Department of Computer Applications

Asansol Engineering College

Kalyanpur, Sen Raleigh Road, Asansol – 713304

CERTIFICATE

*This is to certify that the project work entitled "...Housing Price Prediction..." is a bonafide record of work carried out in the **Department of Computer Applications**, Asansol Engineering College, Asansol.*

Name	Roll Number	Registration Number
Yash Kumar Dhawan	10871020018	201080571010016
Ria Prasad	10871020024	201080571010022
Lekha Kumari	10871020060	201080571010058
Juhi Kumari	10871020061	201080571010059
Nikhilesh Routh	10871020012	201080571010010

The students of 4th Semester MCA 2020-22 under my / our supervision in requirement of partial fulfillment of the Award of Degree of MCA from Maulana Abul Kalam Azad University of Technology, West Bengal.

Signature of
The Project Guide
Name: Dr./Mr Shubhas Chandra Nath.
Designation: Assistant Professor

Recommendation & Signature of
The Principal
Name: Dr. P. P. Bhattacharya

Recommendation &Signature of
The Head of Department
Name: Dr. D. K. PAL

Recommendation &Signature of
Internal / External Examiners

Housing price prediction

Made by:

Yash Kumar Dhawan

Lekha Kumari

Ria Prasad

Juhi Kumari

Nikhilesh Routh

TOC

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him all the time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Introduction

We need a proper prediction on the real estate and the houses in the buying and selling market. We can see a mechanism that runs throughout the properties buying and selling. Buying a house will be a life time goal for most of the individuals but there are lots of people making huge mistakes when buying properties. Most of the people are buying properties unseen from the people they don't know by seeing the advertisements and all over the grooves coming around the market. One of the common mistakes is buying the properties that are too expensive but not worth it.

According to a survey average house prices rose only 2.5% last year, according to Reuters calculations based on the Reserve Bank of India's House Price Index. That compares with double-digit gains in nearly every other property market.
But as Asia's third-largest economy gets back on track, the housing market was expected to pick up pace. The Feb. 9-28 poll of 13 property analysts predicted national house prices to go up a median 5.0% this year, an upgrade from the 3.75% expected in a December survey. Average house prices were expected to rise 7.0% next year and in 2024.

Many methods have been used in the price prediction like a regression. In this we are trying to predict the real estate house price for the future using machine learning techniques and python. We have used random forest, linear regression and more algorithms with different tools to predict the house price. So, it would be helpful for the people, so they will be aware of both current and future situations, as it may help them avoid making mistakes.

Project Objective

In this project we have a shortened ‘housing price prediction’. In this dataset, the target is to solve problems faced during buying a house. Buying a house is a stressful thing. Buyers are generally not aware of factors that influence the house prices. Our project is a machine learning app, based on certain specifications of your future home it will try to guess the most accurate price.

Our objective in this project is to predict the efficient house pricing for real estate customers with respect to their budgets and priorities. By analysing previous market trends and price ranges, and also upcoming developments future prices will be predicted. In this thesis, we explore how predictive modelling can be applied in housing sale price prediction by analysing the housing dataset and using machine learning models. We tried four different models, namely, linear regression, Decision Tree, Adaboost, Random Forest.

Our methodology for solving the problems in the given project is described below:

- Load the required dataset.
- Study the dataset.
- Describe the dataset.
- Visualise the dataset.
- Find out if the dataset needs to be pre-processed.
    - It will be determined on the basis of whether the dataset has null values or outliers or any such discrepancy that might affect the output of the models to be trained.
    - If the dataset is required to be pre-processed, take the necessary steps to pre-process the data.
    - Find out the principal attributes for training.
    - Split the given dataset for training and testing purposes.
    - Fit the previously split train data in the aforementioned 4 models.
- Calculate the accuracy of the 4 models and find out the classification reports.
    - Plot the necessary graphs.
    - Use each trained model to predict the outcomes of the given test dataset.
    - Choose the best model among the 4 trained models based on the accuracy and classification reports.

Project Scope

The broad scope of ‘Housing Price Prediction' project is given below:

- The given dataset has attributes based on which the status of the location and other factors House price will be predicted.
- It is a useful project as the Classifier models can be used to quickly determine the best price in the large datasets.
- Various real estate marketers can use these models and modify them according to their needs to use in their business as well as of their own. This will reduce the manual labour and time spent on determining the best suited house as per their requirements.
- Customers who intend to take a House can use these trained models to check whether the available house fulfills their needs or not. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI.
- The dataset given to us is a shortened form of the original dataset from UCI. So, the results might have some mismatch with the real-world applications. But that can be avoided if the models are trained accordingly.

Technology

Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Libraries used

Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Sklearn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

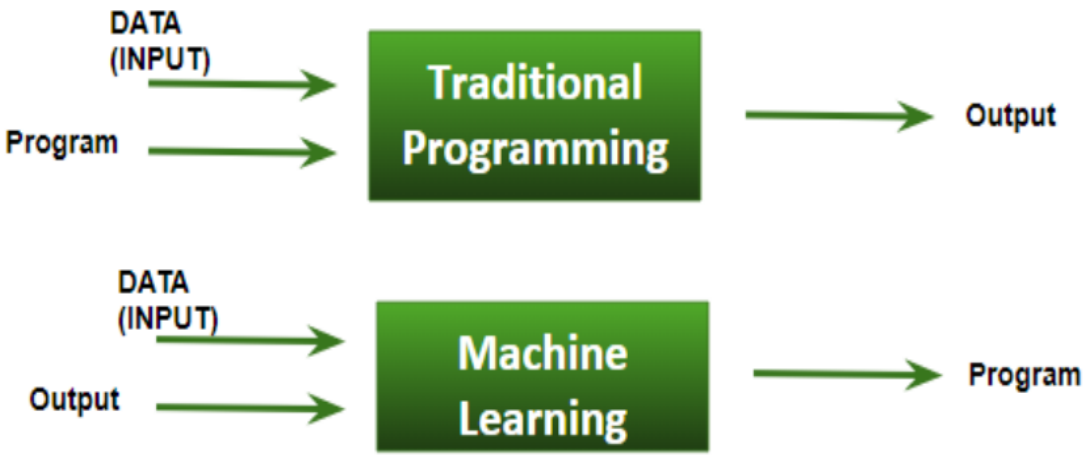
Machine Learning

Machine learning (ML) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks.

It is seen as a part of artificial intelligence.

Machine learning algorithms build a model based on sample data, known as training data.

In order to make predictions or decisions without being explicitly programmed to do so.Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and Computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.



- Traditional Programming : We feed in DATA (Input) + PROGRAM (logic), run it on a machine and get output.
- Machine Learning : We feed in DATA(Input) + Output, run it on a machine during training and the machine creates its own program(logic), which can be evaluated while testing.

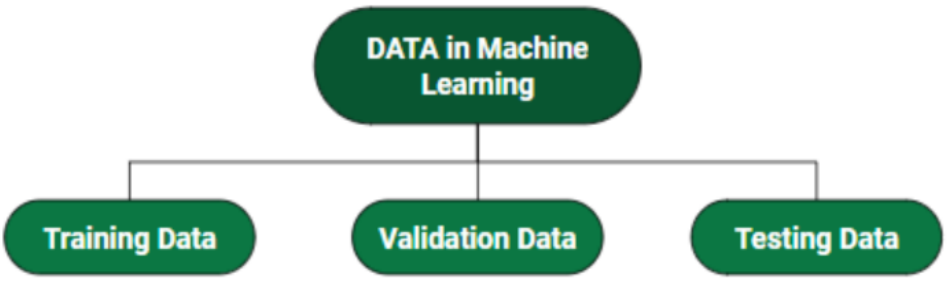
DATA: It can be any unprocessed fact, value, text, sound, or picture that is not being interpreted and analyzed. Without data, we can't train any model and all modern research and automation will go in vain.

Splitting data

Training Data: The part of data we use to train our model. This is the data that your model actually sees(both input and output) and learns from.

Validation Data: The part of data that is used to do a frequent evaluation of the model, fit on the training dataset along with improving involved hyperparameters (initially set parameters before the model begins learning). This data plays its part when the model is actually training.

Testing Data: Once our model is completely trained, testing data provides an unbiased evaluation. When we feed in the inputs of Testing data, our model will predict some values(without seeing actual output). After prediction, we evaluate our model by comparing it with the actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in as training data, set at the time of training.



Data Description

Source of the data: UCI.

The given dataset is a shortened version of the original dataset in UCI.

Data Description: The given train dataset has 404 rows and 14 columns.

Columns	Attribute Name	Type	Target Attribute
CRIM	CRIM	Non-categorical	NO
ZN	ZN	Non-categorical	NO
INDUS	INDUS	Non-categorical	NO
CHAS	CHAS	Non-categorical	NO
NOX	NOX	Non-categorical	NO
RM	RM	Non-categorical	NO
AGE	AGE	Non-categorical	NO
DIS	DIS	Non-categorical	NO
RAD	RAD	Non-categorical	NO
TAX	TAX	Non-categorical	NO
PTRATIO	PTRATIO	Non-categorical	NO
B	B	Non-categorical	NO
LSTAT	LSTAT	Non-categorical	NO
MEDV	MEDV	Non-categorical	YES

Table 1: Data Description

Attribute description

Attribute	Description
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 centres
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000
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
MEDV	Median value of owner-occupied homes in $1000's

Table-2: Attribute description

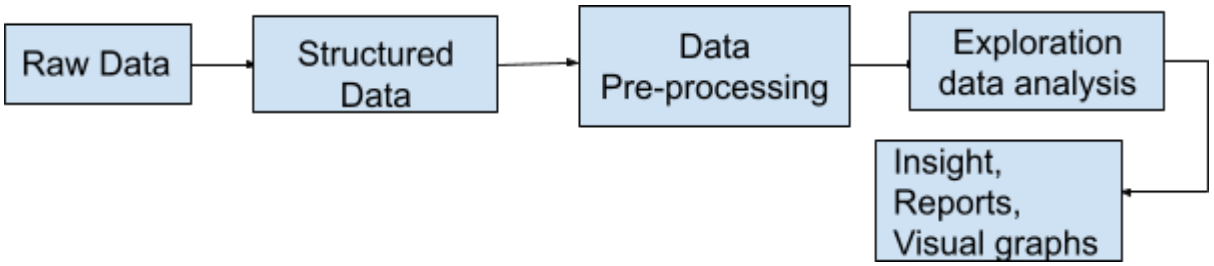
Now we will pre-process the data. The methodology followed is given below:

- Checking for null values:
  If null values are present, we will fill them or drop the row containing the null value based on the dataset.

- Checking for outliers:
  If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

Data Pre-Processing

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.



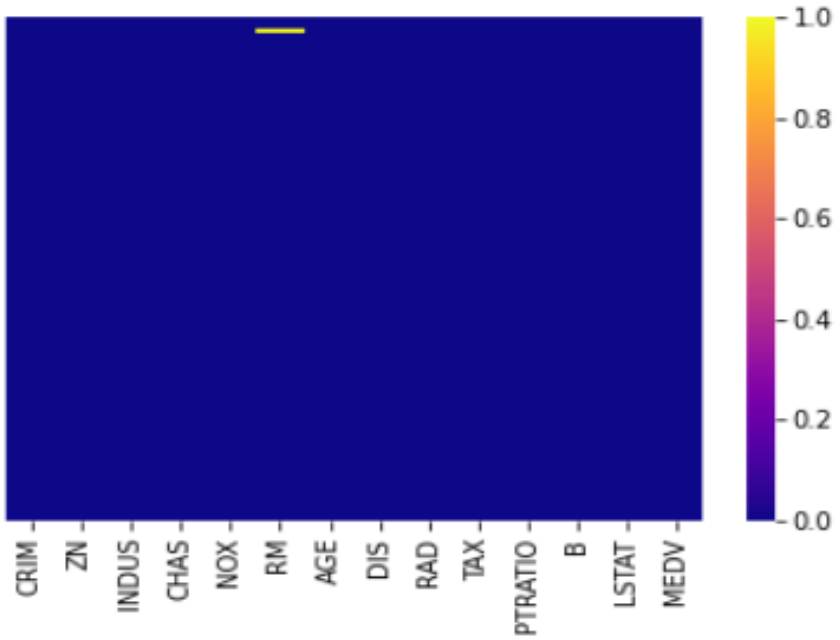
We searched for null values in our dataset and formed the following table:

Column	Count of Null Values
CRIM	0
ZN	0
INDUS	0
CHAS	0
NOX	0
RM	5
AGE	0
DIS	0
RAD	0
TAX	0
PTRATIO	0
B	0

LSTAT	0
MEDV	0

Table-3: Count of null values

To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:



Heatmap of the given Data

The heatmap shows that the dataset has null values.

To remove the null values, we had filled the gaps in an attribute with the median value for that particular attribute using SimpleImputer with median as its strategy.

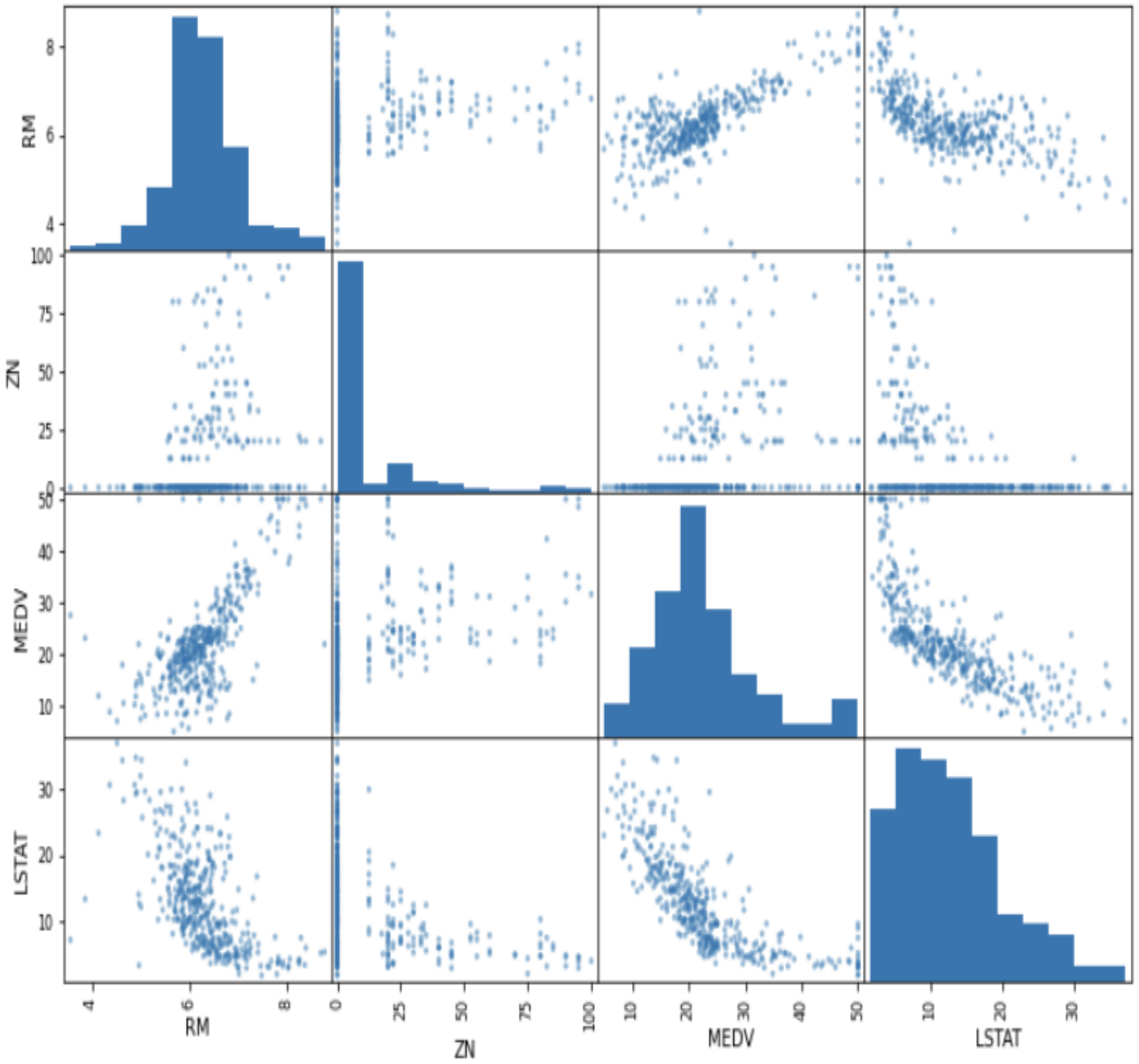
Now we have successfully handled Null values.

So, we are moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. 'MEDV' column in the dataset. The following table gives the correlation value of each attribute with our target attribute i.e. ‘MEDV’ :

Columns	Correlation Values
RM	0.679821
B	0.361761
ZN	0.339741
DIS	0.240451
CHAS	0.205066
AGE	-0.36456
RAD	-0.374693
CRIM	-0.393715
NOX	-0.422873
TAX	-0.456657
INDUS	-0.473516
PTRATIO	-0.493534
LSTAT	-0.740494

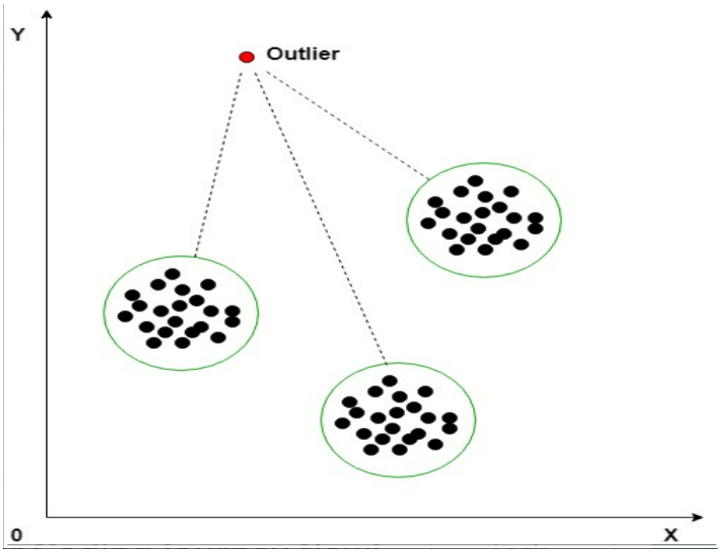
Table-4: Correlation values with Target Attribute

Showing Scatter Matrix of highest correlating features in the datasets which is ‘RM’, ‘ZN’ and ‘LSTAT’ with the target Attribute ‘MEDV’.



Scatter Matrix of highest correlated attributes

Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.The analysis of outlier data is referred to as outlier analysis or outlier mining.



Most common causes of outliers on a data set:

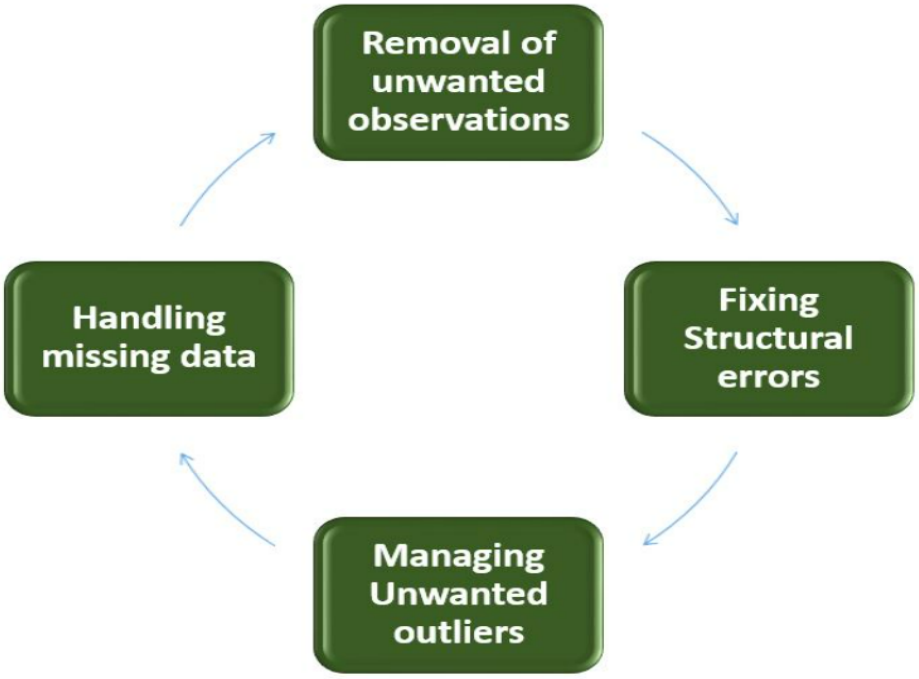
- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (data extraction or experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation or data set unintended mutations)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

Data Cleaning

Data cleaning is one of the important parts of machine learning. It plays a significant part in building a model. It surely isn't the fanciest part of machine learning and at the same time, there aren't any hidden tricks or secrets to uncover. However, the success or failure of a project relies on proper data cleaning. Professional data scientists usually invest a very large portion of their time in this step because of the belief that **"Better data beats fancier algorithms"**.

If we have a well-cleaned dataset, there are chances that we can achieve good results with simple algorithms also, which can prove very beneficial at times especially in terms of computation when the dataset size is large.

Steps involved in Data Cleaning:



Some data cleansing tools :

- Openrefine
- Trifacta Wrangler
- TIBCO Clarity
- Cloudingo
- IBM Infosphere Quality Stage

Model building

Splitting data for training and testing purposes.

We split the given dataset into two parts for training and testing purposes. The split ratio we used is 8:2 which indicates 80% of the data is used for training purposes and the remaining 20% of the data is used for the purpose of testing.

Now we will be training our required models. Our project goal requires us to train specific 4 classifier models viz.

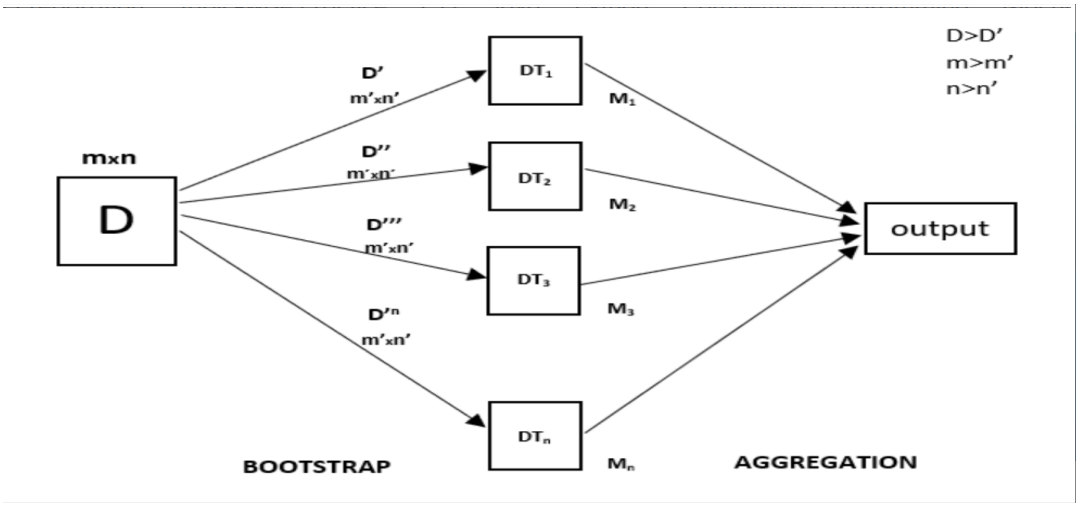
1. Random Forest Model
2. Decision Tree Model
3. AdaBoost Model
4. Linear Regression Model

We will be using the final dataset obtained after pre-processing the given train dataset to train our required models.

Random Forest Model

Random forest is an algorithm which can be used both for classification and regression. Random forest models are constructed by using a collection of decision trees based on the training data. Instead of taking the target value from a single tree, the Random forest algorithm makes a prediction on the average prediction of a collection of trees. The decision trees themselves are constructed by fitting to randomly drawn groups of rows and columns in the training data. This method is called bagging, and results in a reduction of bias as each tree is built on different parts of the input at random. The method of averaging the predictions of decision trees reduces the overfitting that can occur when using single decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap. In the case of a regression problem, the final output is the mean of all the outputs. This part is called **Aggregation**.



Here is the graph of the Random Forest Model which consists of the values of mean, standard deviation and RMSE which stands for Root Mean Square Error.

The formula for calculating **RMSE** is:

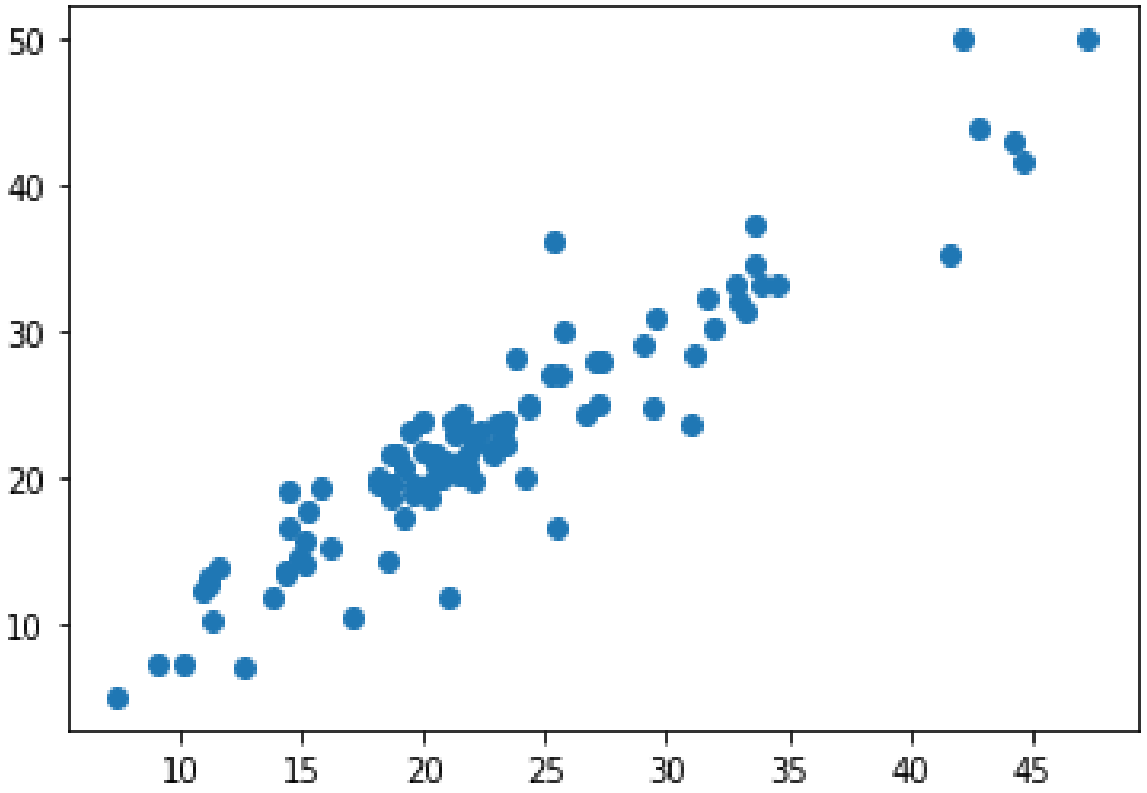
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Mean: 3.3232811423389905

Standard deviation: 0.6323460275894798

RMSE : 2.95013707724061

Scatter Plot



Scatter Plot of Random Forest Model

The following table consists of the feature importance obtained for our random forest classifier:

Attributes	Feature Importance
MEDV	1.000000
RM	0.679821
B	0.361761
ZN	0.339741
DIS	0.240451
CHAS	0.205066
AGE	-0.364596
RAD	-0.374693
CRIM	-0.393715
NOX	-0.422873
TAX	-0.456657
INDUS	-0.473516
PTRATIO	-0.493534
TAXRM	-0.527643
LSTAT	-0.740494

Table-5

As we can see from the calculated feature importance values, the attribute viz. 'MEDV' has higher values compared to other attributes. So, we will select this attribute as our principal attribute to train our required models.

Now we obtained a final dataset which we will use for training purpose of our required models.

Given below is the description of the final dataset obtained:

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.283138
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.715738
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.878750
50%	0.286735	0.00000	9.900000	0.000000	0.538000	6.217500
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.632000
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000

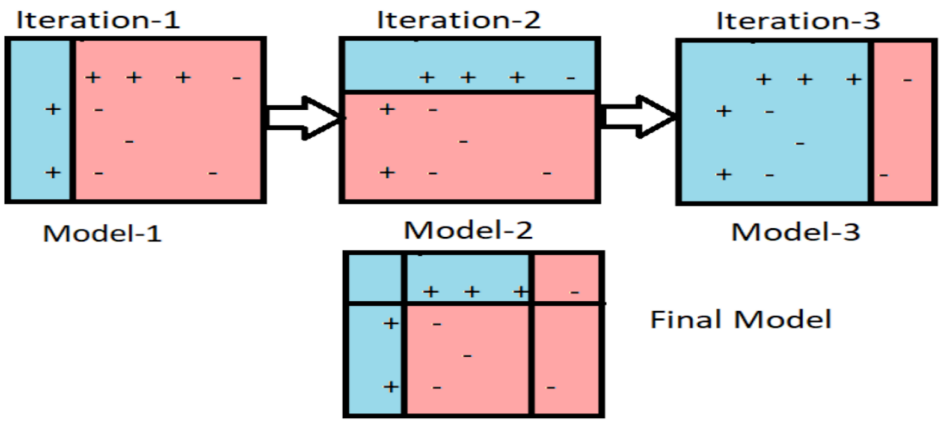
Table-6

	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.00000
mean	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.791609
std	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.235740
min	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	1.730000
25%	44.850000	2.035975	4.000000	284.000000	17.400000	374.617500	6.847500
50%	78.200000	3.122200	5.000000	337.000000	19.000000	390.955000	11.570000
75%	94.100000	5.100400	24.000000	666.000000	20.200000	395.630000	17.102500
max	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	36.98000

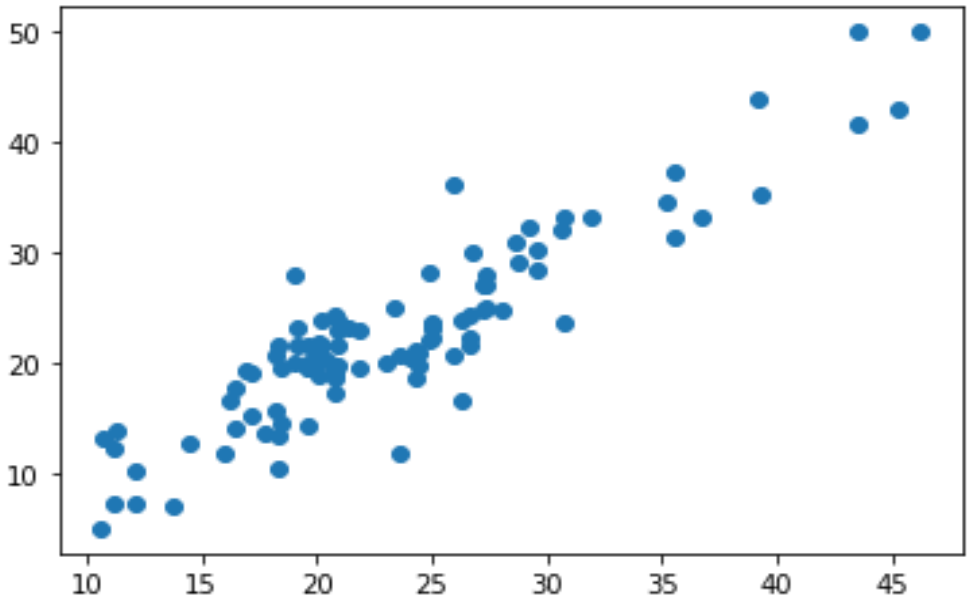
Table:7

AdaBoost Model

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. These trees are also called Decision Stumps. This algorithm does is that it builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.



Scatter Plot



Scatter Plot of AdaBoost Model

The main key of this algorithm is in the way they give weights to the instances in the dataset. Due to this the algorithm needs to pay less attention to the instances while constructing subsequent models.

Observations from the Scatter plot :

Mean: 3.6912528076335667

Standard deviation: 0.7419117853727191

RMSE : 3.56136463924601

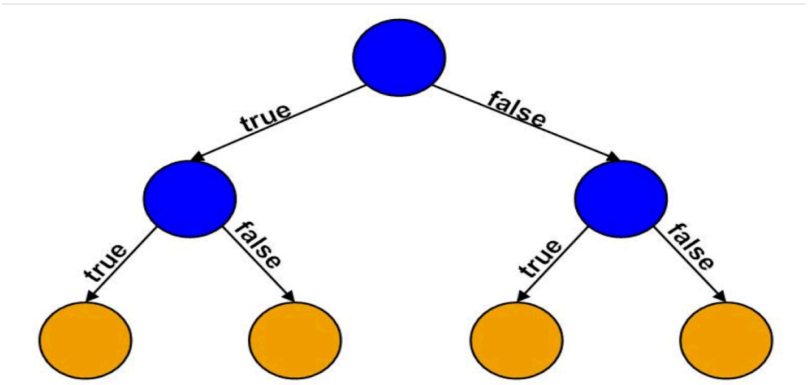
Decision Tree Model

Decision tree algorithm falls under the category of supervised learning.

Decision tree learning uses a decision tree as a predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining, and machine learning. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunction of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real regression) are called regression trees. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making.

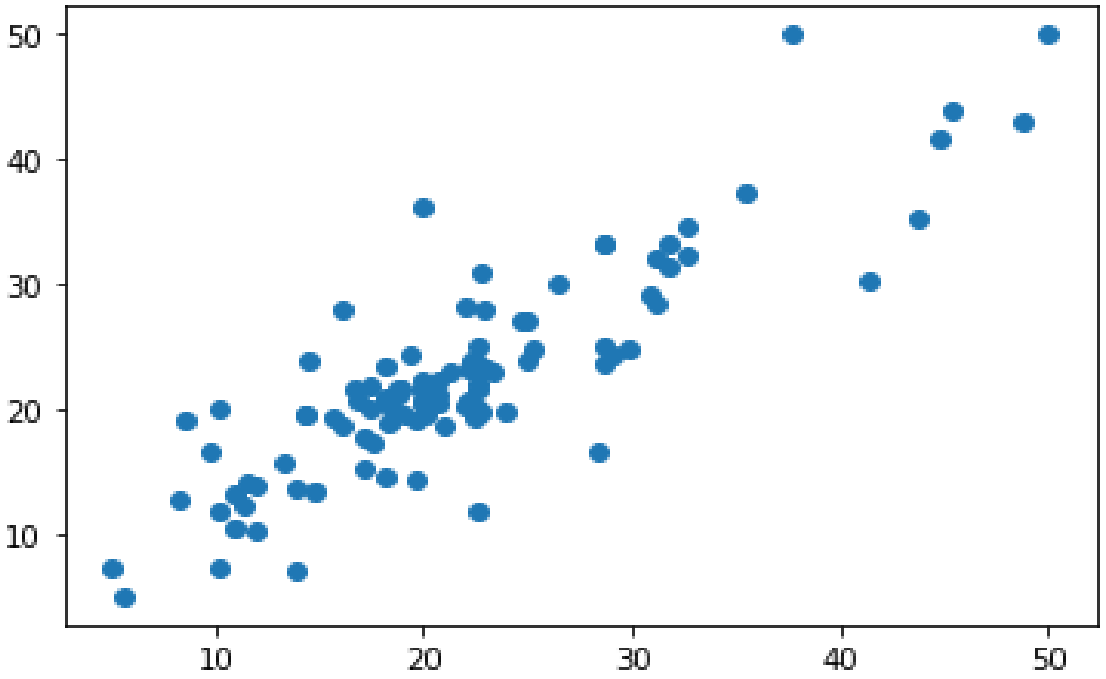
The decision tree regressor observes features of an attribute and trains a model in the form of a tree to predict data in the future to produce meaningful output. Decision tree regressor learns from the max depth, min depth of a graph and according to the system analyzes the data. Grid Search CV is a way to deal with parameter tuning that will efficiently manufacture and assess a model for every mix of calculation parameters indicated in a grid. Grid Search CV in this algorithm is used to assess the best value for max-depth, using which the decision tree is constructed.

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, it’s also widely used in machine learning.



Scatter plot that represents graph from our data set;

Scatter Plot



Scatter Plot of Decision Tree model

Observations from the Scatter plot :

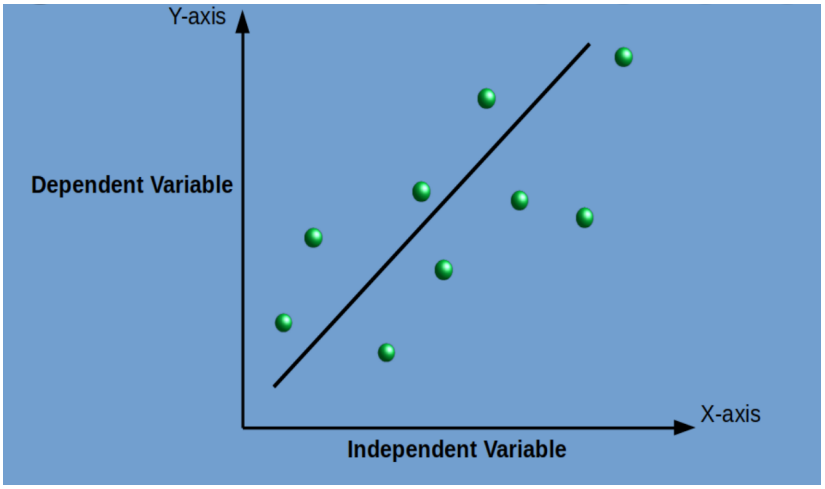
Mean: 4.191768321080035
Standard deviation: 0.9005646878620126
RMSE : 4.49651281005407

Linear Regression:

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

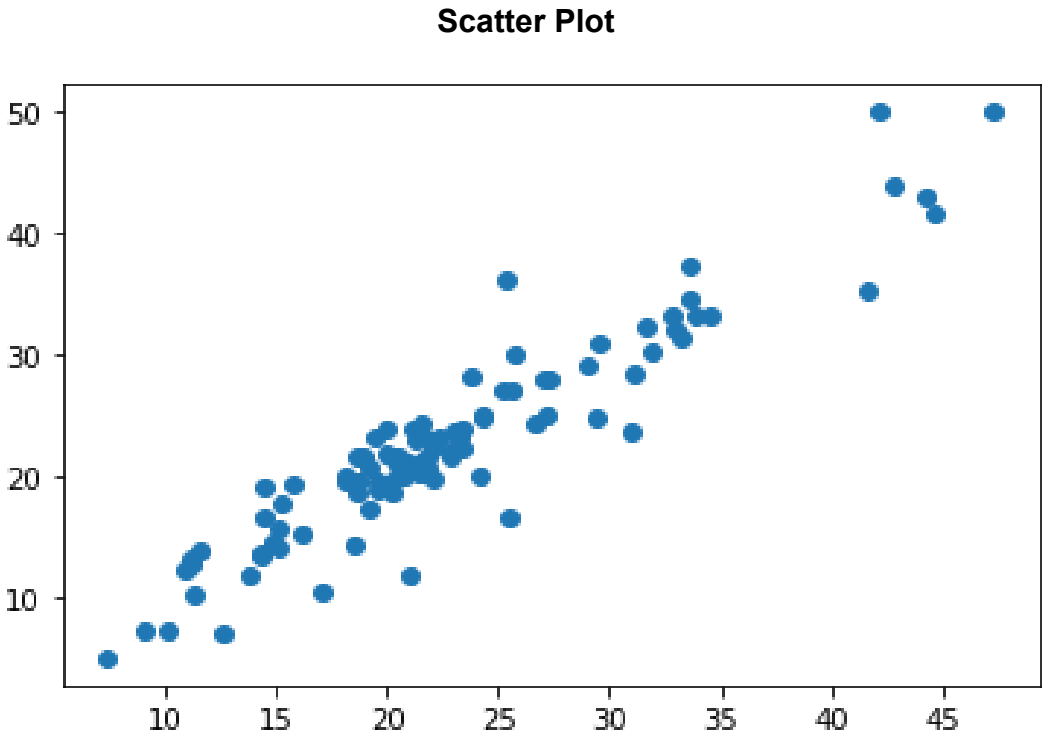
In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.



linear regression

Scatter plot that represents our dataset:



Scatter Plot of Linear Regression

Observations from the Scatter plot :

Mean: 5.032043501135085

Standard deviation: 1.0578695545386638

RMSE : 4.14187936743179

Comparison of the Models trained :

We trained 4 models using the 4 algorithms viz.

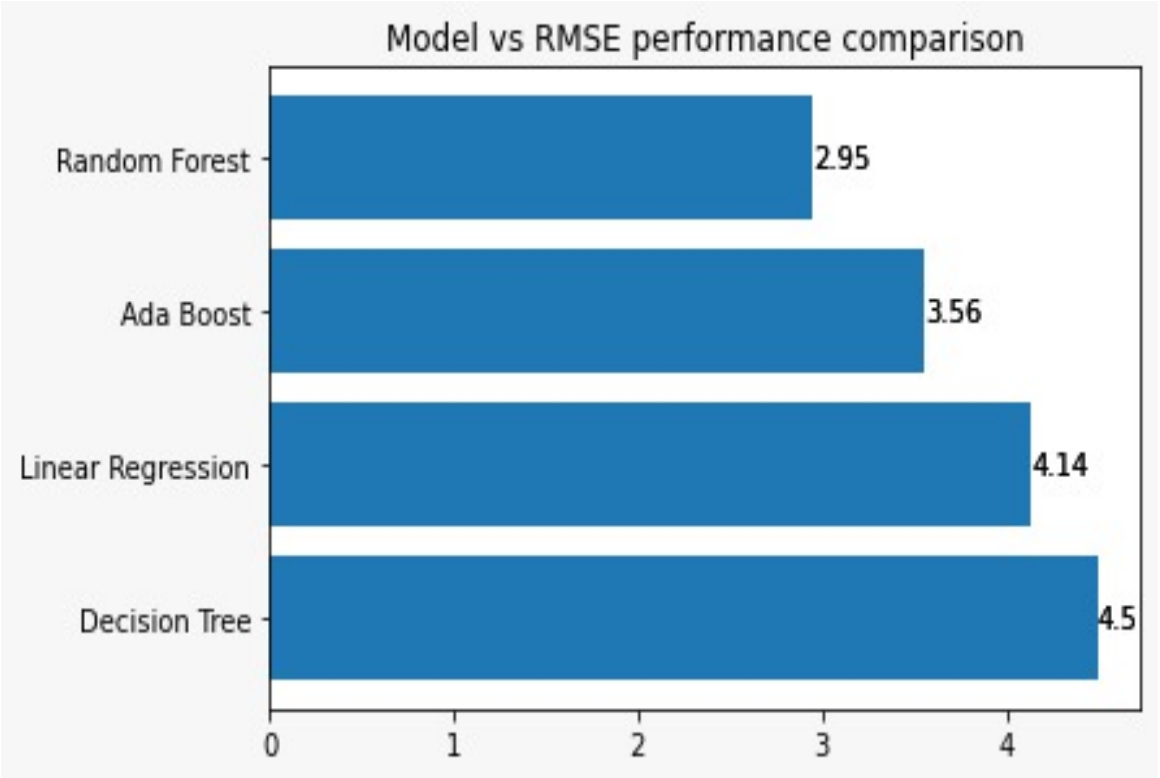
1. Random Forest Model
2. AdaBoost Model
3. Decision Tree Model
4. Linear Regression Model

The 4 models had different accuracy. The comparison of the accuracies of the models are given below:

Model	Accuracy %
Random Forest Model	2.95013707724061
AdaBoost Model	3.56136463924601
Decision Tree Model	4.49651281005407
Linear Regression Model	4.14187936743179

Table-8: Accuracy Comparison table

The following bar graph shows the accuracy comparison in graphical way:



Comparison of accuracy of the 4 different models trained

Test Dataset

We were given a test dataset for this loan approval problem. We pre-process the given test dataset in a similar way we pre-processed our train dataset. The methodology followed is given below:

Change the categorical values to numeric values using same process we used during changing values in our train dataset.

Checking for null values.
If null values are present, we will fill them or drop the row containing the null value based on the dataset.

Checking for outliers.
If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

N.B.: We handled the outliers using a non-parametric statistics method. We used Median Absolute Deviation or MAD to handle the outliers. In this method, data outside the range 3 * mad is excluded and replaced with the median of the data. This is the same procedure we used to handle outliers in our train dataset.

After pre-processing of the test dataset had been done, we retrained the models with 100% of the train dataset given and used those trained models to predict the outcome for each input in the test dataset.

There were a total of 102 inputs in the test dataset.

Source code

Importing module

```
In [1]: import pandas as pd
```

Loading Data Set

```
In [1]: housing = pd.read_csv("data.csv")
```

Getting Details of Data

```
In [3]: # Get a glance at the data
housing.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Checking entries and missing data

In [4]:
```
# Check entries of each type. Check for any missing data.
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       501 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
```

```
In [36]: # Printing the null value count of each field
         print("Null values count:\n", housing.isnull().sum())
```

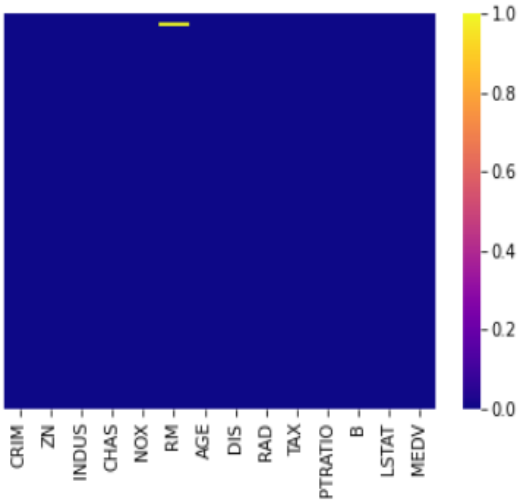
```
Null values count:
 CRIM       0
ZN          0
INDUS       0
CHAS        0
NOX         0
RM          5
AGE         0
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT       0
MEDV        0
```

As we find out, we have 5 null values in the RM attribute.

Lets see the graphical representation for the same.

```
In [42]: import seaborn as sb
         sb.heatmap(housing.isnull(), yticklabels = False, cbar = True, cmap = 'plasma')
```

```
Out[42]: <AxesSubplot:>
```



Counting Value for an attribute with binary values(0/1)

In [5]:
```
#Count the value of each type
housing['CHAS'].value_counts()
```

Out[5]:
```
0    471
1     35
Name: CHAS, dtype: int64
```

In [6]:
```
housing.describe()
```

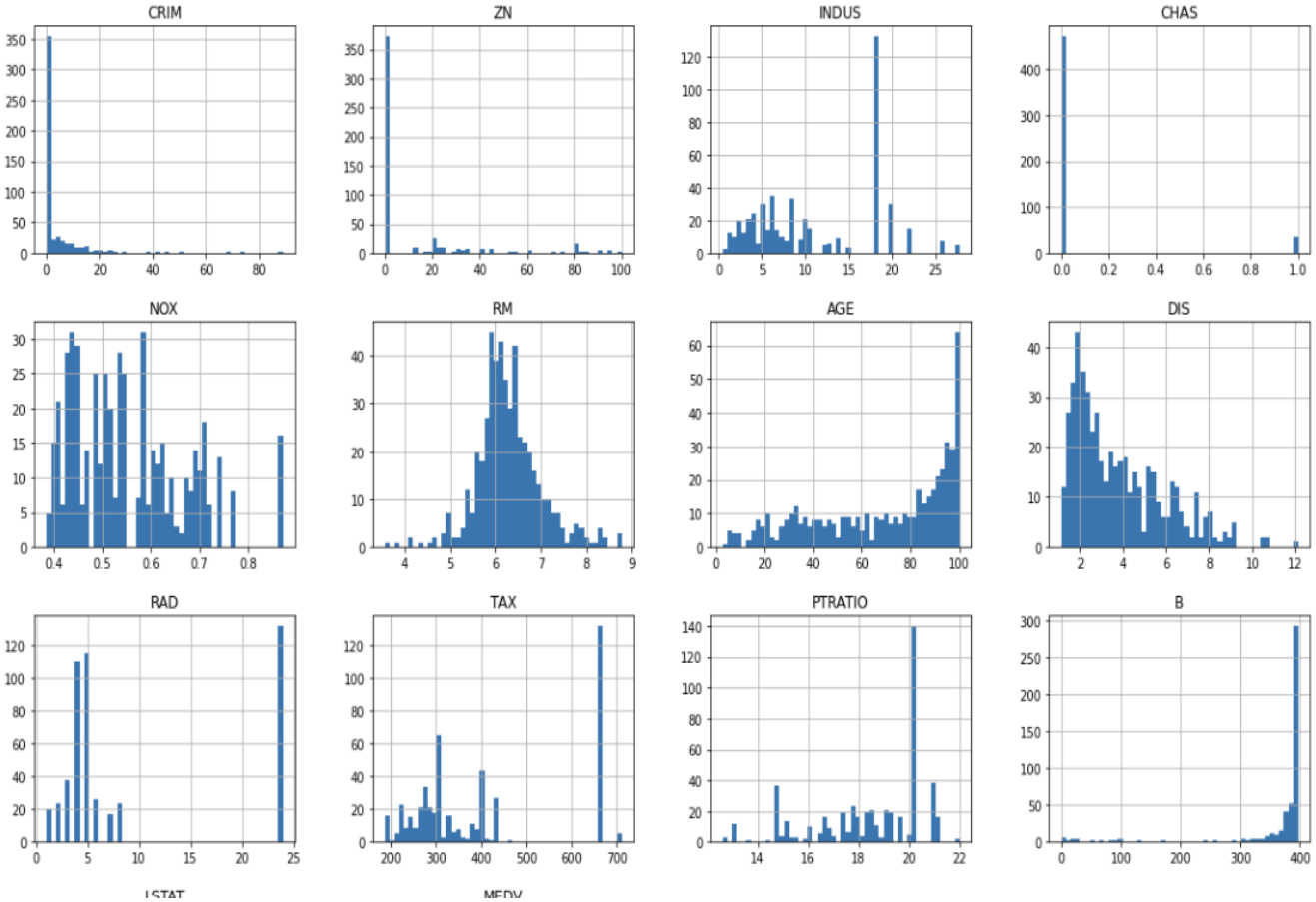
Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.287920	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.6
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.705287	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.1
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	1.7
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885000	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.9
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.211000	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.629000	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.9
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	37.9

Histogram representation of all attributes

```
In [8]: import matplotlib.pyplot as plt
```

```
In [9]: housing.hist(bins = 50, figsize = (20, 15))
        plt.show()
```



Split Representation of Test Data

In [10]:
```
from sklearn .model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size = 0.2, random_state=42)
```

In [11]:
```
test_set
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
173	0.09178	0.0	4.05	0	0.510	6.416	84.1	2.6463	5	296	16.6	395.50	9.04	23.6
274	0.05644	40.0	6.41	1	0.447	6.758	32.9	4.0776	4	254	17.6	396.90	3.53	32.4
491	0.10574	0.0	27.74	0	0.609	5.983	98.8	1.8681	4	711	20.1	390.11	18.07	13.6
72	0.09164	0.0	10.81	0	0.413	6.065	7.8	5.2873	4	305	19.2	390.91	5.52	22.8
452	5.09017	0.0	18.10	0	0.713	6.297	91.8	2.3682	24	666	20.2	385.09	17.27	16.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
412	18.81100	0.0	18.10	0	0.597	4.628	100.0	1.5539	24	666	20.2	28.79	34.37	17.9
436	14.42080	0.0	18.10	0	0.740	6.461	93.3	2.0026	24	666	20.2	27.49	18.05	9.6
411	14.05070	0.0	18.10	0	0.597	6.657	100.0	1.5275	24	666	20.2	35.05	21.22	17.2
86	0.05188	0.0	4.49	0	0.449	6.015	45.1	4.4272	3	247	18.5	395.99	12.86	22.5
75	0.09512	0.0	12.83	0	0.437	6.286	45.0	4.5026	5	398	18.7	383.23	8.94	21.4

102 rows × 14 columns

Split Representation of Training Data

In [12]:
```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['CHAS']):
    strat_train_data = housing.loc[train_index]
    strat_test_data = housing.loc[test_index]
```

In [13]:
```
strat_train_data
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57	21.9
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99	24.5
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68	16.7
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6.87	23.1
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6.15	23.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
155	3.53501	0.0	19.58	1	0.871	6.152	82.6	1.7455	5	403	14.7	88.01	15.02	15.6
423	7.05042	0.0	18.10	0	0.614	6.103	85.1	2.0218	24	666	20.2	2.52	23.29	13.4
98	0.08187	0.0	2.89	0	0.445	7.820	36.9	3.4952	2	276	18.0	393.53	3.57	43.8
455	4.75237	0.0	18.10	0	0.713	6.525	86.5	2.4358	24	666	20.2	50.92	18.13	14.1
216	0.04560	0.0	13.89	1	0.550	5.888	56.0	3.1121	5	276	16.4	392.80	13.51	23.3

404 rows × 14 columns

CHAS[0/1] value count under training data

In [14]:
```
housing  = strat_train_data.copy()
```

In [15]:
```
strat_train_data['CHAS'].value_counts()
```

Out[15]:
```
0    376
1     28
Name: CHAS, dtype: int64
```

Correlation

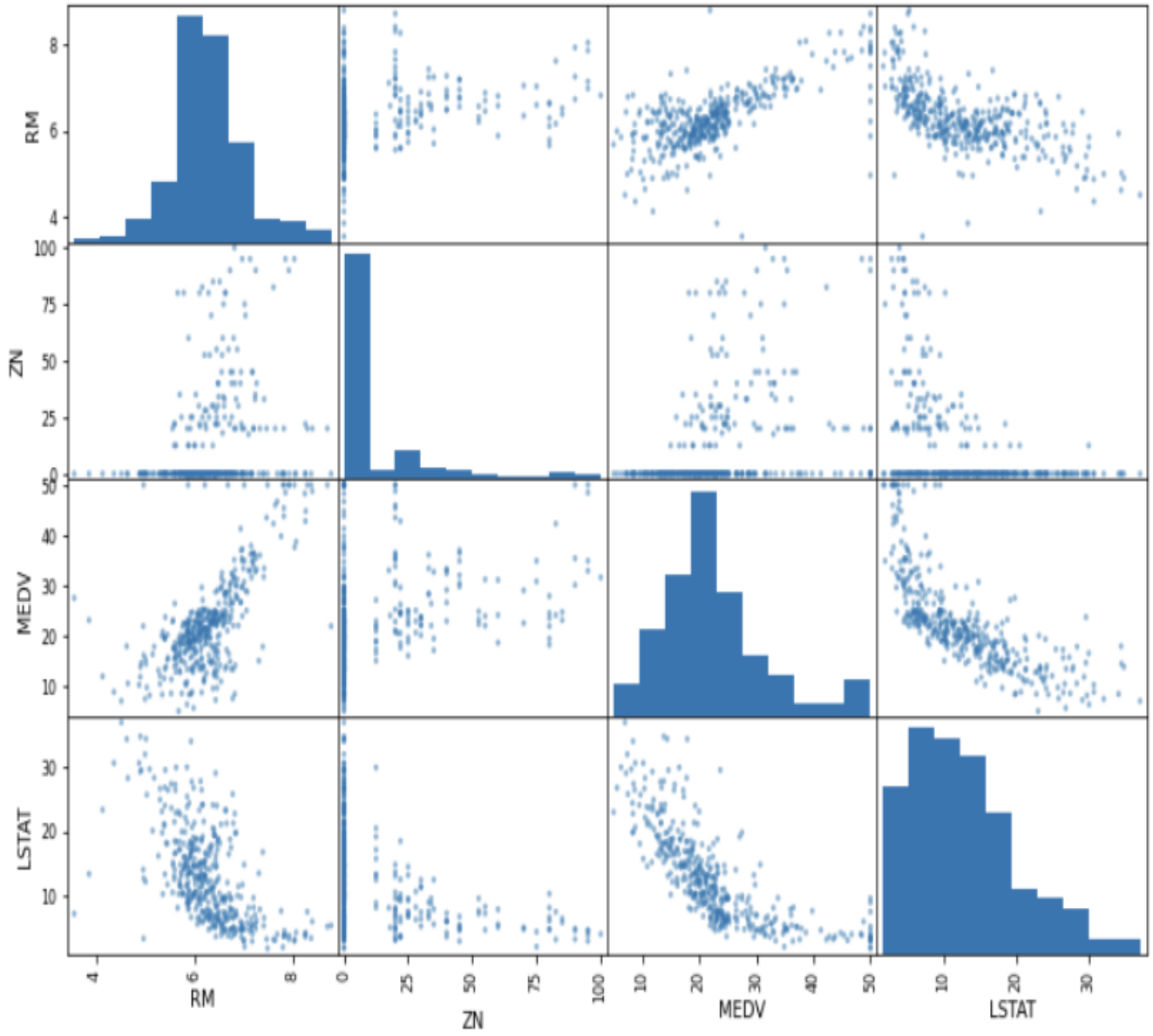
```
In [16]: corr_matrix = housing.corr()
         corr_matrix['MEDV'].sort_values(ascending = False)
```

```
Out[16]: MEDV       1.000000
         RM         0.679821
         B          0.361761
         ZN         0.339741
         DIS        0.240451
         CHAS       0.205066
         AGE       -0.364596
         RAD       -0.374693
         CRIM      -0.393715
         NOX       -0.422873
         TAX       -0.456657
         INDUS     -0.473516
         PTRATIO   -0.493534
         LSTAT     -0.740494
```

Plotting between high correlating features

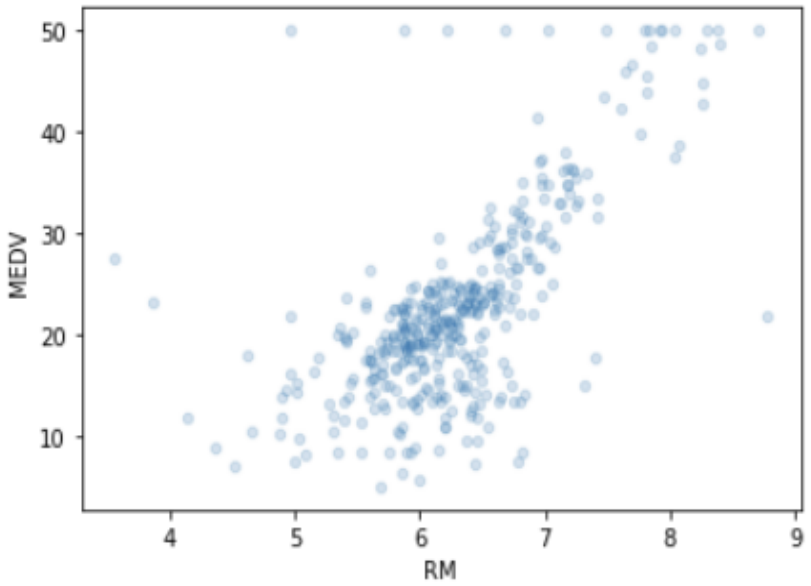
```
In [17]: from pandas.plotting import scatter_matrix
         attributes = ['RM', 'ZN', 'MEDV', 'LSTAT']
         scatter_matrix(housing[attributes], figsize=(12, 8))
```

```
Out[17]: array([[<AxesSubplot:xlabel='RM', ylabel='RM'>,
                 <AxesSubplot:xlabel='ZN', ylabel='RM'>,
                 <AxesSubplot:xlabel='MEDV', ylabel='RM'>,
                 <AxesSubplot:xlabel='LSTAT', ylabel='RM'>],
                [<AxesSubplot:xlabel='RM', ylabel='ZN'>,
                 <AxesSubplot:xlabel='ZN', ylabel='ZN'>,
                 <AxesSubplot:xlabel='MEDV', ylabel='ZN'>,
                 <AxesSubplot:xlabel='LSTAT', ylabel='ZN'>],
                [<AxesSubplot:xlabel='RM', ylabel='MEDV'>,
                 <AxesSubplot:xlabel='ZN', ylabel='MEDV'>,
                 <AxesSubplot:xlabel='MEDV', ylabel='MEDV'>,
                 <AxesSubplot:xlabel='LSTAT', ylabel='MEDV'>],
                [<AxesSubplot:xlabel='RM', ylabel='LSTAT'>,
                 <AxesSubplot:xlabel='ZN', ylabel='LSTAT'>,
                 <AxesSubplot:xlabel='MEDV', ylabel='LSTAT'>,
                 <AxesSubplot:xlabel='LSTAT', ylabel='LSTAT'>]], dtype=object)
```


RM
ZN
MEDV
LSTAT
RM
ZN
MEDV
LSTAT

Individual plotting between two correlating features(RM, MEDV)

```
In [18]: housing.plot(kind='scatter', x='RM', y='MEDV', alpha=0.2)
Out[18]: <AxesSubplot:xlabel='RM', ylabel='MEDV'>
```



Attribute combination

Here we have tried to introduce a new attribute(TAXRM) combining two existing attributes(TAX and RM) having high correlation.

```
In [19]: housing['TAXRM'] = housing['TAX'] / housing['RM']
         housing.head()
```

Out[19]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	TAXRM
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57	21.9	51.571709
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99	24.5	42.200452
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68	16.7	102.714374
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6.87	23.1	45.012547
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6.15	23.0	45.468948

As we see in the following output, the newly added attribute(TAXRM) has a standard correlating value too.

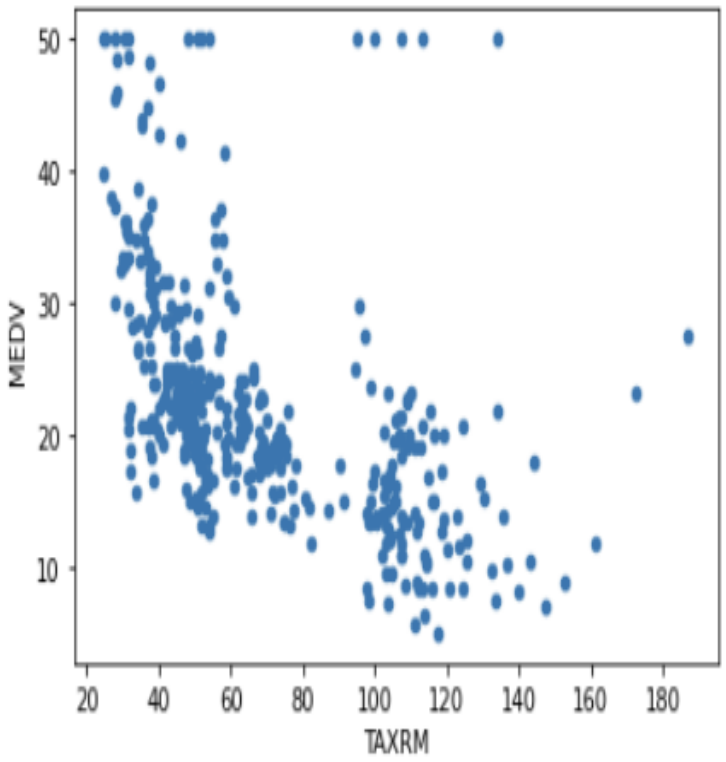
```
In [20]: corr_matrix = housing.corr()
         corr_matrix['MEDV'].sort_values(ascending=False)
```

```
Out[20]: MEDV       1.000000
         RM         0.679821
         B          0.361761
         ZN         0.339741
         DIS        0.240451
         CHAS       0.205066
         AGE       -0.364596
         RAD       -0.374693
         CRIM      -0.393715
         NOX       -0.422873
         TAX       -0.456657
         INDUS     -0.473516
         PTRATIO   -0.493534
         TAXRM     -0.527643
         LSTAT     -0.740494
         Name: MEDV, dtype: float64
```

Plotting between TAXRM vs MEDV

```
In [21]: housing.plot(kind='scatter', x='TAXRM', y='MEDV', alpha=1)
```

```
Out[21]: <AxesSubplot:xlabel='TAXRM', ylabel='MEDV'>
```




```
In [22]: housing = strat_train_data.drop('MEDV', axis=1)
         housing_labels = strat_train_data['MEDV'].copy()
```

Missing attributes

In [23]:
```
# There are three options to take care of missing attributes:
# 1. Get rid of the missing data points
# 2. Get rid of the whole attribute
# 3. Set the missing values to 0 or mean or median
```

In [24]:
```
# Option 1
# Till we mention emplace=True no changes will be made to the housing data
# and it will just return a copy of the new data
a = housing.dropna(subset=['RM'])
a.shape
```

Out[24]: (400, 13)

In [25]:
```
#Option 2
housing.drop("RM", axis=1).shape
# Note that the RM column will be dropped in the newly obtained copy of housing data
# The number of columns is reduced to 14 from 15
# Also note that the original housing data will remain unchanged
```

Out[25]: (404, 12)

In [26]:
```
median = housing['RM'].median()
median
```

Out[26]: 6.2175

In [27]:
```
# Option 3
housing['RM'].fillna(median)
```

Out[27]:
```
254    6.108
348    6.635
476    6.484
321    6.376
326    6.312
       ...
155    6.152
423    6.103
98     7.820
455    6.525
216    5.888
Name: RM, Length: 404, dtype: float64
```

```
In [28]: housing['RM'].shape
Out[28]: (404,)

In [29]: housing.shape
Out[29]: (404, 13)
```

```
In [30]: housing.describe()
```

Out[30]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	404.000000	404.000000	404.000000	404.000000	404.000000	400.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.0
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.283138	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.7
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.715738	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.2
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	1.7
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.878750	44.850000	2.035975	4.000000	284.000000	17.400000	374.617500	6.8
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.217500	78.200000	3.122200	5.000000	337.000000	19.000000	390.955000	11.5
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.632000	94.100000	5.100400	24.000000	666.000000	20.200000	395.630000	17.1
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	36.9

```
In [31]: from sklearn.impute import SimpleImputer
         imputer = SimpleImputer(strategy = 'median')
         imputer.fit(housing)
```

```
In [32]: imputer.statistics_
```

```
Out[32]: array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
                6.21750e+00, 7.82000e+01, 3.12220e+00, 5.00000e+00, 3.37000e+02,
                1.90000e+01, 3.90955e+02, 1.15700e+01])
```

```
In [33]: x = imputer.transform(housing)
```

```
In [34]: housing_new = pd.DataFrame(x, columns=housing.columns)
         housing_new.shape
```

```
Out[34]: (404, 13)
```

```
In [35]: housing_new.describe()
```

Out[35]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.0
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.282488	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.7
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.712207	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.2
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	1.7
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.879750	44.850000	2.035975	4.000000	284.000000	17.400000	374.617500	6.8
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.217500	78.200000	3.122200	5.000000	337.000000	19.000000	390.955000	11.5
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630250	94.100000	5.100400	24.000000	666.000000	20.200000	395.630000	17.1
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	36.9

Pipeline

In [36]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
myPipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('std scaler', StandardScaler())
])
```

In [37]:
```
housing_num_new = myPipeline.fit_transform(housing_new)
```

In [38]:
```
housing_num_new
```

Out[38]:
```
array([[-0.43942006,  3.12628155, -1.12165014, ..., -0.97491834,
         0.41164221, -0.86091034],
       [-0.44352175,  3.12628155, -1.35893781, ..., -0.69277865,
         0.39131918, -0.94116739],
       [ 0.15682292, -0.4898311 ,  0.98336806, ...,  0.81196637,
         0.44624347,  0.81480158],
       ...,
       [-0.43525657, -0.4898311 , -1.23083158, ..., -0.22254583,
         0.41831233, -1.27603303],
       [ 0.14210728, -0.4898311 ,  0.98336806, ...,  0.81196637,
        -3.15239177,  0.73869575],
       [-0.43974024, -0.4898311 ,  0.37049623, ..., -0.97491834,
         0.41070422,  0.09940681]])
```

In [39]:
```
housing_num_new.shape
```

Out[39]:
```
(404, 13)
```

Model contrasting

Linear Regression

Selecting a model

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
model = LinearRegression()
# model = DecisionTreeRegressor()
# model = AdaBoostRegressor()
# model = RandomForestRegressor()
model.fit(housing_num_new, housing_labels)
```

```
▾ LinearRegression
LinearRegression()
```

Model evaluation

```
import numpy as np
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_new)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

```
mse
```

```
23.339190986772923
```

Using better evaluation technique: Cross validation

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_new, housing_labels, scoring='neg_mean_squared_error', cv=10)
rmse_scores = np.sqrt(-scores)
```

```
rmse_scores
```

```
array([4.21344506, 4.25892467, 5.10040197, 3.84937261, 5.34544966,
       4.37746471, 7.46842477, 5.4912994 , 4.15006097, 6.06559119])
```

```
def print_scores(scores):
    print("Scores: ", scores)
    print("Mean: ", scores.mean())
    print("Standard deviation: ", scores.std())
```

```
print_scores(rmse_scores)
```

```
Scores:  [4.21344506 4.25892467 5.10040197 3.84937261 5.34544966 4.37746471
 7.46842477 5.4912994  4.15006097 6.06559119]
Mean:  5.032043501135085
Standard deviation:  1.0578695545386638
```

Testing model against test data

```
x_test = strat_test_data.drop('MEDV', axis=1)
y_test = strat_test_data['MEDV'].copy()
x_ready = myPipeline.transform(x_test)
final_predictions = model.predict(x_ready)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
print(final_predictions, '\n', list(y_test))
```

```
[22.65162407 17.22518999 30.01241486 30.72336136  8.81545633 13.31241205
 17.26347963 17.73849451 32.49106968 36.05339992 16.3623327   0.5587445
 22.97101227 20.46009882 20.01741986 12.93905755 31.15356226 13.40765401
 25.00668445 24.16396532 20.41213851 17.05084195 17.80585097 25.58596586
 19.51509285 32.85972862 19.42430063 33.72664932  8.02404507 34.67065873
 19.50209901 21.43788013 29.30576277 16.35267073 26.97982996 18.35464122
 37.30171381 24.53426593 22.17895719 37.15193206 25.14237742 34.47608627
 23.44419191 23.98350521 18.5202786  32.68555692 38.4545224  21.37097626
 17.64421563 16.22550737 21.23145923 12.38342884 19.85377526 20.40220783
 27.91570132 33.0694495  40.12097174 31.36388554 14.94354534 20.73100271
 40.49290758 18.11169065 15.09007198 27.65525894 19.49741525 32.49392077
 23.40509752 20.30582366 21.13606742 33.81151323 34.11311759 27.62516391
 24.50098481 21.85485186 36.17141346  8.58520595 17.40079078 21.45622819
 20.51014981 22.98020194 25.97538003 22.54412794 14.19663395 25.42717414
 21.20418679 23.80974179 19.88337555 21.70360292 22.4734981  21.66272935
 17.77015369 28.34914223  7.43450461 28.45854137 18.8525554  30.8376402
 20.50919773 31.11611729 13.87644804 27.7066178  22.54449075 25.63246381]
 [16.5, 10.2, 30.1, 23.0, 14.4, 15.6, 19.4, 14.1, 30.3, 35.2, 23.1, 13.8, 25.0, 27.9, 19.5, 12.3, 32.2, 13.5, 23.
8, 21.7, 19.2, 19.5, 10.4, 23.2, 18.6, 28.5, 15.2, 32.0, 7.2, 34.6, 20.1, 20.6, 23.6, 13.1, 23.8, 12.7, 43.1, 24.
7, 22.2, 44.0, 28.1, 31.0, 21.7, 23.4, 19.5, 33.1, 41.7, 18.7, 19.9, 20.6, 21.2, 13.6, 20.3, 17.8, 27.1, 31.5, 50.
0, 29.1, 18.9, 20.4, 50.0, 7.2, 17.2, 36.2, 14.6, 33.2, 23.8, 19.9, 21.5, 37.3, 27.0, 22.0, 24.3, 19.8, 33.3, 7.0,
19.4, 20.9, 21.1, 20.4, 22.2, 11.9, 11.7, 21.6, 19.7, 23.0, 16.7, 21.7, 20.6, 23.3, 19.6, 28.0, 5.0, 24.4, 20.8, 2
4.8, 21.8, 23.6, 19.0, 25.0, 20.3, 21.5]
```

Final RMSE

```
final_rmse
```

```
4.141879367431788
```

Store model

```
In [51]: from joblib import dump, load
         dump(model, 'Model.joblib')
```

```
Out[51]: ['Model.joblib']
```

Testing of the developed module using a dataset

```
from joblib import dump, load
model = load('Model.joblib')
```

```
import numpy as np
features = np.array([[-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.74529952, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.21164221, -0.86091034]])
model.predict(features)
```

```
array([22.38781341])
```

Decision Tree

Selecting a model

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
# model = LinearRegression()
model = DecisionTreeRegressor()
# model = AdaBoostRegressor()
# model = RandomForestRegressor()
model.fit(housing_num_new, housing_labels)
```

▾ DecisionTreeRegressor

DecisionTreeRegressor()

Model evaluation

```
import numpy as np
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_new)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

```
mse
```

0.0

Using better evaluation technique: Cross validation

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_new, housing_labels, scoring='neg_mean_squared_error', cv=10)
rmse_scores = np.sqrt(-scores)
```

```
rmse_scores
```

```
array([3.96038308, 5.49278906, 5.28513793, 3.98322703, 4.17279882,
       3.06924258, 5.12528048, 3.83777279, 3.56419696, 4.19970237])
```

```
def print_scores(scores):
    print("Scores: ", scores)
    print("Mean: ", scores.mean())
    print("Standard deviation: ", scores.std())
```

```
print_scores(rmse_scores)
```

```
Scores:  [3.96038308 5.49278906 5.28513793 3.98322703 4.17279882 3.06924258
 5.12528048 3.83777279 3.56419696 4.19970237]
Mean:  4.269053110757295
Standard deviation:  0.7467597817771174
```

Testing model against test data

```
x_test = strat_test_data.drop('MEDV', axis=1)
y_test = strat_test_data['MEDV'].copy()
x_ready = myPipeline.transform(x_test)
final_predictions = model.predict(x_ready)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
print(final_predictions, '\n', list(y_test))
```

```
[29.8 11.9 29.  23.3 19.7 13.1 22.4 11.5 41.3 43.8 21.7 11.9 21.6 15.3
 19.  11.3 28.7 15.2 24.2 16.6 18.5 14.5 10.9 19.4 16.1 31.2 17.1 33.1
 10.5 32.7 20.  18.5 22.6 10.9 22.3  8.3 48.8 25.3 23.1 46.  24.3 22.8
 19.5 19.4 14.5 31.6 48.3 20.6 16.8 23.1 19.2 13.9 22.4 17.1 28.6 31.7
 37.6 30.8 18.7 19.9 50.   6.3 14.5 19.9 16.7 34.7 15.3 13.2 19.9 35.4
 24.3 22.6 19.1 24.3 32.7 13.8 15.6 18.6 22.  22.  19.9 22.  13.4 22.6
 26.4 21.2  8.7 18.9 19.9 20.7 24.5 22.9  5.6 29.1 16.8 29.9 17.4 28.7
  8.5 28.6 22.  18.8]
 [16.5, 10.2, 30.1, 23.0, 14.4, 15.6, 19.4, 14.1, 30.3, 35.2, 23.1, 13.8, 25.0, 27.9, 19.5, 12.3, 32.2, 13.5, 23.
8, 21.7, 19.2, 19.5, 10.4, 23.2, 18.6, 28.5, 15.2, 32.0, 7.2, 34.6, 20.1, 20.6, 23.6, 13.1, 23.8, 12.7, 43.1, 24.
7, 22.2, 44.0, 28.1, 31.0, 21.7, 23.4, 19.5, 33.1, 41.7, 18.7, 19.9, 20.6, 21.2, 13.6, 20.3, 17.8, 27.1, 31.5, 50.
0, 29.1, 18.9, 20.4, 50.0, 7.2, 17.2, 36.2, 14.6, 33.2, 23.8, 19.9, 21.5, 37.3, 27.0, 22.0, 24.3, 19.8, 33.3, 7.0,
19.4, 20.9, 21.1, 20.4, 22.2, 11.9, 11.7, 21.6, 19.7, 23.0, 16.7, 21.7, 20.6, 23.3, 19.6, 28.0, 5.0, 24.4, 20.8, 2
4.8, 21.8, 23.6, 19.0, 25.0, 20.3, 21.5]
```

Final RMSE

```
final_rmse
```

```
4.652882302991361
```

Store model

```
In [51]: from joblib import dump, load
         dump(model, 'Model.joblib')
```

```
Out[51]: ['Model.joblib']
```

Testing of the developed module using a dataset

```
from joblib import dump, load
model = load('Model.joblib')
```

```
import numpy as np
features = np.array([[-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.74529952, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.21164221, -0.86091034]])
model.predict(features)
```

```
array([21.9])
```

AdaBoost Regression

Selecting a model

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
# model = LinearRegression()
# model = DecisionTreeRegressor()
model = AdaBoostRegressor()
# model = RandomForestRegressor()
model.fit(housing_num_new, housing_labels)
```

```
▾ AdaBoostRegressor
AdaBoostRegressor()
```

Model evaluation

```
import numpy as np
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_new)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

```
mse
```

```
7.447474545445552
```

Using better evaluation technique: Cross validation

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_new, housing_labels, scoring='neg_mean_squared_error', cv=10)
rmse_scores = np.sqrt(-scores)
```

```
rmse_scores
```

```
array([3.52421674, 3.36952135, 4.43453651, 2.92571773, 3.28464873,
       3.41657665, 5.64815371, 3.70914394, 3.33170592, 3.34881972])
```

```
def print_scores(scores):
    print("Scores: ", scores)
    print("Mean: ", scores.mean())
    print("Standard deviation: ", scores.std())
```

```
print_scores(rmse_scores)
```

```
Scores:  [3.52421674 3.36952135 4.43453651 2.92571773 3.28464873 3.41657665
 5.64815371 3.70914394 3.33170592 3.34881972]
Mean:  3.6993040996762803
Standard deviation:  0.7474200987941813
```

Testing model against test data

```
x_test = strat_test_data.drop('MEDV', axis=1)
y_test = strat_test_data['MEDV'].copy()
x_ready = myPipeline.transform(x_test)
final_predictions = model.predict(x_ready)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
print(final_predictions, '\n', list(y_test))
```

```
print(final_predictions, '\n', list(y_test))
```

```
[25.00434783 12.39102564 26.93275862 22.86631579 19.21888889 17.1443609
 20.05       16.19090909 32.675      38.45862069 19.80478261 11.26271186
 23.73557047 19.03333333 22.86631579 11.47894737 28.33191489 17.71666667
 25.69313725 19.91616162 20.47195122 18.46213592 20.14705882 21.94666667
 20.32692308 29.57529412 16.65972222 30.32916667 10.9673913  34.99268293
 23.4031746  20.47195122 24.42039474 11.2        21.94666667 14.76125
 43.263      26.76296296 25.73731343 40.09473684 23.95153846 27.94210526
 19.80478261 24.38672566 18.46213592 32.75151515 43.52017544 23.95153846
 19.21888889 25.69313725 21.15964912 17.2326087  19.91616162 16.19090909
 26.93275862 34.22352941 42.79208633 29.47905405 19.27631579 23.95153846
 45.37391304 12.39102564 21.15964912 25.01149425 19.03333333 30.55591398
 21.15964912 20.05       20.32692308 34.71470588 27.43680982 25.01149425
 21.58333333 24.38672566 36.26545455 13.98181818 16.79038462 23.94125
 24.12820513 23.95153846 24.42039474 24.12820513 16.24615385 24.42039474
 22.1043956  21.58333333 15.15636364 22.45       24.12820513 25.01149425
 19.80478261 27.43846154 10.6        25.00434783 18.20107527 27.87777778
 20.32692308 30.55591398 17.36190476 27.43680982 21.42807882 20.21343284]
 [16.5, 10.2, 30.1, 23.0, 14.4, 15.6, 19.4, 14.1, 30.3, 35.2, 23.1, 13.8, 25.0, 27.9, 19.5, 12.3, 32.2, 13.5, 23.
8, 21.7, 19.2, 19.5, 10.4, 23.2, 18.6, 28.5, 15.2, 32.0, 7.2, 34.6, 20.1, 20.6, 23.6, 13.1, 23.8, 12.7, 43.1, 24.
7, 22.2, 44.0, 28.1, 31.0, 21.7, 23.4, 19.5, 33.1, 41.7, 18.7, 19.9, 20.6, 21.2, 13.6, 20.3, 17.8, 27.1, 31.5, 50.
0, 29.1, 18.9, 20.4, 50.0, 7.2, 17.2, 36.2, 14.6, 33.2, 23.8, 19.9, 21.5, 37.3, 27.0, 22.0, 24.3, 19.8, 33.3, 7.0,
19.4, 20.9, 21.1, 20.4, 22.2, 11.9, 11.7, 21.6, 19.7, 23.0, 16.7, 21.7, 20.6, 23.3, 19.6, 28.0, 5.0, 24.4, 20.8, 2
4.8, 21.8, 23.6, 19.0, 25.0, 20.3, 21.5]
```

Final RMSE

```
final_rmse
```

```
3.59998411791391
```

Store model

```
In [51]: from joblib import dump, load
         dump(model, 'Model.joblib')
```

```
Out[51]: ['Model.joblib']
```

Testing of the developed module using a dataset

```
from joblib import dump, load
model = load('Model.joblib')
```

```
import numpy as np
features = np.array([[-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.74529952, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.21164221, -0.86091034]])
model.predict(features)
```

```
array([26.00825688])
```

Random Forest

Selecting a model

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
# model = LinearRegression()
# model = DecisionTreeRegressor()
# model = AdaBoostRegressor()
model = RandomForestRegressor()
model.fit(housing_num_new, housing_labels)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

Model evaluation

In [45]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_new)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

In [46]:
```
mse
```

Out[46]:
```
1.3995934900990077
```

Using better evaluation technique: Cross validation

In [47]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_new, housing_labels, scoring='neg_mean_squared_error', cv=10)
rmse_scores = np.sqrt(-scores)
```

In [48]:
```
rmse_scores
```

Out[48]:
```
array([2.91524447, 2.95600355, 4.46141842, 2.60459303, 3.46936637,
       2.4829617 , 4.57062754, 3.39839661, 3.38022811, 3.19867748])
```

In [49]:
```
def print_scores(scores):
    print("Scores: ", scores)
    print("Mean: ", scores.mean())
    print("Standard deviation: ", scores.std())
```

In [50]:
```
print_scores(rmse_scores)
```

```
Scores:  [2.91524447 2.95600355 4.46141842 2.60459303 3.46936637 2.4829617
 4.57062754 3.39839661 3.38022811 3.19867748]
Mean:  3.343751727671568
Standard deviation:  0.6645921625590131
```

Testing model against test data

In [52]:
```
x_test = strat_test_data.drop('MEDV', axis=1)
y_test = strat_test_data['MEDV'].copy()
x_ready = myPipeline.transform(x_test)
final_predictions = model.predict(x_ready)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
In [53]: print(final_predictions, '\n', list(y_test))
```

```
[24.394 11.386 25.474 21.759 19.706 15.062 20.123 14.428 31.84  41.762
 20.023 12.114 22.971 30.205 19.58  11.23  31.389 14.218 23.703 19.214
 19.996 18.136 18.792 22.199 18.634 30.977 15.769 33.171  9.146 33.043
 24.362 21.33  22.439 10.855 21.257 11.236 44.142 24.326 23.186 41.888
 23.95  29.841 20.398 21.109 19.39  33.241 43.848 20.084 20.155 22.391
 21.548 14.661 21.17  15.01  24.616 33.576 41.83  29.033 19.822 20.409
 46.772  9.776 18.828 25.51  14.555 33.42  20.283 17.941 19.332 33.964
 25.155 22.794 21.735 22.227 33.837 12.972 15.999 20.313 20.736 21.512
 21.992 20.672 14.099 22.393 20.577 21.006 14.192 21.687 20.612 23.295
 18.44  27.109  7.306 26.08  19.469 29.493 20.287 31.104 14.485 26.63
 21.339 20.344]
 [16.5, 10.2, 30.1, 23.0, 14.4, 15.6, 19.4, 14.1, 30.3, 35.2, 23.1, 13.8, 25.0, 27.9, 19.5, 12.3, 32.2, 13.5, 23.8, 21.7, 19.2,
19.5, 10.4, 23.2, 18.6, 28.5, 15.2, 32.0, 7.2, 34.6, 20.1, 20.6, 23.6, 13.1, 23.8, 12.7, 43.1, 24.7, 22.2, 44.0, 28.1, 31.0, 2
1.7, 23.4, 19.5, 33.1, 41.7, 18.7, 19.9, 20.6, 21.2, 13.6, 20.3, 17.8, 27.1, 31.5, 50.0, 29.1, 18.9, 20.4, 50.0, 7.2, 17.2, 36.
2, 14.6, 33.2, 23.8, 19.9, 21.5, 37.3, 27.0, 22.0, 24.3, 19.8, 33.3, 7.0, 19.4, 20.9, 21.1, 20.4, 22.2, 11.9, 11.7, 21.6, 19.7,
23.0, 16.7, 21.7, 20.6, 23.3, 19.6, 28.0, 5.0, 24.4, 20.8, 24.8, 21.8, 23.6, 19.0, 25.0, 20.3, 21.5]
```

Final RMSE

```
final_rmse
```

```
2.949417740809856
```

Store model

```
In [51]: from joblib import dump, load
         dump(model, 'Model.joblib')
```

```
Out[51]: ['Model.joblib']
```

Testing of the developed module using a dataset

```
In [4]: from joblib import dump, load
        model = load('Model.joblib')
```

```
In [10]: import numpy as np
         features = np.array([[-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
                -0.74529952, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
                -0.97491834,  0.21164221, -0.86091034]])
         model.predict(features)
```

```
Out[10]: array([21.396])
```

Future Scope of Improvement

- The RMSE value could be lowered further.
- Better model may be used.
- Prediction can be made more accurate by increasing the volume of training data.
- Better correlating features might be introduced in future.