IND708: How To Run Python Mapping Code

The process for running the mapping code is very simple. There are just a few things to do in the beginning to ensure the csv file is imported correctly then the rest of running the code should be simple.

#1)

```
import pandas as pd

df = pd.read_csv('vending_machine_sales_final 2.csv')
df
```

	Status	ID	Location	Machine	Product	Category	Transaction	TransDate	Type	RCoil	RPrice	RQty	MCoil	MPrice	MQty	LineTotal	TransTotal	Date
0	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14515778905	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
1	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516018629	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	5.0	1/1/2022
2	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516018629	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	5.0	1/1/2022
3	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516020373	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	1.5	1/1/2022
4	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516021756	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6440	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Lindens - Chocolate Chippers	Food	15603201222	Wednesday, August 31, 2022	Credit	122	2.0	1	122	2.0	1	2.0	6.0	8/31/2022
6441	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Wonderful Pistachios - Variety	Food	15603201222	Wednesday, August 31, 2022	Credit	131	2.0	1	131	2.0	1	2.0	6.0	8/31/2022
6442	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Hungry Buddha - Chocolate Chip	Food	15603201222	Wednesday, August 31, 2022	Credit	137	2.0	1	137	2.0	1	2.0	6.0	8/31/2022
6443	Processed	VJ300320609	GuttenPlans	GuttenPlans x1367	Snapple Tea - Lemon	Non Carbonated	15603853105	Wednesday, August 31, 2022	Credit	145	2.5	1	145	2.5	1	2.5	2.5	8/31/2022
6444	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Goldfish Baked - Cheddar	Food	15603921383	Wednesday, August 31, 2022	Cash	125	1.5	1	125	1.5	1	1.5	1.5	8/31/2022

6445 rows × 18 columns

Figure 1

- For this initial step make sure the file for the vending machine is called '`vending_machine_sales_final 2.csv`'

#2) Import the csv file to 'google colab' as such:

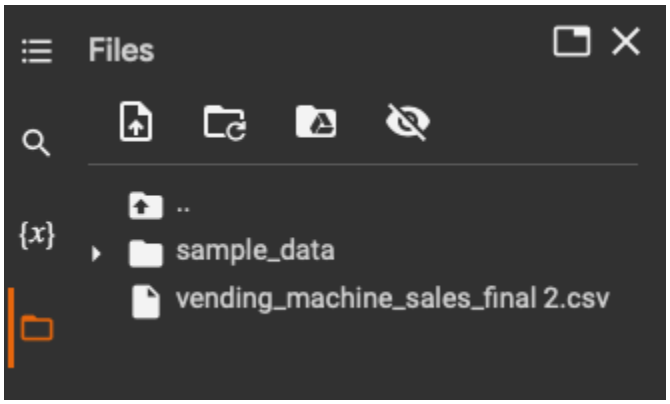


Figure 2

- Once the file is added you can run the initial code in figure 1 to display the data set

#3) From here you can run the code one by one in sequence. Make sure when you get to the mapping codes to run each cell in sequence to ensure the data allocates the right parameters for the latitude and longitude of each route.

#4) Below the sequence for each code inputs and outputs are displayed to show the imputed code as well as what should be seen after running the code.

4.1: Importing File

```
import pandas as pd

df = pd.read_csv('vending_machine_sales_final 2.csv')
df
```

	Status	ID	Location	Machine	Product	Category	Transaction	TransDate	Type	RCoil	RPrice	RQty	MCoil	MPrice	MQty	LineTotal	TransTotal	Date
0	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14515778905	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
1	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516018629	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	5.0	1/1/2022
2	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516018629	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	5.0	1/1/2022
3	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516020373	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	1.5	1/1/2022
4	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516021756	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6440	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Lindens - Chocolate Chippers	Food	15603201222	Wednesday, August 31, 2022	Credit	122	2.0	1	122	2.0	1	2.0	6.0	8/31/2022
6441	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Wonderful Pistachios - Variety	Food	15603201222	Wednesday, August 31, 2022	Credit	131	2.0	1	131	2.0	1	2.0	6.0	8/31/2022
6442	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Hungry Buddha - Chocolate Chip	Food	15603201222	Wednesday, August 31, 2022	Credit	137	2.0	1	137	2.0	1	2.0	6.0	8/31/2022
6443	Processed	VJ300320609	GuttenPlans	GuttenPlans x1367	Snapple Tea - Lemon	Non Carbonated	15603853105	Wednesday, August 31, 2022	Credit	145	2.5	1	145	2.5	1	2.5	2.5	8/31/2022
6444	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Goldfish Baked - Cheddar	Food	15603921383	Wednesday, August 31, 2022	Cash	125	1.5	1	125	1.5	1	1.5	1.5	8/31/2022

6445 rows × 18 columns

Figure 3

4.2: Cleaning NaN Values

```
[ ] #dropping all NaN values found in the vending machines data set
    df = df.dropna(axis=0)
    df
```

	Status	ID	Location	Machine	Product	Category	Transaction	TransDate	Type	RCoil	RPrice	RQty	MCoil	MPrice	MQty	LineTotal	TransTotal	Date
0	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14515778905	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
1	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516018629	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	5.0	1/1/2022
2	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516018629	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	5.0	1/1/2022
3	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516020373	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	1.5	1/1/2022
4	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516021756	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6440	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Lindens - Chocolate Chippers	Food	15603201222	Wednesday, August 31, 2022	Credit	122	2.0	1	122	2.0	1	2.0	6.0	8/31/2022
6441	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Wonderful Pistachios - Variety	Food	15603201222	Wednesday, August 31, 2022	Credit	131	2.0	1	131	2.0	1	2.0	6.0	8/31/2022
6442	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Hungry Buddha - Chocolate Chip	Food	15603201222	Wednesday, August 31, 2022	Credit	137	2.0	1	137	2.0	1	2.0	6.0	8/31/2022
6443	Processed	VJ300320609	GuttenPlans	GuttenPlans x1367	Snapple Tea - Lemon	Non Carbonated	15603853105	Wednesday, August 31, 2022	Credit	145	2.5	1	145	2.5	1	2.5	2.5	8/31/2022
6444	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Goldfish Baked - Cheddar	Food	15603921383	Wednesday, August 31, 2022	Cash	125	1.5	1	125	1.5	1	1.5	1.5	8/31/2022

6260 rows × 18 columns

Figure 4

4.3: Determining Unique Locations

```
[ ]  #determining the unique locations

     df = pd.read_csv('vending_machine_sales_final 2.csv')

     location = df['Location']
     unique_location = location.unique()

     unique_location
```

```
array(['Brunswick Sq Mall', 'Earle Asphalt', 'GuttenPlans',
       'EB Public Library'], dtype=object)
```

Figure 5

4.4: Determining which Location sold the most

```
[ ]  #what vending machine sold the most products

     df['Location'].value_counts(ascending=True)
```

```
Earle Asphalt          676
Brunswick Sq Mall     1279
EB Public Library     1922
GuttenPlans           2568
Name: Location, dtype: int64
```

Figure 6

4.5: Creating new dataset and gave latitude and longitude values for each location

```
[ ]  #creating a new dataset with locations and their corresponding volumes, locations, etc.
     #will find loaction for storage facility afterwards
     df1 = pd.DataFrame({'Location': ['Brunswick Sq Mall', 'Earle Asphalt', 'GuttenPlans', 'EB Public Library'],
                         'Latitude': [43.60022, 43.58916, 43.54102, 43.54998],'Longitude': [-79.71238, -79.64498, -79.60217, -79.68079],
                         'Volume': [1279, 676, 2568, 1922],
                         'Location Type':['Demand', 'Demand', 'Demand', 'Demand'],
                         'Color':['red','purple','orange','blue'],
                         'Weight': [15,10,40,35]
                         })
     df1.sort_values('Volume', inplace = True)
     df1['Rank'] = df1['Volume'].rank(method='min')
     display(df1)
```

	Location	Latitude	Longitude	Volume	Location Type	Color	Weight	Rank
1	Earle Asphalt	43.58916	-79.64498	676	Demand	purple	10	1.0
0	Brunswick Sq Mall	43.60022	-79.71238	1279	Demand	red	15	2.0
3	EB Public Library	43.54998	-79.68079	1922	Demand	blue	35	3.0
2	GuttenPlans	43.54102	-79.60217	2568	Demand	orange	40	4.0

Figure 6

4.6: Graphical representation of volume of orders at each location

```
#graphical representation of the various volumes sold at each location

import matplotlib.pyplot as plt

plt.bar(df1['Location'], df1['Volume'])
plt.title('Volume For Vending Machine Locations')
plt.xlabel('Location')
plt.ylabel('Volume')
plt.show()
```

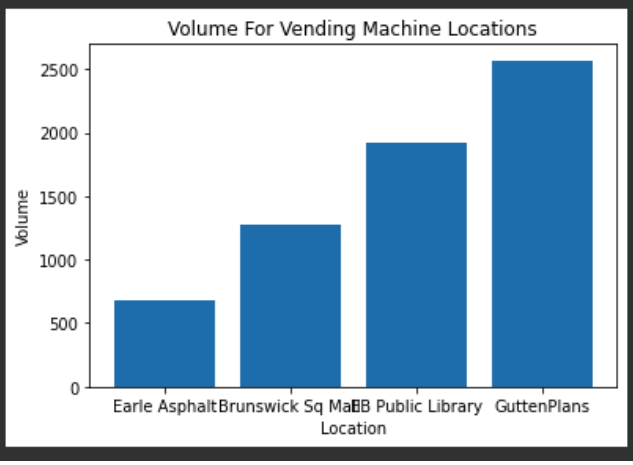


Figure 7

4.7: Solving for warehouse latitude

```
[ ] # ordering the Latitudes in ascending order
    df2 = df1[['Latitude','Weight']].copy()
    df2 = df2.sort_values('Latitude')
    df3 = pd.DataFrame(df2)
    sumweight = [40,75,85,100]
    df3['sumweight'] = sumweight
    display(df3)

    #the first sum weight that exceeds 50 is the corresponding latitude for the warehouse
```

	Latitude	Weight	sumweight
2	43.54102	40	40
3	43.54998	35	75
1	43.58916	10	85
0	43.60022	15	100

Figure 8

4.8: Solving for warehouse longitude

```
[ ]  df4 = df1[['Longitude','Weight']].copy()
     df4 = df4.sort_values('Longitude')
     df5 = pd.DataFrame(df4)
     sumweight = [15,50,60,100]
     df5['sumweight'] = sumweight
     display(df5)

     #the first sum weight that exceeds 50 is the corresponding longitude for the warehouse
```

	Longitude	Weight	sumweight
0	-79.71238	15	15
3	-79.68079	35	50
1	-79.64498	10	60
2	-79.60217	40	100

Figure 9

4.9: Adding warehouse location to the new dataset

```
[ ]  # warehouse location is Latitude: 43.54998 Long: -79.64498

[ ]  df1 = pd.DataFrame({'Location': ['Brunswick Sq Mall', 'Earle Asphalt', 'GuttenPlans', 'EB Public Library','Warehouse'],
                         'Latitude': [43.60022, 43.58916, 43.54102, 43.54998, 43.54998],'Longitude': [-79.71238, -79.64498, -79.60217, -79.68079, -79.64498],
                          'Volume': [1279, 676, 2568, 1922, 1279+676+2568+1922],
                          'Location Type':['Demand', 'Demand', 'Demand', 'Demand','Supply'],
                          'Color':['red','purple','orange','blue', 'green'],
                          'Weight': [15,10,40,35,0]
                          })
     df1.sort_values('Volume', inplace = True)
     df1['Rank'] = df1['Volume'].rank(method='min')
     display(df1)
```

	Location	Latitude	Longitude	Volume	Location Type	Color	Weight	Rank
1	Earle Asphalt	43.58916	-79.64498	676	Demand	purple	10	1.0
0	Brunswick Sq Mall	43.60022	-79.71238	1279	Demand	red	15	2.0
3	EB Public Library	43.54998	-79.68079	1922	Demand	blue	35	3.0
2	GuttenPlans	43.54102	-79.60217	2568	Demand	orange	40	4.0
4	Warehouse	43.54998	-79.64498	6445	Supply	green	0	5.0

Figure 10

4.10: Plotting each location on map using ‘Folium’ library (You will need to zoom in on the coordinates to see them the way they are displayed in the figure below)

```
[ ] import folium
    import pandas as pd

    my_map = folium.Map()

    for i in range(5):

        city = df1.loc[i]
        folium.Marker(
            location = [city['Latitude'], city['Longitude']],
            popup = city['Location'],
            tooltip = city['Location'],
            icon = folium.Icon(color=city['Color'])

        ).add_to(my_map)


    my_map
```

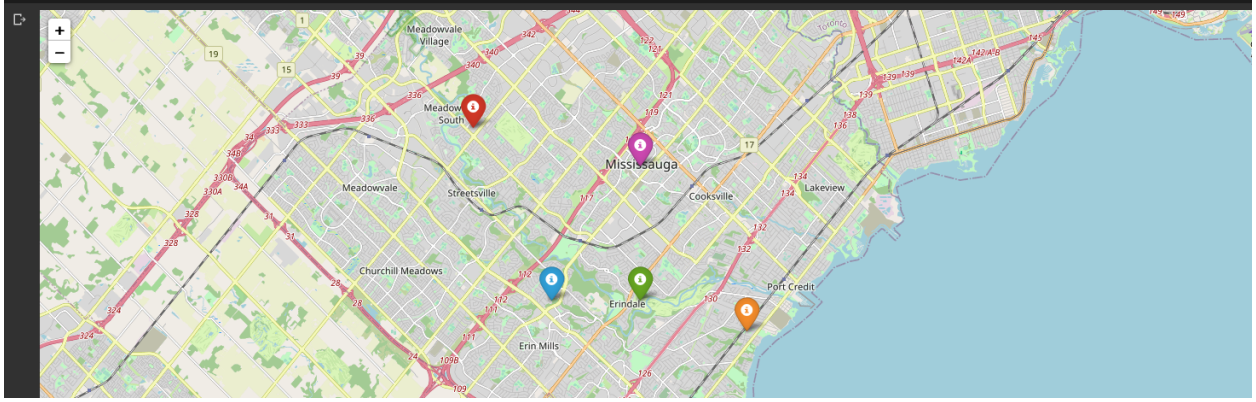


Figure 11

4.11: Installing mapping library ‘osmnx’

```
[ ] %pip install osmnx

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: osmnx in /usr/local/lib/python3.7/dist-packages (1.1.2)
    Requirement already satisfied: geopandas>=0.10 in /usr/local/lib/python3.7/dist-packages (from osmnx) (0.10.2)
    Requirement already satisfied: Shapely<2.0,>=1.7 in /usr/local/lib/python3.7/dist-packages (from osmnx) (1.8.5.post1)
    Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.7/dist-packages (from osmnx) (2.28.1)
    Requirement already satisfied: matplotlib>=3.4 in /usr/local/lib/python3.7/dist-packages (from osmnx) (3.5.3)
    Requirement already satisfied: Rtree>=0.9 in /usr/local/lib/python3.7/dist-packages (from osmnx) (1.0.1)
    Requirement already satisfied: numpy>=1.21 in /usr/local/lib/python3.7/dist-packages (from osmnx) (1.21.6)
    Requirement already satisfied: networkx>=2.6 in /usr/local/lib/python3.7/dist-packages (from osmnx) (2.6.3)
    Requirement already satisfied: pandas>=1.3 in /usr/local/lib/python3.7/dist-packages (from osmnx) (1.3.5)
    Requirement already satisfied: pyproj>=3.2 in /usr/local/lib/python3.7/dist-packages (from osmnx) (3.2.1)
    Requirement already satisfied: fiona>=1.8 in /usr/local/lib/python3.7/dist-packages (from geopandas>=0.10->osmnx) (1.8.22)
    Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (2.5.0)
    Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (7.1.2)
    Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (22.1.0)
    Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (1.15.0)
    Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (57.4.0)
    Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (2022.9.24)
    Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (1.1.1)
    Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas>=0.10->osmnx) (0.7.2)
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (7.1.2)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (21.3)
    Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (3.0.9)
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (2.8.2)
    Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (4.38.0)
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (0.11.0)
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.4->osmnx) (1.4.4)
    Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=3.4->osmnx) (4.1.1)
    Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=1.3->osmnx) (2022.6)
    Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->osmnx) (2.1.1)
    Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->osmnx) (1.24.3)
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->osmnx) (2.10)
```

Figure 12

4.12: Determining shortest route using ‘drive’ between warehouse and Earle Asphalt (Takes 1-2 mins to run)

```
[ ] import osmnx as ox
    import networkx as nx


    ox.config(log_console=True, use_cache=True)
    # Location of Warehouse and Earle Asphalt

    start_latitudelongitude = (43.54998,-79.64498) #warehouse location
    end_latitudelongitude = (43.58916,-79.64498) # Earle Asphalt

    place = 'Mississauga, Ontario, Canada'

    mode = 'drive'

    optimizer = 'time'

    graph = ox.graph_from_place(place, network_type = mode)

    orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

    dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

    shortest_route = nx.shortest_path(graph,
                                      orig_node,
                                      dest_node,
                                      weight=optimizer)

    shortest_route

    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the mc
      warnings.warn(msg)
    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the mc
      warnings.warn(msg)
    [449223260,
     449223262,
     449223276,
     50781511,
     3956981106,
     3956981114,
     3956981120,
     1585267344,
     1585267358,
     1585267361,
     1585267392,
     448855373,
     50781501,
     319230991
```

Figure 13

4.13: Distance calculation between warehouse and Earle Asphalt

```
[ ]  from geopy.distance import distance

     km = distance(start_latitudelongitude, end_latitudelongitude).km

     miles = distance(start_latitudelongitude, end_latitudelongitude).miles

     km, miles

     (4.353048495928998, 2.7048589337823343)
```

Figure 14

4.14: Time calculation between warehouse and Earle Asphalt

```
[ ]  speed = 50 #km/h

     averagetime = (km/speed)*60 #multipling by 60 to get the minutes it takes to travel from warehouse to Earle Asphalt

     print("The approximate time to travel from warehouse to Earle Asphalt is:", averagetime, "mins")

     The approximate time to travel from warehouse to Earle Asphalt is: 5.223658195114797 mins
```

Figure 15

4.15: Optimal 'Drive' route between warehouse and Earle Asphalt

```
[ ]  import folium

     shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                               tiles='openstreetmap')
     folium.TileLayer('openstreetmap').add_to(shortest_route_map)
     folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
     folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
     folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
     folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
     folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
     folium.LayerControl().add_to(shortest_route_map)

     start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
     end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
     start_marker = folium.Marker(
                 location = start_latitudelongitude,
                 popup = 'Warehouse',
                 tooltip = 'Warehouse',
                 icon = folium.Icon(color='green'))
     end_marker = folium.Marker(
                 location = end_latitudelongitude,
                 popup = 'Earle Asphalt',
                 tooltip = 'Earle Asphalt',
                 icon = folium.Icon(color='purple'))
     # add the circle marker to the map
     start_marker.add_to(shortest_route_map)
     end_marker.add_to(shortest_route_map)

     shortest_route_map
```

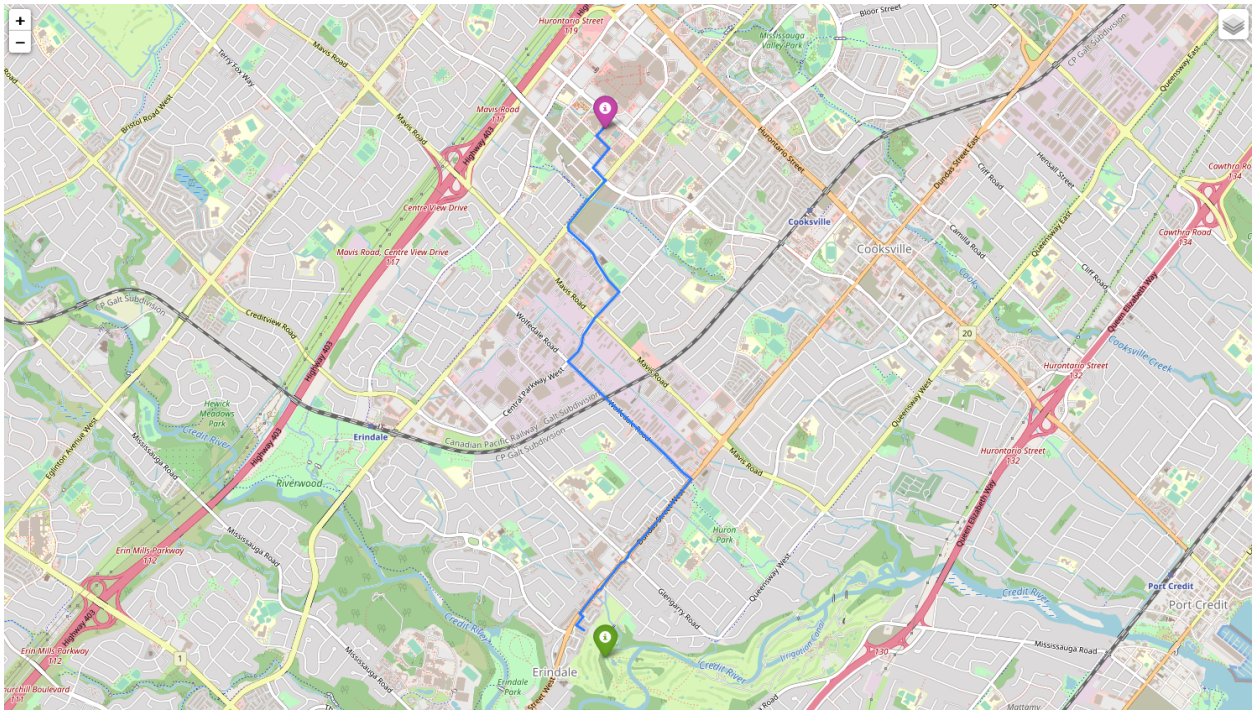


Figure 16

4.16: Routing optimal route between warehouse and Earle Asphalt when using ‘bike’ (Takes 1-2 mins to run)

```
[ ]  #mapping the route from Warehouse location to Earle Asphalt same as before but changing the mode to 'bike' instead of 'drive'

     import osmnx as ox
     import networkx as nx


     ox.config(log_console=True, use_cache=True)
     # Location of Warehouse and Earle Asphalt

     start_latitudelongitude = (43.54998,-79.64498) #warehouse location
     end_latitudelongitude = (43.58916,-79.64498) # Earle Asphalt

     place = 'Mississauga, Ontario, Canada'

     mode = 'bike'

     optimizer = 'time'

     graph = ox.graph_from_place(place, network_type = mode)

     orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

     dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

     shortest_route = nx.shortest_path(graph,
                                       orig_node,
                                       dest_node,
                                       weight=optimizer)

     shortest_route

     /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the n
       warnings.warn(msg)
     /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the n
       warnings.warn(msg)
     [4482788603,
      4482811426,
      4613864250,
      4613864252,
      3956980939,
      3956980943,
      3956980953,
      3956981067,
      450510135,
      9460802838,
      9460802834,
```

Figure 17

4.17: Mapping the optimal ‘bike’ route between warehouse and Earle Asphalt

```
[ ]  import folium

     shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                               tiles='openstreetmap')
     folium.TileLayer('openstreetmap').add_to(shortest_route_map)
     folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
     folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
     folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
     folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
     folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
     folium.LayerControl().add_to(shortest_route_map)

     start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
     end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
     start_marker = folium.Marker(
                 location = start_latitudelongitude,
                 popup = 'Warehouse',
                 tooltip = 'Warehouse',
                 icon = folium.Icon(color='green'))
     end_marker = folium.Marker(
                 location = end_latitudelongitude,
                 popup = 'Earle Asphalt',
                 tooltip = 'Earle Asphalt',
                 icon = folium.Icon(color='purple'))
     # add the circle marker to the map
     start_marker.add_to(shortest_route_map)
     end_marker.add_to(shortest_route_map)

     shortest_route_map
```

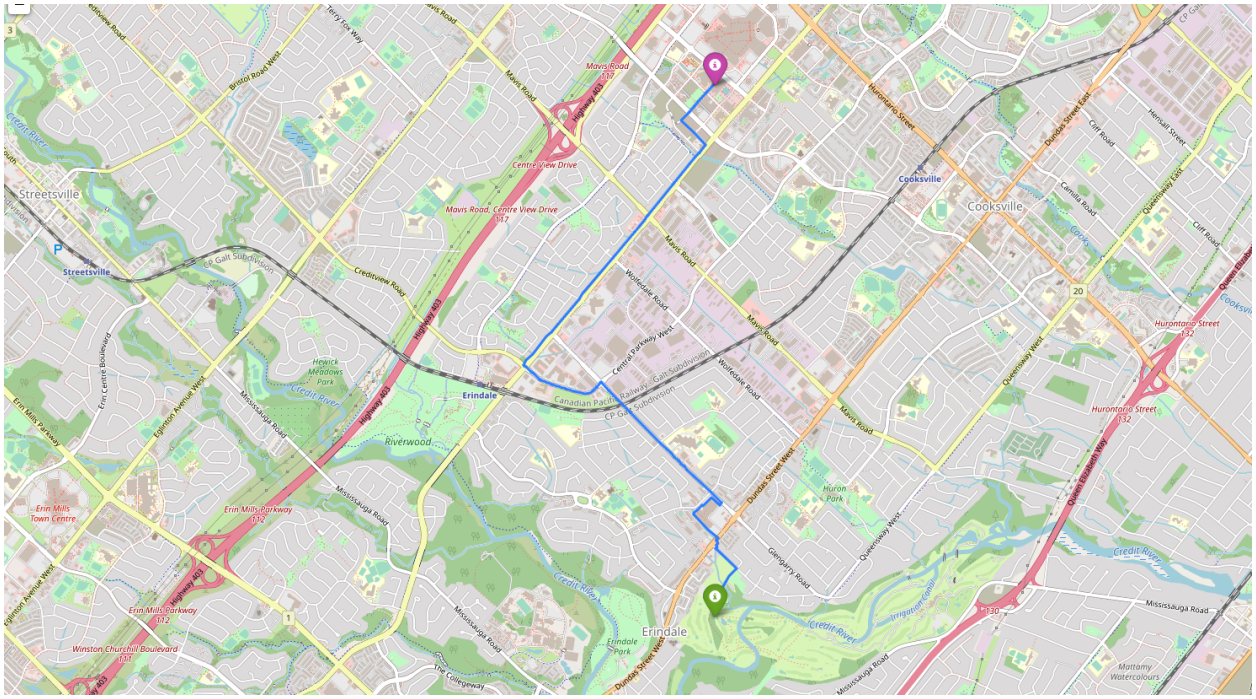
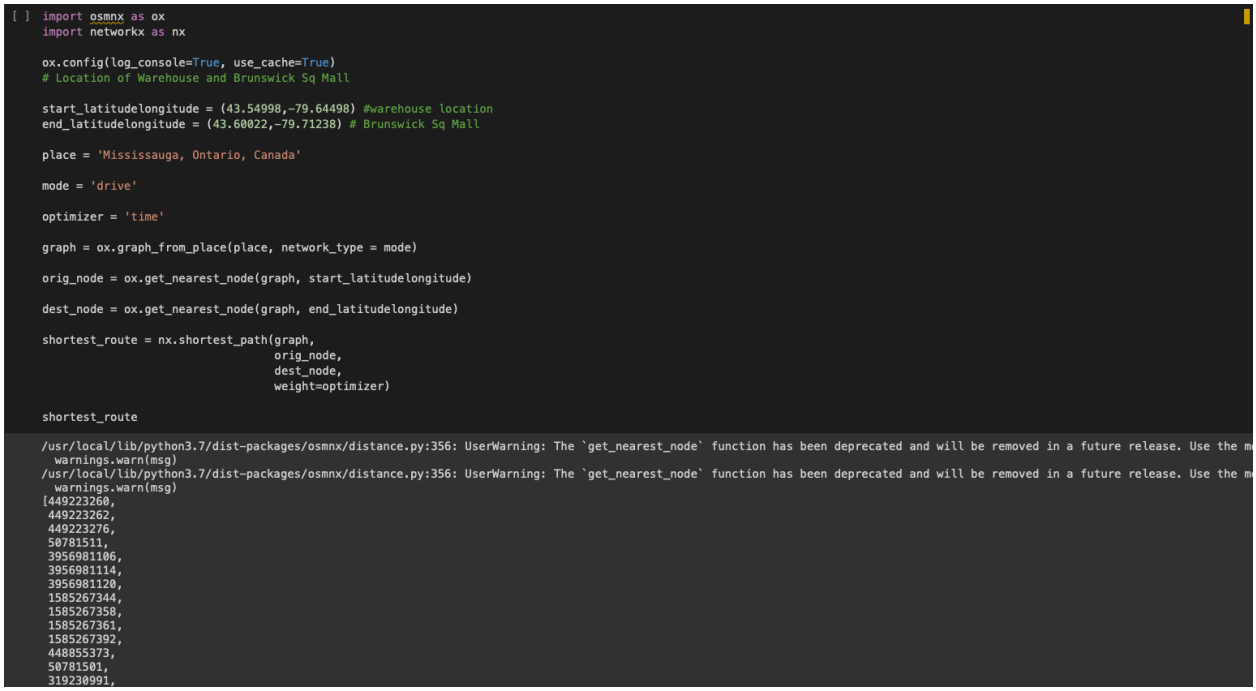


Figure 18

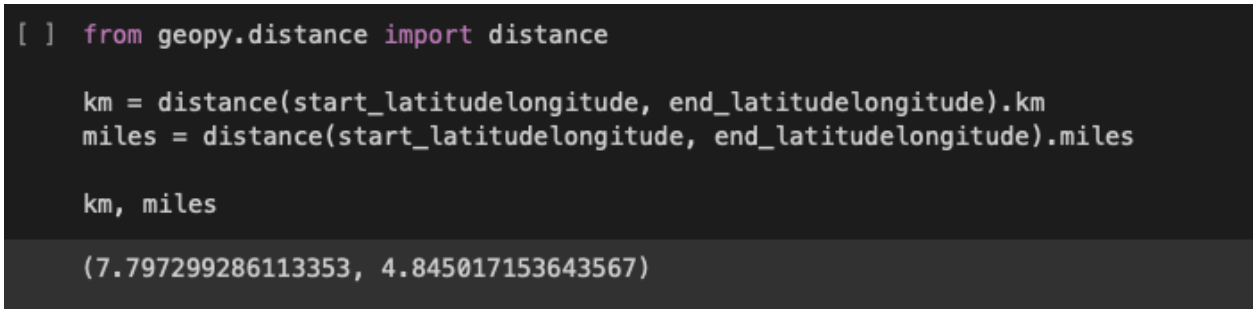
4.18: Determining shortest route using ‘drive’ between warehouse and Brunswick Sq Mall (Takes 1-2 mins to run)



```
[ ] import osmnx as ox
    import networkx as nx

    ox.config(log_console=True, use_cache=True)
    # Location of Warehouse and Brunswick Sq Mall

    start_latitudelongitude = (43.54998,-79.64498) #warehouse location
    end_latitudelongitude = (43.60022,-79.71238) # Brunswick Sq Mall

    place = 'Mississauga, Ontario, Canada'

    mode = 'drive'

    optimizer = 'time'

    graph = ox.graph_from_place(place, network_type = mode)

    orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

    dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

    shortest_route = nx.shortest_path(graph,
                                      orig_node,
                                      dest_node,
                                      weight=optimizer)

    shortest_route

    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the m
      warnings.warn(msg)
    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the m
      warnings.warn(msg)
    [449223260,
     449223262,
     449223276,
     50781511,
     3956981106,
     3956981114,
     3956981120,
     1585267344,
     1585267358,
     1585267361,
     1585267392,
     448855373,
     50781501,
     319230991,
```

Figure 19

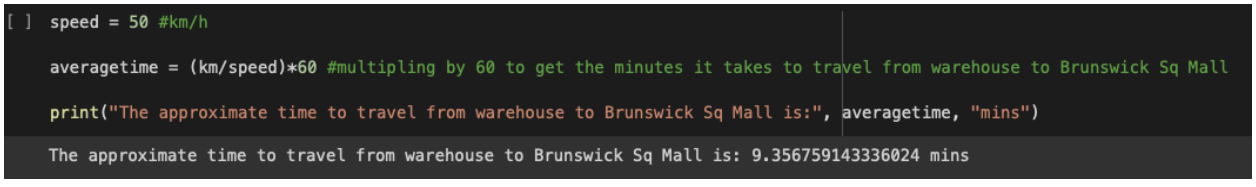
4.19: Distance calculation between warehouse and Brunswick Sq Mall



```
[ ] from geopy.distance import distance

    km = distance(start_latitudelongitude, end_latitudelongitude).km
    miles = distance(start_latitudelongitude, end_latitudelongitude).miles

    km, miles

    (7.797299286113353, 4.845017153643567)
```

Figure 20

4.20: Time calculation between warehouse and Brunswick Sq Mall



```
[ ] speed = 50 #km/h

    averagetime = (km/speed)*60 #multipling by 60 to get the minutes it takes to travel from warehouse to Brunswick Sq Mall

    print("The approximate time to travel from warehouse to Brunswick Sq Mall is:", averagetime, "mins")

    The approximate time to travel from warehouse to Brunswick Sq Mall is: 9.356759143336024 mins
```

Figure 21

4.21: Optimal ‘Drive’ route between warehouse and Brunswick Sq Mall

```
[ ]  import folium

    shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                              tiles='openstreetmap')
    folium.TileLayer('openstreetmap').add_to(shortest_route_map)
    folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
    folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
    folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
    folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
    folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
    folium.LayerControl().add_to(shortest_route_map)

    start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
    end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
    start_marker = folium.Marker(
                location = start_latitudelongitude,
                popup = 'Warehouse',
                tooltip = 'Warehouse',
                icon = folium.Icon(color='green'))
    end_marker = folium.Marker(
                location = end_latitudelongitude,
                popup = 'Brunswick Sq Mall',
                tooltip = 'Brunswick Sq Mall',
                icon = folium.Icon(color='red'))
    # add the circle marker to the map
    start_marker.add_to(shortest_route_map)
    end_marker.add_to(shortest_route_map)

    shortest_route_map
```

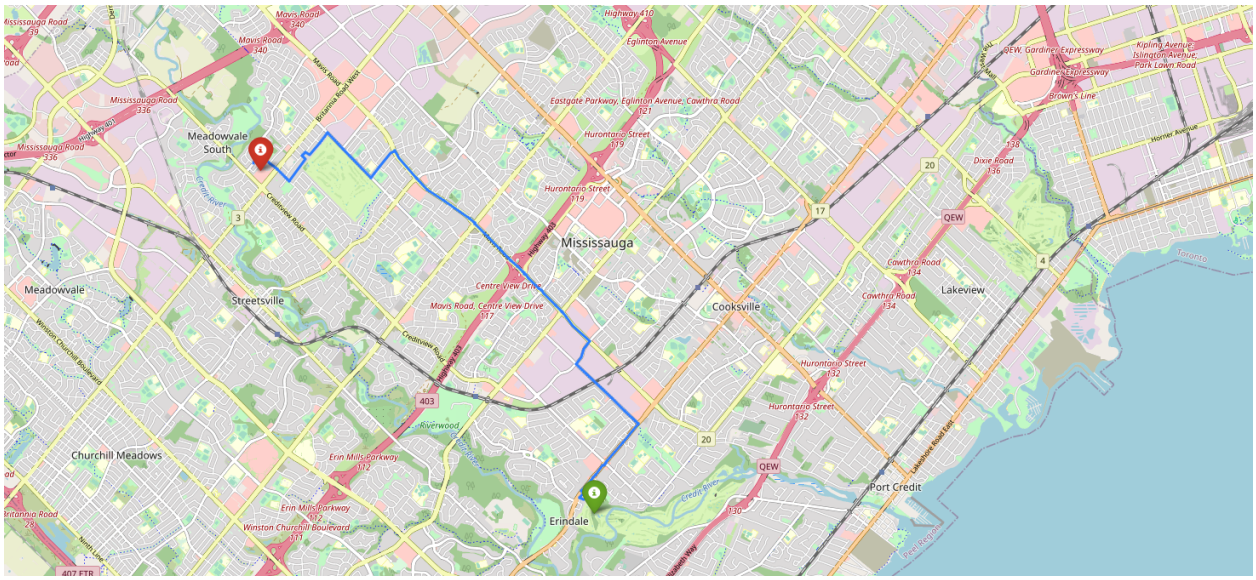


Figure 22

4.22: Routing optimal route between warehouse and Brunswick Sq Mall when using ‘bike’ (Takes 1-2 mins to run)

```
[ ] import osmnx as ox
    import networkx as nx

    ox.config(log_console=True, use_cache=True)
    # Location of Warehouse and Brunswick Sq Mall

    start_latitudelongitude = (43.54998,-79.64498) #warehouse location
    end_latitudelongitude = (43.60022,-79.71238) # Brunswick Sq Mall

    place = 'Mississauga, Ontario, Canada'

    mode = 'bike'

    optimizer = 'time'

    graph = ox.graph_from_place(place, network_type = mode)

    orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

    dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

    shortest_route = nx.shortest_path(graph,
                                      orig_node,
                                      dest_node,
                                      weight=optimizer)

    shortest_route

    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the n
      warnings.warn(msg)
    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the n
      warnings.warn(msg)
    [4482788603,
     4482811426,
     4613864250,
     4613864252,
     3956980939,
     3956980943,
     3956980953,
     3956981067,
     450510135,
     9460802838,
     9460802834,
     3956981120,
     4613899884,
     4613899870,
```

Figure 23

4.23: Mapping the optimal ‘bike’ route between warehouse and Brunswick Sq Mall

```
[ ] import folium

    shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                              tiles='openstreetmap')
    folium.TileLayer('openstreetmap').add_to(shortest_route_map)
    folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
    folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
    folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
    folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
    folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
    folium.LayerControl().add_to(shortest_route_map)

    start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
    end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
    start_marker = folium.Marker(
                location = start_latitudelongitude,
                popup = 'Warehouse',
                tooltip = 'Warehouse',
                icon = folium.Icon(color='green'))
    end_marker = folium.Marker(
                location = end_latitudelongitude,
                popup = 'Brunswick Sq Mall',
                tooltip = 'Brunswick Sq Mall',
                icon = folium.Icon(color='red'))
    # add the circle marker to the map
    start_marker.add_to(shortest_route_map)
    end_marker.add_to(shortest_route_map)

    shortest_route_map
```

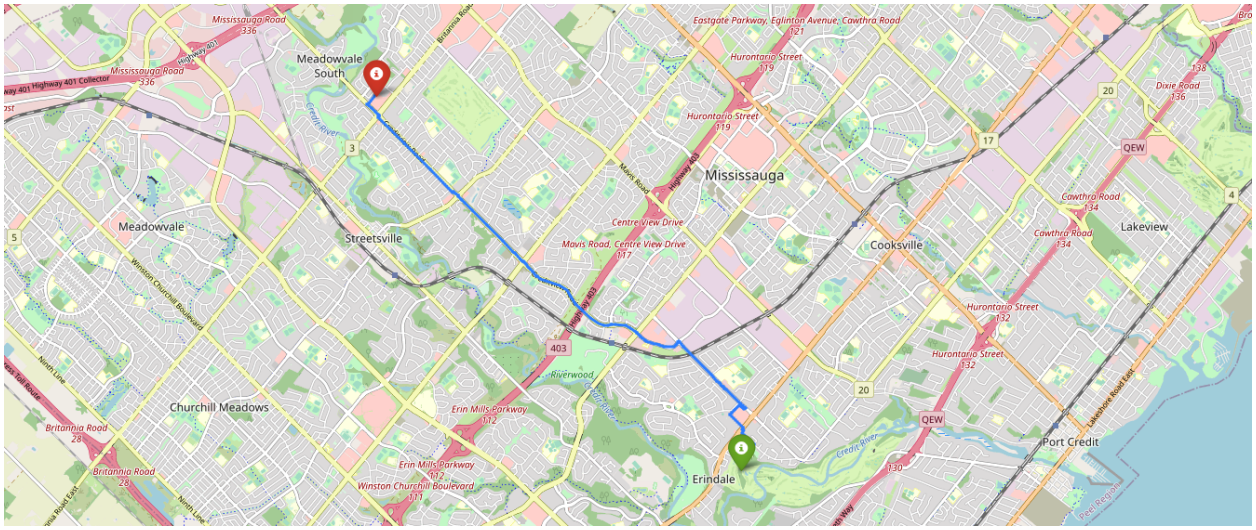


Figure 24

4.24: Determining shortest route using ‘drive’ between warehouse and EB Public Library (Takes 1-2 mins to run)

```
[ ] import osmnx as ox
    import networkx as nx

    ox.config(log_console=True, use_cache=True)
    # Location of Warehouse and EB Public Library

    start_latitudelongitude = (43.54998,-79.64498) #warehouse location
    end_latitudelongitude = (43.54998,-79.68079) # EB Public Library

    place = 'Mississauga, Ontario, Canada'

    mode = 'drive'

    optimizer = 'time'

    graph = ox.graph_from_place(place, network_type = mode)

    orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

    dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

    shortest_route = nx.shortest_path(graph,
                                      orig_node,
                                      dest_node,
                                      weight=optimizer)

    shortest_route

    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the m
      warnings.warn(msg)
    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the m
      warnings.warn(msg)
    [449223260,
     449223262,
     449223278,
     449223255,
     449224111,
     449224121,
     460052735,
     369995045,
     369995048,
     369995052,
     460052709,
     369995162,
     2383794463,
```

Figure 25

4.25: Distance calculation between warehouse and EB Public Library

```
[ ] from geopy.distance import distance

    km = distance(start_latitudelongitude, end_latitudelongitude).km
    miles = distance(start_latitudelongitude, end_latitudelongitude).miles

    km, miles

    (2.893803734451265, 1.7981262765768318)
```

Figure 26

4.26: Time calculation between warehouse and EB Public Library

```
[ ]  speed = 50 #km/h

     averagetime = (km/speed)*60 #multipling by 60 to get the minutes it takes to travel from warehouse to EB Public Library

     print("The approximate time to travel from warehouse to EB Public Library is:", averagetime, "mins")

     The approximate time to travel from warehouse to EB Public Library is: 3.4725644813415175 mins
```

Figure 27

4.27: Optimal 'Drive' route between warehouse and EB Public Library

```
[ ]  import folium

     shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                               tiles='openstreetmap')
     folium.TileLayer('openstreetmap').add_to(shortest_route_map)
     folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
     folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
     folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
     folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
     folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
     folium.LayerControl().add_to(shortest_route_map)

     start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
     end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
     start_marker = folium.Marker(
                 location = start_latitudelongitude,
                 popup = 'Warehouse',
                 tooltip = 'Warehouse',
                 icon = folium.Icon(color='green'))
     end_marker = folium.Marker(
                 location = end_latitudelongitude,
                 popup = 'EB Public Library',
                 tooltip = 'EB Public Library',
                 icon = folium.Icon(color='blue'))
     # add the circle marker to the map
     start_marker.add_to(shortest_route_map)
     end_marker.add_to(shortest_route_map)

     shortest_route_map
```

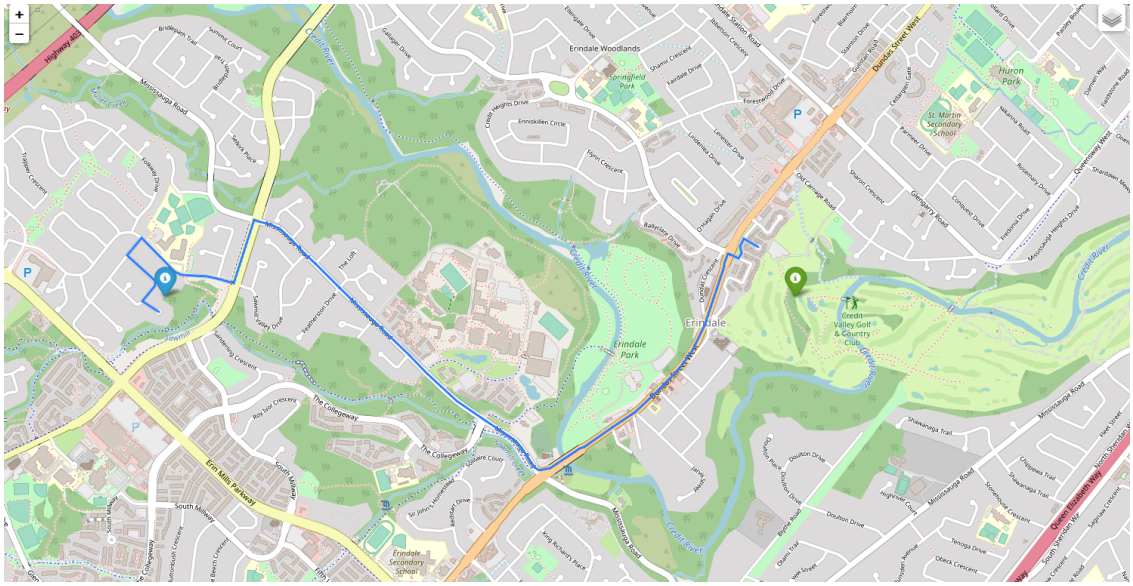


Figure 28

4.28: Routing optimal route between warehouse and EB Public Library when using ‘bike’ (Takes 1-2 mins to run)

```
[ ] import osmnx as ox
    import networkx as nx

    ox.config(log_console=True, use_cache=True)
    # Location of Warehouse and EB Public Library

    start_latitudelongitude = (43.54998,-79.64498) #warehouse location
    end_latitudelongitude = (43.54998,-79.68079) # EB Public Library

    place = 'Mississauga, Ontario, Canada'

    mode = 'bike'

    optimizer = 'time'

    graph = ox.graph_from_place(place, network_type = mode)

    orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

    dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

    shortest_route = nx.shortest_path(graph,
                                      orig_node,
                                      dest_node,
                                      weight=optimizer)

    shortest_route

    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use th
      warnings.warn(msg)
    /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use th
      warnings.warn(msg)
    [4482788603,
     4482811426,
     4613864250,
     4613864252,
     3956980939,
     3956980943,
     3956980953,
     3956981067,
     450510135,
     9460802838,
     3956981114,
     50781507,
     3956981106,
     5919986777,
```

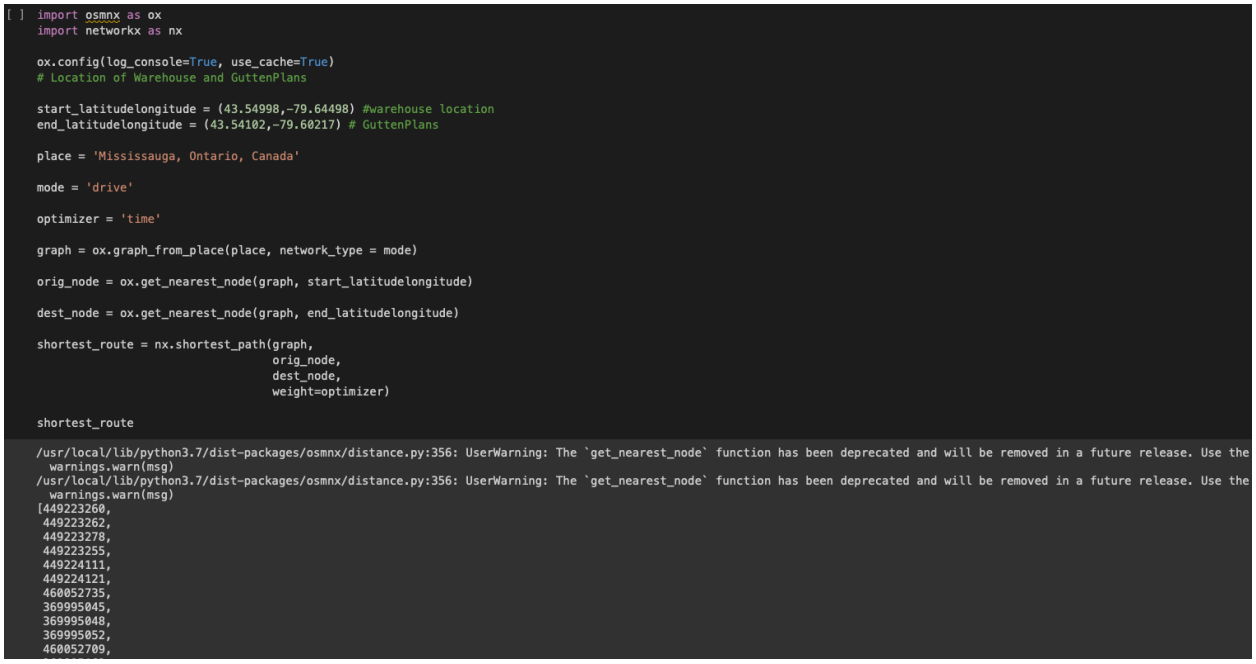
Figure 29

4.29: Mapping the optimal ‘bike’ route between warehouse and EB Public Library

```
[ ]  import folium

     shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                               tiles='openstreetmap')
     folium.TileLayer('openstreetmap').add_to(shortest_route_map)
     folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
     folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
     folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
     folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
     folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
     folium.LayerControl().add_to(shortest_route_map)

     start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
     end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
     start_marker = folium.Marker(
                location = start_latitudelongitude,
                popup = 'Warehouse',
                tooltip = 'Warehouse',
                icon = folium.Icon(color='green'))
     end_marker = folium.Marker(
                location = end_latitudelongitude,
                popup = 'EB Public Library',
                tooltip = 'EB Public Library',
                icon = folium.Icon(color='blue'))
     # add the circle marker to the map
     start_marker.add_to(shortest_route_map)
     end_marker.add_to(shortest_route_map)

     shortest_route_map
```

Figure 30

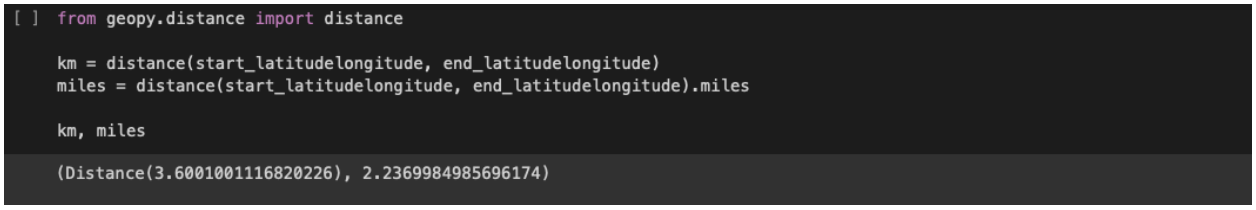
4.30: Determining shortest route using ‘drive’ between warehouse and Gutten Plans (Takes 1-2 mins to run)



```
[ ]  import osmnx as ox
     import networkx as nx

     ox.config(log_console=True, use_cache=True)
     # Location of Warehouse and GuttenPlans

     start_latitudelongitude = (43.54998,-79.64498) #warehouse location
     end_latitudelongitude = (43.54102,-79.60217) # GuttenPlans

     place = 'Mississauga, Ontario, Canada'

     mode = 'drive'

     optimizer = 'time'

     graph = ox.graph_from_place(place, network_type = mode)

     orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

     dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

     shortest_route = nx.shortest_path(graph,
                                       orig_node,
                                       dest_node,
                                       weight=optimizer)

     shortest_route

     /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the
       warnings.warn(msg)
     /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the
       warnings.warn(msg)
     [449223260,
      449223262,
      449223278,
      449223255,
      449224111,
      449224121,
      460052735,
      369995045,
      369995048,
      369995052,
      460052709,
```

Figure 31

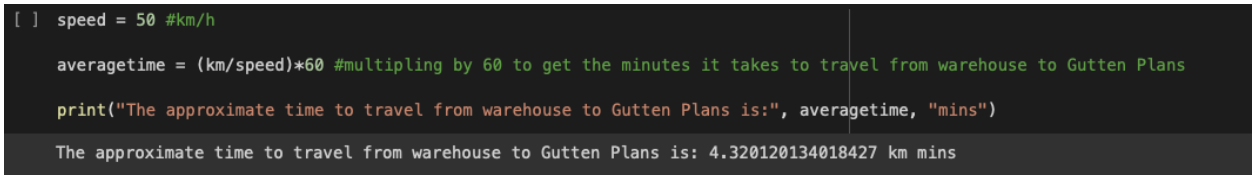
4.31: Distance calculation between warehouse and Gutten Plans



```
[ ]  from geopy.distance import distance

     km = distance(start_latitudelongitude, end_latitudelongitude)
     miles = distance(start_latitudelongitude, end_latitudelongitude).miles

     km, miles

     (Distance(3.6001001116820226), 2.2369984985696174)
```

Figure 32

4.32: Time calculation between warehouse and Gutten Plans



```
[ ]  speed = 50 #km/h

     averagetime = (km/speed)*60 #multipling by 60 to get the minutes it takes to travel from warehouse to Gutten Plans

     print("The approximate time to travel from warehouse to Gutten Plans is:", averagetime, "mins")

     The approximate time to travel from warehouse to Gutten Plans is: 4.320120134018427 km mins
```

Figure 33

4.33: Optimal ‘Drive’ route between warehouse and Gutten Plans

```
[ ]  import folium

    shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                              tiles='openstreetmap')
    folium.TileLayer('openstreetmap').add_to(shortest_route_map)
    folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
    folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
    folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
    folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
    folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
    folium.LayerControl().add_to(shortest_route_map)

    start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
    end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
    start_marker = folium.Marker(
                location = start_latitudelongitude,
                popup = 'Warehouse',
                tooltip = 'Warehouse',
                icon = folium.Icon(color='green'))
    end_marker = folium.Marker(
                location = end_latitudelongitude,
                popup = 'Gutten Plans',
                tooltip = 'Gutten Plans',
                icon = folium.Icon(color='orange'))
    # add the circle marker to the map
    start_marker.add_to(shortest_route_map)
    end_marker.add_to(shortest_route_map)

    shortest_route_map
```

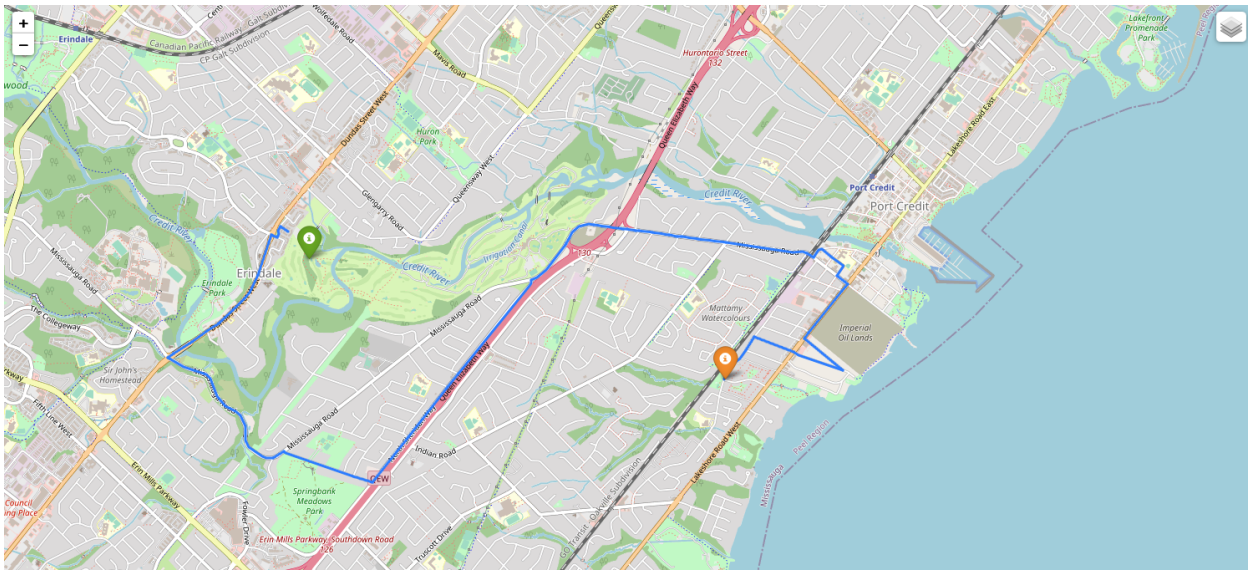
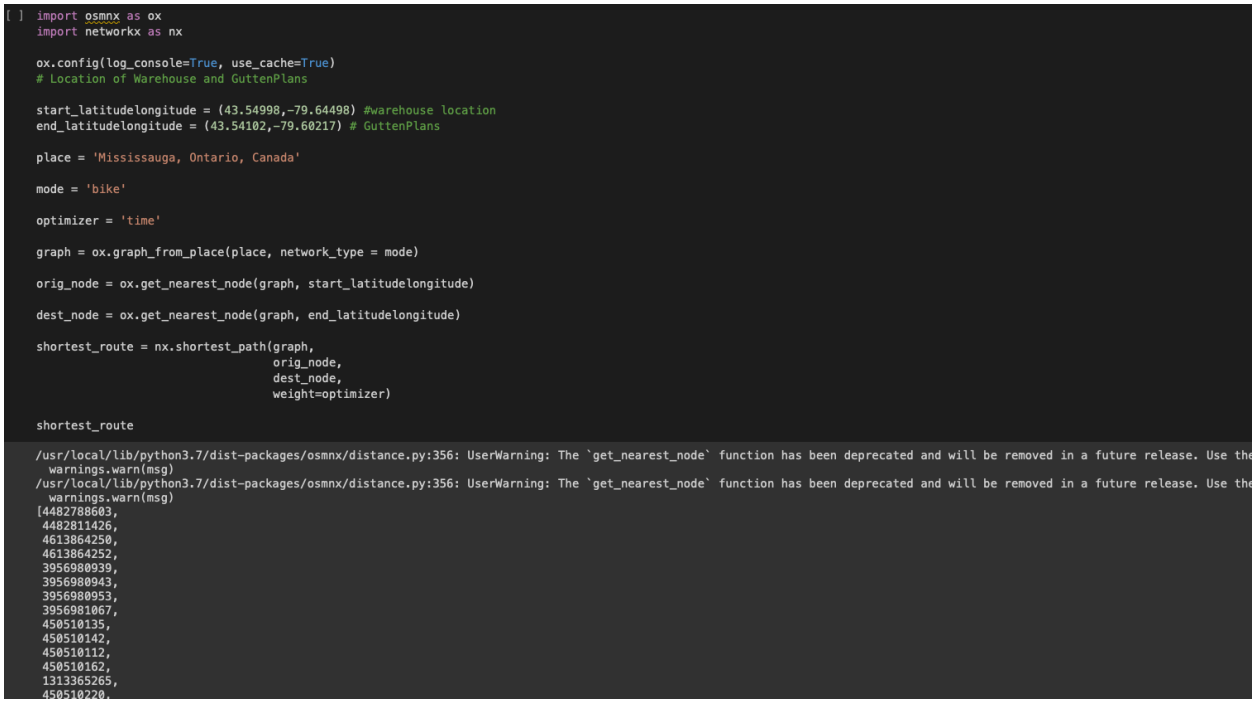


Figure 34

4.34: Routing optimal route between warehouse and Gutten Plans when using ‘bike’ (Takes 1-2 mins to run)



```
[ ]  import osmnx as ox
     import networkx as nx

     ox.config(log_console=True, use_cache=True)
     # Location of Warehouse and GuttenPlans

     start_latitudelongitude = (43.54998,-79.64498) #warehouse location
     end_latitudelongitude = (43.54102,-79.60217) # GuttenPlans

     place = 'Mississauga, Ontario, Canada'

     mode = 'bike'

     optimizer = 'time'

     graph = ox.graph_from_place(place, network_type = mode)

     orig_node = ox.get_nearest_node(graph, start_latitudelongitude)

     dest_node = ox.get_nearest_node(graph, end_latitudelongitude)

     shortest_route = nx.shortest_path(graph,
                                       orig_node,
                                       dest_node,
                                       weight=optimizer)

     shortest_route

     /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the
       warnings.warn(msg)
     /usr/local/lib/python3.7/dist-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future release. Use the
       warnings.warn(msg)
     [4482788603,
      4482811426,
      4613864250,
      4613864252,
      3956980939,
      3956980943,
      3956980953,
      3956981067,
      450510135,
      450510142,
      450510112,
      450510162,
      1313365265,
      450510220,
```

Figure 35

4.35: Mapping the optimal ‘bike’ route between warehouse and Gutten Plans

```
[ ]  import folium

     shortest_route_map = ox.plot_route_folium(graph, shortest_route,
                                               tiles='openstreetmap')
     folium.TileLayer('openstreetmap').add_to(shortest_route_map)
     folium.TileLayer('Stamen Terrain').add_to(shortest_route_map)
     folium.TileLayer('Stamen Toner').add_to(shortest_route_map)
     folium.TileLayer('Stamen Water Color').add_to(shortest_route_map)
     folium.TileLayer('cartodbpositron').add_to(shortest_route_map)
     folium.TileLayer('cartodbdark_matter').add_to(shortest_route_map)
     folium.LayerControl().add_to(shortest_route_map)

     start_latitudelongitude = (start_latitudelongitude[0],start_latitudelongitude[1])
     end_latitudelongitude  = (end_latitudelongitude[0],end_latitudelongitude[1])
     start_marker = folium.Marker(
                 location = start_latitudelongitude,
                 popup = 'Warehouse',
                 tooltip = 'Warehouse',
                 icon = folium.Icon(color='green'))
     end_marker = folium.Marker(
                 location = end_latitudelongitude,
                 popup = 'Gutten Plans',
                 tooltip = 'Gutten Plans',
                 icon = folium.Icon(color='orange'))
     # add the circle marker to the map
     start_marker.add_to(shortest_route_map)
     end_marker.add_to(shortest_route_map)

     shortest_route_map
```

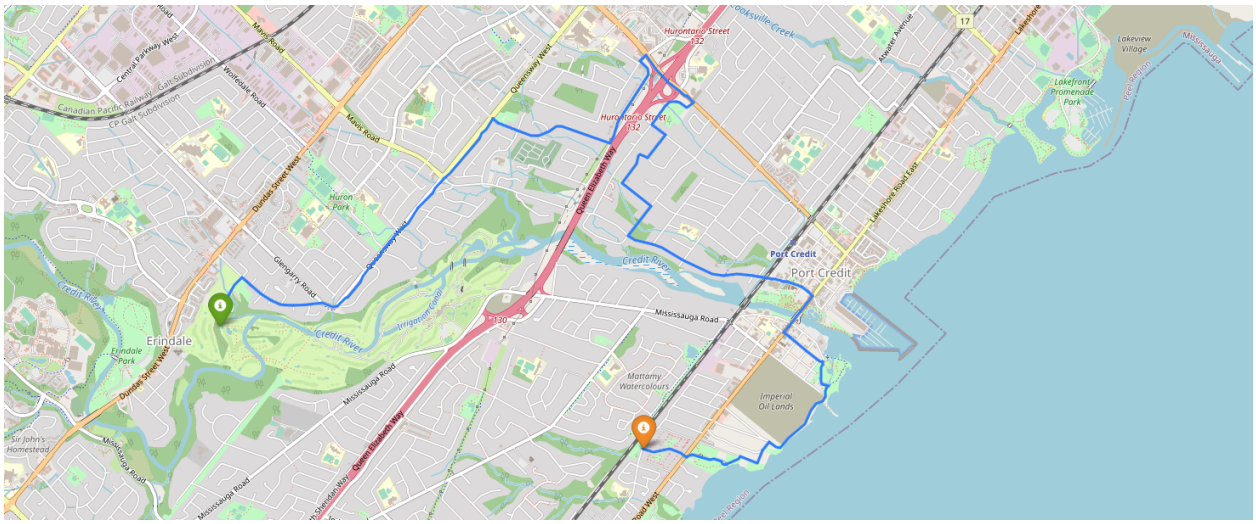


Figure 36