ENSF592 FINAL: TRAFFIC ANALYSIS FOR THE CITY OF CALGARY

DATE: AUG 13, 2020

INTRODUCTION

PURPOSE: The purpose of this project is to analyze potential factors that can lead to an increase in traffic incidents in the City of Calgary, and determine how strongly these factors correlate to an increase in incidents. Our team analyzes data provided by the City of Calgary itself, readily available at https://data.calgary.ca/browse (https://data.calgary.ca/browse). The potential factors that we consider for our analysis include various road features (speed limits, traffic volume, and the number of traffic cameras, signs, and signals) and changing weather conditions (temperature and visibility). The potential factors, as well as incident data, are analyzed for the year 2018. These factors will be mathematically and graphically compared so that we can make a data-driven conclusion on which factors result in increased traffic incidents.

HYPOTHESIS: All scientific experiments start with an initial hypothesis, and this study is no different for us. While this data analysis isn't a true experiment, we can still treat it like such and hypothesize how the various factors listed above can potentially affect traffic incident numbers. Our intial hypothesis is as follows:

- Areas with higher average speed limits (such as highways) will see lower incident numbers when compared to areas with relatively lower speed limits (such as inner-city roads). We base this assumption on the fact that highway driving is more streamlined with less stopping and going.
- Areas with higher traffic volume will see higher incident numbers (more cars means more chance of accidents occuring).
- Areas with more traffic cameras, that are more closely monitored by the City, will have lower incident numbers (the presence of cameras will encourage drivers to obey the law more and drive more carefully).
- Areas with more traffic signs will see lower incident numbers (presence of road signs will encourage drivers to obey the law more and drive more carefully).
- Areas with more traffic signals will see higher incident numbers (the presence of more intersections increases the risk of accidents due to failure to stop, yield, or give right of way).
- Colder temperatures (which also brings about ground frost and snow) will see higher incident numbers, with incident spikes occuring around 0 degrees C and single negative digits (temperatures hovering around 0 and just below result in the roads being at their most slippery due to frost)
- Higher visibility will result in lower incident numbers (drivers who can see clearer are less likely to fall victims to traffic incidents).

PROJECT FLOW: The project is broken down and presented in the following format:

- Data Preparation and Aggregation
- Data Analysis and Visualization
- Conclusion of Findings

DATA PREPARATION AND AGGREGATION

In this section, the City of Calgary will be approximately split up into 100 grids which will each be analyzed. The analysis of each grid will include:

- Average Speed Limit
- Average Traffic Volume
- Number of Traffic Cameras
- Number of Traffic Signals
- Number of Traffic Signs
- Number of Traffic Incidents

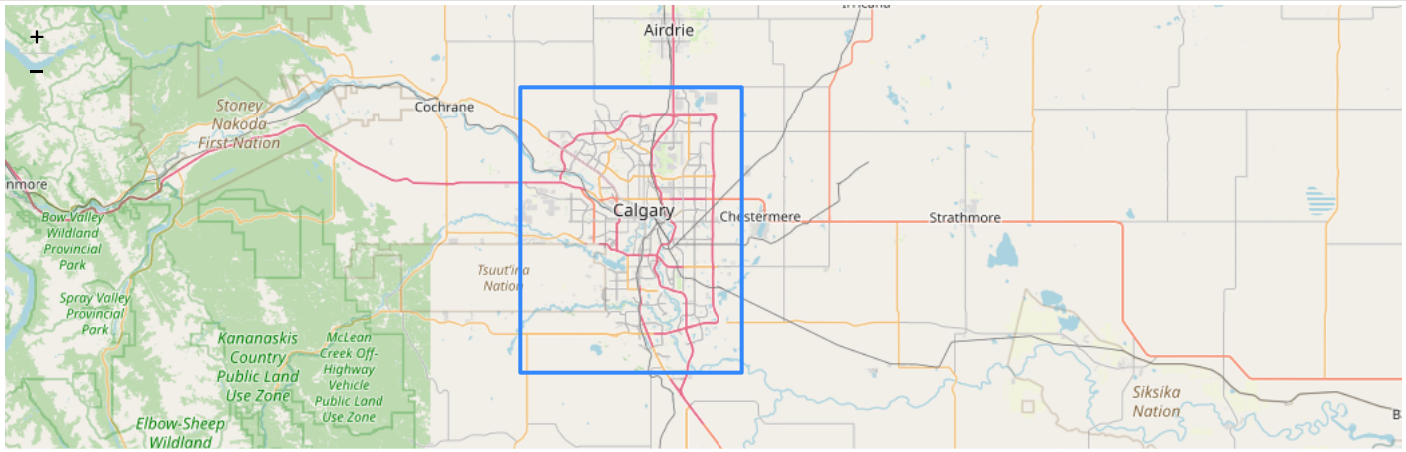
The resulting calculations from each grid's analysis will be presented in a DataFrame, as well as visually, in the next section.

In [1]:
```
# Import required packages to aid with data analysis:

%matplotlib inline

import pandas as pd
import numpy as np
import folium
import re
from statistics import mean
from folium.plugins import HeatMap
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import json
import geojson
import branca
import branca.colormap as cm
import pdb
from geojson import MultiLineString
import gmaps
```

Data Preparation starts with finding the City of Calgary boundary. Data Analysis will only be performed within the City limits. We read from the City_Boundary_layer CSV file and clean the data before obtaining the max and min coordinates for the City limits. We then visually display the boundary for ease of viewing our project scope.

In [2]:
```
#Read in csv for data cleaning:
test_df = pd.read_csv('City_Boundary_layer.csv')
boundary_coordinates = test_df.the_geom.get(0)
boundary_coordinates = boundary_coordinates.replace('(', '').replace(')', '').replace('POLYGON', '').replace(',', '')
coordinate_list = boundary_coordinates.split()

#Create a list of latitude and longitude coordinates the City falls within:
lat_list = [float(coordinate_list[i]) for i in range(len(coordinate_list)) if i % 2 == 1]
long_list = [float(coordinate_list[j]) for j in range(len(coordinate_list)) if j % 2 == 0]

#Find the maximum and minimum coordinates from the coordinate list to show the maximum city boundary:
min_lat = min(lat_list)
max_lat = max(lat_list)
min_long = min(long_list)
max_long = max(long_list)

#Use central coordinates for the City to center the Folium map:
calgary_lat = 51.0447
calgary_long = -114.0719

#Pass the max and min boundary coordinates to Folium to generate a Calgary map showing our boundary scope:
my_map = folium.Map(location=[calgary_lat, calgary_long], zoom_start=10)
folium.vector_layers.Rectangle([[max_lat, max_long], [min_lat, min_long]]).add_to(my_map)
my_map
```

Out[2]:



Next, we want to split our scope into 100 grids to aid with our analysis of the City. We can split the City into grids by considering the maximum and minimum coordinates, and iterating over these coordinates in equal length segments. It is important to note that Latitude DECREASES going from top to bottom and Longitude INCREASES going from left to right. The grids are zero-indexed, meaning the upper-leftmost grid has the coordinates 0,0 (col,row) and the bottom-rightmost grid has the coordinates 9,9. The grid data will be stored in a DataFrame where each DF element is a dict that has a key-value pair representing the min and max longitude and latitude.

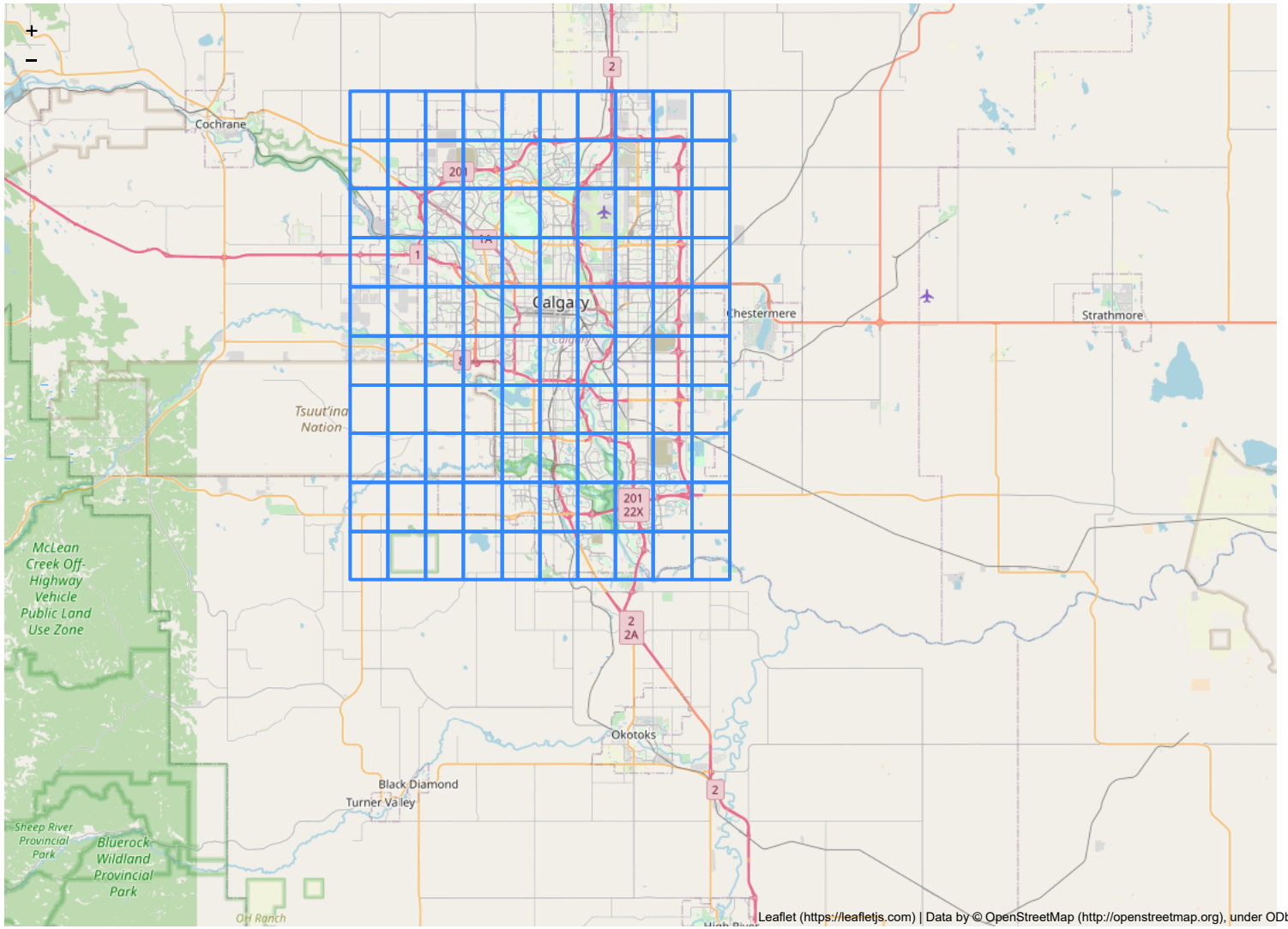
In [3]:
```
# Generate a list of lists that will represent the grid coordinates
grid = [[0 for x in range(10)] for y in range(10)]

# Calculate grid boundaries:
for i in range(10):
    # Get the min/max longitude for each grid:
    grid_min_long = min_long + ((max_long - min_long) / 10) * i
    grid_max_long = min_long + ((max_long - min_long) / 10) * (i + 1)

    for j in range(10):
        # Get the min/max latitude for each grid:
        grid_max_lat = max_lat - ((max_lat - min_lat) / 10) * j
        grid_min_lat = max_lat - ((max_lat - min_lat) / 10) * (j + 1)
         # Populate data into a dataframe for each grid:
        grid[j][i] = {'min_lat':grid_min_lat,'max_lat':grid_max_lat,'min_long':grid_min_long,'max_long':grid_max_long}

grid_df = pd.DataFrame(grid)
grid_df # See Appendix of PDF report for entire grid_df
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9
0	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....	{'min_lat': 51.175464700000006, 'max_lat': 51....
1	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...	{'min_lat': 51.1385044, 'max_lat': 51.17546470...
2	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...	{'min_lat': 51.1015441, 'max_lat': 51.1385044,...
3	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...	{'min_lat': 51.0645838, 'max_lat': 51.1015441,...
4	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....	{'min_lat': 51.027623500000004, 'max_lat': 51....
5	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...	{'min_lat': 50.9906632, 'max_lat': 51.02762350...
6	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....	{'min_lat': 50.953702899999996, 'max_lat': 50....
7	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...	{'min_lat': 50.9167426, 'max_lat': 50.95370289...
8	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...	{'min_lat': 50.8797823, 'max_lat': 50.9167426,...
9	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...	{'min_lat': 50.842822, 'max_lat': 50.8797823, ...

In [4]:
```
#Generate City of Calgary map split into 100 individual grids to be analyzed:
my_map2 = folium.Map(location=[calgary_lat, calgary_long], zoom_start=10)
# Place rectangle for each grid:
for i in range(10):
    for j in range(10):
        grid_bounds = [[grid_df[j][i]['max_lat'],grid_df[j][i]['max_long']],\
                       [grid_df[j][i]['min_lat'], grid_df[j][i]['min_long']]]

        folium.vector_layers.Rectangle(grid_bounds).add_to(my_map2)

my_map2
```

Out[4]:



Road Features:

Next, we create a master_df that will store the analysis results for all the grids in the City based on road features. We start by creating an empty DataFrame with column names, and values representing which grid the calculations belong to. Then we analyze multiple CSV files available from the City of Calgary to populate the elements of the DataFrame. See comments below in each code cell for further information on how the data values were calculated. The entire filled DataFrame can be viewed in the Appendix of the PDF report.

In [5]:
```
#Create lists to prepare empty dataframe (master_df) to store analysis results for all 100 grids
colname_list = ['column', 'row', 'avg_speed_limit', 'avg_volume', 'traffic_cameras', 'traffic_signals', 'traffic_signs', 'traffic_incidents']
row_list = []
col_list = []
for i in range(10):
    for j in range(10):
        col_list.append(i)
        row_list.append(j)

#Create dataframe with columns and rows values (cols and rows represent grids)
master_df = pd.DataFrame(columns=colname_list)
master_df.row = row_list
master_df.column = col_list
```

In [6]:
```
#Adding the number of traffic cameras for each grid to the master_df:
camera_df = pd.read_csv('Traffic_Camera_Locations.csv')
num_camera_list = [] #generate empty list that will hold number of cameras in a grid

#Code loops through the grid_df and checks if the cameras fall within the min and max coordinates of a grid.
#If they do, append the total number of cameras in the grid to the num_camera_list (note: 0 is a valid entry)
for i in range(10):
    for j in range(10):
        #returns the size of the dataframe that has cameras that fall within a specified grid after filtering using
        #conditional statements
        num_camera_list.append(camera_df[(camera_df.longitude > grid_df[i][j]['min_long']) &\
                               (camera_df.longitude < grid_df[i][j]['max_long']) &\
                               (camera_df.latitude > grid_df[i][j]['min_lat']) &\
                               (camera_df.latitude < grid_df[i][j]['max_lat'])].shape[0])

master_df.traffic_cameras = num_camera_list #add the number of cameras in each grid to the master_df
```

In [7]:
```
#Adding the number of traffic signals for each grid to the master_df:
signals_df = pd.read_csv('Traffic_Signals.csv')
num_signals_list = [] #generate empty list that will hold number of signals in a grid

#Code loops through the grid_df and checks if the signals fall within the min and max coordinates of a grid.
#If they do, append the total number of signals in the grid to the num_signals_list (note: 0 is a valid entry)
for i in range(10):
    for j in range(10):
        #returns the size of the dataframe that has signals that fall within a specified grid after filtering using
        #conditional statements
        num_signals_list.append(signals_df[(signals_df.longitude > grid_df[i][j]['min_long']) &\
                               (signals_df.longitude < grid_df[i][j]['max_long']) &\
                               (signals_df.latitude > grid_df[i][j]['min_lat']) &\
                               (signals_df.latitude < grid_df[i][j]['max_lat'])].shape[0])

master_df.traffic_signals = num_signals_list #add the number of signals in each grid to the master_df
```

In [8]:
```
#Adding the number of traffic incidents for each grid to the master_df:
traffic_incidents_df = pd.read_csv('Traffic_Incidents.csv')
criterion = traffic_incidents_df['id'].map(lambda x : x.startswith('2018'))
traffic_incidents_df = traffic_incidents_df[criterion] #returns dataframe with only 2018 data
incident_list = [] #generate empty list that will hold number of signals in a grid

#Code loops through the grid_df and checks if the incidents fall within the min and max coordinates of a grid.
#If they do, append the total number of incidents in the grid to the incidents_list (note: 0 is a valid entry)
for i in range(10):
    for j in range(10):
        #returns the size of the dataframe that has incidents that fall within a specified grid after filtering using
        #conditional statements
        incident_list.append(traffic_incidents_df[(traffic_incidents_df.Longitude > grid_df[i][j]['min_long']) &\
                             (traffic_incidents_df.Longitude < grid_df[i][j]['max_long']) &\
                             (traffic_incidents_df.Latitude > grid_df[i][j]['min_lat']) &\
                             (traffic_incidents_df.Latitude < grid_df[i][j]['max_lat'])].shape[0])

master_df.traffic_incidents = incident_list #add the number of incidents in each grid to the master_df
```

In [9]:
```
#Dataframe that contain data from 'Speed_Limit.csv'
speedlimit_df = pd.read_csv('Speed_Limits.csv')

#check to make sure column "SPEED' in the csv is numeric. If error return NA
pd.to_numeric(speedlimit_df['SPEED'], errors='coerce')

#Initialize new dataframe that pull column "SPEED" from the csv
speedlimitclean_df = pd.DataFrame(speedlimit_df['SPEED'])

# since for every row under columnm "multiline", there are multiple coordinates that represent multilines.
# Store these arrays of corrdinates into lists
coordinateLong_list = []
coordinateLat_list = []

# List to represent all arrays of coordinates
coordinateLong_list_merge = []
coordinateLat_list_merge = []

# Loop through every rows in the dataset, do operations to turn each cell under column "multiline" into arrays of coordinates,
# then store all lat and long coordinates in coordinateLong_list_merge and coordinateLat_list_merge respectively
for i, j in speedlimit_df.iterrows():
    speedlimi_coordinates = j.multiline
    #replace to isolate lat and long coordinates
    speedlimi_coordinates = speedlimi_coordinates.replace('(', '').replace(')', '').replace('MULTILINESTRING', '').replace(',', '')
    # split into individual coordinates
    speedlimitcoordinate_list = speedlimi_coordinates.split()
    #since lat and long go in pair, use even odds logic to seperate into lat and long
    coordinateLat_list = [float(speedlimitcoordinate_list[k]) for k in range(len(speedlimitcoordinate_list)) if k % 2 == 1]
    coordinateLong_list = [float(speedlimitcoordinate_list[l]) for l in range(len(speedlimitcoordinate_list)) if l % 2 == 0]
    #append arrays of lat and long into one list for Lat and one list for Long
    coordinateLat_list_merge.append(coordinateLat_list)
    coordinateLong_list_merge.append(coordinateLong_list)

#assign values for Lat and Long columns in the previous dataframe that contain only the "SPEED" column
speedlimitclean_df['Lat'] = coordinateLat_list_merge
speedlimitclean_df['Long'] = coordinateLong_list_merge

#unpivot columns "Lat" and "Long" by using explode function, stored these exploded values into a new dataframe
speedlimitclean_df_Lat = speedlimitclean_df.explode('Lat')
speedlimitclean_df_Lat = speedlimitclean_df_Lat.drop(["Long"], axis = 1)
speedlimitclean_df_Long = speedlimitclean_df.explode('Long')
speedlimitclean_df_Long = speedlimitclean_df_Long.drop(["Lat"], axis = 1)

#Drop the previous Long and Lat column to prepare to add new ones
speedlimitclean_df = speedlimitclean_df.drop(["Long"], axis = 1)
speedlimitclean_df = speedlimitclean_df.drop(["Lat"], axis = 1)

# Merge "Lat" and "Long" columns together, and assign that into a clean new dataframe
speedlimitclean_df_Long['Lat'] = speedlimitclean_df_Lat.Lat
speedlimitclean_df = speedlimitclean_df_Long

#list containing all the averages of speed limit
speedaverage_list = []

#Loop through every grid and store the average values of speed liimit in speedaverage_list
for i in range(10):
    for j in range(10):
        #array to contain all speed limits within a certain grid
        countspeed =[]
        #temporary store the filtered value grid to grid as a new dataframe
        temp = speedlimitclean_df[(speedlimitclean_df.Long > grid_df[i][j]['min_long']) &\
                                  (speedlimitclean_df.Long < grid_df[i][j]['max_long']) &\
                                  (speedlimitclean_df.Lat > grid_df[i][j]['min_lat']) &\
                                  (speedlimitclean_df.Lat < grid_df[i][j]['max_lat'])]
```

```
        # remove any na value
        temp.dropna()
        #check if temp is none or empty
        if (temp is not None and temp.shape[0] > 0):
            # add all speed limits that fall within this specific grid
            for row,col in temp.iterrows():
                countspeed.append(col.SPEED)
            # Average out the speed limit within this grid, then add to speedaverage_list
            speedaverage_list.append(np.nanmean(countspeed))
        else:
            #if no data for speed limit, default to NaN
            speedaverage_list.append(np.nan)

#assign speed limit average values to  master dataframe
master_df.avg_speed_limit = speedaverage_list
```

In [10]:
```
#Dataframe that contain data from 'Traffic_Volumes_for_2018.csv'
trafficvolume_df = pd.read_csv('Traffic_Volumes_for_2018.csv')

#check to make sure column "VOLUME" is numeric, if not then put na
pd.to_numeric(trafficvolume_df['VOLUME'], errors='coerce')

# initialzie a fresh dataframe that contain only the "VOLUME" column from csv data
trafficvolumeclean_df = pd.DataFrame(trafficvolume_df['VOLUME'])

# since for every row under columnm "multiline", there are multiple coordinates that represent multilines.
# Store these arrays of corrdinates into lists
trafficcoordinateLong_list = []
trafficcoordinateLat_list = []

# List of list of coordinates, append to every row from trafficcoordinateLat_list_merge and trafficcoordinateLong_list_merge
trafficcoordinateLat_list_merge = []
trafficcoordinateLong_list_merge = []

# Loop through every rows in the dataset, do operations to turn each cell under column "multiline" into arrays of coordinates,
# then store all lat and long coordinates in trafficcoordinateLat_list_merge and trafficcoordinateLong_list_merge respectively
for i, j in trafficvolume_df.iterrows():
    traffic_coordinates = j.multilinestring
    #replace to isolate lat and long coordinates
    traffic_coordinates = traffic_coordinates.replace('(', '').replace(')', '').replace('MULTILINESTRING ', '').replace(',', '')
    # split into individual coordinates
    traffic_coordinates_list = traffic_coordinates.split()
    #since lat and long go in pair, use even odds logic to seperate into lat and long
    trafficcoordinateLat_list = [float(traffic_coordinates_list[k]) for k in range(len(traffic_coordinates_list)) if k % 2 == 1]
    trafficcoordinateLong_list = [float(traffic_coordinates_list[l]) for l in range(len(traffic_coordinates_list)) if l % 2 == 0]
    #append arrays of lat and long into one list for Lat and one list for Long
    trafficcoordinateLat_list_merge.append(trafficcoordinateLat_list)
    trafficcoordinateLong_list_merge.append(trafficcoordinateLong_list)

#assign values for Lat and Long columns in the previous dataframe that contain only the "VOLUME" column
trafficvolumeclean_df['Lat'] = trafficcoordinateLat_list_merge
trafficvolumeclean_df['Long'] = trafficcoordinateLong_list_merge

#unpivot columsn "Lat" and "Long" by using explode function, stored these exploded values into a new dataframe
trafficvolumeclean_df_Lat = trafficvolumeclean_df.explode('Lat')
trafficvolumeclean_df_Lat = trafficvolumeclean_df_Lat.drop(["Long"], axis = 1)
trafficvolumeclean_df_Long = trafficvolumeclean_df.explode('Long')
trafficvolumeclean_df_Long = trafficvolumeclean_df_Long.drop(["Lat"], axis = 1)

#Drop the previous Long and Lat column to prepare to add new ones
trafficvolumeclean_df = speedlimitclean_df.drop(["Long"], axis = 1)
trafficvolumeclean_df = speedlimitclean_df.drop(["Lat"], axis = 1)

# Merge "Lat" and "Long" columns together, and assign that into a clean new dataframe
trafficvolumeclean_df_Long['Lat'] = trafficvolumeclean_df_Lat.Lat
trafficvolumeclean_df = trafficvolumeclean_df_Long


#List containing all the averages of traffic volume
trafficspeedaverage_list = []

#Loop through every grid and store the average values of traffic volume in trafficspeedaverage_list
for i in range(10):
    for j in range(10):
        #array to contain all traffic volume data within a certain grid
        countspeed =[]
        #temporary store the filtered value grid to grid as a new dataframe
        temp = trafficvolumeclean_df[(trafficvolumeclean_df.Long > grid_df[i][j]['min_long']) &\
                                     (trafficvolumeclean_df.Long < grid_df[i][j]['max_long']) &\
                                     (trafficvolumeclean_df.Lat > grid_df[i][j]['min_lat']) &\
```

```
                            (trafficvolumeclean_df.Lat < grid_df[i][j]['max_lat'])]
        # remove any na value
        temp.dropna()
        #check if temp is none or empty
        if (temp is not None and temp.shape[0] > 0):
            # add all traffic volume data that fall within this specific grid
            for row,col in temp.iterrows():
                countspeed.append(col.VOLUME)
            # Average out the traffic volume data within this grid, then add to trafficspeedaverage_list
            trafficspeedaverage_list.append(np.nanmean(countspeed))
        else:
            #if no data for speed limit, default to NaN
            trafficspeedaverage_list.append(np.nan)

#assign traffic volume data average values to  master dataframe
master_df.avg_volume = trafficspeedaverage_list
```

In [11]:
```
#Dataframe that contain data from 'Traffic_Signs.csv'
signvolume_df = pd.read_csv('Traffic_Signs.csv')

#check to make sure column "SGN_COUNT_NOO" is numeric, if not then put na
pd.to_numeric(signvolume_df['SGN_COUNT_NO'], errors='coerce')

# initialzie a fresh dataframe that contain only the "SGN_COUNT_NO" column from csv data
signvolumeclean_df = pd.DataFrame(signvolume_df['SGN_COUNT_NO'])

#Assign a column for type of sign
signvolumeclean_df['SignName'] = signvolume_df['BLADE_TYPE']

# list of signs that will not be included in the master dataframe or the analysis, since these don't relate to traffic incidents
# or having any meaningful value
signnotinclude = ['Parking Restrictions', 'Timed Parking', 'Park Plus','Street Name','Disabled Parking','Residential Parking', 'Hospital', 'Halo']

# since for every row under columnm "POINT", there is a pair of lat and long coordinate, store these pair for each row
# in the lists below
signcoordinateLong_list = []
signccoordinateLat_list = []

# List of list of coordinates, append for every row to signcoordinateLat_list_merge and signcoordinateLong_list_merge
signcoordinateLat_list_merge = []
signcoordinateLong_list_merge = []

# Loop through every rows in the dataset, do operations to turn each cell under column "POINT" into individual Lat and Long coordinates,
# then store all lat and long coordinates in signcoordinateLat_list_merge and signcoordinateLong_list_merge respectively
for i, j in signvolume_df.iterrows():
    sign_coordinates = j.POINT
    #replace to isolate lat and long coordinates
    sign_coordinates = sign_coordinates.replace('(', '').replace(')', '').replace('POINT', '').replace(',', '')
    # split into individual coordinates
    sign_coordinates_list = sign_coordinates.split()
    #since lat and long go in pair, use even odds logic to seperate into lat and long
    signccoordinateLat_list = float(sign_coordinates_list[1].strip())
    signcoordinateLong_list = float(sign_coordinates_list[0].strip())
    #append arrays of lat and long into one list for Lat and one list for Long
    signcoordinateLat_list_merge.append(signccoordinateLat_list)
    signcoordinateLong_list_merge.append(signcoordinateLong_list)

#assign values for Lat and Long columns in the previous dataframe that contain only the "SGN_COUNT_NO" and "BLADE_TYPE" columns
signvolumeclean_df['Lat'] = signcoordinateLat_list_merge
signvolumeclean_df['Long'] = signcoordinateLong_list_merge

# drop rows that contain signs that we don't want to include in analysis
for sign in range(0,len(signnotinclude)):
    signvolumeclean_df = signvolumeclean_df[signvolumeclean_df.SignName != signnotinclude[sign] ]

#list containing all the averages of sign count
signaverage_list = []

#Loop through every grid and store the average values of count of signs in signaverage_list
for i in range(10):
    for j in range(10):
        #array to contain all traffic sign data within a certain grid
        countsign =[]
        #temporary store the filtered value grid to grid as a new dataframe
        temp = signvolumeclean_df[(signvolumeclean_df.Long > grid_df[i][j]['min_long']) &\
                                  (signvolumeclean_df.Long < grid_df[i][j]['max_long']) &\
                                  (signvolumeclean_df.Lat > grid_df[i][j]['min_lat']) &\
                                  (signvolumeclean_df.Lat < grid_df[i][j]['max_lat'])]
        # remove any na value
        temp.dropna()
        #check if temp is none or empty
```

```
        if (temp is not None and temp.shape[0] > 0):
            # add all count of signs data that fall within this specific grid
            for row,col in temp.iterrows():
                countsign.append(col.SGN_COUNT_NO)
            # sum of sign count within this grid, then add to signaverage_list
            signaverage_list.append(np.nansum(countsign))
        else:
            #if no data for sign count, default to 0
            signaverage_list.append(0)

#assign sign count data values to  master dataframe
master_df.traffic_signs = signaverage_list
```

Weather Features:

To find the relation between number of incidents and weather, we create a separate dataframe that analyzes this. We clean the Incident Data, and group the total incidents by day. We then obtain and clean the weather data, calculating the average temperature and visibility per day. Both dataframes hold 365 rows, representing each day of the year in 2018. We then merge these two dataframes on the day to get the average weather conditions and the number of incidents on the days that have these weather conditions.

It is important to note that Count in the weather_and_incidents_df is Traffic Incidents.

In [12]:
```
#Cleaning Incident Data from CSV and grouping total number of incidents by day:

#Loads csv into dataframe, using Datetime as index
incident_df = pd.read_csv('Traffic_Incidents.csv', parse_dates=[2], date_parser=pd.to_datetime,index_col=[2])
criterion = incident_df['id'].map(lambda x : x.startswith('2018'))
incident_df = incident_df[criterion] #ensures only 2018 data is used

#Creates dataframe that only has days and total incidents in those given days
incident_cleaned_df = incident_df.loc[:, incident_df.columns.intersection(['Count'])]
incident_cleaned_df = incident_cleaned_df.resample('D').sum()
```

In [13]:
```
#Obtaining and cleaning hourly weather data, and merging with incidents data:

#Create new dataframe that the weather data for each month will append to:
weather_and_incidents_df = pd.DataFrame(columns=['Temp (C)', 'Visibility (km)'])

#URL to obtain hourly weather data for YYC International Airport:
url = "https://climate.weather.gc.ca/climate_data/bulk_data_e.html?format=csv&stationID=50430&Year=2018&Month={i}&Day=14&timeframe=1&submit=Download+Data"

#Loops through 1-12, representing the months Jan-Dec. Code gets the Temp and Visibility data for every month,
#resamples to get the average temp and visibility for each day, and appends the daily averages to the
#weather_and_incidents_df
for i in range(1,13):
    #Loads csv into dataframe, using Datetime as index
    weather_data = pd.read_csv(url.format(i=i), parse_dates=[4], date_parser=pd.to_datetime,index_col=[4])
    weather_data.columns = [col.replace('\xb0', '') for col in weather_data.columns]
    weather_data_cleaned_df = weather_data.loc[:, weather_data.columns.intersection(['Temp (C)','Visibility (km)'])]
    weather_data_cleaned_df = weather_data_cleaned_df.resample('D').mean()
    weather_and_incidents_df = weather_and_incidents_df.append(weather_data_cleaned_df)

#Merge dataframes together to have one df representing weather conditions and sum of incidents for each day of the year
weather_and_incidents_df = pd.merge(weather_and_incidents_df, incident_cleaned_df, left_index=True, right_index=True)
weather_and_incidents_df = weather_and_incidents_df.reset_index().rename(columns={'index':'Date'})
```

DATA ANALYSIS AND VISUALIZATION

In this section, we take a look at our findings and try to analyze the data we have calculated. We will start by viewing the master_df, which now contains road feature calculations for every grid in the City limits. We will then look at the weather_and_incidents_df to view the average weather conditions and total number of incidents per day of the year. Then, using the data from both these dataframes, we will analyze our findings using two methods and draw conclusions on which features affect traffic incident numbers, and how these features affect incidents. These two methods are:

- Method 1: Spearman's Rank Correlation Coefficient
- Method 2: Graphical Representation using various plots

Displaying DataFrames:

In [14]:
```
master_df #Entire DataFrame included in Appendix section of PDF report.
```

Out[14]:

	column	row	avg_speed_limit	avg_volume	traffic_cameras	traffic_signals	traffic_signs	traffic_incidents
0	0	0	NaN	NaN	0	0	0.0	0
1	0	1	NaN	NaN	0	0	0.0	0
2	0	2	NaN	NaN	0	0	43.0	0
3	0	3	110.000000	44000.0	0	0	109.0	5
4	0	4	NaN	NaN	0	0	0.0	0
...	...	...	...	...	...	...	...	...
95	9	5	80.000000	NaN	0	0	6.0	0
96	9	6	NaN	10200.0	0	0	86.0	0
97	9	7	58.075314	NaN	0	0	95.0	0
98	9	8	76.341463	9000.0	0	0	51.0	2
99	9	9	NaN	NaN	0	0	4.0	0

100 rows × 8 columns

In [46]:
```
weather_and_incidents_df
```

Out[46]:

	Date	Temp (C)	Visibility (km)	Count
0	2018-01-01	-16.683333	42.570833	12
1	2018-01-02	-3.787500	40.891667	30
2	2018-01-03	-2.391667	39.212500	26
3	2018-01-04	-5.016667	40.891667	22
4	2018-01-05	-0.345833	40.891667	20
...	...	...	...	...
360	2018-12-27	-7.225000	12.650000	21
361	2018-12-28	-4.633333	40.891667	10
362	2018-12-29	1.600000	38.475000	4
363	2018-12-30	-7.066667	16.958333	11
364	2018-12-31	-11.916667	38.145833	13

365 rows × 4 columns

Displaying DataFrame for Top 5 Grid Locations With Greatest Number of Incidents:

In [16]:
```
#Sort master_df by descending number of incidents:
top_5_incidents= master_df.sort_values(by=['traffic_incidents'], ascending=False)
top_5_incidents.head()
```

Out[16]:

	column	row	avg_speed_limit	avg_volume	traffic_cameras	traffic_signals	traffic_signs	traffic_incidents
54	5	4	44.259012	15870.370370	22	223	18374.0	465
64	6	4	74.371061	48142.142857	5	38	4009.0	362
63	6	3	67.253333	37966.824645	6	36	1610.0	354
44	4	4	57.833133	29086.923077	8	77	9621.0	326
55	5	5	58.443322	31340.814630	9	61	5475.0	307

Method 1: Spearman's Rank Correlation Coefficient

The Spearman's Rank Correlation Coefficient is used to analyze the dependency relation between two variables. The coefficient ranges between -1 to +1. A coefficient closing in on -1 (negative correlation) means the dependency strongly disagrees (in other words, as X increases, Y decreases) while a coefficient closing in on +1 (positive correlation) means the dependency strongly agrees (as X increases, so does Y). More about the Spearman's Rank Correlation Coefficient can be read here: https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient (https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient)

We use the Pandas built-in correlation function to return the Spearman Rank Correlation Coefficient:

In [17]:
```
#Display Spearman Rank Correlation Coefficient for Traffic Incidents vs Road and Weather Features
#Note: Count and traffic_incidents both represent Traffic Incident count in the City of Calgary
correlation_df = master_df.corr(method='spearman')
weather_corr_df = weather_and_incidents_df.corr(method='spearman')
display(correlation_df['traffic_incidents'].iloc[2:7],weather_corr_df['Count'].iloc[:2])
```

```
avg_speed_limit   -0.319823
avg_volume         0.578726
traffic_cameras    0.800618
traffic_signals    0.940578
traffic_signs      0.919933
Name: traffic_incidents, dtype: float64

Temp (C)          -0.162125
Visibility (km)   -0.123895
Name: Count, dtype: float64
```

Analysis:

Based on the Spearman results, we can see the following:

- Very strong positive correlation between Incidents and Number of Traffic Signs & Number of Traffic Signals
- Strong positive correlation between Incidents and Number of Traffic Cameras
- Medium positive correlation between Incidents and Average Traffic Volume
- Weak negative correlation between Incidents and Average Speed Limit
- Very weak negative correlation between Incidents and Temperature and Visibility

Based on these results, our hypothesis has already been partially debunked. However, we will hold off on a final conclusion until we view our graphical results, so that we can see whether both methods agree in their findings.

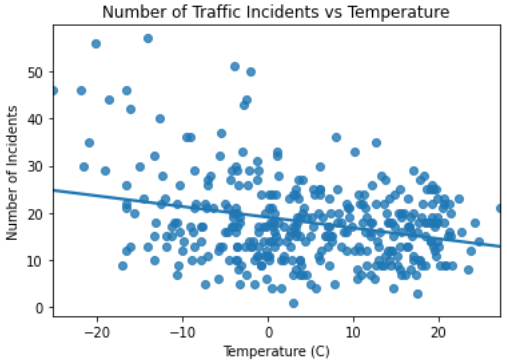
Method 2: Graphical Representation

In this section, we shall graphically compare Number of Incidents with various Road and Weather features to see if we can notice trends that agree with our findings from Method 1 (Spearman Rank).

Plot #1: The graph below shows the relation between Number of Incidents and Average Temperature. This was created by using the weather_and_incidents_df and grouping the Temperature data while summing up

the Count (incidents) column for the grouped temperature values. The relation is then represented using a Seaborn regression plot, which is excellent for showing expected linear relations.

In [18]:
```
# Regression plot showing Number of Incidents vs Weather (Temperature)
temp_incidents = weather_and_incidents_df.groupby(['Temp (C)']).Count.sum().reset_index()
ax = sns.regplot('Temp (C)', 'Count', data = temp_incidents, ci=None)
ax.set(title='Number of Traffic Incidents vs Temperature', xlabel='Temperature (C)',ylabel='Number of Incidents')
plt.show()
```

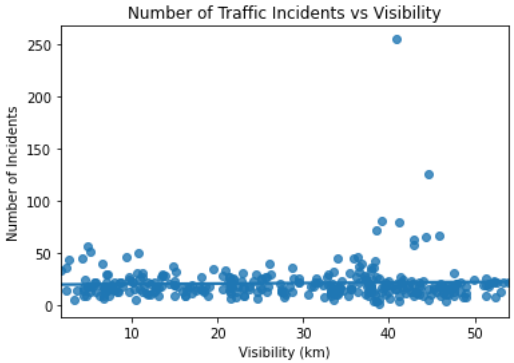


Analysis:

Viewing the regression plot, we can see a trend that shows number of incidents generally decreasing as the temperature increases. We expected colder temperatures to lead to more incidents, so this visualization, along with our negative Spearman Coefficient confirm this. We also notice the cluster get more dense around the 0C temperature mark. This was also expected since roads are the most slippery during this freeze/thaw cycle.

Plot #2: The graph below shows the relation between Number of Incidents and Average Visibility. This graph was also created by using weather_and_incidents_df but grouping the Visibility data while summing the incident count for the grouped data. The relation between Visibility and Incident count is then represented using a Seaborn regression plot.

In [19]:
```
#Generate Incidents vs Visibility Graph
vis_incidents = weather_and_incidents_df.groupby(['Visibility (km)']).Count.sum().reset_index()
ax = sns.regplot('Visibility (km)', 'Count', data=vis_incidents, ci=None)
ax.set(title='Number of Traffic Incidents vs Visibility', xlabel='Visibility (km)', ylabel='Number of Incidents')
plt.show()
```

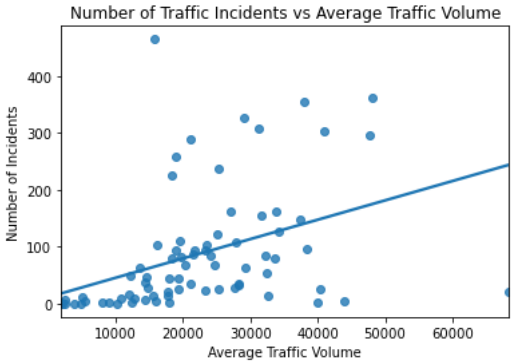


Analysis:

Viewing the regression plot, we cannot really see a trend between visibility and number of incidents. The data cluster stays relatively consistent and leveled throughout the length of the y-axis. This, coupled with a very weak Spearman Coefficient, leads us to believe that visibility is not an apparent factor that affects Incident numbers in the city. This was not expected, since logic would assume that reduced visibility would lead to more incidents.

Plot #3: The graph below shows the relation between number of incidents and average traffic volume. The graph was created by grouping average volume calculated for each grid in master_df and summing the indicent count for the grouped data. The relation between volume and incident count is also represented using a Seaborn regression plot.

In [20]:
```
#Generate Incidents vs Volume Graph
incidents_by_vol = master_df.groupby(['avg_volume']).traffic_incidents.sum().reset_index()
ax = sns.regplot('avg_volume', 'traffic_incidents', data=incidents_by_vol, ci=None)
ax.set(title='Number of Traffic Incidents vs Average Traffic Volume', xlabel='Average Traffic Volume',\
       ylabel='Number of Incidents')
plt.show()
```

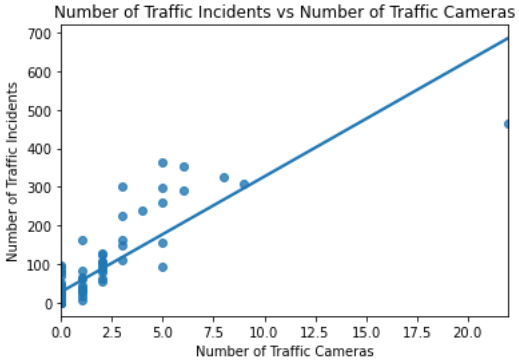


Analysis:

Viewing the regression plot, we can see an upward trend. This upward trend, coupled with a medium strength positive Spearman Coefficient, shows us that number of incidents generally increases as traffic volume increases. This was expected since logic would dictate that the more cars there are in an area, the more likely they will be involved in some form of incident.

Plot #4: The graph below shows the relation between number of incidents and number of traffic cameras. The graph was created by plotting the number of cameras calculated for each grid in master_df vs the indicent count for the grid data. The relation between cameras and incident count is then represented using a Seaborn regression plot. The reason the camera data is not grouped is because grouping by count seems to condense the information too much to visualize, and alters our results. By looking at the non-grouped data, we can see a clearer correlation between the two.

In [21]:
```
#Generate incident graphs by number of cameras
ax = sns.regplot('traffic_cameras', 'traffic_incidents', data=master_df, ci=None)
ax.set(title='Number of Traffic Incidents vs Number of Traffic Cameras',xlabel='Number of Traffic Cameras',\
      ylabel='Number of Traffic Incidents')
plt.show()
```

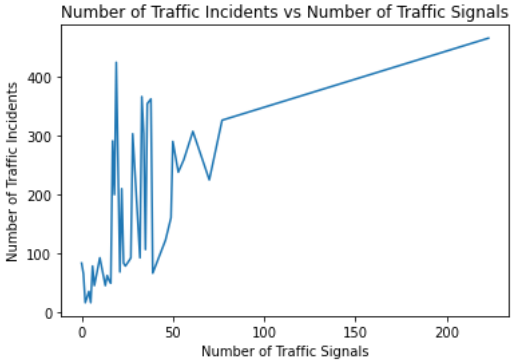


Analysis:

Viewing the regression plot, we can see an upward trend in the relationship between the number of cameras and incidents. This upward trend, combined with a strong Spearman Coefficient, shows us that incident numbers increase as the number of cameras increases. This was unexpected, as we made the assumption that an increase in the number of cameras in a given area would result in drivers being more careful and diligent, resulting in fewer incidents.

Plot #5: The graph below shows the relation between number of incidents and number of traffic signals. The graph was created by grouping number of signals calculated for each grid in master_df and summing the indicent count for the grouped data. The relation between signals and incident count is then represented using the Matplotlib line graphs.

In [22]:
```
#Generate incident graph based on traffic signals count
incidents_by_signals = master_df.groupby(['traffic_signals']).traffic_incidents.sum().reset_index()
ax = plt.plot('traffic_signals', 'traffic_incidents', data=incidents_by_signals)
plt.title('Number of Traffic Incidents vs Number of Traffic Signals')
plt.xlabel('Number of Traffic Signals')
plt.ylabel('Number of Traffic Incidents')
plt.show()
```

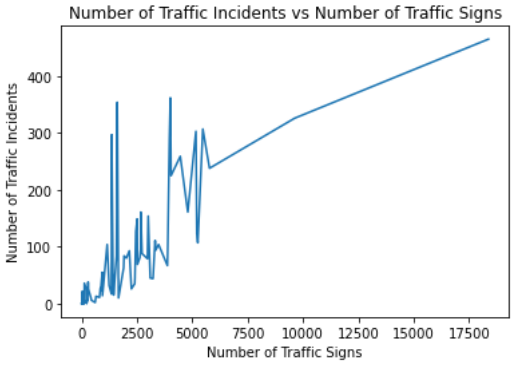


Analysis:

Viewing the line graph, we see an upward trend in the relationship between number of signals and incidents. While the beginning of the graph has rises and falls, the overall trend shows that as the number of traffic signals increases, the number of incidents typically increases. This visual, along with a very strong positive Spearman coefficient, confirms this is the case. This was expected since more signals typically means more stop and go traffic, and more intersections, both which can increase the likelihood of incidents occuring.

Plot #6: The graph below shows the relation between the number of incidents and the number of signs. The graph was created by grouping the number of signs for each grid in the master_df and summing the incident count for the grouped data. The relation between signs and incident count is then represented using the Matplotlib line graphs.

In [23]:
```
#Generate Incident vs Signs graph showing relation between the two
incidents_by_signs = master_df.groupby(['traffic_signs']).traffic_incidents.sum().reset_index()
ax = plt.plot('traffic_signs', 'traffic_incidents', data=incidents_by_signs)
plt.title('Number of Traffic Incidents vs Number of Traffic Signs')
plt.xlabel('Number of Traffic Signs')
plt.ylabel('Number of Traffic Incidents')
plt.show()
```



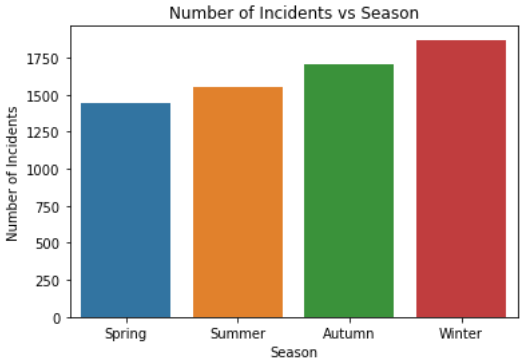
Analysis:

Viewing the line graph, we see an upward trend in the relationship between number of signs and incidents. While the beginning of the graph has rises and falls, the overall trend shows that as the number of traffic signs increases, the number of incidents typically increases. This visual, along with a very strong positive Spearman coefficient, confirms this is the case. This was not expected since we assumed more signs would result in more order, and safer driving from citizens. However, after viewing these results, we can make sense of them. The traffic signs include yields, stops, and multi-way stops. These all represent signs at city intersections, which are typically areas of high incident rates, as we have seen from our Signals vs Incidents data above.

Plot #7: The graph below represents the relation between number of incidents and the season of the year. This graph was created by including a new 'Season' column in a copied weather_and_incidents_df (leaves the original unchanged). The code block below iterates through the dataframe, and checks if the dates fall within the dates of a given season's range. The code then appends the season name to the Season column depending on which season it falls in. These seasons are then grouped while summing the incidents for each group. The data is then represented using Seaborn barplots.

Range of days within a season was taken from 'Northern Meteorological Seasons' found at: https://www.timeanddate.com/calendar/aboutseasons.html (https://www.timeanddate.com/calendar/aboutseasons.html)

In [24]:
```
#Add season information to database to allow for grouping
season_and_incidents_df = weather_and_incidents_df.copy()
pd.options.mode.chained_assignment = None #turn off SettingWithoutCopy Warning

season_and_incidents_df['Season'] = ''

for ind in season_and_incidents_df.index: #iterates through the dataframe

    if (season_and_incidents_df['Date'][ind] >= datetime.datetime(day=1, month=3, year=2018))\
    & (season_and_incidents_df['Date'][ind] <= datetime.datetime(day=31, month=5, year=2018)):
        season_and_incidents_df['Season'][ind] = 'Spring'

    elif (season_and_incidents_df['Date'][ind] >= datetime.datetime(day=1, month=6, year=2018))\
    & (season_and_incidents_df['Date'][ind] <= datetime.datetime(day=31, month=8, year=2018)):
        season_and_incidents_df['Season'][ind] = 'Summer'

    elif (season_and_incidents_df['Date'][ind] >= datetime.datetime(day=1, month=9, year=2018))\
    & (season_and_incidents_df['Date'][ind] <= datetime.datetime(day=30, month=11, year=2018)):
        season_and_incidents_df['Season'][ind] = 'Autumn'
    else:
        season_and_incidents_df['Season'][ind] = 'Winter'

#Group data by season and create graph
grouped_seasons = season_and_incidents_df.groupby(['Season']).sum().reset_index().reindex([1,2,0,3])
ax = sns.barplot(x='Season', y='Count', data=grouped_seasons).set(title='Number of Incidents vs Season',\
                                                                 ylabel='Number of Incidents')
plt.show()
```

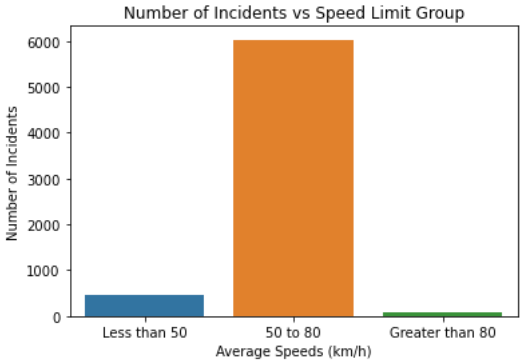


Analysis:

Viewing the barplot, we can see that the Winter season has the highest number of incidents for 2018. Winter is followed closely by Autumn. Calgary winters are known to be cold and snowy, which can explain the high number of incidents. Calgary Autumns also typically go into freeze/thaw cycles late in the night and early in the morning. It is also not uncommon for Calgary to experience sudden snowfalls during the Autumn season which catches drivers off-guard. The City can perform further studies to determine the cause of high incidents during the Winter and Autumn seasons, and possibly come up with ways of mitigating the causes.

Plot #8: The graph below represents the relation between different speed groups and the number of incidents. The graph is created by first adding a new 'Speed Limit Group' column to a copy of the master_df that extracts only the speed limit and incident columns. The code iterates through the dataframe and checks which group the average speed limit falls into. The groups are 'Less than 50', which represents more residential-heavy areas in the City, '50 to 80' which represent typical inner-city streets, and 'Greater than 80' which represent highway-type roads in the City. The data is then grouped based on these self-defined groups, where the incident count is summed about these groups. The number of incidents per speed limit group is then represented by a Seaborn barplot.

In [25]:
```
#Create incidents vs average speed limit (grouped) graph
incidents_by_speedlimit = master_df[['avg_speed_limit', 'traffic_incidents']].copy()
incidents_by_speedlimit['Speed Limit Group'] = ''

for ind in incidents_by_speedlimit.index: #iterates through dataframe
    if incidents_by_speedlimit['avg_speed_limit'][ind] <= 50:
        incidents_by_speedlimit['Speed Limit Group'][ind] = 'Less than 50'

    elif (incidents_by_speedlimit['avg_speed_limit'][ind] > 50)\
        & (incidents_by_speedlimit['avg_speed_limit'][ind] < 80):
        incidents_by_speedlimit['Speed Limit Group'][ind] = '50 to 80'

    else:
        incidents_by_speedlimit['Speed Limit Group'][ind] = 'Greater than 80'
incidents_by_speedlimit = incidents_by_speedlimit.groupby(['Speed Limit Group']).sum().reset_index().reindex([2,0,1])
ax = sns.barplot(x='Speed Limit Group', y='traffic_incidents', data=\
                 incidents_by_speedlimit).set(title='Number of Incidents vs Speed Limit Group',\
                                              ylabel='Number of Incidents', xlabel= 'Average Speeds (km/h)')
plt.show()
```

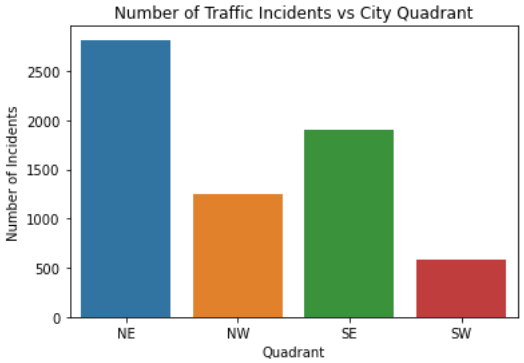


Analysis:

Viewing the barplot, we can see that inner-city roads see significantly more accidents than any other type of road. Residential areas having fewer incidents might be explained by assuming that slower speeds allow for drivers to be more in control of their vehicles. Highway roads having the fewest incidents might be explained by assuming that the constant speeds, mixed with little to no intersections and constant turning, results in smoother flowing traffic that reduces the chance of incidents occuring.

Plot #9: The graph below represents the relation between different city quadrants and the number of incidents. The graph is created by first adding a new 'Quadrant' column to a copy of the master_df that extracts only the column, row, and incidents columns. The code then iterates through the dataframes, checking which grids the data corresponds to, and appending either NW, NE, SW or SE to the Quadrant column depending on the grid location. The data is then grouped by quadrant, summing the incident count for each, and represented by a Seaborn barplot.

In [26]:
```
#Generate incident graph based on city quadrant
incidents_by_quadrant = master_df[['column', 'row', 'traffic_incidents']].copy()
incidents_by_quadrant['Quadrant'] = ''
for ind in incidents_by_quadrant.index:
    if (incidents_by_quadrant.row[ind] >= 0) & (incidents_by_quadrant.row[ind] <= 4) & (incidents_by_quadrant.column[ind] >= 0) & (incidents_by_quadrant.column[ind] <= 4):
        incidents_by_quadrant.Quadrant[ind] = 'NW'
    elif (incidents_by_quadrant.row[ind] >= 0) & (incidents_by_quadrant.row[ind] <= 4) & (incidents_by_quadrant.column[ind] >= 5) & (incidents_by_quadrant.column[ind] <= 9):
        incidents_by_quadrant.Quadrant[ind] = 'NE'
    elif (incidents_by_quadrant.row[ind] >= 5) & (incidents_by_quadrant.row[ind] <= 9) & (incidents_by_quadrant.column[ind] >= 0) & (incidents_by_quadrant.column[ind] <= 4):
        incidents_by_quadrant.Quadrant[ind] = 'SW'
    else:
        incidents_by_quadrant.Quadrant[ind] = 'SE'

incidents_by_quadrant = incidents_by_quadrant.groupby(['Quadrant']).sum().reset_index()
ax = sns.barplot(x='Quadrant', y='traffic_incidents', data= incidents_by_quadrant).set(title='Number of Traffic Incidents vs City Quadrant', ylabel='Number of Incidents', xlab
plt.show()
```



Analysis:

Viewing the barplot, we can see that the NE quadrant has significantly more incidents than any other quadrant in the city, followed by the SE quadrant with the second most incidents. These quadrants should be the focus of further analysis by the City to determine the root cause of such high incident rates. Possible ways to mitigate these incidents is to determine if permit testing is adequate at NE and SE registries, and whether drivers are obeying the traffic laws in these areas.

2018 Traffic Heatmap

Method of Choice:

In this analysis we use two different methods to generate heatmap:

- Folium
- Google Map API

We wanted to use two different methods to see which yields better results and is more user friendly. We found:

- Folium is easier to use without the need for an approved API, whereas Google needed their own API
- Google Map has more features and is more reliable compared to using Folium. We found there was a bug in Folium that doesn't take into account the weight of the data. We were able to fix this bug by installing a new Folium package online. This package is not readily available through the Folium download.

How The Heatmap Was Generated:

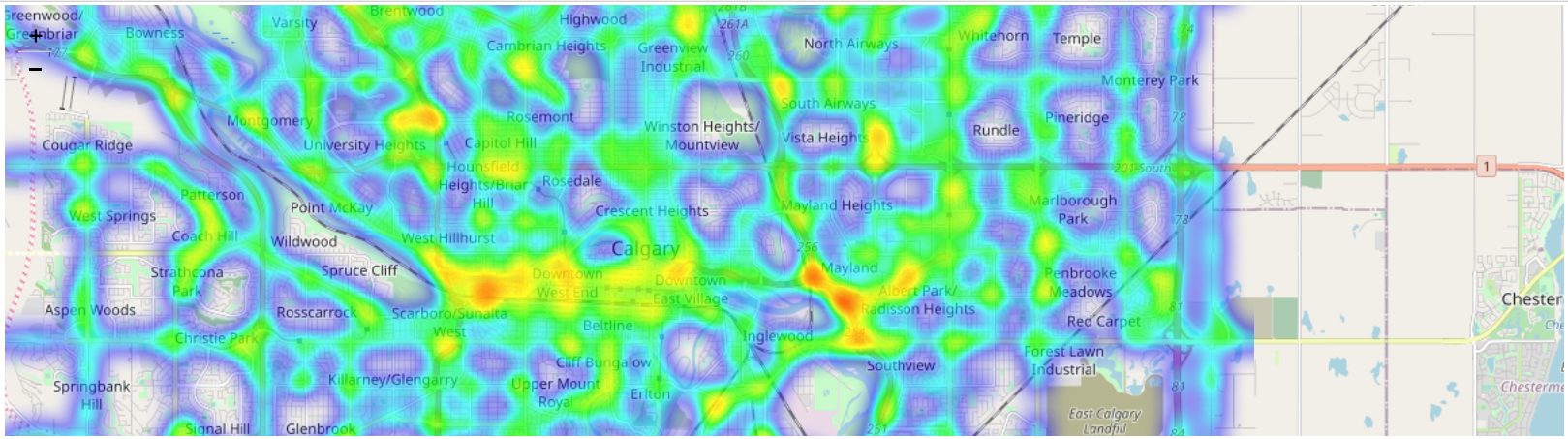
The heatmap was generated using a previous volume dataframe that we generated. We can take out the "lat", "Long" and "Volume" to use as latitude coordinates, longitude coordinates and weight for the heatmap.

Using Folium:

In [27]:
```
#Function to generate a base map, with default location set to calgary
def generateBaseMap(default_location=[51.0447, -114.0719], default_zoom_start=11):
    base_map = folium.Map(location=default_location, control_scale=True, zoom_start=default_zoom_start)
    return base_map
```

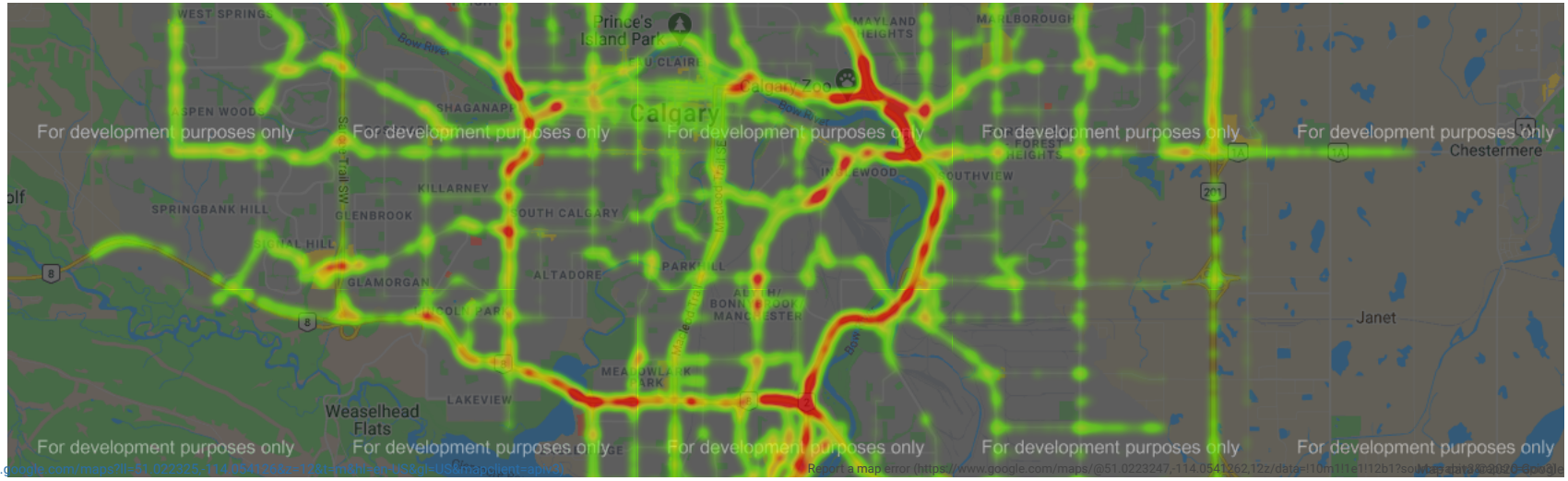
In [28]:
```
Trafficheat_Map= generateBaseMap()
# Traverse the traffic volume dataframe that contain lat/Long coordinates and their corresponding traffic volume.
# heatmap is weighted base on the amount of traffic volume
heat_data = [[row.Lat,row.Long, row.VOLUME] for index, row in trafficvolumeclean_df.iterrows()]

# Plot heatmap
HeatMap(heat_data, radius =12).add_to(Trafficheat_Map)
Trafficheat_Map
```

Out[28]:



Using Google API:

In [29]:
```
#configure API from google maps
gmaps.configure(api_key='AIzaSyDbcpzf8_b5n8CmjSPMUOME1z2jMGoCWTA')

#Initialize map with lat, long and Volume data from traffic volume dataframe
fig = gmaps.figure()
heatmap_layer = gmaps.heatmap_layer(
    trafficvolumeclean_df[['Lat', 'Long']], weights=trafficvolumeclean_df['VOLUME'],)

#Add heat map to main frame
fig.add_layer(heatmap_layer)
fig
```



Analysis:

Traffics seem to accumulate on highway and interesections, especially at the ones surrounding Downtown. Deerfoot Trail seems to be the worst road to be on during traffic hours, I won't be taking this route to work in the future. Crowchild Trail looks to be the second worst with most traffic at intersections of Bow Trail and Shaganappi Trail. All these roads are most commonly used to enter the Downtown core, where majority of Calgarians work, so it comes as no surprise that these areas see increased amounts of traffic volume.

Speed Limit Map for the City of Calgary

To generate the speed limit map, we first parse our Lat, Long data of multilines into proper geojson format. We then use folium.Geojson to draw these multilines on the map. The Speed categories and color gradient is handled by the style function that base the color of the lines on the magnitude of the speed limit.

In [41]:
```
# Color gradient for each speed limit category
linear = cm.LinearColormap(['green','yellow','red'], vmin=20., vmax=110.)
linear.caption = 'Speed Limit (Km/h)'
```

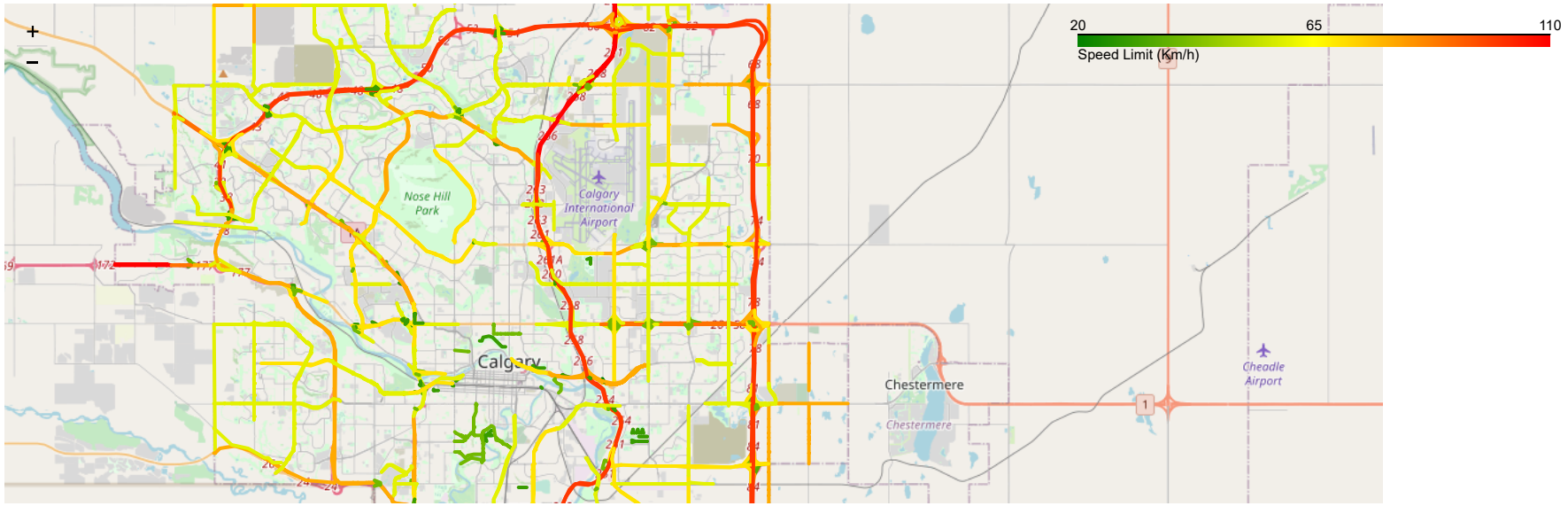
In [42]:
```
# function to get unique values of speed limit
def unique(list1):
    x = np.array(list1)
    return np.unique(x)
#Array of unique speed limit taken from speed limit data
uniqueSpeed = unique(speedlimitclean_df.SPEED)
```

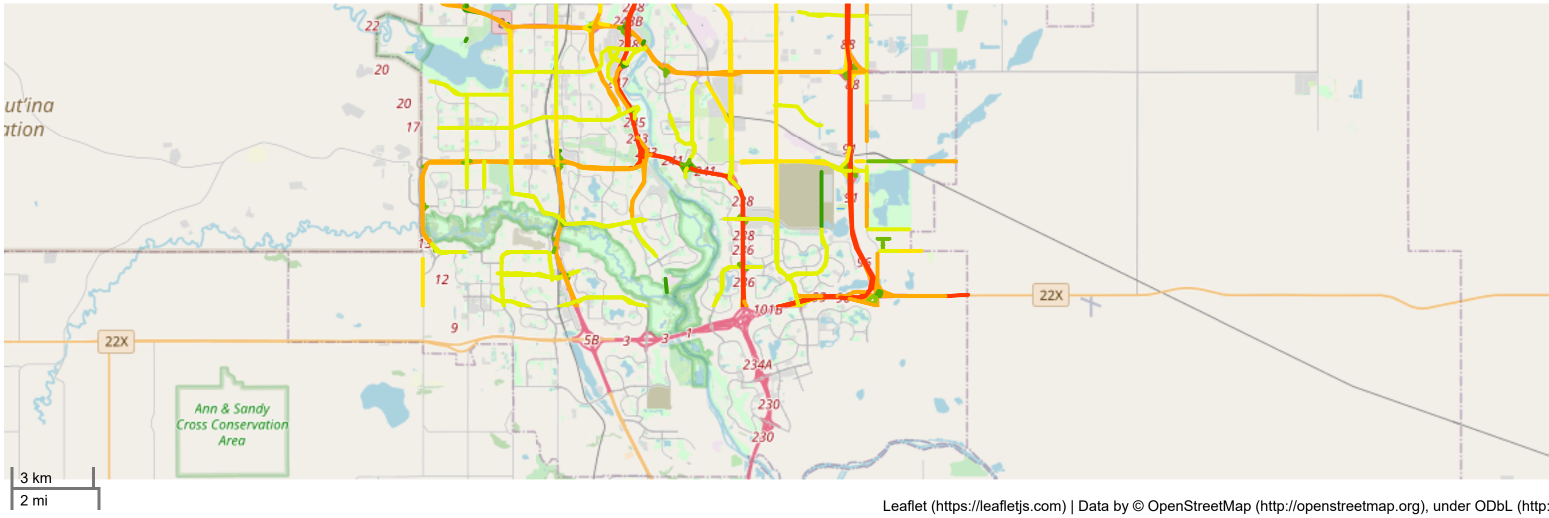
In [43]:
```
#Array of unique speed limit
uniqueSpeed = unique(speedlimitclean_df.SPEED)
```

In [44]:
```
# Function to define the color of each speed limit category
def style_function(feature):
    speed_category = int(feature['properties']['speedLimit'])
    return {
            'fillOpacity': 1,
            'weight': 3,

            'color': linear(speed_category)
    }
```

In [45]:
```
# Generate base map
Volume_Map= generateBaseMap()

#Initialize list and dicts to convert Long/Lat data into proper geojson format
speedlimit_category_dict = {}
speedlimit_category_list = []
speedlimit_category_dict_merge = {}

# Loop through every rows in the speed limit dataframe, and convert the data for Lat and Long coordinates into proper geojson format
for i, j in speedlimit_df.iterrows():
    speedlimi_multiline = j.multiline
    # do replace operations to get the data under "multiline" column to get into proper geojson format, take into account the multiple multilines within one cell
    speedlimi_multiline = speedlimi_multiline.replace('((', '[[[').replace('))', ']]]').replace('(', '[').replace(')', ']').replace('MULTILINESTRING', '').replace(', ', '],[')
    speedlimi_multiline = speedlimi_multiline.strip()
    speedlimi_multiline =speedlimi_multiline.replace(' ', ',')
    # Convert each list into a json object
    speedlimit_converttoList = json.loads(speedlimi_multiline)
    for speed in range(0, len(uniqueSpeed)):
        if (str(j.SPEED) == str(uniqueSpeed[speed])):
            #generate a dict for every speed limit category
            speedlimit_category_dic = {'type': "Feature",
                                'geometry':{'type':'MultiLineString', 'coordinates': speedlimit_converttoList },
                                'properties': {'speedLimit': str(uniqueSpeed[speed]) }}
            #append to the general list that has all the speed limit categories
            speedlimit_category_list.append(speedlimit_category_dic)

# merge overall speedlimit data with coordinates into a dictioary that can be loaded into geojson
speedlimit_category_dict_merge = {"type": "FeatureCollection", "features": speedlimit_category_list}

# load data into geojson object
s = json.dumps(speedlimit_category_dict_merge)
g1 = geojson.loads(s)

#apply style function to categorize speed limit
folium.GeoJson(g1, name ='Speed Limit Map', style_function=style_function).add_to(Volume_Map)
# add a legend color gradient for each speed limit
Volume_Map.add_child(linear)
Volume_Map
```

Out[45]:





Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).

Conclusion

Based on our various calculations and analysis, we have observed that some areas of our initial hypothesis have been proven true, and others have been proven false. After our in-depth analysis, we have observed the following:

- Core Downtown (grid 5,4) has the greatest number of incidents overall in the City for 2018.
- Inner-city streets (Speed limits of 50-80 kmph) have significantly more incidents compared to other speeds.
- As traffic volume increases, the number of incidents increases.
- As the number of traffic cameras increases, the number of incidents increases.
- As the number of road signs increases, the number of incidents increases.
- As the number of road signals increases, the number of incidents increases.
- As the temperature decreases, the number of incidents increases, with frequent incidents occurring approximately in the +7 to -7 temperature ranges.
- Visibility does not seem to have an effect on the number of incidents in the City of Calgary.
- The NE quadrant of the City sees the highest number of incidents out of all quadrants for 2018.
- The Winter season sees the highest number of incidents out of all seasons for 2018.
- Major roads leading into the City center (Deerfoot, Crowchild, etc) see the most traffic volume.

It is important to note that our analysis observations are limited to those in 2018. To further solidify our observations and ensure they are accurate, the same analysis should be performed on data observed over multiple years to more accurately draw conclusions.

(

Appendix: grid_df

	0	1	2	3	4	5	6	7	8	9
0	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.0878505, 'max_long': -	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -113.9966723, 'max_long': -	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -113.9510832, 'max_long': -	{'min_lat': 51.175464700000006, 'max_lat': 51.212425, 'min_long': -113.9054941, 'max_long': -
1	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 51.1385044, 'max_lat': 51.175464700000006, 'min_long': -113.9054941, 'max_long': -113.859905}
2	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 51.1015441, 'max_lat': 51.1385044, 'min_long': -113.9054941, 'max_long': -113.859905}
3	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 51.0645838, 'max_lat': 51.1015441, 'min_long': -113.9054941, 'max_long': -113.859905}
4	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 51.027623500000004, 'max_lat': 51.0645838, 'min_long': -113.9054941, 'max_long': -113.859905}
5	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 50.9906632, 'max_lat': 51.027623500000004, 'min_long': -113.9054941, 'max_long': -113.859905}
6	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 50.953702899999996, 'max_lat': 50.9906632, 'min_long': -113.9054941, 'max_long': -113.859905}
7	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 50.9167426, 'max_lat': 50.953702899999996, 'min_long': -113.9054941, 'max_long': -113.859905}
8	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 50.8797823, 'max_lat': 50.9167426, 'min_long': -113.9054941, 'max_long': -113.859905}
9	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.315796, 'max_long': -114.2702069}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.2702069, 'max_long': -114.2246178}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.2246178, 'max_long': -114.1790287}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.1790287, 'max_long': -114.1334396}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.1334396, 'max_long': -114.0878505}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.0878505, 'max_long': -114.0422614}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -114.0422614, 'max_long': -113.9966723}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -113.9966723, 'max_long': -113.9510832}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -113.9510832, 'max_long': -113.9054941}	{'min_lat': 50.842822, 'max_lat': 50.8797823, 'min_long': -113.9054941, 'max_long': -113.859905}

Appendix: master_df

	column	row	avg_speed_limit	avg_volume	traffic_cameras	traffic_signals	traffic_signs	traffic_incidents
0	0	0			0	0	0	0
1	0	1			0	0	0	0
2	0	2			0	0	43	0
3	0	3	110	44000	0	0	109	5
4	0	4			0	0	0	0
5	0	5			0	0	0	0
6	0	6			0	0	0	0
7	0	7			0	0	0	0
8	0	8			0	0	0	0
9	0	9			0	0	0	0
10	1	0	75.07936508	3935.483871	0	0	18	0
11	1	1	64.94252874	12898.77301	0	7	1667	10
12	1	2	61.75438596	25233.19328	0	10	2249	26
13	1	3	70.859375	32582.8877	0	0	942	14
14	1	4	60	8000	0	0	81	3
15	1	5	73.96825397	17906.97674	0	0	59	1
16	1	6			0	0	0	0
17	1	7			0	0	0	0
18	1	8			0	0	0	0
19	1	9			0	0	0	0
20	2	0	71.18421053	4842.105263	0	0	82	0
21	2	1	64.84554815	28350.93168	1	10	1243	32
22	2	2	64.86463257	20248.64539	1	39	3875	67
23	2	3	63.48336595	17951.36778	1	17	3225	44
24	2	4	61.0202864	19318.89081	0	22	3100	45
25	2	5	65.70743405	15631.57895	0	2	1446	15
26	2	6			0	0	0	0
27	2	7			0	0	2	0
28	2	8	70	12352.94118	0	0	122	1
29	2	9			0	0	0	0
30	3	0	63.63636364	5068.100358	0	1	808	11
31	3	1	63.7347561	24163.93443	1	22	1917	84
32	3	2	67.98728814	21647.1846	2	32	3888	93

33	3	3	65.62115621	25056.28141	2	46	5203	123
34	3	4	66.25495376	27780.4878	2	35	5248	107
35	3	5	73.40909091	24648.82943	0	21	2522	69
36	3	6	60	2571.428571	0	1	228	0
37	3	7	72.18533887	10875.91241	0	0	430	9
38	3	8	70.45454545	14358.97436	0	0	442	6
39	3	9			0	0	0	0
40	4	0	60	1961.206897	0	2	613	2
41	4	1	65.63218391	33696.93095	2	24	2976	79
42	4	2	71.52993348	23391.00346	0	7	859	23
43	4	3	61.79112272	27171.07811	3	49	4796	161
44	4	4	57.83313325	29086.92308	8	77	9621	326
45	4	5	59.43378995	41029.41176	3	28	5163	303
46	4	6	62.34993614	21442.66667	2	26	2729	88
47	4	7	64.78494624	14508.22669	0	13	2410	46
48	4	8	62.04255319	12285.47579	0	16	2411	50
49	4	9		2000	0	0	58	0
50	5	0	75.04950495	5485.294118	0	1	273	5
51	5	1	63.56513676	18897.17046	0	33	3347	94
52	5	2	70.43478261	33751.9084	1	33	2673	161
53	5	3	61.48484848	25199.29141	4	53	5777	238
54	5	4	44.25901202	15870.37037	22	223	18374	465
55	5	5	58.44332176	31340.81463	9	61	5475	307
56	5	6	62.78884462	18290.61224	3	70	4039	225
57	5	7	62.06577119	31692.23573	5	34	3008	154
58	5	8	63.67415194	19579.35735	3	33	3320	111
59	5	9	59.67320261	2509.433962	1	1	247	7
60	6	0	88.55238095	68347.82609	0	0	44	22
61	6	1	72.75438596	32529.95392	2	15	933	55
62	6	2	67.54098361	19294.11765	0	6	220	25
63	6	3	67.25333333	37966.82464	6	36	1610	354
64	6	4	74.37106056	48142.14286	5	38	4009	362
65	6	5	70.82397004	47646.04462	5	19	1360	297
66	6	6	68.22074566	37399.01881	3	34	2511	149

67	6	7	66.55727156	38376.11408	0	18	2667	96
68	6	8	71.49141631	27660.76696	1	6	1437	27
69	6	9	66.20689655	11914.8265	1	5	1348	17
70	7	0	69.62298025	17750	0	0	34	20
71	7	1	68.57015192	29355.32995	2	14	1414	63
72	7	2	61.29476584	18288.26152	0	17	2022	80
73	7	3	59.10411622	21158.99123	6	50	3966	290
74	7	4	75.69165143	19009.92556	5	56	4457	259
75	7	5	62.31227652	13521.73913	1	17	1899	63
76	7	6	70.02633889	16257.98212	2	18	1155	104
77	7	7	69.9034062	23227.1028	5	27	2156	93
78	7	8	67.21440397	23421.97802	2	17	3477	104
79	7	9	80.44554455	17805.12821	0	7	650	13
80	8	0	98.83333333	40000	0	0	4	1
81	8	1	83.72116349	40318.58407	0	3	304	26
82	8	2	76.88504326	19832.71375	0	22	2632	81
83	8	3	71.12003781	34263.04802	2	19	2451	127
84	8	4	69.50310559	32221.95704	0	23	1577	84
85	8	5	72.13842975	28179.48718	0	4	124	36
86	8	6	72.98299845	14462.26415	1	1	295	38
87	8	7	64.29906542	14836.06557	0	6	850	27
88	8	8	70.76843198	21129.65723	1	10	2400	35
89	8	9			0	0	25	0
90	9	0			0	0	0	0
91	9	1			0	0	12	0
92	9	2			0	0	0	0
93	9	3			0	0	14	0
94	9	4	80	16000	0	1	83	5
95	9	5	80		0	0	6	0
96	9	6		10200	0	0	86	0
97	9	7	58.07531381		0	0	95	0
98	9	8	76.34146341	9000	0	0	51	2
99	9	9			0	0	4	0