COMPSCIX 415.2 Homework 4

Bryan Hee

February 20, 2018

Section 5.6.7

Question 2

The following R chunks represents both the original code and the updated code that gives the same tibble without using "count"

```
# original code chunk
    not_cancelled <- flights %>%
      filter(!is.na(dep_delay), !is.na(arr_delay))
    not_cancelled %>% count(dest)
```

```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
## 10 BHM     269
## # ... with 94 more rows
```

```
# updated code chunk
    not_cancelled <- flights %>%
      filter(!is.na(dep_delay), !is.na(arr_delay))
    not_cancelled %>%
    group_by(dest) %>%
    summarise(
      n = n()
    )
```

```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
```

```
## 10 BHM     269
## # ... with 94 more rows
```

```
# original code chunk
    not_cancelled <- flights %>%
      filter(!is.na(dep_delay), !is.na(arr_delay))
    not_cancelled %>% count(tailnum, wt = distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 N0EGMQ  239143
##  3 N10156  109664
##  4 N102UW   25722
##  5 N103US   24619
##  6 N104UW   24616
##  7 N10575  139903
##  8 N105UW   23618
##  9 N107US   21677
## 10 N108UW   32070
## # ... with 4,027 more rows
```

```
# updated code chunk
    not_cancelled <- flights %>%
      filter(!is.na(dep_delay), !is.na(arr_delay))
    not_cancelled %>%
    group_by(tailnum) %>%
    summarise(
      n = sum(distance)
    )
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 N0EGMQ  239143
##  3 N10156  109664
##  4 N102UW   25722
##  5 N103US   24619
##  6 N104UW   24616
##  7 N10575  139903
##  8 N105UW   23618
##  9 N107US   21677
## 10 N108UW   32070
## # ... with 4,027 more rows
```

Question 4

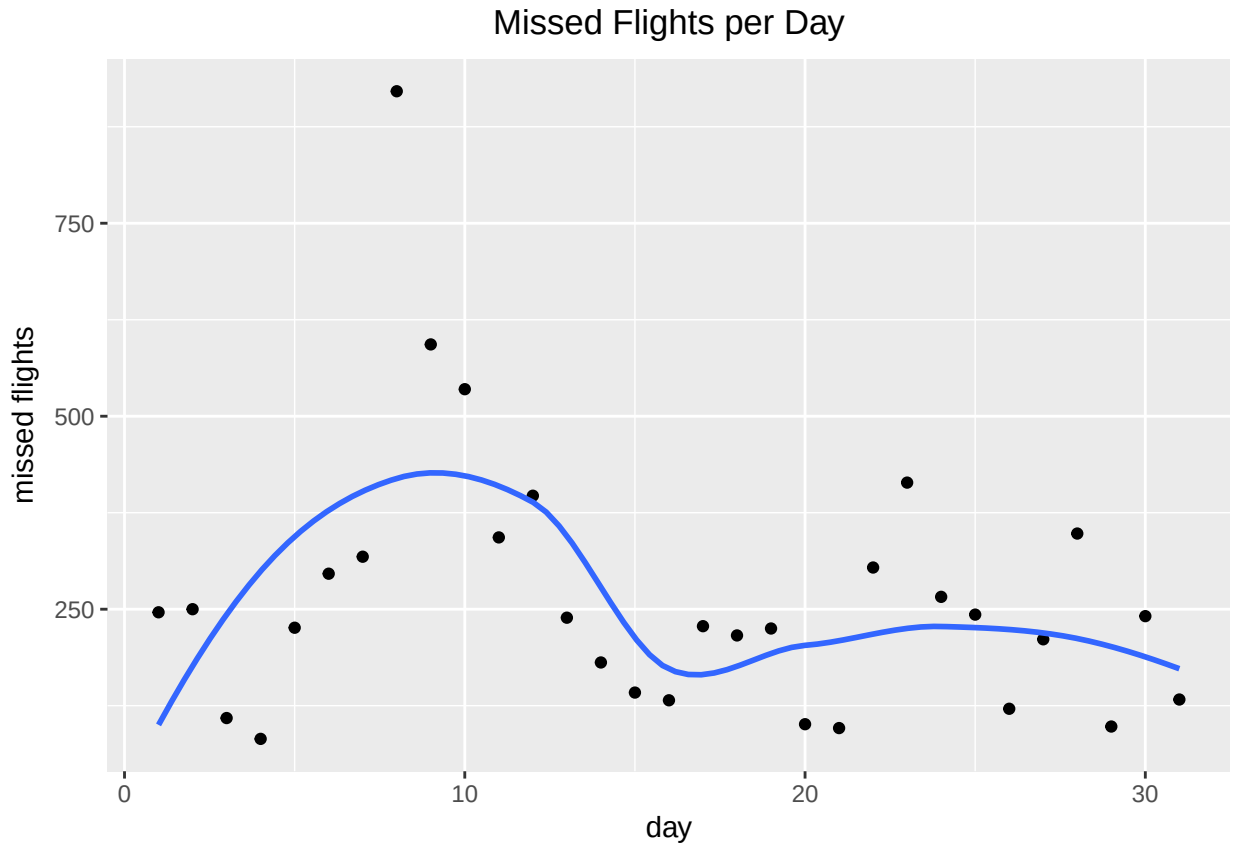
Based off of the graph below, there doesn't appear to be any correlation between day of the month and number of cancelled flights. There also shouldn't be a correlation between cancelled flights per day and average delay time because if the flight is cancelled the delay by default is infite or NA.

```
cancelled <- flights %>%
  filter(is.na(dep_time), is.na(arr_time))
```

```r
cancelled %>%
  group_by(day) %>%
  summarise(
    missed_flights = n()
  ) %>%

ggplot(mapping = aes(x = day, y = missed_flights)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  labs(y ="missed flights", title = "Missed Flights per Day") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## `geom_smooth()` using method = 'loess'
```

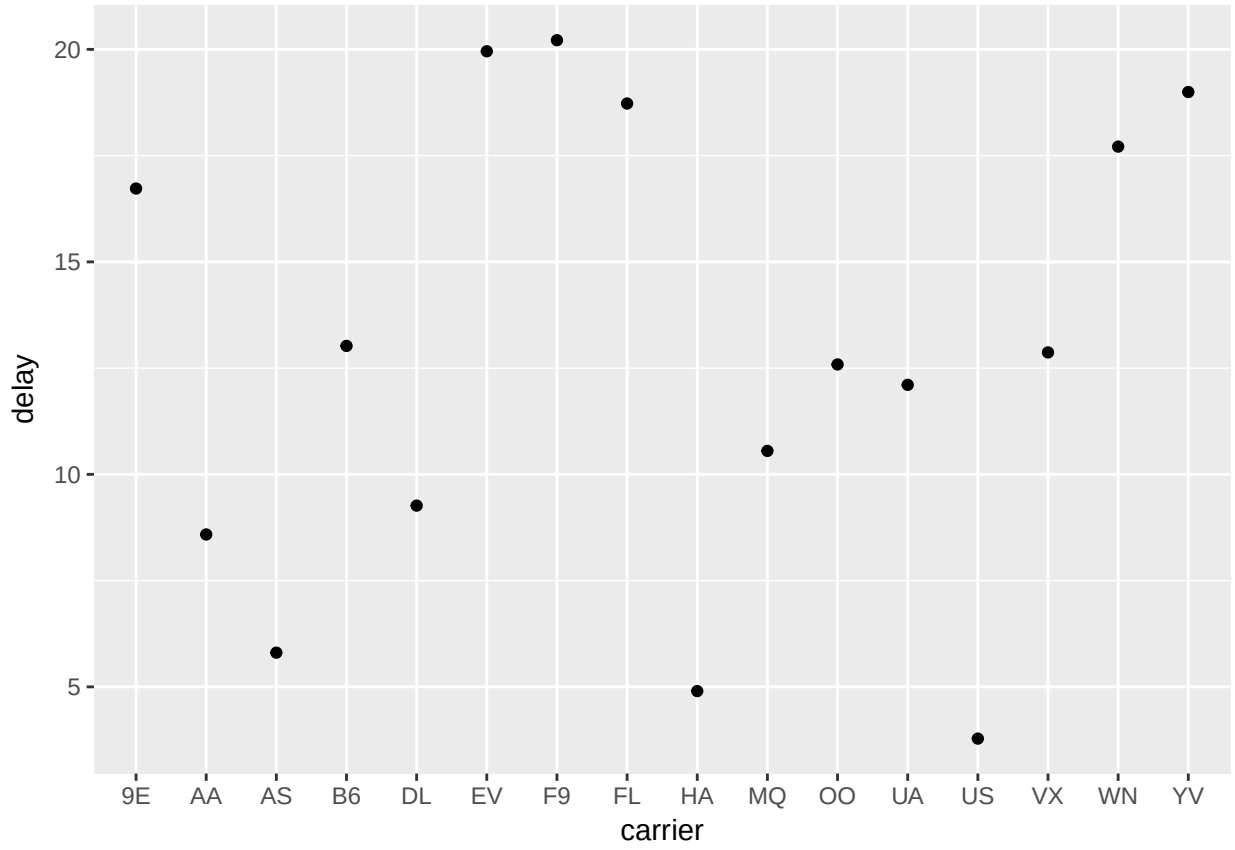


Question 5

Based off of the average delay time, the worst carrier is F9, or Frontier Airlines. However, based off of the total delay times (in the flights data set) per airline, EV, or ExpressJet Airlines, Inc. is the worst.

```r
carriers <- flights %>%
  group_by(carrier) %>%
  summarise(
    delay = (mean(dep_delay, na.rm = TRUE)),
    total_flights = n(),
    total_delay = delay * total_flights
  )
```

```
ggplot(data = carriers, mapping = aes(x = carrier, y = delay)) +
  geom_point()
```



Question 6

The sort function, when applied to count() should arrange the variable in question from smallest count to largest. You could use it to rank different attributes. I.e. on the previous question you could sort carriers based on the number of delays or number of cancelled flights to get an understanding of what is the worst and what is the best carrier.

Section 10.5

Question 1

You can tell if an object is a tibble by running it in the console and seeing what the result is. If it is a tibble, the first line will be "# A tibble:" and then define the tibble size. It will also display only the first few columns' names, variable type, and will only print the first 10 rows in the object. It will then tell you the number of additional rows and the other variable columns (by name) included in the tibble.

A data frame on the otherhand, won't display the metadata (variable type, number of rows, number of variables) and will try to output the entire dataframe until it reaches a max.print limit.

Question 2

See the comments in the R chunk below for the differences between the code run on a data frame versus tibble. Also, with data frames using "[" will return either a data frame or a vector, however on a tibble it will only return a tibble. Hence why, you get three tibbles when you run the code on a tibble, but only two data frames (because one result is a vector)

```
df <- tibble(abc = 1, xyz = "a")

#because there is no column named "x" running this subset on the tibble will not work and will generate
df$x
```

```
## Warning: Unknown or uninitialised column: 'x'.
```

```
## NULL
```

```
#this prints the full column of the tibble, but only the first observation of the dataframe
df[, "xyz"]
```

```
## # A tibble: 1 x 1
##   xyz
##   <chr>
## 1 a
```

```
#this works the same on both a tibble and data.frame
df[, c("abc", "xyz")]
```

```
## # A tibble: 1 x 2
##     abc xyz
##   <dbl> <chr>
## 1  1.00 a
```

Question 3

If you have a name of a variable stored in an object, you can extract it using the following code: "tibble_name$variable_name" or by using "tibble_name[[variable_name]]" or "tibble_name[[position]]"

Question 6

The option that controls the number of additional column names printed at the footer of a tibble is driven by "options(tibble.width =)" which will allow the user to define the number of columns to print. The remaining columns will be printed in the footer.

Section 12.3.3

Question 2

The following code failed because instead of passing gather() the names of the columns that represent values as a string, it passed them as values. The following is the corrected gather function calling for '1999' rather than 1999, etc.

```
table4a %>%
  gather('1999', '2000', key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Question 3

Spreading the provided tibble fails because there are two entries for Pillip Wood's age. You could add a new firs column that indexes each row to assign Phillip Woods' two entries as unique. I.e. Phillip Woods 1 and Phillip Woods 2. See the spread tibble below:

```
people <- tribble(
  ~index, ~name,              ~key,    ~value,
  #-----------------/--------/------
  "1", "Phillip Woods",   "age",       45,
  "1", "Phillip Woods",   "height",   186,
  "2", "Phillip Woods",   "age",       50,
  "3", "Jessica Cordero", "age",       37,
  "3", "Jessica Cordero", "height",   156
) %>%

  spread(key = "key", value = "value")
```

Question 4

Gather is needed to tidy the tibble because "male" and "female" are values not variables. See below for the gathered tibble:

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes",     NA,    10,
  "no",      20,    12
) %>%

gather("male", "female", key = "gender", value = "number of pregnant")
```

Section 12.4.3

Question 1

In separate, the extra argument controls what happens when the separator is a character vector and there are too many pieces. If you use "drop" it will drop any extra values without warning. The default is "warn" which will print a warning message, and if you use "merge" the tibble will concatenate any extra observations to fit the required number of column dileneations. The fill argument controls what happens if there are not enough pieces. The default is again "warn" with "left" moving all observations to the right, and "right" pulling all observations from the right and therefore moving them to the left.

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"), extra = "merge")
```

```
## # A tibble: 3 x 3
##   one   two   three
## * <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f,g
## 3 h     i     j
```

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), fill = "left")
```

```
## # A tibble: 3 x 3
##   one   two   three
## * <chr> <chr> <chr>
## 1 a     b     c
## 2 <NA>  d     e
## 3 f     g     i
```

Question 2

The remove argument from the unite() and separate() functions remove the input column(s) from the output dataframe. This would be done if you did not require the initial columns. I.e. if you were to remove "century" and "year" columns from the table5 example, you would not have the three columns in the your data frame: "century", "year", "new"; but instead would just have the "new" column providing one cohesive year.

Additional Question

```
holder_name <- 'C:/Users/BryanHee/OneDrive - stok LLC/Intro to Data Science/baby_names.txt'
baby_name_data <- read_delim(file = holder_name, "|")
```

```
## Parsed with column specification:
## cols(
##   year = col_integer(),
##   sex = col_character(),
##   name = col_character(),
##   n = col_integer(),
##   prop = col_double()
## )
```

```
glimpse(baby_name_data)
```

```
## Observations: 30,000
## Variables: 5
## $ year <int> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 188...
## $ sex  <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F...
## $ name <chr> "Mary", "Anna", "Emma", "Elizabeth", "Minnie", "Margaret"...
## $ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 128...
## $ prop <dbl> 0.072384329, 0.026679234, 0.020521700, 0.019865989, 0.017...
```

```
write_rds(baby_name_data, 'C:/Users/BryanHee/OneDrive - stok LLC/Intro to Data Science/baby_names.rds')
read_rds('C:/Users/BryanHee/OneDrive - stok LLC/Intro to Data Science/baby_names.rds') %>%
  glimpse()
```

```
## Observations: 30,000
## Variables: 5
## $ year <int> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 188...
## $ sex  <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F...
## $ name <chr> "Mary", "Anna", "Emma", "Elizabeth", "Minnie", "Margaret"...
## $ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 128...
## $ prop <dbl> 0.072384329, 0.026679234, 0.020521700, 0.019865989, 0.017...
```