Group Project 3 - Data Carpentry Data Transformations

Group 3: Ryo Iwata, Stephan Quintin, Zachary Strickland

Group 3 members:

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

Group Assignment 3

Instructions - There are fewer assigned problems this time because you are expected to spend time going back and carefully re-reading (r4ds 2e ch 1-19, but elide over ch 8). **Be sure to go back and carefully read all chapters previously covered as well as those covered in these exercises. It is important that you get to a point of being able to relatively quickly approach whatever data problem comes your way, and this is the foundation for that.**

Ch 11

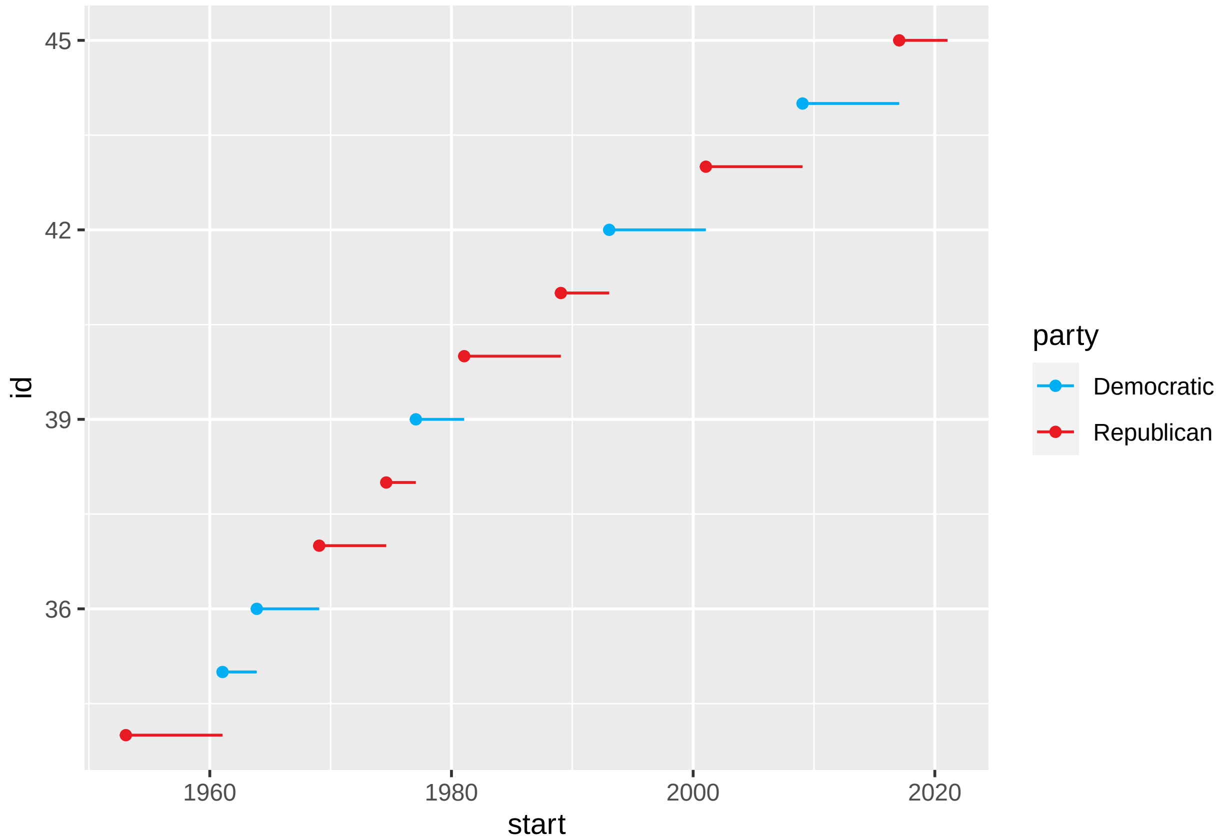
11.4.6

3

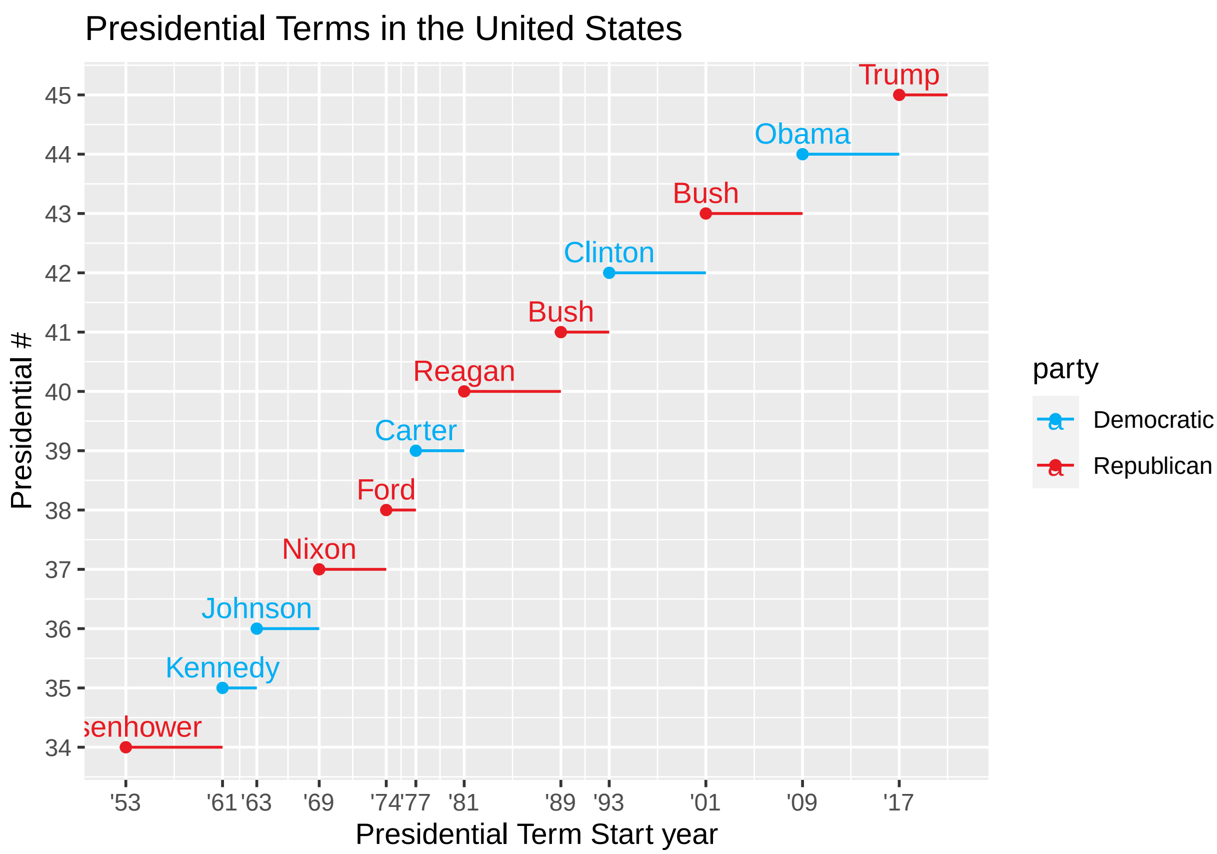
Change the display of the presidential terms by:

Combining the two variants that customize colors and x axis breaks. Improving the display of the y axis. Labelling each term with the name of the president. Adding informative plot labels. Placing breaks every 4 years (this is trickier than it seems!).

```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(values = c(Republican = "#E81B23", Democratic = "#00AEF3"))
```



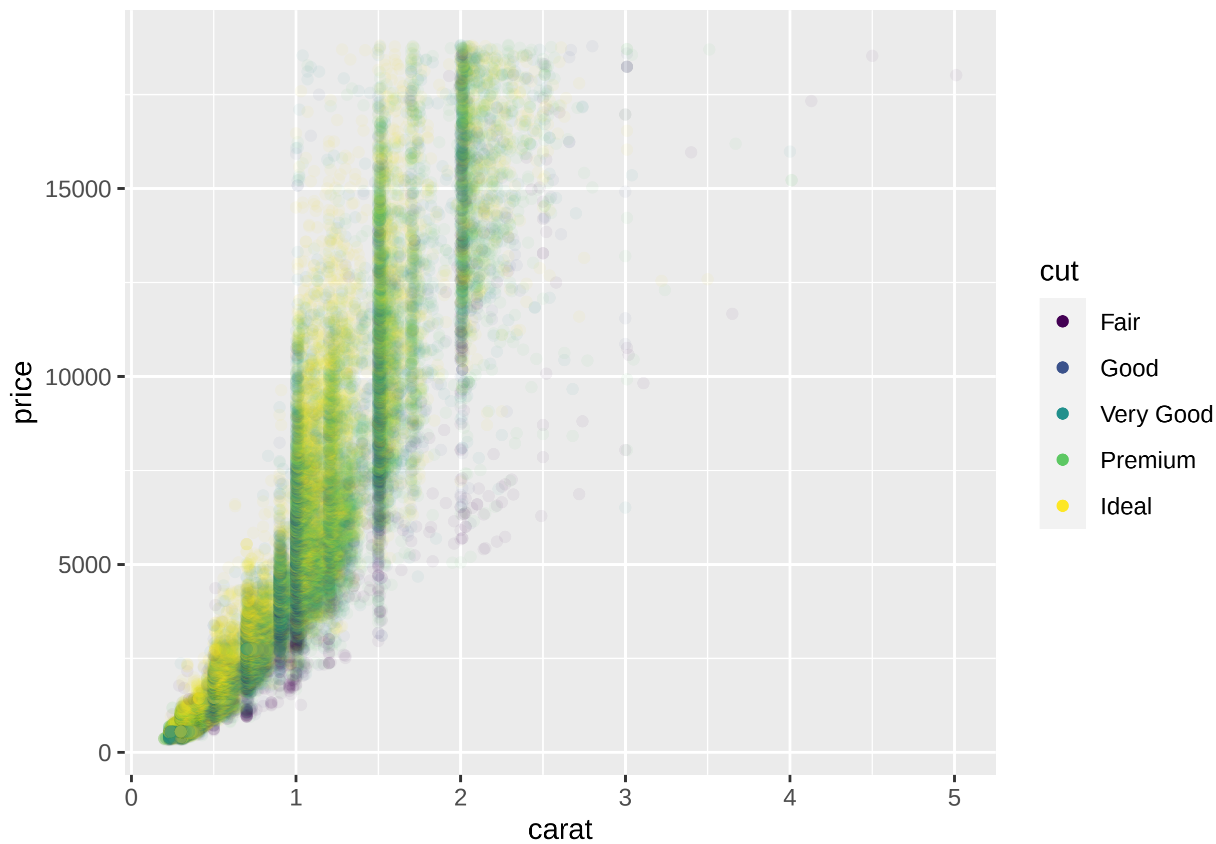

```r
numbered_presidential <- presidential |>
  mutate(id = 33 + row_number())

numbered_presidential |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  labs(title = "Presidential Terms in the United States",
       x = "Year",
       y = "President #") +
  # Combining the two variants that customize colors and x axis breaks.
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(values = c(Republican = "#E81B23", Democratic = "#00AEF3")) +
  # Labelling each term with the name of the president.
  geom_text(aes(label = name), vjust = -0.5, hjust = 0.5) +
  # Placing breaks every 4 years (this is trickier than it seems!).
  scale_x_date(name = "Presidential Term Start year",
               breaks = numbered_presidential$start, date_labels = "'%y") +
  # Improving the display of the y axis.
  scale_y_continuous(name = "Presidential #",
               breaks = numbered_presidential$id)
```



4

First, create the following plot. Then, modify the code using override.aes to make the legend easier to see.

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(aes(color = cut), alpha = 1/20) +
  # Making the transparency lower
  guides(colour = guide_legend(override.aes = list(alpha = 1)))
```

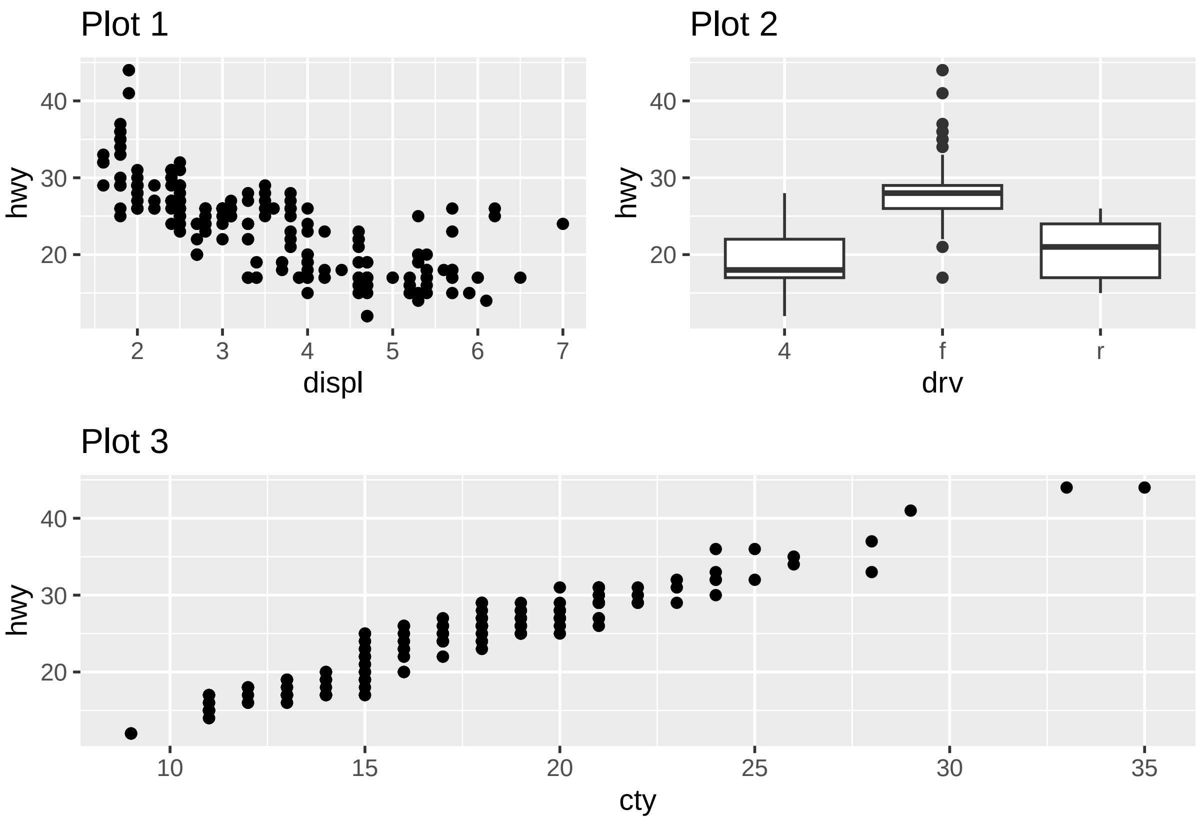


11.6.1

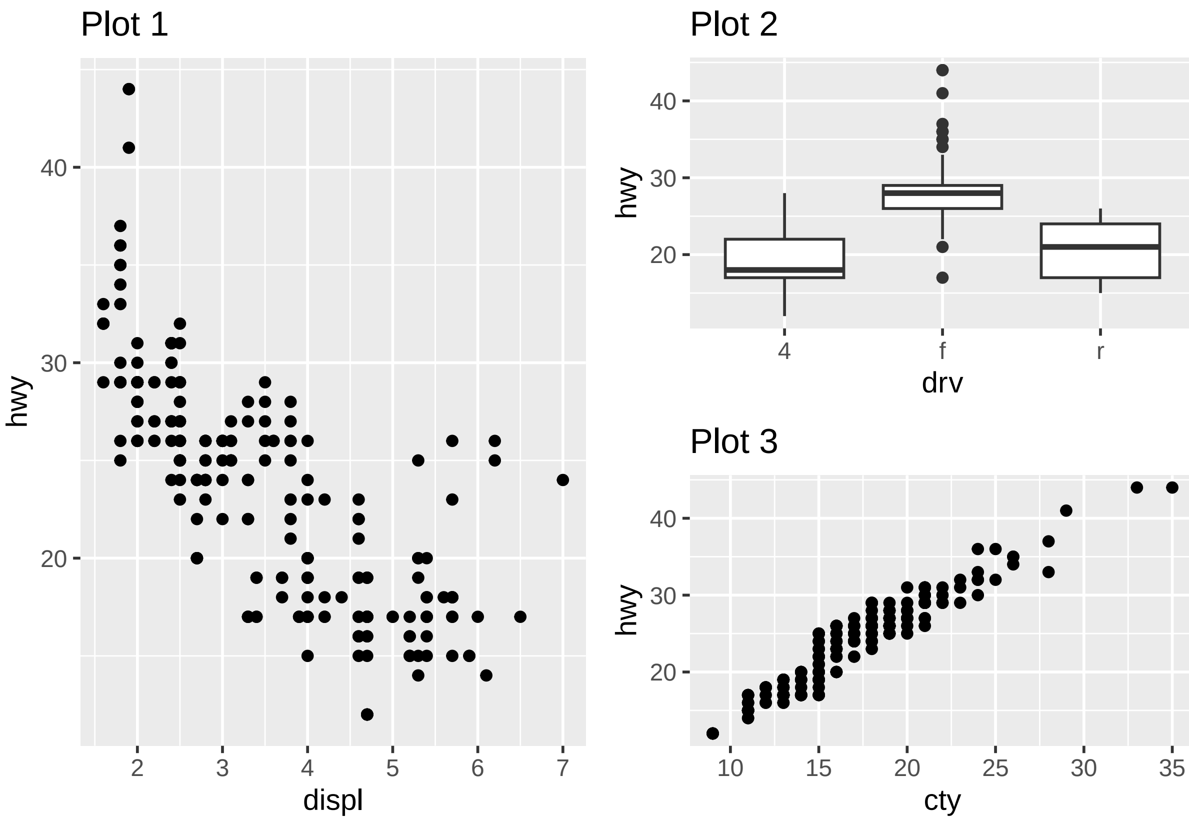
1 What happens if you omit the parentheses in the following plot layout. Can you explain why this happens?

Answer: Omitting the parentheses changes the order of operation of the plot layout. Instead of p3 being placed below both p1 and p2, it is only under p2 with p1 to the next on the left.

```
p1 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  labs(title = "Plot 1")
p2 <- ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_boxplot() +
  labs(title = "Plot 2")
p3 <- ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point() +
  labs(title = "Plot 3")

(p1 | p2) / p3
```


Plot 1
hwy
displ
40
30
20
2
3
4
5
6
7
Plot 2
hwy
drv
40
30
20
4
f
r
Plot 3
hwy
cty
40
30
20
10
15
20
25
30
35

```
p1 | p2 / p3
```



2

Using the three plots from the previous exercise, recreate the following patchwork.

```
p1 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  labs(tag = "Fig A:", title = "Plot 1")
p2 <- ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_boxplot() +
  labs(tag = "Fig B:", title = "Plot 2")
p3 <- ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point() +
  labs(tag = "Fig C:", title = "Plot 3")

p1 / (p2 | p3)
```

Fig A:

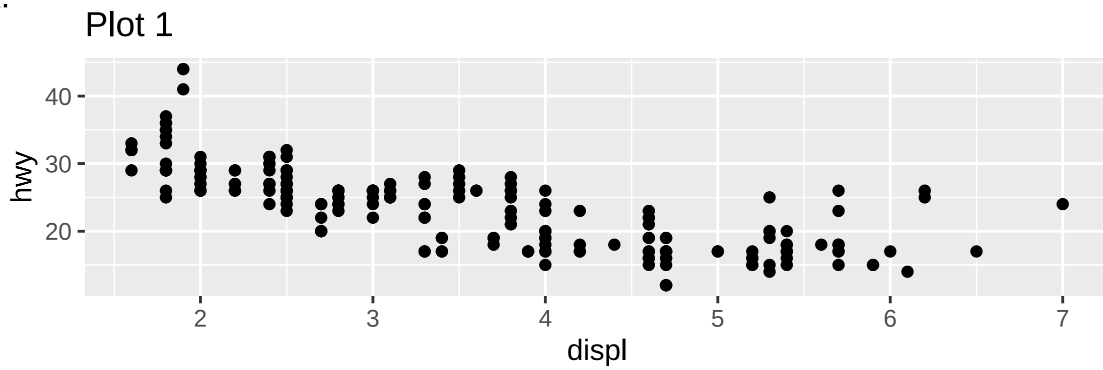


Fig B:

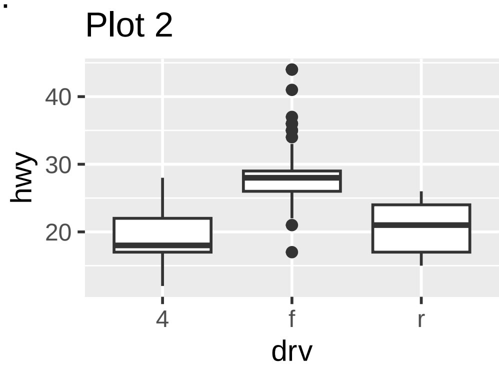
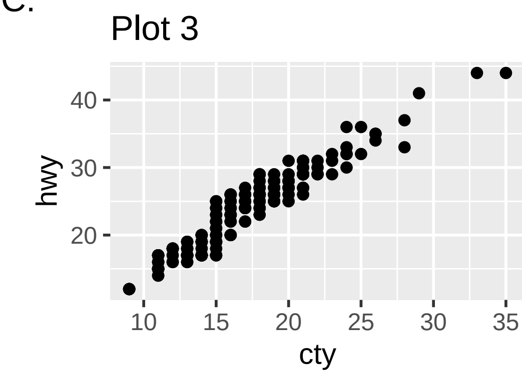


Fig C:



12.2.4

1

How does dplyr::near() work? Type near to see the source code. Is sqrt(2)^2 near 2?

Answer: near() is a way of comparing if two numbers are (pairwise) equal which includes a tolerance into its calculation which accounts for differences between the two numbers that are allowed. This is useful for seeing if sqrt(2)^2 near 2, because there are a limited number of numbers after the decimal that is stored. So the computer will think that this equation is not true because there are missing numbers when storing the `sqrt(2)`

```
sqrt(2)^2 == 2
```

```
## [1] FALSE
```

```
near(sqrt(2) ^ 2, 2)
```

```
## [1] TRUE
```

2

Use mutate(), is.na(), and count() together to describe how the missing values in dep_time, sched_dep_time and dep_delay are connected.

Answer: All flights have a `sched_dep_time`. `dep_time` and `dep_delay` are always missing together.

```
#This demonstrates that every flight has a scheduled departure time
flights |>
  count(is.na(sched_dep_time))
```

```
## # A tibble: 1 x 2
##   `is.na(sched_dep_time)`      n
##   <lgl>                    <int>
## 1 FALSE                   336776
```

```
# There are same amount of missing values in dep_time and dep_delay
flights |>
  count(is.na(dep_time))
```

```
## # A tibble: 2 x 2
##   `is.na(dep_time)`      n
##   <lgl>              <int>
## 1 FALSE             328521
## 2 TRUE                8255
```

```
flights |>
  count(is.na(dep_delay))
```

```
## # A tibble: 2 x 2
##   `is.na(dep_delay)`      n
##   <lgl>               <int>
## 1 FALSE              328521
## 2 TRUE                 8255
```

```
# This demonstrates that whenever a dep_time is missig, del_delay is missing as well.
flights |>
  mutate(dep_time_is.na = is.na(dep_time),
         sched_dep_time_is.na = is.na(sched_dep_time),
         dep_delay_is.na = is.na(dep_delay)) |>
  count(dep_time_is.na, sched_dep_time_is.na, dep_delay_is.na)
```

```
## # A tibble: 2 x 4
##   dep_time_is.na sched_dep_time_is.na dep_delay_is.na      n
##   <lgl>          <lgl>                <lgl>            <int>
## 1 FALSE          FALSE                FALSE           328521
## 2 TRUE           FALSE                TRUE              8255
```

12.3.4

1

Find all flights where arr_delay is missing but dep_delay is not. Find all flights where neither arr_time nor sched_arr_time are missing, but arr_delay is.

```
filter(flights, is.na(arr_delay) & !is.na(dep_delay)) |>
  relocate(arr_delay) |>
  relocate(dep_delay)
```

```
## # A tibble: 1,175 x 19
##    dep_delay arr_delay  year month   day dep_time sched_dep_time arr_time
##        <dbl>     <dbl> <int> <int> <int>    <int>          <int>    <int>
##  1        -5        NA  2013     1     1     1525           1530     1934
##  2        29        NA  2013     1     1     1528           1459     2002
##  3        -5        NA  2013     1     1     1740           1745     2158
##  4        29        NA  2013     1     1     1807           1738     2251
##  5        59        NA  2013     1     1     1939           1840       29
```

```
##  6        22         NA  2013     1     1     1952          1930     2358
##  7        46         NA  2013     1     1     2016          1930       NA
##  8        43         NA  2013     1     2      905           822     1313
##  9       120         NA  2013     1     2     1125           925     1445
## 10         8         NA  2013     1     2     1848          1840     2333
## # i 1,165 more rows
## # i 11 more variables: sched_arr_time <int>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
filter(flights, is.na(arr_delay) & !is.na(dep_delay) & !is.na(sched_arr_time)) |>
  relocate(arr_delay) |>
  relocate(dep_delay) |>
  relocate(sched_arr_time)
```

```
## # A tibble: 1,175 x 19
##    sched_arr_time dep_delay arr_delay  year month   day dep_time sched_dep_time
##             <int>     <dbl>     <dbl> <int> <int> <int>    <int>          <int>
##  1           1805        -5        NA  2013     1     1     1525           1530
##  2           1647        29        NA  2013     1     1     1528           1459
##  3           2020        -5        NA  2013     1     1     1740           1745
##  4           2103        29        NA  2013     1     1     1807           1738
##  5           2151        59        NA  2013     1     1     1939           1840
##  6           2207        22        NA  2013     1     1     1952           1930
##  7           2220        46        NA  2013     1     1     2016           1930
##  8           1045        43        NA  2013     1     2      905            822
##  9           1146       120        NA  2013     1     2     1125            925
## 10           2151         8        NA  2013     1     2     1848           1840
## # i 1,165 more rows
## # i 11 more variables: arr_time <int>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

2

How many flights have a missing dep_time? What other variables are missing in these rows? What might these rows represent?

Answer: The number of flights with a missing `dep_time` is 8255. The other missing variables for flights that have a missing `dep_time` is `dep_delay` `arr_time` `arr_delay` `tailnum` `air_time` . This likely represents flights that are cancelled.

```
# Create a subset of flights with missing dep_time
flights_missing_dep_time<- flights |>
  filter(is.na(dep_time))

# Count the number of flights with missing dep_time
num_flights_missing_dep_time <- flights_missing_dep_time |>
  nrow()

num_flights_missing_dep_time
```

```
## [1] 8255
```

```
# Getting all the columns that also have only missing values
flights_missing_dep_time |>
  select(where(~all(is.na(.)))) |>
```

```r
  names()
```

```
## [1] "dep_time"  "dep_delay" "arr_time"  "arr_delay" "air_time"
```

3

Assuming that a missing dep_time implies that a flight is cancelled, look at the number of cancelled flights per day. Is there a pattern? Is there a connection between the proportion of cancelled flights and the average delay of non-cancelled flights?

Answer: There seems to be a positive correlation between the average delay and the proportion of cancelled flights

```r
# Getting the number of cancelled flights
cancelled_flights <- flights |>
  filter(is.na(dep_time)) |>
  group_by(month, day) |>
  summarise(cancelled_flights_num = n())
```

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

```r
cancelled_flights
```

```
## # A tibble: 358 x 3
## # Groups:   month [12]
##    month   day cancelled_flights_num
##    <int> <int>                 <int>
##  1     1     1                     4
##  2     1     2                     8
##  3     1     3                    10
##  4     1     4                     6
##  5     1     5                     3
##  6     1     6                     1
##  7     1     7                     3
##  8     1     8                     4
##  9     1     9                     5
## 10     1    10                     3
## # i 348 more rows
```

```r
# Getting the number of total flights
total_flights <- flights |>
  group_by(month, day) |>
  summarise(total_flights_num = n())
```

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

```r
# Getting the proportion of cancelled flights
proportion_flights <- inner_join(cancelled_flights, total_flights, by=c("month", "day")) |>
  mutate(cancelled_flights_ratio = cancelled_flights_num / total_flights_num)

proportion_flights
```

```
## # A tibble: 358 x 5
## # Groups:   month [12]
##    month   day cancelled_flights_num total_flights_num cancelled_flights_ratio
##    <int> <int>                 <int>             <int>                   <dbl>
```

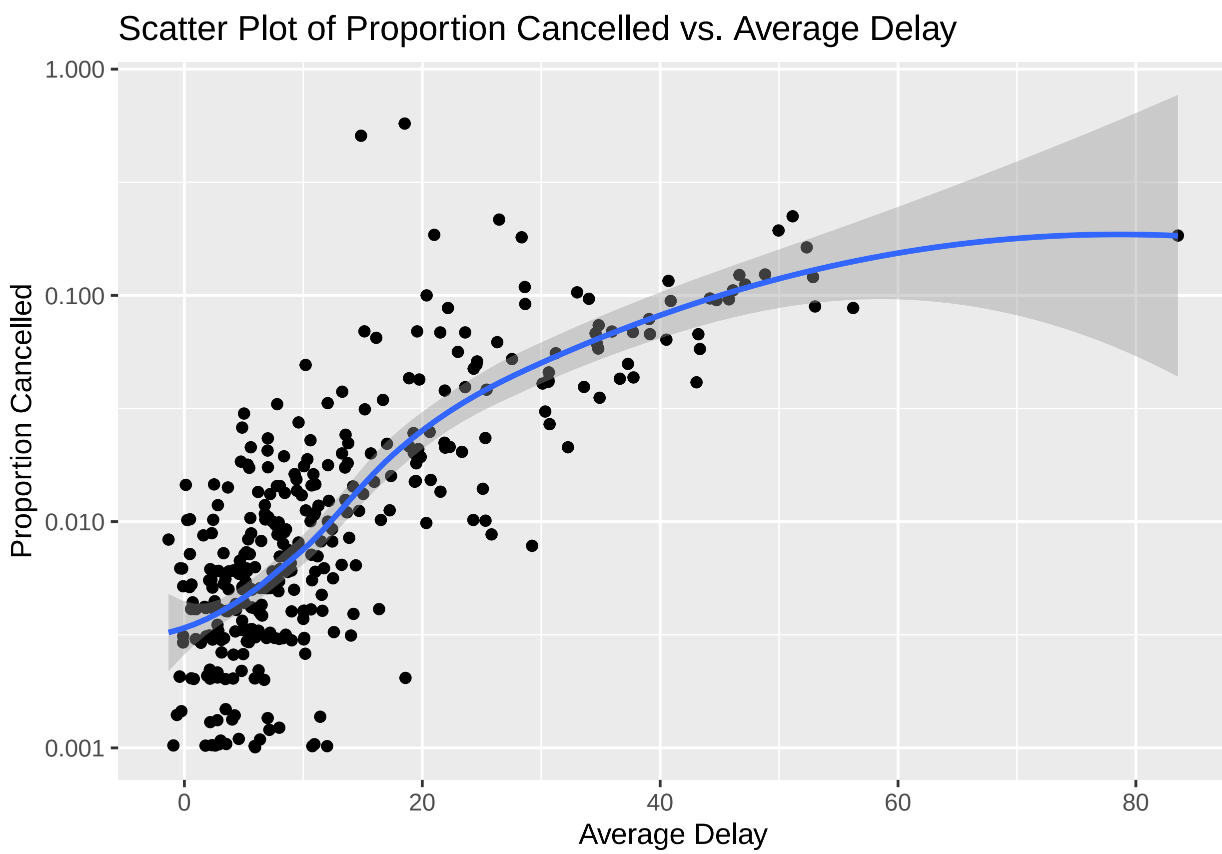
```
##  1     1     1                     4          842            0.00475
##  2     1     2                     8          943            0.00848
##  3     1     3                    10          914            0.0109
##  4     1     4                     6          915            0.00656
##  5     1     5                     3          720            0.00417
##  6     1     6                     1          832            0.00120
##  7     1     7                     3          933            0.00322
##  8     1     8                     4          899            0.00445
##  9     1     9                     5          902            0.00554
## 10     1    10                     3          932            0.00322
## # i 348 more rows
```

```r
# Calculating the average delay
proportion_flights <- flights |>
  filter(!is.na(dep_time)) |>
  group_by(month, day) |>
  summarise(average_delay = mean(dep_delay, na.rm = TRUE)) |>
  inner_join(proportion_flights, by = c("month", "day"))
```

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

```r
# Graph proportion of cancelled flights and average delay
proportion_flights |>
ggplot(aes(average_delay, cancelled_flights_ratio)) +
  geom_point() +
  geom_smooth(se = TRUE) +
  labs(title = "Scatter Plot of Proportion Cancelled vs. Average Delay",
       x = "Average Delay",
       y = "Proportion Cancelled") +
  scale_y_continuous(trans='log10')
```

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



12.4.4 Exercises

1

What will sum(is.na(x)) tell you? How about mean(is.na(x))?

Answer: `sum(is.na(x))` gets the number of items that are `Nan` in vector. While `mean(is.na(x))` gets the proportions of items that are `nan` in the vector.

```
sum(is.na(flights$dep_delay))
```

```
## [1] 8255
```

```
mean(is.na(flights$dep_delay))
```

```
## [1] 0.02451184
```

```
flights |>
  filter(is.na(dep_delay)) |>
  nrow()
```

```
## [1] 8255
```

```
nrow(filter(flights, is.na(dep_delay))) / nrow(flights)
```

```
## [1] 0.02451184
```

2

What does prod() return when applied to a logical vector? What logical summary function is it equivalent to? What does min() return when applied to a logical vector? What logical summary function is it equivalent to? Read the documentation and perform a few experiments.

Answer: When prod() is applied to a logical vector, it returns the product of the vector's elements, treating TRUE as 1 and FALSE as 0. Therefore, if any value in the logical vector is FALSE (equivalent to 0), prod() will return 0 because 1 times 0 is 0 no matter the number if 1s and 0s. If all values are TRUE (equivalent to 1), prod() will return 1. This behavior makes prod() on a logical vector equivalent to the logical all() function, which checks if all values are TRUE.

On the other hand, min() applied to a logical vector returns the minimum value of the vector, treating TRUE as 1 and FALSE as 0. Therefore, if any value in the vector is FALSE, min() will return 0 If all values are TRUE, min() will return 1 because 0 is smaller than 1. This behavior makes min() on a logical vector equivalent to the logical all() function, which checks if all values are TRUE.

```
# ?prod

prod(c(TRUE, TRUE))
```

```
## [1] 1
```

```
prod(c(TRUE, FALSE, TRUE))
```

```
## [1] 0
```

```
all(c(TRUE, FALSE, TRUE))
```

```
## [1] FALSE
```

```
# ?min
min(c(TRUE, TRUE))
```

```
## [1] 1
```

```
min(c(TRUE, FALSE, TRUE))
```

```
## [1] 0
```

12.5.4 Exercises

1

A number is even if it's divisible by two, which in R you can find out with x %% 2 == 0. Use this fact and if_else() to determine whether each number between 0 and 20 is even or odd.

```
one_to_twenty <- 1:20

even_odd <- if_else(one_to_twenty %% 2 == 0, "Even", "Odd")
tibble(one_to_twenty, even_odd)
```

```
## # A tibble: 20 x 2
##    one_to_twenty even_odd
##            <int> <chr>
##  1             1 Odd
##  2             2 Even
##  3             3 Odd
##  4             4 Even
```

```
## 5               5 Odd
## 6               6 Even
## 7               7 Odd
## 8               8 Even
## 9               9 Odd
## 10             10 Even
## 11             11 Odd
## 12             12 Even
## 13             13 Odd
## 14             14 Even
## 15             15 Odd
## 16             16 Even
## 17             17 Odd
## 18             18 Even
## 19             19 Odd
## 20             20 Even
```

4

Write a case_when() statement that uses the month and day columns from flights to label a selection of important US holidays (e.g., New Years Day, 4th of July, Thanksgiving, and Christmas). First create a logical column that is either TRUE or FALSE, and then create a character column that either gives the name of the holiday or is NA.

```r
# Define a function to check if a date is a specific holiday
is_holiday <- function(month, day) {
  case_when(
    month == 1 & day == 1 ~ TRUE,           # New Year's Day
    month == 7 & day == 4 ~ TRUE,           # 4th of July
    month == 11 & day == 28 ~ TRUE,         # Thanksgiving
    month == 12 & day == 25 ~ TRUE,         # Christmas
    TRUE ~ FALSE                           # Other days
  )
}

# Apply the function to create a logical column
flights |>
  mutate(is_holiday = is_holiday(month, day),
         holiday_name = case_when(
           is_holiday & month == 1 & day == 1 ~ "New Year's Day",
           is_holiday & month == 7 & day == 4 ~ "4th of July",
           is_holiday & month == 11 & day == 28 ~ "Thanksgiving",
           is_holiday & month == 12 & day == 25 ~ "Christmas",
           TRUE ~ NA_character_
         )) |>
  relocate(is_holiday) |>
  relocate(holiday_name)
```

```
## # A tibble: 336,776 x 21
##    holiday_name   is_holiday  year month   day dep_time sched_dep_time dep_delay
##    <chr>          <lgl>      <int> <int> <int>    <int>          <int>     <dbl>
##  1 New Year's Day TRUE        2013     1     1      517            515         2
##  2 New Year's Day TRUE        2013     1     1      533            529         4
##  3 New Year's Day TRUE        2013     1     1      542            540         2
##  4 New Year's Day TRUE        2013     1     1      544            545        -1
```

```
##  5 New Year's Day TRUE        2013     1     1      554            600         -6
##  6 New Year's Day TRUE        2013     1     1      554            558         -4
##  7 New Year's Day TRUE        2013     1     1      555            600         -5
##  8 New Year's Day TRUE        2013     1     1      557            600         -3
##  9 New Year's Day TRUE        2013     1     1      557            600         -3
## 10 New Year's Day TRUE        2013     1     1      558            600         -2
## # i 336,766 more rows
## # i 13 more variables: arr_time <int>, sched_arr_time <int>, 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>
```

13.3.1 Exercises

1

How can you use count() to count the number rows with a missing value for a given variable?

Answer: Filter for rows that have a missing value with `filter()` and then use count() to get the number of filtered rows

```
count(filter(flights, is.na(dep_delay)))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  8255
```

2

Expand the following calls to count() to instead use group_by(), summarize(), and arrange():

flights |> count(dest, sort = TRUE)

flights |> count(tailnum, wt = distance)

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

```
## # A tibble: 105 x 2
##    dest      n
##    <chr> <int>
##  1 ORD   17283
##  2 ATL   17215
##  3 LAX   16174
##  4 BOS   15508
##  5 MCO   14082
##  6 CLT   14064
##  7 SFO   13331
##  8 FLL   12055
##  9 MIA   11728
## 10 DCA    9705
## # i 95 more rows
```

```
flights |>
  group_by(dest) |>
  summarize(total_flight = n()) |>
  arrange(desc(total_flight))
```

```
## # A tibble: 105 x 2
##    dest  total_flight
##    <chr>        <int>
##  1 ORD          17283
##  2 ATL          17215
##  3 LAX          16174
##  4 BOS          15508
##  5 MCO          14082
##  6 CLT          14064
##  7 SFO          13331
##  8 FLL          12055
##  9 MIA          11728
## 10 DCA           9705
## # i 95 more rows
```

```
flights |> count(tailnum, wt = distance)
```

```
## # A tibble: 4,044 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 N0EGMQ  250866
##  3 N10156  115966
##  4 N102UW   25722
##  5 N103US   24619
##  6 N104UW   25157
##  7 N10575  150194
##  8 N105UW   23618
##  9 N107US   21677
## 10 N108UW   32070
## # i 4,034 more rows
```

```
flights |>
  group_by(tailnum) |>
  summarize(weighted_tailnum = sum(distance)) |>
  arrange(tailnum)
```

```
## # A tibble: 4,044 x 2
##    tailnum weighted_tailnum
##    <chr>              <dbl>
##  1 D942DN              3418
##  2 N0EGMQ            250866
##  3 N10156            115966
##  4 N102UW             25722
##  5 N103US             24619
##  6 N104UW             25157
##  7 N10575            150194
##  8 N105UW             23618
##  9 N107US             21677
## 10 N108UW             32070
## # i 4,034 more rows
```

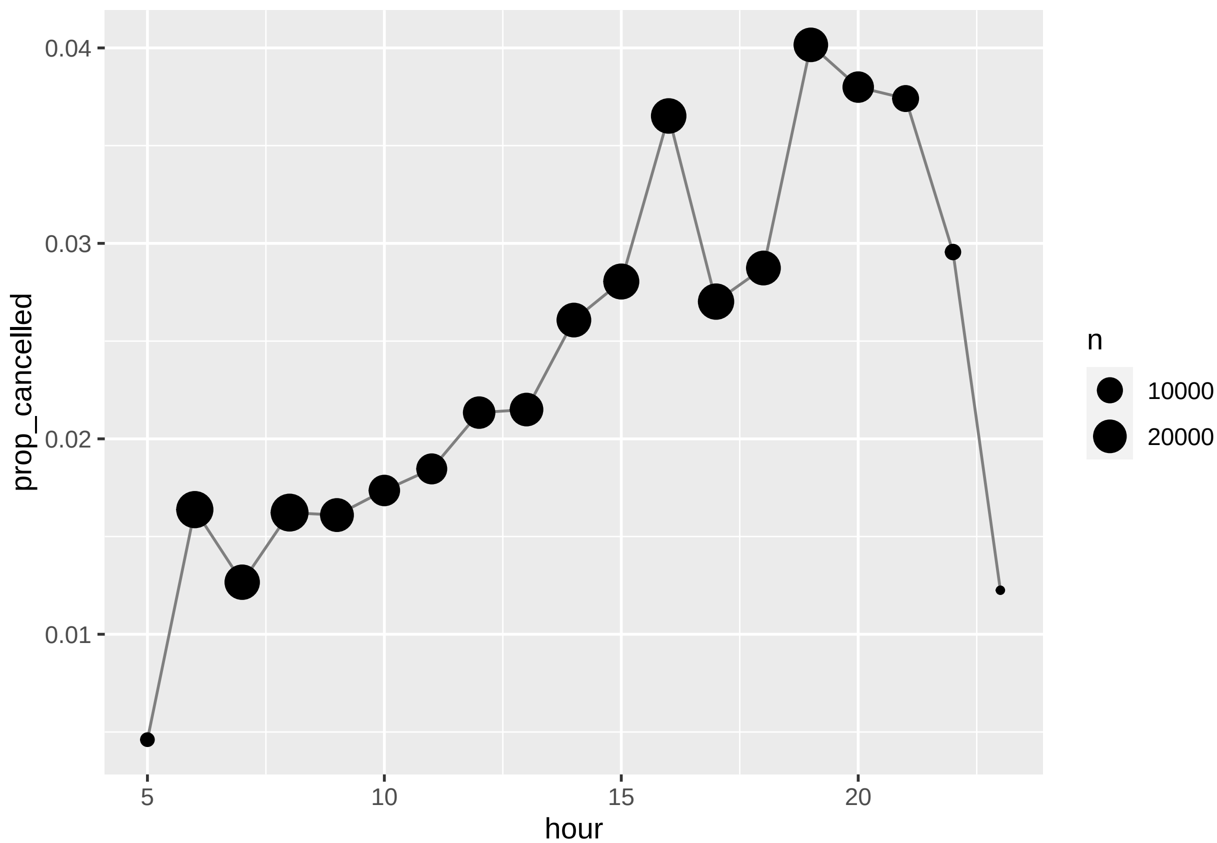
13.4.8 Exercises

1

Explain in words what each line of the code used to generate Figure 13.1 does.

13.5.4 Exercises

```
# Pipes in data
flights |>
# Groups the data by the hour of the scheduled departure time.
# The sched_dep_time %/% 100 extracts the hour by integer division (ignoring the minutes).
# The result is a grouped dataframe with an additional column named hour.
  group_by(hour = sched_dep_time %/% 100) |>
# Calculates summary statistics within each group.
# It calculates the mean of the logical vector
# is.na(dep_time) (proportion of cancelled flights) and
# counts the total number of flights (n) within each hour.
  summarize(prop_cancelled = mean(is.na(dep_time)), n = n()) |>
# Filters out rows where the hour is less than or equal to 1.
# This is done to exclude any potential outliers or data issues.
  filter(hour > 1) |>
# Initializes a ggplot object, setting the aesthetics (aes)
# for the x-axis as hour and the y-axis as prop_cancelled.
  ggplot(aes(x = hour, y = prop_cancelled)) +
# Adds a line layer to the ggplot.
# This line connects the proportion of cancelled flights across different hours.
# The color is set to "grey50".
  geom_line(color = "grey50") +
# Adds a point layer to the ggplot. Points represent the individual hours,
# and the size of each point is determined by the number of flights (n) in that hour.
  geom_point(aes(size = n))
```



1

Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for min_rank().

Answer: I want to handle ties like done in min_rank() where we rank all the ties by having them the smallest rank between all numbers that were in tie as if they all had different values that wouldn't change the order of the numbers.

```
flights |>
  mutate(delay_rank = min_rank(desc(dep_delay))) |>
  arrange(delay_rank) |>
  filter(delay_rank <= 10)
```

```
## # A tibble: 10 x 20
##     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 12 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>, delay_rank <int>
```

#2
Which plane (tailnum) has the worst on-time record?

Answer: N240AT, N392SW, N635SW has all delayed flights and the most number of delayed flights

```
# Getting the counts of delayed flights per tailnum
arr_delay_flights <- flights |>
  filter(arr_delay >= 1) |>
  group_by(tailnum) |>
  summarize(arr_delay_num = n())

# Getting the counts of all flights for each tailnum
total_flights <- flights |>
  group_by(tailnum) |>
  summarize(total_flights_num = n())

# Combining the tibbles by tailnum
joined_delay_flights <- inner_join(arr_delay_flights, total_flights, by="tailnum")

# Getting the proportion of delayed flights
joined_delay_flights <- joined_delay_flights |>
  mutate(delay_ratio = arr_delay_num / total_flights_num)

# Getting the tailnum with the worst delay ratio
# Then the most number of delays if there are ties
joined_delay_flights |>
  filter(delay_ratio == max(delay_ratio)) |>
  filter(arr_delay_num == max(arr_delay_num))
```

```
## # A tibble: 3 x 4
##   tailnum arr_delay_num total_flights_num delay_ratio
##   <chr>           <int>             <int>       <dbl>
## 1 N240AT              5                 5           1
## 2 N392SW              5                 5           1
## 3 N635SW              5                 5           1
```

13.6.7 Exercises

1

Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights.

Answer: Mean and Median of departure delays, distribution of delay times (via histograms, density plots or box plots), percentage of flights with delays, variability metrics (Standard dev, IQR, etc), time trends (do delays vary across different times of day, day of week/month, etc)

When is mean() useful?

Answer: `mean()` is useful when you want to understand the central tendency of the delays. Mean is most appropriate if you are analyzing data that is symmetrically distributed and

does not contain outliers that can skew the average. The mean is also useful when comparing different distributions.

When is median() useful?

Answer: Median is useful when you want to understand the central tendency of the delays and the distribution contains outliers that can skew the average. Median is robust because it is based on ranking of the numbers.

When might you want to use something else?

Answer: Using standard deviation allows you to better understand the overall spread of your data.

Should you use arrival delay or departure delay?

Answer: Using arrival or departure delay depends on what insights you are focusing on. Such as if you are seeing how long the flight takes off from the scheduled time or how different the arrival time is from the scheduled time.

Why might you want to use data from planes?

Answer: Looking at planes will give you insights into specific aircraft or aircraft types. This may help us better understand why some flights have frequent or long delays.

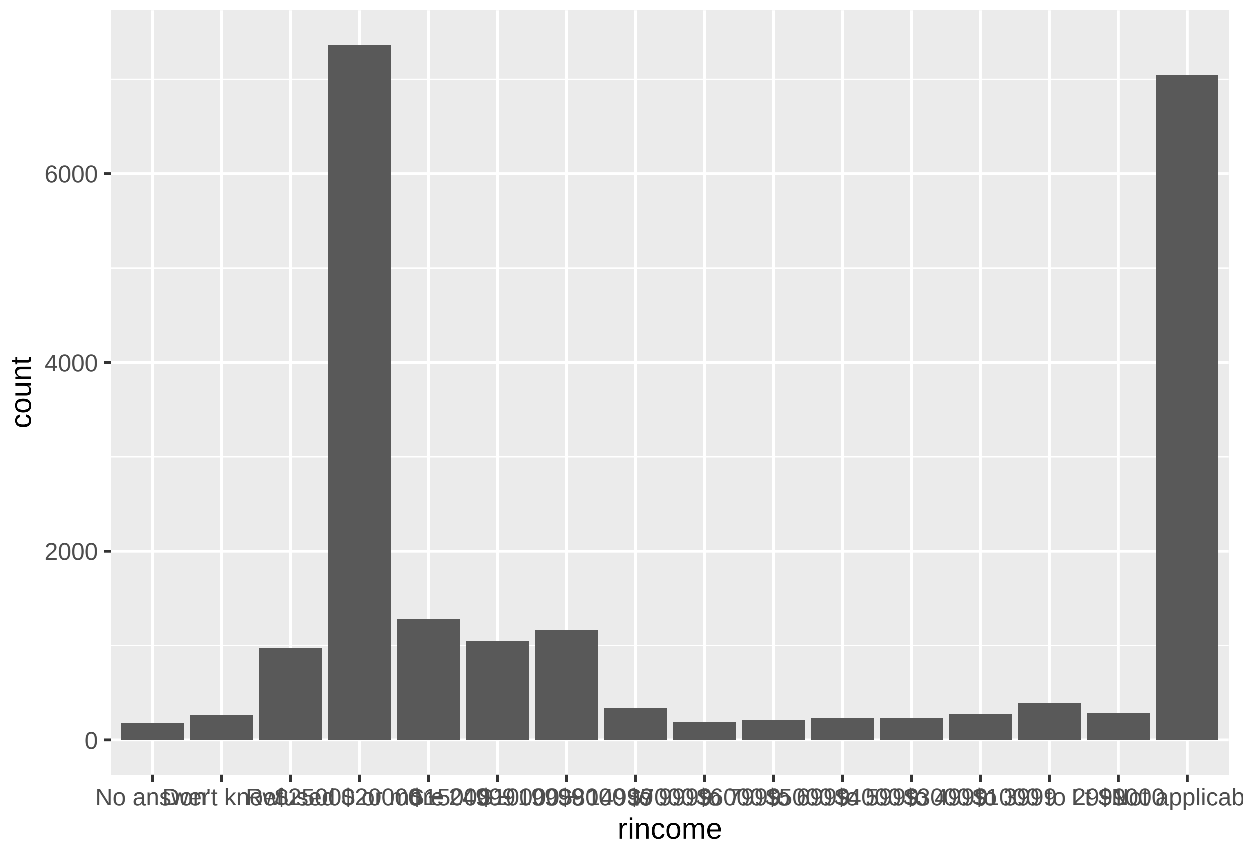
16.3.1 Exercise

1

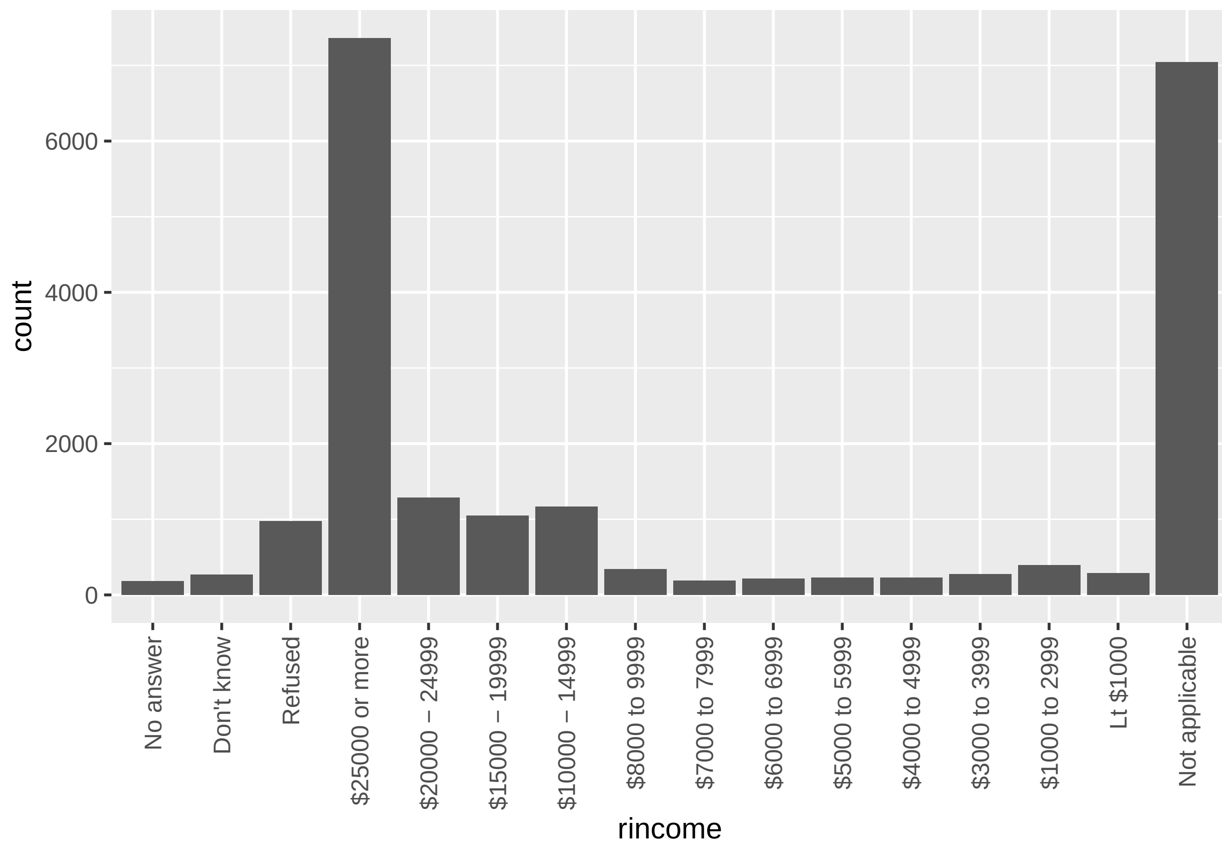
Explore the distribution of rincome (reported income). What makes the default bar chart hard to understand? How could you improve the plot?

Answer: The default bar chart is difficult to understand because the x-labels are long and overlap each other. This makes reading the label for any bar difficult. You can improve the plot by rotating the x-labels

```
gss_cat |>
  ggplot(aes(x=rincome)) +
  geom_bar()
```




```r
gss_cat |>
  ggplot(aes(x=rincome)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



2

What is the most common relig in this survey? What's the most common partyid?

Answer: Protestant is the most common `relig` and independent is the most common `partyid`

```
gss_cat |>
  group_by(relig) |>
  summarise(relig_num = n()) |>
  filter(relig_num == max(relig_num))
```

```
## # A tibble: 1 x 2
##   relig      relig_num
##   <fct>          <int>
## 1 Protestant     10846
```

```
gss_cat |>
  group_by(partyid) |>
  summarise(partyid_num = n()) |>
  filter(partyid_num == max(partyid_num))
```

```
## # A tibble: 1 x 2
##   partyid     partyid_num
##   <fct>             <int>
## 1 Independent        4119
```

3

Which relig does denom (denomination) apply to? How can you find out with a table? How can you find out with a visualization?

Answer: Denomination mainly applies to Protestant.

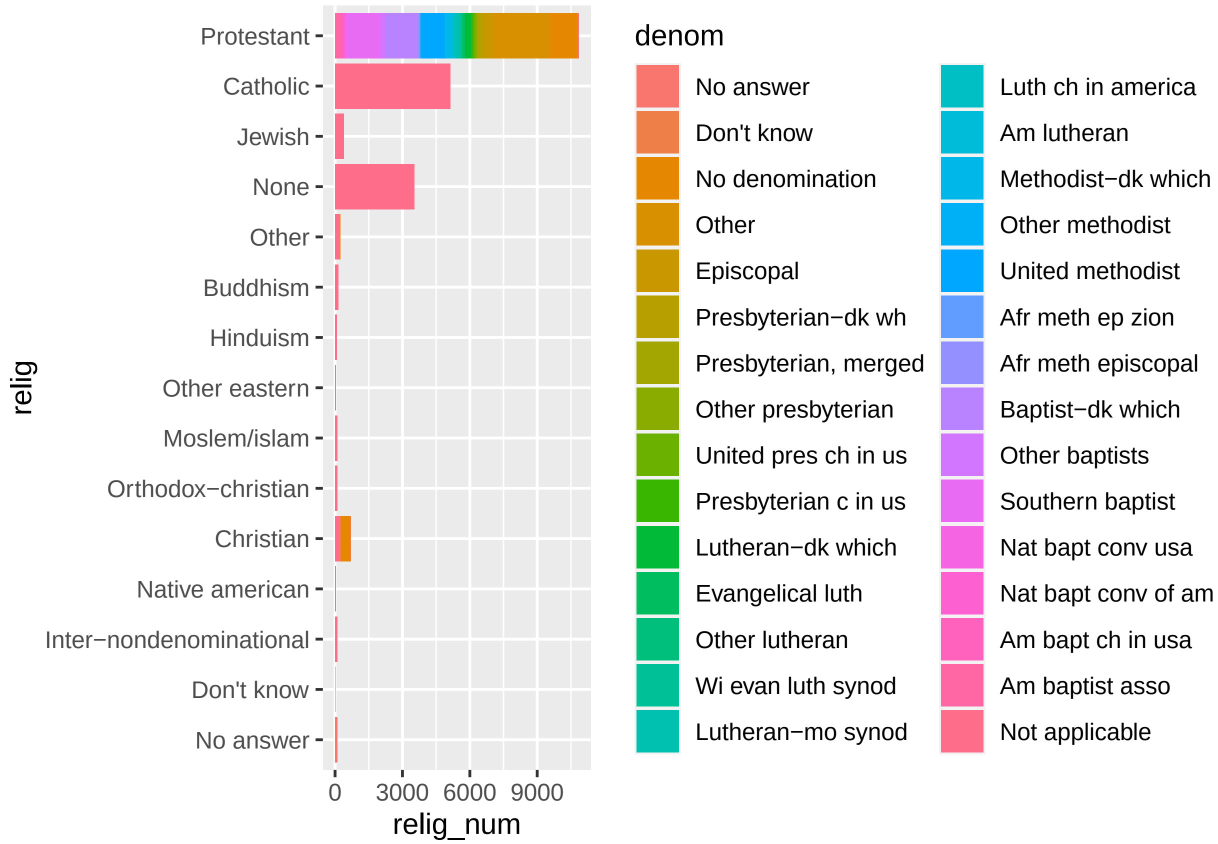
```
grouped_relig_denom <- gss_cat |>
  group_by(relig, denom) |>
  summarise(relig_num = n())
```

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

```
grouped_relig_denom
```

```
## # A tibble: 47 x 3
## # Groups:   relig [15]
##    relig                   denom            relig_num
##    <fct>                   <fct>                <int>
##  1 No answer               No answer               93
##  2 Don't know              Not applicable          15
##  3 Inter-nondenominational Not applicable         109
##  4 Native american         Not applicable          23
##  5 Christian               No answer                2
##  6 Christian               Don't know              11
##  7 Christian               No denomination        452
##  8 Christian               Not applicable         224
##  9 Orthodox-christian      Not applicable          95
## 10 Moslem/islam            Not applicable         104
## # i 37 more rows
```

```
grouped_relig_denom |>
  ggplot(aes(y = relig, fill = denom, x = relig_num)) +
  geom_col()
```



16.4.1 Exercises

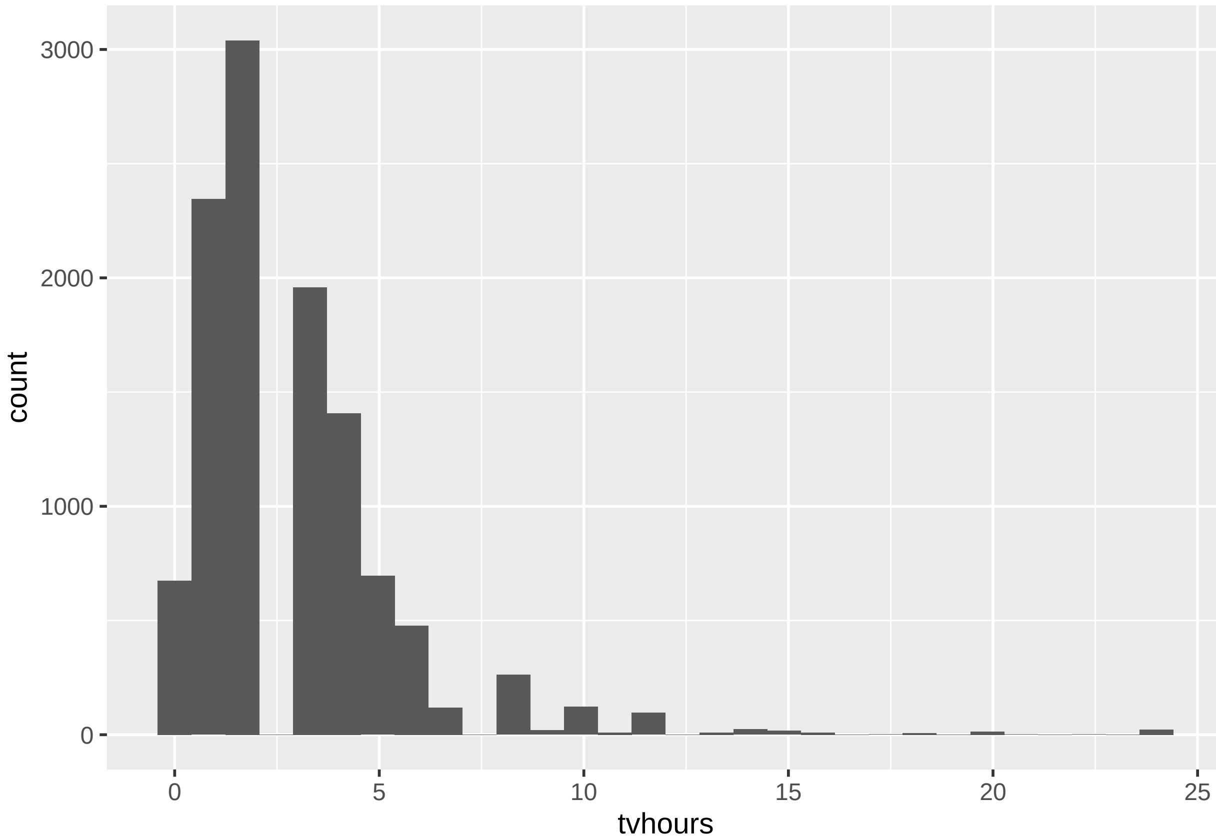
1

There are some suspiciously high numbers in tvhours. Is the mean a good summary?

Answer: Yes, there is a datapoint with 24 hours which is unlikely for the number of hours watched per day. Mean might not be a good summary if there's a potential outlier such as this

```
gss_cat |>
  ggplot(aes(tvhours)) +
  geom_histogram(na.rm=TRUE)
```

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




```
max(gss_cat$tvhours, na.rm=TRUE)
```

```
## [1] 24
```

```
mean(gss_cat$tvhours, na.rm=TRUE)
```

```
## [1] 2.980771
```

```
median(gss_cat$tvhours, na.rm=TRUE)
```

```
## [1] 2
```

2

For each factor in gss_cat identify whether the order of the levels is arbitrary or principled.

Answer: Arbitrary: Marital(depending on whether marriage status can be ordered), race, partid, relig, denom. Principled: rincome

```
str(gss_cat)
```

```
## tibble [21,483 x 9] (S3: tbl_df/tbl/data.frame)
##  $ year   : int [1:21483] 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
##  $ marital: Factor w/ 6 levels "No answer","Never married",..: 2 4 5 2 4 6 2 4 6 6 ...
##  $ age    : int [1:21483] 26 48 67 39 25 25 36 44 44 47 ...
##  $ race   : Factor w/ 4 levels "Other","Black",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ rincome: Factor w/ 16 levels "No answer","Don't know",..: 8 8 16 16 16 5 4 9 4 4 ...
##  $ partyid: Factor w/ 10 levels "No answer","Don't know",..: 6 5 7 6 9 10 5 8 9 4 ...
##  $ relig  : Factor w/ 16 levels "No answer","Don't know",..: 15 15 15 6 12 15 5 15 15 15 ...
```

```
##  $ denom  : Factor w/ 30 levels "No answer","Don't know",..: 25 23 3 30 30 25 30 15 4 25 ...
##  $ tvhours: int [1:21483] 12 NA 2 4 1 NA 3 NA 0 3 ...
```

```r
factored_gss_cat <- gss_cat |>
  mutate(marital_num = as.integer(marital)) |>
  mutate(race_num = as.integer(race)) |>
  mutate(rincome_num = as.integer(rincome)) |>
  mutate(partyid_num = as.integer(partyid)) |>
  mutate(relig_num = as.integer(relig)) |>
  mutate(denom_num = as.integer(denom))

factored_gss_cat |>
  group_by(marital_num, marital) |>
  summarize(marital_count = n())
```

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

## # A tibble: 6 x 3
## # Groups:   marital_num [6]
##   marital_num marital       marital_count
##         <int> <fct>                 <int>
## 1           1 No answer                17
## 2           2 Never married          5416
## 3           3 Separated               743
## 4           4 Divorced               3383
## 5           5 Widowed                1807
## 6           6 Married               10117
```

```r
factored_gss_cat |>
  group_by(race_num, race) |>
  summarize(race_count = n())
```

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

## # A tibble: 3 x 3
## # Groups:   race_num [3]
##   race_num race  race_count
##      <int> <fct>      <int>
## 1        1 Other       1959
## 2        2 Black       3129
## 3        3 White      16395
```

```r
factored_gss_cat |>
  group_by(rincome_num, rincome) |>
  summarize(rincome_count = n())
```

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

## # A tibble: 16 x 3
## # Groups:   rincome_num [16]
##    rincome_num rincome        rincome_count
##          <int> <fct>                  <int>
##  1           1 No answer                183
##  2           2 Don't know               267
##  3           3 Refused                  975
```

```
##  4           4 $25000 or more          7363
##  5           5 $20000 - 24999          1283
##  6           6 $15000 - 19999          1048
##  7           7 $10000 - 14999          1168
##  8           8 $8000 to 9999            340
##  9           9 $7000 to 7999            188
## 10          10 $6000 to 6999            215
## 11          11 $5000 to 5999            227
## 12          12 $4000 to 4999            226
## 13          13 $3000 to 3999            276
## 14          14 $1000 to 2999            395
## 15          15 Lt $1000                 286
## 16          16 Not applicable          7043
```

```
factored_gss_cat |>
  group_by(partyid_num, partyid) |>
  summarize(partyid_count = n())
```

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

```
## # A tibble: 10 x 3
## # Groups:   partyid_num [10]
##    partyid_num partyid            partyid_count
##          <int> <fct>                      <int>
##  1           1 No answer                    154
##  2           2 Don't know                     1
##  3           3 Other party                  393
##  4           4 Strong republican           2314
##  5           5 Not str republican          3032
##  6           6 Ind,near rep                1791
##  7           7 Independent                 4119
##  8           8 Ind,near dem                2499
##  9           9 Not str democrat            3690
## 10          10 Strong democrat             3490
```

```
factored_gss_cat |>
  group_by(relig_num, relig) |>
  summarize(relig_count = n())
```

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

```
## # A tibble: 15 x 3
## # Groups:   relig_num [15]
##    relig_num relig                   relig_count
##        <int> <fct>                         <int>
##  1         1 No answer                        93
##  2         2 Don't know                       15
##  3         3 Inter-nondenominational         109
##  4         4 Native american                  23
##  5         5 Christian                       689
##  6         6 Orthodox-christian               95
##  7         7 Moslem/islam                    104
##  8         8 Other eastern                    32
##  9         9 Hinduism                         71
```

```
## 10          10 Buddhism                         147
## 11          11 Other                            224
## 12          12 None                            3523
## 13          13 Jewish                           388
## 14          14 Catholic                        5124
## 15          15 Protestant                     10846
```

```
factored_gss_cat |>
  group_by(denom_num, denom) |>
  summarize(denom_count = n())
```

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

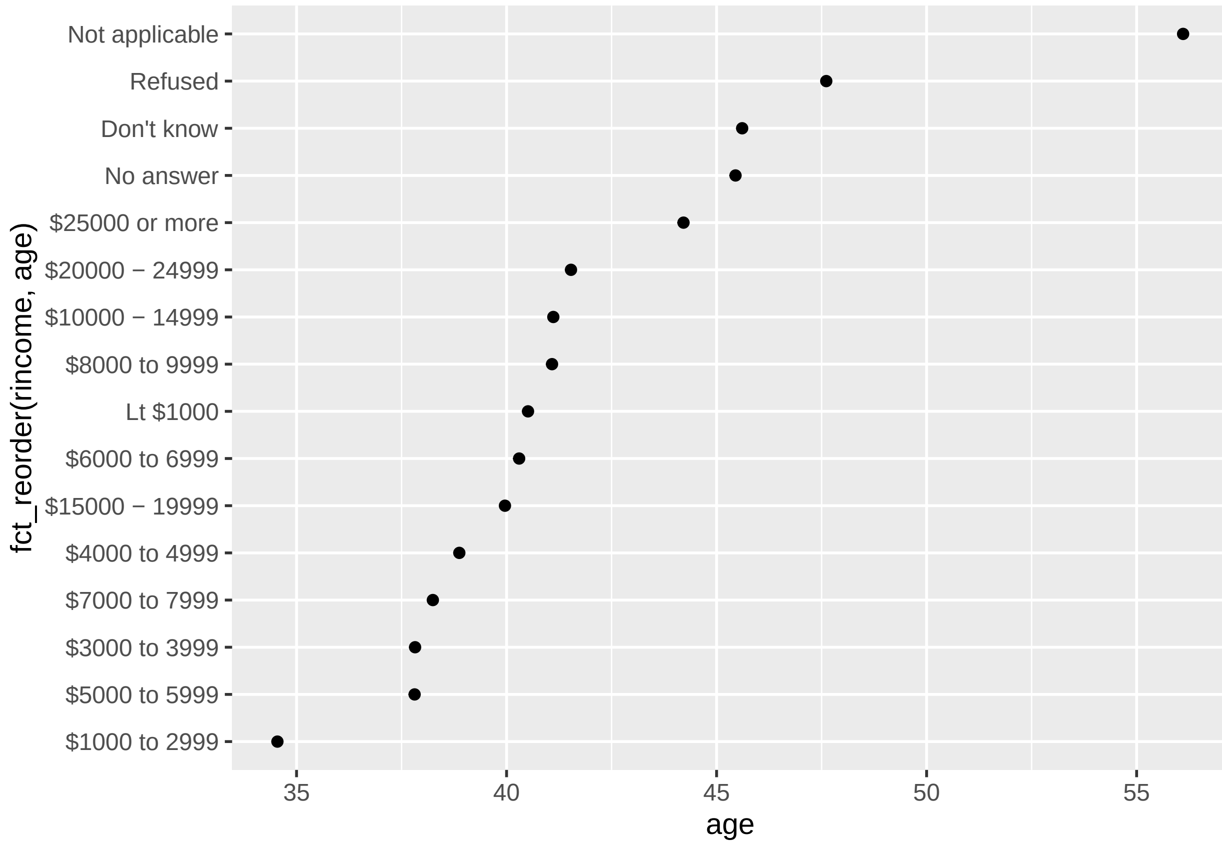
```
## # A tibble: 30 x 3
## # Groups:   denom_num [30]
##    denom_num denom                   denom_count
##        <int> <fct>                         <int>
##  1         1 No answer                       117
##  2         2 Don't know                       52
##  3         3 No denomination                1683
##  4         4 Other                          2534
##  5         5 Episcopal                       397
##  6         6 Presbyterian-dk wh              244
##  7         7 Presbyterian, merged             67
##  8         8 Other presbyterian               47
##  9         9 United pres ch in us            110
## 10        10 Presbyterian c in us            104
## # i 20 more rows
```

3

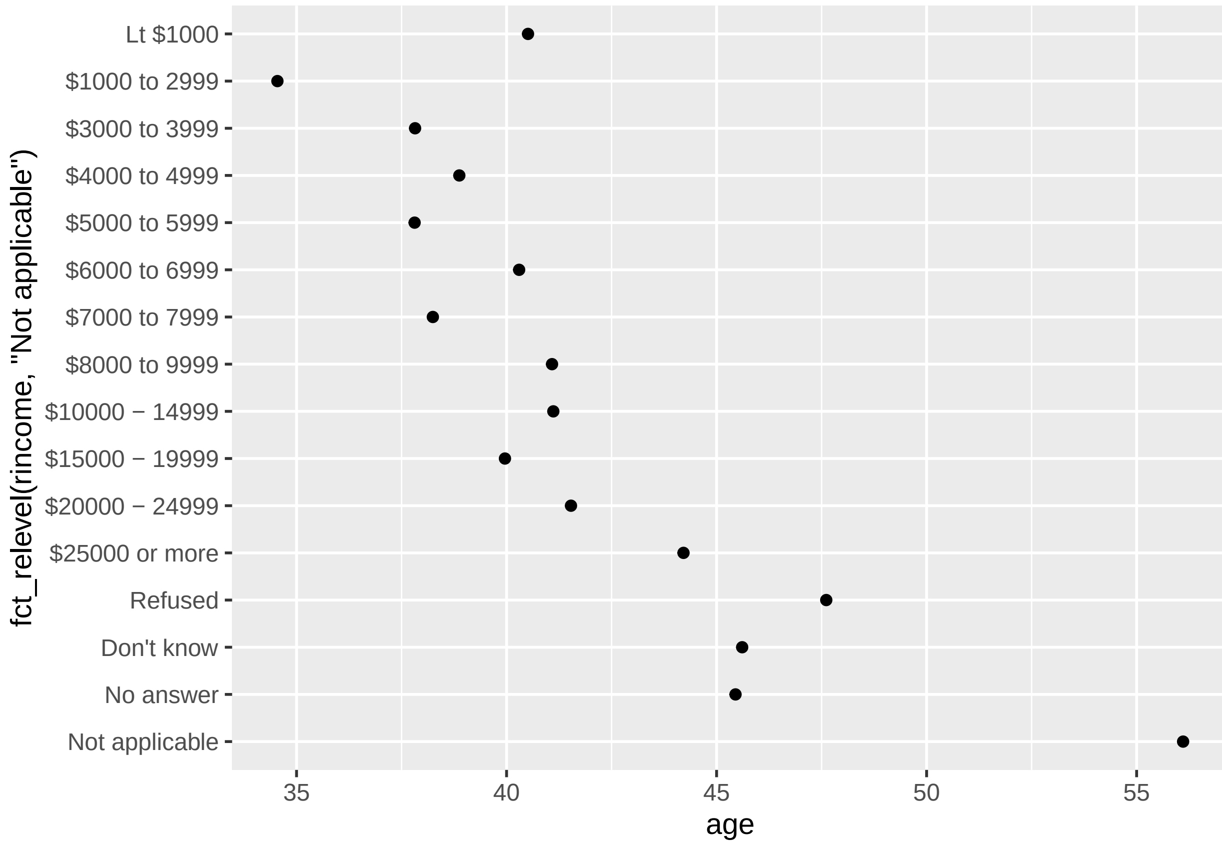
Why did moving "Not applicable" to the front of the levels move it to the bottom of the plot?

Answer: Moving "Not applicable" to the front of the levels moved it to the bottom because the plot is plotted from bottom to top going from first in the factor to the last factor.

```
rincome_summary <- gss_cat |>
  group_by(rincome) |>
  summarize(
    age = mean(age, na.rm = TRUE),
    n = n()
  )

# ?fct_reorder

ggplot(rincome_summary, aes(x = age, y = fct_reorder(rincome, age))) +
  geom_point()
```




```r
# ?fct_relevel

ggplot(rincome_summary, aes(x = age, y = fct_relevel(rincome, "Not applicable"))) +
  geom_point()
```



19.3.4 Exercises

1

Find the 48 hours (over the course of the whole year) that have the worst delays. Cross-reference it with the weather data. Can you see any patterns?

Answer: Given time constraints we were only able to see that there might be a difference in wind speed

```
# Grouping each flight by month/day

worst_flights <- flights |>
  filter(dep_delay < 0) |>
  group_by(month, day) |>
  summarise(n_flights = n(), .groups = "drop")

# Grouping by every 48 hours

worst_flights <- worst_flights |>
  mutate(row = row_number()) |>
  mutate(index_48_hours_second = row_number() %/% 2) |>
  mutate(index_48_hours_first = ceiling(row_number() / 2) + 1000)

# Adding the index to worst flights

worst_flights <- inner_join(flights, worst_flights, by=c("month", "day")) |>
```

```r
  filter(dep_delay < 0)

# Getting the average delay for each 48 hours

two_days_worst_flights_first <- worst_flights |>
  group_by(index_48_hours_first) |>
  summarise(avg_48_delay = mean(dep_delay))

two_days_worst_flights_second <- worst_flights |>
  group_by(index_48_hours_second) |>
  summarise(avg_48_delay = mean(dep_delay))

# Seeing which 48 hours have the worst delays on average

two_days_worst_flights <- bind_rows(two_days_worst_flights_first, two_days_worst_flights_second) |>
  filter(avg_48_delay == min(avg_48_delay))

two_days_worst_index <- two_days_worst_flights$index_48_hours_second[1]

one_worst_flight <- worst_flights |>
  filter(index_48_hours_second == two_days_worst_index)

worst_day <- one_worst_flight$day[1]
worst_month <- one_worst_flight$month[1]

# Trying to see any patterns with weather

weather
```

```
## # A tibble: 26,115 x 15
##    origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
##  1 EWR     2013     1     1     1  39.0  26.1  59.4      270      10.4
##  2 EWR     2013     1     1     2  39.0  27.0  61.6      250       8.06
##  3 EWR     2013     1     1     3  39.0  28.0  64.4      240      11.5
##  4 EWR     2013     1     1     4  39.9  28.0  62.2      250      12.7
##  5 EWR     2013     1     1     5  39.0  28.0  64.4      260      12.7
##  6 EWR     2013     1     1     6  37.9  28.0  67.2      240      11.5
##  7 EWR     2013     1     1     7  39.0  28.0  64.4      240      15.0
##  8 EWR     2013     1     1     8  39.9  28.0  62.2      250      10.4
##  9 EWR     2013     1     1     9  39.9  28.0  62.2      260      15.0
## 10 EWR     2013     1     1    10  41    28.0  59.6      260      13.8
## # i 26,105 more rows
## # i 5 more variables: wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dttm>
```

```r
weather |>
  filter(month == worst_month) |>
  filter(day == worst_day)
```

```
## # A tibble: 72 x 15
##    origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
##  1 EWR     2013     9     5     0  71.1  57.0  61.2      250       9.21
##  2 EWR     2013     9     5     1  68    57.9  70.2      230       9.21
```

```
##   3 EWR     2013     9     5     2  66.9  57.9  72.8       230       9.21
##   4 EWR     2013     9     5     3  66.0  57.9  75.1       220       5.75
##   5 EWR     2013     9     5     4  63.0  57.9  83.6       210       6.90
##   6 EWR     2013     9     5     5  64.0  57.9  80.5       220       8.06
##   7 EWR     2013     9     5     6  63.0  57.9  83.6       210       5.75
##   8 EWR     2013     9     5     7  68    57.9  70.2       250       9.21
##   9 EWR     2013     9     5     8  70.0  57.9  65.5       260      10.4
##  10 EWR     2013     9     5     9  73.9  59    59.6       260       8.06
## # i 62 more rows
## # i 5 more variables: wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dttm>
```

```r
weather |>
  filter(month == worst_month) |>
  filter(day == worst_day) |>
  summarise(mean(wind_speed, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   `mean(wind_speed, na.rm = TRUE)`
##                              <dbl>
## 1                             11.0
```

```r
weather |>
  summarise(mean(wind_speed, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   `mean(wind_speed, na.rm = TRUE)`
##                              <dbl>
## 1                             10.5
```

2

Imagine you've found the top 10 most popular destinations using this code:

```r
flights2 <- flights |>
  select(year, time_hour, origin, dest, tailnum, carrier)

top_dest <- flights2 |>
  count(dest, sort = TRUE) |>
  head(10)
```

How can you find all flights to those destinations?

```r
left_join(top_dest, flights, by="dest")
```

```
## # A tibble: 141,145 x 20
##    dest      n  year month   day dep_time sched_dep_time dep_delay arr_time
##    <chr> <int> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1 ORD   17283  2013     1     1      554            558        -4      740
##  2 ORD   17283  2013     1     1      558            600        -2      753
##  3 ORD   17283  2013     1     1      608            600         8      807
##  4 ORD   17283  2013     1     1      629            630        -1      824
##  5 ORD   17283  2013     1     1      656            700        -4      854
##  6 ORD   17283  2013     1     1      709            700         9      852
##  7 ORD   17283  2013     1     1      715            713         2      911
##  8 ORD   17283  2013     1     1      739            745        -6      918
##  9 ORD   17283  2013     1     1      749            710        39      939
```

```
## 10 ORD   17283  2013     1     1      828            830        -2     1027
## # i 141,135 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>
```

4

What do the tail numbers that don't have a matching record in planes have in common? (Hint: one variable explains ~90% of the problems.)

Answer: We were not able to uncover what was common betweeen tail numbers without a matching plane.

```r
unmatched_flights <- flights |>
  left_join(planes, by = c("tailnum", "year"))|>
  filter(is.na(type))

unmatched_flights
```

```
## # A tibble: 332,146 x 26
##     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      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 332,136 more rows
## # i 18 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>, type <chr>, manufacturer <chr>,
## #   model <chr>, engines <int>, seats <int>, speed <int>, engine <chr>
```

```r
# Print or visualize the results
unmatched_flights |>
  select(tailnum) |>
  unique()
```

```
## # A tibble: 3,952 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 3,942 more rows
```