Subject: artificial intelligence

Name: MD AYNUL ISLAM

Chinese Name: 叶子

Student ID: 4420190030

Topic Name: Boston House Price Prediction

Administrator
[Date]

Table of Contents

ABSTRACT

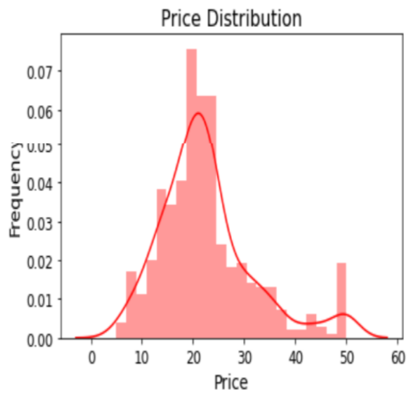
In this paper, I explore how predictive modeling can be applied in housing sale price prediction by analyzing the housing dataset and use machine learning model. . Actually, I tried several different models, namely as,svm, linear regression, randomforest and xgboost. Additionally, as my dataset import from sklearn, so there are no missing values. I just did explorary data analysis, feature enginnering before model fitting. After I try four different models, I get some results. As for the first model DecisionTreeRegressor, it doesn't meet the assumption of equality of the variances. Therefore,we can't use the DecisionTreeRegressor as the candidate of our final model. This model is underfitting .Then I try random forest classifier ,the RMSE and R-squared looks so good. Then I try Random forest. The R squared in this model of training set is very good, in the test set the R squared is also good. All of the results of this random forest classifier model seem very good. Therefore, I will use this random forest classifier model as my final model to predict the housing price. The random forest classifier model also shows which variables have important effects on sale price.

INTRODUCTION

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. In this project I discuss about boston house price prediction using sklearn.ensemble.RandomForestClassifier. "A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree."[1] Modeling uses machine learning algorithms, where machine learns from the data and uses them to predict a new data. The most frequently used model for predictive analysis is regression. As we know, the proposed model for accurately predicting future outcomes has applications in economics, business, banking sector, healthcare industry, e-commerce, entertainment, sports etc. One such method used to forecast house prices are based on multiple factors. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications, the random forest algorithm is fast enough but there can certainly be situations where run-time performance is important and other approaches would be preferred.

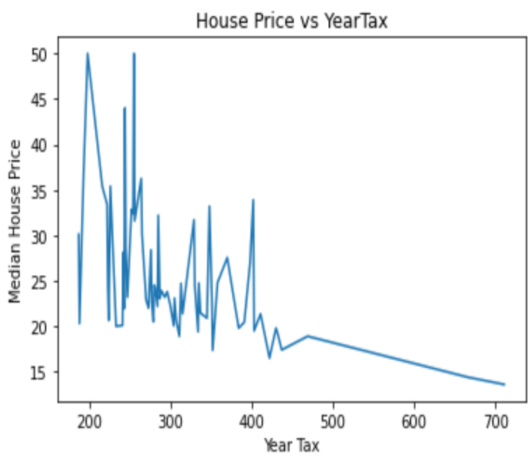
Data Analysis

Distribution Plot



In this distribution plot we distribution the price vs frequenc. In here we express the house price distribute with frequence.

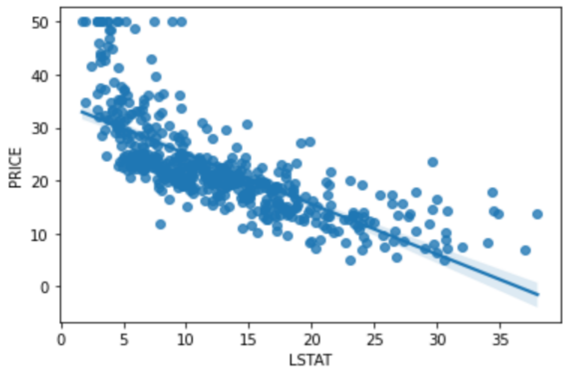
House Price Vs Year Tax



In House Price Vs Year Tax we make a graph made a median house price vs year tax. (“house price vs yeartax”, 0.5,1.0) .

PRICE VS LSTAT

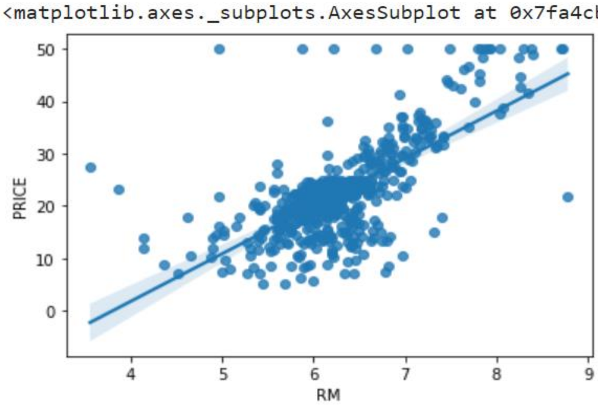
<matplotlib.axes._subplots.AxesSubplot at 0x7fa4d94bd610>



In this regplot we compare price with LSTAT.

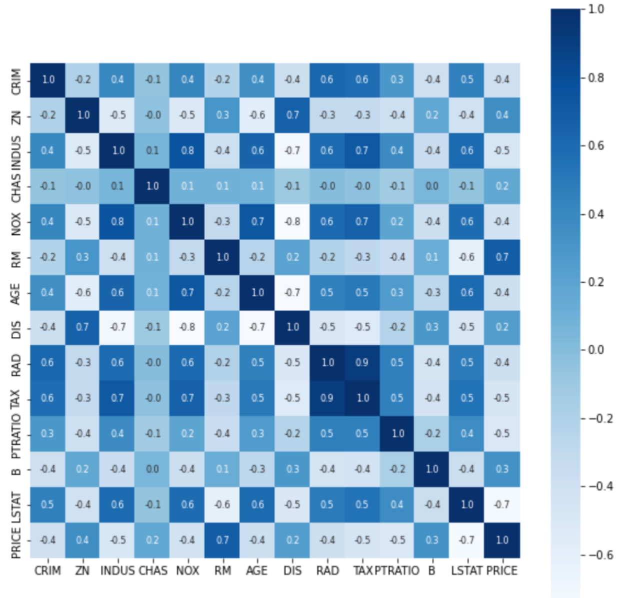
PRICE VS RM

<matplotlib.axes._subplots.AxesSubplot at 0x7fa4cbef36d0>



In this regplot we compare price with RM.

CONSTRUCTING A HEATMAP



In this heatmap. We define correlations with all data.

ALGORITHM ANALYSIS

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. “Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.”[2]

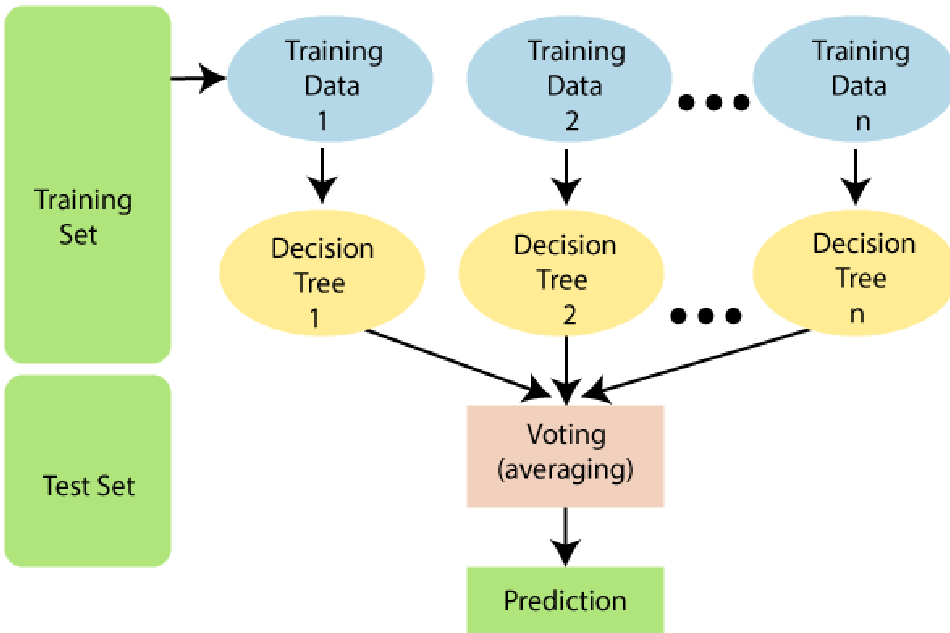
THE RANDOM FORESTS ALGORITHM

It technically is an ensemble method (based on the divide-and-conquer approach) of decision trees generated on a randomly split dataset. This collection of decision tree classifiers is also known as the forest. The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute. Each tree depends on an independent random sample. In a classification problem, each tree votes and the most popular class is chosen as the final result. In the case of regression, the average of all the tree outputs is considered as the final result. It is simpler and more powerful compared to the other non-linear classification algorithms.

HOW DOES THE ALGORITHM WORK?

It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.



IMPORTANT FEATURES

Random forests also offers a good feature selection indicator. Scikit-learn provides an extra variable with the model, which shows the relative importance or contribution of each feature in the prediction. It automatically computes the relevance score of each feature in the training phase. “Random forest uses gini importance or mean decrease in impurity (MDI) to calculate the importance of each feature. Gini importance is also known as the total decrease in node impurity. This is how much the model fit or accuracy decreases when you drop a variable. The larger the decrease, the more significant the variable is. Here, the mean decrease is a significant parameter for variable selection. The Gini index can describe the overall explanatory power of the variables.”[2]

IMPORTANT HYPERPARAMETERS

The hyperparameters in random forest are either used to increase the predictive power of the model or to make the model faster. Let's look at the hyperparameters of sklearns built-in random forest function.

Increasing the predictive power

Firstly, there is the n_estimators hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.

Another important hyperparameter is max_features, which is the maximum number of features random forest considers to split a node. Sklearn provides several options, all described in the documentation.

Increasing the model's speed

The n_jobs hyperparameter tells the engine how many processors it is allowed to use. If it has a value of one, it can only use one processor. A value of “-1” means that there is no limit.

Lastly, there is the oob_score (also called oob sampling), which is a random forest cross-validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out-of-bag samples. It's very similar to the leave-one-out-cross-validation method, but almost no additional computational burden goes along with it.[3]

CRITERIA TO MEASURE PERFORMANCE

For measuring how good predictions the model makes, four error metrics havebeen used. Mean absolute error (MAE), Mean squared error (MSE), Medianabsolute error (MedAE) and Coefficient of determination (R2). They are alldefined below.

MEAN ABSOLUTE ERROR (MAE)

Mean absolute error measures the prediction error by taking the mean of allabsolute values of all errors, that is:

$$MAE = \frac{\sum_{i=0}^{n} |y_i - \hat{y_i}|}{n}$$

Where n is the number of samples, y are the target values andˆyare the pre-dicted values. A MAE closer to 0 means that the model predicts with lowererror and that the prediction is better the closer the MAE is to 0.[4]

MEAN SQUARED ERROR (MSE)

Mean squared error is similar to MAE, but the impact of a term is quadraticallyproportional to its size. It measures the prediction error by taking the mean of all squared absolute values of all errors, that is:

$$MSE = \frac{\sum_{i=0}^{n} (y_i - \hat{y_i})^2}{n}$$

This is the prediction error by taking the mean of all squared absolute values.[5]

MEDIAN ABSOLUTE ERROR (MEDAE)

The median absolute error (MedAE) is the median of all absolute differencesbetween the predicted value and the target value. In difference to MAE andMSE, the median absolute error is more robust to outliers by virtue of usingthe median instead of the mean.

$$MedAE = median(|y_1 - \hat{y_1}|...|y_n - \hat{y_n}|)$$

A low MedAE means little error and a good prediction.[6]

COEFFICIENT OF DETERMINATION

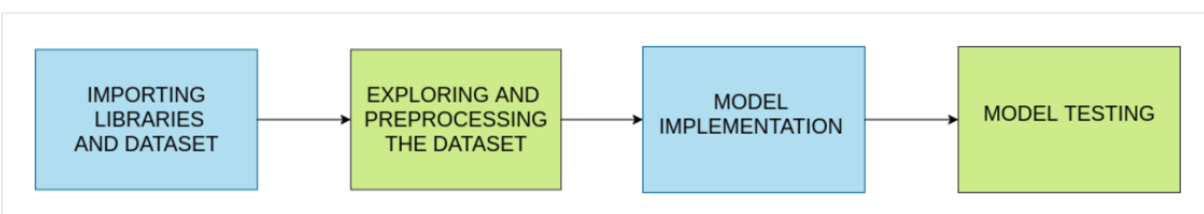
The coefficient of determination, R2, is the ratio between the explained vari-ance and the total variance. Another, perhaps more intuitive way of under-standing it is that it is the fraction of the variance that is predictable by (orexplained by) by the model. It is frequently used to evaluate how well a re-gression model fits the data. The coefficient value ranges from 0 to 1 where 1is better, meaning perfect determination and 0 meaning no determination.[7]

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1}(y_i - \bar{y})^2}$$

Where yi is the predicted value of theith sample, y is the corresponding actual value over n samples and ̄y is the arithmetic mean as such:

$$\bar{y} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} y_i$$

SOURCE CODE




```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston
```

```python
from sklearn.model_selection  import train_test_split
from sklearn.preprocessing import StandardScaler
#from sklearn.svm import SVR
#from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
#from sklearn.tree import DecisionTreeRegressor
# from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_absolute_error,r2_score
```

```python
boston = load_boston()
boston
```

```
{'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n---------------------------\n\n**Data Set Characteristics:**  \n\n
 'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'filename': '/usr/local/lib/python3.7/dist-packages/sklearn/datasets/data/boston_house_prices.csv',
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
```

```python
boston_df = pd.DataFrame(boston.data, columns = boston.feature_names)
boston_df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

```
boston_df['PRICE']=boston.target
boston_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
boston_df.shape
```

```
(506, 14)
```

```
boston_df.isnull().sum()
```

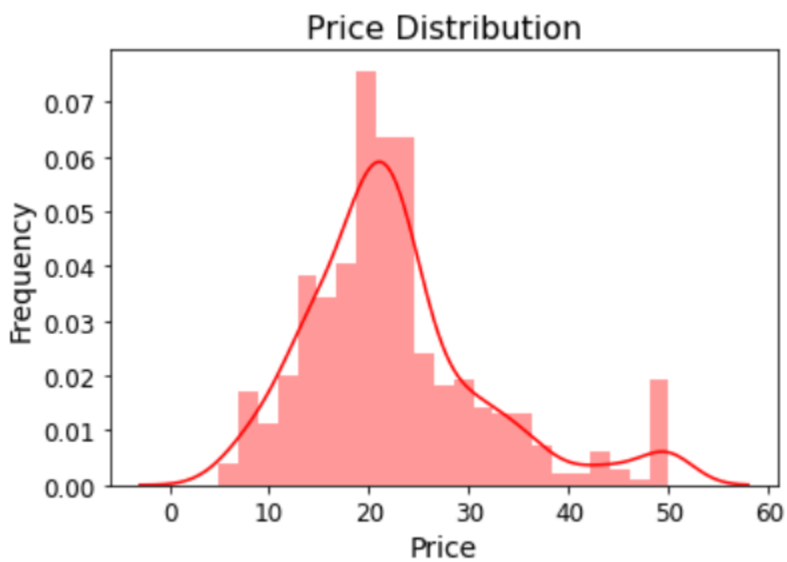
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

```
boston_df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.65
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.14
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.73
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.95
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.36
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.95
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.97

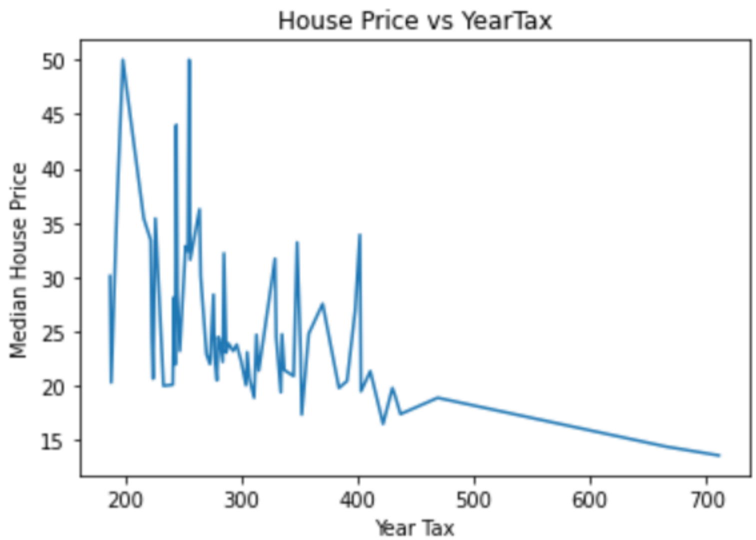
```
sns.distplot(boston_df['PRICE'], color = 'r')
```

```
plt.title('Price Distribution', fontsize = 16)
plt.xlabel('Price', fontsize = 14)
plt.ylabel('Frequency', fontsize = 14)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)

plt.savefig('distplot.png')
plt.show()
```



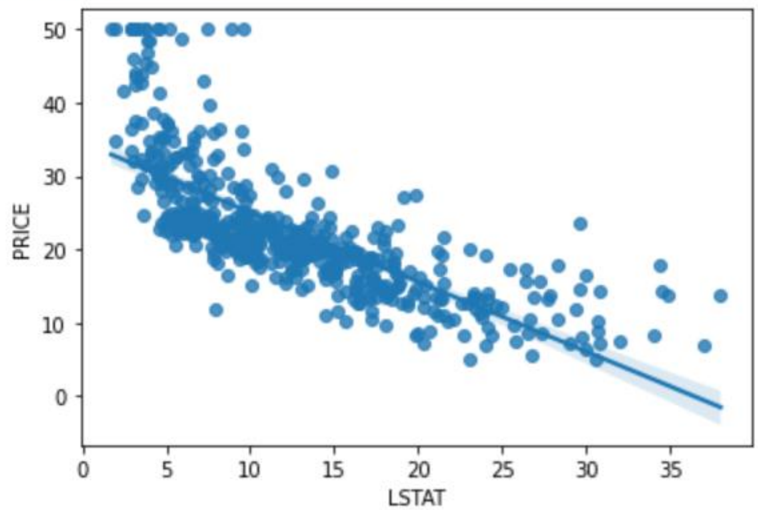

```
boston_df.groupby('TAX')['PRICE'].median().plot()
plt.xlabel('Year Tax')
plt.ylabel('Median House Price')
plt.title("House Price vs YearTax")
```

```
Text(0.5, 1.0, 'House Price vs YearTax')
```



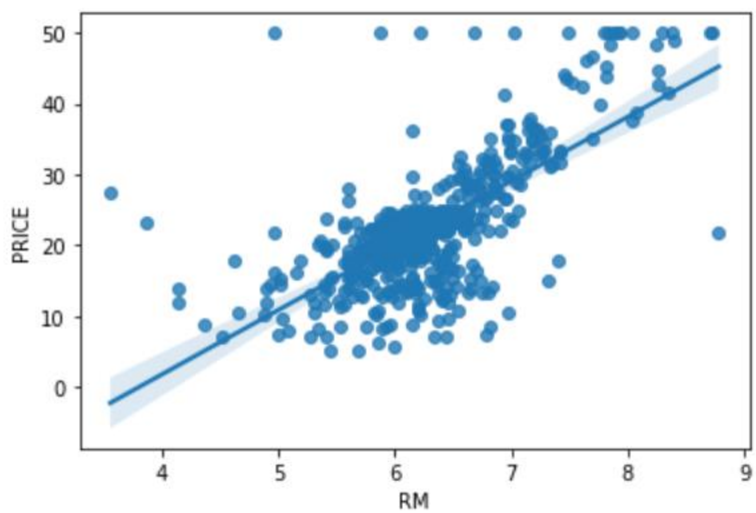
```
sns.regplot(y=boston_df['PRICE'], x=boston_df['LSTAT'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fa4d94bd610>



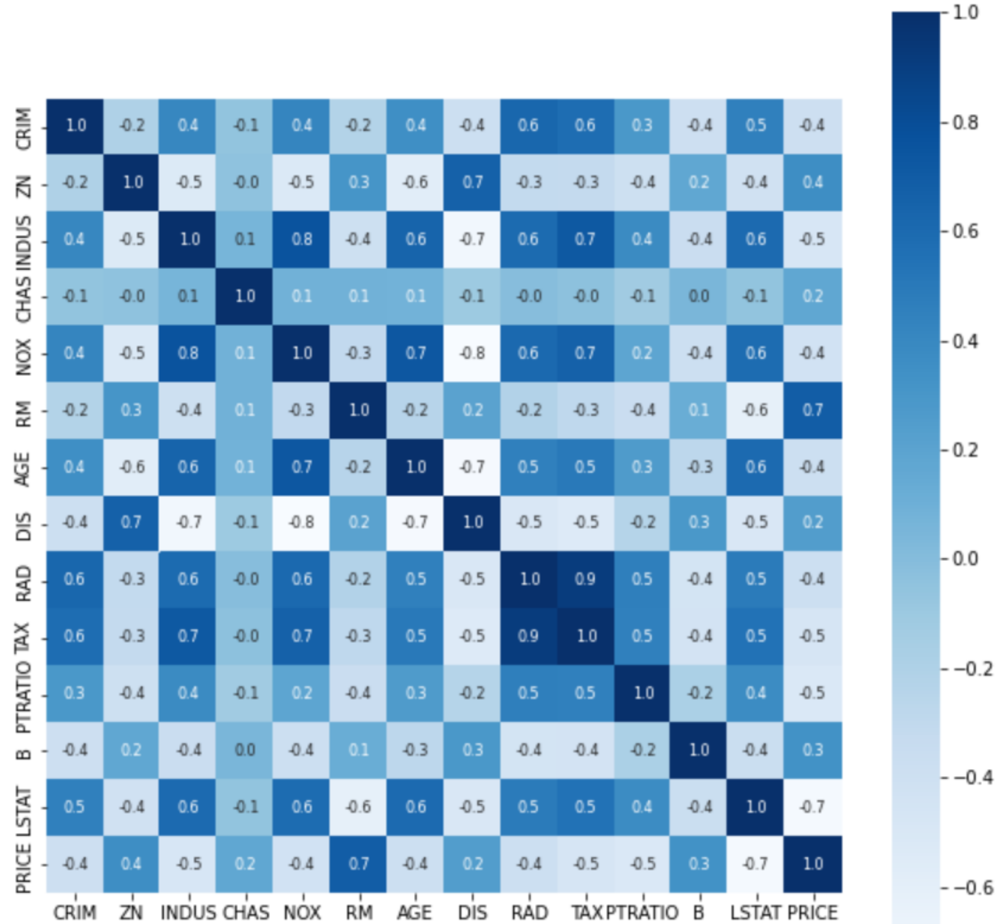

```
sns.regplot(y=boston_df['PRICE'], x=boston_df['RM'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fa4cbef36d0>




```
correlation = boston_df.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True,
annot_kws={'size':8}, cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa4db48aa50>
```




```
X = boston_df.drop(['PRICE'], axis=1)
y = boston_df['PRICE']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(404, 13) (102, 13) (404,) (102,)
```

```
sc_x=StandardScaler()
X=sc_x.fit(X_train)
X=sc_x.fit(X_test)
```

```
#modeling
house_Price_model=RandomForestRegressor()
house_Price_model.fit(X_train,y_train)
```

```
#modeling
house_Price_model=RandomForestRegressor()
house_Price_model.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
X_train_prediction = house_Price_model.predict(X_train)
X_train_prediction
```

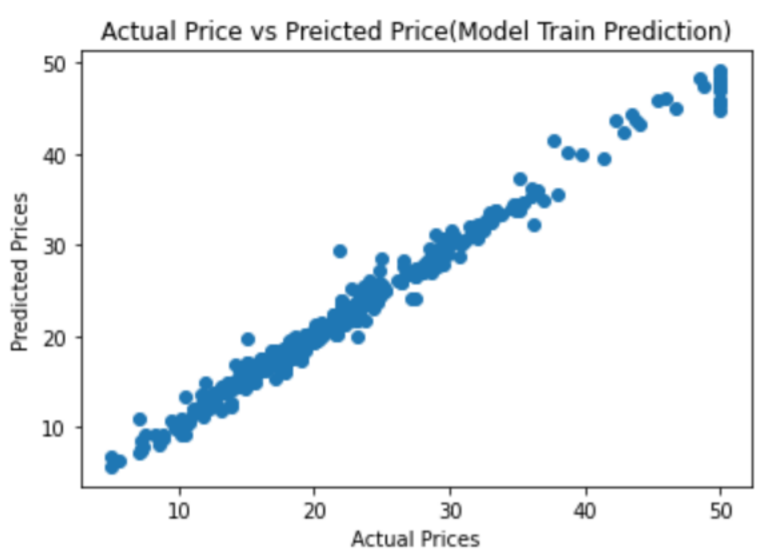
```
array([24.901, 23.649,  7.475, 20.801, 13.516, 26.659, 27.823, 26.187,
       44.387, 23.767, 12.034, 33.815, 36.205, 36.01 , 19.317, 19.546,
       34.522, 47.117, 20.239, 14.284, 28.232, 19.804, 24.631, 20.819,
       23.51 , 20.257, 22.177, 27.838, 19.984, 22.752, 24.914,  9.744,
       24.548, 35.343, 14.171, 13.564, 39.994, 13.226, 21.122, 16.722,
       19.256, 24.125, 29.6  , 23.056, 23.704, 17.786, 28.228, 19.846,
       33.263, 14.75 , 20.453, 17.34 , 13.756, 30.073, 27.372, 25.47 ,
       23.827, 25.514, 43.708, 29.357, 26.195, 15.001, 20.544, 21.331,
       23.757, 11.891, 16.272, 24.342, 20.992, 21.76 , 14.977, 29.151,
       20.034, 23.551, 30.798, 19.536, 47.593, 20.732, 20.268, 23.087,
       17.444, 31.189, 12.076, 20.714, 20.398, 37.354, 20.   , 21.785,
       21.45 , 14.36 , 21.253, 17.068, 14.866, 19.363, 39.461, 19.211,
       45.994, 45.59 , 17.533, 16.727, 26.874, 16.137, 17.998, 28.054,
       25.491, 20.751, 19.592, 18.937, 22.038, 18.537, 30.387, 18.802,
       20.517, 24.594, 17.431, 14.023, 23.038, 21.405, 21.228, 18.958,
       19.721, 19.24 , 22.471, 32.711, 26.152, 10.949, 19.317, 32.494,
       13.993, 15.394, 27.304, 23.549, 10.643, 24.03 , 19.027, 43.211,
       48.033, 23.692, 20.202,  8.95 , 21.117, 48.358, 21.667, 31.776,
       23.722, 17.843, 29.527, 10.424, 16.577, 14.447, 32.274, 17.408,
        9.116, 27.613, 14.594, 19.722, 27.991, 31.747, 18.305, 28.454,
        9.251, 17.558, 31.033, 24.071, 45.011, 26.612, 16.708, 22.372,
       31.306, 14.432, 23.868, 17.129, 25.971, 25.13 , 25.097, 21.531,
       20.949, 18.804,  6.231, 19.098, 19.918, 21.262, 20.272, 13.23 ,
       23.788, 17.044, 14.394, 32.832, 10.131,  9.121, 26.529, 19.067,
       23.555, 12.602, 34.67 , 22.034, 28.518,  7.67 , 16.202,  9.047,
       22.079, 13.774, 19.613, 25.314,  8.014, 17.429, 28.16 , 21.745,
       19.725, 33.025, 24.687, 22.921, 15.592, 26.725, 19.326, 24.169,
       20.833, 33.834, 17.627, 15.964, 25.985, 22.324, 15.604, 31.101,
       21.228, 22.149, 14.813, 12.494, 15.248, 31.922, 26.987, 24.358,
       17.237,  8.412, 12.24 , 12.56 , 32.296, 40.145, 19.703, 27.541,
       19.766, 11.759, 18.186,  8.879, 19.007, 22.989, 21.389, 29.57 ,
       33.964, 24.56 , 21.683, 35.636,  7.184, 19.535, 19.472, 26.869,
       19.373, 30.201, 20.706, 44.872, 18.648, 21.297, 30.615, 14.319,
       16.071, 43.731, 16.962, 18.906, 20.956, 17.157, 21.29 , 22.37 ,
       21.132, 23.387, 31.461, 23.989,  9.091, 21.892, 20.651, 18.162,
       13.925, 11.166, 20.064, 22.36 , 33.537, 21.007, 30.551, 18.239,
       14.945, 10.557, 34.504, 25.586, 47.402, 42.315, 20.071, 31.549,
       22.919, 47.178, 23.355, 31.499, 19.346, 15.501, 25.912, 35.013,
       24.339, 25.729, 23.947, 10.82 , 22.344, 24.628, 14.829, 13.743,
       28.834, 22.996, 17.047, 49.168, 23.224, 19.454, 16.673, 16.919,
       11.129, 33.893, 15.911, 48.974, 23.172, 19.831, 19.84 , 18.296,
       46.101, 19.955, 41.507, 21.874, 14.659, 33.537, 32.594, 20.272,
       20.293, 45.463, 19.532, 19.319, 25.221, 17.143, 20.283, 24.747,
       45.948, 33.897, 21.051, 19.757,  6.636,  7.753, 19.863, 13.316,
       20.576, 16.963, 18.281, 19.282, 22.74 , 36.084, 19.205, 18.615,
        8.655, 22.332, 22.238, 15.508, 19.835, 21.289, 24.164, 18.473,
       14.454, 21.835, 19.847, 20.47 , 20.056, 24.862, 18.041, 19.968,
        5.645, 16.354, 14.416, 30.754, 13.827, 18.307, 12.258, 20.99 ,
       22.355, 33.564, 28.931, 34.762, 14.882, 31.999, 21.273, 16.179,
       14.994, 12.822, 14.722, 20.897, 18.107, 48.26 , 14.573, 20.813,
       22.704, 13.691, 23.645, 20.703])
```

```
# R squared error
score_1 =r2_score(y_train, X_train_prediction)

# Mean Absolute Error
score_2 =mean_absolute_error(y_train, X_train_prediction)

print("R squared error : {} %".format(score_1*100))
print('Mean Absolute Error : {} %'.format(score_2))
```


```
# R squared error
score_1 =r2_score(y_train, X_train_prediction)

# Mean Absolute Error
score_2 =mean_absolute_error(y_train, X_train_prediction)

print("R squared error : {} %".format(score_1*100))
print('Mean Absolute Error : {} %'.format(score_2))
```


```
R squared error : 98.27104360761231 %
Mean Absolute Error : 0.7985618811881182 %
```

```
plt.scatter(y_train, X_train_prediction)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Preicted Price(Model Train Prediction)")
plt.show()
```



```
# accuracy for prediction on test data
X_test_prediction = house_Price_model.predict(X_test)
X_test_prediction
```

```
array([30.363, 27.338, 20.155, 20.563, 20.151, 19.923, 28.316, 19.079,
       20.606, 23.708, 30.008, 31.175, 20.598, 20.133, 20.485, 26.635,
       11.847, 40.988, 24.158, 14.579, 19.919, 16.337, 24.118, 23.679,
       25.191,  9.256, 14.602, 19.397, 43.643, 12.43 , 26.529, 19.59 ,
       47.899, 15.924, 23.074, 20.842, 15.522, 33.111, 13.445, 19.532,
       24.708, 23.009, 25.583, 16.23 , 15.886, 11.677, 47.79 , 11.82 ,
       21.484, 18.516, 23.489, 21.457, 24.667, 20.501, 10.853, 23.81 ,
       11.22 , 23.639, 18.644, 43.259, 14.267, 26.833, 13.05 , 14.468,
       18.238, 33.738, 42.73 , 24.846, 21.575, 20.173, 23.856,  6.594,
       18.633, 21.008, 19.497, 20.537, 40.752, 24.44 , 27.6  , 32.642,
       17.616, 20.628, 34.255, 11.743, 24.626, 25.227, 14.474, 24.501,
       19.635, 17.275, 26.53 , 45.616, 16.237, 20.651, 14.822, 20.12 ,
       24.027, 23.938, 42.321, 20.671, 15.964, 15.278])
```

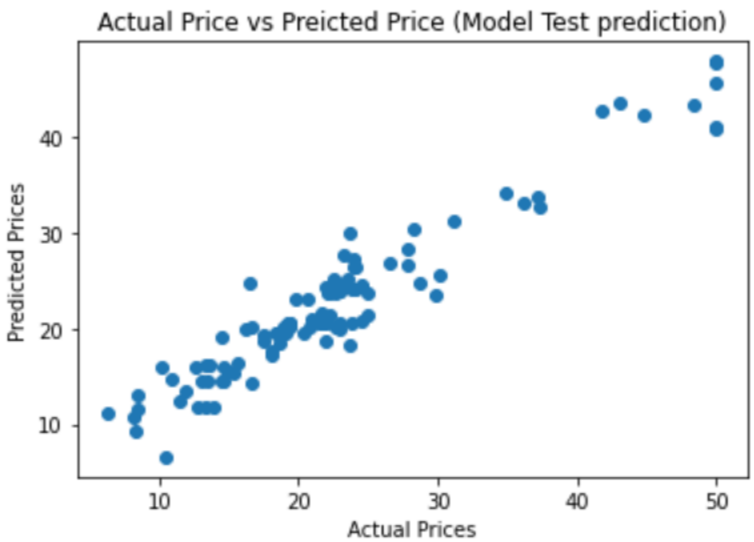
```
# R squared error
score_1 =r2_score(y_test, X_test_prediction)

# Mean Absolute Error
score_2 =mean_absolute_error(y_test, X_test_prediction)

print("R squared error : {} %".format(score_1*100))
print('Mean Absolute Error : {} %'.format(score_2))
```


```
# R squared error
score_1 =r2_score(y_test, X_test_prediction)

# Mean Absolute Error
score_2 =mean_absolute_error(y_test, X_test_prediction)

print("R squared error : {} %".format(score_1*100))
print('Mean Absolute Error : {} %'.format(score_2))
```


```
R squared error : 91.01491236686833 %
Mean Absolute Error : 2.2834509803921588 %
```

```
plt.scatter(y_test, X_test_prediction)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Preicted Price (Model Test prediction)")
```

```
plt.show()
```

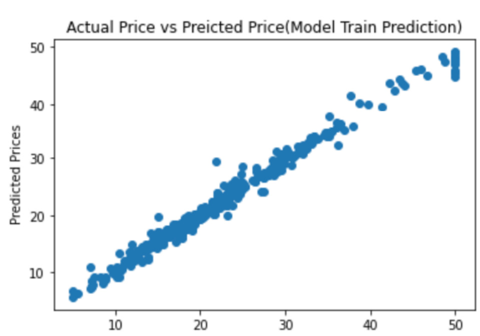


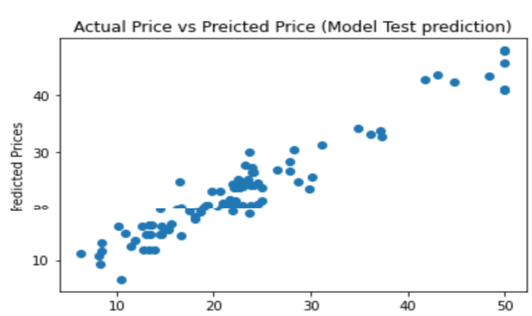
RESULTS ANALYSIS

In this part I am going to explain about all over results of Random forests and also discuss about status of other algorithm. In this datasets I used several algorithm .but I got only two valuable algorithm such as random forests and DecisionTreeRegressor.

RESULTS AFTER USING RANDOM FOREST

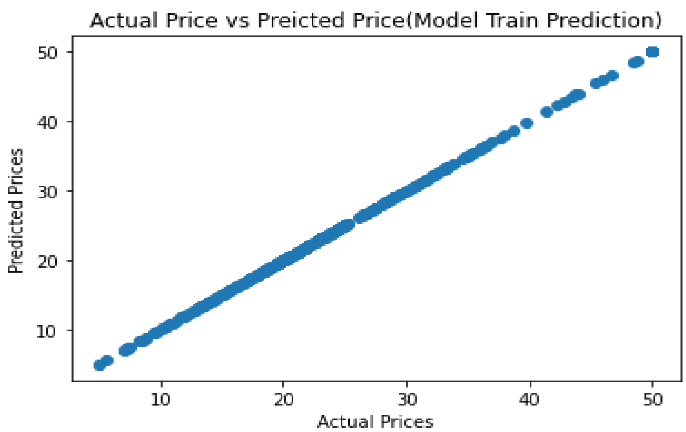
After using random forest ,the Model Train Prediction and Model test prediction both having good result . in model train prediction R squared error :98.271 % and Mean Absolute error :0.7985%. Model Test Prediction R squared error :91.014%.Mean Absolute error:2.283%.

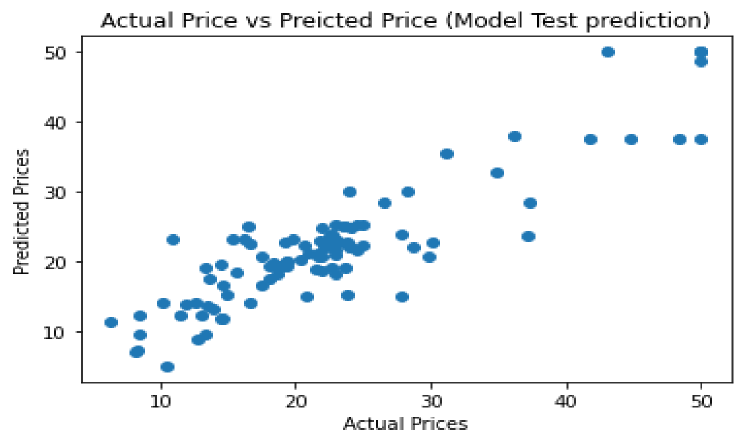




RESULTS AFTER USING DECISIONTREEREGRESSOR

After using DecisionTreeRegressor , the Model Train Prediction and Model test prediction both having good result but this model is little bit underfitting model. Here we see this model train prediction R squared error :100 % and Mean Absolute error :0.00%. Model Test Prediction R squared error : 80.4861%.Mean Absolute error: 3.0637%. so ,we don't use this model.





CONCLUSION

The objective of this paper is to fit models to predict the housing sale price and find some important aspects of the house. In order to achieve my goal, I fit two models to the dataset: random forest and DecisionTreeRegressor .As for the second model - DecisionTreeRegressor, it doesn't meet the assumption of equality of the variances. we have found that the Random forest regression algorithm performsbetter at predicting house prices. How-ever, there is still a difference between the actual prices in our testing data and the prices predicted by the Random forest regression algorithm. random forest is a (mostly) fast, simple and flexible tool, but not without some limitations.

REFERENCE

[1] "scikit-learn: machine learning in Python — scikit-learn 1.0.1 documentation." https://scikit-learn.org/stable/index.html (accessed Nov. 13, 2021).

[2] "Sklearn Random Forest Classifiers in Python - DataCamp." https://www.datacamp.com/community/tutorials/random-forests-classifier-python (accessed Nov. 14, 2021).

[3] N. Donges, "Random Forest Algorithms: A Complete Guide," *Built In*, 2021. https://builtin.com/data-science/random-forest-algorithm (accessed Nov. 14, 2021).

[4] I. Engström and A. Ihre, "Predicting house prices with machine learning methods," 2019, Accessed: Nov. 14, 2021. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-260140.

[5] Wikipedia, "Mean squared error - Wikipedia," 2021. https://en.wikipedia.org/wiki/Mean_squared_error#In_regression (accessed Nov. 14, 2021).

[6] A. Ihre, "Predicting house prices with machine learning methods," *Examensarbete Inom Teknik, Grundnivå, 15 Hp Stockholm, Sverige 2019*, 2019. https://www.diva-portal.org/smash/get/diva2:1354741/FULLTEXT01.pdf (accessed Nov. 14, 2021).

[7] V. S. Pandiri and V. Shiva, "Machine Learning Models to Predict House Prices based on Home Features," Aug. 2017, Accessed: Nov. 14, 2021. [Online]. Available: http://dspace.calstate.edu/handle/10211.3/194683.

[1] "scikit-learn: machine learning in Python — scikit-learn 1.0.1 documentation." https://scikit-learn.org/stable/index.html (accessed Nov. 13, 2021).

[2] "Sklearn Random Forest Classifiers in Python - DataCamp." https://www.datacamp.com/community/tutorials/random-forests-classifier-python (accessed Nov. 14, 2021).

[3] N. Donges, "Random Forest Algorithms: A Complete Guide," *Built In*, 2021. https://builtin.com/data-science/random-forest-algorithm (accessed Nov. 14, 2021).

[4] I. Engström and A. Ihre, "Predicting house prices with machine learning methods," 2019, Accessed: Nov. 14, 2021. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-260140.

[5] Wikipedia, "Mean squared error - Wikipedia," 2021. https://en.wikipedia.org/wiki/Mean_squared_error#In_regression (accessed Nov. 14, 2021).

[6] A. Ihre, "Predicting house prices with machine learning methods," *Examensarbete Inom Teknik, Grundnivå, 15 Hp Stockholm, Sverige 2019*, 2019. https://www.diva-portal.org/smash/get/diva2:1354741/FULLTEXT01.pdf (accessed Nov. 14, 2021).

[7] V. S. Pandiri and V. Shiva, "Machine Learning Models to Predict House Prices based on Home Features," Aug. 2017, Accessed: Nov. 14, 2021. [Online]. Available: http://dspace.calstate.edu/handle/10211.3/194683.