Project Phase II

Goal

Finding the features that can affect the number of accident in Calgary City. All data provided is based on https://data.calgary.ca/browse (https://data.calgary.ca/browse)

Features

Road Features

1. Road Speed

```
https://data.calgary.ca/Health-and-Safety/Speed-Limits-Map/rbfp-3tic
```

2. Average Traffic Volume

```
2018 (Traffic_Volumes_for_2018.csv)
```

3. Road Signals

```
a. Traffic Signals (Traffic_Signals.csv)
b. Traffic Signs (Traffic_Signs.csv)
c. Traffic cameras (Traffic_Camera_Locations.csv)
```

Weather Features

```
• Temperature
• Visibility
Ref: climate.weather.gc.ca
```

Marking

```
• Analysing Data- (Visualization: 10 Marks + Conclusion: 5 Marks) (15 Marks)
• Visualizing speed limit (5 Marks)
• Visualizing Traffic heatmap (5 Marks)
• Project Demo (5 Marks)
• Total Mark: 30 Marks
```

Due date

To upload the report(Presentation Slides) and source code: 13-Aug 11:59 midnight.

1. Data Preparation

1.1 Data Cleaning and Data Merging

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import geopandas as gpd
from geopandas import GeoDataFrame
import matplotlib.pyplot as plt
import seaborn as sns
import re
from shapely.geometry import Polygon
import folium
import shapely.wkt
from shapely.geometry import Point, Polygon
import math
```

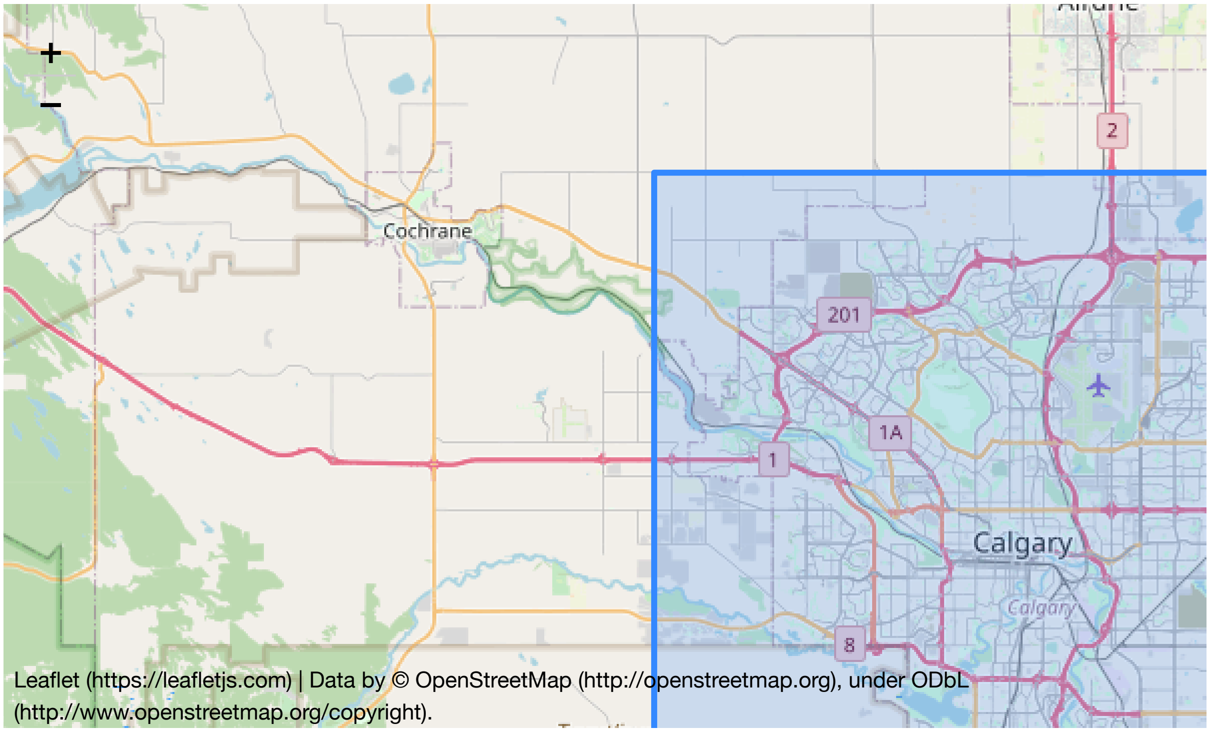
1.1.1 Load City Boundary Geometry and Create Boundary Geometry Object

In [2]:
```
# load boundary from csv and store long/lat separately in df
city_boundary_df = pd.read_csv('City_Boundary_layer.csv')
geom = city_boundary_df.iloc[0]['the_geom']
g=re.split("POLYGON", geom)[1].strip()
temp = pd.DataFrame(re.sub('[()]', '', g).split(', '))
boundary_coordinates_df=temp[0].str.split(" ", n = 1, expand = True).ast
ype(float)
boundary_coordinates_df.columns=['Longitude', 'Latitude']
#boundary_coordinates_df.describe()
```

In [3]:
```
# boundaries in four directions w, e, n, s
w = boundary_coordinates_df['Longitude'].max()
e = boundary_coordinates_df['Longitude'].min()
n = boundary_coordinates_df['Latitude'].max()
s = boundary_coordinates_df['Latitude'].min()
polygon_geom = Polygon([(w, n), (w, s), (e, s), (e, n)])
crs = 'epsg:4326'
city_boundary_polygon = gpd.GeoDataFrame(index=[0], crs=crs, geometry=[p
olygon_geom])
```

In [4]:
```
# create map object with folium
citymap = folium.Map(location=[51.03011, -114.08529], zoom_start = 10)
# add city boundary Polygon to map object
folium.GeoJson(city_boundary_polygon).add_to(citymap)
# add coordinate popups
folium.LatLngPopup().add_to(citymap)
citymap
```

Out[4]:



1.1.2 Load and Reorganize Features Data

In [5]:
```
# save only useful columns into DataFrame
speed_df = pd.read_csv('Speed_Limits.csv', usecols=['SPEED', 'multiline'
])

volume_df = pd.read_csv('Traffic_Volumes_for_2018.csv',
                        usecols=['YEAR', 'VOLUME', 'multilinestring'])
volume_df = volume_df[volume_df['YEAR']==2018].drop(columns=['YEAR'])

cameras_df = pd.read_csv('Traffic_Camera_Locations.csv', usecols=['longi
tude', 'latitude'])

signals_df = pd.read_csv('Traffic_Signals.csv',
                         usecols=['longitude', 'latitude', 'Point', 'Cou
nt'])

signs_df = pd.read_csv('Traffic_Signs.csv', usecols=['BLADE_TYPE', 'POIN
T'])

incident_df = pd.read_csv('Traffic_Incidents.csv', usecols=['START_DT',
'Longitude', 'Latitude'])
incident_df = incident_df[incident_df['START_DT'].str.contains('2018')]
incident_df['DateTime'] = pd.to_datetime(incident_df['START_DT'])
incident_df['Date'] = incident_df['DateTime'].dt.strftime('%Y-%m-%d')
incident_df['Time'] = incident_df['DateTime'].dt.strftime('%H:%M')
incident_df=incident_df.sort_values(by=['Date']).reset_index(drop=True)
```

1.2 Divide City Area into 10X10 Grids

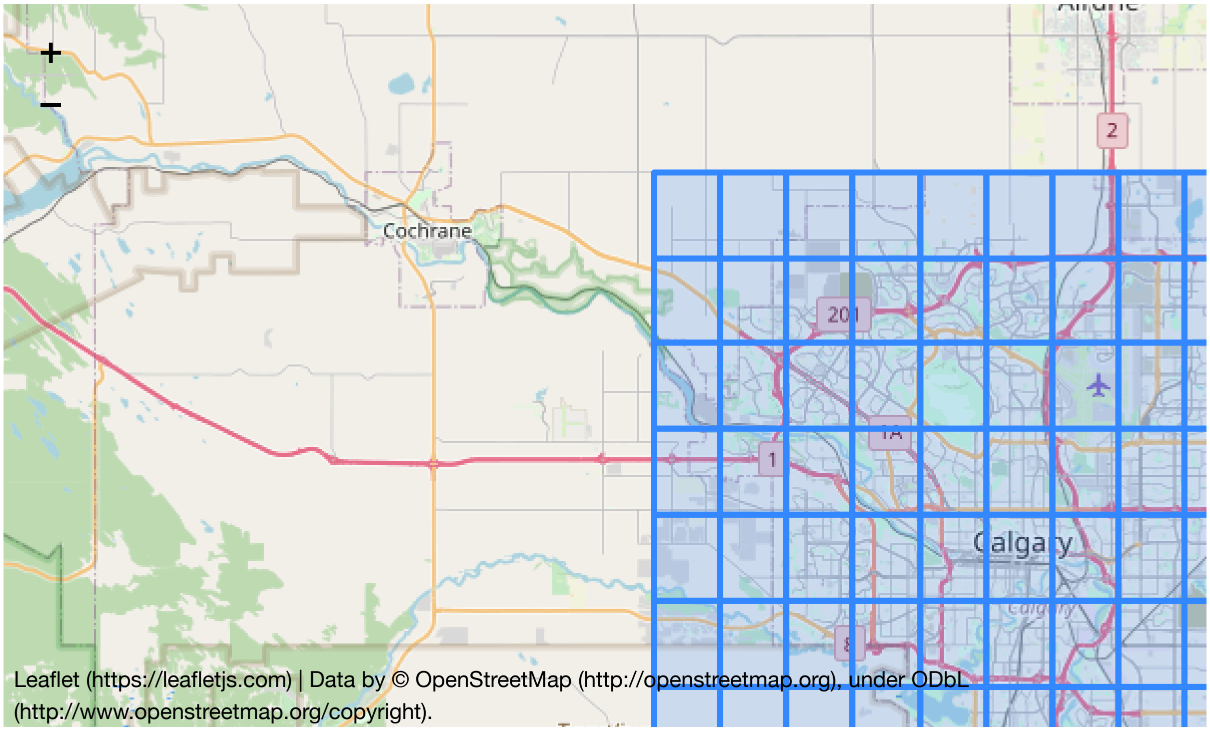
- Read the calgary boundary from City_Boundary_layer.csv.
- Draw a rectangle on Calgary map that shows the boundary of Calgary City.
- Divide calgary to a 10x10 matrix of areas.
  You need to investigate each area according to different features.

In [6]:
```
# generate 11*1 1-d array, listing longitude from w to e
x = np.linspace(w, e, num=11)[::-1]
# generate 1*11 1-d array, listing latitude from n to s
y = np.linspace(n, s, num=11)
# generate 11*11 2-d array, listing longitude and latitude separately in
xv and yv
xv, yv = np.meshgrid(x, y, indexing='ij')
# Ref: https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.h
tml

# grids are named following the pattern "grid" + x + y,
# e.g. northwest corner is named grid00 and southeast corner grid is nam
ed grid99
grid_names=[]
west_boundary=[]
east_boundary=[]
north_boundary=[]
south_boundary=[]
polygons=[]
for i in range(10):
    for j in range(10):
        # write grid names into df
        grid_names.append('grid{}{}'.format(i,j))
        # write 4 boundaries into df
        west_boundary.append(x[i])
        east_boundary.append(x[i+1])
        north_boundary.append(y[j])
        south_boundary.append(y[j+1])
        # find the nw, sw, se, ne corner coordinates and store in df
        grid_corners=[
            (xv[i][j], yv[i][j]),
            (xv[i][j+1], yv[i][j+1]),
            (xv[i+1][j+1], yv[i+1][j+1]),
            (xv[i+1][j], yv[i+1][j])
        ]
        polygons.append(Polygon(grid_corners))

grid_df = pd.DataFrame({'Grid Names':grid_names,
                       'West Boundary':west_boundary,
                       'East Boundary':east_boundary,
                       'North Boundary':north_boundary,
                       'South Boundary':south_boundary})

polygon_gdf = gpd.GeoDataFrame(crs='epsg:4326', geometry=polygons)
grid_df = pd.concat([polygon_gdf['geometry'], grid_df], axis=1)
```

In [7]:

```
# Create separate map object to display grids
gridmap = folium.Map(location=[51.03011, -114.08529], zoom_start = 10)
folium.GeoJson(polygon_gdf).add_to(gridmap)
gridmap
```

Out[7]:



2. Data Aggregation

For Each area (grid) calculate the following features: (15 Marks)

- Average speed limit
- Average Traffic volume
- Average number of traffic cameras
- Number of Traffic Signals
- Number of Traffic Signs
- Daily Weather Condition
  - Temperature
  - Visibility
- Target: Average number of Traffic accidents
- Analyse the data and interpret what is the relation between the number of accidents and the above feature in 2018. (Use different techniques of visualizing data like histogram, scatter plot, line graph, heatmap to interpret your answer)

2.1 Analysing a specific group of data

2.1.1 Average Speed Limit

is calculated to be a road length weighted speed limit value.

1. Use x = a.intersection(b) to returns a intersected geometry of MULTILINESTRING and Polygon.
2. Use geometry_object.length to return the length of the intersected geometry.
3. (speed limit * road segment length)/(total length of all segments) in grid

In [8]:
```
# (speed limit * road segment length) / total length of all segments in
 cell
grid_df.assign(Speed_Limit=np.nan)
i=0
for polygon in grid_df['geometry']:
    weighted_total_speed=0
    total_length=0
    j=0
    for m in speed_df['multiline']:
        # convert string m to MULTILINESTRING object MultiLineString
        MultiLineString = shapely.wkt.loads(m)
        intersection=polygon.intersection(MultiLineString)
        speed=speed_df.iloc[j, 0]
        weighted_total_speed+=speed*intersection.length
        total_length+=intersection.length
        j+=1
    if total_length>0:
        grid_df.at[i, 'Speed_Limit']=math.trunc(weighted_total_speed/tot
al_length)
    else:
        grid_df.at[i, 'Speed_Limit']=math.trunc(0)
    i+=1
```

In [9]:
```
grid_df.dtypes
```

Out[9]:
```
geometry          geometry
Grid Names          object
West Boundary      float64
East Boundary      float64
North Boundary     float64
South Boundary     float64
Speed_Limit        float64
dtype: object
```

2.1.2 Average Traffic Volume

is calculated to be a road length weighted traffic volume value.

1. Use x = a.intersection(b) to returns a intersected geometry of MULTILINESTRING and Polygon.
2. Use geometry_object.length to return the length of the intersected geometry.
3. (traffic volume * road segment length)/(total length of all segments) in grid

In [10]:
```
grid_df.assign(Traffic_Volume=np.nan)
i=0
for polygon in grid_df['geometry']:
    weighted_total_volume=0
    total_length=0
    j=0
    for m in volume_df['multilinestring']:
        # convert string m to MULTILINESTRING object MultiLineString
        MultiLineString = shapely.wkt.loads(m)
        intersection=polygon.intersection(MultiLineString)
        volume=volume_df.iloc[j, 0]
        weighted_total_volume+=volume*intersection.length
        total_length+=intersection.length
        j+=1
    if total_length>0:
        grid_df.at[i, 'Traffic_Volume']=math.trunc(weighted_total_volume
/total_length)
    else:
        grid_df.at[i, 'Traffic_Volume']=math.trunc(0)
    i+=1
```

2.1.3 Total (not average) Number of Traffic Cameras

In [11]:
```
grid_df.assign(Traffic_Cameras=np.nan)
idx=0
for polygon in grid_df['geometry']:
    count=0
    for long, lat in zip(cameras_df['longitude'], cameras_df['latitude'
]):
        point = Point(long, lat)
        # check if points are inside of grid polygon
        if point.within(polygon):
            count+=1
    grid_df.at[idx, 'Traffic_Cameras'] = count
    idx+=1
```

2.1.4 Total Number of Traffic Signals

In [12]:
```
grid_df.assign(Traffic_Signals=np.nan)
idx=0
for polygon in grid_df['geometry']:
    count=0
    for long, lat in zip(signals_df['longitude'], signals_df['latitude'
]):
        point = Point(long, lat)
        if point.within(polygon):
            count+=1
    grid_df.at[idx, 'Traffic_Signals'] = count
    idx+=1
```

2.1.5 Total Number of Traffic Signs

In [13]:
```
grid_df.assign(Traffic_Signs=np.nan)
idx=0
for polygon in grid_df['geometry']:
    count=0
    for p in signs_df['POINT']:
        # convert string p to Point object point
        point = shapely.wkt.loads(p)
        if point.within(polygon):
            count+=1
    grid_df.at[idx, 'Traffic_Signs'] = count
    idx+=1
```

2.2 Traffic Accidents Analysis

2.2.1 Total (not average) number of Traffic Accidents

In [14]:
```
grid_df.assign(Accidents=np.nan)
idx=0
for polygon in grid_df['geometry']:
    count=0
    for long, lat in zip(incident_df['Longitude'], incident_df['Latitud
e']):
        point = Point(long, lat)
        if point.within(polygon):
            count+=1
    grid_df.at[idx, 'Accidents'] = count
    idx+=1
grid_df
```

Out[14]:

	geometry	Grid Names	West Boundary	East Boundary	North Boundary	South Boundary	Speed_Limit	Traffic_Vol
0	POLYGON ((-114.31580 51.21243, -114.31580 51.1...	grid00	-114.315796	-114.270207	51.212425	51.175465	0.0	
1	POLYGON ((-114.31580 51.17546, -114.31580 51.1...	grid01	-114.315796	-114.270207	51.175465	51.138504	0.0	
2	POLYGON ((-114.31580 51.13850, -114.31580 51.1...	grid02	-114.315796	-114.270207	51.138504	51.101544	0.0	
3	POLYGON ((-114.31580 51.10154, -114.31580 51.0...	grid03	-114.315796	-114.270207	51.101544	51.064584	109.0	440
4	POLYGON ((-114.31580 51.06458, -114.31580 51.0...	grid04	-114.315796	-114.270207	51.064584	51.027624	0.0	
...	...	...	...	...	...	...	...	
95	POLYGON ((-113.90549 51.02762, -113.90549 50.9...	grid95	-113.905494	-113.859905	51.027624	50.990663	80.0	
96	POLYGON ((-113.90549 50.99066, -113.90549 50.9...	grid96	-113.905494	-113.859905	50.990663	50.953703	0.0	121
97	POLYGON ((-113.90549 50.95370, -113.90549 50.9...	grid97	-113.905494	-113.859905	50.953703	50.916743	60.0	
98	POLYGON ((-113.90549 50.91674, -113.90549 50.8...	grid98	-113.905494	-113.859905	50.916743	50.879782	80.0	90
99	POLYGON ((-113.90549 50.87978, -113.90549 50.8...	grid99	-113.905494	-113.859905	50.879782	50.842822	0.0	

100 rows × 12 columns

2.2.2 Daily Weather Conditions

In [15]:
```
def download_weather_data(month=1, daily=True):
    """ returns a DataFrame with weather data from climate.weather.gc.c
a"""
    # url string with station 51430, year of 2018 and user defined mont
h, and daily or hourly option
    url_template = "https://climate.weather.gc.ca/climate_data/bulk_data
_e.html?format=csv&stationID=50430&Year=2018&Month={month}&Day=14&timefr
ame={tf}&submit=Download+Data"

    if daily == False:
        tf = 1
    else: # hourly
        tf = 2
    url = url_template.format(month=month, tf = tf)

    # 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=T
rue)

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

    return weather_data
```

In [16]:
```
daily_weather_df=pd.DataFrame()
for mo in range(1,13):
    hourly_weather_df=download_weather_data(month = mo, daily = False)
    daily_average_weather_df=hourly_weather_df.groupby(by='Day').mean()
    daily_weather_df=pd.concat([daily_weather_df, daily_average_weather_
df])
pd.set_option('display.max_rows', 400)
daily_weather_df=daily_weather_df.reset_index()
# Add Date type object to Date column
daily_weather_df['Date']=pd.to_datetime(daily_weather_df[['Day', 'Month'
, 'Year']]).dt.strftime('%Y-%m-%d')
```

In [17]:
```
# Add weather conditions(daily average temperature and visibility) to in
cident_df
incident_df=incident_df.assign(Temperature=np.nan)
incident_df=incident_df.assign(Visibility=np.nan)
row=0
for date in daily_weather_df['Date']:
    index=incident_df[incident_df['Date']==date].index
    incident_df.loc[index, ['Temperature']]=daily_weather_df.iloc[row, 6
]
    incident_df.loc[index, ['Visibility']]=daily_weather_df.iloc[row, -9
]
    row+=1
```

In [18]:

```
# match each accident to grid, and add grid name and grid polygon
temp_df=pd.DataFrame()
for long, lat in zip(incident_df['Longitude'], incident_df['Latitude']):
    point = Point(long, lat)
    # check which Polygon this Points is located in
    idx_grid=0
    temp=pd.DataFrame([np.nan])
    for polygon in grid_df['geometry']:
        if point.within(polygon):
            # append grid information that matched the coordinate of Poi
nt
            temp=grid_df.iloc[idx_grid]
        idx_grid+=1
    temp_df = pd.concat([temp_df, temp], axis=1)
temp_df=temp_df.T.reset_index(drop=True).drop([0], axis=1)
incident_df=pd.concat([incident_df, temp_df], axis=1)
incident_df
```

Out[18]:

	START_DT	Longitude	Latitude	DateTime	Date	Time	Temperature	Visibility	Accid
0	01/01/2018 02:15:43 PM	-114.129824	51.165068	2018-01-01 14:15:43	2018-01-01	14:15	-16.683333	42.570833	
1	01/01/2018 04:28:29 PM	-114.083473	51.049899	2018-01-01 16:28:29	2018-01-01	16:28	-16.683333	42.570833	
2	01/01/2018 02:45:41 PM	-114.068263	51.041568	2018-01-01 14:45:41	2018-01-01	14:45	-16.683333	42.570833	
3	01/01/2018 11:13:46 AM	-114.025651	50.888942	2018-01-01 11:13:46	2018-01-01	11:13	-16.683333	42.570833	
4	01/01/2018 10:35:28 AM	-114.170252	51.123058	2018-01-01 10:35:28	2018-01-01	10:35	-16.683333	42.570833	
...	...	...	...	...	...	...	...	...	
6562	12/31/2018 01:33:45 PM	-114.033912	50.948342	2018-12-31 13:33:45	2018-12-31	13:33	-11.916667	38.145833	
6563	12/31/2018 01:14:38 PM	-113.994847	51.033832	2018-12-31 13:14:38	2018-12-31	13:14	-11.916667	38.145833	
6564	12/31/2018 08:00:47 PM	-114.079493	51.054765	2018-12-31 20:00:47	2018-12-31	20:00	-11.916667	38.145833	
6565	12/31/2018 03:34:01 PM	-113.989219	51.067086	2018-12-31 15:34:01	2018-12-31	15:34	-11.916667	38.145833	
6566	12/31/2018 08:55:03 PM	-114.076108	51.048764	2018-12-31 20:55:03	2018-12-31	20:55	-11.916667	38.145833	

6567 rows × 20 columns

In [19]:

```
# passing DataFrames to part 2, so that when implementing analysis and v
isualization
# we won't need to run the data part again
grid_df.to_csv('grid_df.csv', index=False)
incident_df.to_csv('incident_df.csv', index=False)
speed_df.to_csv('speed_df.csv', index=False)
volume_df.to_csv('volume_df.csv', index=False)
```