SI618 Project I
Iain Graham

1. Motivation

For this project I have chosen to focus on uncovering potential relationships that exist between receiving parking tickets and the weather. Having received many parking violations, I have always been curious if there is a way to lower my chances of getting a ticket. Uncovering patterns in ticket writing could potentially save people time and money by avoiding costly citations. In an effort to uncover a possible relationship I decided to try and answer three main questions:

1. What is the specific ticket breakdown for different types of weather?
2. What does the average weather look like for each type of traffic ticket issued?
3. If you get a ticket, what is the probability that you will get it during a certain weather type?

2. Data Sources

City of Chicago Parking Ticket Data: (https://www.propublica.org/datastore/dataset/chicago-parking-ticket-data) The first dataset I looked at was parking citation data from the city of Chicago. I downloaded this data from ProPublica.org. The downloaded data was a CSV file with 28,272,580 total rows of data starting from November 1, 2003 and ending on May 3, 2018. Although I had data for 15 years back, I only chose to focus on the year 2017 for this project which gave me roughly 2 million rows to work with. The variables that I was interested in were the issue date and violation description from this dataset.

Weather dataset: (https://www.ncdc.noaa.gov/cdo-web/search)
The second dataset that I used was weather data from the National Oceanic and Atmospheric Administration. The NOAA website allowed me to enter specific search parameters that let me access the specific data that I was interested in. I chose historical weather data for all of 2017 collected from the Chicago O'Hare weather station. The dataset that I got back from my search was 365 rows and 33 columns as a CSV file, however I wouldn't be using all of the columns for this project. The variables from the weather data that I was interested in were date, precipitation, snowfall, and average temperature.

3. Data Manipulation Methods

Fortunately, both of my datasets were already clean and did not contain any missing values that I needed to account for. My columns of interest were also in the correct format and ready to be used without any sort of conversion or manipulation. Whereas my weather dataset only contained data from 2017, my parking ticket dataset contained data since 2003, so I needed to extract only the data from 2017 so I could make my join.

Manipulation Step 1: Extracting Columns of Interest

Once I had my data loaded into Spark, I made a sparkSQL query to return my parking ticket data from 2017. The dates in this dataset were listed in *YYYY/MM/DD HH:MM:SS* format, so to make this query, I used a *like* statement that returned all of the rows that started with 2017 in the date column. In addition to selecting the date, I also returned the violation description. Now that I had the relevant data from the ticket dataset, I wrote another simple query to extract the date, precipitation, average temperature, and snowfall from the weather dataset.

Manipulation Step 2: Making the Join

Once I had created my two new tables I was ready to make my join by using the date column that was shared between both datasets. Recall that the date column in the tickets dataset was listed in *YYYY/MM/DD HH:MM:SS* format, which was different than the weather dataset. Since the weather table only had the date and not time, I joined the tables together on a substring of the first part of the date column in the tickets table (*YYYY/MM/DD*) on the date column from the weather table that was given in *YYYY/MM/DD* format. The resulting joined table had date, violation description, average temperature, precipitation, and snowfall columns.

Manipulation Step 3: Breaking my Data Up into Weather Types

For multiple parts of this project I wanted to look at the ticketing behavior during specific types of weather. To achieve this, I needed to query my joined table to return the different weather categories I was interested in. I used sparkSQL again and queried my joined table using *where* statements to get all of my weather categories. The weather types I was interested in are outlined below:

Light rain: where precipitation > 0 and < 0.1 inches
Moderate rain: where precipitation >= 0.1 and < 0.3
Heavy rain: where precipitation > 0.29
No snow: where snow = 0
Moderate snow: where snow > 0.0 and < 2.1
Heavy snow: where snow > 2.0
Freezing temperature: where average temperature <= 32
Cold temperature: where average temperature > 32 and <=50
Moderate temperature: where average temperature >=51 and <=69
Warm temperature: where average temperature >= 7

Manipulation Source Code:

```
>>> q1 = sqlContext.sql("select DATE, PRCP, SNOW, TAVG, WSF5 from weather")
```

>>> selecting the columns of interest from the weather dataset

```
>>> q2 = sqlContext.sql("select issue_date, community_area_name, violation_description from tickets where issue_date like '2017%'")
```

>>> selecting the columns of interest and filtering the date from the tickets dataset

```
>>> q3 = sqlContext.sql("select weather.DATE, tickets.violation_description, tickets.community_area_name, weather.PRCP, weather.SNOW, weather.TAVG,
weather.WSF5 from tickets inner join weather on substring(tickets.issue_date,1,10)=weather.DATE")
```

>>> making my join on the date using a substring from the tickets query

```
>>> q3.show()
+----------+--------------------+--------------------+----+----+----+----+
|      DATE|violation_description|community_area_name|PRCP|SNOW|TAVG|WSF5|
+----------+--------------------+--------------------+----+----+----+----+
|2017-01-01| NO STANDING/PARKI...|          LAKE VIEW|0.00| 0.0|  27|19.0|
|2017-01-01| DOUBLE PARKING OR...|    NEAR NORTH SIDE|0.00| 0.0|  27|19.0|
|2017-01-01| NO CITY STICKER V...| WASHINGTON HEIGHTS|0.00| 0.0|  27|19.0|
|2017-01-01| NO CITY STICKER V...| WASHINGTON HEIGHTS|0.00| 0.0|  27|19.0|
|2017-01-01| RESIDENTIAL PERMI...|                null|0.00| 0.0|  27|19.0|
|2017-01-01| NO STANDING/PARKI...|    NEAR NORTH SIDE|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|       NORTH CENTER|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|    LOWER WEST SIDE|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|    LOWER WEST SIDE|0.00| 0.0|  27|19.0|
|2017-01-01|    3-7 AM SNOW ROUTE|                LOOP|0.00| 0.0|  27|19.0|
|2017-01-01| DOUBLE PARKING OR...|    NEAR NORTH SIDE|0.00| 0.0|  27|19.0|
|2017-01-01| DOUBLE PARKING OR...|    NEAR NORTH SIDE|0.00| 0.0|  27|19.0|
|2017-01-01|    3-7 AM SNOW ROUTE|       PORTAGE PARK|0.00| 0.0|  27|19.0|
|2017-01-01|    3-7 AM SNOW ROUTE|          WEST TOWN|0.00| 0.0|  27|19.0|
|2017-01-01|    3-7 AM SNOW ROUTE|          WEST TOWN|0.00| 0.0|  27|19.0|
|2017-01-01|    3-7 AM SNOW ROUTE|          WEST TOWN|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|     GARFIELD RIDGE|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|     GARFIELD RIDGE|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|     GARFIELD RIDGE|0.00| 0.0|  27|19.0|
|2017-01-01| EXPIRED PLATES OR...|     GARFIELD RIDGE|0.00| 0.0|  27|19.0|
+----------+--------------------+--------------------+----+----+----+----+
only showing top 20 rows
```

>>> the final joined table that I use for my analysis

```
>>> light_rain = sqlContext.sql("select * from main_table where PRCP > 0.0 and PRCP < 0.1 and SNOW = 0.0")
>>> mod_rain = sqlContext.sql("select * from main_table where PRCP >= 0.1 and PRCP < 0.3 and SNOW = 0.0")
>>> heavy_rain = sqlContext.sql("select * from main_table where PRCP > 0.29 and SNOW = 0.0")
>>> no_snow = sqlContext.sql("select * from main_table where SNOW = 0.0")
>>> mod_snow = sqlContext.sql("select * from main_table where SNOW < 2.1")
>>> heavy_snow = sqlContext.sql("select * from main_table where SNOW > 2.0")
>>> low_temp = sqlContext.sql("select * from main_table where TAVG <= 32")
>>> mid1_temp = sqlContext.sql("select * from main_table where TAVG > 32 and TAVG <=50")
>>> mid2_temp = sqlContext.sql("select * from main_table where TAVG >=51 and TAVG <=69")
>>> high_temp = sqlContext.sql("select * from main_table where TAVG >= 70")
```

>>> filtering to get my weather types I'm interested in

4. Analysis and Visualization

Question I: What is the specific ticket breakdown for different types of weather?

I asked this question to get more comfortable with the data and uncover and initial observations that stick out as unusual.

For this part of the project, I want to get a total count of each of the violations for each of my 10 different weather categories. For this task, I will take each weather type data frame, and add up how many times the different traffic violations occur. I used mrjob to perform this task. Since I am only interested in the *violation_description* column in my tables, my first step was to isolate this column and convert it to an *rdd*. After creating my *rdd* object I applied a regular expression to match all the words. To combine the *violation_description* entries into one string rather than multiple words, I used the *.join* method in a lambda function so that I won't get any errors when I run my reducer. I then add an iterator to each of my datapoints followed by my *reduceByKey*. I

then decide to sort my totaled list in descending order so I can see which ticket types are the most common across the different weather categories.

Figure 1: Total tickets administered across the different weather categories for the most ticketed violations

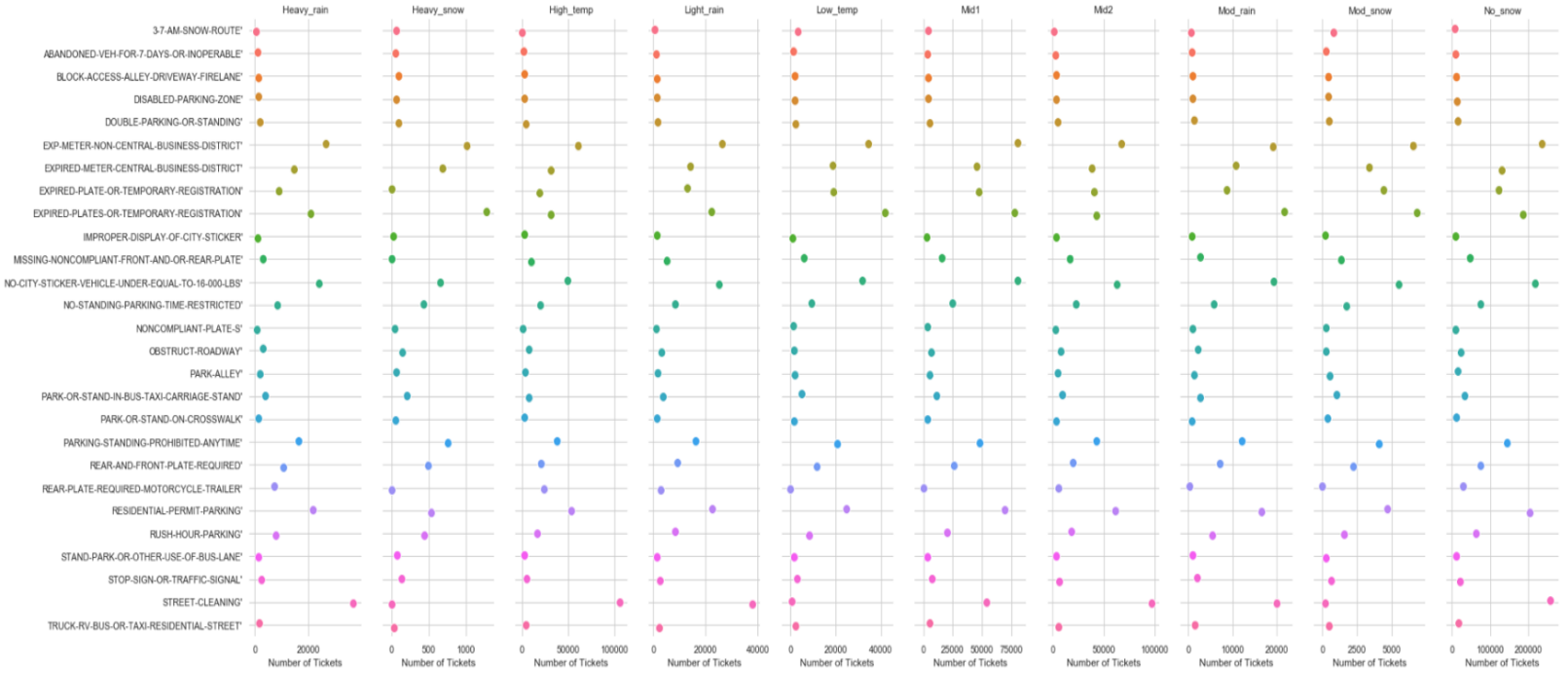

Figure 2: Street cleaning ticket breakdown to see size differences

After graphing the results of our mrjob task, we can notice the consistencies across different weather types in terms of proportions of tickets given. Although the number of tickets vary drastically across different weather types, the same ticket types are near the top of all the different weather categories. The difference in total number of tickets issued can be explained by the frequency of the weather rather than interpreting it as less ticketing during a certain weather type. Figure 2 does a good job showing the breakdown of the distribution of street cleaning tickets across the weather types that these tickets rank high in. Looking back at figure 1 we can see that expired meters, no city sticker, and street cleaning violations are consistently high across the weather types. Although street cleaning tickets seem to be one of the most ticketed

violations, in heavy snow, moderate snow, and low temperatures the ticketing rate drops dramatically leading me to think there must not be street cleaning in Chicago when these conditions are present. The results of this question lead me to believe that people are still issuing the same types of tickets no matter the weather conditions.

Question II: What does the average weather look like for each type of traffic ticket issued?

Because the results of the first question don't do a good job at differentiating the gap between frequency of weather compared to administered tickets, I want to now look at the average rainfall, snowfall, and temperature for each of the ticket types. This will give us a good idea of the types of conditions certain tickets are more likely to be given in. For this question I used sparkSQL to select the entire precipitation, snowfall, and average temperature columns from my main table. My resulting tables had two columns, date and a type of weather. To implement another map-reduce task I mapped my tables into an *rdd* of tuples. After I created my tuples I used the *.mapValues* method to convert the weather to a float type for the division in the next step. I then set an iterator and used the *.reduceByKey* method which resulted in something that looks like *(12-31-2017, (25.6, 250))* where 25.6 represents the sum of the weather type (e.g. total rainfall), and 250 represents the number of rows we added together. From here, I divided *x[1][0] / x[1][1]* to get the average rainfall, snowfall, and temperature for 2017.

Figure 3: Average precipitation, snowfall, and temperature for each unique ticket type

Looking at the graphic resulting from our second question we can see that the average precipitation across the different ticket types is relatively consistent with the exception of a few spikes. We notice a distinct difference as we move to the average snow column. We can say that more than four front mounted lamp and snow route tickets are much more likely to occur in snow. Moreover, there are not many other tickets that are likely to be issued during snow. As we look at the average temperature column, we notice a consistency around 55 degrees. There are two ticket types that occur with an average temperature of over 80 degrees – improper running board lamps and safety chains required. This breakdown helps us understand the likelihood of getting a ticket in a certain weather a little more clearly than in question 1 because we can see the specific weather conditions that make a ticket more likely.

Question III: If you get a ticket, what is the probability that you will get it during a certain weather type?

To answer this question, I relied on sparkSQL. To start solving this problem I started by creating my 10 different categories of weather that I outlined in the manipulation section. Because I am trying to find probability, I need to calculate the total number of tickets, so I run another sparkSQL query to find the total count of all tickets given in 2017. I then run another sparkSQL query to calculate the count of tickets given for each different weather type. As a resukt I can divide the total count by this number and get the probability of getting a ticket during a certain weather type. Before I complete the division, I first convert the two numbers to float types using the *float()* function so I get back a decimal. Now that I have found the probability, I have decided to include average tickets given per day given a weather type. To do this, I take the count of the weather type specific tickets and need to divide it by the number of days that weather type existed. To find this number, I run another sparkSQL query to select the *count(distinct date)* from my table. I do not need to worry about converting these to float types since I want this to return an integer.

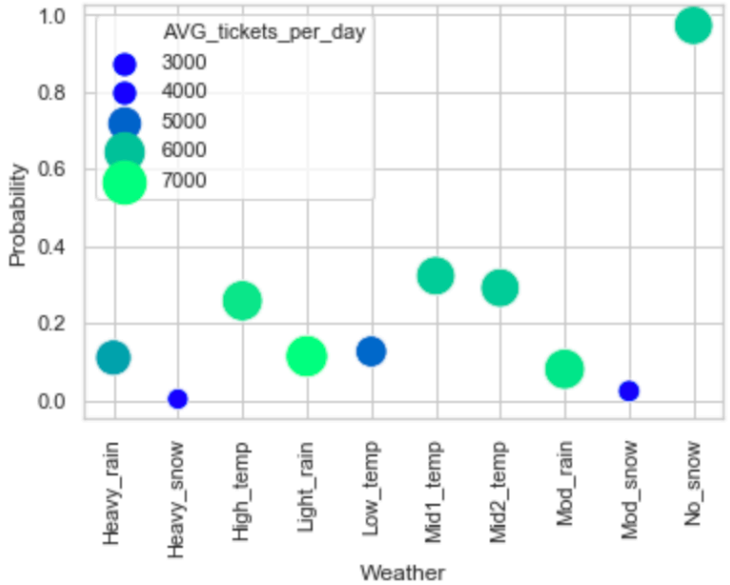

Figure 4: This graph explains the probability if you get a ticket, that you will get the ticket on a specific weather type. Additionally, the size of the plot-markers represents the average number of tickets administered for a certain weather type.

This graph is interesting for a couple of reasons. First, we can see that on average, days with light rain have the most tickets administered. We can also see that any type of cold or rainy weather leads to a low probability of you getting a ticket during that weather. Additionally, low temperatures seem to mean low probability as well as low average ticket counts. I noticed that the probabilities shown on the graph could be similar to the fraction of days that a certain weather exists, so I plotted it.

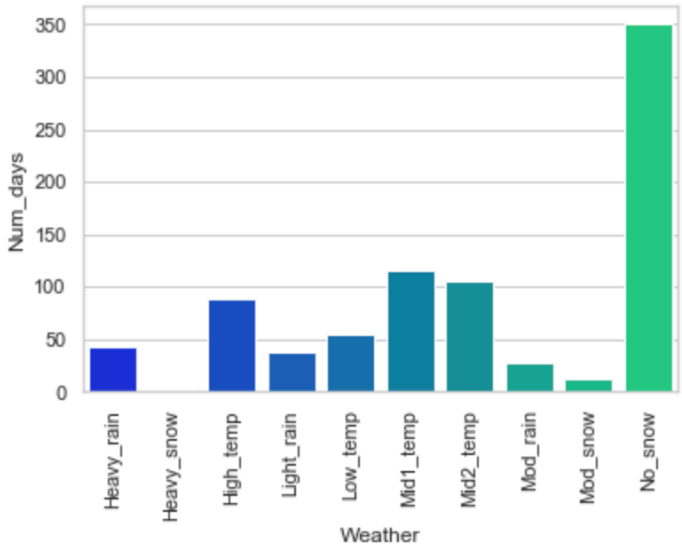

Figure 5: This figure shows the number of days out of the year that a specific weather type exists

As you can see, the probability is near identical to the distribution in the second graph. This means that if you get a ticket, the probability is actually measuring the chance of a weather type occurring, and not measuring whether or not you are more likely to get a ticket. To try and overcome this, I looked at the difference of the two graphs.

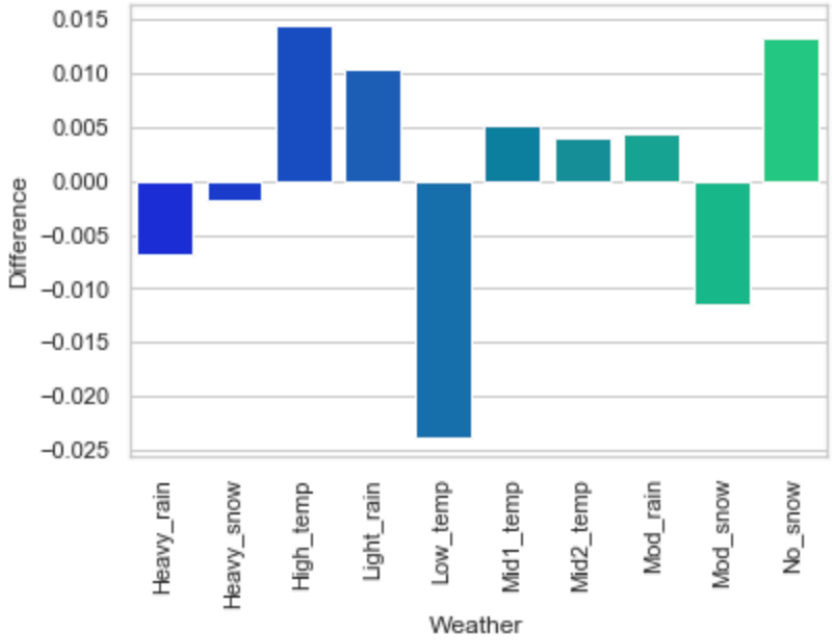

Figure 6: This graph shows us the difference in probability from figure 4 and the fraction of weather type occurring from figure 5.

After interpreting this graph, we can see that heavy rain, low temperature, and moderate snow all have negative values. This means that the probability from figure 4 is greater than that of figure 5. If we think about the fraction of days that a certain weather occurs as the baseline probability of getting a ticket during a certain weather type, we can compare the results from figure 4 against this baseline. For example, noticing that the low temp difference is the negative tells us that you are roughly 0.024 less times likely to get a ticket when the temp is low against the base line. Conversely, getting a ticket in a high temp is actually 0.015 more times likely. With these results we have, we should not be trying to detect the probability of getting a ticket during a specific type of weather, but rather how much more likely are we to get a ticket during a certain type of weather compared to the baseline.

5. Conclusion

After diving deep into these two datasets, I learned that weather may not have as large as an effect that I initially thought it would. Answering my first question showed me that the most commonly given tickets are given out in similar proportions across different weather types. Moreover, I learned that the average weather type across the three different types of varies. We saw a lot of consistency looking at rain, but noticed some possible outliers looking at snow and temperature. Attempting to find the probability of getting a ticket during a specific weather type resulted in something similar to the fraction of that the weather type occurred in 2017. To overcome this, I decided to compare the probability I found to the fraction of times a certain weather occurred. I think this adjustment gave me a better idea whether or not the probability I found using the data could explain any change in likelihood of parking tickets.

6. Challenges

One of the challenges I encountered during this project came to me in the second part. As I was trying to divide to find the averages in my lambda function. I thought of changing the type of the numbers in the lambda function, but it was still throwing me an error even after trying numerous combinations of converting to float and int. After googling and reading up about converting types on Stack Overflow, I tried to simplify the problem by converting to a float before the main lambda function. To my surprise this fix ended up working!

I also ran into some trouble as I was trying to write some of my outputs to file. Although this was frustrating, it was a quick fix for me after I realized I needed to convert to an rdd. After converting the type I was easily able to write all my files to csv files.

Another problem I encountered was not being able to initially run my *reduceByKey* in part 1. I kept getting errors when I would run it and wasn't sure what was happening. After looking at the example we did in class with word count, I got the idea to try and make the ticket descriptions one long string because the only difference that I noticed between my project and the homework was that the homework was reducing a single word key, where I was trying to reduce a phrase. After replacing the spaces with "-" my reducer ran and gave me the desired results.