1 Property Recommender System

1.1 1) The Executive Summary:

The goal of the project is to produce a tool in python that allows users to find real estate property in NYC based on several search criteria. These criteria will include the user's estimated budget, the neighborhood of the property, and the purpose/tax bracket of the property (Residential/Small Office, Rental Building/Co-op unit, Utility, or Commercial/Industrial Property). The tool will then display a list of properties that fit into the user's search criteria, including information like the address, the neighborhood, and the type of property.

Afterwards, the user is given the option to see additional properties that didn't fit exactly into the initial search criteria, but are still similar. Also included is a cell that displays a graphical google map with markers pointing at the relevant properties from the data output. In the end, we were able to create a tool that allows people to find information about various property that fits their search criteria and needs.

1.2 2) Background:

As current students, some of whom are new to the city, the problem of finding a place to live is very personal to us, so we wanted to try to create something that could be used as a resource to get information about property that we found would be relevant. Because of this, we decided to leave a certain degree of input parameters up to the user such as the price and the neighborhood.

The data comes from NYC Open Data and we started by pulling it directly and assembling it into a local file (sales.csv). The data spans through Fall of 2019 and reflects the following data categories:

Location Information
BOROUGH
NEIGHBORHOOD
ADDRESS
APARTMENT NUMBER
ZIP CODE
Property Physical Details
BUILDING CLASS CATEGORY
TAX CLASS AT PRESENT
BLOCK
LOT

EASE-MENT
BUILDING CLASS AT PRESENT
RESIDENTIAL UNITS
COMMERCIAL UNITS
TOTAL UNITS
LAND SQUARE FEET
GROSS SQUARE FEET
YEAR BUILT
Sales Information
SALE PRICE
SALE DATE
TAX CLASS AT TIME OF SALE
BUILDING CLASS AT TIME OF SALE

Since the scope of our group's goal was limited to properties in New York due to our use of NYC Open Data, as well as the fact that the end-goal of the project was to help people with finding properties to buy or live in, the most relevant location-based metrics were neighborhood and address. This is because aside from returning specific addresses and properties, the quantity of data returned by the output would show various availability of property with the right price range and room requirements across different neighborhoods. Combined with the use of the Google maps static API, this also allows users to use the tool to discover new neighborhoods to consider living in.

The most important physical detail related data elements were the the number of units and the square footage. Number of units allows users to properly filter down the lists of properties by their needs. On the other hand, square footage allows users to compare different properties, or even neighborhoods by the amount of space they could get at each price point after their data was filtered. One potential use of the square footage with the maps API was to put the square footage in the markers as text, but this resulted in a cluttered map and we decided to just label the markers numerically instead.

When it came to sales related information, the sales price and data were the important elements that we needed to consider here. The sale price is both an important search metric because of user budgets as well as an important part of the output that lets users compare properties and the overall expensiveness of neighborhoods on average. Conversely, the inclusion of sale date allows users to see how recent and accurate the sales price might be, and also allows them to more accurately price similar units in the same neighborhood that may not have been sold in a long time. For example, if a 2-bedroom unit in Alphabet City sold this past October for 3 million and a similar property on the same block has not been sold in 30 years, the first property's price may be a more accurate metric to users than the raw data held by NYC Open Data.

1.3 3) Project Description: What were the steps involved, in acquiring the data, transforming it, analyzing it and presenting it.

The google maps static API is not available on mashape/rapidAPI, so we had to set up a billing account with google to access the API and get output from it. Fortunately, they offer a sizeable amount of free pulls per month that were more than adequate for our needs. After registering properly and enabling the relevant API, we were able to obtain an API key that we used. After

that, the next step was to convert the addresses from the data output to something that would work in the get method for the API. At first, we started by converting each of the addresses in the tool's output to geographic coordinates (latitude, longitude) but eventually realized that the Google maps static API can also use addresses in a string format to search for locations. With New York as the scope of our data, we fed a python list of the addresses into a code cell that created a string to be inserted into the get method that put markers on a google map of the first five locations so that users can get a geographic idea of where in New York the properties they were looking at were located.

1.4 4) Conclusion and Further Steps

In conclusion, we were able to leverage several of the tools and resources from this class, from using MySQL with Python to calling APIs to create a tool that takes in a set of parameters to help users find a dataset of properties that fit their search criteria. For futher steps, we might be able to incorporate other data sets and create more detailed search parameters or output. For example, if we could find a way to standardize data outputs from other cities, we could expand the tool's scope beyond that of NY. Alternatively, we could find other datasets about NY such as neighborhood crime statistics or nearby amenities to display additional information about different neighborhoods and properties. We might also be able to accomplish these things by looking into other APIs to use beyond or even in conjunction with google maps. For example, we could create a user input parameter with their place of employment and have the google maps directions API calculate the commute distance by public transit. Additionally, as the users start using this system and with their consent if they allow us to store their search results, it will eventually lead to creation of a big-data that can be used for creating a personalized experience for every user and alternatively the data can be used to derive insights on the most watched listings and most attractive neighborhood to stay.

1.5 5) Lessons Learned

We learned how to use many of the different data tools from the class: acquiring data, putting it into a MySQL database, using SQL to search through it, using python to facilitate the entire process, and putting the data output into a format readable by a get method of an API to create extra visualizations. We learned some of the difficulties in clearly defining a project goal, as well as some of the difficulties involved in acquiring data using python. We also experimented with using different methods of input for different APIs and finding various ways they could interact with our data outputs as well as finding different ways a data set could be represented other than just having raw data displayed.

[1]:
```
import requests
import pandas as pd
import numpy as np
import json
import re
import MySQLdb as mdb
import matplotlib.pyplot as plt
```

[2]:
```
import MySQLdb as mdb

#con = mdb.connect(host = 'localhost',
#                   user = 'root',
#                   passwd = 'dwdstudent2015',
#                   charset='utf8', use_unicode=True);


con = mdb.connect(host = 'bigdata.stern.nyu.edu',
                  user = 'dealingF19GB3',
                  passwd = 'dealingF19GB3!!',
                  charset='utf8',
                  use_unicode=True);
```

[3]:
```
cursor = con.cursor()
table_name = 'Customer'
db = 'dealingF19GB3'
# Create a table
# Drop it if it exists
#create_drop_docks=''' drop table {db}.{table}'''.
 ↪format(db=db_name,table=table_name)
#cursor.execute(create_drop_docks)

# The {db} and {table} are placeholders for the parameters in the format(....)␣
 ↪statement
create_table_query = '''CREATE TABLE IF NOT EXISTS {db}.{table}
                                (
                                Customer_ID int NOT NULL AUTO_INCREMENT,
                                Name varchar(250),
                                Budget int,
                                Neighborhood varchar(100),
                                Tax_class int,
                                PRIMARY KEY(Customer_ID)
                                )'''.format(db=db, table=table_name)
cursor.execute(create_table_query)

cursor.close()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:19: Warning: (1050,
"Table 'Customer' already exists")
```

[4]:
```
# Importing uszipcode data to work on retriving zipcodes of the given latitute␣
 ↪and Longitide
import csv
from uszipcode import SearchEngine, SimpleZipcode, Zipcode
from tabulate import tabulate
```

[5]:
```
# Let's get the most recent property data from NYC Open Data
pd.options.display.width=None
pd.options.display.max_columns = None
data = pd.read_csv('sales.csv')
data.head(10)
```

[5]:

	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY
0	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS
1	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS
2	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS
3	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS
4	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS
5	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS
6	1	ALPHABET CITY	03 THREE FAMILY DWELLINGS
7	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS
8	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS
9	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS

	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDING CLASS AT PRESENT
0	1	376	43	NaN	S1
1	1	390	61	NaN	A4
2	1	400	19	NaN	A4
3	1	404	1	NaN	B9
4	1	404	1	NaN	B9
5	1	404	1	NaN	B9
6	1	377	56	NaN	C0
7	2	372	23	NaN	C1
8	2	373	19	NaN	C7
9	2	375	28	NaN	C4

	ADDRESS	APARTMENT NUMBER	ZIP CODE	RESIDENTIAL UNITS
0	743 EAST 6TH STREET	NaN	10009.0	1
1	189 EAST 7TH STREET	NaN	10009.0	1
2	526 EAST 5TH STREET	NaN	10009.0	1
3	166 AVENUE A	NaN	10009.0	2
4	166 AVENUE A	NaN	10009.0	2
5	166 AVENUE A	NaN	10009.0	2
6	263 EAST 7TH STREET	NaN	10009.0	3
7	300 EAST 3RD STREET	NaN	10009.0	12
8	332 EAST 4TH STREET	NaN	10009.0	28
9	738 EAST 6TH STREET	NaN	10009.0	11

	COMMERCIAL UNITS	TOTAL UNITS	LAND SQUARE FEET	GROSS SQUARE FEET
0	1	2	2090	3680
1	0	1	987	2183
2	0	1	1883	5200
3	0	2	1510	4520

```
4                0            2              1510              4520
5                0            2              1510              4520
6                0            3              2430              3600
7                0           12              2393              7989
8                2           30              4651             17478
9                0           11              1750              6500

   YEAR BUILT  TAX CLASS AT TIME OF SALE BUILDING CLASS AT TIME OF SALE  \
0      1940.0                         1                            S1
1      1860.0                         1                            A4
2      1900.0                         1                            A4
3      1900.0                         1                            B9
4      1900.0                         1                            B9
5      1900.0                         1                            B9
6      1899.0                         1                            C0
7      2001.0                         2                            C1
8      1920.0                         2                            C7
9      1900.0                         2                            C4

   SALE PRICE            SALE DATE
0     3200000  2019-07-24 00:00:00
1           0  2019-09-25 00:00:00
2     6100000  2018-12-03 00:00:00
3           0  2019-07-22 00:00:00
4           0  2018-11-29 00:00:00
5           0  2018-11-29 00:00:00
6     6300000  2019-04-30 00:00:00
7     1950000  2019-08-08 00:00:00
8    14000000  2019-01-09 00:00:00
9           0  2019-09-11 00:00:00
```

[6]:
```
# Replacing spaces in the Column names to '_'
data.columns = data.columns.str.replace(' ', '_')
data.columns = data.columns.str.strip('_')
```

[7]:
```
data
```

[7]:
```
       BOROUGH              NEIGHBORHOOD    BUILDING_CLASS_CATEGORY  \
0            1             ALPHABET CITY    01 ONE FAMILY DWELLINGS
1            1             ALPHABET CITY    01 ONE FAMILY DWELLINGS
2            1             ALPHABET CITY    01 ONE FAMILY DWELLINGS
3            1             ALPHABET CITY    02 TWO FAMILY DWELLINGS
4            1             ALPHABET CITY    02 TWO FAMILY DWELLINGS
…          …                         …                          …
17746        1  WASHINGTON HEIGHTS UPPER         22 STORE BUILDINGS
17747        1  WASHINGTON HEIGHTS UPPER  31 COMMERCIAL VACANT LAND
17748        1  WASHINGTON HEIGHTS UPPER         22 STORE BUILDINGS
```

```
17749        1  WASHINGTON HEIGHTS UPPER         22 STORE BUILDINGS
17750        1  WASHINGTON HEIGHTS UPPER  31 COMMERCIAL VACANT LAND

      TAX_CLASS_AT_PRESENT  BLOCK  LOT  EASE-MENT BUILDING_CLASS_AT_PRESENT  \
0                        1    376   43        NaN                        S1
1                        1    390   61        NaN                        A4
2                        1    400   19        NaN                        A4
3                        1    404    1        NaN                        B9
4                        1    404    1        NaN                        B9
…                      …    …  …         …                      …
17746                    4   2154    5        NaN                        K4
17747                    4   2174   97        NaN                        V1
17748                    4   2154    1        NaN                        K2
17749                    4   2154    5        NaN                        K4
17750                    4   2174   97        NaN                        V1

                      ADDRESS APARTMENT_NUMBER  ZIP_CODE  RESIDENTIAL_UNITS  \
0          743 EAST 6TH STREET             NaN   10009.0                  1
1          189 EAST 7TH STREET             NaN   10009.0                  1
2          526 EAST 5TH STREET             NaN   10009.0                  1
3                166 AVENUE A              NaN   10009.0                  2
4                166 AVENUE A              NaN   10009.0                  2
…                         …              …       …                 …
17746  1428 ST NICHOLAS AVENUE             NaN   10033.0                  0
17747         N/A NAGLE AVENUE             NaN       NaN                  0
17748  575-599 WEST 181 STREET             NaN   10033.0                  0
17749  1428 ST NICHOLAS AVENUE             NaN   10033.0                  0
17750         N/A NAGLE AVENUE             NaN       NaN                  0

       COMMERCIAL_UNITS  TOTAL_UNITS  LAND_SQUARE_FEET  GROSS_SQUARE_FEET  \
0                     1            2              2090               3680
1                     0            1               987               2183
2                     0            1              1883               5200
3                     0            2              1510               4520
4                     0            2              1510               4520
…                   …          …                …                …
17746                 2            2              2000               4880
17747                 0            0              2800                  0
17748                 9            9             17500              35000
17749                 2            2              2000               4880
17750                 0            0              2800                  0

       YEAR_BUILT  TAX_CLASS_AT_TIME_OF_SALE BUILDING_CLASS_AT_TIME_OF_SALE  \
0          1940.0                         1                            S1
1          1860.0                         1                            A4
2          1900.0                         1                            A4
3          1900.0                         1                            B9
```

```
4          1900.0                          1                          B9
…           …                        …                        …
17746      1905.0                          4                          K4
17747         0.0                          4                          V1
17748      1959.0                          4                          K2
17749      1905.0                          4                          K4
17750         0.0                          4                          V1

       SALE_PRICE            SALE_DATE
0         3200000  2019-07-24 00:00:00
1               0  2019-09-25 00:00:00
2         6100000  2018-12-03 00:00:00
3               0  2019-07-22 00:00:00
4               0  2018-11-29 00:00:00
…             …                  …
17746           0  2019-05-03 00:00:00
17747       15000  2019-06-12 00:00:00
17748           0  2019-07-25 00:00:00
17749           0  2019-05-03 00:00:00
17750       15000  2019-06-12 00:00:00

[17751 rows x 21 columns]
```

Data Cleaning Remove the rows with 0 in sale price

Removing entries from the dataset with empty Sale Price value.

[8]:
```
# Removing lines with 0 in the SALE PRICE
data = data[(data.SALE_PRICE != '0')]
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/ops/__init__.py:1115:
FutureWarning: elementwise comparison failed; returning scalar instead, but in
the future will perform elementwise comparison
  result = method(y)
```

Removing duplicate entries from the dataset if any.

[9]:
```
# Removing duplicate lines from the data set
data = data.drop_duplicates(subset=None, keep='first', inplace=False)
```

[10]:
```
# Convert the data type of column 'DOB' from string (DD/MM/YYYY) to datetime64
data['SALE_DATE'] = pd.to_datetime(data['SALE_DATE'])
```

[57]:
```
# sorting by Address
data.sort_values("ADDRESS", inplace = True)
data.head(10)
```

[57]:

	BOROUGH	NEIGHBORHOOD	BUILDING_CLASS_CATEGORY
3350	1	GREENWICH VILLAGE-CENTRAL	10 COOPS - ELEVATOR APARTMENTS
3599	1	GREENWICH VILLAGE-CENTRAL	13 CONDOS - ELEVATOR APARTMENTS
3598	1	GREENWICH VILLAGE-CENTRAL	13 CONDOS - ELEVATOR APARTMENTS
2154	1	FINANCIAL	21 OFFICE BUILDINGS
9612	1	MIDTOWN WEST	13 CONDOS - ELEVATOR APARTMENTS
9604	1	MIDTOWN WEST	13 CONDOS - ELEVATOR APARTMENTS
7567	1	MANHATTAN-UNKNOWN	13 CONDOS - ELEVATOR APARTMENTS
15970	1	UPPER WEST SIDE (59-79)	45 CONDO HOTELS
15968	1	UPPER WEST SIDE (59-79)	45 CONDO HOTELS
15174	1	UPPER WEST SIDE (59-79)	13 CONDOS - ELEVATOR APARTMENTS

	TAX_CLASS_AT_PRESENT	BLOCK	LOT	EASE-MENT	BUILDING_CLASS_AT_PRESENT
3350	2	550	22	NaN	D4
3599	2	529	1318	NaN	R4
3598	2	529	1315	NaN	R4
2154	4	13	1	NaN	O6
9612	2	1274	1619	NaN	R4
9604	2	1274	1479	NaN	R4
7567	2	1274	1480	NaN	R4
15970	4	1113	1384	NaN	RH
15968	4	1113	1347	NaN	RH
15174	2	1113	1426	NaN	R4

	ADDRESS	APARTMENT_NUMBER	ZIP_CODE
3350	1 5 AVENUE, 4D	NaN	10003.0
3599	1 BOND STREET, 4D	4D	10012.0
3598	1 BOND STREET, 6C	6C	10012.0
2154	1 BROADWAY	NaN	10004.0
9612	1 CENTRAL PARK SOUTH, 1809	1809	10019.0
9604	1 CENTRAL PARK SOUTH, 603	603	10019.0
7567	1 CENTRAL PARK SOUTH, 605	NaN	10019.0
15970	1 CENTRAL PARK WEST	1524	10023.0
15968	1 CENTRAL PARK WEST, 1218	1218	10023.0
15174	1 CENTRAL PARK WEST, 27D	27D	10023.0

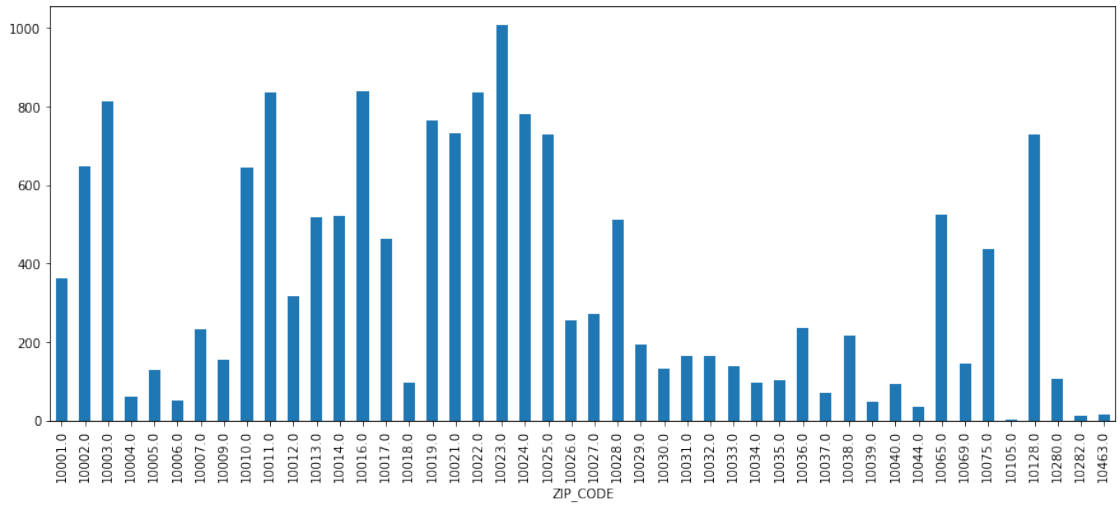
	RESIDENTIAL_UNITS	COMMERCIAL_UNITS	TOTAL_UNITS	LAND_SQUARE_FEET
3350	0	0	0	0
3599	0	0	0	0
3598	0	0	0	0
2154	0	16	16	17025
9612	0	0	0	0
9604	0	0	0	0
7567	0	0	0	0
15970	0	0	0	0
15968	0	0	0	0
15174	0	0	0	0

	GROSS_SQUARE_FEET	YEAR_BUILT	TAX_CLASS_AT_TIME_OF_SALE	\
3350	0	1927.0	2	
3599	1603	0.0	2	
3598	2527	0.0	2	
2154	180646	1921.0	4	
9612	5655	0.0	2	
9604	1994	0.0	2	
7567	1202	0.0	2	
15970	456	0.0	4	
15968	691	0.0	4	
15174	2165	0.0	2	

	BUILDING_CLASS_AT_TIME_OF_SALE	SALE_PRICE	SALE_DATE
3350	D4	0	2019-05-01
3599	R4	2537500	2019-06-19
3598	R4	0	2019-03-06
2154	O6	139500000	2018-11-16
9612	R4	31250000	2019-10-07
9604	R4	19600000	2019-03-08
7567	R4	19600000	2019-03-08
15970	RH	1425000	2018-11-16
15968	RH	950000	2018-12-28
15174	R4	6100000	2019-09-16

[12]:
```
#Let's use bar graph to plot the count of houses in each zipcode
data.groupby('ZIP_CODE').ZIP_CODE.count().plot(kind='bar', figsize=(15,6))
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2aee0fa2f550>



[15]:
```
# Let's get the most recent crime data from NYC Open Data (Data used is cleaned
 ↪by removing the unwanted columns)
crime = pd.read_csv('crime_man.csv')
crime.head(10)
```

[15]:

	CMPLNT_DT	CMPLNT_TM	OFNS_DESC	LAW_CAT_CD	BORO_NM	\
0	6/30/2019	19	ASSAULT 3 & RELATED OFFENSES	MISDEMEANOR	MANHATTAN	
1	6/30/2019	16	GRAND LARCENY	FELONY	MANHATTAN	
2	6/30/2019	17	PETIT LARCENY	MISDEMEANOR	MANHATTAN	
3	6/30/2019	20	HARRASSMENT 2	VIOLATION	MANHATTAN	
4	6/30/2019	18	MISCELLANEOUS PENAL LAW	FELONY	MANHATTAN	
5	6/30/2019	19	ASSAULT 3 & RELATED OFFENSES	MISDEMEANOR	MANHATTAN	
6	6/30/2019	3	ASSAULT 3 & RELATED OFFENSES	MISDEMEANOR	MANHATTAN	
7	6/30/2019	4	HARRASSMENT 2	VIOLATION	MANHATTAN	
8	6/30/2019	11	GRAND LARCENY	FELONY	MANHATTAN	
9	6/30/2019	0	ASSAULT 3 & RELATED OFFENSES	MISDEMEANOR	MANHATTAN	

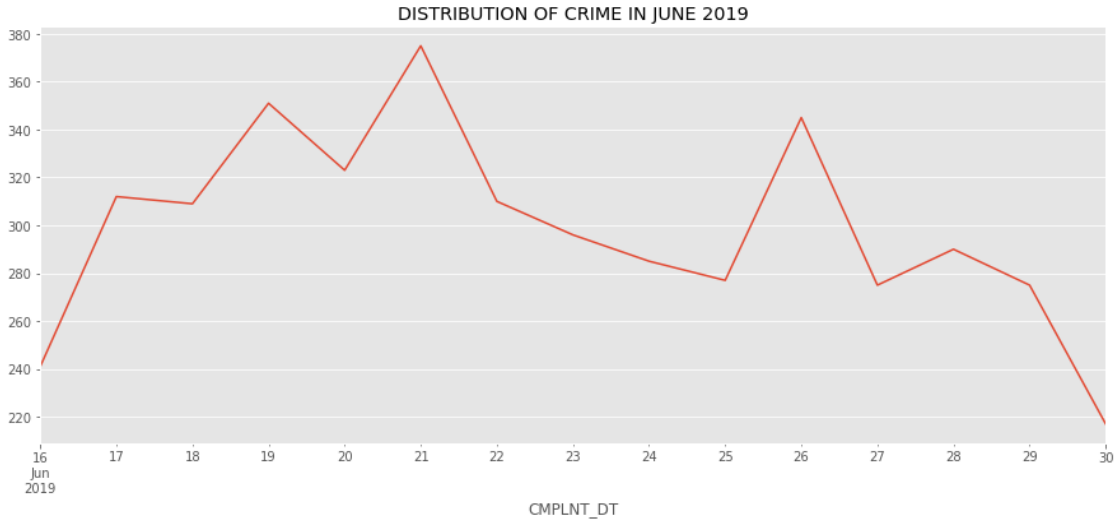
	LATITUDE	LONGITUDE
0	40.736570	-74.001095
1	40.745309	-73.986847
2	40.803708	-73.935819
3	40.724644	-73.974238
4	40.784245	-73.947071
5	40.736570	-74.001095
6	40.756555	-73.992147
7	40.729417	-74.001697
8	40.709177	-74.009012
9	40.823182	-73.941675

[16]:
```
# Convert the data type of column 'DOB' from string (DD/MM/YYYY) to datetime64
crime['CMPLNT_DT'] = pd.to_datetime(crime['CMPLNT_DT'])
```

[17]:
```
# Sorting crime data based on the date of crime
crime.sort_values("CMPLNT_DT", ascending=False, inplace = True)
```

[18]:
```
#Let's use bar graph to plot the count of houses in each zipcode
crime.groupby('CMPLNT_DT').CMPLNT_DT.count().plot(title = 'DISTRIBUTION OF
 ↪CRIME IN JUNE 2019' ,kind='line', figsize=(15,6))
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2aee0fa2fac8>
```



[19]:
```
# Retriving zipcodes to the latitude and Longitude values
search = SearchEngine()
zip = []
with open('crime_man.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        lat = row[5]
        long = row[6]
        if lat=="" or long =="" or lat=="LATITUDE" or long=="LONGITUDE":
            continue
        result = search.by_coordinates(float(lat), float(long), radius=30,␣
 ↪returns=1)
        zipcode = result[0]
        zip.append(zipcode.zipcode)
csvFile.close()
```

[20]:
```
# Creating a new column-ZIPCODE in the CRIME dataframe
crime['ZIPCODE'] = zip
crime.head(10)
```

[20]:

	CMPLNT_DT	CMPLNT_TM	OFNS_DESC	LAW_CAT_CD	\
0	2019-06-30	19	ASSAULT 3 & RELATED OFFENSES	MISDEMEANOR	
149	2019-06-30	2	PETIT LARCENY	MISDEMEANOR	
138	2019-06-30	17	HARRASSMENT 2	VIOLATION	
139	2019-06-30	2	BURGLARY	FELONY	
140	2019-06-30	9	OFF. AGNST PUB ORD SENSBLTY &	MISDEMEANOR	
141	2019-06-30	18	GRAND LARCENY	FELONY	
142	2019-06-30	2	GRAND LARCENY	FELONY	
143	2019-06-30	22	ASSAULT 3 & RELATED OFFENSES	MISDEMEANOR	

```
144 2019-06-30         15                  HARRASSMENT 2    VIOLATION
145 2019-06-30         19                  PETIT LARCENY  MISDEMEANOR

       BORO_NM   LATITUDE  LONGITUDE ZIPCODE
0    MANHATTAN  40.736570 -74.001095   10011
149  MANHATTAN  40.862177 -73.919403   10001
138  MANHATTAN  40.721286 -73.983450   10035
139  MANHATTAN  40.867911 -73.920484   10009
140  MANHATTAN  40.765024 -73.984836   10028
141  MANHATTAN  40.756642 -73.988372   10011
142  MANHATTAN  40.784245 -73.947071   10018
143  MANHATTAN  40.763438 -73.992702   10012
144  MANHATTAN  40.807009 -73.951601   10271
145  MANHATTAN  40.816801 -73.947240   10030
```

[21]:
```
# Changing the Object type of ZIPCODE to Float
crime['ZIPCODE'] = pd.to_numeric(crime['ZIPCODE'],downcast='float')
```

[22]:
```
# Printing the data types of the columnin the Crime dataframe
# Data profiling of crime dataset
print(crime.dtypes)
```

```
CMPLNT_DT     datetime64[ns]
CMPLNT_TM              int64
OFNS_DESC             object
LAW_CAT_CD            object
BORO_NM               object
LATITUDE             float64
LONGITUDE            float64
ZIPCODE              float32
dtype: object
```

[23]:
```
# Data profiling of NY Property Sales Data
print(data.dtypes)
```

```
BOROUGH                           int64
NEIGHBORHOOD                     object
BUILDING_CLASS_CATEGORY          object
TAX_CLASS_AT_PRESENT             object
BLOCK                             int64
LOT                               int64
EASE-MENT                       float64
BUILDING_CLASS_AT_PRESENT        object
ADDRESS                          object
APARTMENT_NUMBER                 object
ZIP_CODE                        float64
RESIDENTIAL_UNITS                 int64
```

```
COMMERCIAL_UNITS                                int64
TOTAL_UNITS                                     int64
LAND_SQUARE_FEET                                int64
GROSS_SQUARE_FEET                               int64
YEAR_BUILT                                    float64
TAX_CLASS_AT_TIME_OF_SALE                       int64
BUILDING_CLASS_AT_TIME_OF_SALE                 object
SALE_PRICE                                      int64
SALE_DATE                             datetime64[ns]
dtype: object
```

[24]:
```
# Function to retreive SAles data with the respective input of the user
def query(amount, neighborhood, tax_class):
    return data[(data.SALE_PRICE <= int(amount)) & (data.NEIGHBORHOOD ==␣
 ↪neighborhood) & (data.TAX_CLASS_AT_PRESENT ==␣
 ↪tax_class)][['NEIGHBORHOOD','BUILDING_CLASS_CATEGORY', 'ADDRESS',␣
 ↪'LAND_SQUARE_FEET', 'GROSS_SQUARE_FEET', 'SALE_DATE']]
```

[25]:
```
# Function to retreive wider SAles data with the respective input of the user
def wideQuery(amount, neighborhood, tax_class):
    var = data[(data.NEIGHBORHOOD == neighborhood)][['ZIP_CODE']]
    var = var.ZIP_CODE.unique()
    val = data[(data.SALE_PRICE <= int(amount)) & (data.ZIP_CODE.
 ↪isin(var))][['NEIGHBORHOOD','BUILDING_CLASS_CATEGORY', 'ADDRESS',␣
 ↪'ZIP_CODE', 'LAND_SQUARE_FEET', 'GROSS_SQUARE_FEET', 'SALE_DATE']]
    val.dropna(subset=['ZIP_CODE'], inplace=True)
    return val
```

[26]:
```
# Function to retrieve the crime happened in the given zipcode
def zipfunc(var):
    list = (crime[crime.ZIPCODE.
 ↪isin(var)][['CMPLNT_DT','OFNS_DESC','LAW_CAT_CD','BORO_NM']])
    return list
```

[27]:
```
# Function call for the zipcode match in the crime set to the sales data frame
def crimeQuery(neighborhood):
    var = data[data.NEIGHBORHOOD == neighborhood][['ZIP_CODE']]
    var = var.ZIP_CODE.unique()
    val = zipfunc(var)
    return val
```

User Interface

[68]:
```
        new_name =""
        new_neighborhood = ""
        import time
        if __name__ == "__main__":
```

```
                table={}
                st = ""
                name = ""
                amount = ""
                neighborhood = ""
                print("Do you want to start the search!! (Yes/Quit)")
                while(st!="quit" and st!="Quit"):
                        st = input()
                        if(st!='Yes' and st!='yes'):
                                time.sleep(2)
                                print("Quitting..")
                                time.sleep(2)
                                print("Have a great time ahead!!")
                                break
                        print("Kindy Enter Your Full Name:")
                        name = input()
                        new_name = "'"+ name +"'"
                        if( name == 'quit' or name == 'Quit'):
                                break
                        print("How much are you estimating to spend on your␣
↪Dream House:")
                        amount = input()
                        if( amount == 'quit' or amount == 'Quit'):
                                break
                        print("What Neighborhood would you like: (eg: CHELSEA,␣
↪ALPHABET CITY)")
                        neighborhood = input()
                        new_neighborhood = "'" + neighborhood +"'"
                        if( neighborhood == 'quit' or neighborhood == 'Quit' or␣
↪neighborhood == 'QUIT'):
                                break
                        print("What is the purpose of the buy?")
                        print("Tax class 1: Residential/Small Office")
                        print("Tax class 2: Rental buildings/Cooperative unit")
                        print("Tax class 3: Utility")
                        print("Tax class 4: Commercial or industrial␣
↪properties")
                        tax_class = input()
                        if( tax_class == 'quit' or tax_class == 'Quit'):
                                break
                        output = query(amount, neighborhood.upper(), tax_class)
                        if len(output)==0:
                                print("Unfortunately there are no listings for␣
↪your search criteria, kindly search again")
                                time.sleep(2)
                                print("Quitting..")
                                time.sleep(2)
```

```
                                print("Have a great time ahead!!")
                                break
                        else:
                                print("According to your inputs, following are␣
↪the places that people have prefered before:")
                                table = tabulate(output.head(20),␣
↪headers='keys', tablefmt="simple")
                                print(table)
                                print("These were some of the top listings")
                                print("Do you find anything that might interest␣
↪you? Or do you want to look out for more? (more/quit)")
                                option = input()
                                if( option == 'quit' or option == 'Quit'):
                                        break
                                else:
                                        output_more = wideQuery(amount,␣
↪neighborhood.upper(), tax_class)
                                        table = tabulate(output_more.head(20),␣
↪headers='keys', tablefmt="simple")
                                        print(table)
                                        print("These were some of the top 20␣
↪listings")
                                        print("Do you need the deailed list(Yes/
↪No)")
                                        res = input()
                                        if (res == 'Yes' or res == 'yes'):
                                                print(output)
                                        elif( res == 'quit' or res == 'Quit'):
                                                break
                                        print("Please find below the crime␣
↪details of the region you have selected")
                                        output_crime = crimeQuery(neighborhood.
↪upper())
                                        table = tabulate(output_crime.head(20),␣
↪headers='keys', tablefmt="simple")
                                        print(table)
                                        print("These were some of the recent␣
↪crimes in the area you are looking to live in")
                                        print("Quit??")
                                        st = input()
                                        if( st == 'quit' or st == 'Quit'):
                                                print("Quitting..")
                                                time.sleep(2)
                                                print("Have a great time ahead!!
↪")
                                                break
```

```
Do you want to start the search!! (Yes/Quit)
Yes
Kindy Enter Your Full Name:
Sundar Pichai
How much are you estimating to spend on your Dream House:
5000000
What Neighborhood would you like: (eg: CHELSEA, ALPHABET CITY)
chelsea
What is the purpose of the buy?
Tax class 1: Residential/Small Office
Tax class 2: Rental buildings/Cooperative unit
Tax class 3: Utility
Tax class 4: Commercial or industrial properties
1
According to your inputs, following are the places that people have prefered
before:
     NEIGHBORHOOD    BUILDING_CLASS_CATEGORY    ADDRESS
LAND_SQUARE_FEET    GROSS_SQUARE_FEET  SALE_DATE
---  --------------  -------------------------  --------------------
-----------------  ------------------  -------------------
146  CHELSEA         03 THREE FAMILY DWELLINGS  211 WEST 22ND STREET
847                 2100  2019-08-26 00:00:00
143  CHELSEA         02 TWO FAMILY DWELLINGS    213 WEST 22ND STREET
1199                2448  2019-09-03 00:00:00
137  CHELSEA         01 ONE FAMILY DWELLINGS    218 WEST 15TH STREET
2141                3935  2019-05-22 00:00:00
144  CHELSEA         02 TWO FAMILY DWELLINGS    278 WEST 25TH STREET
937                 2052  2019-08-27 00:00:00
141  CHELSEA         02 TWO FAMILY DWELLINGS    329 WEST 20TH STREET
1104                2460  2018-12-27 00:00:00
142  CHELSEA         02 TWO FAMILY DWELLINGS    353 WEST 22ND STREET
1234                3300  2019-01-03 00:00:00
145  CHELSEA         03 THREE FAMILY DWELLINGS  448 WEST 25TH STREET
1925                2666  2019-10-07 00:00:00
133  CHELSEA         01 ONE FAMILY DWELLINGS    504 WEST 22ND STREET
1102                5390  2019-02-28 00:00:00
These were some of the top listings
Do you find anything that might interest you? Or do you want to look out for
more? (more/quit)
more
      NEIGHBORHOOD            BUILDING_CLASS_CATEGORY            ADDRESS
ZIP_CODE    LAND_SQUARE_FEET    GROSS_SQUARE_FEET  SALE_DATE
----  ----------------------  ---------------------------------
------------------------  ----------  ------------------  --------------------
-------------------
2621  GRAMERCY                10 COOPS - ELEVATOR APARTMENTS     1 LEXINGTON
AVENUE, 9D          10010                   0                   0  2019-03-18
00:00:00
```

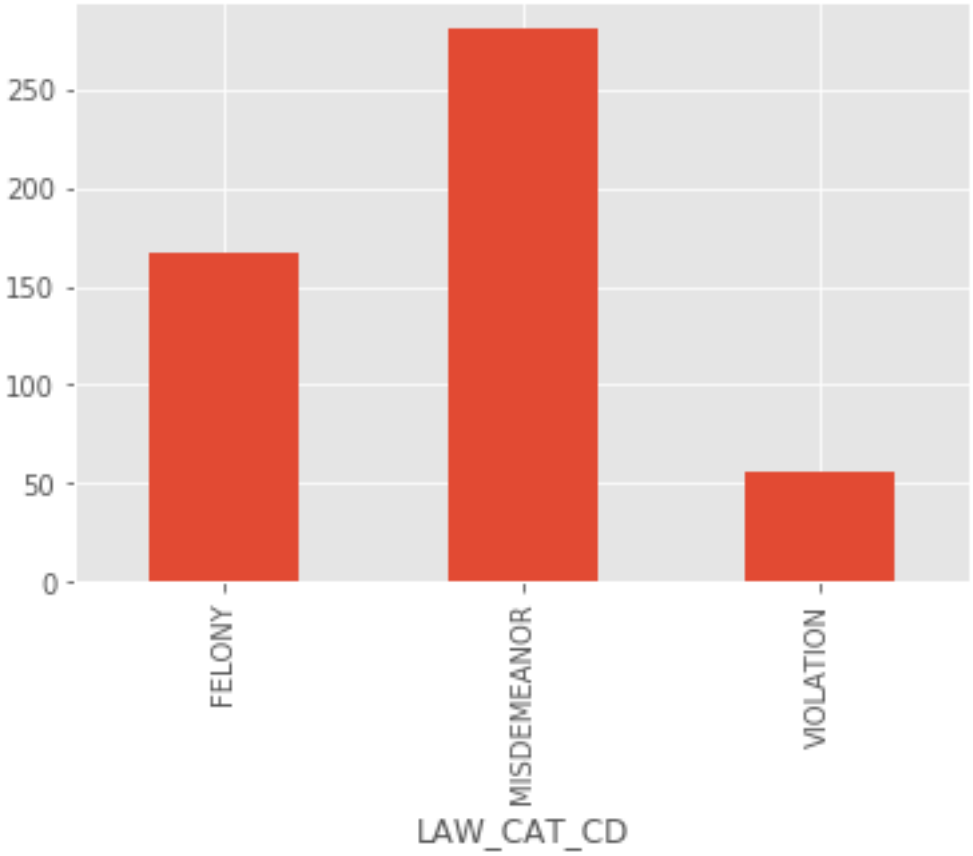
```
4250  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON SQUARE
10014                    0                1581  2018-11-06 00:00:00
4256  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 14A E          10014                    0                  813
2019-03-04 00:00:00
4254  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 14A E          10014                    0                  813
2019-07-23 00:00:00
4240  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 2E W           10014                    0                 1476
2019-04-23 00:00:00
4242  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 4A W           10014                    0                 1187
2019-01-18 00:00:00
4244  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 6GW            10014                    0                 1295
2019-08-27 00:00:00
4252  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 8C E           10014                    0                 1410
2019-05-22 00:00:00
4246  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 MORTON
SQUARE, 9E W           10014                    0                 1160
2019-07-25 00:00:00
4224  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 SHERIDAN
SQUARE                 10014                    0                  466  2018-12-06
00:00:00
4225  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 SHERIDAN
SQUARE, 4C             10014                    0                  460  2019-02-21
00:00:00
4226  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 SHERIDAN
SQUARE, 4D             10014                    0                  440  2019-02-21
00:00:00
4227  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 SHERIDAN
SQUARE, 5E             10014                    0                  450  2019-06-12
00:00:00
4228  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 SHERIDAN
SQUARE, 8C             10014                    0                  500  2019-01-01
00:00:00
4229  GREENWICH VILLAGE-WEST  13 CONDOS - ELEVATOR APARTMENTS    1 SHERIDAN
SQUARE, PHN            10014                    0                  800  2019-08-26
00:00:00
3987  GREENWICH VILLAGE-WEST  07 RENTALS - WALKUP APARTMENTS     1-3 CHARLES
STREET                 10014                 1562                 7482  2019-07-31
00:00:00
4467  GREENWICH VILLAGE-WEST  15 CONDOS - 2-10 UNIT RESIDENTIAL  10 BEDFORD
STREET, 1              10014                    0                  804  2019-03-14
00:00:00
4468  GREENWICH VILLAGE-WEST  15 CONDOS - 2-10 UNIT RESIDENTIAL  10 BEDFORD
```

```
STREET, 2            10014                    0                   625  2019-03-14
00:00:00
4469  GREENWICH VILLAGE-WEST  15 CONDOS - 2-10 UNIT RESIDENTIAL  10 BEDFORD
STREET, 4            10014                    0                   891  2019-03-14
00:00:00
4407  GREENWICH VILLAGE-WEST  17 CONDO COOPS                      10 CHRISTOPHER
STREET, 7C       10014                    0                    0  2019-01-28
00:00:00
These were some of the top 20 listings
Do you need the deailed list(Yes/No)
no
Please find below the crime details of the region you have selected
     CMPLNT_DT            OFNS_DESC                       LAW_CAT_CD    BORO_NM
---  -------------------  ------------------------------  ------------
---------
  0  2019-06-30 00:00:00  ASSAULT 3 & RELATED OFFENSES    MISDEMEANOR
MANHATTAN
149  2019-06-30 00:00:00  PETIT LARCENY                   MISDEMEANOR
MANHATTAN
141  2019-06-30 00:00:00  GRAND LARCENY                   FELONY
MANHATTAN
146  2019-06-30 00:00:00  ASSAULT 3 & RELATED OFFENSES    MISDEMEANOR
MANHATTAN
153  2019-06-30 00:00:00  GRAND LARCENY                   FELONY
MANHATTAN
154  2019-06-30 00:00:00  MISCELLANEOUS PENAL LAW         FELONY
MANHATTAN
155  2019-06-30 00:00:00  HARRASSMENT 2                   VIOLATION
MANHATTAN
156  2019-06-30 00:00:00  DANGEROUS WEAPONS               FELONY
MANHATTAN
159  2019-06-30 00:00:00  HARRASSMENT 2                   VIOLATION
MANHATTAN
136  2019-06-30 00:00:00  ASSAULT 3 & RELATED OFFENSES    MISDEMEANOR
MANHATTAN
111  2019-06-30 00:00:00  MISCELLANEOUS PENAL LAW         FELONY
MANHATTAN
113  2019-06-30 00:00:00  PETIT LARCENY                   MISDEMEANOR
MANHATTAN
120  2019-06-30 00:00:00  CRIMINAL MISCHIEF & RELATED OF  MISDEMEANOR
MANHATTAN
131  2019-06-30 00:00:00  OFFENSES AGAINST PUBLIC ADMINI  MISDEMEANOR
MANHATTAN
205  2019-06-30 00:00:00  CRIMINAL MISCHIEF & RELATED OF  MISDEMEANOR
MANHATTAN
202  2019-06-30 00:00:00  OFFENSES AGAINST THE PERSON     MISDEMEANOR
MANHATTAN
204  2019-06-30 00:00:00  POSSESSION OF STOLEN PROPERTY   FELONY
```

```
MANHATTAN
168   2019-06-30 00:00:00   GRAND LARCENY                     FELONY
MANHATTAN
182   2019-06-30 00:00:00   ASSAULT 3 & RELATED OFFENSES      MISDEMEANOR
MANHATTAN
189   2019-06-30 00:00:00   GRAND LARCENY                     FELONY
MANHATTAN
These were some of the recent crimes in the area you are looking to live in
Quit??
quit
Quitting..
Have a great time ahead!!
```

[69]:
```
# count the number of crime by crime_level
output_crime.groupby('LAW_CAT_CD').LAW_CAT_CD.count().plot(kind='bar')
```

[69]: <matplotlib.axes._subplots.AxesSubplot at 0x2aee13c670b8>

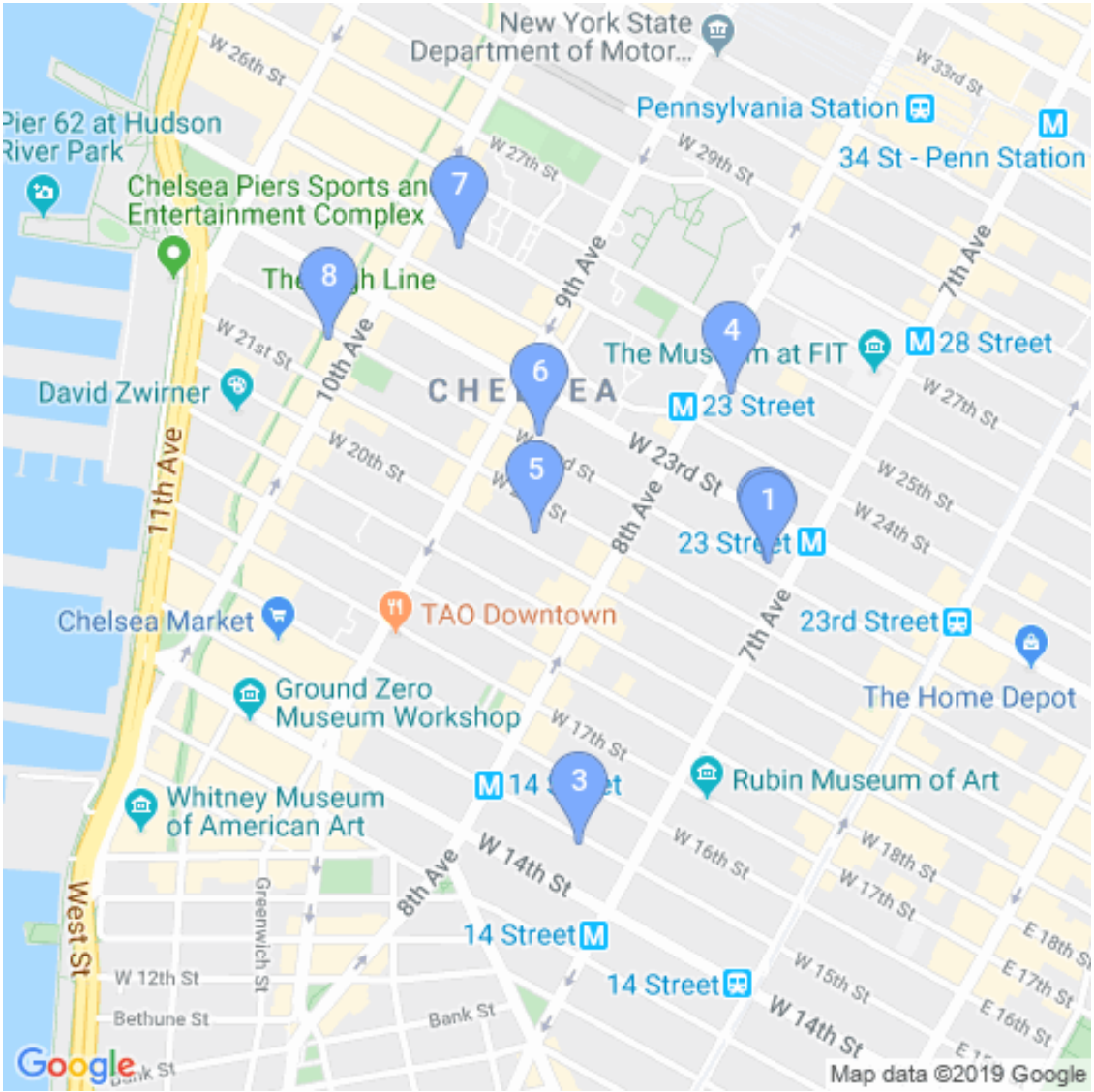


1.5.1 Using the Google Maps API to plot the listings for search results of entered neighborhood

[70]:
```
# this cell uses the google maps static API to display a map with markers on␣
 ↪the first 5 addresses listed by the output.

# this controls how many markers are displayed on the map
numDisplayed = 15

# pull in the list of addresses from the data output
addressList = []
addressList = output['ADDRESS']

# this function converts the list of addresses to text to be inserted into the␣
 ↪get request to google maps static API
def address2markers(addList, maxMarkers):
    markerOutput = ""
    numLabel = 1
    for i in addList:
        if numLabel > maxMarkers:
            return markerOutput
        markerOutput = markerOutput + "markers=color:blue%7Clabel:" +␣
 ↪str(numLabel) + "%7C" + i + " New York, NY&"
        numLabel = numLabel + 1

    return markerOutput

# here we import requests to make http requests and Image to display the map
import requests
from IPython.display import Image

# google maps static API key
api_key = "AIzaSyCpzQc9GnqxvUbxWZ9LjU5rn97dtGVpI94"

# google maps static API URL
url = "https://maps.googleapis.com/maps/api/staticmap?"

# defines the center of the map
center = "Midtown Manhattan, NY"

# defines the zoom level of the map
zoom = 11.5

# uses the address2markers function to create the marker information for the␣
 ↪requests
markertext = address2markers(addressList, numDisplayed)
```

```
# get for google maps static API with parameters included
# there is no "&" after the markertext variable because it is included by the␣
 ↪function that creates it
# this is the case so that it can handle multiple markers
r = requests.get(url + "zoom=" +
                 str(zoom) + "&size=500x500&" + markertext +␣
 ↪"sensor=false&key= " +
                            api_key)

#display the map
Image(r.content)
```

[70]:

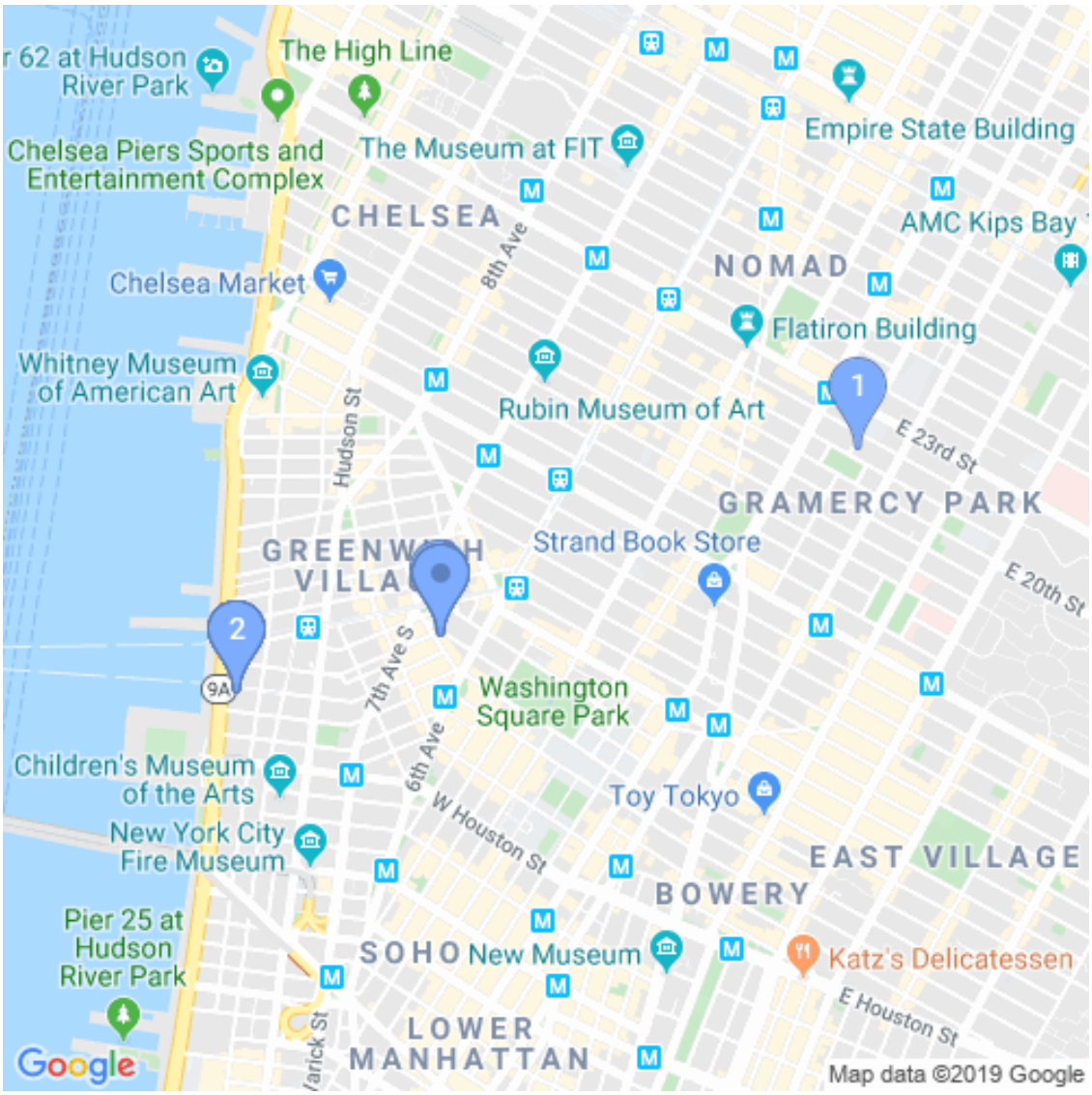


1.5.2 Using the Google Maps API to plot the listings for detailed search results around the entered neighborhood

[71]:
```
# this cell uses the google maps static API to display a map with markers on␣
 ↪the first 5 addresses listed by the output.

# this controls how many markers are displayed on the map
numDisplayed = 15

# pull in the list of addresses from the data output
addressList = []
addressList = output_more['ADDRESS']

# this function converts the list of addresses to text to be inserted into the␣
 ↪get request to google maps static API
def address2markers(addList, maxMarkers):
    markerOutput = ""
    numLabel = 1
    for i in addList:
        if numLabel > maxMarkers:
            return markerOutput
        markerOutput = markerOutput + "markers=color:blue%7Clabel:" +␣
 ↪str(numLabel) + "%7C" + i + " New York, NY&"
        numLabel = numLabel + 1

    return markerOutput

# here we import requests to make http requests and Image to display the map
import requests
from IPython.display import Image

# google maps static API key
api_key = "AIzaSyCpzQc9GnqxvUbxWZ9LjU5rn97dtGVpI94"

# google maps static API URL
url = "https://maps.googleapis.com/maps/api/staticmap?"

# defines the center of the map
center = "Midtown Manhattan, NY"

# defines the zoom level of the map
zoom = 11.5

# uses the address2markers function to create the marker information for the␣
 ↪requests
markertext = address2markers(addressList, numDisplayed)
```

```
# get for google maps static API with parameters included
# there is no "&" after the markertext variable because it is included by the␣
 ↪function that creates it
# this is the case so that it can handle multiple markers
r = requests.get(url + "zoom=" +
                  str(zoom) + "&size=500x500&" + markertext +␣
 ↪"sensor=false&key= " +
                             api_key)

#display the map
Image(r.content)
```

[71]:



[72]:
```
#Here we are inserting the customer information that can one day grow into a␣
 ↪big-data and can be used to display the most
#Searched neighborhood and the most watched propery listing with respect to the␣
 ↪user search terms just like any real-estate application

from datetime import date, datetime, timedelta
table_name = 'Customer'
db = 'dealingF19GB3'

#We capture the details entered by the user for prediction at a later stage␣
 ↪when the database consist of many records for
#further intelligent prediction and recommendation
query_template = '''INSERT INTO {db}.{table}(Name,
                                             Budget,
                                             Neighborhood,
                                             Tax_class
                                             )
                    VALUES ({Name}, {Budget}, {Neighborhood}, {Tax_Class})'''.
 ↪format(db=db, table=table_name,
                                                                              ␣
 ↪      Name = new_name, Budget = amount, Neighborhood = new_neighborhood,␣
 ↪Tax_Class = tax_class)
cursor = con.cursor()
cursor.execute(query_template)
print(cursor.rowcount, "Record inserted successfully into table")
con.commit()
cursor.close()
```

```
1 Record inserted successfully into table
```

[73]:
```
#Here we are displaying the data that have been entered by the users of the␣
 ↪system
import pandas as pd
table_name = 'Customer'
db = 'dealingF19GB3'

cur = con.cursor(mdb.cursors.DictCursor)
cur.execute("SELECT * FROM {db}.{table}".format(db=db, table=table_name))
rows = cur.fetchall()
cur.close()
```

[74]:
```
rows #To display the customer records.
```

[74]:
```
({'Customer_ID': 1,
  'Name': 'XYZ',
  'Budget': 1000,
  'Neighborhood': 'Manhattan',
```

```
 'Tax_class': 1},
{'Customer_ID': 2,
 'Name': 'Mark Zukerberg',
 'Budget': 50000,
 'Neighborhood': 'Manhattan',
 'Tax_class': 1},
{'Customer_ID': 3,
 'Name': 'SS',
 'Budget': 3200000,
 'Neighborhood': 'ALPHABET CITY',
 'Tax_class': 1},
{'Customer_ID': 4,
 'Name': 'Elon Musk',
 'Budget': 320000,
 'Neighborhood': 'Chelsea',
 'Tax_class': 1},
{'Customer_ID': 5,
 'Name': 'Jeff Bezos',
 'Budget': 3200000,
 'Neighborhood': 'chelsea',
 'Tax_class': 1},
{'Customer_ID': 6,
 'Name': 'Bill Gates',
 'Budget': 5000000,
 'Neighborhood': 'Alphabet city',
 'Tax_class': 1},
{'Customer_ID': 7,
 'Name': 'Sundar Pichai',
 'Budget': 5000000,
 'Neighborhood': 'chelsea',
 'Tax_class': 1})
```

[ ]: