R for Data Science: Exercise Solutions

Code

Nick Giangreco

Welcome

This document details a solutions guide to the chapter exercises in *R for Data Science* by Hadley Wickham. The author created the document while in a study group with Jason Baik (https://www.linkedin.com/in/jason-baik-0718a1134/). Inspiration for answers also came from jarnold (https://jrnold.github.io/e4qf/)-thanks to him!

1 Introduction

No exercises

Part I

2 Introduction

No exercises

3 Data visualization

Exercise 3.2.4

1. Run *ggplot()*. This will instantiate an empty coordinate system.

Hide

```
library(tidyverse)
ggplot()
```


2. How many rows and columns are in *mpg*?

Hide

```
nrow(mpg)
```

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

Hide

```
ncol(mpg)
```

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

3. What does the *drv* variable describe?

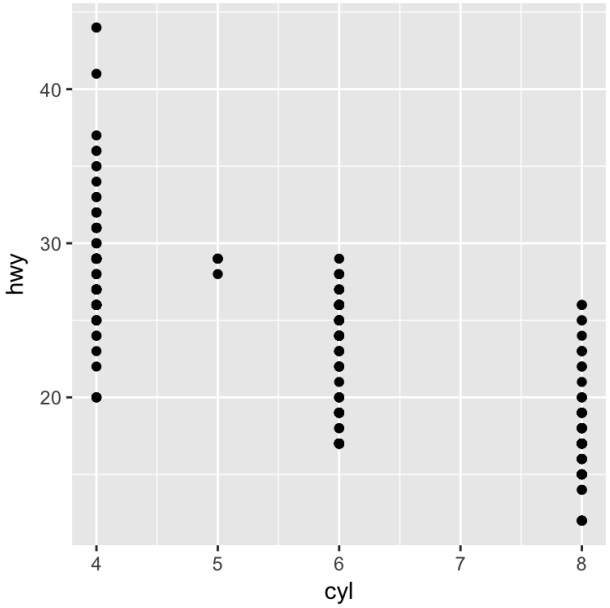
Hide

```
?mpg
```

4. Make a scatterplot between *hwy* and *cyl*.

Hide

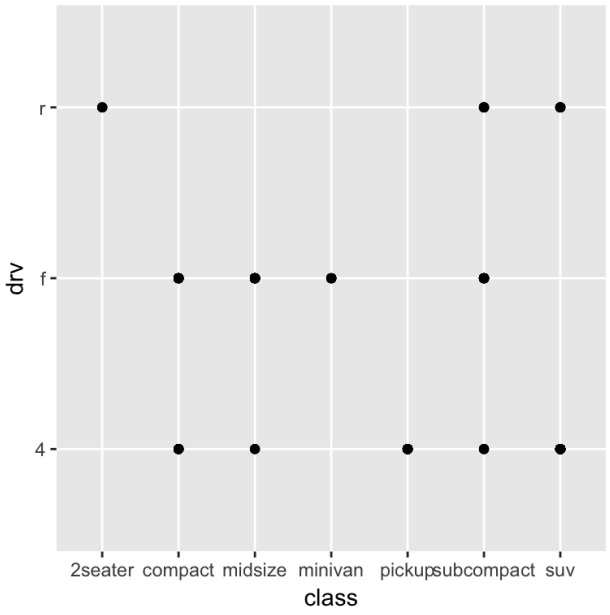
```
ggplot( data = mpg ) +
  geom_point( mapping = aes( x = cyl, y = hwy ) )
```



5. Make a scatterplot between *class* and *drv*. The scatterplot isn't particularly useful because we are not viewing trends-The class of cars and type of drive are both independent variables. The plot would be useful when we are viewing a independent variable and its dependent variable.

Hide

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

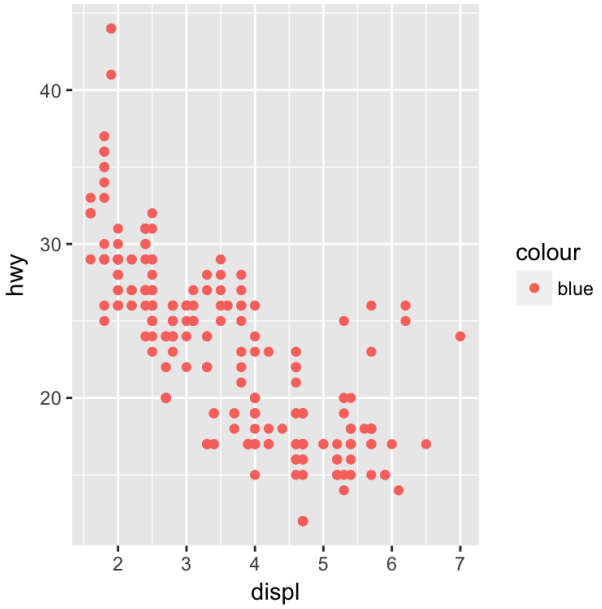


Exercise 3.3.1

1. What has gone wrong with this code? Why are the points not blue?

Hide

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



Because assigning the color within *aes()* provides a mapping of the levels given to color to the x and y coordinates, which in this case is does not because the "blue" by itself does not map to x and y. However, to make all the points blue, one would set the color outside *aes()* within *geom_point()*, which doesn't require a mapping and just passes the property "blue" to all the data points.

2. Which variables in *mpg* are categorical? Which variables are continuous?

Hide

```
mpg
```

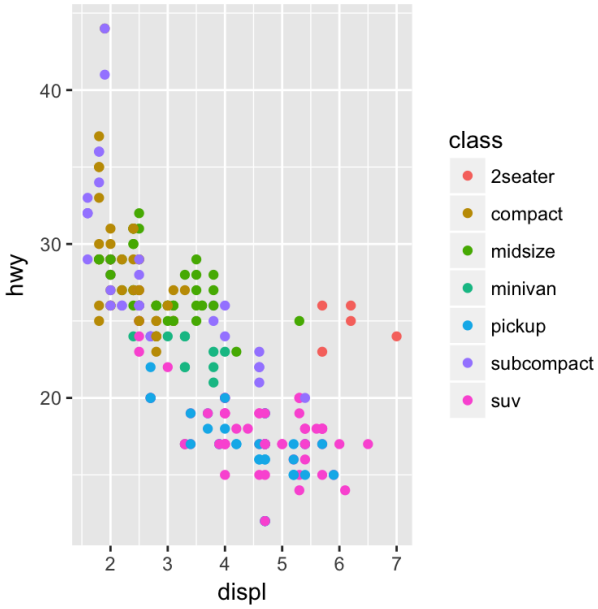
```
## # A tibble: 234 x 11
##    manufacturer      model displ  year   cyl      trans   drv   cty   hwy
##           <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int>
##  1         audi         a4   1.8  1999     4   auto(l5)     f    18    29
##  2         audi         a4   1.8  1999     4 manual(m5)     f    21    29
##  3         audi         a4   2.0  2008     4 manual(m6)     f    20    31
##  4         audi         a4   2.0  2008     4   auto(av)     f    21    30
##  5         audi         a4   2.8  1999     6   auto(l5)     f    16    26
##  6         audi         a4   2.8  1999     6 manual(m5)     f    18    26
##  7         audi         a4   3.1  2008     6   auto(av)     f    18    27
##  8         audi a4 quattro   1.8  1999     4 manual(m5)     4    18    26
##  9         audi a4 quattro   1.8  1999     4   auto(l5)     4    16    25
## 10         audi a4 quattro   2.0  2008     4 manual(m6)     4    20    28
## # ... with 224 more rows, and 2 more variables: fl <chr>, class <chr>
```

Underneath the variable names, a class indicator is given. For example the categorical variables are denoted by “chr” meaning character and are manufacturer, model, trans, drv, fl and class. The continuous variables are denoted by “int” and “dbl” meaning integer and double, respectively, and are year, cyl, trans, cty, and hwy. They are continuous because consecutive numbers are smaller or larger for these variables while categorical variables do not have this relation.

3. Map a continuous variable to color, size, and shape. How do these aesthetics behave differently for categorical vs. continuous variables?

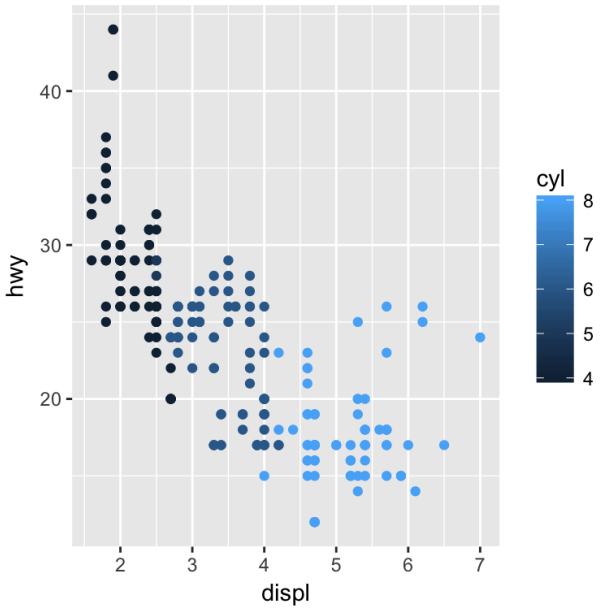
Hide

```
#Color

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



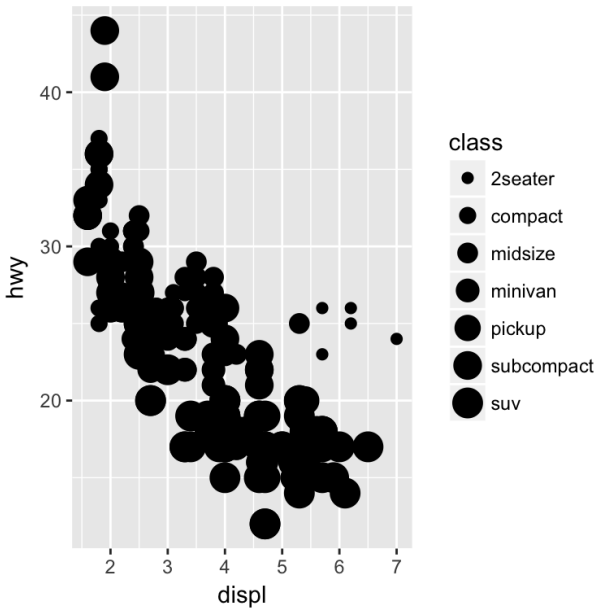
Hide

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



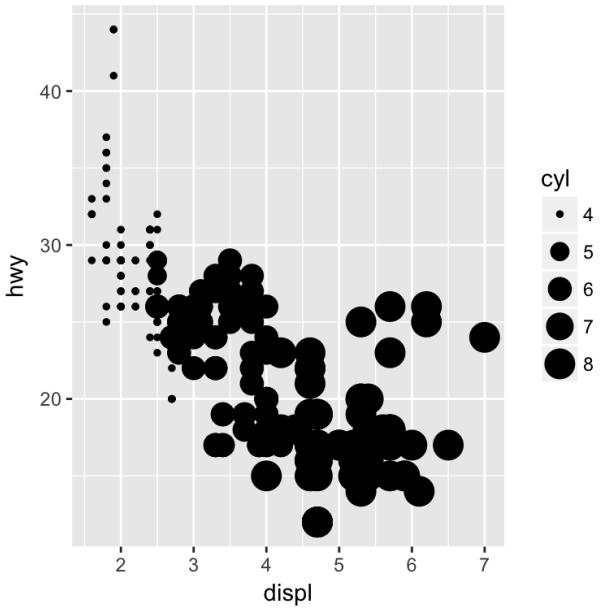
Hide

```
#Size

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



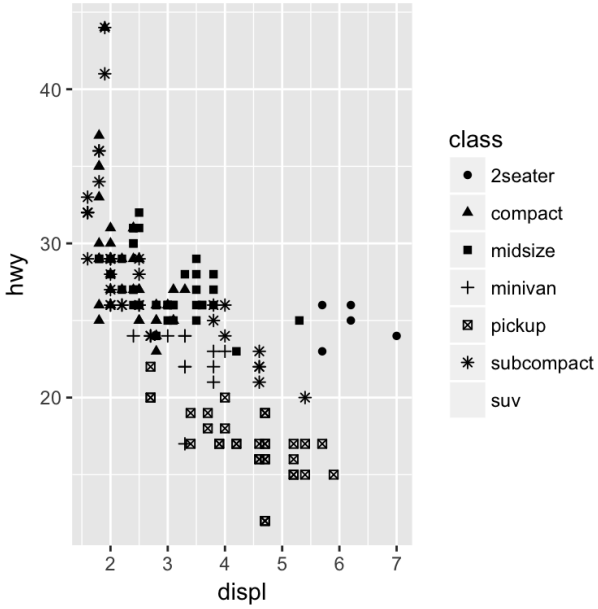
Hide

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



Hide

```
#Shape

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



Hide

```
#continuous-error
#ggplot(data = mpg) +
#  geom_point(mapping = aes(x = displ, y = hwy, shape = cyl ) )
```

For color aesthetics, categorical variables are given discrete colors to map to the color levels, where continuous variables are assigned colors along a gradient to designate a lighter or darker color tone to how large or small the value is. Foe size aesthetics, assigning it a categorucal variable doesn't make sense because increasing size doesn't necessarily mean a larger value, so this only makes sense for continuous variables. For shape aesthetics, only up to 6 discrete levels are assigned shapes because beyond 6 is hard to interpret. Assigning a continuous variable gives an error because a shape has no property for indicating similarity between values.

4. What happens if you map the same variable to multiple aesthetics?

Hide

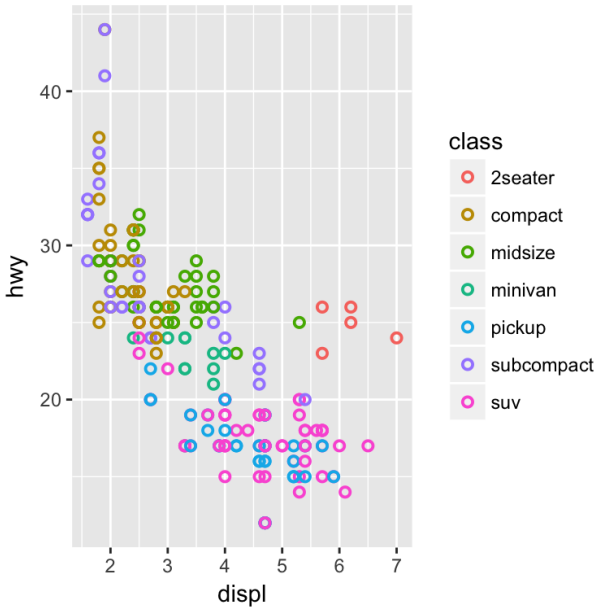
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = class, color = class ) )
```

You're essentially representing a variable twice which is redundant.

5. What does the stroke aesthetic do? What shapes does it work with? (Hint: use ?geom_point)

Hide

```
?geom_point

ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class ), shape = 21, stroke = 1  )
```

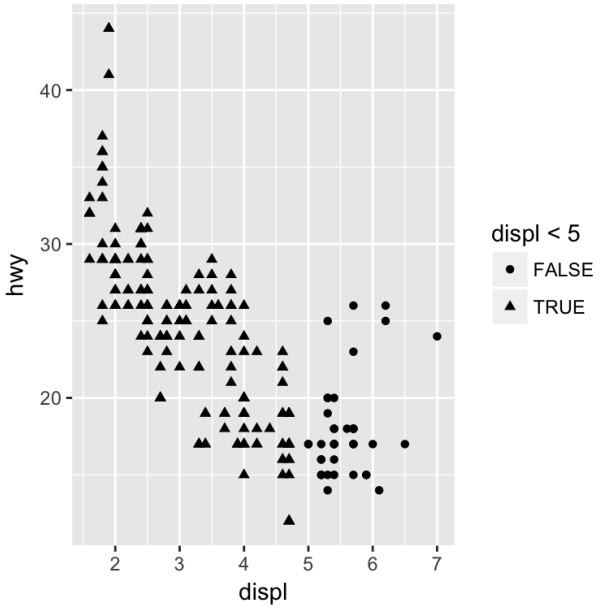


The *stroke* easthetic indicates the width of the shape border which is given as a property to *geom_points()*.

6. What happens if you map an aesthetic to something other than a variable name, like aes(colour = displ < 5)?

Hide

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = displ<5 ))
```



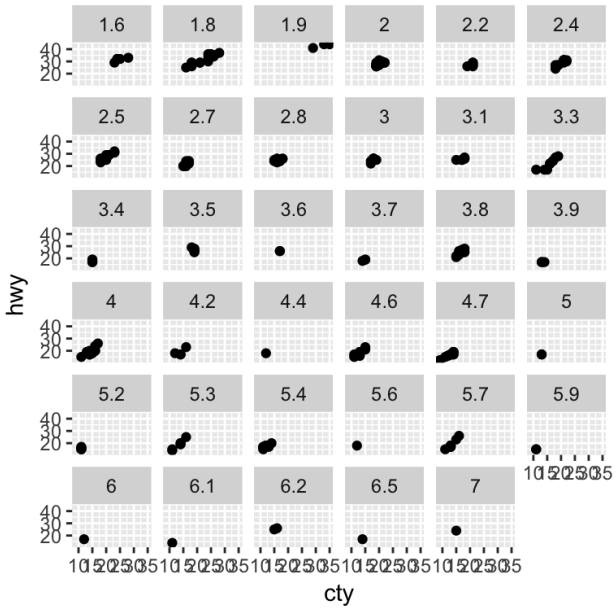
displ<5 in actuallity divides the observations into two, which provides it's own mapping to the observations. So you would get two colors, two shapes, etc. based on how one divides the data.

Exercise 3.5.1

1. What happens if you facet on a continuous variable?

Hide

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

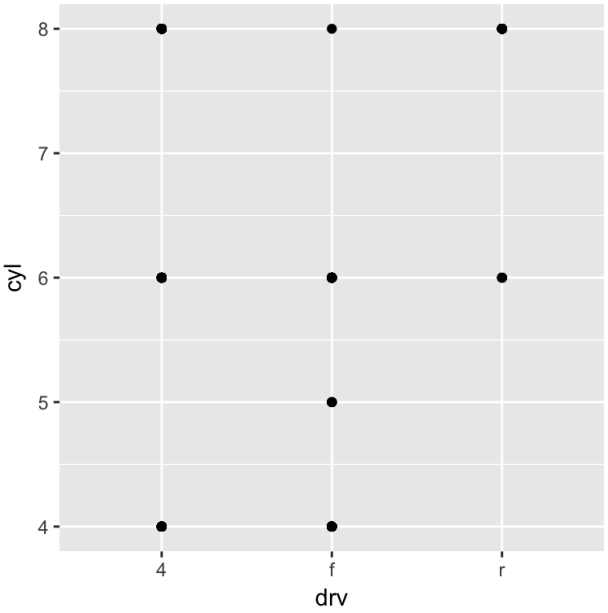


You'll get as many plots as there are distinct values.

2. What do the empty cells in plot with facet_grid(drv ~ cyl) mean? How do they relate to this plot?

Hide

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



Hide

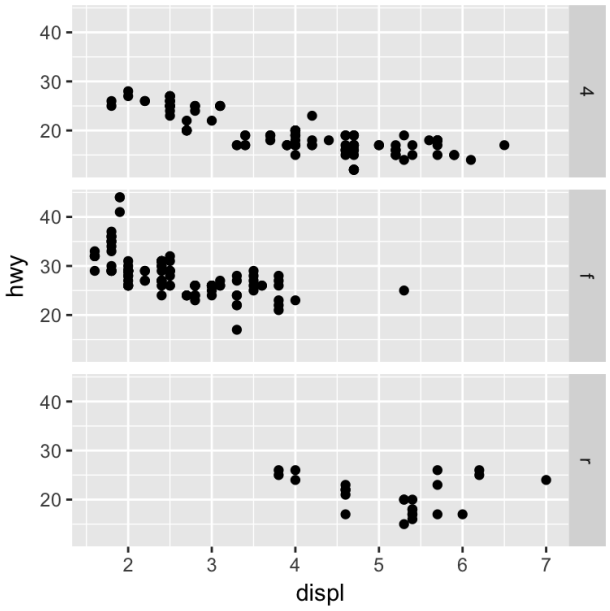
```
  #facet_grid(drv ~ cyl)
```

The empty grids using *facet_grid()* compare to the above plot because there are no points for those combination of variables and hence the empty grids are representing the same as the absence of a point for those combination of variables.

3. What plots does the following code make? What does . do?

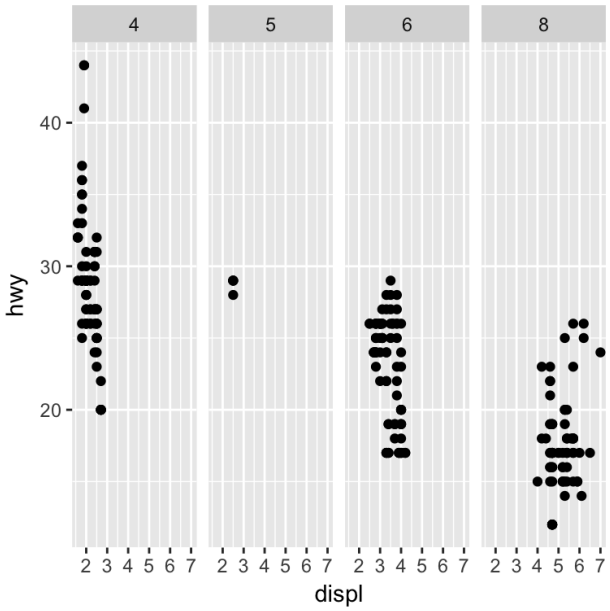
Hide

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



Hide

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

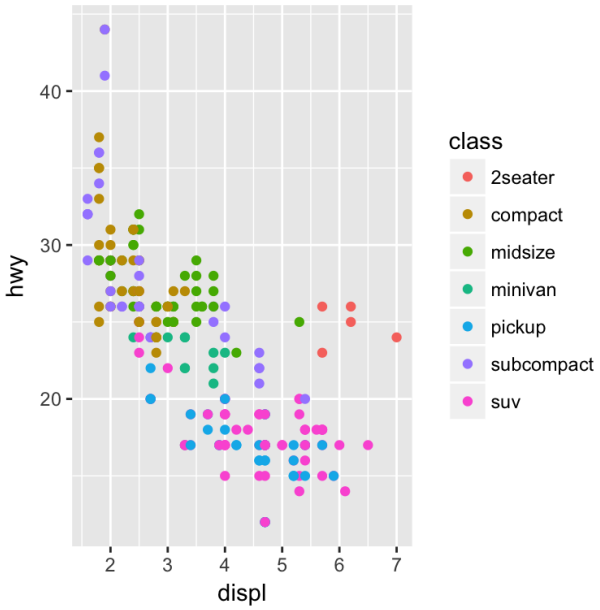


The above plots compare displ vs hwy but for either the drv classes in the rows or the cyl classes in the columns.

4. Take the first faceted plot in this section. What are the advantages to using faceting instead of the colour aesthetic? What are the disadvantages? How might the balance change if you had a larger dataset?

Hide

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



Hide

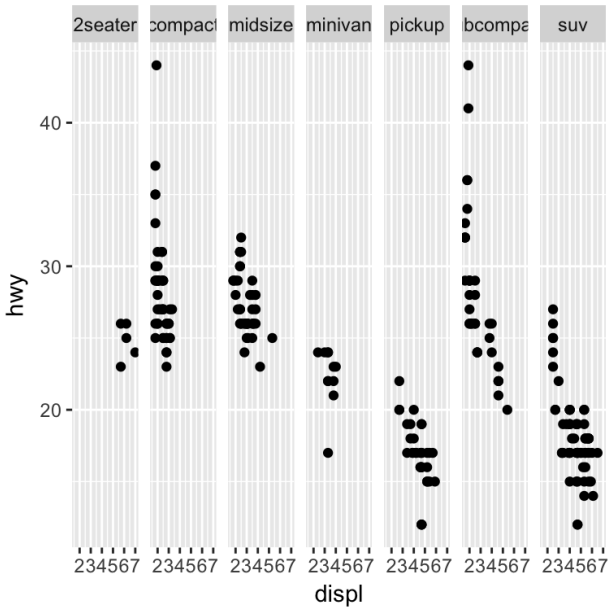
```
#facet_wrap(~ class, nrow = 2)
```

The advantages of labeling the classes by different colors versus showing points for each combination of variables in each grid is that showing all the points in one grid may be too cluttered to show especially with a large dataset and if there are many classes. But it may be worth showing each class in a different color in the same grid if the dataset is smaller and there aren't that many classes to distinguish.

5. Read ?facet_wrap. What does nrow do? What does ncol do? What other options control the layout of the individual panels? Why doesn't facet_grid() have nrow and ncol argument?

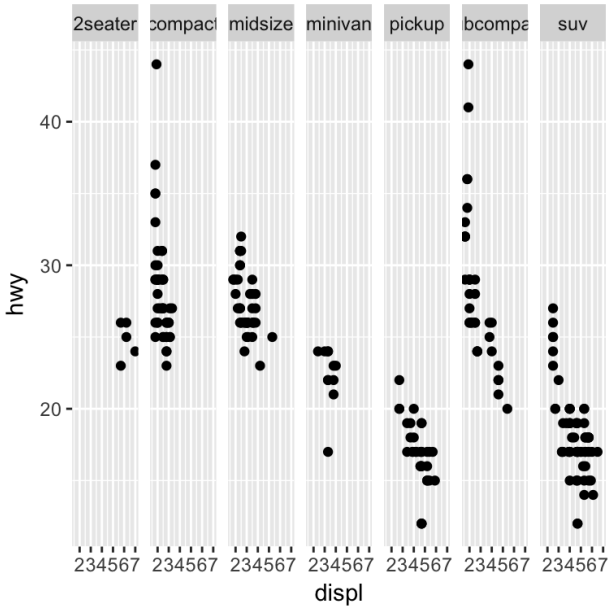
Hide

```
?facet_wrap

ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow=1)
```



Hide

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



Nrow establishes the number of rows to display and ncol establishes the number of columns to display when plotting each facet of the variable. Other options to control the layout of the individual panels includes *scales* (either frees or fixes the x and y scales of facets), *shrink* (shrinks scales to fit output of the statistics, not raw data), *labeller* (a function that takes one data frame of labels and returns a list or data frame of character vectors-changes the labelling on the facets see ?labellers for options), and *switch* (changes the facet labelling from top to bottom or from right to left of plots). *facet_grid()* doesn't have nrow and ncol options because it is the simple case of facet_wrap with nrow = 1. *facet_wrap* allows one to better use screen space.

6. When using facet_grid() you should usually put the variable with more unique levels in the columns. Why?

Because if there are many levels, you'll have many panels in the rows which is not a good use of screen space. *facet_grid()* works well with only a few levels to break up into separate panels.

Exercise 3.6.1

1. What geom would you use to draw a line chart? A boxplot? A histogram? An area chart?

bar chart: geom_bar()

histogram: geom_histogram()

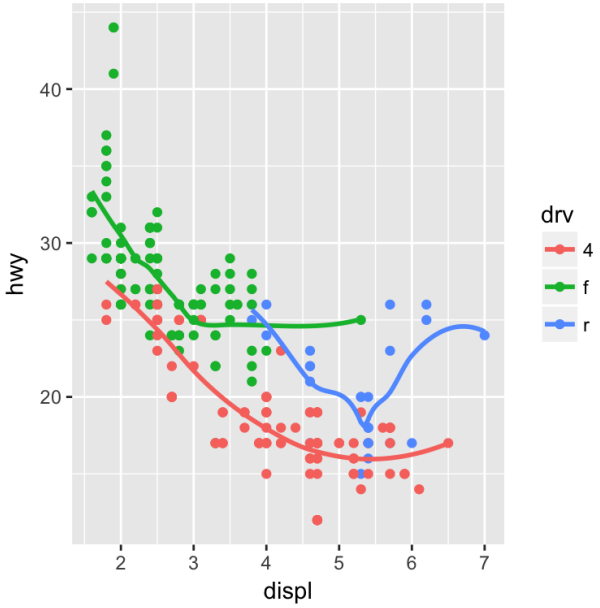
line chart: geom_smooth()

area chart: geom_area()

2. Run this code in your head and predict what the output will look like. Then, run the code in R and check your predictions.

Hide

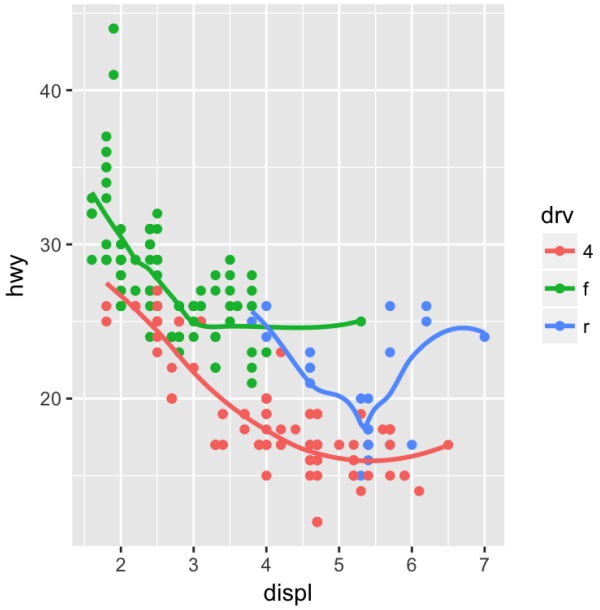
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)
```



I predict this will show a line chart of display vs hwy, without se gray areas around the line, with the corresponding points, and with drive type as color. Also there's 3 different lines per color (the mapping in ggplot() is global so it passes on to geom_point() and geom_smooth()).

Hide

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



3. What does show.legend = FALSE do? What happens if you remove it?

Why do you think I used it earlier in the chapter?

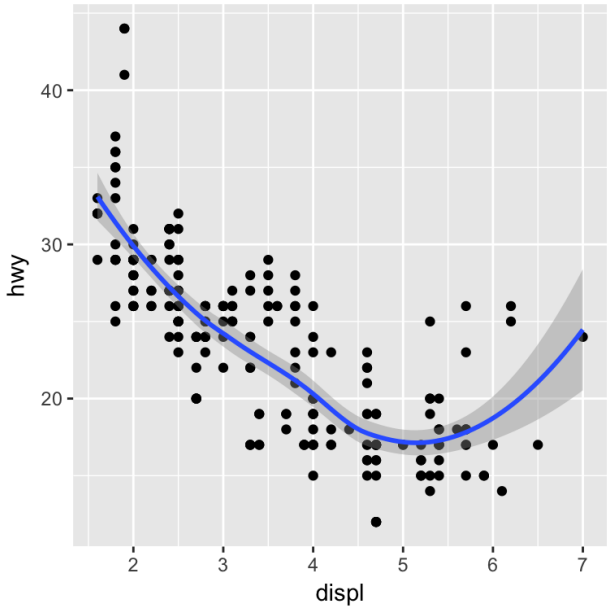
show.legend = FALSE removes the legend key from the side of the plot, which should be done to remove clutter.

4. What does the se argument to geom_smooth() do? It gives standard error gray areas around a line in the line chart to denote where the data lies about the average.

5. Will these two graphs look different? Why/why not?

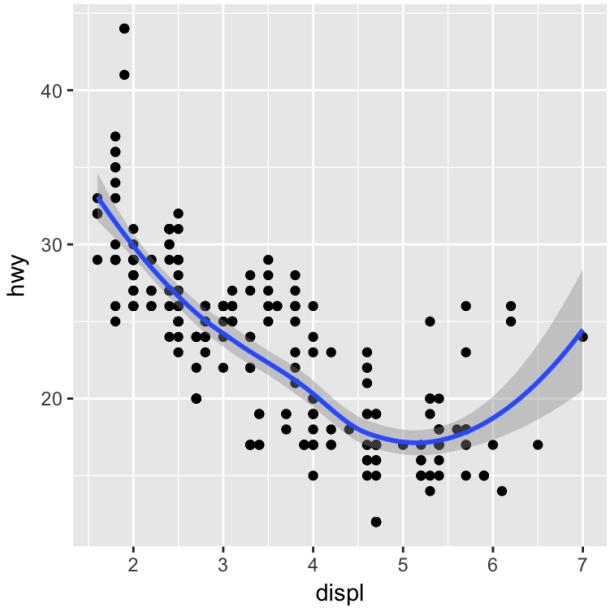
Hide

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



Hide

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

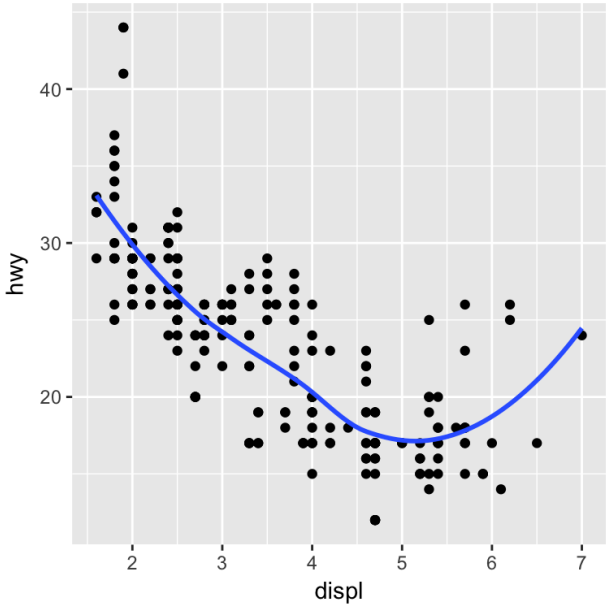


No because both have thesame mappings passed on to geom_point() and geom_smooth() (either globally or locally).

6. Recreating graphs in book

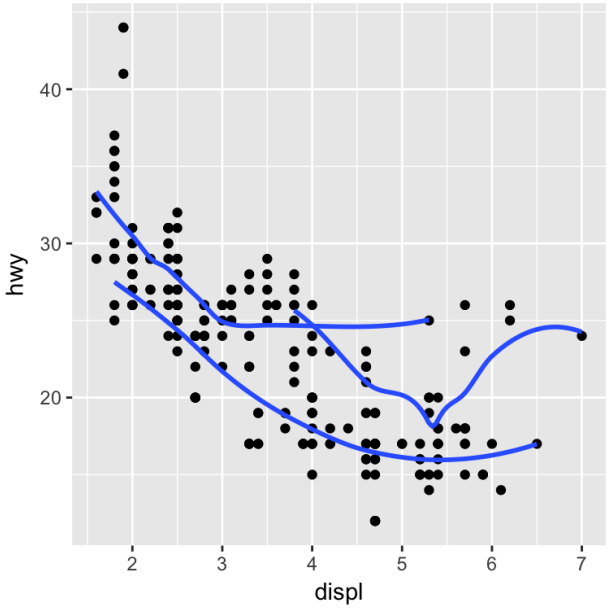
Hide

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



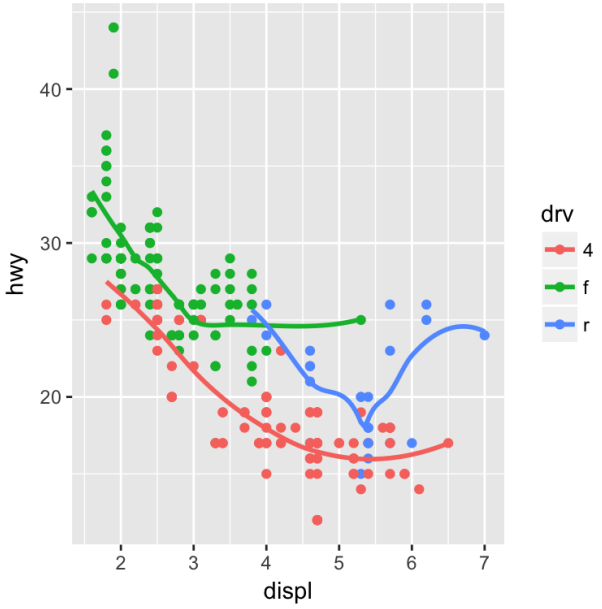
Hide

```
ggplot(data = mpg, mapping = aes(x=displ,y = hwy))+
  geom_point()+
  geom_smooth(se=FALSE, mapping = aes(x=displ,y = hwy, group = drv))
```



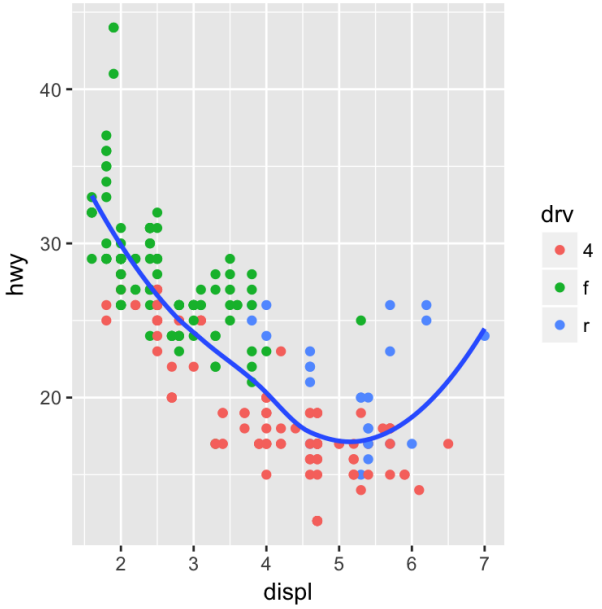
Hide

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



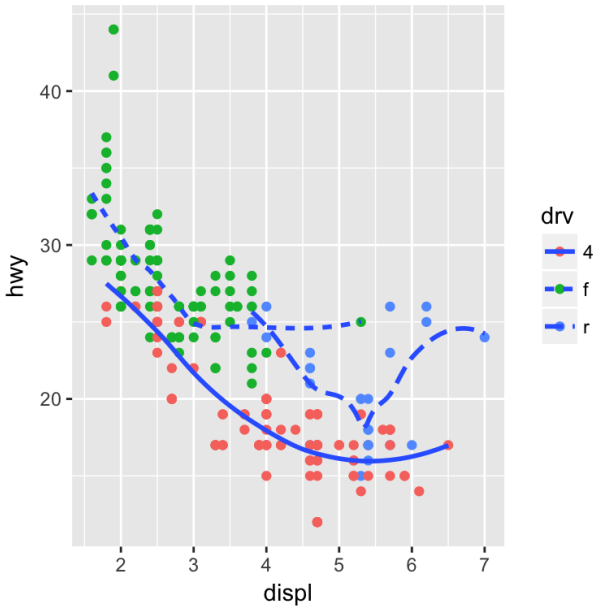
Hide

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



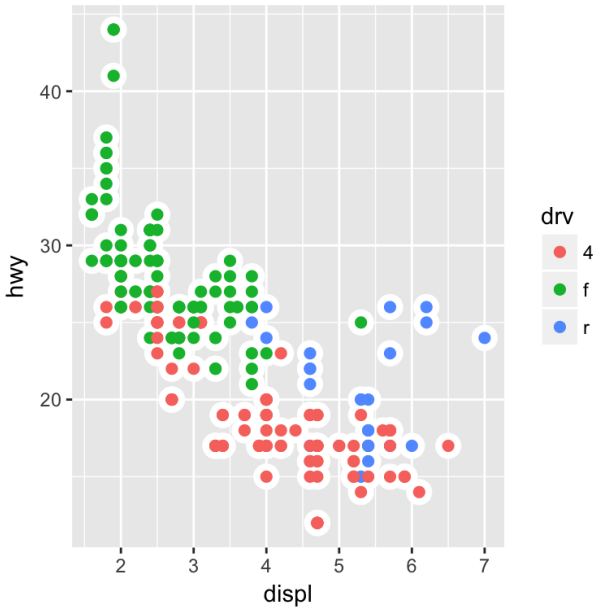
Hide

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



Hide

```
ggplot(data = mpg, mapping = aes(x=displ,y = hwy))+
  geom_point(aes(x=displ,y = hwy),shape=21,size=5,fill="white",color="white")+
  geom_point(aes(x=displ,y = hwy,fill = drv, color = drv),shape=21,size=2)
```



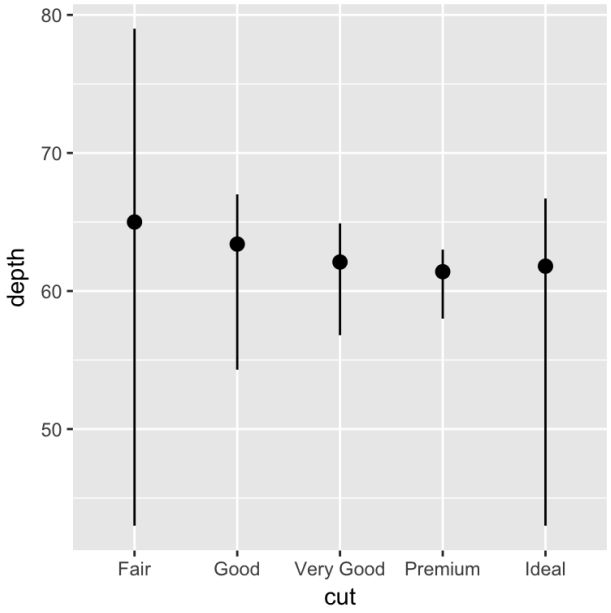
Exercise 3.7.1

1. What is the default geom associated with stat_summary()? How could you rewrite the previous plot to use that geom function instead of the stat function?

The default geom for stat_summary() is geom_pointrange. For geom_pointrange, the default stat is “identity”, so in order to duplicate the previous plot we need to change the stat to summary and change the min, max and midpoint to reflect the same parameters as previously.

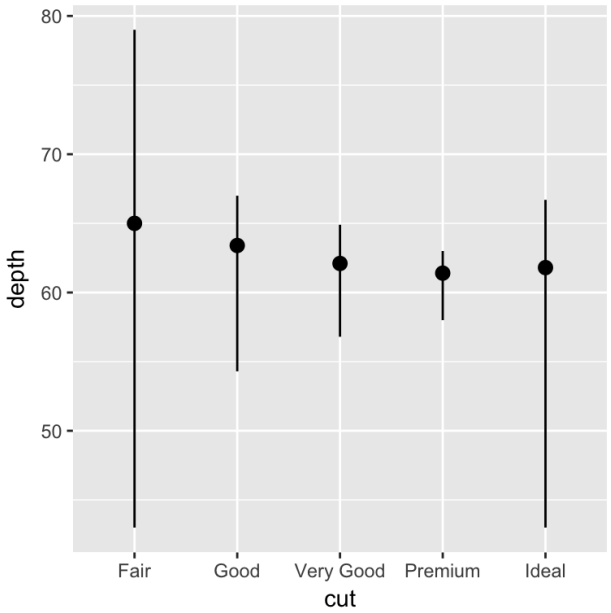
Hide

```
ggplot(data = diamonds) +
  stat_summary(
    mapping = aes(x = cut, y = depth),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  )
```



Hide

```
ggplot(data = diamonds) +
  geom_pointrange(
    mapping = aes(x = cut, y = depth),
    stat = "summary",
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  )
```

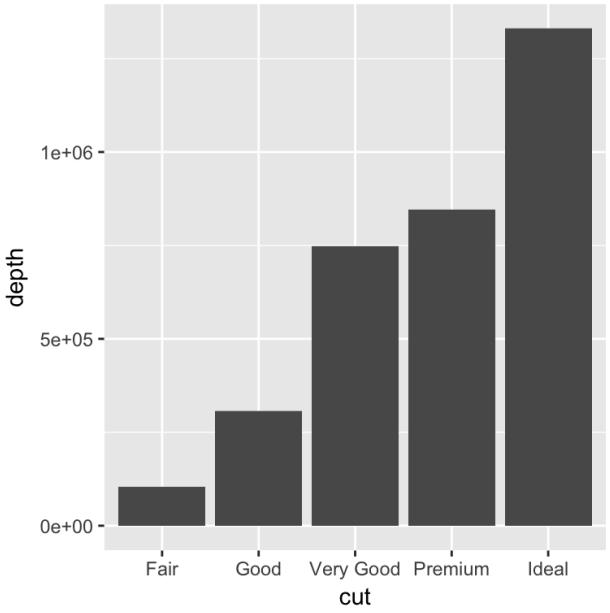


2. What does geom_col() do? How is it different to geom_bar()?

Geom_bar makes the height of the bar proportional to the number of cases in each group (or if the weight aesthetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use geom_col instead.

Hide

```
ggplot(data = diamonds, mapping = aes(x = cut, y = depth)) +
  geom_col()
```



3. Most geoms and stats come in pairs that are almost always used in concert. Read through the documentation and make a list of all the pairs. What do they have in common?

see http://ggplot2.tidyverse.org/reference/ (http://ggplot2.tidyverse.org/reference/)

4. What variables does stat_smooth() compute? What parameters control its behaviour?

stat_smooth calculates:

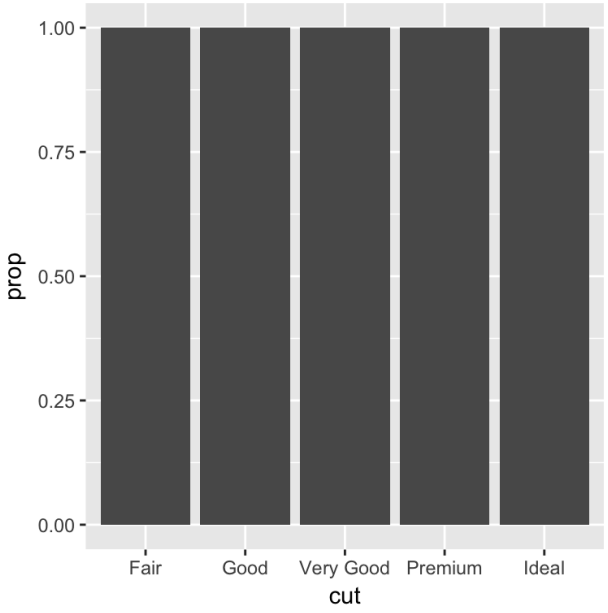
y: predicted value ymin: lower value of the confidence interval ymax: upper value of the confidence interval se: standard error

There's parameters such as method which determines which method is used to calculate the predictions and confidence interval, and some other arguments that are passed to that.

5. In our proportion bar chart, we need to set group = 1. Why? In other words what is the problem with these two graphs?

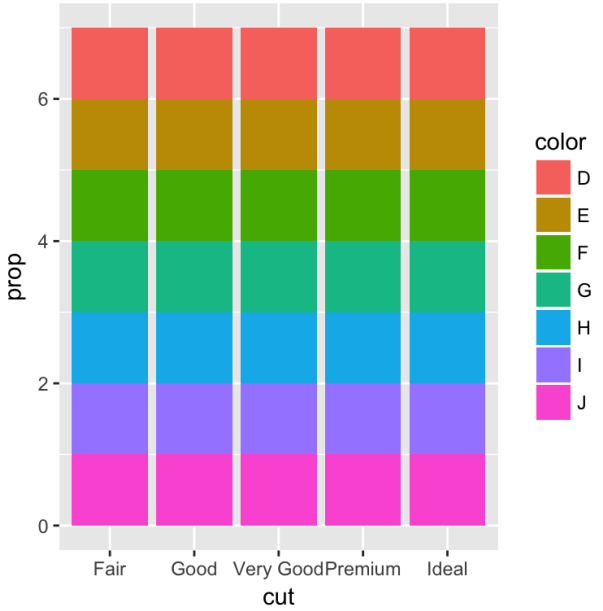
Hide

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop..))
```



Hide

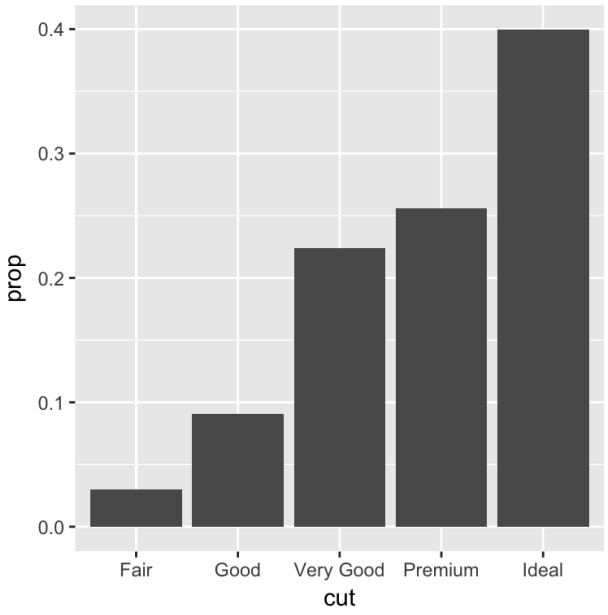
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = color, y = ..prop..))
```



If group is not set to 1, then all the bars have prop == 1. The function geom_bar assumes that the groups are equal to the x values, since the stat computes the counts within the group.

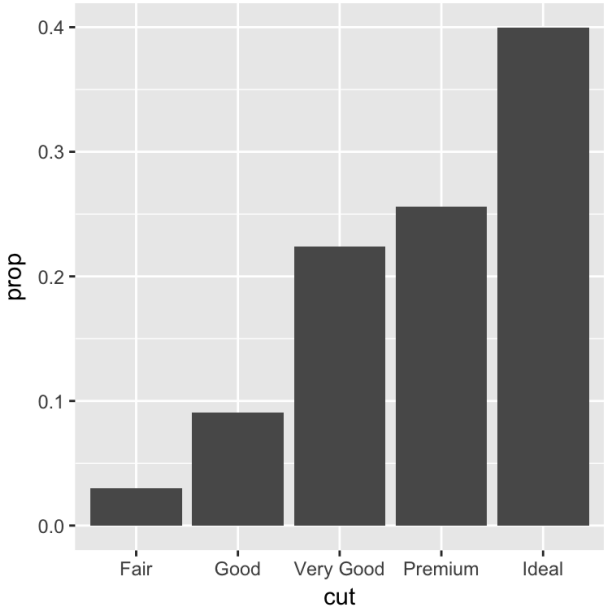
Hide

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop..,group=1))
```



Hide

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = color, y = ..prop..,group=1))
```

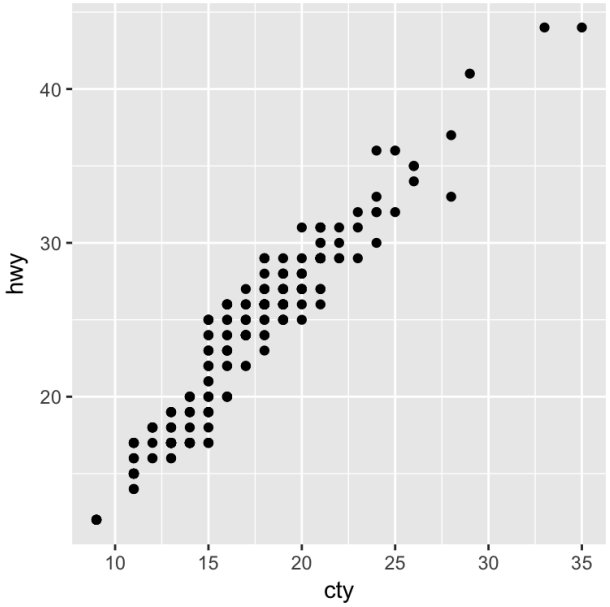


Exercise 3.8.1

1. What is the problem with this plot? How could you improve it?

Hide

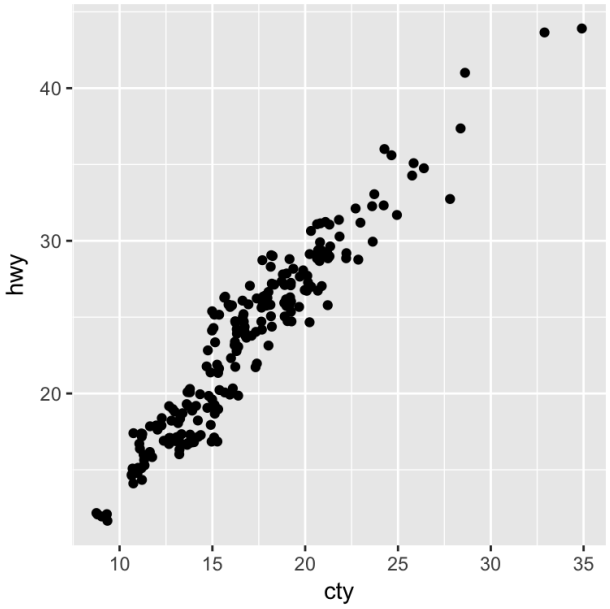
```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point()
```



A lot of points aren't shown here because they overlap. Using geom_jitter() allows you to see them all.

Hide

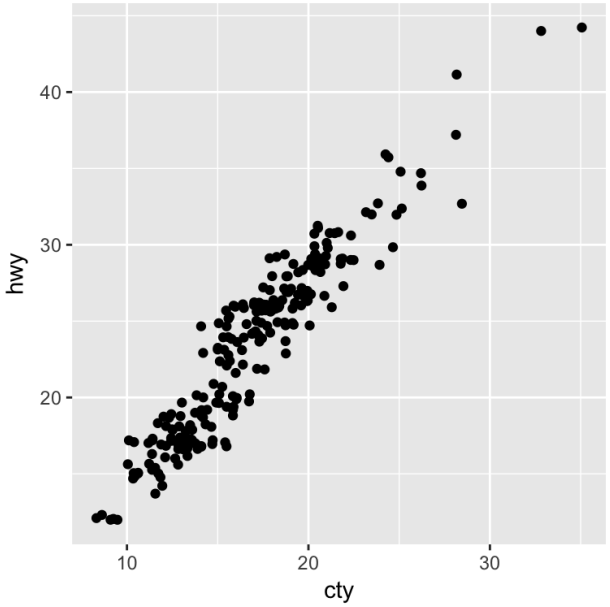
```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_jitter()
```



2. What parameters to geom_jitter() control the amount of jittering?

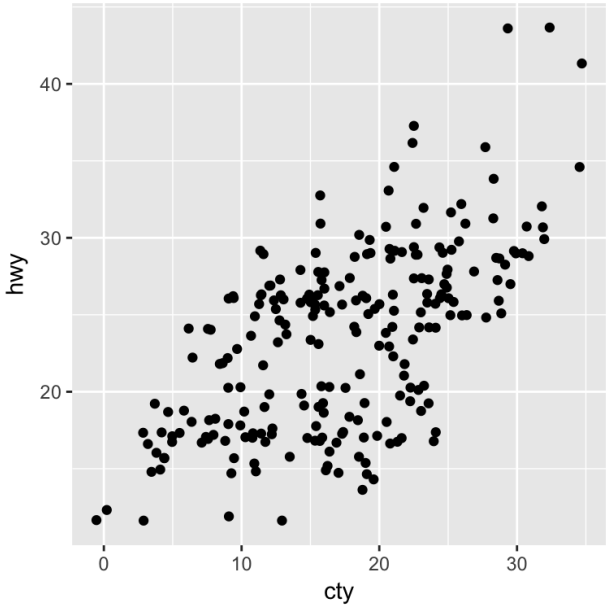
Hide

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_jitter(width=1)
```



Hide

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_jitter(width=10)
```

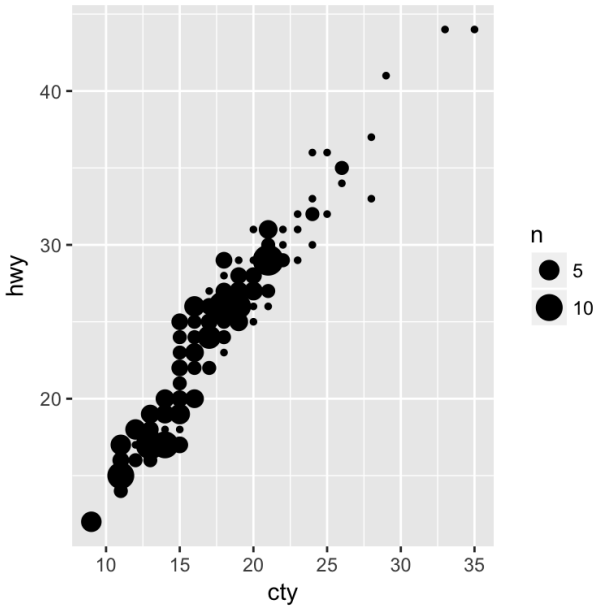


The amount of jitter is controlled by the width argument-increases the distance (noise) between the points.

3. Compare and contrast geom_jitter() with geom_count().

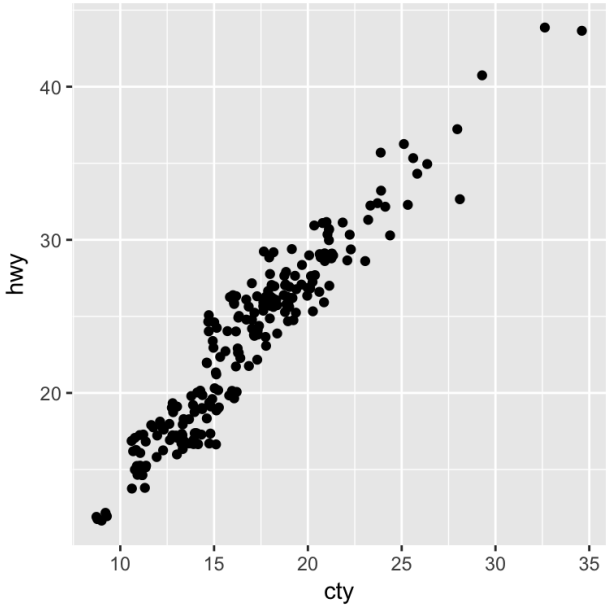
Hide

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_count()
```



Hide

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_jitter()
```



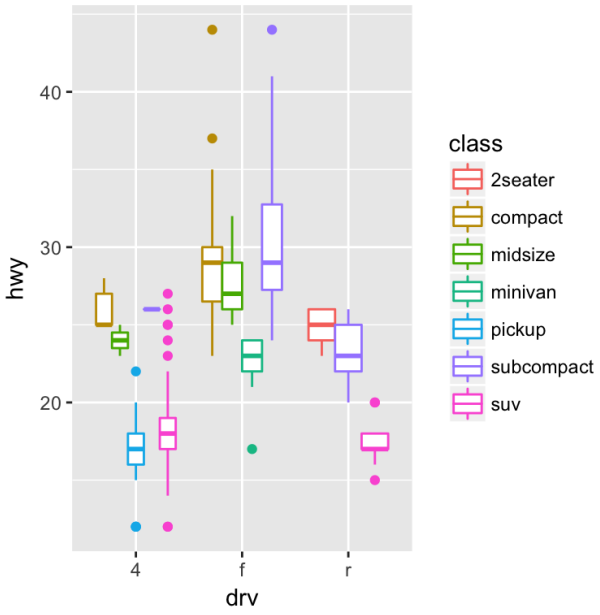
Seems like geom_count() increases the size of the points when there are more overlapping points. Kind of like estimating the density of points in that location. geom_jitter() just makes all the points visible and the same size.

4. What's the default position adjustment for geom_boxplot()? Create a visualisation of the mpg dataset that demonstrates it.

The deault is for the boxplots to be non overlapping or dodged.

Hide

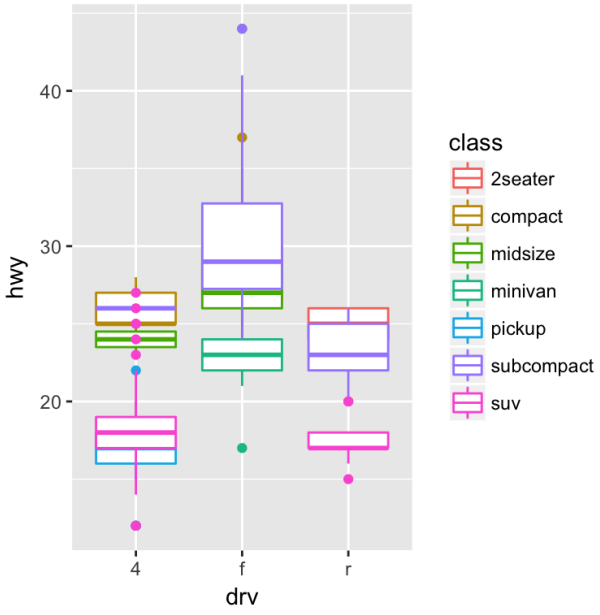
```
ggplot(data = mpg, mapping = aes(x = drv, y = hwy, color = class)) +
  geom_boxplot(position="dodge")
```



In contrast, we can have them overlapping by using identity.

Hide

```
ggplot(data = mpg, mapping = aes(x = drv, y = hwy, color = class)) +
  geom_boxplot(position="identity")
```

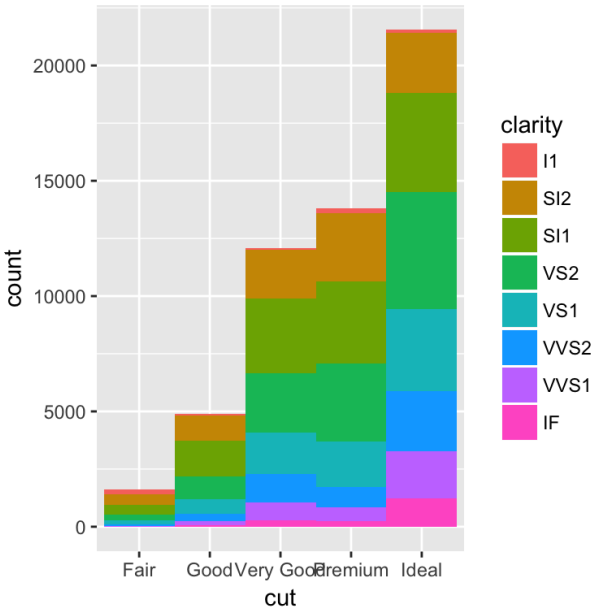


Exercise 3.9.1

1. Turn a stacked bar chart into a pie chart using coord_polar().

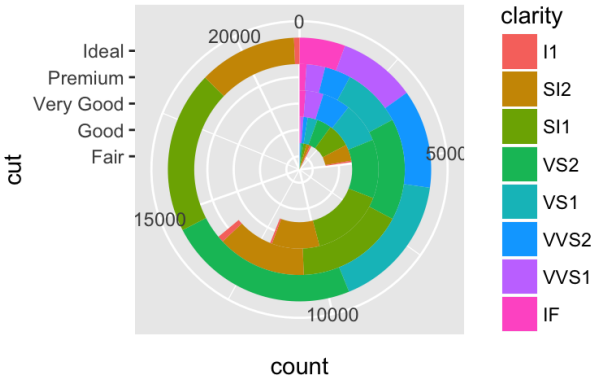
Hide

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), width = 1)
```



Hide

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), width=1)+coord_polar(theta = "y")
```



2. What does labs() do? Read the documentation.

Hide

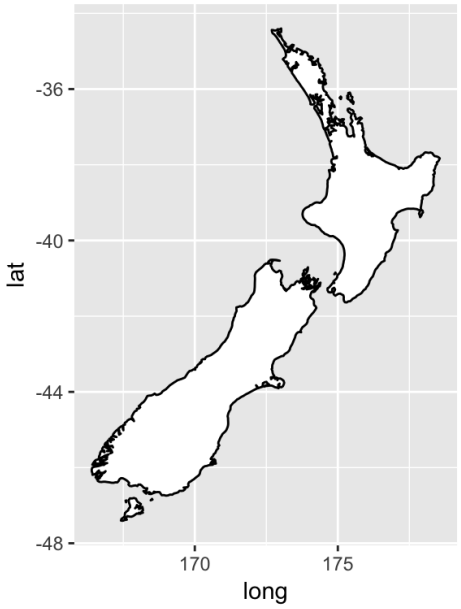
```
?labs
```

Allows one to label the coordinates.

3. What's the difference between coord_quickmap() and coord_map()?

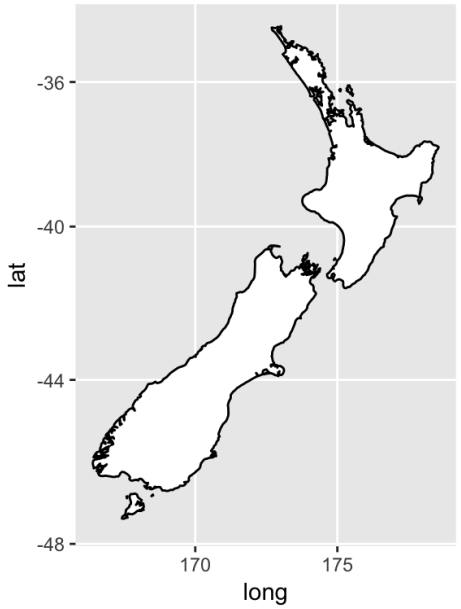
Hide

```
nz<-map_data("nz")

ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black") +
  coord_quickmap()
```



Hide

```
ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black") +
  coord_map()
```



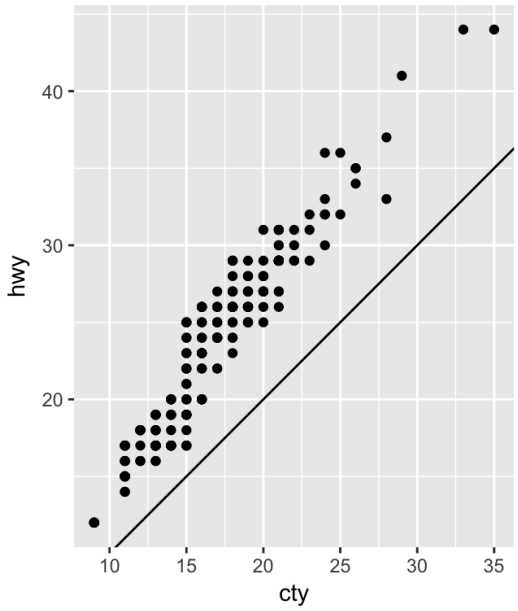
Seems like coord_map() eliminates some grid lines and shrinks the map a tiny bit.

- coord_map uses a 2D projection: by default the Mercatur project of the sphere to the plot. But this requires transforming all geoms.
- coord_quickmap uses a quick approximation by using the lat/long ratio as an approximation. This is “quick” because the shapes don’t need to be transformed.

4. What does the plot below tell you about the relationship between city and highway mpg? Why is coord_fixed() important? What does geom_abline() do?

Hide

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_fixed()
```



The abline shown with the scatter points between cty and hwy tell me that one gets higher highway mpg compared to city mpg, but they are positively correlated. geom_abline() gives the x = y line where if the points were on that line the highway and city mpg would be the same. coord_fixed() fixes the ratio between the physical representation of data units on the axes-the ratio represents the number of units on the y-axis equivalent to one unit on the x-axis.Also, coord_fixed() ensures that the abline is at a 45 degree angle, which makes it easy to compare the highway and city mileage against what it would be if they were exactly the same.

4 Workflow: Basics

Exercise 4.4

1. Why does this code not work?

Hide

```
my_variable <- 10
#my_varıable #Error: object 'my_varıable' not found
```

Because my_variable that is assigned is different from my_varıable, which is not assigned and thus R will throw an error since it's not an object in the environment.

2. Tweak each of the following R commands so that they run correctly:

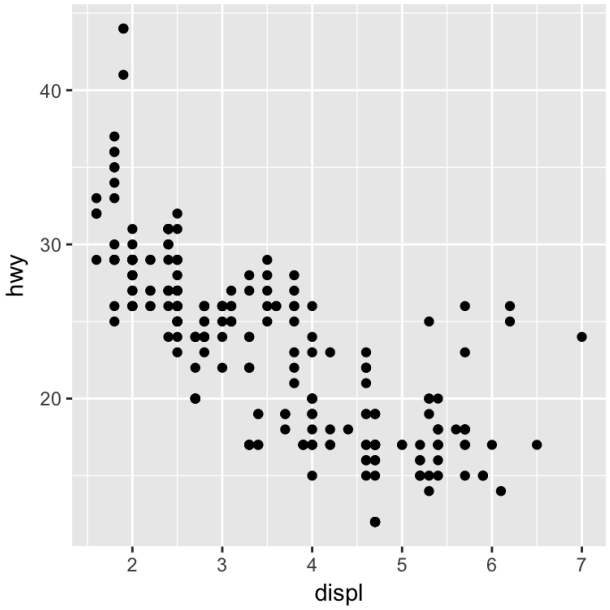
Hide

```
library(tidyverse)

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

fliter(mpg, cyl = 8)
filter(diamond, carat > 3)
```

'dota' should be 'data', 'fliter' should be 'filter', '=' should be '==', and 'diamond' should be 'diamonds'. It should be:

Hide

```
library(tidyverse)

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



Hide

```
filter(mpg, cyl == 8)
```

```
## # A tibble: 70 x 11
##    manufacturer              model displ  year   cyl      trans   drv
##           <chr>              <chr> <dbl> <int> <int>      <chr> <chr>
##  1         audi         a6 quattro   4.2  2008     8   auto(s6)     4
##  2    chevrolet c1500 suburban 2wd   5.3  2008     8   auto(l4)     r
##  3    chevrolet c1500 suburban 2wd   5.3  2008     8   auto(l4)     r
##  4    chevrolet c1500 suburban 2wd   5.3  2008     8   auto(l4)     r
##  5    chevrolet c1500 suburban 2wd   5.7  1999     8   auto(l4)     r
##  6    chevrolet c1500 suburban 2wd   6.0  2008     8   auto(l4)     r
##  7    chevrolet           corvette   5.7  1999     8 manual(m6)     r
##  8    chevrolet           corvette   5.7  1999     8   auto(l4)     r
##  9    chevrolet           corvette   6.2  2008     8 manual(m6)     r
## 10    chevrolet           corvette   6.2  2008     8   auto(s6)     r
## # ... with 60 more rows, and 4 more variables: cty <int>, hwy <int>,
## #   fl <chr>, class <chr>
```

Hide

```
filter(diamonds, carat > 3)
```

```
## # A tibble: 32 x 10
##    carat     cut color clarity depth table price     x     y     z
##    <dbl>   <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  3.01 Premium     I      I1  62.7    58  8040  9.10  8.97  5.67
##  2  3.11    Fair     J      I1  65.9    57  9823  9.15  9.02  5.98
##  3  3.01 Premium     F      I1  62.2    56  9925  9.24  9.13  5.73
##  4  3.05 Premium     E      I1  60.9    58 10453  9.26  9.25  5.66
##  5  3.02    Fair     I      I1  65.2    56 10577  9.11  9.02  5.91
##  6  3.01    Fair     H      I1  56.1    62 10761  9.54  9.38  5.31
##  7  3.65    Fair     H      I1  67.1    53 11668  9.53  9.48  6.38
##  8  3.24 Premium     H      I1  62.1    58 12300  9.44  9.40  5.85
##  9  3.22   Ideal     I      I1  62.6    55 12545  9.49  9.42  5.92
## 10  3.50   Ideal     H      I1  62.8    57 12587  9.65  9.59  6.03
## # ... with 22 more rows
```

5 Data Transformation

Exercise 5.2.4

1. Find all flights that

(a) Had an arrival delay of two hours.

Hide

```
library(nycflights13)
library(tidyverse)
#glimpse(flights)
filter(flights, arr_delay > 119 & arr_delay < 121)
```

```
## # A tibble: 166 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     2     1806           1629        97     2008
##  2  2013     1    10     1801           1619       102     1923
##  3  2013     1    13     1958           1836        82     2231
##  4  2013     1    13     2145           2005       100        4
##  5  2013     1    14     1652           1445       127     1806
##  6  2013     1    15     1603           1446        77     1957
##  7  2013     1    21     1957           1815       102     2237
##  8  2013     1    22     1550           1420        90     1820
##  9  2013     1    23     1031            825       126     1337
## 10  2013     1    23     1805           1619       106     1926
## # ... with 156 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

(b) flew to Houston

Hide

```
filter(flights, dest == 'HOU' | dest == 'IAH')
```

```
## # A tibble: 9,313 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      623            627        -4      933
##  4  2013     1     1      728            732        -4     1041
##  5  2013     1     1      739            739         0     1104
##  6  2013     1     1      908            908         0     1228
##  7  2013     1     1     1028           1026         2     1350
##  8  2013     1     1     1044           1045        -1     1352
##  9  2013     1     1     1114            900       134     1447
## 10  2013     1     1     1205           1200         5     1503
## # ... with 9,303 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

(c) Were operated by American, United or Delta airlines.

Hide

```
filter(flights, carrier == "UA" | carrier == 'AA' | carrier == 'DA')
```

```
## # A tibble: 91,394 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      554            558        -4      740
##  5  2013     1     1      558            600        -2      753
##  6  2013     1     1      558            600        -2      924
##  7  2013     1     1      558            600        -2      923
##  8  2013     1     1      559            600        -1      941
##  9  2013     1     1      559            600        -1      854
## 10  2013     1     1      606            610        -4      858
## # ... with 91,384 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

(d) Departed in the summeer (July, August and September)

Hide

```
filter(flights, month == 7 | month == 8 | month == 9)
```

```
## # A tibble: 86,326 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     7     1        1           2029       212      236
##  2  2013     7     1        2           2359         3      344
##  3  2013     7     1       29           2245       104      151
##  4  2013     7     1       43           2130       193      322
##  5  2013     7     1       44           2150       174      300
##  6  2013     7     1       46           2051       235      304
##  7  2013     7     1       48           2001       287      308
##  8  2013     7     1       58           2155       183      335
##  9  2013     7     1      100           2146       194      327
## 10  2013     7     1      100           2245       135      337
## # ... with 86,316 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

(e) Arrived more than two hours late, but didn't leave late

Hide

```
filter(flights,arr_delay > 120 & dep_delay <= 0)
```

```
## # A tibble: 29 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1    27     1419           1420        -1     1754
##  2  2013    10     7     1350           1350         0     1736
##  3  2013    10     7     1357           1359        -2     1858
##  4  2013    10    16      657            700        -3     1258
##  5  2013    11     1      658            700        -2     1329
##  6  2013     3    18     1844           1847        -3       39
##  7  2013     4    17     1635           1640        -5     2049
##  8  2013     4    18      558            600        -2     1149
##  9  2013     4    18      655            700        -5     1213
## 10  2013     5    22     1827           1830        -3     2217
## # ... with 19 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

(f) Were delayed by at least an hour, but made up over 30 minutes in flight

Hide

```
filter(flights, dep_delay > 60 & air_time > 30)
```

```
## # A tibble: 26,185 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      811            630       101     1047
##  2  2013     1     1      826            715        71     1136
##  3  2013     1     1      848           1835       853     1001
##  4  2013     1     1      957            733       144     1056
##  5  2013     1     1     1114            900       134     1447
##  6  2013     1     1     1120            944        96     1331
##  7  2013     1     1     1301           1150        71     1518
##  8  2013     1     1     1337           1220        77     1649
##  9  2013     1     1     1400           1250        70     1645
## 10  2013     1     1     1505           1310       115     1638
## # ... with 26,175 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

(g) Departed between midnight and 6am (inclusive)

Hide

```
filter(flights, hour >= 0 & hour <= 6)
```

```
## # A tibble: 27,905 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 27,895 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

2. Another useful dplyr filtering helper is between(). What does it do? Can you use it to simplify the code needed to answer the previous challenges?

between() is a shortcut for an inclusive range, implemented efficiently in C++. So instead of filter(flights, hour >= 0 & hour <= 6), we can have between(flights$hour,0,6) which would give TRUE for rows of flights in the range. Or, to simplify 1g, do filter(flights, between(dep_time, 0, 600)). (I don't think it simplifies it.)

3. How many flights have a missing dep_time? What other variables are missing? What might these rows represent?

Hide

```
filter(flights, is.na(dep_time) )
```

```
## # A tibble: 8,255 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1       NA           1630        NA       NA
##  2  2013     1     1       NA           1935        NA       NA
##  3  2013     1     1       NA           1500        NA       NA
##  4  2013     1     1       NA            600        NA       NA
##  5  2013     1     2       NA           1540        NA       NA
##  6  2013     1     2       NA           1620        NA       NA
##  7  2013     1     2       NA           1355        NA       NA
##  8  2013     1     2       NA           1420        NA       NA
##  9  2013     1     2       NA           1321        NA       NA
## 10  2013     1     2       NA           1545        NA       NA
## # ... with 8,245 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Along with departure time, the arrival time, delays, and air time are missing. But these flights are scheduled-it seems that these flights were scheduled but wasn't recorded to actually fly.

4. Why is NA ^ 0 not missing? Why is NA | TRUE not missing? Why is FALSE & NA not missing? Can you figure out the general rule? (NA * 0 is a tricky counterexample!)

Hide

```
NA^0
```

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

Hide

```
NA | TRUE
```

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

Hide

```
FALSE & NA
```

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

Hide

```
NA * 0
```

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

Hmm. Anything to the 0 is 1, NA | TRUE is TRUE because of the TRUE, FALSE & NA is false because anything and false is always false, because the value of the missing element matters in NA | FALSE and NA & TRUE, these are missing (NA), and the reason that NA * 0 is not equal to 0 is that x * 0 = NaN is undefined when x = Inf or x = -Inf.

Exercise 5.3.1

1. How could you use arrange() to sort all missing values to the start? (Hint: use is.na()).

for example:

Hide

```
arrange(flights, desc(is.na(dep_time)), dep_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1       NA           1630        NA       NA
##  2  2013     1     1       NA           1935        NA       NA
##  3  2013     1     1       NA           1500        NA       NA
##  4  2013     1     1       NA            600        NA       NA
##  5  2013     1     2       NA           1540        NA       NA
##  6  2013     1     2       NA           1620        NA       NA
##  7  2013     1     2       NA           1355        NA       NA
##  8  2013     1     2       NA           1420        NA       NA
##  9  2013     1     2       NA           1321        NA       NA
## 10  2013     1     2       NA           1545        NA       NA
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

2. Sort flights to find the most delayed flights. Find the flights that left earliest

Hide

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     9      641            900      1301     1242
##  2  2013     6    15     1432           1935      1137     1607
##  3  2013     1    10     1121           1635      1126     1239
##  4  2013     9    20     1139           1845      1014     1457
##  5  2013     7    22      845           1600      1005     1044
##  6  2013     4    10     1100           1900       960     1342
##  7  2013     3    17     2321            810       911      135
##  8  2013     6    27      959           1900       899     1236
##  9  2013     7    22     2257            759       898      121
## 10  2013    12     5      756           1700       896     1058
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Hide

```
arrange(flights, dep_delay)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013    12     7     2040           2123       -43       40
##  2  2013     2     3     2022           2055       -33     2240
##  3  2013    11    10     1408           1440       -32     1549
##  4  2013     1    11     1900           1930       -30     2233
##  5  2013     1    29     1703           1730       -27     1947
##  6  2013     8     9      729            755       -26     1002
##  7  2013    10    23     1907           1932       -25     2143
##  8  2013     3    30     2030           2055       -25     2213
##  9  2013     3     2     1431           1455       -24     1601
## 10  2013     5     5      934            958       -24     1225
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

3. Sort flights to find the fastest flights.

Hide

```
arrange(flights, air_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1    16     1355           1315        40     1442
##  2  2013     4    13      537            527        10      622
##  3  2013    12     6      922            851        31     1021
##  4  2013     2     3     2153           2129        24     2247
##  5  2013     2     5     1303           1315       -12     1342
##  6  2013     2    12     2123           2130        -7     2211
##  7  2013     3     2     1450           1500       -10     1547
##  8  2013     3     8     2026           1935        51     2131
##  9  2013     3    18     1456           1329        87     1533
## 10  2013     3    19     2226           2145        41     2305
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

4. Which flights travelled the longest? Which travelled the shortest?

Hide

```
arrange(flights,desc(distance))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      857            900        -3     1516
##  2  2013     1     2      909            900         9     1525
##  3  2013     1     3      914            900        14     1504
##  4  2013     1     4      900            900         0     1516
##  5  2013     1     5      858            900        -2     1519
##  6  2013     1     6     1019            900        79     1558
##  7  2013     1     7     1042            900       102     1620
##  8  2013     1     8      901            900         1     1504
##  9  2013     1     9      641            900      1301     1242
## 10  2013     1    10      859            900        -1     1449
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Hide

```
arrange(flights,distance)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     7    27       NA            106        NA       NA
##  2  2013     1     3     2127           2129        -2     2222
##  3  2013     1     4     1240           1200        40     1333
##  4  2013     1     4     1829           1615       134     1937
##  5  2013     1     4     2128           2129        -1     2218
##  6  2013     1     5     1155           1200        -5     1241
##  7  2013     1     6     2125           2129        -4     2224
##  8  2013     1     7     2124           2129        -5     2212
##  9  2013     1     8     2127           2130        -3     2304
## 10  2013     1     9     2126           2129        -3     2217
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Exercise 5.4.1

1. Brainstorm as many ways as possible to select dep_time, dep_delay, arr_time, and arr_delay from flights.

I could reorder, I could select a subset or I could remove variables.

2. What happens if you include the name of a variable multiple times in a select() call?

Hide

```
select(flights, dep_time, dep_time)
```

```
## # A tibble: 336,776 x 1
##    dep_time
##       <int>
##  1      517
##  2      533
##  3      542
##  4      544
##  5      554
##  6      554
##  7      555
##  8      557
##  9      557
## 10      558
## # ... with 336,766 more rows
```

It ignores the call to the same variable if it already occurred.

3. What does the one_of() function do? Why might it be helpful in conjunction with this vector?

Hide

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")

select(flights, one_of(vars) )
```

```
## # A tibble: 336,776 x 5
##     year month   day dep_delay arr_delay
##    <int> <int> <int>     <dbl>     <dbl>
##  1  2013     1     1         2        11
##  2  2013     1     1         4        20
##  3  2013     1     1         2        33
##  4  2013     1     1        -1       -18
##  5  2013     1     1        -6       -25
##  6  2013     1     1        -4        12
##  7  2013     1     1        -5        19
##  8  2013     1     1        -3       -14
##  9  2013     1     1        -3        -8
## 10  2013     1     1        -2         8
## # ... with 336,766 more rows
```

one_of(), when used in select(), makes a table of the variables in the vector vars found in flights.

4. Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

Hide

```
select(flights, contains("TIME"))
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time
##       <int>          <int>    <int>          <int>    <dbl>
##  1      517            515      830            819      227
##  2      533            529      850            830      227
##  3      542            540      923            850      160
##  4      544            545     1004           1022      183
##  5      554            600      812            837      116
##  6      554            558      740            728      150
##  7      555            600      913            854      158
##  8      557            600      709            723       53
##  9      557            600      838            846      140
## 10      558            600      753            745      138
## # ... with 336,766 more rows, and 1 more variables: time_hour <dttm>
```

This command makes sense-any variable in flights containing the word TIME (case insensitive) is part of a new table.

Exercise 5.5.2

1. Currently dep_time and sched_dep_time are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

Hide

```
transmute(flights,
  dep_time_since_midnight = (dep_time %% 100) + ((dep_time %/% 100) * 60),
  sched_dep_time_since_midnight = (sched_dep_time %% 100) + ((sched_dep_time %/% 100) * 60)
  )
```

```
## # A tibble: 336,776 x 2
##    dep_time_since_midnight sched_dep_time_since_midnight
##                      <dbl>                         <dbl>
##  1                     317                           315
##  2                     333                           329
##  3                     342                           340
##  4                     344                           345
##  5                     354                           360
##  6                     354                           358
##  7                     355                           360
##  8                     357                           360
##  9                     357                           360
## 10                     358                           360
## # ... with 336,766 more rows
```

2. Compare air_time with arr_time - dep_time. What do you expect to see? What do you see? What do you need to do to fix it?

I expect to see air_time to equal arr_time - dep_time

Hide

```
transmute(flights,
          air_time,
          tmp = arr_time - dep_time
          )
```

```
## # A tibble: 336,776 x 2
##    air_time   tmp
##       <dbl> <int>
##  1      227   313
##  2      227   317
##  3      160   381
##  4      183   460
##  5      116   258
##  6      150   186
##  7      158   358
##  8       53   152
##  9      140   281
## 10      138   195
## # ... with 336,766 more rows
```

Actually, arr_time and dep_time is in different units than air_time, so let me convert.

Hide

```
transmute(flights,
          air_time,
          arr_minutes = (arr_time %% 100) + ((arr_time %/% 100) * 60),
          dep_minutes = (dep_time %% 100) + ((dep_time %/% 100) * 60),
          arr_dep_minutes_diff = arr_minutes - dep_minutes
          )
```

```
## # A tibble: 336,776 x 4
##    air_time arr_minutes dep_minutes arr_dep_minutes_diff
##       <dbl>       <dbl>       <dbl>                <dbl>
##  1      227         510         317                  193
##  2      227         530         333                  197
##  3      160         563         342                  221
##  4      183         604         344                  260
##  5      116         492         354                  138
##  6      150         460         354                  106
##  7      158         553         355                  198
##  8       53         429         357                   72
##  9      140         518         357                  161
## 10      138         473         358                  115
## # ... with 336,766 more rows
```

Interesting-arr_dep_minutes_diff should be the same but it's a bit inflated compared to air_time. There might be a disconnect between actual air time and the logging of departure and arrival times.

3. Compare dep_time, sched_dep_time, and dep_delay. How would you expect those three numbers to be related?

I would expect dep_delay = dep_time - sched_dep_time

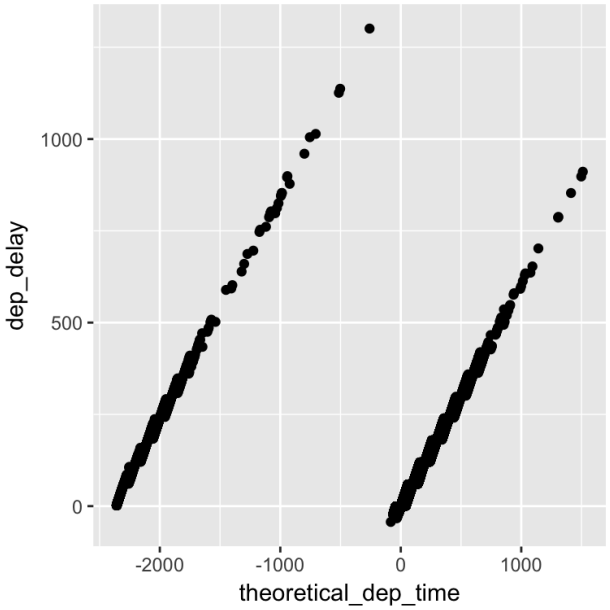
Hide

```
transmute(flights,
          dep_delay,
          theoretical_dep_time = dep_time - sched_dep_time)
```

```
## # A tibble: 336,776 x 2
##    dep_delay theoretical_dep_time
##        <dbl>                <int>
##  1         2                    2
##  2         4                    4
##  3         2                    2
##  4        -1                   -1
##  5        -6                  -46
##  6        -4                   -4
##  7        -5                  -45
##  8        -3                  -43
##  9        -3                  -43
## 10        -2                  -42
## # ... with 336,766 more rows
```

Hide

```
tmp <- transmute(flights,
          dep_delay,
          theoretical_dep_time = dep_time - sched_dep_time)

ggplot( data = tmp, mapping = aes( x = theoretical_dep_time, y = dep_delay)) +
  geom_point()
```



Looks like the early departures deviate more than those that are actual delays. Very strange looking graph-there's like a whole other line at extremely early departures.

4. Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for min_rank().

Hide

```
mutate(flights,
       dep_delay_rank = min_rank(-dep_delay)) %>%
  arrange(dep_delay_rank)
```

```
## # A tibble: 336,776 x 20
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     9      641            900      1301     1242
##  2  2013     6    15     1432           1935      1137     1607
##  3  2013     1    10     1121           1635      1126     1239
##  4  2013     9    20     1139           1845      1014     1457
##  5  2013     7    22      845           1600      1005     1044
##  6  2013     4    10     1100           1900       960     1342
##  7  2013     3    17     2321            810       911      135
##  8  2013     6    27      959           1900       899     1236
##  9  2013     7    22     2257            759       898      121
## 10  2013    12     5      756           1700       896     1058
## # ... with 336,766 more rows, and 13 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, dep_delay_rank <int>
```

5. What does 1:3 + 1:10 return? Why?

Hide

```
1:3
```

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

Hide

```
1:10
```

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

Hide

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

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

Because 1:3 is short than 1:10, 1:3 is recycled and it goes 1+1,2+2,3+3,1+4,2+5,3+6, etc.

6. What trigonometric functions does R provide?

R provides:

cos(x) sin(x) tan(x)

acos(x) asin(x) atan(x) atan2(y, x)

cospi(x) sinpi(x) tanpi(x)

From ?Trig, They respectively compute the cosine, sine, tangent, arc-cosine, arc-sine, arc-tangent, and the two-argument arc-tangent.

Exercise 5.6.7

1. Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights. Consider the following scenarios. Which is more important: arrival delay or departure delay?

- A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.
- A flight is always 10 minutes late.
- A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.
- 99% of the time a flight is on time. 1% of the time it's 2 hours late.

I would say arrival delay is more important-if I depart late but arrive on time that's ok but I usually like to adhere to adhere to when my arrival is supposed to be.

So flights can have an arrival delay that modulates the departure and thus that delay, it can also change consecutive flight arrivals/departures. The delay characteristics can vary e.g. large and small delays, or can be consistent e.g. typical amount of delay. The scenarios above vary from having small to large amounts of delays but either either have small variation or are skewed like the last scenario. I would also look to see if the delay of a flight correlates with a delay in a consecutive flight, which airlines attribute to the most delays and even if there's an airport with more or loss delays, especially if one is cchoosing to fly into say LGA or JFK or Newark in the NYC area, which can advise people on choosing flights. Also, is the arrival delay correlative with the departure delay?

2. Come up with another approach that will give you the same output as not_cancelled %>% count(dest) and not_cancelled %>% count(tailnum, wt = distance) (without using count()).

Hide

```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>% count(dest)
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
##  1   ABQ   254
##  2   ACK   264
##  3   ALB   418
##  4   ANC     8
##  5   ATL 16837
##  6   AUS  2411
##  7   AVL   261
##  8   BDL   412
##  9   BGR   358
## 10   BHM   269
## # ... with 94 more rows
```

Hide

```
not_cancelled %>% count(tailnum, wt = distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##      <chr>  <dbl>
##  1  D942DN   3418
##  2  N0EGMQ 239143
##  3  N10156 109664
##  4  N102UW  25722
##  5  N103US  24619
##  6  N104UW  24616
##  7  N10575 139903
##  8  N105UW  23618
##  9  N107US  21677
## 10  N108UW  32070
## # ... with 4,027 more rows
```

I can group by the airport (dest) and sum the amount of non-na values instead of using count().

Hide

```
not_cancelled %>%
  group_by(dest) %>%
  summarise(n = sum(!is.na(dest)))
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
##  1   ABQ   254
##  2   ACK   264
##  3   ALB   418
##  4   ANC     8
##  5   ATL 16837
##  6   AUS  2411
##  7   AVL   261
##  8   BDL   412
##  9   BGR   358
## 10   BHM   269
## # ... with 94 more rows
```

Since count() is a shorthand for group_by() and tally(), I can just break up the command.

Hide

```
not_cancelled %>%
  group_by(tailnum) %>%
tally(wt = distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##      <chr>  <dbl>
##  1  D942DN   3418
##  2  N0EGMQ 239143
##  3  N10156 109664
##  4  N102UW  25722
##  5  N103US  24619
##  6  N104UW  24616
##  7  N10575 139903
##  8  N105UW  23618
##  9  N107US  21677
## 10  N108UW  32070
## # ... with 4,027 more rows
```

3. Our definition of cancelled flights (is.na(dep_delay) | is.na(arr_delay)) is slightly suboptimal. Why? Which is the most important column?

There could be flights that depart but never arrive, which could mean a cancelled flight. However, a flight that never departs will never arrive. I would say arr_delay would be more important-if a flight did arrive it will depart but not true the other way around. For making an optimal cancelled flight table, I would just do

Hide

```
flights %>%
  filter(is.na(arr_delay))
```

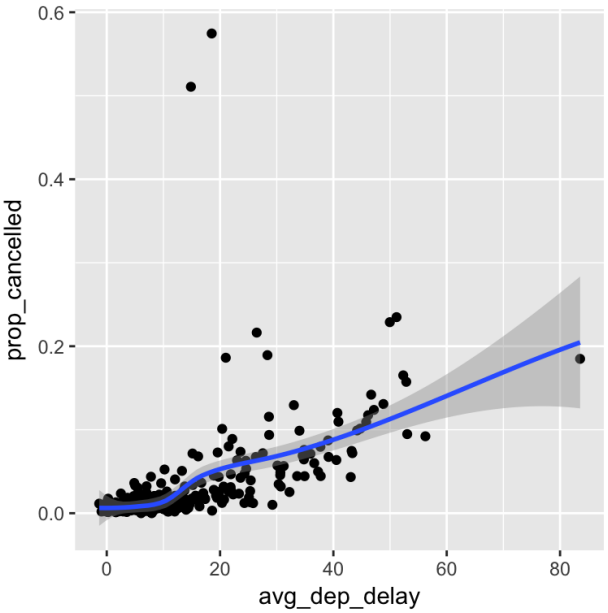
```
## # A tibble: 9,430 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1     1525           1530        -5     1934
##  2  2013     1     1     1528           1459        29     2002
##  3  2013     1     1     1740           1745        -5     2158
##  4  2013     1     1     1807           1738        29     2251
##  5  2013     1     1     1939           1840        59       29
##  6  2013     1     1     1952           1930        22     2358
##  7  2013     1     1     2016           1930        46       NA
##  8  2013     1     1       NA           1630        NA       NA
##  9  2013     1     1       NA           1935        NA       NA
## 10  2013     1     1       NA           1500        NA       NA
## # ... with 9,420 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

The extra is.na(dep_delay) isn't necessary.

4. Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

Hide

```
cancelled_delayed <-
  flights %>%
  mutate(cancelled = (is.na(arr_delay) | is.na(dep_delay))) %>%
  group_by(year, month, day) %>%
  summarise(prop_cancelled = mean(cancelled),
            avg_dep_delay = mean(dep_delay, na.rm = TRUE))

ggplot(cancelled_delayed, aes(x = avg_dep_delay, prop_cancelled)) +
  geom_point() +
  geom_smooth()
```



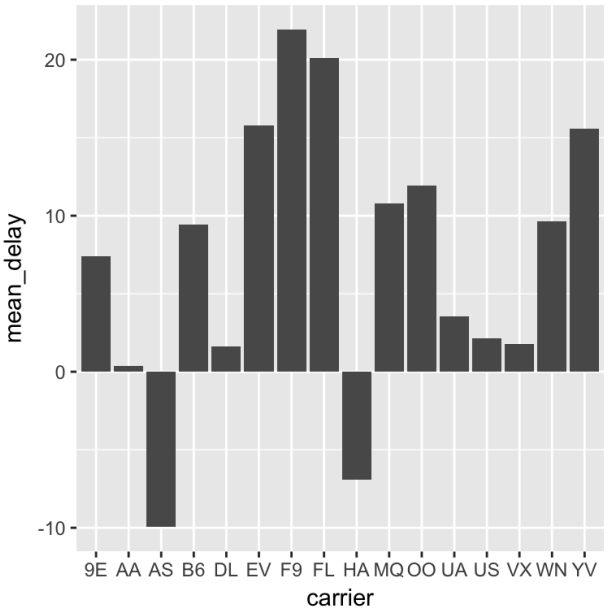
As the average dep_delay increases, the proportion of cancelled flights vary more and more. Also, less and less flights have large delays, with all but one under 60 minutes. With knowing that,the variation in the proportion of cancelled flights seems more or less constant.

5. Which carrier has the worst delays? Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about flights %>% group_by(carrier, dest) %>% summarise(n()))

So we want to group by carrier. And since we want to take into account the airport (dest), we can group by dest and carrier.

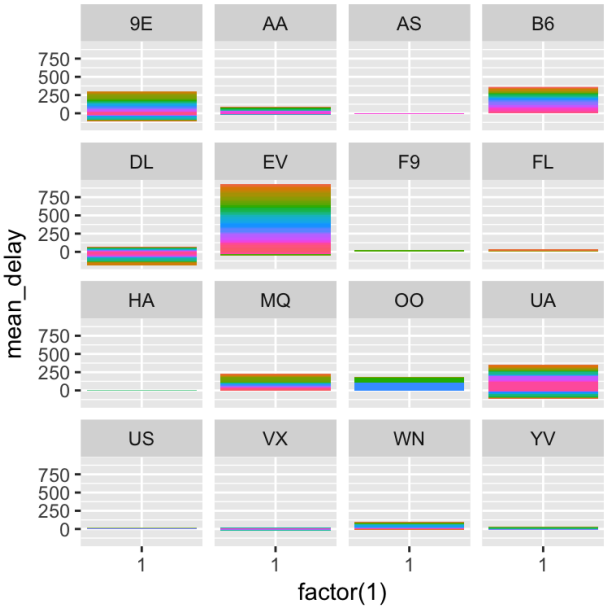
Hide

```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

tmp <- not_cancelled %>% group_by(carrier) %>% summarise( "mean_delay" = mean(arr_delay))

ggplot( data = tmp , mapping = aes(x = carrier, y = mean_delay) ) +
  geom_bar(stat="identity")
```



Hide

```
tmp <- not_cancelled %>% group_by(carrier,dest) %>% summarise( "mean_delay" = mean(arr_delay))

ggplot( data = tmp , mapping = aes(x = factor(1), y = mean_delay, fill = dest) ) +
  geom_bar(stat="identity") +
  facet_wrap(~carrier) +
  theme(
    legend.position = "none"
  )
```

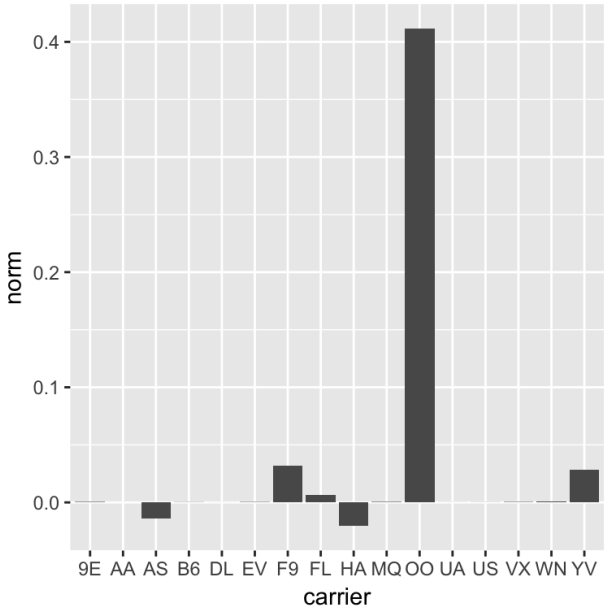


The barplot shows the mean delay time across all carriers, clearly showing those with more average delays where others have early arrivals.

We see that some carriers are at a lot more aiports. We should take that into account.

Hide

```
tmp <- not_cancelled %>%
  group_by(carrier) %>%
  summarise( "mean_delay" = mean(arr_delay),
             "num_flights" = n(),
             "norm" = mean_delay / num_flights)
ggplot( data = tmp , mapping = aes(x = carrier, y = norm) ) +
  geom_bar(stat="identity")
```



There are multiple and better ways to disentangle worse airport/carrier delays. See http://fivethirtyeight.com/features/the-best-and-worst-airlines-airports-and-flights-summer-2015-update/ (http://fivethirtyeight.com/features/the-best-and-worst-airlines-airports-and-flights-summer-2015-update/)

6. What does the sort argument to count() do. When might you use it?

Hide

```
not_cancelled %>% count(dest)
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
##  1   ABQ   254
##  2   ACK   264
##  3   ALB   418
##  4   ANC     8
##  5   ATL 16837
##  6   AUS  2411
##  7   AVL   261
##  8   BDL   412
##  9   BGR   358
## 10   BHM   269
## # ... with 94 more rows
```

Hide

```
not_cancelled %>% count(dest,sort=T)
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
##  1   ATL 16837
##  2   ORD 16566
##  3   LAX 16026
##  4   BOS 15022
##  5   MCO 13967
##  6   CLT 13674
##  7   SFO 13173
##  8   FLL 11897
##  9   MIA 11593
## 10   DCA  9111
## # ... with 94 more rows
```

The sort argument of count() sorts from largest to smallest counts. One might use this to view the highest count of something among the counts of others.

Exercise 5.7.1

1. Refer back to the lists of useful mutate and filtering functions. Describe how each operation changes when you combine it with grouping.

When you combine the mutate and filtering functions with groupings, you operate on the grouped data as opposed to the entire data frame.

2. Which plane (tailnum) has the worst on-time record?

Hide

```
not_cancelled %>%
  group_by(tailnum) %>%
  summarise(arr_delay = mean(arr_delay, na.rm = TRUE)) %>%
  ungroup() %>%
  filter(rank(desc(arr_delay)) <= 1)
```

```
## # A tibble: 1 x 2
##   tailnum arr_delay
##     <chr>     <dbl>
## 1  N844MH       320
```

3. What time of day should you fly if you want to avoid delays as much as possible?

Hide

```
not_cancelled %>%
  group_by(hour) %>%
  summarise( mean_arr_delay = mean(arr_delay, na.rm=T) ) %>%
  ungroup() %>%
  arrange(mean_arr_delay)
```

```
## # A tibble: 19 x 2
##     hour mean_arr_delay
##    <dbl>          <dbl>
##  1     7     -5.3044716
##  2     5     -4.7969072
##  3     6     -3.3844854
##  4     9     -1.4514074
##  5     8     -1.1132266
##  6    10      0.9539401
##  7    11      1.4819300
##  8    12      3.4890104
##  9    13      6.5447397
## 10    14      9.1976501
## 11    23     11.7552783
## 12    15     12.3241920
## 13    16     12.5976412
## 14    18     14.7887244
## 15    22     15.9671618
## 16    17     16.0402670
## 17    19     16.6558736
## 18    20     16.6761098
## 19    21     18.3869371
```

You want to leave as early as possible.

4. For each destination, compute the total minutes of delay. For each, flight, compute the proportion of the total delay for its destination.

Hide

```
not_cancelled %>%
  group_by(dest) %>%
  summarise( sum_arr_delay = sum(arr_delay) )
```

```
## # A tibble: 104 x 2
##     dest sum_arr_delay
##    <chr>         <dbl>
##  1   ABQ          1113
##  2   ACK          1281
##  3   ALB          6018
##  4   ANC           -20
##  5   ATL        190260
##  6   AUS         14514
##  7   AVL          2089
##  8   BDL          2904
##  9   BGR          2874
## 10   BHM          4540
## # ... with 94 more rows
```

Hide

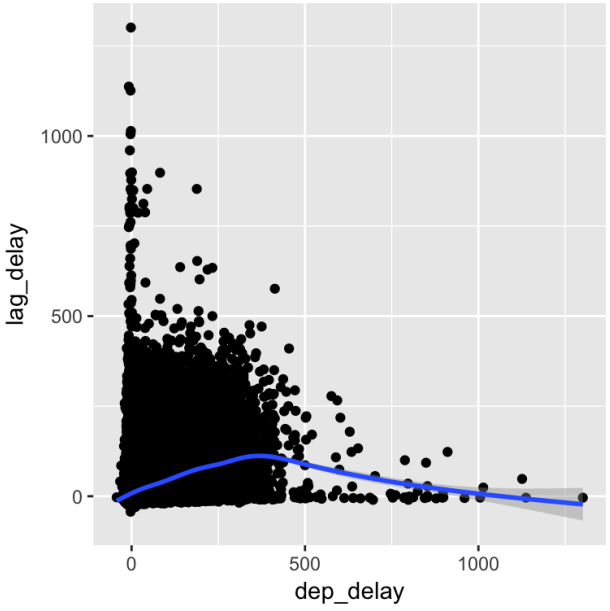
```
not_cancelled %>%
  group_by(flight) %>%
  transmute( prop_arr_delay = arr_delay / sum(arr_delay) )
```

```
## # A tibble: 327,346 x 2
## # Groups:   flight [3,835]
##    flight prop_arr_delay
##     <int>          <dbl>
##  1   1545    0.186440678
##  2   1714    0.024183797
##  3   1141    0.036789298
##  4    725   -0.193548387
##  5    461    0.010521886
##  6   1696   -0.122448980
##  7    507    0.005950517
##  8   5708   -0.121739130
##  9     79    0.021390374
## 10    301   -0.002056555
## # ... with 327,336 more rows
```

5. Delays are typically temporally correlated: even once the problem that caused the initial delay has been resolved, later flights are delayed to allow earlier flights to leave. Using lag() explore how the delay of a flight is related to the delay of the immediately preceding flight.

Hide

```
flights %>%
  group_by(year, month, day) %>%
  filter(!is.na(dep_delay)) %>%
  mutate(lag_delay = lag(dep_delay)) %>%
  filter(!is.na(lag_delay)) %>%
  ggplot(aes(x = dep_delay, y = lag_delay)) +
  geom_point() +
  geom_smooth()
```



6. Look at each destination. Can you find flights that are suspiciously fast? (i.e. flights that represent a potential data entry error). Compute the air time a flight relative to the shortest flight to that destination. Which flights were most delayed in the air?

I also computed a observed vs expected ratio using the median air time. I'm sure there's better ways to find outliers.

Hide

```
flights %>%
  filter(!is.na(air_time)) %>%
  group_by(dest) %>%
  mutate( med_air_time = median(air_time),
          o_vs_e = (air_time - med_air_time) / med_air_time,
          air_time_diff = air_time - min(air_time) ) %>%
  arrange(desc(air_time_diff)) %>%
  select(air_time, o_vs_e, air_time_diff, dep_time, sched_dep_time, arr_time, sched_arr_time) %>%
  head(15)
```

```
## # A tibble: 15 x 8
## # Groups:   dest [8]
##     dest air_time    o_vs_e air_time_diff dep_time sched_dep_time arr_time
##    <chr>    <dbl>     <dbl>         <dbl>    <int>          <int>    <int>
##  1   SFO      490 0.4202899           195     1727           1730     2242
##  2   LAX      440 0.3414634           165     1812           1815     2302
##  3   EGE      382 0.5098814           163     1806           1700     2253
##  4   DEN      331 0.4711111           149     1513           1507     1914
##  5   LAX      422 0.2865854           147     1814           1815     2240
##  6   LAS      399 0.3255814           143     2142           1729      143
##  7   SFO      438 0.2695652           143     1727           1730     2206
##  8   SAN      413 0.2707692           134     1646           1620     2107
##  9   HNL      695 0.1282468           133     1337           1335     1937
## 10   SFO      426 0.2347826           131     1746           1745     2225
## 11   SNA      405 0.2310030           131     1820           1819     2242
## 12   LAX      405 0.2347561           130     1814           1815     2240
## 13   LAX      404 0.2317073           129     1655           1655     2110
## 14   HNL      691 0.1217532           129      853            900     1542
## 15   LAX      403 0.2286585           128     1113           1112     1515
## # ... with 1 more variables: sched_arr_time <int>
```

7. Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.

Hide

```
not_cancelled %>%
  group_by(dest, carrier) %>%
  count(carrier) %>%
  filter(n >= 2) %>%
  group_by(carrier) %>%
  count(sort = TRUE)
```

```
## # A tibble: 16 x 2
## # Groups:   carrier [16]
##    carrier    nn
##      <chr> <int>
##  1      EV    60
##  2      9E    45
##  3      UA    43
##  4      B6    42
##  5      DL    36
##  6      MQ    20
##  7      AA    19
##  8      WN    11
##  9      US     5
## 10      VX     4
## 11      FL     3
## 12      OO     3
## 13      YV     3
## 14      AS     1
## 15      F9     1
## 16      HA     1
```

8. For each plane, count the number of flights before the first delay of greater than 1 hour.

Hide

```
not_cancelled %>%
  group_by(tailnum) %>%
  mutate(delay_gt1hr = dep_delay > 60) %>%
  mutate(before_delay = cumsum(delay_gt1hr)) %>%
  filter(before_delay < 1) %>%
  count(sort = TRUE)
```

```
## # A tibble: 3,818 x 2
## # Groups:   tailnum [3,818]
##    tailnum     n
##      <chr> <int>
##  1  N952UW   214
##  2  N315NB   160
##  3  N705TW   159
##  4  N706TW   149
##  5  N961UW   139
##  6  N346NB   127
##  7  N713TW   127
##  8  N765US   122
##  9  N721TW   119
## 10   N744P   118
## # ... with 3,808 more rows
```

6. Workflow: Scripts

Exercise 6.3

1. Go to the RStudio Tips twitter account, https://twitter.com/rstudiotips (https://twitter.com/rstudiotips) and find one tip that looks interesting. Practice using it!

This is a good one from https://twitter.com/vuorre/status/869545189056053248 (https://twitter.com/vuorre/status/869545189056053248): Cmd+Option+o collapses all sections in your Rmarkdown document-pretty useful!

2. What other common mistakes will RStudio diagnostics report? Read https://support.rstudio.com/hc/en-us/articles/205753617-Code-Diagnostics (https://support.rstudio.com/hc/en-us/articles/205753617-Code-Diagnostics) to find out.

Very useful doc-must read!

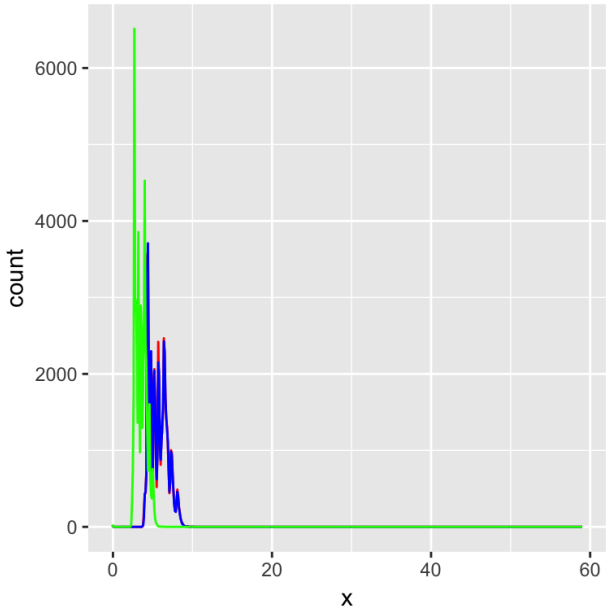
7. Exploratory Data Analysis

Exercise 7.3.4

1. Explore the distribution of each of the x, y, and z variables in diamonds. What do you learn? Think about a diamond and how you might decide which dimension is the length, width, and depth.

Hide

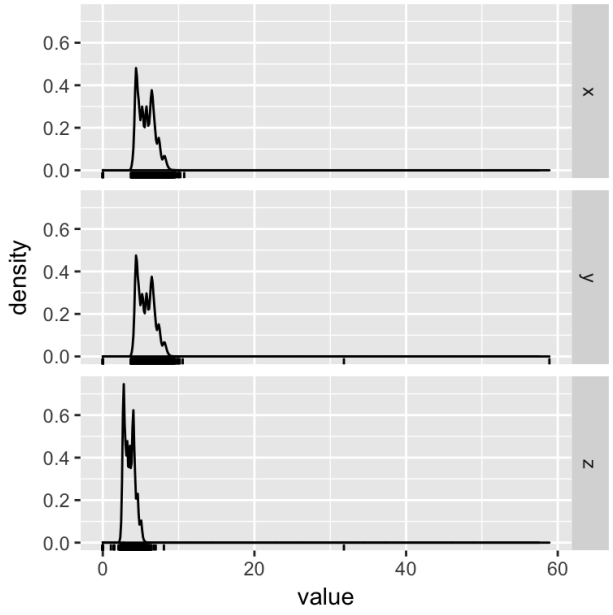
```
library(tidyverse)

ggplot( data = diamonds ) +
  geom_freqpoly(binwidth=0.1,aes(x = x ), color = "red") +
  geom_freqpoly(binwidth=0.1,aes(x = y ), color = "blue") +
  geom_freqpoly(binwidth=0.1,aes(x = z ), color  ="green")
```



Looks like the y and z axis have similar value frequencies, and the z axis has more smaller values, which I think might represent the depth of diamonds. I realy like the graph representation from https://jrnold.github.io/e4qf/exploratory-data-analysis.html (https://jrnold.github.io/e4qf/exploratory-data-analysis.html) so I'm plotting it here for reference.

Hide

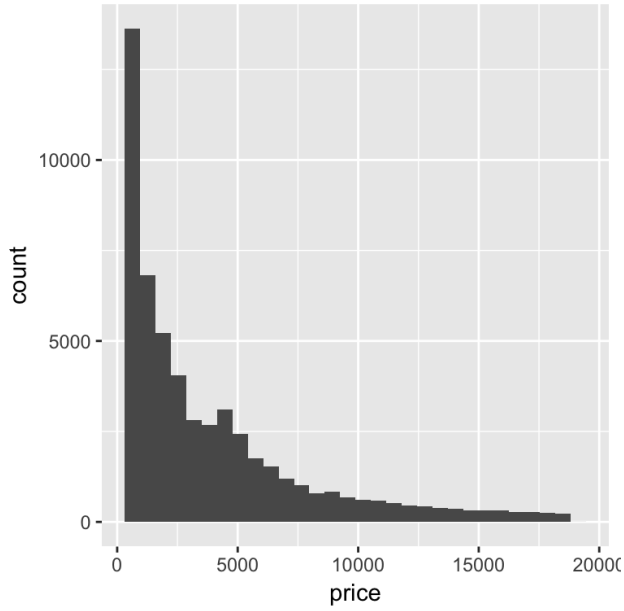
```
diamonds %>%
  mutate(id = row_number()) %>%
  select(x, y, z, id) %>%
  gather(variable, value, -id)  %>%
  ggplot(aes(x = value)) +
  geom_density() +
  geom_rug() +
  facet_grid(variable ~ .)
```



2. Explore the distribution of price. Do you discover anything unusual or surprising? (Hint: Carefully think about the binwidth and make sure you try a wide range of values.)

Hide

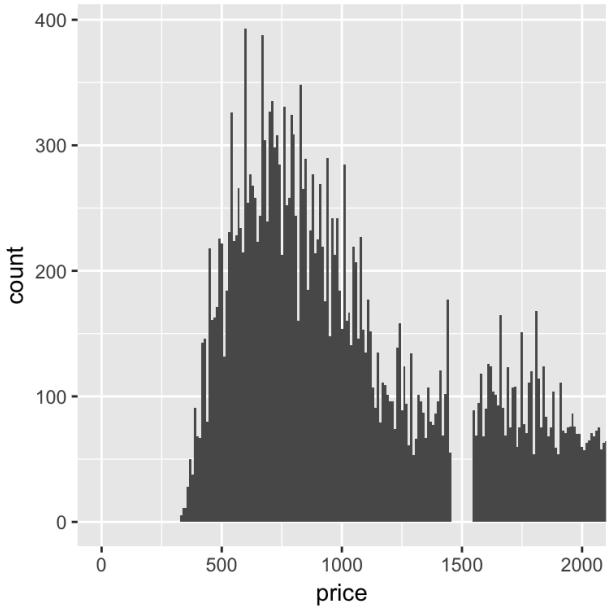
```
ggplot( data = diamonds , aes( x = price ) ) +
  geom_histogram()
```



There is a skew towards 0 then it tails off to very high prices. But why are diamonds priced so low?

Hide

```
ggplot( data = diamonds , aes( x = price ) ) +
  geom_histogram(binwidth=10) +
  coord_cartesian(xlim=c(0,2000))
```

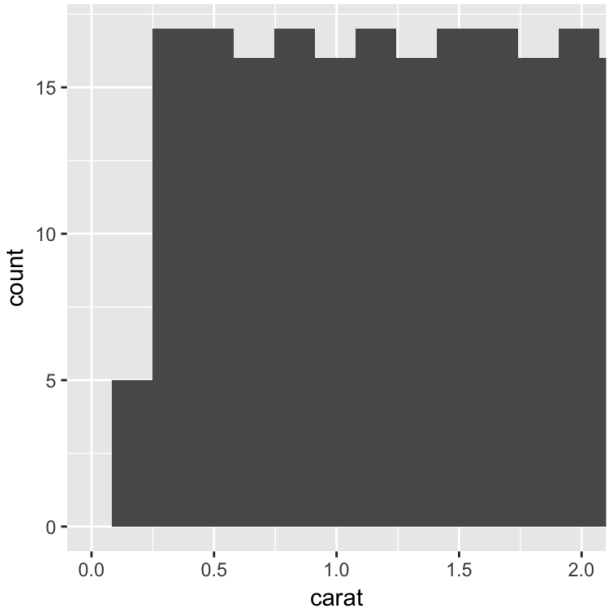


Actually the minimum price is between 250 and 300. Then there's no diamonds priced around 1500. Wow, with more domain expertise and interesting questions we can learn a lot more about this data!

3. How many diamonds are 0.99 carat? How many are 1 carat? What do you think is the cause of the difference?

Hide

```
diamonds %>% group_by(carat) %>% count() %>%
  ggplot() +
  geom_histogram( aes( x = carat ) ) +
  coord_cartesian(xlim=c(0,2))
```



Hide

```
filter(diamonds,carat==0.99) %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    23
```

Hide

```
filter(diamonds,carat==1) %>% count()
```

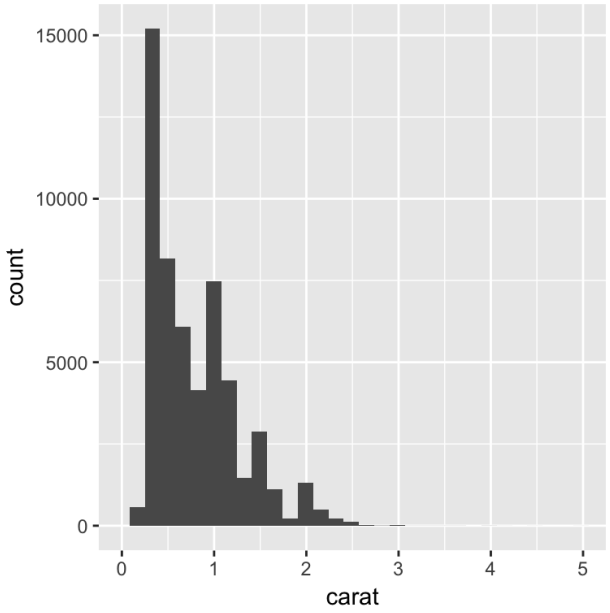
```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  1558
```

There's so many more at 1 carat-maybe makes sense since I wouldn't be asking for a .99 carat diamond. I'll ask for a nice round 1!

4. Compare and contrast coord_cartesian() vs xlim() or ylim() when zooming in on a histogram. What happens if you leave binwidth unset? What happens if you try and zoom so only half a bar shows?

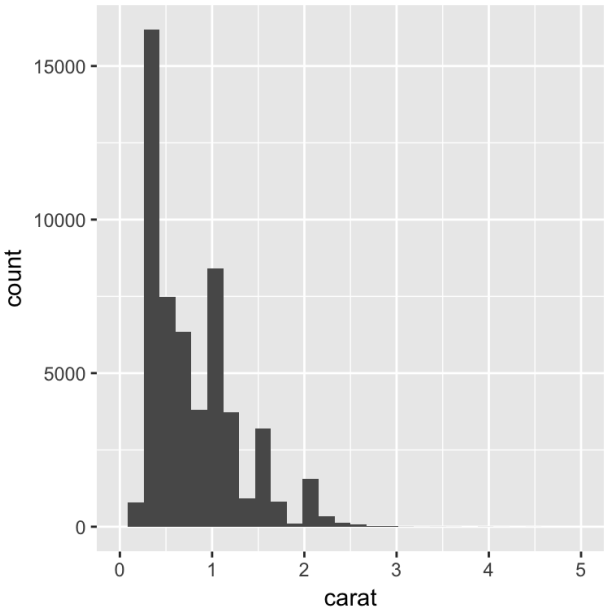
Hide

```
diamonds %>%
  ggplot() +
  geom_histogram( aes( x = carat ) ) +
  coord_cartesian(xlim=c(0,5))
```



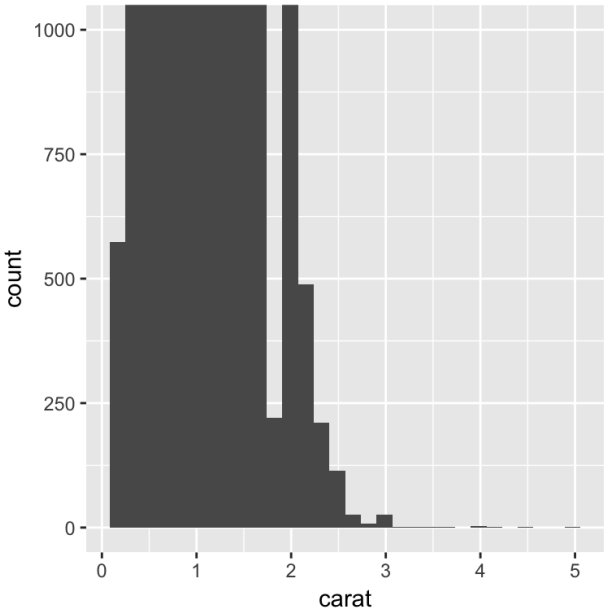
Hide

```
diamonds %>%
  ggplot() +
  geom_histogram( aes( x = carat ) ) +
  xlim( c(0,5) )
```



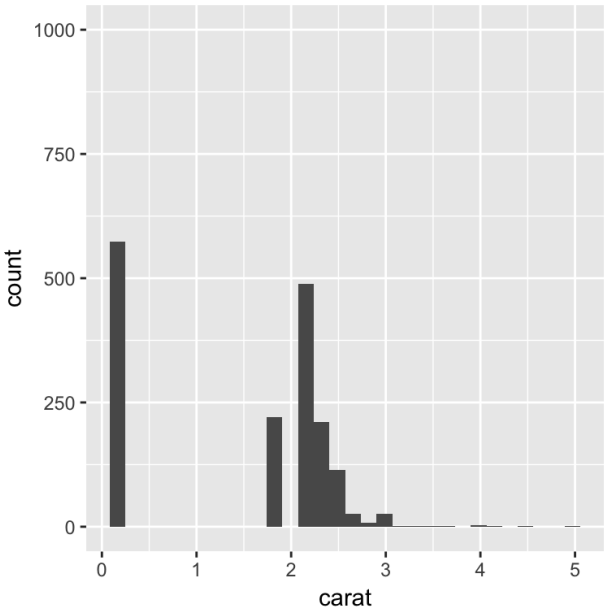
Hide

```
diamonds %>%
  ggplot() +
  geom_histogram( aes( x = carat ) ) +
  coord_cartesian(ylim=c(0,1000))
```



Hide

```
diamonds %>%
  ggplot() +
  geom_histogram( aes( x = carat ) ) +
  ylim( c(0,1000) )
```



That's weird. Using ylim vs. coord_cartesian cuts of the values to only include those with the max or lower bin height. For the x axis it's virtually the same but not sure why one row is removed...

Exercise 7.4.1

1. What happens to missing values in a histogram? What happens to missing values in a bar chart? Why is there a difference?

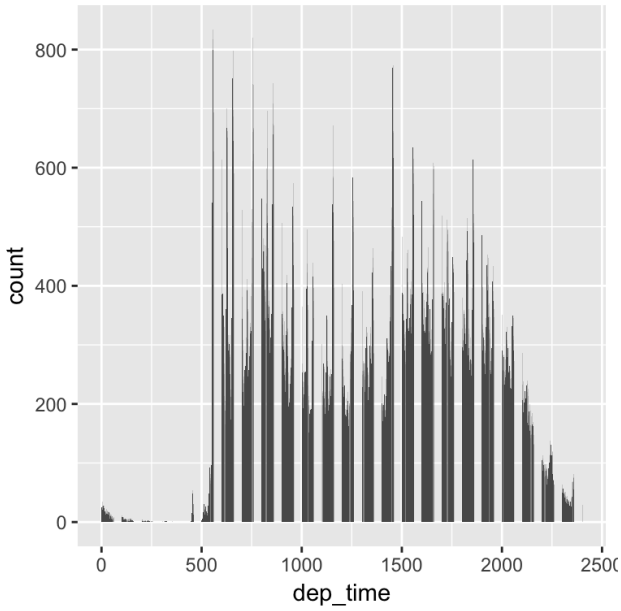
Hide

```
nycflights13::flights %>%
  count(is.na(dep_time)==TRUE)
```

```
## # A tibble: 2 x 2
##   `is.na(dep_time) == TRUE`      n
##                       <lgl>  <int>
## 1                     FALSE 328521
## 2                      TRUE   8255
```

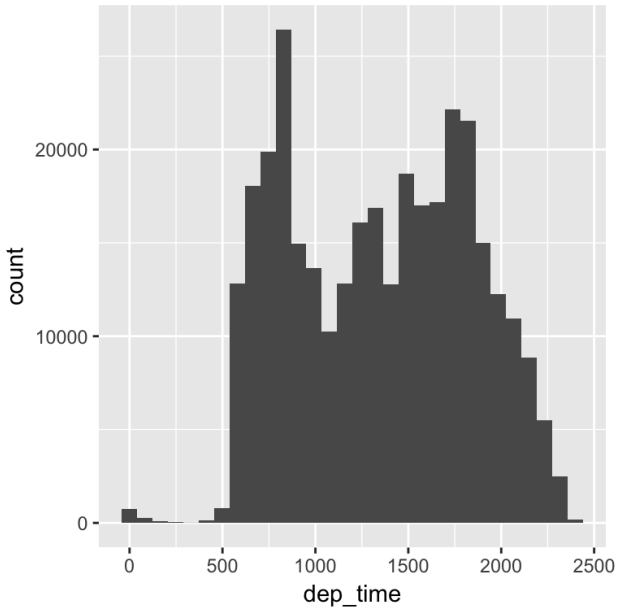
Hide

```
nycflights13::flights %>%
  ggplot() +
  geom_bar( aes(x = dep_time ) )
```



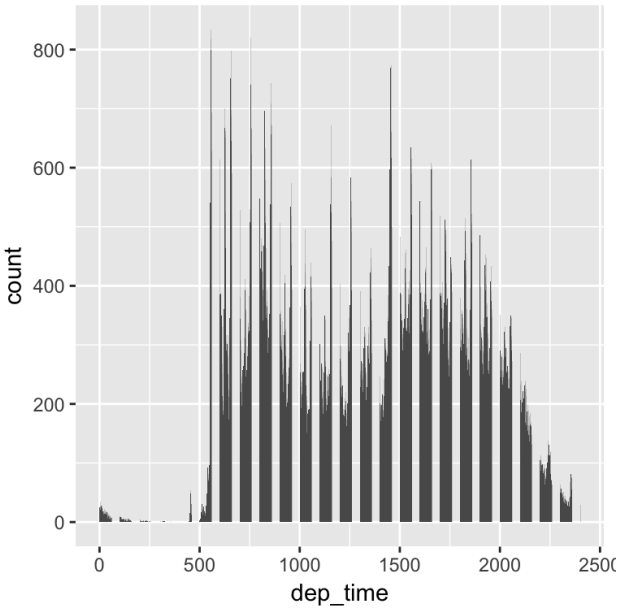
Hide

```
nycflights13::flights %>%
  ggplot() +
  geom_histogram( aes(x = dep_time ) )
```



Hide

```
nycflights13::flights %>%
  ggplot() +
  geom_histogram( binwidth=1,aes(x = dep_time ) )
```



I think there's a difference in the bar plot vs. histogram because the binwidth matters for showing that there are gaps present in the data. Trying out different bin widths might be helpful in exploring missingness in your data.

2. What does na.rm = TRUE do in mean() and sum()?

Hide

```
nycflights13::flights %>%
  group_by(dep_time) %>%
  count()
```

```
## # A tibble: 1,319 x 2
## # Groups:   dep_time [1,319]
##    dep_time     n
##       <int> <int>
##  1        1    25
##  2        2    35
##  3        3    26
##  4        4    26
##  5        5    21
##  6        6    22
##  7        7    22
##  8        8    23
##  9        9    28
## 10       10    22
## # ... with 1,309 more rows
```

Hide

```
nycflights13::flights %>%
  group_by(dep_time) %>%
  count() %>%
  sum()
```

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

Hide

```
nycflights13::flights %>%
  group_by(dep_time) %>%
  count() %>%
  sum(na.rm=TRUE)
```

```
## [1] 1995334
```

Seems like adding a na.rm=TRUE argument to sum (and mean) gets rid of NA values so that the sum and mean can be computed by R.

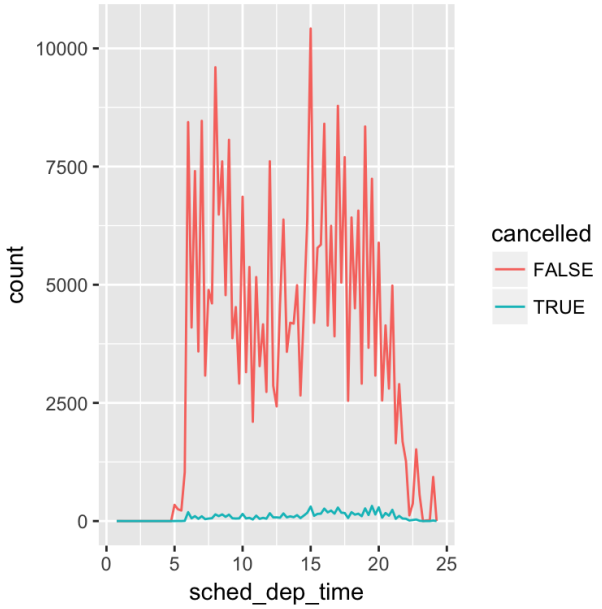
Exercise 7.5.1.1

1. Use what you've learned to improve the visualisation of the departure times of cancelled vs. non-cancelled flights.

Here's the original

Hide

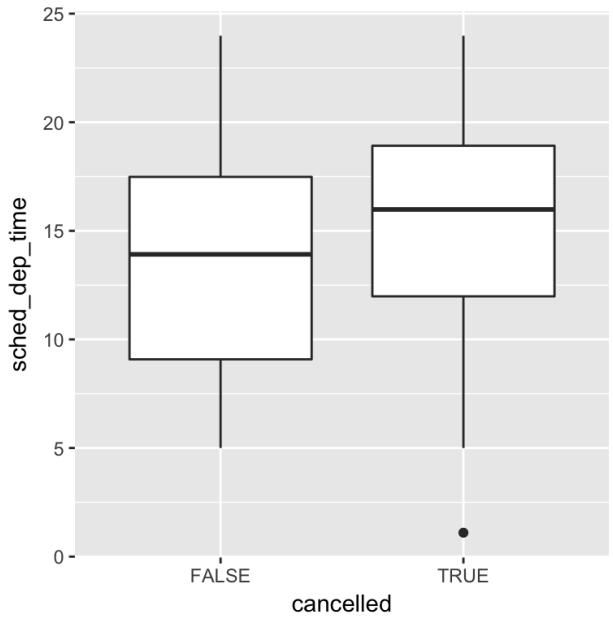
```
cancellation <- nycflights13::flights %>%
  mutate(
    cancelled = is.na(dep_time),
    sched_hour = sched_dep_time %/% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + sched_min / 60
  )

  ggplot( data = cancellation, mapping = aes(sched_dep_time)) +
    geom_freqpoly(mapping = aes(colour = cancelled), binwidth = 1/4)
```



So we see the cancelled flights are less frequent then not cancelled flights so it's hard to see the distribution. We can instead view it as a boxplot.

Hide

```
ggplot(data = cancellation , mapping = aes( x = cancelled , y = sched_dep_time) ) +
  geom_boxplot()
```



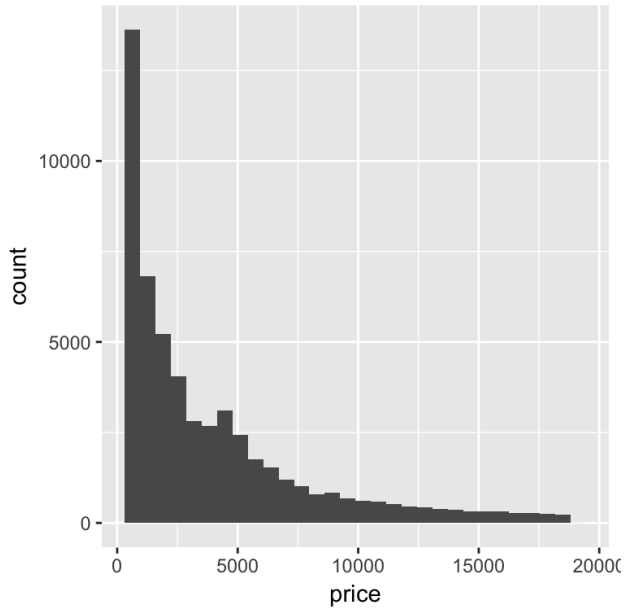
The distributions are more comparable now.

2. What variable in the diamonds dataset is most important for predicting the price of a diamond? How is that variable correlated with cut? Why does the combination of those two relationships lead to lower quality diamonds being more expensive?

So the question is asking a question regarding these

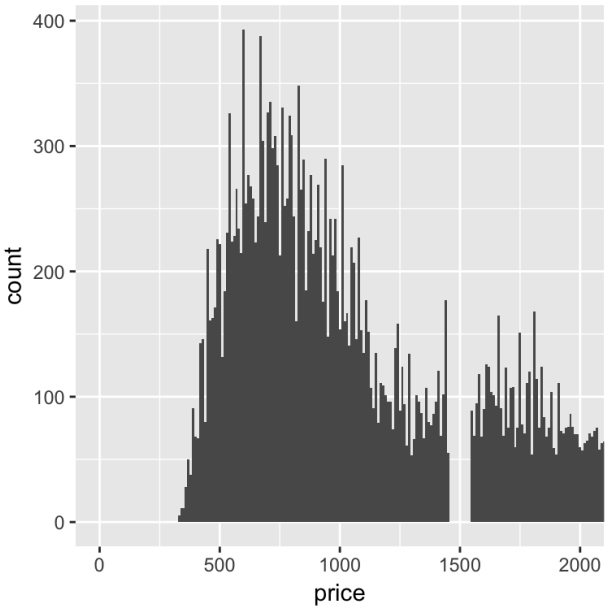
Hide

```
ggplot(data = diamonds, mapping = aes(x = price)) +
  geom_histogram()
```



Hide

```
ggplot( data = diamonds , aes( x = price ) ) +
  geom_histogram(binwidth=10) +
  coord_cartesian(xlim=c(0,2000))
```

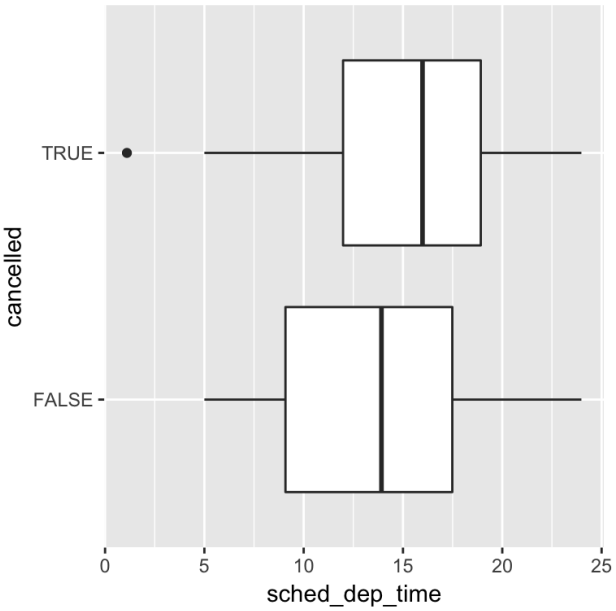


I assume the question is asking something about variable covariation or predictive value of a variable for proce, but approaches to tackle those problems haven't been addressed in the book thus far.

3. Install the ggstance package, and create a horizontal boxplot. How does this compare to using coord_flip()?

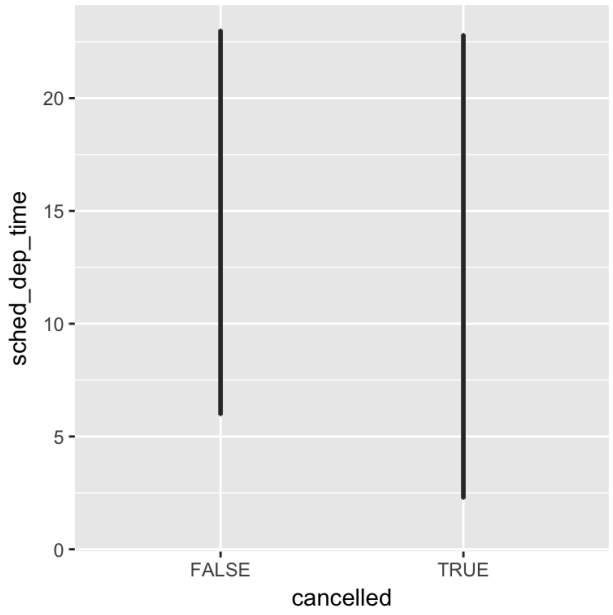
Hide

```
require(ggstance)
ggplot( data = cancellation ) +
  geom_boxplot( mapping = aes( x = cancelled , y = sched_dep_time ) ) +
  coord_flip()
```



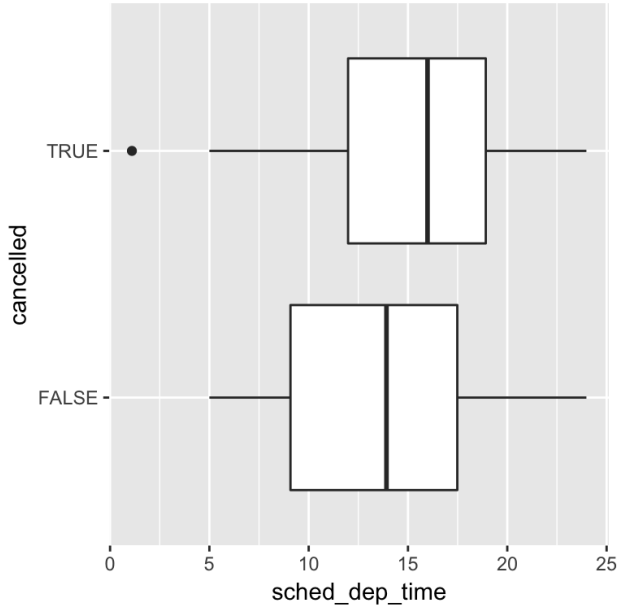
Hide

```
ggplot( data = cancellation ) +
  geom_boxploth( mapping = aes( x = cancelled , y = sched_dep_time ) )
```



Hide

```
ggplot( data = cancellation ) +
  geom_boxploth( mapping = aes( y = cancelled , x = sched_dep_time ) )
```

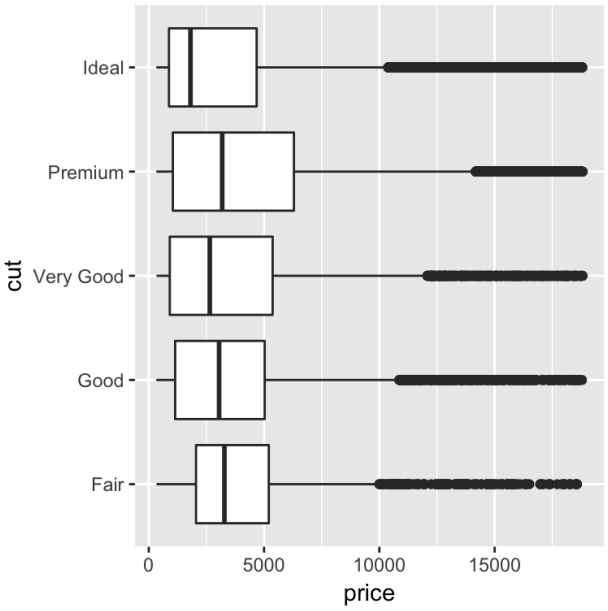


ggstance is a new package to me, so just using geom_boxploth it says we can't have overlapping y axes, but apparently I need to switch the x and y assignment to show the same thing. Hard to see the advantage with ggstance here except you save one line of code?

4. One problem with boxplots is that they were developed in an era of much smaller datasets and tend to display a prohibitively large number of "outlying values". One approach to remedy this problem is the letter value plot. Install the lvplot package, and try using geom_lv() to display the distribution of price vs cut. What do you learn? How do you interpret the plots?

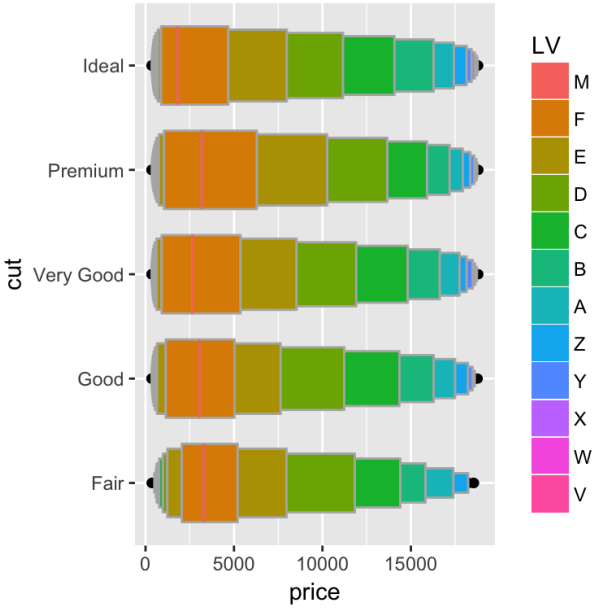
Hide

```
ggplot(diamonds,aes(x=cut,y=price))+
  geom_boxplot()+coord_flip()
```



Hide

```
require(lvplot)

ggplot(diamonds,aes(x=cut,y=price))+
  geom_lv(aes(fill=..LV..))+coord_flip()
```

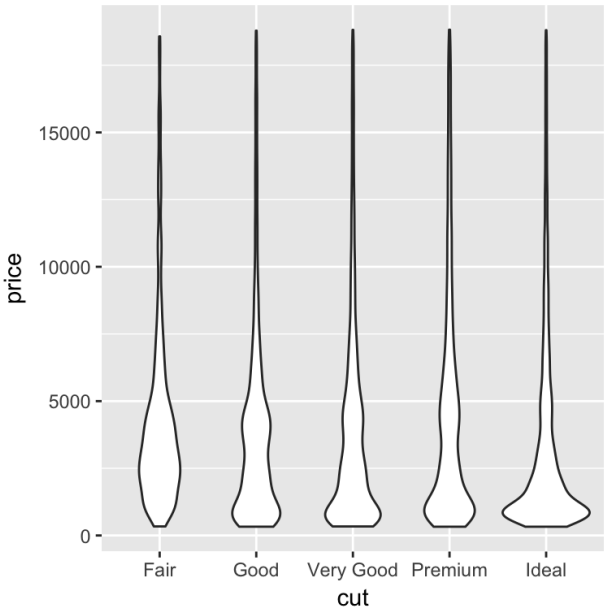


Very cool! Try ?geom_lv for the explanation of these distributions. The letter values correspond to quantiles of the data. So we see a better representation of the data quantiles than with regular boxplots. This is useful when we have lots of data.

5. Compare and contrast geom_violin() with a facetted geom_histogram(), or a coloured geom_freqpoly(). What are the pros and cons of each method?

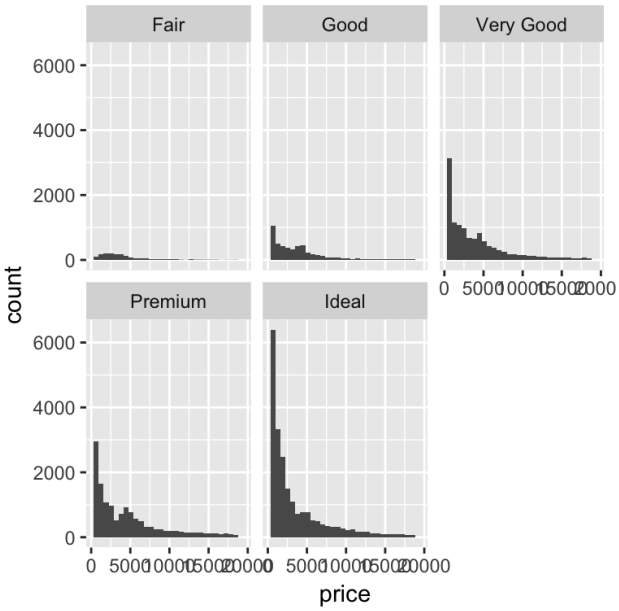
Hide

```
ggplot(diamonds,aes(x=cut,y=price))+
  geom_violin()
```



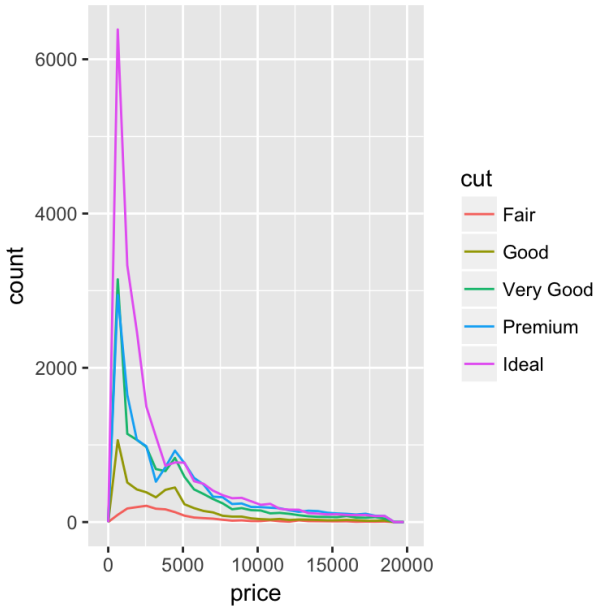
Hide

```
ggplot(diamonds,aes(x=price))+
  geom_histogram()+facet_wrap(~cut)
```



Hide

```
ggplot(diamonds,aes(color=cut,x=price))+
  geom_freqpoly()
```



Hmm. It's harder to compare the price distribution of different cuts with geom_histogram, geom_violin provides a better indication of the density of values in the distribution but I think geom_freqpoly provides a better way to compare and contrast the price distribution of different cuts.

6. If you have a small dataset, it's sometimes useful to use geom_jitter() to see the relationship between a continuous and categorical variable. The ggbeeswarm package provides a number of methods similar to geom_jitter(). List them and briefly describe what each one does.

Hide

```
require(ggbeeswarm)
```

geom_beeswarm-points jittered using the beeswarm package

geom_quasirandom-points jittered using the vipor package

ggbeeswarm-extends ggplot2 with violin point/beeswarm plots

position_beeswarm-Violin point-style plots to show overlapping points. x must be discrete.

position_quasirandom-Violin point-style plots to show overlapping points. x must be discrete.

Exercise 7.5.2.1

1. How could you rescale the count dataset above to more clearly show the distribution of cut within colour, or colour within cut?

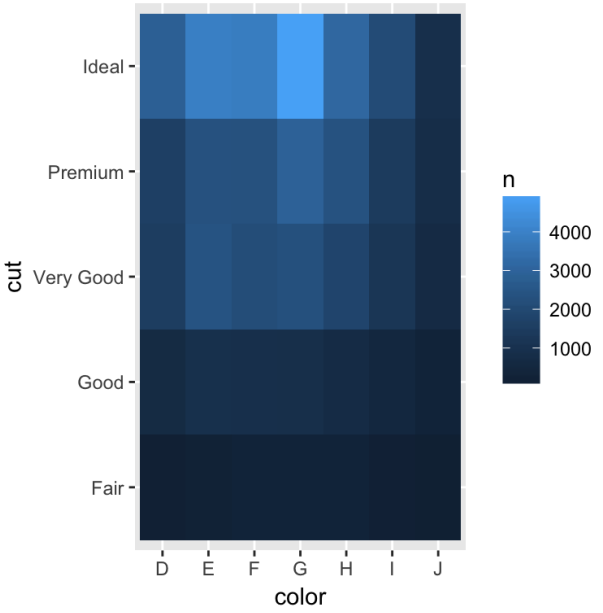
Hide

```
d <- diamonds %>%
  count(color, cut)

d
```

```
## # A tibble: 35 x 3
##    color       cut     n
##    <ord>     <ord> <int>
##  1     D      Fair   163
##  2     D      Good   662
##  3     D Very Good  1513
##  4     D   Premium  1603
##  5     D     Ideal  2834
##  6     E      Fair   224
##  7     E      Good   933
##  8     E Very Good  2400
##  9     E   Premium  2337
## 10     E     Ideal  3903
## # ... with 25 more rows
```

Hide

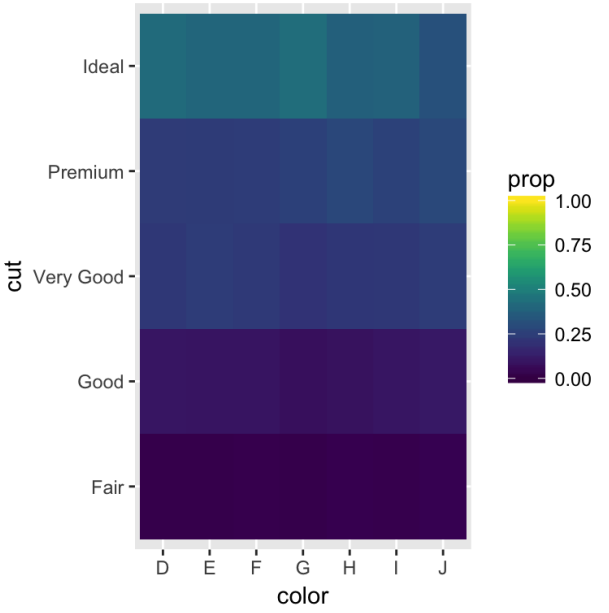
```
d %>%
  ggplot(mapping = aes(x = color, y = cut)) +
    geom_tile(mapping = aes(fill = n))
```



Per jarnold's answer you can do this by

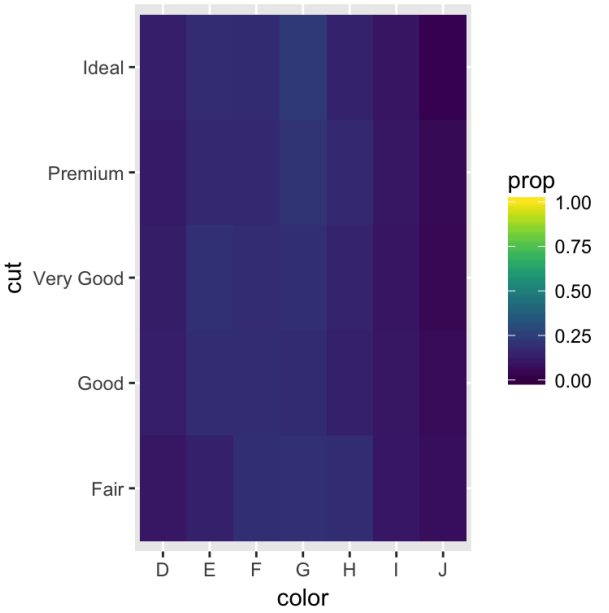
Hide

```
require(viridis)
#cut within color
diamonds %>%
  count(color, cut) %>%
  group_by(color) %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(mapping = aes(x = color, y = cut)) +
  geom_tile(mapping = aes(fill = prop)) +
  scale_fill_viridis(limits = c(0, 1))
```



Hide

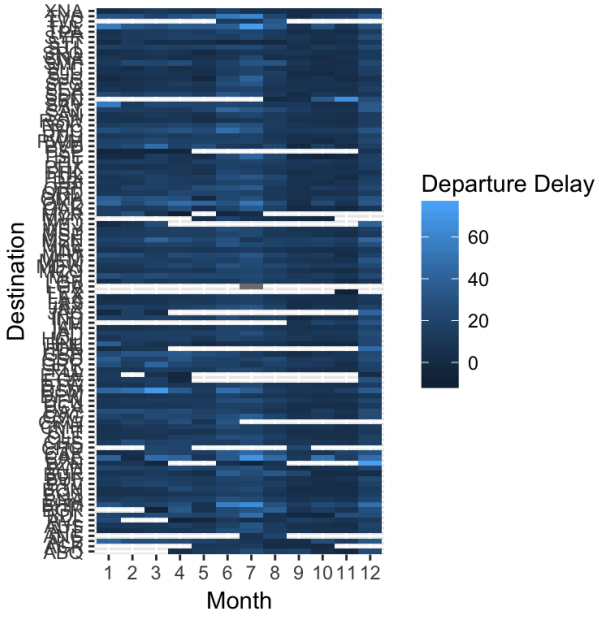
```
#color within cut
diamonds %>%
  count(color, cut) %>%
  group_by(cut) %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(mapping = aes(x = color, y = cut)) +
  geom_tile(mapping = aes(fill = prop)) +
  scale_fill_viridis(limits = c(0, 1))
```



2. Use geom_tile() together with dplyr to explore how average flight delays vary by destination and month of year. What makes the plot difficult to read? How could you improve it?

Hide

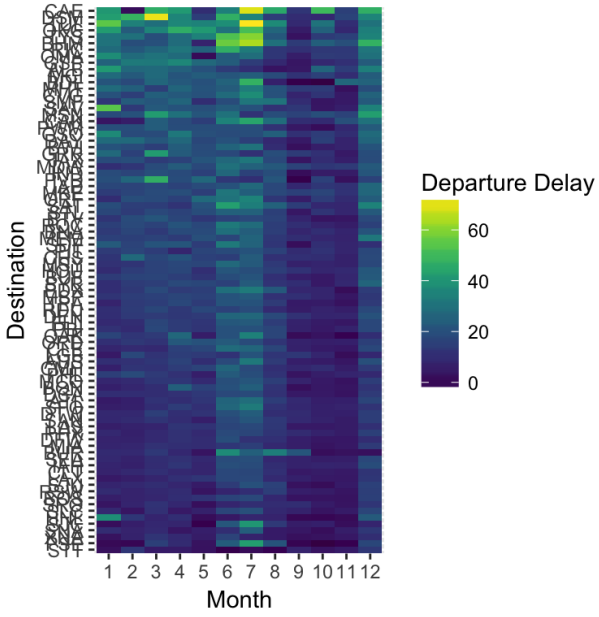
```
nycflights13::flights %>%
  group_by(month,dest) %>%
  summarize(dep_delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x=factor(month),y=dest,fill=dep_delay))+geom_tile()+
  labs(x = "Month", y = "Destination", fill = "Departure Delay")
```



So many of the tiles couldn't be drawn. Again, per jarnold's answer, we filtered for those with 12 destinations,reorder destination and dep_delay factors to get the higher delay destinations toward the top of the plot, and changed the color scheme.

Hide

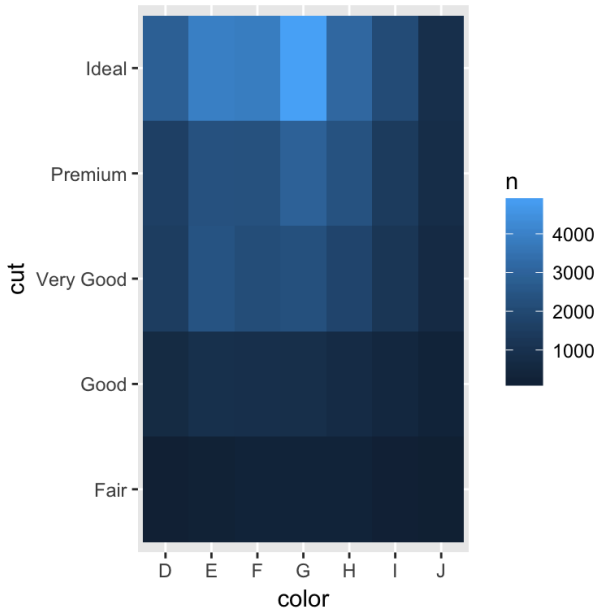
```
require(forcats)
nycflights13::flights %>%
  group_by(month,dest) %>%
  summarize(dep_delay = mean(dep_delay, na.rm = TRUE)) %>%
  group_by(dest) %>%
  filter(n() == 12) %>%
  ungroup() %>%
  mutate(dest = fct_reorder(dest, dep_delay)) %>%
  ggplot(aes(x=factor(month),y=dest,fill=dep_delay))+geom_tile()+
  labs(x = "Month", y = "Destination", fill = "Departure Delay")+
    scale_fill_viridis()
```



3. Why is it slightly better to use aes(x = color, y = cut) rather than aes(x = cut, y = color) in the example above?

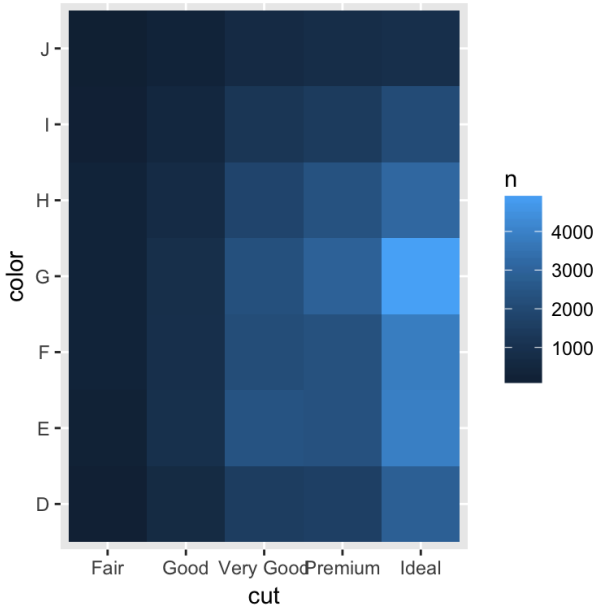
Hide

```
diamonds %>%
  count(color, cut) %>%
  ggplot(mapping = aes(x = color, y = cut)) +
    geom_tile(mapping = aes(fill = n))
```



Hide

```
diamonds %>%
  count(color, cut) %>%
  ggplot(mapping = aes(x = cut, y = color)) +
    geom_tile(mapping = aes(fill = n))
```



Longer cut names so better to have them on the y axis and the tiles are less wide in the former so it looks less stretched.

Exercise 7.5.3.1

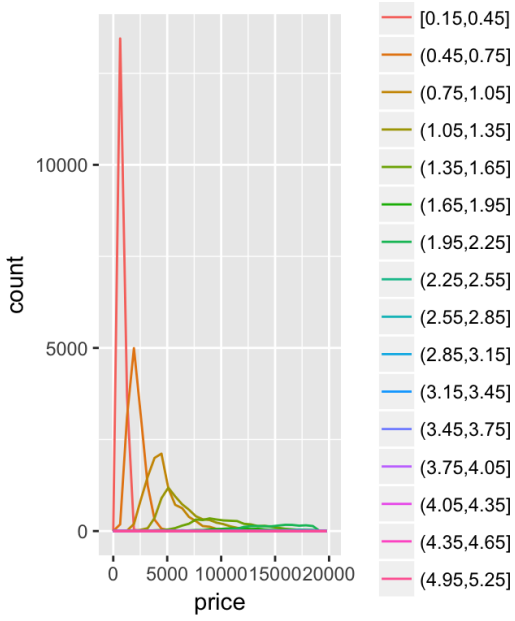
1. Instead of summarising the conditional distribution with a boxplot, you could use a frequency polygon. What do you need to consider when using cut_width() vs cut_number()? How does that impact a visualisation of the 2d distribution of carat and price?

Per jarnold:

“When using cut_width the number in each bin may be unequal. The distribution of carat is right skewed so there are few diamonds in those bins.”

Hide

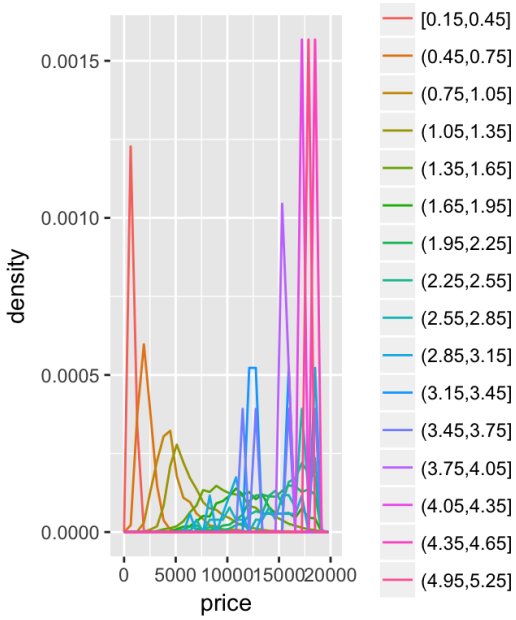
```
ggplot(data = diamonds,
       mapping = aes(x = price,
                     colour = cut_width(carat, 0.3))) +
  geom_freqpoly()
```



“Plotting the density instead of counts will make the distributions comparable, although the bins with few observations will still be hard to interpret.”

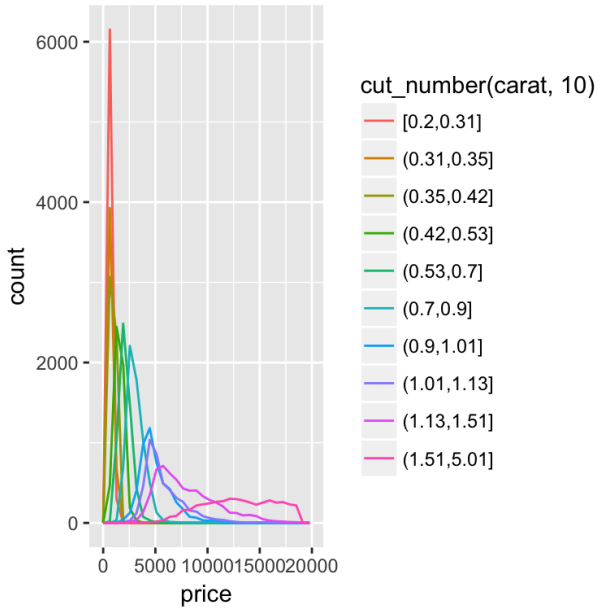
Hide

```
#cut width
ggplot(data = diamonds,
       mapping = aes(x = price,
                     y = ..density..,
                     colour = cut_width(carat, 0.3))) +
  geom_freqpoly()
```



Hide

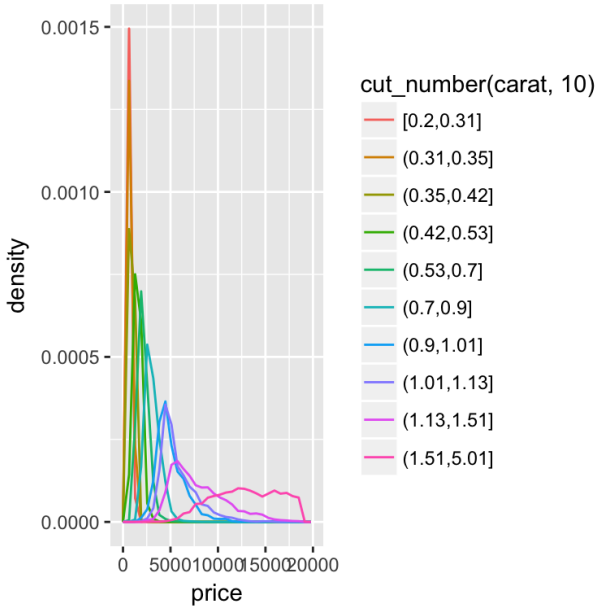
```
#cut number
ggplot(data = diamonds,
       mapping = aes(x = price,
                     colour = cut_number(carat, 10))) +
  geom_freqpoly()
```



“Since there are equal numbers in each bin, the plot looks the same if density is used for the y aesthetic (although the values are on a different scale).”

Hide

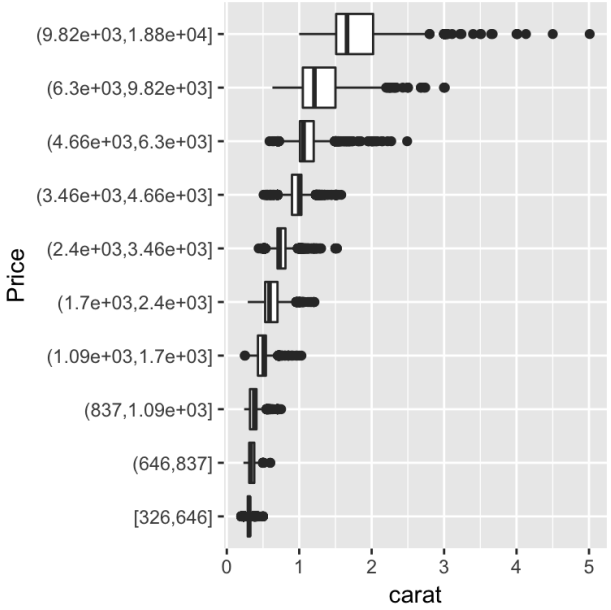
```
ggplot(data = diamonds,
       mapping = aes(x = price,
                     y = ..density..,
                     colour = cut_number(carat, 10))) +
  geom_freqpoly()
```



2. Visualise the distribution of carat, partitioned by price

Hide

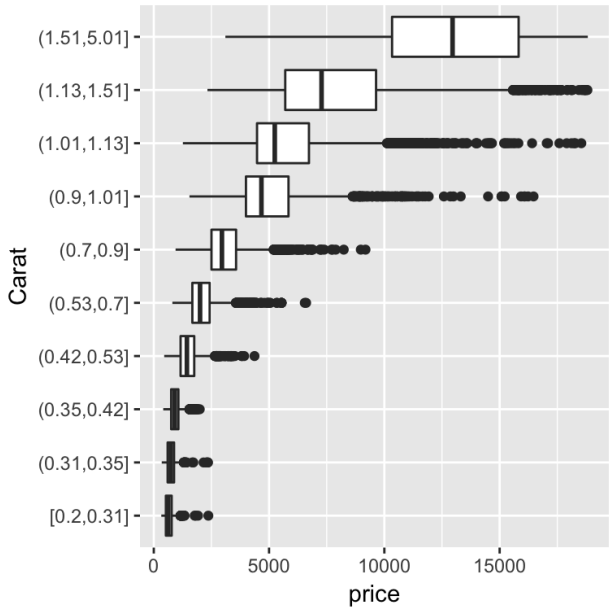
```
ggplot(diamonds, aes(x = cut_number(price, 10), y = carat)) +
  geom_boxplot() +
  coord_flip() +
  xlab("Price")
```



3. How does the price distribution of very large diamonds compare to small diamonds. Is it as you expect, or does it surprise you?

Hide

```
ggplot(diamonds, aes(x = cut_number(carat, 10), y = price)) +
  geom_boxplot() +
  coord_flip() +
  xlab("Carat")
```

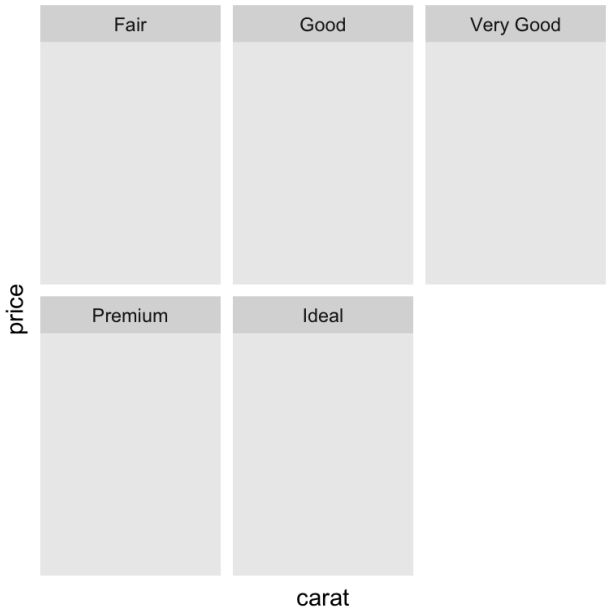


I would expect larger carat diamonds to be more expensive, but there's substantial overlap with that, depending on other factors.

4. Combine two of the techniques you've learned to visualise the combined distribution of cut, carat, and price.

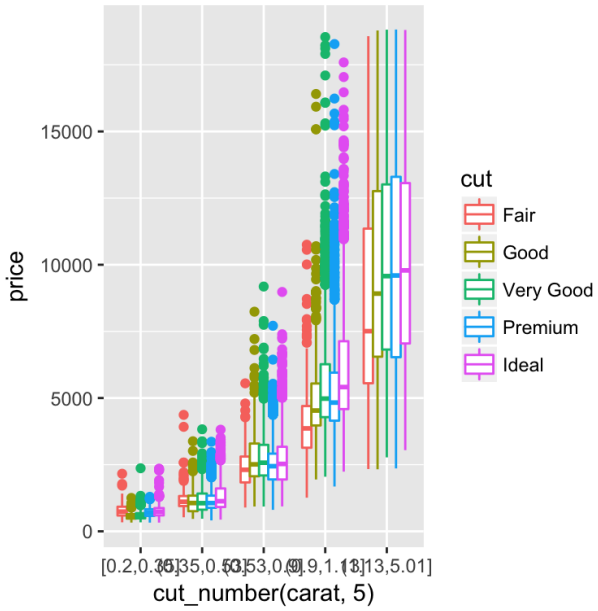
Hide

```
require(hexbin)
diamonds %>%
ggplot(mapping = aes(x = carat, y = price)) +
  geom_hex()+
  facet_wrap(~ cut, ncol=3) +
  scale_fill_viridis()
```



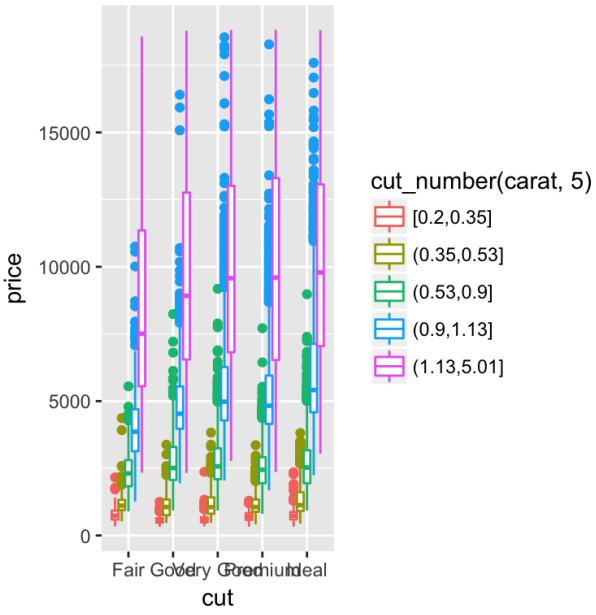
Hide

```
ggplot(diamonds, aes(x = cut_number(carat, 5), y = price, color = cut)) +
  geom_boxplot()
```



Hide

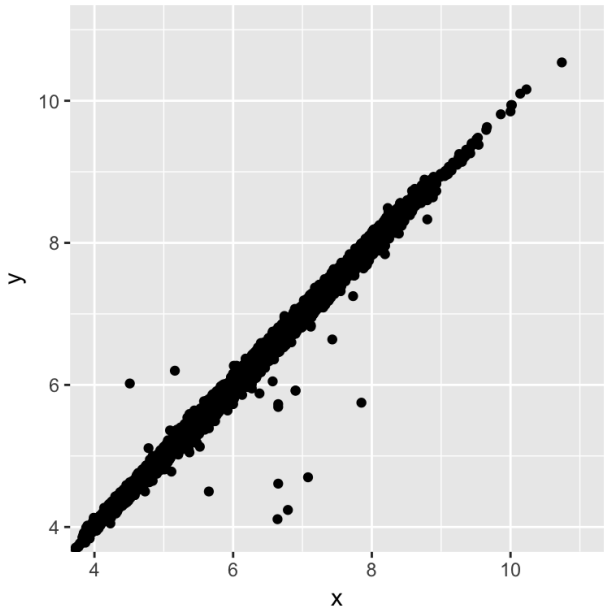
```
ggplot(diamonds, aes(color = cut_number(carat, 5), y = price, x = cut)) +
  geom_boxplot()
```



5. Two dimensional plots reveal outliers that are not visible in one dimensional plots. For example, some points in the plot below have an unusual combination of x and y values, which makes the points outliers even though their x and y values appear normal when examined separately. Why is a scatterplot a better display than a binned plot for this case?

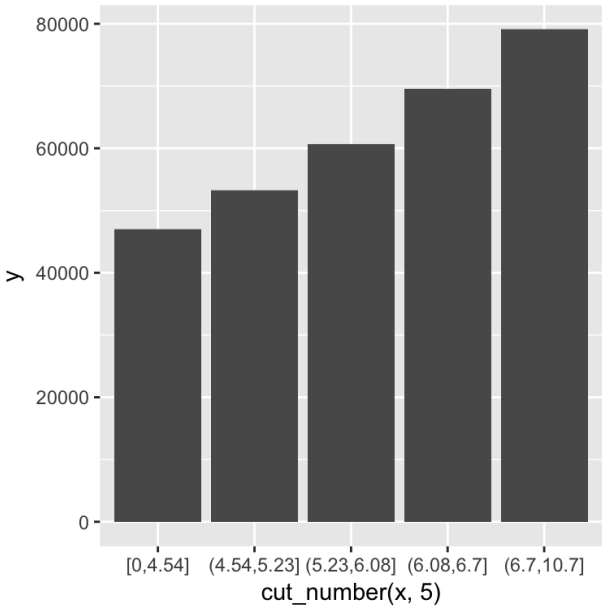
Hide

```
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = x, y = y)) +
  coord_cartesian(xlim = c(4, 11), ylim = c(4, 11))
```



Hide

```
ggplot(data = diamonds) +
  geom_bar(stat="identity",mapping = aes(x = cut_number(x,5), y = y))
```



geom_point gives far greater resolution for picking out outliers in the distributions.

Part II Wrangle

8. Workflow: projects

No exercises

9. Introduction

No Exercises

10. Tibbles

Exercise 10.5

1. How can you tell if an object is a tibble? (Hint: try printing mtcars, which is a regular data frame)

You can tell if an object is a tibble, as opposed to a dataframe, in that the class tible is printed, each column class is printed out below the variable name, and the trailing variable names are not printed out but only listed. Addtionally, tibbles have class "tbl_df" and "tbl_" in addition to "data.frame".

Hide

```
as.tibble(mtcars)
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##  * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
##  2  21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
##  3  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
##  4  21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
##  5  18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
##  6  18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
##  7  14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
##  8  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
##  9  22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
## 10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
## # ... with 22 more rows
```

2. Compare and contrast the following operations on a data.frame and equivalent tibble. What is different? Why might the default data frame behaviours cause you frustration?

Hide

```
df <- data.frame(abc = 1, xyz = "a")
df$x
```

```
## [1] a
## Levels: a
```

Hide

```
df[, "xyz"]
```

```
## [1] a
## Levels: a
```

Hide

```
df[, c("abc", "xyz")]
```

```
##   abc xyz
## 1   1   a
```

On a tibble,

Hide

```
tbl <- as.tibble(df)

#tbl %>% .$x #Doesn't work because "x" is not a variable name

tbl %>% .[["xyz"]]
```

```
## [1] a
## Levels: a
```

Hide

```
tbl[,c("abc","xyz")]
```

```
## # A tibble: 1 x 2
##     abc    xyz
##   <dbl> <fctr>
## 1     1      a
```

$dfx * iscompletedto * dfxyz$ which could be good but also could call the wrong variable and therefore troublesome.

From jarnold "With data.frames, with [the type of object that is returned differs on the number of columns. If it is one column, it won't return a data.frame, but instead will return a vector. With more than one column, then it will return a data.frame. This is fine if you know what you are passing in, but suppose you did df[, vars] where vars was a variable. Then you what that code does depends on length(vars) and you'd have to write code to account for those situations or risk bugs."

3. If you have the name of a variable stored in an object, e.g. var <- "mpg", how can you extract the reference variable from a tibble?

Hide

```
tbl <- as.tibble(mtcars)

var <- "mpg"

tbl[[var]]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

Hide

```
#or

pull(tbl,var)
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

4. Practice referring to non-syntactic names in the following data frame by:

Extracting the variable called 1.

Plotting a scatterplot of 1 vs 2.

Creating a new column called 3 which is 2 divided by 1.

Renaming the columns to one, two and three.

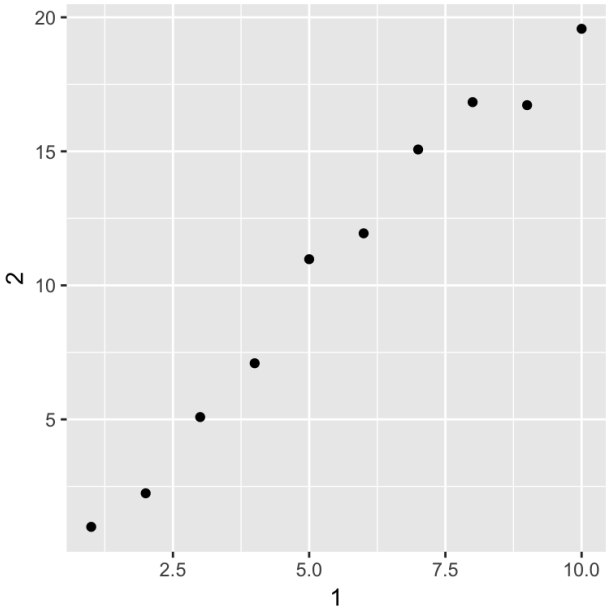
Hide

```
annoying <- tibble(
  `1` = 1:10,
  `2` = `1` * 2 + rnorm(length(`1`))
)

pull(annoying,`1`)
```

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

Hide

```
ggplot(annoying,aes(x= `1`,y = `2`)) + geom_point()
```



Hide

```
annoying <- annoying %>% mutate(`3` = `2` / `1`)

annoying
```

```
## # A tibble: 10 x 3
##      `1`        `2`       `3`
##    <int>      <dbl>     <dbl>
##  1     1  0.9907587 0.9907587
##  2     2  2.2437326 1.1218663
##  3     3  5.0863258 1.6954419
##  4     4  7.0942603 1.7735651
##  5     5 10.9757141 2.1951428
##  6     6 11.9408270 1.9901378
##  7     7 15.0688189 2.1526884
##  8     8 16.8350357 2.1043795
##  9     9 16.7230989 1.8581221
## 10    10 19.5716049 1.9571605
```

Hide

```
annoying <- rename(annoying, one = `1`, two = `2`, three = `3`)

annoying
```

```
## # A tibble: 10 x 3
##      one        two     three
##    <int>      <dbl>     <dbl>
##  1     1  0.9907587 0.9907587
##  2     2  2.2437326 1.1218663
##  3     3  5.0863258 1.6954419
##  4     4  7.0942603 1.7735651
##  5     5 10.9757141 2.1951428
##  6     6 11.9408270 1.9901378
##  7     7 15.0688189 2.1526884
##  8     8 16.8350357 2.1043795
##  9     9 16.7230989 1.8581221
## 10    10 19.5716049 1.9571605
```

5. What does tibble::enframe() do? When might you use it?

It converts named vectors to a data frame with names and values

Hide

```
enframe(c(a = 1, b = 2, c = 3))
```

```
## # A tibble: 3 x 2
##    name value
##   <chr> <dbl>
## 1     a     1
## 2     b     2
## 3     c     3
```

6. What option controls how many additional column names are printed at the footer of a tibble?

print(n_extra = ?) sets the number of column names printed

Hide

```
mtcars %>% print(n_extra = 5)
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

11. Data Import

Exercise 11.2.2

1. What function would you use to read a file where fields were separated with "|"?

I would use read_delim(), and indicate with sep="|".

2. Apart from file, skip, and comment, what other arguments do read_csv() and read_tsv() have in common?

Look up the arguments at ?read_tsv or ?read_csv. Other arguments in common include col_names, quote and delim.

Hide

```
library(readr)
union(names(formals(read_csv)), names(formals(read_tsv)))
```

```
##  [1] "file"      "col_names" "col_types" "locale"    "na"
##  [6] "quoted_na" "quote"     "comment"   "trim_ws"   "skip"
## [11] "n_max"     "guess_max" "progress"
```

3. What are the most important arguments to read_fwf()?

The most important arguments to read_fwf which reads in a fixed width file includes the file path and col_positions which tells the function where data columns begin and end. Read ?read_fwf for more details.

4. Sometimes strings in a CSV file contain commas. To prevent them from causing problems they need to be surrounded by a quoting character, like " or '. By convention, read_csv() assumes that the quoting character will be ", and if you want to change it you'll need to use read_delim() instead. What arguments do you need to specify to read the following text into a data frame?

Hide

```
x <- "x,y\n1,'a,b'"
read_delim(x, ",", quote = "'")
```

```
## # A tibble: 1 x 2
##       x     y
##   <int> <chr>
## 1     1   a,b
```

5. Identify what is wrong with each of the following inline CSV files. What happens when you run the code?

Hide

```
read_csv("a,b\n1,2,3\n4,5,6") #implying different number of rows so chops of the 3rd element in rows 2 and 3.
```

```
## # A tibble: 2 x 2
##       a     b
##   <int> <int>
## 1     1     2
## 2     4     5
```

Hide

```
read_csv("a,b,c\n1,2\n1,2,3,4") #adds NA to 2nd column and chops of 4th element in 4th column
```

```
## # A tibble: 2 x 3
##       a     b     c
##   <int> <int> <int>
## 1     1     2    NA
## 2     1     2     3
```

Hide

```
read_csv("a,b\n\"1") #2nd element in 1st row is missing
```

```
## # A tibble: 1 x 2
##       a     b
##   <int> <chr>
## 1     1  <NA>
```

Hide

```
read_csv("a,b\n1,2\na,b") #nothing that I see
```

```
## # A tibble: 2 x 2
##       a     b
##   <chr> <chr>
## 1     1     2
## 2     a     b
```

Hide

```
read_csv("a;b\n1;3") #need to indicate sep=";"
```

```
## # A tibble: 1 x 1
##   `a;b`
##   <chr>
## 1   1;3
```

Exercise 11.3.5

1. What are the most important arguments to locale()?

Arguably, the most important arguments are decimal_mark which tells of the decimal separator and grouping_mark which tells of the grouping separator. Use ?locale

2. What happens if you try and set decimal_mark and grouping_mark to the same character? What happens to the default value of grouping_mark when you set decimal_mark to ",”? What happens to the default value of decimal_mark when you set the grouping_mark to “.”?

Hide

```
parse_number("1,000.00",locale=locale(grouping_mark=",",decimal_mark="."))
```

```
## [1] 1000
```

Hide

```
#parse_number("1,000.00",locale=locale(grouping_mark=",",decimal_mark=",")) # Error: `decimal_mark` and `g
rouping_mark` must be different

parse_number("1.6,500,000.60",locale=locale(decimal_mark=",")) #grouping_mark default is also ","
```

```
## [1] 16.5
```

Hide

```
parse_number("1.666,500,000.60",locale=locale(grouping_mark=".")) # the "." is ignored
```

```
## [1] 1666.5
```

3. I didn't discuss the date_format and time_format options to locale(). What do they do? Construct an example that shows when they might be useful.

In locale, the date_format and time_format option indicates the rules for parsing a date and time, respectively.

4. If you live outside the US, create a new locale object that encapsulates the settings for the types of file you read most commonly.

Outside the US there seems to be a lack of standards for encoding/decoding.

5. What's the difference between read_csv() and read_csv2()?

sep=“,” in read_csv and sep=“;” in read_csv2

6. What are the most common encodings used in Europe? What are the most common encodings used in Asia? Do some googling to find out.

Most frequent encodings overall on the web: https://stackoverflow.com/questions/8509339/what-is-the-most-common-encoding-of-each-language (https://stackoverflow.com/questions/8509339/what-is-the-most-common-encoding-of-each-language)

UTF-8 (89.2%) ISO-8859-1 (5.0%) Windows-1251 (1.6%) Shift JIS (0.9%) Windows-1252 (0.8%) GB2312 (0.7%) EUC-KR (0.4%) EUC-JP (0.3%)

Also read http://unicodebook.readthedocs.io/encodings.html (http://unicodebook.readthedocs.io/encodings.html) and http://kunststube.net/encoding/ (http://kunststube.net/encoding/)

7. Generate the correct format string to parse each of the following dates and times:

Hide

```
d1 <- "January 1, 2010"
parse_date(d1, format="%B %d, %Y")
```

```
## [1] "2010-01-01"
```

Hide

```
d2 <- "2015-Mar-07"
parse_date(d2, format="%Y-%b-%d")
```

```
## [1] "2015-03-07"
```

Hide

```
d3 <- "06-Jun-2017"
parse_date(d3, format="%d-%b-%Y")
```

```
## [1] "2017-06-06"
```

Hide

```
d4 <- c("August 19 (2015)", "July 1 (2015)")
#parse_date(d4, format=c("%B %b (%Y)","%B %b (%Y)"))
d5 <- "12/30/14" # Dec 30, 2014
parse_date(d5, format="%m/%d/%y")
```

```
## [1] "2014-12-30"
```

Hide

```
t1 <- "1705"
parse_time(t1, format="%H%M")
```

```
## 17:05:00
```

Hide

```
t2 <- "11:15:10.12 PM"
#parse_datetime(t2) #can't figure out
```

12. Tidy data

Exercise 12.2.1

1. Using prose, describe how the variables and observations are organised in each of the sample tables.

In table 1, each observation is a (country, year) and variables are cases and count.

In table 2, each observation is a (county, year, type) and the variable is count.

In table 3, each observation is (country, year) and the variable is rate (cases/population)

In the table 4s, each observation is country and variables are years. Table 4a is cases and 4b is population.

2. Compute the rate for table2, and table4a + table4b. You will need to perform four operations:

Extract the number of TB cases per country per year.

Hide

```
library(tidyverse)

table2 %>%
  filter(type=="cases") %>%
  group_by(country) %>%
  summarize(TB = sum(count))
```

```
## # A tibble: 3 x 2
##       country     TB
##         <chr>  <int>
## 1 Afghanistan   3411
## 2      Brazil 118225
## 3       China 426024
```

Extract the matching population per country per year.

Hide

```
table2 %>% filter(type == "population")
```

```
## # A tibble: 6 x 4
##       country  year       type      count
##         <chr> <int>      <chr>      <int>
## 1 Afghanistan  1999 population   19987071
## 2 Afghanistan  2000 population   20595360
## 3      Brazil  1999 population  172006362
## 4      Brazil  2000 population  174504898
## 5       China  1999 population 1272915272
## 6       China  2000 population 1280428583
```

Divide cases by population, and multiply by 10000.

Hide

```
tb2_cases <- filter(table2, type == "cases")[["count"]]
tb2_country <- filter(table2, type == "cases")[["country"]]
tb2_year <- filter(table2, type == "cases")[["year"]]
tb2_population <- filter(table2, type == "population")[["count"]]
options(scipen=-99)
table2_clean <- tibble(country = tb2_country,
       year = tb2_year,
       rate = tb2_cases / tb2_population)
table2_clean
```

```
## # A tibble: 6 x 3
##       country  year         rate
##         <chr> <int>        <dbl>
## 1 Afghanistan  1999 3.727410e-05
## 2 Afghanistan  2000 1.294466e-04
## 3      Brazil  1999 2.193930e-04
## 4      Brazil  2000 4.612363e-04
## 5       China  1999 1.667495e-04
## 6       China  2000 1.669488e-04
```

Store back in the appropriate place.

It's stored in table2_clean.

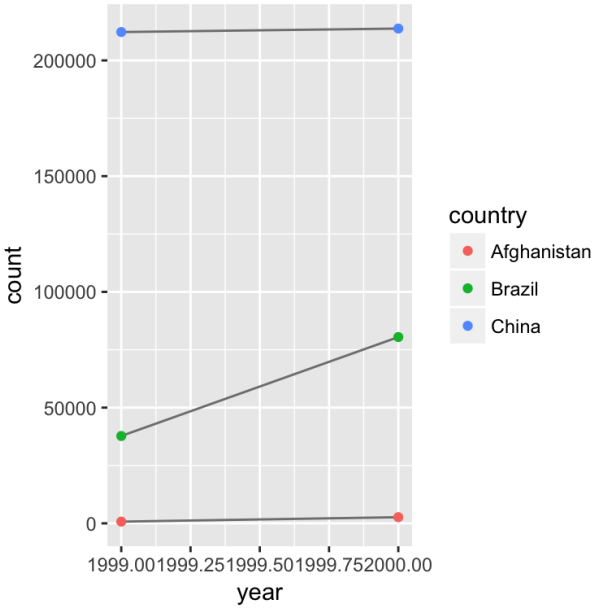
Which representation is easiest to work with? Which is hardest? Why?

Hmm. It's easier to work with table2 than table4a or 4b because all the data is in table2 but is separated in table4a and 4b which makes doing operations a bit more complicated.

3. Recreate the plot showing change in cases over time using table2 instead of table1. What do you need to do first?

Hide

```
data <- table2 %>%
  filter(type == "cases")

options(scipen=99)

  ggplot(data,aes(year, count)) +
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country)) +
    theme(legend.box.just = "centre")
```



Exercise 12.3.3

1. Why are gather() and spread() not perfectly symmetrical? Carefully consider the following example:

Hide

```
stocks <- tibble(
  year   = c(2015, 2015, 2016, 2016),
  half  = c(   1,    2,     1,    2),
  return = c(1.88, 0.59, 0.92, 0.17)
)
stocks %>%
  spread(year, return) %>%
  gather("year", "return", `2015`:`2016`)
```

```
## # A tibble: 4 x 3
##    half  year return
##   <dbl> <chr>  <dbl>
## 1     1  2015   1.88
## 2     2  2015   0.59
## 3     1  2016   0.92
## 4     2  2016   0.17
```

(Hint: look at the variable types and think about column names.) Both spread() and gather() have a convert argument. What does it do?

From jarnold

" The functions spread and gather are not perfectly symmetrical because column type information is not transferred between them. In the original table the column year was numeric, but after the spread-gather cyle it is character, because with gather, variable names are always converted to a character vector.

The convert argument tries to convert character vectors to the appropriate type. In the background this uses the type.convert function. "

Hide

```
stocks %>%
  spread(year, return) %>%
  gather("year", "return", `2015`:`2016`, convert = TRUE)
```

```
## # A tibble: 4 x 3
##    half  year return
##   <dbl> <int>  <dbl>
## 1     1  2015   1.88
## 2     2  2015   0.59
## 3     1  2016   0.92
## 4     2  2016   0.17
```

2. Why does this code fail?

table4a %>% gather(1999, 2000, key = “year”, value = “cases”)

Error in combine_vars(vars, ind_list) : Position must be between 0 and n

The code fails because the column names 1999 and 2000 are not standard and thus needs to be quoted. The tidyverse functions will interpret 1999 and 2000 without quotes as looking for the 1999th and 2000th column of the data frame.

3. Why does spreading this tibble fail? How could you add a new column to fix the problem?

Hide

```
people <- tribble(
  ~name,             ~key,    ~value,
  #-----------------|--------|------
  "Phillip Woods",   "age",       45,
  "Phillip Woods",   "height",   186,
  "Phillip Woods",   "age",       50,
  "Jessica Cordero", "age",       37,
  "Jessica Cordero", "height",   156
)
glimpse(people)
```

```
## Observations: 5
## Variables: 3
## $ name  <chr> "Phillip Woods", "Phillip Woods", "Phillip Woods", "Jess...
## $ key   <chr> "age", "height", "age", "age", "height"
## $ value <dbl> 45, 186, 50, 37, 156
```

spread(people, key, value) Error: Duplicate identifiers for rows (1, 3)

Spreading the data frame fails because there are two rows with “age” for “Phillip Woods”. We would need to add another column with an indicator for the number observation it is,

Hide

```
people <- tribble(
  ~name,             ~key,    ~value, ~obs,
  #-----------------|--------|------|------
  "Phillip Woods",   "age",       45, 1,
  "Phillip Woods",   "height",   186, 1,
  "Phillip Woods",   "age",       50, 2,
  "Jessica Cordero", "age",       37, 1,
  "Jessica Cordero", "height",   156, 1
)
spread(people, key, value)
```

```
## # A tibble: 3 x 4
##              name   obs   age height
## *           <chr> <dbl> <dbl>  <dbl>
## 1 Jessica Cordero     1    37    156
## 2   Phillip Woods     1    45    186
## 3   Phillip Woods     2    50     NA
```

4. Tidy the simple tibble below. Do you need to spread or gather it? What are the variables?

Hide

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes",     NA,    10,
  "no",      20,    12
)
```

You need to gather male (values are logical) and count (integer count of gender)

Hide

```
gather(preg, sex, count, male, female) %>%
  mutate(pregnant = pregnant == "yes",
         female = sex == "female") %>%
  select(-sex)
```

```
## # A tibble: 4 x 3
##   pregnant count female
##      <lgl> <dbl>  <lgl>
## 1     TRUE    NA  FALSE
## 2    FALSE    20  FALSE
## 3     TRUE    10   TRUE
## 4    FALSE    12   TRUE
```

Exercise 12.4.3

1. What do the extra and fill arguments do in separate()? Experiment with the various options for the following two toy datasets.

Hide

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j"))
```

```
## # A tibble: 3 x 1
##         x
##     <chr>
## 1   a,b,c
## 2 d,e,f,g
## 3   h,i,j
```

Hide

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"),extra="merge")
```

```
## # A tibble: 3 x 3
##     one   two three
## * <chr> <chr> <chr>
## 1     a     b     c
## 2     d     e   f,g
## 3     h     i     j
```

Hide

```
tibble(x = c("a,b,c", "d,e", "f,g,i"))
```

```
## # A tibble: 3 x 1
##       x
##   <chr>
## 1 a,b,c
## 2   d,e
## 3 f,g,i
```

Hide

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"),fill="left")
```

```
## # A tibble: 3 x 3
##     one   two three
## * <chr> <chr> <chr>
## 1     a     b     c
## 2  <NA>     d     e
## 3     f     g     i
```

fill and extra dictate how extra or missing values in rows are delt with when making a table.

If there's an extra value (like having multiple rows with the minority of the rows having more values), then we can indicate whether to give a warning about beingh dropped, drop the value without warning, or merge the last two values together. Or we can fill the missing value with NA frrom the left or right.

2. Both unite() and separate() have a remove argument. What does it do? Why would you set it to FALSE?

From jarnold, "You would set it to FALSE if you want to create a new variable, but keep the old one."

3. Compare and contrast separate() and extract(), Why are there three variations of separation (by position, by separator, and with groups), but only one unite?

From jarnold, “The function extract uses a regular expression to find groups and split into columns. In unite it is unambigous since it is many columns to one, and once the columns are specified, there is only one way to do it, the only choice is the sep. In separate, it is one to many, and there are multiple ways to split the character string.”

Exercise 12.5.1

1. Compare and contrast the fill arguments to spread() and complete().

From jarnold, “In spread, the fill argument explicitly sets the value to replace NAs. In complete, the fill argument also sets a value to replace NAs but it is named list, allowing for different values for different variables. Also, both cases replace both implicit and explicit missing values.”

2. What does the direction argument to fill() do?

From jarnold, “With fill, it determines whether NA values should be replaced by the previous non-missing value (”down“) or the next non-missing value (”up“).”

Exercise 12.6.1

1. In this case study I set na.rm = TRUE just to make it easier to check that we had the correct values. Is this reasonable? Think about how missing values are represented in this dataset. Are there implicit missing values? What's the difference between an NA and zero?

With help from jarnold's solutions,

Hide

```
who1 <- who %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "cases", na.rm = TRUE)

who1 %>%
  filter(cases == 0) %>%
  nrow()
```

```
## [1] 11080
```

Starting the tidy process gives us the number of cases of a type of TB for a country. We see above there's 11080 rows with 0 cases, indicating no representation of TB cases for a country. Setting na.rm=T or F doesn't matter since WHO seems to have no data entered if none were measured for a country (as far as we can tell just from the data). In the end, it doesn't matter the specification.

2. What happens if you neglect the mutate() step? (mutate(key = stringr::str_replace(key, “newrel”, “new_rel”))

Neglecting the mutate step can be OK if we know that all cases are new and we just parse the case type after the 3rd character. But we may not know that so better to mutate.

3. I claimed that iso2 and iso3 were redundant with country. Confirm this claim.

From jarnold,

Hide

```
select(who, country, iso2, iso3) %>%
  distinct() %>%
  group_by(country) %>%
  filter(n() > 1)
```

```
## # A tibble: 0 x 3
## # Groups:   country [0]
## # ... with 3 variables: country <chr>, iso2 <chr>, iso3 <chr>
```

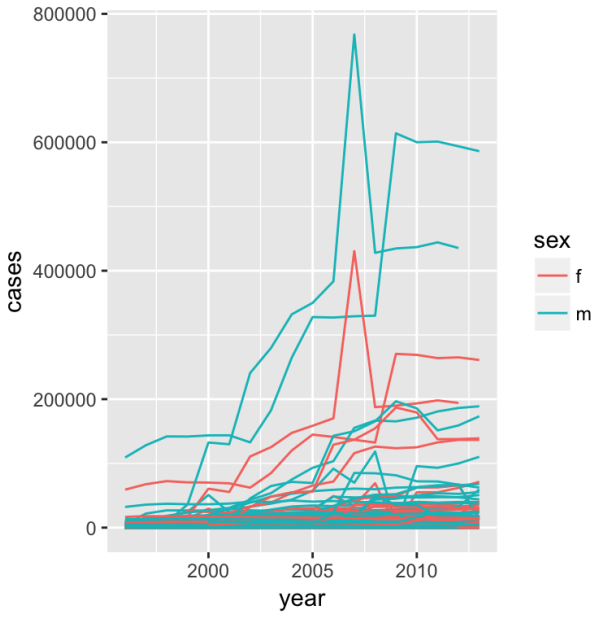
If there were redundant values, after displaying only distinct rows, there would be a country represented >1 if it paired with a differet iso2 or iso3, thus they're redundant columns.

4. For each country, year, and sex compute the total number of cases of TB. Make an informative visualisation of the data

Because of the 0s for cases of TB types, it's better to just sum cases of all types. Also because there are no or few cases before 1195, we'll filter for after then. Also to distinguish cases by country and sex, we're making a new variable to group by to have the graph less cluttered. This is adapted from jarnold.

Hide

```
who_new <- who %>%
  gather(code, value, new_sp_m014:newrel_f65, na.rm = TRUE) %>%
  mutate(code = stringr::str_replace(code, "newrel", "new_rel")) %>%
  separate(code, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)

who_new %>%
  group_by(country, year, sex) %>%
  filter(year > 1995) %>%
  summarise(cases = sum(value)) %>%
  unite(country_sex, country, sex, remove = FALSE) %>%
  ggplot(aes(x = year, y = cases, group = country_sex, colour = sex)) +
  geom_line()
```



13. Relational data

Exercise 13.2.1

1. Imagine you wanted to draw (approximately) the route each plane flies from its origin to its destination. What variables would you need? What tables would you need to combine?

To get the route, I need the origin and dest variables in the flights table. And I would connect the flights table to the tailnum variable in the planes table to account for each plane. The route would be constructed from the latitude and longtiude variablesin the airports table.

2. I forgot to draw the relationship between weather and airports. What is the relationship and how should it appear in the diagram?

The tables weather and airports are connected by the airports (NYC only) in the faa or origins variables in the tables, respectively.

3. Weather only contains information for the origin (NYC) airports. If it contained weather records for all airports in the USA, what additional relation would it define with flights?

year, month, day, hour, origin in weather would be matched to year, month, day, hour, dest in flight.

4. We know that some days of the year are "special", and fewer people than usual fly on them. How might you represent that data as a data frame? What would be the primary keys of that table? How would it connect to the existing tables?

If I wanted to make a "special" table, I would indicate the year, month and day variables to be keys and include other variables which would connect to tables.

Exercise 13.3.1

1. Add a surrogate key to flights.

Hide

```
library(tidyverse)
library(nycflights13)

flights %>%
  arrange(year,month,day,sched_dep_time,carrier,flight) %>%
  mutate(flight_id = row_number()) %>%
  count(flight_id) %>%
  filter(n > 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: flight_id <int>, n <int>
```

Without a surrogate key like above, we'd need to combine these 5 different variables to get a parent key.

2. Identify the keys in the following datasets

- **Lahman::Batting,**
- **babynames::babynames**
- **nasaweather::atmos**
- **fueleconomy::vehicles**
- **ggplot2::diamonds**

(You might need to install some packages and read some documentation.)

Hide

```
Lahman::Batting %>%
  group_by(playerID, yearID, stint) %>%
  filter(n() > 1) %>%
  nrow()
```

```
## [1] 0
```

Hide

```
babynames::babynames %>%
  group_by(year, sex, name) %>%
  filter(n() > 1) %>%
  nrow()
```

```
## [1] 0
```

Hide

```
nasaweather::atmos %>%
  group_by(lat, long, year, month) %>%
  filter(n() > 1) %>%
  nrow()
```

```
## [1] 0
```

Hide

```
fueleconomy::vehicles %>%
  group_by(id) %>%
  filter(n() > 1) %>%
  nrow()
```

```
## [1] 0
```

There is no primary key for ggplot2::diamonds. Using all variables in the data frame, the number of distinct rows is less than the total number of rows, meaning no combination of variables uniquely identifies the observations.

3. Draw a diagram illustrating the connections between the Batting, Master, and Salaries tables in the Lahman package. Draw another diagram that shows the relationship between Master, Managers, AwardsManagers.

How would you characterise the relationship between the Batting, Pitching, and Fielding tables?

Hide

```
colnames(Lahman::Batting)
```

```
##  [1] "playerID" "yearID"   "stint"    "teamID"   "lgID"     "G"
##  [7] "AB"       "R"        "H"        "X2B"      "X3B"      "HR"
## [13] "RBI"      "SB"       "CS"       "BB"       "SO"       "IBB"
## [19] "HBP"      "SH"       "SF"       "GIDP"
```

Hide

```
colnames(Lahman::Master)
```

```
##  [1] "playerID"     "birthYear"    "birthMonth"   "birthDay"
##  [5] "birthCountry" "birthState"   "birthCity"    "deathYear"
##  [9] "deathMonth"   "deathDay"     "deathCountry" "deathState"
## [13] "deathCity"    "nameFirst"    "nameLast"     "nameGiven"
## [17] "weight"       "height"       "bats"         "throws"
## [21] "debut"        "finalGame"    "retroID"      "bbrefID"
## [25] "deathDate"    "birthDate"
```

Hide

```
colnames(Lahman::Salaries)
```

```
## [1] "yearID"   "teamID"   "lgID"     "playerID" "salary"
```

From jarnold (I'm too lazy to give this a good college try):

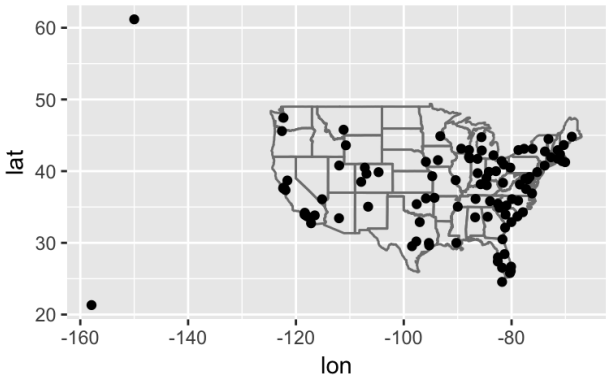
Batting primary key: playerID, yearID, stint foreign keys: playerID -> Master.playerID Master primary key: playerID Salaries primary key: yearID, teamID, playerID foreign keys: playerID -> Master.playerID Managers: primary key: yearID, playerID, teamID, inseason foreign keys: playerID -> Master.teamID Managers: primary key: awardID, yearID AwardsManagers: primary key: playerID, awardID, yearID (since there are ties and while tie distinguishes those awards it has NA values) foreign keys: playerID -> Master.playerID playerID, yearID, lgID -> Managers.playerID, yearID, lgID lgID and teamID appear in multiple tables, but should be primary keys for league and team tables.

Exercise 13.4.6

1. Compute the average delay by destination, then join on the airports data frame so you can show the spatial distribution of delays. Here's an easy way to draw a map of the United States:

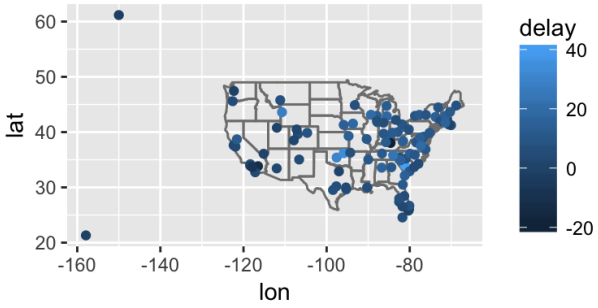
Hide

```
airports %>%
  semi_join(flights, c("faa" = "dest")) %>%
  ggplot(aes(lon, lat)) +
    borders("state") +
    geom_point() +
    coord_quickmap()
```



Hide

```
avg_dest_delays <-
  flights %>%
  group_by(dest) %>%
  # arrival delay NA's are cancelled flights
  summarise(delay = mean(arr_delay, na.rm = TRUE)) %>%
  inner_join(airports, by = c(dest = "faa"))

avg_dest_delays %>%
  ggplot(aes(lon, lat, colour = delay)) +
    borders("state") +
    geom_point() +
    coord_quickmap()
```



2. Add the location of the origin and destination (i.e. the lat and lon) to flights.

Hide

```
flights %>%
  left_join(airports, c(origin = "faa")) %>%
  left_join(airports, c(dest = "faa"))
```

```
## # A tibble: 336,776 x 33
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 26 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, name.x <chr>, lat.x <dbl>,
## #   lon.x <dbl>, alt.x <int>, tz.x <dbl>, dst.x <chr>, tzone.x <chr>,
## #   name.y <chr>, lat.y <dbl>, lon.y <dbl>, alt.y <int>, tz.y <dbl>,
## #   dst.y <chr>, tzone.y <chr>
```

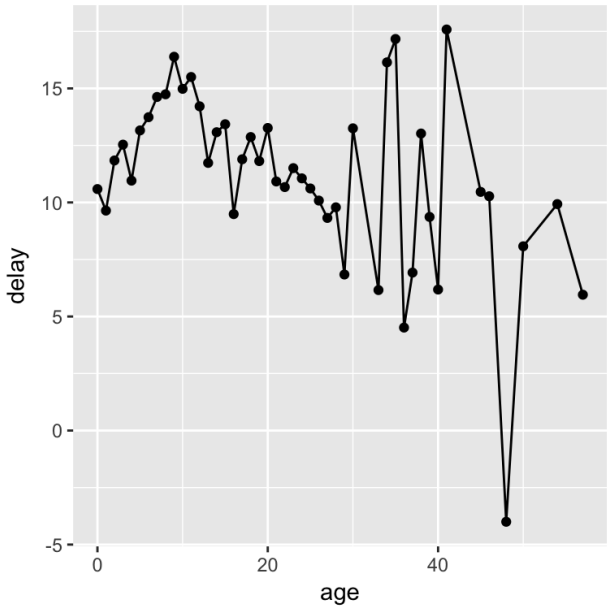
3. Is there a relationship between the age of a plane and its delays?

from jarnold,

Suprisingly not. If anything (departure) delay seems to decrease slightly with age (perhaps because of selection):

Hide

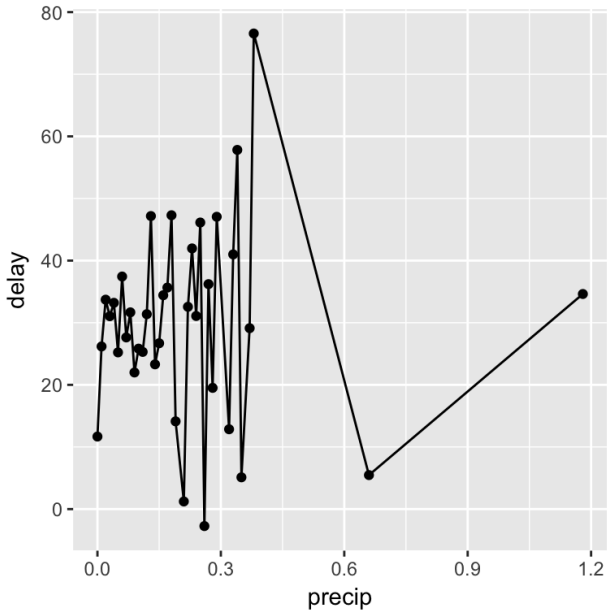
```
plane_ages <-
  planes %>%
  mutate(age = 2013 - year) %>%
  select(tailnum, age)

flights %>%
  inner_join(plane_ages, by = "tailnum") %>%
  group_by(age) %>%
  filter(!is.na(dep_delay)) %>%
  summarise(delay = mean(dep_delay)) %>%
  ggplot(aes(x = age, y = delay)) +
  geom_point() +
  geom_line()
```



4. What weather conditions make it more likely to see a delay?

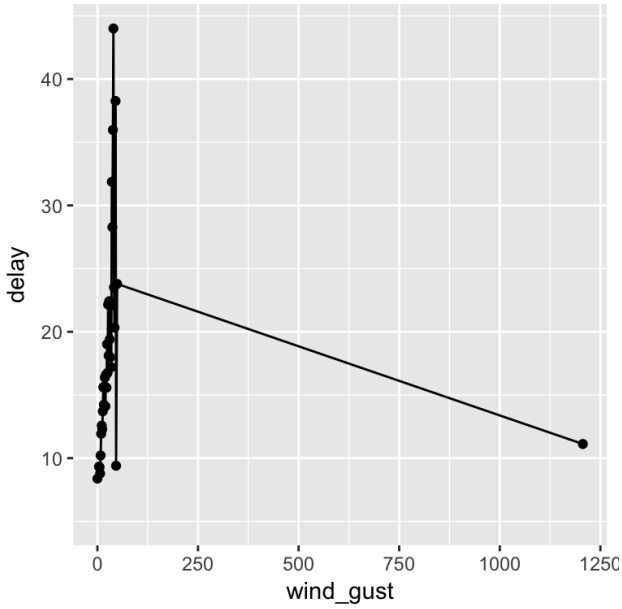
Hide

```
flight_weather <- flights %>%
  inner_join(weather,by=c("origin" = "origin",
                          "year" = "year",
                          "month" = "month",
                          "day" = "day",
                          "hour" = "hour")
             )

flight_weather %>%
  group_by(precip) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = precip, y = delay)) +
    geom_line() + geom_point()
```



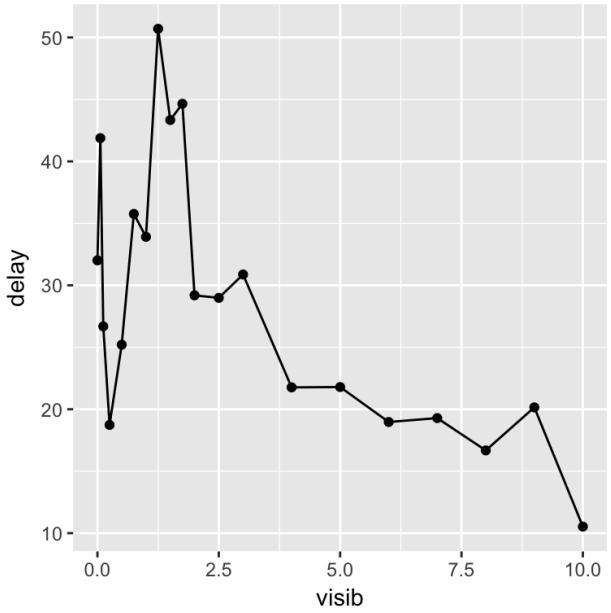
Hide

```
flight_weather %>%
  group_by(wind_gust) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = wind_gust, y = delay)) +
    geom_line() + geom_point()
```



Hide

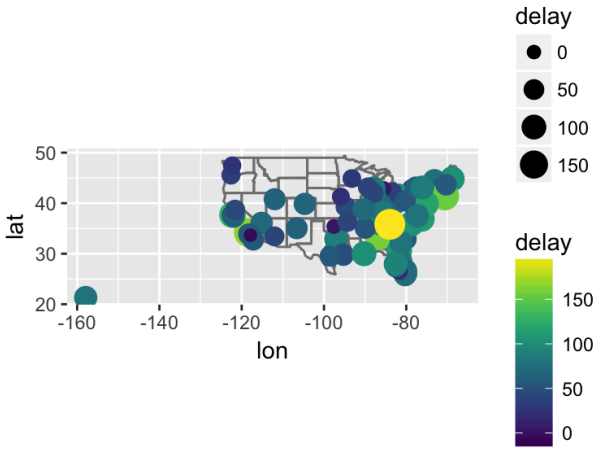
```
flight_weather %>%
  group_by(visib) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = visib, y = delay)) +
    geom_line() + geom_point()
```



5. What happened on June 13 2013? Display the spatial pattern of delays, and then use Google to cross-reference with the weather.

Hide

```
library(viridis)
flights %>%
  filter(year == 2013, month == 6, day == 13) %>%
  group_by(dest) %>%
  summarise(delay = mean(arr_delay, na.rm = TRUE)) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  ggplot(aes(y = lat, x = lon, size = delay, colour = delay)) +
  borders("state") +
  geom_point() +
  coord_quickmap() +
  scale_color_viridis()
```



Exercises 13.5.1

1. What does it mean for a flight to have a missing tailnum? What do the tail numbers that don't have a matching record in planes have in common? (Hint: one variable explains ~90% of the problems.)

Looks like NA tailnums never departed though they were scheduled to

American and Envoy airlines planes don't have tail numbers.

Hide

```
na_flights <- filter(flights,is.na(flights$tailnum))

flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(carrier, sort = TRUE)
```

```
## # A tibble: 10 x 2
##    carrier     n
##      <chr> <int>
##  1      MQ 25397
##  2      AA 22558
##  3      UA  1693
##  4      9E  1044
##  5      B6   830
##  6      US   699
##  7      FL   187
##  8      DL   110
##  9      F9    50
## 10      WN    38
```

2. Filter flights to only show flights with planes that have flown at least 100 flights.

Hide

```
flights %>%
  group_by(tailnum) %>%
  filter(n() > 100)
```

```
## # A tibble: 229,202 x 19
## # Groups:   tailnum [1,201]
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      544            545        -1     1004
##  4  2013     1     1      554            558        -4      740
##  5  2013     1     1      555            600        -5      913
##  6  2013     1     1      557            600        -3      709
##  7  2013     1     1      557            600        -3      838
##  8  2013     1     1      558            600        -2      849
##  9  2013     1     1      558            600        -2      853
## 10  2013     1     1      558            600        -2      923
## # ... with 229,192 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

3. Combine fueleconomy::vehicles and fueleconomy::common to find only the records for the most common models.

Hide

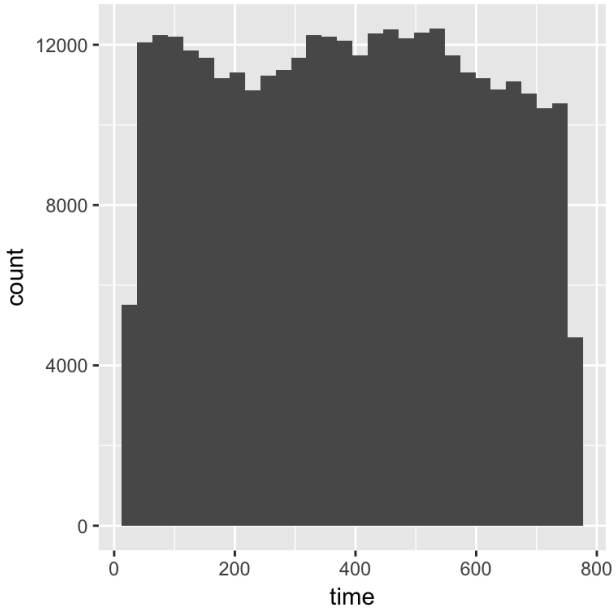
```
com_models <- fueleconomy::common %>%
  semi_join(fueleconomy::vehicles)
```

4. Find the 48 hours (over the course of the whole year) that have the worst delays. Cross-reference it with the weather data. Can you see any patterns?

2 day window with highest delay times

Hide

```
flight_weather %>%
  mutate(time = day*24 + hour + (minute/60) ) %>%
  ggplot(aes(time)) +
  geom_histogram()
```



Thought I had something...

5. What does anti_join(flights, airports, by = c("dest" = "faa")) tell you? What does anti_join(airports, flights, by = c("faa" = "dest")) tell you?

anti_join(flights, airports, by = c("dest" = "faa")) are flights that go to an airport that is not in FAA list of destinations, likely foreign airports.

anti_join(airports, flights, by = c("faa" = "dest")) are US airports that don't have a flight in the data, meaning that there were no flights to that aiport from New York in 2013.

6. You might expect that there's an implicit relationship between plane and airline, because each plane is flown by a single airline. Confirm or reject this hypothesis using the tools you've learned above.

Is there a plane flown by more than 1 airline? Nope.

Hide

```
flights %>%
  group_by(tailnum, carrier) %>%
  distinct() %>%
  distinct(tailnum) %>%
  group_by(tailnum) %>%
  count(carrier) %>%
  filter(n>1) %>%
  arrange(desc(n))
```

```
## # A tibble: 0 x 3
## # Groups:   tailnum [0]
## # ... with 3 variables: tailnum <chr>, carrier <chr>, n <int>
```

14. Strings

Exercise 14.2.5

1. In code that doesn't use stringr, you'll often see paste() and paste0(). What's the difference between the two functions? What stringr function are they equivalent to? How do the functions differ in their handling of NA?

The pasteo separator in nothing ("") and paste the default separator is space (" "). The equivalent stringr functions are str_c(), stri_paste(), stri_join(), and stri_flatten(). In paste, missing values, NA, are treated as strings, "NA".

2. In your own words, describe the difference between the sep and collapse arguments to str_c().

The collapse argument joins elements in a character vector while the sep argument joins elements from multiple character vectors.

3. Use str_length() and str_sub() to extract the middle character from a string. What will you do if the string has an even number of characters?

Hide

```
library(stringr)

x <- "Apple"

med <- ifelse(
  str_length(x) %% 2 == 0,
  ceiling(str_length(x) / 2),
  floor(str_length(x) / 2)
  )

str_sub(x,med,med)
```

```
## [1] "p"
```

4. What does str_wrap() do? When might you want to use it?

This is useful for reformatting text so that it doesn't run off the visible page or you want to indent.

5. What does str_trim() do? What's the opposite of str_trim()?

str_trim() trims whitespace from the beginning or end of a string. The opposite, str_pad(), adds whitespace.

6. Write a function that turns (e.g.) a vector c("a", "b", "c") into the string a, b, and c. Think carefully about what it should do if given a vector of length 0, 1, or 2.

Hide

```
func <- function(x) {
  if( length(x) < 2 ){warning("need string with atleast two characters")}else{
    str_c(x,collapse=", ")
  }
}

x0 <- c()
x1 <- c("a")
x2 <- c("a","b")

func(x2)
```

```
## [1] "a, b"
```

Exercise 14.3.1.1

1. Explain why each of these strings don't match a : "",”\“,”\".

“" doesn't match ‘' because that is a special character. “\” doesn't match” because even though it escaped the ‘' in the regexp, we wrap it in a string first, and a string requires more escaping outside the special behavior. “\" doesn't match a literal ” because in strings ” is also used as an escape so we need”\\" to match a”.

2. How would you match the sequence "' ?

Hide

```
x <- "\"\'\\"
x
```

```
## [1] "\"'\\"
```

Hide

```
str_view(x,"\\\"\\'\\\\")
```

"'\

Escaping " three times, once for escaping the speical behavior then because we're in a string and then because we're escaping special behavior in a string.

Escaping ’ twice, twice for escaping special behavior in the regexp and then in the string.

Escaping for the reason stated in the question above.

3. What patterns will the regular expression `{r chapter14,eval=F}\..\..\..` match? How would you represent it as a string?

Hide

```
x <- "\\..\\..\\.."
x
```

```
## [1] "\\..\\..\\.."
```

Can't figure out how to get rid of that second backslash...

Exercise 14.3.2.1

1. How would you match the literal string "^"?

Hide

```
x <- '"$^$"'

str_view(x,'"$^$"')
```

"$^$"

I think that matches it...

2. Given the corpus of common words in stringr::words, create regular expressions that find all words that:

- Start with "y".
- End with "x"
- Are exactly three letters long. (Don't cheat by using str_length()!)
- Have seven letters or more.
- Since this list is long, you might want to use the match argument to str_view() to show only the matching or non-matching words.

Hide

```
words <- stringr::words

str_view(words,"^y",match=T)
```

year

yes

yesterday

yet

you

young

Hide

```
str_view(words,"x$",match=T)
```

box

sex

six

tax

Hide

```
str_view(words,"\\b[[:alpha:]]{3}\\b",match = T)
```

act

add

age

ago

air

all

and

any

arm

art

ask

bad

bag

bar

bed

bet

big

bit

box

boy

bus

but

buy

can

car

cat

cup

cut

dad

day

die

dog

dry

due

eat

egg

end

eye

far

few

fit

fly

for

fun

gas

get

god

guy

hit

hot

how

job

key

kid

lad

law

lay

leg

let

lie

lot

low

man

may

mrs

new

non

not

now

odd

off

old

one

out

own

pay

per

put

red

rid

run

say

see

set

sex

she

sir

sit

six

son

sun

tax

tea

ten

the

tie

too

top

try

two

use

war

way

wee

who

why

win

yes

yet

you

Hide

```
str_match(words,"\\b[[:alpha:]]{7,}\\b")
```

```
##        [,1]
##   [1,] NA
##   [2,] NA
##   [3,] NA
##   [4,] "absolute"
##   [5,] NA
##   [6,] "account"
##   [7,] "achieve"
##   [8,] NA
##   [9,] NA
##  [10,] NA
##  [11,] NA
##  [12,] NA
##  [13,] "address"
##  [14,] NA
##  [15,] "advertise"
##  [16,] NA
##  [17,] NA
##  [18,] NA
##  [19,] "afternoon"
##  [20,] NA
##  [21,] "against"
##  [22,] NA
##  [23,] NA
##  [24,] NA
##  [25,] NA
##  [26,] NA
##  [27,] NA
##  [28,] NA
##  [29,] NA
##  [30,] NA
##  [31,] "already"
##  [32,] "alright"
##  [33,] NA
##  [34,] "although"
##  [35,] NA
##  [36,] "america"
##  [37,] NA
##  [38,] NA
##  [39,] "another"
##  [40,] NA
##  [41,] NA
##  [42,] NA
##  [43,] "apparent"
##  [44,] NA
##  [45,] NA
##  [46,] "appoint"
##  [47,] "approach"
##  [48,] "appropriate"
##  [49,] NA
##  [50,] NA
##  [51,] NA
##  [52,] NA
##  [53,] "arrange"
##  [54,] NA
##  [55,] NA
##  [56,] NA
##  [57,] "associate"
```

```
##  [58,] NA
##  [59,] NA
##  [60,] NA
##  [61,] "authority"
##  [62,] "available"
##  [63,] NA
##  [64,] NA
##  [65,] NA
##  [66,] NA
##  [67,] NA
##  [68,] NA
##  [69,] NA
##  [70,] "balance"
##  [71,] NA
##  [72,] NA
##  [73,] NA
##  [74,] NA
##  [75,] NA
##  [76,] NA
##  [77,] NA
##  [78,] NA
##  [79,] NA
##  [80,] "because"
##  [81,] NA
##  [82,] NA
##  [83,] NA
##  [84,] NA
##  [85,] NA
##  [86,] "believe"
##  [87,] "benefit"
##  [88,] NA
##  [89,] NA
##  [90,] "between"
##  [91,] NA
##  [92,] NA
##  [93,] NA
##  [94,] NA
##  [95,] NA
##  [96,] NA
##  [97,] NA
##  [98,] NA
##  [99,] NA
## [100,] NA
## [101,] NA
## [102,] NA
## [103,] NA
## [104,] NA
## [105,] NA
## [106,] NA
## [107,] NA
## [108,] NA
## [109,] NA
## [110,] NA
## [111,] NA
## [112,] "brilliant"
## [113,] NA
## [114,] "britain"
## [115,] "brother"
```

```
## [116,] NA
## [117,] NA
## [118,] NA
## [119,] "business"
## [120,] NA
## [121,] NA
## [122,] NA
## [123,] NA
## [124,] NA
## [125,] NA
## [126,] NA
## [127,] NA
## [128,] NA
## [129,] NA
## [130,] NA
## [131,] NA
## [132,] NA
## [133,] NA
## [134,] NA
## [135,] NA
## [136,] NA
## [137,] "certain"
## [138,] NA
## [139,] "chairman"
## [140,] NA
## [141,] NA
## [142,] NA
## [143,] "character"
## [144,] NA
## [145,] NA
## [146,] NA
## [147,] NA
## [148,] NA
## [149,] NA
## [150,] NA
## [151,] "Christmas"
## [152,] NA
## [153,] NA
## [154,] NA
## [155,] NA
## [156,] NA
## [157,] NA
## [158,] NA
## [159,] NA
## [160,] NA
## [161,] NA
## [162,] NA
## [163,] NA
## [164,] NA
## [165,] NA
## [166,] "colleague"
## [167,] "collect"
## [168,] "college"
## [169,] NA
## [170,] NA
## [171,] "comment"
## [172,] NA
## [173,] "committee"
```

```
## [174,] NA
## [175,] "community"
## [176,] "company"
## [177,] "compare"
## [178,] "complete"
## [179,] "compute"
## [180,] "concern"
## [181,] "condition"
## [182,] NA
## [183,] "consider"
## [184,] "consult"
## [185,] "contact"
## [186,] "continue"
## [187,] "contract"
## [188,] "control"
## [189,] "converse"
## [190,] NA
## [191,] NA
## [192,] NA
## [193,] "correct"
## [194,] NA
## [195,] NA
## [196,] "council"
## [197,] NA
## [198,] "country"
## [199,] NA
## [200,] NA
## [201,] NA
## [202,] NA
## [203,] NA
## [204,] NA
## [205,] NA
## [206,] NA
## [207,] "current"
## [208,] NA
## [209,] NA
## [210,] NA
## [211,] NA
## [212,] NA
## [213,] NA
## [214,] NA
## [215,] NA
## [216,] NA
## [217,] NA
## [218,] "decision"
## [219,] NA
## [220,] "definite"
## [221,] NA
## [222,] "department"
## [223,] NA
## [224,] "describe"
## [225,] NA
## [226,] NA
## [227,] "develop"
## [228,] NA
## [229,] "difference"
## [230,] "difficult"
## [231,] NA
```

```
## [232,] NA
## [233,] "discuss"
## [234,] "district"
## [235,] NA
## [236,] NA
## [237,] NA
## [238,] "document"
## [239,] NA
## [240,] NA
## [241,] NA
## [242,] NA
## [243,] NA
## [244,] NA
## [245,] NA
## [246,] NA
## [247,] NA
## [248,] NA
## [249,] NA
## [250,] NA
## [251,] NA
## [252,] NA
## [253,] NA
## [254,] NA
## [255,] NA
## [256,] NA
## [257,] "economy"
## [258,] "educate"
## [259,] NA
## [260,] NA
## [261,] NA
## [262,] NA
## [263,] NA
## [264,] "electric"
## [265,] NA
## [266,] NA
## [267,] NA
## [268,] "encourage"
## [269,] NA
## [270,] NA
## [271,] "english"
## [272,] NA
## [273,] NA
## [274,] NA
## [275,] "environment"
## [276,] NA
## [277,] "especial"
## [278,] NA
## [279,] NA
## [280,] "evening"
## [281,] NA
## [282,] NA
## [283,] "evidence"
## [284,] NA
## [285,] "example"
## [286,] NA
## [287,] NA
## [288,] "exercise"
## [289,] NA
```

```
## [290,] NA
## [291,] "expense"
## [292,] "experience"
## [293,] "explain"
## [294,] "express"
## [295,] NA
## [296,] NA
## [297,] NA
## [298,] NA
## [299,] NA
## [300,] NA
## [301,] NA
## [302,] NA
## [303,] NA
## [304,] NA
## [305,] NA
## [306,] NA
## [307,] NA
## [308,] NA
## [309,] NA
## [310,] NA
## [311,] NA
## [312,] NA
## [313,] NA
## [314,] NA
## [315,] NA
## [316,] NA
## [317,] "finance"
## [318,] NA
## [319,] NA
## [320,] NA
## [321,] NA
## [322,] NA
## [323,] NA
## [324,] NA
## [325,] NA
## [326,] NA
## [327,] NA
## [328,] NA
## [329,] NA
## [330,] NA
## [331,] NA
## [332,] NA
## [333,] NA
## [334,] NA
## [335,] NA
## [336,] "fortune"
## [337,] "forward"
## [338,] NA
## [339,] NA
## [340,] NA
## [341,] NA
## [342,] NA
## [343,] NA
## [344,] NA
## [345,] NA
## [346,] NA
## [347,] "function"
```

```
## [348,] NA
## [349,] "further"
## [350,] NA
## [351,] NA
## [352,] NA
## [353,] NA
## [354,] "general"
## [355,] "germany"
## [356,] NA
## [357,] NA
## [358,] NA
## [359,] NA
## [360,] NA
## [361,] NA
## [362,] NA
## [363,] "goodbye"
## [364,] NA
## [365,] NA
## [366,] NA
## [367,] NA
## [368,] NA
## [369,] NA
## [370,] NA
## [371,] NA
## [372,] NA
## [373,] NA
## [374,] NA
## [375,] NA
## [376,] NA
## [377,] NA
## [378,] NA
## [379,] NA
## [380,] NA
## [381,] NA
## [382,] NA
## [383,] NA
## [384,] NA
## [385,] NA
## [386,] NA
## [387,] NA
## [388,] NA
## [389,] NA
## [390,] NA
## [391,] NA
## [392,] NA
## [393,] NA
## [394,] NA
## [395,] "history"
## [396,] NA
## [397,] NA
## [398,] "holiday"
## [399,] NA
## [400,] NA
## [401,] NA
## [402,] NA
## [403,] "hospital"
## [404,] NA
## [405,] NA
```

```
## [406,] NA
## [407,] NA
## [408,] "however"
## [409,] NA
## [410,] "hundred"
## [411,] "husband"
## [412,] NA
## [413,] "identify"
## [414,] NA
## [415,] "imagine"
## [416,] "important"
## [417,] "improve"
## [418,] NA
## [419,] "include"
## [420,] NA
## [421,] "increase"
## [422,] NA
## [423,] "individual"
## [424,] "industry"
## [425,] NA
## [426,] NA
## [427,] "instead"
## [428,] NA
## [429,] "interest"
## [430,] NA
## [431,] "introduce"
## [432,] NA
## [433,] "involve"
## [434,] NA
## [435,] NA
## [436,] NA
## [437,] NA
## [438,] NA
## [439,] NA
## [440,] NA
## [441,] NA
## [442,] NA
## [443,] NA
## [444,] NA
## [445,] NA
## [446,] NA
## [447,] NA
## [448,] NA
## [449,] "kitchen"
## [450,] NA
## [451,] NA
## [452,] NA
## [453,] NA
## [454,] NA
## [455,] NA
## [456,] "language"
## [457,] NA
## [458,] NA
## [459,] NA
## [460,] NA
## [461,] NA
## [462,] NA
## [463,] NA
```

```
## [464,] NA
## [465,] NA
## [466,] NA
## [467,] NA
## [468,] NA
## [469,] NA
## [470,] NA
## [471,] NA
## [472,] NA
## [473,] NA
## [474,] NA
## [475,] NA
## [476,] NA
## [477,] NA
## [478,] NA
## [479,] NA
## [480,] NA
## [481,] NA
## [482,] NA
## [483,] NA
## [484,] NA
## [485,] NA
## [486,] NA
## [487,] NA
## [488,] NA
## [489,] NA
## [490,] NA
## [491,] NA
## [492,] NA
## [493,] NA
## [494,] NA
## [495,] NA
## [496,] NA
## [497,] "machine"
## [498,] NA
## [499,] NA
## [500,] NA
## [501,] NA
## [502,] NA
## [503,] NA
## [504,] NA
## [505,] NA
## [506,] NA
## [507,] NA
## [508,] NA
## [509,] NA
## [510,] NA
## [511,] NA
## [512,] "meaning"
## [513,] "measure"
## [514,] NA
## [515,] NA
## [516,] "mention"
## [517,] NA
## [518,] NA
## [519,] NA
## [520,] NA
## [521,] "million"
```

```
## [522,] NA
## [523,] "minister"
## [524,] NA
## [525,] NA
## [526,] NA
## [527,] NA
## [528,] NA
## [529,] NA
## [530,] NA
## [531,] NA
## [532,] NA
## [533,] "morning"
## [534,] NA
## [535,] NA
## [536,] NA
## [537,] NA
## [538,] NA
## [539,] NA
## [540,] NA
## [541,] NA
## [542,] NA
## [543,] NA
## [544,] NA
## [545,] NA
## [546,] "necessary"
## [547,] NA
## [548,] NA
## [549,] NA
## [550,] NA
## [551,] NA
## [552,] NA
## [553,] NA
## [554,] NA
## [555,] NA
## [556,] NA
## [557,] NA
## [558,] NA
## [559,] NA
## [560,] NA
## [561,] NA
## [562,] NA
## [563,] NA
## [564,] NA
## [565,] "obvious"
## [566,] "occasion"
## [567,] NA
## [568,] NA
## [569,] NA
## [570,] NA
## [571,] NA
## [572,] NA
## [573,] NA
## [574,] NA
## [575,] NA
## [576,] NA
## [577,] NA
## [578,] NA
## [579,] NA
```

```
## [580,] "operate"
## [581,] "opportunity"
## [582,] NA
## [583,] NA
## [584,] NA
## [585,] "organize"
## [586,] "original"
## [587,] NA
## [588,] "otherwise"
## [589,] NA
## [590,] NA
## [591,] NA
## [592,] NA
## [593,] NA
## [594,] NA
## [595,] NA
## [596,] NA
## [597,] NA
## [598,] "paragraph"
## [599,] NA
## [600,] NA
## [601,] NA
## [602,] NA
## [603,] "particular"
## [604,] NA
## [605,] NA
## [606,] NA
## [607,] NA
## [608,] NA
## [609,] "pension"
## [610,] NA
## [611,] NA
## [612,] "percent"
## [613,] "perfect"
## [614,] "perhaps"
## [615,] NA
## [616,] NA
## [617,] "photograph"
## [618,] NA
## [619,] "picture"
## [620,] NA
## [621,] NA
## [622,] NA
## [623,] NA
## [624,] NA
## [625,] NA
## [626,] NA
## [627,] NA
## [628,] NA
## [629,] "politic"
## [630,] NA
## [631,] "position"
## [632,] "positive"
## [633,] "possible"
## [634,] NA
## [635,] NA
## [636,] NA
## [637,] "practise"
```

```
## [638,] "prepare"
## [639,] "present"
## [640,] NA
## [641,] "pressure"
## [642,] "presume"
## [643,] NA
## [644,] "previous"
## [645,] NA
## [646,] NA
## [647,] "private"
## [648,] "probable"
## [649,] "problem"
## [650,] "proceed"
## [651,] "process"
## [652,] "produce"
## [653,] "product"
## [654,] "programme"
## [655,] "project"
## [656,] NA
## [657,] "propose"
## [658,] "protect"
## [659,] "provide"
## [660,] NA
## [661,] NA
## [662,] "purpose"
## [663,] NA
## [664,] NA
## [665,] "quality"
## [666,] "quarter"
## [667,] "question"
## [668,] NA
## [669,] NA
## [670,] NA
## [671,] NA
## [672,] NA
## [673,] NA
## [674,] NA
## [675,] NA
## [676,] NA
## [677,] NA
## [678,] NA
## [679,] NA
## [680,] NA
## [681,] "realise"
## [682,] NA
## [683,] NA
## [684,] "receive"
## [685,] NA
## [686,] NA
## [687,] "recognize"
## [688,] "recommend"
## [689,] NA
## [690,] NA
## [691,] NA
## [692,] NA
## [693,] NA
## [694,] NA
## [695,] "relation"
```

```
## [696,] "remember"
## [697,] NA
## [698,] "represent"
## [699,] "require"
## [700,] "research"
## [701,] "resource"
## [702,] "respect"
## [703,] "responsible"
## [704,] NA
## [705,] NA
## [706,] NA
## [707,] NA
## [708,] NA
## [709,] NA
## [710,] NA
## [711,] NA
## [712,] NA
## [713,] NA
## [714,] NA
## [715,] NA
## [716,] NA
## [717,] NA
## [718,] NA
## [719,] NA
## [720,] NA
## [721,] "saturday"
## [722,] NA
## [723,] NA
## [724,] NA
## [725,] NA
## [726,] "science"
## [727,] NA
## [728,] "scotland"
## [729,] NA
## [730,] NA
## [731,] "secretary"
## [732,] "section"
## [733,] NA
## [734,] NA
## [735,] NA
## [736,] NA
## [737,] NA
## [738,] NA
## [739,] NA
## [740,] "separate"
## [741,] "serious"
## [742,] NA
## [743,] "service"
## [744,] NA
## [745,] NA
## [746,] NA
## [747,] NA
## [748,] NA
## [749,] NA
## [750,] NA
## [751,] NA
## [752,] NA
## [753,] NA
```

```
## [754,] NA
## [755,] NA
## [756,] NA
## [757,] NA
## [758,] NA
## [759,] NA
## [760,] NA
## [761,] NA
## [762,] "similar"
## [763,] NA
## [764,] NA
## [765,] NA
## [766,] NA
## [767,] NA
## [768,] NA
## [769,] NA
## [770,] NA
## [771,] "situate"
## [772,] NA
## [773,] NA
## [774,] NA
## [775,] NA
## [776,] NA
## [777,] NA
## [778,] NA
## [779,] NA
## [780,] NA
## [781,] "society"
## [782,] NA
## [783,] NA
## [784,] NA
## [785,] NA
## [786,] NA
## [787,] NA
## [788,] NA
## [789,] NA
## [790,] NA
## [791,] "special"
## [792,] "specific"
## [793,] NA
## [794,] NA
## [795,] NA
## [796,] NA
## [797,] NA
## [798,] NA
## [799,] NA
## [800,] NA
## [801,] "standard"
## [802,] NA
## [803,] NA
## [804,] "station"
## [805,] NA
## [806,] NA
## [807,] NA
## [808,] NA
## [809,] NA
## [810,] NA
## [811,] "straight"
```

```
## [812,] "strategy"
## [813,] NA
## [814,] NA
## [815,] NA
## [816,] "structure"
## [817,] "student"
## [818,] NA
## [819,] NA
## [820,] NA
## [821,] "subject"
## [822,] "succeed"
## [823,] NA
## [824,] NA
## [825,] "suggest"
## [826,] NA
## [827,] NA
## [828,] NA
## [829,] NA
## [830,] NA
## [831,] "support"
## [832,] "suppose"
## [833,] NA
## [834,] "surprise"
## [835,] NA
## [836,] NA
## [837,] NA
## [838,] NA
## [839,] NA
## [840,] NA
## [841,] NA
## [842,] NA
## [843,] NA
## [844,] NA
## [845,] "telephone"
## [846,] "television"
## [847,] NA
## [848,] NA
## [849,] NA
## [850,] NA
## [851,] "terrible"
## [852,] NA
## [853,] NA
## [854,] NA
## [855,] NA
## [856,] NA
## [857,] NA
## [858,] "therefore"
## [859,] NA
## [860,] NA
## [861,] NA
## [862,] "thirteen"
## [863,] NA
## [864,] NA
## [865,] NA
## [866,] NA
## [867,] "thousand"
## [868,] NA
## [869,] "through"
```

```
## [870,] NA
## [871,] "thursday"
## [872,] NA
## [873,] NA
## [874,] NA
## [875,] NA
## [876,] "together"
## [877,] "tomorrow"
## [878,] "tonight"
## [879,] NA
## [880,] NA
## [881,] NA
## [882,] NA
## [883,] NA
## [884,] NA
## [885,] NA
## [886,] "traffic"
## [887,] NA
## [888,] "transport"
## [889,] NA
## [890,] NA
## [891,] NA
## [892,] "trouble"
## [893,] NA
## [894,] NA
## [895,] NA
## [896,] "tuesday"
## [897,] NA
## [898,] NA
## [899,] NA
## [900,] NA
## [901,] NA
## [902,] NA
## [903,] "understand"
## [904,] NA
## [905,] NA
## [906,] NA
## [907,] "university"
## [908,] NA
## [909,] NA
## [910,] NA
## [911,] NA
## [912,] NA
## [913,] NA
## [914,] NA
## [915,] "various"
## [916,] NA
## [917,] NA
## [918,] NA
## [919,] "village"
## [920,] NA
## [921,] NA
## [922,] NA
## [923,] NA
## [924,] NA
## [925,] NA
## [926,] NA
## [927,] NA
```

```
## [928,] NA
## [929,] NA
## [930,] NA
## [931,] NA
## [932,] NA
## [933,] NA
## [934,] NA
## [935,] NA
## [936,] "wednesday"
## [937,] NA
## [938,] NA
## [939,] NA
## [940,] "welcome"
## [941,] NA
## [942,] NA
## [943,] NA
## [944,] NA
## [945,] NA
## [946,] "whether"
## [947,] NA
## [948,] NA
## [949,] NA
## [950,] NA
## [951,] NA
## [952,] NA
## [953,] NA
## [954,] NA
## [955,] NA
## [956,] NA
## [957,] NA
## [958,] NA
## [959,] NA
## [960,] NA
## [961,] NA
## [962,] "without"
## [963,] NA
## [964,] NA
## [965,] NA
## [966,] NA
## [967,] NA
## [968,] NA
## [969,] NA
## [970,] NA
## [971,] NA
## [972,] NA
## [973,] NA
## [974,] NA
## [975,] NA
## [976,] NA
## [977,] "yesterday"
## [978,] NA
## [979,] NA
## [980,] NA
```

Go to ?regex in the base package to see more examples and details about regular expressions.

Exercise 14.3.3.1

1. Create regular expressions to find all words that:

- Start with a vowel.
- That only contain consonants. (Hint: thinking about matching “not”-vowels.)
- End with ed, but not with eed.
- End with ing or ise.

Hide

```
sub_words <- sample(words,20)

str_view(sub_words,"^[aeiou]",match=T)
```

and

eleven

art

old

instead

Hide

```
str_view(sub_words,"^[aeiou]",match=T)
```

and

eleven

art

old

instead

Hide

```
str_view(sub_words,"[^e]ed$",match=T)
```

Hide

```
str_view(sub_words,"ing$|ise$",match=T)
```

rise

practise

2. Empirically verify the rule “i before e except after c”.

Hide

```
str_view(words,"cei",match=T)
```

receive

Hide

```
str_view(words,"cie",match=T)
```

science

society

Hide

```
str_view(words,"iec",match=T)
```

piece

Hide

```
str_view(words,"eic",match=T)
```

I think it's only "iec" not "eic".

3. Is "q" always followed by a "u"?

Hide

```
str_view(words,"uq|qu",match=T)
```

equal

quality

quarter

question

quick

quid

quiet

quite

require

square

yes

4. Write a regular expression that matches a word if it's probably written in British English, not American English.

colour

Hide

```
str_view(words,"colour",match=T)
```

colour

5. Create a regular expression that will match telephone numbers as commonly written in your country.

Hide

```
("[0-9]{3}-[0-9]{3}-[0-9]{4}")
```

```
## [1] "[0-9]{3}-[0-9]{3}-[0-9]{4}"
```

Exercise 14.3.4.1

1. Describe the equivalents of ?, +, * in {m,n} form.

? <- {0,1}

- <- {1,}
- <- {0.}

2. Describe in words what these regular expressions match: (read carefully to see if I'm using a regular expression or a string that defines a regular expression.)

^.*$ # This matches a 0 or more periods at the beginning of the string to the end

“\{.+\}” #this string of a regex that matches one or more periods

-- #this regex matches 4 digits, a dash, 2 digits, a dash then 2 digits

“\\{4}” # this regex is in a string and matches 4 's.

3. Create regular expressions to find all words that:

- Start with three consonants.
- Have three or more vowels in a row.
- Have two or more vowel-consonant pairs in a row.

Hide

```
str_view(words,"^[^aeiou]{3}.*",match=T)
```

year

yes

yesterday

yet

you

young

Hide

```
str_view(words,"[aeiou]{3,}",match=T)
```

box

sex

six

tax

Hide

```
str_view(words,"([aeiou][^aeiou]){2,}.*",match=T)
```

act

add

age

ago

air

all

and

any
arm
art
ask
bad
bag
bar
bed
bet
big
bit
box
boy
bus
but
buy
can
car
cat
cup
cut
dad
day
die
dog
dry
due
eat
egg
end
eye
far
few
fit
fly
for
fun
gas
get
god

guy

hit

hot

how

job

key

kid

lad

law

lay

leg

let

lie

lot

low

man

may

mrs

new

non

not

now

odd

off

old

one

out

own

pay

per

put

red

rid

run

say

see

set

sex

she

sir

`sit`

`six`

`son`

`sun`

`tax`

`tea`

`ten`

`the`

`tie`

`too`

`top`

`try`

`two`

`use`

`war`

`way`

`wee`

`who`

`why`

`win`

`yes`

`yet`

`you`

4. Solve the beginner regexp crosswords at https://regexcrossword.com/challenges/beginner (https://regexcrossword.com/challenges/beginner).

No thanks I'll stick with this.

Exercise 14.3.5.1

1. Describe, in words, what these expressions will match:

(.) #words with two of the same letter once

"(.)(.)\2\1" # words with any two letters repeated twice consecutively once in the word

(..) # words with two letters repeated once

"(.).\1.\1" # words with any letter, a specific letter repeated once then another letter repeated once

"(.)(.)(.).*\3\2\1" # words with one letter, then two of the same letter, 3 of the same letter then 0 or more of a letter.

2. Construct regular expressions to match words that:

- Start and end with the same character.
- Contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice.)

- Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)

Hide

```
str_view(words,"^(.).*\\1$",match=T) #couldn't figure out how top feasibly expand to any character.
```

year

yes

yesterday

yet

you

young

Hide

```
str_view(words,"(..).*\\1",match = T)
```

box

sex

six

tax

Hide

```
str_view(words,"(.).*\\1.*\\1",match = T)
```

act

add

age

ago

air

all

and

any

arm

art

ask

bad

bag

bar

bed

bet

big

bit

box

boy

bus

but

buy

can

car

cat

cup

cut

dad

day

die

dog

dry

due

eat

egg

end

eye

far

few

fit

fly

for

fun

gas

get

god

guy

hit

hot

how

job

key

kid

lad

law

lay

leg

let

lie

lot

low

man

may

mrs

new

non

not

now

odd

off

old

one

out

own

pay

per

put

red

rid

run

say

see

set

sex

she

sir

sit

six

son

sun

tax

tea

ten

the

tie

too

top

try

two

use

`war`

`way`

`wee`

`who`

`why`

`win`

`yes`

`yet`

`you`

Exercise 14.4.2

1. For each of the following challenges, try solving it by using both a single regular expression, and a combination of multiple str_detect() calls.

- Find all words that start or end with x.

Hide

```
# one regex
words[str_detect(words, "^x|x$")]
```

```
## [1] "box" "sex" "six" "tax"
```

Hide

```
# split regex into parts
start_with_x <- str_detect(words, "^x")
end_with_x <- str_detect(words, "x$")
words[start_with_x | end_with_x]
```

```
## [1] "box" "sex" "six" "tax"
```

- Find all words that start with a vowel and end with a consonant.

Hide

```
str_subset(words, "^[aeiou].*[^aeiou]$") %>% head()
```

```
## [1] "about"   "accept"  "account" "across"  "act"     "actual"
```

Hide

```
start_with_vowel <- str_detect(words, "^[aeiou]")
end_with_consonant <- str_detect(words, "[^aeiou]$")
words[start_with_vowel & end_with_consonant] %>% head()
```

```
## [1] "about"   "accept"  "account" "across"  "act"     "actual"
```

- Are there any words that contain at least one of each different vowel?

Hide

```
library(purrr)

pattern <-
  cross(rerun(5, c("a", "e", "i", "o", "u")),
        .filter = function(...) {
          x <- as.character(unlist(list(...)))
          length(x) != length(unique(x))
        }) %>%
  map_chr(~ str_c(unlist(.x), collapse = ".*")) %>%
  str_c(collapse = "|")

str_subset(words, pattern)
```

```
## character(0)
```

Hide

```
words[str_detect(words, "a") &
        str_detect(words, "e") &
        str_detect(words, "i") &
        str_detect(words, "o") &
        str_detect(words, "u")]
```

```
## character(0)
```

all from jarnold

2. What word has the highest number of vowels? What word has the highest proportion of vowels? (Hint: what is the denominator?)

Hide

```
ind <- which.max(str_count(words,"[aeiou]"))

words[ind]
```

```
## [1] "appropriate"
```

Exercise 14.4.3.1

1. In the previous example, you might have noticed that the regular expression matched "flickered", which is not a colour. Modify the regex to fix the problem.

Add the "" before and after the pattern

Hide

```
colours <- c("red", "orange", "yellow", "green", "blue", "purple")

colour_match <- str_c(colours, collapse = "|")

more <- sentences[str_count(sentences, colour_match) > 1]
str_view_all(more, colour_match)
```

and

eleven

art

old

instead

Hide

```
colour_match2 <- str_c("\\b(", str_c(colours, collapse = "|"), ")\\b")

more2 <- sentences[str_count(sentences, colour_match) > 1]

str_view_all(more2, colour_match2, match = TRUE)
```

and

eleven

art

old

instead

2. From the Harvard sentences data, extract:

- The first word from each sentence.
- All words ending in ing.
- All plurals.

Hide

```
str_extract(sentences, "[a-zA-X]+") %>% head()
```

```
## [1] "The"   "Glue"  "It"    "These" "Rice"  "The"
```

Hide

```
pattern <- "\\b[A-Za-z]+ing\\b"
sentences_with_ing <- str_detect(sentences, pattern)
unique(unlist(str_extract_all(sentences[sentences_with_ing], pattern))) %>%
  head()
```

```
## [1] "spring"  "evening" "morning" "winding" "living"  "king"
```

Hide

```
unique(unlist(str_extract_all(sentences, "\\b[A-Za-z]{3,}s\\b"))) %>%
  head()
```

```
## [1] "planks" "days"   "bowls"  "lemons" "makes"  "hogs"
```

Exercise 14.4.4.1

1. Find all words that come after a "number" like "one", "two", "three" etc. Pull out both the number and the word.

from jarnold

Hide

```
numword <- "(one|two|three|four|five|six|seven|eight|nine|ten) +(\\S+)"
sentences[str_detect(sentences, numword)] %>%
  str_extract(numword)
```

```
##  [1] "ten served"    "one over"      "seven books"   "two met"
##  [5] "two factors"   "one and"       "three lists"   "seven is"
##  [9] "two when"      "one floor."    "ten inches."   "one with"
## [13] "one war"       "one button"    "six minutes."  "ten years"
## [17] "one in"        "ten chased"    "one like"      "two shares"
## [21] "two distinct"  "one costs"     "ten two"       "five robins."
## [25] "four kinds"    "one rang"      "ten him."      "three story"
## [29] "ten by"        "one wall."     "three inches"  "ten your"
## [33] "six comes"     "one before"    "three batches" "two leaves."
```

2. Find all contractions. Separate out the pieces before and after the apostrophe.

Hide

```
contraction <- "([A-Za-z]+)'([A-Za-z]+)"

sentences %>%
  `[`(str_detect(sentences, contraction)) %>%
  str_extract(contraction)
```

```
##  [1] "It's"       "man's"      "don't"      "store's"    "workmen's"
##  [6] "Let's"      "sun's"      "child's"    "king's"     "It's"
## [11] "don't"      "queen's"    "don't"      "pirate's"   "neighbor's"
```

Exercise 14.4.5.1

1. Replace all forward slashes in a string with backslashes.

Hide

```
x <- c("apples, pears, and bananas")
str_split(x, ", +(and +)?")[[1]]
```

```
## [1] "apples"  "pears"   "bananas"
```

2. Implement a simple version of str_to_lower() using replace_all()

Can't find a replce_all function...

3. Switch the first and last letters in words. Which of those strings are still words?

Couldn't figure out how to do this...

Exercise 14.4.6.1

1. Split up a string like “apples, pears, and bananas” into individual components.

Hide

```
x <- c("apples, pears, and bananas")
str_split(x, ", +(and +)?")[[1]]
```

```
## [1] "apples"  "pears"   "bananas"
```

2. Why is it better to split up by boundary(“word”) than " “?

Splitting by boundary(“word”) splits on punctuation and not just whitespace.

3. What does splitting with an empty string (“”) do? Experiment, and then read the documentation.

Hide

```
str_split("ab. cd|agt", "")[[1]]
```

```
##  [1] "a" "b" "." " " "c" "d" "|" "a" "g" "t"
```

from jarnold.

Exercise 14.5.1

1. How would you find all strings containing with regex() vs. with fixed()?

Hide

```
str_subset(c("a\\b", "ab"), "\\\\")
```

```
## [1] "a\\b"
```

Hide

```
str_subset(c("a\\b", "ab"), fixed("\\"))
```

```
## [1] "a\\b"
```

2. What are the five most common words in sentences?

Hide

```
library(dplyr)
str_extract_all(sentences, boundary("word")) %>%
  unlist() %>%
  str_to_lower() %>%
  tibble() %>%
  set_names("word") %>%
  group_by(word) %>%
  count(sort = TRUE) %>%
  head(5)
```

```
## # A tibble: 5 x 2
## # Groups:   word [5]
##    word     n
##   <chr> <int>
## 1   the   751
## 2     a   202
## 3    of   132
## 4    to   123
## 5   and   118
```

from jarnold.

Exercise 14.7.1

1. Find the stringi functions that:

- Count the number of words

stri_count_words

- Find duplicated strings

stri_duplicated

- Generate random text

There are several functions beginning with stri_rand_. stri_rand_lipsum generates lorem ipsum text, stri_rand_strings generates random strings, stri_rand_shuffle randomly shuffles the code points in the text.

2. How do you control the language that stri_sort() uses for sorting?

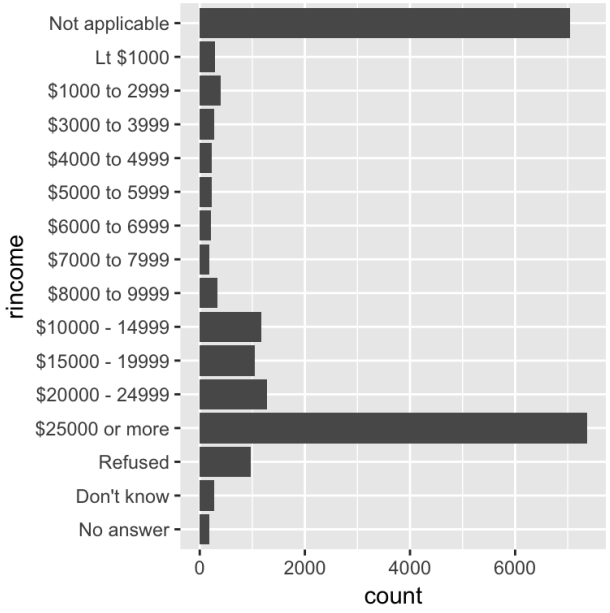
Use the locale argument to the opts_collator argument. # 15. Factors

Exercise 15.3.1

1. Explore the distribution of rincome (reported income). What makes the default bar chart hard to understand? How could you improve the plot?

Hide

```
library(forcats)
library(tidyverse)
library(ggplot2)

ggplot(gss_cat,aes(rincome)) +
  geom_bar(drop=F) +
  coord_flip()
```



This is ok-would be better if we sorted by decreasing order or in another meaningful way.

2. What is the most common relig in this survey? What's the most common partyid?

Hide

```
gss_cat %>% group_by(relig) %>% count() %>% arrange(desc(n))
```

```
## # A tibble: 15 x 2
## # Groups:   relig [15]
##                      relig     n
##                     <fctr> <int>
##  1              Protestant 10846
##  2                Catholic  5124
##  3                    None  3523
##  4               Christian   689
##  5                  Jewish   388
##  6                   Other   224
##  7                Buddhism   147
##  8 Inter-nondenominational   109
##  9            Moslem/islam   104
## 10      Orthodox-christian    95
## 11               No answer    93
## 12                Hinduism    71
## 13           Other eastern    32
## 14         Native american    23
## 15              Don't know    15
```

Hide

```
gss_cat %>% group_by(partyid) %>% count() %>% arrange(desc(n))
```

```
## # A tibble: 10 x 2
## # Groups:   partyid [10]
##                partyid     n
##                 <fctr> <int>
##  1        Independent  4119
##  2   Not str democrat  3690
##  3    Strong democrat  3490
##  4 Not str republican  3032
##  5       Ind,near dem  2499
##  6  Strong republican  2314
##  7       Ind,near rep  1791
##  8        Other party   393
##  9          No answer   154
## 10         Don't know     1
```

3. Which relig does denom (denomination) apply to? How can you find out with a table? How can you find out with a visualisation?

Hide

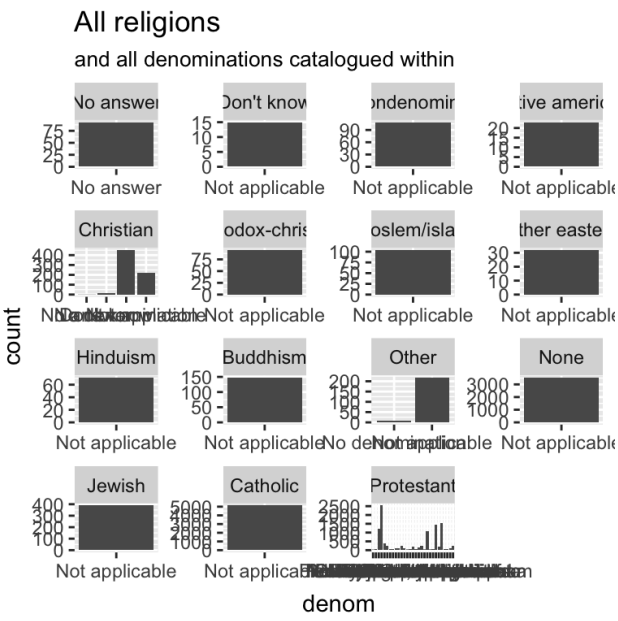
```
gss_cat %>% group_by(denom) %>% count()
```

```
## # A tibble: 30 x 2
## # Groups:   denom [30]
##                    denom     n
##                   <fctr> <int>
##  1            No answer   117
##  2           Don't know    52
##  3      No denomination  1683
##  4                Other  2534
##  5            Episcopal   397
##  6   Presbyterian-dk wh   244
##  7 Presbyterian, merged    67
##  8   Other presbyterian    47
##  9 United pres ch in us   110
## 10 Presbyterian c in us   104
## # ... with 20 more rows
```

My educated guess is different sects of the religions but let's try to use our programming skillset.

Hide

```
gss_cat %>%
  ggplot() +
  geom_bar(aes(denom)) +
  scale_x_discrete(drop=T) +
  facet_wrap(~relig,scales = "free") +
  ggtitle("All religions",subtitle = "and all denominations catalogued within")
```



Hide

```
  coord_flip()
```

```
## <ggproto object: Class CoordFlip, CoordCartesian, Coord>
##     aspect: function
##     distance: function
##     expand: TRUE
##     is_linear: function
##     labels: function
##     limits: list
##     range: function
##     render_axis_h: function
##     render_axis_v: function
##     render_bg: function
##     render_fg: function
##     train: function
##     transform: function
##     super:  <ggproto object: Class CoordFlip, CoordCartesian, Coord>
```

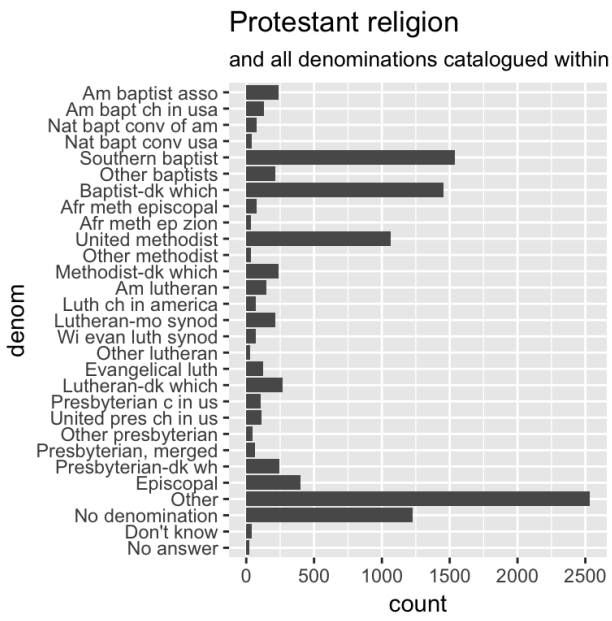
Hide

```
denoms <- levels(gss_cat$denom)

tmp <- filter(gss_cat,relig=="Protestant") %>% select(denom) %>% unique()
protestant_denoms <- pull(tmp,denom)

setdiff(denoms,protestant_denoms)
```

```
## [1] "Not applicable"
```

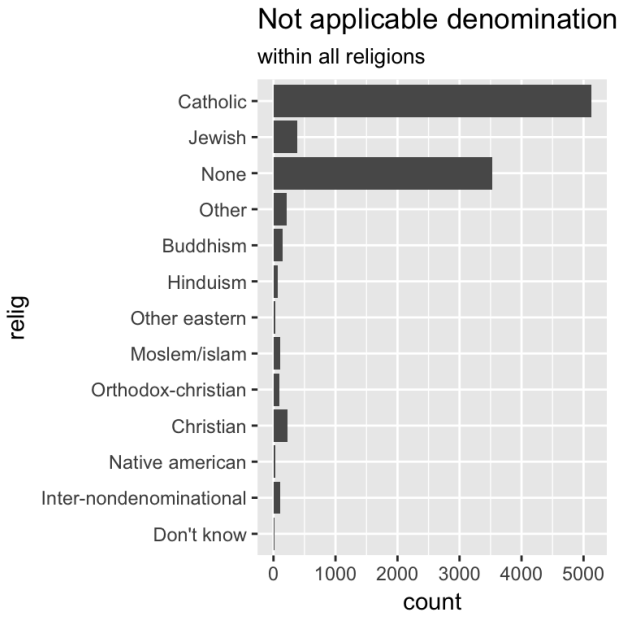
Hide

```
filter(gss_cat,relig=="Protestant") %>%
  ggplot() +
  geom_bar(aes(denom)) +
  ggtitle("Protestant religion",subtitle = "and all denominations catalogued within") +
  coord_flip()
```



Hide

```
filter(gss_cat,denom=="Not applicable") %>%
  ggplot() +
  geom_bar(aes(relig)) +
  ggtitle("Not applicable denomination",subtitle = "within all religions") +
  coord_flip()
```

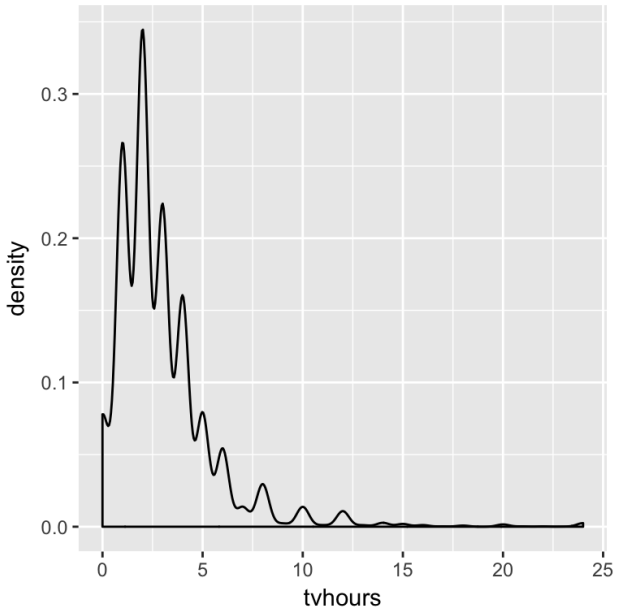


Exercise 15.4.1

1. There are some suspiciously high numbers in tvhours. Is the mean a good summary?

Hide

```
gss_cat %>%
  ggplot() +
  geom_density(aes(tvhours))
```



The distribution of tvhours is a little wonkey-a lot of peaks and valleys. But depending on your question the mean (or median) wuld be what you want to use. Or viewing the histogram for different groups.

2. For each factor in gss_cat identify whether the order of the levels is arbitrary or principled.

Hide

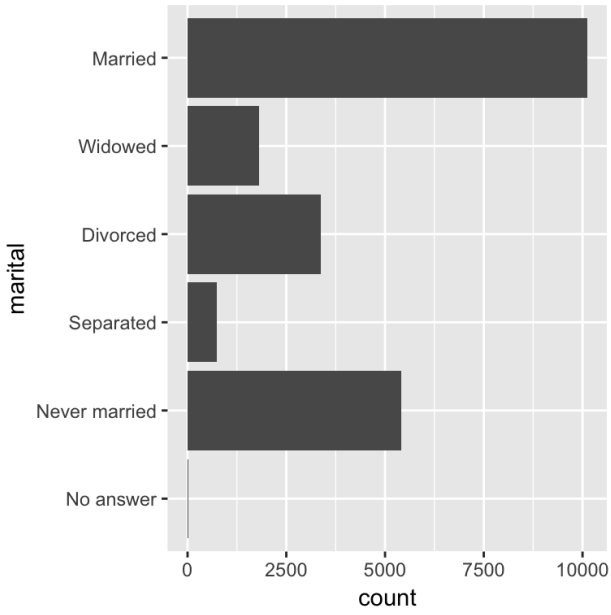
```
gss_cat
```

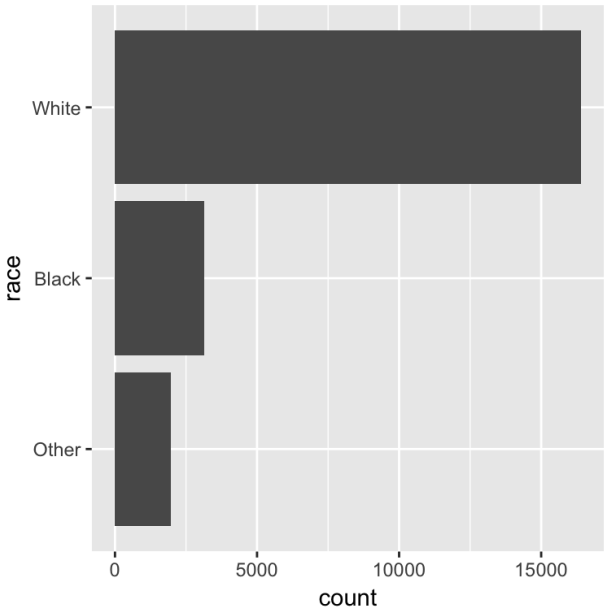
```
## # A tibble: 21,483 x 9
##     year       marital   age   race        rincome            partyid
##    <int>        <fctr> <int> <fctr>         <fctr>             <fctr>
##  1  2000 Never married    26  White  $8000 to 9999       Ind,near rep
##  2  2000      Divorced    48  White  $8000 to 9999 Not str republican
##  3  2000       Widowed    67  White Not applicable        Independent
##  4  2000 Never married    39  White Not applicable       Ind,near rep
##  5  2000      Divorced    25  White Not applicable   Not str democrat
##  6  2000       Married    25  White $20000 - 24999    Strong democrat
##  7  2000 Never married    36  White $25000 or more Not str republican
##  8  2000      Divorced    44  White  $7000 to 7999       Ind,near dem
##  9  2000       Married    44  White $25000 or more   Not str democrat
## 10  2000       Married    47  White $25000 or more  Strong republican
## # ... with 21,473 more rows, and 3 more variables: relig <fctr>,
## #   denom <fctr>, tvhours <int>
```

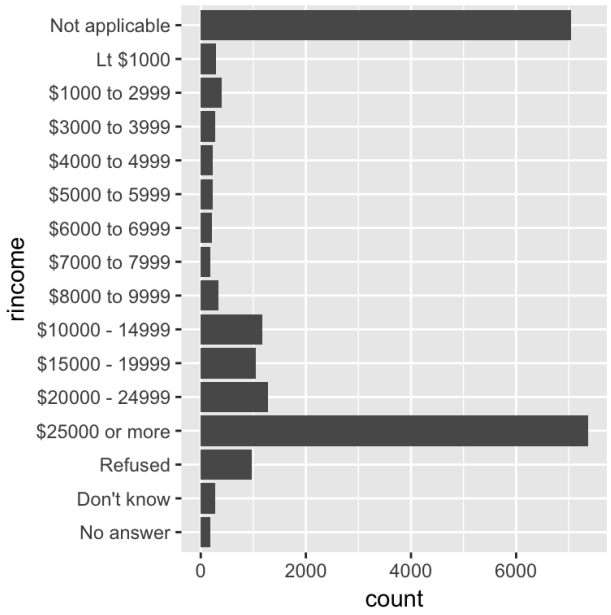
We want to look at marital, race, rincome, partyid, relig, and denom.

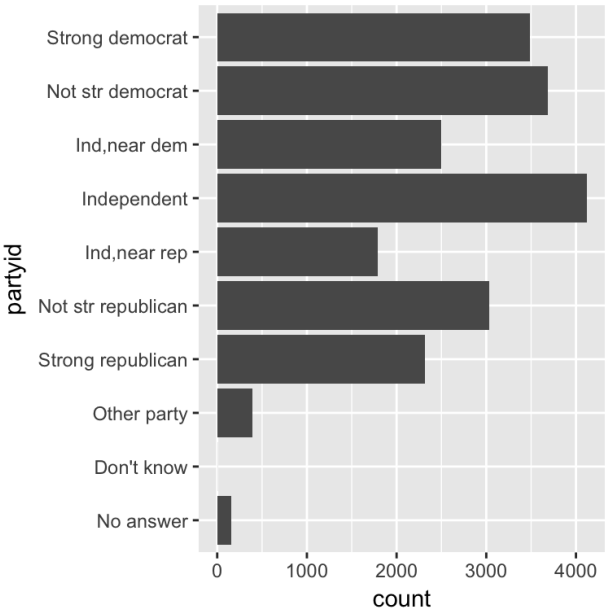
Hide

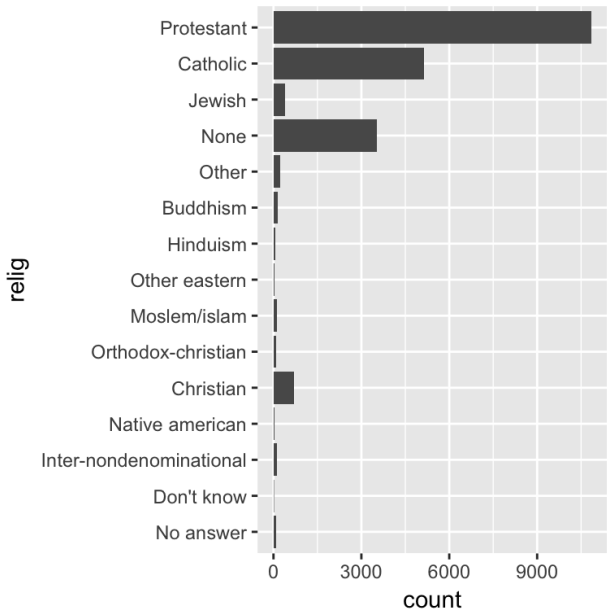
```
levs <- c("marital", "race", "rincome", "partyid", "relig", "denom")

for(i in levs){
  gg <- gss_cat %>%
    ggplot() + geom_bar(aes_string(x = i)) + coord_flip()
  print(gg)
}
```

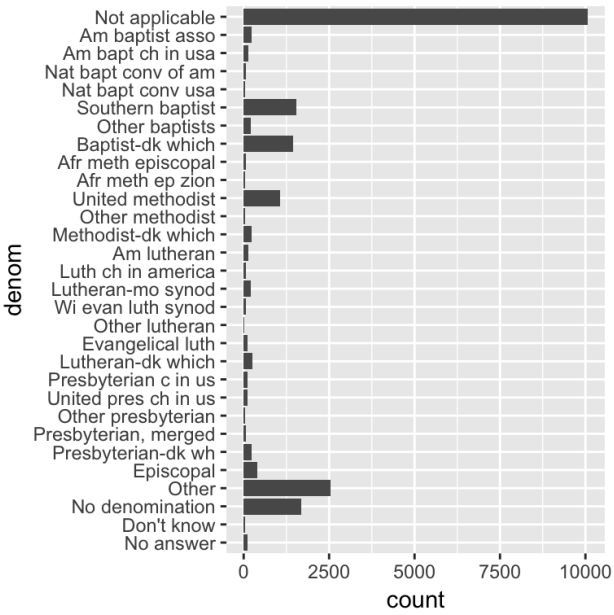












Only race is in descending order, the rest are more or less arbitrary.

3. Why did moving "Not applicable" to the front of the levels move it to the bottom of the plot?

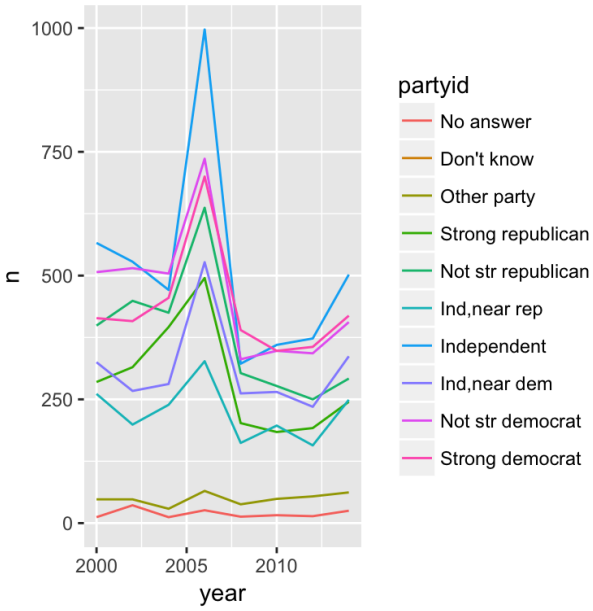
From jarnold, "Because that gives the level "Not applicable" an integer value of 1."

Exercise 15.5.1

1. How have the proportions of people identifying as Democrat, Republican, and Independent changed over time?

Hide

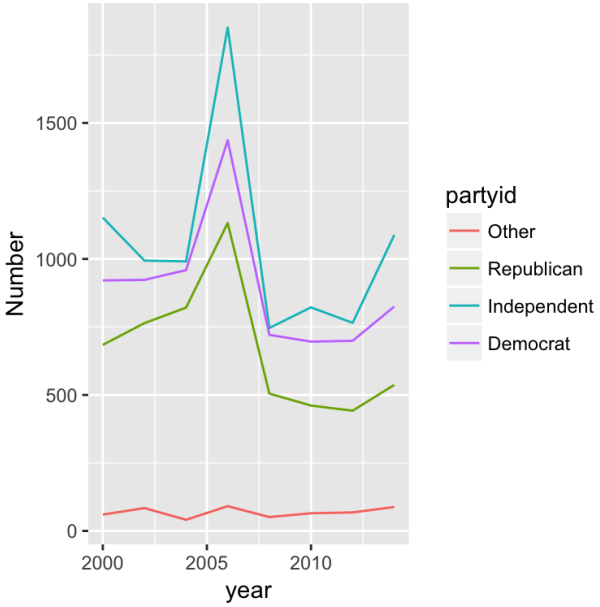
```
gss_cat %>%
  group_by(year,partyid) %>%
  count(partyid) %>%
  ggplot() +
  geom_line(aes(year,n,group=partyid,color=partyid))
```



We need to do some factor reciding...

Hide

```
col_gss_cat <- gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
    Other = c("No answer", "Don't know", "Other party"),
    Republican = c("Strong republican", "Not str republican"),
    Independent = c("Ind,near rep", "Independent", "Ind,near dem"),
    Democrat = c("Not str democrat", "Strong democrat")
  ))

col_gss_cat %>%
  group_by(year,partyid) %>%
  count(partyid) %>%
  ggplot() +
  ylab("Number") +
  geom_line(aes(year,n,group=partyid,color=partyid))
```



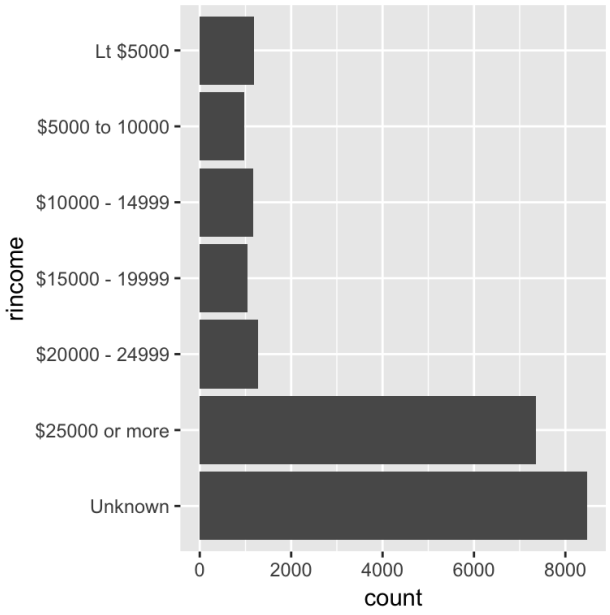
2. How could you collapse rincome into a small set of categories?

From jarnold (just being lazy myself :-)):

“Group all the non-responses into one category, and then group other categories into a smaller number. Since there is a clear ordering, we wouldn’t want to use something like fct_lump.”

Hide

```
library("stringr")
gss_cat %>%
  mutate(rincome =
           fct_collapse(
             rincome,
             `Unknown` = c("No answer", "Don't know", "Refused", "Not applicable"),
             `Lt $5000` = c("Lt $1000", str_c("$", c("1000", "3000", "4000"),
                                              " to ", c("2999", "3999", "4999"))),
             `$5000 to 10000` = str_c("$", c("5000", "6000", "7000", "8000"),
                                      " to ", c("5999", "6999", "7999", "9999"))
           )) %>%
  ggplot(aes(x = rincome)) +
  geom_bar() +
  coord_flip()
```



16. Dates and times

Exercise 16.2.4

1. What happens if you parse a string that contains invalid dates?

Hide

```
library(tidyverse)
library(lubridate)
library(nycflights13)

ymd(c("2010-10-10", "bananas"))
```

```
## [1] "2010-10-10" NA
```

2. What does the tzone argument to today() do? Why is it important?

Time zone is very important because the time changes e.g. between NYC and London, and you want to have the right time.

3. Use the appropriate lubridate function to parse each of the following dates:

Hide

```
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2017"
d4 <- c("August 19 (2015)", "July 1 (2015)")
d5 <- "12/30/14" # Dec 30, 2014
```

Hide

```
mdy(d1)
```

```
## [1] "2010-01-01"
```

Hide

```
ymd(d2)
```

```
## [1] "2015-03-07"
```

Hide

```
dmy(d3)
```

```
## [1] "2017-06-06"
```

Hide

```
mdy(d4)
```

```
## [1] "2015-08-19" "2015-07-01"
```

Hide

```
mdy(d5)
```

```
## [1] "2014-12-30"
```

Exercise 16.3.4

1. How does the distribution of flight times within a day change over the course of the year?

making date time variables using jarnold's code

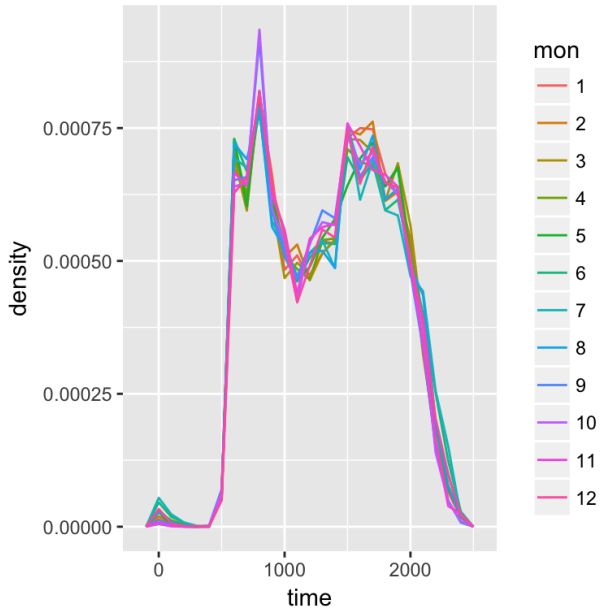
Hide

```
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
}

flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(
    dep_time = make_datetime_100(year, month, day, dep_time),
    arr_time = make_datetime_100(year, month, day, arr_time),
    sched_dep_time = make_datetime_100(year, month, day, sched_dep_time),
    sched_arr_time = make_datetime_100(year, month, day, sched_arr_time)
  ) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))
```

We can show, over time in seconds and separated by each month, the departure time.

Hide

```
flights_dt %>%
  mutate(time = hour(dep_time) * 100 + minute(dep_time),
         mon = as.factor(month
                         (dep_time))) %>%
  ggplot(aes(x = time, y = ..density.., group = mon, color = mon)) +
  geom_freqpoly(binwidth = 100)
```



The distribution looks the same from month to month.

2. Compare dep_time, sched_dep_time and dep_delay. Are they consistent? Explain your findings.

If they are consistent, then dep_time = sched_dep_time + dep_delay

Hide

```
flights_dt %>%
  mutate(dep_time_ = sched_dep_time + dep_delay * 60) %>%
  filter(dep_time_ != dep_time) %>%
  select(dep_time_, dep_time, sched_dep_time, dep_delay)
```

```
## # A tibble: 1,205 x 4
##              dep_time_            dep_time      sched_dep_time dep_delay
##                 <dttm>              <dttm>              <dttm>     <dbl>
##  1 2013-01-02 08:48:00 2013-01-01 08:48:00 2013-01-01 18:35:00       853
##  2 2013-01-03 00:42:00 2013-01-02 00:42:00 2013-01-02 23:59:00        43
##  3 2013-01-03 01:26:00 2013-01-02 01:26:00 2013-01-02 22:50:00       156
##  4 2013-01-04 00:32:00 2013-01-03 00:32:00 2013-01-03 23:59:00        33
##  5 2013-01-04 00:50:00 2013-01-03 00:50:00 2013-01-03 21:45:00       185
##  6 2013-01-04 02:35:00 2013-01-03 02:35:00 2013-01-03 23:59:00       156
##  7 2013-01-05 00:25:00 2013-01-04 00:25:00 2013-01-04 23:59:00        26
##  8 2013-01-05 01:06:00 2013-01-04 01:06:00 2013-01-04 22:45:00       141
##  9 2013-01-06 00:14:00 2013-01-05 00:14:00 2013-01-05 23:59:00        15
## 10 2013-01-06 00:37:00 2013-01-05 00:37:00 2013-01-05 22:30:00       127
## # ... with 1,195 more rows
```

From jarnold:

“There exist discrepencies. It looks like there are mistakes in the dates. These are flights in which the actual departure time is on the next day relative to the scheduled departure time. We forgot to account for this when creating the date-times. The code would have had to check if the departure time is less than the scheduled departure time. Alternatively, simply adding the delay time is more robust because it will automatically account for crossing into the next day.”

3. Compare air_time with the duration between the departure and arrival. Explain your findings.

Hide

```
flights_dt %>%
    mutate(flight_duration = as.numeric(arr_time - dep_time),
         air_time_mins = air_time,
         diff = flight_duration - air_time_mins) %>%
  select(origin, dest, flight_duration, air_time_mins, diff)
```

```
## # A tibble: 328,063 x 5
##    origin  dest flight_duration air_time_mins  diff
##     <chr> <chr>           <dbl>         <dbl> <dbl>
##  1    EWR   IAH             193           227   -34
##  2    LGA   IAH             197           227   -30
##  3    JFK   MIA             221           160    61
##  4    JFK   BQN             260           183    77
##  5    LGA   ATL             138           116    22
##  6    EWR   ORD             106           150   -44
##  7    EWR   FLL             198           158    40
##  8    LGA   IAD              72            53    19
##  9    JFK   MCO             161           140    21
## 10    LGA   ORD             115           138   -23
## # ... with 328,053 more rows
```

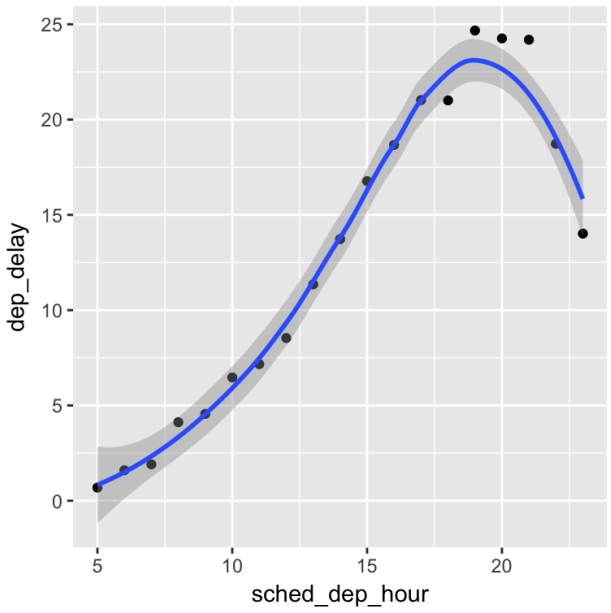
There seems to be a discrepency-air time should be always equal or less than flight duration right?

4. How does the average delay time change over the course of a day? Should you use dep_time or sched_dep_time? Why?

sched_dep_time is what's most interesting for passengers. From jarnold,

Hide

```
flights_dt %>%
  mutate(sched_dep_hour = hour(sched_dep_time)) %>%
  group_by(sched_dep_hour) %>%
  summarise(dep_delay = mean(dep_delay)) %>%
  ggplot(aes(y = dep_delay, x = sched_dep_hour)) +
  geom_point() +
  geom_smooth()
```



5. On what day of the week should you leave if you want to minimise the chance of a delay?

from jarnold,

Hide

```
flights_dt %>%
  mutate(dow = wday(sched_dep_time)) %>%
  group_by(dow) %>%
 summarise(dep_delay = mean(dep_delay),
            arr_delay = mean(arr_delay, na.rm = TRUE),
           tot_delay = dep_delay + arr_delay)
```

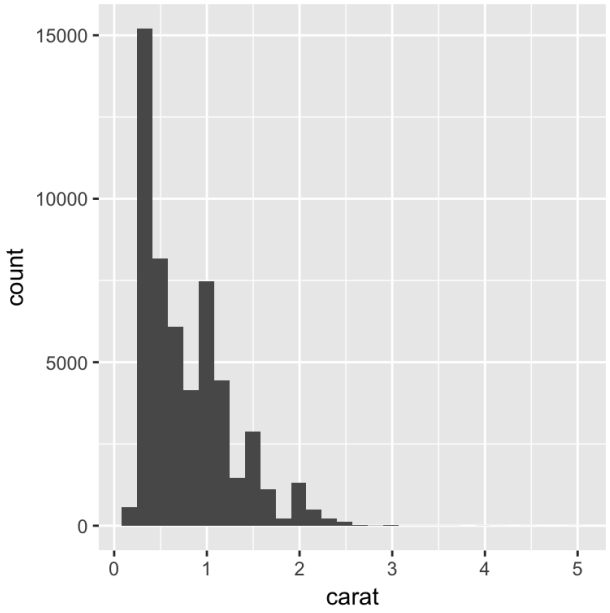
```
## # A tibble: 7 x 4
##     dow dep_delay arr_delay tot_delay
##   <dbl>     <dbl>     <dbl>     <dbl>
## 1     1 11.495974  4.820024  16.31600
## 2     2 14.734220  9.653739  24.38796
## 3     3 10.591532  5.388526  15.98006
## 4     4 11.699198  7.051119  18.75032
## 5     5 16.064837 11.740819  27.80566
## 6     6 14.660643  9.070120  23.73076
## 7     7  7.620118 -1.448828   6.17129
```

Sunday

6. What makes the distribution of diamonds$carat and flights$sched_dep_time similar?

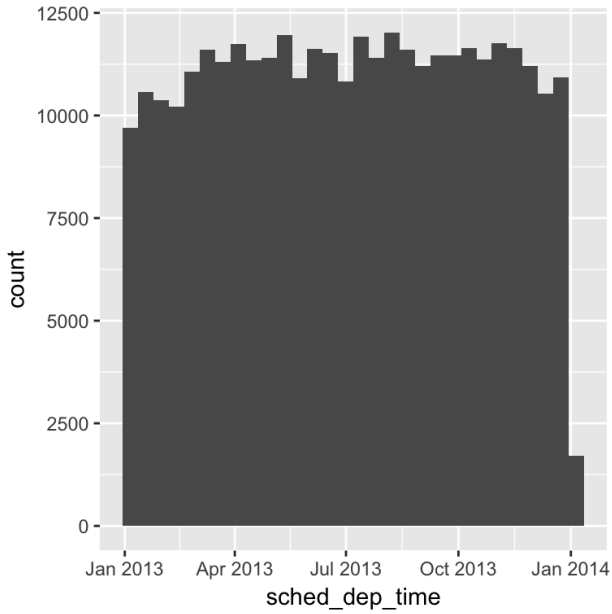
Hide

```
diamonds %>%
  ggplot(aes(carat)) +
  geom_histogram()
```



Hide

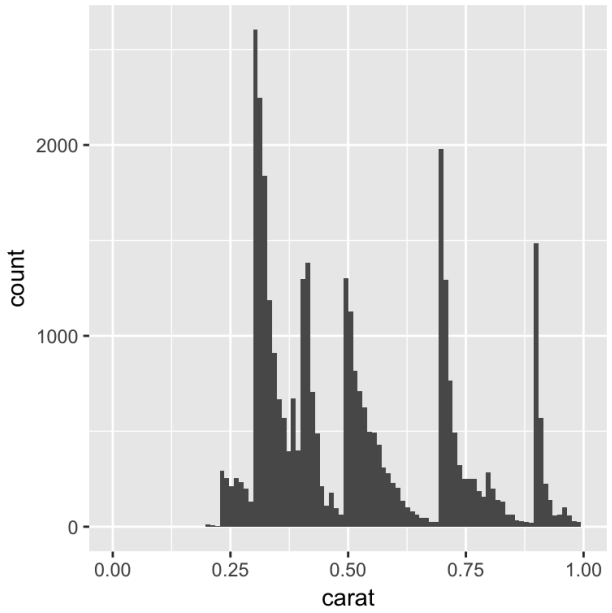
```
flights_dt %>%
  ggplot(aes(sched_dep_time)) +
  geom_histogram()
```



Let's look at more granularity,

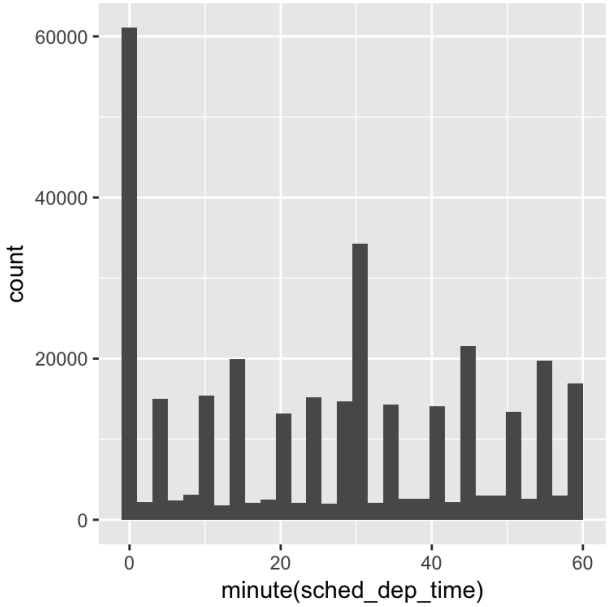
Hide

```
diamonds %>%
  ggplot(aes(carat)) +
  geom_histogram(bins=100) +
  xlim(0,1)
```



Hide

```
flights_dt %>%
  ggplot(aes(minute(sched_dep_time))) +
  geom_histogram()
```

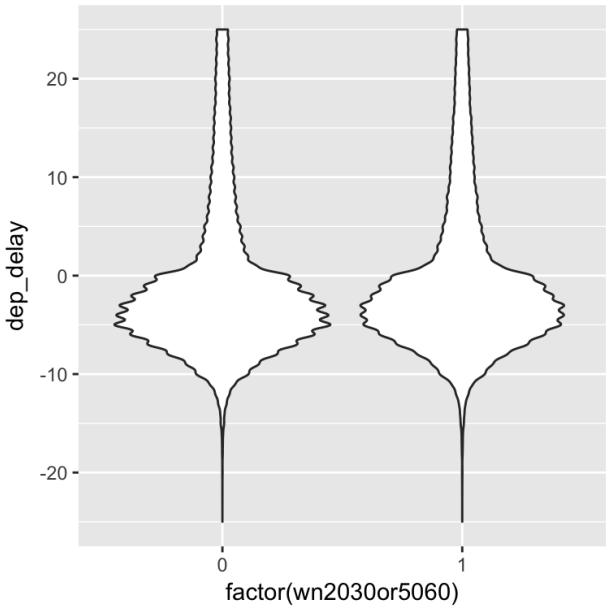


Both distributions have multi-modal distributions, reflecting diamond carats at 1/3, 1/2, 2/3, etc and sched_dep_time of flights being close to every 5 minutes of the hour.

7. Confirm my hypothesis that the early departures of flights in minutes 20-30 and 50-60 are caused by scheduled flights that leave early. Hint: create a binary variable that tells you whether or not a flight was delayed.

Hide

```
flights_dt %>%
  mutate( wn2030or5060 = ifelse(
    (minute(sched_dep_time)>=20 & minute(sched_dep_time)<=30) |
      (minute(sched_dep_time)>=50 & minute(sched_dep_time)<=60),
    1,0)) %>%
  select(sched_dep_time,wn2030or5060,dep_delay) %>%
  ggplot() +
  geom_violin(aes(factor(wn2030or5060),dep_delay)) +
  ylim(-25,25)
```



I would expect there to be a lower-value distribution to go with the hypothesis. Doesn't seem like it.

Exercise 16.4.5

1. Why is there months() but no dmonths()?

From jarnold,

“There is no direct unambigous value of months in seconds:

31 days: Jan, Mar, May, Jul, Aug, Oct, 30 days: Apr, Jun, Sep, Nov, Dec 28 or 29 days: Feb

Though in the past, in the pre-computer era, for arithmetic convenience, bankers adopoted a 360 day year with 30 day months."

2. Explain days(overnight * 1) to someone who has just started learning R. How does it work?

In R we can easily manipulate the concept of time using variables. This is helpful when we have messy time data, such as times for overnight flights. We can clean up time discrepencies by accomidating for whether there was an overnight flight-the variables overnight is either TRUE or FALSE so by multiplying by 1 all overnight flights get added a day and those not overnight stay the same.

3. Create a vector of dates giving the first day of every month in 2015. Create a vector of dates giving the first day of every month in the current year.

from jarnold,

Hide

```
ymd("2015-01-01") + months(0:11)
```

```
##  [1] "2015-01-01" "2015-02-01" "2015-03-01" "2015-04-01" "2015-05-01"
##  [6] "2015-06-01" "2015-07-01" "2015-08-01" "2015-09-01" "2015-10-01"
## [11] "2015-11-01" "2015-12-01"
```

Hide

```
floor_date(today(), unit = "year") + months(0:11)
```

```
##  [1] "2017-01-01" "2017-02-01" "2017-03-01" "2017-04-01" "2017-05-01"
##  [6] "2017-06-01" "2017-07-01" "2017-08-01" "2017-09-01" "2017-10-01"
## [11] "2017-11-01" "2017-12-01"
```

4. Write a function that given your birthday (as a date), returns how old you are in years.

Hide

```
func <- function(date){
  todays_yr <- year(today())
  birth_yr <- year(ymd(date))
  cat("You are ",todays_yr - birth_yr, " years old!")
}

func("1992-07-15")
```

```
## You are  25  years old!
```

5. Why can't (today() %–% (today() + years(1))) / months(1) work?

It doesn't throw an error, but the functions provide different time classes.

Hide

```
(today() %--% (today() + years(1))) / months(1)
```

```
## [1] 12
```

Part III Program

17. Introduction

No Exercises

18. Pipes

No Exercises

19. Functions

19.2.1 Practice

1. Why is TRUE not a parameter to rescale01()? What would happen if x contained a single missing value, and na.rm was FALSE?

Hide

```
x <- c(0,.5,1, NA)
range(x,na.rm = T)
```

```
## [1] 0 1
```

Hide

```
range(x,na.rm = F)
```

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

na.rm = F makes all the elements NA values. Instead with na.rm = T, you can tell how many NA values you have.

2. In the second variant of rescale01(), infinite values are left unchanged. Rewrite rescale01() so that -Inf is mapped to 0, and Inf is mapped to 1.

from jarnold (I can't beat his work...)

Hide

```
x <- c(1:10, Inf, -Inf)

#second variant
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])

}

rescale01(x)
```

```
##  [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
##  [8] 0.7777778 0.8888889 1.0000000       Inf      -Inf
```

Hide

```
rescale01_alt <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  y <- (x - rng[1]) / (rng[2] - rng[1])
  y[ y== Inf] <- 1
  y[ y== -Inf] <- 0
}

rescale01_alt(x)
```

3. Practice turning the following code snippets into functions. Think about what each function does. What would you call it? How many arguments does it need? Can you rewrite it to be more expressive or less duplicative?

Hide

```
mean(is.na(x))
```

```
## [1] 0
```

Hide

```
x / sum(x, na.rm = TRUE)
```

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

Hide

```
sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
```

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

Not really sure what the mean of a logical is doing...

The second one I can do which has a transparent name and reducing duplication a little.

And the third one I would do which does the same as above. This is the coefficient of variation so maybe *coef_var* would be a better name.

4. Follow http://nicercode.github.io/intro/writing-functions.html (http://nicercode.github.io/intro/writing-functions.html) to write your own functions to compute the variance and skew of a numeric vector.

Looks easy enough, but I'll move on.

5. Write both_na(), a function that takes two vectors of the same length and returns the number of positions that have an NA in both vectors.

jarnold has a better answer, of course, but here's mine.

Hide

```
both_na <- function(x,y){

  if(length(x) != length(y)){
    stop("You're two vectors need to be the same length")
  }

  x_ints <- as.integer( is.na(x) )
  y_ints <- as.integer( is.na(y) )

  sum((x_ints + y_ints)==2)
}

x <- c(NA,2,NA)
y <- c(NA,1,2)
both_na(x,y)
```

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

6. What do the following functions do? Why are they useful even though they are so short?

Hide

```
is_directory <- function(x) file.info(x)$isdir
is_readable <- function(x) file.access(x, 4) == 0
```

is_directory extracts whether x is a directory and *is_readable* extracts how many files have read permission in my directory. Their still useful even though they're short because the function name is a bit more intuitive and requires a bit less thinking about what it's doing compared to the code in the function calls.

To me, I'd not do functions for these unless there's a lot of repitition because I would still take the time to see what the functions are doing and then read the help pages on the called functions.

7. Read the complete lyrics (https://en.wikipedia.org/wiki/Little_Bunny_Foo_Foo) to "Little Bunny Foo Foo". There's a lot of duplication in this song. Extend the initial piping example to recreate the complete song, and use functions to reduce the duplication.

I like jarnold's answer, but it's broken...it still gets the point across of writing functions to reduce duplication and increase interpretability.

Hide

```
library(dplyr)

threat <- function(chances) {
  give_chances(from = Good_Fairy,
               to = foo_foo,
               number = chances,
               condition = "Don't behave",
               consequence = turn_into_goon)
}

lyric <- function() {
  foo_foo %>%
    hop(through = forest) %>%
    scoop(up = field_mouse) %>%
    bop(on = head)

  down_came(Good_Fairy)
  said(Good_Fairy,
      c("Little bunny Foo Foo",
        "I don't want to see you",
        "Scooping up the field mice",
        "And bopping them on the head."))
}

lyric()
threat(3)
lyric()
threat(2)
lyric()
threat(1)
lyric()
turn_into_goon(Good_Fairy, foo_foo)
```

Exercise 19.3.1

Quick note: in an R script use **Cmd/Ctrl + Shift + R** to get a cool comment line!

1. Read the source code for each of the following three functions, puzzle out what they do, and then brainstorm better names.

Hide

```
f1 <- function(string, prefix) {
  substr(string, 1, nchar(prefix)) == prefix
}
f2 <- function(x) {
  if (length(x) <= 1) return(NULL)
  x[-length(x)]
}
f3 <- function(x, y) {
  rep(y, length.out = length(x))
}
```

from jarnold,

f1 returns whether a function has a common prefix.

The function *f2* drops the last elemen. A better name for f2 is drop_last().

The function *f3* repeats y once for each element of x. This is a harder one to name. I would say something like recycle (R's name for this behavior), or expand.

2. Take a function that you've written recently and spend 5 minutes brainstorming a better name for it and its arguments.

Haven't written a function in a while, so I'll pass.

3. Compare and contrast rnorm() and MASS::mvrnorm(). How could you make them more consistent?

from jarnold,

You can ignore.

rnorm samples from the univariate normal distribution, while MASS::mvrnorm samples from the multivariate normal distribution. The main arguments in rnorm are n, mean, sd. The main arguments is MASS::mvrnorm are n, mu, Sigma. To be consistent they should have the same names. However, this is difficult. In general, it is better to be consistent with more widely used functions, e.g. rmvnorm should follow the conventions of rnorm. However, while mean is correct in the multivariate case, sd does not make sense in the multivariate case. Both functions an internally consistent though; it would be bad to have mu and sd or mean and Sigma.

4. Make a case for why norm_r(), norm_d() etc would be better than rnorm(), dnorm(). Make a case for the opposite.

from jarnold,

If named norm_r and norm_d, it groups the family of functions related to the normal distribution. If named rnorm, and dnorm, functions related to are grouped into families by the action they perform. r* functions always sample from distributions: rnorm, rbinom, runif, rexp. d* functions calculate the probability density or mass of a distribution: dnorm, dbinom, dunif, dexp.

Exercise 19.4.4

1. What's the difference between if and ifelse()? Carefully read the help and construct three examples that illustrate the key differences.

if statements execute upon a true condition and *ifelse* executes always because the otherwise option is given.

Hide

```
x <- FALSE

if(x) "Hi"

ifelse(x,"Hi","FALSE condition")
```

```
## [1] "FALSE condition"
```

2. Write a greeting function that says "good morning", "good afternoon", or "good evening", depending on the time of day. (Hint: use a time argument that defaults to lubridate::now(). That will make it easier to test your function.)

Hide

```
library(lubridate)

greet <- function(x){
  hr <- hour(x)
  if(hr<12){
    "Good morning"
  } else if(hr>17){
    "Good evening"
  } else{
    "Good afternoon"
  }
}

greet( now() )
```

```
## [1] "Good evening"
```

3. Implement a fizzbuzz function. It takes a single number as input. If the number is divisible by three, it returns "fizz". If it's divisible by five it returns "buzz". If it's divisible by three and five, it returns "fizzbuzz". Otherwise, it returns the number. Make sure you first write working code before you create the function.

Hide

```
fizzbuzz <- function(x){
  stopifnot( length(x)==1 && is.numeric(x) )
  if(x %% 3==0){
    print("fizz")
  } else if(x %% 5==0){
    print("buzz")
  } else if((x %% 3==0) && (x %% 5==0)){
    print("fizzbuzz")
  } else{
    print(x)
  }
}

fizzbuzz(9)
```

```
## [1] "fizz"
```

4. How could you use cut() to simplify this set of nested if-else statements?

I like jarnold's explanation that *cut* works with vectors while *if* works only with single values.

Hide

```
temp <- seq(-10, 50, by = 5)

# this
if (temp <= 0) {
  "freezing"
} else if (temp <= 10) {
  "cold"
} else if (temp <= 20) {
  "cool"
} else if (temp <= 30) {
  "warm"
} else {
  "hot"
}
```

```
## [1] "freezing"
```

Hide

```
#is converted to this
cut(temp, c(-Inf, 0, 10, 20, 30, Inf), right = TRUE,
    labels = c("freezing", "cold", "cool", "warm", "hot"))
```

```
##  [1] freezing freezing freezing cold     cold     cool     cool
##  [8] warm     warm     hot      hot      hot      hot
## Levels: freezing cold cool warm hot
```

This is cool-the intervals could be equally spaced by specifying an integer for breaks or you can specify the breakpoints like above.

5. What happens if you use switch() with numeric values?

from jarnold,

It selects that number argument from ...

two

6. What does this switch() call do? What happens if x is “e”?

Hide

```
x <- "e"
switch(x,
  a = ,
  b = "ab",
  c = ,
  d = "cd"
)

x <- "a"
switch(x,
  a = ,
  b = "ab",
  c = ,
  d = "cd"
)
```

```
## [1] "ab"
```

Nothing happens because e isn't a named argument in *switch*. But if a named argument is given, it points to that-but if nothing is given like in "a" above it'll point to the next argument in *switch*.

Exercise 19.5.5

1. What does commas(letters, collapse = "-") do? Why?

letters is a, b, c and

Hide

```
library(stringr)

letters = c("a", "b", "c")

commas <- function(...) stringr::str_c(..., collapse = ", ")

#commas(letters, collapse = "-")
```

is supposed to concatenates the letters and connects them by '-'. But it throws an error: The argument collapse is passed to str_c as part of ..., so it tries to run str_c(letters, collapse = "-", collapse = ",").

2. It'd be nice if you could supply multiple characters to the pad argument, e.g. rule("Title", pad = "-+"). Why doesn't this currently work? How could you fix it?

from jarnold,

Hide

```
rule <- function(..., pad = "-") {
  title <- paste0(...)
  width <- getOption("width") - nchar(title) - 5
  cat(title, " ", stringr::str_dup(pad, width), "\n", sep = "")
}
rule("Important output")
```

```
## Important output ------------------------------------------------------
```

Hide

```
rule("Important output", pad = "-+")
```

```
## Important output -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-+-+
```

It does not work because it duplicates pad by the width minus the length of the string. This is implictly assuming that pad is only one character. I could adjust the code to calculate the length of pad. The trickiest part is handling what to do if width is not a multiple of the number of characters of pad.

Hide

```
rule <- function(..., pad = "-") {
  title <- paste0(...)
  width <- getOption("width") - nchar(title) - 5
  padchar <- nchar(pad)
  cat(title, " ",
      stringr::str_dup(pad, width %/% padchar),
      # if not multiple, fill in the remaining characters
      stringr::str_sub(pad, 1, width %% padchar),
      "\n", sep = "")
}
rule("Important output")
```

```
## Important output ----------------------------------------------------
```

Hide

```
rule("Important output", pad = "-+")
```

```
## Important output -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Hide

```
rule("Important output", pad = "-+-")
```

```
## Important output -+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-
```

3. What does the trim argument to mean() do? When might you use it?

The *trim* argument omits the end proportion of elements in 'x'. Why do this? Maybe if you know the end data is faulty? Remove outliers from calculation?

4. The default value for the method argument to cor() is c("pearson", "kendall", "spearman"). What does that mean? What value is used by default?

Those are the options-if others are given the function will stop. The first is used by default.

20. Vectors

Exercise 20.3.5

1. Describe the difference between is.finite(x) and !is.infinite(x).

From the docs,

- is.finite returns a vector of the same length as x the jth element of which is TRUE if x[j] is finite (i.e., it is not one of the values NA, NaN, Inf or -Inf) and FALSE otherwise.
- is.infinite returns a vector of the same length as x the jth element of which is TRUE if x[j] is infinite (i.e., equal to one of Inf or -Inf) and FALSE otherwise.

So *!is.infinite()* will be true for everything but *Inf* or *-Inf*. is.finite() will be false for NA, NaN while !is.infinite() will be true.

2. Read the source code for dplyr::near() (Hint: to see the source code, drop the ()). How does it work?

It determines of the difference between the two numbers given is less than a tolerance threshold, which is 1.490116e-08 encoded by .Machine$double.eps^0.5. This gives greater precision for determining equality of two numbers. Also you can change the threshold to your liking.

3. A logical vector can take 3 possible values. How many possible values can an integer vector take? How many possible values can a double take? Use google to do some research.

from the docs of ?is.integer,

An integer vector can actually contain onlyin the range of +/-2*10^9. Doubles can hold much larger integers exactly.

4. Brainstorm at least four functions that allow you to convert a double to an integer. How do they differ? Be precise.

from jarnold,

Broadly, could convert a double to an integer by truncating or rounding to the nearest integer. For truncating or for handling ties (doubles ending in 0.5), there are multiple methods for determing which integer value to go to.

For rounding, R and many programming languages use the IEEE standard. This is “round to nearest, ties to even”. This is not the same as what you See the value of looking at the value of .Machine$double.rounding and its documentation.

He provides a list (https://www.ma.utexas.edu/users/arbogast/misc/disasters.html) of cool links for rounding errors.

5. What functions from the readr package allow you to turn a string into logical, integer, and double vector?

The parse_* functions for those types

Exerise 20.4.6

1. What does mean(is.na(x)) tell you about a vector x? What about sum(!is.finite(x))?

The former tells is you the average amount of NA values in vector x, which the the sum of NAs divided by the length of NAs and non-NAs. The latter returns the number of elements in x that are either NA, NaN, Inf or -Inf.

2. Carefully read the documentation of is.vector(). What does it actually test for? Why does is.atomic() not agree with the definition of atomic vectors above?

from the help page,

is.vector returns TRUE if x is a vector of the specified mode having no attributes other than names. It returns FALSE otherwise.

is.atomic returns TRUE if x is of an atomic type (or NULL) and FALSE otherwise.

it is common to call the atomic types ‘atomic vectors’, but note that is.vector imposes further restrictions: an object can be atomic but not a vector (in that sense).

is.atomic(list()) gives FALSE and is.vector(list()) gives TRUE

3. Compare and contrast setNames() with purrr::set_names().

Hide

```
setNames
```

```
## function (object = nm, nm)
## {
##     names(object) <- nm
##     object
## }
## <bytecode: 0x7fc562eca4a8>
## <environment: namespace:stats>
```

Hide

```
purrr::set_names
```

```
## function (x, nm = x, ...)
## {
##     if (!is_vector(x)) {
##         abort("`x` must be a vector")
##     }
##     if (is_function(nm) || is_formula(nm)) {
##         nm <- as_function(nm)
##         nm <- nm(names2(x), ...)
##     }
##     else if (!is_null(nm)) {
##         nm <- as.character(nm)
##         nm <- chr(nm, ...)
##     }
##     if (!is_null(nm) && !is_character(nm, length(x))) {
##         abort("`nm` must be `NULL` or a character vector the same length as `x`")
##     }
##     names(x) <- nm
##     x
## }
## <environment: namespace:rlang>
```

set_names adds a few sanity checks: x has to be a vector, and the lengths of the object and the names have to be the same.

Create functions that take a vector as input and returns:

- The last value. Should you use [or [[? 2 The elements at even numbered positions.
- Every element except the last value.
- Only even numbers (and no missing values).

4. Create functions that take a vector as input and returns:

1. The last value. Should you use [or [[?
2. The elements at even numbered positions.
3. Every element except the last value.
4. Only even numbers (and no missing values).

see jarnold (https://jrnold.github.io/e4qf/vectors.html#using-atomic-vectors)

5. Why is x[-which(x > 0)] not the same as x[x <= 0]?

from jarnold,

Hide

```
x <- c(-5:5, Inf, -Inf, NaN, NA)
x[-which(x > 0)]
```

```
##  [1]   -5   -4   -3   -2   -1    0 -Inf  NaN   NA
```

Hide

```
-which(x > 0)
```

```
## [1]  -7  -8  -9 -10 -11 -12
```

Hide

```
x[x <= 0]
```

```
## [1]   -5   -4   -3   -2   -1    0 -Inf   NA   NA
```

Hide

```
x <= 0
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE  TRUE    NA    NA
```

-which(x > 0) which calculates the indexes for any value that is TRUE and ignores NA. Thus is keeps NA and NaN because the comparison is not TRUE. x <= 0 works slightly differently. If x <= 0 returns TRUE or FALSE it works the same way. Hoewver, if the comparison generates a NA, then it will always keep that entry, but set it to NA. This is why the last two values of x[x <= 0] are NA rather than c(NaN, NA).

6. What happens when you subset with a positive integer that's bigger than the length of the vector? What happens when you subset with a name that doesn't exist?

From jarnold,

When you subset with positive integers that are larger than the length of the vector, NA values are returned for those integers larger than the length of the vector.

When a vector is subset with a name that doesn't exist, an error is generated.

Exercise 20.5.4

1. Draw the following lists as nested sets:

list(a, b, list(c, d), list(e, f)) list(list(list(list(list(list(a))))))

Sorry, not drawing-see the chapter for drawing representations.

2. What happens if you subset a tibble as if you're subsetting a list? What are the key differences between a list and a tibble?

From jarnold,

Subsetting a tibble works the same way as a list; a data frame can be thought of as a list of columns. The key different between a list and a tibble is that a tibble (data frame) has the restriction that all its elements (columns) must have the same length.

Exercise 20.7.4

1. What does hms::hms(3600) return? How does it print? What primitive type is the augmented vector built on top of? What attributes does it use?

Hide

```
x <- hms::hms(3600)
class(x)
```

```
## [1] "hms"      "difftime"
```

Hide

```
str(x)
```

```
## Classes 'hms', 'difftime'  atomic [1:1] 3600
##   ..- attr(*, "units")= chr "secs"
```

Hide

```
typeof(x)
```

```
## [1] "double"
```

Hide

```
attributes(x)
```

```
## $units
## [1] "secs"
##
## $class
## [1] "hms"      "difftime"
```

2. Try and make a tibble that has columns with different lengths. What happens?

Hide

```
library(dplyr)

tibble(x = 1:2, y = 1:3)
#Error: Column `x` must be length 1 or 3, not 2
```

3. Based on the definition above, is it ok to have a list as a column of a tibble?

Only if the length is equal to the other column vectors/lists.

21. Iteration

Exercise 21.2.1

1. Write for loops to:

1. Compute the mean of every column in mtcars.
2. Determine the type of each column in nycflights13::flights.
3. Compute the number of unique values in each column of iris.
4. Generate 10 random normals for each of μ= −10, 0, 10 , and 100.

Think about the output, sequence, and body before you start writing the loop.

Hide

```
# 1
col_means <- vector("double",length = ncol(mtcars))
for(i in seq_along(mtcars)){
  col_means[[i]] <- mean(mtcars[[i]])
}
col_means
```

```
##  [1]  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##  [7]  17.848750   0.437500   0.406250   3.687500   2.812500
```

Hide

```
#2
types <- vector("list",length = ncol(nycflights13::flights)) #class can  have multiple values
names(types) <- names(nycflights13::flights)
for(i in seq_along(nycflights13::flights)){
  types[[i]] <- class(nycflights13::flights[[i]])
}
types
```

```
## $year
## [1] "integer"
##
## $month
## [1] "integer"
##
## $day
## [1] "integer"
##
## $dep_time
## [1] "integer"
##
## $sched_dep_time
## [1] "integer"
##
## $dep_delay
## [1] "numeric"
##
## $arr_time
## [1] "integer"
##
## $sched_arr_time
## [1] "integer"
##
## $arr_delay
## [1] "numeric"
##
## $carrier
## [1] "character"
##
## $flight
## [1] "integer"
##
## $tailnum
## [1] "character"
##
## $origin
## [1] "character"
##
## $dest
## [1] "character"
##
## $air_time
## [1] "numeric"
##
## $distance
## [1] "numeric"
##
## $hour
## [1] "numeric"
##
## $minute
## [1] "numeric"
##
## $time_hour
## [1] "POSIXct" "POSIXt"
```

Hide

```
#3
uniq_in_col <- vector("integer",ncol(iris))
names(uniq_in_col) <- names(iris)
for(i in seq_along(iris)){
  uniq_in_col[[i]] <- length(unique(iris[[i]]))
}
uniq_in_col
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##           35           23           43           22            3
```

Hide

```
#4
means <- c(-10,0,10,100)
ran_norms <- vector("list",length=length(means))
n <- 10
names(ran_norms) <- means
for(i in seq_along(ran_norms)){
  ran_norms[[i]] <- rnorm(n = n, mean=means[i])
}
ran_norms
```

```
## $`-10`
##  [1]  -9.782210  -9.783519  -8.713145  -9.699514 -11.355749 -11.505992
##  [7] -10.646917  -9.439703 -10.322458  -9.335176
##
## $`0`
##  [1]  1.0560113  0.4293358  0.5480951 -1.1116816  0.1633620 -0.8094876
##  [7] -0.2377104  0.3179641 -1.8586886 -1.0749297
##
## $`10`
##  [1] 11.321203 10.686546  9.782033 11.400680 10.083576 10.727077  8.952497
##  [8]  9.502575 10.794428  8.909179
##
## $`100`
##  [1] 100.32602  99.88585  98.13351 100.06831 100.38857 100.23556  99.71547
##  [8]  98.84670 100.73822  99.40213
```

Hide

```
#jarnold notes we can just make a matrix because rnorm recycles the means (vectorized operation)
matrix(rnorm(n * length(means), mean = means), ncol = n)
```

```
##             [,1]        [,2]         [,3]       [,4]        [,5]
## [1,] -10.0523338 -10.5643940 -10.20917153 -11.518092 -10.0557345
## [2,]   0.9700397   0.4048292  -0.04495856   2.218319   0.1312883
## [3,]  10.2047852   9.4786549   9.45667593   9.086414  10.6150856
## [4,] 100.5452153 100.0354454  99.84771066 100.305060  99.2797483
##            [,6]       [,7]      [,8]         [,9]     [,10]
## [1,] -10.522843 -8.7190236 -10.42173 -11.05073285 -6.856809
## [2,]  -1.764183 -0.7201603   2.36971  -0.03026167 -1.713422
## [3,]   8.383111  9.0472588  10.33865   9.12435959  8.243709
## [4,] 100.751077 99.7857383  99.52560  99.10466249 99.273852
```

2. Eliminate the for loop in each of the following examples by taking advantage of an existing function that works with vectors:

Hide

```
#1
out <- ""
for (x in letters) {
  out <- stringr::str_c(out, x)
}
#2
x <- sample(100)
sd <- 0
for (i in seq_along(x)) {
  sd <- sd + (x[i] - mean(x)) ^ 2
}
sd <- sqrt(sd / (length(x) - 1))
#3
x <- runif(100)
out <- vector("numeric", length(x))
out[1] <- x[1]
for (i in 2:length(x)) {
  out[i] <- out[i - 1] + x[i]
}
```

from jarnold-takes advantage of the fact that many of these functions are vectorized (know to work through each element in a vector).

Hide

```
#1
stringr::str_c(letters, collapse = "")
```

```
## [1] "abc"
```

Hide

```
#2
x <- sample(100)
sd(x) #or
```

```
## [1] 29.01149
```

Hide

```
sqrt(sum((x - mean(x)) ^ 2) / (length(x) - 1))
```

```
## [1] 29.01149
```

Hide

```
#3-this is a cumulative summation so we just use the cumsum function
x <- runif(100)
cumsum(x)
```

```
##   [1]  0.1585237  0.4955077  0.8746246  1.0127657  1.2579870  1.4299225
##   [7]  1.9900154  2.7079516  3.0328263  3.4117178  4.2827850  5.2428236
##  [13]  5.7240713  6.5145766  7.2684402  7.9535453  8.5283808  8.7555752
##  [19]  9.2210613  9.3179354  9.4374757  9.6260160  9.9355400 10.2116301
##  [25] 11.0322993 11.6172869 12.5815333 13.0098218 13.2954146 13.9561334
##  [31] 14.7877616 14.8420736 15.0975497 15.5626921 16.5214859 16.8348926
##  [37] 16.8597550 17.8347179 17.8502785 18.3934595 18.5537550 18.8951895
##  [43] 19.1981427 19.4788238 19.5107065 19.7670737 20.7511489 20.9087983
##  [49] 21.4272117 22.2743143 22.9163670 23.8264348 24.2381663 24.2630072
##  [55] 25.0678384 25.2808299 25.9409406 26.1956704 27.1185219 27.6052099
##  [61] 28.1732145 28.5461514 29.0448202 29.0931784 29.1818545 29.4814858
##  [67] 29.8924843 30.3276582 30.6004047 31.2561873 32.1578170 32.7051952
##  [73] 33.2658342 33.5007759 34.1604669 34.7843218 35.3188907 35.8099561
##  [79] 35.9350050 36.4674255 36.5813578 37.0869016 37.6270703 37.8291504
##  [85] 38.5884003 39.1523516 39.5579204 40.2984624 40.7025057 40.8446495
##  [91] 41.0116712 41.4858749 41.7671875 41.8597004 42.1691953 42.6564239
##  [97] 42.9437491 43.7527671 44.5986954 45.4420465
```

3 Combine your function writing and for loop skills:

1. Write a for loop that prints() the lyrics to the children's song "Alice the camel"" (https://www.kididdles.com/lyrics/a012.html).
2. Convert the nursery rhyme "ten in the bed" (https://supersimpleonline.com/song/ten-in-the-bed/) to a function. Generalise it to any number of people in any sleeping structure.
3. Convert the song "99 bottles of beer on the wall" to a function. Generalise to any number of any vessel containing any liquid on any surface.

just going to use jarnold's solutions

Hide

```
library(stringr)
#1
humps <- c("five", "four", "three", "two", "one", "no")
for (i in humps) {
  cat(str_c("Alice the camel has ", rep(i, 3), " humps.",
            collapse = "\n"), "\n")
  if (i == "no") {
    cat("Now Alice is a horse.\n")
  } else {
    cat("So go, Alice, go.\n")
  }
  cat("\n")
}
```

```
## Alice the camel has five humps.
## Alice the camel has five humps.
## Alice the camel has five humps.
## So go, Alice, go.
##
## Alice the camel has four humps.
## Alice the camel has four humps.
## Alice the camel has four humps.
## So go, Alice, go.
##
## Alice the camel has three humps.
## Alice the camel has three humps.
## Alice the camel has three humps.
## So go, Alice, go.
##
## Alice the camel has two humps.
## Alice the camel has two humps.
## Alice the camel has two humps.
## So go, Alice, go.
##
## Alice the camel has one humps.
## Alice the camel has one humps.
## Alice the camel has one humps.
## So go, Alice, go.
##
## Alice the camel has no humps.
## Alice the camel has no humps.
## Alice the camel has no humps.
## Now Alice is a horse.
```

Hide

```
#2
numbers <- c("ten", "nine", "eight", "seven", "six", "five",
             "four", "three", "two", "one")
for (i in numbers) {
  cat(str_c("There were ", i, " in the bed\n"))
  cat("and the little one said\n")
  if (i == "one") {
    cat("I'm lonely...")
  } else {
    cat("Roll over, roll over\n")
    cat("So they all rolled over and one fell out.\n")
  }
  cat("\n")
}
```

```
## There were ten in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were nine in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were eight in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were seven in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were six in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were five in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were four in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were three in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were two in the bed
## and the little one said
## Roll over, roll over
## So they all rolled over and one fell out.
##
## There were one in the bed
## and the little one said
## I'm lonely...
```

Hide

```
#3
bottles <- function(i) {
  if (i > 2) {
   bottles <- str_c(i - 1, " bottles")
  } else if (i == 2) {
   bottles <- "1 bottle"
  } else {
   bottles <- "no more bottles"
  }
  bottles
}

beer_bottles <- function(n) {
  # should test whether n >= 1.
  for (i in seq(n, 1)) {
     cat(str_c(bottles(i), " of beer on the wall, ", bottles(i), " of beer.\n"))
     cat(str_c("Take one down and pass it around, ", bottles(i - 1),
                " of beer on the wall.\n\n"))
  }
  cat("No more bottles of beer on the wall, no more bottles of beer.\n")
  cat(str_c("Go to the store and buy some more, ", bottles(n), " of beer on the wall.\n"))
}
beer_bottles(3)
```

```
## 2 bottles of beer on the wall, 2 bottles of beer.
## Take one down and pass it around, 1 bottle of beer on the wall.
##
## 1 bottle of beer on the wall, 1 bottle of beer.
## Take one down and pass it around, no more bottles of beer on the wall.
##
## no more bottles of beer on the wall, no more bottles of beer.
## Take one down and pass it around, no more bottles of beer on the wall.
##
## No more bottles of beer on the wall, no more bottles of beer.
## Go to the store and buy some more, 2 bottles of beer on the wall.
```

4. It's common to see for loops that don't preallocate the output and instead increase the length of a vector at each step:

Hide

```
output <- vector("integer", 0)
for (i in seq_along(x)) {
  output <- c(output, lengths(x[[i]]))
}
output
```

How does this affect performance? Design and execute an experiment.

How I would do this:

Hide

```
output <- vector("integer", 0)
system.time( for (i in seq_along(x)) {
  output <- c(output, lengths(x[[i]]))
} )
```

```
##    user  system elapsed
##   0.004   0.000   0.004
```

Hide

```
output
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Hide

```
output <- vector("integer", length(length(x)))
system.time( for (i in seq_along(x)) {
  output <- c(output, lengths(x[[i]]))
} )
```

```
##    user  system elapsed
##   0.005   0.001   0.006
```

Hide

```
output
```

```
##   [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

How jarnold does this:

Hide

```
#not allocated
library(microbenchmark)
add_to_vector <- function(n) {
  output <- vector("integer", 0)
  for (i in seq_len(n)) {
    output <- c(output, i)
  }
  output
}
microbenchmark(add_to_vector(10000), times = 3,unit = "ms")
```

```
## Unit: milliseconds
##                  expr      min       lq     mean  median       uq      max
##  add_to_vector(10000) 161.2413 177.7291 199.1119 194.217 218.0472 241.8774
##  neval
##      3
```

Hide

```
#pre-allocated
add_to_vector_2 <- function(n) {
  output <- vector("integer", n)
  for (i in seq_len(n)) {
    output[[i]] <- i
  }
  output
}
microbenchmark(add_to_vector_2(10000), times = 3,unit = "ms")
```

```
## Unit: milliseconds
##                    expr      min        lq     mean   median       uq
##  add_to_vector_2(10000) 0.604179 0.6377025 2.011321 0.671226 2.714892
##       max neval
##  4.758558     3
```

Dang, what a difference! I better pre-allocate...

Exercise 21.3.5

1. Imagine you have a directory full of CSV files that you want to read in. You have their paths in a vector, files <- dir("data/", pattern ="//.csv", full.names = TRUE), and now want to read each one with read_csv(). Write the for loop that will load them into a single data frame. (use two back slashes instead of forward slashes-back slashes are special characters so had to change here).

from jarnold

Hide

```
df <- vector("list", length(files))
for (fname in seq_along(files)) {
  df[[i]] <- read_csv(files[[i]])
}
df <- bind_rows(df)
```

2. What happens if you use for (nm in names(x)) and x has no names? What if only some of the elements are named? What if the names are not unique?

names(x) will yield NULL if there's no names, nothing will happen-the for loop will quietly end. If only some elements are named, then you'll get an error. If there are duplicates, that's fine-that'll work.

3. Write a function that prints the mean of each numeric column in a data frame, along with its name. For example, show_mean(iris) would print:

Hide

```
show_mean <- function(df, digits = 2) {
  # Get max length of any variable in the dataset
  maxstr <- max(str_length(names(df)))
  for (nm in names(df)) {
    if (is.numeric(df[[nm]])) {
      cat(str_c(str_pad(str_c(nm, ":"), maxstr + 1L, side = "right"),
                format(mean(df[[nm]]), digits = digits, nsmall = digits),
                sep = " "),
          "\n")
    }
  }
}
show_mean(iris)
```

```
## Sepal.Length: 5.84
## Sepal.Width:  3.06
## Petal.Length: 3.76
## Petal.Width:  1.20
```

from jarnold:

Hide

```
show_mean(iris)
```

```
## Sepal.Length: 5.84
## Sepal.Width:  3.06
## Petal.Length: 3.76
## Petal.Width:  1.20
```

4. What does this code do? How does it work?

Hide

```
trans <- list(
  disp = function(x) x * 0.0163871,
  am = function(x) {
    factor(x, labels = c("auto", "manual"))
  }
)
for (var in names(trans)) {
  mtcars[[var]] <- trans[[var]](mtcars[[var]])
}
```

trans is a named list of functions, and the for loop changes the same named columns in mtcars per those functions.

Exercise 21.4.1

1. Read the documentation for apply(). In the 2d case, what two for loops does it generalise?

It generalises for loops applied to the column, then elements in the resulting vector of a matrix.

2. Adapt col_summary() so that it only applies to numeric columns You might want to start with an is_numeric() function that returns a logical vector that has a TRUE corresponding to each numeric column.

from jarnold:

Hide

```
col_summary2 <- function(df, fun) {
  # test whether each colum is numeric
  numeric_cols <- vector("logical", length(df))
  for (i in seq_along(df)) {
    numeric_cols[[i]] <- is.numeric(df[[i]])
  }
  # indexes of numeric columns
  idxs <- seq_along(df)[numeric_cols]
  # number of numeric columns
  n <- sum(numeric_cols)
  out <- vector("double", n)
  for (i in idxs) {
    out[i] <- fun(df[[i]])
  }
  out
}
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = letters[1:10],
  d = rnorm(10)
)
col_summary2(df, mean)
```

```
## [1]  0.03865459  0.07936705  0.00000000 -0.87282580
```

Exercise 21.5.3

1. Write code that uses one of the map functions to:

1. Compute the mean of every column in mtcars.
2. Determine the type of each column in nycflights13::flights.
3. Compute the number of unique values in each column of iris.
4. Generate 10 random normals for each of μ = -10, 0, 10, 100

from jarnold

Hide

```
#1
purrr::map_dbl(mtcars, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##       qsec         vs         am       gear       carb
##  17.848750   0.437500   0.406250   3.687500   2.812500
```

Hide

```
#2
purrr::map(nycflights13::flights, class)
```

```
## $year
## [1] "integer"
##
## $month
## [1] "integer"
##
## $day
## [1] "integer"
##
## $dep_time
## [1] "integer"
##
## $sched_dep_time
## [1] "integer"
##
## $dep_delay
## [1] "numeric"
##
## $arr_time
## [1] "integer"
##
## $sched_arr_time
## [1] "integer"
##
## $arr_delay
## [1] "numeric"
##
## $carrier
## [1] "character"
##
## $flight
## [1] "integer"
##
## $tailnum
## [1] "character"
##
## $origin
## [1] "character"
##
## $dest
## [1] "character"
##
## $air_time
## [1] "numeric"
##
## $distance
## [1] "numeric"
##
## $hour
## [1] "numeric"
##
## $minute
## [1] "numeric"
##
## $time_hour
## [1] "POSIXct" "POSIXt"
```

Hide

```
purrr::map_chr(nycflights13::flights, typeof)
```

```
##           year          month            day       dep_time sched_dep_time
##      "integer"      "integer"      "integer"      "integer"      "integer"
##      dep_delay       arr_time sched_arr_time      arr_delay        carrier
##       "double"      "integer"      "integer"       "double"    "character"
##         flight        tailnum         origin           dest       air_time
##      "integer"    "character"    "character"    "character"       "double"
##       distance           hour         minute      time_hour
##       "double"       "double"       "double"       "double"
```

Hide

```
#3
purrr::map_int(iris, ~ length(unique(.)))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##           35           23           43           22            3
```

Hide

```
#4
purrr::map(c(-10, 0, 10, 100), rnorm, n = 10)
```

```
## [[1]]
##  [1]  -9.714045  -9.337733  -8.292421  -9.387098  -9.041342  -9.443563
##  [7] -10.318711  -8.969552  -9.422471 -10.758078
##
## [[2]]
##  [1]  1.2765188 -0.8132919 -0.5986963  0.4913592  0.9580939  1.5092699
##  [7] -0.3148165  1.5760822 -0.1095791  0.9415330
##
## [[3]]
##  [1] 10.275683  9.787535 10.223332 10.330234  8.878974  9.231961  9.299927
##  [8]  8.834318  9.746394  8.848766
##
## [[4]]
##  [1]  99.03635  99.74021 101.82065 100.12630  99.49620 101.29579  99.98857
##  [8]  99.59608 101.27721 101.52840
```

2. How can you create a single vector that for each column in a data frame indicates whether or not it's a factor?

from jarnold

Hide

```
map_lgl(mtcars, is.factor)
```

```
##   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

3. What happens when you use the map functions on vectors that aren't lists? What does map(1:5, runif) do? Why?

Hide

```
purrr::map(1:5,runif)
```

```
## [[1]]
## [1] 0.1057802
##
## [[2]]
## [1] 0.7530251 0.3675610
##
## [[3]]
## [1] 0.2397398 0.7586907 0.4187146
##
## [[4]]
## [1] 0.7071993 0.5419450 0.6088576 0.5425923
##
## [[5]]
## [1] 0.6672391 0.9163422 0.2454936 0.2145345 0.4969023
```

applies runif(n = each_element_of_1:5)

4. What does map(-2:2, rnorm, n = 5) do? Why? What does map_dbl(-2:2, rnorm, n = 5) do? Why?

Hide

```
#1
purrr::map(-2:2, rnorm, n = 5)
```

```
## [[1]]
## [1] -3.0809401 -1.8159872 -1.0138979 -1.1328492  0.4046854
##
## [[2]]
## [1] -0.5246244 -1.7789792 -0.9934616 -0.5246491  1.5434798
##
## [[3]]
## [1] -0.3253424  0.8292342 -0.8311809 -0.6667747 -1.7118878
##
## [[4]]
## [1]  0.9946182  0.2268039  0.7243463  0.4898557 -0.3413743
##
## [[5]]
## [1] 0.9239205 2.7275130 1.7610069 2.5645018 1.2867388
```

Hide

```
#2
#purrr::map_dbl(-2:2, rnorm, n = 5)
```

#1 uses the mean value given in -2:2 to generate 5 rnorms, and in #2 throws an error because map_dbl expects to return a single vector-we can do that by nesting the map function call in flatten_dbl.

5. Rewrite map(x, function(df) lm(mpg ~ wt, data = df)) to eliminate the anonymous function.

put mtcars into a list and the map functions can then use it.

Hide

```
purrr::map( list(mtcars), ~lm(mpg ~ wt, data = .) )
```

```
## [[1]]
##
## Call:
## lm(formula = mpg ~ wt, data = .)
##
## Coefficients:
## (Intercept)           wt
##      37.285       -5.344
```

Exercise 21.9.3

1. Implement your own version of every() using a for loop. Compare it with purrr::every(). What does purrr's version do that your version doesn't?

from jarnold

Hide

```
# Use ... to pass arguments to the function
every2 <- function(.x, .p, ...) {
  for (i in .x) {
    if (!.p(i, ...)) {
      # If any is FALSE we know not all of then were TRUE
      return(FALSE)
    }
  }
  # if nothing was FALSE, then it is TRUE
  TRUE
}

every2(1:3, function(x) {x > 1})
```

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

Hide

```
every2(1:3, function(x) {x > 0})
```

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

The function purrr::every does fancy things with .p, like taking a logical vector instead of a function, or being able to test part of a string if the elements of .x are lists.

2. Create an enhanced col_sum() that applies a summary function to every numeric column in a data frame.

from jarnold

Hide

```
#Note this question has a typo. It is referring to col_summary.

#I will use map to apply the function to all the columns, and keep to only select numeric columns.

col_sum2 <- function(df, f, ...) {
  purrr::map(keep(df, is.numeric), f, ...)
}
col_sum2(iris, mean)
```

```
## $Sepal.Length
## [1] 5.843333
##
## $Sepal.Width
## [1] 3.057333
##
## $Petal.Length
## [1] 3.758
##
## $Petal.Width
## [1] 1.199333
```

3. A possible base R equivalent of col_sum() is:

Hide

```
col_sum3 <- function(df, f) {
  is_num <- sapply(df, is.numeric)
  df_num <- df[, is_num]

  sapply(df_num, f)
}
```

But it has a number of bugs as illustrated with the following inputs:

Hide

```
df <- dplyr::tibble(
  x = 1:3,
  y = 3:1,
  z = c("a", "b", "c")
)
# OK
col_sum3(df, mean)
# Has problems: don't always return numeric vector
#col_sum3(df[1:2], mean)
#col_sum3(df[1], mean)
#col_sum3(df[0], mean)
```

What causes the bugs?

from jarnold

The problem is that sapply doesn't always return numeric vectors. If no columns are selected, instead of gracefully exiting, it returns an empty list. This causes an error since we can't use a list with [.

Hide

```
sapply(df[0], is.numeric)
```

```
## named list()
```

Hide

```
sapply(df[1], is.numeric)
```

```
##    a
## TRUE
```

Hide

```
sapply(df[1:2], is.numeric)
```

```
##    a    b
## TRUE TRUE
```

Part IV Model

22. Introduction

No Exercises

23. Model Basics

Exercise 23.2.1

1. One downside of the linear model is that it is sensitive to unusual values because the distance incorporates a squared term. Fit a linear model to the simulated data below, and visualise the results. Rerun a few times to generate different simulated datasets. What do you notice about the model?

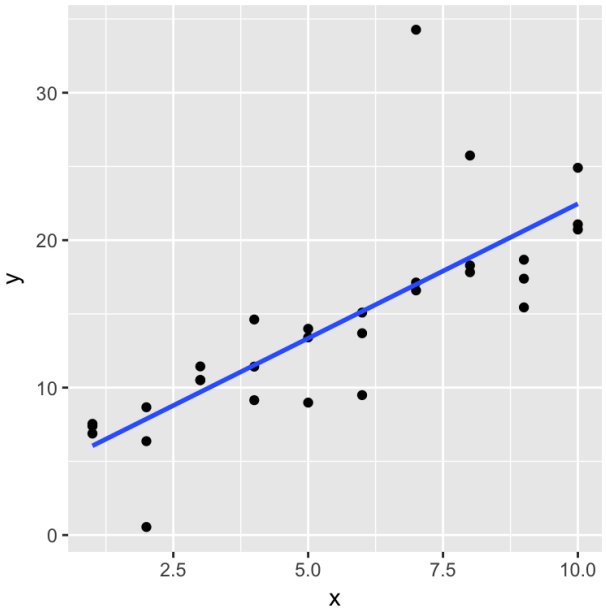
Hide

```
library(dplyr)
library(modelr)

sim1a <- tibble(
  x = rep(1:10, each = 3),
  y = x * 1.5 + 6 + rt(length(x), df = 2)
)
```

Let's view this with a model using ggplot

Hide

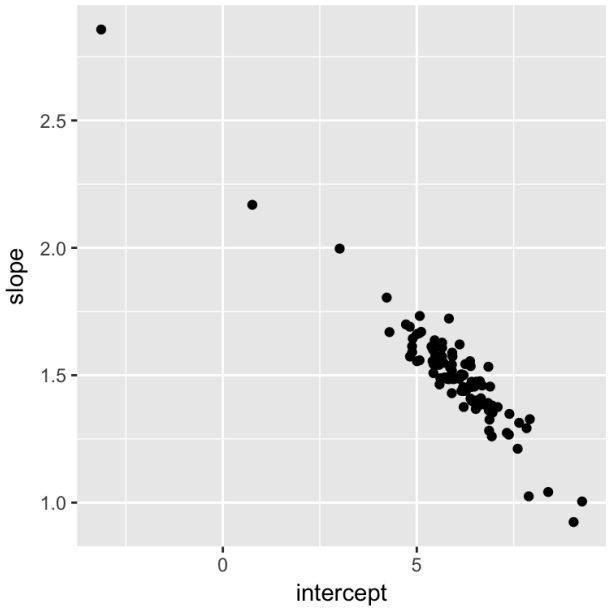
```
library(ggplot2)

ggplot(sim1a, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm",se = F)
```



If we rerun to get a new *sim1a*, I'll use the solution from jarnold that computes this well.

Hide

```
simt <- function(i) {
  tibble(
    x = rep(1:10, each = 3),
    y = x * 1.5 + 6 + rt(length(x), df = 2),
    .id = i
  )
}

lm_df <- function(.data) {
  mod <- lm(y ~ x, data = .data)
  beta <- coef(mod)
  tibble(intercept = beta[1], slope = beta[2])
}

sims <- purrr::map(1:100, simt) %>%
  purrr::map_df(lm_df)

ggplot(sims, aes(x = intercept, y = slope)) +
  geom_point()
```



This is showing that The slope decreases as the intercept increases, which in turn is regulated by where the data lie. So the linear models and the slopes/intercepts are determined by the variability present in the data (outliers).

2. One way to make linear models more robust is to use a different distance measure. For example, instead of root-mean-squared distance, you could use mean-absolute distance:

Hide

```
measure_distance <- function(mod, data) {
  diff <- data$y - make_prediction(mod, data)
  mean(abs(diff))
}
```

Use optim() to fit this model to the simulated data above and compare it to the linear model.

First, there is no *make_prediction* function so I won't be using that measure. However, I'll still use optim to find the best distance.

Hide

```
sim1a <- tibble(
  x = rep(1:10, each = 3),
  y = x * 1.5 + 6 + rt(length(x), df = 2)
)

model1 <- function(a, data) {
  a[1] + data$x * a[2]
}

measure_distance <- function(mod, data) {
  diff <- data$y - model1(mod, data)
  sqrt(mean(diff ^ 2))
}

#finds minimum distance from the initial values
best <- optim(par=c(0, 0), fn=measure_distance, data = sim1a)

#best model (intercept, slope) values
best$par
```

```
## [1] 8.513129 1.224304
```

3. One challenge with performing numerical optimisation is that it's only guaranteed to find one local optima. What's the problem with optimising a three parameter model like this?

Hide

```
model1 <- function(a, data) {
  a[1] + data$x * a[2] + a[3]
}
```

Let's see

Hide

```
sim1a <- tibble(
  x = rep(1:10, each = 3),
  y = x * 1.5 + 6 + rt(length(x), df = 2)
)

measure_distance <- function(mod, data) {
  diff <- data$y - model1(mod, data)
  sqrt(mean(diff ^ 2))
}

#finds minimum distance from the initial values
best <- optim(par=c(0, 0, 0), fn=measure_distance, data = sim1a)

#best model (intercept, slope) values
best$par
```

```
## [1] -5.2391455  0.9267704 15.3017119
```

It works. However, the way the model is doesn't disambiguate between a[1] and a[3], meaning the model can have multiple optimum values. So that might be a problem (actually, it contradicts the term optimum since it implies only one solution).

Exercise 23.3.2

1. Instead of using lm() to fit a straight line, you can use loess() to fit a smooth curve. Repeat the process of model fitting, grid generation, predictions, and visualisation on sim1 using loess() instead of lm(). How does the result compare to geom_smooth()?

I'll use the solution from jarnold, but I'll detail his process

Hide

```
sim1
```

```
## # A tibble: 30 x 2
##        x         y
##    <int>     <dbl>
##  1     1  4.199913
##  2     1  7.510634
##  3     1  2.125473
##  4     2  8.988857
##  5     2 10.243105
##  6     2 11.296823
##  7     3  7.356365
##  8     3 10.505349
##  9     3 10.511601
## 10     4 12.434589
## # ... with 20 more rows
```

Hide

```
sim1_loess <- loess(y ~ x, data = sim1)
sim1_loess
```

```
## Call:
## loess(formula = y ~ x, data = sim1)
##
## Number of Observations: 30
## Equivalent Number of Parameters: 4.19
## Residual Standard Error: 2.262
```

Hide

```
grid_loess <- sim1 %>%
  add_predictions(sim1_loess)
grid_loess
```

```
## # A tibble: 30 x 3
##        x         y      pred
##    <int>     <dbl>     <dbl>
##  1     1  4.199913  5.338000
##  2     1  7.510634  5.338000
##  3     1  2.125473  5.338000
##  4     2  8.988857  8.274913
##  5     2 10.243105  8.274913
##  6     2 11.296823  8.274913
##  7     3  7.356365 10.809582
##  8     3 10.505349 10.809582
##  9     3 10.511601 10.809582
## 10     4 12.434589 12.779762
## # ... with 20 more rows
```

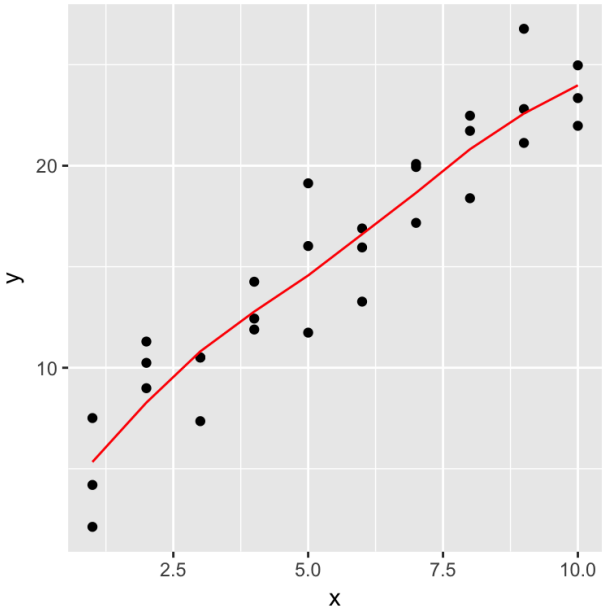
Hide

```
sim1 <- sim1 %>%
  add_residuals(sim1_loess, var = "resid_loess") %>%
  add_predictions(sim1_loess, var = "pred_loess")
sim1
```

```
## # A tibble: 30 x 4
##        x         y resid_loess pred_loess
##    <int>     <dbl>       <dbl>      <dbl>
##  1     1  4.199913  -1.1380871   5.338000
##  2     1  7.510634   2.1726340   5.338000
##  3     1  2.125473  -3.2125273   5.338000
##  4     2  8.988857   0.7139447   8.274913
##  5     2 10.243105   1.9681928   8.274913
##  6     2 11.296823   3.0219105   8.274913
##  7     3  7.356365  -3.4532176  10.809582
##  8     3 10.505349  -0.3042329  10.809582
##  9     3 10.511601  -0.2979815  10.809582
## 10     4 12.434589  -0.3451730  12.779762
## # ... with 20 more rows
```

Hide

```
plot_sim1_loess <-
  ggplot(sim1, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(x = x, y = pred), data = grid_loess, colour = "red")
plot_sim1_loess
```



you could also use the loess method within geom_smooth.

2. add_predictions() is paired with gather_predictions() and spread_predictions(). How do these three functions differ?

From the help of *add_predictions*:

add_prediction adds a single new column, .pred, to the input data. spread_predictions adds one column for each model. gather_prections adds two columns .model and .pred, and repeats the input rows for each model.

So gather makes a model column as well, there's one column for the predictions called *pred*. Spread does not but has the model name as the column name.

3. What does geom_ref_line() do? What package does it come from? Why is displaying a reference line in plots showing residuals useful and important?

It comes from *ggplot2*, and it just adds a line (specified) that allows easy comparison with the plotted data. It's good to use with showing residuals cause then you can see if your model is appropriate for your data (kinda random residuals) versus not (higher residuals in certain places). This can help you choose a useful model.

4. Why might you want to look at a frequency polygon of absolute residuals? What are the pros and cons compared to looking at the raw residuals?

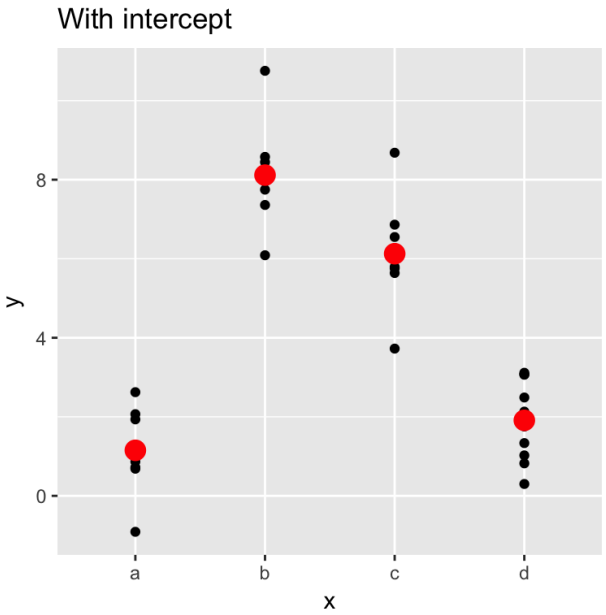
Like I mentioned in Q3, it would be useful to see what the distribution of residuals is. Geom_freqpoly helps with that-so visually any patterns pop out at you.

Exercise 23.4.5

1. What happens if you repeat the analysis of sim2 using a model without an intercept. What happens to the model equation? What happens to the predictions?

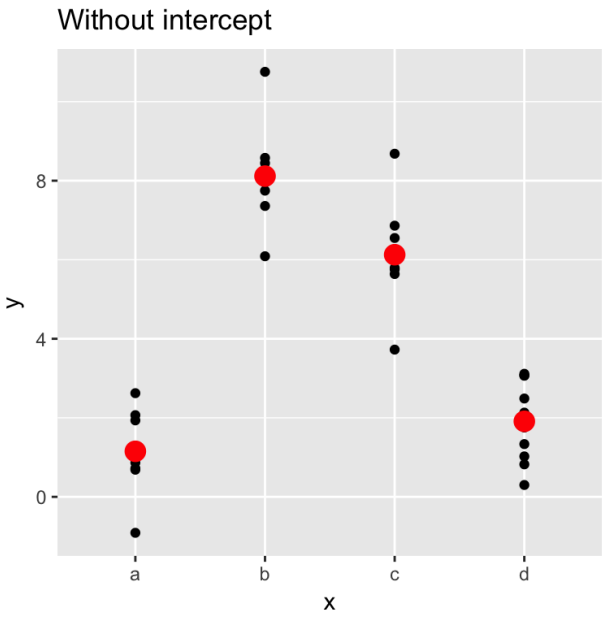
Hide

```
mod2 <- lm(y ~ x, data = sim2)

grid <- sim2 %>%
  data_grid(x) %>%
  add_predictions(mod2)

ggplot(sim2, aes(x)) +
  geom_point(aes(y = y)) +
  geom_point(data = grid, aes(y = pred), colour = "red", size = 4) +
  ggtitle("With intercept")
```



Hide

```
mod2 <- lm(y ~ x - 1, data = sim2)

grid <- sim2 %>%
  data_grid(x) %>%
  add_predictions(mod2)

ggplot(sim2, aes(x)) +
  geom_point(aes(y = y)) +
  geom_point(data = grid, aes(y = pred), colour = "red", size = 4) +
  ggtitle("Without intercept")
```



Nothing happens to the predictions. And the intercept column is just taken out from the model equation. Without the intercept, the other predictor is included in the equation because then the intercept can't be used to make that lost column (xa) with the intercept.

Hide

```
model_matrix(sim2, y ~ x)
```

```
## # A tibble: 40 x 4
##    `(Intercept)`    xb    xc    xd
##            <dbl> <dbl> <dbl> <dbl>
##  1             1     0     0     0
##  2             1     0     0     0
##  3             1     0     0     0
##  4             1     0     0     0
##  5             1     0     0     0
##  6             1     0     0     0
##  7             1     0     0     0
##  8             1     0     0     0
##  9             1     0     0     0
## 10             1     0     0     0
## # ... with 30 more rows
```

Hide

```
model_matrix(sim2, y ~ x - 1)
```

```
## # A tibble: 40 x 4
##       xa    xb    xc    xd
##    <dbl> <dbl> <dbl> <dbl>
##  1     1     0     0     0
##  2     1     0     0     0
##  3     1     0     0     0
##  4     1     0     0     0
##  5     1     0     0     0
##  6     1     0     0     0
##  7     1     0     0     0
##  8     1     0     0     0
##  9     1     0     0     0
## 10     1     0     0     0
## # ... with 30 more rows
```

2. Use model_matrix() to explore the equations generated for the models I fit to sim3 and sim4. Why is * a good shorthand for interaction?

Hide

```
model_matrix(sim3, y ~ x1 + x2) %>% head()
```

```
## # A tibble: 6 x 5
##   `(Intercept)`    x1   x2b   x2c   x2d
##           <dbl> <dbl> <dbl> <dbl> <dbl>
## 1             1     1     0     0     0
## 2             1     1     0     0     0
## 3             1     1     0     0     0
## 4             1     1     1     0     0
## 5             1     1     1     0     0
## 6             1     1     1     0     0
```

Hide

```
model_matrix(sim3, y ~ x1 * x2) %>% head()
```

```
## # A tibble: 6 x 8
##   `(Intercept)`    x1   x2b   x2c   x2d `x1:x2b` `x1:x2c` `x1:x2d`
##           <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>    <dbl>
## 1             1     1     0     0     0        0        0        0
## 2             1     1     0     0     0        0        0        0
## 3             1     1     0     0     0        0        0        0
## 4             1     1     1     0     0        1        0        0
## 5             1     1     1     0     0        1        0        0
## 6             1     1     1     0     0        1        0        0
```

Hide

```
model_matrix(sim4, y ~ x1 + x2) %>% head()
```

```
## # A tibble: 6 x 3
##   `(Intercept)`    x1         x2
##           <dbl> <dbl>      <dbl>
## 1             1    -1 -1.0000000
## 2             1    -1 -1.0000000
## 3             1    -1 -1.0000000
## 4             1    -1 -0.7777778
## 5             1    -1 -0.7777778
## 6             1    -1 -0.7777778
```

Hide

```
model_matrix(sim4, y ~ x1 * x2) %>% head()
```

```
## # A tibble: 6 x 4
##   `(Intercept)`    x1         x2   `x1:x2`
##           <dbl> <dbl>      <dbl>     <dbl>
## 1             1    -1 -1.0000000 1.0000000
## 2             1    -1 -1.0000000 1.0000000
## 3             1    -1 -1.0000000 1.0000000
## 4             1    -1 -0.7777778 0.7777778
## 5             1    -1 -0.7777778 0.7777778
## 6             1    -1 -0.7777778 0.7777778
```

It's a good shorthand because then I don't have to manually specify.

3. Using the basic principles, convert the formulas in the following two models into functions. (Hint: start by converting the categorical variable into 0-1 variables.)

Hide

```
#a_0 + a_1 * x1 + a_2 * x2
mod1 <- lm(y ~ x1 + x2, data = sim3)

#a_0 + a_1 * x1 + a_2 * x2 + a_12 * x1:x2
mod2 <- lm(y ~ x1 * x2, data = sim3)
```

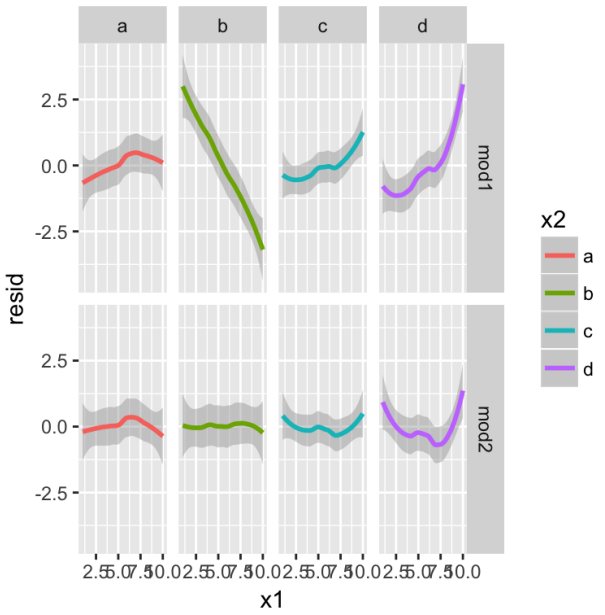
I'm not making the functions, but basically for each predicting you make a column and the value is the value for the predictor, but only if it's non-categorical. For categorical predictors, you spread them into separate columns for each factor. For interactions, you give 1 if both are 1 for the predictors and 0 otherwise.

4. For sim4, which of mod1 and mod2 is better? I think mod2 does a slightly better job at removing patterns, but it's pretty subtle. Can you come up with a plot to support my claim?

To better see the pattern, instead of points I'd use the loess method in ggplot to see how the line deviates from 0, indicating the residuals are larger in particular areas which indicates a bad model.

Hide

```
mod1 <- lm(y ~ x1 + x2, data = sim3)
mod2 <- lm(y ~ x1 * x2, data = sim3)

sim3 <- sim3 %>%
  gather_residuals(mod1, mod2)

ggplot(sim3, aes(x1, resid, colour = x2)) +
  geom_smooth(method="loess") +
  facet_grid(model ~ x2)
```



24. Model Building

Exercise 24.2.3

1. In the plot of lcarat vs. lprice, there are some bright vertical strips. What do they represent?

The bright vertical stripes in the lprice vs. lcarat plot represent areas of high counts (very dense) of diamonds. These areas are actually at round or human-friendly carat cuts.

2. If log(price) = a_0 + a_1 * log(carat), what does that say about the relationship between price and carat?

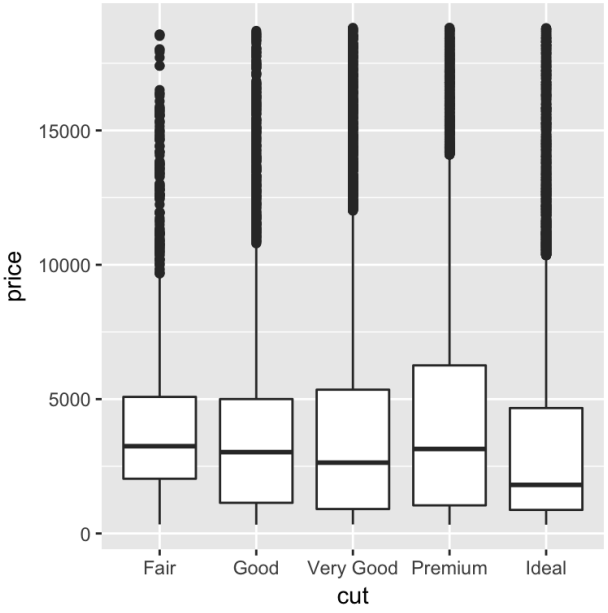
from jarnold,

An 1% increase in carat is associated with an a1a1 % increase in price.

3. Extract the diamonds that have very high and very low residuals. Is there anything unusual about these diamonds? Are the particularly bad or good, or do you think these are pricing errors?

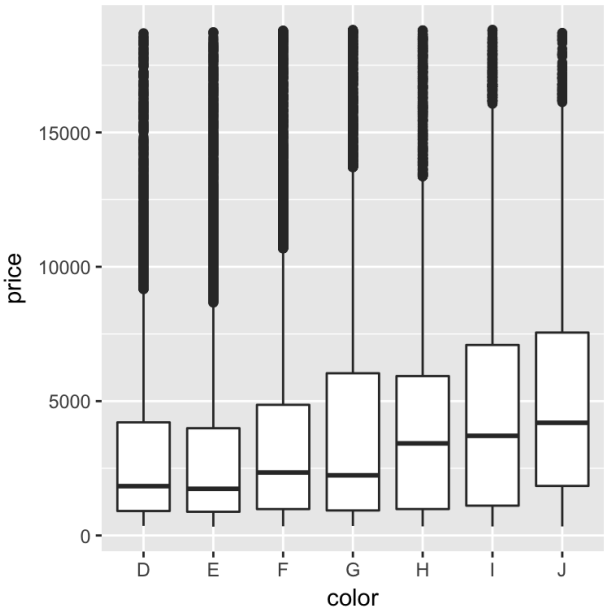
Hide

```
library(dplyr)
library(ggplot2)
library(modelr)

diamonds2 <- diamonds %>%
  filter(carat <= 2.5) %>%
  mutate(lprice = log2(price), lcarat = log2(carat))

mod_diamond <- lm(lprice ~ lcarat, data = diamonds2)

diamonds2 <- diamonds2 %>%
  add_residuals(mod_diamond, "lresid")

resid_quants <- quantile(diamonds2$lresid)

filtered <- diamonds2 %>%
  filter(
    !((lresid < resid_quants[["25%"]]) & (lresid > resid_quants[["75%"]]))
    )

ggplot(filtered, aes(cut, price)) + geom_boxplot()
```



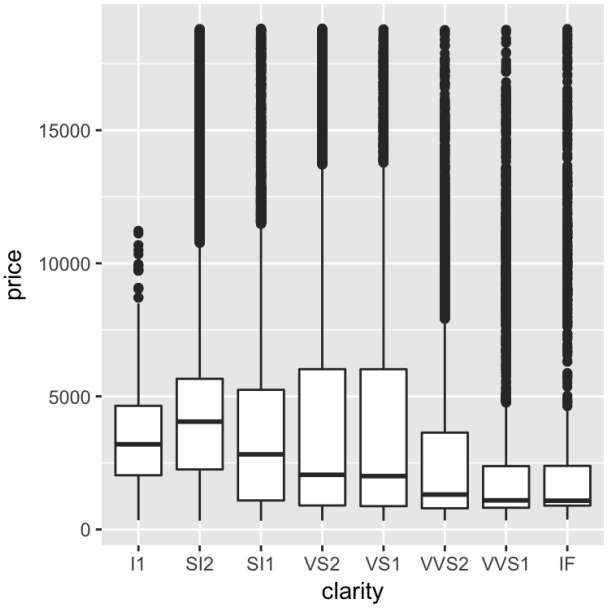
Hide

```
ggplot(filtered, aes(color, price)) + geom_boxplot()
```



Hide

```
ggplot(filtered, aes(clarity, price)) + geom_boxplot()
```

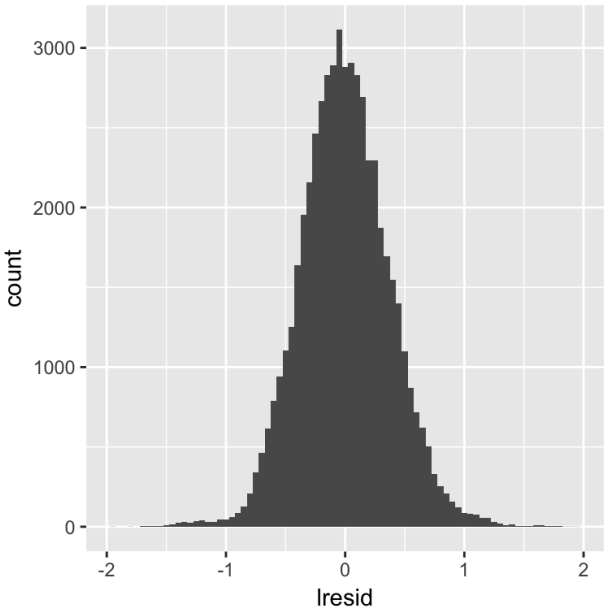


From the plots, they don't seem to be pricing errors since there aren't any strong trends of cut, color or clarity with price after filtering for extreme values of residuals in the predictions.

4. Does the final model, mod_diamonds2, do a good job of predicting diamond prices? Would you trust it to tell you how much to spend if you were buying a diamond?

Hide

```
mod_diamond2 <- lm(lprice ~ lcarat + color + cut + clarity, data = diamonds2)

diamonds2 <- diamonds2 %>%
  add_residuals(mod_diamond2, "lresid2")

diamonds2 %>% ggplot() +
  geom_histogram(aes(lresid),binwidth=.05)
```



from jarnold,

Hide

```
diamonds2 %>%
  add_predictions(mod_diamond2) %>%
  add_residuals(mod_diamond2) %>%
  summarise(sq_err = sqrt(mean(resid^2)),
            abs_err = mean(abs(resid)),
            p975_err = quantile(resid, 0.975),
            p025_err = quantile(resid, 0.025))
```

```
## # A tibble: 1 x 4
##     sq_err   abs_err  p975_err   p025_err
##      <dbl>     <dbl>     <dbl>      <dbl>
## 1 0.191524 0.1491158 0.3844299 -0.3692446
```

The average squared and absolute errors are $2^{0.19}$=1.14 and $2^{0.10}$ so on average, the error is ±10−−15%. And the 95% range of residuals is about $2^{0.37}$=1.3 so within ±30%. This doesn't seem terrible to me.

Exercise 24.3.5

1. Use your Google sleuthing skills to brainstorm why there were fewer than expected flights on Jan 20, May 26, and Sep 1. (Hint: they all have the same explanation.) How would these days generalise to another year?

from jarnold,

These are the Sundays before Monday holidays Martin Luther King Day, Memorial Day, and Labor Day.

2. What do the three days with high positive residuals represent? How would these days generalise to another year?

Hide

```
library("nycflights13")
daily <- flights %>%
  mutate(date = make_date(year, month, day)) %>%
  group_by(date) %>%
  summarise(n = n())

daily <- daily %>%
  mutate(wday = wday(date, label = TRUE))

term <- function(date) {
  cut(date,
    breaks = ymd(20130101, 20130605, 20130825, 20140101),
    labels = c("spring", "summer", "fall")
  )
}

daily <- daily %>%
  mutate(term = term(date))

mod3 <- MASS::rlm(n ~ wday * term, data = daily)

daily <- daily %>%
  add_residuals(mod3, "resid")

daily %>%
  top_n(3, resid)
```

```
## # A tibble: 3 x 5
##         date     n  wday   term    resid
##       <date> <int> <ord> <fctr>    <dbl>
## 1 2013-11-30   857   Sat   fall 160.1007
## 2 2013-12-21   811   Sat   fall 114.1007
## 3 2013-12-28   814   Sat   fall 117.1007
```

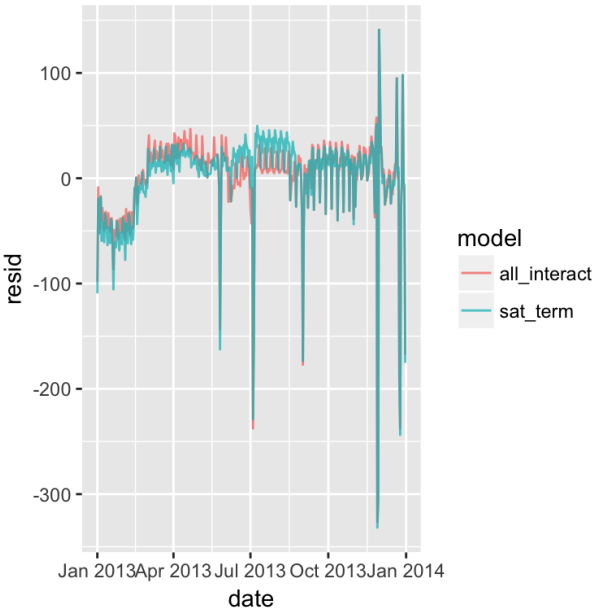
These three days represent saturdays before or after major holidays (thanksgiving and christmas). Traveling on those days from home is probably particularly high.

3. Create a new variable that splits the wday variable into terms, but only for Saturdays, i.e. it should have Thurs, Fri, but Sat-summer, Sat-spring, Sat-fall. How does this model compare with the model with every combination of wday and term?

from jarnold,

Hide

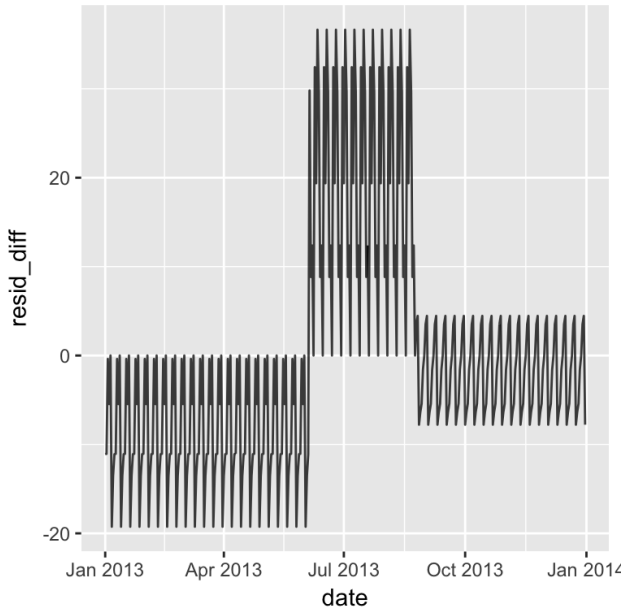
```
daily <- daily %>%
  mutate(wday2 =
         case_when(.$wday == "Sat" & .$term == "summer" ~ "Sat-summer",
         .$wday == "Sat" & .$ term == "fall" ~ "Sat-fall",
         .$wday == "Sat" & .$term == "spring" ~ "Sat-spring",
         TRUE ~ as.character(.$wday)))

mod2 <- lm(n ~ wday * term, data = daily)

mod4 <- lm(n ~ wday2, data = daily)

daily %>%
  gather_residuals(sat_term = mod4, all_interact = mod2) %>%
  ggplot(aes(date, resid, colour = model)) +
    geom_line(alpha = 0.75)
```



I think the overlapping plot is hard to understand. If we are interested in the differences, it is better to plot the differences directly. In this code I use spread_residuals to add one column per model, rather than gather_residuals which creates a new row for each model.

Hide

```
daily %>%
  spread_residuals(sat_term = mod4, all_interact = mod2) %>%
  mutate(resid_diff = sat_term - all_interact) %>%
  ggplot(aes(date, resid_diff)) +
    geom_line(alpha = 0.75)
```



The model with terms x Saturday has higher residuals in the fall, and lower residuals in the spring than the model with all interactions.

Using overall model comparison terms, mod4 has a lower R2 and regression standard error, σ^ , despite using fewer variables. More importantly for prediction purposes, it has a higher AIC - which is an estimate of the out of sample error.

Hide

```
library(broom)

glance(mod4) %>% select(r.squared, sigma, AIC, df)
```

```
##   r.squared    sigma      AIC df
## 1 0.7356615 47.35969 3862.885  9
```

Hide

```
glance(mod2) %>% select(r.squared, sigma, AIC, df)
```

```
##   r.squared    sigma     AIC df
## 1  0.757289 46.16568 3855.73 21
```

4. Create a new wday variable that combines the day of week, term (for Saturdays), and public holidays. What do the residuals of that model look like?

from jarnold,

The question is unclear how to handle the public holidays. We could include a dummy for all public holidays? or the Sunday before public holidays?

Including a level for the public holidays themselves is insufficient because (1) public holiday's effects on travel varies dramatically, (2) the effect can occur on the day itself or the day before and after, and (3) with Thanksgiving and Christmas there are increases in travel as well.

Hide

```
daily <- daily %>%
  mutate(wday3 =
         case_when(
           .$date %in% lubridate::ymd(c(20130101, # new years
                                        20130121, # mlk
                                        20130218, # presidents
                                        20130527, # memorial
                                        20130704, # independence
                                        20130902, # labor
                                        20131028, # columbus
                                        20131111, # veterans
                                        20131128, # thanksgiving
                                        20131225)) ~
             "holiday",
           .$wday == "Sat" & .$term == "summer" ~ "Sat-summer",
           .$wday == "Sat" & .$ term == "fall" ~ "Sat-fall",
           .$wday == "Sat" & .$term == "spring" ~ "Sat-spring",
           TRUE ~ as.character(.$wday)))

mod5 <- lm(n ~ wday3, data = daily)

daily %>%
  spread_residuals(mod5) %>%
  arrange(desc(abs(resid))) %>%
  slice(1:20) %>% select(date, wday, resid)
```

```
## # A tibble: 20 x 3
##          date  wday      resid
##        <date> <ord>      <dbl>
##  1 2013-11-28 Thurs -347.53717
##  2 2013-11-29   Fri -321.43256
##  3 2013-07-04 Thurs -257.03213
##  4 2013-12-25   Wed -248.56250
##  5 2013-12-24  Tues -200.14703
##  6 2013-12-31  Tues -185.14703
##  7 2013-09-01   Sun -182.08451
##  8 2013-07-05   Fri -170.03213
##  9 2013-05-26   Sun -164.49563
## 10 2013-11-30   Sat  160.10068
## 11 2013-12-28   Sat  117.10068
## 12 2013-12-21   Sat  114.10068
## 13 2013-01-01  Tues -110.83875
## 14 2013-01-20   Sun -107.49563
## 15 2013-12-01   Sun   86.91549
## 16 2013-02-03   Sun  -79.49563
## 17 2013-01-19   Sat  -72.81905
## 18 2013-01-27   Sun  -70.49563
## 19 2013-01-26   Sat  -66.81905
## 20 2013-01-13   Sun  -65.49563
```

5. What happens if you fit a day of week effect that varies by month (i.e. n ~ wday * month)? Why is this not very helpful?

You don't have many samples (4-5) per month so your power to predict is pretty small.

6. What would you expect the model n ~ wday + ns(date, 5) to look like? Knowing what you know about the data, why would you expect it to be not particularly effective?

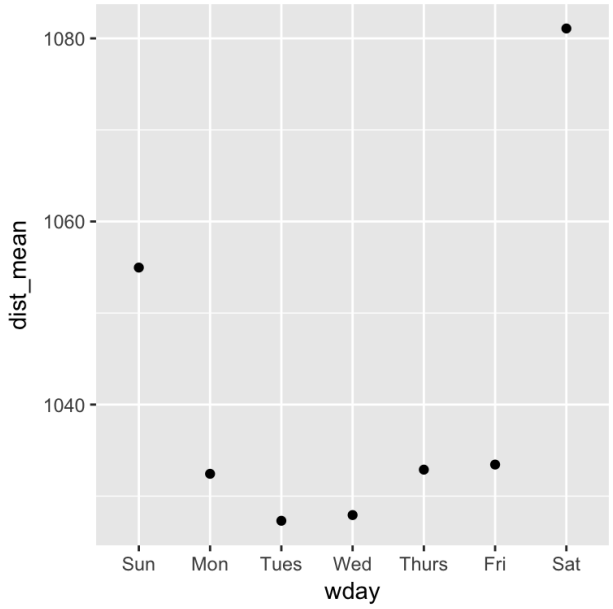
There isn't really a weekday trend across months, but really a trend around holidays and other miscallaneous things so the model wouldn't really take advantage of that.

7. We hypothesised that people leaving on Sundays are more likely to be business travellers who need to be somewhere on Monday. Explore that hypothesis by seeing how it breaks down based on distance and time: if it's true, you'd expect to see more Sunday evening flights to places that are far away

from jarnold,

Hide

```
flights %>%
  mutate(date = make_date(year, month, day),
         wday = wday(date, label = TRUE)) %>%
  group_by(wday) %>%
  summarise(dist_mean =  mean(distance),
            dist_median = median(distance)) %>%
  ggplot(aes(y = dist_mean, x = wday)) +
  geom_point()
```



Hide

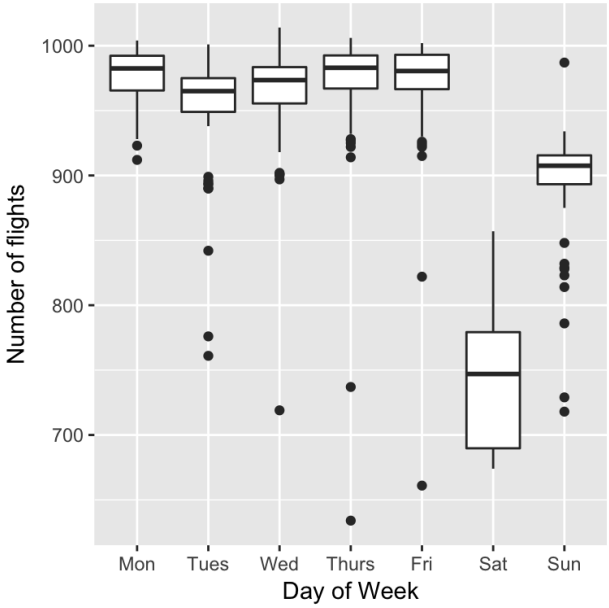
```
flights %>%
  mutate(date = make_date(year, month, day),
         wday = wday(date, label = TRUE)) %>%
  group_by(wday, hour) %>%
  summarise(dist_mean =  mean(distance),
            dist_median = median(distance)) %>%
  ggplot(aes(y = dist_mean, x = hour, colour = wday)) +
  geom_point() +
  geom_line()
```

Maybe someone can look at the Sunday evening/Monday scheduled arrivals?

8. It's a little frustrating that Sunday and Saturday are on separate ends of the plot. Write a small function to set the levels of the factor so that the week starts on Monday.

Hide

```
monday_first <- function(x) {
  forcats::fct_relevel(x, levels(x)[-1])
}

daily <- daily %>%
  mutate(wday = wday(date, label = TRUE))
ggplot(daily, aes(monday_first(wday), n)) +
  geom_boxplot() +
  labs(x = "Day of Week", y = "Number of flights")
```



25. Many Models

Exercise 25.2.5

1. A linear trend seems to be slightly too simple for the overall trend. Can you do better with a quadratic polynomial? How can you interpret the coefficients of the quadratic? (Hint you might want to transform year so that it has mean zero.)

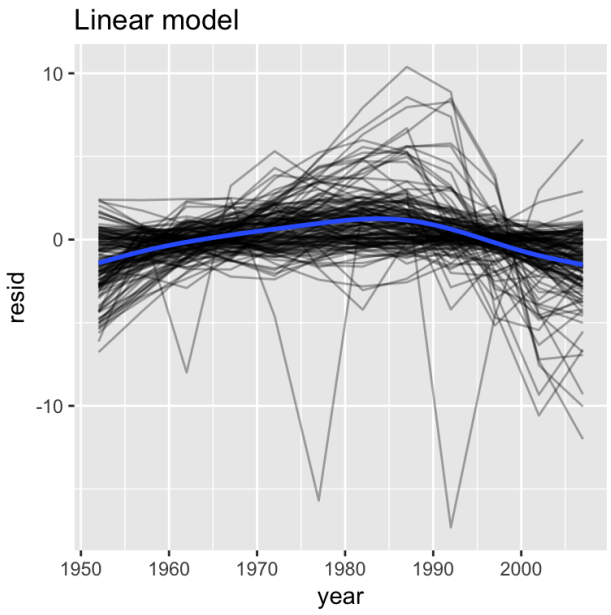
Hide

```
print("here")
```

```
## [1] "here"
```

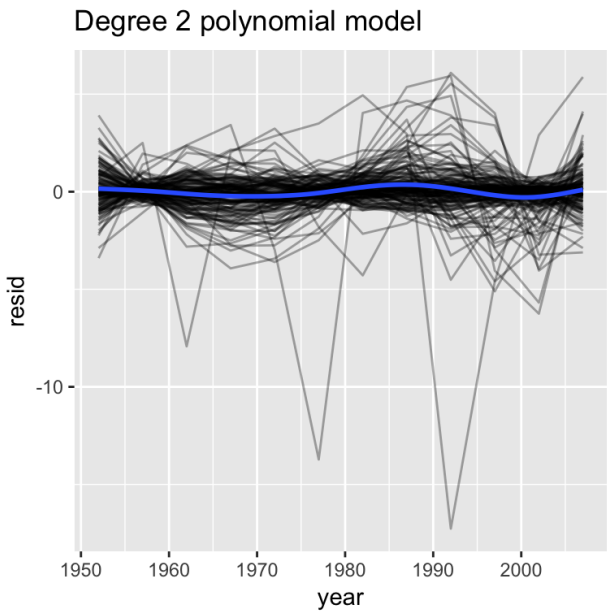
Hide

```
library(modelr)
library(tidyverse)
if(!require(gapminder)){install.packages("gapminder")}
library(gapminder)

by_country <- gapminder %>%
  group_by(country, continent) %>%
  nest()

by_country <- by_country %>%
  mutate(
    one_mod = purrr::map(data, ~ lm(lifeExp ~ poly(year,1),data = .)),
    two_mod = purrr::map(data, ~ lm(lifeExp ~ poly(year,2),data = .)),
    three_mod = purrr::map(data, ~ lm(lifeExp ~ poly(year,3),data = .))
         )

by_country <- by_country %>%
  mutate(
    one_resids = purrr::map2(data, one_mod, add_residuals),
    two_resids = purrr::map2(data, two_mod, add_residuals),
    three_resids = purrr::map2(data, three_mod, add_residuals)
  )

one_resids <- unnest(by_country, one_resids)
two_resids <- unnest(by_country, two_resids)
three_resids <- unnest(by_country, three_resids)

plot <- function(df,title){
    df %>%
    ggplot(aes(year, resid)) +
    geom_line(aes(group = country), alpha = 1 / 3) +
    geom_smooth(se = FALSE) +
    ggtitle(title)
}

plot(one_resids,"Linear model")
```



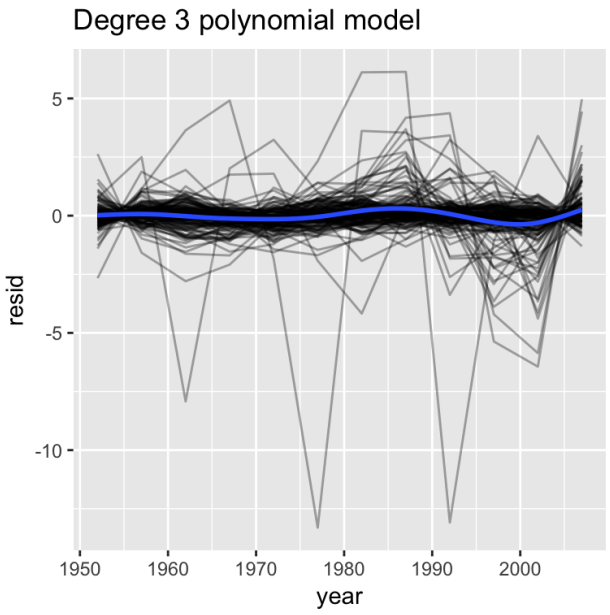
Hide

```
plot(two_resids,"Degree 2 polynomial model")
```



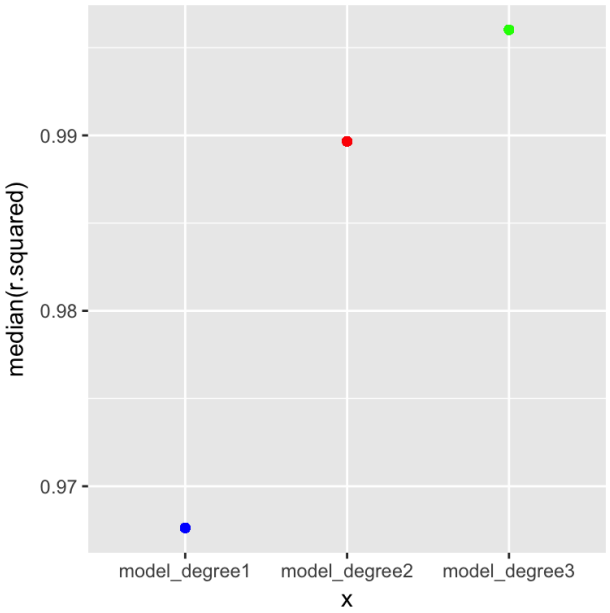
Hide

```
plot(three_resids,"Degree 3 polynomial model")
```



Hide

```
glance <- by_country %>%
  mutate(one_glance = purrr::map(one_mod, broom::glance),
         two_glance = purrr::map(two_mod, broom::glance),
         three_glance = purrr::map(three_mod, broom::glance)) %>%
  unnest(one_glance,two_glance,three_glance, .drop = TRUE)



glance %>%
  ggplot() +
  geom_point(aes("model_degree1",median(r.squared)),color="blue") +
  geom_point(aes("model_degree2",median(r.squared1)),color="red") +
  geom_point(aes("model_degree3",median(r.squared2)),color="green")
```



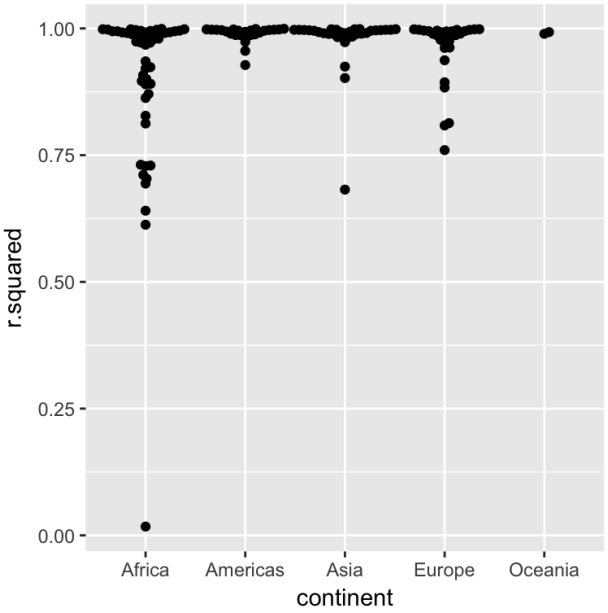
I think I violated the “no copy/paste more than two times” rule. But that’s learning for another time, the point is you can see the model has a better fit when increasing the polynomial degree in the model fitted to the data. How can the coefficients be interpreted? They allow extra flexibilty in fitting the data-so the quadratic terms account for deviations from a linear trend.

2. Explore other methods for visualising the distribution of R2 per continent. You might want to try the ggbeeswarm package, which provides similar methods for avoiding overlaps as jitter, but uses deterministic methods.

from jarnold,

Hide

```
library(gapminder)
country_model <- function(df) {
  lm(lifeExp ~ poly(year - median(year), 2), data = df)
}

by_country <- gapminder %>%
  group_by(country, continent) %>%
  nest()

by_country <- by_country %>%
  mutate(model = purrr::map(data, country_model))

library("ggbeeswarm")
by_country %>%
  mutate(glance = purrr::map(model, broom::glance)) %>%
  unnest(glance, .drop = TRUE) %>%
  ggplot(aes(continent, r.squared)) +
  geom_beeswarm()
```



3. To create the last plot (showing the data for the countries with the worst model fits), we needed two steps: we created a data frame with one row per country and then semi-joined it to the original dataset. It's possible avoid this join if we use unnest() instead of unnest(.drop = TRUE). How?

Hide

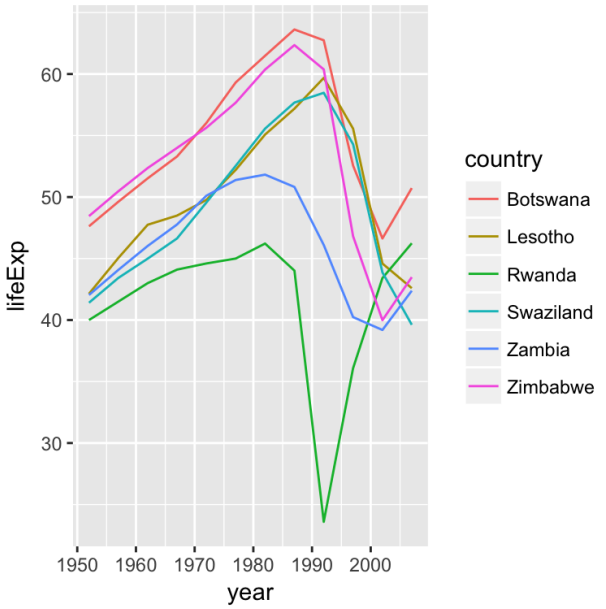
```
glance <- by_country %>%
  mutate(glance = purrr::map(model, broom::glance)) %>%
  unnest(glance)
glance
```

```
## # A tibble: 142 x 15
##        country continent              data    model r.squared
##         <fctr>    <fctr>            <list>   <list>     <dbl>
##  1 Afghanistan      Asia <tibble [12 x 4]> <S3: lm> 0.9892058
##  2     Albania    Europe <tibble [12 x 4]> <S3: lm> 0.9615523
##  3     Algeria    Africa <tibble [12 x 4]> <S3: lm> 0.9920480
##  4      Angola    Africa <tibble [12 x 4]> <S3: lm> 0.9786354
##  5   Argentina  Americas <tibble [12 x 4]> <S3: lm> 0.9955863
##  6   Australia   Oceania <tibble [12 x 4]> <S3: lm> 0.9927021
##  7     Austria    Europe <tibble [12 x 4]> <S3: lm> 0.9946153
##  8     Bahrain      Asia <tibble [12 x 4]> <S3: lm> 0.9973157
##  9  Bangladesh      Asia <tibble [12 x 4]> <S3: lm> 0.9972774
## 10     Belgium    Europe <tibble [12 x 4]> <S3: lm> 0.9957497
## # ... with 132 more rows, and 10 more variables: adj.r.squared <dbl>,
## #   sigma <dbl>, statistic <dbl>, p.value <dbl>, df <int>, logLik <dbl>,
## #   AIC <dbl>, BIC <dbl>, deviance <dbl>, df.residual <int>
```

We can give the data variable directly to ggplot

Hide

```
country_model <- function(df) {
  lm(lifeExp ~ poly(year,1), data = df)
}

by_country <- by_country %>%
  mutate(model = purrr::map(data, country_model))

glance <- by_country %>%
  mutate(glance = purrr::map(model, broom::glance)) %>%
  unnest(glance)

glance %>%
  filter(r.squared < 0.25) %>%
  select(country,data) %>%
  unnest() %>%
  ggplot() +
  geom_line(aes(year,lifeExp,color=country))
```



Interesting how when the polynomial degree is 1 then these 6 countries are below .25 r^2 but when the model is degree 2 only rwanda is there. Thus the degree 2 model allows for greater flexibility to fit those 5 countries' data. Interesting.

Exercise 25.4.5

1. List all the functions that you can think of that take a atomic vector and return a list.

from jarnold, many of the stringr functions

2. Brainstorm useful summary functions that, like quantile(), return multiple values.

range, fivenum etc.

3. What's missing in the following data frame? How does quantile() return that missing piece? Why isn't that helpful here?

Hide

```
mtcars %>%
  group_by(cyl) %>%
  summarise(q = list(quantile(mpg))) %>%
  unnest()
```

```
## # A tibble: 15 x 2
##      cyl     q
##    <dbl> <dbl>
##  1     4 21.40
##  2     4 22.80
##  3     4 26.00
##  4     4 30.40
##  5     4 33.90
##  6     6 17.80
##  7     6 18.65
##  8     6 19.70
##  9     6 21.00
## 10     6 21.40
## 11     8 10.40
## 12     8 14.40
## 13     8 15.20
## 14     8 16.25
## 15     8 19.20
```

What's missing is the list of values given from quantile(). unnest() drops the names of the quantiles so we lose those labels.

4. What does this code do? Why might might it be useful?

Hide

```
mtcars %>%
  group_by(cyl) %>%
  summarise_each(funs(list))
```

```
## # A tibble: 3 x 11
##     cyl        mpg       disp         hp       drat         wt       qsec
##   <dbl>     <list>     <list>     <list>     <list>     <list>     <list>
## 1     4 <dbl [11]> <dbl [11]> <dbl [11]> <dbl [11]> <dbl [11]> <dbl [11]>
## 2     6  <dbl [7]>  <dbl [7]>  <dbl [7]>  <dbl [7]>  <dbl [7]>  <dbl [7]>
## 3     8 <dbl [14]> <dbl [14]> <dbl [14]> <dbl [14]> <dbl [14]> <dbl [14]>
## # ... with 4 more variables: vs <list>, am <list>, gear <list>,
## #   carb <list>
```

Seems like this puts of of the non-grouped variables data together while keeping the grouped variable present but still representing the data from all the other variables. I think this might streamline analyses in parallel but not sure the advantage except for compacting data into view.

Exercise 25.5.3

1. Why might the lengths() function be useful for creating atomic vector columns from list-columns?

from jarnold,

The lengths() function gets the lengths of each element in a list. It could be useful for testing whether all elements in a list-column are the same length. You could get the maximum length to determine how many atomic vector columns to create. It is also a replacement for something like map_int(x, length) or sapply(x, length).

2. List the most common types of vector found in a data frame. What makes lists different?

Most common types would be character, integer, numeric, logical, and factor. Lists are non-atomic, meaning they are a mixture of the mentioned object types. # Part IV Communicate # 26. Introduction

No Exercises

27. R Markdown

Exercise 27.2.1

1. Create a new notebook using File > New File > R Notebook. Read the instructions. Practice running the chunks. Verify that you can modify the code, re-run it, and see modified output.

See _MyNotebook.Rmd

2. Create a new R Markdown document with File > New File > R Markdown… Knit it by clicking the appropriate button. Knit it by using the appropriate keyboard short cut. Verify that you can modify the input and see the output update

See *MyRmarkdown.Rmd*

3. Compare and contrast the R notebook and R markdown files you created above. How are the outputs similar? How are they different? How are the inputs similar? How are they different? What happens if you copy the YAML header from one to the other?

Similar output: Both produce an html-rendered file.

Different output: The notebook html ending is *.nb.html where the RMarkdown ending is *.html. Also, the notebook has a *Code* button in the top right to hide/show the code or download the Rmd.

Similar input: Same 3 components: YAML header, mixed prose and embedded code.

Different input: YAML header in the notebook has *output: html_notebook* and the RMarkdown variant is *output: html_document*.

4. Create one new R Markdown document for each of the three built-in formats: HTML, PDF and Word. Knit each of the three documents. How does the output differ? How does the input differ? (You may need to install LaTeX in order to build the PDF output — RStudio will prompt you if this is necessary.)

Similar output: Some file format with content rendered.

Different output: html, word, or pdf document.

Similar input: Same content.

Different input: In the YAML, the output was either *html_document*, *word_document*, or *pdf_document*.

Exercise 27.3.1

1. Practice what you've learned by creating a brief CV. The title should be your name, and you should include headings for (at least) education or employment. Each of the sections should include a bulleted list of jobs/degrees. Highlight the year in bold.

See *BriefCV.Rmd*

2. Using the R Markdown quick reference, figure out how to:

1. Add a footnote.
2. Add a horizontal rule.
3. Add a block quote.

See *BriefCV.Rmd*

3. Copy and paste the contents of diamond-sizes.Rmd from https://github.com/hadley/r4ds/tree/master/rmarkdown (https://github.com/hadley/r4ds/tree/master/rmarkdown) in to a local R markdown document. Check that you can run it, then add text after the frequency polygon that describes its most striking features.

See *Diamond-sizes.Rmd*

Exercise 27.4.7

1. Add a section that explores how diamond sizes vary by cut, colour, and clarity. Assume you're writing a report for someone who doesn't know R, and instead of setting echo = FALSE on each chunk, set a global option.

See section *Exercise 27.4.7 #1 description* in *Diamond-sizes.Rmd*.

2. Download diamond-sizes.Rmd from https://github.com/hadley/r4ds/tree/master/rmarkdown (https://github.com/hadley/r4ds/tree/master/rmarkdown). Add a section that describes the largest 20 diamonds, including a table that displays their most important attributes.

See section *Exercise 27.4.7 #2 description* in *Diamond-sizes.Rmd*.

3. Modify diamonds-sizes.Rmd to use comma() to produce nicely formatted output. Also include the percentage of diamonds that are larger than 2.5 carats.

I couldn't get a global commas command unfortunately...

4. Set up a network of chunks where d depends on c and b, and both b and c depend on a. Have each chunk print lubridate::now(), set cache = TRUE, then verify your understanding of caching.

See section *Exercise 27.4.7 #2 description* in *Diamond-sizes.Rmd*.

SessionInfo

Hide

```
sessionInfo()
```

```
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] microbenchmark_1.4-2.1 lubridate_1.6.0        stringr_1.2.0
##  [4] forcats_0.2.0          viridis_0.4.0          viridisLite_0.2.0
##  [7] ggbeeswarm_0.6.0       lvplot_0.2.0           ggstance_0.3
## [10] nycflights13_0.2.2     bindrcpp_0.2           maps_3.2.0
## [13] dplyr_0.7.2            purrr_0.2.3            readr_1.1.1
## [16] tidyr_0.6.3            tibble_1.3.3           ggplot2_2.2.1
## [19] tidyverse_1.1.1
##
## loaded via a namespace (and not attached):
##  [1] beeswarm_0.2.3   reshape2_1.4.2   haven_1.1.0      lattice_0.20-35
##  [5] colorspace_1.3-2 htmltools_0.3.6  yaml_2.1.14      rlang_0.1.1
##  [9] foreign_0.8-69   glue_1.1.1       modelr_0.1.1     readxl_1.0.0
## [13] bindr_0.1        plyr_1.8.4       munsell_0.4.3    gtable_0.2.0
## [17] cellranger_1.1.0 rvest_0.3.2      codetools_0.2-15 psych_1.7.5
## [21] evaluate_0.10.1  knitr_1.16       vipor_0.4.5      parallel_3.4.1
## [25] broom_0.4.2      Rcpp_0.12.12     scales_0.4.1     backports_1.1.0
## [29] jsonlite_1.5     gridExtra_2.2.1  mnormt_1.5-5     hms_0.3
## [33] digest_0.6.12    stringi_1.1.5    grid_3.4.1       rprojroot_1.2
## [37] tools_3.4.1      magrittr_1.5     lazyeval_0.2.0   pkgconfig_2.0.1
## [41] xml2_1.1.1       assertthat_0.2.0 rmarkdown_1.6    httr_1.2.1
## [45] R6_2.2.2         nlme_3.1-131     compiler_3.4.1
```