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
```

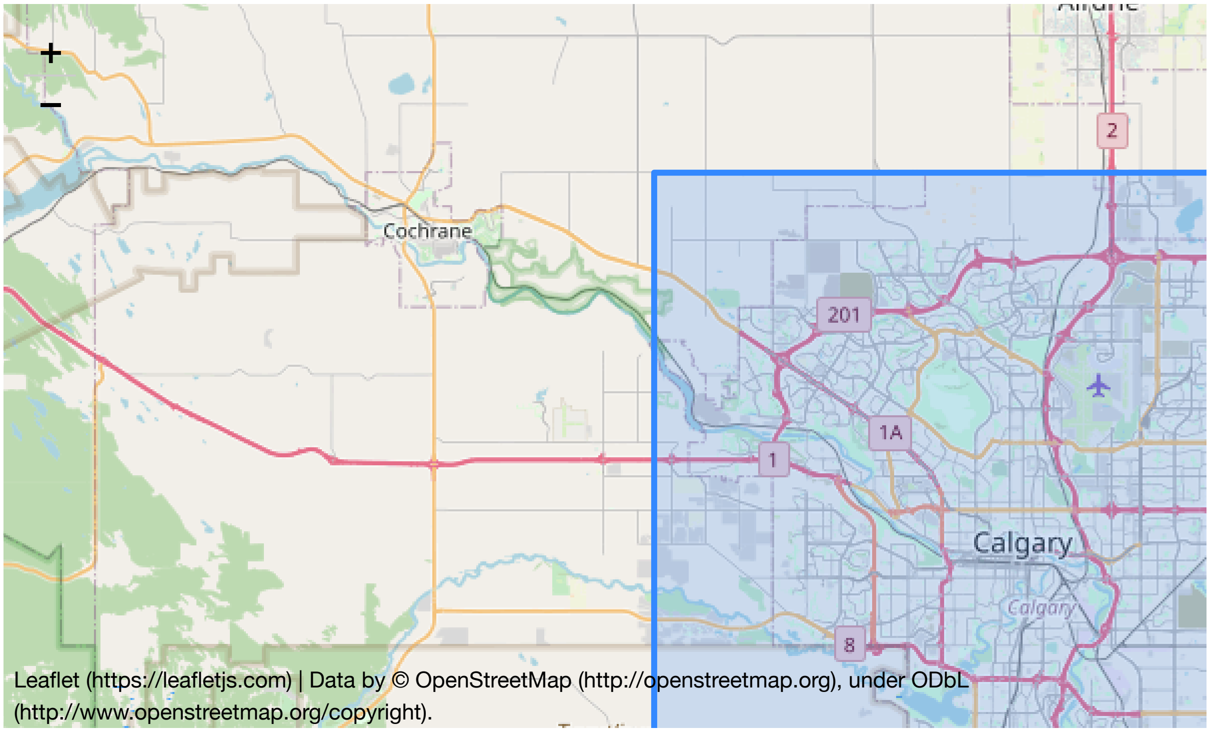
1.1.1 Load City Boundary Geometry and Create Boundary Geometry Object

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

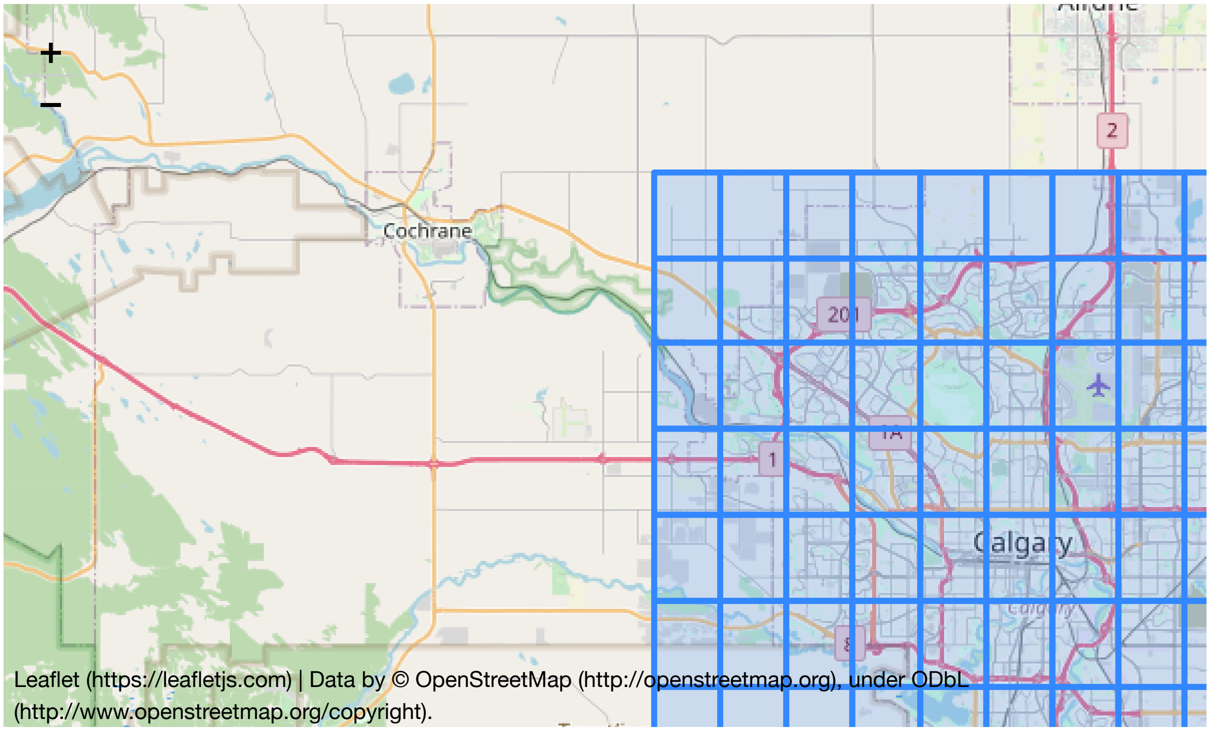
- Read the calgary boundary from City_Boundary_layer.csv.
- Draw a rectangle on Calgary map that shows the boundary of Calgary City.
- Divide calgary to a 10x10 matrix of areas.
  You need to investigate each area according to different features.

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 visual izing
data like histogram, scatter plot, line graph, heatmap to interpret your ans wer)

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	Accic
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)
```

In [1]:
```
import pandas as pd
import re
```

In [2]:
```
volume_df = pd.read_csv('volume_df.csv')
```

4.2 Show traffic heatmap of 2018. (5 Marks) ¶

In [3]:
```
# process multistring into array of coordinates
def get_coordinates(multistring):
    list_coordinate=re.sub('[^-0-9, .]','',multistring).split(",")
    coords=[]
    for i in list_coordinate:
        coord=list(float(j) for j in (i.strip().split(" ")[::-1]))
        coords.append(coord)
        exit()
    return coords
```

In [4]:
```
# process the data to create:
# lat lon volume dataframe for heatmap plotting
def generate_volume_points_df(volume_df):
    volume_points_df=pd.DataFrame(columns=['latitude','longitude','volum
e'])
    for i in range(volume_df.shape[0]):
        volume_coord=volume_df["multilinestring"].values[i]
        coor=get_coordinates(volume_coord)
        for j in range(len(coor)):
            coor[j].append(volume_df["VOLUME"].values[i])
            volume_points_df.loc[len(volume_points_df)]=coor[j]
    return  volume_points_df
```

In [5]:
```
# this line is causing kernel problem, maybe python cannot hold such hug
h list?
# volume_list=generate_traffic_volume_list(volume_df)
volume_points_df = generate_volume_points_df(volume_df)
volume_points_df.to_csv('volume_points_df.csv', index=False)
```

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.wkt import loads
from shapely.geometry import Point, Polygon
import math
```

Load back DataFrames from part 1

In [2]:
```
grid_df = pd.read_csv('grid_df.csv')
incident_df = pd.read_csv('incident_df.csv')
speed_df = pd.read_csv('speed_df.csv')
volume_df = pd.read_csv('volume_df.csv')
# cast geometry column back to GeoDataFrame
grid_df['geometry'] = grid_df['geometry'].apply(shapely.wkt.loads)
grid_gdf = gpd.GeoDataFrame(grid_df, geometry = 'geometry')

speed_df['multiline'] = speed_df['multiline'].apply(shapely.wkt.loads)
speed_gdf = gpd.GeoDataFrame(speed_df, geometry = 'multiline', crs='eps
g:4326')

# volume_df['multilinestring'] = volume_df['multilinestring'].apply(shap
ely.wkt.loads)
# volume_gdf = gpd.GeoDataFrame(volume_df, geometry = 'multilinestring',
crs='epsg:4326')
```

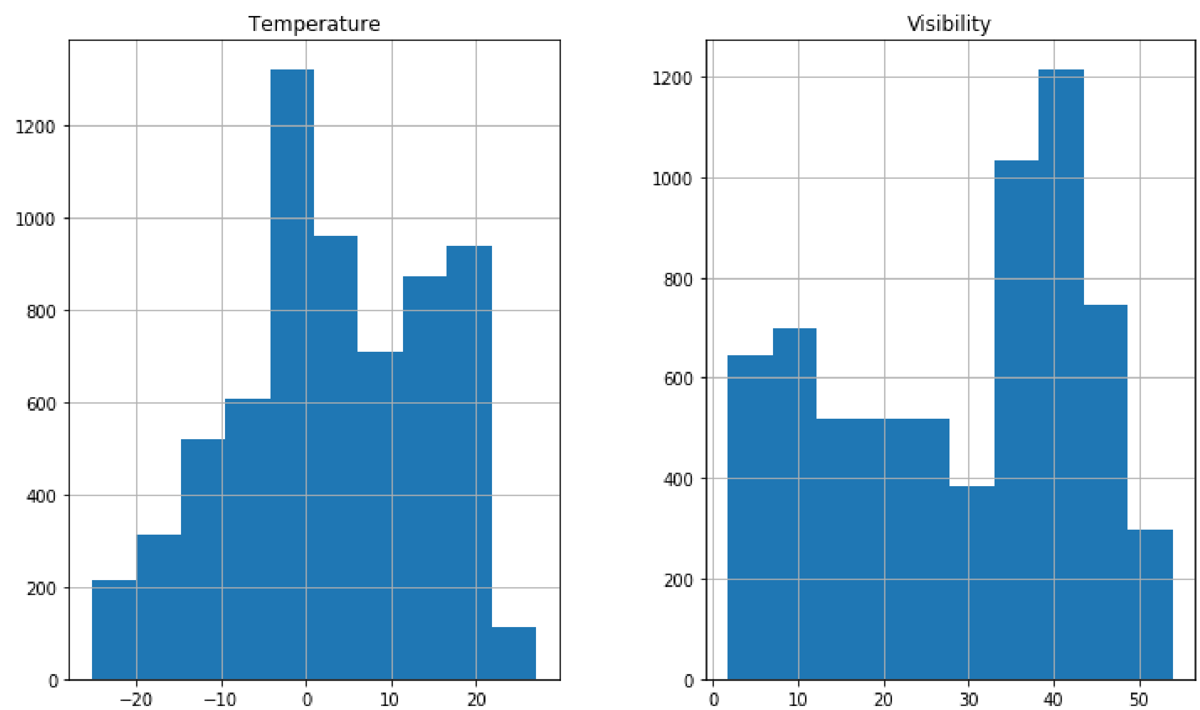
3. Correlation Analysis between features and Traffic Accidents

3.1 Daily Temperature and Visibility Influence on Traffic Accident

The histogram of Accident counts vs Temperature shows no relations between accident counts and temperature. Similarly, Accident counts vs Visibility scatter plot also indicates no relation between those two parameters.

However, by observing the distribution pattern, account count peaked at temperature range (-5, 0), which may indicating that when road just start to freeze, slippery road condition will create more tranffic accidents than any other temperature range.

In [3]:
```
fig, ax = plt.subplots(1, 2, figsize=(12,7))
# sns.scatterplot(x="Temperature", y="Accidents", ax=ax[0], data=inciden
t_df)
# sns.scatterplot(x="Visibility", y="Accidents", ax=ax[1], data=incident
_df)
incident_df.hist("Temperature", ax=ax[0])
incident_df.hist("Visibility", ax=ax[1])
plt.show()
```



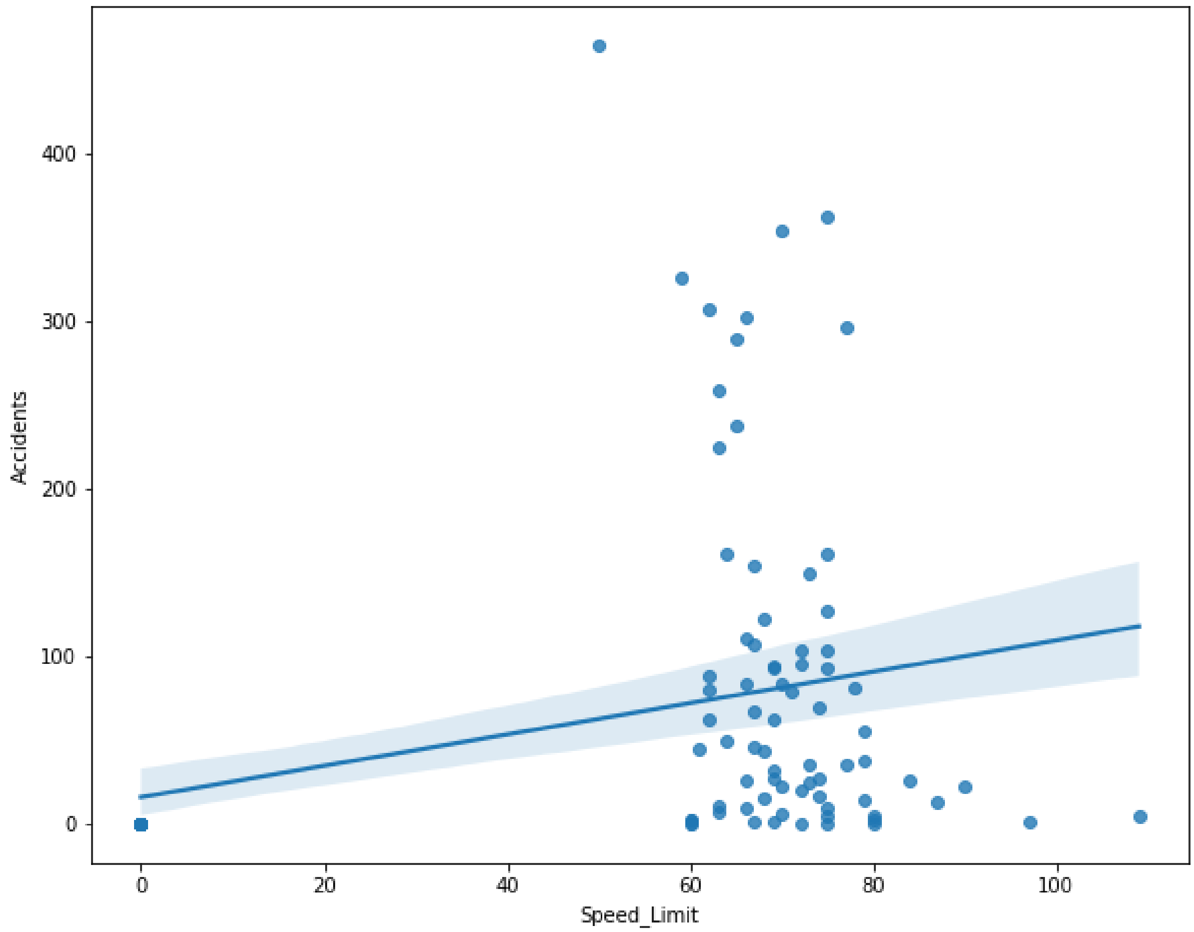
3.2 Other Features Influence on Traffic Accident

3.2.1 Correlation between Speed Limit and Traffic Accident

The linear regression plot of Accident counts vs Speed Limit shows a somewhat positive correlation between those two variables. However, most of the accidents can be seen within the speed limit range 60-80, which may indicates Expressways and Freeways within the city.

In [4]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x='Speed_Limit',y='Accidents', ax=ax, data=grid_df, order=1)
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1660cd50>



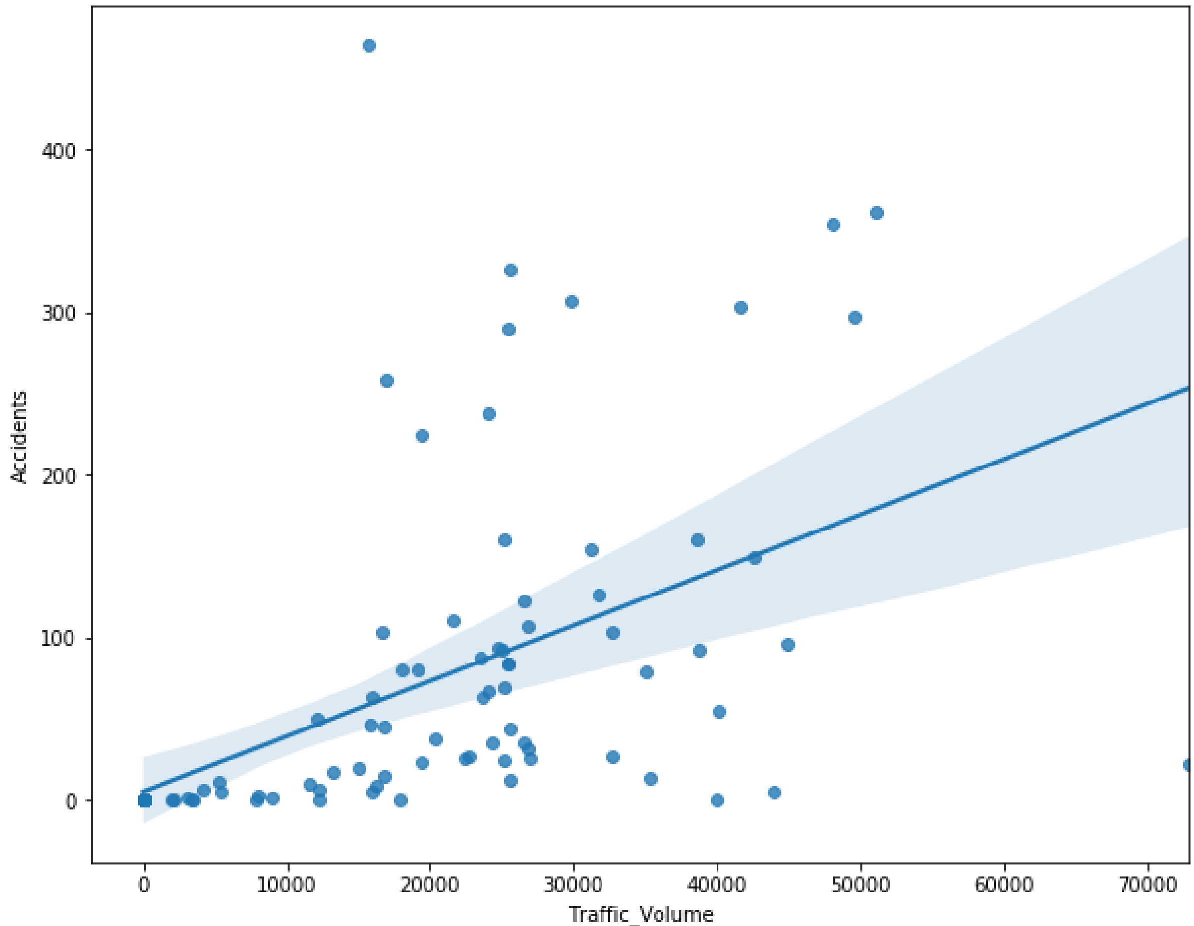
3.2.2 Correlation between Traffic Volume and Traffic Accident

The linear regression plot of Accidents Counts vs Traffic Volume indicates a positive correlation between them. This pattern is within anticipation statistically, as locations with higher traffic volume has the larger sample pool so are prone to have more traffic accidents.

The quantified correlation with be studied at the end of this chapter.

In [5]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x='Traffic_Volume',y='Accidents', ax=ax, data=grid_df, order
=1) # pointplot x vs y
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1a16bab650>

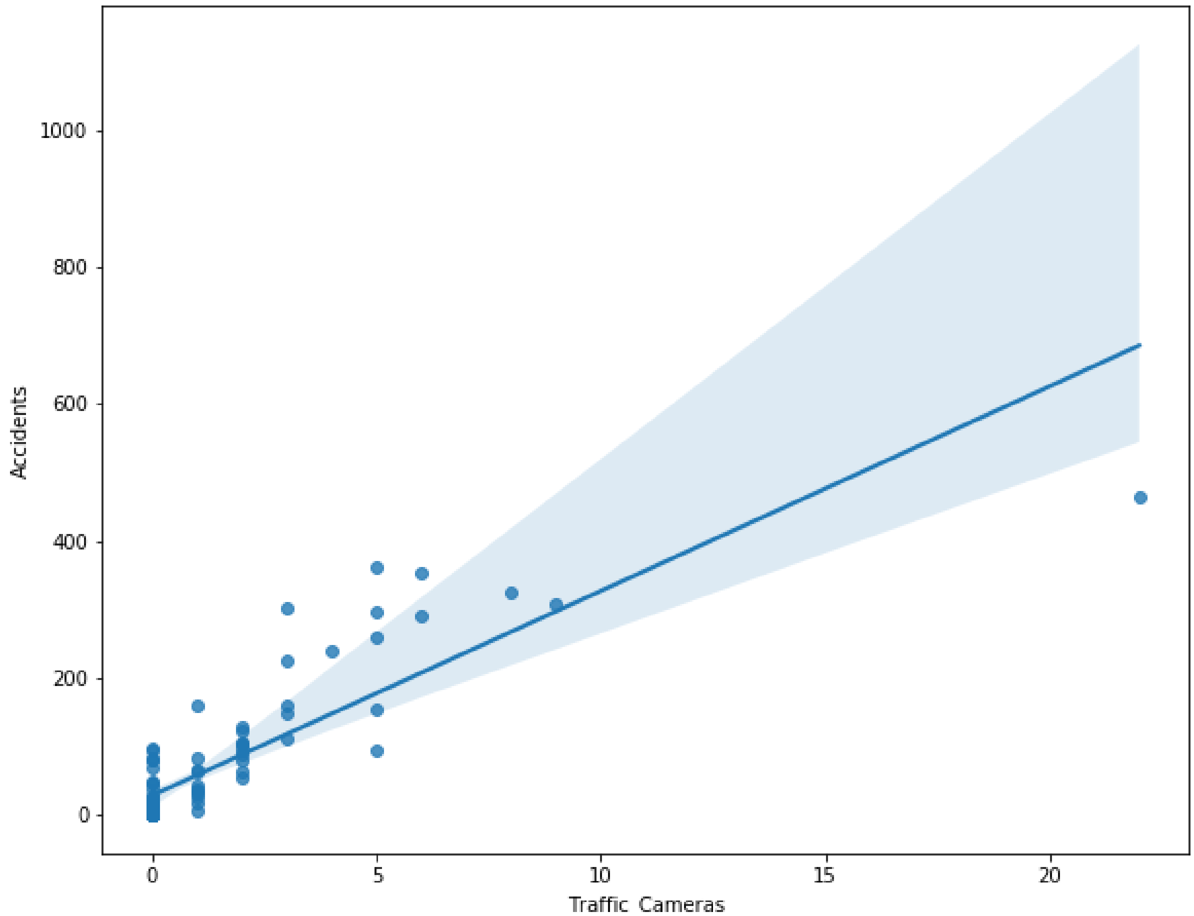


3.2.3 Correlation between Traffic Camera Counts and Traffic Accident

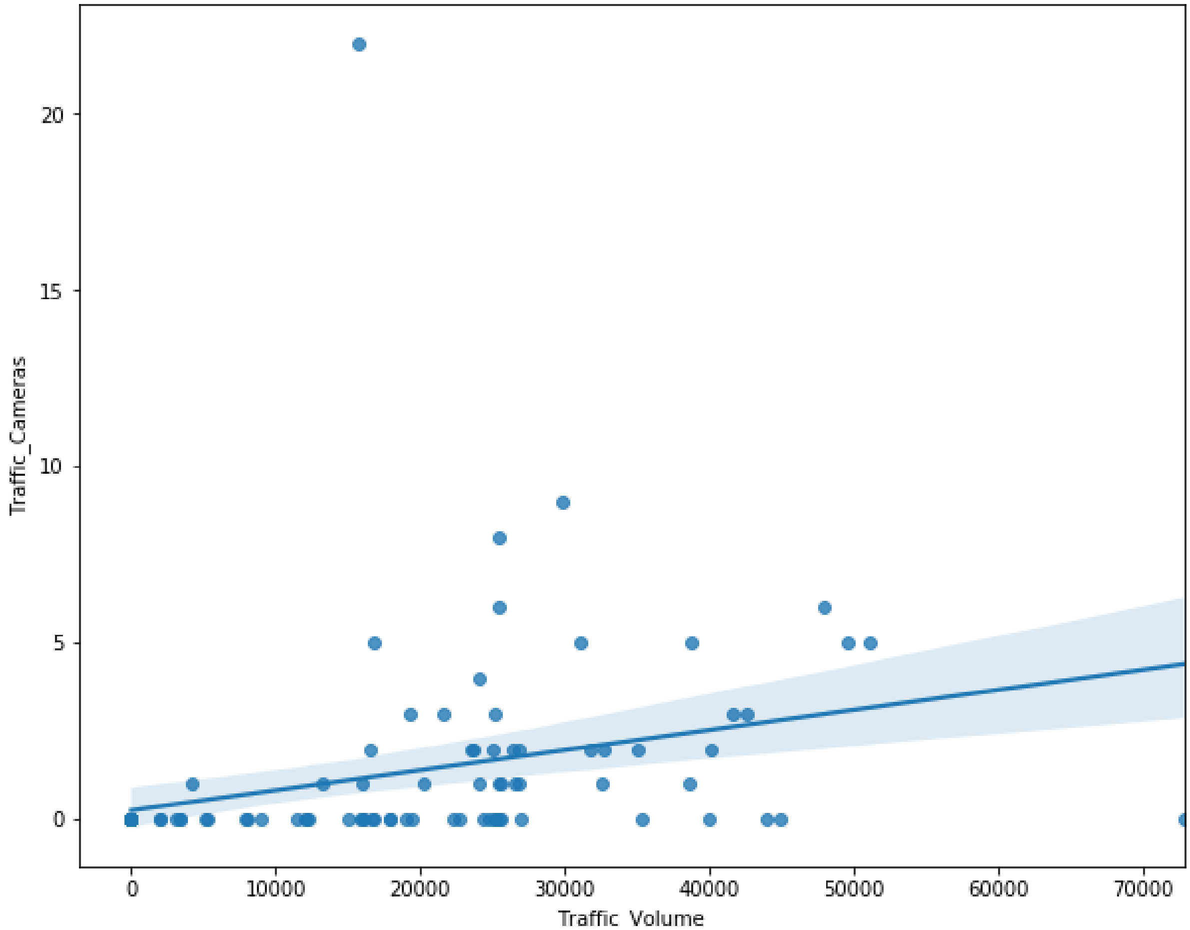
The regression plot of Traffic Accident counts vs Traffic Camera counts indicates a positive correlation between them. This effect may also because of the traffic volume, since the Traffic Cameras count vs Traffic Volume plot indicates a positive correlation between them.

The quantified correlation with be studied at the end of this chapter.

In [6]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x="Traffic_Cameras", y="Accidents", ax=ax, data=grid_df)
plt.show()
```



In [7]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x="Traffic_Volume", y="Traffic_Cameras", ax=ax, data=grid_df
)
plt.show()
```

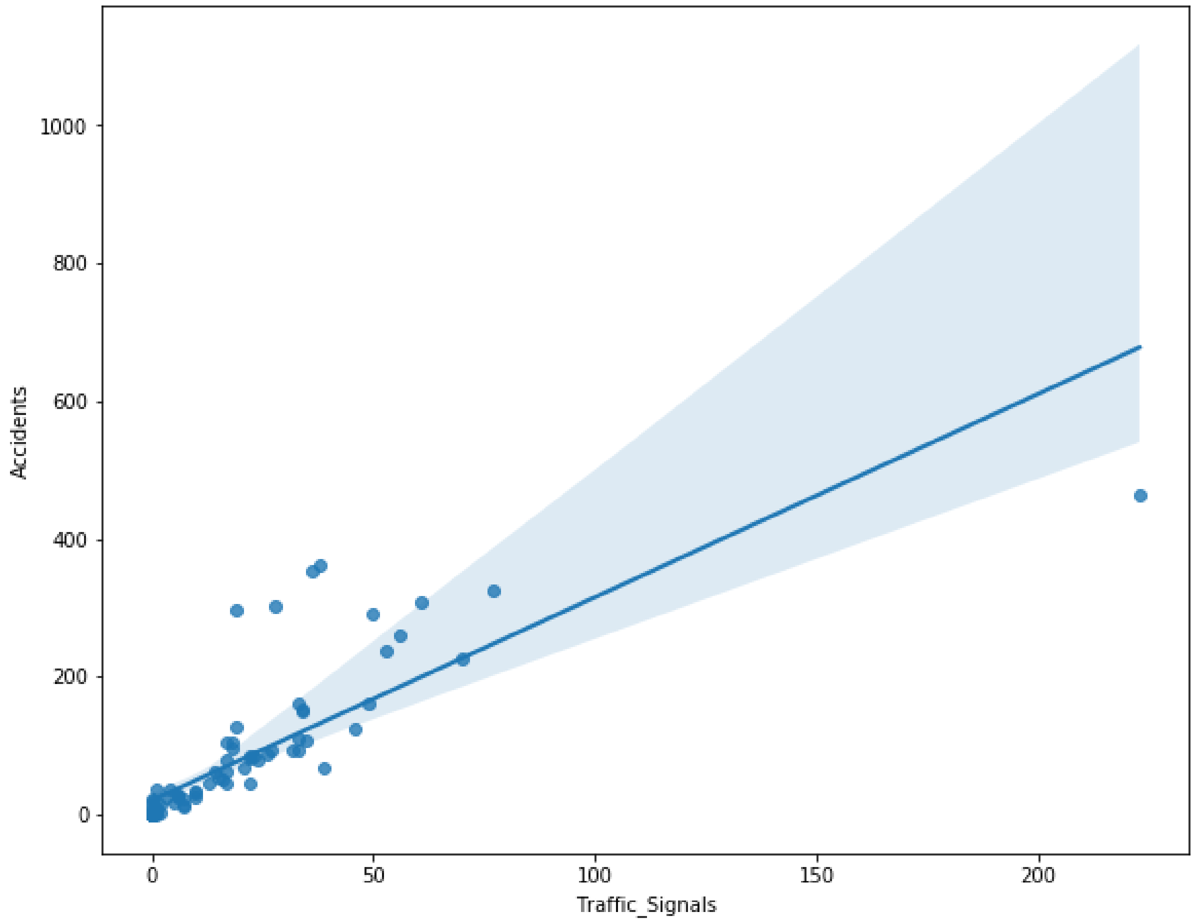


3.2.4 Correlation between Traffic Signal Counts and Traffic Accident

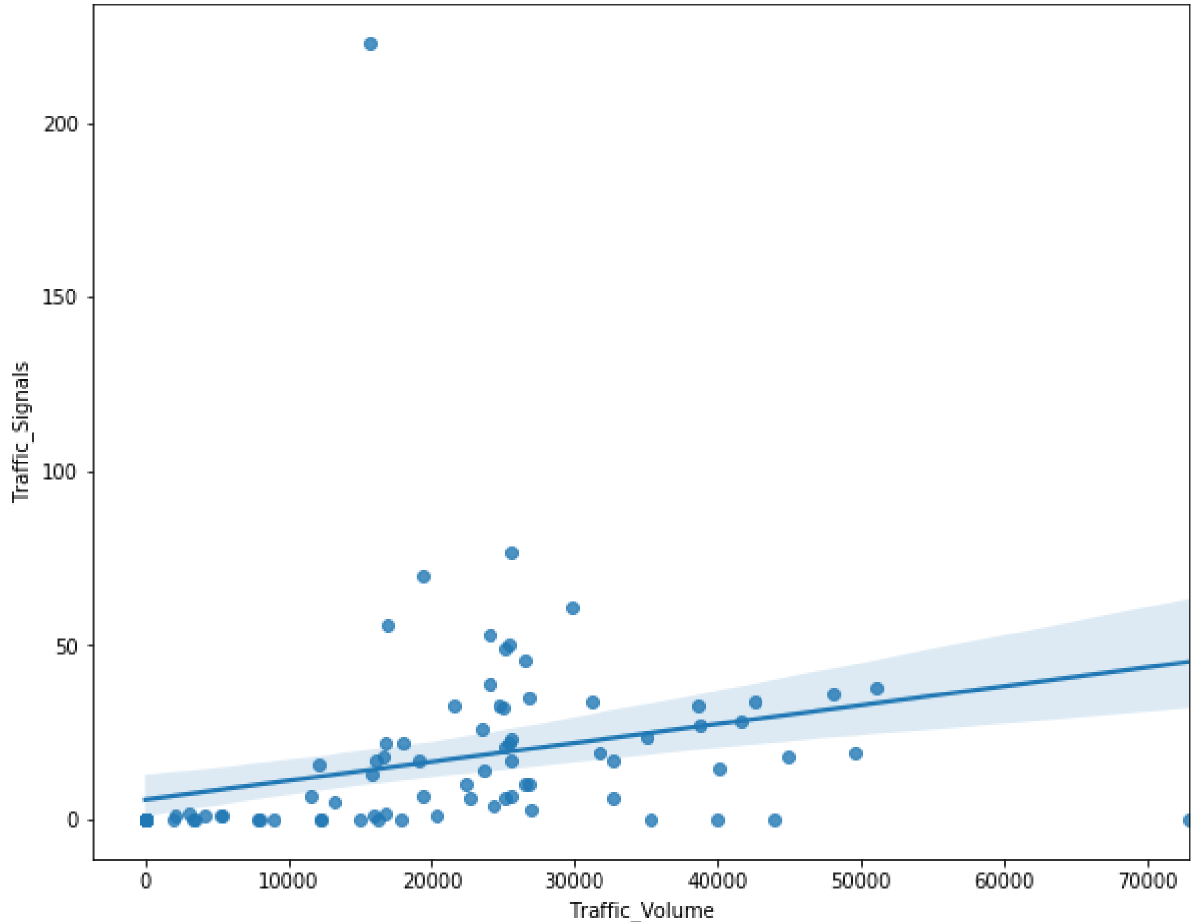
The regression plot of Traffic Accident counts vs Traffic Signal counts indicates a positive correlation between them. Like Traffic Cameras, this correlation may also because of the traffic volume, since the Traffic Signal count vs Traffic Volume plot indicates a positive correlation between them.

The quantified correlation with be studied at the end of this chapter.

In [8]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x="Traffic_Signals", y="Accidents", ax=ax, data=grid_df)
plt.show()
```

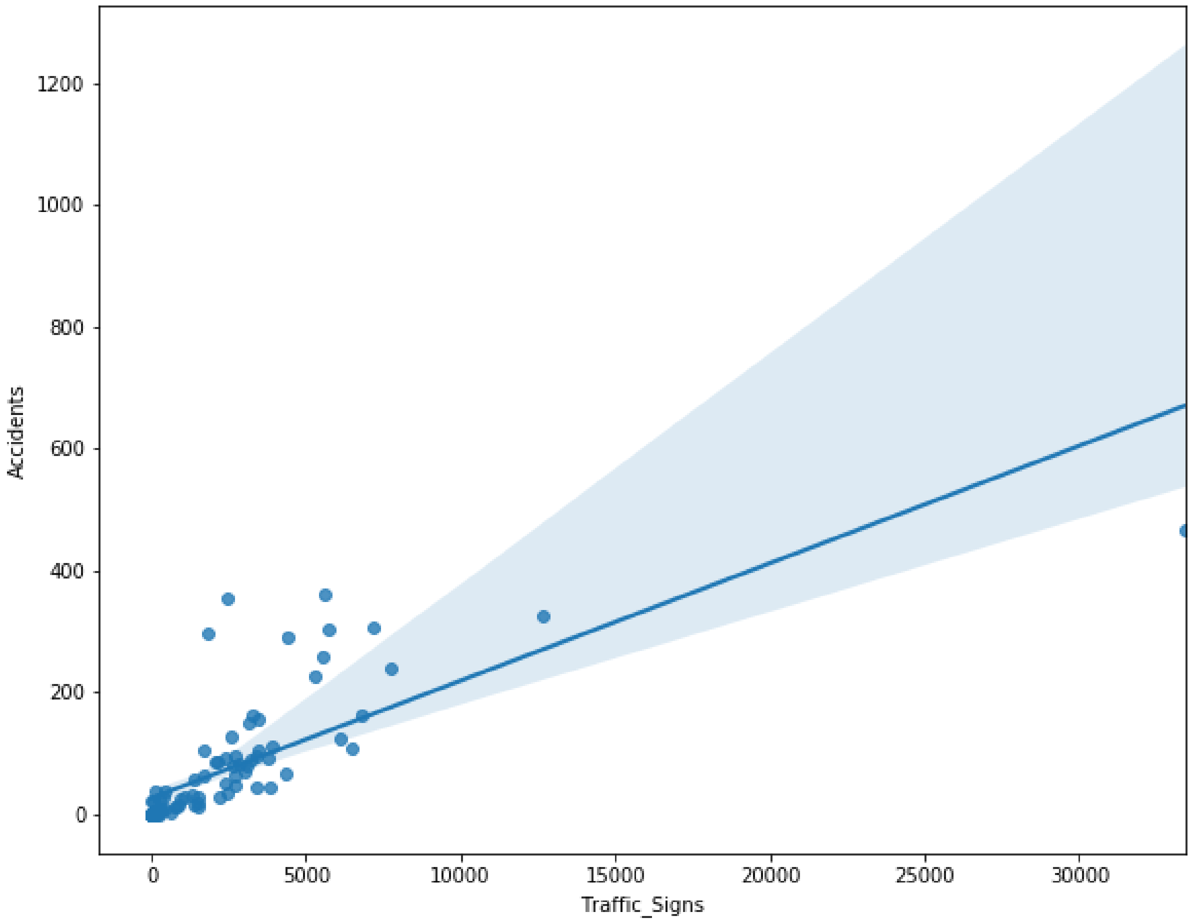


In [9]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x="Traffic_Volume", y="Traffic_Signals", ax=ax, data=grid_df
)
plt.show()
```

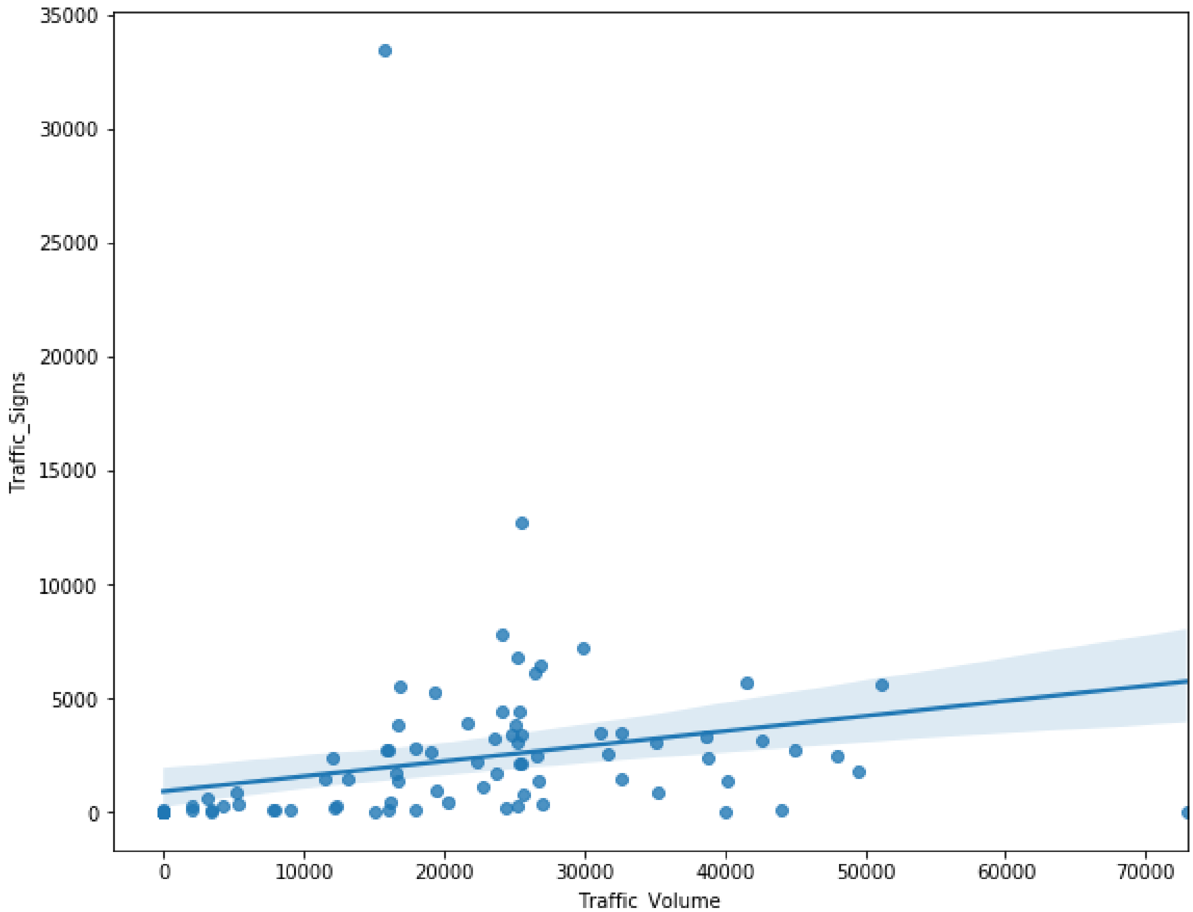


3.2.5 Correlation between Traffic Sign Counts and Traffic Accident

In [10]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x="Traffic_Signs", y="Accidents", ax=ax, data=grid_df)
plt.show()
```



In [11]:
```
fig,ax=plt.subplots(figsize=(10,8))
sns.regplot(x="Traffic_Volume", y="Traffic_Signs", ax=ax, data=grid_df)
plt.show()
```



3.3 Quantitative Study of Factors Contribute to Traffic Accidents

Three quantitative correlation study(Pearson, Kendall, Spearman) have been performed in this section.

In all three studies, we are observing positive correlations between any or the features and accident counts. Amoung those, Traffic Sign counts and Traffic Signal counts are showing the strongest correlation to accidents.

3.3.1 Pearson Correlation Study

In [12]:
```
parameters=['Accidents', 'Speed_Limit','Traffic_Volume', 'Traffic_Camera
s', 'Traffic_Signals', 'Traffic_Signs']
grid_df[parameters].corr(method='pearson')
```

Out[12]:

	Accidents	Speed_Limit	Traffic_Volume	Traffic_Cameras	Traffic_Signals	Traffic_S
Accidents	1.000000	0.299831	0.528462	0.846082	0.818475	0.75
Speed_Limit	0.299831	1.000000	0.707985	0.168168	0.213830	0.19
Traffic_Volume	0.528462	0.707985	1.000000	0.311684	0.302962	0.26
Traffic_Cameras	0.846082	0.168168	0.311684	1.000000	0.919970	0.90
Traffic_Signals	0.818475	0.213830	0.302962	0.919970	1.000000	0.97
Traffic_Signs	0.757298	0.196862	0.261465	0.900012	0.972463	1.00

3.3.2 Kendall Correlation Study

In [13]:
```
grid_df[parameters].corr(method='kendall')
```

Out[13]:

	Accidents	Speed_Limit	Traffic_Volume	Traffic_Cameras	Traffic_Signals	Traffic_S
Accidents	1.000000	0.282082	0.633239	0.701243	0.835438	0.78
Speed_Limit	0.282082	1.000000	0.555923	0.138387	0.169990	0.20
Traffic_Volume	0.633239	0.555923	1.000000	0.473617	0.528490	0.53
Traffic_Cameras	0.701243	0.138387	0.473617	1.000000	0.695502	0.61
Traffic_Signals	0.835438	0.169990	0.528490	0.695502	1.000000	0.82
Traffic_Signs	0.788238	0.201583	0.537230	0.611603	0.825797	1.00

3.3.3 Spearman Correlation Study

In [14]:
```
grid_df[parameters].corr(method='spearman')
```

Out[14]:

	Accidents	Speed_Limit	Traffic_Volume	Traffic_Cameras	Traffic_Signals	Traffic_S
Accidents	1.000000	0.439067	0.803756	0.800618	0.940578	0.92
Speed_Limit	0.439067	1.000000	0.712159	0.193739	0.277248	0.36
Traffic_Volume	0.803756	0.712159	1.000000	0.590118	0.679468	0.71
Traffic_Cameras	0.800618	0.193739	0.590118	1.000000	0.792520	0.74
Traffic_Signals	0.940578	0.277248	0.679468	0.792520	1.000000	0.94
Traffic_Signs	0.929010	0.360214	0.714335	0.743034	0.943018	1.00

3.3.4 Weather conditions

As we are not seeing any correlations between weather conditions and accident counts, it may be helpful to confirm this conclusion by using quantitative tools as well.

As expected, all results from three methods are showing very weak correlations between weather conditions and traffic accident counts. However, on a sidenote, we can observe a somewhat positive correlation between temperature and visibility, which is also true by common sense.

In [15]:
```
parameters=['Accidents', 'Temperature', 'Visibility']
incident_df[parameters].corr(method='pearson')
```

Out[15]:

	Accidents	Temperature	Visibility
Accidents	1.000000	0.044856	0.037522
Temperature	0.044856	1.000000	0.229565
Visibility	0.037522	0.229565	1.000000

In [16]:
```
incident_df[parameters].corr(method='kendall')
```

Out[16]:

	Accidents	Temperature	Visibility
Accidents	1.000000	0.028576	0.024483
Temperature	0.028576	1.000000	0.157612
Visibility	0.024483	0.157612	1.000000

In [17]:
```
incident_df[parameters].corr(method='spearman')
```

Out[17]:

	Accidents	Temperature	Visibility
Accidents	1.000000	0.042133	0.036063
Temperature	0.042133	1.000000	0.230376
Visibility	0.036063	0.230376	1.000000

4. Visualization

4.1 Visualize the speed limit according to the roads. (5 Marks)

In [18]:
```
smap = folium.Map(location=[51.03011, -114.08529], zoom_start = 10)

style20 = {'fillColor': '#FAF9DF', 'color': '#FAF9DF'}
speed20_df = speed_gdf[(speed_gdf['SPEED']>=20) & (speed_gdf['SPEED']<35
)]
folium.GeoJson(speed20_df['multiline'], style_function=lambda x:style20)
.add_to(smap)

style30 = {'fillColor': '#FAF8B9', 'color': '#FAF8B9'}
speed30_df = speed_gdf[(speed_gdf['SPEED']>=30) & (speed_gdf['SPEED']<35
)]
folium.GeoJson(speed30_df['multiline'], style_function=lambda x:style30)
.add_to(smap)

style35 = {'fillColor': '#F7F265', 'color': '#F7F265'}
speed35_df = speed_gdf[(speed_gdf['SPEED']>=35) & (speed_gdf['SPEED']<40
)]
folium.GeoJson(speed35_df['multiline'], style_function=lambda x:style35)
.add_to(smap)

style40 = {'fillColor': '#E7E032', 'color': '#E7E032'}
speed40_df = speed_gdf[(speed_gdf['SPEED']>=40) & (speed_gdf['SPEED']<45
)]
folium.GeoJson(speed40_df['multiline'], style_function=lambda x:style40)
.add_to(smap)

style45 = {'fillColor': '#E7CF3A', 'color': '#E7CF3A'}
speed45_df = speed_gdf[(speed_gdf['SPEED']>=45) & (speed_gdf['SPEED']<60
)]
folium.GeoJson(speed45_df['multiline'], style_function=lambda x:style45)
.add_to(smap)

style60 = {'fillColor': '#D6A40A', 'color': '#D6A40A'}
speed60_df = speed_gdf[(speed_gdf['SPEED']>=60) & (speed_gdf['SPEED']<70
)]
folium.GeoJson(speed60_df['multiline'], style_function=lambda x:style60)
.add_to(smap)

style70 = {'fillColor': '#E0A536', 'color': '#E0A536'}
speed70_df = speed_gdf[(speed_gdf['SPEED']>=70) & (speed_gdf['SPEED']<80
)]
folium.GeoJson(speed70_df['multiline'], style_function=lambda x:style70)
.add_to(smap)

style80 = {'fillColor': '#E17515', 'color': '#E17515'}
speed80_df = speed_gdf[(speed_gdf['SPEED']>=80) & (speed_gdf['SPEED']<90
)]
folium.GeoJson(speed80_df['multiline'], style_function=lambda x:style80)
.add_to(smap)

style90 = {'fillColor': '#E14D15', 'color': '#E14D15'}
speed90_df = speed_gdf[(speed_gdf['SPEED']>=90) & (speed_gdf['SPEED']<10
0)]
folium.GeoJson(speed90_df['multiline'], style_function=lambda x:style90)
.add_to(smap)
```

```
style100 = {'fillColor': '#AA370C', 'color': '#AA370C'}
speed100_df = speed_gdf[(speed_gdf['SPEED']>=100) & (speed_gdf['SPEED']<
110)]
folium.GeoJson(speed100_df['multiline'], style_function=lambda x:style10
0).add_to(smap)

style110 = {'fillColor': '#7B1C0B', 'color': '#7B1C0B'}
speed110_df = speed_gdf[(speed_gdf['SPEED']>=110)]
folium.GeoJson(speed110_df['multiline'], style_function=lambda x:style11
0).add_to(smap)
```

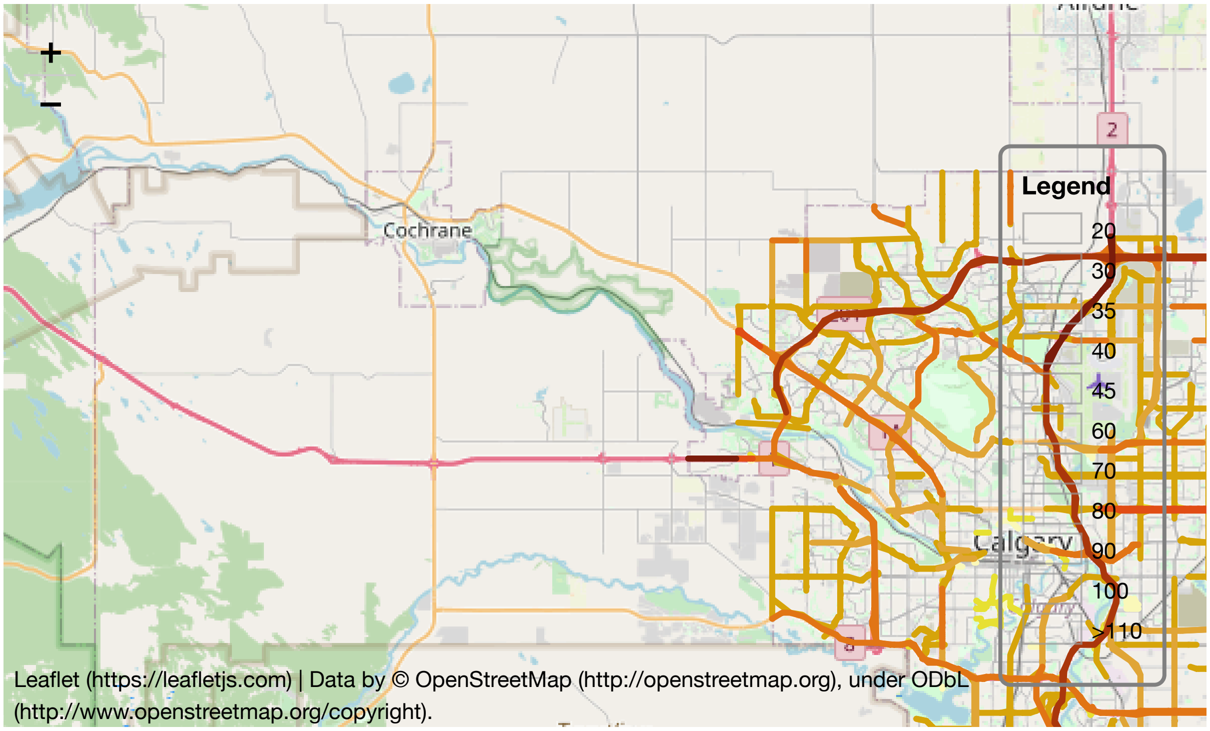
Out[18]: <folium.features.GeoJson at 0x1a179ac8d0>

In [19]:

```
# add legend info on to map and display
from branca.element import Template, MacroElement

template = """
{% macro html(this, kwargs) %}

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>jQuery UI Draggable - Default functionality</title>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/j
query-ui.css">

  <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

  <script>
  $( function() {
    $( "#maplegend" ).draggable({
                    start: function (event, ui) {
                        $(this).css({
                            right: "auto",
                            top: "auto",
                            bottom: "auto"
                        });
                    }
                });
});

  </script>
</head>
<body>


<div id='maplegend' class='maplegend'
    style='position: absolute; z-index:9999; border:2px solid grey; back
ground-color:rgba(255, 255, 255, 0.8);
     border-radius:6px; padding: 10px; font-size:14px; right: 20px; bott
om: 20px;'>

<div class='legend-title'>Legend</div>
<div class='legend-scale'>
  <ul class='legend-labels'>
    <li><span style='background:#FAF9DF;opacity:0.7;'></span>20</li>
    <li><span style='background:#FAF8B9;opacity:0.7;'></span>30</li>
    <li><span style='background:#F7F265;opacity:0.7;'></span>35</li>
    <li><span style='background:#E7E032;opacity:0.7;'></span>40</li>
    <li><span style='background:#E7CF3A;opacity:0.7;'></span>45</li>
    <li><span style='background:#D6A40A;opacity:0.7;'></span>60</li>
    <li><span style='background:#E0A536;opacity:0.7;'></span>70</li>
    <li><span style='background:#E17515;opacity:0.7;'></span>80</li>
    <li><span style='background:#E14D15;opacity:0.7;'></span>90</li>
    <li><span style='background:#AA370C;opacity:0.7;'></span>100</li>
    <li><span style='background:#7B1C0B;opacity:0.7;'></span>>110</li>
```

```
  </ul>
</div>
</div>

</body>
</html>

<style type='text/css'>
  .maplegend .legend-title {
    text-align: left;
    margin-bottom: 5px;
    font-weight: bold;
    font-size: 90%;
    }
  .maplegend .legend-scale ul {
    margin: 0;
    margin-bottom: 5px;
    padding: 0;
    float: left;
    list-style: none;
    }
  .maplegend .legend-scale ul li {
    font-size: 80%;
    list-style: none;
    margin-left: 0;
    line-height: 18px;
    margin-bottom: 2px;
    }
  .maplegend ul.legend-labels li span {
    display: block;
    float: left;
    height: 16px;
    width: 30px;
    margin-right: 5px;
    margin-left: 0;
    border: 1px solid #999;
    }
  .maplegend .legend-source {
    font-size: 80%;
    color: #777;
    clear: both;
    }
  .maplegend a {
    color: #777;
    }
</style>
{% endmacro %}"""

macro = MacroElement()
macro._template = Template(template)

smap.get_root().add_child(macro)

smap
```

Out[19]:



4.2 Show traffic heatmap of 2018. (5 Marks)

In [20]:
```
volume_points_df = pd.read_csv('volume_points_df.csv')
```

In [21]:
```
# create heat map  --hmap
```

In [22]:
```
from folium import FeatureGroup, LayerControl, Map, Marker
from folium.plugins import HeatMap
hmap = folium.Map(location=[51.04011, -114.03529], zoom_start = 13,tiles
='cartodbpositron')
gradient = {.5: 'lightblue', .7: 'orange', .9:'red'}
hm_wide = HeatMap(list(zip(volume_points_df.latitude.values, volume_poin
ts_df.longitude.values, volume_points_df.volume)),
                     min_opacity=0.3,
                     gradient=gradient,
                     radius=8,
                     blur=2,
                     max_zoom=10
                 )
hmap.add_child(hm_wide)
```

Out[22]: