# Library imports

In [1]:

```
import pandas as pd
import numpy as np
import geopandas as gpd
from shapely.geometry import MultiPoint, Point
import random
import zipfile
from geomanip.poly_contains import combine_geom_dfs
from geomanip.haversine import haversine_2_way
from geomanip.endpoint_algorithm import find_dist_ratio
from geomanip.inher_super import inherently_superior
import requests
import folium
import os
```

First, we read the routed generated from the 100 Zip Codes with the most EV registrations. This data was generated in the prior workbook.

In [2]:

```
top_routes = gpd.read_file('assets/top_routes.geojson')
```

In [3]:

```
top_routes.head()
```

Out[3]:

| | id | direction | part | route | route_num | seed_direction | seed_station | seed_traffic | station | traffic | traffic_adj | traffic_perc | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 5.0 | 1 | 1 | 140 | 5.0 | 870855 | 603 | 870855 | 603.000000 | 4.307143 | 1.000000 | MUL |
| **1** | 1 | 5.0 | 2 | 1 | 140 | 5.0 | 870855 | 603 | 870139 | 244.761282 | 1.748295 | 0.405906 | MUL |
| **2** | 2 | 5.0 | 3 | 1 | 140 | 5.0 | 870855 | 603 | 876138 | 145.569626 | 1.039783 | 0.241409 | MUL |

| | id | direction | part | route | route_num | seed_direction | seed_station | seed_traffic | station | traffic | traffic_adj | traffic_perc | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **3** | 3 | 5.0 | 4 | 1 | 140 | 5.0 | 870855 | 603 | 871135 | 88.974253 | 0.635530 | 0.147553 | MUL |
| **4** | 4 | 7.0 | 5 | 1 | 140 | 5.0 | 870855 | 603 | 878017 | 2.148334 | 0.015345 | 0.003563 | MUL |

First, we grouped the dataframe by station and direction and computed the total traffic by section.

In [4]:
```
route_traffic = (top_routes
                  .groupby(['station', 'direction'])
                  .agg({'traffic_adj': 'sum', 'geometry': 'first'})
                  .reset_index()
                 )
```

Next, we sorted by estimated traffic and chose only the 1,000 most trafficked routes.

In [5]:
```
route_traffic = route_traffic.sort_values('traffic_adj', ascending=False).iloc[:1000]
```

We then converted the grouped dataframe into a GeoDataFrame and set the crs.

In [6]:
```
route_traffic = gpd.GeoDataFrame(route_traffic, geometry='geometry')
route_traffic.crs = 'EPSG:4326'
```

We created a function that generates a given number of random points inside a polygon. This was created to generate random points adjacent to each segment.

In [7]:
```
def generate_random(polygon, number, seed=42):
    """
    Generates number of random points within a Shapely Polygon

    Parameters
    ----------
    polygon - Shapely Polygon
        polygon from which random points are drawn
    number - int
```

```
        number of random points to draw
    seed - int
        random seed set for reproducibility

    Returns
    -------
    Shapely MultiPoint shape with all generated points
    """
    points = []
    minx, miny, maxx, maxy = polygon.bounds
    random.seed(seed)
    while len(points) < number:
        pnt = Point(random.uniform(minx, maxx), random.uniform(miny, maxy))
        if polygon.contains(pnt):
            points.append(pnt)
    points = MultiPoint(points)
    return points
```

We created a buffer around each segment to create polygons for generating points.

In [8]:

```
route_traffic.geometry = route_traffic.buffer(0.001)
```

```
<ipython-input-8-cd13f26ec9f0>:1: UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likel
y incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.

  route_traffic.geometry = route_traffic.buffer(0.001)
```

Next, we created a column with MulitPoint objects of 10 randomly selected points within each buffer zone using the previously created function.

In [9]:

```
route_traffic['geom_point'] = route_traffic.geometry.map(lambda x: generate_random(x, 10))
```

We then renamed the geomtry columns and set the point geometries as the main geometry in our GeoDataFrame. This enables the explosion of the MultiPoint geometry into Point geometries.

In [10]:

```
route_traffic.rename({'geometry': 'geom_poly', 'geom_point': 'geometry'}, axis=1, inplace=True)
route_traffic = gpd.GeoDataFrame(route_traffic, geometry='geometry')
route_traffic.crs = 'EPSG:4326'
```

We reset the index and exploded the MultiPoint geometries.

In [11]:
```
route_traffic = route_traffic.reset_index().drop('index', axis=1)
route_traffic = route_traffic.explode().reset_index()
```

We then dropped the unnecessary grouping columns created by the `explode` method.

In [12]:
```
route_traffic.drop(['level_0', 'level_1'], axis=1, inplace=True)
```

Next we extracted files from the NYC zoning geographic database. This contains geomtries of differently zoned areas in NYC.

In [13]:
```
if not os.path.exists('assets/nyc_zoning'):
    with zipfile.ZipFile('assets/nyc_zoning.zip', 'r') as nyc_zoning:
        nyc_zoning.extractall('assets/nyc_zoning')
```

We read the file for commercially zoned areas and set the crs accordingly. We assumed that charging stations would only be placed in areas zoned for commercial development.

In [14]:
```
nyc_zoning = gpd.read_file('assets/nyc_zoning/nyc_zoning/nyco.shp')
nyc_zoning.crs = 'EPSG:2263'
nyc_zoning = nyc_zoning.to_crs('EPSG:4326')
```

We next combined the zoning and traffic dataframes using a vectorized function from our created `geomanip` package. This method finds all generated points that lie withing commercially zoned areas

In [15]:
```
top_results_full = combine_geom_dfs(nyc_zoning, route_traffic, geo_name1 = 'geom_zoning', geo_name2='geom_route
```

In [16]:
```
top_results_full.columns
```

Out[16]:
```
Index(['OVERLAY', 'Shape_Leng', 'Shape_Area', 'geom_zoning', 'station',
       'direction', 'traffic_adj', 'geom_poly', 'geom_route'],
      dtype='object')
```

We read the dataframe with all alternative fuel stations. This was used to determine the distance of commercially zoned points to the nearest existing charging station.

In [17]:
```
stations = pd.read_csv('assets/alt_fuel_stations.csv', low_memory=False)
```

We selected only the stations with are in NY state and are below 41.5$^{\circ}$N.

In [18]:
```
stations_nyc = stations.loc[(stations['State'] == 'NY') & (stations['Latitude'] < 41.5)]
```

We converted the latitude and longitude of the charging stations to Shapely Point objects, then converted the dataframe to a GeoDataFrame and set the crs.

In [19]:
```
stations_nyc['geometry'] = stations_nyc.apply(lambda x: Point(x['Longitude'], x['Latitude']), axis=1)
stations_nyc = gpd.GeoDataFrame(stations_nyc, geometry='geometry')
stations_nyc.crs = 'EPSG:4326'
```

```
<ipython-input-19-10c82622e6f8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  stations_nyc['geometry'] = stations_nyc.apply(lambda x: Point(x['Longitude'], x['Latitude']), axis=1)
```

We recreated the dataframe with commercially zoned points into a GeoDataFrame and set the crs.

In [20]:
```
top_results_full = gpd.GeoDataFrame(top_results_full, geometry='geom_route')
top_results_full.crs = 'EPSG:4326'
```

Using the `haversine_2_way` vectorized function from the `geomanip` package that we created, we found the distance of each generated commercially zoned point to every EV charging station.

In [21]:
```
stations_route = haversine_2_way(top_results_full.reset_index(), stations_nyc, units='imperial',
                                 geo_col1='geom_route', geo_name1='geom_route', geo_name2='geom_station')
```

Next we grouped the resulting dataframe by the index and found the closest charging station to each point.

In [22]:
```
g = stations_route.groupby('index')
station_closest = g.apply(lambda x: x.sort_values('distance').iloc[0])
```

Next we adjusted the indices as they were no longer needed to identify points.

In [23]:
```
station_closest = (station_closest
                   .rename({'index': 'index0'}, axis=1)
                   .reset_index()
                   .drop(['index', 'index0'], axis=1))
```

We used the `inherently superior` vectorized function from our created `geomanip` package. This function takes the dataset, with distance to a charging station and traffic and finds how many other possible locations each possible location is superior to. Points are characterized as superior to other points if both metrics are as high or higher than for another point. The `perc_sup` column is the percent of other points to which each point is superiror.

In [24]:
```
station_closest['perc_sup'] = inherently_superior(station_closest[['distance', 'traffic_adj']])['perc_sup']
```

We then chose the top 10 locations based on this evaluation of superiority.

In [25]:
```
top_loc = station_closest.sort_values('perc_sup', ascending=False).iloc[:10]
```

In [26]:
```
top_loc
```

Out[26]:

| | OVERLAY | Shape_Leng | Shape_Area | geom_zoning | station | direction | traffic_adj | geom_poly | ge |
|---|---|---|---|---|---|---|---|---|---|
| **135** | C1-3 | 1597.641551 | 70321.207270 | POLYGON ((-73.96421219075391 40.62253839844913... | 021189 | 1.0 | 100.195999 | POLYGON ((-73.96599815922355 40.62337155093744... | (-73.964312 40.622657 |
| **123** | C2-4 | 1833.883122 | 85160.054471 | POLYGON ((-73.96556566650732 40.62970644177518... | 021189 | 1.0 | 100.195999 | POLYGON ((-73.96599815922355 40.62337155093744... | (-73.965964 40.6315193 |
| **141** | C2-2 | 1318.762024 | 81526.600657 | POLYGON ((-73.92967827495094 40.58643644135677... | 020906 | 7.0 | 95.032183 | POLYGON ((-73.9093162587283 40.59431571606489,... | (-73.930404 40.5861081 |
| **50** | C2-2 | 706.363115 | 30191.361539 | POLYGON ((-73.75781782182298 40.74816078230577... | 050047 | 7.0 | 159.917042 | POLYGON ((-73.75611525833661 40.74838273387774... | (-73.758357 40.748308 |
| **146** | C1-3 | 1803.514432 | 79833.674600 | POLYGON ((-73.96518777502872 40.62243063917565... | 021138 | 7.0 | 13.456809 | POLYGON ((-73.95748263527133 40.62460333790642... | (-73.96533 40.6240226 |
| **62** | C1-2 | 1620.834235 | 129511.836437 | POLYGON ((-73.73152517420824 40.75949029666447... | 056018 | 1.0 | 14.840996 | POLYGON ((-73.72951177592692 40.75508101552814... | (-73.732102 40.75964 |

| | OVERLAY | Shape_Leng | Shape_Area | geom_zoning | station | direction | traffic_adj | geom_poly | ge |
|---|---|---|---|---|---|---|---|---|---|
| **145** | C1-4 | 453.542650 | 12669.438046 | POLYGON ((-73.96841189251157 40.60892078397171... | 021145 | 7.0 | 25.869827 | POLYGON ((-73.96326138004056 40.60887981352835... | (-73.968414 40.6089307 |
| **133** | C1-4 | 666.978052 | 23431.596882 | POLYGON ((-73.97248586611182 40.60899433508917... | 021144 | 7.0 | 24.440267 | POLYGON ((-73.97342798197243 40.60971399337825... | (-73.972412 40.609123 |
| **101** | C1-4 | 604.732720 | 21808.308130 | POLYGON ((-73.96851475796134 40.60945224235646... | 021144 | 7.0 | 24.440267 | POLYGON ((-73.97342798197243 40.60971399337825... | (-73.968563 40.609573 |
| **97** | C2-3 | 1186.803935 | 47889.466986 | POLYGON ((-73.96157397842951 40.60856720165741... | 021145 | 7.0 | 25.869827 | POLYGON ((-73.96326138004056 40.60887981352835... | (-73.9618 40.609280 |

10 rows × 77 columns

We reformed the dataframe into a GeoDataFrame and set the crs. We then rounded the distance and traffic down to the nearest integer to make tooltips in the Folium map have fewer significant digits. Finally we separated the top locations from the existing EV charging stations for plotting.

In [27]:

```
top_loc =  gpd.GeoDataFrame(top_loc, geometry='geom_route')
top_loc.crs = 'EPSG:4326'
top_loc['distance'] = top_loc['distance'].astype(int)
top_loc['traffic_adj'] = top_loc['traffic_adj'].astype(int)
top_loc_poly = gpd.GeoDataFrame(top_loc, geometry='geom_poly')[['traffic_adj', 'distance', 'geom_poly']]
top_loc_poly.crs = 'EPSG:4326'
ev_chargers = gpd.GeoDataFrame(top_loc, geometry='geom_station')[['traffic_adj', 'distance', 'geom_station']]
ev_chargers.crs = 'EPSG:4326'
```

The chart below shows the top 10 locations based on our analysis. While some of the locations are close together, this may be necessary as many businesses may not agree to installing chargers. The existing charging stations are shown with a half empty battery in red, and the chosen locations are in blue.

In [28]:

```
m = folium.Map(location=(40.7128, -74), zoom_start=10)
points = folium.features.GeoJson(top_loc[['distance', 'traffic_adj', 'geom_route']].to_json(),
                                  tooltip=folium.features.GeoJsonTooltip(fields=['distance', 'traffic_adj'],
```

```
                                                                                aliases=['Distance to Charger (feet)',
                                                                                         'Calculated EV Traffic']),
                                    marker=folium.Marker())
points.add_to(m)

points_ev = folium.features.GeoJson(ev_chargers.to_json(), marker=folium.Marker(icon=folium.Icon(icon='battery-
                                    )
points_ev.add_to(m)
m
```

Out[28]: Make this Notebook Trusted to load map: File -> Trust Notebook

Finally, we wanted to find the exact location of each point using the Mapbox reverse geocoding api. Below is a function used for this purpose.

In [29]:
```
from config import api_token

longitude = top_loc['geom_route'].iloc[0].x
latitude = top_loc['geom_route'].iloc[0].y

def find_address(point):
    """
    Uses Mapbox reverse geocoding API to find address of specific location

    Parameters
    ----------
    point - Shapely Point
        point containing latlong coords of address

    Returns
    -------
    Address of location

    """
    longitude = point.x
    latitude = point.y
    TOKEN = api_token
    URL = f'https://api.mapbox.com/geocoding/v5/mapbox.places/{longitude},{latitude}.json?access_token={TOKEN}'
    address = requests.get(URL).json()['features'][0]['place_name']
    return address
```

The addresses of the chosen locations are found using the above function.

In [30]:
```
top_loc['address'] = top_loc['geom_route'].map(find_address)
```

These are the final chosen locations with traffic, distance to a charging station, and address.

In [31]:
```
final_loc = top_loc[['traffic_adj', 'distance', 'address']]
final_loc.to_csv('assets/top_loc.csv')
final_loc
```

Out[31]:

| | traffic_adj | distance | address |
|---|---|---|---|
| **135** | 100 | 11269 | Mark Halberstam, 1435 Coney Island Ave, New Yo... |
| **123** | 100 | 9655 | 1071 Coney Island Avenue, Brooklyn, New York 1... |

| | traffic_adj | distance | address |
|---|---|---|---|
| **141** | 95 | 5774 | 3939 Shore Parkway, Brooklyn, New York 11235, ... |
| **50** | 159 | 5338 | BP, 21902 Horace Harding Expy, New York, New Y... |
| **146** | 13 | 11819 | 1372 Coney Island Avenue, Brooklyn, New York 1... |
| **62** | 14 | 10163 | 56-20 Marathon Parkway, Queens, New York 11362... |
| **145** | 25 | 7527 | 1622 Ocean Parkway, Brooklyn, New York 11223, ... |
| **133** | 24 | 7978 | Dunkin', 407 Avenue P, New York, New York 1123... |
| **101** | 24 | 7764 | 511 Avenue P, Brooklyn, New York 11230, United... |
| **97** | 25 | 7352 | 1963 Coney Island Avenue, Brooklyn, New York 1... |