MODELLING AND SIMULATION
(IT – 205)

PROJECT REPORT

SUBMITTED BY : -

VARUN KUMAR (2K19 / IT / 140)

YASHIT KUMAR (2K19 / IT / 149)

Under the Supervision of : -

Mr . Ankit Yadav

Asst.Prof.

DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project titled "**House Price Prediction** " which is submitted by Varun Kumar ; Roll No – 2K19/IT/140 ; and Yashit Kumar ; Roll No – 2K19/IT/149 INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in fulfillment of the requirement for the 3rd semester of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place : Delhi **Mr. Ankit Yadav**

Date : 15-11-20 **Supervisor**

DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

We would like to convey our heartfelt thanks to our supervisor Mr. Ankit Yadav for his ingenious ideas, tremendous help and cooperation. We are extremely grateful to our friends who gave valuable suggestions and guidance for completion of our project. The cooperation and healthy criticism came handy and useful with them.

OBJECTIVE

Thousands of houses are sold everyday. There are some questions every buyer asks himself like: What is the actual price that this house deserves? Am I paying a fair price?

In this Project a machine learning model is proposed to predict a house price based on data sets (features) related to the house .

Our model will be used to predict house prices in given area and invest in that area.

While learning about machine learning it is best to actually work with real world data .

The main objectives of this Project are as follows:

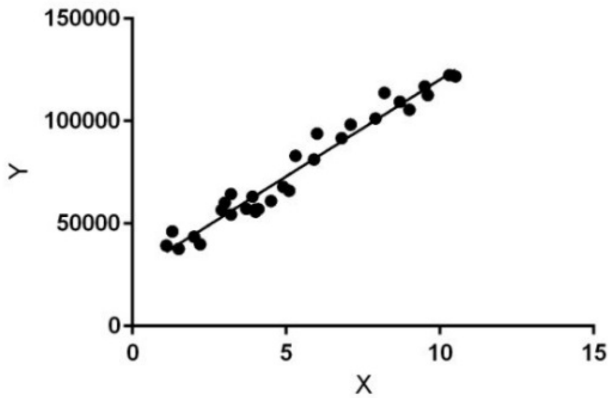
- To apply data preprocessing and preparation techniques in order to obtain clean data
- To build machine learning models able to predict house price based on house features
- To analyze and compare models performance in order to choose the best model

ALGORITHMS USED

We are trying to predict the House price using the machine learning techniques with the help of the previous works. We have used the Simple Linear Regression , Decision Tree Regression and Random Forest Regression So, it would be helpful for the people and it may avoid them in making mistakes.

1. Linear Regression

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

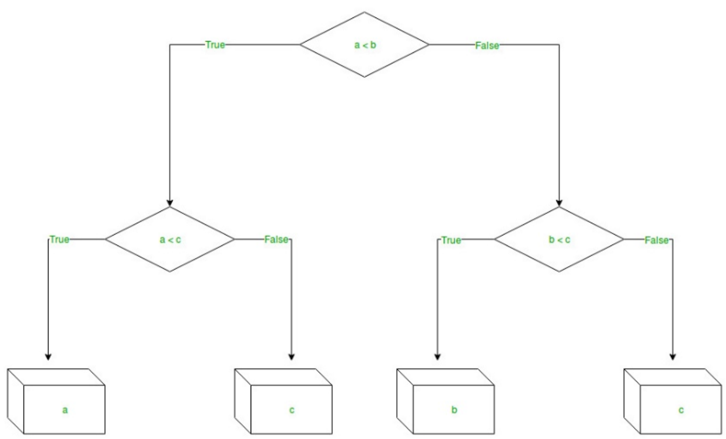
2. Decision Tree Regression

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs and utility.
Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

1. Conditions [Decision Nodes]
2. Result [End Nodes]

The branches/edges represent the truth/falsity of the statement and takes makes a decision based on that in the example below which shows a decision tree that evaluates the smallest of three numbers:

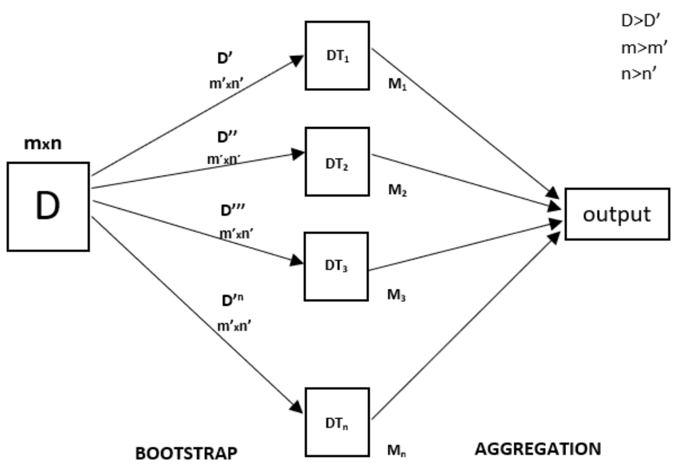


Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Discrete output example: A weather prediction model that predicts whether or not there'll be rain in a particular day.
Continuous output example: A profit prediction model that states the probable profit that can be generated from the sale of a product

3. Random Forest Regression

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual 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.



DATASETS

1. Title: Boston Housing Data

2. Sources:

(a) Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

(b) Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

(c) Date: July 7, 1993

3. Past Usage:

- Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

4. Relevant Information:

Concerns housing values in suburbs of Boston.

5. Number of Instances: 506

6. Number of Attributes: 13 continuous attributes (including "class" attribute "MEDV"), 1 binary-valued attribute.

7. Attribute Information:

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

8. Missing Attribute Values: None.

DATASET ANALYSIS

```
housing.head()
```

	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

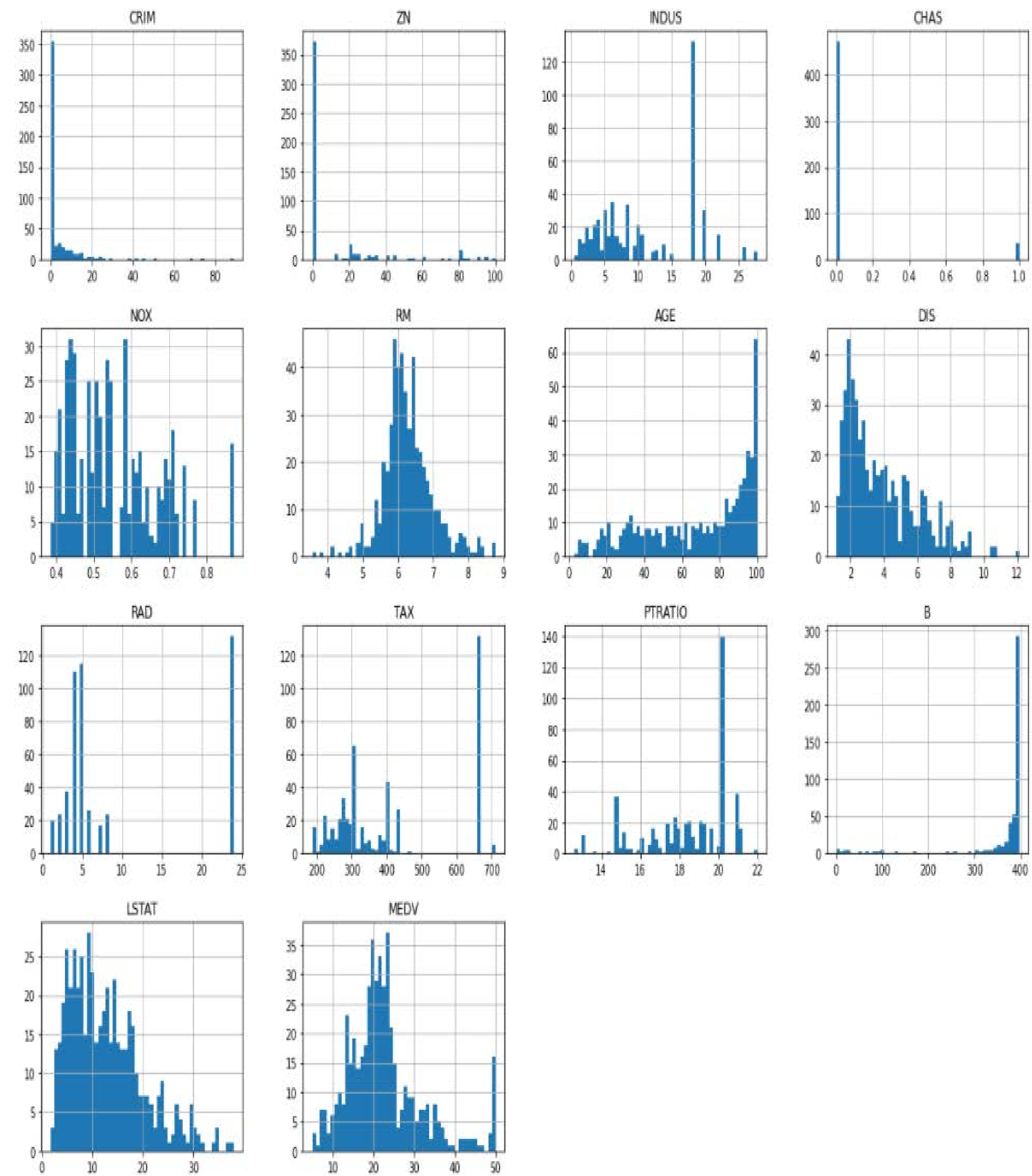
```
housing.info()
```

```
<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
dtypes: float64(11), int64(3)
memory usage: 55.4 KB
```

```python
housing.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
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	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284341	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.705587	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
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	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.884000	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208000	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.625000	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
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	37.970000	50.000000

FEATURES GRAPH ANALYSIS



FEATURES OBSERVATION

Data Science is the process of making some assumptions and hypothesis on the data, and testing them by performing some tasks. Initially we could make the following intuitive assumptions for each feature:

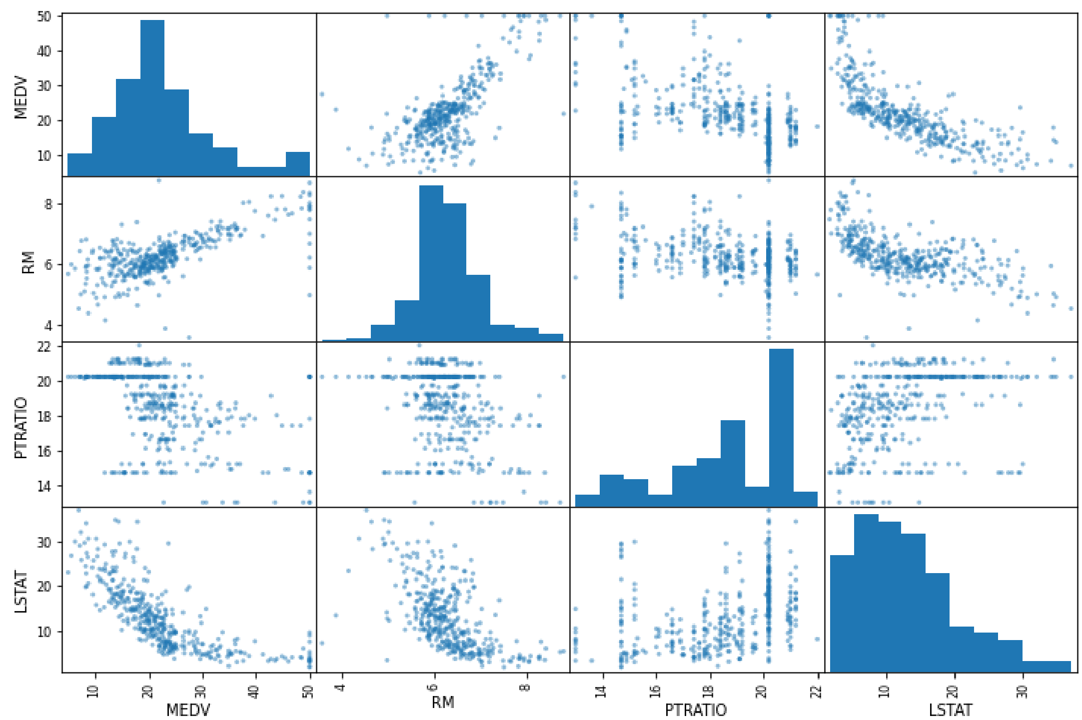
- Houses with more rooms (higher ‘RM’ value) will worth more. Usually houses with more rooms are bigger and can fit more people, so it is reasonable that they cost more money. They are directly proportional variables.
- Neighborhoods with more lower class workers (higher ‘LSTAT’ value) will worth less. If the percentage of lower working class people is higher, it is likely that they have low purchasing power and therefore, they houses will cost less. They are inversely proportional variables.
- Neighborhoods with more students to teachers ratio (higher ‘PTRATIO’ value) will be worth less. If the percentage of students to teachers ratio people is higher, it is likely that in the neighborhood there are less schools, this could be because there is less tax income which could be because in that neighborhood people earn less money. If people earn less money it is likely that their houses are worth less. They are inversely proportional variables.

We’ll find out if these assumptions are correct through the project.

LOOKING FOR CORRELATIONS

We will start by creating a scatterplot matrix that will allow us to visualize the pair-wise relationships and correlations between the different features.

It is also quite useful to have a quick overview of how the data is distributed and wheter it cointains or not outliers.



We can spot a linear relationship between ‘RM’ and House prices ‘MEDV’. In addition, we can infer from the histogram that the ‘MEDV’ variable seems to be normally distributed but contain several outliers

SHUFFLE AND SPLIT DATA

For this section we will take the Boston housing dataset and split the data into training and testing subsets. Typically, the data is also shuffled into a random order when creating the training and testing subsets to remove any bias in the ordering of the dataset.

. TRAINING AND TESTING

It is useful to evaluate our model once it is trained. We want to know if it has learned properly from a training split of the data. There can be 3 different situations:

1) The model didn´t learn well on the data, and can’t predict even the outcomes of the training set, this is called underfitting and it is caused because a high bias.

2) The model learn too well the training data, up to the point that it memorized it and is not able to generalize on new data, this is called overfitting, it is caused because high variance.

3) The model just had the right balance between bias and variance, it learned well and is able predict correctly the outcomes on new data.

SOURCE CODE

Github link to code - https://github.com/varunkmr038/MS-PROJECT-

HOUSE PRICE PREDICTOR

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

In [2]:
```
housing = pd.read_csv("data.csv")
```

In [3]:
```
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

In [4]:
```
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
dtypes: float64(11), int64(3)
memory usage: 55.4 KB
```

In [5]:
```
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	6.284341	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	0.705587	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	5.884000	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	6.208000	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.625000	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

In [7]:
```
%matplotlib inline
```

In [8]:
```
# For plotting histogram
# import matplotlib.pyplot as plt
# housing.hist(bins=50, figsize=(20, 15))
```

Train-Test Splitting

In [9]:
```
import numpy as np
def split_train_test(data, test_ratio):
    np.random.seed(42)
    shuffled = np.random.permutation(len(data))
    print(shuffled)
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled[:test_set_size]
    train_indices = shuffled[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [10]:
```
# train_set, test_set = split_train_test(housing, 0.2)
```

In [11]:
```
# print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_set)}\n")
```

In [12]:
```
from sklearn.model_selection import train_test_split
train_set, test_set  = train_test_split(housing, test_size=0.2, random_state=42)
print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_set)}\n")
```

```
Rows in train set: 404
Rows in test set: 102
```

In [13]:
```
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_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

In [14]:
```
strat_test_set['CHAS'].value_counts()
```

Out[14]:
```
0    95
1     7
Name: CHAS, dtype: int64
```

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

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

In [16]:
```
# 95/7
```

In [17]:
```
# 376/28
```

In [18]:
```
housing = strat_train_set.copy()
```

Looking for Correlations

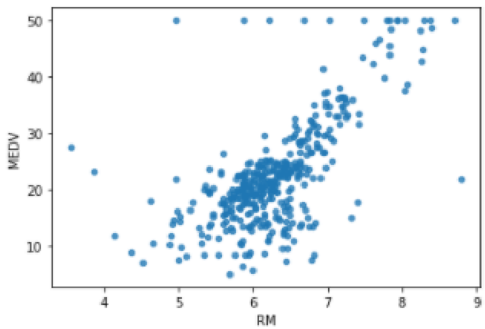
In [19]:
```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[19]:
```
MEDV       1.000000
RM         0.680857
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
Name: MEDV, dtype: float64
```

In [20]:
```
# from pandas.plotting import scatter_matrix
# attributes = ["MEDV", "RM", "PTRATIO", "LSTAT"]
# scatter_matrix(housing[attributes], figsize = (12,8))
```

In [21]:
```python
housing.plot(kind="scatter", x="RM", y="MEDV", alpha=0.8)
```

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



Trying out Attribute combinations

In [22]:
```python
housing["TAXRM"] = housing['TAX']/housing['RM']
```

In [23]:
```python
housing.head()
```

Out[23]:

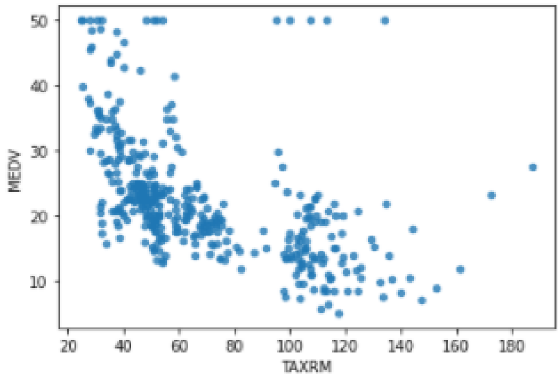
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	TAXRM
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57	21.9	51.571709
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99	24.5	42.200452
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	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

In [24]:
```python
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[24]:
```
MEDV       1.000000
RM         0.680857
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     -0.528626
```

```
In [25]: housing.plot(kind="scatter", x="TAXRM", y="MEDV", alpha=0.8)
```

```
Out[25]: <AxesSubplot:xlabel='TAXRM', ylabel='MEDV'>
```




```
In [26]: housing = strat_train_set.drop("MEDV", axis=1)
         housing_labels = strat_train_set["MEDV"].copy()
```

Missing Attributes

```
In [27]: # To take care of missing attributes, we have three options:
         #     1. Get rid of the missing data points
         #     2. Get rid of the whole attribute
         #     3. Set the value to some value(0, mean or median)
```

```
In [28]: a = housing.dropna(subset=["RM"]) #Option 1
         a.shape
```

```
Out[28]: (399, 13)
```

```
In [29]: housing.drop("RM", axis=1).shape # Option 2
```

```
Out[29]: (404, 12)
```

```
In [30]: median = housing["RM"].median() # Compute median for Option 3
```

```
In [31]: housing["RM"].fillna(median) # Option 3
```

```
Out[31]: 254    6.108
         348    6.635
         476    6.484
         321    6.376
```

In [32]:
```
housing.shape
```
Out[32]:
```
(404, 13)
```

In [33]:
```
housing.describe() # before we started filling missing attributes
```
Out[33]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	I
count	404.000000	404.000000	404.000000	404.000000	404.000000	399.000000	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	6.279481	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	0.716784	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%	0.086963	0.000000	5.190000	0.000000	0.453000	5.876500	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	6.209000	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	6.630500	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 [34]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
imputer.fit(housing)
```
Out[34]:
```
SimpleImputer(strategy='median')
```

In [35]:
```
imputer.statistics_
```
Out[35]:
```
array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
       6.20900e+00, 7.82000e+01, 3.12220e+00, 5.00000e+00, 3.37000e+02,
       1.90000e+01, 3.90955e+02, 1.15700e+01])
```

In [36]:
```
X = imputer.transform(housing)
```

In [37]:
```
housing_tr = pd.DataFrame(X, columns=housing.columns)
```

In [38]:
```
housing_tr.describe()
```
Out[38]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	I
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	6.278609	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	0.712366	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%	0.086963	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	6.209000	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	6.630000	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

Creating a Pipeline

In [39]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])
```

In [40]:
```
housing_num_tr = my_pipeline.fit_transform(housing)
```

In [41]:
```
housing_num_tr.shape
```

Out[41]:
```
(404, 13)
```

Selecting a desired model

In [42]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
# model = LinearRegression()
# model = DecisionTreeRegressor()
model = RandomForestRegressor()
model.fit(housing_num_tr, housing_labels)
```

Out[42]:
```
RandomForestRegressor()
```

In [43]:
```
some_data = housing.iloc[:5]
```

In [44]:
```
some_labels = housing_labels.iloc[:5]
```

In [45]:
```
prepared_data = my_pipeline.transform(some_data)
```

In [46]:
```
model.predict(prepared_data)
```

Out[46]:
```
array([22.273, 25.598, 16.531, 23.315, 23.511])
```

In [47]:
```
list(some_labels)
```

Out[47]:
```
[21.9, 24.5, 16.7, 23.1, 23.0]
```

Evaluating the model

In [48]:
```
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_tr)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

In [49]:
```
rmse
```

Out[49]:
```
1.2735771822347122
```

Using better evaluation technique - Cross Validation

In [50]:
```
# 1 2 3 4 5 6 7 8 9 10
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_tr, housing_labels, scoring="neg_mean_squared_error", cv=10)
rmse_scores = np.sqrt(-scores)
```

In [51]:
```
rmse_scores
```

Out[51]:
```
array([2.75685401, 2.72456915, 4.4506526 , 2.63206438, 3.42436845,
       2.72362479, 4.89403592, 3.44025965, 3.35547506, 3.11180581])
```

In [52]:
```
def print_scores(scores):
    print("Scores:", scores)
    print("Mean: ", scores.mean())
    print("Standard deviation: ", scores.std())
```

In [53]:
```
print_scores(rmse_scores)
```

```
Scores: [2.75685401 2.72456915 4.4506526  2.63206438 3.42436845 2.72362479
 4.89403592 3.44025965 3.35547506 3.11180581]
Mean:  3.3513709824947817
Standard deviation:  0.7291550975961761
```

Saving the model

In [54]:
```
from joblib import dump, load
dump(model, 'Predictor.joblib')
```

Out[54]:
```
['Predictor.joblib']
```

Testing the model on test data

In [55]:
```
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
# print(final_predictions, list(Y_test))
```

In [56]:
```
final_rmse
```

Out[56]:
```
2.9620950654005993
```

Using the model

In [57]:
```
from joblib import dump, load
import numpy as np
model = load('Predictor.joblib')
features = np.array([[-4.43942006, 4.12628155, -1.6165014, -0.67288841, -1.42262747,
       -11.44443979304, -49.31238772,  7.61111401, -26.0016879 , -0.5778192 ,
       -0.97491834,  0.41164221, -66.86091034]])
model.predict(features)
```

Out[57]:
```
array([24.3])
```

CONCLUSION

The main goal of this project is to determine the prediction for prices of houses for which we tried three different machine learning algorithms namely Linear regression, Decision tree regression and Random forest regression , And we got the following results : ---

Model Outputs

1. Linear Regression:
   Mean (rmse error): 5.037482786117751

   Standard deviation: 1.0594382405606948

2. Decision Tree Regression:
   Mean (rmse error) : 4.220181728238616

   Standard deviation: 0.7451258431327811

3. Random Forest Regression
   Mean (rmse error) : 3.3513709824947817

   Standard deviation: 0.7291550975961761

So it's clear from the outputs that the Random forest have more accuracy in prediction when compared to the others . Hence , we have used this model to predict better house prices .

BIBLIOGRAPHY

- ➔ https://archive.ics.uci.edu/ml/machine-learning-databases/housing/?C=N;O=D
- ➔ https://www.kaggle.com/shreayan98c/boston-house-price-prediction
- ➔ https://www.geeksforgeeks.org/random-forest-regression-in-python/
- ➔ https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/
- ➔ https://www.knowledgehut.com/blog/data-science/linear-regression-for-machine-learning