MITA CAPSTONE PROJECT

Boston House Price Prediction

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Program: Masters in Information Technology And Analytics

Semester – Spring 2021

Submitted By: Akshata Chavan

RUID: 190001210

Under Guidence of: Prof. Michail Xyntarakis

INDEX:

1. Introduction:

1.1 Context
This Capstone Project has been done as part of Masters in Information Technology and Analytics at Rutgers Business School, Newark and mentored by Prof. Michail Xyntarakis. The Project was completed in the duration of three months including deciding project topic, research, learning, development and reporting.

1.2 Motivation
The project represent the interest in the field of Machine Learning, Data analytics and Visualization. The skills gained in the curriculum over the past semesters have been applied practically which helped in enhancing the existing theoretical knowledge.

1.3 Idea
Residing in Boston currently, this topic immensely interested me to know better about the city and its residential facts. Also, when the project was completed I was able to verify the results in the real world scenario which is accurate to great extent.

1.4 Objective
The main objective is to evaluate the performance of various models used and predict the price of the houses in the suburbs of Boston, Massachusetts. The project is created in two parts using Machine Learning Algorithms in Python (Jupyter Notebook) and through Tableau Visualizations. In the initial part of the project, models trained and tested on the dataset can be seen as a good fit and thus could be used to make predictions about the monetary value of the houses. In the later part, various charts and representations have been used to get a better idea of various factors that effect the house price in Boston. When an individual plans to buy a house he considers various aspects like price, the number of rooms, the distance from major locations, educational institutes, crime rate, other residents in that area etc. Here, multiple factors are used to predict the price of the house. This model and visualizations can be very useful to individuals who are new to the city and want to explore/buy a house.

2. Dataset

2.1 Getting the dataset

The Dataset obtained is the corrected version of original Boston dataset. This is the updated dataset and contains the Harrison and Rubinfield data corrected for minor errors in the latitude and longitude. Also, the town column has been added in the dataset. The corrected dataset was in a txt format and had to be converted to xlsx format.

```
OBS. TOWN TOWN# TRACT LON LAT MEDV CMEDV CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT
1 Nahant 0 2011 -70.955000 42.255000 24.0 24.0 0.00632 18.0 2.31 0 0.538 6.575 65.2 4.0900 1 296 15.3 396.90 4.98
2 Swampscott 1 2021 -70.950000 42.287500 21.6 21.6 0.02731 0.0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8 396.90 9.14
3 Swampscott 1 2022 -70.936000 42.283000 34.7 34.7 0.02729 0.0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8 392.83 4.03
4 Marblehead 2 2031 -70.928000 42.293000 33.4 33.4 0.03237 0.0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7 394.63 2.94
5 Marblehead 2 2032 -70.922000 42.298000 36.2 36.2 0.06905 0.0 2.18 0 0.458 7.147 54.2 6.0622 3 222 18.7 396.90 5.33
6 Marblehead 2 2033 -70.916500 42.304000 28.7 28.7 0.02985 0.0 2.18 0 0.458 6.430 58.7 6.0622 3 222 18.7 394.12 5.21
7 Salem 3 2041 -70.936000 42.297000 22.9 22.9 0.08829 12.5 7.87 0 0.524 6.012 66.6 5.5605 5 311 15.2 395.60 12.43
8 Salem 3 2042 -70.937500 42.310000 27.1 22.1 0.14455 12.5 7.87 0 0.524 6.172 96.1 5.9505 5 311 15.2 396.90 19.15
9 Salem 3 2043 -70.933000 42.312000 16.5 16.5 0.21124 12.5 7.87 0 0.524 5.631 100.0 6.0821 5 311 15.2 386.63 29.93
10 Salem 3 2044 -70.929000 42.316000 18.9 18.9 0.17004 12.5 7.87 0 0.524 6.004 85.9 6.5921 5 311 15.2 386.71 17.10
11 Salem 3 2045 -70.935000 42.316000 15.0 15.0 0.22489 12.5 7.87 0 0.524 6.377 94.3 6.3467 5 311 15.2 392.52 20.45
12 Salem 3 2046 -70.944000 42.317000 18.9 18.9 0.11747 12.5 7.87 0 0.524 6.009 82.9 6.2267 5 311 15.2 396.90 13.27
13 Salem 3 2047 -70.951000 42.306000 21.7 21.7 0.09378 12.5 7.87 0 0.524 5.889 39.0 5.4509 5 311 15.2 390.50 15.71
14 Lynn 4 2051 -70.964500 42.292000 20.4 20.4 0.62976 0.0 8.14 0 0.538 5.949 61.8 4.7075 4 307 21.0 396.90 8.26
15 Lynn 4 2052 -70.972000 42.287000 18.2 18.2 0.63796 0.0 8.14 0 0.538 6.096 84.5 4.4619 4 307 21.0 380.02 10.26
16 Lynn 4 2053 -70.976500 42.294000 19.9 19.9 0.62739 0.0 8.14 0 0.538 5.834 56.5 4.4986 4 307 21.0 395.62 8.47
17 Lynn 4 2054 -70.987000 42.298500 23.1 23.1 1.05393 0.0 8.14 0 0.538 5.935 29.3 4.4986 4 307 21.0 386.85 6.58
18 Lynn 4 2055 -70.978000 42.285000 17.5 17.5 0.78420 0.0 8.14 0 0.538 5.990 81.7 4.2579 4 307 21.0 386.75 14.67
19 Lynn 4 2056 -70.992500 42.282500 20.2 20.2 0.80271 0.0 8.14 0 0.538 5.456 36.6 3.7965 4 307 21.0 288.99 11.69
20 Lynn 4 2057 -70.988000 42.277600 18.2 18.2 0.72580 0.0 8.14 0 0.538 5.727 69.5 3.7965 4 307 21.0 390.95 11.28
21 Lynn 4 2058 -70.983500 42.277000 13.6 13.6 1.25179 0.0 8.14 0 0.538 5.570 98.1 3.7979 4 307 21.0 376.57 21.02
22 Lynn 4 2059 -70.982000 42.281000 19.6 19.6 0.85204 0.0 8.14 0 0.538 5.965 89.2 4.0123 4 307 21.0 392.53 13.83
23 Lynn 4 2060 -70.977500 42.279000 15.2 15.2 1.23247 0.0 8.14 0 0.538 6.142 91.7 3.9769 4 307 21.0 396.90 18.72
24 Lynn 4 2061 -70.973000 42.279000 14.5 14.5 0.98843 0.0 8.14 0 0.538 5.813 100.0 4.0952 4 307 21.0 394.54 19.88
25 Lynn 4 2062 -70.969300 42.281600 15.6 15.6 0.75026 0.0 8.14 0 0.538 5.924 94.1 4.3996 4 307 21.0 394.33 16.30
26 Lynn 4 2063 -70.964000 42.284000 13.9 13.9 0.84054 0.0 8.14 0 0.538 5.599 85.7 4.4546 4 307 21.0 303.42 16.51
27 Lynn 4 2064 -70.959700 42.287000 16.6 16.6 0.67191 0.0 8.14 0 0.538 5.813 90.3 4.6820 4 307 21.0 376.88 14.81
28 Lynn 4 2065 -70.959700 42.282500 14.8 14.8 0.95577 0.0 8.14 0 0.538 6.047 88.8 4.4534 4 307 21.0 306.38 17.28
29 Lynn 4 2066 -70.957000 42.280000 18.4 18.4 0.77299 0.0 8.14 0 0.538 6.495 94.4 4.4547 4 307 21.0 387.94 12.80
30 Lynn 4 2067 -70.951000 42.278000 21.0 21.0 1.00245 0.0 8.14 0 0.538 6.674 87.3 4.2390 4 307 21.0 380.23 11.98
31 Lynn 4 2068 -70.967000 42.279000 12.7 12.7 1.13081 0.0 8.14 0 0.538 5.713 94.1 4.2330 4 307 21.0 360.17 22.60
32 Lynn 4 2069 -70.964000 42.276500 14.5 14.5 1.35472 0.0 8.14 0 0.538 6.072 100.0 4.1750 4 307 21.0 376.73 13.04
33 Lynn 4 2070 -70.972000 42.276500 13.2 13.2 1.38799 0.0 8.14 0 0.538 5.950 82.0 3.9900 4 307 21.0 232.60 27.71
34 Lynn 4 2071 -70.979500 42.276000 13.1 13.1 1.15172 0.0 8.14 0 0.538 5.701 95.0 3.7872 4 307 21.0 358.77 18.35
35 Lynn 4 2072 -70.977500 42.272000 13.5 13.5 1.61282 0.0 8.14 0 0.538 6.096 96.9 3.7598 4 307 21.0 248.31 20.34
36 Sargus 5 2081 -71.000000 42.270000 18.9 18.9 0.06417 0.0 5.96 0 0.499 5.933 68.2 3.3603 5 279 19.2 396.90 9.68
37 Sargus 5 2082 -71.008000 42.274500 20.0 20.0 0.09744 0.0 5.96 0 0.499 5.841 61.4 3.3779 5 279 19.2 377.56 11.41
38 Sargus 5 2083 -71.004000 42.286500 21.0 21.0 0.08014 0.0 5.96 0 0.499 5.850 41.5 3.9342 5 279 19.2 396.90 8.77
39 Sargus 5 2084 -71.020000 42.287500 24.7 24.2 0.17505 0.0 5.96 0 0.499 5.966 30.2 3.8473 5 279 19.2 393.43 10.13
40 Lynnfield 6 2091 -71.013000 42.313000 30.8 30.8 0.02763 75.0 2.95 0 0.428 6.595 21.8 5.4011 3 252 18.3 395.63 4.32
```

Figure 2.1: The original dataset

2.2. Dataset Description

The dataset used in this project contains 506 rows and 20 columns. The columns in the dataset are as follows:

CRIM - per capita crime rate by town
ZN - proportion of residential land zoned for lots over 25,000 sq. ft.
INDUS - proportion of non-retail business acres per town
CHAS - Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX - nitric oxides concentration (parts per 10 million)
RM - average number of rooms per dwelling
AGE - proportion of owner-occupied units built prior to 1940
DIS - weighted distances to five Boston employment centers
RAD - index of accessibility to radial highways
TAX - full-value property-tax rate per 10,000usd
PTRATIO - pupil-teacher ratio by town
B - 1000(Bk - 0.63)^2 - where Bk is the proportion of blacks by town
LSTAT - % lower status of the population

```
In [4]: df.head(5)
```

Out[4]:

	OBS.	TOWN	TOWN#	TRACT	LON	LAT	MEDV	CMEDV	CRIM	ZN	...	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
0	1	Nahant	0	2011	-70.955	42.2550	24.0	24.0	0.00632	18.0	...	0	0.538	6.575	65.2	4.0900	1	296	15.3	396
1	2	Swampscott	1	2021	-70.950	42.2875	21.6	21.6	0.02731	0.0	...	0	0.469	6.421	78.9	4.9671	2	242	17.8	396
2	3	Swampscott	1	2022	-70.936	42.2830	34.7	34.7	0.02729	0.0	...	0	0.469	7.185	61.1	4.9671	2	242	17.8	392
3	4	Marblehead	2	2031	-70.928	42.2930	33.4	33.4	0.03237	0.0	...	0	0.458	6.998	45.8	6.0622	3	222	18.7	394
4	5	Marblehead	2	2032	-70.922	42.2980	36.2	36.2	0.06905	0.0	...	0	0.458	7.147	54.2	6.0622	3	222	18.7	396

Figure 2.2. : Viewing the dataset

3. Data Exploration and Cleaning

Deriving information about the dataset through exploration and checking for unnecessary columns, null values, repeated values etc.

```
In [9]:  #datatype info
         df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 20 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   TOWN      506 non-null    object
 1   TOWN#     506 non-null    int64
 2   TRACT     506 non-null    int64
 3   LON       506 non-null    float64
 4   LAT       506 non-null    float64
 5   MEDV      506 non-null    float64
 6   CMEDV     506 non-null    float64
 7   CRIM      506 non-null    float64
 8   ZN        506 non-null    float64
 9   INDUS     506 non-null    float64
 10  CHAS      506 non-null    int64
 11  NOX       506 non-null    float64
 12  RM        506 non-null    float64
 13  AGE       506 non-null    float64
 14  DIS       506 non-null    float64
 15  RAD       506 non-null    int64
 16  TAX       506 non-null    int64
 17  PTRATIO   506 non-null    float64
 18  B         506 non-null    float64
 19  LSTAT     506 non-null    float64
dtypes: float64(14), int64(5), object(1)
memory usage: 79.2+ KB
```

Figure 3.1: Exploring the datatype information

```
In [10]:  #checking null values
          df.isnull().sum()

Out[10]:  TOWN       0
          TOWN#      0
          TRACT      0
          LON        0
          LAT        0
          MEDV       0
          CMEDV      0
          CRIM       0
          ZN         0
          INDUS      0
          CHAS       0
          NOX        0
          RM         0
          AGE        0
          DIS        0
          RAD        0
          TAX        0
          PTRATIO    0
          B          0
          LSTAT      0
          dtype: int64
```

Figure 3.2: Checking for null values

```
In [5]:  df.drop(columns=['OBS.'], axis=0, inplace=True)
         df.head()

Out[5]:
```

	TOWN	TOWN#	TRACT	LON	LAT	MEDV	CMEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	Nahant	0	2011	-70.955	42.2550	24.0	24.0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.9
1	Swampscott	1	2021	-70.950	42.2875	21.6	21.6	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9
2	Swampscott	1	2022	-70.936	42.2830	34.7	34.7	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.8
3	Marblehead	2	2031	-70.928	42.2930	33.4	33.4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.6
4	Marblehead	2	2032	-70.922	42.2980	36.2	36.2	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9

Figure 3.3: Dropping unnecessary columns

4. Exploratory Data Analysis

Exploratory Data Analysis (EDA) is important part of any data analysis. EDA helps to generate questions about the data, visualize, transform and summarize the data. It makes easier to find patterns, spot anomalies and check assumptions. EDA was originally developed by American Mathematician John Turkey in the 1970s and these techniques are widely used in data discovery process till date. The specific statistical techniques to perform with EDA tools include Clustering and dimension reduction, Univariate visualization, Bivariate visualizations and summary statistics, Multivariate visualizations, K-means Clustering etc.

The EDA performed on thr Boston Dataset can be seen in the figures below.

In [16]:

```
#Create box plots
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

sns.boxplot(y= 'CRIM', data=df, ax=ax[0])
sns.boxplot(y= 'ZN', data=df, ax=ax[1])
sns.boxplot(y= 'INDUS', data=df, ax=ax[2])
sns.boxplot(y= 'CHAS', data=df, ax=ax[3])
sns.boxplot(y= 'NOX', data=df, ax=ax[4])
sns.boxplot(y= 'RM', data=df, ax=ax[5])
sns.boxplot(y= 'AGE', data=df, ax=ax[6])
sns.boxplot(y= 'DIS', data=df, ax=ax[7])
sns.boxplot(y= 'RAD', data=df, ax=ax[8])
sns.boxplot(y= 'TAX', data=df, ax=ax[9])
sns.boxplot(y= 'PTRATIO', data=df, ax=ax[10])
sns.boxplot(y= 'B', data=df, ax=ax[11])
sns.boxplot(y= 'LSTAT', data=df, ax=ax[12])
sns.boxplot(y= 'CMEDV', data=df, ax=ax[13])
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

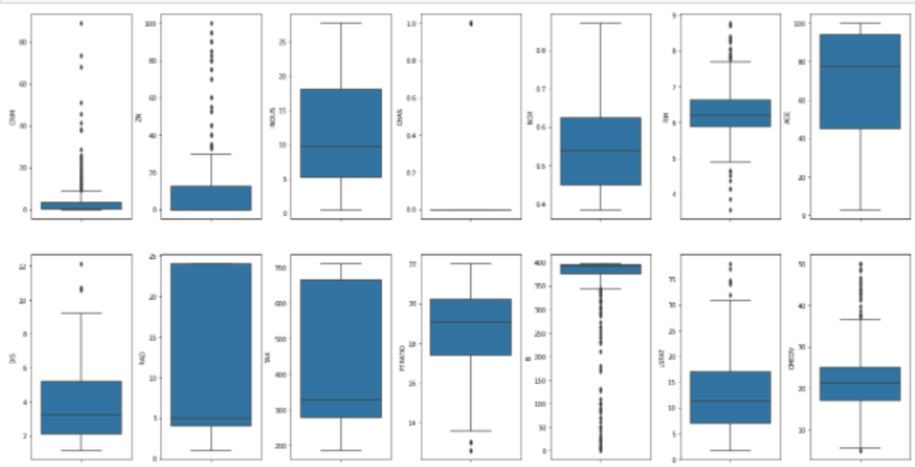

Figure 4.1. : Box and Whisker plot

The above figure is a box plot for various features in the dataset. It is used here to depict the data through quartiles. The lines extending are whiskers and indicate the variability outside the upper and lower quartiles. It can tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

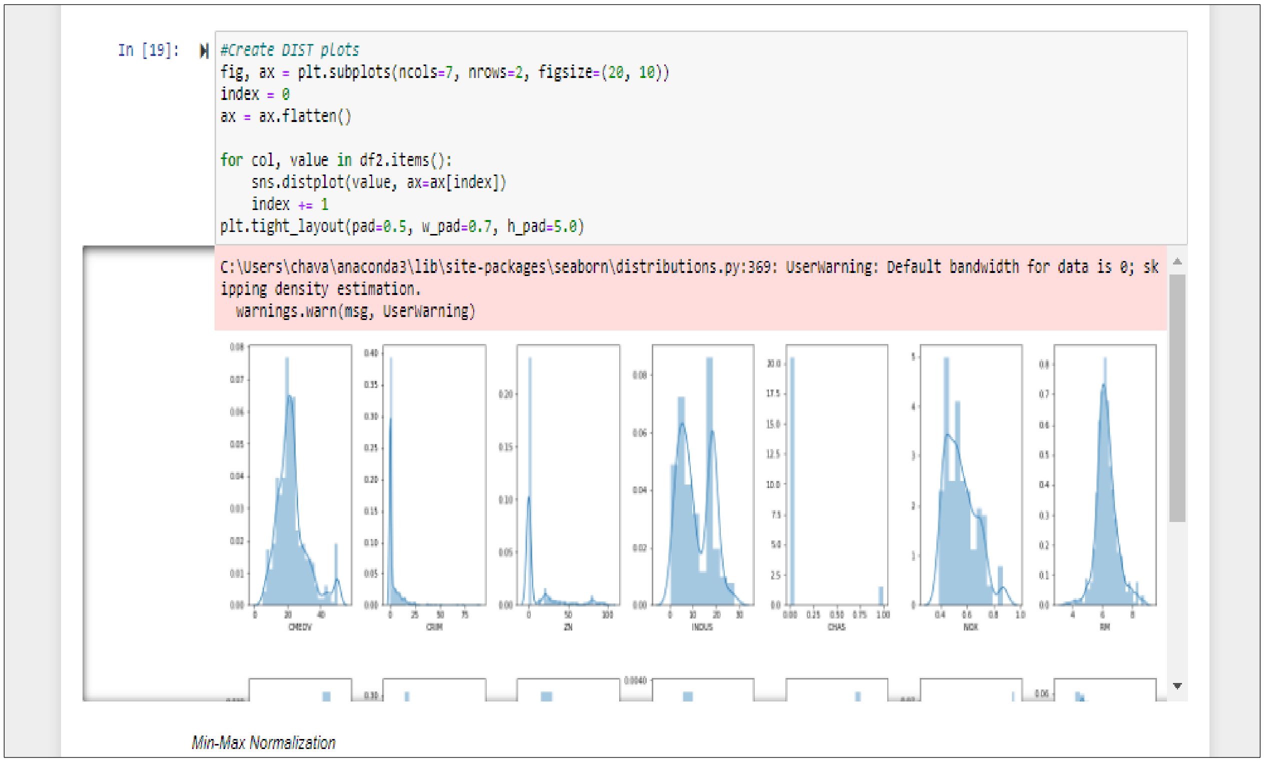

Figure 4.2. Dist plot

The above figure is a seaborn dist plot. It consist mainly of a histogram with line on it. Dist plot is a univariate distribution of observations.

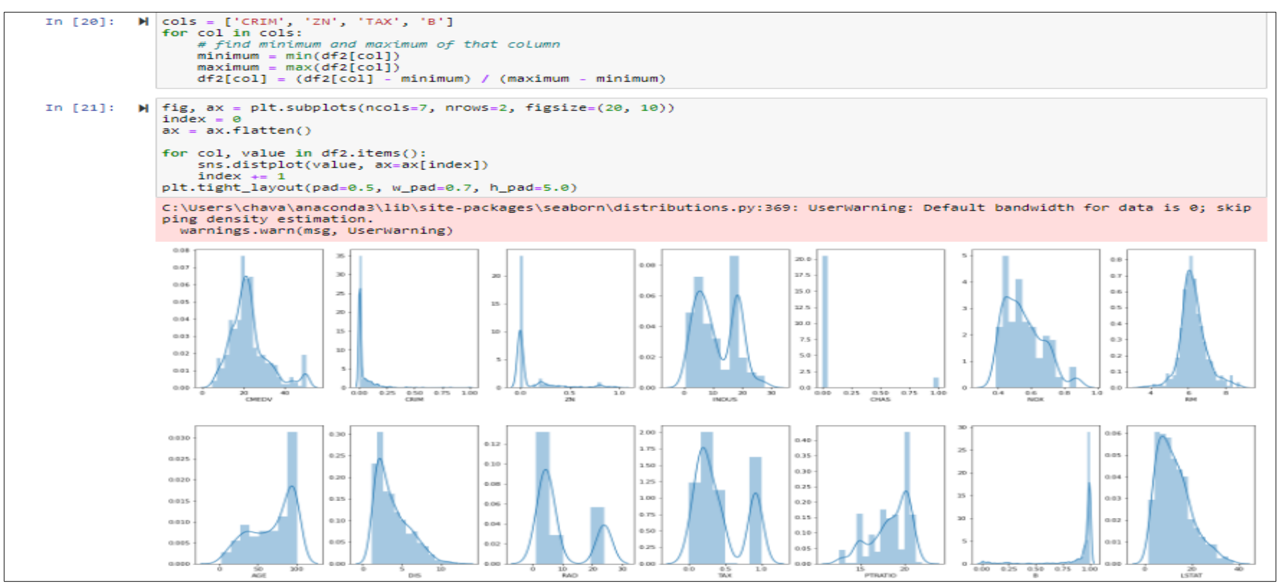

Figure 4.3: Dist plot after Min-Max Scaler

The min-max scaler is a preprocessing technique in the sklearn.preprocessing library. It is used to transform features by scaling each feature to a given range. The formula for transformation is given by

X_std = (X - X.min (axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
Where min, max = feature_range.

The syntax in sklearn is
class sklearn.preprocessing.**MinMaxScaler**(*feature_range=0, 1, *, copy=True, clip=False)*

In [22]:
```
# standardization
from sklearn import preprocessing
scalar = preprocessing.StandardScaler()

# fit our data
scaled_cols = scalar.fit_transform(df2[cols])
scaled_cols = pd.DataFrame(scaled_cols, columns=cols)
scaled_cols.head()
```

Out[22]:

	CRIM	ZN	TAX	B
0	-0.419782	0.284830	-0.666608	0.441052
1	-0.417339	-0.487722	-0.987329	0.441052
2	-0.417342	-0.487722	-0.987329	0.396427
3	-0.416750	-0.487722	-1.106115	0.416163
4	-0.412482	-0.487722	-1.106115	0.441052

In [23]:
```
for col in cols:
    df2[col] = scaled_cols[col]
```

In [24]:
```
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in df2.items():
    sns.distplot(value, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

```
C:\Users\chava\anaconda3\lib\site-packages\seaborn\distributions.py:369: UserWarning: Default bandwidth for data is 0; skip
ping density estimation.
  warnings.warn(msg, UserWarning)
```

Figure 4.4 : Standard scaler technique application

The standard scaler is a preprocessing technique in the sklearn.preprocessing library. It is used to standardize features by removing mean and scaling it to unit variance. The formula for transformation is given by

z = (x - u) / s
where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

The syntax in sklearn is

class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)

Figure 4.5. : Correlation Matrix

The correlation matrix is a table of correlation coefficients between variables. Every cell shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

The common used of correlation matrix is to summarize large amount of data where the aim is to observe patterns, input into another analyses and as a diagnostic. Here, -1 indicates negative linear correlation, 0 indicates no linear correlation and 1 indicates positive linear correlation between two variables.

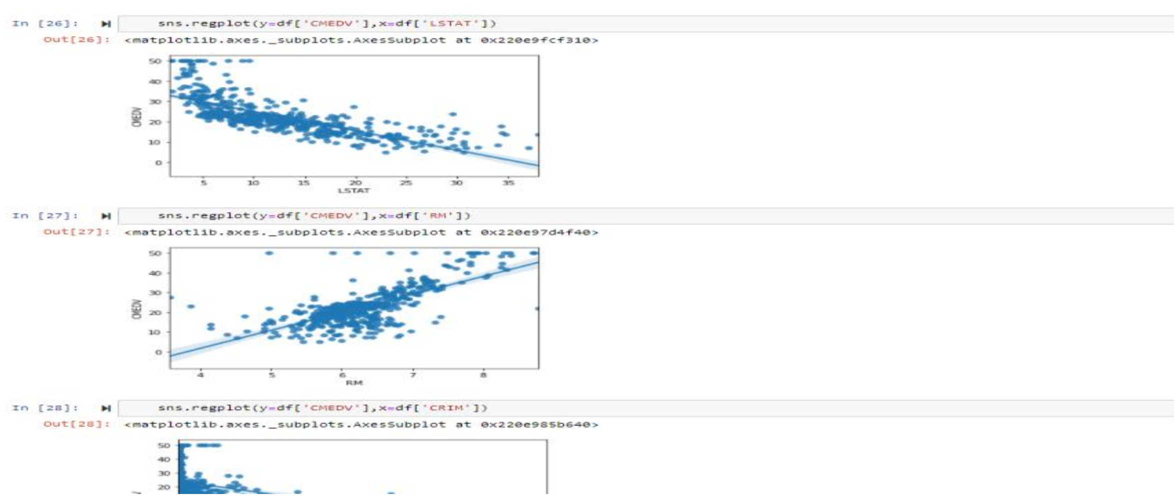

Figure 4.6: Feature Observation

5. Models

5.1. Linear Regression

Linear regression is ML algorithm based on supervised learning. It performs regression tasks and target prediction values based on independent variables. It is used for finding out relationship between variables and forecasting.

Linear Regression

In [31]:
```python
#Linear regression begins
#Training the model

# Import library for Linear regression
from sklearn.linear_model import LinearRegression

# Create a Linear regressor
lm = LinearRegression()

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

Out[31]: LinearRegression()

In [32]:
```python
# Value of y intercept
lm.intercept_
```

Out[32]: 34.04956690327636

In [33]:
```python
#Converting the coefficient values to a dataframe
coeffcients = pd.DataFrame([X_train.columns,lm.coef_]).T
coeffcients = coeffcients.rename(columns={0: 'Attribute', 1: 'Coefficients'})
coeffcients
```

Out[33]:

	Attribute	Coefficients
0	CRIM	-1.04043
1	ZN	1.31891
2	INDUS	-0.00511846
3	CHAS	4.72643
4	NOX	-14.6705
5	RM	3.27964
6	AGE	-0.0042343
7	DIS	-1.57465
8	RAD	0.327912
9	TAX	-2.43257
10	PTRATIO	-0.787248
11	B	0.832628
12	LSTAT	-0.524433

Figure 5.1 : Linear Regression performed in the project

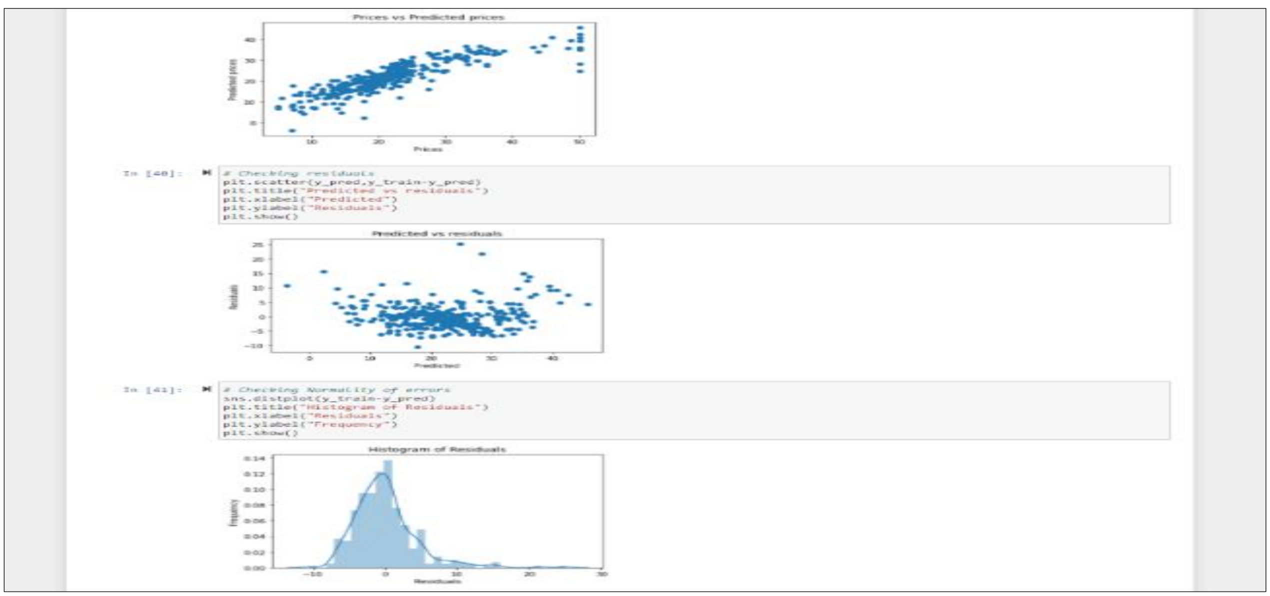

Figure 5.2 : Output of linear regression

5.2. Random Forest Regressor

Random forest regression is supervised learning algorithm using ensemble learning which combines multiple ML algorithms to makes accurate predictions. It is powerful and accurate and performs great on problems like nonlinear relationships. The disadvantage is that overfitting occurs easily.

Random Forest Regressor

```
In [47]:  #Importing Random Forest Regressor
          from sklearn.ensemble import RandomForestRegressor

          #Creating Random Forest Regressor
          reg = RandomForestRegressor()

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

Out[47]:  RandomForestRegressor()

In [48]:  # Model prediction on train data
          y_pred = reg.predict(X_train)

In [49]:  # Model Evaluation
          print('R^2:',metrics.r2_score(y_train, y_pred))
          print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
          print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
          print('MSE:',metrics.mean_squared_error(y_train, y_pred))
          print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))

          R^2: 0.980456226162348
          Adjusted R^2: 0.9797089642214966
          MAE: 0.8155564971751416
          MSE: 1.4673895960451977
          RMSE: 1.2113585745125997

In [111]: cv_score = cross_val_score(reg, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
          cv_score = np.abs(np.mean(cv_score))

In [112]:  print('CV Score:', cv_score)

          CV Score: 31.23542592446512
```

Figure 5.3 : Random Forest Regression performed in the project

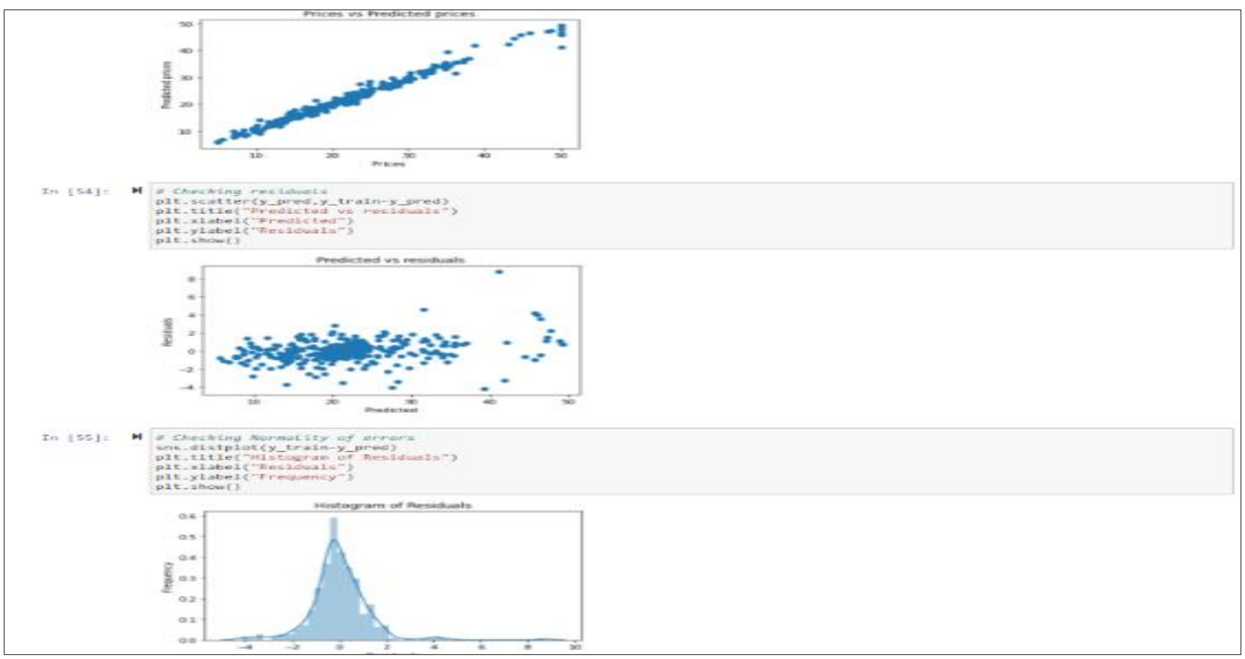

Figure 5.4 : Output of Random forest regression

5.3. Support Vector Regressor

The SVM Regressor works on the principle of SVM classification. It is adapted form of SVM when dependent variable is numerical and not categorical. The main benefit is that it is a non parametric technique. The output does not depend on distributions of underlying dependent and non dependent variables.

SVM Regressor

In [59]:
```
# Creating scaled set to be used in model to improve our results
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Train the model

In [65]:
```
# Import SVM Regressor
from sklearn import svm

# Create a SVM Regressor
svmreg = svm.SVR()
```

In [83]:
```
#Train the model using the training sets
svmreg.fit(X_train, y_train)
```

Out[83]:
```
SVR()
```

In [67]:
```
# Model prediction on train data
y_pred = svmreg.predict(X_train)
```

In [68]:
```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.6459011785476183
Adjusted R^2: 0.6323621059626743
MAE: 2.890924651825334
MSE: 26.58651961936111
RMSE: 5.1562117508264835
```

Figure 5.5: SVM Regression performed in the project

In [64]:
```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [71]:
```
# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```

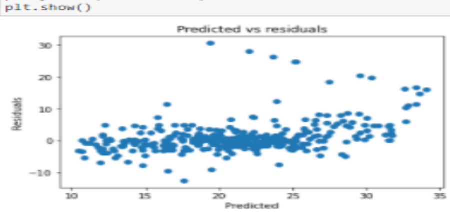

Figure 5.6 : Output of SVM regression

5.4. XGBoost Regressor

XGBoost is a tool for building supervised learning models. The results of the regression problems are continuous or real values. Some commonly used regression algorithms are Linear Regression and Decision Trees. Root-mean-squared error (RMSE) and mean-squared-error (MAE) are metrics involved in the regression. RMSE is square root of mean squared error. MAE is an absolute sum of actual and predicted differences.

XGBoost Regressor

Training the model

In [79]:
```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.3.3-py3-none-win_amd64.whl (95.2 MB)
Requirement already satisfied: scipy in c:\users\chava\anaconda3\lib\site-packages (from xgboost) (1.5.0)
Requirement already satisfied: numpy in c:\users\chava\anaconda3\lib\site-packages (from xgboost) (1.18.5)
Installing collected packages: xgboost
Successfully installed xgboost-1.3.3
```

In [82]:
```
# Import XGBoost Regressor
from xgboost import XGBRegressor

#Create a XGBoost Regressor
xgbreg = XGBRegressor()

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

Out[82]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=12, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Figure 5.7: XGBoost Regression performed in the project

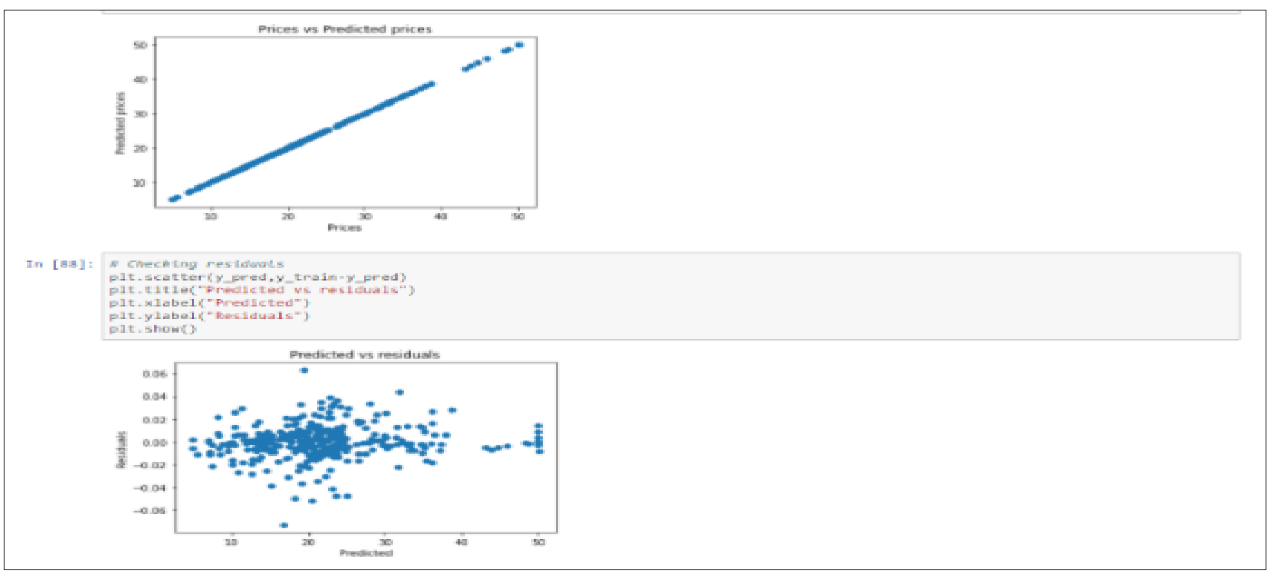

Figure 5.8: Output of XGBoost regression

5.5. Model Evaluation

After evaluating all the four models to find the best model, it is concluded that the XGBoost Regressor is the best model with R-squared score of 88.8% and Cross validation score of 16.34.

Evaluating all the models together

In [92]:
```
models = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost', 'Support Vector Machines'],
    'R-squared Score': [acc_linreg*100, acc_rf*100, acc_xgb*100, acc_svm*100]})
models.sort_values(by='R-squared Score', ascending=False)
```

Out[92]:

	Model	R-squared Score
2	XGBoost	88.814143
1	Random Forest	82.816550
0	Linear Regression	71.667099
3	Support Vector Machines	59.290598

In [122]:
```
models1 = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'Support Vector Machines', 'XGBoost'],
    'CV-Score': [cv_score1, cv_score2, cv_score3, cv_score4]})
models1.sort_values(by='CV-Score', ascending=True)
```

Out[122]:

	Model	CV-Score
3	XGBoost	16.347440
0	Linear Regression	31.756006
1	Random Forest	56.886331
2	Support Vector Machines	56.886331

Figure 5.9: Results of model evaluation

6. Data Visualization using Tableau

The Tableau Story is the other half of this project where the price predictions that were made using the different models can be visualized. Here, the Tableau visualization will give an interactive demonstration of the factors and their direct or indirect effect on the price of the houses.

There are multiple charts used in the workbook for creating a story based on the Boston House price prediction.

There are four dashboards used to create a story. First dashboard consists of a map showing the areas with size varying marks with respect to the interest measure selected. Second visualizes the effect on the average measures due to different price range. Third dashboards consists of top areas with highest and lowest measures. All the dashboards are interactive and one can select the interest measure to visualize the necessary information.

Figure 6.1 : Tableau story point 1

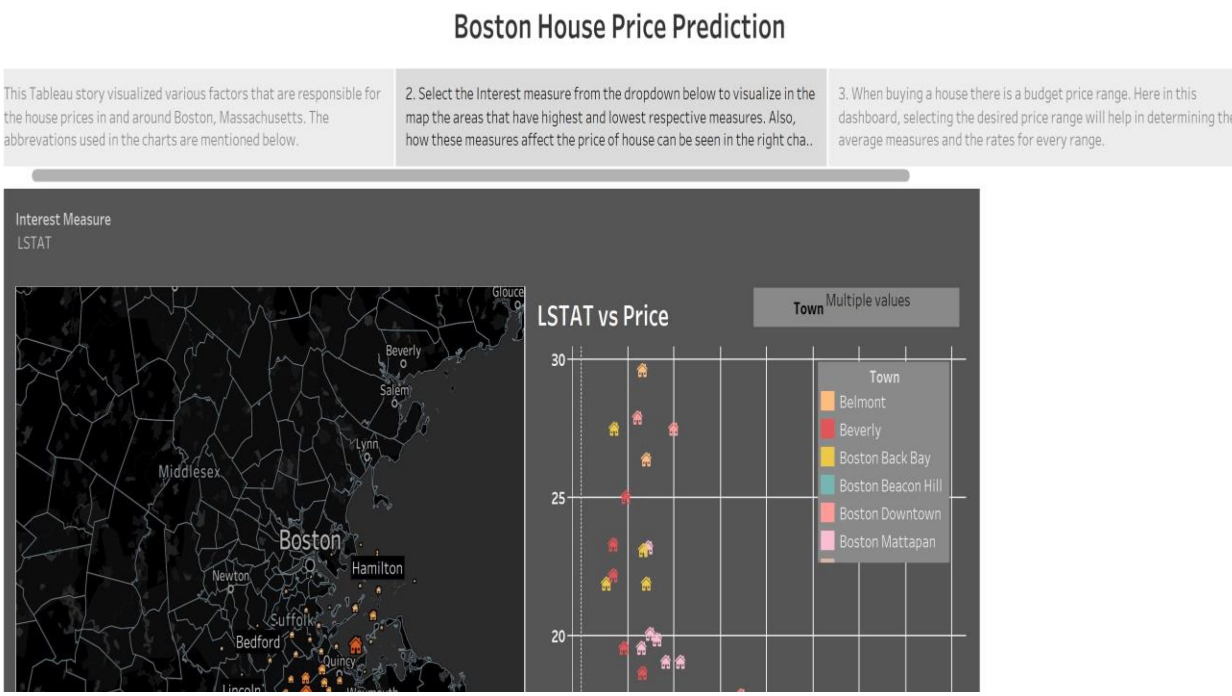

Figure 6.2 : Tableau story point 2

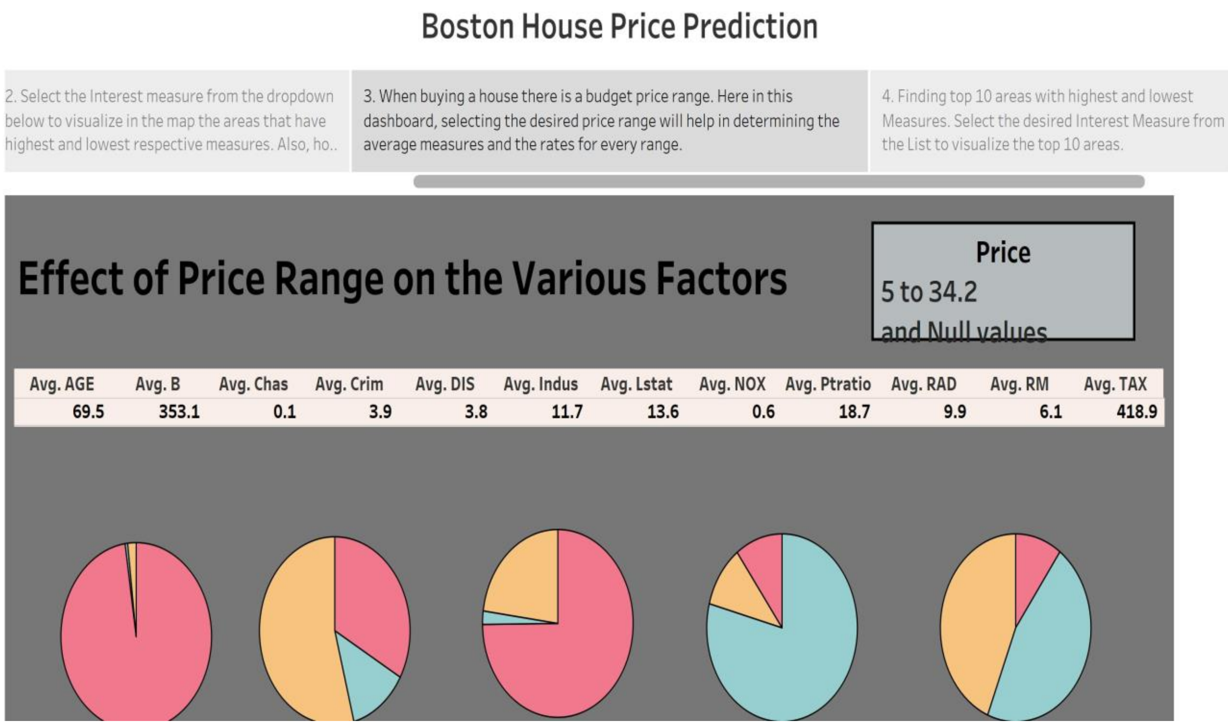

Figure 6.3 : Tableau story point 3

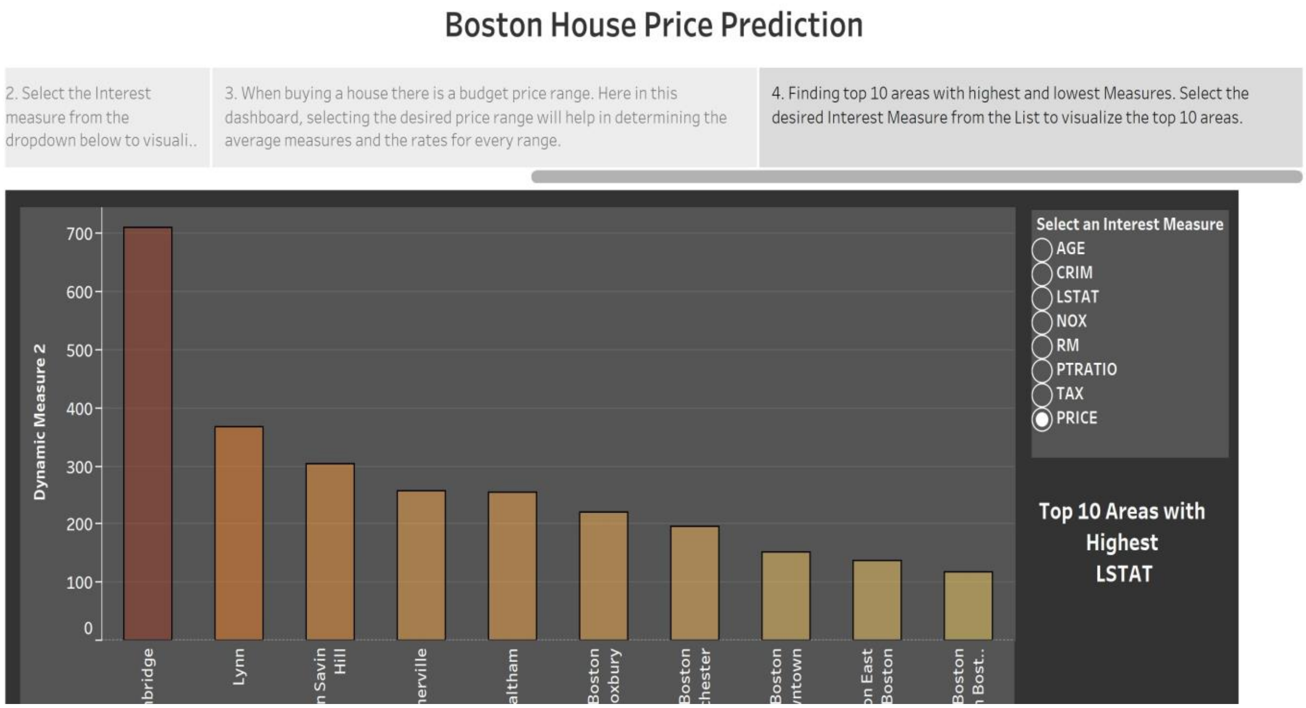

Figure 6.4 : Tableau story point 4

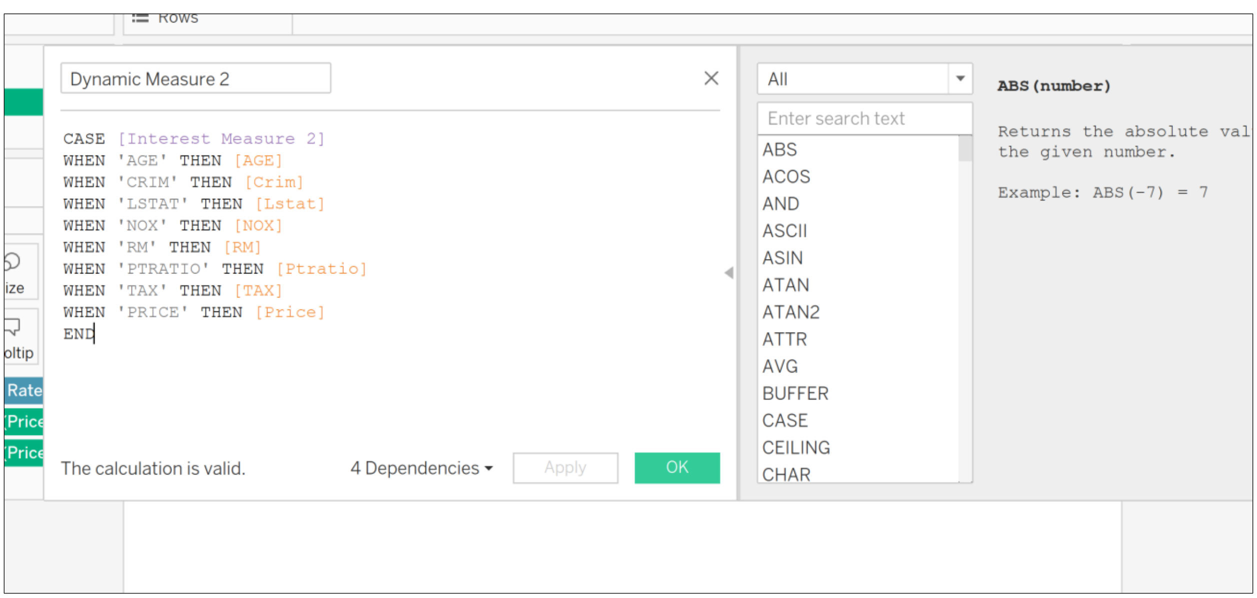

Figure 6.5 : Tableau calculation used in most of the sheets

7. Conclusion

- The XGBoost is the best model for predicting the monetary value of the house price based on various features with R-squared score of 88.8% and cross validation score of 16.34.

- The Price is comparatively low in the areas where the crime rate is high and vice versa in most cases.

- The highest crime rate is in Boston’s Roxbury area but the price is not comparatively low.

- The price is high for the house with more number of rooms but that fails in case of some areas. This is due to other measures having more effect.

- The price is less where the Tax is high in most cases.

- The price is very low where the NOX is high.

- Pupil Teacher Ratio does not affect the price in all the areas.

- The price is low in area where the low status population is high.

8. References

- http://lib.stat.cmu.edu/datasets/boston_corrected.txt
- https://www.ibm.com/cloud/learn/exploratory-data-analysis
- https://pythonbasics.org/seaborn-distplot/
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://www.geeksforgeeks.org/python-linear-regression-using-sklearn/
- https://levelup.gitconnected.com/random-forest-regression-209c0f354c84
- https://www.kdnuggets.com/2017/03/building-regression-models-support-vector-regression.html