Homework 2 hints

Joshua Loftus

Question 1

Preliminaries

- There is a difference between *installing* a package and *loading* it. The `install.packages("nycflights13")` command downloads the package to your computer, and you only need to do this once. But every time you open `R`, you will need to *load* the package with the `library(nycflights13)` command. Otherwise you may get errors like **object 'flights' not found** or **could not find function "ggplot"**. You should also load the `ggplot2` and `dplyr` packages with `library(tidyverse)` (they are both contained in the `tidyverse` bundle).
- Once you load the `nycflights13` library, you now have access to the `flights` data frame. Reading the documentation by running `?flights` will help you understand what variables are in the data frame, as will looking at the first few rows with `head(flights)`.
- Suppose you want to access a variable called `var` in a data frame called `df`, one way of doing this is with the dollar sign: `df$var`. For an example of using `table()`, look about halfway through the lecture notes on summaries where we computed a table of the `age` variable in a data frame called `data`: https://joftius.github.io/notes103/3summaries.html

Commands spread over multiple lines

- The `R` language has this feature which is useful but can be confusing.
- For example, if you type `1 +` and run the command, `R` waits for more input, assuming you haven't finished typing yet.
- If you are typing in the Console and get stuck with this and want to start over, just hit Escape.
- This allows you to enter very long commands by spreading them over multiple lines, making them easier to read. See these examples: (the input code is in a gray box, and `R`'s output appears below each box)

```
5 +
  1
```

```
## [1] 6
```

```
x <- c(1, 2, 3, 4,
       5, 6, 7, 8,
       9, 10)

sum(
  x)
```

```
## [1] 55
```

```
x %>% sum()
```

```
## [1] 55
```

- This is the feature that the ModernDive textbook authors used when they did this:

```
all_alaska_flights <- flights %>%
  filter(carrier == "AS")
```

- If you want to modify this for Hawaiian Airlines, it might be easier to start by putting the whole command on one line:

```r
all_alaska_flights <- flights %>% filter(carrier == "AS")
```

And then change that line to give you the answer you want.

Creating a histogram

- Note that the example plot in ModernDive 3.3 is a scatterplot, not a histogram. Copying the `ggplot()` code they use will not work for us. Scatterplots are about showing the *relationship* between *two* variables. Histograms are about showing the *distribution* of *one* variable.
- For a histogram example, see about halfway through the lecture notes here: https://joftius.github.io/notes103/4plots.html look for the use of `geom_histogram()`
- For part (c), look at the plot and think about what kinds of facts about any of the summary statistics (or relationships between different summary statistics) you can guess from the plot.
- For parts (d) and (e), it may help to look near the end of the lecture notes on summaries https://joftius.github.io/notes103/3summaries.html
- To understand the code from the lecture notes, look at these examples:

```r
x <- c(1, 2, 3, 4)
xbar <- mean(x)
x - xbar
```

```
## [1] -1.5 -0.5  0.5  1.5
```

When you add or subtract one number to a list, `R` interprets that as adding or subtracting the number to each element in the list, and outputs a new list. Subtracting the mean like this is also called "centering."

```r
abs(x - xbar)
```

```
## [1] 1.5 0.5 0.5 1.5
```

This outputs a list of the absolute values.

```r
s <- sd(x)
abs(x - xbar) <= s
```

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

This creates a list of `TRUE` and `FALSE` values, testing the conditiong `<= s` for each element in the list

```r
mean(abs(x - xbar) <= s)
```

```
## [1] 0.5
```

This tells us what proportion of the values are `TRUE`

Understanding piping

- Recall this line from the ModernDive book example:

```
all_alaska_flights <- flights %>% filter(carrier == "AS")
```

- Let's make sure we understand this line. First, ignore the beginning part and focus on `flights %>% filter(carrier == "AS")`

- Many commands in `R` have the format: `function(data, ...)` where `function` is the name of the command, `data` is the name of the data frame you are applying the command to, and `...` is other code that depends on the specific example.
- If you want to run several commands, using the output from the previous one as input to the next, you can do that like this: `function3(function2(function1(data, ...), ...), ...)`
- That can be hard to read and understand, so the `dplyr` package has a useful trick called "piping" which works like this: `data %>% function1(...) %>% function2(...) %>% function3()`
- This makes a line of code read more like English and less like mathematics.
- `flights %>% filter(carrier == "AS")` is the same as `filter(flights, carrier == "AS")`
- The `filter()` command filters down to only those rows of the data that satisfy the condition inside the parentheses.
- The condition `carrier == "AS"` is testing that the variable `carrier` (which the `filter()` command knows about because it was given `flights` as input by piping) equals the value `"AS"`
- So the output from all of this so far should be a new data frame with only those rows of `flights` corresponding to the carrier Alaskan Airlines.
- Now let's look back at the first part of the line.
- `all_alaska_flights` is the name of a new data frame being created.
- The left arrow `<-` means to assign value, i.e. the new data frame gets the value output by all the code to the right of the `<-`.