COMPSCIX 415.2 Homework 4

Ganesh Saravanan

6/29/2018

My Github repository for my assignments can be found at this URL:https://github.com/gsaravanan1/rstudiodemo.git

```
library(mdsr)
library(tidyverse)
library(ggplot2)
library(nycflights13)
library(tibble)
```

#Section 5.6.7: #2, #4 and #6 only. Extra Credit: Do #5

2 Come up with another approach that will give you the same output as not_cancelled %>% count(dest) and not_cancelled %>% count(tailnum, wt = distance) (without using count()).

```
not_cancelled <- filter(flights, !is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>%
  group_by(dest) %>%
  tally()
```

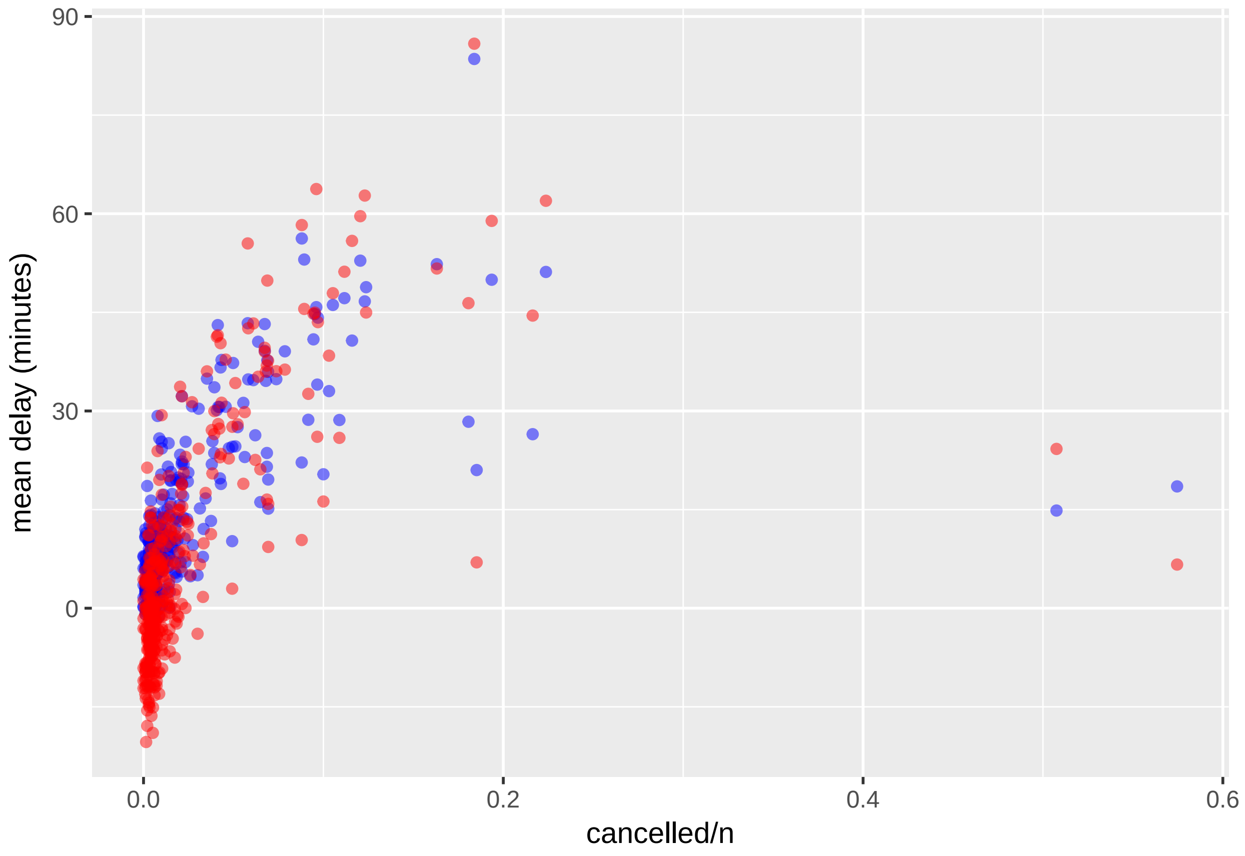
```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
## 10 BHM     269
## # ... with 94 more rows
```

```
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
```

4 Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

```
flights %>%
  mutate(dep_date = lubridate::make_datetime(year, month, day)) %>%
  group_by(dep_date) %>%
  summarise(cancelled = sum(is.na(dep_delay)),
            n = n(),
            mean_dep_delay = mean(dep_delay,na.rm=TRUE),
            mean_arr_delay = mean(arr_delay,na.rm=TRUE)) %>%
    ggplot(aes(x= cancelled/n)) +
    geom_point(aes(y=mean_dep_delay), colour='blue', alpha=0.5) +
    geom_point(aes(y=mean_arr_delay), colour='red', alpha=0.5) +
    ylab('mean delay (minutes)')
```



No, the proportion of cancelled flights not related to the average delay. Mostly the higher the cancelled flights leads to higher delay #'s.

5 Which carrier has the worst delays? Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about flights %>% group_by(carrier, dest) %>% summarise(n()))

```r
flights %>%
    filter(arr_delay > 0) %>%
    group_by(carrier) %>%
    summarise(average_arr_delay = mean(arr_delay, na.rm=TRUE)) %>%
    arrange(desc(average_arr_delay))
```

```
## # A tibble: 16 x 2
##    carrier average_arr_delay
##    <chr>               <dbl>
##  1 OO                   60.6
##  2 YV                   51.1
##  3 9E                   49.3
##  4 EV                   48.3
##  5 F9                   47.6
##  6 VX                   43.8
##  7 FL                   41.1
##  8 WN                  40.7
##  9 B6                  40.0
## 10 AA                  38.3
## 11 MQ                  37.9
## 12 DL                  37.7
## 13 UA                  36.7
## 14 HA                  35.0
## 15 AS                  34.4
## 16 US                  29.0
```

```
flights %>%
  summarise(n_distinct(carrier),
            n_distinct(origin),
            n_distinct(dest))
```

```
## # A tibble: 1 x 3
##   `n_distinct(carrier)` `n_distinct(origin)` `n_distinct(dest)`
##                   <int>                <int>              <int>
## 1                    16                    3                105
```

It seems to be more of an airport issues rather than an airline issue.

6 What does the sort argument to count() do. When might you use it?

The sort argument to count() sorts by descending order of n. This is useful because often the most common group is the most important.

#Section 10.5: #1, #2, #3 and #6 only

1 How can you tell if an object is a tibble? (Hint: try printing mtcars, which is a regular data frame).

```
mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
class(mtcars)
```

```
## [1] "data.frame"
```

Tibbles will only print out a limited number of rows and show the class on top of each column

2 Compare and contrast the following operations on a data.frame and equivalent tibble. What is different? Why might the default data frame behaviours cause you frustration?

```r
df <- data.frame(abc = 1, xyz = "a")
df$x
```

```
## [1] a
## Levels: a
```

```r
df[, "xyz"]
```

```
## [1] a
## Levels: a
```

```r
df[, c("abc", "xyz")]
```

```
##   abc xyz
## 1   1   a
```

```r
tbl <- as_tibble(df)
tbl$x
```

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

```
## NULL
```

```r
tbl[, "xyz"]
```

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

```
tbl[, c("abc", "xyz")]
```

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

A data frame will partially complete the column whereas in tibble, the same operation will return a tibble with a single column.Tibble does not do partial matching. It's perfectly fine to do this in data frame

3 If you have the name of a variable stored in an object, e.g. var <- "mpg", how can you extract the reference variable from a tibble?

We will not be able to use $ to subset the columns. Instead we need to use [“].

```
tibble_mtcars <- as.tibble(mtcars)
var <- 'mpg'
tibble_mtcars[var]
```

```
## # A tibble: 32 x 1
##      mpg
##    <dbl>
##  1  21
##  2  21
##  3  22.8
##  4  21.4
##  5  18.7
##  6  18.1
##  7  14.3
##  8  24.4
##  9  22.8
## 10  19.2
## # ... with 22 more rows
```

6 What option controls how many additional column names are printed at the footer of a tibble?

By default, information of all remaining columns are printed at the footer. To limit the number of additional column information, we can use the argument n_extra.

# Section 12.3.3: #2, #3 and #4 only

2 Why does this code fail?

```
#table4a %>%
 # gather(1999, 2000, key = "year", value = "cases")
#> Error in inds_combine(.vars, ind_list): Position must be between 0 and n
```

The code fails because the column names 1999 and 2000 are not standard and thus needs to be quoted. The tidyverse functions will interpret 1999 and 2000 without quotes as looking for the 1999th and 2000th column of the data frame.

```
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
```

3 Why does spreading this tibble fail? How could you add a new column to fix the problem?

```
people <- tribble(
  ~name,             ~key,    ~value,
  #-----------------|--------|------
  "Phillip Woods",   "age",       45,
  "Phillip Woods",   "height",   186,
  "Phillip Woods",   "age",       50,
  "Jessica Cordero", "age",       37,
  "Jessica Cordero", "height",   156
)
```

Spreading the data frame fails because there are two rows with “age” for “Phillip Woods”

```
people <- tribble(
  ~name,             ~key,    ~value, ~obs,
  #-----------------|--------|------|------
  "Phillip Woods",   "age",       45, 1,
  "Phillip Woods",   "height",   186, 1,
  "Phillip Woods",   "age",       50, 2,
  "Jessica Cordero", "age",       37, 1,
  "Jessica Cordero", "height",   156, 1
)
spread(people, key, value)
```

```
## # A tibble: 3 x 4
##   name              obs   age height
##   <chr>           <dbl> <dbl>  <dbl>
## 1 Jessica Cordero     1    37    156
## 2 Phillip Woods       1    45    186
## 3 Phillip Woods       2    50     NA
```

4 Tidy the simple tibble below. Do you need to spread or gather it? What are the variables?

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

You need to gather it. The variables are:

pregnant: logical ("yes", "no") female: logical count: integer

# Section 12.4.3: #1 and #2 only

1 What do the extra and fill arguments do in separate()? Experiment with the various options for the following two toy datasets.

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

```
## Warning: Expected 3 pieces. Additional pieces discarded in 1 rows [2].
```

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

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

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1 rows [2].
```

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

extra: if sep is a character vector, this controls what happens when there are too many pieces. There are three valid options:

"warn" (the default): emit a warning and drop extra values.

"drop": drop any extra values without a warning.

"merge": only splits at most length(into) times

fill: if sep is a character vector, this controls what happens when there are not enough pieces. There are three valid options:

"warn" (the default): emit a warning and fill from the right

“right”: fill with missing values on the right

“left”: fill with missing values on the left

2 Both unite() and separate() have a remove argument. What does it do? Why would you set it to FALSE?

If TRUE, remove input column from output data frame.

DATA IMPORT:

```r
my_data <- read.delim("~/Downloads/baby_names.txt")
glimpse(my_data)
```

```
## Observations: 30,000
## Variables: 1
## $ year.sex.name.n.prop <fct> 1880|F|Mary|7065|0.0723843285111266, 1880...
```

```r
write.csv(my_data, "~/Downloads/baby_names.csv")
my_data1 <- read.csv("~/Downloads/baby_names.csv")
glimpse(my_data1)
```

```
## Observations: 30,000
## Variables: 2
## $ X                    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13...
## $ year.sex.name.n.prop <fct> 1880|F|Mary|7065|0.0723843285111266, 1880...
```