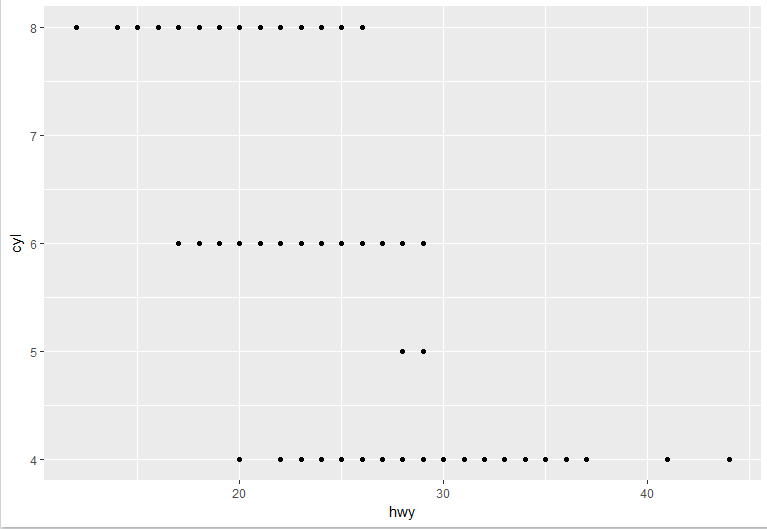
R for Data Science: Introduction and Explore

**3.2.4**

1. Nothing
2. 234 rows, 11 columns (*variables*)
3. f = front-wheel drive, r = rear wheel drive, 4 = 4wd
4. 
5. This scatterplot is not helping answer any useful questions. Knowing the class of car and the type of drive functionality it is is not useful

**3.3.1**

1. The color = “blue” needs to go outside of the aes() for the whole plot to turn blue
2. Categorical = model name, cyl, trans, drv, fl, class 🡪 Continuous = displ, year, cty, hwy
3. Using a discrete variable can sometimes make it useless and challenging to apply aesthetics to
4. Both aesthetics will be used and they will each have their own specific legend
5. **Stroke:** controls the border width of the shapes being displayed. I think it works with all shapes, but it’s quite useful for circles
6. It will create a true/false situation where the data points will receive shading based off the statement created about displ

**3.5.1**

1. R will create a separate facet for each unique continuous variable. It just doesn’t work.
2. It means there is no combination of those values. In this example, there are no 4-wheel drive cars with a 7 cylinder engine.
3. A (.) acts as a holder for a variable. Therefore it plots that data on one dimension rather that the two variables separated by a ~
4. Faceting can help you see all the variables you’re measuring in a separate graph without all of the clutter and different colors clumping in one graph.
5. **nrow:** determines how many rows the graph will have. **ncol:** determines the # of columns a grid will have. **as.table:** if true, the default, the facets are laid out like a table with the highest values at the bottom-right. If false, the facets are laid out like a plot with the highest value at the top right.
6. The variable w/ more unique levels in the columns will help you see the graph in a more extended way.

**3.6.1**

1. Different variations of geoms
   1. Line chart 🡪 geom\_line()
   2. Boxplot 🡪 geom\_boxplot()
   3. Histogram 🡪 geom\_histogram()
   4. Area chart 🡪 geom\_area()
2. N/A
3. **show.legend = FALSE** removes the legend. I don’t see the true point of this
4. **se** argument in geom\_smooth(): shows whether or not to create a confidence interval
5. No they will show the same things. The mapping is in different places, but the function is the same
6. Creating plot examples

**3.7.1**

1. Default geom: geom\_pointrange()
2. Geom\_bar() uses a count stat and the geom\_col() is already transformed
3. Almost all of the geoms and stats are paired. Too many to list.
4. Variables that stat\_smooth() computes
   1. **y**: predicted value
   2. **ymin**: lower pointwise confidence interval around the mean
   3. **ymax**: upper pointwise confidence interval around the mean
   4. **se**: standard error, determines whether confidence interval should be plotted
      1. **method:** controls the smoothing method employed
      2. **level:** determines the level of confidence interval to use
5. Unless you set the proportions of a bar chart to **group=1**, then it will utilize the entire dataset, rather than the variable chosen

**3.8.1**

1. There’s loads of overlapping points, apply jitter to add dat noise
2. **Width** and **height** control the amount of jitter
3. When there is overlapping of data points, geo\_count() counts the number and makes larger points rather than just having a lot of indistinguishable noise
4. Default positioning for a **geo\_boxplot()** is a **position\_dodge()**

**3.9.1**

1. Turning a stacked bar chart into a pie chart using coord\_polar()
   1. ggplot(data = mpg, mapping = aes( x = factor(1), fill = class)) + geom\_bar(width=1) + coord\_polar(theta = “y”)
2. **labs()**: ensures that the axis and legend labels display the full variable name
3. **coord\_map()** helps show a 3-D object onto a 2-D surface 🡪 like the map of New Zealand. **coord\_quickmap()** preserves straight lines and is less accurate. Hence the “quickmap”
4. There’s various learning points from this exercises:
   1. **coord\_fixed():** draws equals intervals on the x and y axes so they directly comparable
   2. **geom\_abline():** draws a line that has an intercept of 0 and slope of 1

**4.4**

1. There is a 1 instead of an “i” so the call breaks
2. Change the o into an a. The second one is fine
3. **Alt + Shift + K:** this will pop a list of all of the keyboard shortcuts! You can also go to Help 🡪 Keyboard Shortcuts Help

**5.2.4**

1. Finding flights
   1. filter(flights, arr\_delay >= 120)
   2. filter(flights, dest == “IAH” | dest == “HOU”)
   3. filter(flights, carrier == “UA” | carrier == “AA” | carrier == “DL”)
   4. filter(flights, month >= 7, month <= 9)
   5. filter(flights, arr\_delay >= 120, dep\_delay <= 0)
   6. filter(flights, dep\_delay >= 60, dep\_delay – arr\_delay >= 30)
   7. filter(flights, dep\_time >= 0, dep\_time <= 600)
2. **between()**: finding observations between two values or dates
3. filter(flights, **is.na**(dep\_time))
4. NA functions
   1. NA ^ 0: anything to the 0 power is 1
   2. NA | TRUE: if one condition is true they are both true
   3. FALSE & NA: NA means the absence of a value so a conditional expression ignores it
   4. In conditional expressions, missing values are ignored

**5.3.1**

1. arrange(flights, !is.na(dep\_time))
2. arrange(flights, desc(arr\_delay)), arrange(flights, dep\_delay)
3. arrange(flights, desc(distance/air\_time))
4. arrange(flights, desc(distance)), arrange(flights, distance)

**5.4.1**

1. Done
2. It’s only included one time
3. **one\_of():** selects any variable which matches one of the strings in the vector
4. The select function by default ignores case

**5.5.2**

1. Transmute(flights, sched\_dep\_time = (sched\_dep\_time %/% 100) \* 60 + sched\_dep\_time %% 100, dep\_time = (dep\_time %/% 100) \* 60 + dep\_time %% 100)
2. Flightsx <- select(flights, air\_time, arr\_time, dep\_time) 🡪 mutate(flightsx, air\_time\_new = arr\_time – dep\_time)
   1. You need to convert the dep\_time and arr\_time to continuous because right now they are numerical digits
3. Dep\_time is the summation of scheduled departure time and the departure delay
4. Delayed <- mutate(flights, most\_delayed = min\_rank(desc(arr\_delay))) 🡪 arrange(delayed, most\_delayed)
5. The two vector variables are not the same length so R shows the shorter of the two
6. All of them: cosine, sine, arc-tangent, arc-sine

**5.6.7**

1. Brainstorming ways to assess typical delay characteristics
   1. flights %>%  
      group\_by(flight) %>%  
      summarize(early\_15\_min = sum(arr\_delay <= -15, na.rm = TRUE)/n(),  
      late\_15\_min = sum(arr\_delay >= 15, na.rm = TRUE/n()) %>%  
      filter(early\_15\_min == 0.5,  
       late\_15\_min == 0.5)
   2. flights %>%  
      group\_by(flight) %>%  
      summarize(late\_10\_min = sum(arr\_delay = 10, na.rm == TRUE)/n()) %>%  
      filter(late\_10\_min == 1)
   3. flights %>%  
      group\_by(flight) %>%  
      summarize(early\_30\_min = sum(arr\_delay <= -30, na.rm = TRUE)/n(), late\_30\_min = sum(arr\_delay >= 30, na.rm = TRUE)/n()) %>%  
      filter(early\_30\_min == 0.5,  
      late\_30\_min == 0.5)
   4. flights %>%  
      group\_by(flight) %>%  
      summarize(on\_time = sum(arr\_delay == 0, na.rm = TRUE)/n(), late\_2\_hours = sum(arr\_delay == 120, na.rm = TRUE)/n()) %>%  
      filter(on\_time == 0.99,  
      late\_2\_hours == 0.01)
2. not\_cancelled <- flights %>%  
   filter(!is.na(dep\_delay), !is.na(arr\_delay))
   1. not\_cancelled %>%  
      group\_by(dest) %>%  
      summarize(n = n())
   2. not\_cancelled %>%  
      group\_by(talinum) %>%  
      summarize(n = sum(distance, na.rm = TRUE))
3. It’s quite counterintuitive because if a flight didn’t depart then it obviously won’t have an arrival delay
4. flights %>%  
   group\_by(day) %>%  
   summarize(percent\_cancel = sum(is.na(dep\_delay))/n(),  
   avg\_delay = mean(dep\_delay, na.rm = TRUE))
5. Worst Delays by Carrier  
   flights %>%  
   group\_by(carrier) %>%  
   summarize(avg\_delay = mean(arr\_delay, na.rm = TRUE)) %>%  
   arrange(desc(avg\_delay))
6. **Sort** will sort the results of **count()** in descending order of n.

**5.7.1**

1. NA
2. On-time is less than 15 minutes of arrival delay:  
   flights %>%  
   group\_by(talinum) %>%  
   summarize(per\_on\_time = sum(arr\_delay <= 15, na.rm = TRUE)/n(),   
   mean\_arr\_delay = mean(arr\_delay, na.rm = TRUE),  
   flights = n()) %>%  
   arrange(per\_on\_time, desc(mean\_arr\_delay))
3. flights %>%  
   group\_by(hour) %>%  
   summarize(avg\_delay = sum(arr\_delay >= 15, na.rm = TRUE)/n()) %>%  
   ggplot(aes(x = hour, y = avg\_delay)) +  
   geom\_col()
4. Total delay by destination:  
   flights %>%  
   group\_by(dest) %>%  
   filter(!is.na(dep\_delay)) %>%  
   summarize(total\_min = sum(dep\_delay(dep\_delay > 0)))  
   Proportion of flight delay:  
   flights %>%  
   filter(!is.na(dep\_delay)) %>%  
   group\_by(talinum, dest) %>%  
   summarize(prop\_delay = mean(dep\_delay > 0)) %>%  
   arrange(desc(prop\_delay))
5. flights %>%  
   group\_by(origin) %>%  
   arrange(year, month, day, hour, minute) %>%  
   mutate(before\_delay = lag(dep\_delay)) %>%  
   ggplot(aes(x = before\_delay, y = dep\_delay) +  
   geom\_point() +  
   geom\_smooth()
6. Suspiciously fast flights:  
   flights %>%  
   group\_by(dest) %>%  
   arrange(air\_time) %>%  
   Flight time relative to that destination:  
   flights %>%  
   group\_by(dest) %>%  
   mutate(shortest\_flight = air\_time – min(air\_time, na.rm = TRUE)) %>%
7. flights %>%  
   group\_by(dest) %>%  
   filter(n\_distinct(carrier) > 2) %>%  
   group\_by(carrier) %>%  
   summarize(x = n\_distinct(dest)) %>%  
   arrange(desc(x)0
8. flights %>%  
   mutate(dep\_date = time\_hour) %>%  
   group\_by(talinum) %>%  
   arrange(dep\_date)

**6.3**

1. You can run SQL directly in an Rstudio Notebook chunk. This could be incredibly useful for when I start building out CTR curves with Search console data
2. Warn if a variable is defined but not used: can be helpful when you want to clean up old code or when you’re trying to figure out other errors

**7.3.4**

1. gpglot(diamonds, aes(x,y, or z)) + geom\_histogram()
   1. It looks like two of the dimensions have the same distribution
2. It looks like there are a much lower number of diamonds priced around $2K in relation to all the other price points. Using the small bin-size allowed me to split the data apart a bit to see it
3. ggplot(diamonds, aes(carat)) +  
   geom(histogram(binwidth = .01) +  
   coord\_cartesian(xlim = c(.95, 1.05)
   1. It looks like no one is interested in buying diamonds less than 1 karat
4. Using xlim() or ylim() removes all the data points which don’t fit into the boundaries created so then the smooth line will be different than coord\_cartesian() which just zooms in.

**7.4.1**

1. The histogram takes out the missing values and bar charts draw them in a separate category.
2. It takes out missing values before you run the mean() or sum() function.

**7.5.1.1**

1. Change the y variable to density rather than count because the non-cancelled flights are skewing the data and not allowing me to see the cancelled flights
2. ggplot(diamonds, aes(carat, price)) +  
   geom\_point() +  
   geom\_smooth()
3. NA
4. Geom\_lv(): conveys more info in the tails using letter values, only out to the depths where the letter values are reliable estimate. Everything shown in the letter-value boxplot is an actual observation
5. NA
6. Geom\_jitter() is handy to see the relationship bet/ continuous and categorical variables

**7.5.2.1**

1. You can calculate a new variable that is the proportion of each cut within a color because then we can style each individual tile with a different shade
2. We could sort destinations and remove missing values so that there isn’t blank space and then sorting it will make it more viewable
3. I feel like it’s better to have the categorical variable on the y-axis because it will have text as the label and is easier to read that way.

**7.5.3.1**

1. Cut\_width() doesn’t take into account the number of observations in each bin. There may be too many or too few data points skewing the visual
2. ggplot(diamonds, aes(x = cut\_number(price, 50), y = carat)) +  
   geom\_boxplot() +  
   coordflip()
3. I would say that the large the diamonds, the more price variation just due to overall quality and other variables starting to come into play
4. Ggplot(diamonds, aes(x = cut\_number(cut, 5), y = carat, color = price)) +  
   geom\_boxplot()
5. It’s much easier to see the relationship of the data points