Python Solutions

M. Tsaqif Wismadi
S2431645

Code Manual: Machine Learning for Traffic Modelling

February 2022

UNIVERSITY OF TWENTE. | ITC

Data Requirement

In order to run this program, three sets of geospatial data along with its complete data format are required. The details of these dataset can be seen in the table below:

Dataset	Format	Description
Eixample District	.cpg, .dbf, .prj, .sbn, .sbx, .shp	Location boundary of study area
Clustering data pre SB	.cpg, .dbf, .prj, .sbn, .sbx, .shp, .shp.xml, .shx	Traffic data and street network parameters which recorded pre-implementation of superblock policy
Clustering data post SB	.cpg, .dbf, .prj, .sbn, .sbx, .shp, .shp.xml, .shx	Traffic data and street network parameters which recorded post-implementation of superblock policy

Package Requirement

After the datasets are checked and provided, several python packages have to be installed before the program can be run. Make sure you have installed all of the packages in the table below before proceeding to the next step.

Package name	Package function
Contextily	Allowing the program to access tile sets exposed through the popular XYZ format and include them in the workflow through matplotlib
Pandas	Data alteration and manipulation
Numpy	Array processing
Geopandas	Extends the datatypes used by pandas for spatial operations on geometric types
Geoplot	Plots latitude and longitude coordinates using Mercator or Lambert transformation
Scikit Learn	To implement machine learning models and statistical modelling
Matplotlib	To create static visualization of data

Step 0: Importing Package Libraries

Write and run all the codes below to import all the methods and functions that we will use in our program.

```
#List of libraries
import contextily as cx
import pandas as pd
import geopandas as gpd
import geoplot
import numpy as np
from sklearn import cluster
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import math
```

Once you successfully run the cell, you will get some notification regarding the latest version of some packages or some warning regarding potential incompatibility. You can ignore this warning, it will not interrupt the program.

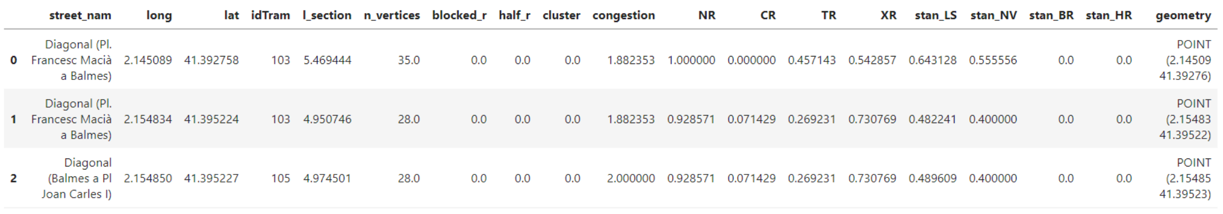
Step 1: Initial Mapping

First, we have to check on the data that we are going to use. To do this, we simply open the files using .read_file function from geopandas package.

- In:

  ```
  #Importing the pre-superblock data
  df1 = gpd.read_file(r"clustering data pre SB.shp")
  df1.head(3)
  ```

  Out:

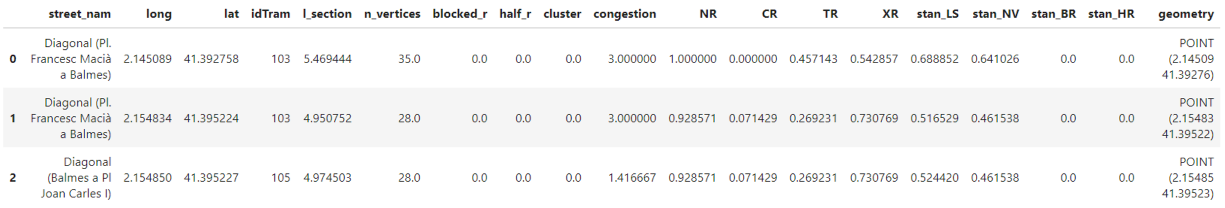


	street_nam	long	lat	idTram	l_section	n_vertices	blocked_r	half_r	cluster	congestion	NR	CR	TR	XR	stan_LS	stan_NV	stan_BR	stan_HR	geometry
0	Diagonal (Pl. Francesc Macià a Balmes)	2.145089	41.392758	103	5.469444	35.0	0.0	0.0	0.0	1.882353	1.000000	0.000000	0.457143	0.542857	0.643128	0.555556	0.0	0.0	POINT (2.14509 41.39276)
1	Diagonal (Pl. Francesc Macià a Balmes)	2.154834	41.395224	103	4.950746	28.0	0.0	0.0	0.0	1.882353	0.928571	0.071429	0.269231	0.730769	0.482241	0.400000	0.0	0.0	POINT (2.15483 41.39522)
2	Diagonal (Balmes a Pl Joan Carles I)	2.154850	41.395227	105	4.974501	28.0	0.0	0.0	0.0	2.000000	0.928571	0.071429	0.269231	0.730769	0.489609	0.400000	0.0	0.0	POINT (2.15485 41.39523)

- In:

  ```
  #Importing the post-superblock data
  df2 = gpd.read_file(r"clustering data post SB.shp")
  df2.head(3)
  ```

  Out:

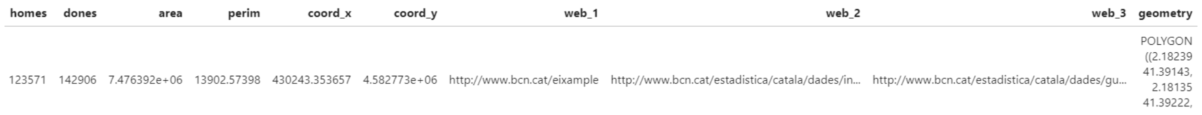


	street_nam	long	lat	idTram	l_section	n_vertices	blocked_r	half_r	cluster	congestion	NR	CR	TR	XR	stan_LS	stan_NV	stan_BR	stan_HR	geometry
0	Diagonal (Pl. Francesc Macià a Balmes)	2.145089	41.392758	103	5.469444	35.0	0.0	0.0	0.0	3.000000	1.000000	0.000000	0.457143	0.542857	0.688852	0.641026	0.0	0.0	POINT (2.14509 41.39276)
1	Diagonal (Pl. Francesc Macià a Balmes)	2.154834	41.395224	103	4.950752	28.0	0.0	0.0	0.0	3.000000	0.928571	0.071429	0.269231	0.730769	0.516529	0.461538	0.0	0.0	POINT (2.15483 41.39522)
2	Diagonal (Balmes a Pl Joan Carles I)	2.154850	41.395227	105	4.974503	28.0	0.0	0.0	0.0	1.416667	0.928571	0.071429	0.269231	0.730769	0.524420	0.461538	0.0	0.0	POINT (2.15485 41.39523)

- In:

  ```
  #Importing Eixample District Boundary
  boundary = gpd.read_file(r"Eixample District.shp")
  boundary.head()
  ```

Out:



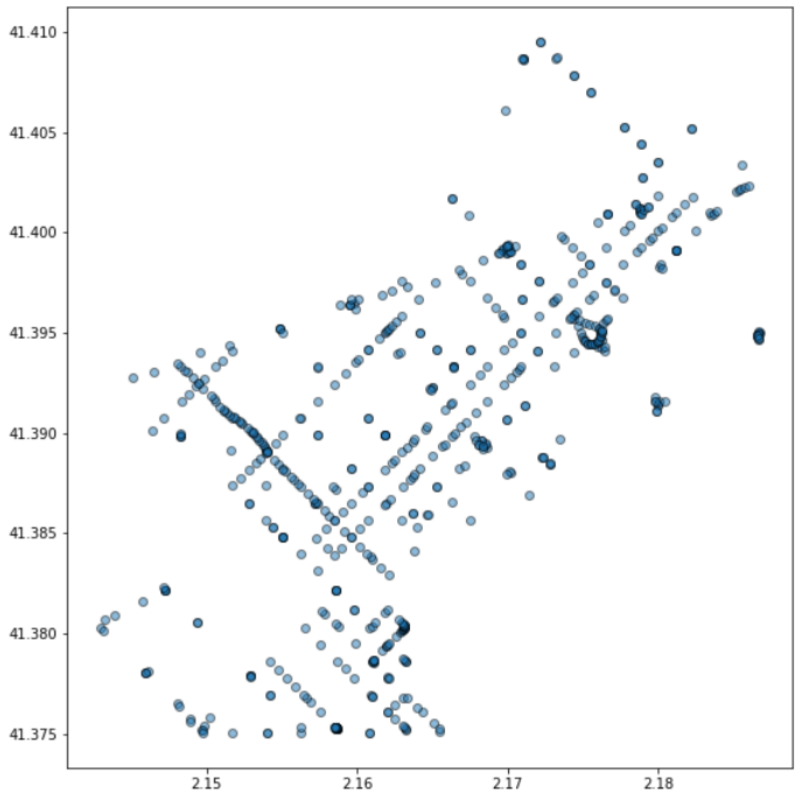
homes	dones	area	perim	coord_x	coord_y	web_1	web_2	web_3	geometry
123571	142906	7.476392e+06	13902.57398	430243.353657	4.582773e+06	http://www.bcn.cat/eixample	http://www.bcn.cat/estadistica/catala/dades/in...	http://www.bcn.cat/estadistica/catala/dades/gu...	POLYGON ((2.18239 41.39143, 2.18135 41.39222,

Once we see our main data, we proceed to plot the data to observe its spatial distribution using .plot method from matplotlib package.

- In:

```
#Check spatial plotting
ax = df1.plot(figsize = (10, 10), alpha = 0.5, edgecolor = 'k')
```

Out:



Now we have to check their coordinate system. Without knowing their coordinate system we will not be able to overlay them with a base map.

- In:

```
#Checking coordinate system
df1.crs
df2.crs
boundary.crs
```

Out:

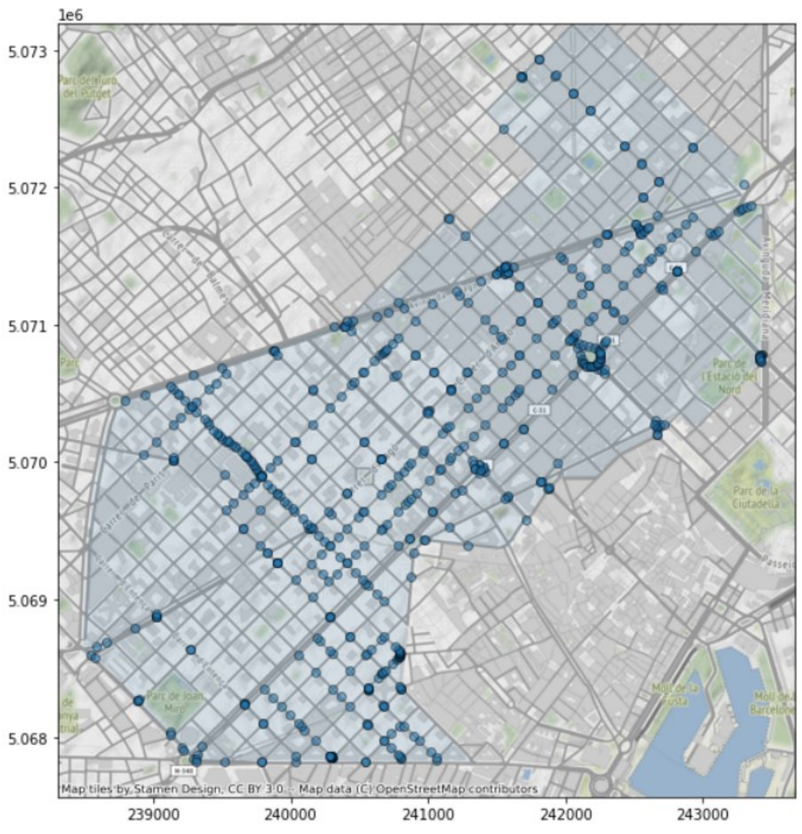
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

After we got the coordinate system, the next step will be projecting it to coordinate system of the base map. In this case, the base map is using WGS 84 or EPSG: 3857.

- In:

```
#Projecting data to the basemap coordinate
df1_wm = df1.to_crs(epsg = 3857)
df2_wm = df2.to_crs(epsg = 3857)
boundary_wm = boundary.to_crs(epsg = 3857)

#Adding basemap to data
ax = df1_wm.plot(figsize = (10, 10), alpha = 0.5, edgecolor = 'k')
cx.add_basemap(ax)
boundary_wm.plot(ax = ax, alpha = 0.1)
```

Out:


1e6
5.073
5.072
5.071
5.070
5.069
5.068
239000
240000
241000
242000
243000
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors

Step 2: Exploratory Spatial Data Analysis (ESDA)

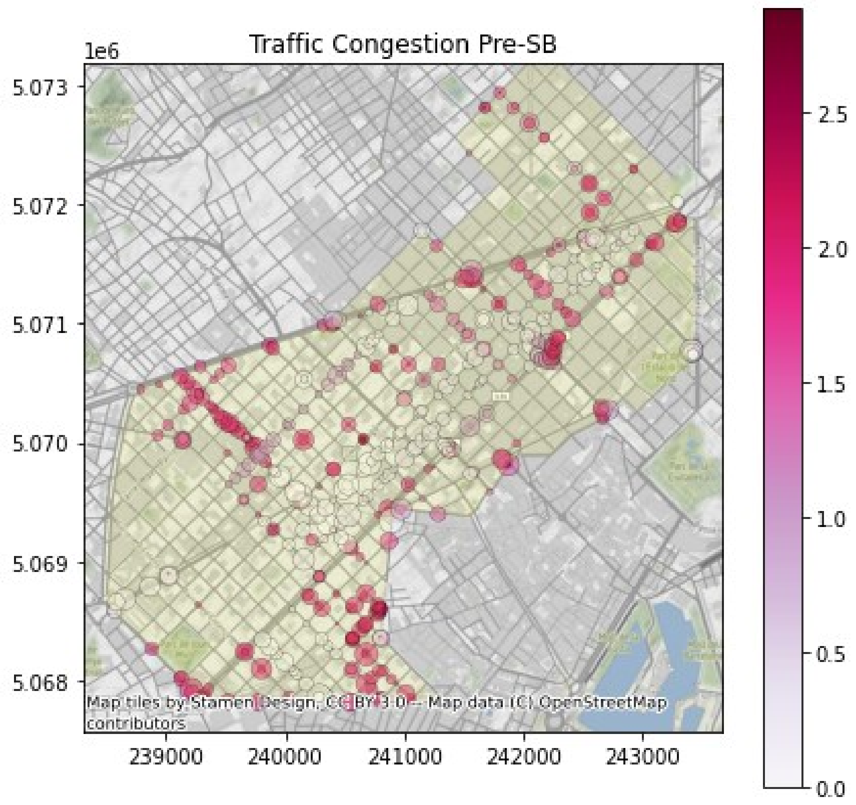
In this section, we will try to map the traffic congestion and clusters the traffic point based on their street parameters. Through this step, we will be able to see the congestion difference in pre and post superblock, as well as change of cluster that the superblock policy created.

For the first cell, let's map the traffic congestion. To make the visual scale more prominent, we will map the dots with marker size. In this way, the different value will not only have different color but also have a different point size.

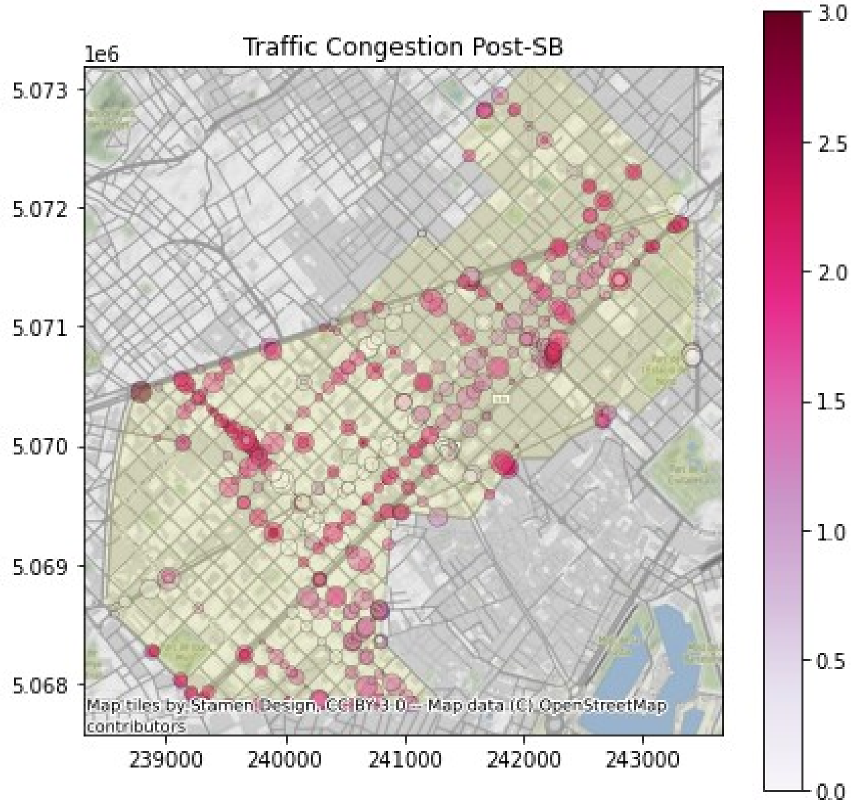
- In:

```
#Congestion map pre-superblcok
df1_wm['congestion'] = np.abs(np.random.randn(len(df1_wm))) * 50
ax = df1_wm.plot(markersize = df1_wm['congestion'],
                 figsize = (7, 7),
                 cmap = 'PuRd',
                 alpha = 0.5,
                 column = df1['congestion'],
                 linewidth = 0.4,
                 legend = True,
                 edgecolor = 'black')

cx.add_basemap(ax)
boundary_wm.plot(ax = ax, alpha = 0.1, color = 'yellow')
plt.title("Traffic Congestion Pre-SB")
plt.show()

#Congestion map post-superblcok
df2_wm['congestion'] = np.abs(np.random.randn(len(df2_wm))) * 50
ax = df2_wm.plot(markersize = df2_wm['congestion'],
                 figsize = (7, 7),
                 cmap = 'PuRd',
                 alpha = 0.5,
                 column = df2['congestion'],
                 linewidth = 0.4,
                 legend = True,
                 edgecolor = 'black')

cx.add_basemap(ax)
boundary_wm.plot(ax = ax, alpha = 0.1, color = 'yellow')
plt.title("Traffic Congestion Post-SB")
plt.show()
```

Out:



Traffic Congestion Pre-SB
1e6
5.073
5.072
5.071
5.070
5.069
5.068
239000
240000
241000
242000
243000
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors
0.0
0.5
1.0
1.5
2.0
2.5



Traffic Congestion Post-SB
1e6
5.073
5.072
5.071
5.070
5.069
5.068
239000
240000
241000
242000
243000
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors
0.0
0.5
1.0
1.5
2.0
2.5
3.0

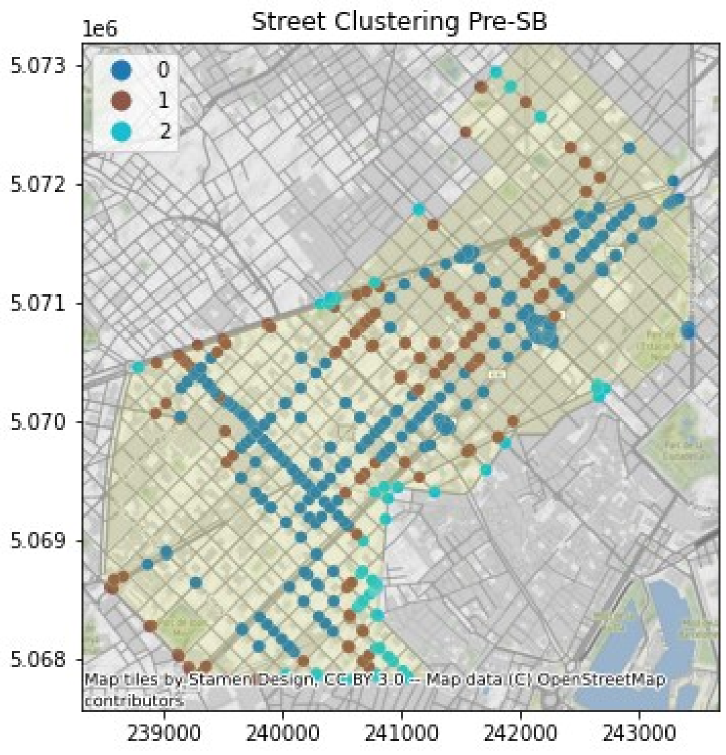
Visually, we can observe that there is an overall increase in traffic congestion after the superblock policy is being implemented. Now we also going to see if there is a change in street cluster post superblock.

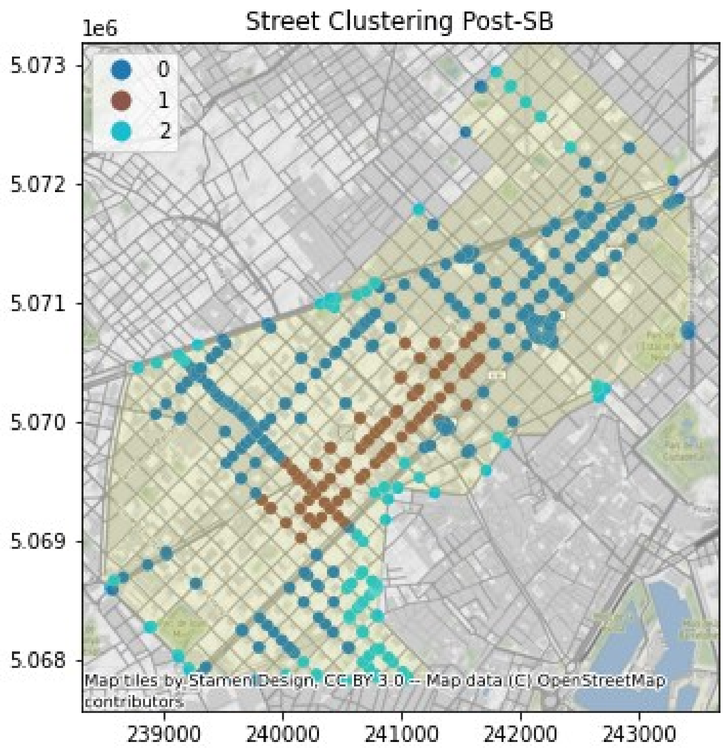
The clustering that we are going to do is based on K-means clustering method. Initially, it will fed the method with various street network parameters and it will determines the cluster distribution from our data. In this task, we will try to cluster the data into three groups, hence we set n_clusters =3.

- In:

```
#Clustering the street parameters pre-superblock

#Set up the number of cluster
km4pre = cluster.KMeans(n_clusters = 3)

#Inputing the street network parameters
km4pre_cls = km4pre.fit(df1_wm[['NR',
                                'CR',
                                'TR',
                                'XR',
                                'stan_LS',
                                'stan_NV',
                                'stan_BR',
                                'stan_HR']].values)

#Mapping the clusters
f, ax = plt.subplots(1, figsize = (6, 6))

df1_wm.assign(cl = km4pre_cls.labels_).plot(column = 'cl',
                                     categorical = True,
                                     legend = True,
                                     linewidth = 0.1,
                                     cmap = 'tab10',
                                     edgecolor = 'white',
                                     ax = ax)

cx.add_basemap(ax)
boundary_wm.plot(ax = ax, alpha = 0.1, color = 'yellow')
plt.title("Street Clustering Pre-SB")
plt.show()

#Clustering the street parameters post-superblock

#Set up the number of cluster
km4pre = cluster.KMeans(n_clusters = 3)

#Inputing the street network parameters
km4pre_cls = km4pre.fit(df2_wm[['NR',
                                'CR',
                                'TR',
                                'XR',
                                'stan_LS',
                                'stan_NV',
                                          'stan_BR',
                                          'stan_HR']].values)

#Mapping the clusters
f, ax = plt.subplots(1, figsize = (6, 6))

df2_wm.assign(cl = km4pre_cls.labels_).plot(column = 'cl',
                                            categorical = True,
                                            legend = True,
                                            linewidth = 0.1,
                                            cmap = 'tab10',
                                            edgecolor = 'white',
                                            ax = ax)

cx.add_basemap(ax)
boundary_wm.plot(ax = ax, alpha = 0.1, color = 'yellow')
plt.title("Street Clustering Post-SB")
plt.show()
```

Out:




Street Clustering Post-SB
1e6
● 0
● 1
● 2
5.073
5.072
5.071
5.070
5.069
5.068
239000
240000
241000
242000
243000
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors

Step 3: Machine Learning Method Selection

In the following sections, we will try to predict the traffic congestion level (dependent variable) by taking into account the street parameters (independent variables) through machine learning modeling.

However, since there are various machine learning methods that we can use for this case, we will try to select the best option between Linear Regression (LR), Decision Tree (DT), and Random Forest (RF). The selection will be based on the r-square and NRMSE value comparison.

To enrich the training dataset, we will combine and concatenate the pre-superblock dataset and post-superblock dataset.

- In:

```
#Concatenating pre-superblock dataset and post-superblock dataset into
one dataset
frames = [df1, df2]
ML_dataset = pd.concat(frames)

ML_dataset.head(3)
```

Out:

	street_nam	long	lat	idTram	l_section	n_vertices	blocked_r	half_r	cluster	congestion	NR	CR	TR	XR	stan_LS	stan_NV	stan_BR	stan_HR	geometry
0	Diagonal (Pl. Francesc Macià a Balmes)	2.145089	41.392758	103	5.469444	35.0	0.0	0.0	0.0	1.882353	1.000000	0.000000	0.457143	0.542857	0.643128	0.555556	0.0	0.0	POINT (2.14509 41.39276)
1	Diagonal (Pl. Francesc Macià a Balmes)	2.154834	41.395224	103	4.950746	28.0	0.0	0.0	0.0	1.882353	0.928571	0.071429	0.269231	0.730769	0.482241	0.400000	0.0	0.0	POINT (2.15483 41.39522)
2	Diagonal (Balmes a Pl Joan Carles I)	2.154850	41.395227	105	4.974501	28.0	0.0	0.0	0.0	2.000000	0.928571	0.071429	0.269231	0.730769	0.489609	0.400000	0.0	0.0	POINT (2.15485 41.39523)

Assuming that the changes of street parameters will affect the traffic congestion, we will set the street parameters as the independent variables of the model, and set the traffic congestion as the dependent variable of the model.

- In:

  ```
  #Defining independent variables (input): all street parameters and
  sensor locations that the dataset contains.
  columns_x = ['long',
               'lat',
               'NR',
               'CR',
               'TR',
               'XR',
               'stan_LS',
               'stan_NV',
               'stan_BR',
               'stan_HR']

  x = ML_dataset[columns_x]
  x.head(3)
  ```

  Out:

	long	lat	NR	CR	TR	XR	stan_LS	stan_NV	stan_BR	stan_HR
0	2.145089	41.392758	1.000000	0.000000	0.457143	0.542857	0.643128	0.555556	0.0	0.0
1	2.154834	41.395224	0.928571	0.071429	0.269231	0.730769	0.482241	0.400000	0.0	0.0
2	2.154850	41.395227	0.928571	0.071429	0.269231	0.730769	0.489609	0.400000	0.0	0.0

- In:

  ```
  #Defining dependent variables (output): the traffic congestion level.
  y = ML_dataset['congestion']
  y
  ```

  Out:

  ```
  0      1.882353
  1      1.882353
  2      2.000000
  3      2.000000
  4      1.764706
           ...
  600    1.833333
  601    1.833333
  602    1.833333
  603    1.833333
  604    1.000000
  Name: congestion, Length: 1210, dtype: float64
  ```

After the independent variables and dependent variable are defined, now we are going to set the train-test split of the data. As the rule of thumb, we will split the data into 70% training and 30% testing, hence we set test_size = 0.3.

- In:

```
# Initializing train and test dataset portion (test dataset 30% from
the whole dataset)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3,
random_state = 42)
```

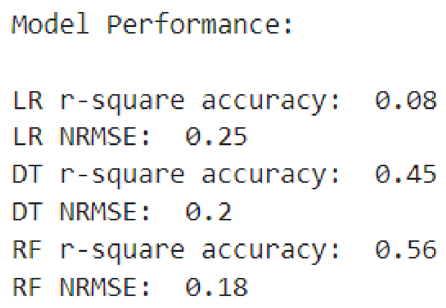
All set, we proceed to start machine learning modeling. In this part, we will try to select the best modeling method between linear regression (LR), decision tree (DT), and random forest (RF). To evaluate the performance, we will calculate the r-square accuracy score and NRMSE score for each model method.

- In:

```
#CONDUCTING MACHINE LEARNING METHODS ACCURACY TEST
#SELECTION BETWEEN LINEAR REGRESSION, DECISION TREE, AND RANDOM FOREST.

print("Model Performance:\n")

#Linear regression (setting)
lr = linear_regression = LinearRegression()

#Linear regression (accuracy test)
lr.fit(xtrain, ytrain)
ypred1 = lr.predict(xtest)
r2 = round(r2_score(ytest, ypred1), 2)
mse = mean_squared_error(ytest, ypred1)
rmse = round((math.sqrt(mse)), 2)
nrmse = round(rmse / (max(ytest)-min(ytest)), 2)

print("LR r-square accuracy: ", r2)
print("LR NRMSE: ", nrmse)

#Decision tree (setting)
dt = DecisionTreeRegressor()

#Decision tree (accuracy test)
dt.fit(xtrain, ytrain)
ypred2 = dt.predict(xtest)
r2 = round(r2_score(ytest, ypred2), 2)
mse = mean_squared_error(ytest, ypred2)
rmse = round((math.sqrt(mse)), 2)
nrmse = round(rmse / (max(ytest)-min(ytest)), 2)

print("DT r-square accuracy: ", r2)
print("DT NRMSE: ", nrmse)
```

```
#Random forest (setting)
rf = RandomForestRegressor()

#Random forest (accuracy test)
rf.fit(xtrain, ytrain)
ypred3 = rf.predict(xtest)
r2 = round(r2_score(ytest, ypred3), 2)
mse = mean_squared_error(ytest, ypred3)
rmse = round((math.sqrt(mse)), 2)
nrmse = round(rmse / (max(ytest)-min(ytest)), 2)

print("RF r-square accuracy: ", r2)
print("RF NRMSE: ", nrmse)
```

Out:



```
Model Performance:

LR r-square accuracy:  0.08
LR NRMSE:  0.25
DT r-square accuracy:  0.45
DT NRMSE:  0.2
RF r-square accuracy:  0.56
RF NRMSE:  0.18
```

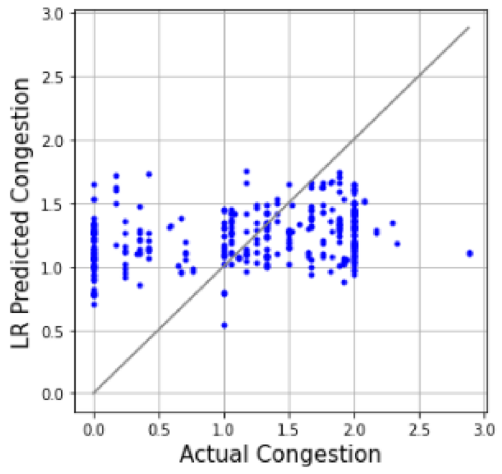
Now we will try to evaluate each method by observing the scatter plot of ‘method predicted value’ against ‘actual value’.

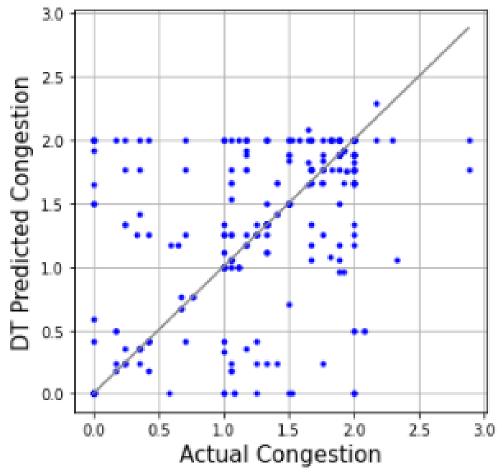
- In:

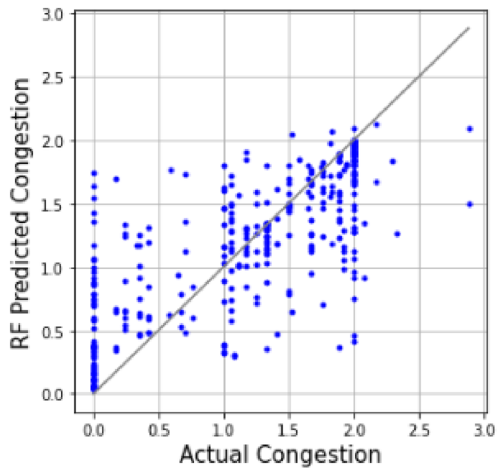
```
#VISUALING MODEL PERFORMANCE FROM DIFFERENT METHODS

#Plotting the linear regression model performance
fig, ax = plt.subplots(figsize = (5,5))
ax.grid()
ax.plot(ytest, ypred1, '.b')
ax.plot([0, max(ytest)], [0, max(ytest)], color = 'grey')
ax.set_ylabel('LR Predicted Congestion', fontsize = 15)
ax.set_xlabel('Actual Congestion', fontsize = 15)

#Plotting the decision tree model performance
fig, ax = plt.subplots(figsize = (5,5))
ax.grid()
ax.plot(ytest, ypred2, '.b')
ax.plot([0, max(ytest)], [0, max(ytest)], color = 'grey')
ax.set_ylabel('DT Predicted Congestion', fontsize = 15)
ax.set_xlabel('Actual Congestion', fontsize = 15)
```

```
#Plotting the random forest model performance
fig, ax = plt.subplots(figsize = (5,5))
ax.grid()
ax.plot(ytest, ypred3, '.b')
ax.plot([0, max(ytest)], [0, max(ytest)], color = 'grey')
ax.set_ylabel('RF Predicted Congestion', fontsize = 15)
ax.set_xlabel('Actual Congestion', fontsize = 15)
```

Out:







Step 4: Feature Importance Analysis

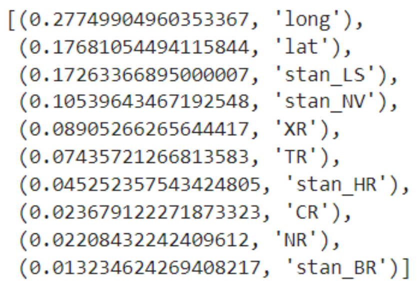
Referring to the r-square accuracy test, NRMSE score, and the data plotting, we found that the random forest method outperforms the other methods, and thus we will go with the random forest as our modeling method.

In this stage, we will try to see which independent variables (or in this case 'street parameters' variable) that have the highest effect on traffic congestion level.

- In:

  ```
  #Checking feature importance of each street parameters
  feature_importance = list (zip (rf.feature_importances_, columns_x))
  feature_importance.sort(reverse = True)
  feature_importance
  ```

  Out:



```
[(0.27749904960353367, 'long'),
 (0.17681054494115844, 'lat'),
 (0.17263366895000007, 'stan_LS'),
 (0.10539643467192548, 'stan_NV'),
 (0.08905266265644417, 'XR'),
 (0.07435721266813583, 'TR'),
 (0.045252357543424805, 'stan_HR'),
 (0.023679122271873323, 'CR'),
 (0.02208432242409612, 'NR'),
 (0.013234624269408217, 'stan_BR')]
```

Step 5: Building Machine Learning Predictor

Now that we have known which method to use and which variables have the highest importance to the model, we can set our machine learning model to predict the traffic congestion level based on street parameters that we insert individually.

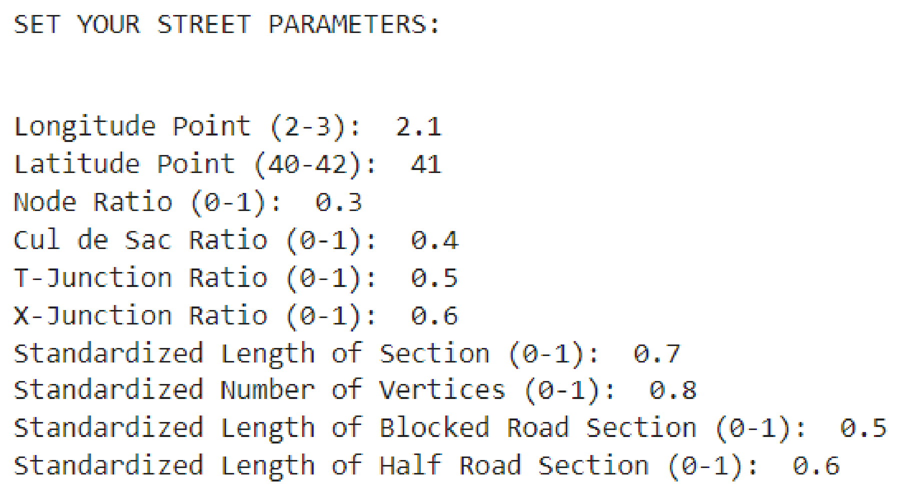
- In:

  ```
  print('SET YOUR STREET PARAMETERS:\n')

  #Creating feedback query
  long = float(input('Longitude Point (2-3): '))
  lat = float(input('Latitude Point (40-42): '))
  NR = float(input('Node Ratio (0-1): '))
  CR = float(input('Cul de Sac Ratio (0-1): '))
  TR = float(input('T-Junction Ratio (0-1): '))
  XR = float(input('X-Junction Ratio (0-1): '))
  stan_LS = float(input('Standardized Length of Section (0-1): '))
  stan_NV = float(input('Standardized Number of Vertices (0-1): '))
  stan_BR = float(input('Standardized Length of Blocked Road Section (0-
  1): '))
  stan_HR = float(input('Standardized Length of Half Road Section (0-1):
  '))
  ```

```
print('\n')
predictions = rf.predict([[long, lat, NR, CR, TR, XR, stan_LS, stan_NV,
stan_BR, stan_HR]])

print('THE ESTIMATED LEVEL OF CONGESTION IN YOUR LOCATION IS: ' +
str(*predictions))
print('\n')
```

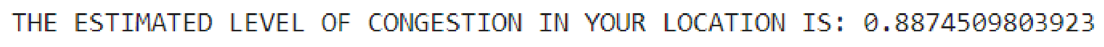
Program Feedback:

**Insert your own value for each street parameters*



```
SET YOUR STREET PARAMETERS:


Longitude Point (2-3):  2.1
Latitude Point (40-42):  41
Node Ratio (0-1):  0.3
Cul de Sac Ratio (0-1):  0.4
T-Junction Ratio (0-1):  0.5
X-Junction Ratio (0-1):  0.6
Standardized Length of Section (0-1):  0.7
Standardized Number of Vertices (0-1):  0.8
Standardized Length of Blocked Road Section (0-1):  0.5
Standardized Length of Half Road Section (0-1):  0.6
```

Out:



```
THE ESTIMATED LEVEL OF CONGESTION IN YOUR LOCATION IS: 0.8874509803923
```