HW 10

Lance

April 2023

Instructions

Write this homework acting as if I don't know what I asked you. For example, don't simply list question numbers for the headings. If you gave this document to someone else who didn't know the assignment, they should be able to understand what you did by reading the headings, code, and accompanying text.

Look to my HW1 and RMarkdown Organization examples for how to write good headings and organize your assignment.

This HW is worth 10 total points.

1. Change the author and date fields in the header above to your name and the date.
2. Make sure to load any packages you may need right at the start. Do *NOT* include the `learnr` package, ever, unless you are writing an interactive Tutorial (which you won't do in this) - this will cause problems.
3. Ensure that no chunks have the `include = FALSE` or `echo = FALSE` option, as I want to be able to see *all* your code and output.
4. Brief but descriptive headings and document organization (answers under headings, text near relevant code, brief explanatory text as indicated below, etc.) (1 pt)
5. Recall this activity from the previous homework: Write a for-loop that will, when you run the code, take the tibble `flights` and export *3* CSV files of flights *from January*: one for each origin airport (EWR, JFK, and LGA).

   We accomplished it with code like this:

```
pacman::p_load(nycflights13, tidyverse, fs)

origin <- c("EWR", "JFK", "LGA")

for(i in origin){

  origin <- flights %>%
#group_by(origin) %>%
  filter(origin == i & month == 1) #%>%
#write_csv(paste0("./origin_", i, ".csv"))

  write_csv(origin, paste0("./origin_", i, ".csv"))
}
```

I'd like you to re-import these files using a `purrr` function.

A complete answer will include: code to read in the files and some sort of output to show me they all read in to a single new file; and a detailed explanation of what the `purrr` function you used does, including a description of what each argument in the code means. (3 pts)

6. Use a `purrr` function to count the number of `NA` values in each column in the `penguins` data frame from the `palmerpenguins` package. You may output the information in any format you want (e.g. list, data frame) as long as each column name and the number of `NA` values is listed.

   If you've mastered tilde dot notation (or maybe even if not), you can accomplish this in a single line.

   HINT: Think about which task with the `gapminder` data this is similar to in Dr. Barter's tutorial. (3 pts)

7. Use a `purrr` function to create a list of 3 plots, split by airport. The plot can be any plot you want that can sensibly be split by airport.

   Access and print the second element of that list. (3 pts)

To submit this assignment:

Ideally, knit straight to PDF by changing `html_document` to `pdf_document` in line 5 above. This should work as long as you properly installed LaTeX in Tutorial 0.1. Otherwise:

1. Knit to HTML. An HTML document should open automatically in another RStudio window.
2. Click "Open in Browser" in that HTML document. It should open as a webpage in your default browser (e.g. Chrome).
3. Click Ctrl+P/Command+P, but instead of printing a hard copy on your printer click "Save as PDF."
4. Save and upload that document to Canvas.

——BEGIN ANSWER BELOW——

Import Packages

```
pacman::p_load(tidyverse, nycflights13, palmerpenguins)
```

Importing Files with `purrr`

Our first task is to import the files that the example code above has created. These files are in the form `origin_XXX.csv`, where `XXX` can be one of three codes corresponding to each of New York City's airports. We accomplish this with the `map_dfr()` function from `purrr`. We first create a character vector `airnames` that contains the airport codes, then create the function `importFiles()` that takes an input `x` and tries to import the file in the local directory called `origin_<x>.csv`. We then use the `dfr` variant of the `map_XXX()` set of functions from `purrr`. The `map_XXX()` functions iterate across a collection and applies a function for every element in the collection, and the `map_dfr()` variant returns a dataframe "created by row-binding" (according to the `purrr` cheat sheet). So for each element in `airnames`, `map_dfr()` applies the function `importFiles()` with that element in `airnames` as the argument, and row-appends it to a dataframe that it returns at the end of the iteration. Since the data is for flights in January, we can quickly verify that this operation has done what we intended by creating the January flights dataframe from `flights`, and comparing the two dataframes.

```
# Filenames are in the form origin_XXX.csv

airnames <- c("EWR", "JFK", "LGA")

importFiles <- function(x){
  read.csv(paste0("./origin_", x, ".csv"))
}

fnew <- map_dfr(.x = airnames,
       .f = importFiles)

# Verify January flights

flights %>%
  filter(month == 1) %>%
  dim()
```

```
## [1] 27004    19
```

```
dim(fnew)
```

```
## [1] 27004    19
```

As the two `dim()`s show, the two dataframes are the same size, meaning we probably have what we want.

Counting `NA`s in `penguins` with `purrr`

We will now count the number of `NA`s in each column of the `penguins` dataframe from `palmerpenguins`. To do so, we do a `map()` with `penguins` as the collection to be iterated over and `sum(is.na())` as the function to be applied. We use tilde dot notation to create the anonymous function inside `map()`, avoiding the need to explicitly define a function outside of this operation.

```
NA_counts <- map(.x = penguins,
    .f = ~sum(is.na(.x)))
NA_counts
```

```
## $species
## [1] 0
##
## $island
## [1] 0
##
## $bill_length_mm
## [1] 2
##
## $bill_depth_mm
## [1] 2
##
## $flipper_length_mm
## [1] 2
```

```
##
## $body_mass_g
## [1] 2
##
## $sex
## [1] 11
##
## $year
## [1] 0
```

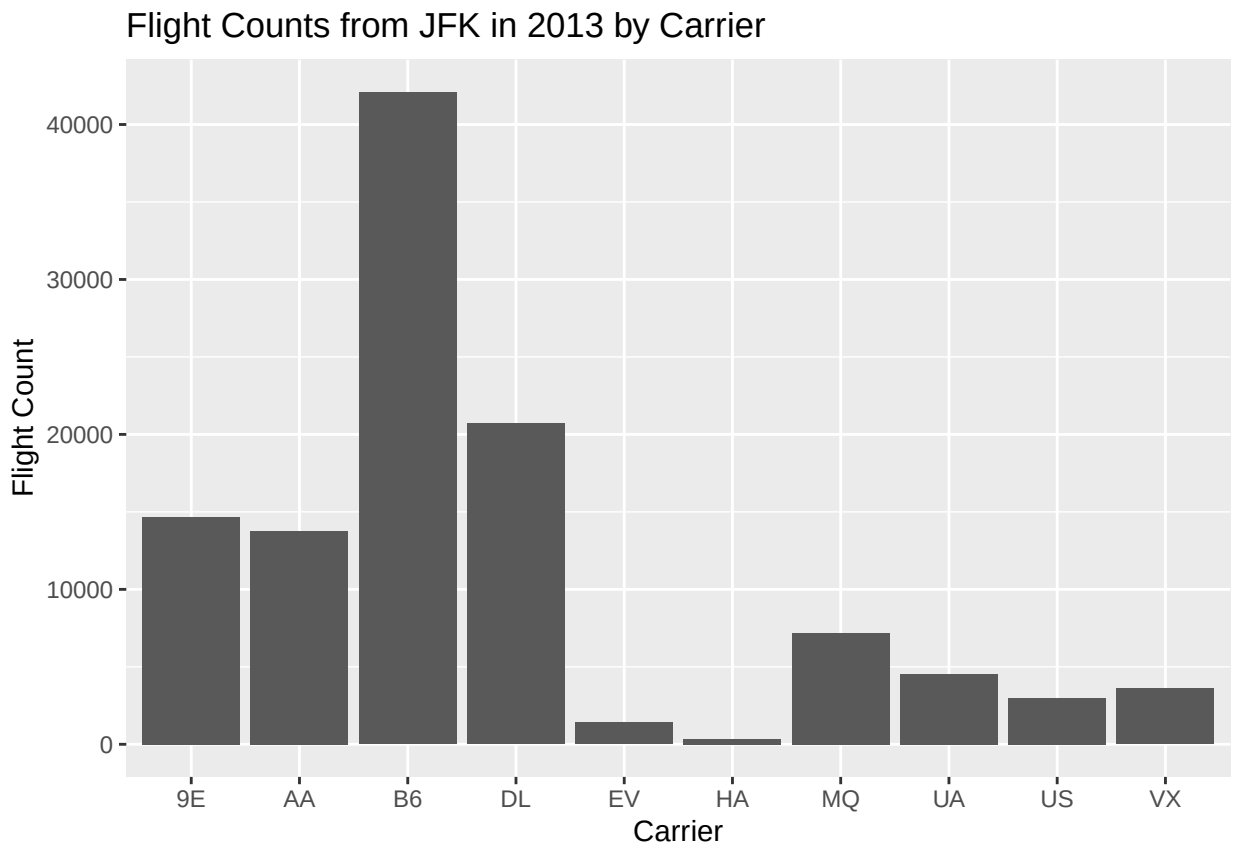
Plots of NYC flight by Carrier and Airport

We will now create bar plots of flight counts by carrier from each airport in NYC. That is to say, we will display the number of flights by each carrier for each of the 3 airports in NYC. To do so, we first define a function `makePlot()`, which takes an input argument that tells it what origin airport to make the plot for. The rest of the function creates the bar plot with the appropriate labels to show the number of flights by each carrier from the airport with the inputted airport code. We then use `map()` to apply the `makePlot()` function on each of the airports whose codes are stored in the `airnames` vector from earlier. Since we used `map()`, the result is of the `list` form, meaning that we can use `[[]]` to extract the element we want.

```
makePlot <- function(orig) {
  flights %>%
    filter(origin == orig) %>%
    ggplot(mapping = aes(x=carrier)) +
      geom_bar() +
      xlab("Carrier") +
      ylab("Flight Count") +
      ggtitle(paste0("Flight Counts from ",
                     orig,
                     " in 2013 by Carrier"))
}

carrierPlots <- map(.x = airnames,
                    .f = ~makePlot(.x))

carrierPlots[[2]]
```


Flight Counts from JFK in 2013 by Carrier
Flight Count
40000
30000
20000
10000
0
9E
AA
B6
DL
EV
HA
MQ
UA
US
VX
Carrier