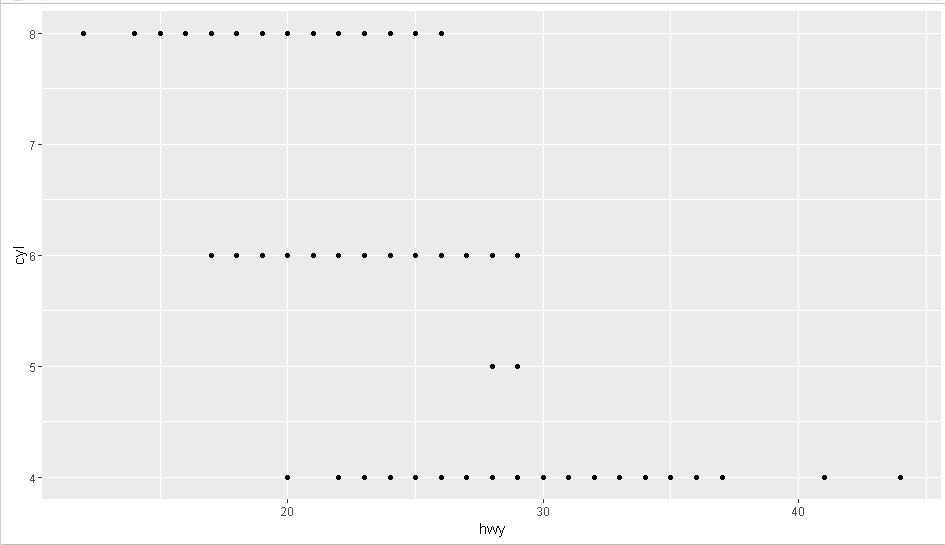
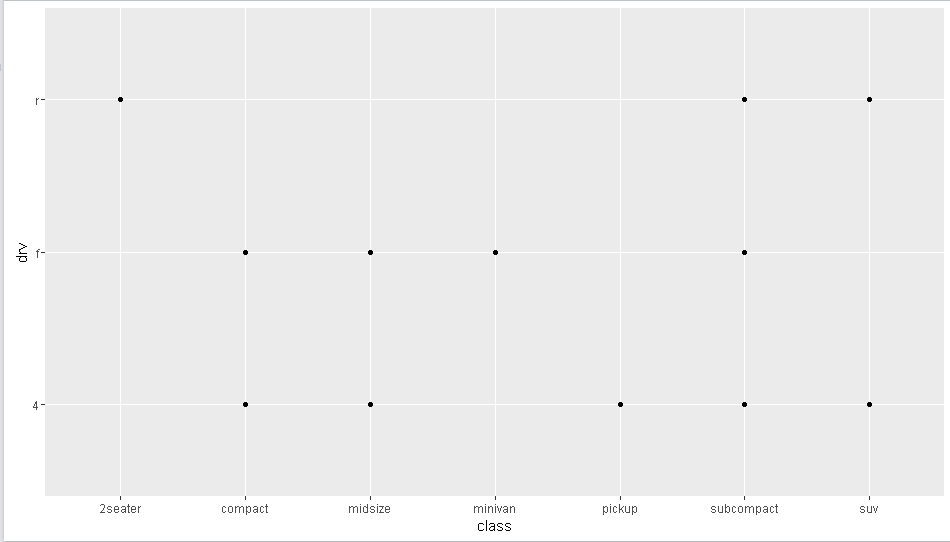
**3.2.4**

1. Run ggplot(data = mpg). What do you see?
   1. Nothing because it’s an empty graph
2. How many rows are in mpg? How many columns?
   1. 234 rows and 11 columns
3. What does the drv variable describe? Read the help for ?mpg to find out.
   1. The type of drive – front wheel, rear wheel, and 4wd.,
4. Make a scatterplot of hwy vs cyl.



1. What happens if you make a scatterplot of class vs drv? Why is the plot not useful?
   1. This happens. They’re both categorical variables, so it doesn’t make sense to use a scatterplot.



**3.3.1**

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

**ggplot**(data = mpg) +

**geom\_point**(mapping = **aes**(x = displ, y = hwy, color = "blue"))

1. To force a variable like that for the whole graph, there need to be parentheses around the x & y variables, with the other variable outside. It should be geom\_point(mapping = aes(x=displ, y=hwy),color=”blue”)
2. Which variables in mpg are categorical? Which variables are continuous? (Hint: type ?mpg to read the documentation for the dataset). How can you see this information when you run mpg?
   1. Categorical: manufacturer, model, year, cyl, trans, drv, fl, class.  
      Variable: displ, cty, hwy  
      The information shows in the Help tab in the viewer.
3. Map a continuous variable to color, size, and shape. How do these aesthetics behave differently for categorical vs. continuous variables?
   1. Color shows as distinct colors for categorical variables, and a gradient of one color for continuous.
   2. Shape can’t be mapped to a continuous variable, while it can to a categorical variable.
   3. Size isn’t advised to be used with categorical variables since it’s not meaningful, while it works fine with continuous variables.
4. What happens if you map the same variable to multiple aesthetics?
   1. You get two separate legends, with the points as combinations of the two aesthetics.
5. What does the stroke aesthetic do? What shapes does it work with? (Hint: use ?geom\_point)
   1. Stroke modifies the width of the border.
6. What happens if you map an aesthetic to something other than a variable name, like aes(colour = displ < 5)?
   1. The legend is two colors to represent True/False based on the logical expression in the aesthetic.

**3.5.1**

1. What happens if you facet on a continuous variable?
   1. You get a new cell for every distinct value of the continuous variable.
2. What do the empty cells in plot with facet\_grid(drv ~ cyl) mean? How do they relate to this plot?

**ggplot**(data = mpg) +

**geom\_point**(mapping = **aes**(x = drv, y = cyl))

* 1. It means there are no data points with that combination of drive type and cylinders. The blank intersections in that plot are the same combinations that are blank cells in the facet\_grid(drv~cyl).

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

**ggplot**(data = mpg) +

**geom\_point**(mapping = **aes**(x = displ, y = hwy)) +

**facet\_grid**(drv ~ .)

1. This plot has three rows of data, separated out by drive type.

**ggplot**(data = mpg) +

**geom\_point**(mapping = **aes**(x = displ, y = hwy)) +

**facet\_grid**(. ~ cyl)

1. This plot has four columns of data, separated out by number of cylinders.
2. . means that whichever side it’s on, x or y, won’t be faceted by another variable.
3. Take the first faceted plot in this section:

**ggplot**(data = mpg) +

**geom\_point**(mapping = **aes**(x = displ, y = hwy)) +

**facet\_wrap**(~ class, nrow = 2)

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?

1. You are able to see trends comparing like with like in the same plot, instead of seeing the data points intermingled with other types of categories. Using faceting makes it easier to see trends within a specific subset. However, you lose the ability to easily see how the subsets fit within the larger picture of the full data set. With a larger data set, I imagine it would be best to use faceting instead of the color aesthetic because too many overlapping or close data points would make it difficult to discern the trends by color
2. 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?
   1. Nrow and ncol tell how many rows and columns to display the data.
   2. Scales determines if the x & y scales should be fixed or free by cell. Switch determines where the location of the labels is shown. Drop determines if levels of the variable not used in the dataset are dropped or displayed. Labeller determines how the values are printed in the cells.
   3. Facet grid is forced into a certain number of rows and columns depending on how many levels there are of the variable(s) so you don’t need the nrow or ncol argument.
3. When using facet\_grid() you should usually put the variable with more unique levels in the columns. Why?
   1. By default there is more horizontal space than vertical space in the viewer, so there is more room for more columns. Additionally, maintaining as much vertical space in the plot itself generally makes it easier to see trends.

**3.6.1**

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

**ggplot**(data = mpg, mapping = **aes**(x = displ, y = hwy, color = drv)) +

**geom\_point**() +

**geom\_smooth**(se = FALSE)

* 1. I did this

1. What does show.legend = FALSE do? What happens if you remove it?  
   Why do you think I used it earlier in the chapter?
   1. It removes the legend from the visualization. My guess is so that the chart where the legend would have been shown by default looked the same as the other two charts that didn’t show a legend by default.
2. What does the se argument to geom\_smooth() do?
   1. It shows the confidence interval around the smooth line.
3. Will these two graphs look different? Why/why not?

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

**geom\_point**() +

**geom\_smooth**()

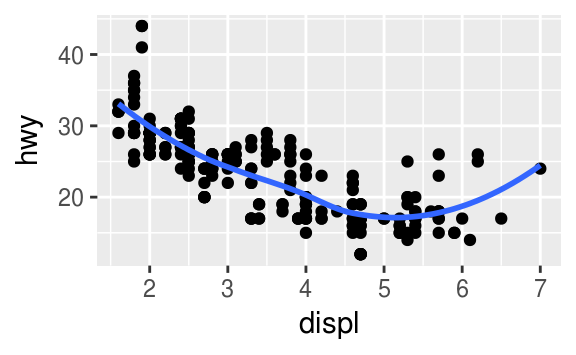
**ggplot**() +

**geom\_point**(data = mpg, mapping = **aes**(x = displ, y = hwy)) +

**geom\_smooth**(data = mpg, mapping = **aes**(x = displ, y = hwy))

* 1. No. The arguments are the same for both geoms when defined in each geom, so defining them up top will create the same visual.

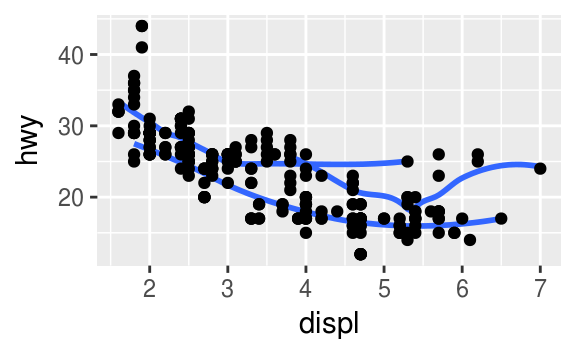
1. Recreate the R code necessary to generate the following graphs.



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

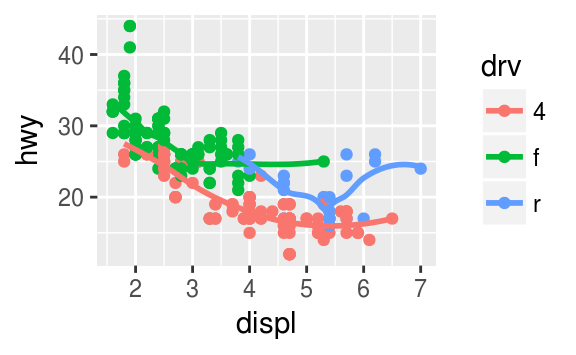
geom\_point() +

geom\_smooth(show.legend = FALSE)



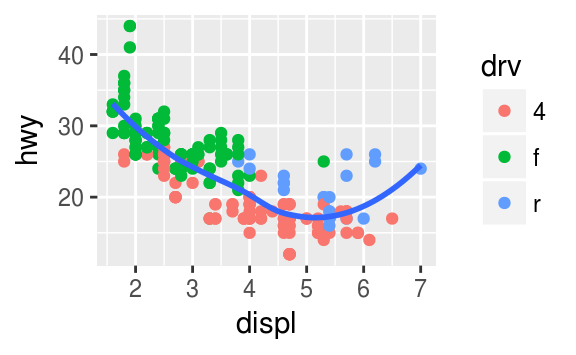
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, group = drv)) +

geom\_point() +

geom\_smooth(se = FALSE)

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

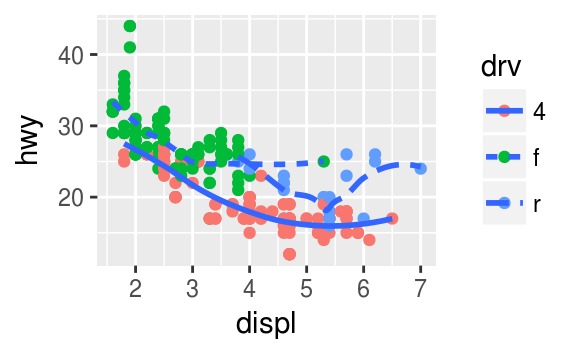
geom\_point() +

geom\_smooth(se = FALSE)

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

geom\_point(mapping = aes(color = drv)) +

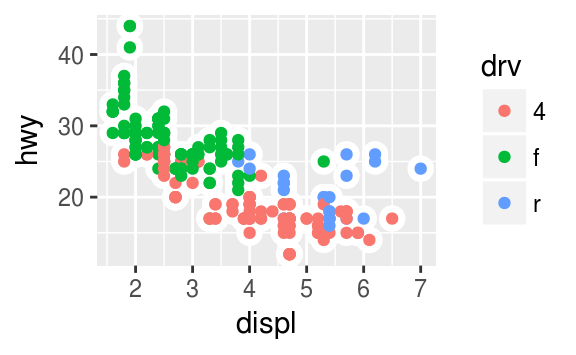
geom\_smooth(se = FALSE)



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

geom\_point(mapping = aes(color = drv)) +

geom\_smooth(mapping = aes(linetype = drv),se = FALSE)



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

geom\_point(mapping = aes(fill = drv), shape = 21, color = "white")

**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?

Geom\_pointrange

ggplot(data = diamonds) +

geom\_pointrange(mapping = aes(x = cut, y = depth),

stat = "summary",

fun.ymin = min,

fun.ymax = max,

fun.y = median)

1. What does geom\_col() do? How is it different to geom\_bar()?
   1. Geom\_col() makes a bar chart but the default stat is stat\_identity instead of stat\_count, so it uses the y value instead of counting the number of observations.
2. 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?
   1. Geom\_point > stat = identity
   2. Geom\_bar > stat = count
   3. Geom\_col > stat = identity
   4. Geom\_smooth > stat = smooth
   5. Geom\_histogram > stat = bin
   6. Geom\_freqpoly > stat = bin
   7. They’re all either utilizing a count of observations, the actual identity of the data value (summed or in place as in a scatterplot) or binning observations and counting observations in that grouping.
3. What variables does stat\_smooth() compute? What parameters control its behaviour?
   1. Stat\_smooth() computes the predicted value y, the lower confidence interval ymin, the upper confidence interval ymax, and the standard error se.
   2. The parameters controlling the behavior are n which tells the number of points to evaluate, span which controls the amount of smoothing, fullrange which controls if the smooth should fit the full plot or the data, and level which determines the confidence interval.
4. In our proportion bar chart, we need to set group = 1. Why? In other words what is the problem with these two graphs?

**ggplot**(data = diamonds) +

**geom\_bar**(mapping = **aes**(x = cut, y = ..prop..))

**ggplot**(data = diamonds) +

**geom\_bar**(mapping = **aes**(x = cut, fill = color, y = ..prop..))

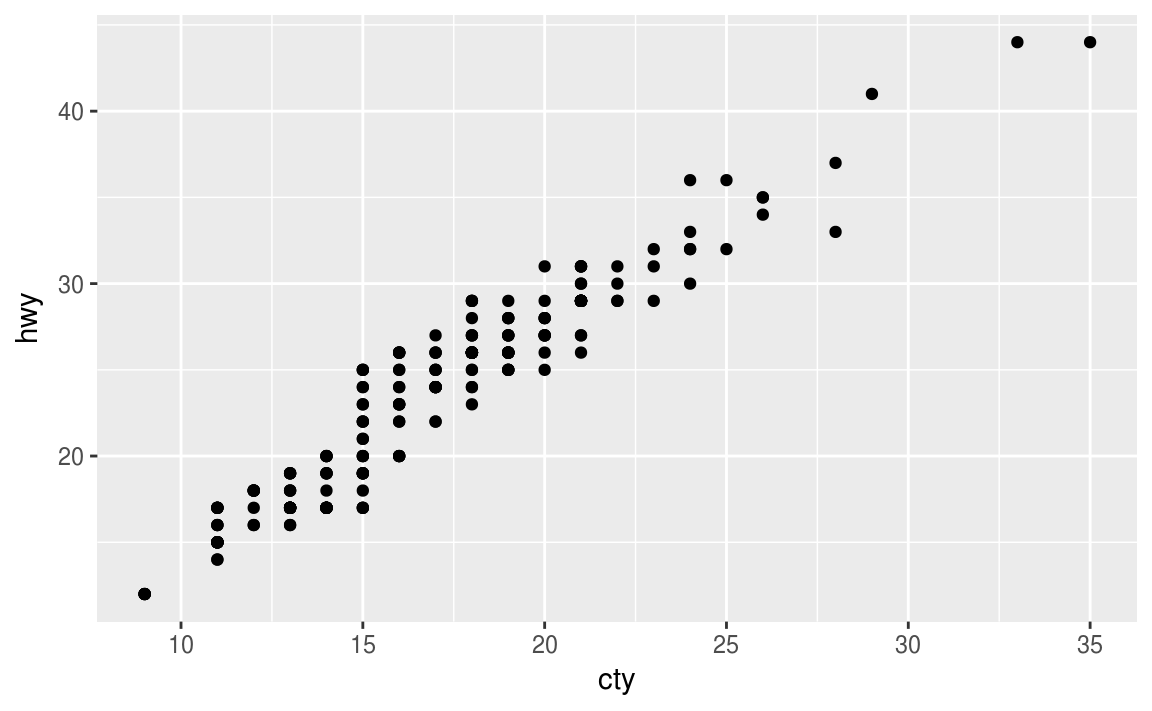
* 1. Running these graphs without the group puts all proportions at 100%. By default, it’s grouping by the x variable, so of course each group has 100% of its own values. The group = 1 overrides that default and tells it to look at the proportion by group of the whole set.

**3.8.1**

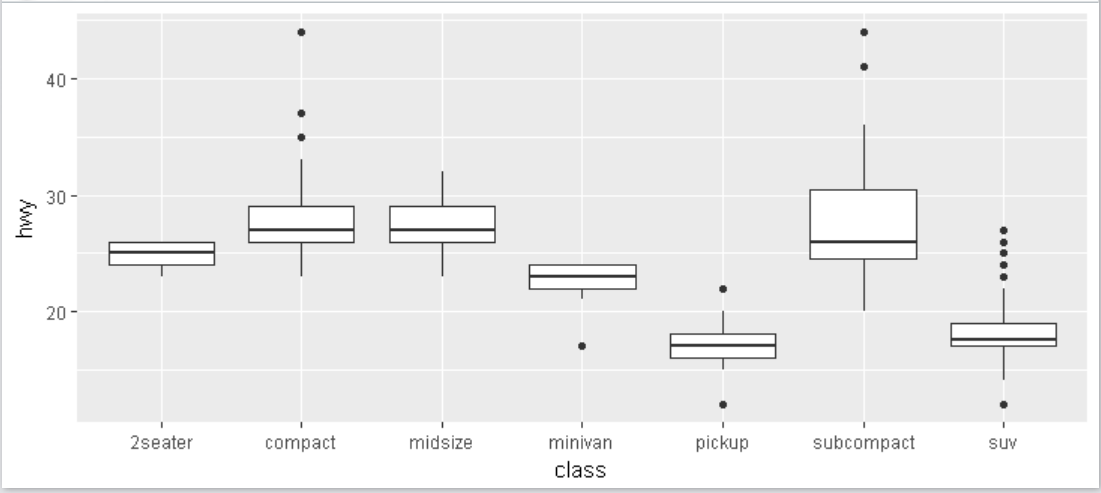
1. What is the problem with this plot? How could you improve it?

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

**geom\_point**()



* 1. It has a lot of overlapping points plotted. Using position = “jitter” will allow you to better see the data in the full data set using random noise.

1. What parameters to geom\_jitter() control the amount of jittering?
   1. Width and height control the amount of jittering.
2. Compare and contrast geom\_jitter() with geom\_count().
   1. Geom\_count() is another way to deal with overplotting, but instead of jittering the points so that you see more points, it increases the size of the point to represent the number of data points plotted at that location. Both allow you to better understand and view the full dataset, but geom\_jitter() works better to see overall trends in the data with more data points, while geom\_count() works better to see where the highest concentration of overplotting is within your data set.
3. What’s the default position adjustment for geom\_boxplot()? Create a visualisation of the mpgdataset that demonstrates it.
   1. The default position adjustment is dodge.
   2. 
   3. ggplot(data = mpg) +
   4. geom\_boxplot(mapping = aes(x = class, y = hwy),position="dodge")

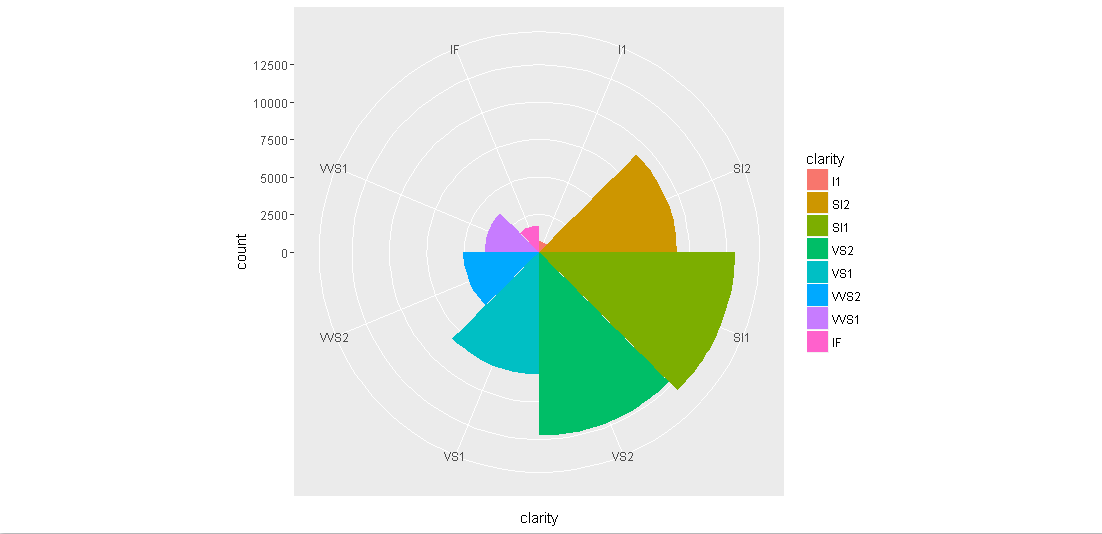
**3.9.1**

1. Turn a stacked bar chart into a pie chart using coord\_polar().

ggplot(data = diamonds) +

geom\_bar(mapping = aes(x = cut, fill = cut)) +

coord\_polar()



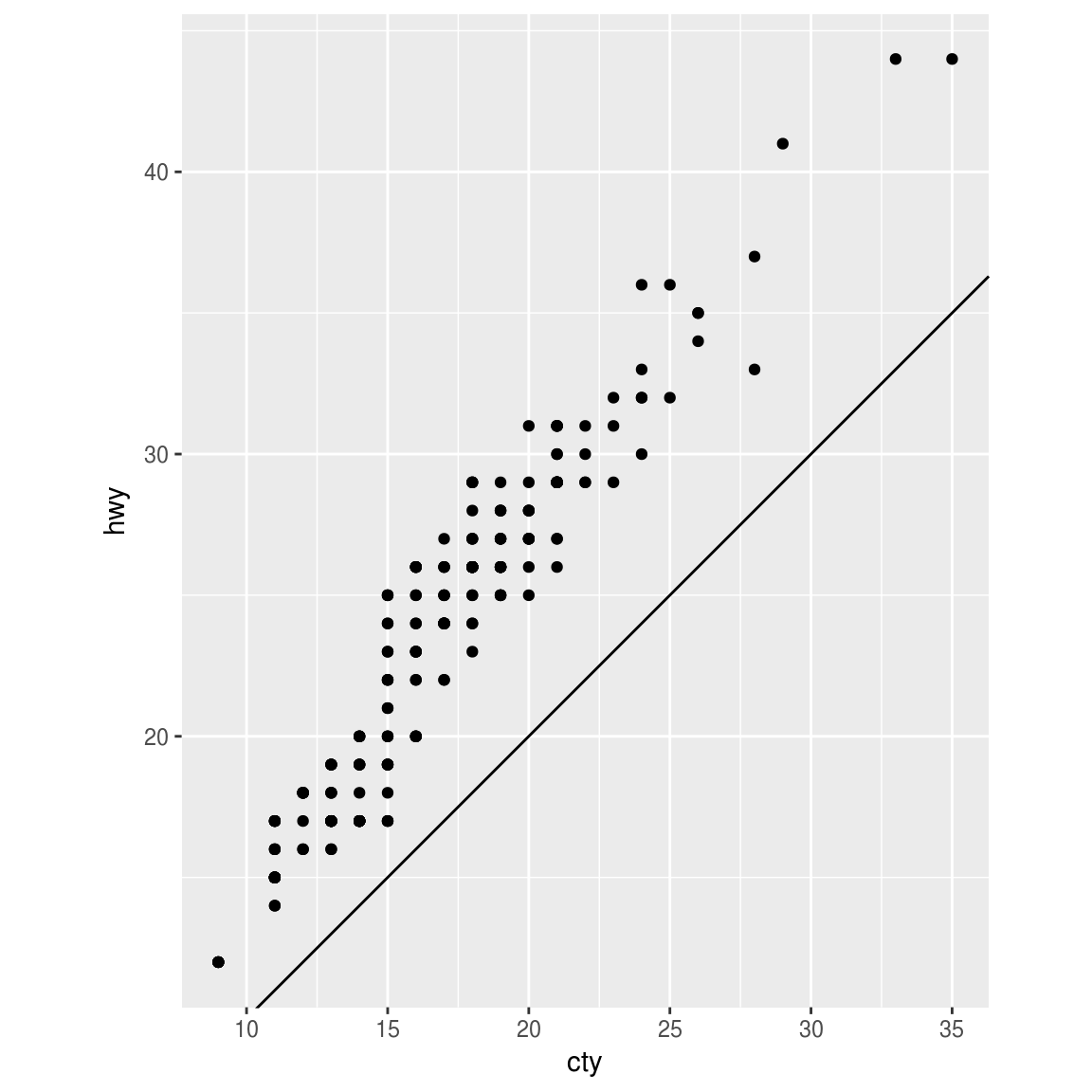
1. What does labs() do? Read the documentation.
   1. Labs() can change the labels of the x and y axis, plot title, and legend name. The NULL in the examples removed the x and y axis labels.
2. What’s the difference between coord\_quickmap() and coord\_map()?
   1. Coord\_map() projects a map area onto a 2d plane without preserving straight lines. Coord\_quickmap() is a less computationally expensive way to do the same thing while preserving straight lines and works best for smaller areas closer to the equator.
3. 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?

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

**geom\_point**() +

**geom\_abline**() +

**coord\_fixed**()



* 1. This plot tells me that there’s a strong relationship between city mileage and highway mileage, which makes sense.
  2. Coord\_fixed() forces a 1:1 ratio between units on the x axis and units on the y axis. It’s important because it more accurately shows the slope of the relationship between variables.
  3. Geom\_abline() adds a reference line to the data that by default is based on the data given, but you can also set the intercept and slope which will override the default.

**4.4**

1. Why does this code not work?

my\_variable <- 10

my\_varıable

1. The “i” in the statement trying to call the variable isn’t actually a lower case I.

*#> Error in eval(expr, envir, enclos): object 'my\_varıable' not found*

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

**library**(tidyverse)

* 1. library(**“**tidyverse**”**)

**ggplot**(dota = mpg) +

**geom\_point**(mapping = **aes**(x = displ, y = hwy))

* 1. ggplot(d**a**ta = mpg) +

geom\_point(mapping = aes(x = displ, y = hwy))

**fliter**(mpg, cyl = 8)

**filter**(diamond, carat > 3)

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

filter(diamond**s**, carat > 3)

1. Press Alt + Shift + K. What happens? How can you get to the same place using the menus?
   1. You see the shortcut command menu.
   2. Tools > Keyboard Shortcuts Help

**5.2.4**

1. Find all flights that
   1. Had an arrival delay of two or more hours
      1. filter(flights, arr\_delay >= 120)
   2. Flew to Houston (IAH or HOU)
      1. filter(flights, dest %in% c("IAH","HOU"))
   3. Were operated by United, American, or Delta
      1. filter(flights, carrier %in% c("UA","AA","DL"))
   4. Departed in summer (July, August, and September)
      1. filter(flights, month %in% c(7,8,9))
   5. Arrived more than two hours late, but didn’t leave late
      1. filter(flights, arr\_delay > 120, dep\_delay < 0)
   6. Were delayed by at least an hour, but made up over 30 minutes in flight
      1. filter(flights, dep\_delay >= 60, dep\_delay - arr\_delay > 30)
   7. Departed between midnight and 6am (inclusive)
      1. filter(flights, dep\_time >= 0, dep\_time <=600)
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?
   * 1. It allows you to give left and right boundaries instead of saying x >= and y <=.
     2. This gives the same answer as #7 in the previous question: filter(flights,between(dep\_time,0,600))
3. How many flights have a missing dep\_time? What other variables are missing? What might these rows represent?
   1. 8,255 flights have a missing dep\_time. filter(flights, is.na(dep\_time))
   2. Other missing variables are: dep\_delay, arr\_time, arr\_delay, talinum, air\_time. These rows are all dependent on a plane taking off, so if the departure time is null then the plane didn’t arrive, didn’t have airtime, didn’t delay, etc.
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!)
   1. Anything ^0 = 1, so it’s TRUE
   2. It’s an “or” evaluation, so TRUE = TRUE even if NA != TRUE.
   3. Since NA itself is missing, it’s basically ignored by the statement, so it’s just FALSE.
   4. Anything \* NA = NA. For conditional statements, NA is ignored.

**5.3.1**

1. How could you use arrange() to sort all missing values to the start? (Hint: use is.na()).
   1. arrange(flights, desc(is.na(dep\_delay)))
2. Sort flights to find the most delayed flights. Find the flights that left earliest.
   1. arrange(flights, desc(dep\_delay))
   2. arrange(flights, dep\_delay)
3. Sort flights to find the fastest flights.
   1. arrange(flights, air\_time)
4. Which flights travelled the longest? Which travelled the shortest?
   1. arrange(flights, desc(distance))
   2. arrange(flights, distance)

**5.4.1**

1. Brainstorm as many ways as possible to select dep\_time, dep\_delay, arr\_time, and arr\_delayfrom flights.
   1. select(flights, dep\_time, dep\_delay, arr\_time, arr\_delay)
   2. select(flights, starts\_with("dep"),starts\_with("arr"))
   3. select(flights,dep\_time:arr\_delay, -sched\_dep\_time, -sched\_arr\_time)
   4. select(flights,ends\_with("time"),ends\_with("delay"),-sched\_arr\_time, -sched\_dep\_time, -air\_time)
2. What happens if you include the name of a variable multiple times in a select() call?
   1. You just get the column once.
3. What does the one\_of() function do? Why might it be helpful in conjunction with this vector?

vars <- **c**("year", "month", "day", "dep\_delay", "arr\_delay")

1. You can use one\_of() for a string of characters within a variable. With this vector, you could use select(flights, one\_of(vars)) to select specifically the columns in the vector.
2. 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?

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

1. The default is set to ignore.case = TRUE. You can override it within the function.
2. This returns no columns: select(flights, contains("TIME", ignore.case = FALSE))

**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.
   1. transmute(flights,

dep\_time,

dep\_time\_mins = (dep\_time %/% 100)\*60 + dep\_time %% 100,

sched\_dep\_time,

sched\_dep\_time\_mins = (sched\_dep\_time %/% 100)\*60 + sched\_dep\_time %% 100)

1. 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?
   1. Conceptually I’d expect to see them matching. They don’t.
   2. air\_time is the actual minutes in air, while arr\_time – dep\_time is looking at the actual time of departure and arrival. So the subtraction doesn’t actually give an accurate time.
   3. Need to use mutate to change arr\_time and dep\_time into minutes from midnight, then create a new column to subtract the two (the numbers don’t actually match up though…)
2. Compare dep\_time, sched\_dep\_time, and dep\_delay. How would you expect those three numbers to be related?
   1. dep\_delay is dep\_time – sched\_dep\_time.
3. Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for min\_rank().

arrange(transmute(flights,

dep\_delay,

most\_delayed = (min\_rank(desc(dep\_delay)))),most\_delayed)

1. min\_rank() handles ties by giving each tie the same ranking, and then giving the next one the next ranking after taking into account the ties (i.e. min\_rank(1, 3, 3, 5, 7) > 1, 2, 2, 4, 5). It’s the same as sports ranking.
2. What does 1:3 + 1:10 return? Why?
3. longer object length is not a multiple of shorter object length
4. R “recycles” vectors, and if you try to compare two of different lengths, R will automatically “recycle” the shorter one over and over again until it matches the length of the longer one. If it doesn’t cleanly divide into the longer one, you will get this error. You can successfully do 1:2 + 1:10 or 1:5 + 1:10 because 2 and 5 divide into 10.
5. What trigonometric functions does R provide?
   1. cos(x), sin(x), tan(x), acos(x), asin(x), atan(x), atan2(y, x), cospi(x), sinpi(x), tanpi(x)

**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:
   1. A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.
   2. A flight is always 10 minutes late.
   3. A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.
   4. 99% of the time a flight is on time. 1% of the time it’s 2 hours late.

Which is more important: arrival delay or departure delay?

1. Overall, I would say that arrival delay is more important, because planes are able to make up time in the sky. Looking at the average difference between arrival delay and dep delay, 60+% of flights are able to make up time in the sky up to a 30 minute delay.
2. Here’s a view of the different delay characteristics I thought of:

not\_cancelled %>%

group\_by(dest) %>%

summarize(n = n(),

arrdelay = mean(arr\_delay, na.rm = TRUE),

depdelay = mean(dep\_delay, na.rm = TRUE),

arrdelay15 = mean(arr\_delay > 15, na.rm = TRUE),

depdelay15 = mean(dep\_delay >15, na.rm =TRUE),

arrdelay30 = mean(arr\_delay >30, na.rm = TRUE),

depdelay30 = mean(dep\_delay >30, na.rm = TRUE),

depdelaysub10 = mean(dep\_delay < 10, na.rm = TRUE),

arrdelaysub10 = mean(arr\_delay <10, na.rm = TRUE),

depdelay60plus = mean(dep\_delay > 60, na.rm = TRUE),

arrdelay60plus = mean(arr\_delay > 60, na.rm = TRUE),

depdelay120plus = mean(dep\_delay >120, na.rm = TRUE),

arrdelay120plus = mean(arr\_delay >120, na.rm = TRUE),

timemadeup = mean(arr\_delay - dep\_delay < 0)) %>%

ggplot(mapping = aes(x = depdelay30, y = timemadeup)) +

geom\_smooth()

1. 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()).

not\_cancelled %>%

group\_by(dest) %>%

summarize(n = sum(!is.na(dest)))

not\_cancelled %>%

group\_by(tailnum) %>%

summarize(n = sum(distance))

1. Our definition of cancelled flights (is.na(dep\_delay) | is.na(arr\_delay) ) is slightly suboptimal. Why? Which is the most important column?
   1. For some reason, there are flights with NA arr\_delay even when there’s an arrival and departure time. There are 1,175 flights with NA arr\_delay and a dep\_delay value existing. Dep\_delay is the most important column.
2. Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?
   1. There is a positive relationship between average delay and percentage of cancelled flights. Below is what I wrote as well as the plot it produced.

flights %>%

group\_by(year, month,day) %>%

mutate(

cancelled = sum(is.na(dep\_delay)),

not\_cancelled = sum(!is.na(dep\_delay))) %>%

summarize(n = n(),

cancelled = sum(cancelled),

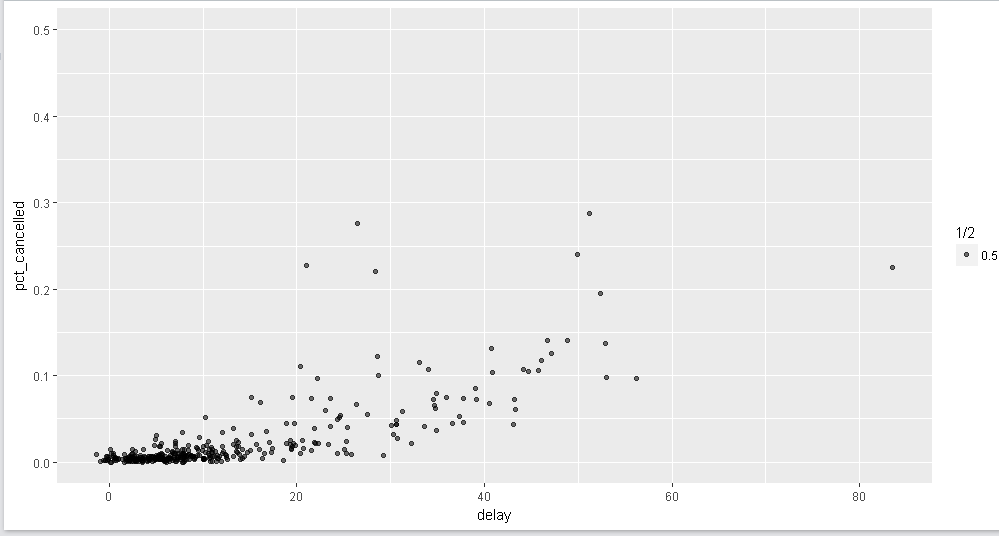
pct\_cancelled = cancelled / sum(not\_cancelled),

delay = mean(dep\_delay, na.rm = TRUE)) %>%

ggplot(mapping = aes(x = delay, y = pct\_cancelled)) +

geom\_point(aes(alpha = 1/2)) +

coord\_cartesian(ylim = c(0,.5))



1. 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()))
   1. Frontier Airlines (F9) has the longest mean delays, just over 20 minutes. It’s not realistic to look at bad airports by destination airport, because there are dozens of destinations and it’s not easy to digest without grouping further (which I’m sure is possible). It’s much easier to look at the effect of carriers and origin airports since there are only 3. Assuming that the airlines with only one origin airport don’t fly out of the other two airports, there doesn’t seem to be a large effect of origin airport on delays within the carriers on a whole, although specific airlines do have a larger proportion of delays from specific airports.

flights %>%

group\_by(carrier, origin) %>%

mutate(

cancelled = sum(is.na(dep\_delay)),

not\_cancelled = sum(!is.na(dep\_delay))

) %>%

summarize(n = n(),

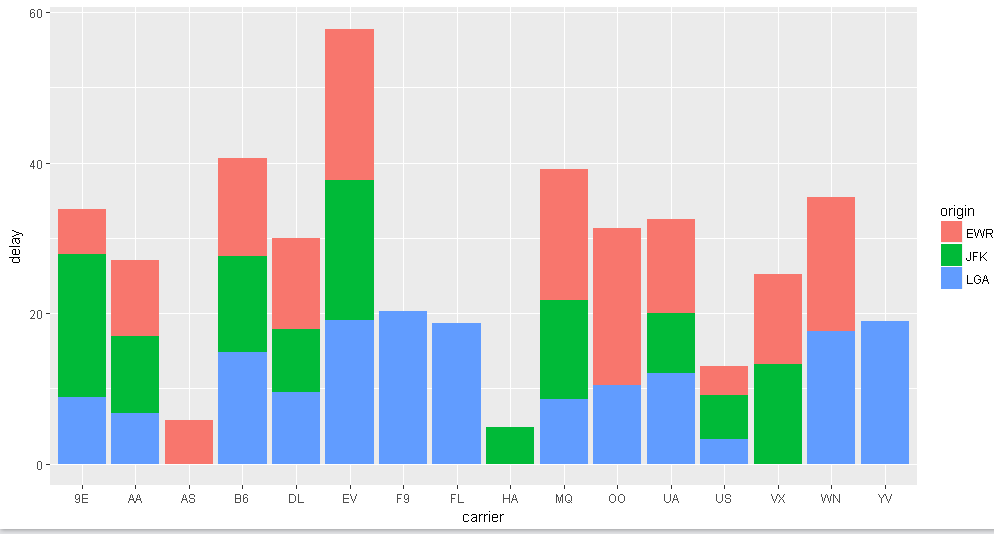
cancelled = sum(cancelled),

pct\_cancelled = cancelled / sum(not\_cancelled),

delay = mean(dep\_delay, na.rm = TRUE)) %>%

ggplot(mapping = aes(x = carrier, y = delay)) +

geom\_col(aes(fill = origin))



1. What does the sort argument to count() do. When might you use it?
   1. It sorts the output in either descending or ascending order. It’s useful if you’re grouping by something and want to see the grouping by most to least or vice versa. Ex:

flights %>%

group\_by(carrier) %>%

count(carrier, sort = TRUE)

**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.
   1. Overall, the grouping forces the filter or mutate to consider all observations within the grouping instead of each individual observation.
2. Which plane (tailnum) has the worst on-time record?
   1. N505MQ (with n > 50 since there were a lot of planes with <10 flights)

flights %>%

group\_by(tailnum) %>%

summarize(n = n(),

total\_delay = sum(arr\_delay >0, na.rm = TRUE),

total\_flight = sum(!is.na(arr\_delay)),

pct\_delayed = mean(total\_delay / total\_flight)) %>%

filter (n > 50) %>%

arrange(desc(pct\_delayed))

1. What time of day should you fly if you want to avoid delays as much as possible?
   1. 5am

flights %>%

group\_by(hour) %>%

summarize (n = n(),

delay = mean(dep\_delay, na.rm = TRUE))

1. For each destination, compute the total minutes of delay. For each, flight, compute the proportion of the total delay for its destination.
2. flights %>%

group\_by(dest) %>%

summarize(n=n(),

total\_delay = sum(arr\_delay, na.rm = TRUE))

1. flights %>%

filter(!is.na(arr\_delay), arr\_delay >0) %>%

group\_by(dest, tailnum) %>%

mutate(total\_delay = sum(arr\_delay, na.rm = TRUE),

prop\_delay = arr\_delay / total\_delay) %>%

select(dest, tailnum, total\_delay, arr\_delay, prop\_delay)

1. 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.
   1. There doesn’t seem to be a strong relationship between the immediately preceding flight (either by Origin only or by combination of Origin + Dest). The majority of delays are concentrated in the <30 minute area, both for previous delays and current delays. There is no sort of linear trend when plotting previous delay to current delay.
2. flights %>%

filter(!is.na(dep\_delay), dep\_delay < 120, month == 7) %>%

group\_by(origin, dest, month, day) %>%

mutate(prev\_delay = lag(dep\_delay, order\_by = dep\_time)) %>%

filter(!is.na(prev\_delay)) %>%

select(month, day, origin, dest, dep\_time, dep\_delay, prev\_delay) %>%

arrange(month, day, origin, dest, dep\_time)%>%

ggplot(mapping = aes(x = prev\_delay, y = dep\_delay)) +

geom\_point(alpha = 1/10)

1. 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?
   1. This flight was 57 minutes faster than the mean airtime:

flights %>%

filter(!is.na(dest), !is.na(origin), !is.na(air\_time)) %>%

group\_by(dest, origin) %>%

mutate(n = n(),

mean\_airtime = mean(air\_time), na.rm = TRUE,

airtime\_diff = air\_time - mean\_airtime, na.rm = TRUE,

min\_airtime = min(air\_time), na.rm = TRUE) %>%

select(tailnum, dest, origin, air\_time, mean\_airtime, airtime\_diff, min\_airtime) %>%

filter(airtime\_diff < -50, mean\_airtime < 200)

A tibble: 1 x 7

# Groups: dest, origin [1]

tailnum dest origin air\_time mean\_airtime airtime\_diff min\_airtime

<chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl>

1 N17196 MSP EWR 93 150.8102 -57.8102 93

* 1. These 4 flights were delayed by over 100 minutes in the air.

flights %>%

filter(!is.na(dest), !is.na(origin), !is.na(air\_time)) %>%

group\_by(dest, origin) %>%

mutate(n = n(),

mean\_airtime = mean(air\_time), na.rm = TRUE,

airtime\_diff = air\_time - mean\_airtime, na.rm = TRUE,

min\_airtime = min(air\_time), na.rm = TRUE) %>%

select(tailnum, dest, origin, air\_time, mean\_airtime, airtime\_diff, min\_airtime) %>%

filter(airtime\_diff > 100)

A tibble: 4 x 7

# Groups: dest, origin [4]

tailnum dest origin air\_time mean\_airtime airtime\_diff min\_airtime

<chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl>

1 N5DBAA EGE JFK 382 256.4455 125.5545 219

2 N178DN LAX JFK 440 329.1511 110.8489 275

3 N703TW SFO JFK 490 347.4036 142.5964 301

4 N578UA DEN LGA 331 227.5160 103.4840 186

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

flights %>%

filter(!is.na(dest), !is.na(carrier), n\_distinct(carrier)>=2) %>%

group\_by(dest) %>%

summarize(num\_carrier = n\_distinct(carrier)) %>%

arrange(desc(num\_carrier))

A tibble: 105 x 2

dest num\_carrier

<chr> <int>

1 ATL 7

2 BOS 7

3 CLT 7

4 ORD 7

5 TPA 7

6 AUS 6

7 DCA 6

8 DTW 6

9 IAD 6

10 MSP 6

# ... with 95 more rows

1. For each plane, count the number of flights before the first delay of greater than 1 hour.
   1. To be perfectly honest, this one confused the crap out of me, and I had to find someone else’s answer to work through and understand. So instead of pretending I wrote this myself, I’m going to explain how I understood someone else’s work (that also really confused me at first glance):

flights %>%

arrange(tailnum, year, month, day) %>% **(putting in order so that the cumsum further down would be chronological)**

group\_by(tailnum)%>% **(for the cumsum to be by flights)**

mutate(delay\_gt1hr = dep\_delay > 60) %>% **(creating a column with true/false as values for True (i.e. 1) when greater than 60 minute delay)**

mutate(before\_delay = cumsum(delay\_gt1hr)) %>% **(getting a cumulative sum by talinum of all flights, in chronological order, so the first flight > 60 would be clear)**

filter(before\_delay < 1) %>% **(limiting to all flights with cumsum of 0, which are all flights before that first 60 minute delay)**

count(sort = TRUE) %>% **(counting the values by flight in descending order of n)**

Source: <https://jrnold.github.io/e4qf/data-transformation.html>

**6.3**

1. Go to the RStudio Tips twitter account, <https://twitter.com/rstudiotips> and find one tip that looks interesting. Practice using it!
   1. I chose this one. Seems very useful when you want to focus in on one specific pane and have the full screen to work or analyze.



1. What other common mistakes will RStudio diagnostics report? Read <https://support.rstudio.com/hc/en-us/articles/205753617-Code-Diagnostics> to find out.
   1. Check arguments to R function calls (i.e. will the function work)
   2. Warn if variable has no definition in scope (isn’t defined)
   3. Warn if variable defined but not used (for cleaner code)
   4. Provide R style diagnostics
   5. There are also diagnostics available for other languages like C++, JS, and Python

**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.
   1. The majority of X observations are between 4-7; y between 3 -9, and Z has a large spike ~2.5
   2. The distribution of X and Y are incredibly similar, leading me to believe those are the length and width. Further, I believe X is the length and Y is the width. X and Y are highly correlated, but the observations outside the overall trend line generally have a lower Y than X. Since people generally wear oval diamonds with the long side vertical instead of horizontal, the higher x value with lower y value are likely oval diamonds, with the majority of diamonds being standard circle cuts.
   3. Z is most likely the depth, because there is a huge spike at ~2.5. I imagine there’s a weaker correlation between diamond size (in mm) and depth, and that is seen when I plot X or Y against Z to see the relationship. Overall the depth increases with the height and width, but there is more variation in depth than there is in the relationship between length and width.
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.)
   1. I was surprised to see such a high number of ideal cut diamonds under 1000. When I mapped price against carat size for ideal cuts however, I saw that the majority of ideal cut diamonds were under 1 carat, with over half of them under .5 carats.
3. How many diamonds are 0.99 carat? How many are 1 carat? What do you think is the cause of the difference?
   1. 23 are .99 carat, 1,558 are 1 carat. With diamonds, at whole and half (and maybe quarter) carats, the price goes up exponentially. So a diamonds that’s 1.05 carats could be significantly more expensive than a .99.
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?
   1. Coord\_cartesian() doesn’t drop data observations outside of the visual zoom, while xlim() and ylim() on their own do.
   2. Binwidth defaults to 30 if you leave it unset
   3. Using coord\_cartesian(), zooming in to see only half a bar still shows that half bar at the edge of the graph, while using xlim will remove data past the limit from the graph.

**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?
   1. Both remove NA values, but because of different stats. Geom\_bar() gives the message: Removed XX rows containing non-finite values (stat\_count) because it’s counting observations by discrete categories (and NA can’t be counted). Geom\_histogram() gives the message: Removed XX rows containing non-finite values (stat\_bin) because it’s counting observations within certain bins of continuous values (and NA can’t fit within a bin).
2. What does na.rm = TRUE do in mean() and sum()?
   1. It removes NA values so that the calculation of mean() or sum() actually returns a value and not just all NA.

**7.5.1.1**

1. Use what you’ve learned to improve the visualisation of the departure times of cancelled vs. non-cancelled flights.
   1. By using ..density.. as the y value in the geom\_freqpoly plot, we can see that the proportion of cancelled flights is significantly higher between the hours of 3pm – 9pm, likely due to delays earlier in the day or afternoon weather.

flights %>%

mutate(

cancelled = is.na(dep\_time),

sched\_hour = sched\_dep\_time %/% 100,

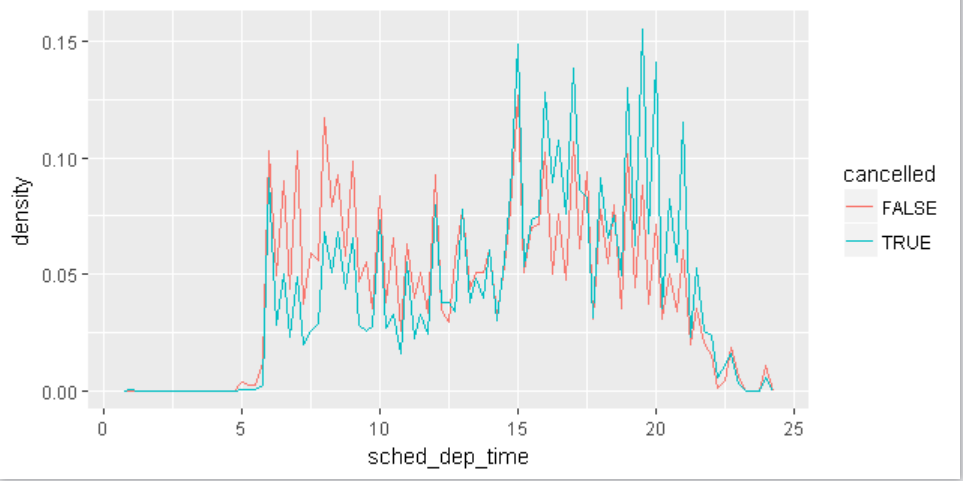
sched\_min = sched\_dep\_time %% 100,

sched\_dep\_time = sched\_hour + sched\_min / 60

) %>%

ggplot(mapping = aes(x = sched\_dep\_time, y = ..density..)) +

geom\_freqpoly(mapping = aes(colour = cancelled), binwidth = 1/4)

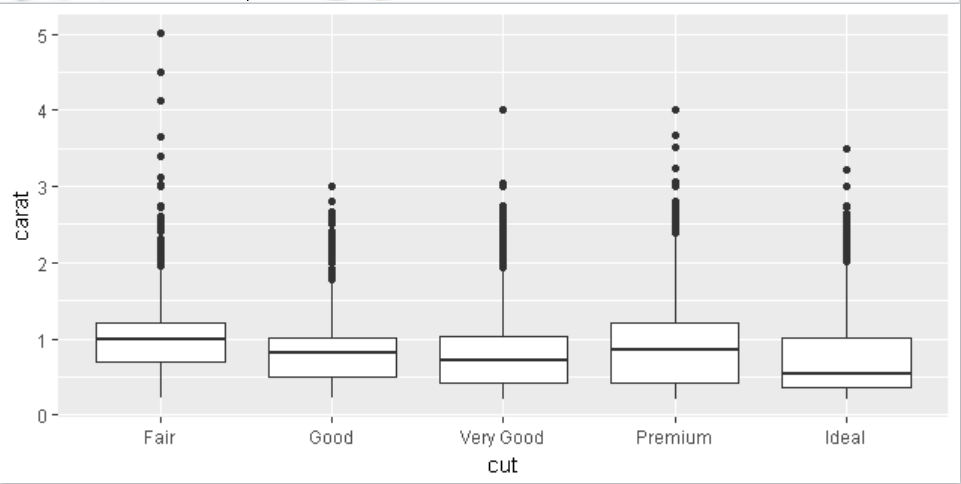


1. What variable in the diamonds dataset is most important for predicting the price of a diamond? How is that variable correlated with cut? Why does the combination of those two relationships lead to lower quality diamonds being more expensive?
   1. Carat is the most important predictor for diamond price. Looking at a boxplot of diamonds by cut and carat, the median carat size is highest for Fair, with Ideal at the lowest. Even though the ideal cut diamonds are higher quality, they are smaller and therefore less expensive on average than the Fair diamonds.

diamonds %>%

ggplot(mapping = aes(x = cut, y = carat)) +

geom\_boxplot()

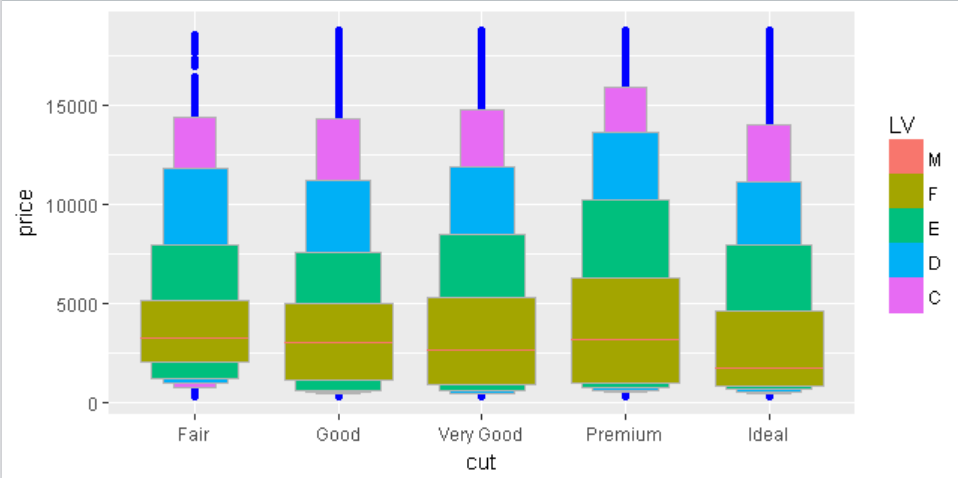


1. Install the ggstance package, and create a horizontal boxplot. How does this compare to using coord\_flip()?
   1. It produces the same visual, but a regular boxplot uses x = discrete, y = continuous, and coord\_flip just flips the axes for the visual. For geom\_boxploth, x = continuous and y = discrete.
2. 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?
   1. Overall, I can see that there are a significant volume of “outliers” outside the middle quartiles. The lv plot shows ntiles beyond the quartiles, that you can define within the plot. Each letter refers to a specific ntile. K = 2 is essentially a regular boxplot, with anything 2+ adding additional ntiles.

diamonds %>%

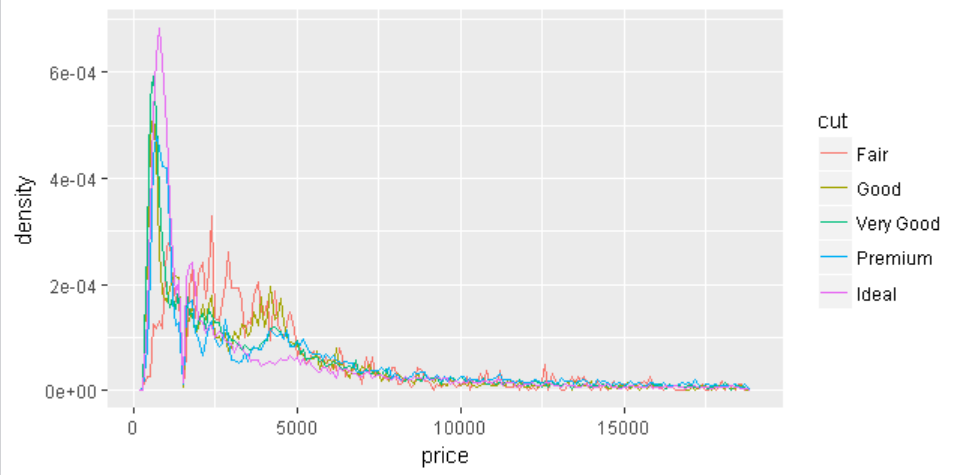
ggplot(mapping = aes(x = cut, y = price)) +

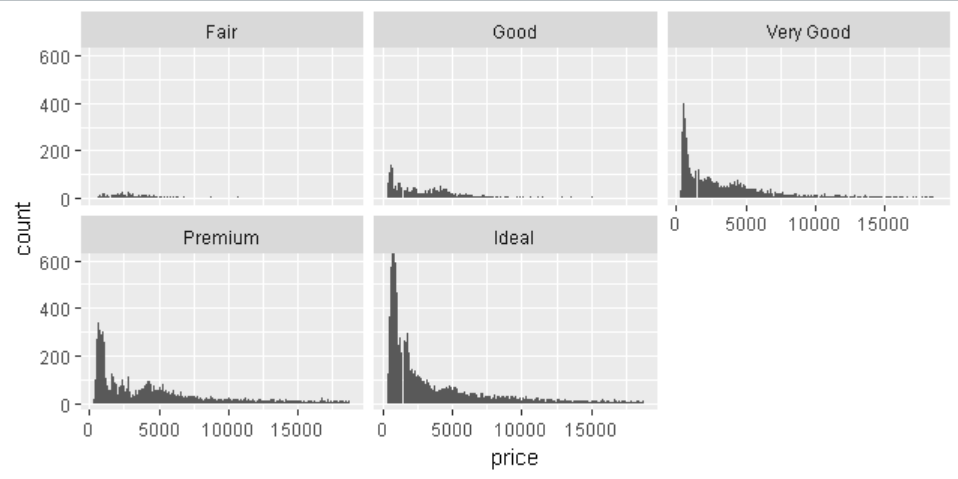
geom\_lv(k = 5,outlier.colour = "blue", aes(fill = ..LV..))



1. Compare and contrast geom\_violin() with a facetted geom\_histogram(), or a coloured geom\_freqpoly(). What are the pros and cons of each method?
   1. Geom\_violin() allows you to see the density and distribution of observations in line, which is really helpful to compare proportion of distributions visually when you’re not concerned about the number of observations in each group but rather the distributions. You can do something similar using density as y with geom\_freqpoly(), but unless the distributions are drastically different, the lines will be overlapping and aren’t necessarily helpful beyond a quick look. Geom\_histogram() using facet\_wrap also allows you to view the distribution and density of each group in plots next to each other, but if one group is large (ideal) and one is very small (fair), the large y scale necessary to show both doesn’t necessarily help you see the differences in distribution very well.







1. 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.
   1. Geom\_quasirandom() produces a plot similar to violin, but with points jittered to avoid overplotting. You can see the density of observations by setting the alpha low.
   2. Geom\_beeswarm() produces a plot similar to a boxplot but with points jittered to show distribution. The width/number of points at a certain y value represents the number/density of observations at that location.

**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?

diamonds %>%

group\_by(cut) %>%

count(cut,color) %>%

mutate(prop = n/sum(n))

1. 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?
   1. There are far too many destinations to fit on either axis, and by default the x axis (months) is showing increments of 2.5 instead of 1.
   2. I forced the x scale to use each month discretely, and limited n > 400 because that’s the lowest number I could see all destinations. I also altered the color scale to have darkest as the highest mean delay since that’s more intuitive.

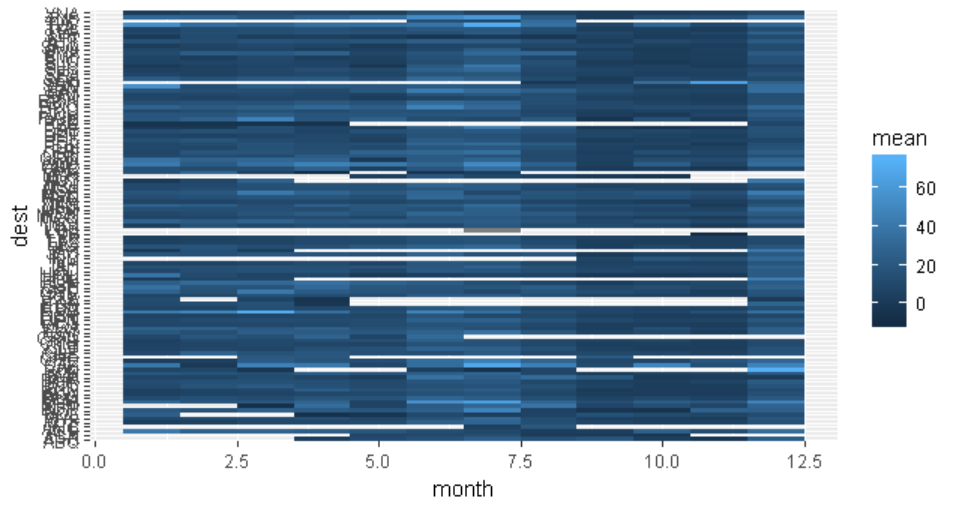
flights %>%

group\_by(dest, month) %>%

summarize(n = n(), mean = mean(dep\_delay, na.rm = TRUE)) %>%

ggplot(mapping = aes(x = month, y = dest)) +

geom\_tile(mapping = aes(fill = mean))



flights %>%

group\_by(dest, month) %>%

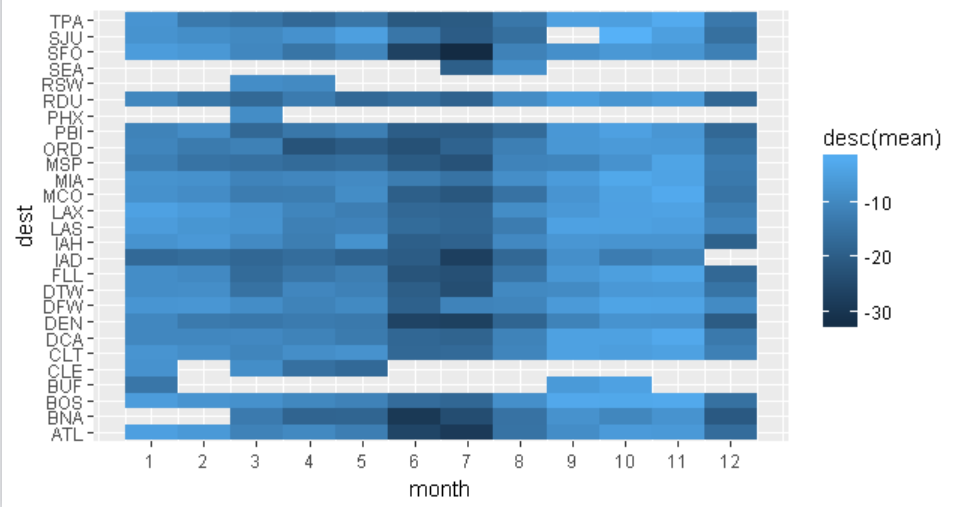
summarize(n = n(), mean = mean(dep\_delay, na.rm = TRUE)) %>%

filter(n > 400) %>%

ggplot(mapping = aes(x = month, y = dest)) +

geom\_tile(mapping = aes(fill = desc(mean))) +

scale\_x\_continuous(breaks = seq(1:12))



1. Why is it slightly better to use aes(x = color, y = cut) rather than aes(x = cut, y = color) in the example above?
   1. There are more options for color than cut, and as we learned a few chapters ago it’s better to have the higher number of options on the x axis since the viewer is a rectangle.

**7.5.3.1**

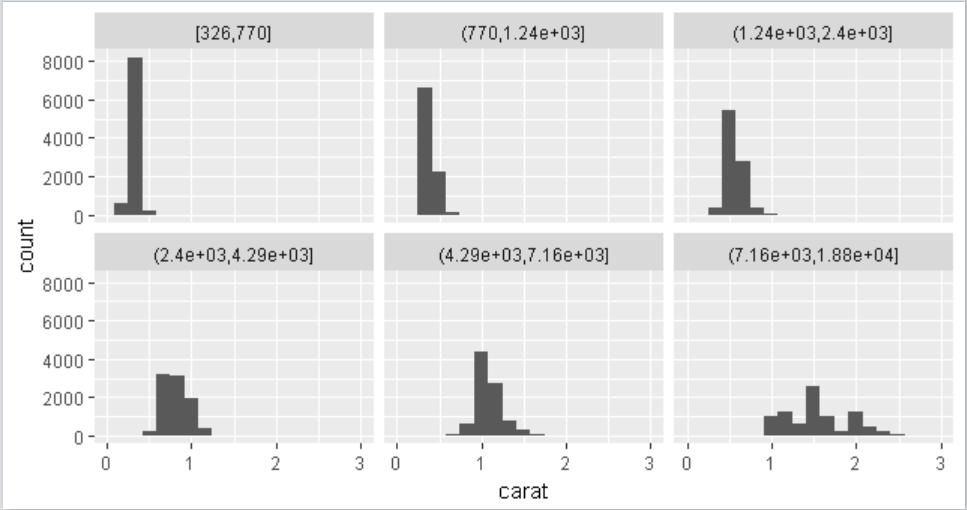
1. Instead of summarising the conditional distribution with a boxplot, you could use a frequency polygon. What do you need to consider when using cut\_width() vs cut\_number()? How does that impact a visualisation of the 2d distribution of carat and price?
   1. Cut\_width is based on the carat value, while cut\_number splits the groupings up into 20 bins without specifically stated limits. Unless you know the range and distribution of carat values and can specifically create a number of bins to match a specific cut\_width number, the visualizations will look very different.
2. Visualise the distribution of carat, partitioned by price.

ggplot(data = diamonds, mapping = aes(x = carat)) +

geom\_histogram(mapping = aes(group = cut\_width(carat, .25))) +

coord\_cartesian(xlim = c(0, 3)) +

facet\_wrap(~cut\_number(price,6))



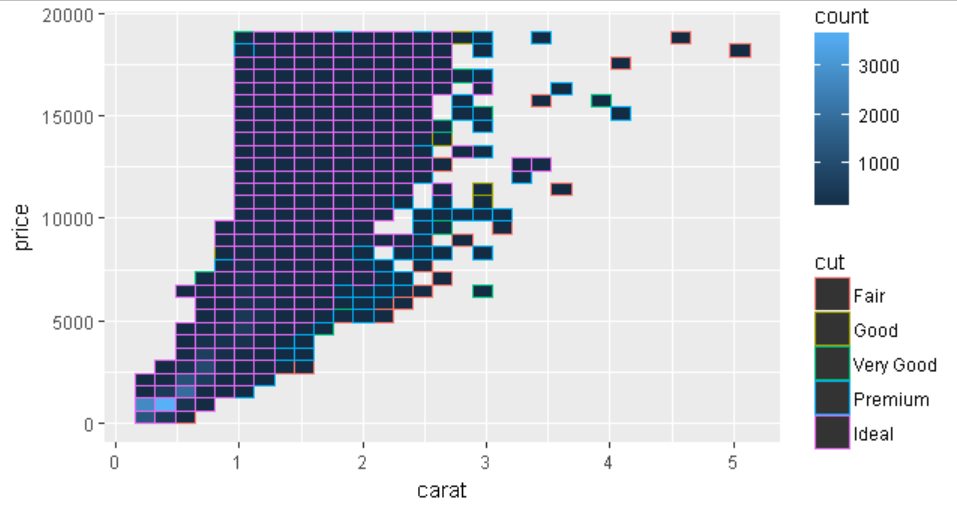
1. How does the price distribution of very large diamonds compare to small diamonds. Is it as you expect, or does it surprise you?
   1. The price distribution is much narrower for small diamonds than large diamonds. This makes sense, because even a high quality diamond that’s small is still a small diamond. I would expect to see a broader range of prices for larger diamonds, because a high quality large diamond will be proportionally more expensive than a low quality large diamond.
2. Combine two of the techniques you’ve learned to visualise the combined distribution of cut, carat, and price.

diamonds %>%

ggplot() +

geom\_bin2d(mapping = aes(x = carat, y = price)) +

facet\_wrap(~cut)

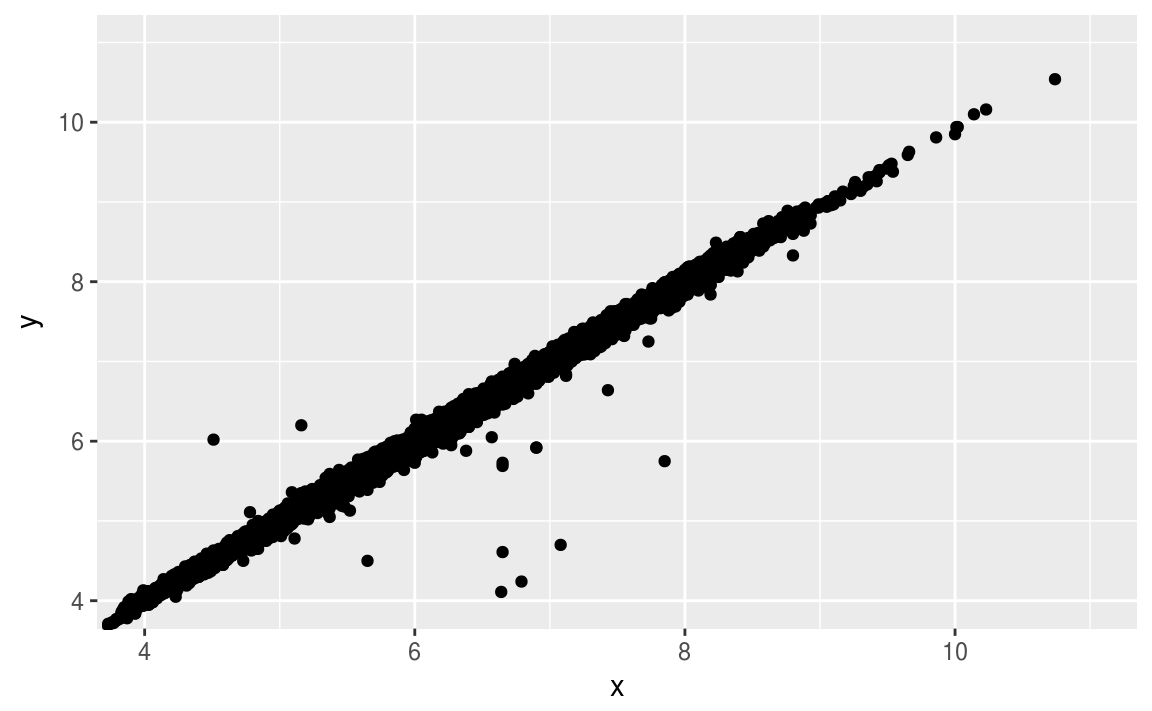


1. Two dimensional plots reveal outliers that are not visible in one dimensional plots. For example, some points in the plot below have an unusual combination of x and y values, which makes the points outliers even though their x and y values appear normal when examined separately.

**ggplot**(data = diamonds) +

**geom\_point**(mapping = **aes**(x = x, y = y)) +

**coord\_cartesian**(xlim = **c**(4, 11), ylim = **c**(4, 11))



Why is a scatterplot a better display than a binned plot for this case?

* 1. In this case, the data set is so large that showing these outliers within a bin would suggest that there’s a much larger number of observations in that bin than actually exist. There could be 5 or 500 observations within the bin, and it would still be almost the exact same color since the range is so large. The scatterplot shows that the observations in that “binned” area are actually few in number.