STAT 231: Problem Set 2A

vaibhav Shah

due by 5 PM on Monday, March 1

In order to most effectively digest the textbook chapter readings – and the new R commands each presents – series A homework assignments are designed to encourage you to read the textbook chapters actively and in line with the textbook's Prop Tip of page 33:

"**Pro Tip**: If you want to learn how to use a particular command, we highly recommend running the example code on your own"

A more thorough reading and light practice of the textbook chapter prior to class allows us to dive quicker and deeper into the topics and commands during class. Furthermore, learning a programming lanugage is like learning any other language – practice, practice, practice is the key to fluency. By having two assignments each week, I hope to encourage practice throughout the week. A little coding each day will take you a long way!

Series A assignments are intended to be completed individually. While most of our work in this class will be collaborative, it is important each individual completes the active readings. The problems should be straightforward based on the textbook readings, but if you have any questions, feel free to ask me!

Steps to proceed:

1. In RStudio, go to File > Open Project, navigate to the folder with the course-content repo, select the course-content project (course-content.Rproj), and click "Open"
2. Pull the course-content repo (e.g. using the blue-ish down arrow in the Git tab in upper right window)
3. Copy ps2A.Rmd from the course repo to your repo (see page 6 of the GitHub Classroom Guide for Stat231 if needed)
4. Close the course-content repo project in RStudio
5. Open YOUR repo project in RStudio
6. In the ps2A.Rmd file in YOUR repo, replace "YOUR NAME HERE" with your name
7. Add in your responses, committing and pushing to YOUR repo in appropriate places along the way
8. Run "Knit PDF"
9. Upload the pdf to Gradescope. Don't forget to select which of your pages are associated with each problem. *You will not get credit for work on unassigned pages (e.g., if you only selected the first page but your solution spans two pages, you would lose points for any part on the second page that the grader can't see).*

1. NYC Flights

a.

In Section 4.3.1, the `flights` and `carrier` tables within the `nycflights13` package are joined together. Recreate the `flightsJoined` dataset from page 80. Hint: make sure you've loaded the `nycflights13` package before referring to the data tables (see code on page 79).

```
library(nycflights13)
flightsJoined <- flights %>%
  inner_join(airlines, by = c("carrier" = "carrier"))
flightsJoined
```

```
## # A tibble: 336,776 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     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
## # ... with 336,766 more rows, and 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>,
## #   name <chr>
```

b.

Now, create a new dataset `flightsJoined2` that:

- creates a new variable, `distance_km`, which is distance in kilometers (note that 1 mile is about 1.6 kilometers)
- keeps only the variables: `name`, `flight`, `arr_delay`, and `distance_km`

- keeps only observations where distance is less than 500 kilometers

Hint: see examples in Section 4.1 for subsetting datasets and creating new variables.

```
flightsJoined2 <- select(flightsJoined, name, flight, arr_delay, distance) %>% mutate(distance_km = dis
flightsJoined2 <- select(filter(flightsJoined2, distance_km < 500), name, flight, arr_delay, distance_k
flightsJoined2
```

```
## # A tibble: 54,921 x 4
##    name                     flight arr_delay distance_km
##    <chr>                     <int>     <dbl>       <dbl>
##  1 ExpressJet Airlines Inc.   5708       -14        366.
##  2 JetBlue Airways             1806        -4        299.
##  3 Southwest Airlines Co.      4646       -19        296
##  4 ExpressJet Airlines Inc.    4144        12        339.
##  5 JetBlue Airways             1002       -10        299.
##  6 JetBlue Airways              102         5        482.
##  7 JetBlue Airways               20        -1        422.
##  8 JetBlue Airways               44         4        334.
##  9 JetBlue Airways             1172       -19        320
## 10 American Airlines Inc.      1838       -22        299.
## # ... with 54,911 more rows
```

c.

Lastly, using the functions introduced in Section 4.1.4, compute the number of flights (call this `N`), the average arrival delay (call this `avg_arr_delay`), and the average distance in kilometers (call this `avg_dist_km`) among these flights with distances less than 500 km (i.e. working off of `flightsJoined2`) *grouping by the carrier name*. Sort the results in descending order based on `avg_arr_delay`.

Getting NAs for `avg_arr_delay`? That happens when some observations are missing that data. Before grouping and summarizing, add a line to exclude observations with missing arrival delay information using `filter(is.na(arr_delay)==FALSE)`.

```
PartC <- flightsJoined2 %>%
  filter(is.na(arr_delay)==FALSE) %>%
  group_by(name) %>%
  summarize(
    N = n(), avg_arr_delay = mean(arr_delay), avg_dist_km = mean(distance_km)) %>%
  arrange(desc(avg_arr_delay))
PartC
```

```
## # A tibble: 11 x 4
##    name                         N avg_arr_delay avg_dist_km
##    <chr>                    <int>         <dbl>       <dbl>
##  1 Mesa Airlines Inc.         286        18.0          360.
##  2 ExpressJet Airlines Inc. 14753        15.6          373.
##  3 Envoy Air                 2741        11.0          351.
##  4 JetBlue Airways          13443         8.66         385.
##  5 Endeavor Air Inc.         6144         6.82         339.
##  6 Southwest Airlines Co.     200         4.92         272.
##  7 United Air Lines Inc.     3307         4.09         320.
##  8 SkyWest Airlines Inc.        1         3            366.
##  9 US Airways Inc.           9093         2.22         308.
## 10 American Airlines Inc.    1428         1.88         299.
## 11 Delta Air Lines Inc.      1201        -0.643        325.
```

2. Baby names

a.

Working with the `babynames` data table in the `babynames` package, create a dataset `babynames2` that only includes years 2000 to 2017.

```
library(babynames)
babynames2 <- babynames %>% filter(year>=2000 & year<=2017)
babynames2
```

```
## # A tibble: 591,925 x 5
##     year sex   name          n    prop
##    <dbl> <chr> <chr>     <int>   <dbl>
##  1  2000 F     Emily     25953 0.0130
##  2  2000 F     Hannah    23080 0.0116
##  3  2000 F     Madison   19967 0.0100
##  4  2000 F     Ashley    17997 0.00902
##  5  2000 F     Sarah     17697 0.00887
##  6  2000 F     Alexis    17629 0.00884
##  7  2000 F     Samantha  17266 0.00866
##  8  2000 F     Jessica   15709 0.00787
##  9  2000 F     Elizabeth 15094 0.00757
## 10  2000 F     Taylor    15078 0.00756
## # ... with 591,915 more rows
```

b.

Following the code presented in Section 5.2.4, create a dataset called `BabyNarrow` that summarizes the total number of people with each name (born between 2000 and 2017), grouped by sex. (Hint: follow the second code chunk on page 102, but don't filter on any particular names.) Look at the dataset. Why have we called this dataset "narrow"?

> ANSWER: We've truncated the number of rows by summing over years.

```
BabyNarrow <- babynames2 %>%
  group_by(name, sex) %>%
  summarise(total = sum(n))
```

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

```
BabyNarrow
```

```
## # A tibble: 73,332 x 3
## # Groups:   name [67,063]
##    name      sex   total
##    <chr>     <chr> <int>
##  1 Aaban     M       107
##  2 Aabha     F        35
##  3 Aabid     M        10
##  4 Aabir     M         5
##  5 Aabriella F         32
##  6 Aada      F          5
##  7 Aadam     M        202
##  8 Aadan     M        130
##  9 Aadarsh   M        199
## 10 Aaden     F          5
## # ... with 73,322 more rows
```

c.

Now, following the code chunk presented on page 103*, put the data into a wide format (call the new dataset `BabyWide`), and only keep observations where both `M` and `F` are greater than 10,000. Compute the `ratio` (as `pmin(M/F, F/M)`) and identify the top three names with the largest ratio. (Note: these names could be different from the ones found on page 103 since we limited the dataset to years 2000-2017 and names with greater than 10,000 individuals.)

- Note: you can use the `pivot_wider()` function instead of the `spread()` function if using the 2nd edition of the textbook (e.g., see Section 6.2.2 and 6.2.3 in the 2nd edition). I find `pivot_wider()` and `pivot_longer()` to be more intuitive than `spread()` and `gather()`.

  ANSWER:Justice, Skyler, and Quinn have the greatest ratios.

```
# this will bring up "Pivoting Introduction" vignette in your Help tab
#vignette("pivot")
BabyWide <- BabyNarrow %>%
  group_by(name, sex) %>%
  spread(key = sex, value = total, fill=0) %>%
  filter( F >10000 & M > 10000) %>%
  mutate(ratio = pmin( M / F, F / M)) %>%
  arrange (desc(ratio))


BabyWide
```

```
## # A tibble: 25 x 4
## # Groups:   name [25]
##    name         F     M ratio
##    <chr>    <dbl> <dbl> <dbl>
##  1 Justice 10947 11267 0.972
##  2 Skyler  17120 22154 0.773
##  3 Quinn   25022 19080 0.763
##  4 Amari   11778 15676 0.751
##  5 Casey   12109 16809 0.720
##  6 Riley   89827 59823 0.666
##  7 Peyton  61217 39261 0.641
##  8 Emerson 18592 11742 0.632
##  9 Charlie 13255 21243 0.624
## 10 Dakota  21950 35840 0.612
## # ... with 15 more rows
```

d.

Lastly, use the `gather()` function (or the `pivot_longer()` function) to put the dataset back into narrow form. Call this dataset `BabyNarrow2`. Hint: see Section 5.2.3. Why are the number of observations in `BabyNarrow2` different from that in `BabyNarrow`?

> ANSWER: BabyNarrow2 splits each name into two rows, one for each gender, and the BabyWide dataset only kept names with at least 10,000 instances in both genders. That's why the second BabyNarrow has much fewer observations even though the names were split by gender.

```r
BabyNarrow2 <- BabyWide %>%
  gather(key = sex, value = total, F, M )
BabyNarrow2
```

```
## # A tibble: 50 x 4
## # Groups:   name [25]
##    name    ratio sex   total
##    <chr>   <dbl> <chr> <dbl>
##  1 Justice 0.972 F     10947
##  2 Skyler  0.773 F     17120
##  3 Quinn   0.763 F     25022
##  4 Amari   0.751 F     11778
##  5 Casey   0.720 F     12109
##  6 Riley   0.666 F     89827
##  7 Peyton  0.641 F     61217
##  8 Emerson 0.632 F     18592
##  9 Charlie 0.624 F     13255
## 10 Dakota  0.612 F     21950
## # ... with 40 more rows
```