Code Challenge

March 3, 2020

1 Code Challenge (Data Analyst)

Required libraries to solve the tasks

[1]:

```
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import sin, cos, sqrt, atan2, radians
from sklearn import preprocessing
from datetime import datetime
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from operator import itemgetter
```

[2]:

```
station_json_path="/home/noor/Desktop/Task/stations.json"
crash_data_path='/home/noor/Downloads/Motor_Vehicle_Collisions_-_Crashes.csv'
```

Read the crashes dataset

[3]:

```
df = pd.read_csv(crash_data_path)
```

```
/home/noor/anaconda2/envs/new_py36/lib/python3.6/site-
packages/IPython/core/interactiveshell.py:3058: DtypeWarning: Columns (3) have
mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

2 Task 1

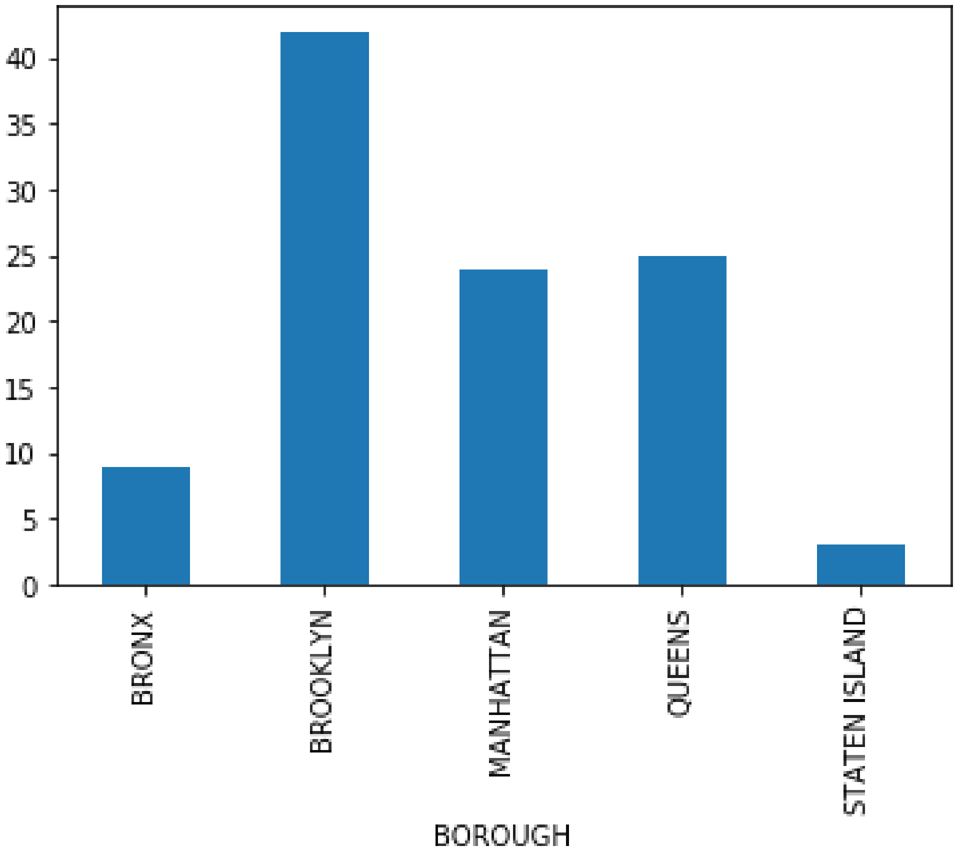
In this task we need to find what is the most dangerous Borough for the Cyclists. There are many concepts that could be taken into consideration to find if a borough is dangerous for cyclists or not. One of these concepts is to see how many cyclists were killed or injured in the crushes that happened in this borough. I created a function that can visualize the borough according to another column in the dataset. Then I called this fucntion to visualize how many cyclists were killed in the crushes that happened in the borough then another call for how many cyclists were injured in the crushes that happened in the borough.

```
[4]: def cyclist_crashes_visualization(df, place, measure,subplot=False,plot=True):
        '''
       Finds the most dangerous borough/place in the dataset according to a specific␣
     ↪measure (injured cyclist, killed cyclist ...) and visulaize the boroughes
        by grouping the data in the dataframe using the bourgh/place and the sum of␣
     ↪the measure values.

        Parameters
        ----------
            df : dataFrame
                Represents the crashes data.

            place : a string or a list of strings
                Represents the place/borough that we need to visualize the data␣
     ↪based on

            measure : String
                Represents the columen name that we need to visulaize the place␣
     ↪according to.

            subplot : boolean
                To visulalize the data in subplots or in one plot

            plot : boolean
                To visualize the data or not
        Return
        ----------
            max_place : list of tupels of string
              Represents the index of the maximum sum of values in the measure column
            max_value : integer
                Represents the maximum sum of values in the measure column
        '''

        # remove nan values
        df.dropna()

        #Remove the lines that has 0 value in the measure column from the dataFrame
        new_df = df[df[measure] != 0]

        #Group the data using place and the sum of the values in the measure column
        place_measure_groups = new_df.groupby(place)[measure].agg('sum')

        #find the place that has the hightest sum of values in the measure column

        max_value = place_measure_groups.max()
        max_place = [key for key in place_measure_groups.keys() if max_value==␣
     ↪place_measure_groups[key]]
```

```
    #plot
    if plot:
        if not subplot:
            place_measure_groups.plot(kind = 'bar')
        if subplot:
            groups = new_df.groupby(place[0])
            for key in groups.groups.keys():
                plt.figure(figsize = (25,6))
                g = groups.get_group(key).groupby(place[1:len(place)])[measure].
 ↪agg('sum')
                plt.bar([str(k) for k in g.keys()],g.values)
                plt.xticks(rotation = 90)
                plt.title(key)
                plt.show()


    return max_place,max_value
```

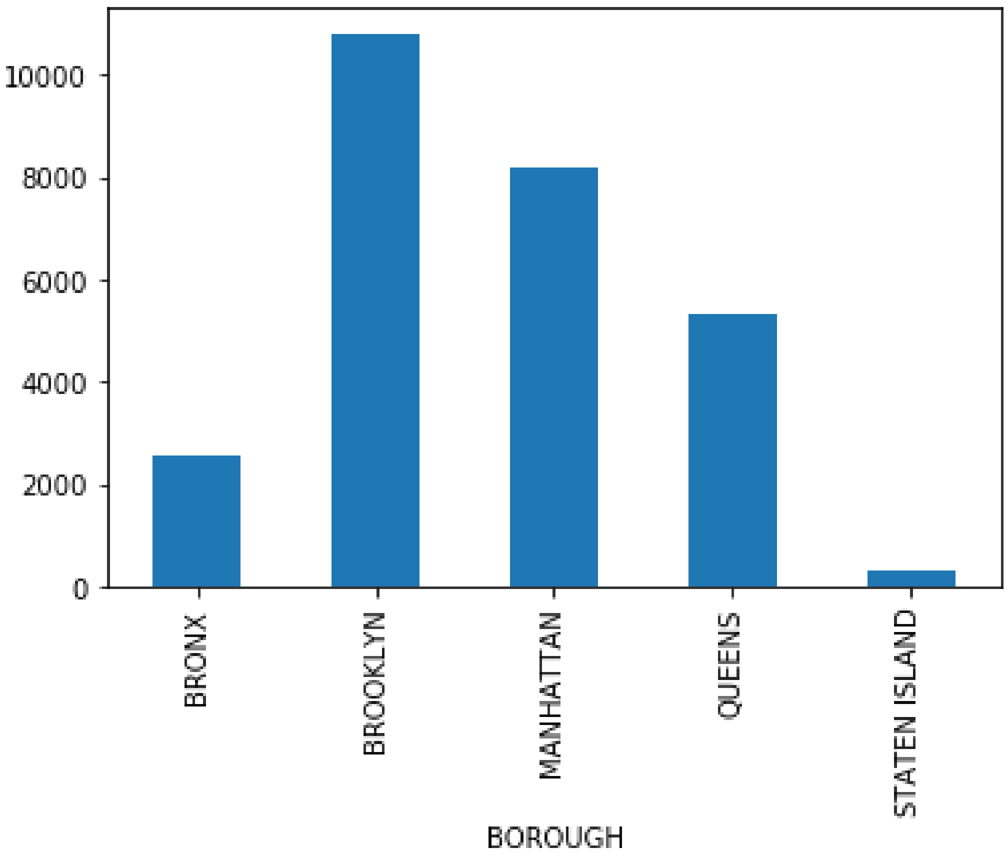
[5]:
```
max_borough,max_value = cyclist_crashes_visualization(df,'BOROUGH', 'NUMBER OF␣
 ↪CYCLIST KILLED')
print('According to sum the number of cyclist killed in the crashes, the Borough␣
 ↪%s is the most dangerous borough, It has the highest number of cyclist killed␣
 ↪in the crashes: %s'%(' and '.join(max_borough), max_value))
```

```
According to sum the number of cyclist killed in the crashes, the Borough
BROOKLYN is the most dangerous borough, It has the highest number of cyclist
killed in the crashes: 42
```



```
[6]: max_borough,max_value = cyclist_crashes_visualization(df,'BOROUGH', 'NUMBER OF␣
      ↪CYCLIST INJURED')
     print('According to sum the number of cyclist injured in the crashes, the␣
      ↪Borough %s is the most dangerous borough, It has the highest number of␣
      ↪cyclist injured in the crashes: %s'%(' and '.join(max_borough), max_value))
```

```
According to sum the number of cyclist injured in the crashes, the Borough
BROOKLYN is the most dangerous borough, It has the highest number of cyclist
injured in the crashes: 10799
```



To find the worst place to have a citybike station in, we need to define what do we mean of worst and how can we define a place.

3 Task 2

First, we can define a place using the zip code, the borough name, the on street name, the cross street name, a coorinate of a point in the place, and so on ...

If we took a coorinate of a point in the place then we are just thinking of one point of the place and when we want to think about a place to build in a station we would think about the area first is it suitable or not. In this case we can take into consideration the zip code, the borough name and the on street name/the cross street name.

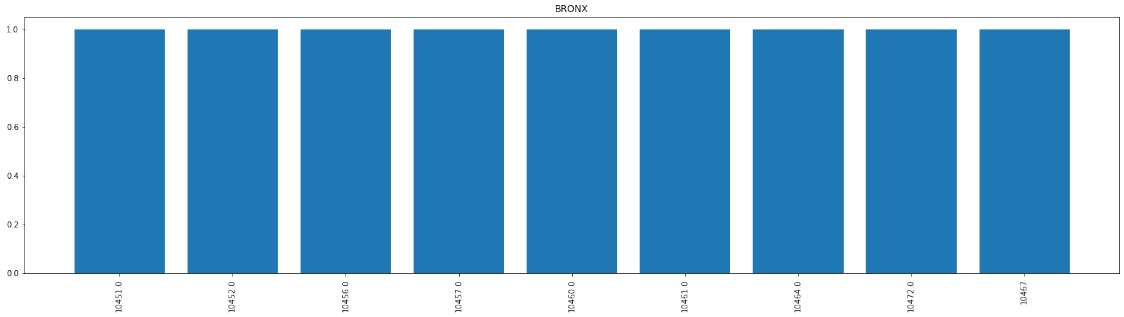
The worst place to build in a citybike station in could be defined as a dangerous place for the cyclists. In other words, it would be similar definiation as in task 1: the place where a cyclist got killed or injured in.

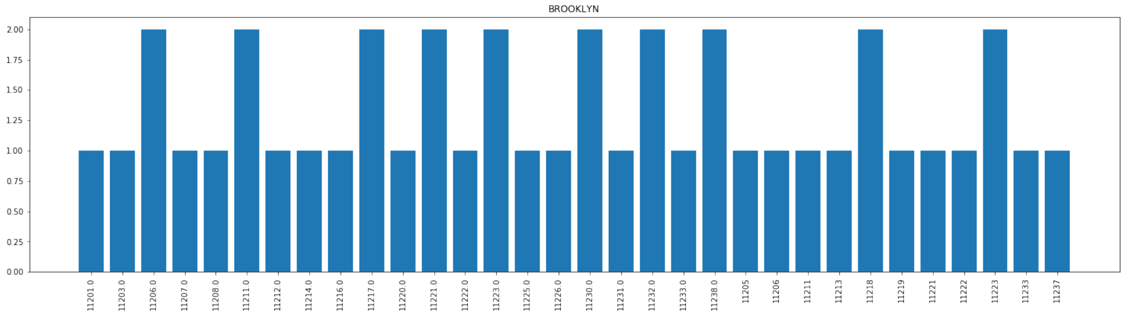
From the first task, we found that Borough BROOKLYN is the most dangerous borough for the cyclists, but it might not have the most dangerous place. we can find this out by visualizing the number of killed/injured cyclists using the borough an the zipcode.

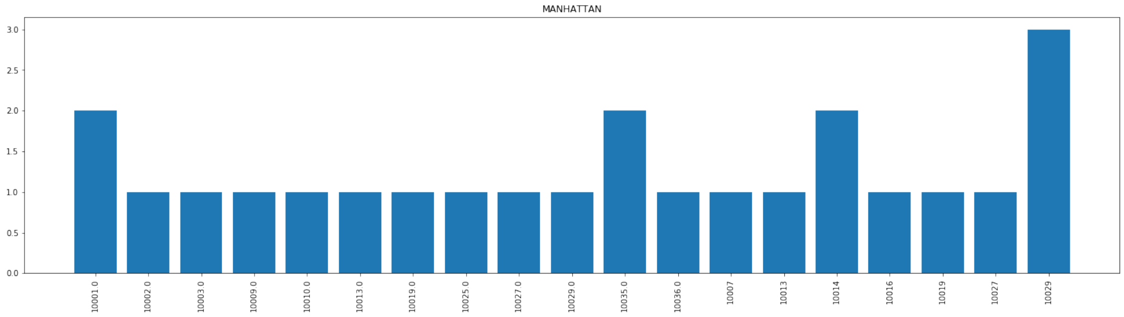
We will start with the number of killed cyclists then we will have a look at the number of injured cyclists

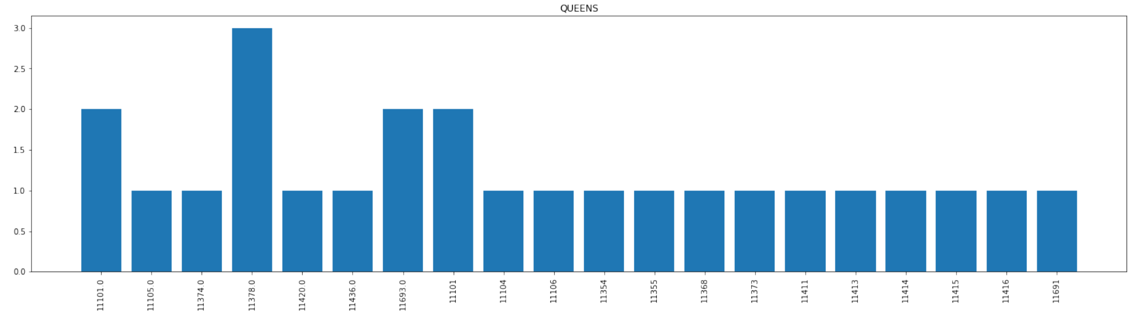
we will use the same function written in task1

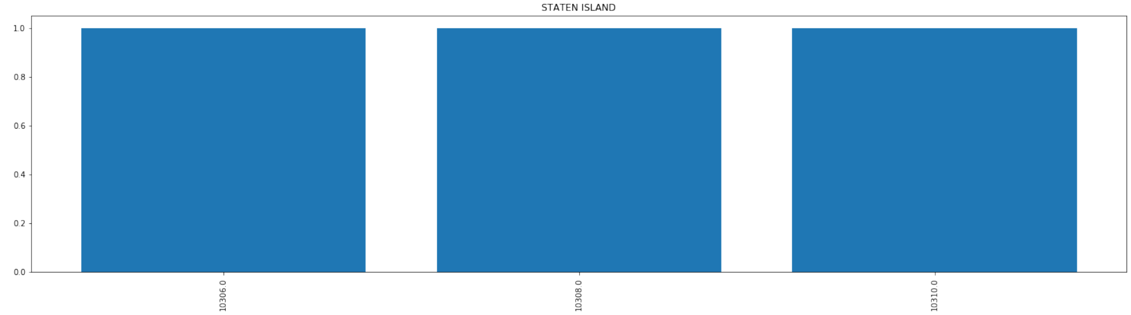
```
[7]: max_place_killed,max_value_killed = cyclist_crashes_visualization(df,['BOROUGH','ZIP CODE'], 'NUMBER OF CYCLIST KILLED',subplot=True)
max_place_killed_print = ['(%s, %s)'%(zipCode,borough) for i ,(borough,zipCode) in enumerate(max_place_killed)]
print('According to sum the number of cyclist killed in the crashes, %s are the most dangerous areas to build a bike station in, they have the highest number of cyclist Killed in the crashes: %s'%(' and '.join(max_place_killed_print), max_value_killed))
```





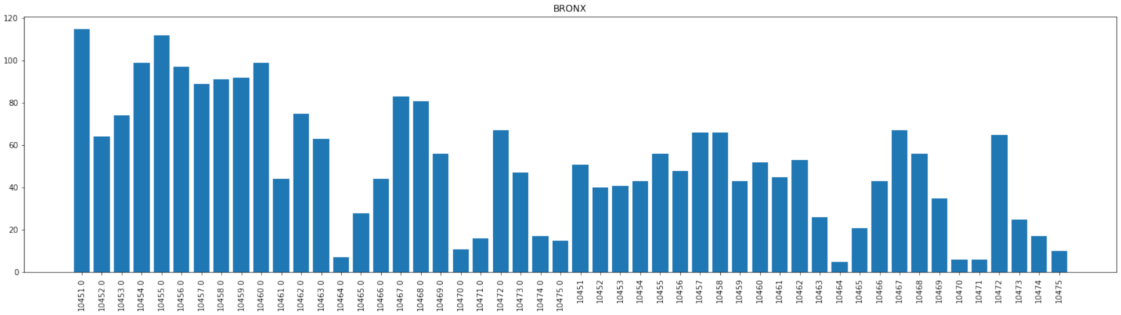


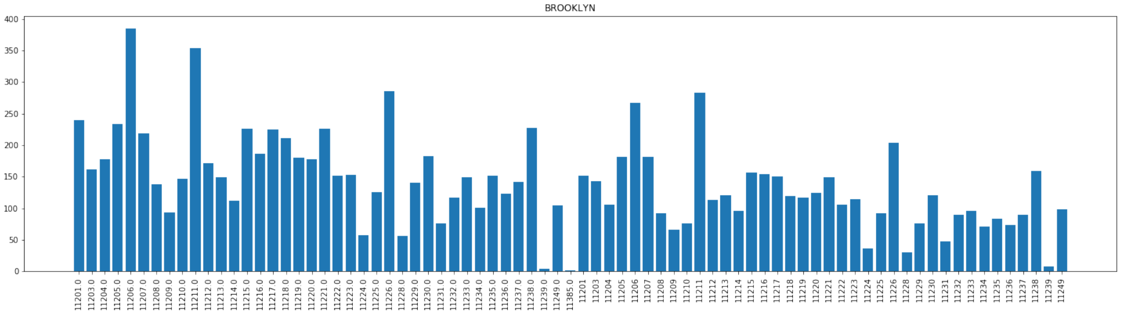


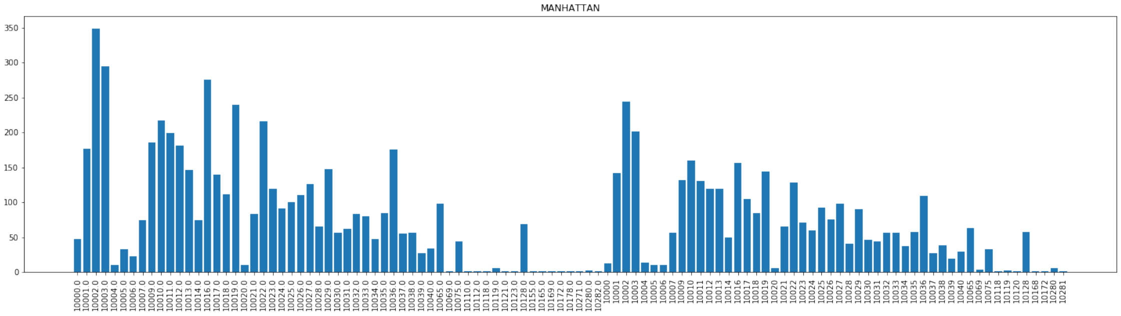


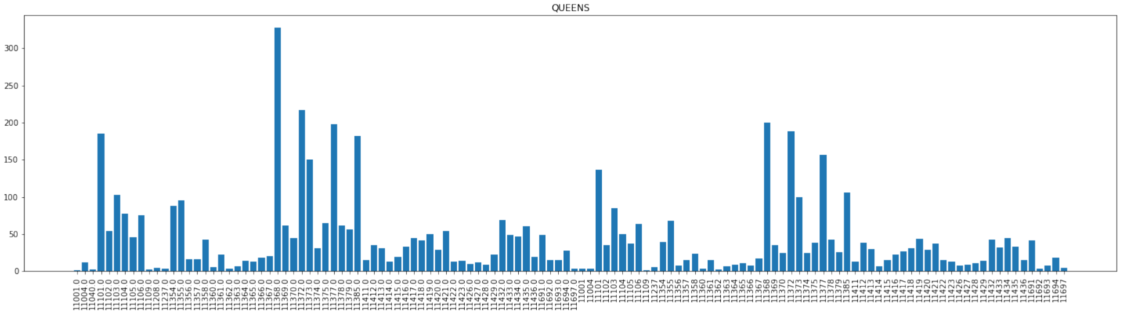
According to sum the number of cyclist killed in the crashes, (10029, MANHATTAN) and (11378.0, QUEENS) are the most dangerous areas to build a bike station in, they have the highest number of cyclist Killed in the crashes: 3

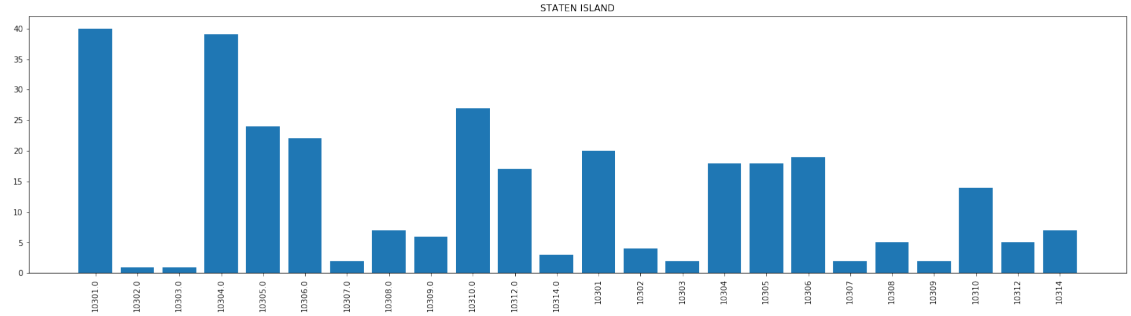
```
[8]: max_place_injured,max_value_injured =␣
      ↪cyclist_crashes_visualization(df,['BOROUGH','ZIP CODE'], 'NUMBER OF CYCLIST␣
      ↪INJURED',subplot=True)
     max_place_injured_print = ['(%s, %s)'%(zipCode,borough) for i␣
      ↪,(borough,zipCode) in enumerate(max_place_injured)]
     print('According to sum the number of cyclist injured in the crashes, %s is the␣
      ↪most dangerous area to build a bike station in, it has the highest number of␣
      ↪cyclist injured in the crashes: %s'%(' and '.join(max_place_injured_print),␣
      ↪max_value_injured))
```


BRONX
0
20
40
60
80
100
120
10451.0
10452.0
10453.0
10454.0
10455.0
10456.0
10457.0
10458.0
10459.0
10460.0
10461.0
10462.0
10463.0
10464.0
10465.0
10466.0
10467.0
10468.0
10469.0
10470.0
10471.0
10472.0
10473.0
10474.0
10475.0
10451
10452
10453
10454
10455
10456
10457
10458
10459
10460
10461
10462
10463
10464
10465
10466
10467
10468
10469
10470
10471
10472
10473
10474
10475


BROOKLYN
0
50
100
150
200
250
300
350
400
11201.0
11203.0
11204.0
11205.0
11206.0
11207.0
11208.0
11209.0
11210.0
11211.0
11212.0
11213.0
11214.0
11215.0
11216.0
11217.0
11218.0
11219.0
11220.0
11221.0
11222.0
11223.0
11224.0
11225.0
11226.0
11228.0
11229.0
11230.0
11231.0
11232.0
11233.0
11234.0
11235.0
11236.0
11237.0
11238.0
11239.0
11249.0
11385.0
11201
11203
11204
11205
11206
11207
11208
11209
11210
11211
11212
11213
11214
11215
11216
11217
11218
11219
11220
11221
11222
11223
11224
11225
11226
11228
11229
11230
11231
11232
11233
11234
11235
11236
11237
11238
11239
11249


MANHATTAN
0
50
100
150
200
250
300
350
10000.0
10001.0
10002.0
10003.0
10004.0
10005.0
10006.0
10007.0
10009.0
10010.0
10011.0
10012.0
10013.0
10014.0
10016.0
10017.0
10018.0
10019.0
10020.0
10021.0
10022.0
10023.0
10024.0
10025.0
10026.0
10027.0
10028.0
10029.0
10030.0
10031.0
10032.0
10033.0
10034.0
10035.0
10036.0
10037.0
10038.0
10039.0
10040.0
10065.0
10069.0
10075.0
10110.0
10112.0
10118.0
10119.0
10121.0
10123.0
10128.0
10155.0
10165.0
10169.0
10172.0
10178.0
10271.0
10280.0
10282.0
10000
10001
10002
10003
10004
10005
10006
10007
10009
10010
10011
10012
10013
10014
10016
10017
10018
10019
10020
10021
10022
10023
10024
10025
10026
10027
10028
10029
10030
10031
10032
10033
10034
10035
10036
10037
10038
10039
10040
10065
10069
10075
10118
10119
10120
10128
10168
10172
10280
10281


QUEENS
0
50
100
150
200
250
300
11001.0
11004.0
11040.0
11101.0
11102.0
11103.0
11104.0
11105.0
11106.0
11109.0
11208.0
11237.0
11354.0
11355.0
11356.0
11357.0
11358.0
11360.0
11361.0
11362.0
11363.0
11364.0
11365.0
11366.0
11367.0
11368.0
11369.0
11370.0
11372.0
11373.0
11374.0
11375.0
11377.0
11378.0
11379.0
11385.0
11411.0
11412.0
11413.0
11414.0
11415.0
11416.0
11417.0
11418.0
11419.0
11420.0
11421.0
11422.0
11423.0
11426.0
11427.0
11428.0
11429.0
11432.0
11433.0
11434.0
11435.0
11436.0
11691.0
11692.0
11693.0
11694.0
11697.0
11001
11004
11101
11102
11103
11104
11105
11106
11109
11237
11354
11355
11356
11357
11358
11360
11361
11362
11363
11364
11365
11366
11367
11368
11369
11370
11372
11373
11374
11375
11377
11378
11379
11385
11411
11412
11413
11414
11415
11416
11417
11418
11419
11420
11421
11422
11423
11426
11427
11428
11429
11432
11433
11434
11435
11436
11691
11692
11693
11694
11697



```
According to sum the number of cyclist injured in the crashes, (11206.0,
BROOKLYN) is the most dangerous area to build a bike station in, it has the
highest number of cyclist injured in the crashes: 385
```

we can see that we got three area according to the borough and the zip code, so we need to find which one is more highly dangerous areas

we can see how many cyclist where injured in the areas that have the highest number of killed cyclist and compare it to the area that has the highest number of injured cyclists

```
[9]: injured_area = df[df['NUMBER OF CYCLIST INJURED'] != 0].groupby(['BOROUGH','ZIP␣
      ↪CODE'])['NUMBER OF CYCLIST INJURED'].agg('sum')
     for i in range(len(max_place_killed)):
         print("The number of injured cyclist in",max_place_killed_print[i],␣
      ↪"is",injured_area[max_place_killed[i]])
```

```
The number of injured cyclist in (10029, MANHATTAN) is 90
The number of injured cyclist in (11378.0, QUEENS) is 62
```

we can also see how many cyclist where killed in the areas that have the highest number of injured cyclist and compare it to the areas that have the highest number of killed cyclists

```
[10]: killed_area = df[df['NUMBER OF CYCLIST KILLED'] != 0].groupby(['BOROUGH','ZIP␣
       ↪CODE'])['NUMBER OF CYCLIST KILLED'].agg('sum')
      for i in range(len(max_place_injured)):
          print("The number of killed cyclist␣
       ↪in",max_place_injured_print[i],"is",killed_area[max_place_injured[i]])
```

```
The number of killed cyclist in (11206.0, BROOKLYN) is 2
```

According to both results, we can see that BROOKLYN is the most dangerous borough and (11206.0, BROOKLYN) is more dangerous area than the others.

To find the highly dangerous street, we can have a look at BROOKLYN crashes.

```
[11]: max_place_killed_on,max_value_killed_on = cyclist_crashes_visualization(df,['ON␣
       ↪STREET NAME','ZIP CODE','BOROUGH'], 'NUMBER OF CYCLIST KILLED',False,False)
```

```
max_place_killed_on_print = ['(%s, %s, %s)'%(" ".join(on_st.
 ↪split()),str(zipCode),borough) for i ,(on_st,zipCode,borough) in␣
 ↪enumerate(max_place_killed_on)]
print('According to sum the number of cyclist killed in the crashes, %s are the␣
 ↪most dangerous streets to build a bike station in, where the highest number␣
 ↪of cyclist killed in the crashes: %s'%(' and '.
 ↪join(max_place_killed_on_print), max_value_killed_on))
```

```
According to sum the number of cyclist killed in the crashes, (3 AVENUE,
11232.0, BROOKLYN) and (56 ROAD, 11378.0, QUEENS) and (BORDEN AVENUE, 11101.0,
QUEENS) and (WEST STREET, 10014, MANHATTAN) are the most dangerous streets to
build a bike station in, where the highest number of cyclist killed in the
crashes: 2
```

[12]:

```
max_place_injured_on,max_value_injured_on =␣
 ↪cyclist_crashes_visualization(df,['ON STREET NAME','ZIP CODE','BOROUGH'],␣
 ↪'NUMBER OF CYCLIST INJURED',False,False)
max_place_injured_on_print = ['(%s, %s, %s)'%(" ".join(on_st.
 ↪split()),str(zipCode),borough) for i ,(on_st,zipCode,borough) in␣
 ↪enumerate(max_place_injured_on)]
print('According to sum the number of cyclist injured in the crashes, %s is the␣
 ↪most dangerous street to build a bike station in, where the highest number of␣
 ↪cyclist injured in the crashes: %s'%(' and '.
 ↪join(max_place_injured_on_print), max_value_injured_on))
```

```
According to sum the number of cyclist injured in the crashes, (GRAND STREET,
11211, BROOKLYN) is the most dangerous street to build a bike station in, where
the highest number of cyclist injured in the crashes: 60
```

[13]:

```
injured_street = df[df['NUMBER OF CYCLIST INJURED'] != 0].groupby(['ON STREET␣
 ↪NAME','ZIP CODE','BOROUGH'])['NUMBER OF CYCLIST INJURED'].agg('sum')
for i in range(len(max_place_killed_on)):
    if max_place_killed_on[i] in injured_street.keys():
        print("The number of injured cyclist in",max_place_killed_on_print[i],␣
 ↪"is",injured_street[max_place_killed_on[i]])
    else:
        print("No cyclist was injured in",max_place_killed_on_print[i])
```

```
The number of injured cyclist in (3 AVENUE, 11232.0, BROOKLYN) is 9
The number of injured cyclist in (56 ROAD, 11378.0, QUEENS) is 3
The number of injured cyclist in (BORDEN AVENUE, 11101.0, QUEENS) is 4
The number of injured cyclist in (WEST STREET, 10014, MANHATTAN) is 2
```

[14]:

```
killed_street = df[df['NUMBER OF CYCLIST KILLED'] != 0].groupby(['ON STREET␣
 ↪NAME','ZIP CODE','BOROUGH'])['NUMBER OF CYCLIST KILLED'].agg('sum')
for i in range(len(max_place_injured_on)):
    if max_place_injured_on[i] in killed_street.keys():
        print("The number of killed cyclist␣
↪in",max_place_injured_on_print[i],"is",killed_street[max_place_injured_on[i]])
    else:
        print("No cyclist was killed in the crashes␣
↪in",max_place_injured_on_print[i])
```

```
No cyclist was killed in the crashes in (GRAND STREET, 11211, BROOKLYN)
```

From the previous analysis we can see that:

1- there are 5 streets that could be considered as dangerous streets to build a citybike in while it differ in the level of dangerousness

2- There is one area that is cocedered as dangerous area (11206.0, BROOKLYN)

4 Task 3

The crashes dataset doesn't have any information about the stations, which means we need to read the stations dataset, we also need a function that can calculate the distacne between two coordinates, we need a function to preprocess to train a model.

```
[15]: def stations_coordinates(path):
    '''
    read the stations data and return a sorted list of stations according to the␣
↪coordinatwes.

    Prameters
    ----------
    path : string
        the path of the json file contains the stations data

    Return
    ----------
    stations_coordinates_sorted : list of lists
        a list of sorted stations according to the corrdinates
    '''

    # read the stations data
    with open(path, 'r') as fin:
        stations = json.load(fin)
        fin.close()


    stationBeanList = stations["stationBeanList"]
    stations_coordinates = []
    # take the needed information
    for station in stationBeanList:
        stations_coordinates.append([station['stationName'],␣
↪round(station['latitude'], 5),
                                     round(station['longitude'], 5)])

    # order the station according to the latitude then longitude then the␣
↪station name
    stations_coordinates_sorted = sorted(stations_coordinates,␣
↪key=itemgetter(2,0,1))

    return stations_coordinates_sorted
```

[16]:

```
def lonLatDistance(point1,point2):
    '''
    Function to calculate the distance between two coordinates

    Parameters
    ----------
    point1: (float, float)
        The first coordinate point
    point2: (float, float)
        The second coordinate point

    Return
    ----------
    Distance between the two points

    '''
    R = 6373.0 # approximate radius of earth in km
    lat1 = float(point1[0])
    lon1 = float(point1[1])
    lat2 = float(point2[0])
    lon2 = float(point2[1])

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

The dataset doesn't contain any safe places where no craches happened. Since our focus is on the cyclists in the city and the citybike stations, any crash that didn't kill or injured a cyclist will be considered as a safe area/street/place for the cyclists

To train the model, I will use crash time, borough, zip code, on street name, longitude, latitiude, contribution factor 1,2,3,4, and 5, vehical type 1,2,3,4, and 5, and the time .

```
[17]: def data_preprocessing(df):
          '''
          This function preprocess the data and prpare oit to train the prediction model

          Parameters
          ----------
          df: dataframe
              contains the data

          Return
          ----------
          featurs : array
              contains the features that are going top ne used as a n input to the␣
       ↪machine learning algorithm.
          labels : list of binary values
              represents the related output to the features in the feature array
          classes : dict
              dictionary that has the mapping values of the encoded values from the␣
       ↪column in the dataframe

          '''

          # remove the rows that contain nan values in the longitude and the latitude␣
       ↪columens
          df_copy = df[pd.notnull(df['LONGITUDE'])]
          df_copy = df_copy[pd.notnull(df_copy['LATITUDE'])]

          classes={}
          # to encode the data
          le = preprocessing.LabelEncoder()

          # encode the borough name, the on street name, the
          borough_encoded = le.fit_transform(list(df_copy['BOROUGH']))
          classes['BOROUGH']=dict(zip(le.transform(le.classes_),le.classes_))

          zipCode_encoded = le.fit_transform(list(df_copy['ZIP CODE']))
          classes['ZIP CODE']=dict(zip(le.transform(le.classes_),le.classes_))

          onStreetName_encoded = le.fit_transform(list(df_copy['ON STREET NAME']))
          classes['ON STREET NAME'] = dict(zip(le.transform(le.classes_),le.classes_))


          # prepare the lonitude and the latitude
          lon_list = list(df_copy['LONGITUDE'].round(5))
          lat_list = list(df_copy['LATITUDE'].round(5))

          # find the list of the contribution factors in 5 columns
```

```
    contribution_factors_list = list(set(list(df_copy['CONTRIBUTING FACTOR VEHICLE 1']) +
                                         list(df_copy['CONTRIBUTING FACTOR VEHICLE 2']) +
                                         list(df_copy['CONTRIBUTING FACTOR VEHICLE 3']) +
                                         list(df_copy['CONTRIBUTING FACTOR VEHICLE 4']) +
                                         list(df_copy['CONTRIBUTING FACTOR VEHICLE 5'])))

    # encode the contribution factors columen in the dataframe according the index of
    # the contribution factor in the contribution list
    Cont_1_encoded = [contribution_factors_list.index(i) for i in df_copy['CONTRIBUTING FACTOR VEHICLE 1']]
    Cont_2_encoded = [contribution_factors_list.index(i) for i in df_copy['CONTRIBUTING FACTOR VEHICLE 2']]
    Cont_3_encoded = [contribution_factors_list.index(i) for i in df_copy['CONTRIBUTING FACTOR VEHICLE 3']]
    Cont_4_encoded = [contribution_factors_list.index(i) for i in df_copy['CONTRIBUTING FACTOR VEHICLE 4']]
    Cont_5_encoded = [contribution_factors_list.index(i) for i in df_copy['CONTRIBUTING FACTOR VEHICLE 5']]

    classes['CONTRIBUTING FACTOR'] = {i:contribution_factors_list[i] for i in range(len(contribution_factors_list))}


    # find the list of the vehical type in 5 columns
    vehical_type_list = list(set(list(df_copy['VEHICLE TYPE CODE 1']) +
                                 list(df_copy['VEHICLE TYPE CODE 2']) +
                                 list(df_copy['VEHICLE TYPE CODE 3']) +
                                 list(df_copy['VEHICLE TYPE CODE 4']) +
                                 list(df_copy['VEHICLE TYPE CODE 5'])))

    # encode the vehical type columen in the dataframe according the index of
    # the vehical type in the vehical type list
    vehical_1_encoded = [vehical_type_list.index(i) for i in df_copy['VEHICLE TYPE CODE 1']]
    vehical_2_encoded = [vehical_type_list.index(i) for i in df_copy['VEHICLE TYPE CODE 2']]
    vehical_3_encoded = [vehical_type_list.index(i) for i in df_copy['VEHICLE TYPE CODE 3']]
    vehical_4_encoded = [vehical_type_list.index(i)for i in df_copy['VEHICLE TYPE CODE 4']]
    vehical_5_encoded = [vehical_type_list.index(i) for i in df_copy['VEHICLE␣
↪TYPE CODE 5']]


    classes['VEHICLE TYPE'] = {i:vehical_type_list[i] for i in␣
↪range(len(vehical_type_list))}

    # extract the minute from the time column
    minutes = [datetime.strptime(x, '%H:%M').time().minute for x in␣
↪df_copy['CRASH TIME']]

    # extract the hours from the time column
    hours = [datetime.strptime(x, '%H:%M').time().hour for x in df_copy['CRASH␣
↪TIME']]

    # combine the number of likked cyclits and the number of injured cyclist in␣
↪one column by summing them
    sum_killed_injured = df_copy['NUMBER OF CYCLIST KILLED']+df_copy['NUMBER OF␣
↪CYCLIST INJURED']

    # the features that are going to be used to train the model
    features = np.column_stack((borough_encoded, zipCode_encoded,␣
↪onStreetName_encoded,
                                lat_list,lon_list,
                               ␣
↪Cont_1_encoded,Cont_2_encoded,Cont_3_encoded,Cont_4_encoded,Cont_5_encoded,
                               ␣
↪vehical_1_encoded,vehical_2_encoded,vehical_3_encoded,vehical_4_encoded,vehical_5_encoded,
                                hours, minutes))

    # the labels that are going to be used to train the model
    label = [x if x == 0 else 1 for x in sum_killed_injured]

    return features, label,classes
```

Prepare the data and split it into 70% training and 30% testing

[18]:
```
# preprocess the data
features, label, classes= data_preprocessing(df)

# split the data into train and test data
X_train, X_test, y_train, y_test = train_test_split(features, label, test_size␣
↪= 0.3,random_state = 109) # 70% training and 30% test
```

Create and train a random forest classifier and generate the prediction on the test dataset

[19]:

```
#Create a Random Forest classifier
model = RandomForestClassifier(n_estimators = 100)

# Train the model using the training sets
model.fit(X_train,y_train)

# Generate the result of prediction from the moel on the test dataset
y_pred = model.predict(X_test)
```

Evalaute the model by calculating the accuracy of the model

[20]:

```
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
print('CLassification report')
print(metrics.classification_report(y_test,y_pred))
```

```
Accuracy: 0.9922370252170584
CLassification report
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    426770
           1       0.78      0.89      0.83      9403

    accuracy                           0.99    436173
   macro avg       0.89      0.94      0.91    436173
weighted avg       0.99      0.99      0.99    436173
```

The number of the available noncrash data according to the numbe rof killed ansd injured cyclist is more than the crash data, This might cause some problms in predicting the crashes and this also didn't let the model learn enough to predict if there is a crash will happen or not.

After we built a model to predict if there is a crash that could be dangerous for the cyclist or not we can see what is the closest bike station to the crashes.

[21]:

```
def find_nearest_station(station_coordinates,crash_point_lat,crash_point_lon):
    '''
    This function finds the closest station to a crash

    Parameters
    ----------
    station_coordinates : list of lists
        list of stations name and coordinate
    crash_point_lat: float
        the latitiude of the crash coordinate
    crash_point_lon: float
        the longitude of the crash coordinate

    '''
    # see if the crash happend exactly at any station
    res1 = [i for i, station in enumerate(station_coordinates) if␣
↪crash_point_lat in station]
    res2 = [i for i, station in enumerate(station_coordinates) if␣
↪crash_point_lon in station]
    crashInStation = list(set(res1) & set(res2))
    # see if the crash happend exactly at any station
    if len(crashInStation) > 0:
        print("The crash will happen exactly in %s bike station"␣
↪%station_coordinates[crashInStation[0]])
    else:
        #find the closest station
        distance_station =␣
↪[lonLatDistance([station[1],station[2]],[crash_point_lat,crash_point_lon])␣
↪for station in station_coordinates]
        min_distance_Idx = distance_station.index(np.min(distance_station))
        print("station:",station_coordinates[min_distance_Idx][0],"is the␣
↪closest citybike station to the crash with a distance is %.2fkm.
↪"%distance_station[min_distance_Idx])
```

[22]:
```
stations_coordinates_sorted = stations_coordinates(station_json_path)
```

Now, for 10 examples from the test dataset that our model predicted to have a crash in we will give some information about the place and the nearest bike station

[23]:
```
num=0
for i in range(len(y_pred)):
    x = X_test[i]
    y = y_pred[i]
    if y == 1:
        print("#####################################################")
        print("According to the informations that we have in %s, %s %s "
              %(" ".join(classes['ON STREET NAME'][x[2]].split()),classes['ZIP␣
↪CODE'][x[1]],classes['BOROUGH'][x[0]]), "a crash will happen")
        find_nearest_station(stations_coordinates_sorted,x[3],x[4])
        num += 1
        if num == 10:
            break
```

```
#####################################################
According to the informations that we have in 5 AVENUE, 11215.0 BROOKLYN  a
crash will happen
station: Carroll St & 5 Ave is the closest citybike station to the crash with a
distance is 10.19km.
#####################################################
```

```
According to the informations that we have in EAST TREMONT AVENUE, 10460.0 BRONX
a crash will happen
station: Lexington Ave & E 127 St is the closest citybike station to the crash
with a distance is 443.91km.
######################################################
According to the informations that we have in LEXINGTON AVENUE, 10016.0
MANHATTAN  a crash will happen
station: E 32 St & Park Ave is the closest citybike station to the crash with a
distance is 9.01km.
######################################################
According to the informations that we have in NOSTRAND AVENUE, nan nan  a crash
will happen
station: Macon St & Nostrand Ave is the closest citybike station to the crash
with a distance is 9.25km.
######################################################
According to the informations that we have in NAGLE AVENUE, 10040.0 MANHATTAN  a
crash will happen
station: Lenox Ave & W 130 St is the closest citybike station to the crash with
a distance is 333.13km.
######################################################
According to the informations that we have in HOYT STREET, 11217 BROOKLYN  a
crash will happen
station: Dean St & Hoyt St is the closest citybike station to the crash with a
distance is 9.25km.
######################################################
According to the informations that we have in EAST 42 STREET, 10017 MANHATTAN  a
crash will happen
station: Pershing Square North is the closest citybike station to the crash with
a distance is 5.32km.
######################################################
According to the informations that we have in JEROME AVENUE, 10453 BRONX  a
crash will happen
station: Lenox Ave & W 130 St is the closest citybike station to the crash with
a distance is 338.87km.
######################################################
According to the informations that we have in WEST 184 STREET, 10033 MANHATTAN
a crash will happen
station: W 129 St & Convent Ave is the closest citybike station to the crash
with a distance is 267.37km.
######################################################
According to the informations that we have in 21 AVENUE, 11214.0 BROOKLYN  a
crash will happen
station: 39 St & 2 Ave is the closest citybike station to the crash with a
distance is 322.52km.
```