# ENSF_592_Final_Report

August 12, 2020

Authors: Junwoo Choi, Adam D'Andrea, Shiyu (Ivan) Zhou

## Contents

# 1 Interpretation

## 1.1 Grid data

The average speed limit, average traffic volume, traffic cameras, traffic signals, traffic signs and traffic incidents were each calculated for each section of a 10x10 grid accordingly. Results were combined into 1 dataframe for visualization analysis. Incident/volume ratio was plotted against average speed limit, traffic cameras, traffic signals and traffic signs respectively to evaluate possible correlations. (Incidents were normalized against volume as naturally higher volumes would lead to higher incident counts).

Incident/volume ratio is indicated by the red line in figures 2.21, 2.31, 2.41 and 2.51. This incident/volume ratio peaks near grids 35, 45, 55 and 65. These grids correspond to the downtown area, with grid 55 (the highest incident/volume ratio) being at the downtown centre of Calgary.

Bar graphs of traffic signals, traffic signs and traffic cameras, shown in cyan color (figures 2.2.1, 2.3.1, 2.4.1 and 2.5.1), followed a similar pattern as the red line representing incident/volume ratio. Grids with greater number of signals, signs and cameras resulted in higher incident/volume ratio. An overall strong positive correlation was shown between incident frequency and the number of traffic signals, signs and cameras. On the other hand, a moderate negative correlation was spotted between incident/volume ratio and average speed limit (figure 2.2.1). Grids with higher average speed limit, such as grid 60, resulted in lower incident/volume ratio. Whereas grids with significantly lower average speed limit, such as grids near downtown area (35, 45, 55, 65), resulted in much higher incident/volume ratio.

Within figure 2.3.2, a bar plot comparing the number of cameras in Calgary that were within ~ a 10m radius of an incident. This plot shows that about 77% of cameras were close to an incident, showing that cameras are at least being placed in good locations to catch incidents. However, when comparing to figure 2.3.3, only 1.4% of incidents occur near cameras. This shows that there are a lot more incidents than cameras in Calgary. On the other hand, 11% of incidents occurred within 10m of a signal. Table 2.4.2 and figure 2.4.5 indicates that ½ signals (intersections where there is only a traffic signal on 2/4 of the streets) have a 23.8 incident to signal count percent. This tells us that ½ signals are more likely to have an incident occur nearby them. However, because there are less ½ signals (21) than other signals (1010 and 257 for 'T-intersection' and 'Overhead Flasher' signals, respectively) this conclusion is not as concrete. It is also interesting that there is a 41.3% incident to signal count in the South Calgary quadrant (table 2.4.3 and figure 2.4.6), and in general, there are higher incident counts in southern Calgary. Part of this may be because downtown is included in South Calgary, and that is where the highest volume of cars is located, and everyone drives to/from downtown every day for work. A bubble plot of incident to signal count can be seen in figure 2.4.7, where each bubble is placed within its respective quadrant in Calgary. Figure 2.4.8 is a visual line plot that show south Calgary has having the highest incident count as well.

As for pedestrian button signals, table 2.4.4 shows an interesting statistic that there is a 20% higher chance of having an incident at a signal with no pedestrian button. The city usually places pedestrian buttons where pedestrians are common, so perhaps cars at these intersections are not often expecting to have a pedestrian crossing the street. Another explanation might be that most intersections without a pedestrian button are "uncontrolled" intersections, where cars

need to regulate themselves and organize themselves when they can turn or drive through an intersection. Whereas an intersection with lights at each road will automatically regulate the traffic for drivers. This relates to possibly why ½ signals are the most common to have incidents, as they require drivers themselves to decide when to drive through the traffic or 'turn into' the traffic.

## 1.2 Weather data

Extra Weather data tables and figures can be found in section 2.6. Overall, these figures indicate that incidents are more common in January and February (figure 2.6.3), and that these two months are on average the coldest months in 2018 at -10oC on average. Figure 2.6.4 agrees with this conclusion, showing that on average 33 incidents occur per day during days of -20oC and -30oC, and any temperature greater than this has little effect on daily incident rates. Figure 2.6.7 agrees with the conclusions so far, showing that the highest daily incidents occur in January and February, but it also shows how visibility (>~3km) had a minor effect on incident rates, and is perhaps overshadowed by temperature. Visibility does however play a role in the daily incident count as seen I figure 2.6.8, where any visibility less than 3km results in an increase in daily incidents.

Of all this data so far, the most important seems to be the time of day. Figure 2.6.9 show the total incidents vs. the time of day (on a 24 hour clock). It is obvious that most incidents occur at 8:00am and 5:00pm, which is exactly when rush hour occurs downtown. The signal data, shows that the highest incident to signal ratio is in downtown, the traffic volume and incident heat maps also show that most of the incidents and traffic volume is located downtown. Final speed data shows that speed is not a major factor in the frequency of incidents, this also correlates to downtown where cars are on average driving at 50-60km/hr, but they have to stop and go frequently at traffic signals and pay attention to when they can turn right onto another street at a red light. The traffic is very dense in downtown at 5:00pm and 8:00am, people are either sleepy or excited to get home, yet they need to be alert and stop/go at frequent intersections, this is a recipe for having incidents, especially in January and February when the weather is cold and snowy.

## 1.3 Conclusions

Based on the interpretations above as well as taking into the spearman coefficients here are the conclusions for each independent variable:

Cameras (spearman = 0.73): While cameras have a high positive correlation with incidents, this is mainly due to the fact that many incidents occur within downtown Calgary where the cameras are located. Taking a closer look into the location, only 1.4% of incidents occur near a camera (10m radius). (However, the placement of the cameras are good as 77% of them were able to capture an incident).

Signals (spearman = 0.92): Signals has the highest correlations with incidents. This makes sense as most signals are placed at intersections and "stop and go" locations where drivers must pay attention to their surroundings as well as potentially self-regulate.

Signs (spearman = 0.88): Signs share the same reasoning with signals.

Speed (spearman = -0.51): Interestingly, speed has a negative spearman coefficient which seems counter intuitive. Faster speeds generally mean less time to react, however, if we look at our average speed limit data to incident/volume ratio, we see that most accidents occur in the downtown area where the speed limit is much lower compared to highway speeds. However, one important thing to note is that this does not measure the severity of the incident (will be mentioned in the following section).

Visibility (spearman = -0.13): Overall, the visibility seems to have little (negative) to no correlation based on the spearman coefficient. However, if we bin the date, we can see that visibility has an impact on incidents with less than ~3km of visibility. After this amount, there seems to be very little correlation which makes sense as >5km of visibility is relatively good (which causes the low spearman value as most points are above this visibility).

Temperature (spearman = -0.02): Similarly with visibility, temperature seems to have little to no correlation based on the spearman coefficient. However, plotting average temperature versus month shows us that the coldest months (January and February) have the most incidents. Binned data also agrees with this (-20C to -30C having the most incidents). The cause of the low spearman coefficient is that most the data is lies above these low temperatures.

### 1.4 Limitations of our analysis/future improvement:

During our analysis, we made several assumptions which could limit the usefulness of our data. We will address these here for potential future improvement:

1. We treat all incidents as the same (not enough granularity):

A multi-car collision incident is treated the same as single car incident. Instead maybe some sort of index of severity would provide more insight. (More data would also be needed, as current data only tells us that there was an incident and not the type/severity). Along this note, a pedestrian incident is the same as a car on car incident.

Reducing accidents could be difficult (it's impossible to reduce all accidents to 0) but instead maybe a good compromise is to reduce severity instead (that's why we need more data for a severity index). A good outcome would be if most accidents were minor.

2. The 10x10 grid:

Our 10x10 grid, while useful so we are able to analyze data, is an arbitrary (and maybe even biased) grouping structure. Instead maybe an unsupervised learning/grouping algorithm to group attributes (such as location of incident, or even weather groups) could possibly provide more insight. Some popular choices are k-means (but we would have to choose amount of groups in this algorithm) or DBSCAN (which would find groups and number of groups for us) could be used to group data together.

3. Driver competency:

We don't know the competency of the drivers (i.e. they are all treated the same). Instead of traffic and weather data, another large factor in reducing incidents can be in educating the

driving population. This could also be the more cost efficient solution.

4. More data:

Finally (and the most obvious), using more data points for the analysis, comparing year to year changes could provide us with valuable insight, however was out of the scope for this project.

# 2 Figures

## 2.1 Spearman correlations table

### 2.1.1 Spearman correlations table

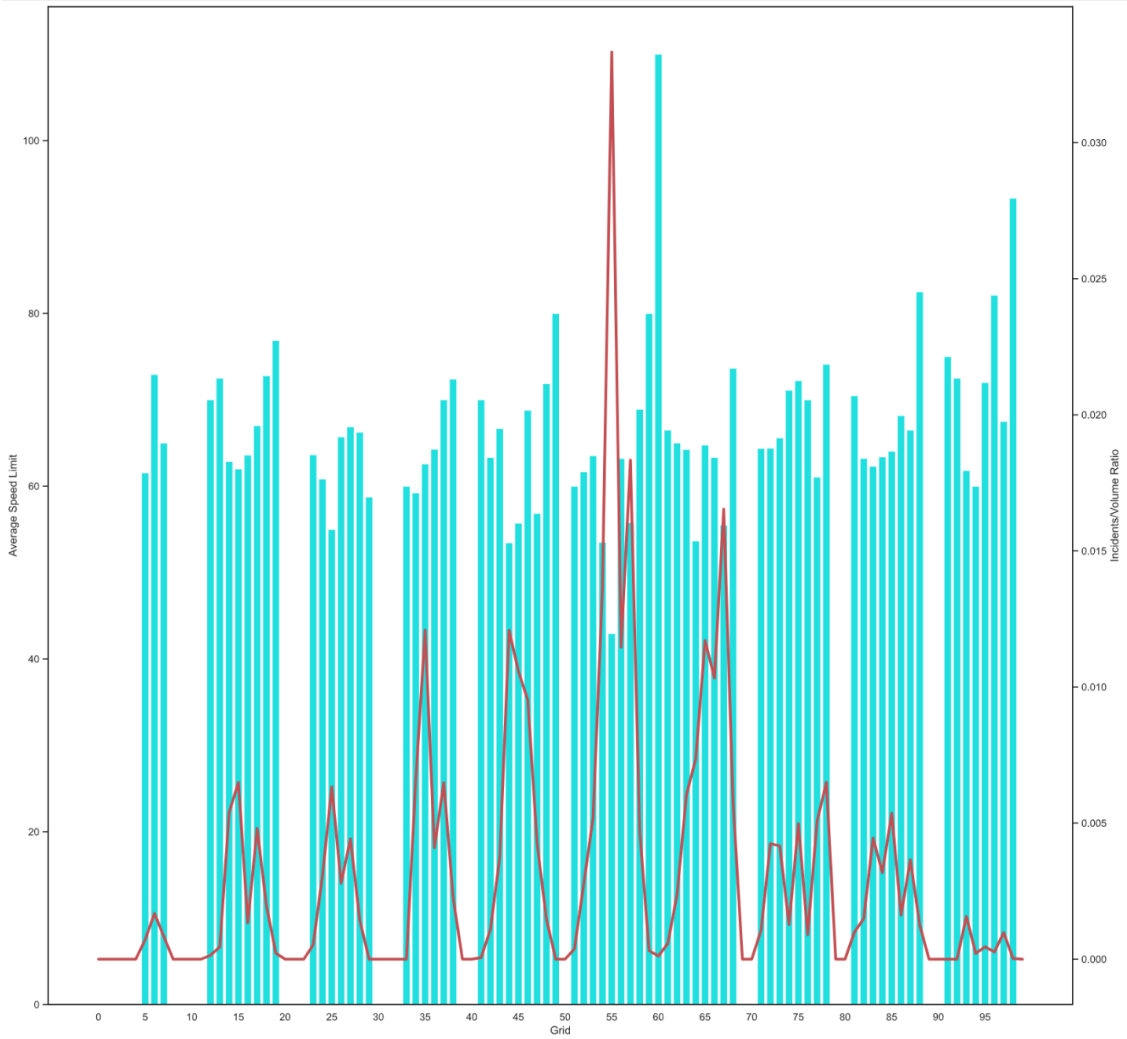
```
Grid data
Spearman correlation against incident/volume
Cameras: 0.730782361308677
Signals: 0.922076813655761
Signs: 0.8823897581792318
Speed: -0.5109218807848945

Weather data
Spearman correlation against average daily incidents
Visibility: -0.12976805363960087
Temperature: -0.02263124255039828
```

Spearman coefficients were calculated using the independent variable (cameras, signals, weather data etc.) rankings against the dependent variable (incident) rankings. The incidents were controlled for volume in the grid data, and for daily average in weather data. From the table, we can see that cameras, signals and signs have a notable positive correlation, speed (limits) have a slight negative correlation and weather data has little correlation. However, further figures will go into each variable and provide an interpretation of these correlations (i.e. the causality).

## 2.2 Traffic speed limit data

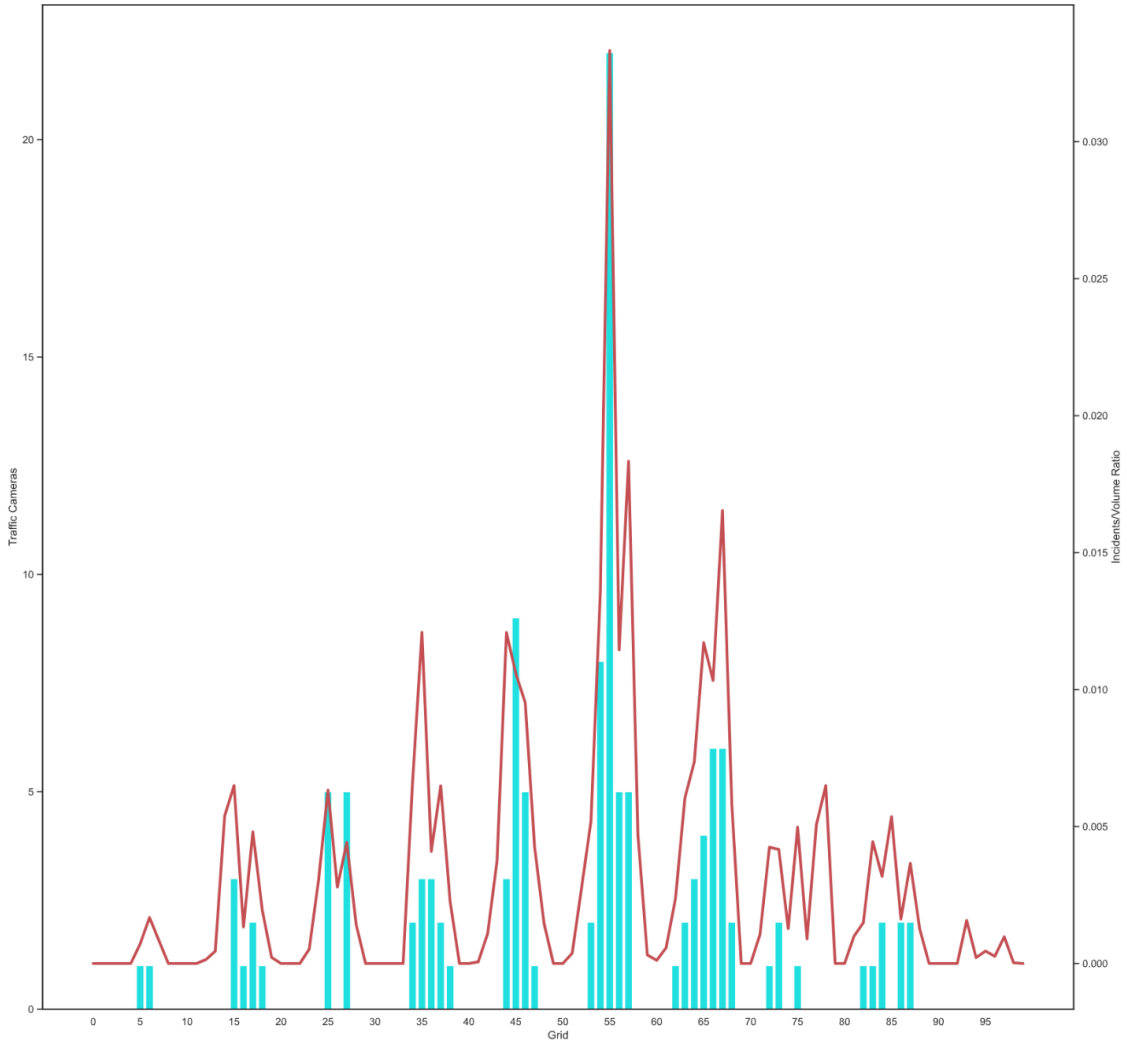
### 2.2.1 Average speed limit (bar graph) and traffic incident/volume ratio (line plot)




Average speed limit seems relatively consistent from grid to grid. All values range from around 50km/h to 70km/h. In fact, some grids with lower average speed corresponds to higher incident/volume ratio, such as near grids 35, 45, 55 and 65. The reason could be that these grids are from the downtown area. Grid 55 is at downtown centre. It has the highest incident/volume ration but the lowest average speed at only around 40km/h. Hence there is no clear evidence of a positive correlation between average speed limit and incidents.

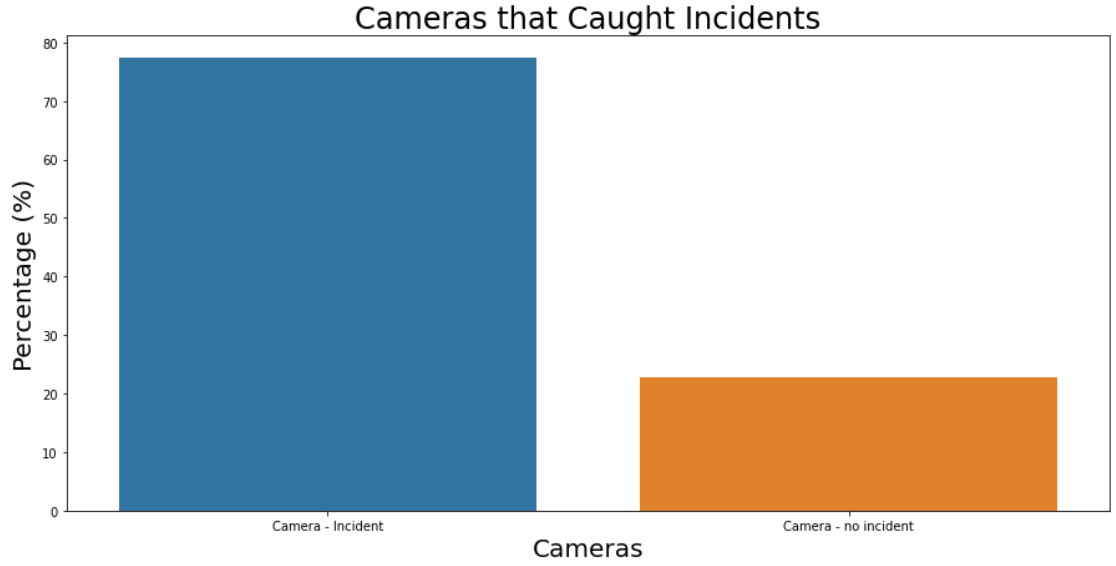
## 2.3 Traffic camera data

### 2.3.1 Traffic cameras (bar graph) and traffic incident/volume ratio (line plot)



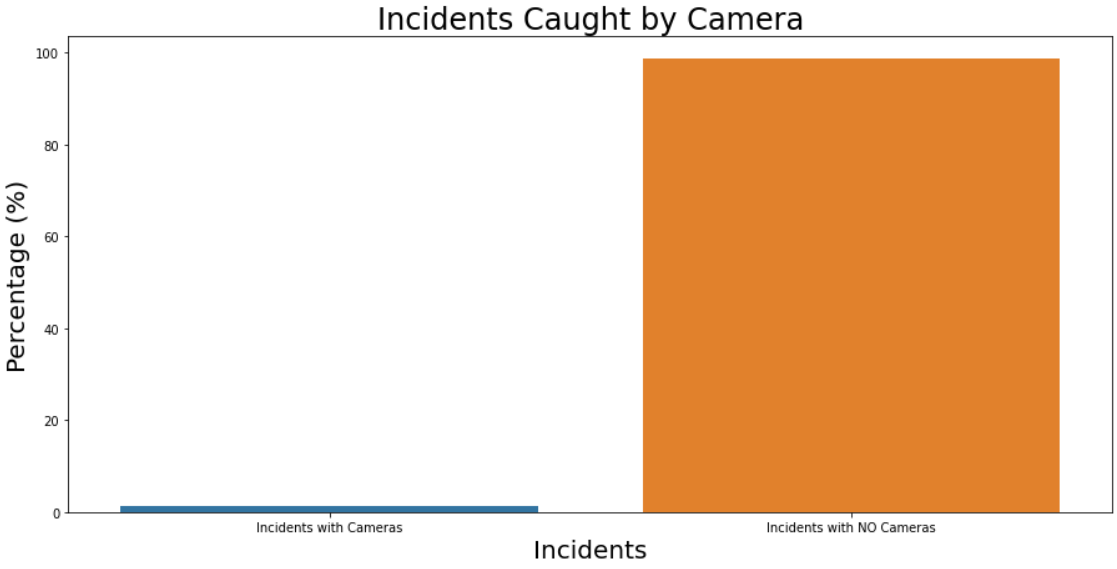

Grids with more cameras tends to have more incidents. The graph shows a relatively strong positive correlation between number of traffic cameras and incidents occurred. One reason could be that more drivers tend to brake harder to avoid red light tickets, increasing rear-end collisions and other "non-angle" collisions.

### 2.3.2 Cameras that caught incidents (bar graph)




The figure above shows that 77% of cameras came within range of an incident and 23% did not. The cameras are potentially well placed.

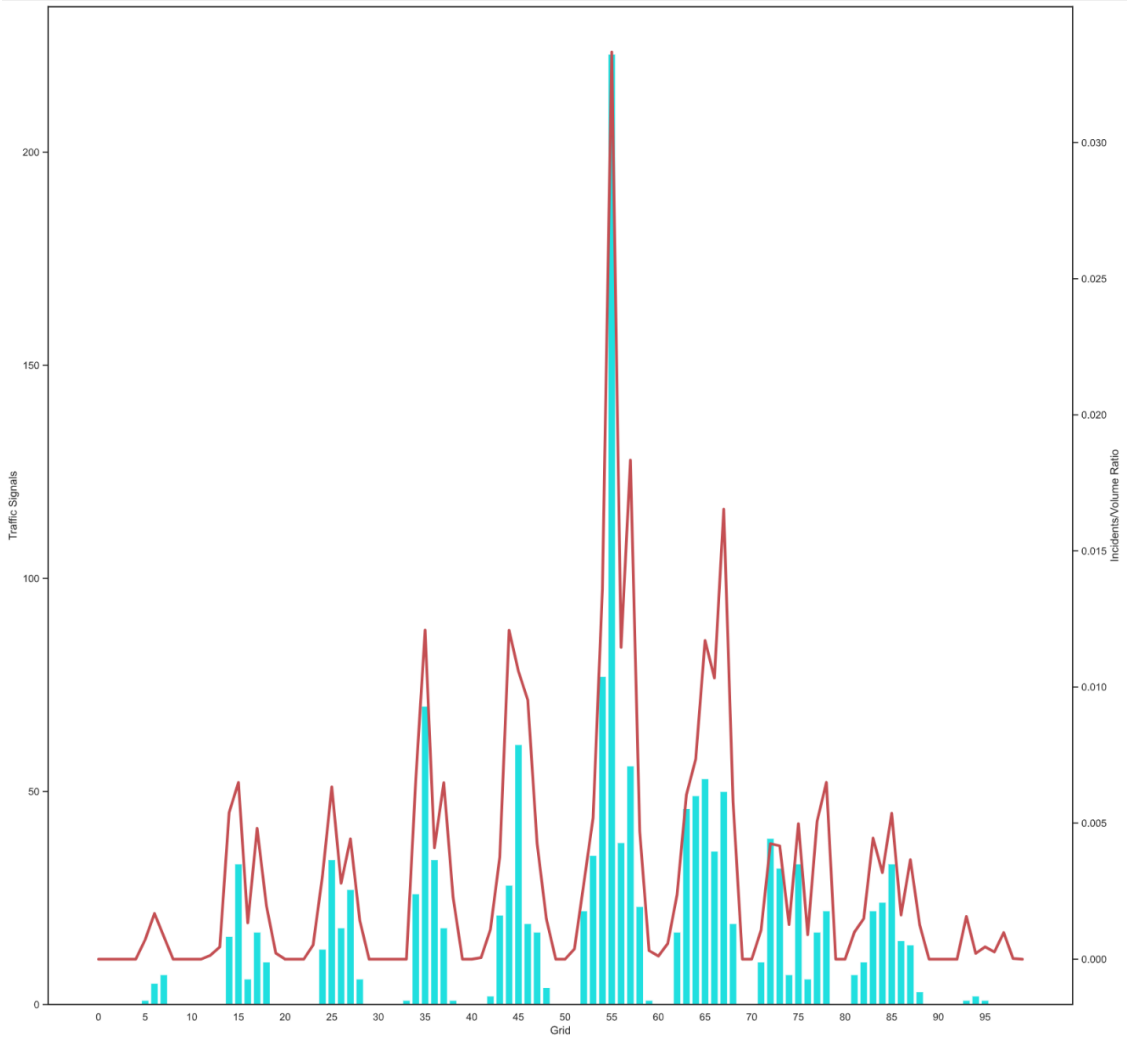
### 2.3.3 Incidents caught by camera (bar graph)




The figure above shows that only 1.4% of incidents occur near cameras. From looking at this data, as well as the signals data (incidents that happen near traffic signals / intersections), most incidents probably occur near locations of speed changes (slowing down/speeding up). Most cameras do not occur near incidents, because there are often "speed" cameras, that are located in the middle of highways where cars are all moving at the same speeds.

## 2.4 Traffic signal data

### 2.4.1 Traffic signals (bar graph) and traffic incident/volume ratio (line plot)




Grids with more traffic signals tends to have higher incidents. Overall, it shows a strong positive correlation between number of traffic signals and incidents occurred, meaning that traffic signals may increase the number of accidents in a given area. This may be due to the higher possibility of rear-end collisions near places with traffic signals.

### 2.4.2 Intersection types table

| | INT_TYPE | Incidents | Signal Count | %Incidents to Signal Count |
|---|---|---|---|---|
| 0 | 1/2 signal | 5.0 | 21 | 23.809524 |
| 1 | Traffic signal | 161.0 | 1010 | 15.940594 |
| 2 | Traffic signal T intersection | 27.0 | 232 | 11.637931 |
| 3 | Overhead Flasher | 14.0 | 257 | 5.447471 |
| 4 | Pedestrian RRFB | 1.0 | 159 | 0.628931 |
| 5 | 1/4 signal | 0.0 | 10 | 0.000000 |
| 6 | Fire signal | 0.0 | 7 | 0.000000 |

There is a higher ratio of incidents that occur near 1/2 signals. However, there are a lot fewer 1/2 signals than other signals, so this result should be taken with a grain of salt

### 2.4.3 Signals by city quadrant table

| | QUADRANT_x | Incidents | Signal Count | %Incidents to Signal Count |
|---|---|---|---|---|
| 0 | S | 19.0 | 46 | 41.304348 |
| 1 | SW | 83.0 | 497 | 16.700201 |
| 2 | SE | 62.0 | 457 | 13.566740 |
| 3 | NE | 30.0 | 294 | 10.204082 |
| 4 | NW | 14.0 | 355 | 3.943662 |
| 5 | E | 0.0 | 11 | 0.000000 |
| 6 | N | 0.0 | 36 | 0.000000 |

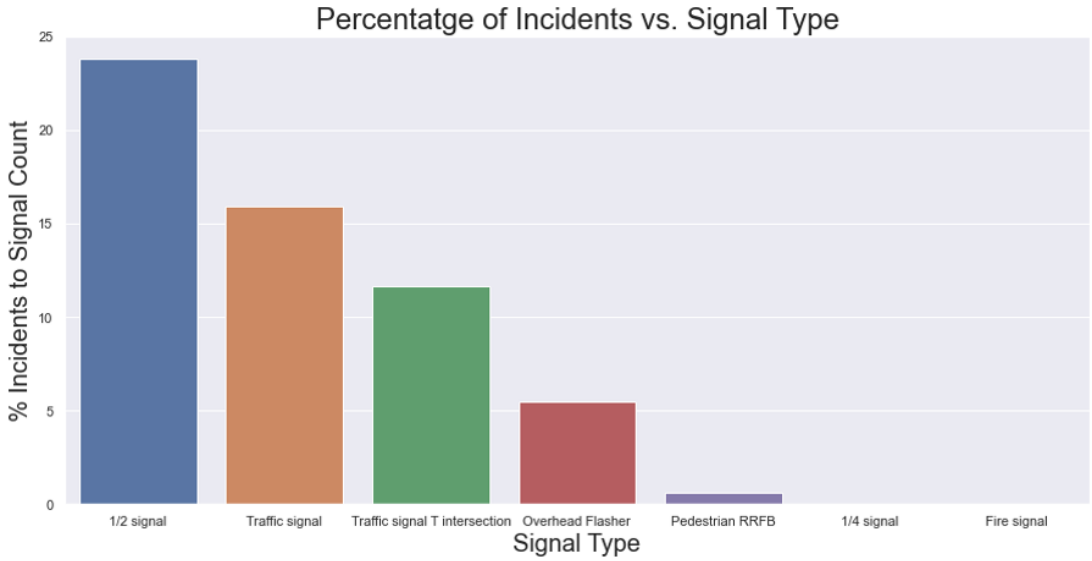
The South quadrant of calgary has the highest incident to traffic signal percent.

### 2.4.4 Signals with/without pedestrian button table

| | PEDBUTTONS | Incidents | Signal Count | %Incidents to Signal Count |
|---|---|---|---|---|
| 0 | No | 94.0 | 330 | 28.484848 |
| 1 | Yes | 114.0 | 1364 | 8.357771 |

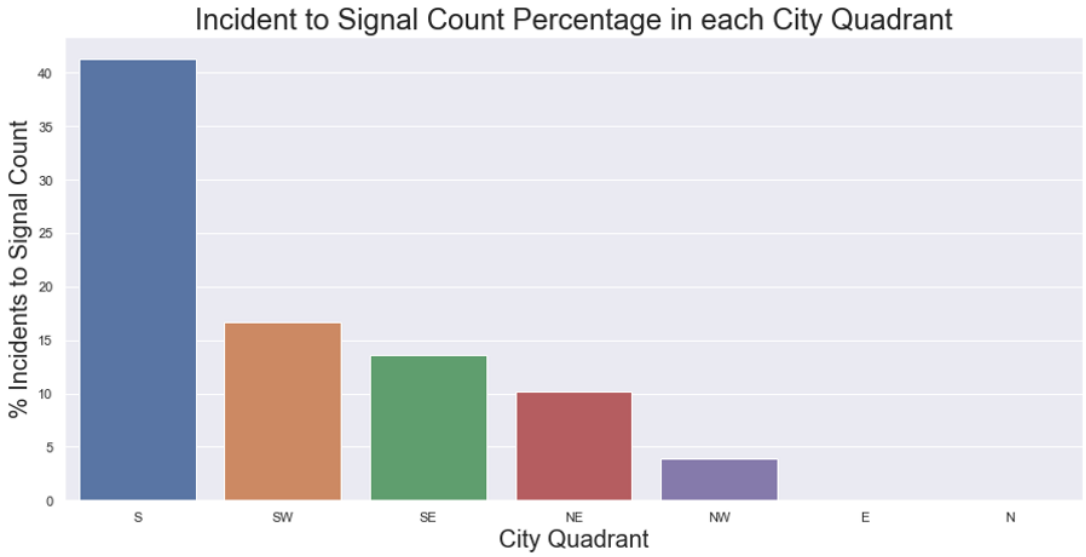
You are 20% more likely to get into an accident at intersections without a pedestrian button.

### 2.4.5 Percentage of incidents vs. signal type (histogram)



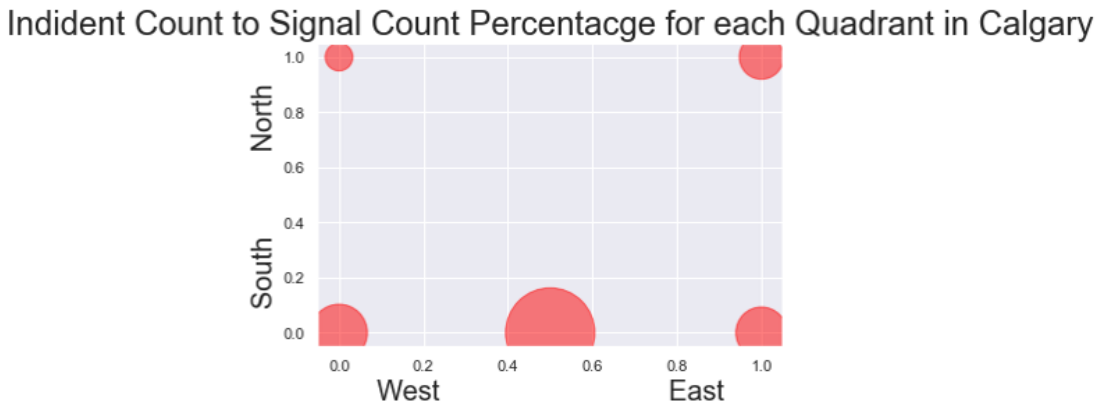

The histogram shown above indicates that the highest percentage of incidents occur by "1/2" signals, followed by the other type of intersections that have traffic signals. A "1/2 Signal" is an intersection where half of the roads have to stop at an intersection signs, where as the others can pass through most of the time. (2/4 of the roads have a traffic signal). This could be dangerous because the intersections are less noticeable and are usually found when a smaller road needs to pass through a larger/faster road.

### 2.4.6 Incident to signal count percentage in each city quadrant (histogram)



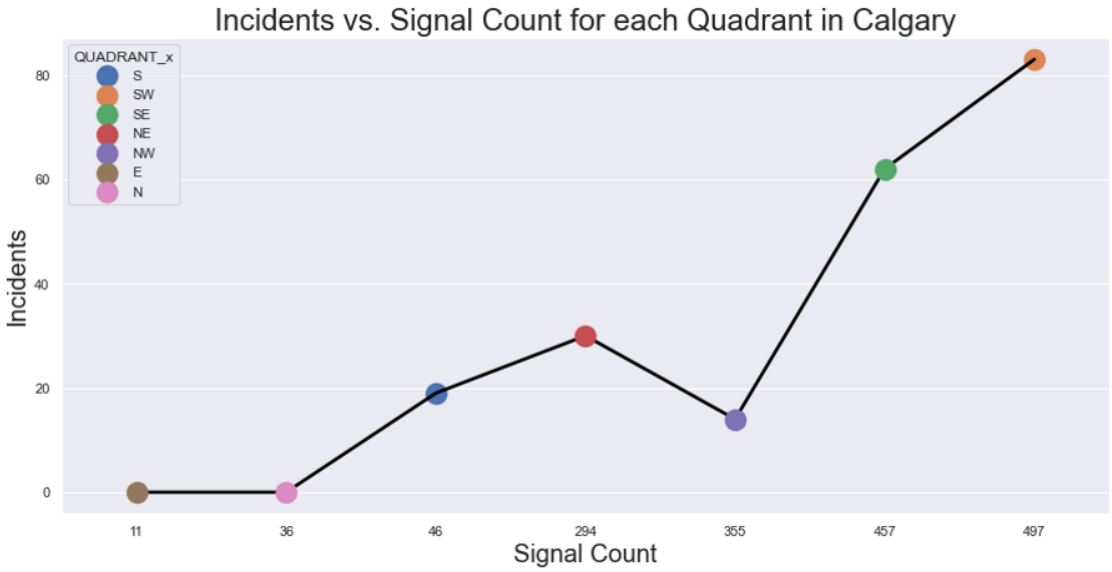

The figure above indicates that the Southern Quadrant of the city has the most accidents per signal ratio.

### 2.4.7 Incident count to signal count percentage in each city quadrant (bubble graph)




This figure shows the percentage incidents to signal count in each quadrant of the city. The south has the highest incident to traffic signal ratio.

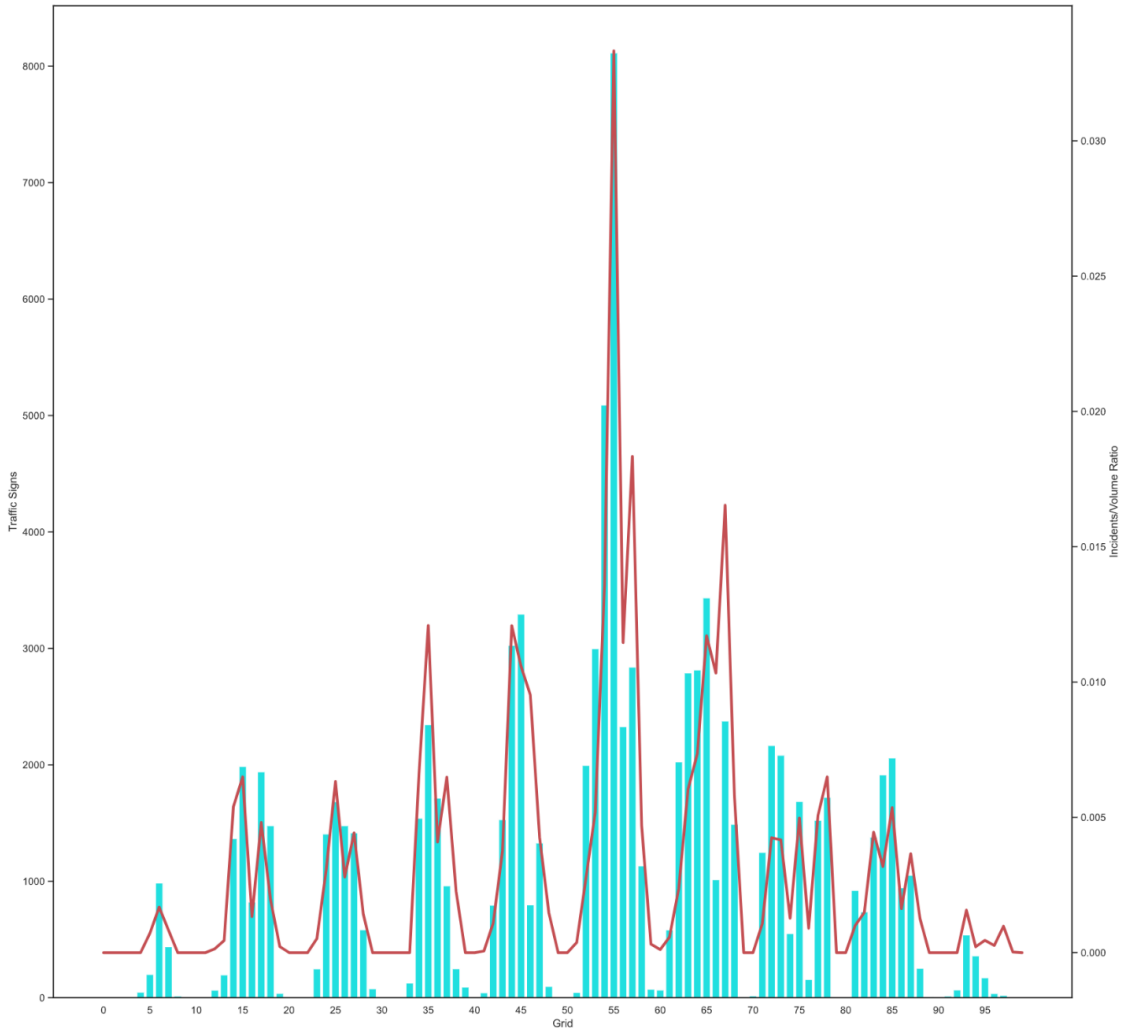
### 2.4.8 Incident count to signal count in each city quadrant (line plot)




The figure above shows that the SW quadrant of the city (orange marker) has the most incidents, however it also has the most signals.

## 2.5 Traffic sign data

### 2.5.1 Traffic signs (bar graph) and traffic incident/volume ratio (line plot)



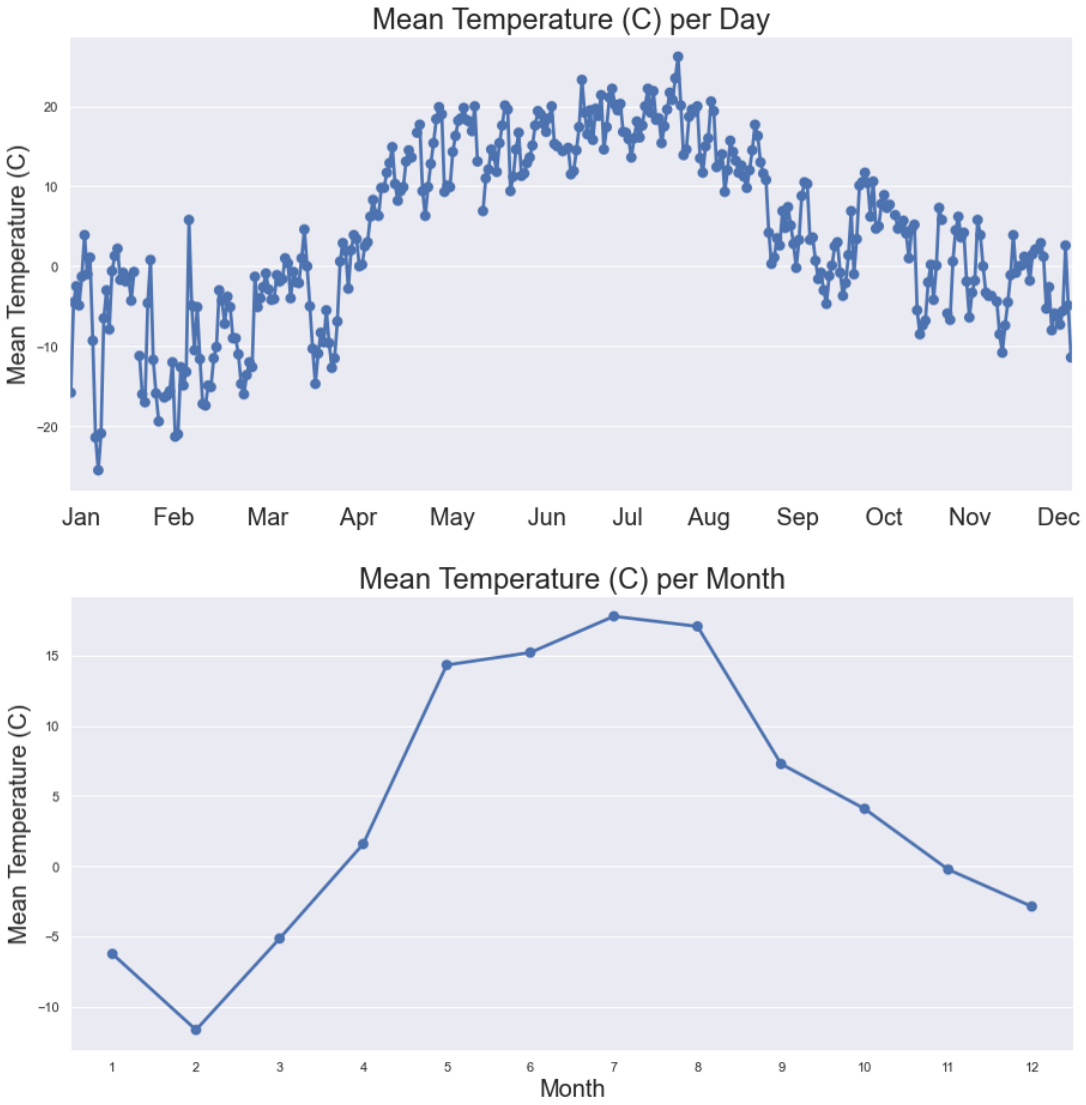

Grids with more traffic signs tends to have more incidents. The graph shows a strong positive correlation between number of traffic signs and incidents occurred. This could be due to that most traffic signs are installed at places with higher incidents, as it provides information to drivers and pedestrians to maintain order and reduce accidents.

## 2.6 Weather data

### 2.6.1 Visibility and mean temperature table

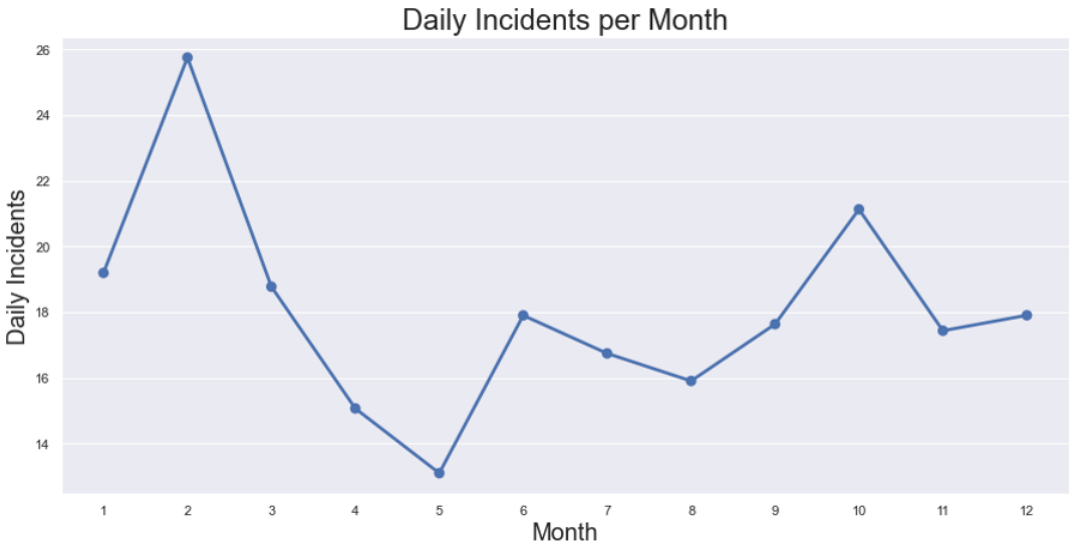
| | Visibility (km) | Mean Temp (C) |
|---|---|---|
| count | 365.000000 | 356.000000 |
| mean | 29.843530 | 4.378933 |
| std | 14.312455 | 10.910681 |
| min | 1.758333 | -25.500000 |
| 25% | 17.033333 | -3.400000 |
| 50% | 33.966667 | 3.900000 |
| 75% | 41.229167 | 13.925000 |
| max | 53.995833 | 26.200000 |

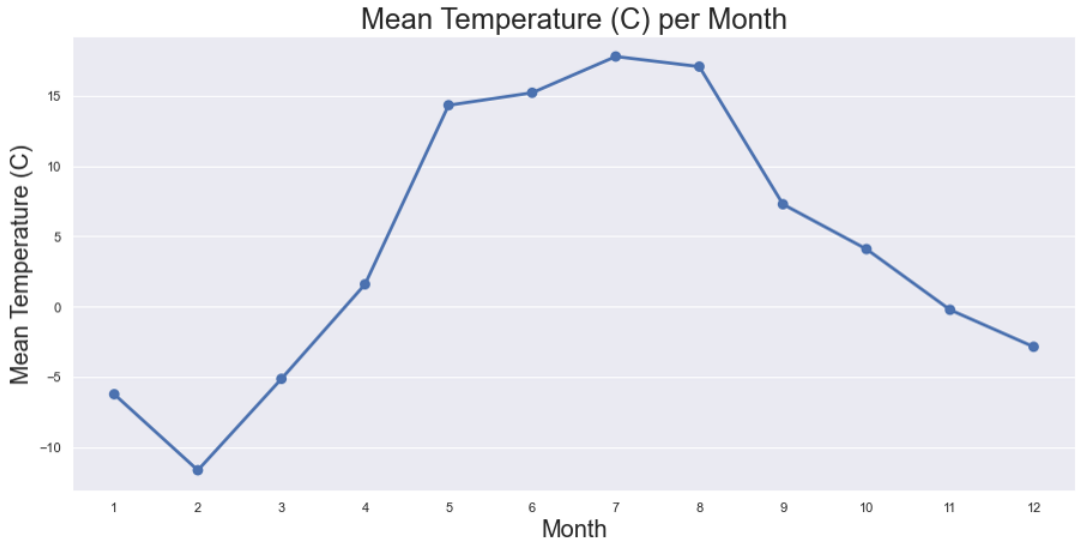
### 2.6.2 Mean temperature per day/month




The above figure shows the average monthly temperature in Calgary in 2018. We have the coldest weather in Jan/Feb, and the best/hottest weather from May until August.

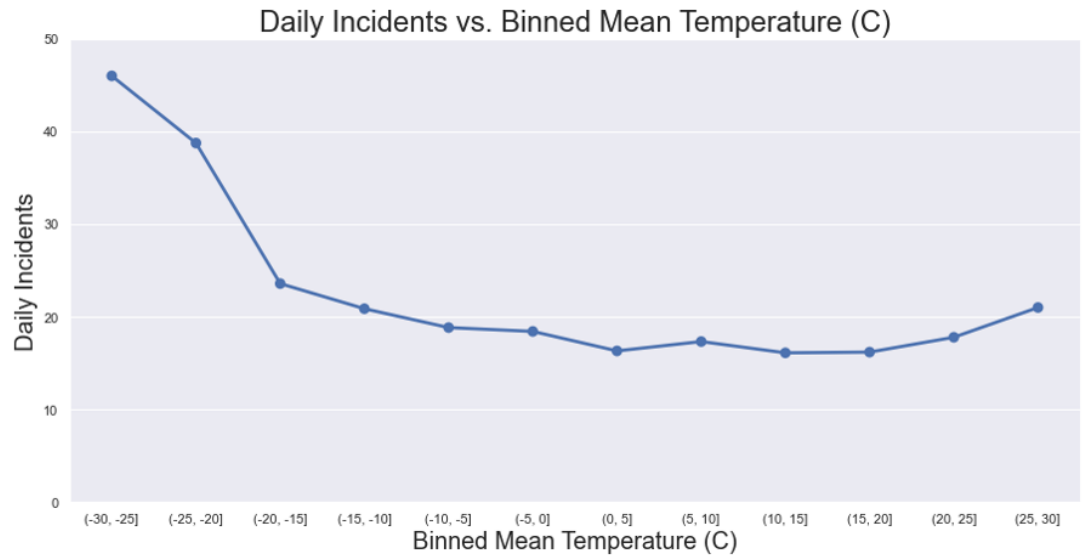
### 2.6.3 Comparing average daily incidents and mean temperature



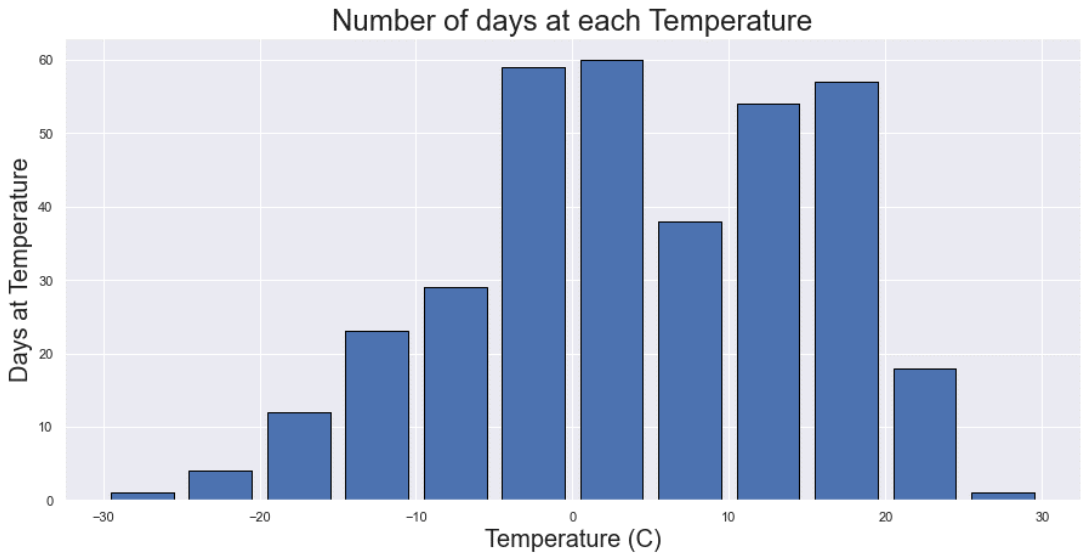

The coldest weather is in Jan/Feb. This is also the temperature and month that we see the most collision/day.

### 2.6.4 Daily incidents vs binned mean temperature



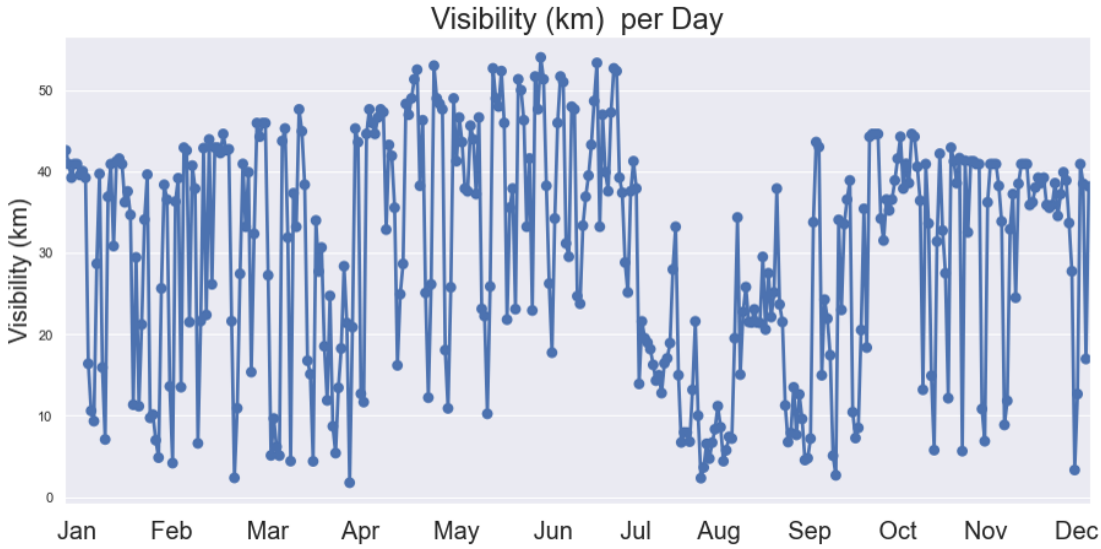

Daily incidents are relatively stable for temperature above -15C. However during bad/cold weather, the daily incidents go up.

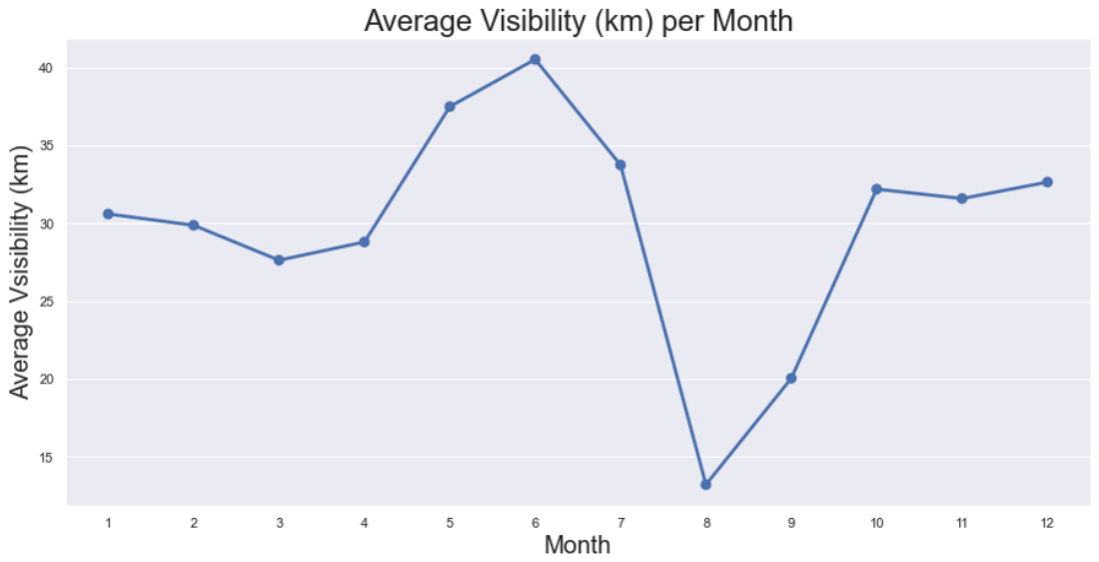
### 2.6.5 Number of days at each temperature (histogram)




The above figure shows that in 2018, the most common temperatures in Calgary are between -5C and +20C.

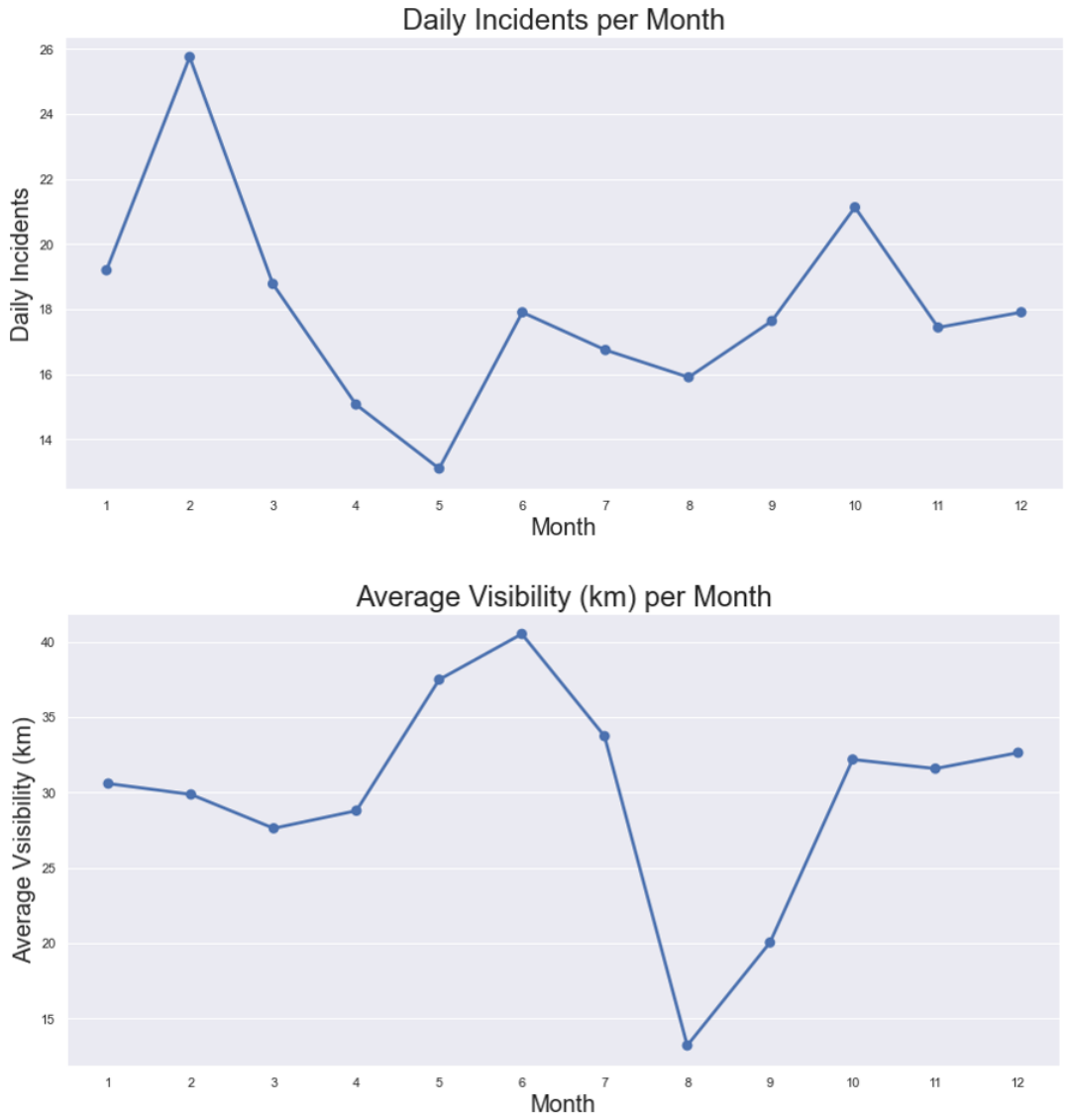
### 2.6.6 Average visiblity per day/month



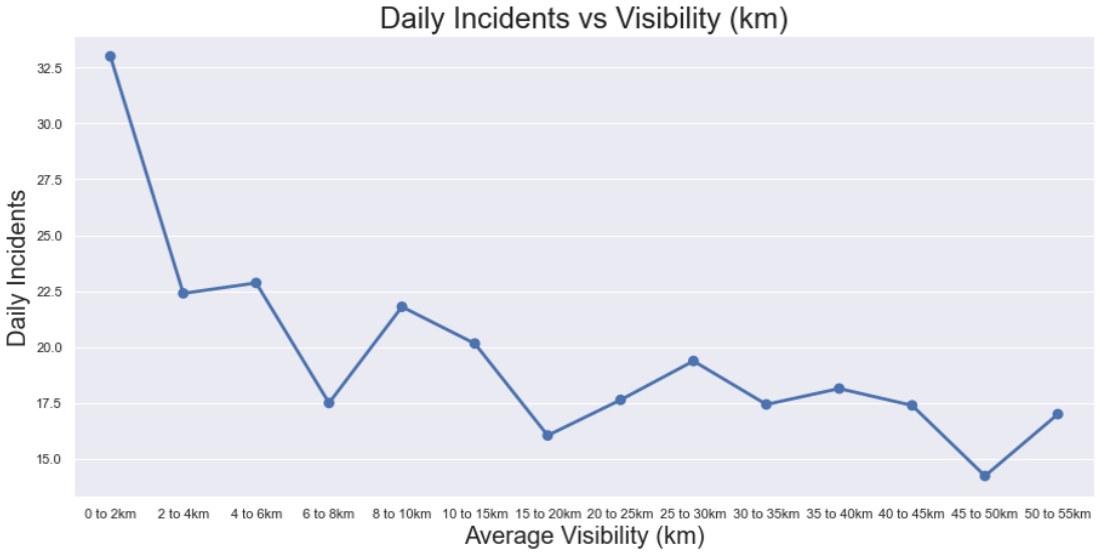

The above figure shows that the average visibility for each month can be seen in 2018. Visibility was at its lowest in August.

### 2.6.7 Comparing average daily incidents and visibility



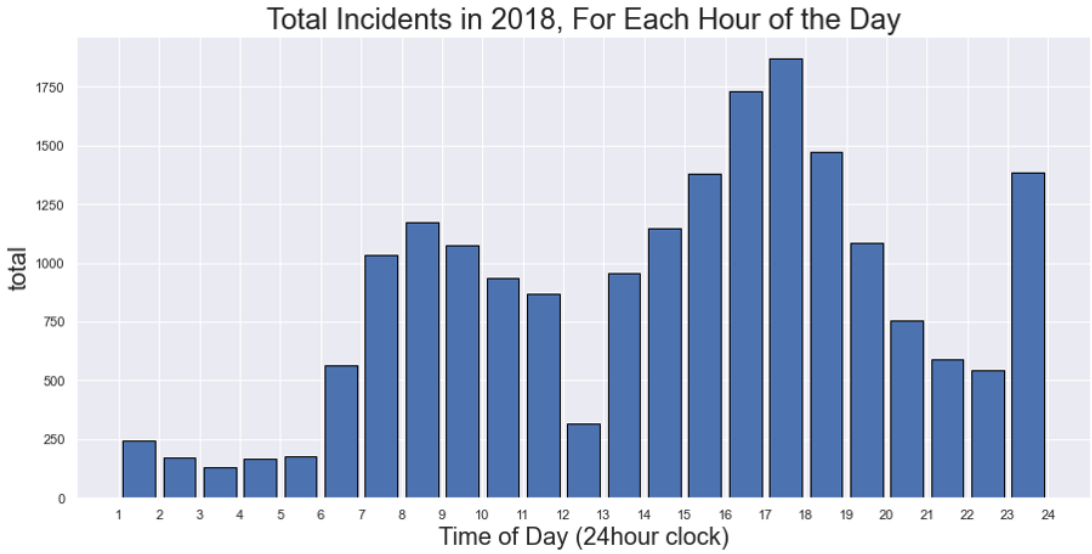

Comparing average visibility per month and average daily incident per month, not much correlation is shown. However, measuring visibility over a month average could be hiding some data. (The next figure will directly compare visibility to incidents)

### 2.6.8 Daily incidents vs binned visibility




Comparing average visibility per month and average daily incident per month, not much correlation is shown. However, measuring visibility over a month average could be hiding some data. (The next figure will directly compare visibility to incidents)

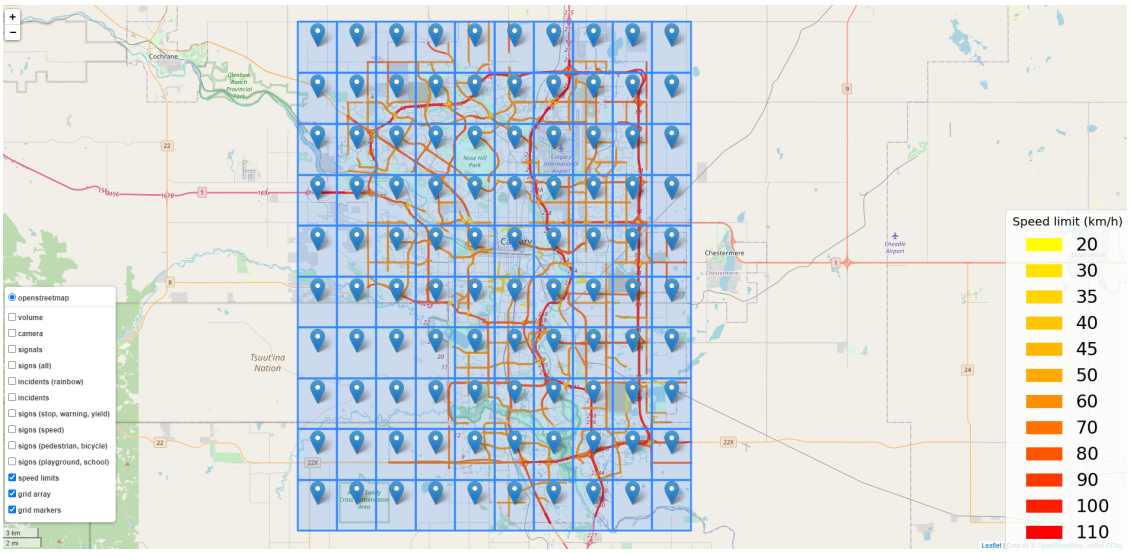
### 2.6.9 Total incidents per hour of the day




The above figure is very interesting. It shows that most incidents occur around 8am, and 5pm. These times correlate perfectly to rush hour times, when everyone is rushing into downtown to get to work, and traffic volumes are very high. Also, incidents are very high at midnight. This might be due to irresponsible driving during these hours or lower visibility.

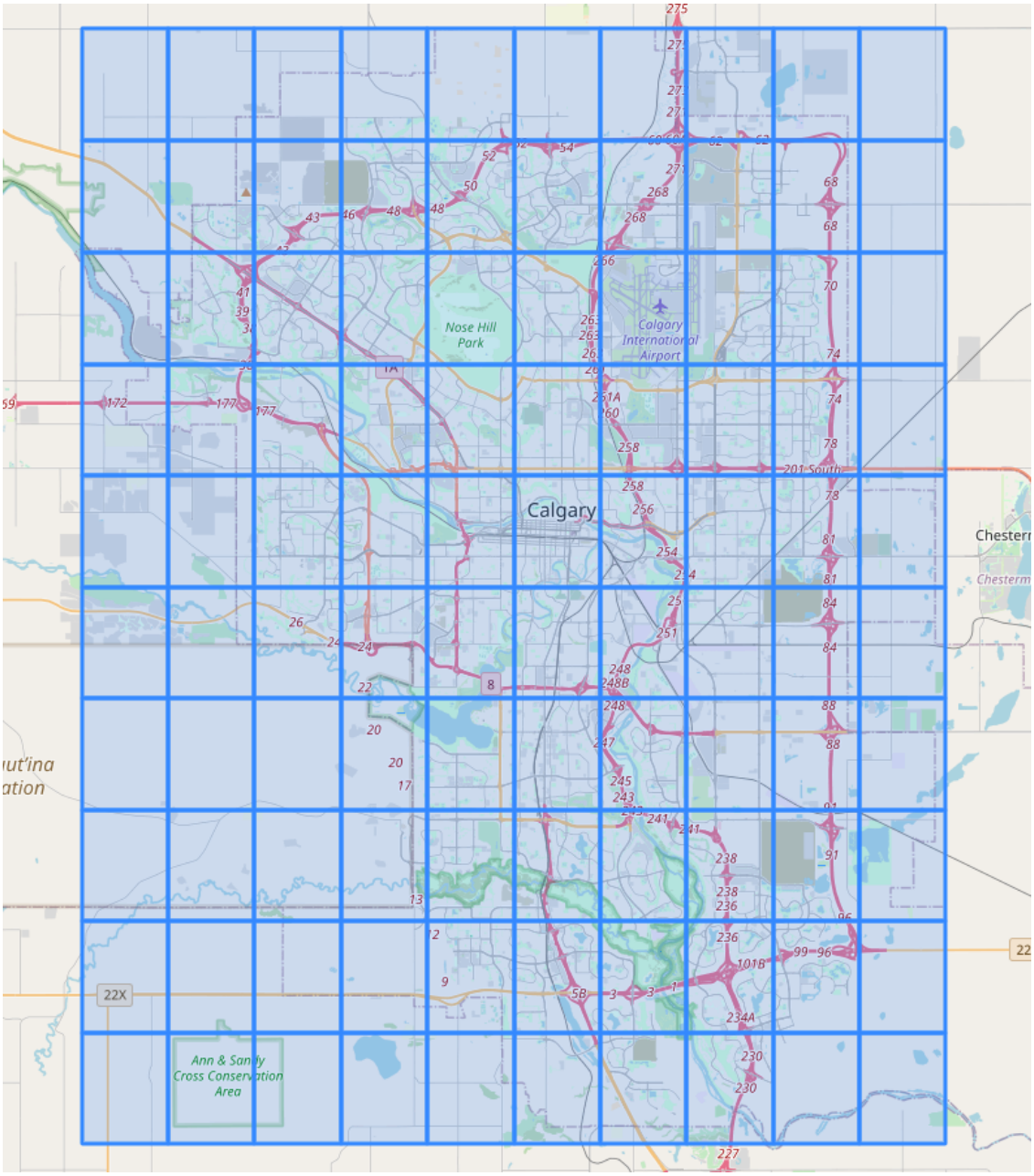
## 2.7 Maps

### 2.7.1 Full map




For full interactive map, see https://jchoi64.github.io/592_project_final/

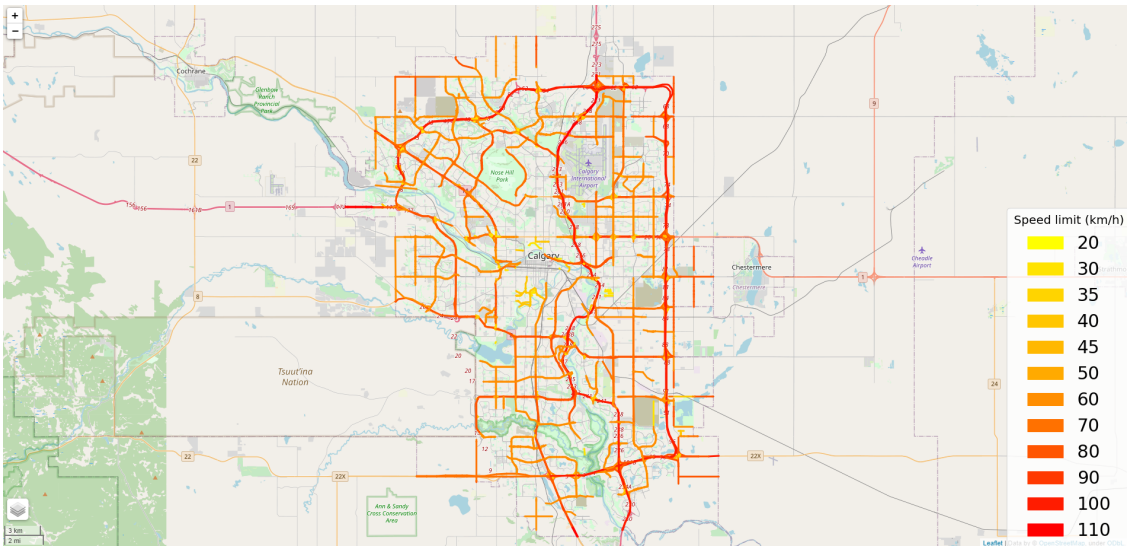
### 2.7.2 10x10 Grid



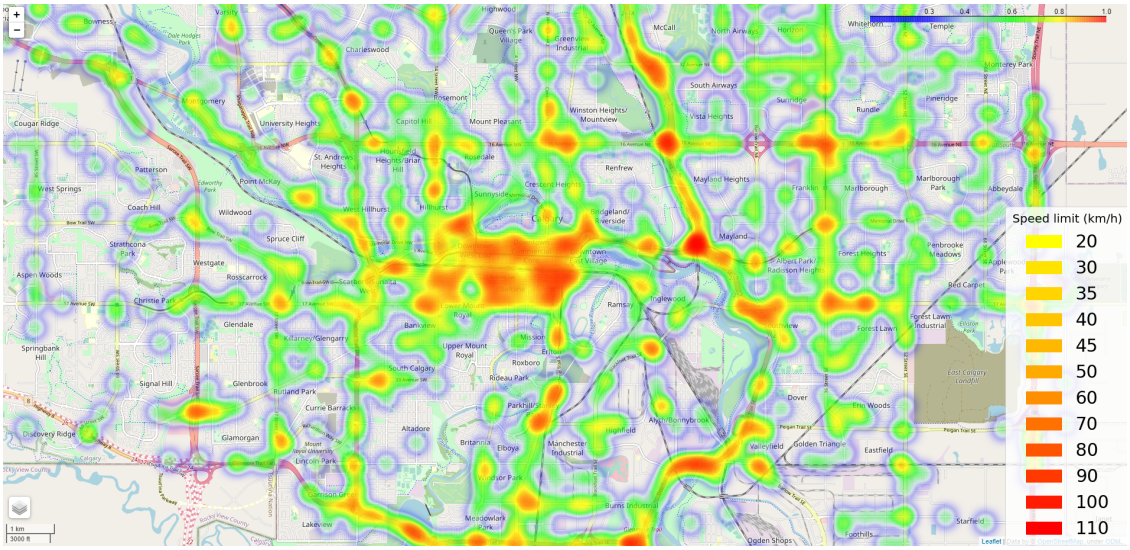
### 2.7.3 Markers


Grid: 55
Average Speed Limit: 42.94117647
Average Traffic Volume: 13951.75439
Total Cameras: 22
Total Signals: 223
Total Incidents: 465
Total Signs: 8114
Calgary
22X

### 2.7.4 Road speed limits

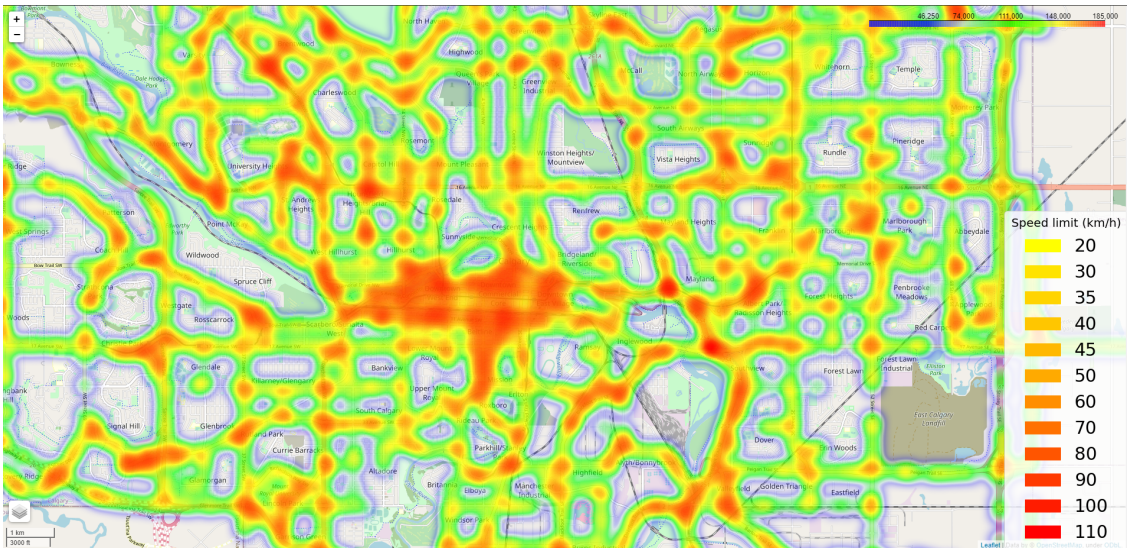



### 2.7.5 Incidents heat map (zoomed adjusted for best resolution)



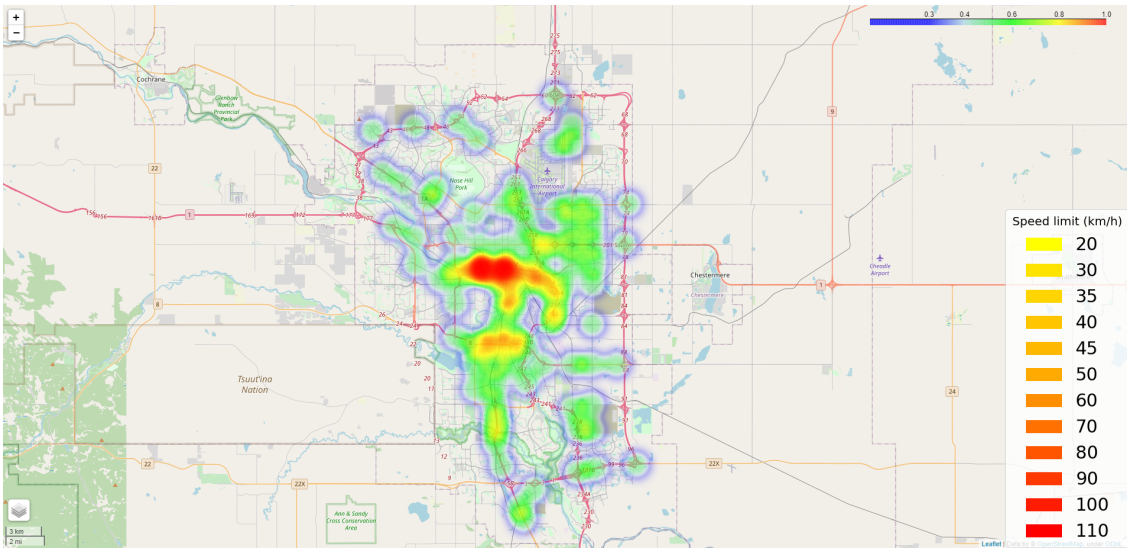

The incident heatmap shows us that most incidents occur in the downtown area, as well as intersections and merging roads. This correlates well with the volume heatmap which will be shown next. Intuitively this makes sense as more volume will generally lead to more incidents by default. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/)

### 2.7.6 Volume heat map (zoomed adjusted for best resolution)



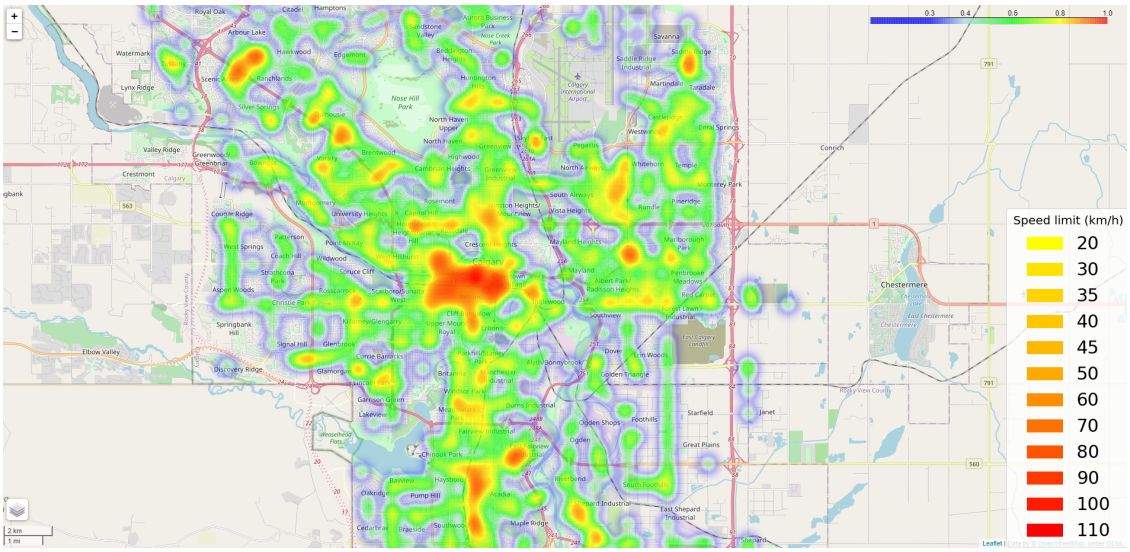

The volume heatmap shows that most traffic occurs within downtown. Other noteworthy areas of high volumes are at intersections and road merges. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/)

### 2.7.7 Camera heat map (zoomed adjusted for best resolution)



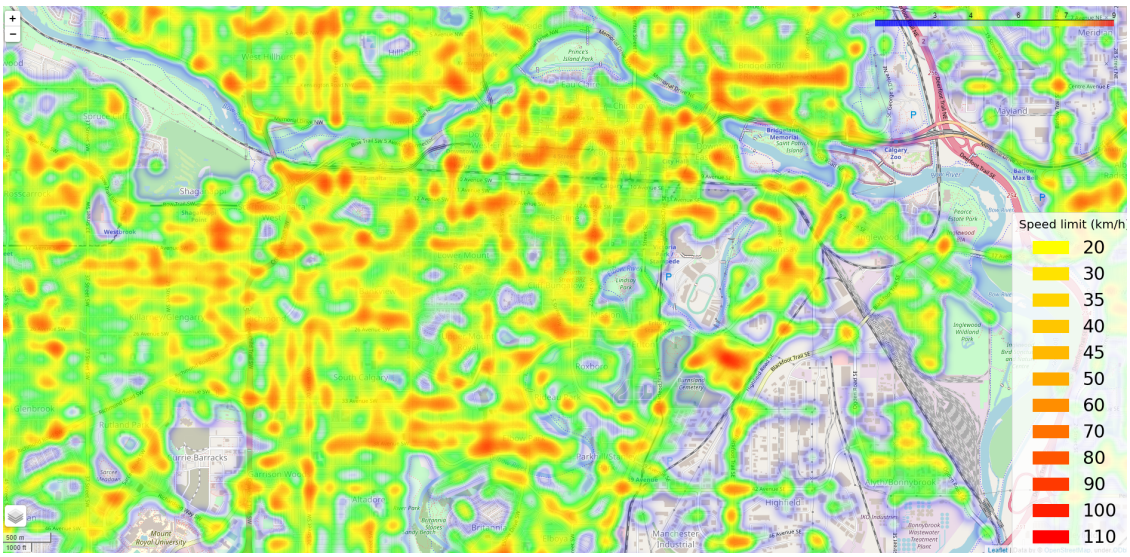

While camera data is sparse (only 126 points throughout all of Calgary), it shows that most are located within the downtown region. This also explains the high correlation between cameras and incidents because most incidents occur in downtown where the cameras are placed. This may also be deliberate by the city of Calgary, by placing cameras in high incident locations. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/)

### 2.7.8 Signals heat map (zoomed adjusted for best resolution)



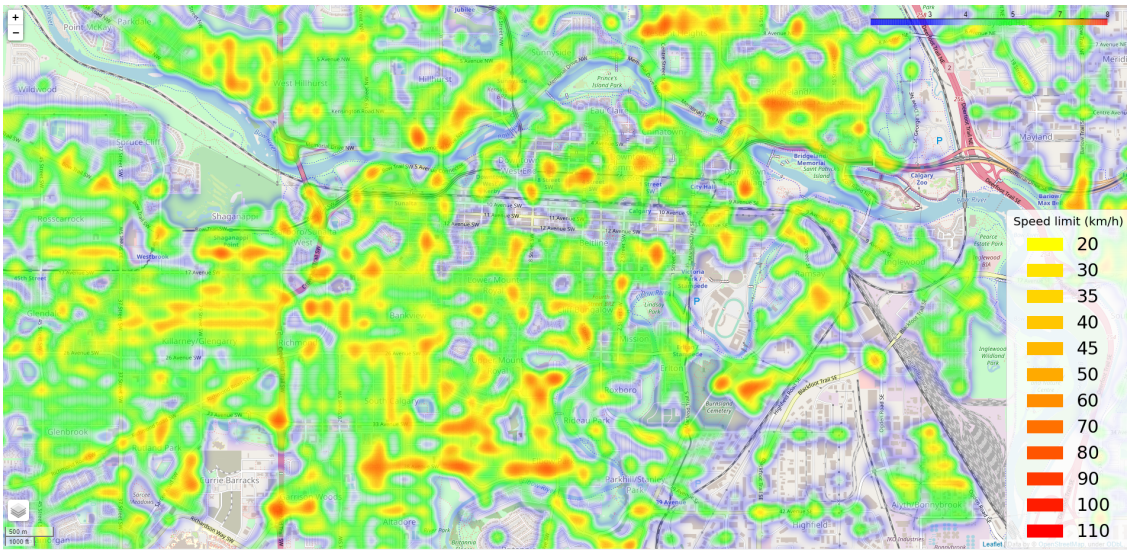

Similarily with cameras, most signals are found within downtown Calgary. This is one reason why the correlation is high with incidents. Another reason is that signals are also placed at intersections which is another area with high incident count. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/)

### 2.7.9 Signs (all signs) heat map (zoomed adjusted for best resolution)



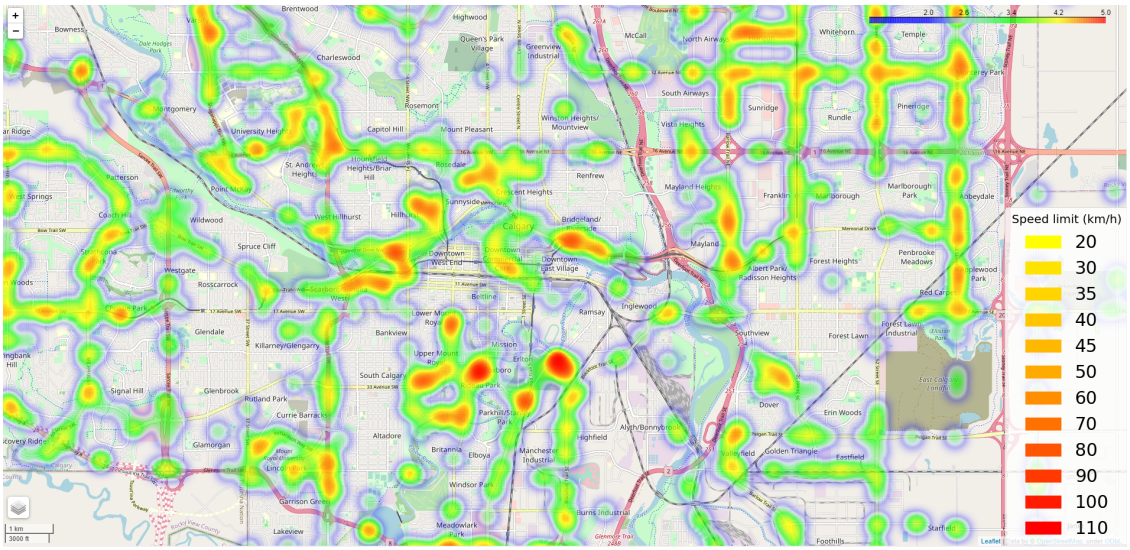

Due to the sheer number of signs, the heatmap (that includes all relevant signs) does not yield too much information. The most that can be determined from this is that since most signs occur at intersections, this would explain the high correlation between signs and incidents. Instead the next figures will show different groupings to determine if more information is available. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/)

### 2.7.10 Signs (stop, warning, yield) heat map (zoomed adjusted for best resolution)



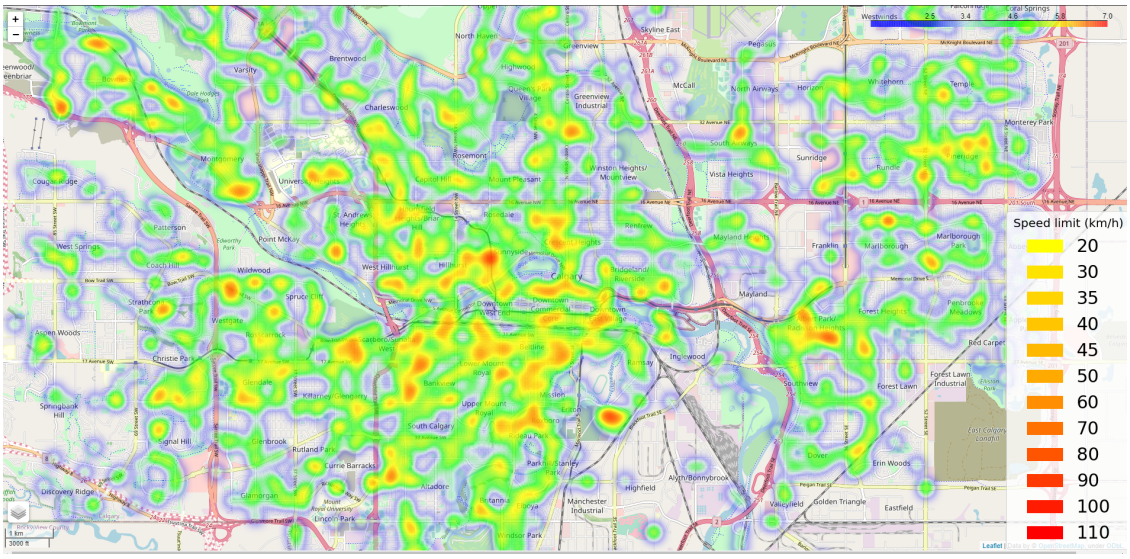

One grouping that was chosen was stop signs, warning signs and yield signs. Failure to follow these signs could lead to an incident which is why these were grouped. However if properly followed (which most driver should be doing), incidents could be lower in these areas. If layered with the incident data and ignoring the downtown region (which has a high incident count and sign count by design), we actually see that they don't overlap too much, which would mean that most people are following the rules of the road.(To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/) (layer signs (stop, warning, yield) first, then incidents to see the overlap)

### 2.7.11 Signs (speed) heat map (zoomed adjusted for best resolution)



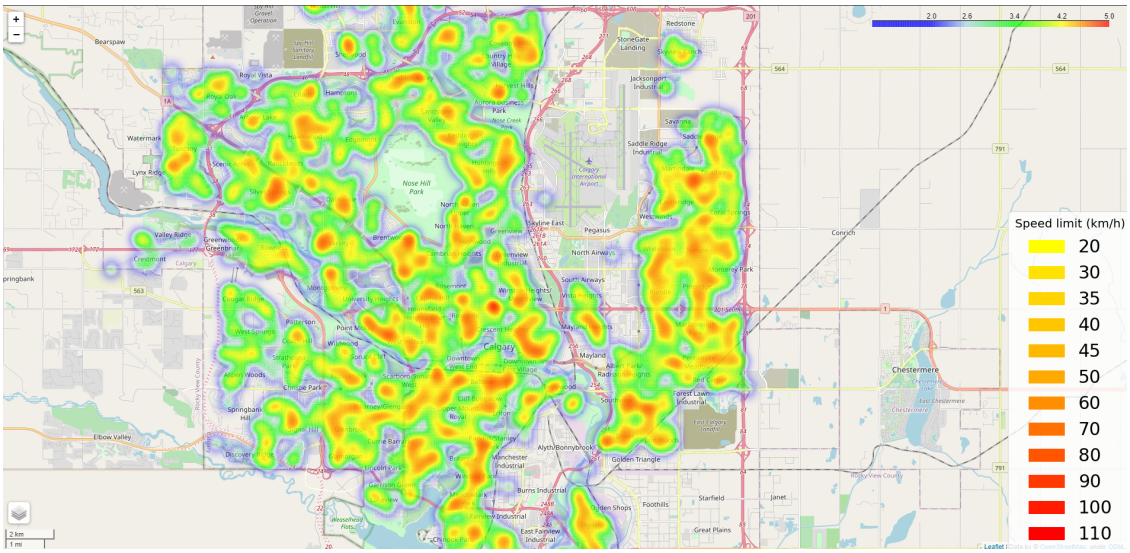

Speed signs were chosen as another grouping with the same reason for plotting the speed limit of the roads as lines. However the sparseness of the data does not yield much information. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/) (layer signs (speed) first, then incidents to see the overlap)

### 2.7.12 Signs (pedestrian, bicycle) heat map (zoomed adjusted for best resolution)




Pedestrian and bicycle paths were chosen as a group but due to the incident data not distinguishing between pedestrian and vehicle incidents, we cannot draw any decisive conclusions from this heatmap. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/) (layer signs (pedestrian, bicycle) first, then incidents to see the overlap)

### 2.7.13 Signs (playground, school) heat map (zoomed adjusted for best resolution)




School and playgrounds were chosen as the final grouping to see if a lower incident rate was found in these areas as drivers are generally more cautious/drive slower in these ares. If we zoom in, we can see that there is very low overlap between incidents and these areas, providing evidence that these areas have lower overall incident rates. (To see the full view of Calgary, please use the interactive map https://jchoi64.github.io/592_project_final/) (layer signs (playground, school) first, then incidents to see the overlap)

# 3 Code

## 3.1 Traffic Cameras.ipynb (Traffic Cameras Analysis file)

### 3.1.1 Libraries and functions

[ ]:
```
%matplotlib inline
import matplotlib

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from csv import reader
import folium
```

### 3.1.2 Reading and merging data

[ ]:
```
#import traffic Camera location
camera_df = pd.read_csv('..\..\CSV_files\Traffic_Camera_Locations.csv')
camera_df['data']='TrafficCameraLocations'
#camera_df.head() #<-- visual QC of dataframe
```

[ ]:
```
#add traffic incidents
incidents_df = pd.read_csv('..\..\CSV_files\Traffic_Incidents.csv')
incidents_df['data']='Camera - Incident' #create a new column, to identify␣
 ↪which dataframe the data came from after the merge
#rename columns so they match other dataframes columns - makes it easier for␣
 ↪the merge
incidents_df=incidents_df.rename(columns={'Latitude':'latitude','Longitude':
 ↪'longitude','Count':'Incidents'})
#incidents_df.head() #<-- visual QC of dataframe
```

[ ]:
```
#data given to 6 significant digets. Here we reduce it to 4, so that camera␣
 ↪data will be matched up with accidents that occured
#within ~10m's
incidents_df = incidents_df.round({'latitude':4, 'longitude':4})
camera_df = camera_df.round({'latitude':4, 'longitude':4})
#camera_df.head() #<-- visual QC of dataframe
```

[ ]:
```
#merging two dataframes into df_total. merging on camera_df so we can determine␣
 ↪which cameras has an incident occur near them
df_total = pd.merge(left=incidents_df, right=camera_df, how='right',␣
 ↪left_on=['latitude','longitude'], right_on=['latitude','longitude'])
#nan data in "data_x" column belong to camera rows that had no incidents
#so replace those nan values with "Camera - no incident", so we can "group-by"␣
 ↪later to count number of "camera no incident"
# and "camera - incident".
df_total['data_x'] = df_total['data_x'].fillna("Camera - no incident")
#need to fill "incidents" with a "1" value for even the non-incident rows, so␣
 ↪that we have something to count when we 'groupby'
df_total['Incidents'] = df_total['Incidents'].fillna(1)
df_total= df_total.rename(columns={"Incidents":"Count"})  #rename incidents to␣
 ↪count, so the title makes more sense
#df_total.head() #<-- visual QC of dataframe
```

### 3.1.3 Table showing the percentage of cameras that caught an incident (within ~10m of a camera (Table))

[ ]:
```
#'groupby-sum' /(count) the number of incident rows, and non-incident rows. and␣
 ↪put into a nice table
df_total = df_total.groupby(['data_x']).sum()
#df_total   #4 decimal points =~10m radius around camera latitudes and␣
 ↪longitudes
```

[ ]:
```
#It would be nice to calculate % of cameras that caught incidents in the above␣
 ↪table
#To do that, need to calculate total first
totalCount = df_total['Count'].sum()
#totalCount #<-- visual QC check
df_total['%'] = (df_total['Count']/totalCount)*100
df_total = df_total[['Count','%']]
df_total #<-- show table
```

### 3.1.4 Cameras near incidents (bar graph)

[ ]:
```
#create histogram plot of above data
df_total=df_total.reset_index()
fig,ax = plt.subplots(figsize=(15,7))

sns.barplot(x = 'data_x', y = '%', data = df_total)

plt.ylabel("Percentage (%)",fontsize=20)
plt.xlabel("Cameras",fontsize=20)
plt.title("Cameras that Caught Incidents",fontsize=24)
```

### 3.1.5 Traffic Cameras (part 2)

[ ]:
```
# Re-Doing above tables and graphs, but merging onto "incident dataframe", so␣
 ↪that we can determine
# how many incidents occured near cameras (oposite of the above)
incidents_df = incidents_df.round({'latitude':4, 'longitude':4}) #rounding lat␣
 ↪and long aloud them to be grouped wihtin a 10m accuracy
camera_df = camera_df.round({'latitude':4, 'longitude':4})      #this is nice,␣
 ↪because accidents and cameras dont need to occur exactly ontop of eachother
camera_df.data = 'Incidents with Cameras' #change input calues of 'data' column␣
 ↪to set up for histogram plot later
#camera_df#<-- visual QC of dataframe
```

[ ]:
```
#merge data on traffic incidents this time. so we have the total amount of␣
 ↪incidents, but only some of them have camera data
#this allows us to compare how many incidents had cameras near them
df_total = pd.merge(left=incidents_df, right=camera_df, how='left',␣
 ↪left_on=['latitude','longitude'], right_on=['latitude','longitude'])
df_total['data_x'] = df_total['data_x'].fillna("Camera")
df_total['data_y'] = df_total['data_y'].fillna('Incidents with NO Cameras')␣
 ↪#filled na's with a proper name, so we can groupby.
df_total= df_total.rename(columns={"Incidents":"Count"})
#df_total.tail(20)#<-- visual QC of dataframe
```

[ ]:
```
totalCount = df_total['Count'].sum() #calculate the total rows, so we can␣
 ↪calculate percentage
#totalCount #<-- visual QC of output
```

### 3.1.6 Number and percentage of incidents that occur with and without a camera nearby (Table)

[ ]:
```
df_total2 = df_total.groupby(['data_y']).sum()
df_total2['%'] = (df_total2['Count']/totalCount)*100
df_total2 = df_total2[['Count','%']]
#JOINED ONTO INCIDENTS, so we see only incident rows, and can see how many have␣
 ↪cameras
df_total2 #<-- show table
```

### 3.1.7 Incidents that occured with and without a camera nearby (bar graph)

[ ]:
```
#create histogram
df_total2=df_total2.reset_index()
fig,ax = plt.subplots(figsize=(15,7))
sns.barplot(x = 'data_y', y = '%', data = df_total2)

plt.ylabel("Percentage (%)",fontsize=20)
plt.xlabel("Incidents",fontsize=20)
plt.title("Incidents Caught by Camera",fontsize=24)
```

## 3.2 Traffic Signals.ipynb (Traffic Signals Analysis)

### 3.2.1 Libraries and functions

[ ]:
```
import numpy as np
import pandas as pd
import folium
import geopandas as gpd
from shapely.geometry import Polygon
from shapely.geometry import box
import matplotlib.pyplot as plt
import seaborn as sns
```

### 3.2.2 Reading and merging data

[ ]:
```
#add traffic incidents
incidents_df = pd.read_csv('..\..\CSV_files\Traffic_Incidents.csv') #call csv
incidents_df['data']='TrafficIncidents' #create column to keep track of where␣
 ↪this data came from when merging dataframes
#rename columns so they match other dataframes
incidents_df=incidents_df.rename(columns={'Latitude':'latitude','Longitude':
 ↪'longitude','Count':'Incidents'})
#incidents_df.head() #<-- visual QC of dataframe
```

[ ]:
```
#data given to 6 significant digets, so this doesnt actualy do much(but we kept␣
 ↪the code, just incase we wanted to
#experiment with radius)
incidents_df = incidents_df.round({'latitude':6, 'longitude':6})
#incidents_df #<-- visual QC of dataframe
```

[ ]:
```
#add trafic signals
signals_df = pd.read_csv('..\..\CSV_files\Traffic_Signals.csv')
signals_df=signals_df.rename(columns={'Count':'Signal Count'})
signals_df['data']='TrafficSignals'
#signals_df #<-- visual QC of dataframe
```

[ ]:
```
signals_df = signals_df.round({'latitude':6, 'longitude':6})
```

[ ]:
```
#DID A LEFT JOIN, BECASUE MORE SIGNALS THAN ACCIDENTS DATA
accidents_vs_signals = pd.merge(left=signals_df, right=incidents_df,␣
 ↪how='left', left_on=['latitude','longitude'],␣
 ↪right_on=['latitude','longitude'])
#accidents_vs_signals #<-- visual QC of dataframe
```

### 3.2.3 Incidents and signal count vs. intersection type (Table)

[ ]:
```
INT_TYPE_TABLE = accidents_vs_signals.groupby(['INT_TYPE']).sum()
INT_TYPE_TABLE = INT_TYPE_TABLE[INT_TYPE_TABLE.columns.
 ↪difference(['latitude','longitude','ACCESSIBLE PEDESTRIAN SIGNAL'])]
INT_TYPE_TABLE['%Incidents to Signal Count'] = (INT_TYPE_TABLE['Incidents'] /␣
 ↪INT_TYPE_TABLE['Signal Count'])*100
INT_TYPE_TABLE = INT_TYPE_TABLE.sort_values(['%Incidents to Signal Count'],␣
 ↪ascending=False) #order table
INT_TYPE_TABLE #<---print table for analysis
```

### 3.2.4 Percentatge of incidents vs. signal type (Histogram)

[ ]:
```
#create histogram of the above table
INT_TYPE_TABLE=INT_TYPE_TABLE.reset_index()
fig,ax = plt.subplots(figsize=(15,7))
sns.barplot(x = 'INT_TYPE', y = '%Incidents to Signal Count', data =␣
 ↪INT_TYPE_TABLE)

plt.ylabel("% Incidents to Signal Count",fontsize=20)
plt.xlabel("Signal Type",fontsize=20)
plt.title("Percentatge of Incidents vs. Signal Type",fontsize=24)
```

### 3.2.5 Incident to signal count percentage in each city quadrant (Table)

[ ]:
```
QUADRANT_TABLE = accidents_vs_signals.groupby(['QUADRANT_x']).sum()
QUADRANT_TABLE = QUADRANT_TABLE[QUADRANT_TABLE.columns.
 ↪difference(['latitude','longitude','ACCESSIBLE PEDESTRIAN SIGNAL'])]
QUADRANT_TABLE['%Incidents to Signal Count'] = (QUADRANT_TABLE['Incidents'] /␣
 ↪QUADRANT_TABLE['Signal Count'])*100
QUADRANT_TABLE = QUADRANT_TABLE.sort_values(['%Incidents to Signal Count'],␣
 ↪ascending=False)
QUADRANT_TABLE=QUADRANT_TABLE.reset_index()
QUADRANT_TABLE #<-- show to see table
```

[ ]:
```
#create histogram of the above table
fig,ax = plt.subplots(figsize=(15,7))
sns.barplot(x = 'QUADRANT_x', y = '%Incidents to Signal Count', data =␣
 ↪QUADRANT_TABLE)

plt.ylabel("% Incidents to Signal Count",fontsize=20)
plt.xlabel("City Quadrant",fontsize=20)
plt.title("Incident to Signal Count Percentage in each City␣
 ↪Quadrant",fontsize=24)
```

### 3.2.6 Pedestrian button at intersection correlation (Table)

[ ]:
```
PEDBUTTON_TABLE = accidents_vs_signals.groupby(['PEDBUTTONS']).sum()
PEDBUTTON_TABLE = PEDBUTTON_TABLE[PEDBUTTON_TABLE.columns.
 ↪difference(['latitude','longitude','ACCESSIBLE PEDESTRIAN SIGNAL'])]
PEDBUTTON_TABLE['%Incidents to Signal Count'] = (PEDBUTTON_TABLE['Incidents'] /␣
 ↪PEDBUTTON_TABLE['Signal Count'])*100
PEDBUTTON_TABLE=PEDBUTTON_TABLE.reset_index()
PEDBUTTON_TABLE #<-- show to see table
```

### 3.2.7 Incident per signal count percentage for each quadrant in Calgary (BubblePlot)

[ ]:
```
x=[0,1,0,0.5,1]
y=[0,1,1,0,0]
z=[1670,1020,394,4130,1357]
plt.scatter(x,y,s=z, c="red", alpha=0.5)

plt.ylabel("South           North",fontsize=20)
plt.xlabel("West                                East",fontsize=20)
plt.title("Incident per Signal Count Percentage for each Quadrant in␣
 ↪Calgary",fontsize=24)
```

### 3.2.8 Incidents vs. signal count for each quadrant in Calgary (pointplot)

[ ]:
```
#bins = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]

fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
g = sns.pointplot(x="Signal Count", y="Incidents", data=QUADRANT_TABLE,␣
 ↪color="black")
g = sns.pointplot(x="Signal Count", y="Incidents", hue='QUADRANT_x',␣
 ↪data=QUADRANT_TABLE, scale=2)

plt.ylabel("Incidents" ,fontsize=20)
plt.xlabel("Signal Count",fontsize=20)
plt.title("Incidents vs. Signal Count for each Quadrant in Calgary",fontsize=24)
```

### 3.3 Weather Analysis.ipynb (Weather Analysis File)

[ ]:
```
import numpy as np
import pandas as pd
import folium
import geopandas as gpd
from shapely.geometry import Polygon
from shapely.geometry import box
import matplotlib.pyplot as plt
import seaborn as sns
```

#### 3.3.1 Reading and merging data

[ ]:
```
#add traffic incidents
incidents_df = pd.read_csv('..\..\CSV_files\Traffic_Incidents.csv') #get data␣
 ↪from CSV file
incidents_df['data']='TrafficIncidents' #create "data" column to record which␣
 ↪dataframe this data came from for later
#rename columns so they match other dataframes
incidents_df=incidents_df.rename(columns={'Latitude':'latitude','Longitude':
 ↪'longitude'})
#incidents_df.head() #<-- used for visual QC
```

[ ]:
```
#split Start_DT into Day/Month/Year (because only interested in 2018)
incidents_df = pd.DataFrame(incidents_df)
incidents_df['MonthDayYear']=incidents_df['START_DT'].str[:10] #re-arrage␣
 ↪columns values, and rename
#incidents_df.head() #<-- used for visual QC
```

[ ]:
```
incidents_df = incidents_df.
 ↪groupby('MonthDayYear')['Count','longitude','latitude','MonthDayYear'].sum()
incidents_df['MonthDayYear']= incidents_df.index
#incidents_df #<-- for visual QC (dont worry, there will be a python warning␣
 ↪when this code executes, that is because
#                   the index has the same title as another column in the␣
 ↪dataframe, but this column will soon
#                    me altered in later steps)
```

[ ]:
```
#Seperating the Year,Day and Month from a combined column in the DataFrame.␣
 ↪This later allows us to plot more interesting plots
incidents_df = pd.DataFrame(incidents_df)
incidents_df['Year']=incidents_df['MonthDayYear'].str[6:10]
incidents_df = incidents_df.loc[incidents_df['Year'] == '2018'] #only␣
 ↪interested in 2018
incidents_df['Day']=incidents_df['MonthDayYear'].str[3:5]
incidents_df['Month']=incidents_df['MonthDayYear'].str[:2]
```

```
#incidents_df #<-- used to visualy QC DataFrame
```

[ ]:
```
#need to change datatypes of Year, Month and Day from Object to Integer for the␣
 ↪merge of dataframes
incidents_df['Year']= incidents_df['Year'].astype(str).astype(int)
incidents_df['Month']= incidents_df['Month'].astype(str).astype(int)
incidents_df['Day']= incidents_df['Day'].astype(str).astype(int)
```

[ ]:
```
#Lets get the Weather Daily Data Now
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# returns a DataFrame with weather data from "climate.weather.gc.ca"
def download_weather_data(station, year, month=1, daily=True):

    # url to retrieve hourly data
    url_template_hourly = "https://climate.weather.gc.ca/climate_data/
 ↪bulk_data_e.html?
 ↪format=csv&stationID={station}&Year={year}&Month={month}&Day=14&timeframe=1&submit=Download

    # url to retrieve daily data
    url_template_daily = "https://climate.weather.gc.ca/climate_data/
 ↪bulk_data_e.html?
 ↪format=csv&stationID={station}&Year={year}&Month={month}&Day=14&timeframe=2&submit=Download

    # daily data by default
    if(daily == True):
        url = url_template_daily.format(station=station, year=year, month=month)

    # hourly data when (daily == False)
    else:
        url = url_template_hourly.format(station=station, year=year,␣
 ↪month=month)

    # read data into dataframe, use headers and set Date/Time column as index
    weather_data = pd.read_csv(url, index_col='Date/Time', parse_dates=True)

    # replace the degree symbol in the column names
    weather_data.columns = [col.replace('\xb0', '') for col in weather_data.
 ↪columns]

    return weather_data
```

[ ]:
```
df = download_weather_data(50430, 2018) #<-- get hourly weather
```

```
[ ]: #rename some columns to make them match other dataframes
df = df.rename(columns={"Longitude (x)":"longitude" , "Latitude (y)":
 ↪"latitude"})
Daily_Weather = df[['latitude','longitude','Station␣
 ↪Name','Year','Month','Day','Mean Temp (C)']]
Daily_Weather['data']='Weather'
#Daily_Weather <-- visual QC
```

```
[ ]: Hourly_Weather = download_weather_data(50430, 2018, daily=False) # <-- get␣
 ↪hourly Data (visibility)
#rename some columns to make them match other dataframes, also add column to␣
 ↪record where this data came from for after merge
Hourly_Weather = Hourly_Weather.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather['data']='Weather'
#Hourly_Weather.head() #<-- visual QC
```

```
[ ]: #groupby.mean() on "Day" column, to get the average visibulity for each day␣
 ↪(daily data was requested for thsi report)
Hourly_Weather = Hourly_Weather.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()
#Hourly_Weather.head() #<-- will get a warning because index has same name as a␣
 ↪column, but it is okay for now.
```

```
[ ]: #continue with above strategy, pulling in all weather data for each month
Hourly_Weather2 = download_weather_data(50430, 2018, month = 2, daily=False)
Hourly_Weather2 = Hourly_Weather2.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather2['data']='Weather'
Hourly_Weather2 = Hourly_Weather2.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather3 = download_weather_data(50430, 2018, month = 3, daily=False)
Hourly_Weather3 = Hourly_Weather3.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather3['data']='Weather'
Hourly_Weather3 = Hourly_Weather3.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather4 = download_weather_data(50430, 2018, month = 4, daily=False)
Hourly_Weather4 = Hourly_Weather4.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather4['data']='Weather'
```

```
Hourly_Weather4 = Hourly_Weather4.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather5 = download_weather_data(50430, 2018, month = 5, daily=False)
Hourly_Weather5 = Hourly_Weather5.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather5['data']='Weather'
Hourly_Weather5 = Hourly_Weather5.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather6 = download_weather_data(50430, 2018, month = 6, daily=False)
Hourly_Weather6 = Hourly_Weather6.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather6['data']='Weather'
Hourly_Weather6 = Hourly_Weather6.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather7 = download_weather_data(50430, 2018, month = 7, daily=False)
Hourly_Weather7 = Hourly_Weather7.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather7['data']='Weather'
Hourly_Weather7 = Hourly_Weather7.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather8 = download_weather_data(50430, 2018, month = 8, daily=False)
Hourly_Weather8 = Hourly_Weather8.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather8['data']='Weather'
Hourly_Weather8 = Hourly_Weather8.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather9 = download_weather_data(50430, 2018, month = 9, daily=False)
Hourly_Weather9 = Hourly_Weather9.rename(columns={"Longitude (x)":"longitude" ,␣
 ↪"Latitude (y)":"latitude"})
Hourly_Weather9['data']='Weather'
Hourly_Weather9 = Hourly_Weather9.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather10 = download_weather_data(50430, 2018, month = 10, daily=False)
```

```
Hourly_Weather10= Hourly_Weather10.rename(columns={"Longitude (x)":"longitude"␣
 ↪, "Latitude (y)":"latitude"})
Hourly_Weather10['data']='Weather'
Hourly_Weather10 = Hourly_Weather10.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather11 = download_weather_data(50430, 2018, month = 11, daily=False)
Hourly_Weather11 = Hourly_Weather11.rename(columns={"Longitude (x)":"longitude"␣
 ↪, "Latitude (y)":"latitude"})
Hourly_Weather11['data']='Weather'
Hourly_Weather11 = Hourly_Weather11.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

Hourly_Weather12 = download_weather_data(50430, 2018, month = 12, daily=False)
Hourly_Weather12 = Hourly_Weather12.rename(columns={"Longitude (x)":"longitude"␣
 ↪, "Latitude (y)":"latitude"})
Hourly_Weather12['data']='Weather'
Hourly_Weather12 = Hourly_Weather12.
 ↪groupby('Day')['longitude','latitude','Visibility␣
 ↪(km)','Year','Month','Day'].mean()

#combined all data
hourly_Weather_Total = pd.
 ↪concat([Hourly_Weather,Hourly_Weather2,Hourly_Weather3,Hourly_Weather4,Hourly_Weather5,\
                                  Hourly_Weather6,Hourly_Weather7,␣
 ↪Hourly_Weather8, Hourly_Weather9, Hourly_Weather10,\
                                 Hourly_Weather11, Hourly_Weather12])
#hourly_Weather_Total
```

[ ]:
```
#JOIN HOURLY WEATHER TO DAILY WEATHER
Daily_Weather.index.name=None
hourly_Weather_Total.index.name=None
total_weather = pd.merge(left=hourly_Weather_Total, right=Daily_Weather,␣
 ↪how='outer', left_on=['Year','Month','Day'], right_on=['Year','Month','Day'])
#total_weather.head() #<-- QC check
```

### 3.3.2 Table of general statistics of temperature and visibility (Table)

```
[ ]: analysis = total_weather[['Visibility (km)','Mean Temp (C)']] # only keep␣
     ↪columns we are interested in
     analysis.describe() # <-- unhide to see table of temperature and visibility␣
     ↪statistics in 2018
```

```
[ ]: #combine Total Weather with collition Data
     Colition_weather_df = pd.merge(left=total_weather, right=incidents_df,␣
     ↪how='left', left_on=['Year','Month','Day'], right_on=['Year','Month','Day'])
     Colition_weather_df=Colition_weather_df.reset_index()
     #Colition_weather_df #<-- visual QC
     #Colition_weather_df.to_csv("C:/Users/adamd/Desktop/WeatherIncidents.csv") <-␣
     ↪hard copy QC
```

### 3.3.3 Number of days at each temperature (Histogram)

```
[ ]: bins = [-30,-25,-20,-15,-10,-5,0,5,10,15,20,25,30]

     fig,ax = plt.subplots(figsize=(15,7))
     sns.distplot(Daily_Weather['Mean Temp (C)'],kde=False , bins = bins,␣
     ↪hist_kws={"rwidth":0.8,'edgecolor':'black', 'alpha':1.0} )
     plt.ylabel("Days at Temperature",fontsize=20)
     plt.xlabel("Temperature (C)",fontsize=20)
     plt.title("Number of days at each Temperature",fontsize=24)
```

### 3.3.4 Daily incidents vs. binned mean temperature (C) (PointPlot)

```
[ ]: # Your solution goes here
     bins = [-30,-25,-20,-15,-10,-5,0,5,10,15,20,25,30]
     Colition_weather_df['Temp Binned'] = pd.cut(Colition_weather_df['Mean Temp␣
     ↪(C)'], bins=bins)

     fig,ax = plt.subplots(figsize=(15,7))
     g = sns.pointplot(x="Temp Binned", y="Count", data=Colition_weather_df , ci =␣
     ↪None)
     g.set(ylim=(0, 50))

     plt.ylabel("Daily Incidents",fontsize=20)
     plt.xlabel("Binned Mean Temperature (C)",fontsize=20)
     plt.title("Daily Incidents vs. Binned Mean Temperature (C)",fontsize=24)
     #Even though it looks like more colisions might happen around the temperatures␣
     ↪of -10 and +10, it is only because there are
     #more days at those temperatures (see above). However % wise, there are very␣
     ↪few days netween -20 and -20 C, yes we still have
     #on average 13 to 16 incidents
```

### 3.3.5 Mean temperature (C) per day (PointPlot)

[ ]:
```
fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
g = sns.pointplot(x="index", y="Mean Temp (C)", data=Colition_weather_df,␣
↪ci=None)
ax.xaxis.set_major_formatter(plt.NullFormatter())

plt.ylabel("Mean Temperature (C)",fontsize=20)
plt.xlabel("Jan        Feb        Mar        Apr        May        Jun      ␣
↪Jul       Aug       Sep       Oct       Nov       Dec",fontsize=20)
plt.title("Mean Temperature (C) per Day",fontsize=24)
```

### 3.3.6 Mean temperature (C) per month (PointPlot)

[ ]:
```
#PLotting some daily Weather Conditions (Temp and Visibility)
fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
g = sns.pointplot(x="Month", y="Mean Temp (C)", data=Colition_weather_df,␣
↪ci=None)

plt.ylabel("Mean Temperature (C)",fontsize=20)
plt.xlabel("Month",fontsize=20)
plt.title("Mean Temperature (C) per Month",fontsize=24)
```

### 3.3.7 Daily incidents per month (PointPlot)

[ ]:
```
#PLotting some daily Weather Conditions (Temp and Visibility)
fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
g = sns.pointplot(x="Month", y="Count", data=Colition_weather_df, ci=None)

plt.ylabel("Daily Incidents",fontsize=20)
plt.xlabel("Month",fontsize=20)
plt.title("Daily Incidents per Month",fontsize=24)
```

### 3.3.8 Average visibility (km) per day (PointPlot)

[ ]:
```
fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
g = sns.pointplot(x="index", y="Visibility (km)", data=Colition_weather_df,␣
 ↪ci=None)

ax.xaxis.set_major_formatter(plt.NullFormatter())

plt.ylabel("Visibility (km)",fontsize=20)
plt.xlabel("Jan        Feb        Mar        Apr        May        Jun      ␣
 ↪Jul       Aug       Sep       Oct       Nov       Dec",fontsize=20)
plt.title("Visibility (km)  per Day",fontsize=24)
```

### 3.3.9 Average visibility (km) per month (PointPlot)

[ ]:
```
fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
g = sns.pointplot(x="Month", y="Visibility (km)", data=Colition_weather_df,␣
 ↪ci=None )

plt.ylabel("Average Vsisibility (km)",fontsize=20)
plt.xlabel("Month",fontsize=20)
plt.title("Average Visibility (km) per Month",fontsize=24)
```

[ ]:
```
#Needed to calculate min and max of visibility data, this will be used to help␣
 ↪determine bin sizes for following figures
maximum = Colition_weather_df['Visibility (km)'].max()
minimum = Colition_weather_df['Visibility (km)'].min()
#print(maximum, minimum) #<-- visual QC
```

### 3.3.10 Daily incidents vs visibility (km) (PointPlot)

[ ]:
```
#create plot of Daily Incidents vs. Average Visibility
Colition_weather_df= Colition_weather_df.sort_values(by=['Visibility (km)'])

labels = ['0 to 2km','2 to 4km','4 to 6km','6 to 8km','8 to 10km','10 to␣
 ↪15km','15 to 20km','20 to 25km','25 to 30km','30 to 35km','35 to 40km','40␣
 ↪to 45km','45 to 50km','50 to 55km']
bins = [0,2,4,6,8,10,15,20,25,30,35,40,45,50,55]

Colition_weather_df['visibility_binned']= pd.
 ↪cut(Colition_weather_df['Visibility (km)'], bins=bins, labels=labels)

fig,ax = plt.subplots(figsize=(15,7))
sns.set(style="darkgrid")
```

```
g = sns.pointplot(x="visibility_binned", y="Count", data=Colition_weather_df,␣
↪ci=None).set_title('Road Incidents vs. Road Visiblity (km)')

plt.ylabel("Daily Incidents",fontsize=20)
plt.xlabel("Average Visibility (km)",fontsize=20)
plt.title("Daily Incidents vs Visibility (km)",fontsize=24)
```

[ ]:
```
#add traffic incidents (but first need to make columns match the weather data␣
↪column formats)
incidents_df = pd.read_csv('..\..\CSV_files\Traffic_Incidents.csv')
incidents_df.head(10)
incidents_df['data']='TrafficIncidents'
#rename columns so they match other dataframes
incidents_df=incidents_df.rename(columns={'Latitude':'latitude','Longitude':
↪'longitude'})
incidents_df=pd.DataFrame(incidents_df)
incidents_df['Time'] = incidents_df['START_DT'].str[10:]
#incidents_df.head() #<-- visual QC
```

[ ]:
```
#split the Time from the hours in the Incident Data. We need to make the␣
↪columns match the weather data so we can combined them
import numpy as np
incidents_df['Time'].str[10:]
incidents_df['Time'].str[:3]
incidents_df['night/day'] = incidents_df['Time'].str[10:]
incidents_df['Hour'] = incidents_df['Time'].str[:3]
incidents_df['Hour'] = incidents_df['Hour'].astype(int)
#incidents_df.head()#<-- visual QC
```

[ ]:
```
#need to change PM/AM time into 24hour clock time, so that we can plot them␣
↪easier on a graph on the X-axis
#create a column for the addition to the current time: 0 for AM's, and +12 for␣
↪PM times:
incidents_df["temp"] = incidents_df["night/day"].map(lambda x: '0' if "AM" in x␣
↪else '12' if "PM" in x else "")
incidents_df['temp'] = incidents_df['temp'].astype(int) #change type to int, so␣
↪we can add columns together
#incidents_df.head() #<-- visual QC
```

[ ]:
```
incidents_df['24HourClock']=incidents_df['Hour']+incidents_df['temp'] #add␣
↪columns to get 24hour clock time
#incidents_df.head()#<-- visual QC
```

### 3.3.11 Total incidents in 2018, for each hour of the day (Histogram)

[ ]:
```
#Finaly, plot total incidents vs. time of day
bins = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]

fig,ax = plt.subplots(figsize=(15,7))
sns.distplot(incidents_df['24HourClock'],kde=False , bins = bins,␣
 ↪hist_kws={"rwidth":0.8,'edgecolor':'black', 'alpha':1.0} )
plt.ylabel("Total Incidents",fontsize=20)
plt.xlabel("Time of Day (24hour clock)",fontsize=20)
plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24])
plt.title("Total Incidents in 2018, For Each Hour of the Day",fontsize=24)
```

## 3.4 stats.ipynb (computing grid data file)

### 3.4.1 Libraries and functions

[ ]:
```
import numpy as np
import pandas as pd
import folium
import geopandas as gpd
from shapely.geometry import Polygon
from shapely.geometry import box
```

### 3.4.2 Creating 10x10 grid

[ ]:
```
df_city_boundary_layer = pd.read_csv('..\..\CSV_files\City_Boundary_Layer.csv')
```

[ ]:
```
string = df_city_boundary_layer["the_geom"].values[0]
string_stripped= string.replace("POLYGON","").replace("(","").replace(")","").
 ↪replace(",","")
string_split = string_stripped.split()
```

[ ]:
```
list_long = []
list_lat = []

#appends latitudes and longitudes based on long/lat/long/lat... pattern
for i in range(len(string_split)):
    if i % 2 == 0:
        list_long.append(float(string_split[i]))
    else:
        list_lat.append(float(string_split[i]))

min_long = min(list_long)
max_long = max(list_long)
min_lat = min(list_lat)
max_lat = max(list_lat)
```

[ ]:
```
coord_box = box(min_long,min_lat,max_long,max_lat)
geo = gpd.GeoSeries([coord_box]).__geo_interface__
```

[ ]:
```
map_osm = folium.Map(location=[min_lat,min_long], zoom_start=10)
folium.GeoJson(geo).add_to(map_osm)
```

[ ]:
```
map_osm
```

[ ]:
```
long_ten_split = np.linspace(min_long,max_long,num = 11)
lat_ten_split = np.linspace(min_lat,max_lat,num = 11)
```

[ ]:
```
grid_array = []

for y in range(len(lat_ten_split)-1):
    for x in range(len(long_ten_split)-1):
        bot_left = [long_ten_split[x],lat_ten_split[y]]
        bot_right = [long_ten_split[x+1],lat_ten_split[y]]
        top_left = [long_ten_split[x],lat_ten_split[y+1]]
        top_right = [long_ten_split[x+1],lat_ten_split[y+1]]
        grid_array.append([bot_left,bot_right,top_left,top_right])

        coord_box = box(bot_left[0],bot_left[1],top_right[0],top_right[1])
        geo = gpd.GeoSeries([coord_box]).__geo_interface__
        folium.GeoJson(geo).add_to(map_osm)

map_osm
```

### 3.4.3 Calculations for each 1x1 grid

[ ]:
```
# This method deals with csv files containing multistring longitude/latitude

# Iterate through each line of the csv file. Parse the multistring from the csv␣
 ↪file to extract the values below for each row and store each value in the␣
 ↪corresponding list:
# minimum longitude
# minimum latitude
# maximum longitude
# maximum latitude

# Check if a coordinate is within each 1x1 grid.

# Return a list of matching speed limit or volume for each 1x1 grid, depending␣
 ↪on the parameters passed.


def compute_average_with_multistring(fileName, columnName, stringName,␣
 ↪columnName2, grid_array):

    df = pd.read_csv(fileName)

    list_long = []
    list_lat = []

    min_long = []
    max_long = []
    min_lat = []
    max_lat = []

    for index, row in df.iterrows():
        # parsing "multiline" column to extract longitude/latitude
        message1 = df[columnName].values[index]
        message1 = message1.replace(stringName, '').replace('(', '').
 ↪replace(',', '').replace(')', '')
        message1 = message1.split()

        # the multiline string follows longitude, latitude, longitude, latitude.
 ↪..... pattern
        for i in range(len(message1)):
            if i % 2 == 0:
                list_long.append(float(message1[i]))
            else:
                list_lat.append(float(message1[i]))

        # finding min/max longitude/latitude for each row
```

```
        min_long.append(min(list_long))
        max_long.append(max(list_long))
        min_lat.append(min(list_lat))
        max_lat.append(max(list_lat))

    # consist of 100 lists corresponding to the 100 grids
    # each individual list stores the extracted speed limit if the coordinate␣
↪is within the grid
    resultList = [[] for _ in range(100)]

    for index, row in df.iterrows():
        for i in range(len(grid_array)):

            # To reduce search time:
            # 1. Check if (minimum longitude, minimum latitude) is greater than␣
↪the top right coordinate of the 1x1 grid.
            # 2. Check if (maximum longitude, maximum latitude) is smaller than␣
↪the bottom left coordinate of the 1x1 grid.
            # If the above condition meets, none of the coordinates from the␣
↪current row of the multiline column would lie within the current 1x1 grid.

            # For each 1x1 grid:
            # grid_array[i][3][0] = longitude of top right coordinate
            # grid_array[i][3][1] = latitude of top right coordinate
            # grid_array[i][0][0] = longitude of bottom left coordinate
            # grid_array[i][0][1] = latitude of bottom left coordinate

            if((min_long[index] > grid_array[i][3][0] and min_lat[index] >␣
↪grid_array[i][3][1]) or (max_long[index] < grid_array[i][0][0] and␣
↪max_lat[index] < grid_array[i][0][1])):
                break

            # If the above condition does not meet, there is the possibility of␣
↪having coordinates lie within the current 1x1 grid. We would need to␣
↪continue our search.
            else:
                # parsing "multiline" column to extract longitude/latitude
                message1 = df[columnName].values[index]
                message1 = message1.replace(stringName, '').replace('(', '').
↪replace(',', '').replace(')', '')
                message1 = message1.split()

                # converting to float type
                for x in range(len(message1)):
                    message1[x] = float(message1[x])
```

```
                # If one of the coordinate from the multistring meets the␣
↪condition, we break outside the loop and check the next row.
                # This means that we only consider each road once within each␣
↪grid.
                # For example, if a road appears 5 times within 1 grid, we only␣
↪count it once.
                for j in range(0, len(message1), 2):
                    if((grid_array[i][0][0] <= message1[j]) and (message1[j] <=␣
↪grid_array[i][3][0]) and (grid_array[i][0][1] <= message1[j+1]) and␣
↪(message1[j+1] <= grid_array[i][3][1])):
                        resultList[i].append(df[columnName2].values[index])
                        break
                    else:
                        continue
    return resultList
```

[ ]:
```
# This method deals with csv files containing only 1 value for longitude/
↪latitude

# Iterate through each line of the csv file.

# Check if a coordinate is within each 1x1 grid.

# Return a list of results for each 1x1 grid.

def compute_result(df, columnLong, columnLat, grid_array):

    resultList = [[] for _ in range(100)]

    for index, row in df.iterrows():
        for i in range(len(grid_array)):
            if((grid_array[i][0][0] <= df[columnLong][index]) and␣
↪(df[columnLong][index] <= grid_array[i][3][0]) and (grid_array[i][0][1] <=␣
↪df[columnLat][index]) and (df[columnLat][index] <= grid_array[i][3][1])):
                resultList[i].append(1)
            else:
                continue

    return resultList
```

### 3.4.4 Computing average speed limit for each area/grid

```
[ ]: # convert to dataframe
df_average_speed_limit = pd.DataFrame(compute_average_with_multistring('..\..
 ↪\CSV_files\Speed_Limits.csv', 'multiline', 'MULTILINESTRING', 'SPEED',␣
 ↪grid_array))
# compute average
df_average_speed_limit['Average Speed Limit'] = df_average_speed_limit.
 ↪mean(axis=1)
df_average_speed_limit = df_average_speed_limit[['Average Speed Limit']].copy()
```

### 3.4.5 Computing average traffic volume for each area/grid

```
[ ]: # convert to dataframe
df_average_traffic_volumes = pd.DataFrame(compute_average_with_multistring('..\.
 ↪.\CSV_files\Traffic_Volumes_for_2018.csv', 'multilinestring',␣
 ↪'MULTILINESTRING', 'VOLUME', grid_array))
# compute average
df_average_traffic_volumes['Average Traffic Volume'] =␣
 ↪df_average_traffic_volumes.mean(axis=1)
df_average_traffic_volumes = df_average_traffic_volumes[['Average Traffic␣
 ↪Volume']].copy()
```

### 3.4.6 Computing traffic cameras for each area/grid

```
[ ]: df_traffic_cameras = pd.read_csv('..\..\CSV_files\Traffic_Camera_Locations.csv')
# convert to dataframe
df_traffic_cameras = pd.DataFrame(compute_result(df_traffic_cameras,␣
 ↪'longitude', 'latitude', grid_array))
# compute sum
df_traffic_cameras['Traffic Cameras'] = df_traffic_cameras.sum(axis=1)
df_traffic_cameras = df_traffic_cameras[['Traffic Cameras']].copy()
```

### 3.4.7 Computing traffic signals for each area/grid

```
[ ]: df_traffic_signals = pd.read_csv('..\..\CSV_files\Traffic_Signals.csv')
# convert to dataframe
df_traffic_signals = pd.DataFrame(compute_result(df_traffic_signals,␣
 ↪'longitude', 'latitude', grid_array))
# compute sum
df_traffic_signals['Traffic Signals'] = df_traffic_signals.sum(axis=1)
df_traffic_signals = df_traffic_signals[['Traffic Signals']].copy()
```

### 3.4.8 Computing traffic signs for each area/grid

[ ]:
```
df_traffic_signs = pd.read_csv('..\..\CSV_files\Traffic_Signs.csv')

# extracting relevant signs
df_traffic_signs = df_traffic_signs.loc[(df_traffic_signs['BLADE_TYPE'] ==␣
↪'Regulatory') | (df_traffic_signs['BLADE_TYPE'] == 'Warning') |␣
↪(df_traffic_signs['BLADE_TYPE'] == 'Stop') | (df_traffic_signs['BLADE_TYPE']␣
↪== 'Playground') | (df_traffic_signs['BLADE_TYPE'] == 'Yield') |␣
↪(df_traffic_signs['BLADE_TYPE'] == 'Pedestrian') |␣
↪(df_traffic_signs['BLADE_TYPE'] == 'Disabled Parking') |␣
↪(df_traffic_signs['BLADE_TYPE'] == 'Speed') |␣
↪(df_traffic_signs['BLADE_TYPE'] == 'Bicycle / Pathway') |␣
↪(df_traffic_signs['BLADE_TYPE'] == 'School')]

# reset index
df_traffic_signs = df_traffic_signs.reset_index(drop=True)
```

[ ]:
```
signs_per_grid = [[] for _ in range(100)]

for index, row in df_traffic_signs.iterrows():
    for i in range(len(grid_array)):
            message1 = df_traffic_signs['POINT'].values[index]
            message1 = message1.replace('POINT', '').replace('(', '').
↪replace(')', '')
            message1 = message1.split()

            for x in range(2):
                message1[x] = float(message1[x])

            if((grid_array[i][0][0] <= message1[0]) and (message1[0] <=␣
↪grid_array[i][3][0]) and (grid_array[i][0][1] <= message1[1]) and␣
↪(message1[1] <= grid_array[i][3][1])):
                signs_per_grid[i].append(1)
                break
            else:
                continue

df_traffic_signs = pd.DataFrame(signs_per_grid)
# convert to dataframe
df_traffic_signs['Traffic Signs'] = df_traffic_signs.sum(axis=1)
# compute sum
df_traffic_signs = df_traffic_signs[['Traffic Signs']].copy()
```

### 3.4.9 Computing traffic incidents for each area/grid

```
[ ]: df_traffic_incidents = pd.read_csv('..\..\CSV_files\Traffic_Incidents.csv')
     # extract 2018 incidents
     df_traffic_incidents = df_traffic_incidents[df_traffic_incidents['id'].str.
      ↪startswith(str(2018))]

     # convert to dataframe
     df_traffic_incidents = pd.DataFrame(compute_result(df_traffic_incidents,␣
      ↪'Longitude', 'Latitude', grid_array))
     # compute sum
     df_traffic_incidents['Traffic Incidents'] = df_traffic_incidents.sum(axis=1)
     df_traffic_incidents = df_traffic_incidents[['Traffic Incidents']].copy()
```

### 3.4.10 Combining results into 1 dataframe

```
[ ]: # index column corresponds to the 100 grids:
     # from bottom left (grid 0) to top right (grid 99)
     df = pd.concat([df_average_speed_limit, df_average_traffic_volumes,␣
      ↪df_traffic_cameras, df_traffic_signals, df_traffic_signs,␣
      ↪df_traffic_incidents], axis=1).reindex(df_average_speed_limit.index)
     df
```

## 3.5 graphs_Ivan.ipynb (figure creation from grid data file)

### 3.5.1 Libraries and functions

[ ]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

### 3.5.2 Read combined data csv (containing grid data)

[ ]:
```
df = pd.read_csv('..\..\CSV_files\Combined_Data.csv')
```

### 3.5.3 Average speed limit vs. incident/volume ratio

[ ]:
```
sns.set(style="white", rc={"lines.linewidth": 3})
fig, ax1 = plt.subplots(figsize=(20,20))
ax2 = ax1.twinx()

sns.barplot(x='Grid', y='Average Speed Limit', data=df, color='cyan', ax=ax1)
sns.lineplot(x='Grid', y='Incidents/Volume Ratio', color='r', data=df, ax=ax2)

plt.locator_params(axis='x', nbins=20)

plt.show()
sns.set()
```

### 3.5.4 Traffic signals vs. incident/volume ratio

[ ]:
```
sns.set(style="white", rc={"lines.linewidth": 3})
fig, ax1 = plt.subplots(figsize=(20,20))
ax2 = ax1.twinx()

sns.barplot(x='Grid', y='Traffic Signals', data=df, color='cyan', ax=ax1)
sns.lineplot(x='Grid', y='Incidents/Volume Ratio', color='r', data=df, ax=ax2)

plt.locator_params(axis='x', nbins=20)

plt.show()
sns.set()
```

### 3.5.5 Traffic cameras vs. incident/volume ratio

[ ]:
```
sns.set(style="white", rc={"lines.linewidth": 3})
fig, ax1 = plt.subplots(figsize=(20,20))
ax2 = ax1.twinx()

sns.barplot(x='Grid', y='Traffic Cameras', data=df, color='cyan', ax=ax1)
sns.lineplot(x='Grid', y='Incidents/Volume Ratio', color='r', data=df, ax=ax2)

plt.locator_params(axis='x', nbins=20)

plt.show()
sns.set()
```

### 3.5.6 Traffic signs vs. incident/volume ratio

[ ]:
```
sns.set(style="white", rc={"lines.linewidth": 3})
fig, ax1 = plt.subplots(figsize=(20,20))
ax2 = ax1.twinx()

sns.barplot(x='Grid', y='Traffic Signs', data=df, color='cyan', ax=ax1)
sns.lineplot(x='Grid', y='Incidents/Volume Ratio', color='r', data=df, ax=ax2)

plt.locator_params(axis='x', nbins=20)

plt.show()
sns.set()
```

## 3.6 spearman.ipynb (spearman calculating file)

### 3.6.1 Libraries and functions

[ ]:
```
import pandas as pd
```

[ ]:
```
#creates a list of rankings from a given list
def ranking(input):
    #saves the location of the items in the original list
    value_index_pair = list(zip(input,range(len(input))))

    #sorts the list by the value
    value_index_pair_sorted = sorted(value_index_pair)

    ranks = [0]*len(input)
    #ranks the items in the list and stores it in the original location
    for i, item in enumerate(value_index_pair_sorted):
        #print(i,item)
        ranks[item[1]] = i+1

    return ranks
```

[ ]:
```
#computes the spearman coefficient given two ranking lists
def compute_spearman_ranks(x_ranks,y_ranks):
    n = len(x_ranks)

    d_square = []

    #d^2 for each row of data ranks input
    for x,y in zip(x_ranks,y_ranks):
        diff = x - y
        d_square.append(diff**2)

    #sum of d^2 terms
    sum_d_square = sum(d_square)

    #calculate spearman coefficient
    rs = 1 - (6*sum_d_square/(n*(n**2-1)))
    return rs
```

### 3.6.2 Read csv files

```
[ ]: #read combined csv file (grid data)
combined_df = pd.read_csv("..\..\CSV_files\Combined_Data.csv")

#drop nan values in traffic volume
combined_dropna_df = combined_df[combined_df["Average Traffic Volume"].notna()]
combined_dropna_df["Incidents/Volume Ratio"] = combined_dropna_df["Total␣
↪Incidents"]/combined_dropna_df["Average Traffic Volume"]

#read weather data with incident count
weather_df = pd.read_csv("..\..\CSV_files\WeatherIncidents.csv")
```

### 3.6.3 Calculate and output spearman coefficients

```
[ ]: #calculate incident ranking normalizing for traffic volume (for grid data) (x␣
↪value)
incident_volume = list(combined_dropna_df["Incidents/Volume Ratio"])
incident_volume_rank = ranking(incident_volume)

#calculate incident ranking normalizing for daily rate(for weather data) (x␣
↪value)
incident_count = list(weather_df["Count"])
incident_count_rank = ranking(incident_count)

#calculate camera number ranking (y value) and spearman coefficient
cameras = list(combined_dropna_df["Total Cameras"])
cameras_rank = ranking(cameras)
cameras_spear = compute_spearman_ranks(cameras_rank, incident_volume_rank)

#calculate signals number ranking (y value) and spearman coefficient
signals = list(combined_dropna_df["Total Signals"])
signals_rank = ranking(signals)
signals_spear = compute_spearman_ranks(signals_rank, incident_volume_rank)

#calculate signs number ranking (y value) and spearman coefficient
signs = list(combined_dropna_df["Total Signs"])
signs_rank = ranking(signs)
signs_spear = compute_spearman_ranks(signs_rank, incident_volume_rank)

#remove nan values in average speed
combined_dropna_df = combined_dropna_df[combined_dropna_df["Average Speed␣
↪Limit"].notna()]
incident_volume = list(combined_dropna_df["Incidents/Volume Ratio"])
incident_volume_rank = ranking(incident_volume)

#calculate speed number ranking (y value) and spearman coefficient
```

```
speed = list(combined_dropna_df["Average Speed Limit"])
speed_rank = ranking(speed)
speed_spear = compute_spearman_ranks(speed_rank, incident_volume_rank)

#calculate visibility ranking (y value) and spearman coefficient
visib = list(weather_df["Visibility (km)"])
visib_rank = ranking(visib)
visib_spear = compute_spearman_ranks(visib_rank, incident_volume_rank)

#calculate temperature ranking (y value) and spearman coefficient
temp = list(weather_df["Mean Temp (C)"])
temp_rank = ranking(temp)
temp_spear = compute_spearman_ranks(temp_rank, incident_volume_rank)
```

[ ]:
```
#grid data spearman coefficients
print("Grid data")
print("Spearman correlation against incident/volume")
print("Cameras:",cameras_spear)
print("Signals:",signals_spear)
print("Signs:",signs_spear)
print("Speed:",speed_spear, "\n")

#weather data spearman coefficients
print("Weather data")
print("Spearman correlation against average daily incidents")
print("Visibility:",visib_spear)
print("Temperature:",temp_spear)
```

## 3.7 mapping_all.ipynb (mapping file)

### 3.7.1 Libraries and functions

[ ]:
```
#import required libraries
import numpy as np
import pandas as pd
import folium
import geopandas as gpd
from folium import plugins
from folium.plugins import HeatMap
from folium.plugins import FloatImage
from shapely.geometry import box
from shapely.geometry import MultiLineString
import branca
from colour import Color
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from PIL import Image
```

[ ]:
```
#CODE BORROWED FROM https://nbviewer.jupyter.org/gist/BibMartin/
 ↪f153aa957ddc5fadc64929abdee9ff2e
from branca.element import MacroElement

from jinja2 import Template

class BindColormap(MacroElement):
    """Binds a colormap to a given layer.

    Parameters
    ----------
    colormap : branca.colormap.ColorMap
        The colormap to bind.
    """
    def __init__(self, layer, colormap):
        super(BindColormap, self).__init__()
        self.layer = layer
        self.colormap = colormap
        self._template = Template(u"""
        {% macro script(this, kwargs) %}
            {{this.colormap.get_name()}}.svg[0][0].style.display = 'block';
            {{this._parent.get_name()}}.on('overlayadd', function (eventLayer) {
                if (eventLayer.layer == {{this.layer.get_name()}}) {
                    {{this.colormap.get_name()}}.svg[0][0].style.display =␣
 ↪'block';
                }});
            {{this._parent.get_name()}}.on('overlayremove', function␣
 ↪(eventLayer) {
```

```
                if (eventLayer.layer == {{this.layer.get_name()}}) {
                    {{this.colormap.get_name()}}.svg[0][0].style.display =␣
↪'none';
                }});
        {% endmacro %}
        """)  # noqa
```

[ ]:
```
#function for creating heatmap given lat, long and weight data
def create_heatmap(list_lat, list_long, map_osm, heat_name, list_weight = [], r␣
↪= 20):
    #create feature layer
    feature = folium.map.FeatureGroup(name = heat_name, overlay = True, show =␣
↪False)

    #if weights are given
    if (len(list_weight) > 0):
        #create heatmap
        plugins.HeatMap(zip(list_lat, list_long, list_weight), radius = r,␣
↪gradient = {0.25: "blue", 0.4: "lightblue", 0.6: "lime", 0.8: "yellow", 1.0:␣
↪"red"}).add_to(feature)

        #create legend
        legend = create_legend(list_weight)

        #add feature to map and bind feature and legend together
        map_osm.add_child(feature)
        map_osm.add_child(legend)
        map_osm.add_child(BindColormap(feature,legend))

    #otherwise don't use any weights
    else:
        #create heatmap
        plugins.HeatMap(zip(list_lat, list_long), radius = r, gradient = {0.25:␣
↪"blue", 0.4: "lightblue", 0.6: "lime", 0.8: "yellow", 1.0: "red"}).
↪add_to(feature)

        #create legend
        legend = create_legend()

        #add feature to map and bind feature and legend together
        map_osm.add_child(feature)
        map_osm.add_child(legend)
        map_osm.add_child(BindColormap(feature,legend))
```

[ ]:
```
#function for creating heatmap given lat, long and weight data (specifically␣
↪for incidents)
```

```
def create_heatmap_incidents(list_lat, list_long, map_osm, heat_name,␣
↪list_weight = [], r = 20):
    #create feature layer
    feature = folium.map.FeatureGroup(name = heat_name, overlay = True, show =␣
↪False)

    #if weights are given
    if (len(list_weight) > 0):
        #create heatmap
        #inferno color scheme (reversed)
        plugins.HeatMap(zip(list_lat, list_long), radius = r, gradient = {0.05:␣
↪"#e4fa15", 0.142857143: "#f6bd27", 0.285714286: "#ee8949",  0.428571429:␣
↪"#d85b69", 0.571428571: "#ba2f8a", 0.714285714: "#9200a6", 0.857142857:␣
↪"#6200a4", 1.0: "#2f0087"}).add_to(feature)

        #create legend
        legend = create_legend_incidents(list_weight)

        #add feature to map and bind feature and legend together
        map_osm.add_child(feature)
        map_osm.add_child(legend)
        map_osm.add_child(BindColormap(feature,legend))

    #otherwise don't use any weights
    else:
        #create heatmap
        #inferno color scheme (reversed)
        plugins.HeatMap(zip(list_lat, list_long), radius = r, gradient = {0.05:␣
↪"#e4fa15", 0.142857143: "#f6bd27", 0.285714286: "#ee8949",  0.428571429:␣
↪"#d85b69", 0.571428571: "#ba2f8a", 0.714285714: "#9200a6", 0.857142857:␣
↪"#6200a4", 1.0: "#2f0087"}).add_to(feature)

        #create legend
        legend = create_legend_incidents()

        #add feature to map and bind feature and legend together
        map_osm.add_child(feature)
        map_osm.add_child(legend)
        map_osm.add_child(BindColormap(feature,legend))
```

[ ]:
```
#function for generating legend (returns a linear interpolated color gradient)
def create_legend(list_weight = []):
    #colors and their respective index values
    colors = ["blue", "lightblue", "lime", "yellow", "red"]
    index = [0.25, 0.4, 0.6, 0.8, 1.0]

    #if weights are given, scale the index
```

```
    if (len(list_weight) > 0):
        #scale index with weight values
        max_weight = max(list_weight)
        min_weight = min(list_weight)
        diff_weight = max_weight - min_weight
        index = [(i*diff_weight) + min_weight for i in index]

        #create a linear interpolation of colors
        colormap = branca.colormap.LinearColormap(colors = colors, index =␣
↪index, vmin = min_weight, vmax = max_weight)

    #else, use standard index
    else:
        colormap = branca.colormap.LinearColormap(colors = colors, index =␣
↪index)

    return colormap
```

[ ]:
```
#function for generating legend (returns a linear interpolated color gradient)␣
↪(specifically for incidents)
def create_legend_incidents(list_weight = []):
    #colors and their respective index values
    # colors = ["#2f0087", "#6200a4", "#9200a6", "#ba2f8a", "#d85b69",␣
↪"#ee8949", "#f6bd27", "#e4fa15"]
    colors = ["#e4fa15", "#f6bd27", "#ee8949", "#d85b69", "#ba2f8a", "#9200a6",␣
↪"#6200a4", "#2f0087"]
    index = [0.05, 0.142857143, 0.285714286, 0.428571429, 0.571428571, 0.
↪714285714, 0.857142857, 1.0]

    #if weights are given, scale the index
    if (len(list_weight) > 0):
        #scale index with weight values
        max_weight = max(list_weight)
        min_weight = min(list_weight)
        diff_weight = max_weight - min_weight
        index = [(i*diff_weight) + min_weight for i in index]

        #create a linear interpolation of colors
        colormap = branca.colormap.LinearColormap(colors = colors, index =␣
↪index, vmin = min_weight, vmax = max_weight)

    #else, use standard index
    else:
        colormap = branca.colormap.LinearColormap(colors = colors, index =␣
↪index)
```

```
    return colormap
```

[ ]:
```
#function for creating grid markers
def create_markers(map_osm, grid_arr):
    #create feature layer
    feature = folium.map.FeatureGroup(name = "grid markers", overlay = True,␣
 ↪show = True)

    #open combined csv file and replace nan with string NaN
    combined_df = pd.read_csv("..\..\CSV_files\Combined_Data.csv")
    combined_df.fillna("NaN", inplace = True)

    #iterate through rows
    for index, row in combined_df.iterrows():
        #find middle point of grid
        grid = grid_arr[row["Grid"]]
        long_coord = (grid[0][0] + grid[3][0]) / 2
        lat_coord = (grid[0][1] + grid[3][1]) / 2
        column_names = list(combined_df.columns.values)

        #create string of row data
        text = ""
        for col_name in column_names:
            text += "<br>" + col_name + ": " + str(row[col_name]) + "<br>"

        #add to feature layer
        iframe = folium.IFrame(text, width=220, height=310)
        popup = folium.Popup(iframe)
        folium.Marker(location=[lat_coord, long_coord], popup=popup).
 ↪add_to(feature)

    #add to map
    map_osm.add_child(feature)
```

[ ]:
```
#function for adding Calgary 10x10 grid array onto the map
def create_grid_array(grid_array, map_osm):
    #create feature layer
    feature = folium.map.FeatureGroup(name = "grid array", overlay = True, show␣
 ↪= True)

    #iterate through each grid
    for grid in grid_array:
        #find the 4 corners of the grid
        coord_box = box(grid[0][0],grid[0][1],grid[3][0],grid[3][1])

        #create box geojson and add to feature layer
        geo = gpd.GeoSeries([coord_box]).__geo_interface__
```

```
        folium.GeoJson(geo).add_to(folium.FeatureGroup(name='grid array')).
↪add_to(feature)

    #add feature layer to map
    map_osm.add_child(feature)
```

### 3.7.2 Creating 10x10 grid

[ ]:
```
#Read city boundary layer
df = pd.read_csv("..\..\CSV_files\City_Boundary_layer.csv")
# df
```

[ ]:
```
#parse the coordinates
string = df["the_geom"].values[0]
string_stripped= string.replace("POLYGON","").replace("(","").replace(")","").
 ↪replace(",","")
string_split = string_stripped.split()
```

[ ]:
```
list_long = []
list_lat = []

#appends latitudes and longitudes based on long/lat/long/lat... pattern
for i in range(len(string_split)):
    if i % 2 == 0:
        list_long.append(float(string_split[i]))
    else:
        list_lat.append(float(string_split[i]))

#find the min and max values of the boundary
min_long = min(list_long)
max_long = max(list_long)
min_lat = min(list_lat)
max_lat = max(list_lat)
# print(min_long)
# print(max_long)
```

[ ]:
```
#creation of grid array for a 10x10 grid
long_ten_split = np.linspace(min_long,max_long,num = 11)
lat_ten_split = np.linspace(min_lat,max_lat,num = 11)

grid_array = []
for y in range(len(lat_ten_split)-1):
    for x in range(len(long_ten_split)-1):
        bot_left = [long_ten_split[x],lat_ten_split[y]]
        bot_right = [long_ten_split[x+1],lat_ten_split[y]]
        top_left = [long_ten_split[x],lat_ten_split[y+1]]
        top_right = [long_ten_split[x+1],lat_ten_split[y+1]]
        grid_array.append([bot_left,bot_right,top_left,top_right])
```

### 3.7.3 Traffic volume heat map

[ ]:
```
#read traffic csv
traffic_df = pd.read_csv("..\..\CSV_files\Traffic_Volumes_for_2018.csv")
# traffic_df
```

[ ]:
```
#parse data for lat, longs and weight (volume)
traffic_data = []

#parse multilinestring
for string,volume in zip(traffic_df['multilinestring'],traffic_df["VOLUME"]):
    string_stripped= string.replace("MULTILINESTRING","").replace("(","").
 ↪replace(")","").replace(",","")
    string_split = string_stripped.split()
    float_split = [float(i) for i in string_split]

    for i in range(int(len(float_split)/2)):
        traffic_data.append([float_split[i*2+1],float_split[i*2],float(volume)])


list_lat = [i[0] for i in traffic_data]
list_long = [i[1] for i in traffic_data]
list_weight = [i[2] for i in traffic_data]
```

[ ]:
```
#create map
map_osm = folium.Map(location = [50.913577283979,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "volume", list_weight␣
 ↪= list_weight)

# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.4 Traffic camera heat map

[ ]:
```
#read traffic camera locations csv
camera_df = pd.read_csv("..\..\CSV_files\Traffic_Camera_Locations.csv")
# camera_df
```

[ ]:
```
#parse data for lats and longs
list_long = [i for i in camera_df["longitude"]]
list_lat = [i for i in camera_df["latitude"]]
```

[ ]:
```
# map_osm = folium.Map(location = [50.913577283979,-114.
↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "camera", r = 25)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.5 Traffic signals heat map

[ ]:
```
#read traffic signals csv
signals_df = pd.read_csv("..\..\CSV_files\Traffic_Signals.csv")
# signals_df
```

[ ]:
```
#parse data for lats and longs
list_long = [i for i in signals_df["longitude"]]
list_lat = [i for i in signals_df["latitude"]]
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.913577283979,-114.
↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "signals", r = 17)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.6 Traffic signs heat map

[ ]:
```
#read traffic signs csv
signs_df = pd.read_csv("..\..\CSV_files\Traffic_Signs.csv")

#drop nan and 0 values in sign count and sign type
signs_df.dropna(subset=["SGN_COUNT_NO"], how='all', inplace=True)
signs_df.dropna(subset=["BLADE_TYPE"], how='all', inplace=True)
signs_df = signs_df[signs_df.SGN_COUNT_NO != 0]

#list of "irrelevant signs" (parking signs, info signs etc.)
signs_nan = ['Timed Parking','Park Plus','Parking Restrictions','Street␣
 ↪Name','Snow Route','Guide / Information','Loading Zone','Residential␣
 ↪Parking', 'Overhead Guide']

#remove irrelevant signs
signs_df = signs_df.loc[~signs_df["BLADE_TYPE"].isin(signs_nan)]

# signs_df
```

[ ]:
```
#parse data for lats, longs and weight (sign count)
list_long = []
list_lat = []
list_weight = [i for i in signs_df["SGN_COUNT_NO"]]

#parse POINT string
for row in signs_df["POINT"]:
    string_stripped = row.replace("POINT", "").replace("(", "").replace(")", "")
    string_split = string_stripped.split()

    list_long.append(float(string_split[0]))
    list_lat.append(float(string_split[1]))
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.9135772839?9,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "signs (all)",␣
 ↪list_weight = list_weight, r = 15)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.7 Traffic incidents

[ ]:
```
#read traffic signs csv
incidents_df = pd.read_csv("..\..\CSV_files\Traffic_Incidents.csv")

#parse for 2018 data
incidents_df = incidents_df[incidents_df.START_DT.str.contains("2018")]
# incidents_df
```

[ ]:
```
#parse data for lats and longs
list_long = [i for i in incidents_df["Longitude"]]
list_lat = [i for i in incidents_df["Latitude"]]
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.913577283979,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "incidents (rainbow)",␣
 ↪r = 20)
create_heatmap_incidents(list_lat, list_long, map_osm, heat_name = "incidents",␣
 ↪r = 20)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.8 Traffic signs (stop, warning yield) specific heat map

[ ]:
```
#read traffic signs csv
signs_df = pd.read_csv("..\..\CSV_files\Traffic_Signs.csv")

#drop nan and 0 values in sign count and sign type
signs_df.dropna(subset=["SGN_COUNT_NO"], how='all', inplace=True)
signs_df.dropna(subset=["BLADE_TYPE"], how='all', inplace=True)
signs_df = signs_df[signs_df.SGN_COUNT_NO != 0]

#list of "irrelevant signs" (parking signs, info signs etc.)
# signs_nan = ['Timed Parking','Park Plus','Parking Restrictions','Street␣
 ↪Name','Snow Route','Guide / Information','Loading Zone','Disabled␣
 ↪Parking','Residential Parking', 'Overhead Guide']

#remove irrelevant signs
signs_df = signs_df.loc[signs_df["BLADE_TYPE"].isin(["Stop","Warning","Yield"])]

# signs_df
```

[ ]:
```
#parse data for lats, longs and weight (sign count)
list_long = []
list_lat = []
list_weight = [i for i in signs_df["SGN_COUNT_NO"]]

#parse POINT string
for row in signs_df["POINT"]:
    string_stripped = row.replace("POINT", "").replace("(", "").replace(")", "")
    string_split = string_stripped.split()

    list_long.append(float(string_split[0]))
    list_lat.append(float(string_split[1]))
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.9135777283979,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "signs (stop, warning,␣
 ↪yield)", list_weight = list_weight, r = 15)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.9 Traffic signs (speed) specific heat map

[ ]:
```
#read traffic signs csv
signs_df = pd.read_csv("..\..\CSV_files\Traffic_Signs.csv")

#drop nan and 0 values in sign count and sign type
signs_df.dropna(subset=["SGN_COUNT_NO"], how='all', inplace=True)
signs_df.dropna(subset=["BLADE_TYPE"], how='all', inplace=True)
signs_df = signs_df[signs_df.SGN_COUNT_NO != 0]

#list of "irrelevant signs" (parking signs, info signs etc.)
# signs_nan = ['Timed Parking','Park Plus','Parking Restrictions','Street␣
 ↪Name','Snow Route','Guide / Information','Loading Zone','Disabled␣
 ↪Parking','Residential Parking', 'Overhead Guide']

#remove irrelevant signs
signs_df = signs_df.loc[signs_df["BLADE_TYPE"].isin(["Speed"])]

# signs_df
```

[ ]:
```
#parse data for lats, longs and weight (sign count)
list_long = []
list_lat = []
list_weight = [i for i in signs_df["SGN_COUNT_NO"]]

#parse POINT string
for row in signs_df["POINT"]:
    string_stripped = row.replace("POINT", "").replace("(", "").replace(")", "")
    string_split = string_stripped.split()

    list_long.append(float(string_split[0]))
    list_lat.append(float(string_split[1]))
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.9135772839?9,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "signs (speed)",␣
 ↪list_weight = list_weight, r = 20)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.10 Traffic signs (pedestrians and bicycle pathways) specific heat map

[ ]:
```
#read traffic signs csv
signs_df = pd.read_csv("..\..\CSV_files\Traffic_Signs.csv")

#drop nan and 0 values in sign count and sign type
signs_df.dropna(subset=["SGN_COUNT_NO"], how='all', inplace=True)
signs_df.dropna(subset=["BLADE_TYPE"], how='all', inplace=True)
signs_df = signs_df[signs_df.SGN_COUNT_NO != 0]

#list of "irrelevant signs" (parking signs, info signs etc.)
# signs_nan = ['Timed Parking','Park Plus','Parking Restrictions','Street␣
 ↪Name','Snow Route','Guide / Information','Loading Zone','Disabled␣
 ↪Parking','Residential Parking', 'Overhead Guide']

#remove irrelevant signs
signs_df = signs_df.loc[signs_df["BLADE_TYPE"].isin(['Pedestrian','Bicycle /␣
 ↪Pathway'])]

# signs_df
```

[ ]:
```
#parse data for lats, longs and weight (sign count)
list_long = []
list_lat = []
list_weight = [i for i in signs_df["SGN_COUNT_NO"]]

#parse POINT string
for row in signs_df["POINT"]:
    string_stripped = row.replace("POINT", "").replace("(", "").replace(")", "")
    string_split = string_stripped.split()

    list_long.append(float(string_split[0]))
    list_lat.append(float(string_split[1]))
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.9135772839?9,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "signs (pedestrian,␣
 ↪bicycle)", list_weight = list_weight, r = 15)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("traffic_heat_map_grid_marker.html")
# map_osm
```

### 3.7.11 Traffic signs (playground and schools) specific heat map

[ ]:
```
#read traffic signs csv
signs_df = pd.read_csv("..\..\CSV_files\Traffic_Signs.csv")

#drop nan and 0 values in sign count and sign type
signs_df.dropna(subset=["SGN_COUNT_NO"], how='all', inplace=True)
signs_df.dropna(subset=["BLADE_TYPE"], how='all', inplace=True)
signs_df = signs_df[signs_df.SGN_COUNT_NO != 0]

#list of "irrelevant signs" (parking signs, info signs etc.)
# signs_nan = ['Timed Parking','Park Plus','Parking Restrictions','Street␣
 ↪Name','Snow Route','Guide / Information','Loading Zone','Disabled␣
 ↪Parking','Residential Parking', 'Overhead Guide']

#remove irrelevant signs
signs_df = signs_df.loc[signs_df["BLADE_TYPE"].isin(['Playground','School'])]

# signs_df
```

[ ]:
```
#parse data for lats, longs and weight (sign count)
list_long = []
list_lat = []
list_weight = [i for i in signs_df["SGN_COUNT_NO"]]

#parse POINT string
for row in signs_df["POINT"]:
    string_stripped = row.replace("POINT", "").replace("(", "").replace(")", "")
    string_split = string_stripped.split()

    list_long.append(float(string_split[0]))
    list_lat.append(float(string_split[1]))
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.9135772839 79,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
create_heatmap(list_lat, list_long, map_osm, heat_name = "signs (playground,␣
 ↪school)", list_weight = list_weight, r = 15)
# create_grid_array(grid_array, map_osm)
# create_markers(map_osm, grid_array)
# folium.LayerControl().add_to(map_osm)

# map_osm.save("layered_heat_map_grid_marker.html")
# map_osm
```

### 3.7.12 Speed limit road colors

[ ]:
```
#import traffic Speed Limits
speed_df = pd.read_csv('..\..\CSV_files\Speed_Limits.csv')

#take relevant columns
speed_df = speed_df[["SPEED","multiline"]]
# speed_df
```

[ ]:
```
#speed list
list_speed = [i for i in speed_df["SPEED"]]
```

[ ]:
```
#parse multiline string for coordinate list
list_coord_row =[]

for row in speed_df["multiline"]:

    #strip MULTILINESTRING and (
    string = row.replace("MULTILINESTRING","").replace("(","")

    #split on every line
    string_split = string.split("),")

    list_coord = []
    for string in string_split:
        #find coordinate pairs
        pairs = string.split(",")
        list_pair = []

        for pair in pairs:
            #convert each coordinate into a float
            pair_strip = pair.strip().replace(")","")
            pair_split = pair_strip.split()
            float_split = [float(i) for i in pair_split]

            list_pair.append(tuple(float_split))
        list_coord.append([list_pair])
    list_coord_row.append(list_coord)
```

[ ]:
```
#create color gradient
yellow = Color("yellow")
colors = list(yellow.range_to(Color("red"),10))
colors = [str(i) for i in colors]
colors.insert(2,'#ffd400')
colors.insert(4,'#ffb800',)

keys = [i for i in range(20,111,10)]
```

```
keys.append(35)
keys.append(45)
keys.sort()

#create dictionary for speeds to corresponding color
colors_dict = dict(zip(keys,colors))
```

[ ]:
```
#creating legend using pyplot
fig, ax = plt.subplots()
fig.set_size_inches(5, 10.5)
handles = [mpatches.Patch(color=colors_dict[x], label=x) for x in colors_dict.
 ↪keys()]
ax.legend(handles = handles, fontsize = 30, loc = "center", title = "Speed␣
 ↪limit (km/h)", title_fontsize = "20")
fig.gca().set_axis_off()
fig.savefig("..\..\HTML_files\legend.png")

#suppress output
plt.close()
```

[ ]:
```
feature_speed = folium.map.FeatureGroup(name = "speed limits", overlay = True,␣
 ↪show = True)

#add legend image to html
FloatImage("legend.png", bottom=-10, left=85).add_to(feature_speed)

#add roads and their corresponding color based on the speed limit
for row,speed in zip(list_coord_row, list_speed):
    for line in row:
        for coordinate in line:
            style = {'fillColor': colors_dict[speed], 'color':␣
 ↪colors_dict[speed]}
            lines = MultiLineString([coordinate])
            geo = gpd.GeoSeries([lines]).__geo_interface__
            folium.GeoJson(geo, style_function=lambda x, fillColor =␣
 ↪style["fillColor"], color = style["color"]: {"fillColor": fillColor, "color":
 ↪ color}).add_to(feature_speed)
```

[ ]:
```
#create map
# map_osm = folium.Map(location = [50.9135772839?9,-114.
 ↪073657541927],control_scale = True, zoom_start=10)

#add heatmap
# create_heatmap(list_lat, list_long, map_osm, heat_name = "signs (playground,␣
 ↪school)", list_weight = list_weight, r = 15)
```

```
#create grid array and markers feature layers
map_osm.add_child(feature_speed)
create_grid_array(grid_array, map_osm)
create_markers(map_osm, grid_array)

# #add layer control
folium.LayerControl(position = "bottomleft").add_to(map_osm)

map_osm.save("..\..\HTML_files\layered_heat_map_grid_marker.html");
# map_osm
```