Introduction

We all have different choices when selecting an apartment. Price can be one obvious measure. Also, some people have different neighbourhood choices. Our problem is “Finding the best neighbourhood to live in Manhattan for a person who is going to gym”. We are trying to find the best place to live based on the specific type of gyms that he is interested in and then predict the average house price that this person has to pay to buy a house in the selected area.

Data

We are going to need two types of data to do this analysis. Those are,

1. Location Data
2. Selling Price Data (Historical)

Location data can be obtained from various APIs like Google Map, OpenstreetMap and Foursqure [1]. I have previously used both Google Map and Openstreetmap APIs So I decided to go with Foursquare. You will have to sign up and obtain a Client ID and a Client Secret in order to access the API. The Manhattan average selling price data [2] were obtained from the New York Department of Finance [3] website.

Methodology

- ## Selecting a place to live
    - ### Download Data

Manhattan has 40 neighborhoods[4]. First, we download it from the above-given URL. The data can be explored as follows.

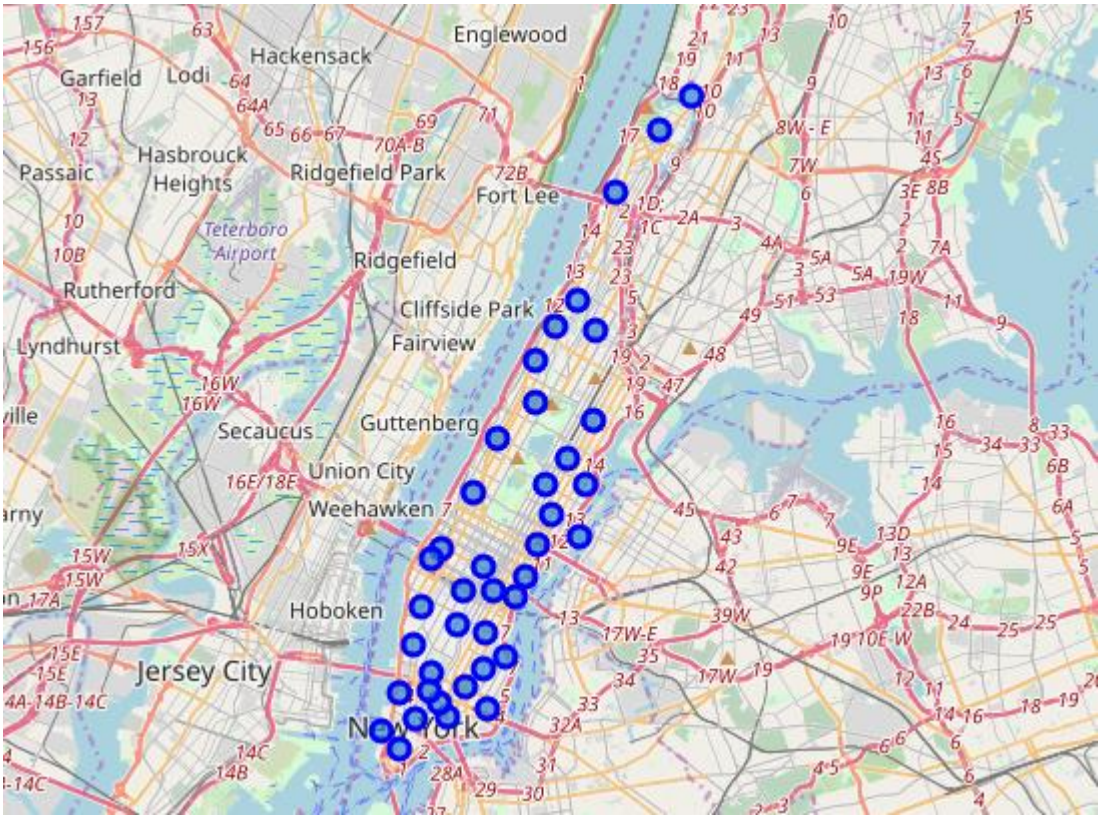
    - ### Feature Analysis

We can see the analyse the features in the data. Following image shows the features in the data.

```
{'geometry': {'coordinates': [-73.84720052054902, 40.89470517661],
  'type': 'Point'},
 'geometry_name': 'geom',
 'id': 'nyu_2451_34572.1',
 'properties': {'annoangle': 0.0,
  'annoline1': 'Wakefield',
  'annoline2': None,
  'annoline3': None,
  'bbox': [-73.84720052054902,
   40.89470517661,
   -73.84720052054902,
   40.89470517661],
  'borough': 'Bronx',
  'name': 'Wakefield',
  'stacked': 1},
 'type': 'Feature'}
```

Visualize neighbourhood

GeoCoder Instance with a user agent which helps us to work with coordinates and addresses. We can visualize the neighbourhood using the coordinates.



Get the Gym data

We can use the FourSqure API to obtain the Gym data in each neighbourhood. 100 Gyms that are in area within a radius of 500 meters. We are using FourSqure CategoryId field to identify the Gym Category. Please visit developer docs[5] to identify the available categories.

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Marble Hill	40.876551	-73.91066	Astral Fitness & Wellness Center	40.876705	-73.906372	Gym
1	Marble Hill	40.876551	-73.91066	Blink Fitness	40.877271	-73.905595	Gym / Fitness Center
2	Marble Hill	40.876551	-73.91066	TCR The Club of Riverdale	40.878628	-73.914568	Gym / Fitness Center
3	Marble Hill	40.876551	-73.91066	Planet Fitness	40.874088	-73.909137	Gym / Fitness Center
4	Marble Hill	40.876551	-73.91066	Bikram Yoga	40.876844	-73.906204	Yoga Studio

Encode data

'Venue Category' column has **categorical variables**. It is better to convert this categorical variable to a **numeric representation** in order to analyse further. There are two main ways and those are **Label encoding** and **One Hot Encoding**. You can learn them [6]. We will be using One Hot Encoding on 'Venue Category'. It will generate 40 new columns based on the feature categories and add 1 to the column relevant to the categorical feature while making all the other column values to 0.

	Neighborhood	Athletics & Sports	Bike Shop	Boxing Gym	Building	Chiropractor	Climbing Gym	Clothing Store	Club House	College Gym	...	Pool	Residential Building (Apartment / Condo)	Spa	Spiritual Center	Sporting Event	Ten Co
0	Marble Hill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	Marble Hill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2	Marble Hill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
3	Marble Hill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	Marble Hill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
5	Marble Hill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
6	Marble Hill	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	
7	Chinatown	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
8	Chinatown	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
9	Chinatown	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
10	Chinatown	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

○ Select Gyms

Our goal is to find the best neighbourhood. So, we group rows by neighbourhood and by taking the mean of the frequency of occurrence of each category. Now we can see the mean values in the feature column instead of the binary values.

```
manhattan_grouped = manhattan_onehot.groupby('Neighborhood').mean().reset_index()
manhattan_grouped.head()
```

	Neighborhood	Athletics & Sports	Bike Shop	Boxing Gym	Building	Chiropractor	Climbing Gym	Clothing Store	Club House	College Gym	...	Pool	Residential Building (Apartment / Condo)	Spa	Spiritual Center	Sporting Event	Tenr Co
0	Battery Park City	0.0	0.03125	0.0625	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
1	Carnegie Hill	0.0	0.00000	0.0000	0.019231	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
2	Central Harlem	0.0	0.00000	0.0000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
3	Chelsea	0.0	0.00000	0.0000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4	Chinatown	0.0	0.00000	0.0500	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

Even though there are several Gym Categories our friend 'Sam' is only interested in 'Climbing Gym','College Gym', 'Community Center', 'Gym', 'Gym / Fitness Center','Gym Pool','Outdoor Gym' and 'Track'. We can filter them.

```
manhattan_grouped=manhattan_grouped[['Neighborhood','Climbing Gym','College Gym', 'Community Center', 'Gym', 'Gym / Fitness
manhattan_grouped
```

	Neighborhood	Climbing Gym	College Gym	Community Center	Gym	Gym / Fitness Center	Gym Pool	Outdoor Gym	Track
0	Battery Park City	0.000000	0.0	0.000000	0.562500	0.281250	0.031250	0.00000	0.000000
1	Carnegie Hill	0.000000	0.0	0.019231	0.326923	0.365385	0.019231	0.00000	0.000000
2	Central Harlem	0.000000	0.0	0.000000	0.428571	0.357143	0.000000	0.00000	0.000000
3	Chelsea	0.000000	0.0	0.000000	0.108696	0.586957	0.021739	0.00000	0.000000
4	Chinatown	0.000000	0.0	0.000000	0.350000	0.350000	0.000000	0.00000	0.000000
5	Civic Center	0.020408	0.0	0.000000	0.193878	0.448980	0.010204	0.00000	0.000000
6	Clinton	0.000000	0.0	0.000000	0.440000	0.460000	0.000000	0.00000	0.020000
7	East Harlem	0.000000	0.0	0.000000	0.100000	0.400000	0.000000	0.00000	0.000000

But as I mentioned before this person is especially interested in 'Gym Pool's. He prefers 'Tracks' after that. We can try to assign the following simple weights-based approach to his preferences as follows.

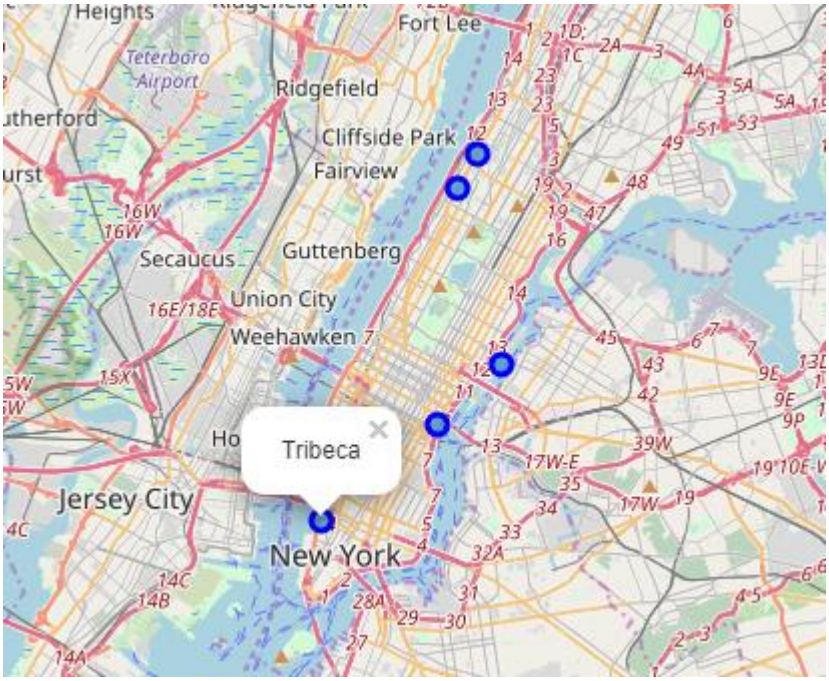
Gym Pool = 3 , Track = 2, Others = 1

We can define a new column called 'Score' and find the weight-based score for the items using a lambda function in python.

```
num_top_venues=5
manhattan_grouped[['Neighborhood','Score','Gym Pool']].sort_values('Score',ascending=False).head(num_top_venues)
```

	Neighborhood	Score	Gym Pool
32	Tribeca	0.090909	0.066667
28	Roosevelt Island	0.090909	0.000000
25	Morningside Heights	0.090909	0.200000
21	Manhattanville	0.090909	0.000000
33	Tudor City	0.088692	0.048780

- Visualize the best neighbourhoods.



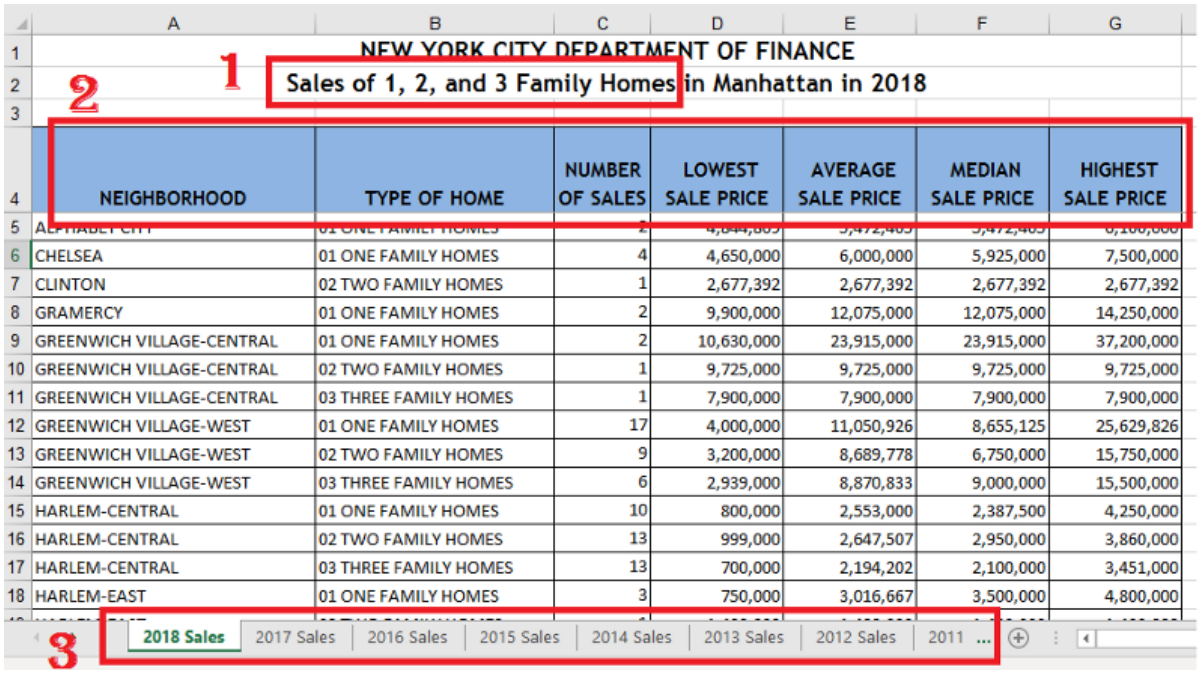
• Predicting the average price

We identified the Neighbourhood("**Tribeca**") based on the Gym Data. Now we can predict the price.

- Data Pre-processing

▪ Format Data

The downloaded Manhattan data set is in .xlsx format and it has separate sheets for each year while there are 6 features are in a sheet. Those are marked as Number 2 in the 'Manhattan Price Data Summary' image. The data set has data from 2005 to 2018.



	A	B	C	D	E	F	G
1	NEW YORK CITY DEPARTMENT OF FINANCE						
2	2 1 Sales of 1, 2, and 3 Family Homes in Manhattan in 2018						
3							
4	NEIGHBORHOOD	TYPE OF HOME	NUMBER OF SALES	LOWEST SALE PRICE	AVERAGE SALE PRICE	MEDIAN SALE PRICE	HIGHEST SALE PRICE
5	ALPHABET CITY	01 ONE FAMILY HOMES	2	4,844,805	5,472,403	5,472,403	6,100,000
6	CHELSEA	01 ONE FAMILY HOMES	4	4,650,000	6,000,000	5,925,000	7,500,000
7	CLINTON	02 TWO FAMILY HOMES	1	2,677,392	2,677,392	2,677,392	2,677,392
8	GRAMERCY	01 ONE FAMILY HOMES	2	9,900,000	12,075,000	12,075,000	14,250,000
9	GREENWICH VILLAGE-CENTRAL	01 ONE FAMILY HOMES	2	10,630,000	23,915,000	23,915,000	37,200,000
10	GREENWICH VILLAGE-CENTRAL	02 TWO FAMILY HOMES	1	9,725,000	9,725,000	9,725,000	9,725,000
11	GREENWICH VILLAGE-CENTRAL	03 THREE FAMILY HOMES	1	7,900,000	7,900,000	7,900,000	7,900,000
12	GREENWICH VILLAGE-WEST	01 ONE FAMILY HOMES	17	4,000,000	11,050,926	8,655,125	25,629,826
13	GREENWICH VILLAGE-WEST	02 TWO FAMILY HOMES	9	3,200,000	8,689,778	6,750,000	15,750,000
14	GREENWICH VILLAGE-WEST	03 THREE FAMILY HOMES	6	2,939,000	8,870,833	9,000,000	15,500,000
15	HARLEM-CENTRAL	01 ONE FAMILY HOMES	10	800,000	2,553,000	2,387,500	4,250,000
16	HARLEM-CENTRAL	02 TWO FAMILY HOMES	13	999,000	2,647,507	2,950,000	3,860,000
17	HARLEM-CENTRAL	03 THREE FAMILY HOMES	13	700,000	2,194,202	2,100,000	3,451,000
18	HARLEM-EAST	01 ONE FAMILY HOMES	3	750,000	3,016,667	3,500,000	4,800,000

3 2018 Sales | 2017 Sales | 2016 Sales | 2015 Sales | 2014 Sales | 2013 Sales | 2012 Sales | 2011 ...

The sheet was manually processed to one sheet adding a 'YEAR' column to the data.

```
df=pd.read_excel('input2.xls')
df.head()
```

	NEIGHBORHOOD	TYPE OF HOME	NUMBER OF SALES	LOWEST SALE PRICE	AVERAGE SALE PRICE	MEDIAN SALE PRICE	HIGHEST SALE PRICE	YEAR
0	ALPHABET CITY	02 TWO FAMILY HOMES	1	2675000	2675000.0	2675000.0	2675000	2005
1	ALPHABET CITY	03 THREE FAMILY HOMES	1	4200000	4200000.0	4200000.0	4200000	2005
2	CHELSEA	01 ONE FAMILY HOMES	2	217500	1958750.0	1958750.0	3700000	2005
3	CHELSEA	02 TWO FAMILY HOMES	3	3200000	3750000.0	3800000.0	4250000	2005
4	CHELSEA	03 THREE FAMILY HOMES	2	2800000	2900000.0	2900000.0	3000000	2005

- Understand the Data

```
df=pd.read_excel('input2.xls')
df.head()
```

	NEIGHBORHOOD	TYPE OF HOME	NUMBER OF SALES	LOWEST SALE PRICE	AVERAGE SALE PRICE	MEDIAN SALE PRICE	HIGHEST SALE PRICE	YEAR
0	ALPHABET CITY	02 TWO FAMILY HOMES	1	2675000	2675000.0	2675000.0	2675000	2005
1	ALPHABET CITY	03 THREE FAMILY HOMES	1	4200000	4200000.0	4200000.0	4200000	2005
2	CHELSEA	01 ONE FAMILY HOMES	2	217500	1958750.0	1958750.0	3700000	2005
3	CHELSEA	02 TWO FAMILY HOMES	3	3200000	3750000.0	3800000.0	4250000	2005
4	CHELSEA	03 THREE FAMILY HOMES	2	2800000	2900000.0	2900000.0	3000000	2005

The unnecessary augmented data were removed from the data.

```
df.dropna(inplace=True)
df.drop(['LOWEST SALE PRICE', 'MEDIAN SALE PRICE', 'HIGHEST SALE PRICE'],inplace=True, axis = 1)
df.head()
```

	NEIGHBORHOOD	TYPE OF HOME	NUMBER OF SALES	AVERAGE SALE PRICE	YEAR
0	ALPHABET CITY	02 TWO FAMILY HOMES	1	2675000.0	2005
1	ALPHABET CITY	03 THREE FAMILY HOMES	1	4200000.0	2005
2	CHELSEA	01 ONE FAMILY HOMES	2	1958750.0	2005
3	CHELSEA	02 TWO FAMILY HOMES	3	3750000.0	2005
4	CHELSEA	03 THREE FAMILY HOMES	2	2900000.0	2005

- Analysing the 'Target' variable.

we can see the summarized description of the target as follows.

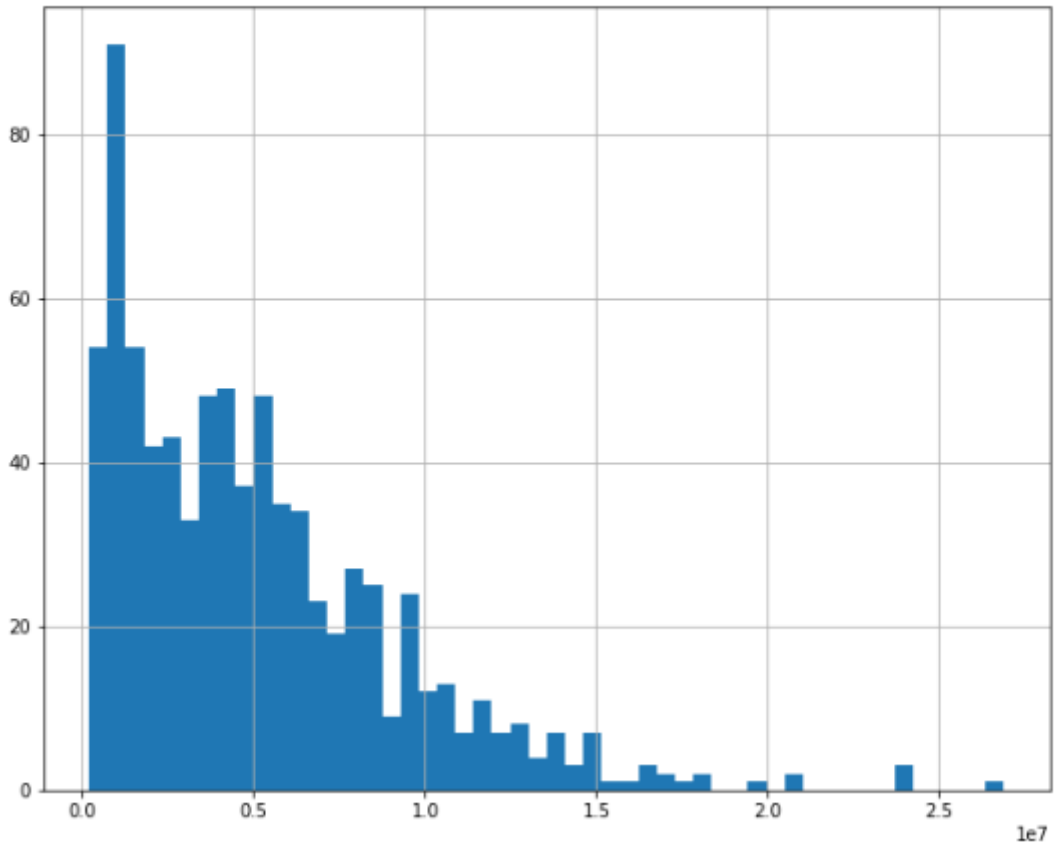
```
df['AVERAGE SALE PRICE'].describe()
```

```
count    7.910000e+02
mean     5.111178e+06
std      4.097849e+06
min      2.000000e+05
25%      1.782600e+06
50%      4.250000e+06
75%      7.157252e+06
max      2.689300e+07
Name: AVERAGE SALE PRICE, dtype: float64
```

Visualizing the statistics helps us understand better. The following histogram shows average prices in X-axis and its count/ frequency on the y-axis. (We have binned X values to 50 bins)

```
1 df['AVERAGE SALE PRICE'].hist(bins = 50,figsize=(10,8))
```

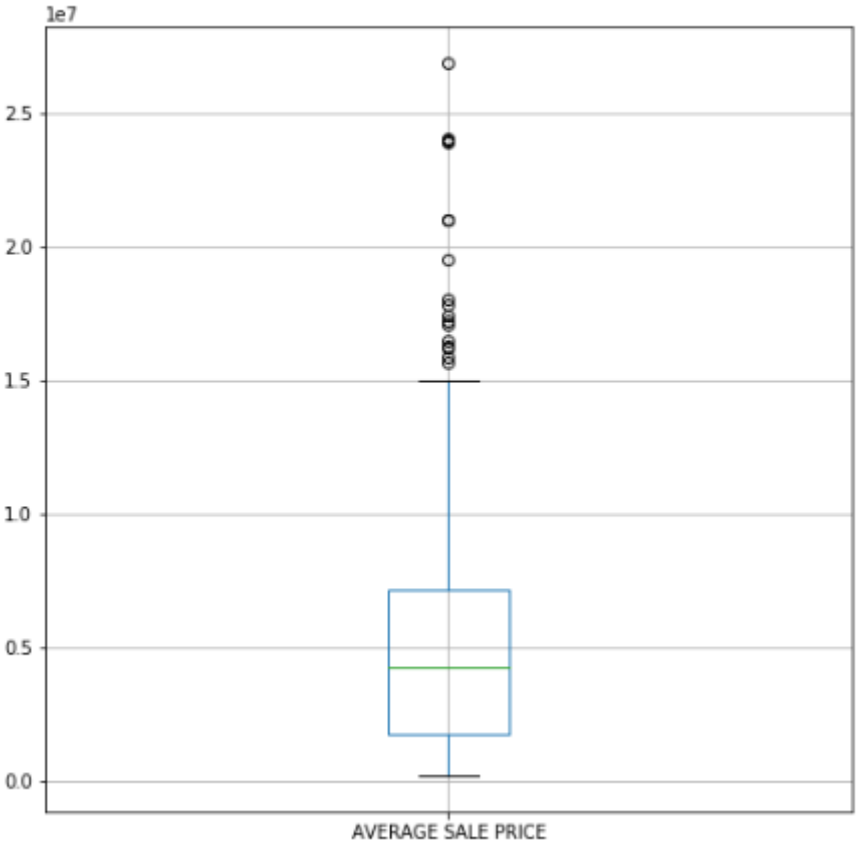
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1381628ba8>
```



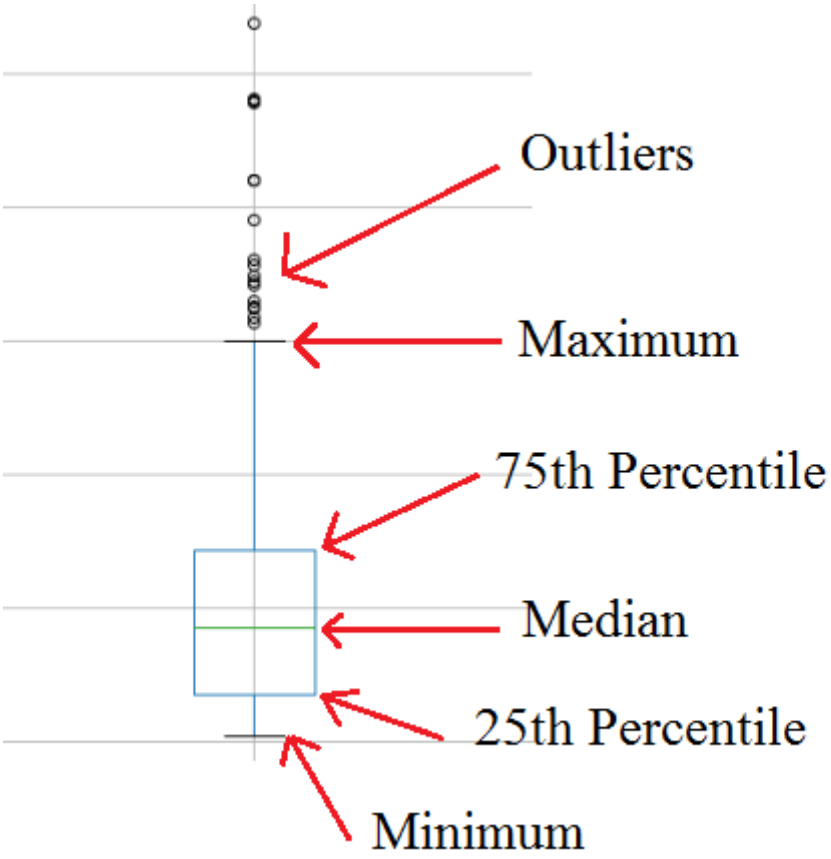
We can see that most of the values are distributed between 0–1.5 (1^{007}). We can use the box plot analysis to identify outliers more clearly.

```
1 df.boxplot(column='AVERAGE SALE PRICE',figsize=(8,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f138053d3c8>
```



Box plot is nothing but a simple representation of the distribution of the data that you have. The following image describes the features in the box plot.



The range between 75th(Q1) and 25th (Q3) percentile is considered as the Interquartile Range and the maximum(possible) and minimum values are calculated as below in this framework.

- `IQR = Q3 - Q1`
- `Maximum = Q3 + 1.5*IQR`
- `Minimum = Q1 - 1.5*IQR`

There can be many reasons to cause values above the Maximum and below the Minimum. These observations are significantly different from others. Removing those outliers increases the accuracy of the models.

```
:  1 # Remove Outliers
   2 Q1 = df['AVERAGE SALE PRICE'].quantile(0.25)
   3 Q3 = df['AVERAGE SALE PRICE'].quantile(0.75)
   4 IQR = Q3 - Q1    #IQR is interquartile range.
   5
   6 outliers=df[(df['AVERAGE SALE PRICE'] < (Q1 - 1.5 * IQR)) | (df['AVERAGE SALE PRICE'] > (Q3 + 1.5 *IQR))]
   7 print("Removed ",len(outliers)," outliers")
   8
   9 df= df[(df['AVERAGE SALE PRICE'] >= Q1 - 1.5 * IQR) & (df['AVERAGE SALE PRICE'] <= Q3 + 1.5 *IQR)]
```

```
Removed  17  outliers
```

▪ Explore ‘’Neighbourhood’ Feature

We can list the unique neighbourhoods as follows.

```
neig_len=len(df.NEIGHBORHOOD.unique())
df.NEIGHBORHOOD.unique()
```

```
array(['ALPHABET CITY            ', 'CHELSEA                 ',
       'CLINTON                  ', 'EAST VILLAGE            ',
       'GRAMERCY                 ', 'GREENWICH VILLAGE-CENTRAL',
       'GREENWICH VILLAGE-WEST   ', 'HARLEM-CENTRAL          ',
       'HARLEM-EAST              ', 'HARLEM-UPPER            ',
       'INWOOD                   ', 'KIPS BAY                ',
       'LITTLE ITALY             ', 'MANHATTAN VALLEY        ',
       'MIDTOWN EAST             ', 'MURRAY HILL             ',
       'SOHO                     ', 'TRIBECA                 ',
       'UPPER EAST SIDE (59-79)  ', 'UPPER EAST SIDE (79-96) ',
       'UPPER WEST SIDE (59-79)  ', 'UPPER WEST SIDE (79-96) ',
       'UPPER WEST SIDE (96-116) ', 'WASHINGTON HEIGHTS LOWER',
       'WASHINGTON HEIGHTS UPPER', 'CIVIC CENTER             ',
       'LOWER EAST SIDE          ', 'MIDTOWN WEST            ',
       'CHINATOWN                ', 'FASHION                 ',
       'HARLEM-WEST              ', 'SOUTHBRIDGE             ',
       'FLATIRON                 ', 'JAVITS CENTER           ',
       'ALPHABET CITY', 'CHELSEA', 'EAST VILLAGE', 'GRAMERCY',
       'GREENWICH VILLAGE-WEST', 'HARLEM-CENTRAL', 'HARLEM-EAST',
       'HARLEM-UPPER', 'INWOOD', 'KIPS BAY', 'MIDTOWN EAST',
       'MURRAY HILL', 'SOHO', 'SOUTHBRIDGE', 'TRIBECA',
       'UPPER EAST SIDE (59-79)', 'UPPER EAST SIDE (79-96)',
       'UPPER WEST SIDE (79-96)', 'UPPER WEST SIDE (96-116)',
       'WASHINGTON HEIGHTS LOWER', 'WASHINGTON HEIGHTS UPPER'],
      dtype=object)
```

The highlighted items are the same but unwanted white spaces have resulted in multiple unique values for the same neighbourhoods. It was required to remove those.

```
df.NEIGHBORHOOD=df.NEIGHBORHOOD.str.strip()
print(" Old Length ",neig_len," New Length ",len(df.NEIGHBORHOOD.unique()))
```

```
Old Length  55  New Length  34
```

```
df.NEIGHBORHOOD.unique()
```

```
array(['ALPHABET CITY', 'CHELSEA', 'CLINTON', 'EAST VILLAGE', 'GRAMERCY',
       'GREENWICH VILLAGE-CENTRAL', 'GREENWICH VILLAGE-WEST',
       'HARLEM-CENTRAL', 'HARLEM-EAST', 'HARLEM-UPPER', 'INWOOD',
       'KIPS BAY', 'LITTLE ITALY', 'MANHATTAN VALLEY', 'MIDTOWN EAST',
       'MURRAY HILL', 'SOHO', 'TRIBECA', 'UPPER EAST SIDE (59-79)',
       'UPPER EAST SIDE (79-96)', 'UPPER WEST SIDE (59-79)',
       'UPPER WEST SIDE (79-96)', 'UPPER WEST SIDE (96-116)',
       'WASHINGTON HEIGHTS LOWER', 'WASHINGTON HEIGHTS UPPER',
       'CIVIC CENTER', 'LOWER EAST SIDE', 'MIDTOWN WEST', 'CHINATOWN',
       'FASHION', 'HARLEM-WEST', 'SOUTHBRIDGE', 'FLATIRON',
       'JAVITS CENTER'], dtype=object)
```

Now the number of neighbourhoods have been reduced to 34. But the neighbourhood field has categorical features. It is required represent this in a numerical format in order to analyse further. It is encoded using One Hot Encoding on Neighbourhood data to get the numerical representation.

```
one_hot_features = ['NEIGHBORHOOD']
one_hot_encoded = pd.get_dummies(df[one_hot_features])
one_hot_encoded.info(verbose=True, memory_usage=True, null_counts=True)

# Replacing categorical columns with dummies
df = df.drop(one_hot_features,axis=1)
df = pd.concat([df, one_hot_encoded] ,axis=1)
```

- Explore '' TYPE OF HOME' Feature

```
df['TYPE OF HOME'].unique()
```

```
array(['02 TWO FAMILY HOMES                          ',
       '03 THREE FAMILY HOMES                        ',
       '01 ONE FAMILY HOMES                          ',
       '01 ONE FAMILY HOMES     ', '02 TWO FAMILY HOMES     ',
       '03 THREE FAMILY HOMES   ', '01  ONE FAMILY HOMES  ',
       '03  THREE FAMILY HOMES', '02  TWO FAMILY HOMES  ',
       '01 ONE FAMILY DWELLINGS', '03 THREE FAMILY DWELLINGS',
       '02 TWO FAMILY DWELLINGS'], dtype=object)
```

Again, we can see the same issue as in neighborhood feature. WHITE SPACES! Also, keep in mind that the price is positively increasing when the 'Family Number' is increasing. With that in mind, We can encode this field using Label Encoding.

```
df['TYPE OF HOME']=df.apply(lambda x: int(x['TYPE OF HOME'].strip().split(" ")[0]),axis=1 )
df['TYPE OF HOME'].unique()
```

```
array([2, 3, 1])
```

We have encoded the data as follows.

- 01 One FAMILY HOMES - 1
- 02 Two FAMILY HOMES - 2
- 03 Three FAMILY HOMES – 3

Data Scaling

Data Scaling helps to train faster, reduce overfitting and improve the accuracies of the machine learning models. This is considered as an important pre-processing step in Deep learning. There are 2 main scaling methods.

1. Data Normalization
2. Data Standardization.

Both Normalization and Standardization techniques were used on the data.
input data was normalized using MinMaxScaler from scikit-learn library. 'YEAR', 'TYPE OF HOME' and 'NUMBER OF SALES' features were scaled into 0–1 range.

```

inputScaler = MinMaxScaler(feature_range=(0, 1))
inputScaler.fit(df[['YEAR','TYPE OF HOME','NUMBER OF SALES']])
df[['YEAR','TYPE OF HOME','NUMBER OF SALES']]=inputScaler.transform(
                                                df[['YEAR','TYPE OF HOME','NUMBER OF SALES']]
                                            )

```

The target variable is Standardized using StandardScaler() in the scikit-learn library.

```
#Lets standarscale the AVERAGE SALE PRICE feild

targetScaler = StandardScaler()
df["AVERAGE SALE PRICE"] = pd.to_numeric(df["AVERAGE SALE PRICE"])

targetScaler.fit(df[['AVERAGE SALE PRICE']])
df[['AVERAGE SALE PRICE']]=targetScaler.transform(df[['AVERAGE SALE PRICE']])

#Rename AVARAGE SALE PRICE to PRICE
df.rename(columns={'AVERAGE SALE PRICE': 'PRICE'},inplace=True)

```

○ Train/ Test Split

The data set was divided into two sets of train/test data. The first data set (Set 1)is designed to test the predictability of this specific problem with the given data features. Set 2 has kept 2018 data for validation and split the other data 7/3 ration for training and testing.

Set 1

```
y_df = df['PRICE']
X_df = df.drop('PRICE', axis=1)
X_train ,X_test, y_train , y_test = train_test_split(X_df , y_df , test_size = 0.3 , random_state =34)
```

Set 2

Scaled value 1 represents the maximum in the year feature and that is the year 2018. We keep that for validation purposes.

```
#with Validation
V_df=df[df['YEAR']==1] #2018 data
V_X=V_df.drop('PRICE', axis=1)
V_Y=V_df.drop('PRICE', axis=1)

#Train and test
T_df=df[df['YEAR']<1]
y_df1 = T_df['PRICE']
X_df1 = T_df.drop('PRICE', axis=1)
X_train1 ,X_test1, y_train1 , y_test1 = train_test_split(X_df1 , y_df1 , test_size = 0.3 , random_state =34)

```

○ Modelling

It's time to build the prediction models. I tried linear and nonlinear regression models for prediction. Please note the models were configured to their default configurations and evaluated using **"Root Mean Squared Error"** (Note that Target is standardized). The results have been explained in the results section.

Results

Liner Models

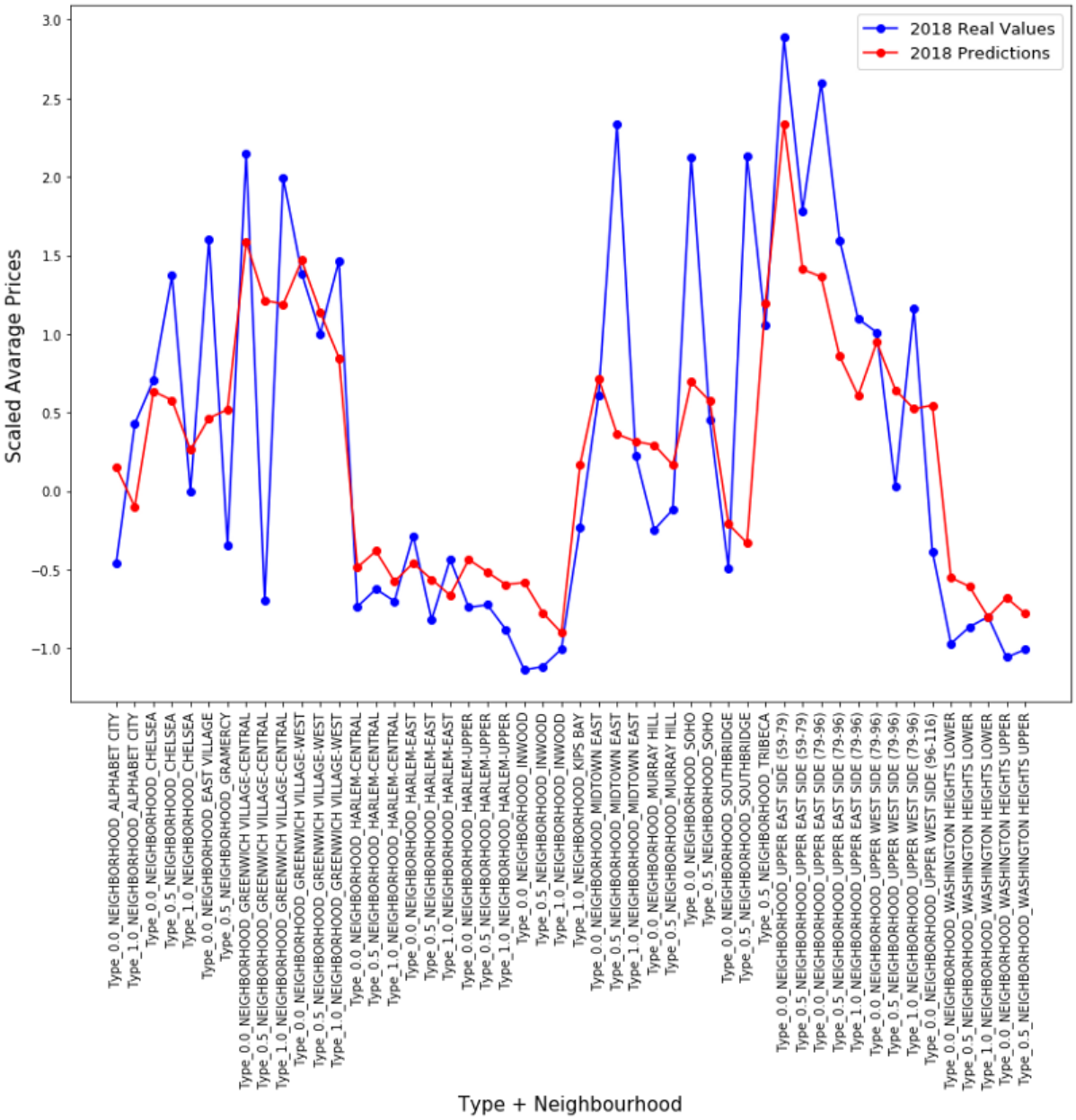
Model	RMS
Linear Regression	2449866754.0338902
Lasso Linear Regression	1.0075
Ridge Regression	0.57
Elastic Net Regression	1.00
Huber Regression	0.58
Lasso Lars Linear Regression	1.00
Passive Aggressive Regression	1.4
Stochastic Gradient Descent Regression	0.6

Nonlinear Models

Model	RMS
k-Nearest Neighbors	0.69
Classification and Regression Trees	0.80
Extra Tree	0.81
Support Vector Regression	0.71

In addition to the above models, I tried a two-layer (with no hidden layers) Neural Network for our regression problem. It resulted an RMS value of 0.563 so far, the best!

We can try to visualize the prediction for the year 2018 vs the actual average prices.



It seems the model has the capability of following the general pattern in the target while missing the spikes. That could be a reason to have higher RMS values.

Discussion

Trying out each possible model and comparing the results is not a good practice. It's always better to have a theoretical background and to know why this model performs better in the specific type of problem (Especially in deep learning). Yet our goal here is to find the best model without spending much time. That's why I used a brute force approach for this problem.

Please note that I did not apply any Hyper Parameter Optimization techniques on the NN model (e.g. Grid Search/ Random Search, Trial & Error or Analysing Literature) or any techniques to stop overfitting (e.g. Cross-validation). Our Aim here is to see the possibility of the predicting target value using the given features.

We can also try an AutoML approach (brute force) to find the best model and hyperparameters without performing an exhaustive manual method. I will leave it as a future work.

Conclusion

We previously modelled the data with YEAR, TYPE OF HOME and NUMBER OF SALES. But when we are predicting the average price for a new year we only have YEAR and TYPE OF HOME that we are looking for. We don't have the NUMBER OF SALES. So lets train the model with YEAR AND TYPE OF HOME features. We will be using the model with the best accuracy assuming that it still can predict better even one feature is reduced.

The imaginary person in our problem is looking for a 'One Family Home'. So, our input variables will be,

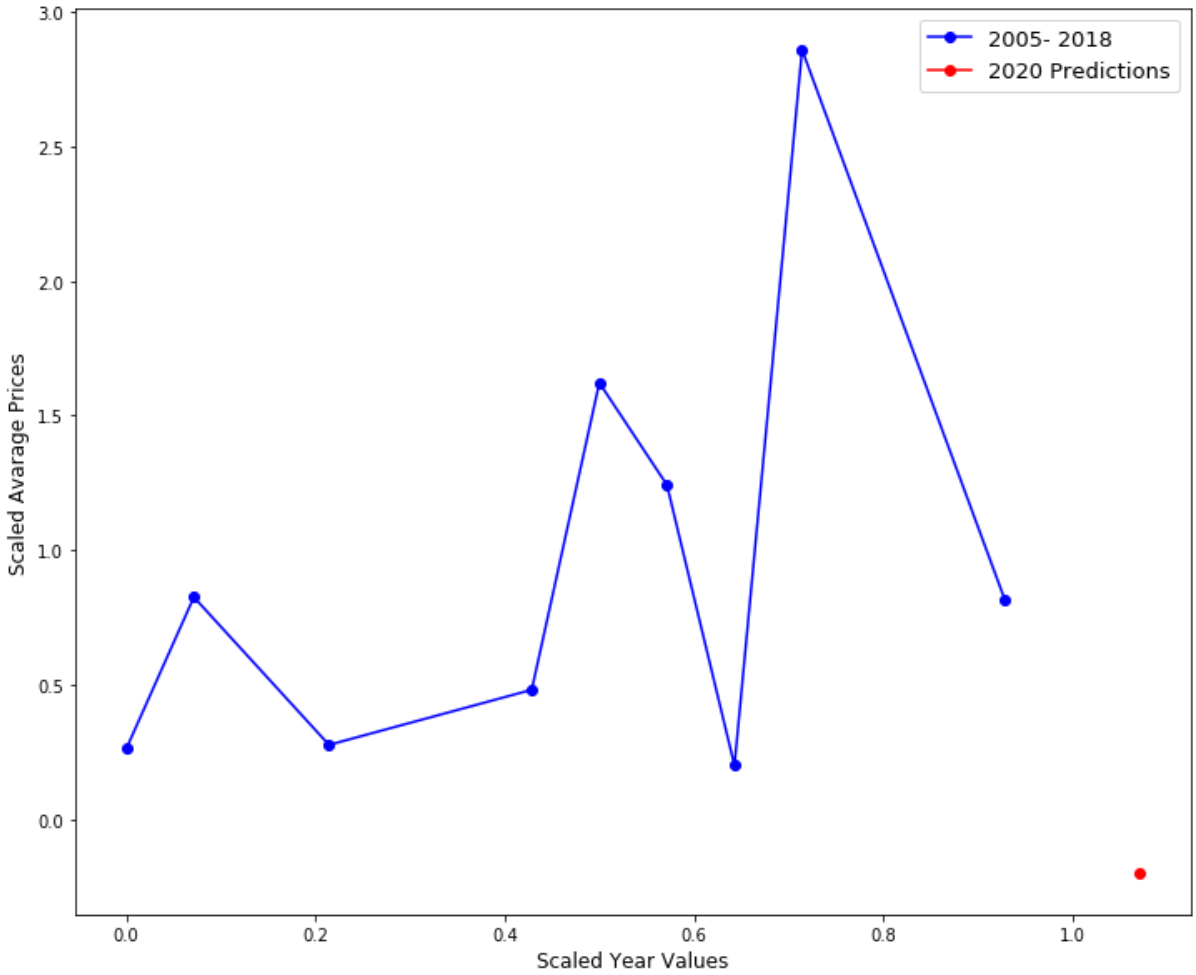
- YEAR = 2020
- TYPE OF HOME = 1
- NEIGHBORHOOD= 'Tribeca'

We should transform the first two features to their scaled values and apply One Hot encoding to the 'NEIGHBORHOOD' feature. (using Scikit-learn pipelines for the process is the better option. but doing it manually as we don't have to worry about the reproducibility ATM)

```
nhoods=['NEIGHBORHOOD_ALPHABET CITY', 'NEIGHBORHOOD_CHELSEA',
        'NEIGHBORHOOD_CHINATOWN', 'NEIGHBORHOOD_CIVIC CENTER', 'NEIGHBORHOOD_CLINTON',
        'NEIGHBORHOOD_EAST VILLAGE', 'NEIGHBORHOOD_FASHION', 'NEIGHBORHOOD_FLATIRON',
        'NEIGHBORHOOD_GRAMERCY', 'NEIGHBORHOOD_GREENWICH VILLAGE-CENTRAL',
        'NEIGHBORHOOD_GREENWICH VILLAGE-WEST', 'NEIGHBORHOOD_HARLEM-CENTRAL',
        'NEIGHBORHOOD_HARLEM-EAST', 'NEIGHBORHOOD_HARLEM-UPPER', 'NEIGHBORHOOD_HARLEM-WEST',
        'NEIGHBORHOOD_INWOOD', 'NEIGHBORHOOD_JAVITS CENTER', 'NEIGHBORHOOD_KIPS BAY',
        'NEIGHBORHOOD_LITTLE ITALY', 'NEIGHBORHOOD_LOWER EAST SIDE', 'NEIGHBORHOOD_MANHATTAN VALLEY',
        'NEIGHBORHOOD_MIDTOWN EAST', 'NEIGHBORHOOD_MIDTOWN WEST', 'NEIGHBORHOOD_MURRAY HILL',
        'NEIGHBORHOOD_SOHO', 'NEIGHBORHOOD_SOUTHBRIDGE', 'NEIGHBORHOOD_TRIBECA',
        'NEIGHBORHOOD_UPPER EAST SIDE (59-79)', 'NEIGHBORHOOD_UPPER EAST SIDE (79-96)',
        'NEIGHBORHOOD_UPPER WEST SIDE (59-79)', 'NEIGHBORHOOD_UPPER WEST SIDE (79-96)',
        'NEIGHBORHOOD_UPPER WEST SIDE (96-116)', 'NEIGHBORHOOD_WASHINGTON HEIGHTS LOWER',
        'NEIGHBORHOOD_WASHINGTON HEIGHTS UPPER']
nhoods_input=listofzeros = [0] * len(nhoods)
nhoods_input[nhoods.index('NEIGHBORHOOD_TRIBECA')]=1
```

```
input = [2020,1,0] # 2020 for year 2020 and 1 for single home,
# dummy number of sales feature is added ans will be removed later
YT_input=inputScaler.transform([input])
input=YT_input[0][:-1]
input=list(input) + nhoods_input
input=np.asarray(input, dtype=np.float32).reshape(1,36)
```

The prediction shows the average price for a single home in 'Tribeca' as **$4093886.2.** We can compare this value with the values of the past years.



References

[1]https://developer.foursquare.com/

[2]https://www1.nyc.gov/assets/finance/downloads/pdf/rolling_sales/rollingsales_manhattan.xls

[3] https://www1.nyc.gov/site/finance/taxes/property-annualized-sales-update.page

[4] https://cocl.us/new_york_dataset

[5]https://medium.com/r/?url=https%3A%2F%2Fdeveloper.foursquare.com%2Fdocs%2Fbuild-with-foursquare%2Fcategories

[6]https://medium.com/r/?url=https%3A%2F%2Ftowardsdatascience.com%2Fchoosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b