Selected Exercises for STT 2860

Material from R for Data Science (2e)

Last modified on August 19, 2024 14:56:51 Eastern Daylight Time

Table of contents

Preface

This document takes selected Exercises from R for Data Science (2e) and puts them in a Quarto book so you can modify the files to add your answers to the questions.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Exercises (Chapter 1)

1. How many rows are in `penguins`? How many columns?

   Answer

   ```
   library(tidyverse)
   library(palmerpenguins)
   # Your R Code here
   ```

   Your answer here.

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

   Answer

   Your answer here.

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.

   Answer

   ```
   # Your R code here
   ```

   Your answer here.

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

   Answer

   ```
   # Your R code here
   ```

   Your answer here.

```
# Your R code here
```

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

```
library(tidyverse)
ggplot(data = penguins) +
  geom_point()
```

Answer

Your answer here.

```
# Correct code here
```

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`.

Answer

Your answer here.

```
# Your R code here
```

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()`.

Answer

```
# Your R code here
```

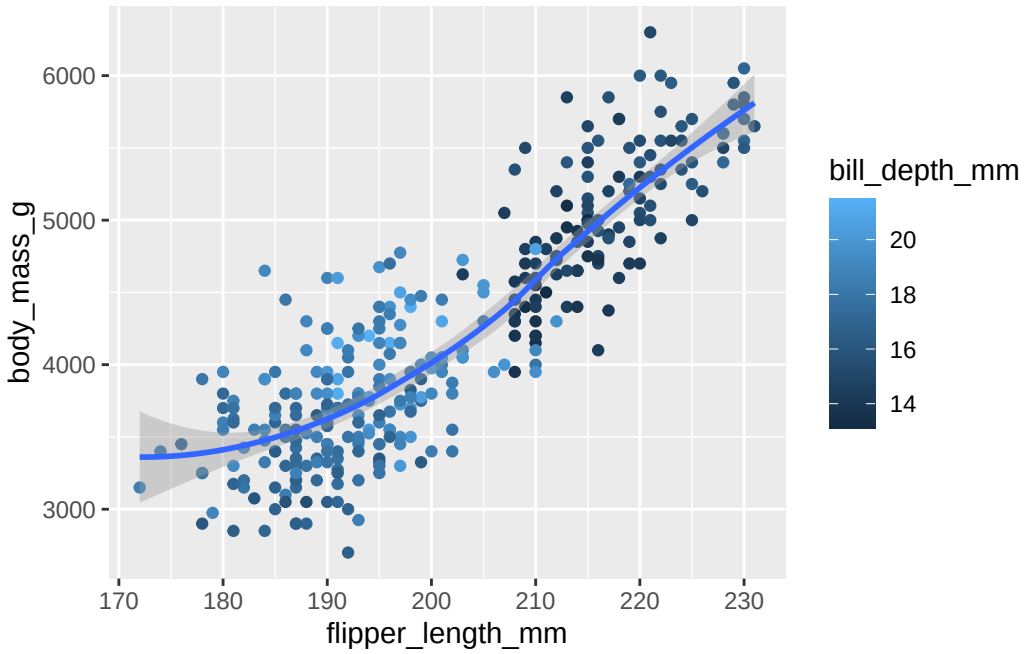
8. Recreate the following visualization. What aesthetic should `bill_depth_mm` be mapped to? And should it be mapped at the global level or at the geom level?

Answer

```
# Your R Code here
```

Your answer here.

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

Answer

Your answer here.

```
# Your R code here
```

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

```
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)
  )
```

Answer

Your answer here.

```
library(patchwork)
# Your R code here
```

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

Answer

Your answer here.

```
# Your R code here
```

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

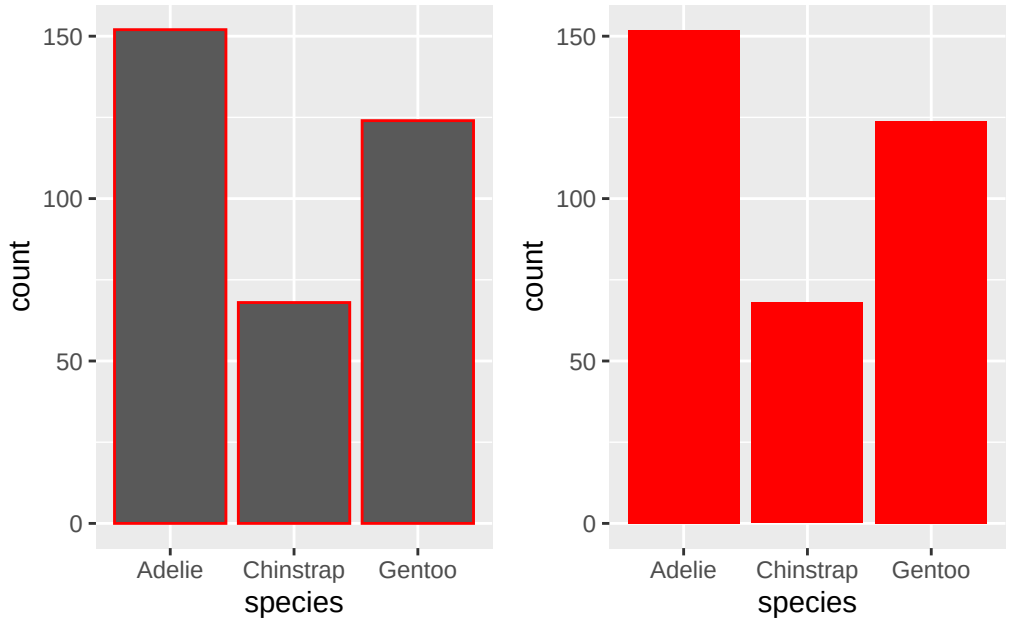
```
ggplot(penguins, aes(x = species)) +
  geom_bar(color = "red")

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

Answer

```
ggplot(penguins, aes(x = species)) +
    geom_bar(color = "red") -> p1

ggplot(penguins, aes(x = species)) +
  geom_bar(fill = "red") -> p2
p1 + p2
```



Your answer here.

13. What does the `bins` argument in `geom_histogram()` do?

Answer

Your answer here.

14. 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

```
# Your R code here
```

Your answer here.

15. 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`?

Answer

```
# Your R code here
```

Your answer here.

16. 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?

Answer

```
# Your R code here
```

Your answer here.

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

Answer

```
# Your R code here
```

Your answer here.

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

Answer

```
# Your R code here
```

Your answer here.

19. 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`?

Answer

```
# Your R code here
```

Your answer here.

20. 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")
```

Answer

```
# Your R fix here
```

Your answer here.

21. 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")
```

Answer

```
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar(position = "fill") -> p1
ggplot(penguins, aes(x = species, fill = island)) +
  geom_bar(position = "fill") -> p2
p1 / p2
```

Your answer here.

22. Run the following lines of code. Which of the two plots is saved as `mpg-plot.png`? Why?

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

Answer

```
# Your R code here
```

Your answer here.

23. What do you need to change in the code above to save the plot as a PDF instead of a PNG? How could you find out what types of image files would work in `ggsave()`?

Answer

Your answer here.

2 Exercises (Chapter 2)

1. Why does this code not work?

   ```
   my_variable <- 10
   my_variable
   ```

   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
   >
   > *Your answer here.*

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

   ```
   libary(todyverse)
   ggplot(dTA = mpg) +
     geom_point(maping = aes(x = displ y = hwy)) +
     geom_smooth(method = "lm)
   ```

   > Answer
   >
   > ```
   > # Your R code here
   > ```

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

   > Answer
   >
   > *Your answer here.*

4. Let's revisit an exercise from "Saving Your Plots". Run the following lines of code. Which of the two plots is saved as `mpg-plot.png`? Why?

```r
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)
```

Answer

```r
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)
```

```
Saving 5.5 x 3.5 in image
```

Your answer here.

3 Exercises (Chapter 3)

1. In a single pipeline for each condition, find all flights that meet the condition:

   - Had an arrival delay of two or more hours

     Answer

     ```
     library(tidyverse)
     library(nycflights13)
     # Your code here
     ```

   - Flew to Houston (`IAH` or `HOU`)

     Answer

     ```
     # Your code here
     ```

   - Were operated by United, American, or Delta

     Answer

     ```
     # Your code here
     ```

   - Departed in summer (July, August, and September)

     Answer

     ```
     # Your code here
     ```

   - Arrived more than two hours late, but didn't leave late.

Answer

```
# Your code here
```

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

Answer

```
# Your code here
```

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

Answer

```
# Your code here
```

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

Answer

```
# Your code here
```

4. Was there a flight on every day of 2013?

Answer

```
# Your code here
```

Your text answer here.

5. Which flights traveled the farthest distance? Which traveled the least distance?

Answer

```
# Your code here
```

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
>
> *Your text answer here.*

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

> Answer
>
> ```
> # Your code here
> ```
>
> *Your text answer here.*

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

> Answer
>
> ```
> # Your code here
> ```

9. What happens if you specify the name of the same variable multiple times in a `select()` call?

> Answer
>
> ```
> # Your code here
> ```
>
> *Your text answer here.*

10. 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")
# Try below first
flights |>
  select(variables)
```

```
Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
i Please use `all_of()` or `any_of()` instead.
  # Was:
  data %>% select(variables)
```

```
  # Now:
  data %>% select(all_of(variables))

See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```

```
# 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
```

```
  # Or
  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
```

Answer

Your text answer here.

11. 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?

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

Answer

```
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
```

Your text answer here.

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

Answer

```
# Your code here
```

13. Why doesn't the following work, and what does the error mean?

```
flights |>
  select(tailnum) |>
  arrange(arr_delay)
```

```
Error in `arrange()`:
```

```
i In argument: `..1 = arr_delay`.
Caused by error:
! object 'arr_delay' not found
```

```
flights |>
  select(tailnum)
```

```
# A tibble: 336,776 x 1
   tailnum
   <chr>
 1 N14228
 2 N24211
 3 N619AA
 4 N804JB
 5 N668DN
 6 N39463
 7 N516JB
 8 N829AS
 9 N593JB
10 N3ALAA
# i 336,766 more rows
```

Answer

Your text answer here.

14. 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

```
# Your code here
```

Your text answer here.

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

Answer

```
# Your code here
```

Your text answer here.

16. How do delays vary over the course of the day. Illustrate your answer with a plot.

Answer

```
# Your code here
```

Your text answer here.

17. What happens if you supply a negative `n` to `slice_min()` and friends?

Answer

```
flights |>
  slice_min(dep_delay, n = -5) |>
  relocate(dep_delay)
```

```
# A tibble: 336,776 x 19
   dep_delay  year month   day dep_time sched_dep_time arr_time sched_arr_time
       <dbl> <int> <int> <int>    <int>          <int>    <int>          <int>
 1       -43  2013    12     7     2040           2123       40           2352
 2       -33  2013     2     3     2022           2055     2240           2338
 3       -32  2013    11    10     1408           1440     1549           1559
 4       -30  2013     1    11     1900           1930     2233           2243
 5       -27  2013     1    29     1703           1730     1947           1957
 6       -26  2013     8     9      729            755     1002            955
 7       -25  2013    10    23     1907           1932     2143           2143
 8       -25  2013     3    30     2030           2055     2213           2250
 9       -24  2013     3     2     1431           1455     1601           1631
10       -24  2013     5     5      934            958     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>
```

```
flights |>
  slice_min(dep_delay, n = 5) |>
  relocate(dep_delay)
```

```
# A tibble: 5 x 19
  dep_delay  year month   day dep_time sched_dep_time arr_time sched_arr_time
      <dbl> <int> <int> <int>    <int>          <int>    <int>          <int>
1       -43  2013    12     7     2040           2123       40           2352
```

```
2       -33  2013     2     3     2022           2055     2240           2338
3       -32  2013    11    10     1408           1440     1549           1559
4       -30  2013     1    11     1900           1930     2233           2243
5       -27  2013     1    29     1703           1730     1947           1957
# 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_max(dep_delay, n = -5) |>
  relocate(dep_delay)
```

```
# A tibble: 336,776 x 19
   dep_delay  year month   day dep_time sched_dep_time arr_time sched_arr_time
       <dbl> <int> <int> <int>    <int>          <int>    <int>          <int>
 1      1301  2013     1     9      641            900     1242           1530
 2      1137  2013     6    15     1432           1935     1607           2120
 3      1126  2013     1    10     1121           1635     1239           1810
 4      1014  2013     9    20     1139           1845     1457           2210
 5      1005  2013     7    22      845           1600     1044           1815
 6       960  2013     4    10     1100           1900     1342           2211
 7       911  2013     3    17     2321            810      135           1020
 8       899  2013     6    27      959           1900     1236           2226
 9       898  2013     7    22     2257            759      121           1026
10       896  2013    12     5      756           1700     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>
```

```
flights |>
  slice_max(dep_delay, n = 5) |>
  relocate(dep_delay)
```

```
# A tibble: 5 x 19
  dep_delay  year month   day dep_time sched_dep_time arr_time sched_arr_time
      <dbl> <int> <int> <int>    <int>          <int>    <int>          <int>
1      1301  2013     1     9      641            900     1242           1530
2      1137  2013     6    15     1432           1935     1607           2120
3      1126  2013     1    10     1121           1635     1239           1810
4      1014  2013     9    20     1139           1845     1457           2210
```

```
5      1005  2013     7    22      845           1600     1044           1815
# 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>
```

Your text answer here.

18. Explain what `count()` does in terms of the dplyr verbs you just learned. What does the `sort` argument to `count()` do?

Answer

```
flights |>
  count(origin, dest, sort = FALSE) # sort = FALSE by   default
```

```
# A tibble: 224 x 3
   origin dest      n
   <chr>  <chr> <int>
 1 EWR    ALB     439
 2 EWR    ANC       8
 3 EWR    ATL    5022
 4 EWR    AUS     968
 5 EWR    AVL     265
 6 EWR    BDL     443
 7 EWR    BNA    2336
 8 EWR    BOS    5327
 9 EWR    BQN     297
10 EWR    BTV     931
# i 214 more rows
```

```
flights |>
  count(origin, dest, sort = TRUE)
```

```
# A tibble: 224 x 3
   origin dest      n
   <chr>  <chr> <int>
 1 JFK    LAX   11262
 2 LGA    ATL   10263
 3 LGA    ORD    8857
 4 JFK    SFO    8204
 5 LGA    CLT    6168
 6 EWR    ORD    6100
```

```
 7 JFK    BOS    5898
 8 LGA    MIA    5781
 9 JFK    MCO    5464
10 EWR    BOS    5327
# i 214 more rows
```

Your text answer here.

19. 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")
)
```

a. Write down what you think the output will look like, then check if you were correct, and describe what `group_by()` does.

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

Answer

```
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
```

Your text answer here.

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).

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

Answer

```
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
```

Your text answer here.

c. Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does.

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

Answer

```
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
```

Your text answer here.

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.

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

Answer

```
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
```

Your text answer here.

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)?

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

Answer

```
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
```

Your text answer here.

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?

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

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

Answer

```
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
```

```
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
```

Your text answer here.

4 Exercises (Chapter 4)

1. Restyle the following pipelines following the guidelines above.

```
flights|>filter(dest=="IAH")|>group_by(year,month,day)|>summarize(n=n(),
delay=mean(arr_delay,na.rm=TRUE))|>filter(n>10)

flights|>filter(carrier=="UA",dest%in%c("IAH","HOU"),sched_dep_time>
0900,sched_arr_time<2000)|>group_by(flight)|>summarize(delay=mean(
arr_delay,na.rm=TRUE),cancelled=sum(is.na(arr_delay)),n=n())|>filter(n>10)
```

Answer

```
# Your R code here
```

Provide an easy way to restyle the code.

5 Exercises (Chapter 5)

Tables

```
table1
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

```
table2
```

```
# A tibble: 12 x 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

```
table3
```

```
# A tibble: 6 x 3
  country      year rate
  <chr>       <dbl> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

1. For each of the sample tables, describe what each observation and each column represents.

   Answer

   Your text answer here.

2. Sketch out the process you'd use to calculate the `rate` for `table2` and `table3`. You will need to perform four operations:

   a. Extract the number of TB cases per country per year.
   b. Extract the matching population per country per year.
   c. Divide cases by population, and multiply by 10000.
   d. Store back in the appropriate place.

   You haven't yet learned all the functions you'd need to actually perform these operations, but you should still be able to think through the transformations you'd need.

   Answer

   ```
   table2 |>
     pivot_wider(names_from = type,
                 values_from = count) |>
     mutate(rate = cases / population * 10000)
   ```

   ```
   # A tibble: 6 x 5
     country      year  cases population  rate
     <chr>       <dbl>  <dbl>      <dbl> <dbl>
   1 Afghanistan  1999    745   19987071 0.373
   2 Afghanistan  2000   2666   20595360 1.29
   3 Brazil       1999  37737  172006362 2.19
   4 Brazil       2000  80488  174504898 4.61
   5 China        1999 212258 1272915272 1.67
   6 China        2000 213766 1280428583 1.67
   ```

```r
#
table3 |>
  separate_wider_delim(
    cols = rate,
    delim = "/",
    names = c("cases", "population"),
  ) |>
  mutate(
    cases = as.numeric(cases),
    population = as.numeric(population),
    rate = cases / population * 10000
  )
```

```
# A tibble: 6 x 5
  country      year  cases population  rate
  <chr>       <dbl>  <dbl>      <dbl> <dbl>
1 Afghanistan  1999    745   19987071 0.373
2 Afghanistan  2000   2666   20595360 1.29
3 Brazil       1999  37737  172006362 2.19
4 Brazil       2000  80488  174504898 4.61
5 China        1999 212258 1272915272 1.67
6 China        2000 213766 1280428583 1.67
```

For `table2`, we need to reshape the data to have a column for cases and a column for population and then divide the two to calculate the rate. A possible approach is shown above.
For `table3`, we need to separate cases and population into their own columns and then divide them. A possible approach is shown above.

6 Exercises (Chapter 6)

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

   > Answer
   >
   > *Your text answer here.*

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

   > Answer
   >
   > *Your text answer here.*

7 Exercises (Chapter 7)

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

   Answer

   Your text answer here.

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

   Answer

   Your text answer here.

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

   Answer

   Your text answer here.

4. Sometimes strings in a CSV file contain commas. To prevent them from causing problems, they need to be surrounded by a quoting character, like `"` or `'`. By default, `read_csv()` assumes that the quoting character will be `"`. To read the following text into a data frame, what argument to `read_csv()` do you need to specify?

   ```
   "x,y\n1,'a,b'"
   ```

   Answer

   We need to specify the `quote` argument.

   ```
   read_csv("x,y\n1,'a,b'", quote = "\'")
   ```

   ```
   Rows: 1 Columns: 2
   -- Column specification --------------------------------------------------------
   Delimiter: ","
   chr (1): y
   ```

```
dbl (1): x

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 1 x 2
      x y
  <dbl> <chr>
1     1 a,b
```

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

```
read_csv("a,b\n1,2,3\n4,5,6")
read_csv("a,b,c\n1,2\n1,2,3,4")
read_csv("a,b\n\"1")
read_csv("a,b\n1,2\na,b")
read_csv("a;b\n1;3")
```

Answer

```
read_csv("a,b\n1,2,3\n4,5,6")
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
Rows: 2 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (1): a
num (1): b

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 2 x 2
      a     b
  <dbl> <dbl>
1     1    23
2     4    56
```

There are only two column headers but three values in each row, so the last two

get merged.

Answer

```
read_csv("a,b,c\n1,2\n1,2,3,4")
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
Rows: 2 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): a, b
num (1): c

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 2 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1     1     2    NA
2     1     2    34
```

here are only three column headers, first row is missing a value in the last column so gets an NA there, the second row has four values so the last two get merge

Answer

```
read_csv("a,b\n\"1")
```

```
Rows: 0 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): a, b

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 0 x 2
# i 2 variables: a <chr>, b <chr>
```

No rows are read in.

Answer

```
read_csv("a,b\n1,2\na,b")
```

```
Rows: 2 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): a, b

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 2 x 2
  a     b
  <chr> <chr>
1 1     2
2 a     b
```

Each column has a numerical and a character value, so the column type is coerced to character.

Answer

```
read_csv("a;b\n1;3")
```

```
Rows: 1 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): a;b

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 1 x 1
  `a;b`
  <chr>
1 1;3
```

The delimiter is ; but it's not specified, therefore this is read in as a single-column data frame with a single observation.

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

```
set.seed(321)
annoying <- tibble(
  `1` = 1:10,
  `2` = `1` * 2 + rnorm(length(`1`))
)
```

a. Extracting the variable called `1`.

Answer

```
  annoying |>
    select(`1`)
```

```
# A tibble: 10 x 1
     `1`
   <int>
 1     1
 2     2
 3     3
 4     4
 5     5
 6     6
 7     7
 8     8
 9     9
10    10
```

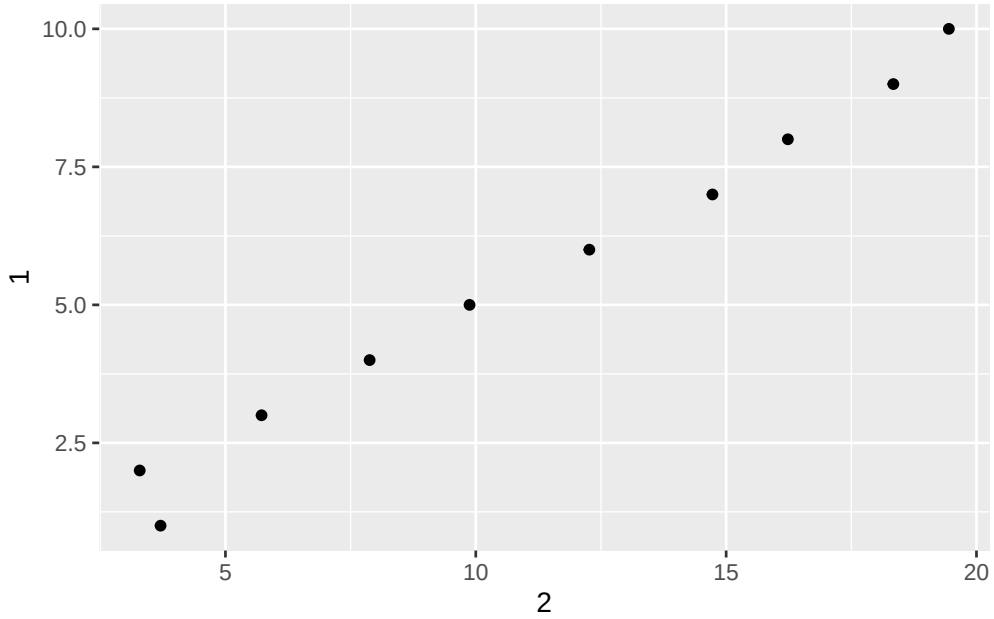
```
  # or
  annoying$`1`
```

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

b. Plotting a scatterplot of `1` vs. `2`.

Answer

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



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

Answer

```
annoying |>
  mutate(`3` = `2`/`1`)
```

```
# A tibble: 10 x 3
     `1`   `2`   `3`
   <int> <dbl> <dbl>
 1     1  3.70  3.70
 2     2  3.29  1.64
 3     3  5.72  1.91
 4     4  7.88  1.97
 5     5  9.88  1.98
 6     6 12.3   2.04
 7     7 14.7   2.10
 8     8 16.2   2.03
 9     9 18.3   2.04
10    10 19.4   1.94
```

d. Renaming the columns to `one`, `two`, and `three`.

Answer

```
  annoying |>
    mutate(`3` = `2`/`1`) |>
      rename(
      "one" = `1`,
      "two" = `2`,
      "three" = `3`
      )
```

```
# A tibble: 10 x 3
     one   two three
   <int> <dbl> <dbl>
 1     1  3.70  3.70
 2     2  3.29  1.64
 3     3  5.72  1.91
 4     4  7.88  1.97
 5     5  9.88  1.98
 6     6 12.3   2.04
 7     7 14.7   2.10
 8     8 16.2   2.03
 9     9 18.3   2.04
10    10 19.4   1.94
```

8 Exercises (Chapter 9)

1. Create a scatterplot of `hwy` vs. `displ` where the points are pink filled in triangles.

   Answer

   ```
   # Your R code here
   ```

2. Why did the following code not result in a plot with blue points?

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

   Answer

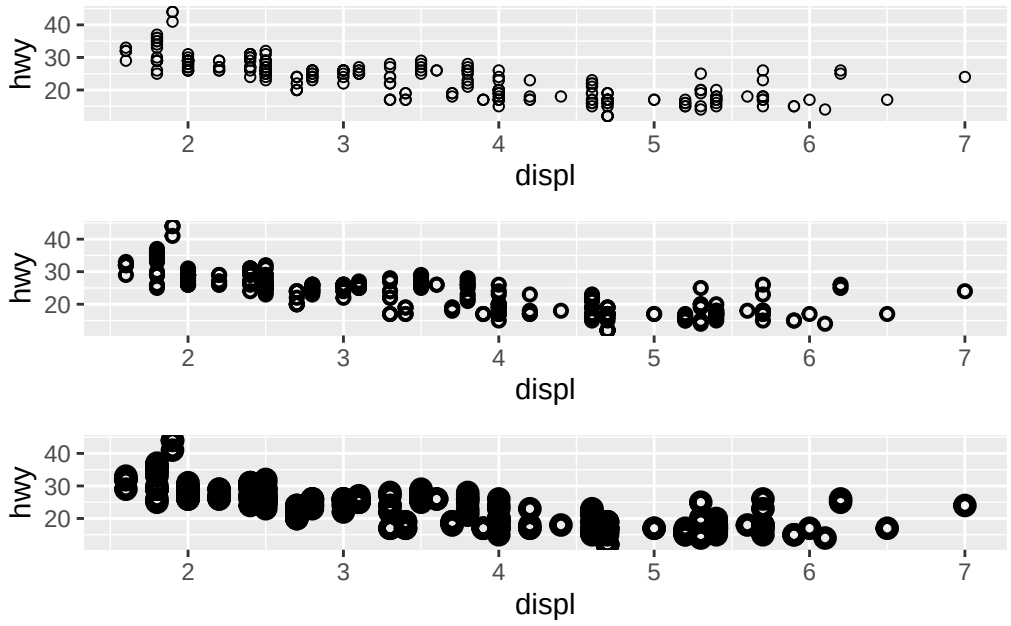
   ```
   # Proper R code here
   ```

   Your text answer here.

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

Answer

```
mpg |>
  ggplot(aes(x = displ, y = hwy)) +
    geom_point(shape = 21, stroke = 0.5) -> p1
mpg |>
  ggplot(aes(x = displ, y = hwy)) +
    geom_point(shape = 21, stroke = 1) -> p2
mpg |>
  ggplot(aes(x = displ, y = hwy)) +
    geom_point(shape = 21, stroke = 2) -> p3
library(patchwork)
p1 / p2 / p3
```

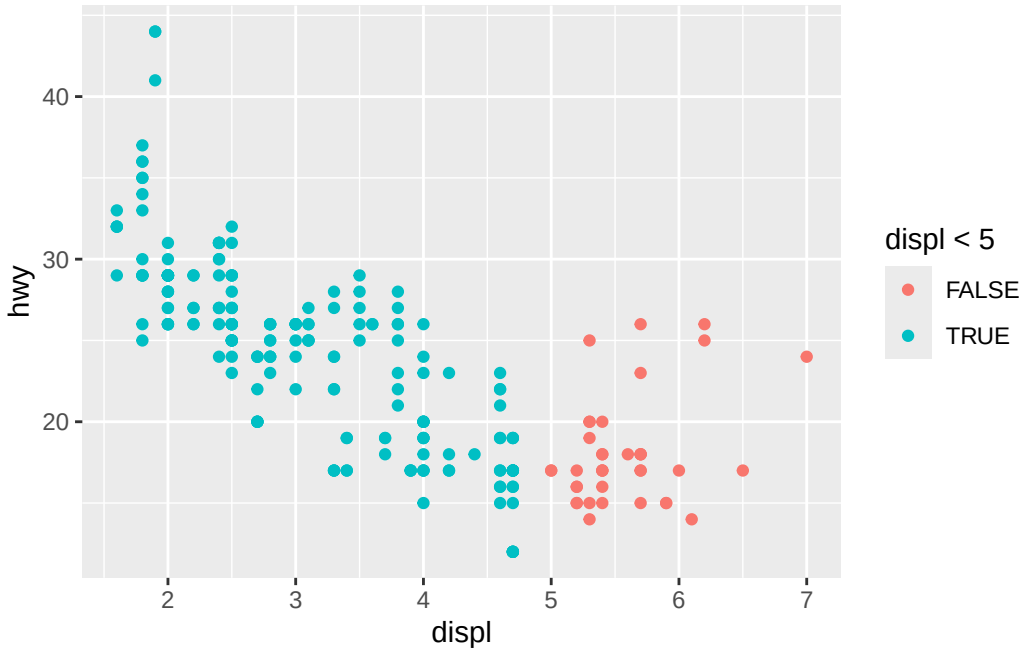


Your text answer here.

4. What happens if you map an aesthetic to something other than a variable name, like `aes(color = displ < 5)`? Note, you'll also need to specify x and y.

Answer

```
mpg |>
  ggplot(aes(x = displ, y = hwy, color = displ < 5)) +
    geom_point()
```



Your text answer here.

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

Answer

Your text answer here.

```
# R Code here
```

Answer

Your text answer here.

```
# R code here
```

Answer

Your text answer here.

```
# R code here
```

Answer

Your text answer here.

```
# Youe R code here
```

6. Earlier in this chapter we used `show.legend` without explaining it:

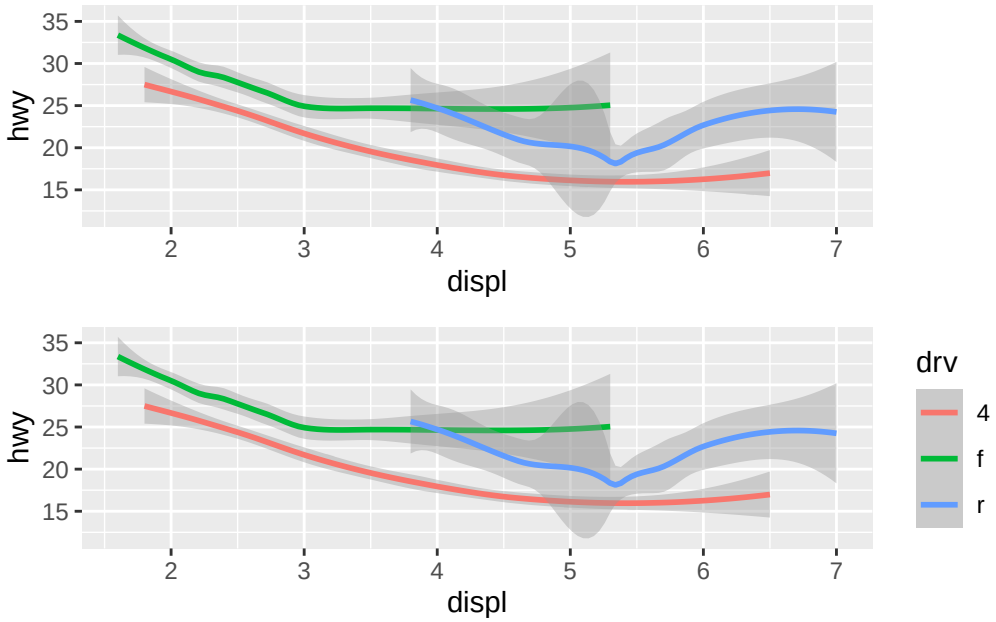
```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(color = drv), show.legend = FALSE)
```

What does `show.legend = FALSE` do here? What happens if you remove it? Why do you think we used it earlier?

Answer

Your text answer here.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(color = drv), show.legend = FALSE) -> p1
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(color = drv), show.legend = TRUE) -> p2
p1 / p2
```

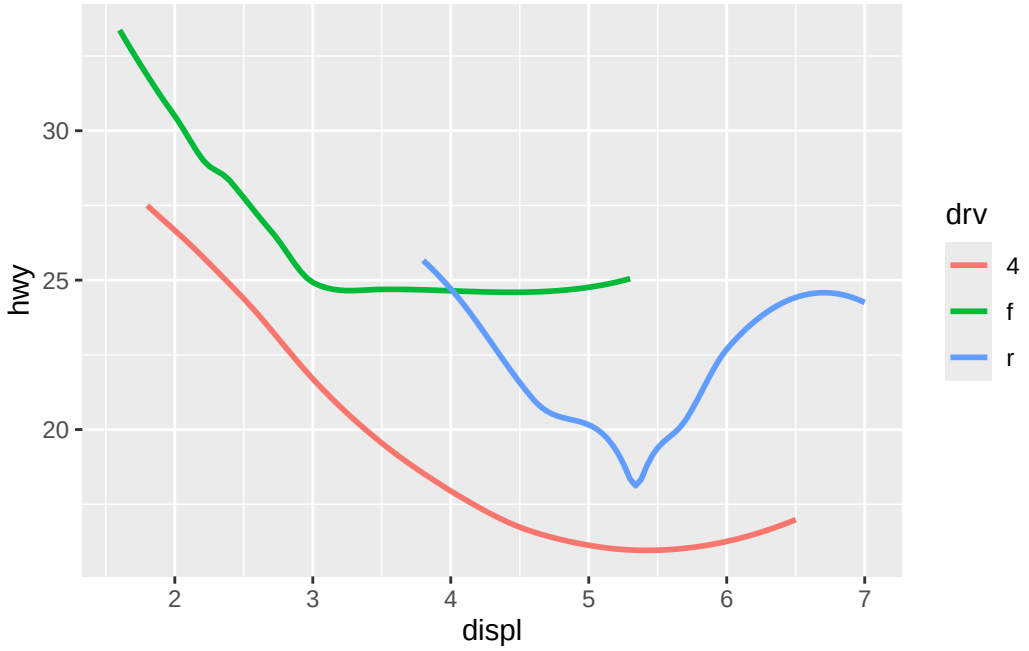


7. What does the `se` argument to `geom_smooth()` do?

Answer

Your text answer here.

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



8. Recreate the R code necessary to generate the following graphs. Note that wherever a categorical variable is used in the plot, it's `drv`.

Answer

The code for each of the plots is given below.

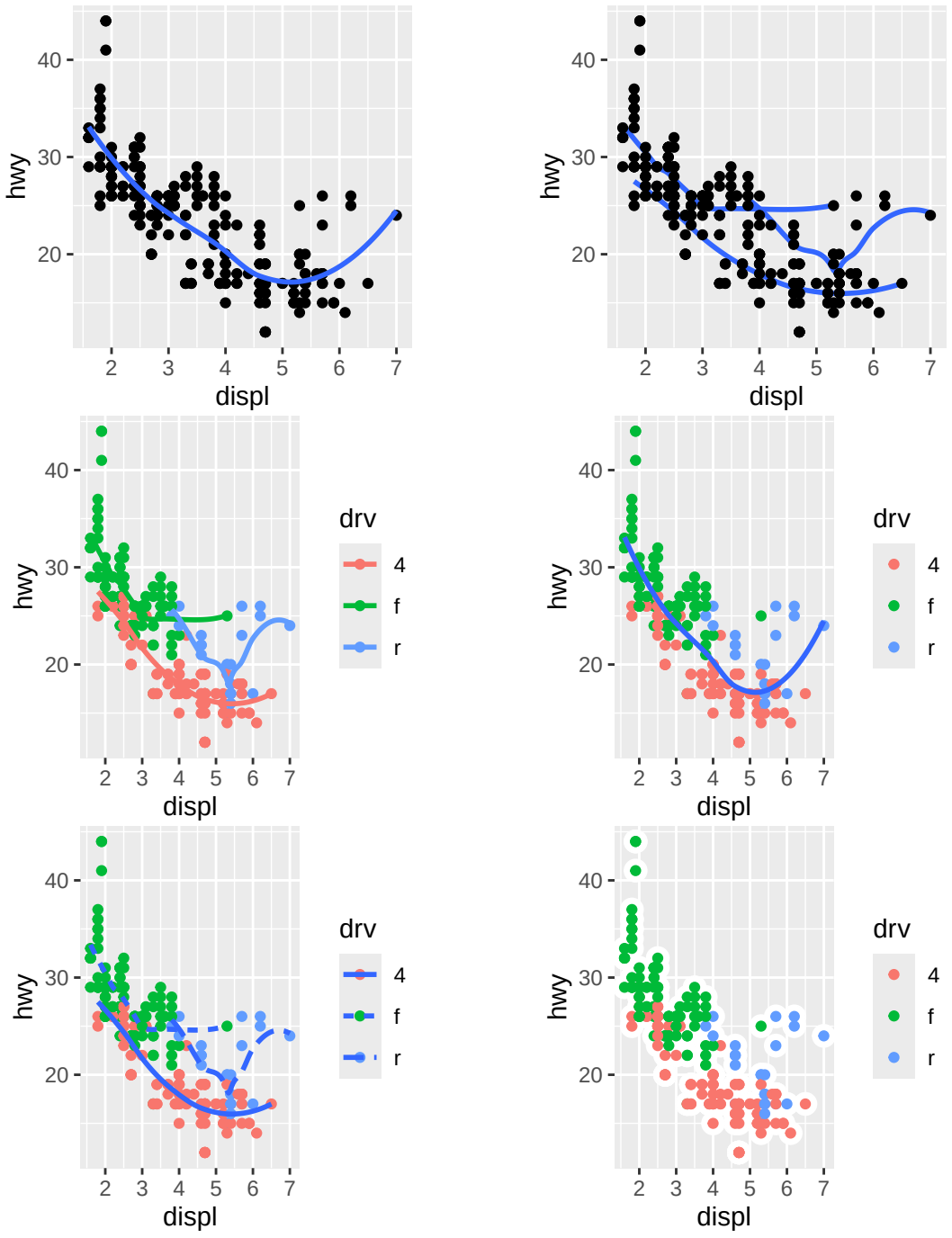
```
# Your R code here
```

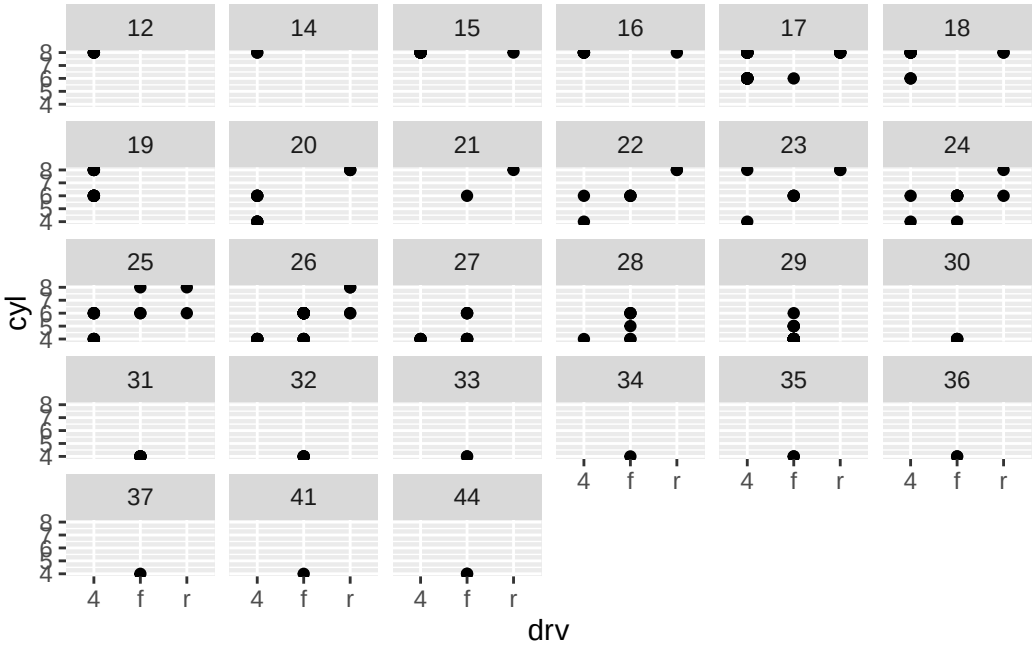
9. What happens if you facet on a continuous variable?

Answer

Your text answer here.

```
mpg |>
  ggplot(aes(x = drv, y = cyl)) +
  geom_point() +
  facet_wrap(~hwy)
```


hwy
displ
drv
4
f
r

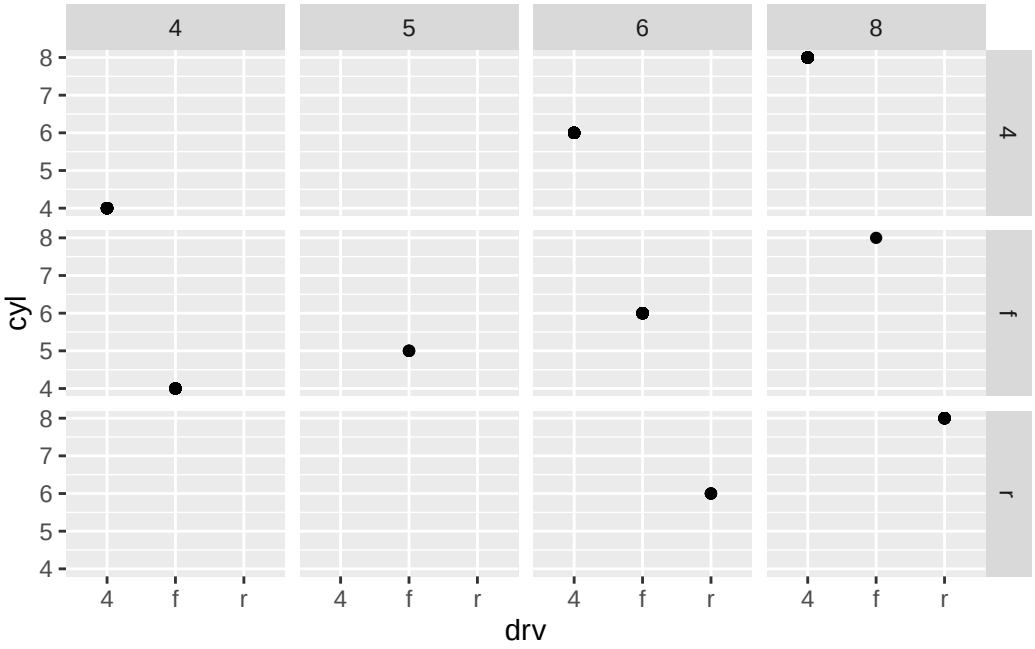


10. What do the empty cells in the plot above with `facet_grid(drv ~ cyl)` mean? Run the following code. How do they relate to the resulting plot?

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

Answer

```
ggplot(mpg) +
  geom_point(aes(x = drv, y = cyl)) +
  facet_grid(drv ~ cyl)
```



Your text answer here.

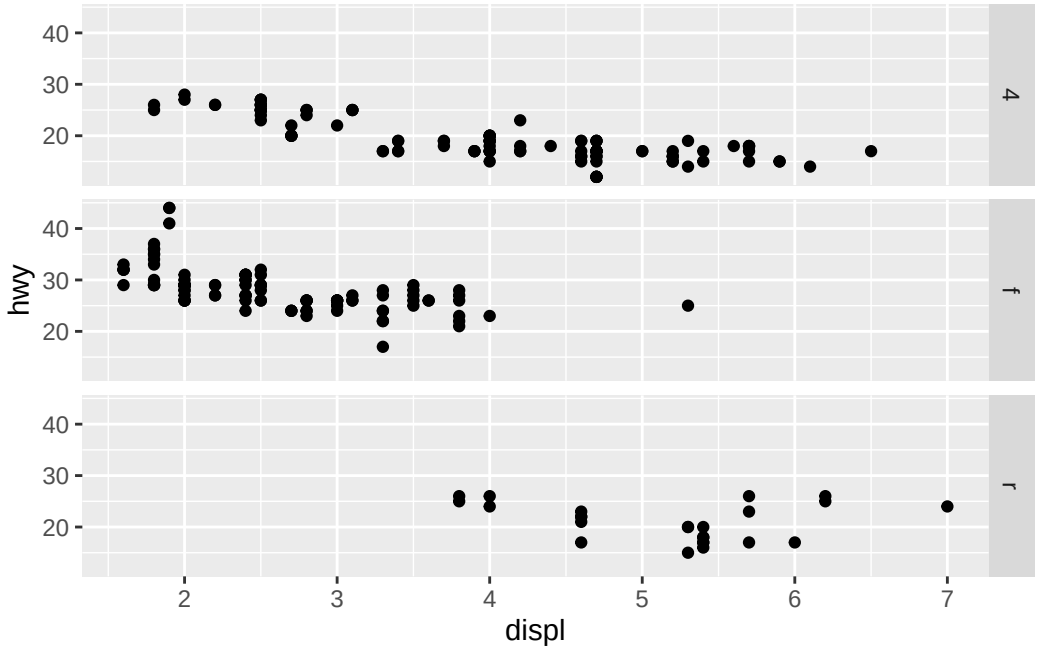
11. What plots does the following code make? What does `.` do?

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

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

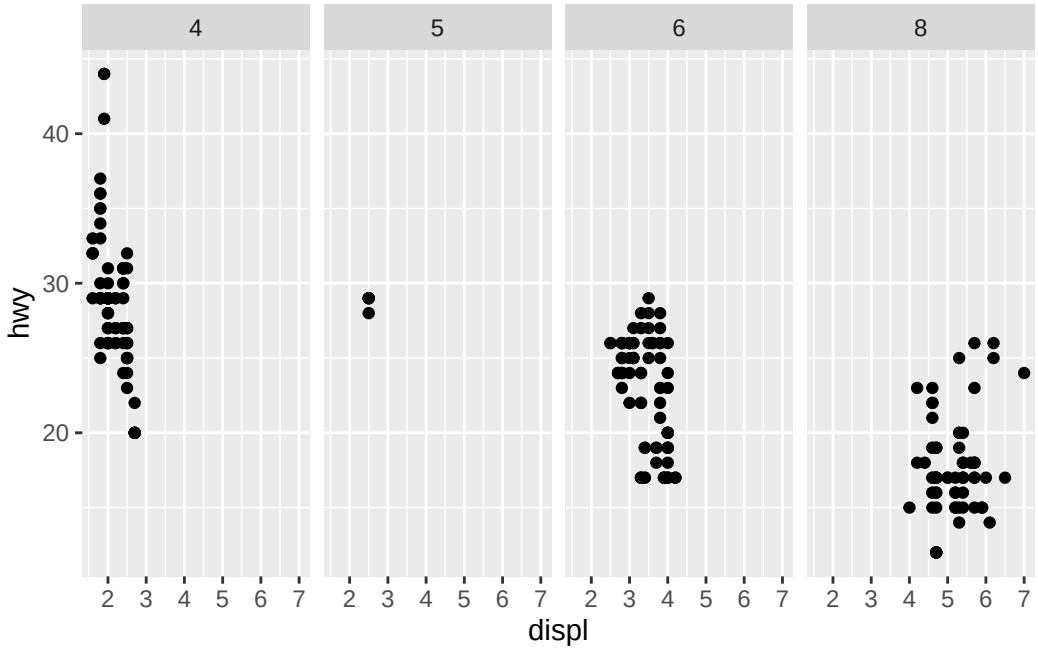
Answer

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



Your text answer here.

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



Your text answer here.

12. Take the first faceted plot in this section:

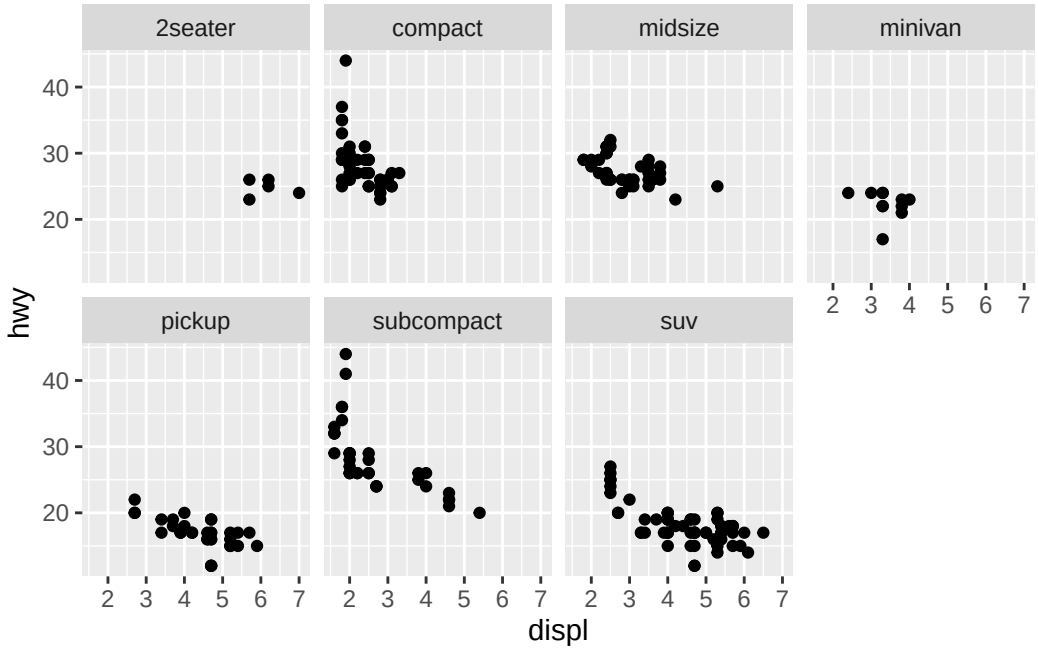
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ cyl, nrow = 2)
```

What are the advantages to using faceting instead of the color aesthetic? What are the disadvantages? How might the balance change if you had a larger dataset?

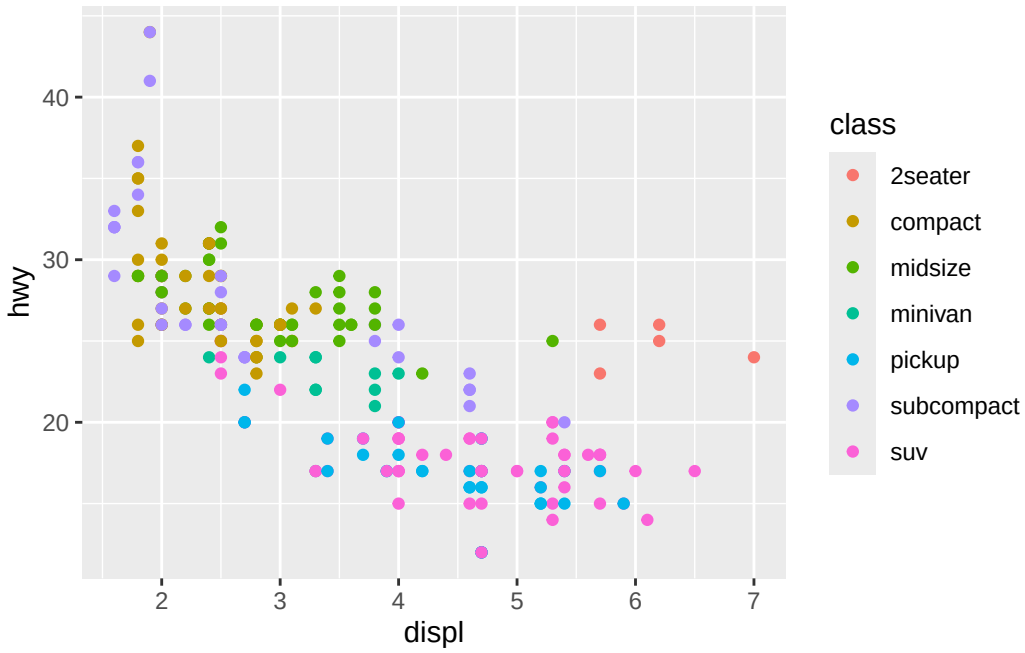
Answer

Your text answer here.

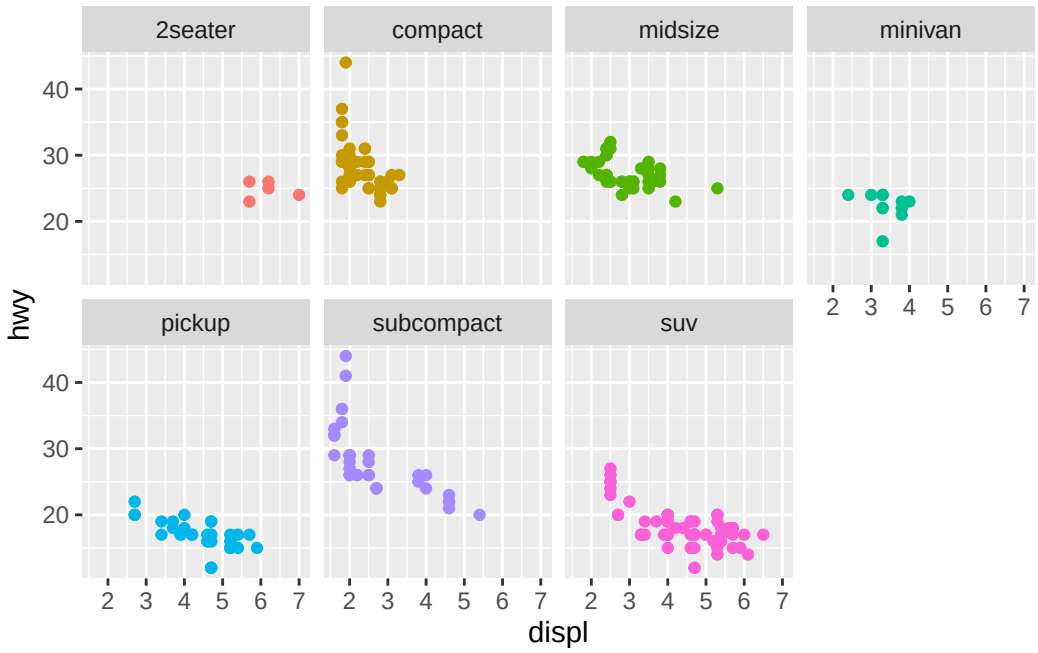
```
# facet
ggplot(mpg) +
 geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



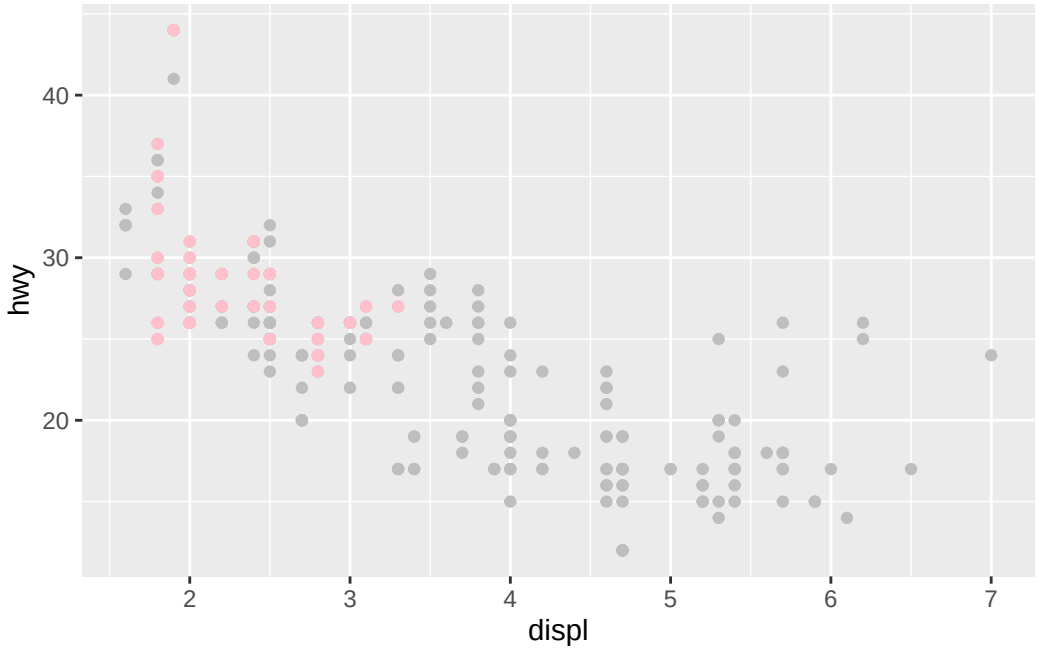

```
# color
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, color = class))
```




```
# both
ggplot(mpg) +
  geom_point(
    aes(x = displ, y = hwy, color = class),
    show.legend = FALSE) +
 facet_wrap(~ class, nrow = 2)
```




```
# highlighting
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(color = "gray") +
  geom_point(
    data = mpg |> filter(class == "compact"),
    color = "pink"
  )
```



13. 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` arguments?

Answer

Your text answer here.

14. Which of the following plots makes it easier to compare engine size (`displ`) across cars with different drive trains? What does this say about when to place a faceting variable across rows or columns?

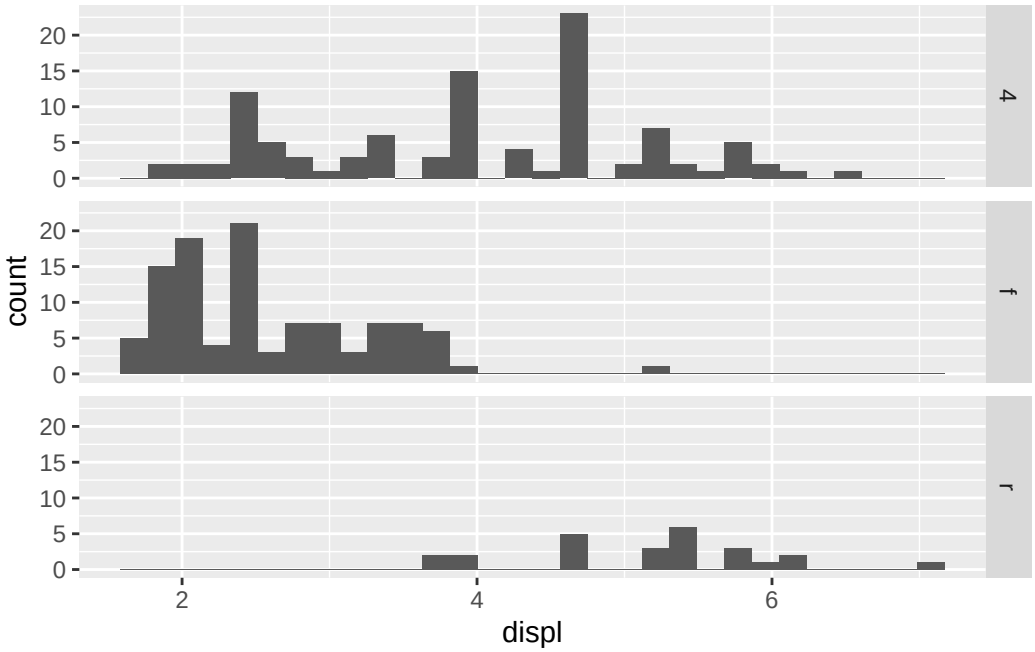
```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(drv ~ .)
```

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(. ~ drv)
```

Answer

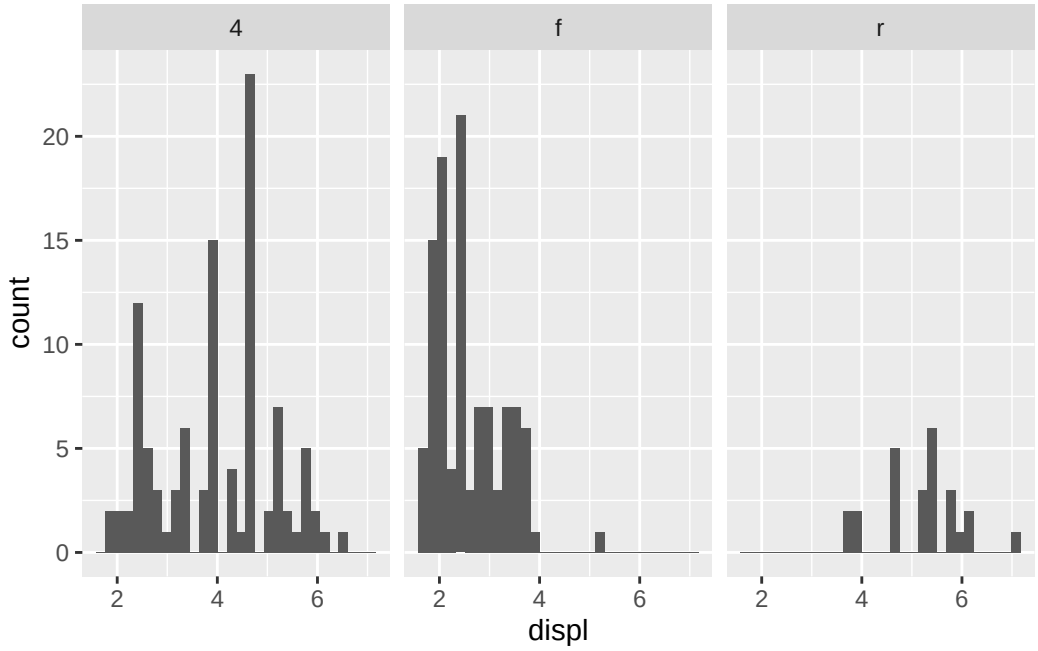
```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(drv ~ .)
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(. ~ drv)
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



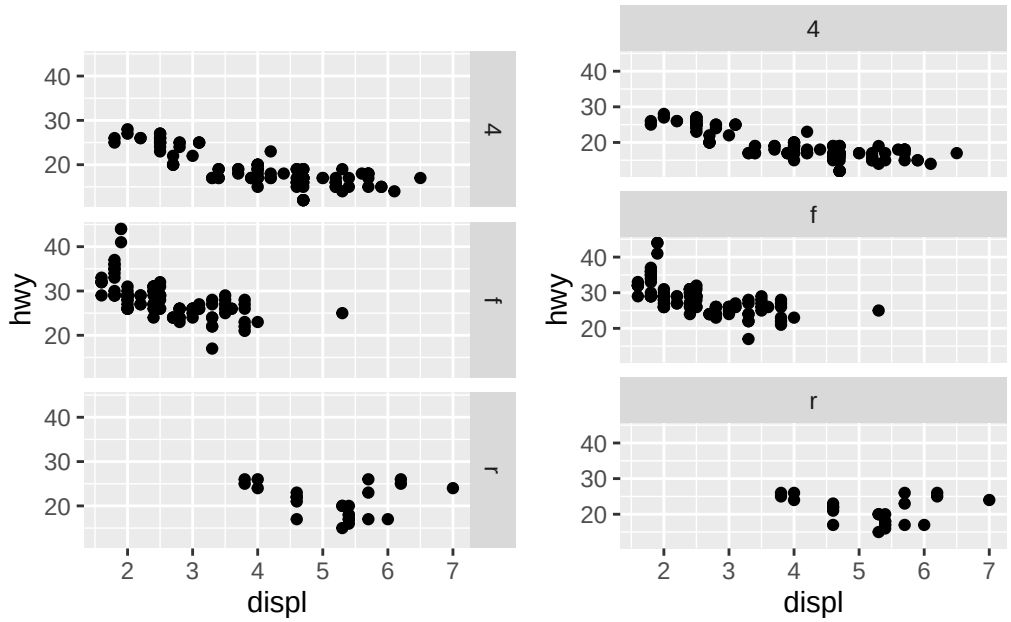
Your text answer here.

15. Recreate the following plot using `facet_wrap()` instead of `facet_grid()`. How do the positions of the facet labels change?

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

Answer

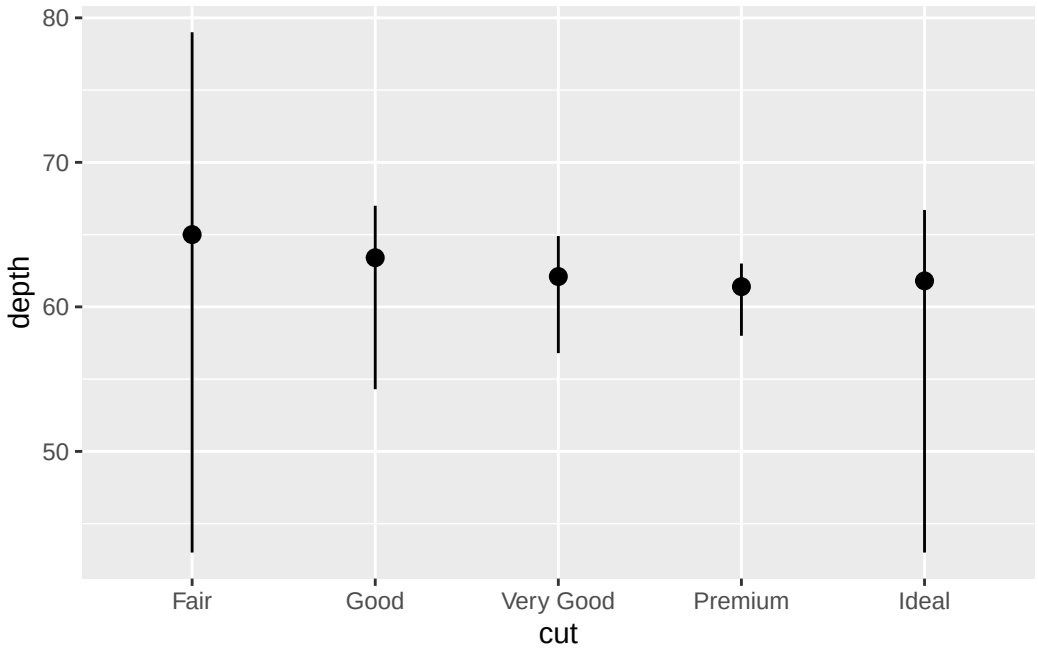
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(drv ~ .) -> p1
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~drv, nrow = 3) -> p2
p1 + p2
```



Your text answer here.

16. 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?

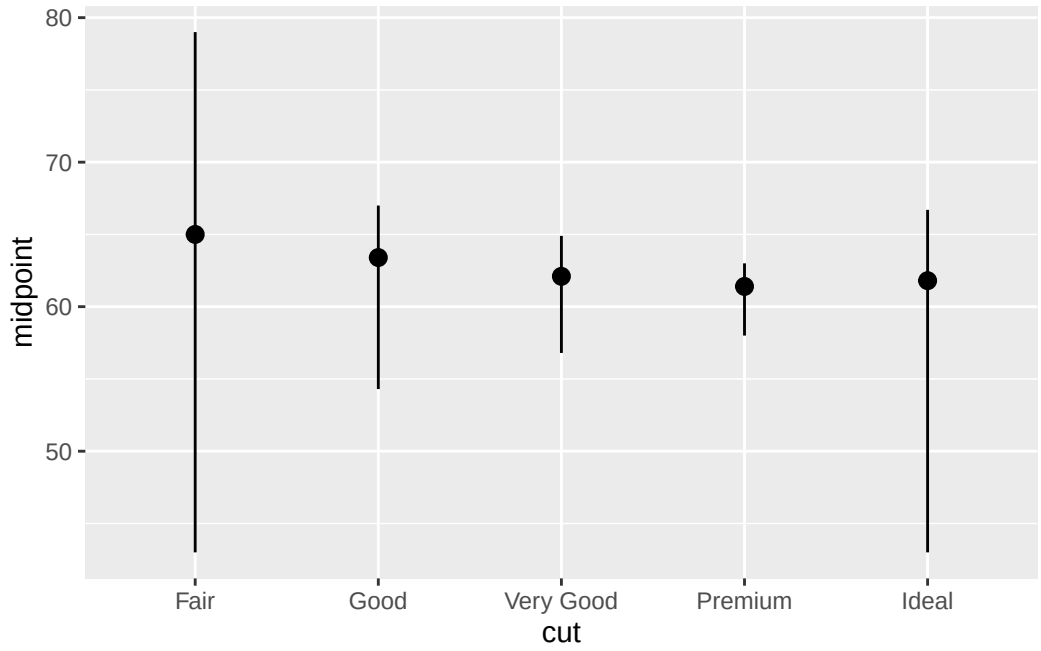
```
ggplot(diamonds) +
  stat_summary(
    aes(x = cut, y = depth),
    fun.min = min,
    fun.max = max,
    fun = median
  )
```



Answer

Your text answer here.

```
diamonds |>
  group_by(cut) |>
  summarize(
    lower = min(depth),
    upper = max(depth),
    midpoint = median(depth)
  ) |>
  ggplot(aes(x = cut, y = midpoint)) +
  geom_pointrange(aes(ymin = lower, ymax = upper))
```



17. What does `geom_col()` do? How is it different from `geom_bar()`?

Answer

Your text answer here.

18. Most geoms and stats come in pairs that are almost always used in concert. Make a list of all the pairs. What do they have in common? (Hint: Read through the documentation.)

Answer

Geoms and stats that are almost always used in concert are listed below:

geom	stat
`geom_bar()`	`stat_count()`
`geom_bin2d()`	`stat_bin_2d()`
`geom_boxplot()`	`stat_boxplot()`
`geom_contour_filled()`	`stat_contour_filled()`
`geom_contour()`	`stat_contour()`
`geom_count()`	`stat_sum()`
`geom_density_2d()`	`stat_density_2d()`
`geom_density()`	`stat_density()`
`geom_dotplot()`	`stat_bindot()`

geom	stat
geom_function()	stat_function()
geom_sf()	stat_sf()
geom_sf()	stat_sf()
geom_smooth()	stat_smooth()
geom_violin()	stat_ydensity()
geom_hex()	stat_bin_hex()
geom_qq_line()	stat_qq_line()
geom_qq()	stat_qq()
geom_quantile()	stat_quantile()

19. What variables does `stat_smooth()` compute? What arguments control its behavior?

> Answer
>
> `stat_smooth()` computes the following variables:
>
> - `y` or `x`: Predicted value
> - `ymin` or `xmin`: Lower pointwise confidence interval around the mean
> - `ymax` or `xmax`: Upper pointwise confidence interval around the mean
> - `se`: Standard error

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

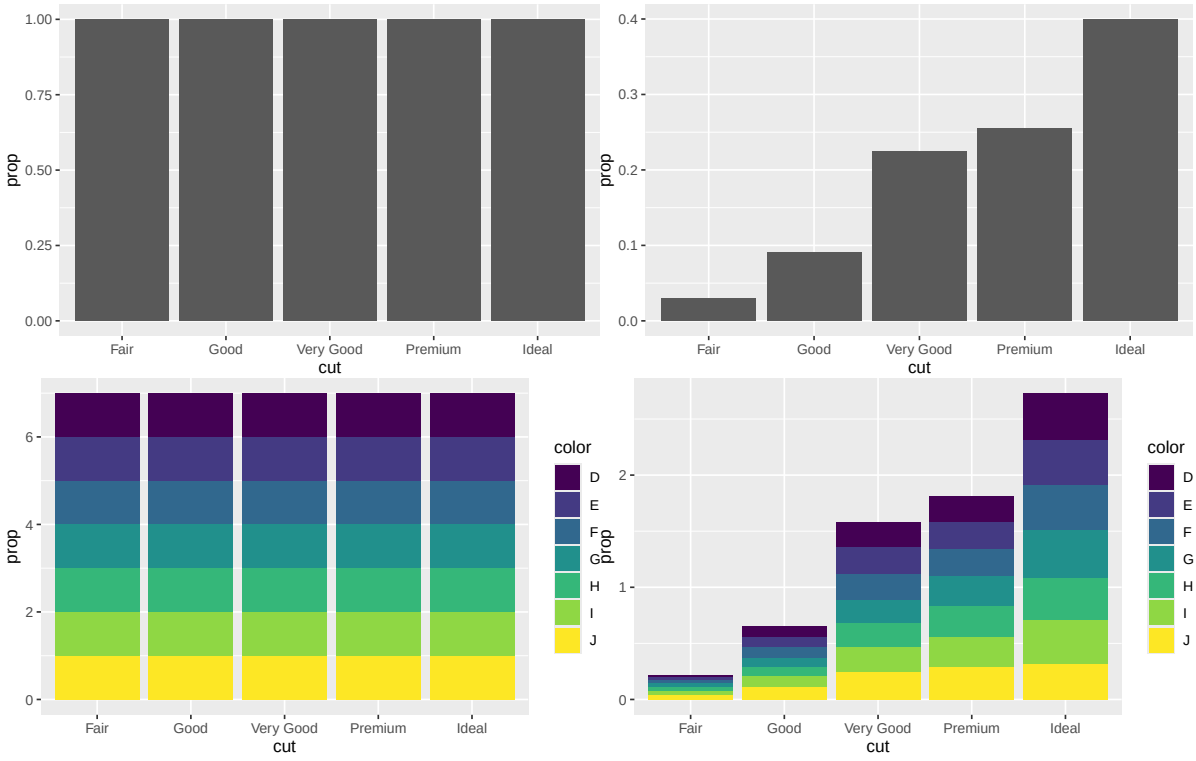
```
ggplot(diamonds, aes(x = cut, y = after_stat(prop))) +
  geom_bar()
```

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

> Answer
>
> *Your text answer here.*

```
# one variable
ggplot(diamonds, aes(x = cut,
                     y = after_stat(prop))) +
  geom_bar()
ggplot(diamonds, aes(x = cut,
                     y = after_stat(prop),
```

```
                      group = 1)) +
  geom_bar()
# two variables
ggplot(diamonds, aes(x = cut,
                      fill = color,
                      y = after_stat(prop))) +
  geom_bar()
ggplot(diamonds, aes(x = cut,
                      fill = color,
                      y = after_stat(prop),
                      group = color)) +
  geom_bar()
```



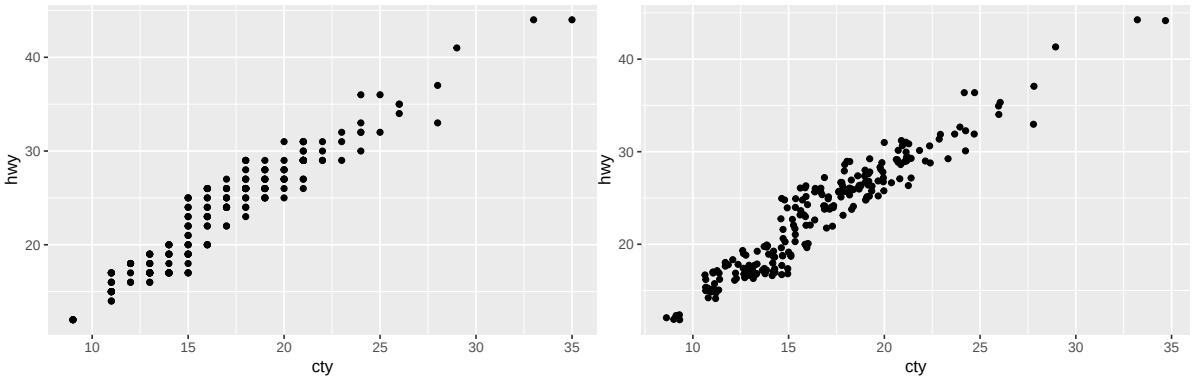
21. What is the problem with the following plot? How could you improve it?

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

Answer

Your text answer here.

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



22. What, if anything, is the difference between the two plots? Why?

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

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(position = "identity")
```

Answer

Your text answer here.

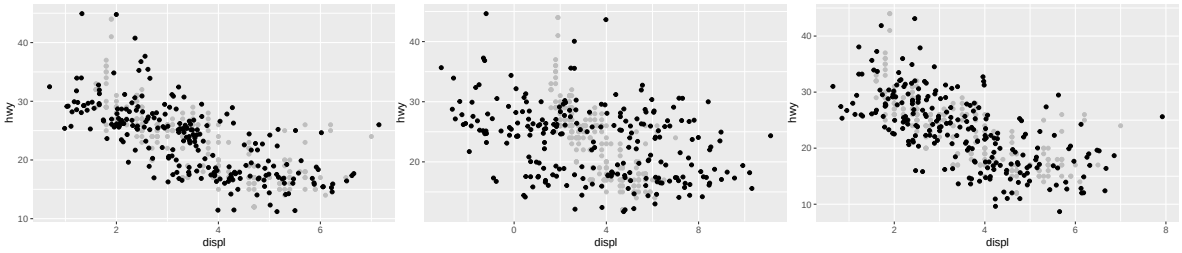
```
# Your R code here
```

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

Answer

Your text answer here.

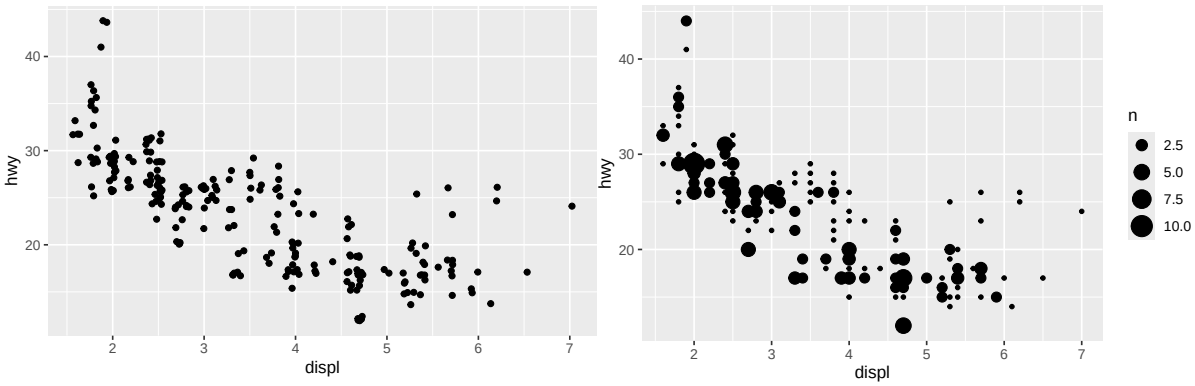
```
set.seed(321)
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(color = "gray") +
  geom_jitter(height = 1, width = 1)
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(color = "gray") +
  geom_jitter(height = 1, width = 5)
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(color = "gray") +
  geom_jitter(height = 5, width = 1)
```



24. Compare and contrast `geom_jitter()` with `geom_count()`.

> Answer
>
> *Your text answer here.*

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter()
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_count()
```

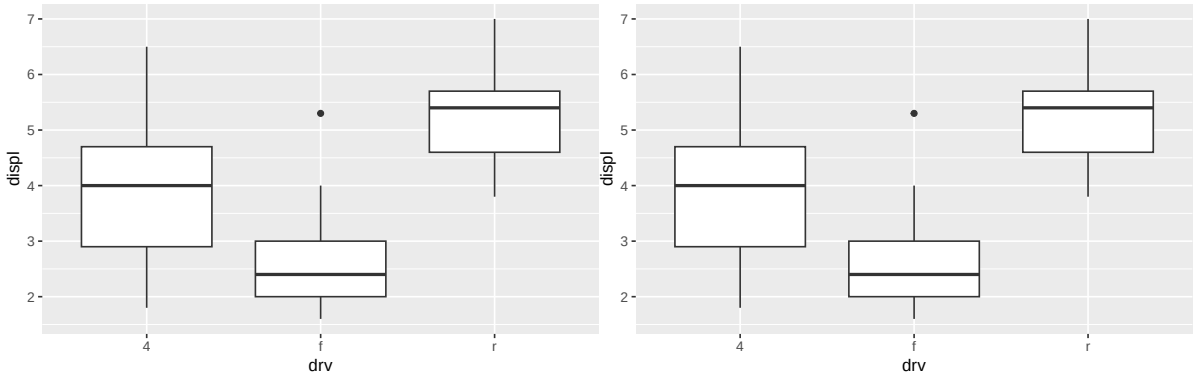


25. What's the default position adjustment for `geom_boxplot()`? Create a visualization of

the `mpg` dataset that demonstrates it.

> Answer
>
> *Your text answer here.*

```
ggplot(mpg, aes(x = drv, y = displ)) +
  geom_boxplot()
ggplot(mpg, aes(x = drv, y = displ)) +
  geom_boxplot(position = "dodge2")
```



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

> Answer
>
> *Your text answer here.*

```
# Your R code here
```

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

> Answer
>
> *Your text answer here.*

28. What does the following plot 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()
```

Answer

Your text answer here.

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