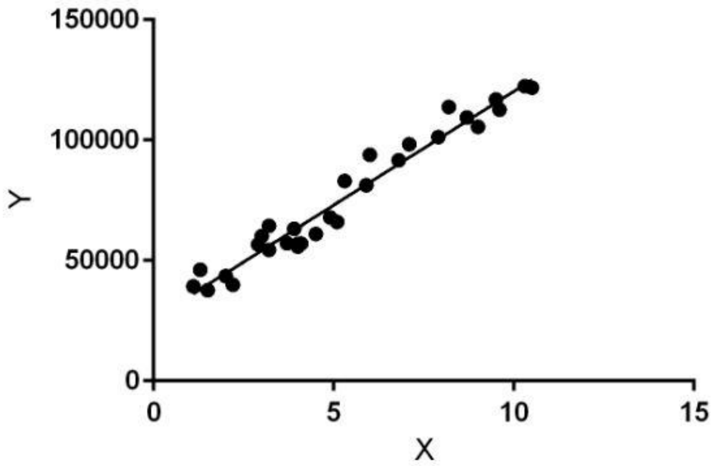
Experiment No. 1
Analyze the Boston Housing dataset and apply appropriate Regression Technique
Date of Performance:
Date of Submission:

Aim: Analyze the Boston Housing dataset and apply appropriate Regression Technique.

Objective: Ablility to perform various feature engineering tasks, apply linear regression on the given dataset and minimise the error.

Theory:

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 getting 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.

Dataset:

The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset columns:

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

Code:

```
[1] import numpy as np
    import pandas as pd
    from sklearn.model_selection import train_test_split
```

```
[2] data = pd.read_csv('/HousingData.csv')
    prices = data['MEDV']
    features = data.drop('MEDV', axis=1)
```

```
[3] print("Boston housing dataset has {} data points with {} variables each.".format(*data.shape))
    print(data[:10])
    print(features)
    print(prices)
```

```
Boston housing dataset has 506 data points with 14 variables each.
      CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31   0.0  0.538  6.575   65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07   0.0  0.469  6.421   78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07   0.0  0.469  7.185   61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18   0.0  0.458  6.998   45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18   0.0  0.458  7.147   54.2  6.0622    3  222     18.7
5  0.02985   0.0   2.18   0.0  0.458  6.430   58.7  6.0622    3  222     18.7
6  0.08829  12.5   7.87   NaN  0.524  6.012   66.6  5.5605    5  311     15.2
7  0.14455  12.5   7.87   0.0  0.524  6.172   96.1  5.9505    5  311     15.2
8  0.21124  12.5   7.87   0.0  0.524  5.631  100.0  6.0821    5  311     15.2
9  0.17004  12.5   7.87   NaN  0.524  6.004   85.9  6.5921    5  311     15.2
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
```

```
df = pd.read_csv('./boston.csv')
```

```
df.keys() #return all the keys of the dictionary
```

```
Index(['Unnamed: 0', 'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
       'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'Price'],
      dtype='object')
```

```
df.describe()
```

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	252.500000	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3
std	146.213884	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2
min	0.000000	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1
25%	126.250000	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2
50%	252.500000	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3
75%	378.750000	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5
max	505.000000	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  506 non-null    int64  
 1   CRIM        506 non-null    float64
 2   ZN          506 non-null    float64
 3   INDUS       506 non-null    float64
 4   CHAS        506 non-null    float64
 5   NOX         506 non-null    float64
 6   RM          506 non-null    float64
 7   AGE         506 non-null    float64
 8   DIS         506 non-null    float64
 9   RAD         506 non-null    float64
 10  TAX         506 non-null    float64
 11  PTRATIO     506 non-null    float64
 12  B           506 non-null    float64
 13  LSTAT       506 non-null    float64
 14  Price       506 non-null    float64
dtypes: float64(14), int64(1)
memory usage: 59.4 KB
```

```
df.head(5)
```

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

Check if the dataset contains any null value or not

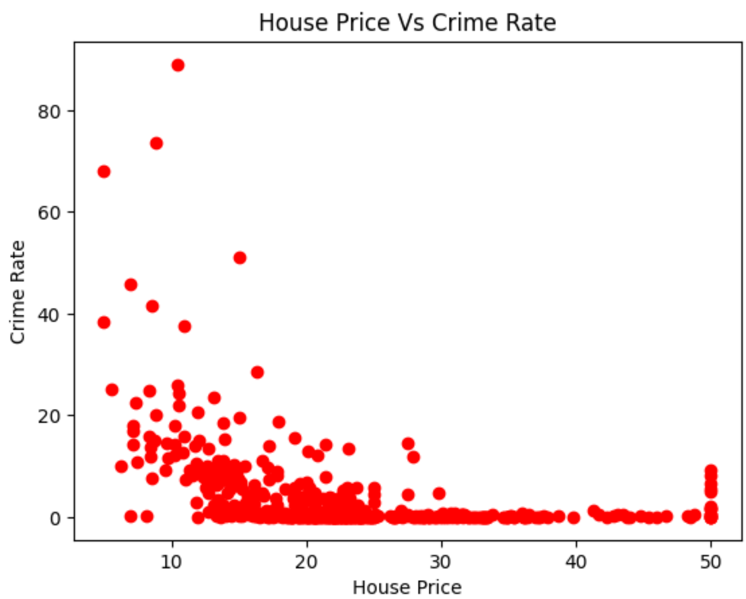
```
df.isnull()
```

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Pr
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
501	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
502	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
503	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
504	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
505	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F

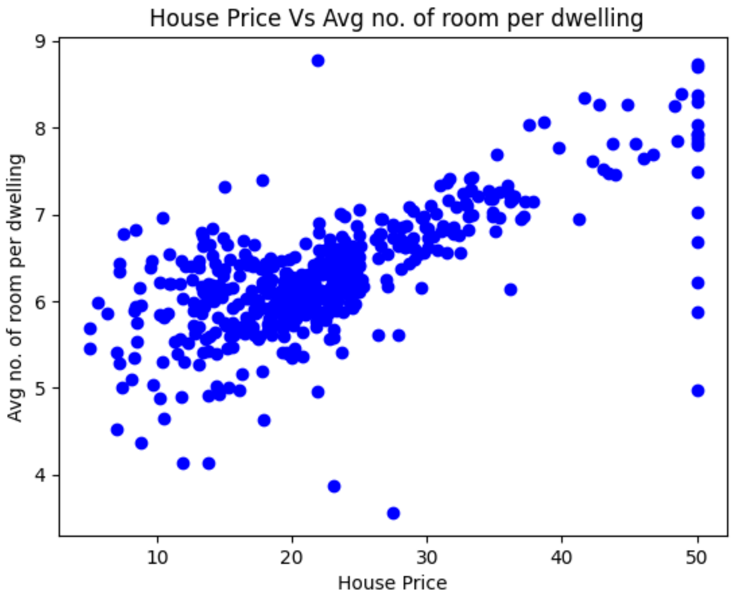
```
df.isnull().sum()
```

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

```
import matplotlib.pyplot as plt

plt.scatter(df['Price'],df['CRIM'], color='red')
plt.title(" House Price Vs Crime Rate ")
plt.xlabel("House Price")
plt.ylabel("Crime Rate")
plt.show()
```

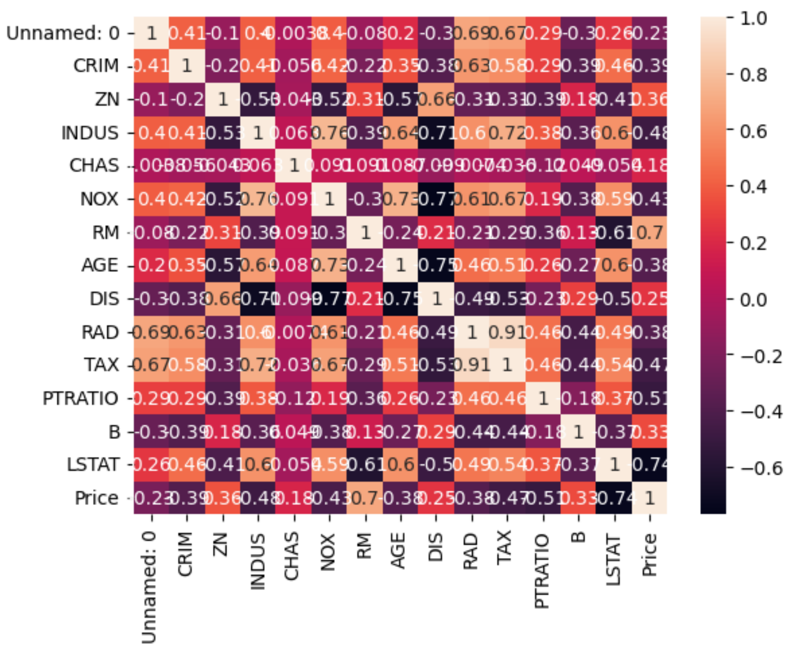



```
plt.scatter(df['Price'],df['RM'], color='blue')
plt.title(" House Price Vs Avg no. of room per dwelling")
plt.xlabel("House Price")
plt.ylabel("Avg no. of room per dwelling")
plt.show()
```




```
sns.heatmap(df.corr(), annot=True)
```

```
<Axes: >
```



We never train the model on all the data that we have, we split the data into two; one is training data and other is testing data to compare the result after traing the model with the testing data.

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop(['Price'], axis=1)
```

```
Y = df['Price']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.15, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(430, 14)
(76, 14)
(430,)
(76,)
```

Importing the linear regression model and train it on the training dataset

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Fitting the model on the training data

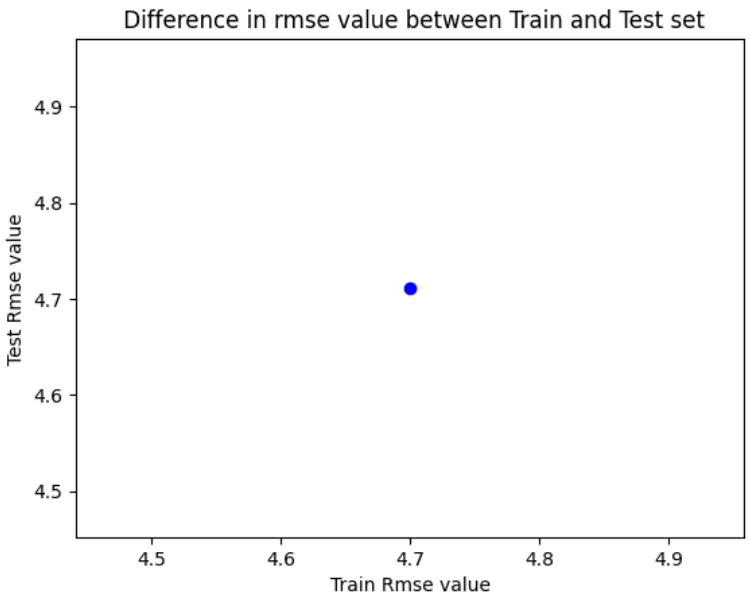
```
lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

```
▸ LinearRegression
```

```
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))

print("The model performance for the training set")
print('RMSE is {}'.format(rmse))
print("\n")

#on testing set

y_test_predict = lin_model.predict(X_test)
rmsee = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

print("The model performance for the testing set")
print('RMSE is {}'.format(rmsee))
```

```
The model performance for the training set
RMSE is 4.700268480051523


The model performance for the testing set
RMSE is 4.711340264707373
```

```
plt.scatter(rmse, rmsee, color='blue')
plt.title(" Difference in rmse value between Train and Test set")
plt.xlabel("Train Rmse value")
plt.ylabel("Test Rmse value")
plt.show()
```



Conclusion:

The selected features for model development include 'LSTAT,' 'RM,' and 'PTRATIO,' known for their strong correlation with the target 'MEDV.' These variables are intuitive predictors of housing demand and desirability. Additionally, features like 'INDUS,' 'TAX,' 'NOX,' 'RAD,' 'AGE,' and 'CRIM' provide insights into socio-economic and environmental factors impacting housing values. Integrating these features improves the model's capacity to capture nuances, potentially leading to more precise 'MEDV' predictions.

The Mean Squared Error (MSE) evaluates the accuracy of a predictive model by measuring the average of the squared differences between predicted and actual values. It gives more weight to significant deviations, and a lower MSE reflects superior performance in minimizing prediction errors