Data Wrangling Case Assignment

In this assignment we'll be getting used to the general "tidyverse" approach to working with and manipulating data. You'll demonstrate your knowledge of the `tidyverse` approach and vocabulary for manipulating and examining a dataset in order to uncover answers to various questions. We'll also get some experience dealing with some common pitfalls of joining data and discover some ways to sniff out potential issues.

As a reminder, you'll start this code assignment (and all code assignments) by cloning the repository to your local machine.

All of the activities for this assignment are to be completed in a separate R script in this repository. Because this is your first real case assignment, I've provided quite a lot of narrative commentary in these instructions, as well as some starter code in the `data_wrangling_case.R` file. You can write all of your code in that file from within RStudio, then save your changes, commit, and push those changes to GitHub before the deadline to submit.

Have fun!

Assignment instructions

Intro / Setup

Follow the download and import instructions in the beginning section ("Intro / Setup") in the `data_wrangling_case.R` file. This will guide you through downloading, importing, and cleaning the column names of the data we'll use for this assignment. (Download here, but follow the import and cleaning instructions in the R script file.)

We're now going to move to the actual exploration of the data. From now on, I'll count on you to use the common data manipulation vocabulary from the `tidyverse` to wrangle the data into the required view(s) in order to answer the questions. You'll be using the dplyr functions like `select()`, `rename()`, `filter()`, `mutate()`, `arrange()`, etc. You should **save the manipulated tibble to a new variable when told to do so**, and make sure to use the variable names exactly as they are requested. (This will ensure that you get credit when we grade your code.)

Sound good? Let's roll.

Filter, Arrange, Mutate, Rename

First select just the carrier, `tail_num`, `actual_elapsed_time` columns. No need to save the result.

Now select the carrier, `tail_num` and `dep_delay` columns, as well as any column with "taxi" in the column name.

> *Partial Answer:*

```
1 AA      N576AA           0       7       13
```

1. Save the result to `selection_with_taxi`.

Now show the same columns, but only for Delta (DL) flights with delayed (i.e., positive) departures.

> *Partial Answer:*

```
1 DL      N760NC          71      11       14
```

2. Save the result in a new tibble called `delta_delayed`.

Now copy and alter the code you wrote above to answer the following: Find the flight with the largest delay. How many minutes was it delayed?

> *You probably just changed the filter order (which is fine), but now to help us actually grade this question, use a dplyr function to limit the result to just the first row in that filtered/arranged set. (Don't just filter to the specific tail number...you'll need to go find a slice function for this.) (Answer: 730)*

3. Save the resulting tibble (which should have just one row and 5 columns) to a new variable called `longest_delay`

Displaying only the tail number, origin airport code, and actual elapsed time columns, find the American Airlines (AA) flight that had the shortest actual elapsed time. *(Answer: N568AA)*

4. Without further filtering the data, save the result in a new tibble called `american_fast_flights`

Select the `distance` and `time_in_air` columns, then create a new column called `speed` that calculates the average airspeed (in mph) of each flight using the following formula: `speed = distance / time_in_air * 60`.

5. After ensuring that the math is working, ensure that the new variable is saved to the existing `flights_clean` tibble.

> *How many columns does the `flights_clean` tibble have now? If your answer is 3, then what did you do wrong in the previous steps? The tibble should have 22 columns (21 original + the new speed column). If you made a mistake, you'll need to go re-create the `flights_clean` tibble from your janitor operation above, then run the correct mutate() command to get to the right number of columns. Go ahead; I'll wait.*

All good? Now find the top speed of all flights *(Answer: 764)*. Limit that set to just the first row.

6. Save the result to a new tibble (with one row and the `speed` column) called `top_speed`

How many distinct carriers are represented in the data? There are a few different ways you can answer this, but your end result should be a tibble with a row count equal to the number of distinct carriers. *(Answer: 15)*

7. Save the resulting tibble to a variable called `carriers_distinct`

Let's rename a few columns just to prove we can. Change the following to the name to which the arrow is pointing:

- `month` --> `month_number`
- `diverted` --> `was_diverted`
- `distance` --> `distance_in_miles`

> *What do you notice about each of the renamings I listed above? Why would someone make changes like these?*

Now let's move those columns to the "front" (left-most) position in the column order. Answer this one using two methods:

1. Use `select()` with the `everything()` helper, saved as `renamed_1`
2. Use `relocate()` function, saved as `renamed_2`

> *(Note: I know you could manually include all column names in my requested order in a simple select call to get to the same end results, but do yourself a favor and figure out how to do it programatically. Answer: Both results should be identical (22 columns) and have the following first 4 columns:* `month_number`, `was_diverted`, `distance_in_miles`, `year`*)*

8. (Make sure the above two are saved as `renamed_1` and `renamed_2`, respectively)

How many flights from any of this list of carriers arrived more than two hours late, but didn't leave late?

- Carriers: AA, AS, CO, DL, F9, FL, MQ, OO, WN, XE, YV

> *Hint: if you filter to those carriers by including them in a huge string of 'or' logic, I'll be very disappointed. :) Do that part of the logic with a single, simple logical operation. Answer: 5*

9. Save the results to a tibble called `miracle_departures`

Grouping, Summarizing, Counting

Let's do some grouping and summarizing practice. We've got lots of interesting categories in our data. Let's look at a few to see if anything interesting jumps out.

Let's start by just getting the average count of flights per day of the week. Use a `group_by()` and then `summarize()` to get a count of flights (i.e., dataset rows) per day of the week. The count column should be called `flight_count`.

> *Hint: within a summarize, you can use `n()` to get the row count for each group. (Answer: There are 34,902 Wednesday flights in the dataset.)*

10. Save the resulting tibble as `flights_per_day`.

Shortcut time! The way you just did a row count by group is a lot of typing. It's such a common data exploration activity that a convenient shortcut has been added to the Tidyverse. Figure out how to replicate the answer from above using the `count()` function. *(Note that the actual count column is named `n` automatically; there is no need to rename it to exactly match the `flight_count` column from above.)*

11. Save your result to `flights_per_day_2`.

Now let's look at the average flight duration (time in air) for the entire dataset. Use a `summarize()` to add a column called `mean_tia`.

> *Hint: When summarize is performed without grouping the data, the function operates on the whole dataset. Answer: 108, and `overall_time_in_air` should be a tibble with just 1 row and 1 column.*

12. Save the resulting tibble to a new variable called `overall_time_in_air`.

Now let's see which airline has the lowest average departure delays. Using a `group_by()` function, calculate the mean delay, and order the results from lowest to highest average.

> *Partial Answer:*

```
1 YV           1.54
```

13. Save the result to `airline_delays`.

What is the most delayed flight to each location? Keep only the destination, departure delay, and departure time columns.

> *Partial Answer:*

```
 6 ASE          269           1336
 7 ATL          730           2310
```

14. Saving the resulting tibble to `most_delayed`

Now let's see the top 5 airports that RECEIVE the most flights. For those 5, what are the average arrival and departure delays? How many unique tail numbers flew into each? Use whatever functions you need, save the answers to columns named whatever you like. *(Answer (which should have 5 rows and 5 columns): DAL, 9820, 806, 8.53, 11.3)*

> *Partial Answer (answer should have 5 rows and 5 columns):*

```
1 DAL             9820             806          8.53         11.3
```

15. Save the resulting tibble to `most_flights`

What are the average `taxi_in` time and speed on each day of the week for the top 3 most delayed arrivals for each carrier?

> *Hint: You'll need to do two `group_by()` calls, which means you'll want to have an `ungroup()` in between. This is a tough one, so I'll give you the full answer:*

```
# A tibble: 7 × 3
  day_of_week ave_taxi_in ave_speed
        <dbl>       <dbl>     <dbl>
1           1        8         427.
2           2        7.67      448.
3           3        7.86      475.
4           4        6.67      417.
5           5       12.5       411.
6           6        7.67      448.
7           7        5.5       446.
```

16. Save the resulting tibble to `group_ungroup_group`

During the first six months of the year, find the time in air for the longest delayed departure from each carrier at each destination airport. Keep only the carrier, destination, time in air, and dep delay columns. *Answer tibble should have 235 or 236 rows, depending on how you get there.*

> *Partial Answer:*

```
 1 CO       ABQ            117       164
```

17. Save the result in `long_delays_each`.

Binding, Joining

Let's do some practice with joining different datasets together. For this, we'll assume we want to do some data verification of the `distance` column in our flights dataset using a manual calculation of the actual distance between two airports, given their lat/long positions.

IMPORTANT: This is one of the few instances where the approach discussed in the second edition of our "R for Data Science" book is just a bit ahead of the currently released version of our dplyr library. So you'll see them mention a special new `join_by()` function which will give you an error if you try to use it. If you need to see some examples or discussion of joins to support the exercises in this section, please consult the older textbook page on the topic.

There are two files containing airport_lat_long data, zipped together and available for download here.

Download, unzip, and then place both files in your `/data` folder.

18. Read them into `lat_long_1` and `lat_long_2` .

Both of these files are obviously small, and as such don't really need to be broken into two files, but I've created them manually for the purposes of this exercise. It is very common for real datasets to be broken up in to tens or even hundreds of different .csv files when they are large. Let's combine the two lat_long files into one. We don't use binding operations nearly as much as joining operations, but it's good to know about them for cases where you want to "stack" data together rather than join it based on a join key.

The easiest way to do this is with the `bind_rows()` function, which is equivalent to a "UNION ALL" operation in a standard SQL query. Use `bind_rows()` to stack both lat_long tibble together.

> *Hint: bind_rows will help you not make assembly mistakes by only truly stacking the data when the column names match. Otherwise, it'll keep all columns, which usually results in a bunch of empty values for either half of the rows in the result.*

```
A tibble: 7,827 × 3
airport_code airport_lat airport_long
<chr>              <dbl>        <dbl>
FAQ                -14.2       -169.
Z08                -14.2       -170.
PPG                -14.3       -171.
```

19. Saving the result as `lat_long` .

Let's join in that lat_long data for our origin airport codes in the flights dataset. Use a left_join() to ensure that the result has the same number of rows as `flights_clean` . After joining, select the carrier, origin, dest, and all columns starting with "airport".

> *Hint: you'll need to specify the join keys (using the `by` parameter) since the two datasets do not have compatible column names. (Just another reminder to make sure you've read the "**IMPORTANT**" note above about using the older textbook discussion of joins if you need help here.)*

> *Answer: (Top three rows shown)*

```
A tibble: 227,496 × 5
carrier origin dest  airport_lat airport_long
<chr>   <chr>  <chr>       <dbl>        <dbl>
```

```
AA      IAH    DFW          30.0        -95.3
AA      IAH    DFW          30.0        -95.3
AA      IAH    DFW          30.0        -95.3
```

20. Save the result to `flights_origin_latlong`

That brought in lat_long info for the origin airport, but let's also bring in the lat_long for the destination airports in the flights dataset. Starting again with the `flights_clean` data, do two joins so that we have lat_long for both the `origin` airport and the `dest` airport. Select just the carrier, origin, dest, and the FOUR new columns that result from the join.

> *Hint 1: Join operations can be added sequentially with pipes like all the other data manipulation verbs.*
> *Hint 2: Make sure you name each `lat_long` column appropriately so we know whether they are for the `origin` or `dest`. Try not to use the lame way of just using `rename()` after the fact. Look into either using a rename on-the-fly in each join statements or use the `suffix` parameter on the second join statement.*

> *Answer: (Top three rows shown)*

```
A tibble: 229,884 × 7
carrier origin dest  airport_lat_origin airport_long_origin airport_lat_dest
airport_long_dest
<chr>   <chr>  <chr>              <dbl>               <dbl>            <dbl>
<dbl>
AA      IAH    DFW                 30.0               -95.3             32.9
-97.0
AA      IAH    DFW                 30.0               -95.3             32.9
-97.0
AA      IAH    DFW                 30.0               -95.3             32.9
-97.0
```

21. Save the resulting tibble to `flights_latlong`

When we do a left join like this, I like to make sure that nothing unexpected happened with the key matching and the resulting row count. In the current scenario, we wanted to bring in lat/long columns for the origin and destination airport codes. We should expect that the result tibble would have the same row count as the source `flights_clean`. Is this the case? Which of the join operations has created an issue?

> *Hint: Compare the row counts for `flights_origin_latlong` and `flights_latlong`. No answer to save here.*

So we know that the result has more rows than the original flights dataset. Do some sleuthing to see if you can figure out what happened.

> *Answer: There is a right answer, but you might get to it in various ways. So just save the tibble that demonstrates that you figured it out.*

22. Save the tibble that helped you figure out the issue to `join_issue_1`.

Normally, we'd make a change to ensure that we fixed the issue you just identified, but we're going to move on so that the final exercise has a consistent starting point for everyone. Finally, let's just make sure that we're able to bring in all of the airport data. By definition, a left join will keep all rows on the left hand side of

the join and join in data where it can, leaving missing data in the result if any of the left-side keys can't be found on the right side.

So let's check to make sure both joins worked. First filter to find and rows where `airport_lat_origin` is NA. (There shouldn't be any.) Then filter to find any rows where `airport_lat_dest` is NA. (No need to save either.)

Uh oh! Looks like something's up with the second join with the destination airports. Using the `flights_latlong` data, figure out why the `airport_lat_dest` and `airport_long_dest` have missing values.

> *Answer: no single right answer. Just save something that demonstrates that you figured it out.*

23. Save the tibble that helped you figure it out to the variable `join_issue_2`

We always want to be defensive when it comes to joins, making sure we're fully aware of what is dropping out or coming up missing along the way. For this reason, I tend to prefer left joins and manual checking of the results like we just did. Depending on your scenario, though, it might be fully expected that your two datasets won't match up perfectly and that you're really only interested in the stuff that does match. This is the perfect scenario for the `inner_join()` operation, just like in any SQL query. Let's fix the issues with our joined data. Specifically, let's first make sure that the `lat_long` dataset is properly, uniquely identified (`join_issue_1`), and then let's just fully IGNORE the missing data that we identified (`join_issue_2`) by using an inner_join for the destination airport).

> *Answer: (Top three rows shown)*

```
A tibble: 223,720 × 7
carrier origin dest  airport_lat_origin airport_long_origin airport_lat_dest
airport_long_dest
<chr>   <chr>  <chr>              <dbl>               <dbl>            <dbl>
<dbl>
AA      IAH    DFW                 30.0               -95.3             32.9
-97.0
AA      IAH    DFW                 30.0               -95.3             32.9
-97.0
AA      IAH    DFW                 30.0               -95.3             32.9
-97.0
```

24. Save the joined result to `flights_latlong_inner`.

Finalize, Commit, Push

BEFORE YOU COMMIT AND PUSH YOUR SUBMISSION

Please make sure you to do the following before you submit this assignment:

1. Ensure that you have just one `setwd()` command near the beginning of the script and that all other calls to the filesystem are relative to that directory.
2. Prior to your final commit / push, please COMMENT OUT that one `setwd()` command so that your code doesn't break our grading procedures.
3. The list of objects below is what I have in memory after writing this case assignment. Any of those objects are fair game for the grading procedure to evaluate. You may want to clear your R session

(Session -> Restart R) and then check the list against what you have in memory after running your script from the beginning.

Hint: you can see what's in your environment by executing the following code:

```
ls() %>% tibble() %>% print(n=30)
   1   airline_delays
   2   american_fast_flights
   3   carriers_distinct
   4   delta_delayed
   5   flights_clean
   6   flights_latlong
   7   flights_latlong_inner
   8   flights_origin_latlong
   9   flights_per_day
   10  flights_per_day_2
   11  flights_raw
   12  group_ungroup_group
   13  join_issue_1
   14  join_issue_2
   15  lat_long
   16  lat_long_1
   17  lat_long_2
   18  long_delays_each
   19  longest_delay
   20  miracle_departures
   21  most_delayed
   22  most_flights
   23  renamed_1
   24  renamed_2
   25  overall_time_in_air
   26  selection_with_taxi
   27  top_speed
```