Group 04 Project 1 - Data Carpentry & Visualization Basics

Stephan Quintin, Ryo Iwata, Zachary Strickland

Group 4 members:

names here Stephan Quintin, Ryo Iwata, Zachary Strickland Note who is assigned leader: Stephan Quintin

Group Assignment 1

Instructions - Everyone should work on their own computer on a separate copy of this, then work together to create a single very well polished version to submit for the assignment. The assigned group leader should submit the assignment for the group, but every individual should thoroughly understand every answer. In most cases, everyone should roughly be on the same question as the same time when working together simultaneously (e.g., in class or during group meetings). When asked, you (all) need to be ready to explain the thought process behind the plan, approach, revisions, and final code that was written. Now is the time to build your understanding of the language and approach to these data carpentry and visualization methods. My expectation is that every submitted assignment will be nearly perfect given how we will be approaching this series of problems both in class and out.

I have copied together all the questions in one place so everyone doesn't have to waste their time doing so, but these are directly from the reading (R4DS 2E). I am leaving the formatting plain, so refer to the book itself for the example plots and pretty formatting.

Please make it easy to grade this! Your text answers should be in the main document (i.e., not in a code chunk), and *please make them bold so they stand out.* Write code, including using comments, as if you are writing code that will be published. Build good habits now!

1.2.5

1

How many rows are in penguins? How many columns?

Answer: 344 rows and 8 columns

```
# Looks at the structure of penguins
str(penguins)
```

```
## tibble [344 x 8] (S3: tbl_df/tbl/data.frame)
##  $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
##  $ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
##  $ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
##  $ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

2

What does the bill_depth_mm variable in the penguins data frame describe? Read the help for ?penguins to find out.

Answer: a number denoting bill depth (millimeters)

```
# ?penguins
```

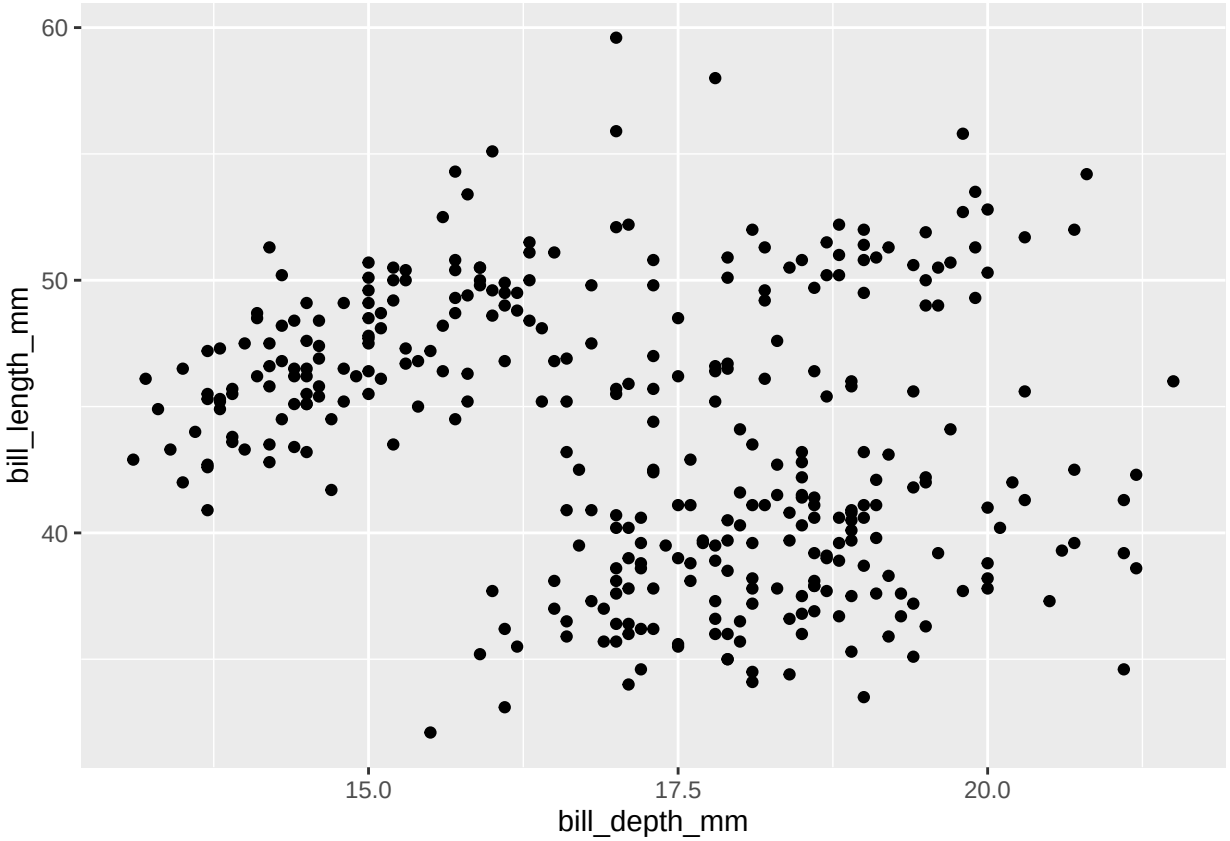
3

Make a scatterplot of bill_depth_mm vs. bill_length_mm. That is, make a scatterplot with bill_depth_mm on the y-axis and bill_length_mm on the x-axis. Describe the relationship between these two variables.

No meaningful visual relationship, there might be some clusters

```
ggplot(penguins, aes(bill_depth_mm, bill_length_mm)) + geom_point()
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



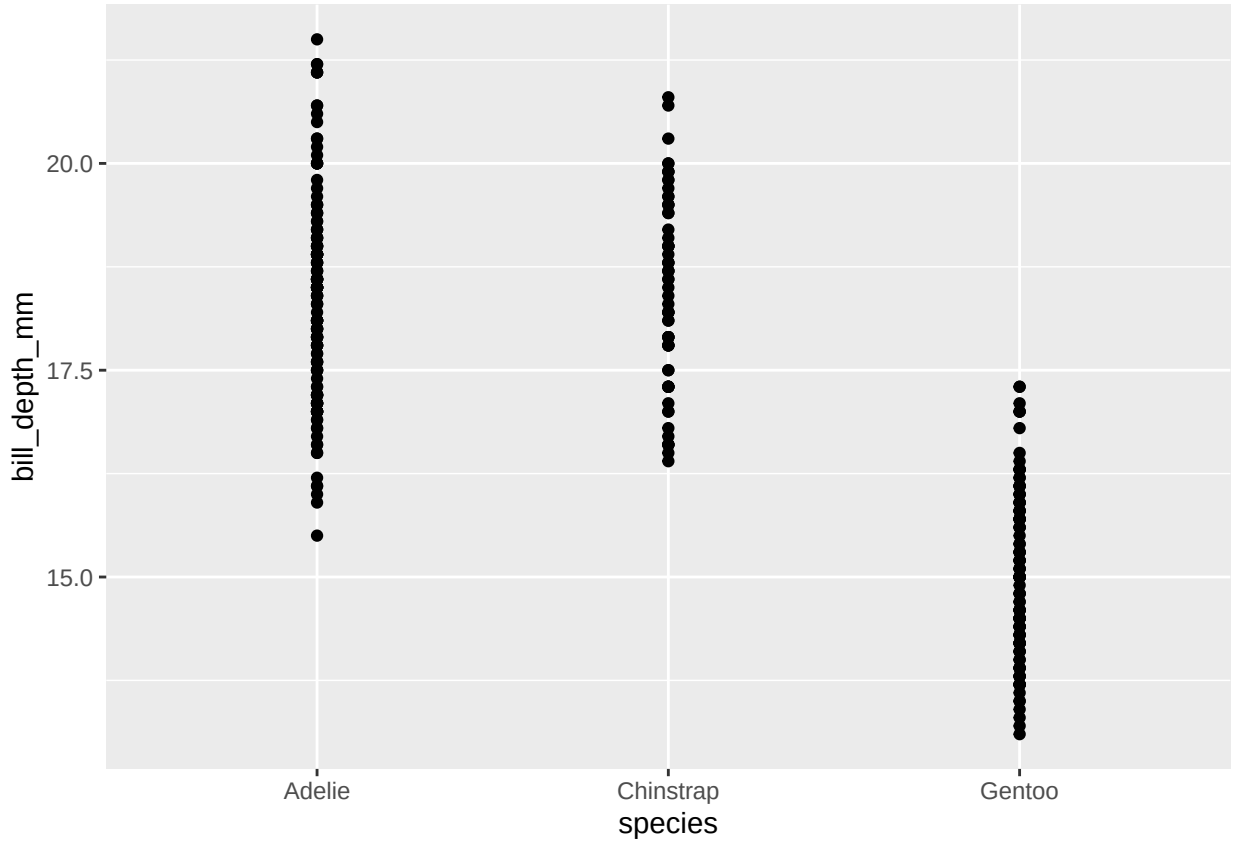
4

What happens if you make a scatterplot of species vs. bill_depth_mm? What might be a better choice of geom?

It creates a scatter plot for each species. geom_violin could better show the distribution

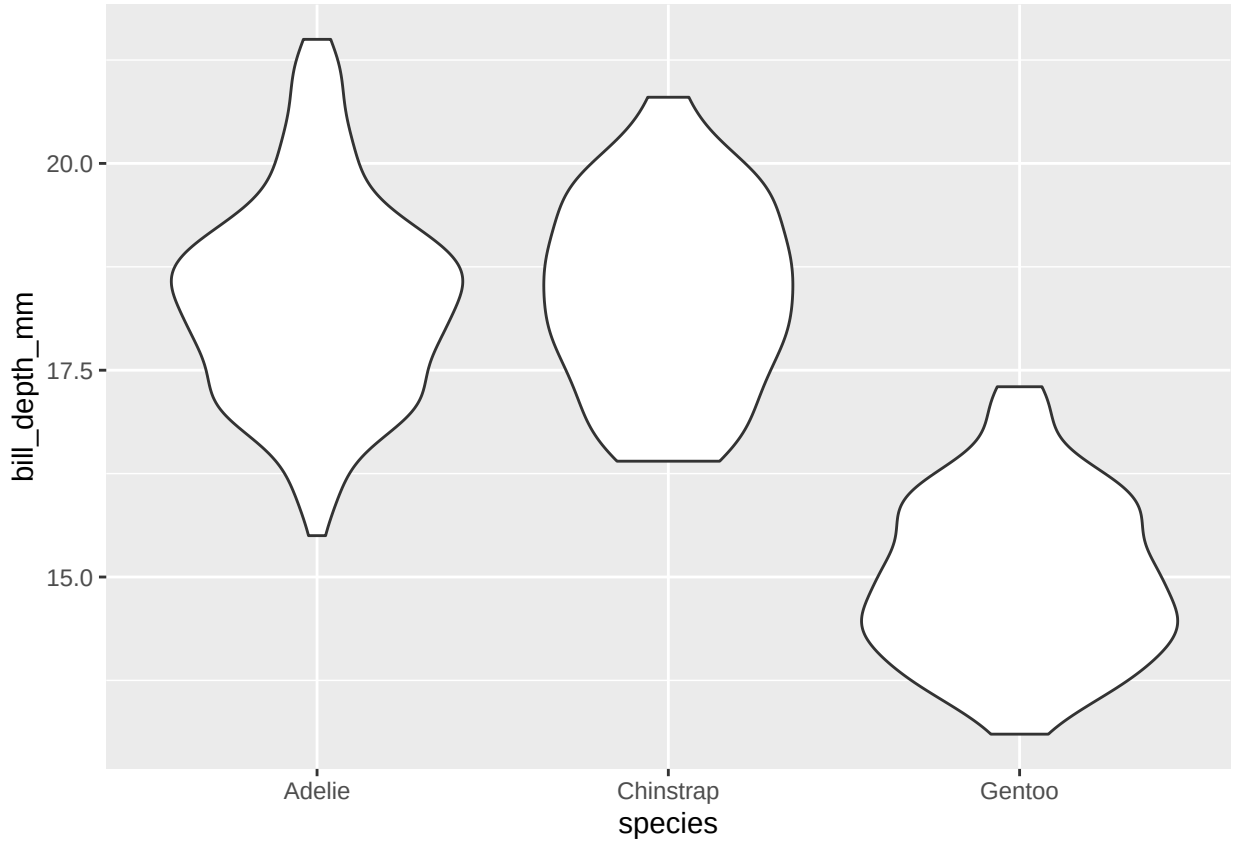
```
ggplot(penguins, aes(species, bill_depth_mm)) + geom_point()
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```




```
ggplot(penguins, aes(species, bill_depth_mm)) + geom_violin()
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_ydensity()`).
```



5

Why does the following give an error and how would you fix it?

X and Y arguments are missing in the aesthetic function. We would fix this by defining X and Y in the aesthetic function

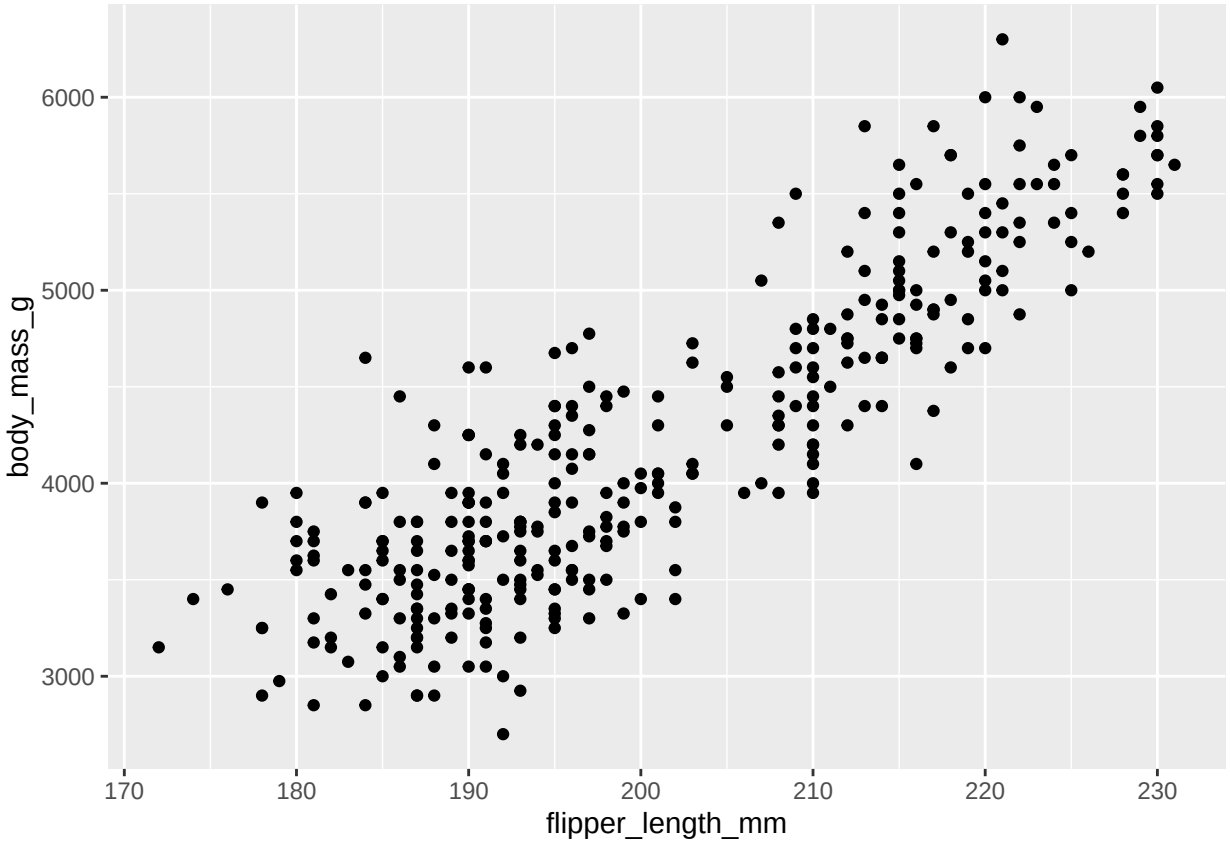
```
# ggplot(data = penguins, aes()) +
#  geom_point()
```

6

What does the na.rm argument do in geom_point()? What is the default value of the argument? Create a scatterplot where you successfully use this argument set to TRUE.

*na.rm is whether or not there is a warning of missing data points. Default value is FALSE which shows a warning.

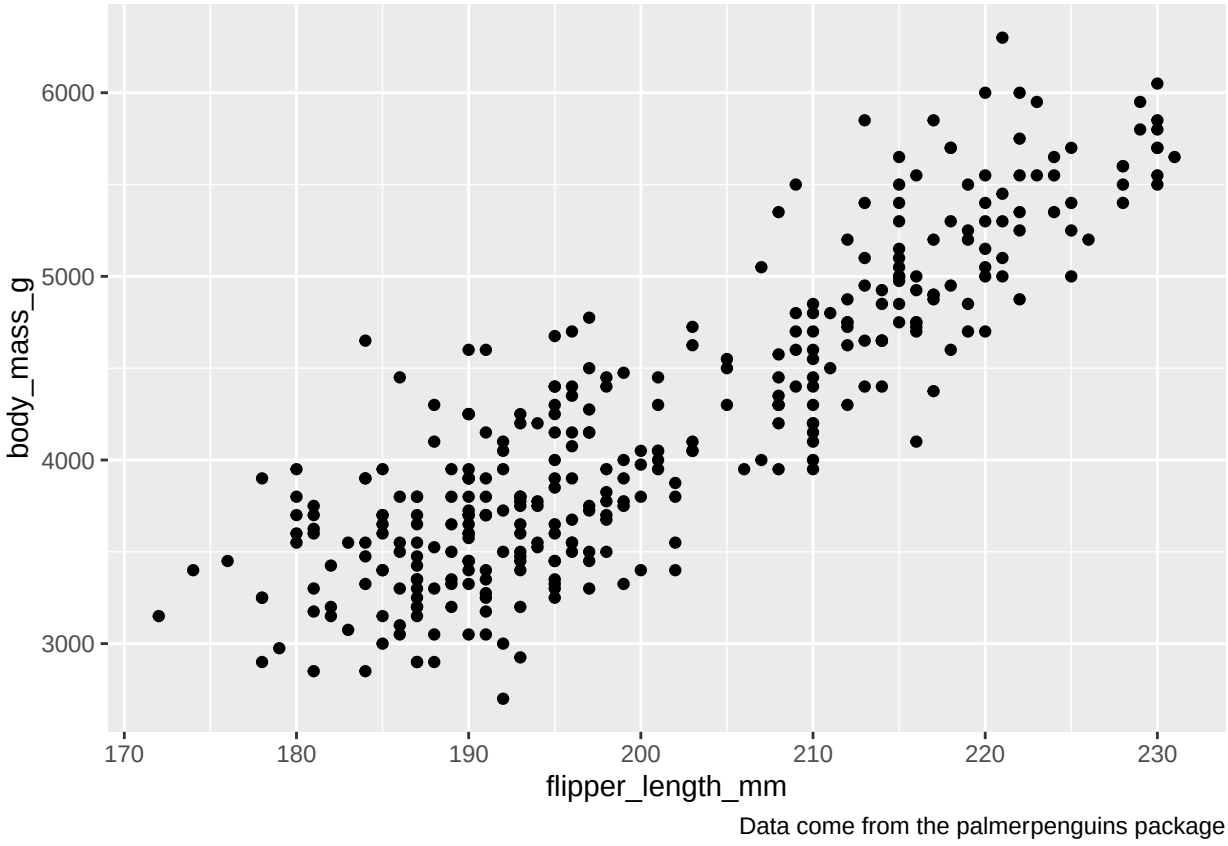
```
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) + geom_point(na.rm = TRUE,)
```



7

Add the following caption to the plot you made in the previous exercise: "Data come from the palmerpenguins package." Hint: Take a look at the documentation for labs().

```
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) + geom_point(na.rm = TRUE,) + labs(caption="Data
```



8

Recreate the following visualization. What aesthetic should bill_depth_mm be mapped to? And should it be mapped at the global level or at the geom level?

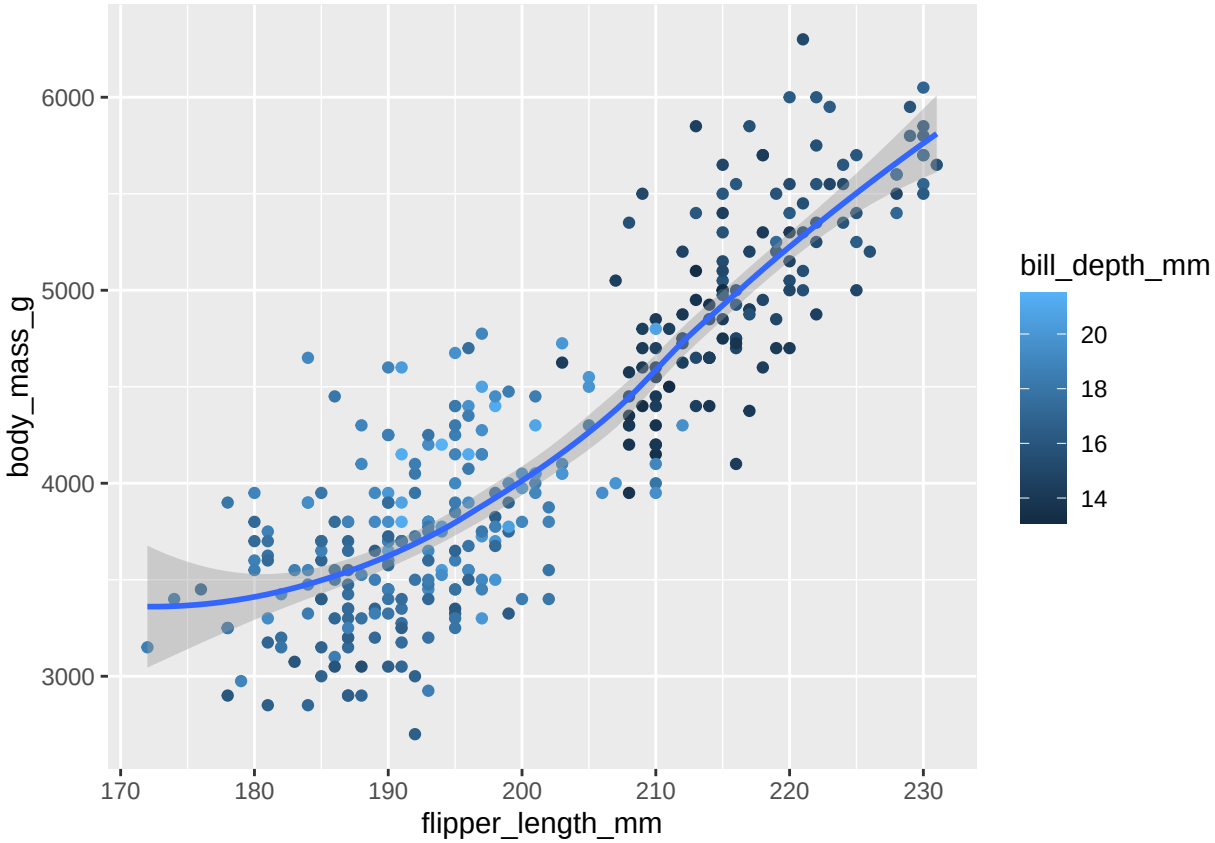
```
ggplot(penguins, aes(flipper_length_mm, body_mass_g, colour=bill_depth_mm)) + geom_point() +  geom_smoo
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: The following aesthetics were dropped during statistical transformation: colour
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



9

Run this code in your head and predict what the output will look like.

ggplot(data = penguins, mapping = aes(x = flipper_length_mm, y = body_mass_g, color = island)) + geom_point() + geom_smooth(se = FALSE)

**Answer: It will be a scatter plot with flipper length as the x, body mass as the y, and each point is colored based on the associated island. A smoothed line that best passes through the points going from the start to end of x will be drawn.There will be one line for each island that the penguins came from. Because `se` is `FALSE`, there will be no confidence interval drawn.

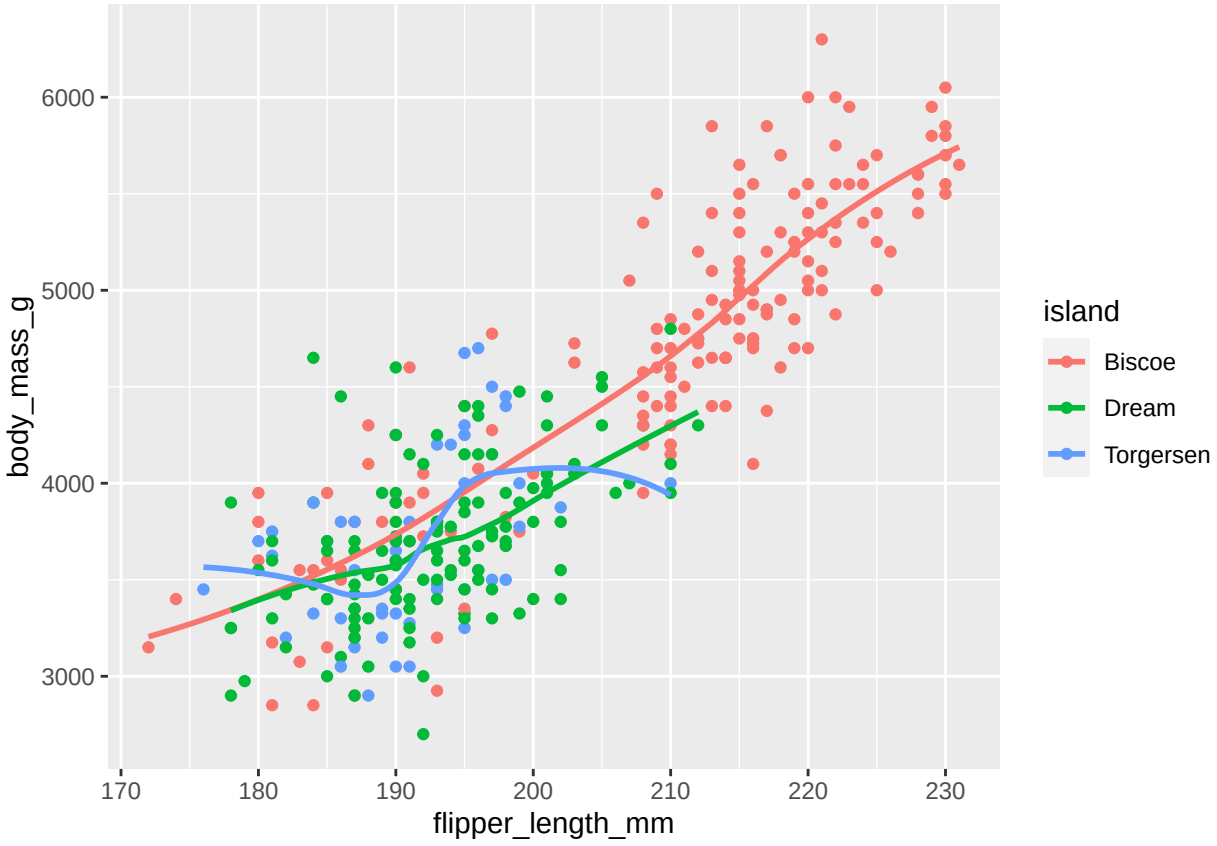
9b

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = island)
) +
  geom_point() +
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



10

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

** Answer: The two graphs will look the same. This is because both sections of code are achieving the same thing in different ways. The first is setting the data and mapping for the entire plot with the `ggplot` function. `geom_point() and geom_smooth()` inherit this data and mapping because they have no arguments. The second calls `ggplot` without any arguments, so it creates an empty plot. `geom_point() and geom_smooth()` both have the data and mapping as arguments that's identical to that of the first section, so the plot will be identical. **

ggplot(data = penguins, mapping = aes(x = flipper_length_mm, y = body_mass_g)) + geom_point() + geom_smooth()

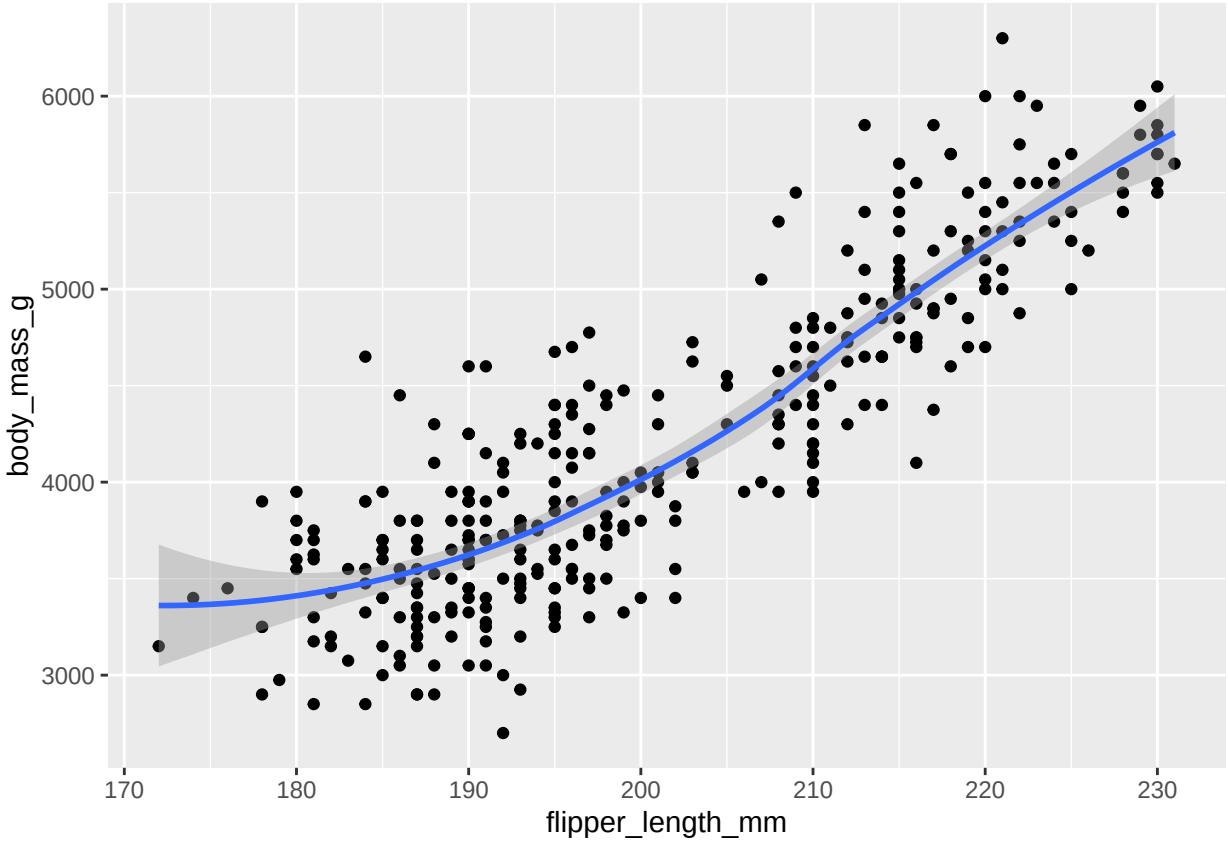
ggplot() + geom_point(data = penguins, mapping = aes(x = flipper_length_mm, y = body_mass_g)) + geom_smooth(data = penguins, mapping = aes(x = flipper_length_mm, y = body_mass_g))

10b confirm your answer:

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point() +
  geom_smooth()
```

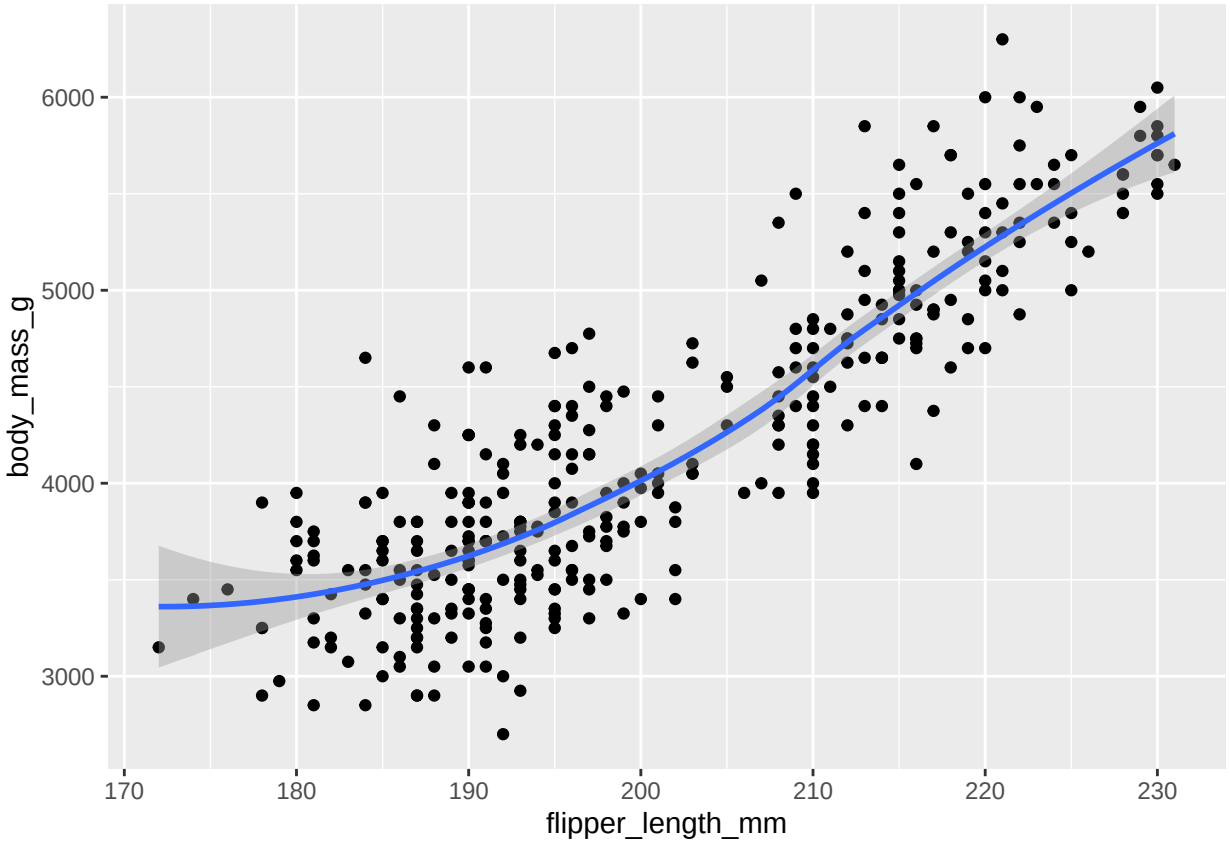
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```




```
ggplot() +
  geom_point(
    data = penguins,
    mapping = aes(x = flipper_length_mm, y = body_mass_g)
  ) +
  geom_smooth(
    data = penguins,
    mapping = aes(x = flipper_length_mm, y = body_mass_g)
  )
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
## Removed 2 rows containing missing values (`geom_point()`).
```


body_mass_g
6000
5000
4000
3000
170
180
190
200
210
220
230
flipper_length_mm

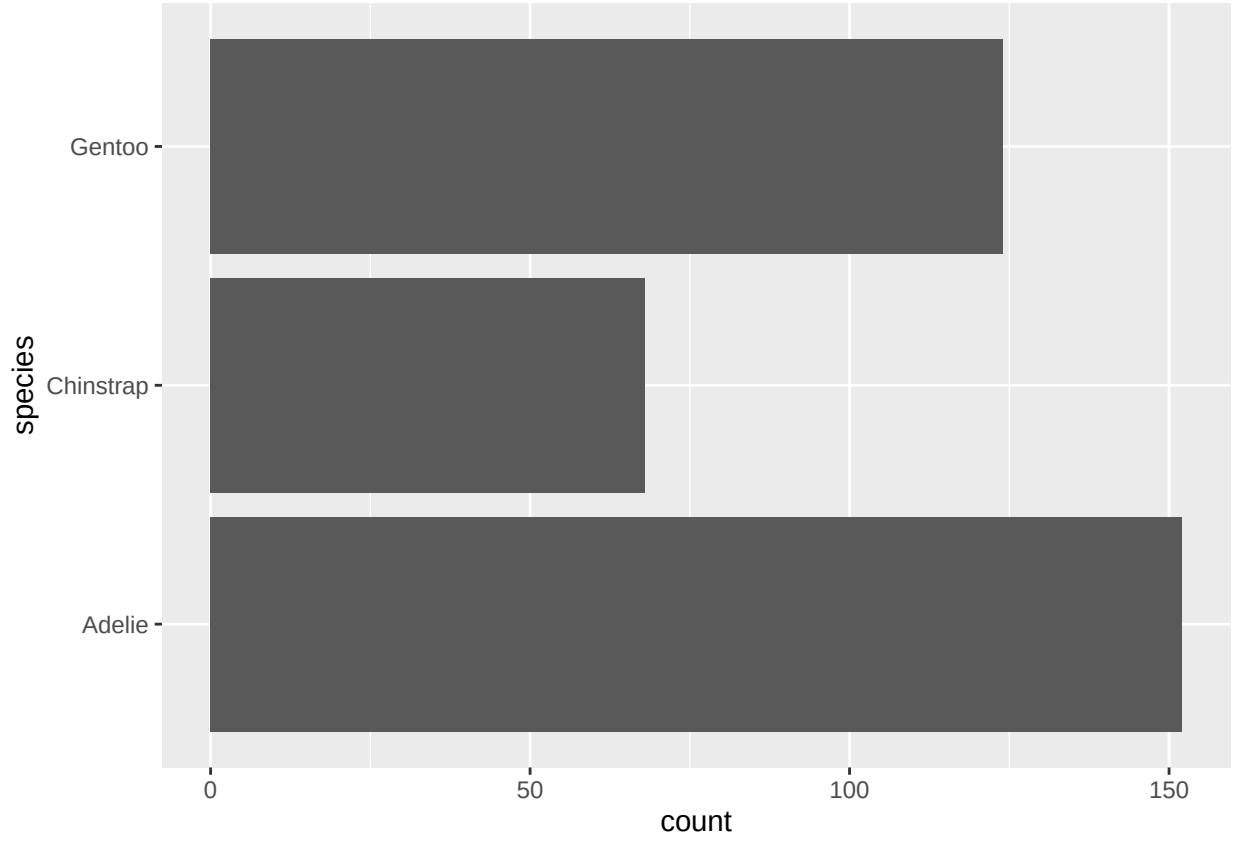
1.4.3 Exercises

1

Make a bar plot of species of penguins, where you assign species to the y aesthetic. How is this plot different?

** Answer: This plot is different because it only requires 1 variable as an argument. It uses the count of data points that are within a given category as the x-value.**

```
ggplot(
  data = penguins,
  mapping = aes(y = species)
) +
  geom_bar()
```



2

How are the following two plots different? Which aesthetic, color or fill, is more useful for changing the color of bars?

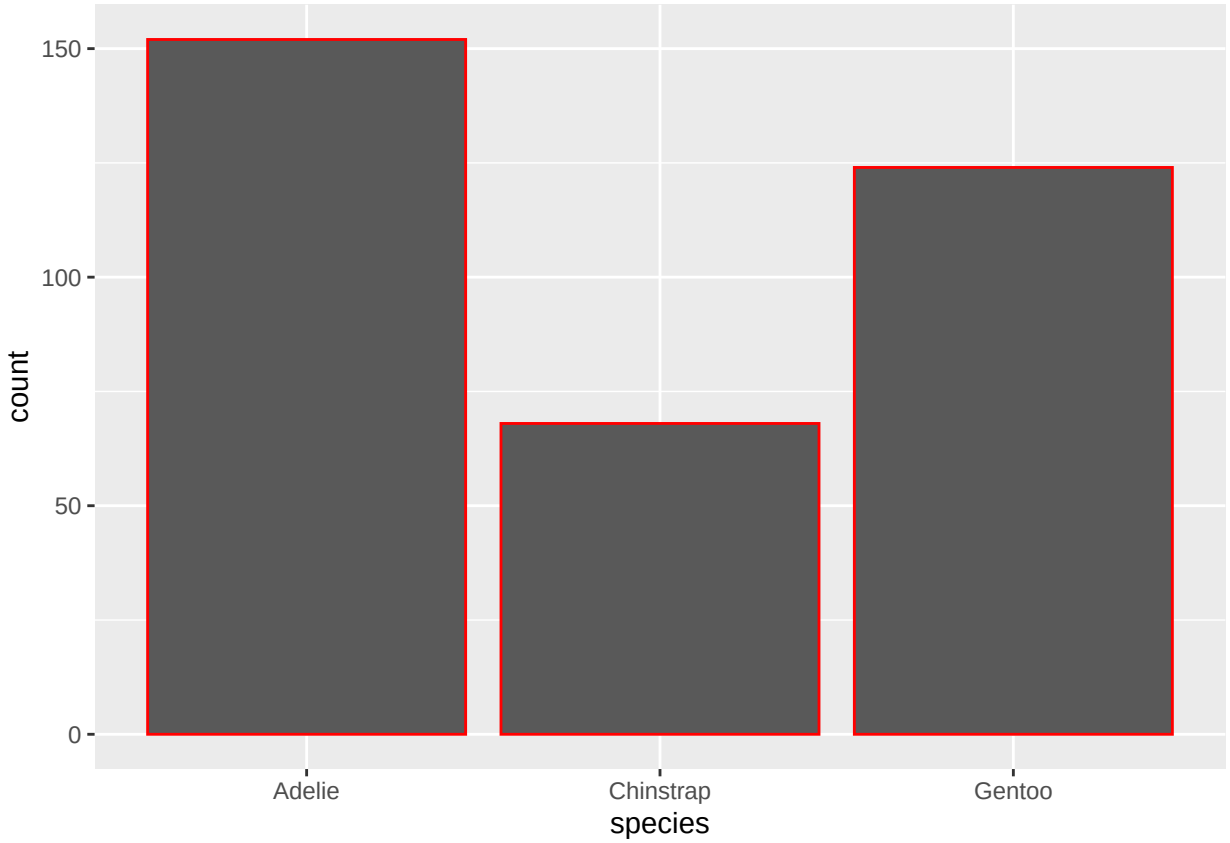
```
ggplot(penguins, aes(x = species)) +
  geom_bar(color = "red")

ggplot(penguins, aes(x = species)) +
  geom_bar(fill = "red")
```

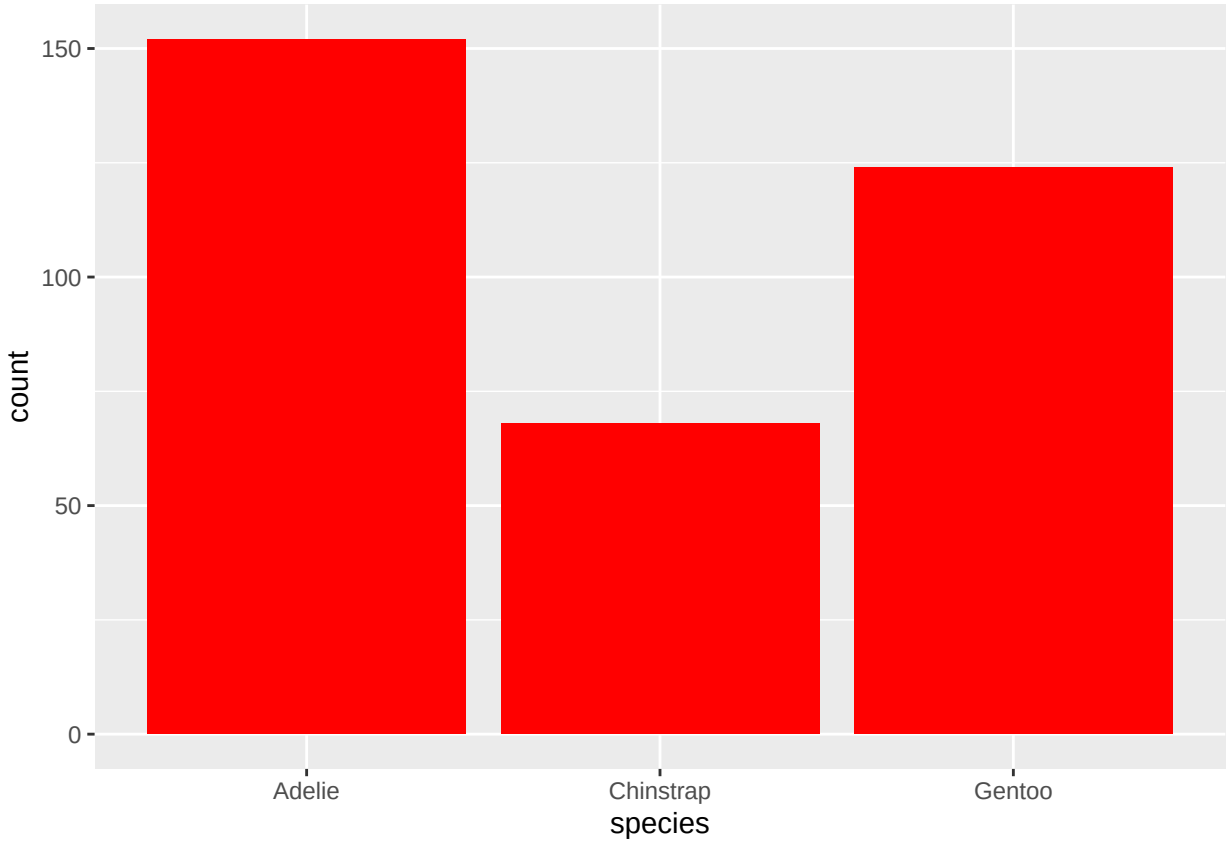
** Answer: The first section only changes the outline of the bars to red. The second changes the color of the insides of the bar. The second if more useful because the outline is small and ususally the focus of labeling

for a plot.

```
ggplot(penguins, aes(x = species)) +
  geom_bar(color = "red")
```




```
ggplot(penguins, aes(x = species)) +
  geom_bar(fill = "red")
```



3

What does the bins argument in geom_histogram() do?

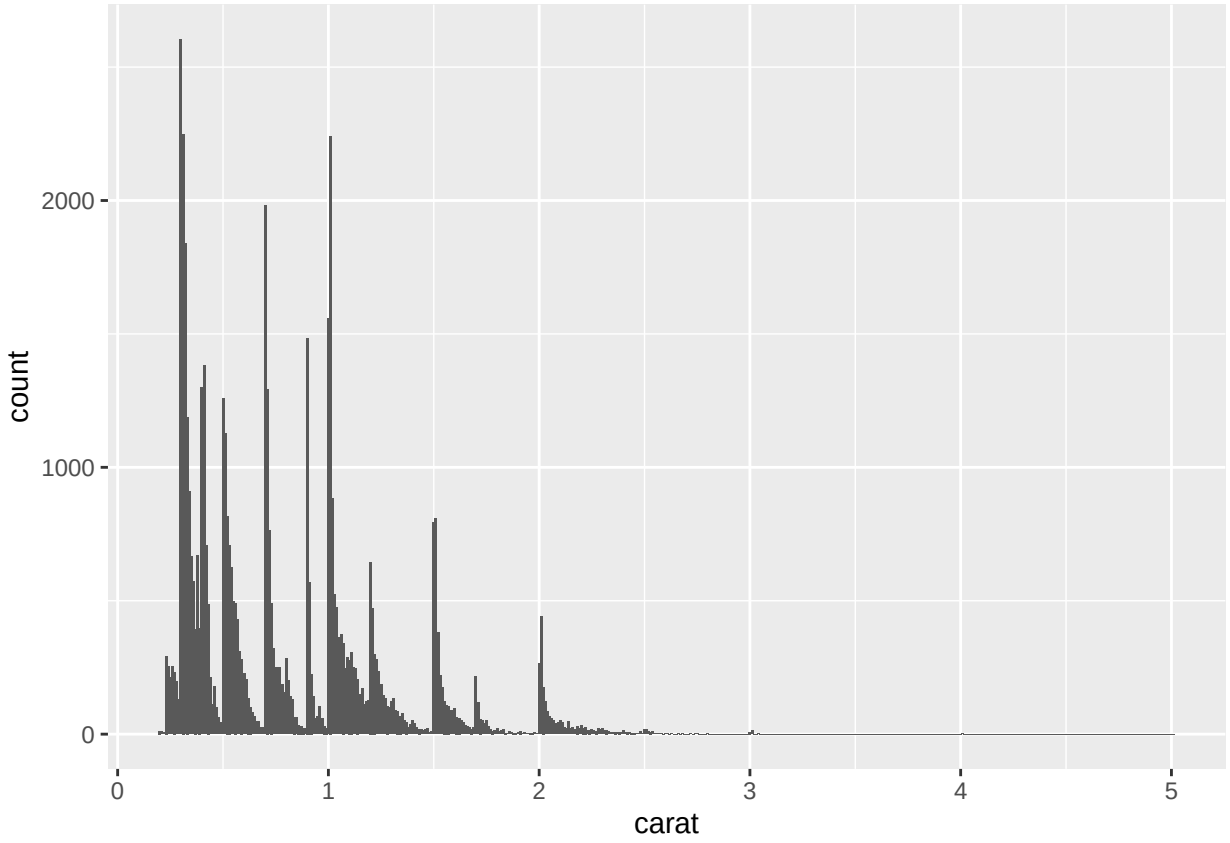
** The bins argument determines the number of bins in the histogram. **

4

Make a histogram of the carat variable in the diamonds dataset that is available when you load the tidyverse package. Experiment with different binwidths. What binwidth reveals the most interesting patterns?

** Answer: Setting the binwidth to 0.01 shows that there are sudden jumps in count in carats that are divisible by 0.25. This may reflect carats being rounded or cut to this size. **

```
ggplot(diamonds, aes(carat)) +
  geom_histogram(binwidth = 0.01)
```



1.5.5 Exercises

1

The mpg data frame that is bundled with the ggplot2 package contains 234 observations collected by the US Environmental Protection Agency on 38 car models. Which variables in mpg are categorical? Which variables are numerical? (Hint: Type ?mpg to read the documentation for the dataset.) How can you see this information when you run mpg?

**

Categorical: manufacturer, model, trans, dry, cty, class Numerical: displ, year, cyl, hwy, fl

You can see if a variable is categorical or numerical by using the `str` function on the tibble.

**

manufacturer: chr [1:234] “audi” “audi” “audi” “audi” … $ model : chr [1:234] “a4” “a4” “a4” “a4” … $ displ : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 … $ year : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 … $ cyl : int [1:234] 4 4 4 4 6 6 6 4 4 4 … $ trans : chr [1:234] “auto(l5)” “manual(m5)” “manual(m6)” “auto(av)” … $ drv : chr [1:234] “f” “f” “f” “f” … $ cty : int [1:234] 18 21 20 21 16 18 18 18 16 20 … $ hwy : int [1:234] 29 29 31 30 26 26 27 26 25 28 … $ fl : chr [1:234] “p” “p” “p” “p” … $ class

```
str(mpg)
```

```
## tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
##  $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
##  $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
##  $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
```

```
##  $ year       : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl        : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
##  $ trans      : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv        : chr [1:234] "f" "f" "f" "f" ...
##  $ cty        : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy        : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
##  $ fl         : chr [1:234] "p" "p" "p" "p" ...
##  $ class      : chr [1:234] "compact" "compact" "compact" "compact" ...
```

2

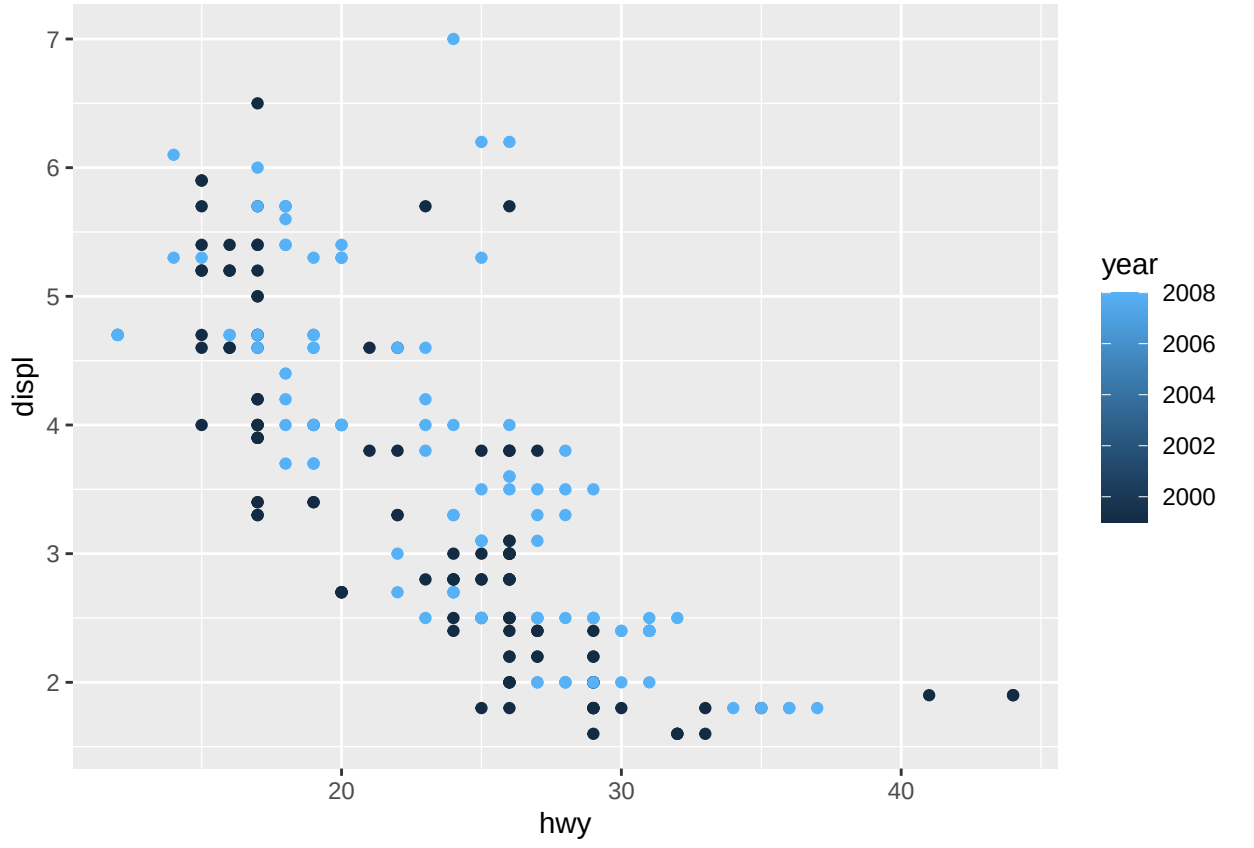
Make a scatterplot of hwy vs. displ using the mpg data frame. Next, map a third, numerical variable to color, then size, then both color and size, then shape. How do these aesthetics behave differently for categorical vs. numerical variables?

** color: For numerical there is a gradient of lighter to darker blue as the number changes. For categorical there are different colors that are associated with each category.

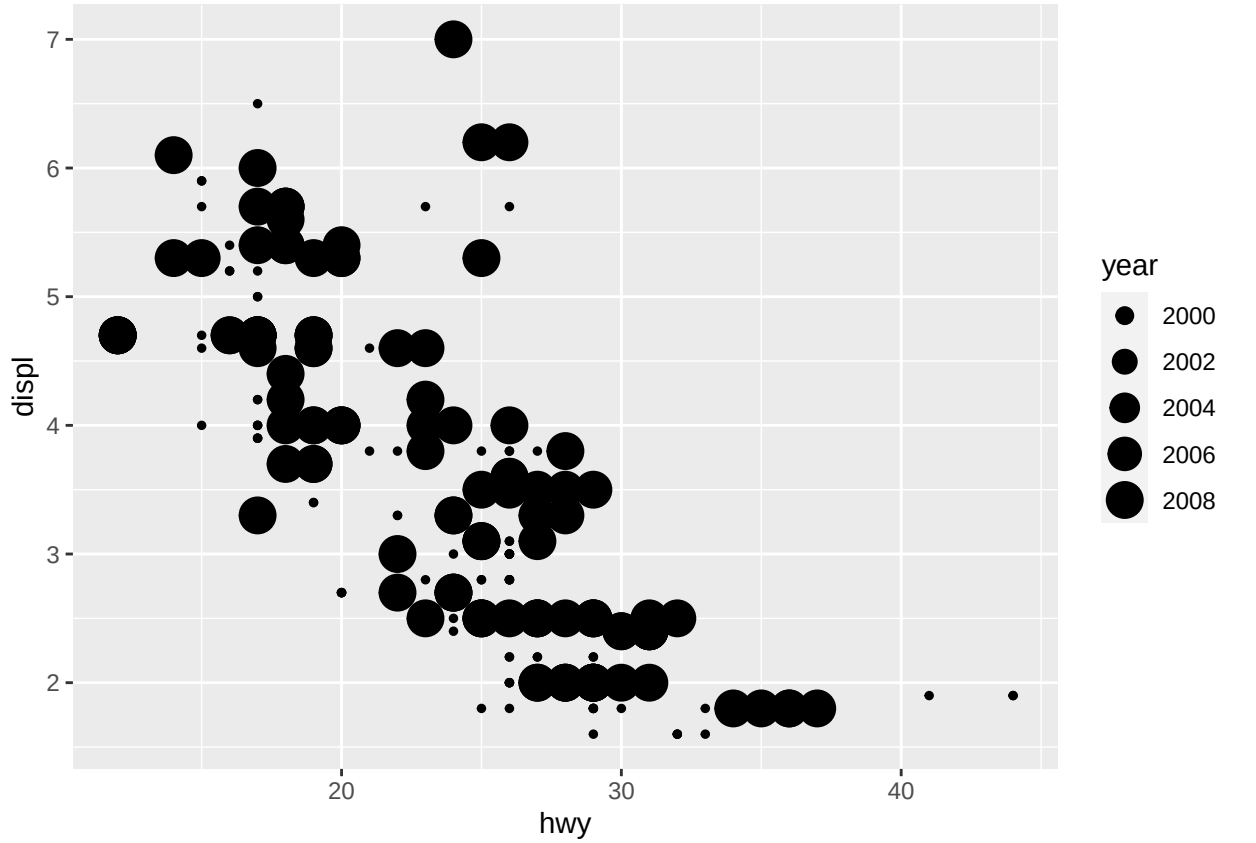
size: For numerical there is a gradient of sizes as the number changes. For categorical there are different discrete sizes that are associated with each category.

**

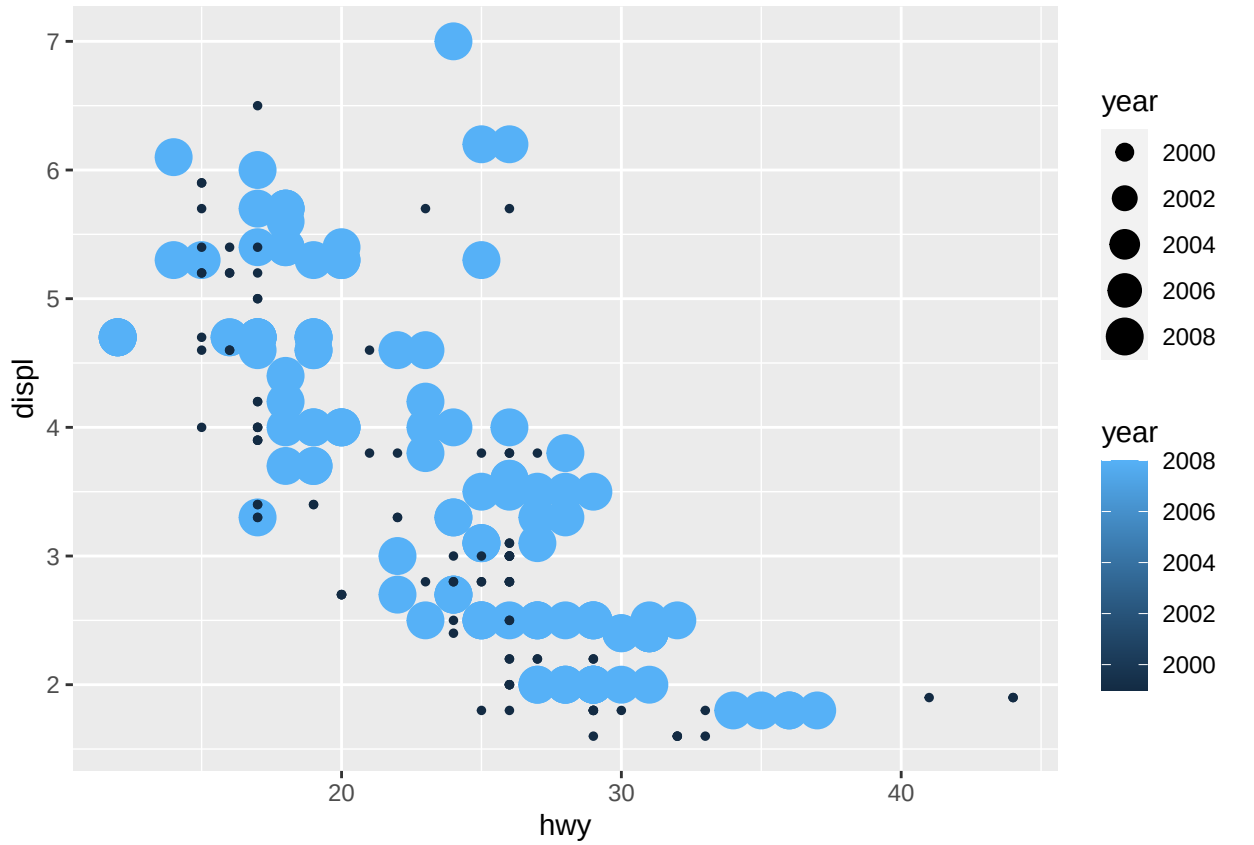
```
ggplot(mpg, aes(x=hwy, y=displ, colour=year)) + geom_point()
```



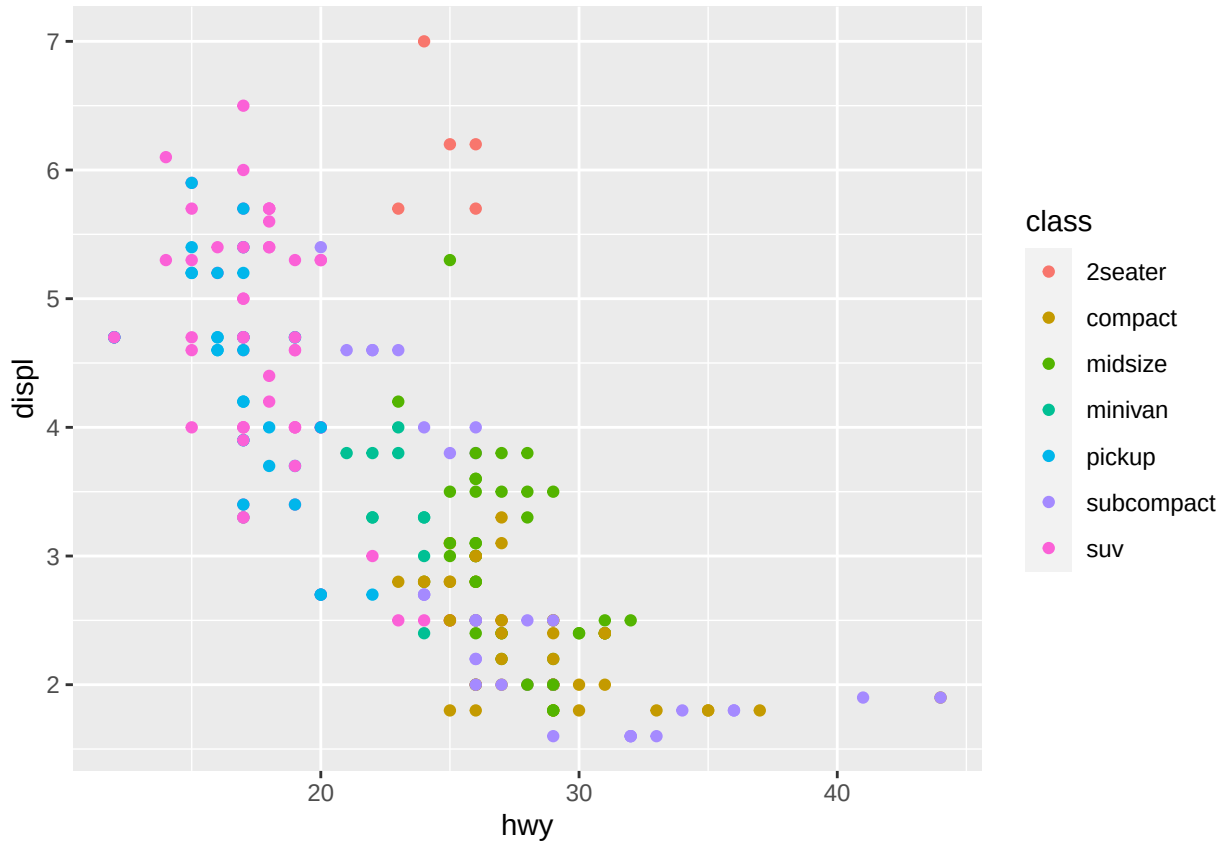

```
ggplot(mpg, aes(x=hwy, y=displ, size=year)) + geom_point()
```




```
ggplot(mpg, aes(x=hwy, y=displ, colour=year, size=year)) + geom_point()
```

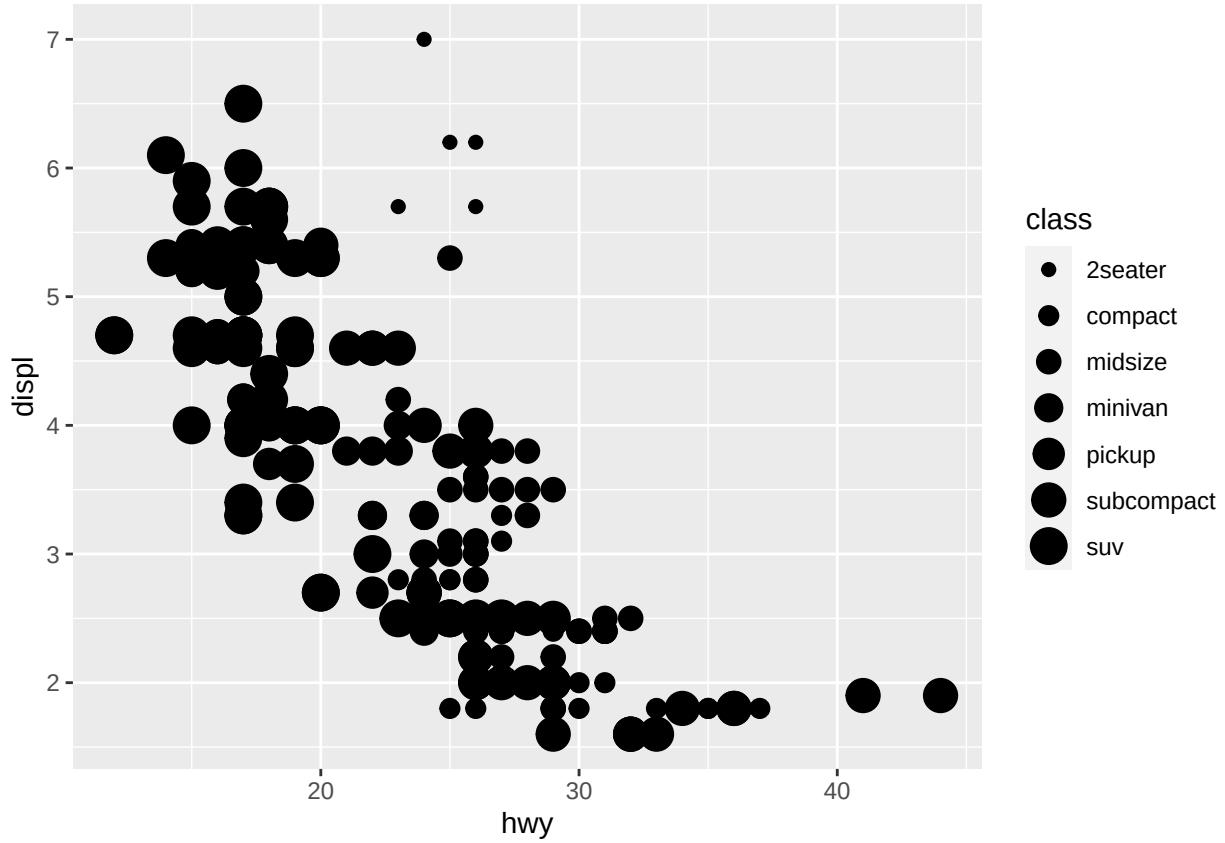



```
ggplot(mpg, aes(x=hwy, y=displ, colour=class)) + geom_point()
```



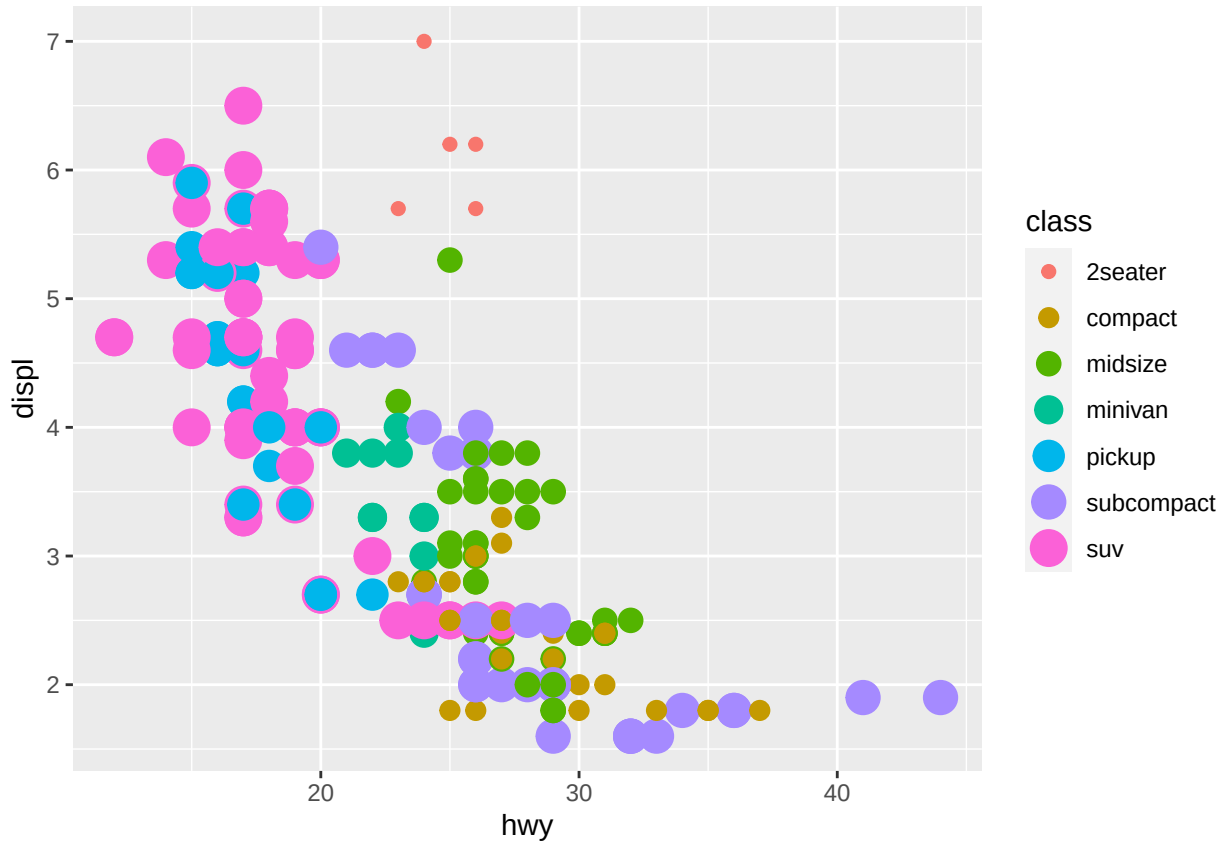

```
ggplot(mpg, aes(x=hwy, y=displ, size=class)) + geom_point()
```

```
## Warning: Using size for a discrete variable is not advised.
```




```
ggplot(mpg, aes(x=hwy, y=displ, colour=class, size=class)) + geom_point()
```

```
## Warning: Using size for a discrete variable is not advised.
```

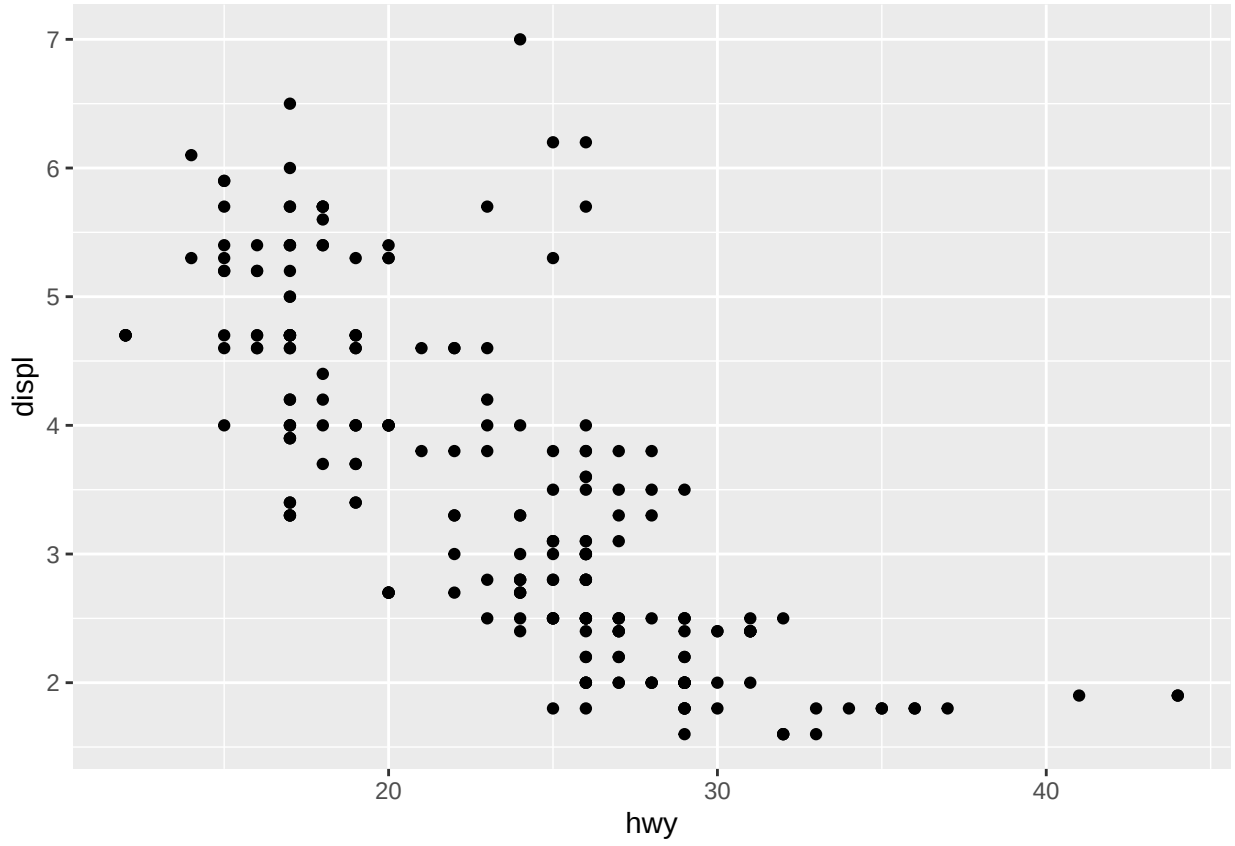


3

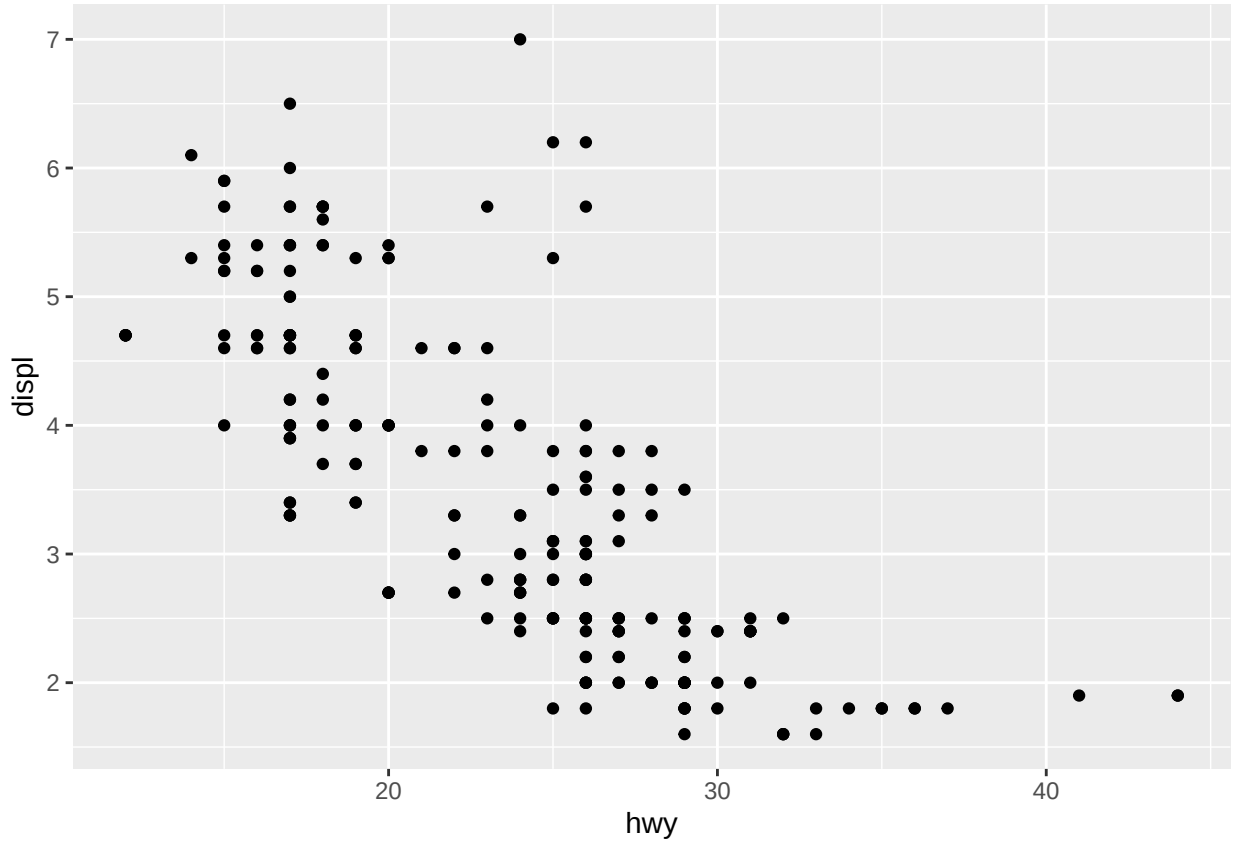
In the scatterplot of hwy vs. displ, what happens if you map a third variable to linewidth?

Nothing, because there are no lines in the scatterplot

```
#simple scatter plot of hwy MPG (x axis) vs engine displacement volume (y axis).
ggplot(data = mpg, aes(x = hwy, y = displ)) +
  geom_point()
```




```
#added a third variable to linewidth
#simple scatter plot of hwy MPG (x axis) vs engine displacement volume (y axis).
ggplot(data = mpg, aes(x = hwy, y = displ, linewidth = cty)) +
  geom_point()
```

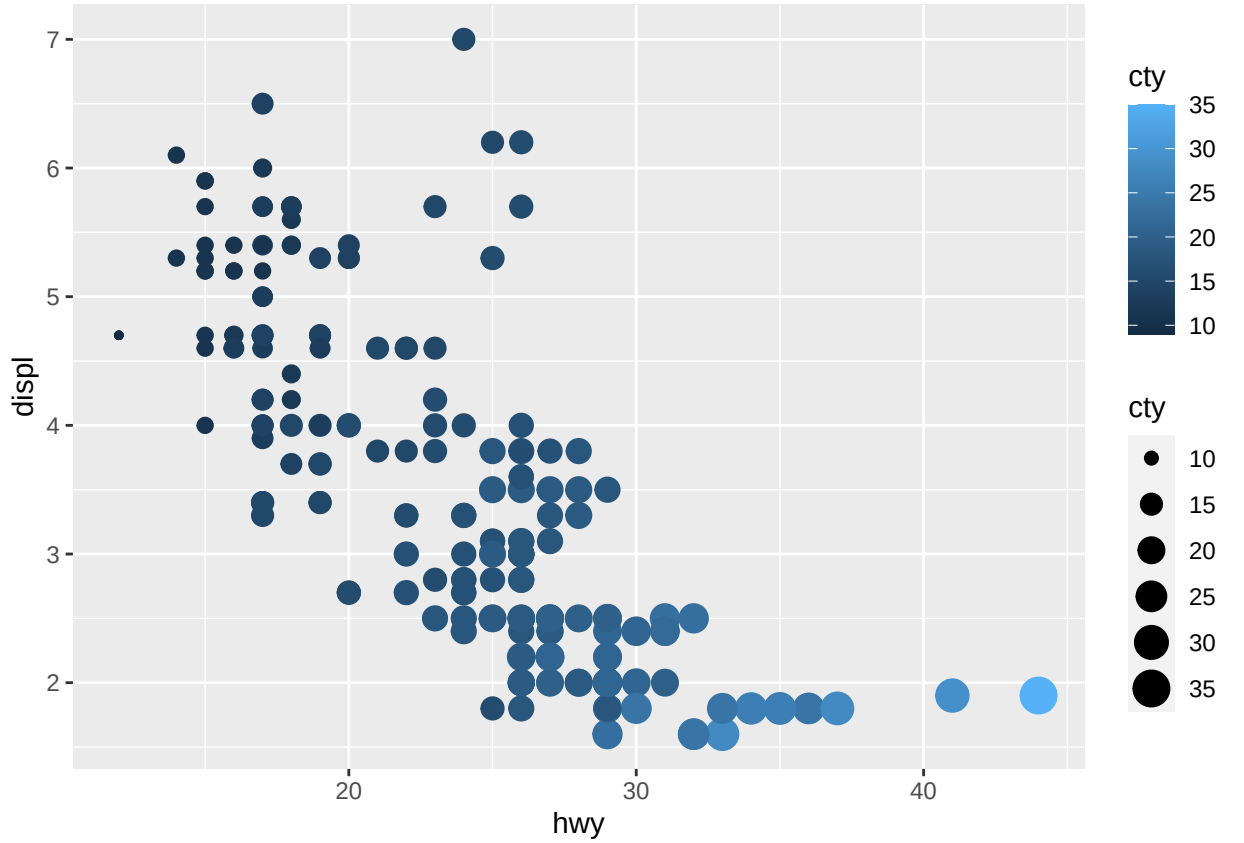


4

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

Then the same variable will be mapped to multiple aesthetics

```
#let's try it out...
ggplot(data = mpg, aes(x = hwy, y = displ, color = cty, size = cty)) +
  geom_point()
```



5

Make a scatterplot of bill_depth_mm vs. bill_length_mm and color the points by species. What does adding coloring by species reveal about the relationship between these two variables? What about faceting by species?

6

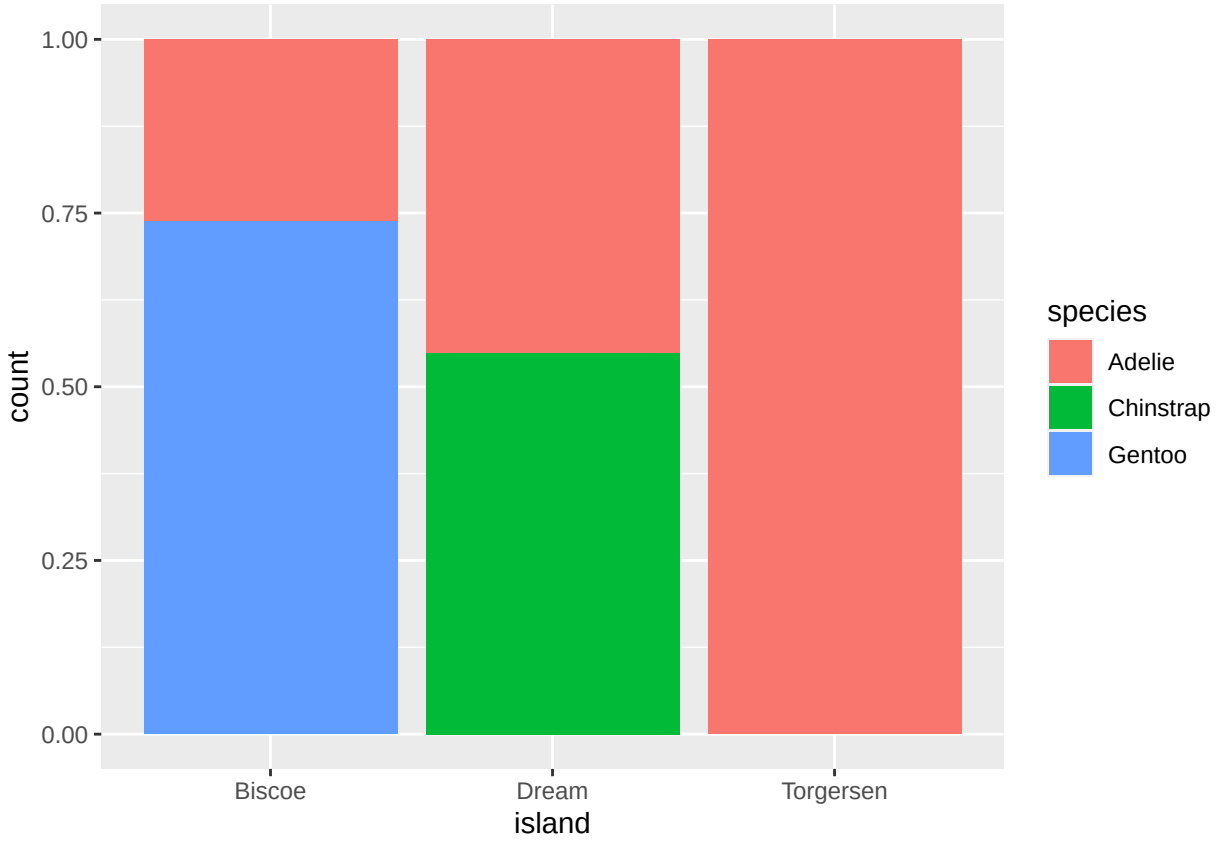
Why does the following yield two separate legends? How would you fix it to combine the two legends?

ggplot(data = penguins, mapping = aes(x = bill_length_mm, y = bill_depth_mm, color = species, shape = species)) + geom_point() + labs(color = "Species")

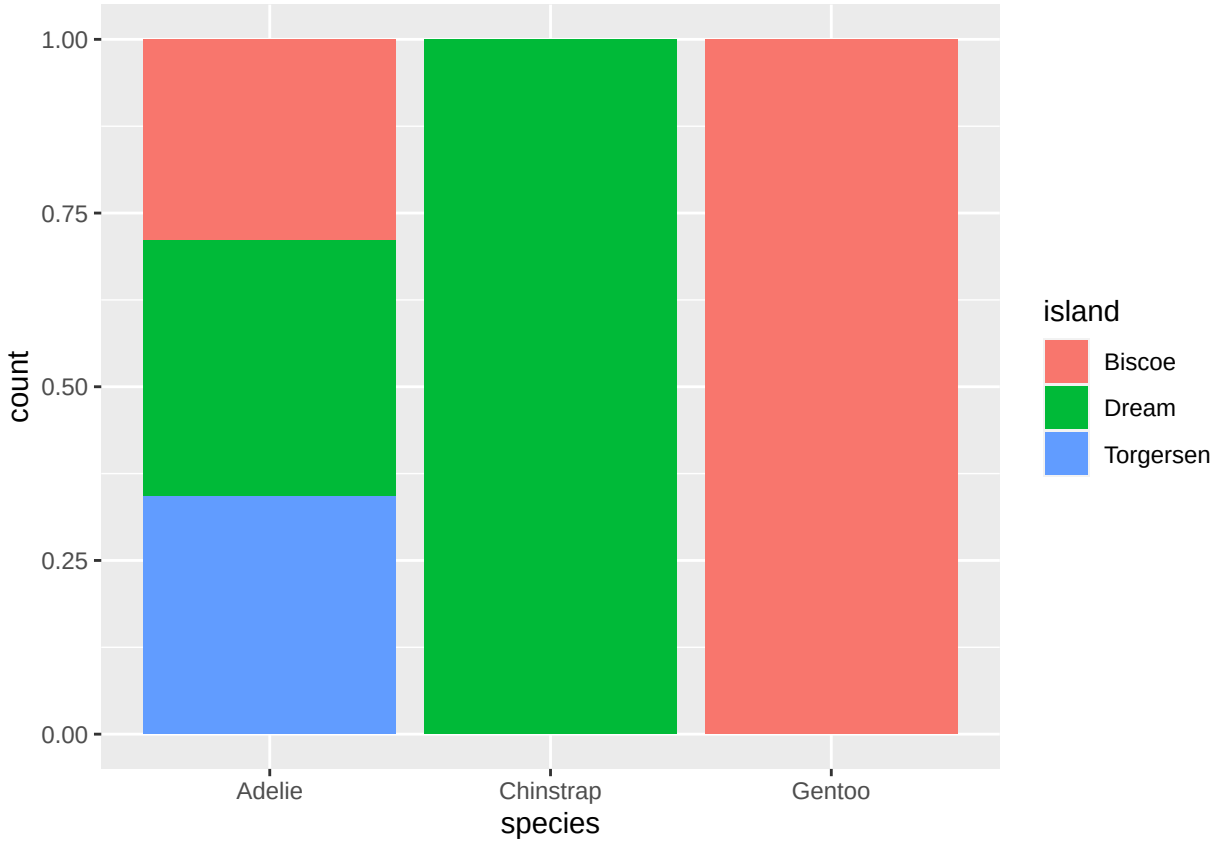
7

Create the two following stacked bar plots. Which question can you answer with the first one? Which question can you answer with the second one?

```
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar(position = "fill")
```




```
ggplot(penguins, aes(x = species, fill = island)) +
  geom_bar(position = "fill")
```



2.5 Exercises

1

Why does this code not work?

my_variable <- 10 my_vaıable #> Error in eval(expr, envir, enclos): object 'my_vaıable' not found

Look carefully! (This may seem like an exercise in pointlessness, but training your brain to notice even the tiniest difference will pay off when programming.)

** Answer: my_vaıable is spelled differently than the variable that was defined, so my_vaıable was not defined

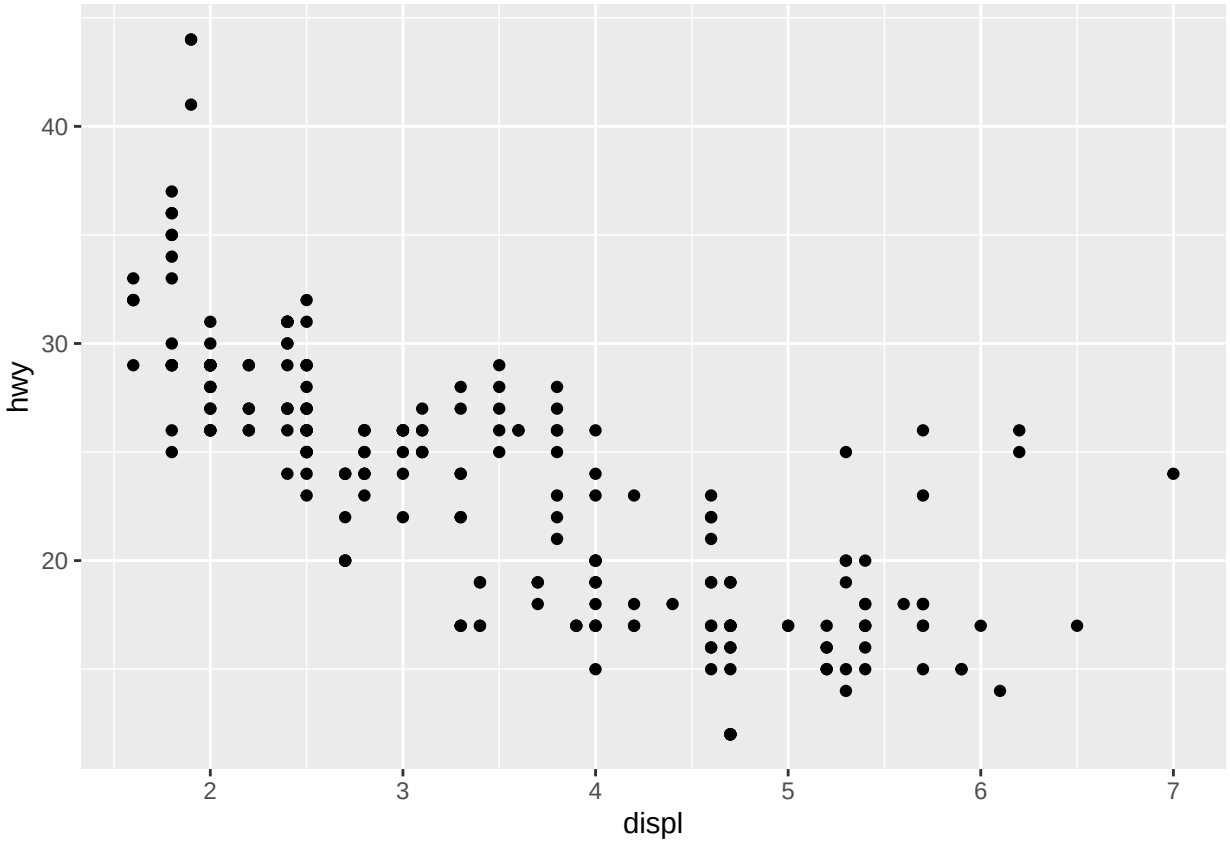
2

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

```
# remove the comment, i.e. # symbol from the below lines by selecting and pressing control-shift-c (i.e
# libary(todyverse)
#
# ggplot(dTA = mpg) +
#   geom_point(maping = aes(x = displ y = hwy)) +
#   geom_smooth(method = "lm)

library(tidyverse)

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




```
 geom_smooth(method = "lm")
```

```
## geom_smooth: na.rm = FALSE, orientation = NA, se = TRUE
## stat_smooth: na.rm = FALSE, orientation = NA, se = TRUE, method = lm
## position_identity
```

3

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

** Answer: The keyboard shortcuts appear. You can go to `Help >> Keyboard Shortcuts Help` **

4

Let's revisit an exercise from the Section 1.6. Run the following lines of code. Which of the two plots is saved as mpg-plot.png? Why?

** `my_bar_plot` is saved because it is speficied in the `ggsave` function under the `plot` variable. **

```
my_bar_plot <- ggplot(mpg, aes(x = class)) +
  geom_bar()
my_scatter_plot <- ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point()
#ggsave(filename = "mpg-plot.png", plot = my_bar_plot)
```

3.2.5 Exercises

1

In a single pipeline for each condition, find all flights that meet the condition: Had an arrival delay of two or more hours Flew to Houston (IAH or HOU) Were operated by United, American, or Delta Departed in summer (July, August, and September) Arrived more than two hours late, but didn't leave late Were delayed by at least an hour, but made up over 30 minutes in flight

```
#for my own edification view/str
View(flights)
str(flights)
```

```
## tibble [336,776 x 19] (S3: tbl_df/tbl/data.frame)
##  $ year          : int [1:336776] 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int [1:336776] 515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num [1:336776] 2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int [1:336776] 819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num [1:336776] 11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr [1:336776] "UA" "UA" "AA" "B6" ...
##  $ flight        : int [1:336776] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr [1:336776] "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num [1:336776] 1400 1416 1089 1576 762 ...
##  $ hour          : num [1:336776] 5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num [1:336776] 15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct[1:336776], format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

```
#flights with arrival delay of >120min
flights |>
  filter(arr_delay > 120)
```

```
## # A tibble: 10,034 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      811            630       101     1047            830
##  2  2013     1     1      848           1835       853     1001           1950
##  3  2013     1     1      957            733       144     1056            853
##  4  2013     1     1     1114            900       134     1447           1222
##  5  2013     1     1     1505           1310       115     1638           1431
##  6  2013     1     1     1525           1340       105     1831           1626
##  7  2013     1     1     1549           1445        64     1912           1656
##  8  2013     1     1     1558           1359       119     1718           1515
##  9  2013     1     1     1732           1630        62     2028           1825
## 10  2013     1     1     1803           1620       103     2008           1750
## # i 10,024 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# Filter flights to those with destination of IAH or HOU
flights |>
  filter(dest == "IAH"|dest == "HOU")
```

```
## # A tibble: 9,313 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      623            627        -4      933            932
##  4  2013     1     1      728            732        -4     1041           1038
##  5  2013     1     1      739            739         0     1104           1038
##  6  2013     1     1      908            908         0     1228           1219
##  7  2013     1     1     1028           1026         2     1350           1339
##  8  2013     1     1     1044           1045        -1     1352           1351
##  9  2013     1     1     1114            900       134     1447           1222
## 10  2013     1     1     1205           1200         5     1503           1505
## # i 9,303 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# filter to flights operated by UA, AA, & DL
flights |>
  filter(carrier == "UA" | carrier == "AA" | carrier == "DL")
```

```
## # A tibble: 139,504 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      554            600        -6      812            837
##  5  2013     1     1      554            558        -4      740            728
##  6  2013     1     1      558            600        -2      753            745
##  7  2013     1     1      558            600        -2      924            917
##  8  2013     1     1      558            600        -2      923            937
##  9  2013     1     1      559            600        -1      941            910
## 10  2013     1     1      559            600        -1      854            902
## # i 139,494 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# filter to months July-Sept
flights |>
  filter(month == 7| month == 8| month == 9)
```

```
## # A tibble: 86,326 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7     1        1           2029       212      236           2359
##  2  2013     7     1        2           2359         3      344            344
##  3  2013     7     1       29           2245       104      151              1
##  4  2013     7     1       43           2130       193      322             14
##  5  2013     7     1       44           2150       174      300            100
```

```
##  6  2013     7     1       46           2051       235      304           2358
##  7  2013     7     1       48           2001       287      308           2305
##  8  2013     7     1       58           2155       183      335             43
##  9  2013     7     1      100           2146       194      327             30
## 10  2013     7     1      100           2245       135      337            135
## # i 86,316 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# flights that arrived >120min late, but left on time
flights |>
  filter(dep_delay == 0 & arr_delay > 120)
```

```
## # A tibble: 3 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013    10     7     1350           1350         0     1736           1526
## 2  2013     5    23     1810           1810         0     2208           2000
## 3  2013     7     1      905            905         0     1443           1223
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
#flights that left at least an hour late but arrived within 30 minutes of their scheduled arrival time.
flights |>
  filter(dep_delay > 60 & arr_delay <30)
```

```
## # A tibble: 181 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     3     1850           1745        65     2148           2120
##  2  2013     1     3     1950           1845        65     2228           2227
##  3  2013     1     6     1019            900        79     1558           1530
##  4  2013     1     7     1543           1430        73     1758           1735
##  5  2013     1    12     1706           1600        66     1949           1927
##  6  2013     1    12     1953           1845        68     2154           2137
##  7  2013     1    19     1456           1355        61     1636           1615
##  8  2013     1    21     1531           1430        61     1843           1815
##  9  2013     1    21     1648           1545        63     1939           1910
## 10  2013    10    10     1938           1835        63     2158           2148
## # i 171 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

2

Sort flights to find the flights with longest departure delays. Find the flights that left earliest in the morning.

```
# sort(flights, dep_delay)

flights %>%
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
```

```
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
filter(flights, dep_time > 800) %>%
  arrange(dep_time)
```

```
## # A tibble: 275,083 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      801            805        -4      900            919
##  2  2013     1     2      801            730        31     1136           1040
##  3  2013     1     2      801            810        -9      951            955
##  4  2013     1     3      801            803        -2     1038           1025
##  5  2013     1     3      801            810        -9      940            955
##  6  2013     1     3      801            759         2     1025           1025
##  7  2013     1     4      801            720        41     1000            920
##  8  2013     1     4      801            759         2     1002            959
##  9  2013     1     5      801            805        -4     1025           1027
## 10  2013     1     5      801            805        -4      919            920
## # i 275,073 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

3

Sort flights to find the fastest flights. (Hint: Try including a math calculation inside of your function.)

```
flights %>%
  arrange(desc(air_time / distance))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1    28     1917           1825        52     2118           1935
##  2  2013     6    29      755            800        -5     1035            909
##  3  2013     8    28      932            940        -8     1116           1051
##  4  2013     1    30     1037            955        42     1221           1100
##  5  2013    11    27      556            600        -4      727            658
##  6  2013     5    21      558            600        -2      721            657
##  7  2013    12     9     1540           1535         5     1720           1656
##  8  2013     6    10     1356           1300        56     1646           1414
##  9  2013     7    28     1322           1325        -3     1612           1432
```

```
## 10  2013     4    11     1349           1345         4     1542           1453
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4

Was there a flight on every day of 2013? **Answer: When we counted all flights that occurred in month day pair there are 365 rows corresponding to data points for each day of the year.

```
# Filtering for all flights that occured in 2013
# Piping all 2013 flights and looking at all the unique month day pairings
flights %>%
  count(month,day)
```

```
## # A tibble: 365 x 3
##    month   day     n
##    <int> <int> <int>
##  1     1     1   842
##  2     1     2   943
##  3     1     3   914
##  4     1     4   915
##  5     1     5   720
##  6     1     6   832
##  7     1     7   933
##  8     1     8   899
##  9     1     9   902
## 10     1    10   932
## # i 355 more rows
```

5

Which flights traveled the farthest distance? Which traveled the least distance? **Answer: Least distance was flight 1632, Farthest distance was 51.**

```
flights %>%
  arrange(distance)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7    27       NA            106        NA       NA            245
##  2  2013     1     3     2127           2129        -2     2222           2224
##  3  2013     1     4     1240           1200        40     1333           1306
##  4  2013     1     4     1829           1615       134     1937           1721
##  5  2013     1     4     2128           2129        -1     2218           2224
##  6  2013     1     5     1155           1200        -5     1241           1306
##  7  2013     1     6     2125           2129        -4     2224           2224
##  8  2013     1     7     2124           2129        -5     2212           2224
##  9  2013     1     8     2127           2130        -3     2304           2225
## 10  2013     1     9     2126           2129        -3     2217           2224
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights %>%
  arrange(desc(distance))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      857            900        -3     1516           1530
##  2  2013     1     2      909            900         9     1525           1530
##  3  2013     1     3      914            900        14     1504           1530
##  4  2013     1     4      900            900         0     1516           1530
##  5  2013     1     5      858            900        -2     1519           1530
##  6  2013     1     6     1019            900        79     1558           1530
##  7  2013     1     7     1042            900       102     1620           1530
##  8  2013     1     8      901            900         1     1504           1530
##  9  2013     1     9      641            900      1301     1242           1530
## 10  2013     1    10      859            900        -1     1449           1530
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

6

Does it matter what order you used filter() and arrange() if you're using both? Why/why not? Think about the results and how much work the functions would have to do. **Answer:The order that the functions are used does not change the output of the data because filtering and arranging are both maintained when followed by the other respective function.**

```
flights %>%
filter(arr_delay>2) %>%
arrange(arr_delay)
```

```
## # A tibble: 123,096 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      622            630        -8     1017           1014
##  2  2013     1     1      728            732        -4     1041           1038
##  3  2013     1     1      743            730        13     1059           1056
##  4  2013     1     1      830            830         0     1018           1015
##  5  2013     1     1      902            903        -1     1048           1045
##  6  2013     1     1      937            940        -3     1238           1235
##  7  2013     1     1     1113           1115        -2     1318           1315
##  8  2013     1     1     1130           1131        -1     1345           1342
##  9  2013     1     1     1133           1129         4     1440           1437
## 10  2013     1     1     1231           1238        -7     1449           1446
## # i 123,086 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights %>%
  arrange(arr_delay) %>%
filter(arr_delay>2)
```

```
## # A tibble: 123,096 x 19
```

```
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      622            630        -8     1017           1014
##  2  2013     1     1      728            732        -4     1041           1038
##  3  2013     1     1      743            730        13     1059           1056
##  4  2013     1     1      830            830         0     1018           1015
##  5  2013     1     1      902            903        -1     1048           1045
##  6  2013     1     1      937            940        -3     1238           1235
##  7  2013     1     1     1113           1115        -2     1318           1315
##  8  2013     1     1     1130           1131        -1     1345           1342
##  9  2013     1     1     1133           1129         4     1440           1437
## 10  2013     1     1     1231           1238        -7     1449           1446
## # i 123,086 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.3.5 Exercises

1

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

```
flights %>%
  select(dep_time, sched_dep_time, dep_delay)
```

```
## # A tibble: 336,776 x 3
##    dep_time sched_dep_time dep_delay
##       <int>          <int>     <dbl>
##  1      517            515         2
##  2      533            529         4
##  3      542            540         2
##  4      544            545        -1
##  5      554            600        -6
##  6      554            558        -4
##  7      555            600        -5
##  8      557            600        -3
##  9      557            600        -3
## 10      558            600        -2
## # i 336,766 more rows
```

2

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

```
flights %>%
select(dep_time, dep_delay, arr_time, arr_delay)
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
##  1      517         2      830        11
##  2      533         4      850        20
##  3      542         2      923        33
##  4      544        -1     1004       -18
```

```
##  5      554        -6      812       -25
##  6      554        -4      740        12
##  7      555        -5      913        19
##  8      557        -3      709       -14
##  9      557        -3      838        -8
## 10      558        -2      753         8
## # i 336,766 more rows
```

```
flights %>%
select(starts_with("dep"), starts_with("arr") )
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
##  1      517         2      830        11
##  2      533         4      850        20
##  3      542         2      923        33
##  4      544        -1     1004       -18
##  5      554        -6      812       -25
##  6      554        -4      740        12
##  7      555        -5      913        19
##  8      557        -3      709       -14
##  9      557        -3      838        -8
## 10      558        -2      753         8
## # i 336,766 more rows
```

```
flights %>%
  select(contains("time"),contains("delay"),-contains("sched"),-contains("air"),-contains("hour"))
```

```
## # A tibble: 336,776 x 4
##    dep_time arr_time dep_delay arr_delay
##       <int>    <int>     <dbl>     <dbl>
##  1      517      830         2        11
##  2      533      850         4        20
##  3      542      923         2        33
##  4      544     1004        -1       -18
##  5      554      812        -6       -25
##  6      554      740        -4        12
##  7      555      913        -5        19
##  8      557      709        -3       -14
##  9      557      838        -3        -8
## 10      558      753        -2         8
## # i 336,766 more rows
```

```
flights %>%
  select(c(4,6,7,9))
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
##  1      517         2      830        11
##  2      533         4      850        20
##  3      542         2      923        33
##  4      544        -1     1004       -18
##  5      554        -6      812       -25
##  6      554        -4      740        12
```

```
##  7      555        -5      913        19
##  8      557        -3      709       -14
##  9      557        -3      838        -8
## 10      558        -2      753         8
## # i 336,766 more rows
```

3

What happens if you specify the name of the same variable multiple times in a select() call? It ignores redundant column calls.

```
flights %>%
  select(dep_time, dep_time)
```

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

4

What does the any_of() function do? Why might it be helpful in conjunction with this vector? variables <- c("year", "month", "day", "dep_delay", "arr_delay") ** Answer: 'any_of() function allows you to select for multiple items. In this case, it can be used to filter for all columns that are in the vector without raising an error if a column in the vector is missing in the tibble. **

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


flights %>% select(any_of(variables))
```

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

5

Does the result of running the following code surprise you? How do the select helpers deal with upper and lower case by default? How can you change that default? **Answer=The select function defaults to ignoring case so despite being in all caps the function still calls all columns that contain "time". To change the default you change the ignore.case argument to False.**

```
flights |> select(contains("TIME"))
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##       <int>          <int>    <int>          <int>    <dbl> <dttm>
##  1      517            515      830            819      227 2013-01-01 05:00:00
##  2      533            529      850            830      227 2013-01-01 05:00:00
##  3      542            540      923            850      160 2013-01-01 05:00:00
##  4      544            545     1004           1022      183 2013-01-01 05:00:00
##  5      554            600      812            837      116 2013-01-01 06:00:00
##  6      554            558      740            728      150 2013-01-01 05:00:00
##  7      555            600      913            854      158 2013-01-01 06:00:00
##  8      557            600      709            723       53 2013-01-01 06:00:00
##  9      557            600      838            846      140 2013-01-01 06:00:00
## 10      558            600      753            745      138 2013-01-01 06:00:00
## # i 336,766 more rows
```

```
flights |> select(contains("TIME",ignore.case=FALSE))
```

```
## # A tibble: 336,776 x 0
```

6

Rename air_time to air_time_min to indicate units of measurement and move it to the beginning of the data frame.

```
flights %>%
  rename(air_time_min=air_time) %>%
  relocate(air_time_min)
```

```
## # A tibble: 336,776 x 19
##    air_time_min  year month   day dep_time sched_dep_time dep_delay arr_time
##           <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1          227  2013     1     1      517            515         2      830
##  2          227  2013     1     1      533            529         4      850
##  3          160  2013     1     1      542            540         2      923
##  4          183  2013     1     1      544            545        -1     1004
##  5          116  2013     1     1      554            600        -6      812
##  6          150  2013     1     1      554            558        -4      740
##  7          158  2013     1     1      555            600        -5      913
##  8           53  2013     1     1      557            600        -3      709
##  9          140  2013     1     1      557            600        -3      838
## 10          138  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

7

Why doesn't the following work, and what does the error mean? **Answer: The code doesn't work because only the column `tailnum` was selected for, and `arr_delay` was no longer part of the new tibble. So it wouldn't have been able to filter by `arr_delay`. The error shows what function caused the error which is `arrange()`, and that the error was because `arr_delay` was not in the tibble.**

```
# remove comments by selecting all desired lines of code and using ctrl-C (control-shift-c) i.e. commen
#
#flights |>
#  select(tailnum)
#   arrange(arr_delay)
# #> Error in `arrange()`:
# #> i In argument: `..1 = arr_delay`.
# #> Caused by error:
# #> ! object 'arr_delay' not found
```

3.5.7 Exercises

1

Which carrier has the worst average delays? Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about flights |> group_by(carrier, dest) |> summarize(n())) **Answer: F9 airline is the worst**

```
flights %>%
  group_by(carrier) %>%
  summarise(avg = mean(dep_delay, na.rm = TRUE) + mean(arr_delay, na.rm = TRUE)) %>%
  arrange(desc(avg))
```

```
## # A tibble: 16 x 2
##    carrier   avg
##    <chr>   <dbl>
##  1 F9      42.1
##  2 FL      38.8
##  3 EV      35.8
##  4 YV      34.6
##  5 WN      27.4
##  6 OO      24.5
##  7 9E      24.1
##  8 B6      22.5
##  9 MQ      21.3
## 10 UA      15.7
## 11 VX      14.6
## 12 DL      10.9
## 13 AA       8.95
## 14 US       5.91
## 15 HA      -2.01
## 16 AS      -4.13
```

2

Find the flights that are most delayed upon departure from each destination.

```
flights %>%
  group_by(dest) %>%
```

```
  slice_max(dep_delay, n = 1) %>%
  relocate(dest)
```

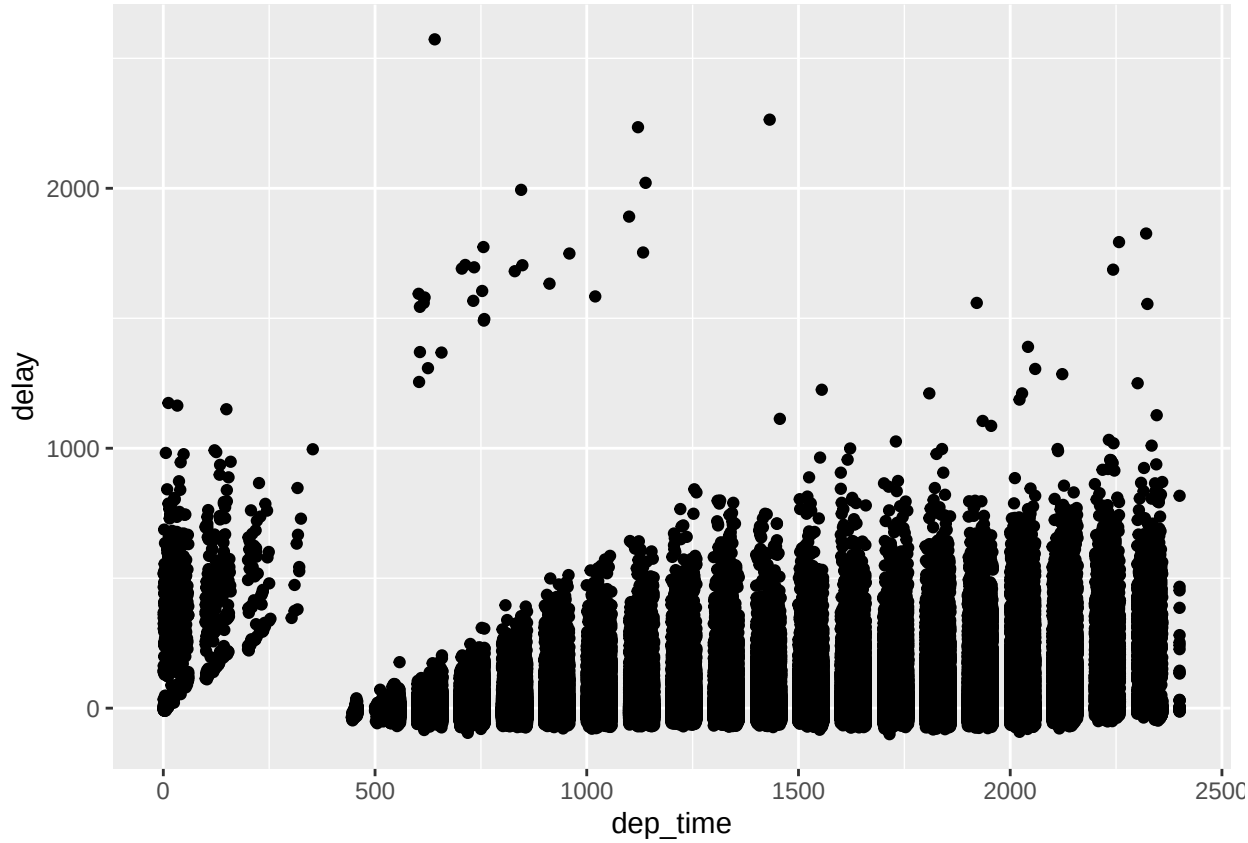
```
## # A tibble: 105 x 19
## # Groups:   dest [105]
##    dest   year month   day dep_time sched_dep_time dep_delay arr_time
##    <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1 ABQ    2013    12    14     2223           2001       142      133
##  2 ACK    2013     7    23     1139            800       219     1250
##  3 ALB    2013     1    25      123           2000       323      229
##  4 ANC    2013     8    17     1740           1625        75     2042
##  5 ATL    2013     7    22     2257            759       898      121
##  6 AUS    2013     7    10     2056           1505       351     2347
##  7 AVL    2013     6    14     1158            816       222     1335
##  8 BDL    2013     2    21     1728           1316       252     1839
##  9 BGR    2013    12     1     1504           1056       248     1628
## 10 BHM    2013     4    10       25           1900       325      136
## # i 95 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

3

How do delays vary over the course of the day. Illustrate your answer with a plot. **Answer: Delays get progressively longer throughout the day**

```
flights %>%
  mutate(delay=arr_delay+dep_delay) %>%
  ggplot() + geom_point(aes(x = dep_time, y = delay))
```

```
## Warning: Removed 9430 rows containing missing values (`geom_point()`).
```



4

What happens if you supply a negative n to slice_min() and friends? **answer: with a negative n, functions like slice_min() will return all rows instead of a specified number.**

```
flights %>%
slice_min(dep_delay, n = 1)
```

```
## # A tibble: 1 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013    12     7     2040           2123       -43       40           2352
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights %>%
slice_min(dep_delay, n = -1)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    12     7     2040           2123       -43       40           2352
##  2  2013     2     3     2022           2055       -33     2240           2338
##  3  2013    11    10     1408           1440       -32     1549           1559
##  4  2013     1    11     1900           1930       -30     2233           2243
##  5  2013     1    29     1703           1730       -27     1947           1957
```

```
##  6  2013     8     9      729            755       -26     1002            955
##  7  2013    10    23     1907           1932       -25     2143           2143
##  8  2013     3    30     2030           2055       -25     2213           2250
##  9  2013     3     2     1431           1455       -24     1601           1631
## 10  2013     5     5      934            958       -24     1225           1309
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

5

Explain what count() does in terms of the dplyr verbs you just learned. What does the sort argument to count() do? **Answer: `count()` function counts the number of rows that have a given value in a specified column. Then it creates a new table with the counts of each group.The sort argument order the rows in decending value.**

```
flights %>% count(carrier, sort=TRUE)
```

```
## # A tibble: 16 x 2
##    carrier     n
##    <chr>   <int>
##  1 UA      58665
##  2 B6      54635
##  3 EV      54173
##  4 DL      48110
##  5 AA      32729
##  6 MQ      26397
##  7 US      20536
##  8 9E      18460
##  9 WN      12275
## 10 VX       5162
## 11 FL       3260
## 12 AS        714
## 13 F9        685
## 14 YV        601
## 15 HA        342
## 16 OO         32
```

6

Suppose we have the following tiny data frame:

```
    df <- tibble(
      x = 1:5,
      y = c("a", "b", "a", "a", "b"),
      z = c("K", "K", "L", "L", "K")
    )
```

6.a Write down what you think the output will look like, then check if you were correct, and describe what group_by() does. **Answer: It looks like the `group_by()` function will create groups based on the rows that share the same `y` value from `df`. According to the documentation, `group_by`creates a new table that is grouped where operations can performed per group.**

```
df |>
 group_by(y)
```

```
## # A tibble: 5 x 3
## # Groups:   y [2]
##       x y     z
##   <int> <chr> <chr>
## 1     1 a     K
## 2     2 b     K
## 3     3 a     L
## 4     4 a     L
## 5     5 b     K
```

6.b Write down what you think the output will look like, then check if you were correct, and describe what arrange() does. Also comment on how it's different from the group_by() in part (a)? **Answer: The output might be a table that is the sorted view of `df` based on the `y` value. It's different than `group_by()` because there seems to be no group functionality with the output of this code.**

```
df |>
  arrange(y)
```

```
## # A tibble: 5 x 3
##       x y     z
##   <int> <chr> <chr>
## 1     1 a     K
## 2     3 a     L
## 3     4 a     L
## 4     2 b     K
## 5     5 b     K
```

6.c Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does. **Answer: The output will probably be a table that has a row for each group of rows from `df` that had a given `y` value. Then the mean of `x` was calculated for all rows in a given group.The pipeline outputs all of the `df` table into `group_by(y)` function, which creates a group of rows that had a given `y` value. These groupings are outputted to the `summarize(mean_x = mean(x))` where the mean `x` was calculated for all rows in a given group.**

```
 df |>
   group_by(y) |>
   summarize(mean_x = mean(x))
```

```
## # A tibble: 2 x 2
##   y     mean_x
##   <chr>  <dbl>
## 1 a       2.67
## 2 b       3.5
```

6.d Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does. Then, comment on what the message says. **Answer: The output will probably be a table that has a row for each group of rows from `df` that had a given `y` and `z` unique pair. Then the mean of `x` was calculated for all rows in a given group.The pipeline outputs all of the `df` table into `group_by(y, z)` function, which creates a group of rows that had a given `y` and `z` unique pair. These groupings are outputted to the `summarize(mean_x = mean(x))` where the mean `x` was calculated for all rows in a given group.**

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

```
## `summarise()` has grouped output by 'y'. You can override using the `.groups`
## argument.
```

```
## # A tibble: 3 x 3
## # Groups:   y [2]
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

6.e Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does. How is the output different from the one in part (d). **Answer: The output will probably be similar to the output from problem 6.d with a table that has a row for each group of rows from `df` that had a given `y` and `z` unique pair. Then the mean of `x` was calculated for all rows in a given group. The pipeline outputs all of the `df` table into `group_by(y, z)` function, which creates a group of rows that had a given `y` and `z` unique pair. These groupings are outputted to the `summarize(mean_x = mean(x))` where the mean `x` was calculated for all rows in a given group.The main difference is that the `group` functionality is removed in this output, and new tibble is created.**

```
df |>
   group_by(y, z) |>
   summarize(mean_x = mean(x), .groups = "drop")
```

```
## # A tibble: 3 x 3
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

6.f Write down what you think the outputs will look like, then check if you were correct, and describe what each pipeline does. How are the outputs of the two pipelines different? **Answer: The output will probably be similar to the output from problem 6.d with a table that has a row for each group of rows from `df` that had a given `y` and `z` unique pair. Then the mean of `x` was calculated for all rows in a given group. The pipeline outputs all of the `df` table into `group_by(y, z)` function, which creates a group of rows that had a given `y` and `z` unique pair. These groupings are outputted to the `summarize(mean_x = mean(x))` where the mean `x` was calculated for all rows in a given group. The main difference is that the `group` functionality is removed in this output, and new tibble is created.**

```
 df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

```
## `summarise()` has grouped output by 'y'. You can override using the `.groups`
## argument.
```

```
## # A tibble: 3 x 3
## # Groups:   y [2]
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

```r
df |>
 group_by(y, z) |>
 mutate(mean_x = mean(x))
```

```
## # A tibble: 5 x 4
## # Groups:   y, z [3]
##       x y     z     mean_x
##   <int> <chr> <chr>  <dbl>
## 1     1 a     K        1
## 2     2 b     K        3.5
## 3     3 a     L        3.5
## 4     4 a     L        3.5
## 5     5 b     K        3.5
```