# RV R Exploration Exercises

## 3.2.4 Exercises

**Run ggplot(data = mpg). What do you see?**

An empty gray rectangle with no labels.

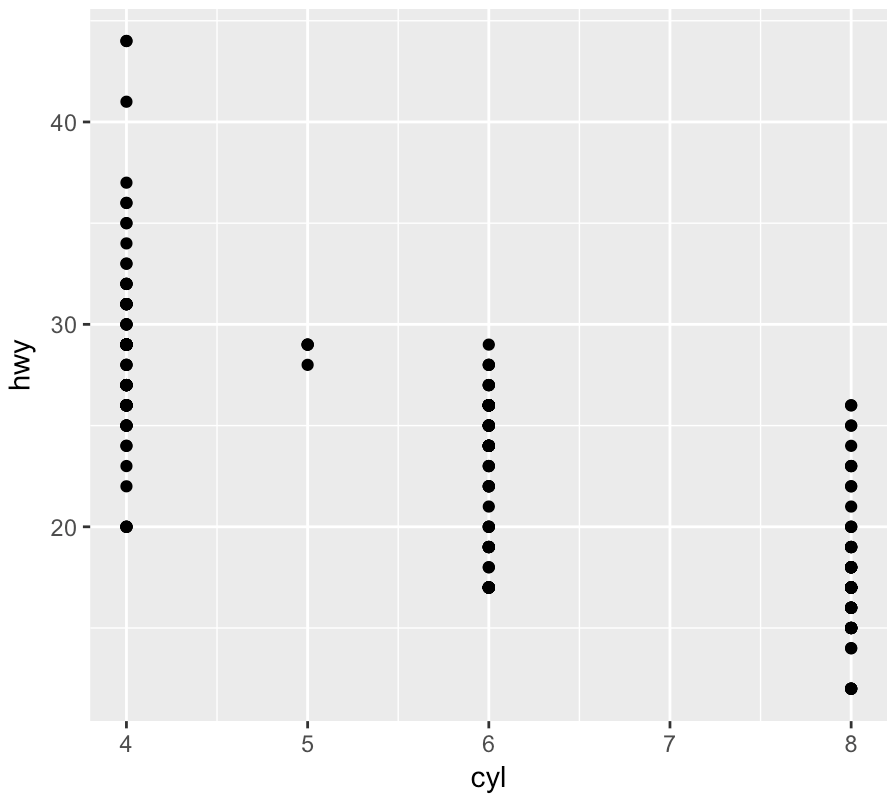
**How many rows are in mpg? How many columns?**

234 rows, 11 columns

**What does the drv variable describe? Read the help for ?mpg to find out.**

drv specifies whether a vehicle has a front-wheel drive, rear-wheel drive or 4-wheel drive.

**Make a scatterplot of hwy vs cyl.**



**What happens if you make a scatterplot of class vs drv? Why is the plot not useful?**

“class” is a qualitative variable so the graph doesn’t suggest a line of regression. In addition, points stack on top of one another, so the viewer has no sense how many of each class of car corresponds with each type of drivetrain.

## 3.3.1 Exercises

**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"))

color = “blue” should be outside of aes() for manual assignment. E.g:

**ggplot**(data = mpg) +

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

**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?**

Categorical: manufacturer, model, trans, drv, fl, class

Continuous: displ, cty, hwy

year and cyl are technically discrete variables since their values are quantized to integers.

Note the data types by simply entering **mpg.** <chr> (character/string), <dbl> (double precision decimal), and <int> (integer). String values are categorical.

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

Color maps continuous variables along a gradient.

Shape returns an error when trying to make a continuous variable.

Size maps points of varying sizes corresponding to the variable’s value.

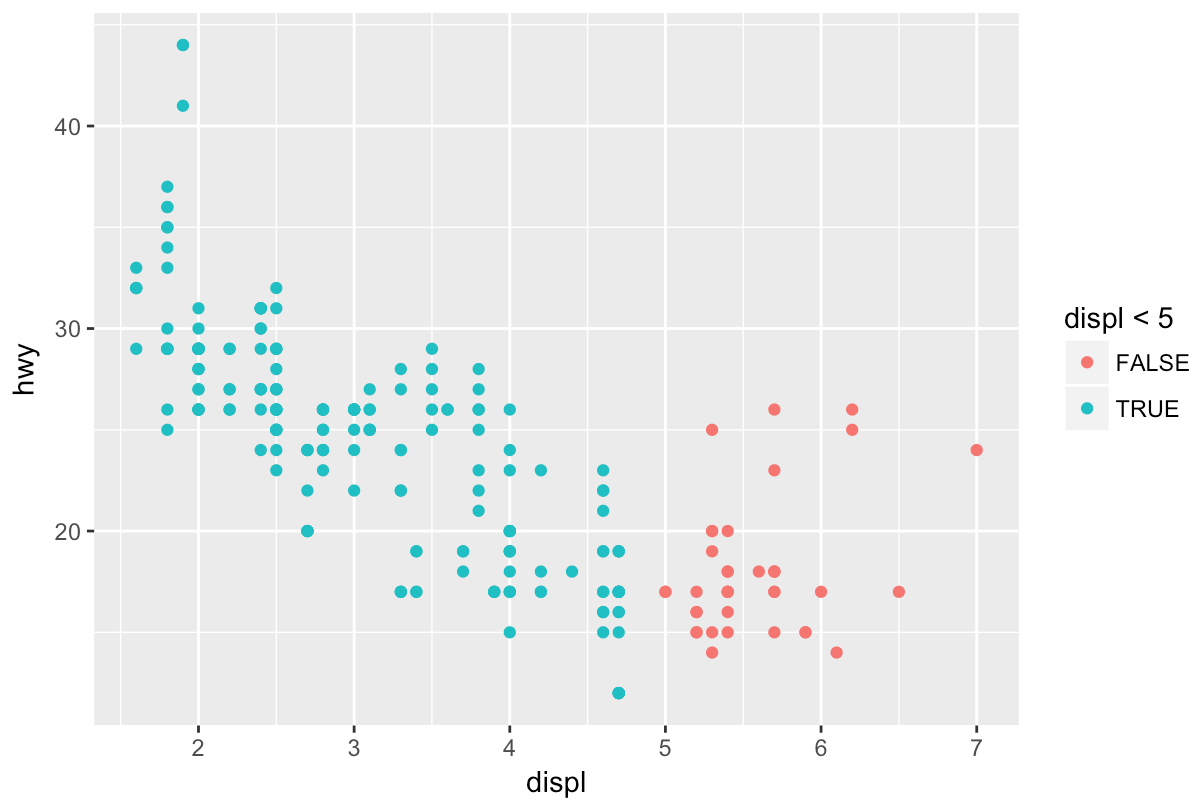
**What happens if you map the same variable to multiple aesthetics?**

You get a diagonal line of dots with a slope of 1.

**What does the stroke aesthetic do? What shapes does it work with? (Hint: use ?geom\_point)**

Changes border width. Works with shapes that have borders.

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

Ggplot changes the color for values greater than five and displays a true/false key:  


## 3.5.1 Exercises

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

ggplot rounds to the nearest integer and creates facet for each value

**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))

The empty cells show the types of cars with combinations of numbers of cylinders and types of drivetrains. The plot shows the instances of models across those variables. There are no 7-cylinder vehicles or 5-cylinder vehicles with 4-wheel or rear drive.

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

**ggplot**(data = mpg) +

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

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

**ggplot**(data = mpg) +

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

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

Creates a facet grid of displacement versus highway mileage based on drivetrain (in the first) and number of cylinders (in the second). The period tells the function there is no variable for that dimension of the grid.

**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?**

It’s easier to compare data points within a given class with facets. It’s easier to see how classes relate to one another with colors, since they’re layered on the same plot. Increasing classes would make facets less practical as they would have to display smaller or require more space.

**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?**

nrow specifies the number of rows to use, and ncol specifies the number of columns.

Other options (from documentation):

|  |  |
| --- | --- |
| scales | should Scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free\_x", "free\_y").  i.e., fixed constrains proportions of panels |
| shrink | If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary. |

(i.e., shrink scales panels to existing data)

|  |  |
| --- | --- |
| switch | By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both". |

…etc. Facet\_grid’s dimensions are determined by the number of categories per variable.

**When using facet\_grid() you should usually put the variable with more unique levels in the columns. Why?**

Because the display is laid out horizontally.

## 3.6.1 Exercises

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

geom\_line, geom\_box, geom\_histogram, geom\_area, respectively.

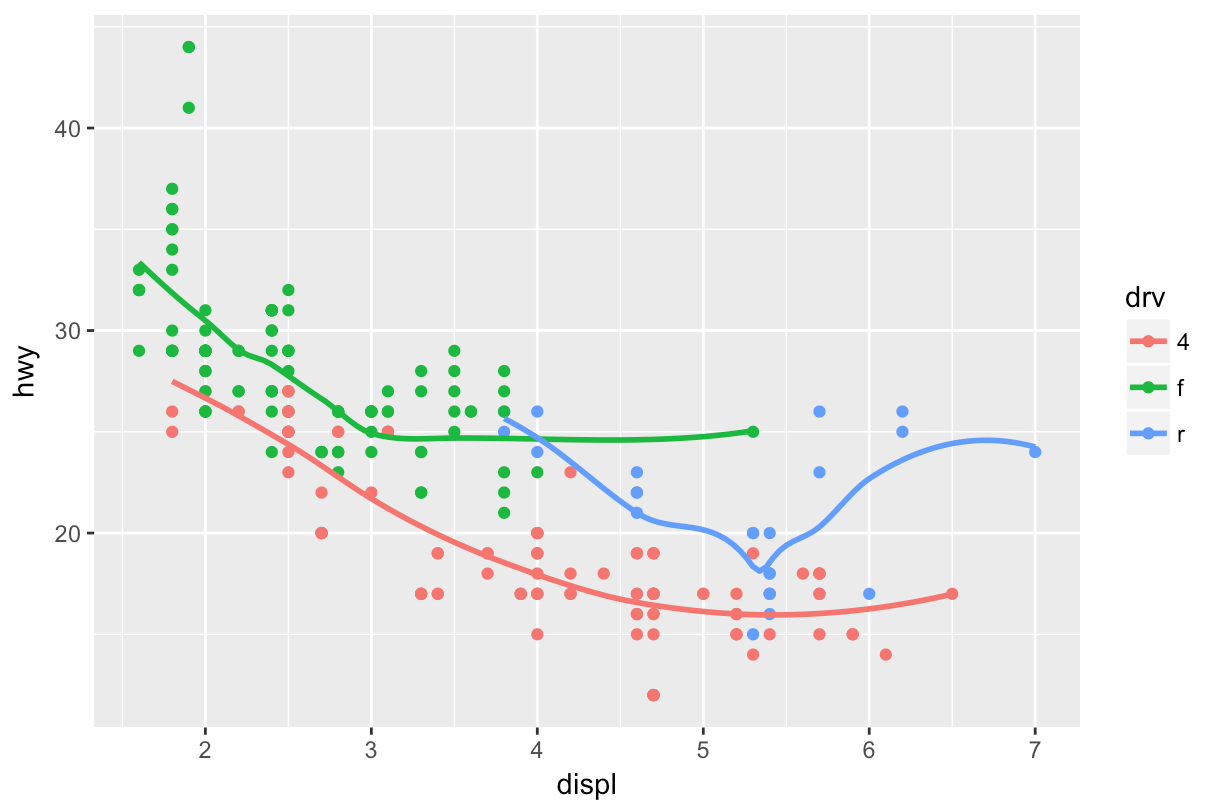
**Run this code in your head and predict what the output will look like. Then, run the code in R and check your predictions.**

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

**geom\_point**() +

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

I didn’t anticipate that there would be more than one line of fit.



**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 omits the color key from the right. Probably omitted in the series of 3 graphs to keep the last scaled consistently with the first two.

**What does the se argument to geom\_smooth() do?**

Hides (or shows) the confidence interval on the line of fit.

**Will these two graphs look different? Why/why not?**

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

**geom\_point**() +

**geom\_smooth**()

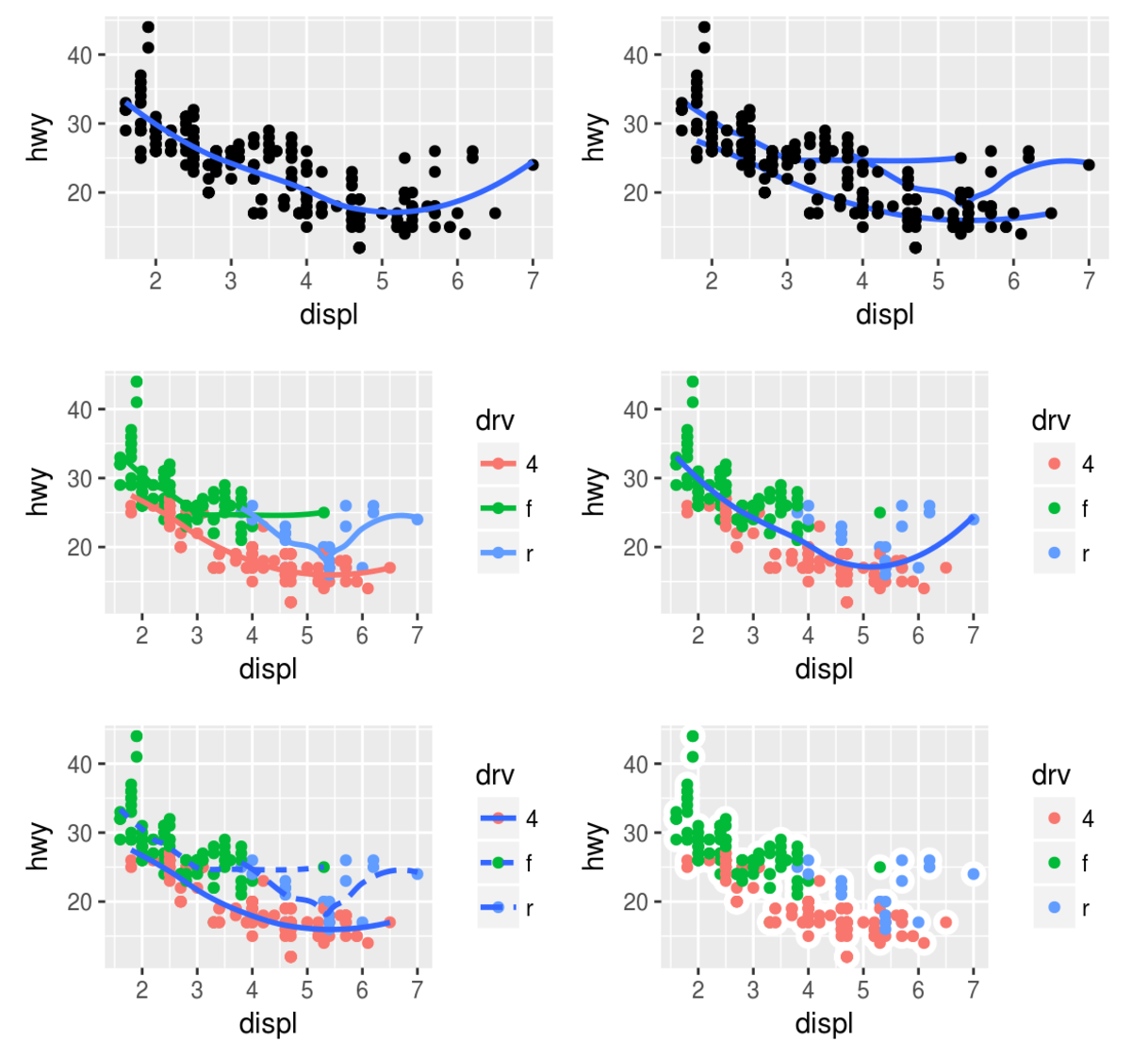
**ggplot**() +

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

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

No. The inner functions receive the same data either way.

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



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

+ geom\_point() +

+ geom\_smooth(se = FALSE)

ggplot(data = mpg) +

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

+ geom\_smooth(mapping = aes(x = displ, y = hwy, group = drv), se = FALSE)  
  
gplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +

+ geom\_point() +

+ geom\_smooth(se = FALSE)

ggplot(data = mpg) +

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

geom\_smooth(mapping = aes(x = displ, y = hwy), se = FALSE)

ggplot(data = mpg) +

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

geom\_smooth(mapping = aes(x = displ, y = hwy, linetype = drv), se = FALSE)

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

geom\_point(mapping = aes(size = 2), color = "white") +

geom\_point(mapping = aes(color = drv), size = 1.75)

## 3.7.1 Exercises

**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)

**What does geom\_col() do? How is it different to geom\_bar()?**

geom\_col makes a bar chart based on numbers in the observations, rather than the count of rows fitting each x value. It uses stat\_identity instead of stat\_count.

**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?**

They’re capable of transforming and displaying data in identical plots. Edited from: <http://ggplot2.tidyverse.org/reference/#section-layer-geoms>

|  |  |  |
| --- | --- | --- |
| [**geom\_bar**](http://ggplot2.tidyverse.org/reference/geom_bar.html) [**geom\_col**](http://ggplot2.tidyverse.org/reference/geom_bar.html) [**stat\_count**](http://ggplot2.tidyverse.org/reference/geom_bar.html) | Bars charts | |
| [**geom\_bin2d**](http://ggplot2.tidyverse.org/reference/geom_bin2d.html) [**stat\_bin\_2d**](http://ggplot2.tidyverse.org/reference/geom_bin2d.html) | Heatmap of 2d bin counts | |
| [**geom\_boxplot**](http://ggplot2.tidyverse.org/reference/geom_boxplot.html) [**stat\_boxplot**](http://ggplot2.tidyverse.org/reference/geom_boxplot.html) | A box and whiskers plot (in the style of Tukey) | |
| [**geom\_contour**](http://ggplot2.tidyverse.org/reference/geom_contour.html) [**stat\_contour**](http://ggplot2.tidyverse.org/reference/geom_contour.html) | 2d contours of a 3d surface | |
| [**geom\_count**](http://ggplot2.tidyverse.org/reference/geom_count.html) [**stat\_sum**](http://ggplot2.tidyverse.org/reference/geom_count.html) | Count overlapping points | |
| [**geom\_density\_2d**](http://ggplot2.tidyverse.org/reference/geom_density_2d.html) [**stat\_density\_2d**](http://ggplot2.tidyverse.org/reference/geom_density_2d.html) | Contours of a 2d density estimate | |
| [**geom\_density**](http://ggplot2.tidyverse.org/reference/geom_density.html) [**stat\_density**](http://ggplot2.tidyverse.org/reference/geom_density.html) | Smoothed density estimates | |
| [**geom\_hex**](http://ggplot2.tidyverse.org/reference/geom_hex.html) [**stat\_bin\_hex**](http://ggplot2.tidyverse.org/reference/geom_hex.html) | Hexagonal heatmap of 2d bin counts | |
| [**geom\_freqpoly**](http://ggplot2.tidyverse.org/reference/geom_histogram.html) [**geom\_histogram**](http://ggplot2.tidyverse.org/reference/geom_histogram.html) [**stat\_bin**](http://ggplot2.tidyverse.org/reference/geom_histogram.html) | Histograms and frequency polygons | |
| [**geom\_qq**](http://ggplot2.tidyverse.org/reference/geom_qq.html) [**stat\_qq**](http://ggplot2.tidyverse.org/reference/geom_qq.html) | A quantile-quantile plot | |
| [**geom\_quantile**](http://ggplot2.tidyverse.org/reference/geom_quantile.html) [**stat\_quantile**](http://ggplot2.tidyverse.org/reference/geom_quantile.html) | Quantile regression | |
| [**geom\_smooth**](http://ggplot2.tidyverse.org/reference/geom_smooth.html) [**stat\_smooth**](http://ggplot2.tidyverse.org/reference/geom_smooth.html) | Smoothed conditional means | |
| [**geom\_violin**](http://ggplot2.tidyverse.org/reference/geom_violin.html) [**stat\_ydensity**](http://ggplot2.tidyverse.org/reference/geom_violin.html) | Violin plot | |
| **What variables does stat\_smooth() compute? What parameters control its behaviour?**  Computes: y -predicted value  Ymin-lower pointwise confidence interval around the mean  Ymax-upper pointwise confidence interval around the mean  Se-standard error.  *Params:*  span - controls amount of smoothingn - number of points to evaluate smoother at  also: level, formula, se, fullrange, etc.  **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..))  group = 1, within aes(), tells ggplot to compare each group to the whole rather than to itself | |  |
|  | |  |

## 3.8.1 Exercises

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

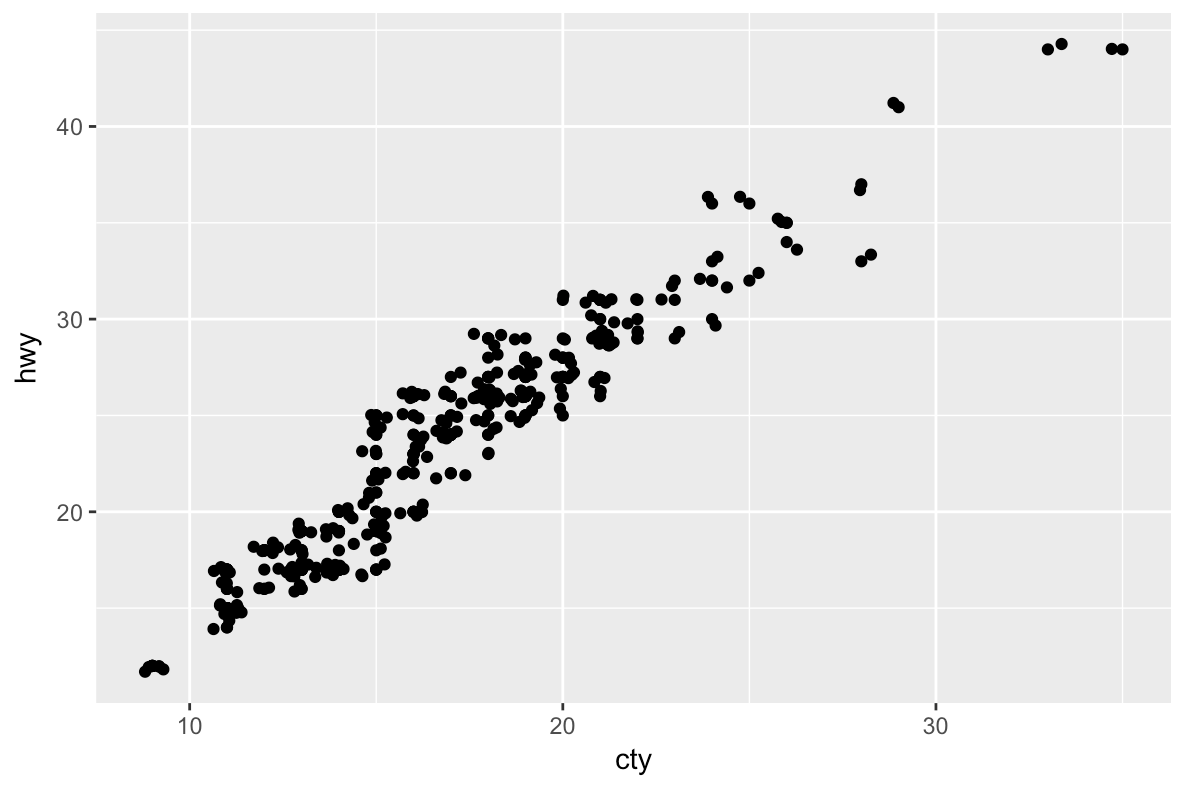
**ggplot**(data = mpg, mapping = **aes**(x = cty, y = hwy)) +

**geom\_point**()

I’d jitter that bad boy up to help with overplotting.

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

geom\_point() +

geom\_jitter()  


**What parameters to geom\_jitter() control the amount of jittering?**

width, height

**Compare and contrast geom\_jitter() with geom\_count().**

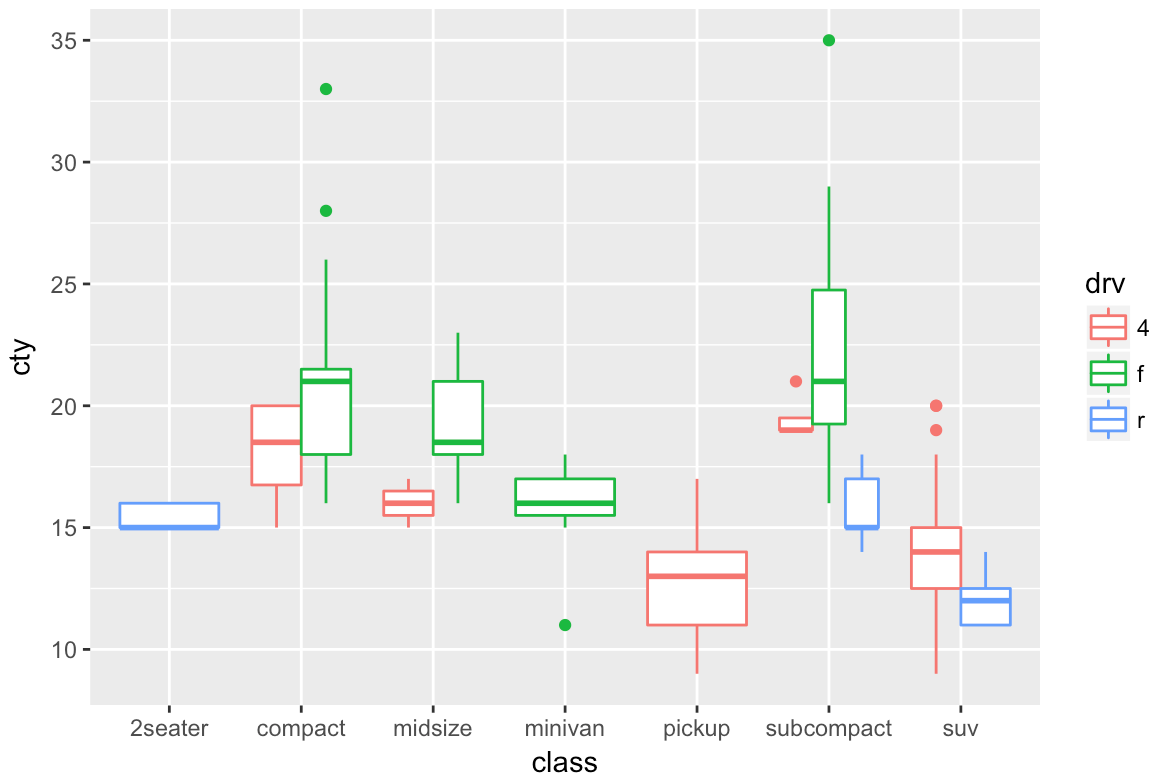
Both are methods of mitigating overplotting. geom\_count uses point size to show stacked points. geom\_count is distinct in that it doesn’t reduce the accuracy of the plot at smaller sizes.

**What’s the default position adjustment for geom\_boxplot()? Create a visualisation of the mpgdataset that demonstrates it.**

dodge.

p <- ggplot(mpg, aes(class, cty))

p + geom\_boxplot(aes(colour = drv))



## 3.9.1 Exercises

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

prem <- filter(diamonds, cut == "Premium")

cutplot <- ggplot(data = prem) +

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

cutplot

cutplot + coord\_polar("y", start = 0)

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

Allows you to customize your plot’s labels.

**What’s the difference between coord\_quickmap() and coord\_map()?**

coord\_quickmap approximates map projections without accounting for the roundness of the earth. Much less processor/memory intensive, but not as accurate for large landmasses closer to the poles.

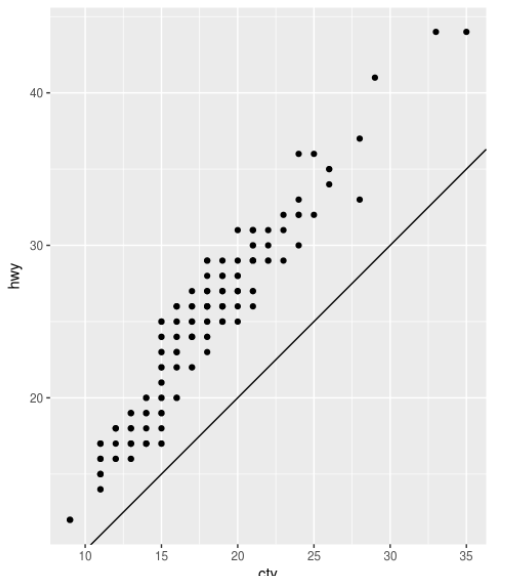
**What does the plot below tell you about the relationship between city and highway mpg? Why is coord\_fixed() important? What does geom\_abline() do?**

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

**geom\_point**() +

**geom\_abline**() +

**coord\_fixed**()

geom\_abline is a reference line showing a slope of 1–perfect correlation. coord\_fixed constrains aspect ratio of grid to be 1:1 for accurate slope.

## 4.1.1 Exercises

**Why does this code not work?**

my\_variable <- 10

my\_varıable

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

my\_variable is spelled with a 1.

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

**library**(tidyverse)

**ggplot**(dota = mpg) +

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

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

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

library(tidyverse)

ggplot(data = mpg) +

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

filter(mpg, cyl == 8)

filter(diamonds, carat > 3)

**Press Alt + Shift + K. What happens? How can you get to the same place using the menus?**

Displays keyboard shortcut quick reference. Available at Tools > Keyboard Shortcuts Help.

## 5.2.1 Exercises

**Find all flights that**

1. **Had an arrival delay of two or more hours**

filter(flights, arr\_delay >= 120)

1. **Flew to Houston (IAH or HOU)**

filter(flights, dest == "IAH" | dest == "HOU")

1. **Were operated by United, American, or Delta**

filter(flights, carrier == "DL" | carrier == "UA" | carrier == "AA")

1. **Departed in summer (July, August, and September)**

filter(flights, month %in% c(7, 8,9))

1. **Arrived more than two hours late, but didn’t leave late**

filter(flights, arr\_delay > 120, dep\_delay <= 0)

1. **Were delayed by at least an hour, but made up over 30 minutes in flight**

filter(flights, dep\_delay >= 60, arr\_delay <= (dep\_delay - 30))

1. **Departed between midnight and 6am (inclusive)**

filter(flights, dep\_time >= 0, dep\_time <= 600)

**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?**

Filters values between two parameters, inclusive. Yump:

filter(flights, between(dep\_time, 0, 600))

**How many flights have a missing dep\_time? What other variables are missing? What might these rows represent?**

8245. They’re canceled flights and thusly lack arrival times and delay times, etc.   
filter(flights, is.na(dep\_time))

**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!)**

Anything to the zeroeth power is 1. The TRUE on the right trumps the unknown value, regardless of whether it would be TRUE or FALSE, because of the OR operator (false or true still evaluates to true). FALSE & NA is not missing because FALSE & TRUE evaluates to FALSE and FALSE & FALSE evaluates to FALSE. The general rule is if it makes sense to you, or failing that, the designer of R (There is no spoon.).

## 5.3.1 Exercises

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

arrange(flights, desc(is.na(dep\_time)))

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

arrange(flights, dep\_delay)

arrange(flights, desc(dep\_delay))

1. **Sort flights to find the fastest flights.**

arrange(flights, air\_time)

1. **Which flights travelled the longest? Which travelled the shortest?**

arrange(flights, desc(distance))

arrange(flights, distance)

## 5.4.1 Exercises

**Brainstorm as many ways as possible to select dep\_time, dep\_delay, arr\_time, and arr\_delay from flights.**

select(flights, dep\_time, dep\_delay, arr\_time, arr\_delay)

select(flights, dep\_time:arr\_delay, -(starts\_with("sched")))

select(flights, -(year:day), -(starts\_with("sched")), -(hour), -(minute), -(carrier), -(tailnum), -(flight), -(origin), -(dest), -(air\_time), -(distance), -(time\_hour))

transmute(flights, dep\_time, dep\_delay, arr\_time, arr\_delay)

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

R honors the first position of the variable name and ignores subsequent mentions.

**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")

one\_of(), counterintuitively given the name, allows you to select all variables from within the character vector as dataframe columns. E.g.:  
select(flights, one\_of(vars))

**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"))

Yup. Apparently they’re not case sensitive. Change that with ignore.case = false:  
  
select(flights, contains("TIME", ignore.case = FALSE))

## 5.5.2 Exercises

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.**

mutate(flights, dep\_time\_min =

dep\_time %/% 100\*60 + dep\_time %% 100,

sched\_dep\_time\_min = sched\_dep\_time %/% 100 \* 60 + sched\_dep\_time %% 100) %>%

select(dep\_time\_min, sched\_dep\_time\_min, everything())

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?**

That looks like an attempt to calculate air time, however it doesn’t take into account hours -> minutes conversion. Let’s run our above minutes-from-midnight conversions and then calculate the differences.

mutate(flights,

arr\_time\_min = arr\_time %/% 100 \* 60 + arr\_time %% 100,

dep\_time\_min = dep\_time %/% 100 \* 60 + dep\_time %% 100,

new\_at\_min = arr\_time\_min - dep\_time\_min,

new\_at = new\_at\_min %/% 60\*100 + new\_at\_min %% 60) %>%

select(arr\_time, dep\_time, arr\_time\_min, dep\_time\_min, new\_at\_min, new\_at, air\_time, everything())

Seems even then that the math is right but the new air time is more than that listed in the original data. Perhaps the difference is in taxying and waiting from gate to runway.

1. **Compare dep\_time, sched\_dep\_time, and dep\_delay. How would you expect those three numbers to be related?**

I would expect dep\_delay to equal dep\_time - sched\_dep\_time (with hr/min conversions).

mutate(flights,

dep\_time\_min = dep\_time %/% 100 \* 60 + dep\_time %% 100,

sched\_dep\_time\_min = sched\_dep\_time %/% 100 \* 60 + sched\_dep\_time %% 100,

dep\_delay\_min = dep\_time\_min - sched\_dep\_time\_min,

# Why doesn't this work?

# new\_dep\_delay = (dep\_delay\_min %/% 60)\*100 + (dep\_delay\_min %% 60),

select(dep\_time, sched\_dep\_time, dep\_delay, dep\_delay\_min,

math\_test, dep\_time\_min, sched\_dep\_time\_min, everything())

Weird. This works, but converting negative times from minutes to hours/minutes doesn’t work right. Something to do with R and negative integers with %/%.

1. **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(flights, min\_rank(desc(flights$dep\_delay)))

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

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

Warning message:

In 1:3 + 1:10 :

longer object length is not a multiple of shorter object length

• It adds the values of the 3 positions in the first to the first three values in the corresponding positions of the second. If it were a multiple, R would add the first three positions of the first again to the fourth, fifth and sixth positions of the second, and so on.

1. **What trigonometric functions does R provide?**

cos(x), sin(x), tan(x), acos(x), asin(x), atan(x), atan2(y, x), cospi(x), sinpi(x), tanpi(x),

## 5.6.7  Exercises

1. **Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights. Consider the following scenarios:**

I’m not *entirely* sure what the author’s looking for here…giving it my best shot.

* + **A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.**

A closely-timed connecting flight periodically introduces a 30-minute delay. Get arrivals data for flights arriving in the 2 hours before.

Check flights data against weather data for wind speed differences. Easy first step would be to group by flight # and check air time variation.

Overbooking may cause delays as patrons are asked to take another flight. Find flight capacity info.

Time of day might have something to do with it if it’s route that flies more than once a day. Perhaps later flight times get delayed along with the rest of airport operations.

Older, crappier planes might cause delays. Examine delays by tail number to see if there’s a connection.

* + **A flight is always 10 minutes late.**

Perhaps that flight has to wait its turn for the runway behind another that leaves right before it. Check gap between earlier flight times against the mean gap between flight times.

Perhaps time estimates are based on a standard airspeed. If this flight goes west against the jet stream, it may have longer-than-predicted air times. Check distance / scheduled air time against other flights.

Could be the same old, issue-prone plane for every flight. Check tail number against delay times for other flight numbers.

Could be that the flight occurs at a particularly busy terminal or time.

Could be that the flight occurs late in the day when more airport operations are cumulatively behind schedule.

* + **A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.**

Check departure times against scheduled departure. Is the discrepancy happening there or later in the flight?

Check against earlier arrivals for connecting flights that cause delays.

Check departure times—more than one per day? Is there a cumulative delay effect throughout the day?

Are there two planes that service the route? Maybe one tends to experience higher delays.

Does the flight always leave from the same terminal?

* + **99% of the time a flight is on time. 1% of the time it’s 2 hours late.**

Are the delays seasonal? Perhaps caused by snow storms. Check against weather data.

Check against a plot of airport-wide delay times over days of the year. There may be a decommissioned flight strip or other overarching problem.

This might be a fluke if it’s only one or two instances. Check total number of flights in the year. Might not be any value in determining cause if it’s less than a few hundred.

Check whether the bulk of delay occurs before or after departure.

Check air times on late days. Perhaps it flies a route that occasionally experiences stronger-than-average headwinds.

Failing these methods, I would try to access non-numerical information about this case. Are there maintenance records?

**Which is more important: arrival delay or departure delay?**

Arrival delays are more problematic because travelers miss connections.

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) %>%

summarise(n = n())

not\_cancelled %>% group\_by(tailnum) %>%

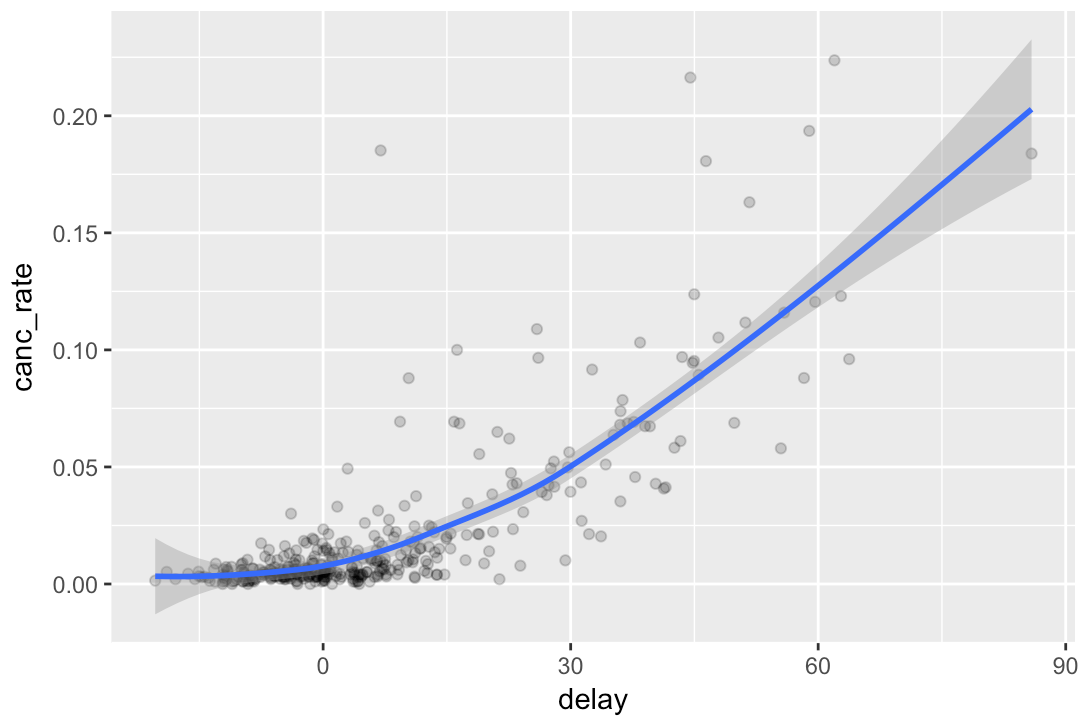
summarise(n = sum(distance))

1. **Our definition of cancelled flights (is.na(dep\_delay) | is.na(arr\_delay) ) is slightly suboptimal. Why? Which is the most important column?**

Although arrival delays are more important for traveler connections, departure delays would seem to give a clearer view of whether the flight took off. There are 1175 flights with departure delay values but no arrival delay values. There are no flights without departure delays that nevertheless have arrival delays. This suggests to me that those flights were re-routed midflight and thus not canceled.

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

Sure is. Cancellation rate is positively correlated with average delay:



by\_day <- group\_by(flights, month, day)

canc\_prop <- summarise(by\_day,

count = n(),

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

canc\_rate = cancels / count,

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

)

canc\_prop <- filter(canc\_prop, canc\_rate < .3)

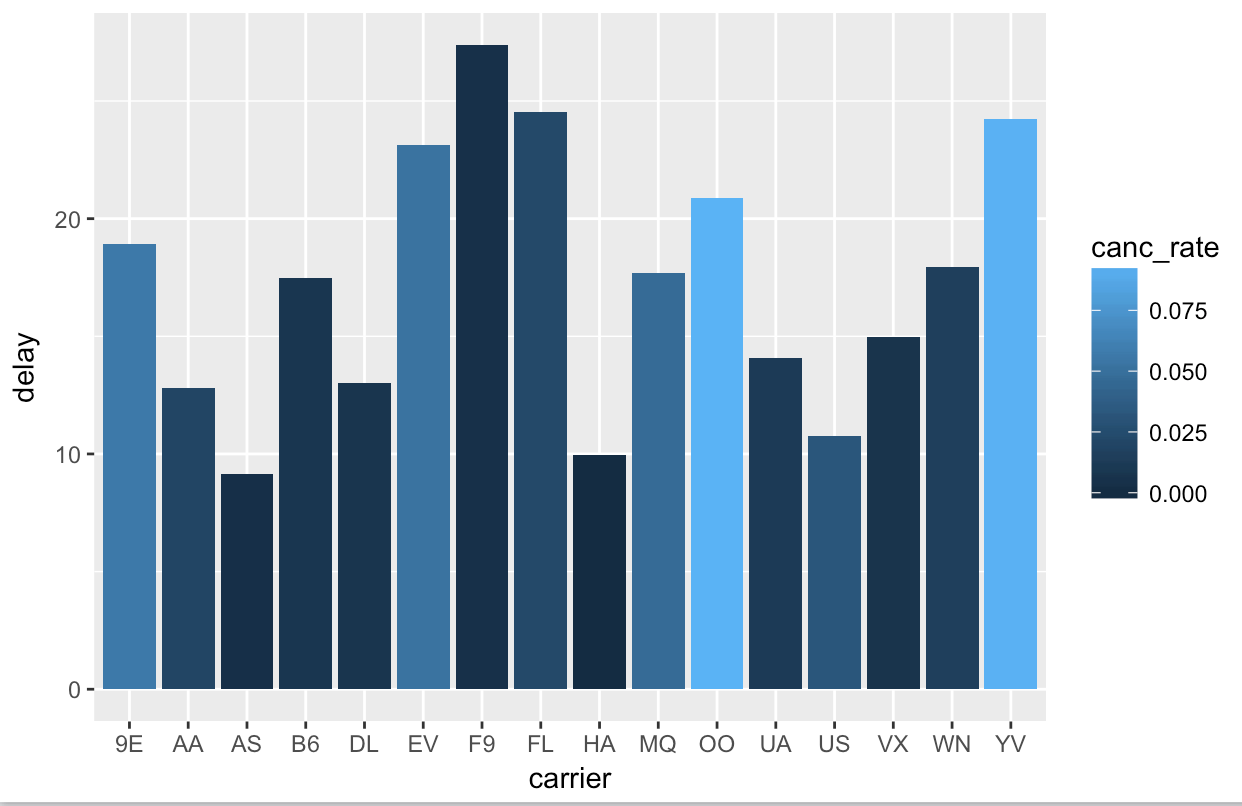
ggplot(data = canc\_prop, mapping = aes(x = delay, y = canc\_rate)) +

geom\_point(alpha = 1/5) +

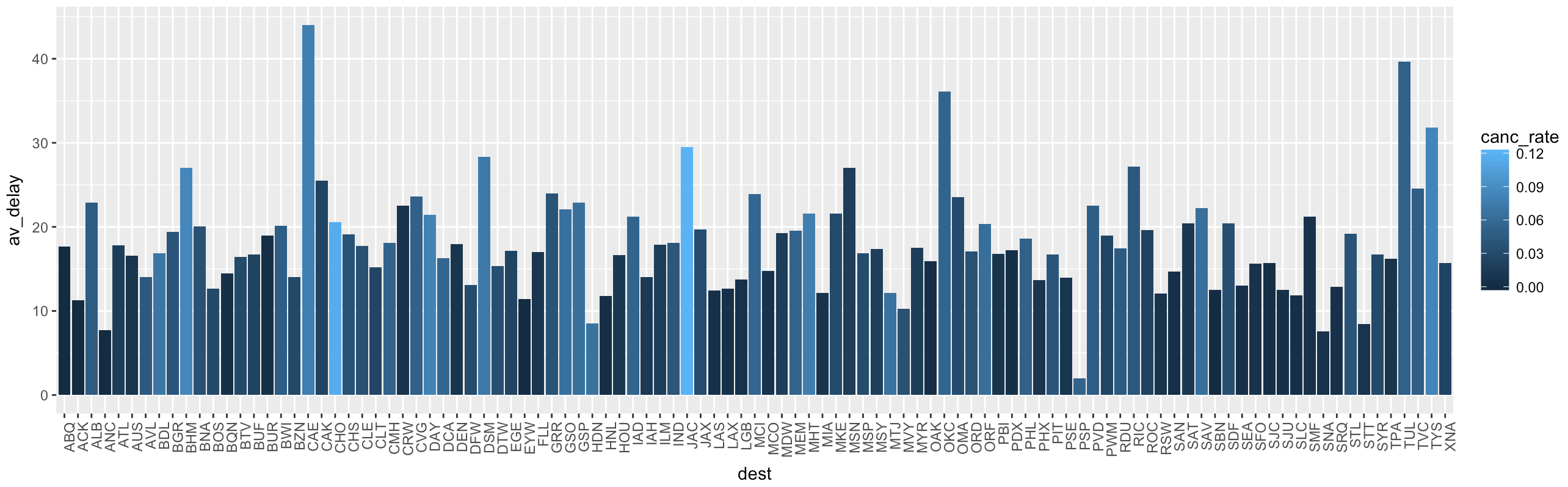
geom\_smooth( na.rm = TRUE)

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()))**

Frontier has the worst average non-negative delays (because being early on some flights doesn’t mitigate being late on others): 27.3 minutes.



This is as close as I was able to get to answering airline vs. airport being fault for delays: a bar chart of delays by airport. A first glance shows us that Columbia, SC airport has the worst delays (go figure), and that except for a few extremes, both airports and airlines experience similar variation in average delay time.



We could explore further by rendering 16 of the preceding graph by carrier. We could create grades (0-1) for airports and then calculate airport-related delay scores for each airline by computing weighted averages of # of flights to each airport \* airport grade. We could then compare that score to the airline’s overall lateness, as indicated above. High correlation would suggest more airport responsibility, while low correlation would put the airlines at fault. We could also do the process in reverse, starting by assigning airline grades.

#-----------------------------------

# this chunk generates bar chart of avg delays by carrier

by\_carrier <- flights

#filter out negative delays

by\_carrier <- mutate(by\_carrier, arr\_delay = ifelse(arr\_delay >= 0, arr\_delay, 0))

by\_carrier <- group\_by(by\_carrier, carrier)

canc\_prop <- summarise(by\_carrier,

count = n(),

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

canc\_rate = cancels / count,

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

ggplot(canc\_prop, aes(carrier, delay)) +

geom\_col(aes(fill = canc\_rate))

#-------------------------

# Who's most responsible for the delays? Carriers or airports?

pos\_delays <- flights

#get rid of negative arrival delays

pos\_delays <- mutate(pos\_delays, arr\_delay = ifelse(arr\_delay >= 0, arr\_delay, 0))

#find av\_delay by airport

pos\_delays <- group\_by(pos\_delays, dest)

pos\_delays <- summarise(pos\_delays, count = n(),

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

canc\_rate = canceled / count,

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

#sort with highest delays & cancel rates first

(pos\_delays <- arrange(pos\_delays, desc(av\_delay), desc(canc\_rate)))

#remove rows w/ small counts

(pos\_delays <- filter(pos\_delays, count > 4))

#now we want to see airports w/ highest avg delays

p <- ggplot(pos\_delays, aes(dest, av\_delay)) +

geom\_col(aes(fill = canc\_rate))

p + theme(axis.text.x = element\_text(angle = 90, hjust = 1))

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

if TRUE will sort output in descending order of n

Would be helpful when we want to see which category has the most observations of something—cancellations, for example.

## 5.7.1 Exercises

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

This question isn’t clear—but generally, grouping will cause the functions for every row in a group bucket and then start over on the next bucket.

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

pos\_delays <- flights %>%

group\_by(tailnum) %>%

#get rid of small samples

filter(n()>5) %>%

#replace negative arrival delays w/ 0

mutate(arr\_delay = ifelse(arr\_delay >= 0, arr\_delay, 0)) %>%

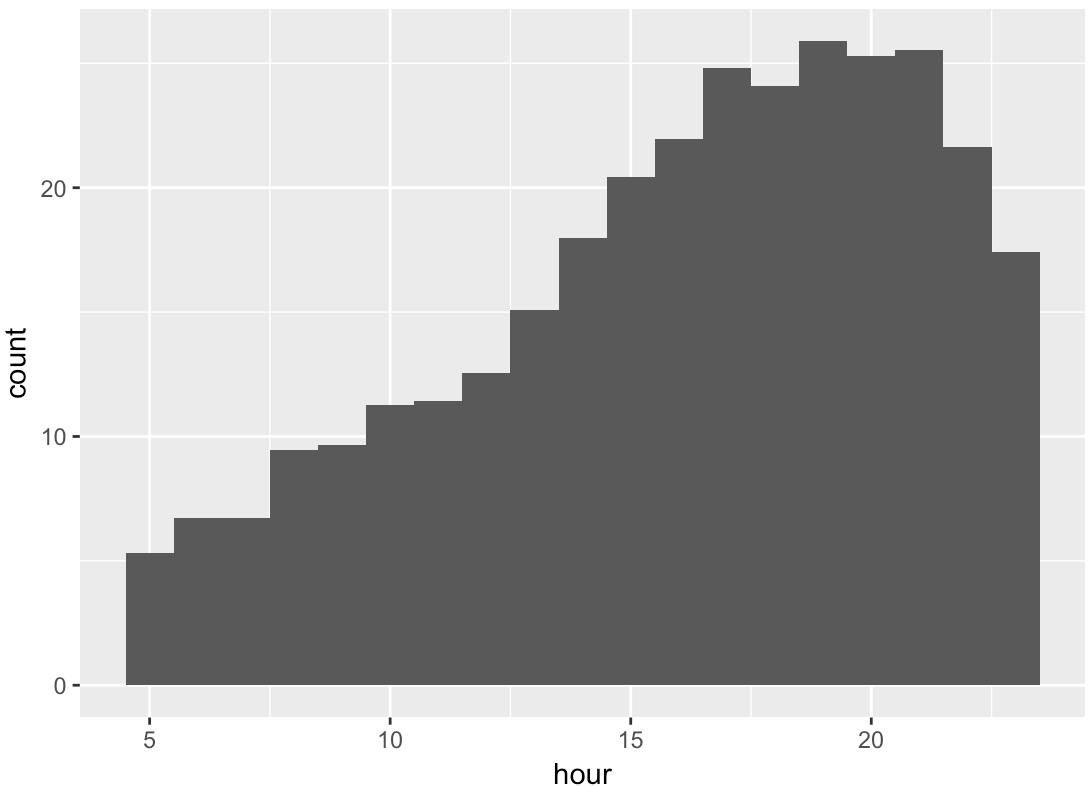
summarise(av\_delay = mean(arr\_delay)) %>%

arrange(rank(desc(av\_delay)))

The plane with the worst delay times and more than 5 flights in the year was N665MQ. The overall worst was N844MH, but it was for a single, 320-minute delayed flight.

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

5 a.m.



hour av\_delay

<dbl> <dbl>

1 19 25.903886

2 21 25.542226

3 20 25.303157

4 17 24.820890

5 18 24.081767

6 16 21.965298

7 22 21.641517

8 15 20.421801

9 14 17.979260

10 23 17.434741

11 13 15.103613

12 12 12.570334

13 11 11.432596

14 10 11.270434

15 9 9.656816

16 8 9.447670

17 7 6.721201

18 6 6.706409

19 5 5.320619

pos\_delays <- flights %>%

group\_by(hour) %>%

#get rid of small samples

filter(n()>1) %>%

#replace negative arrival delays w/ 0

mutate(arr\_delay = ifelse(arr\_delay >= 0, arr\_delay, 0)) %>%

summarise(av\_delay = mean(arr\_delay, na.rm = TRUE)) %>%

#rank hour of day from highest av\_delay to lowest

arrange(rank(desc(av\_delay)))

#print list

pos\_delays

#plot histogram

ggplot(pos\_delays, aes(hour)) +

geom\_histogram(aes(weight = av\_delay), binwidth = 1)

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

#find total delay by dest

pos\_delays <- flights %>%

group\_by(dest) %>%

#get rid of small samples

filter(n()>15) %>%

#sum arrival and departure delay for total delay

mutate(tot\_delay = arr\_delay + dep\_delay,

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

#compute average delay by grouped variable, dest

summarise(tot\_delay = mean(tot\_delay, na.rm = TRUE))

#print list

pos\_delays

**#PART 2**

#find proportion of arrival delay to total delay by flight

pos\_delays <- flights %>%

group\_by(flight) %>%

#get rid of small samples

filter(n()>5) %>%

#replace arrival delays w/ 0

mutate(arr\_delay = ifelse(arr\_delay >= 0, arr\_delay, 0),

dep\_delay = ifelse(dep\_delay >= 0, arr\_delay, 0)) %>%

#sum arrival and departure delay for total delay

mutate(tot\_delay = arr\_delay + dep\_delay,

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

#compute average delay by grouped variable, dest

summarise(tot\_delay = mean(tot\_delay, na.rm = TRUE),

prop\_delay = mean(prop\_delay, na.rm = TRUE))

#print list

pos\_delays

flight tot\_delay prop\_delay

<int> <dbl> <dbl>

1 1 18.175036 0.6603053

2 2 7.686275 0.7500000

3 3 15.659236 0.7014218

4 4 17.127877 0.6151079

5 5 15.743827 0.7111111

6 6 17.315534 0.5925926

7 7 33.813559 0.5989011

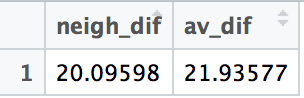
8 8 18.713675 0.5808824

9 9 38.973684 0.6168831

10 10 53.442623 0.5600000

# ... with 3,041 more rows

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.**

Easy way to do this: calculate a year-average positive departure delay time. Use lag to show: 1. the difference between a flight’s non-negative departure delay and the year average and 2. the difference between the flight and its neighbor. I expect to find that neighbors are much closer to each other than to the average. We can average these measurements over the year to get: 

pos\_delays <- flights

#replace negative dep delays w/ 0

pos\_delays <- mutate(flights, dep\_delay = ifelse(dep\_delay >= 0, dep\_delay, 0)) %>%

select(-(carrier:time\_hour))

#add year\_avg, neigh\_dif, av\_dif columns

year\_av <- summarize(pos\_delays, av\_delay = mean(dep\_delay, na.rm = TRUE))

pos\_delays <- mutate(pos\_delays,

year\_avg = year\_av[[1,1]],

#subtract predecessor's delay from this row's delay

neigh\_dif = dep\_delay - lag(dep\_delay),

av\_dif = dep\_delay - year\_avg

)

#average neighbor difference and year difference

View(summarize(pos\_delays, neigh\_dif = mean(abs(neigh\_dif), na.rm = TRUE), av\_dif = mean(abs(av\_dif), na.rm = TRUE)))

If we also wanted to demonstrate that delay times are more likely experience a sharp increase than a sharp decrease, then we could use lead() and lag() to created year-averaged delay\_before and delay\_after variables. Prediction is that delay\_before would be lower. This step wouldn’t really be necessary since our histogram breakdown of average delay by hour of day tells this story pretty well already.

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?**

Boston has the shortest flight time at 54% air time of the average, although I’m not sure that’s an erroneous value. Other short flights have similar proportions of average air time.  
  
Boston has the most delayed air\_time at 533% of its shortest (although we’ve shown that the short air time is the shortest proportional to average of all destinations--)

DC and Akron also have long in-air delays, around 400%.

#-------  
# Airtime variance  
# 1 Look for short-duration airtime outliers by destination  
# 2 Calculate a proportion of shortest flight time it takes  
# other flights to arrive at same dest. (E.G. 1.25 -- 25% longer than shortest.)  
# 3 Which flights are proportionally slowest of their dest group?  
library(scales)  
  
flight\_time <- flights %>%  
#let's put flights in dest- and air time-order and remove cancellations / re-routes  
arrange(dest, air\_time) %>%  
filter(!(is.na(dep\_delay)) & !(is.na(arr\_delay)))  
#let's add some useful columns  
flight\_time <- mutate(flight\_time,  
 #convert some times to minutes  
 arr\_time\_min = arr\_time %/% 100 \* 60 + arr\_time %% 100,  
 dep\_time\_min = dep\_time %/% 100 \* 60 + dep\_time %% 100,  
 sched\_arr\_time\_min = arr\_time %/% 100 \* 60 + arr\_time %% 100,  
 sched\_dep\_time\_min = dep\_time %/% 100 \* 60 + dep\_time %% 100,  
 #calculate gate-to-gate, scheduled air time, difference from scheduled air time, & speed  
 dep\_to\_arr = arr\_time\_min - dep\_time\_min,  
 sched\_at = sched\_arr\_time\_min - sched\_dep\_time\_min,  
 at\_dif = air\_time - sched\_at,  
 at\_skew = air\_time / sched\_at,  
 speed = distance / air\_time \* 60  
 ) %>%  
  
#move irrelevant columns right  
select(dest, dep\_to\_arr, air\_time, sched\_at, at\_dif, at\_skew,  
 speed, dep\_time, arr\_time, distance, everything()) %>%  
  
  
#let's find the shortest flight times by dest  
group\_by(dest) %>%  
summarise( ct=n(),  
 shortest = min(air\_time, na.rm = TRUE),  
 longest = max(air\_time, na.rm = TRUE),  
 #let's add some useful columns by group  
 avg\_airtime = mean(air\_time, na.rm = TRUE),  
 short\_airtime = min(air\_time, na.rm = TRUE),  
 prop\_of\_avg = percent(short\_airtime / avg\_airtime),  
 prop\_of\_shortest = percent(longest / short\_airtime)) %>%  
  
arrange(desc(prop\_of\_shortest))   
  
flight\_time

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

carrier\_num <- flights %>%  
 select(dest, carrier, everything()) %>%  
 group\_by(dest) %>%  
 summarize(ncarrier = n()) %>%  
 arrange(ncarrier) %>%  
 filter(ncarrier >= 2)  
  
carrier\_num  
  
  
 dest ncarrier  
 <chr> <int>  
 1 ANC 8  
 2 SBN 10  
 3 HDN 15  
 4 MTJ 15  
 5 EYW 17  
 6 PSP 19  
 7 JAC 25  
 8 BZN 36  
 9 CHO 52  
10 MYR 59  
# ... with 93 more rows

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

#-------------------------------

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

# group by tail num

# filter by <= 1 hr

# arrange by delay

tails <- flights

tails <- filter(tails, arr\_delay <= 60)

tails <- arrange(tails, desc(arr\_delay))

tails <- group\_by(tails, tailnum)

tails <- summarise(tails, nflights = n())

tails <- arrange(tails, nflights)

tails

# A tibble: 4,014 x 2

tailnum nflights

<chr> <int>

1 N137DL 1

2 N143DA 1

3 N14628 1

4 N152DL 1

5 N162PQ 1

6 N206UA 1

7 N228UA 1

8 N26906 1

9 N27901 1

10 N287AT 1

# ... with 4,004 more rows

## 6.3 Practice

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!**

Found and installed this nifty addin to automatically format text into Markdown with key commands and a dropdown from the Addins menu: <https://github.com/ThinkR-open/remedy>

1. **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.**

pretty whitespace useage, unused variables, diagnostics for other languages, etc.