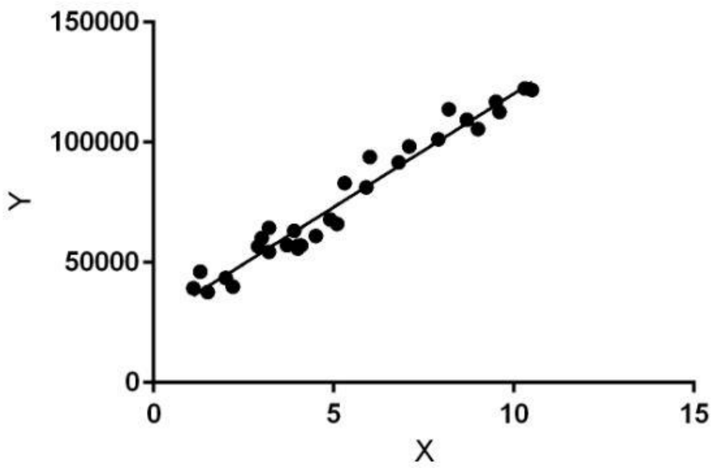
Experiment No. 1
Analyze the Boston Housing dataset and apply appropriate Regression Technique
Date of Performance:
Date of Submission:

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:

Conclusion:

1. What are features have been chosen to develop the model? Justify the features chosen to estimate the price of a house.
    - Column LSTAT is selected as X coordinate and Column MEDV is selected as Y coordinate. LSTAT here stands for (% lower status of the population) and MEDV here stands for (median value of owner-occupied homes in $1000's.
    - The heat map which is produced depicts a strong correlation between LSTAT and MEDV. The scatterplot depicts that the prices tend to decrease as there is an increase in LSTAT.
    - Therfore, the data is splitted into training and test sets which comprise of 80% of sample for training and 20% for testing the trained model. For developing the model Columns LSTAT & MEDV are used to predict the values using Linear Regression.
2. Comment on the Mean Squared Error calculated.
    - The Mean Squared Error (MSE) evaluates linear regression's performance by measuring the squared difference between actual and predicted house prices. Calculated MSE values for training and testing datasets reveal model accuracy.

- A lower MSE would indicate better predictive accuracy, as it signifies that the predictions are closer to the actual values. However, the interpretation of whether a obtained value is good or not depends on the specific context of the problem you're working on and the scale of the data.
- MAE: 5.078127727696937
  MSE: 46.994820919547124

```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('housing.csv')
print(df)
```

```
      0.00632  18.00   2.310  0  0.5380  6.5750  65.20  4.0900   1  296.0  15.30 396.90   4.98  24.00
0      0.02731   0.00   7.070  0  0.4690  6.4210  78...
1      0.02729   0.00   7.070  0  0.4690  7.1850  61...
2      0.03237   0.00   2.180  0  0.4580  6.9980  45...
3      0.06905   0.00   2.180  0  0.4580  7.1470  54...
4      0.02985   0.00   2.180  0  0.4580  6.4300  58...
..                                                ...
500    0.06263   0.00  11.930  0  0.5730  6.5930  69...
501    0.04527   0.00  11.930  0  0.5730  6.1200  76...
502    0.06076   0.00  11.930  0  0.5730  6.9760  91...
503    0.10959   0.00  11.930  0  0.5730  6.7940  89...
504    0.04741   0.00  11.930  0  0.5730  6.0300  80...

[505 rows x 1 columns]
```

```
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'L
df = pd.read_csv('housing.csv',header=None,  delimiter=r"\s+", names=column_names)
df.head()
```

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

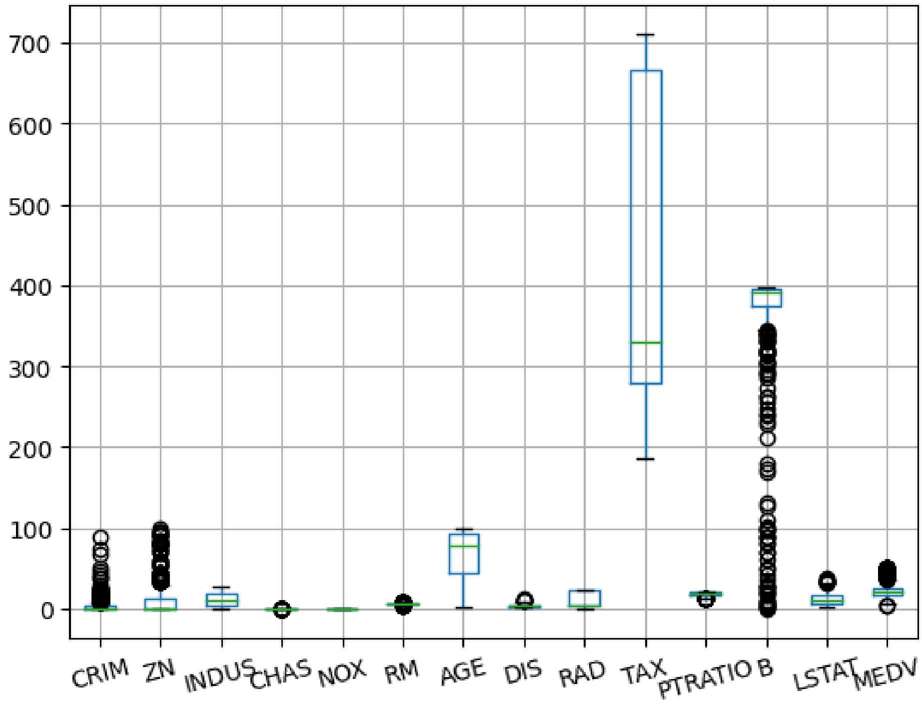
```
print(df)
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0
..       ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273.0
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273.0
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273.0
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273.0
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273.0

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       ...     ...    ...   ...
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
```

```
504      21.0  393.45   6.48  22.0
505      21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

```
df.boxplot(column_names, rot=15)
```

```
<Axes: >
```

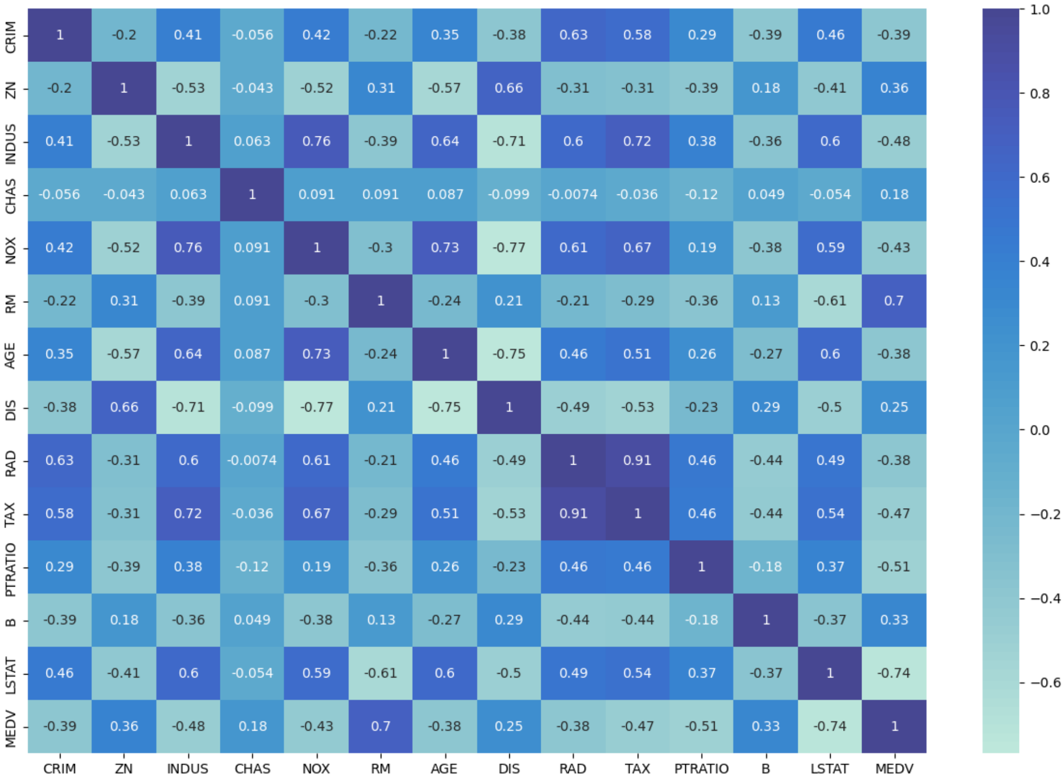



```
df.notnull()
```

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

506 rows × 14 columns

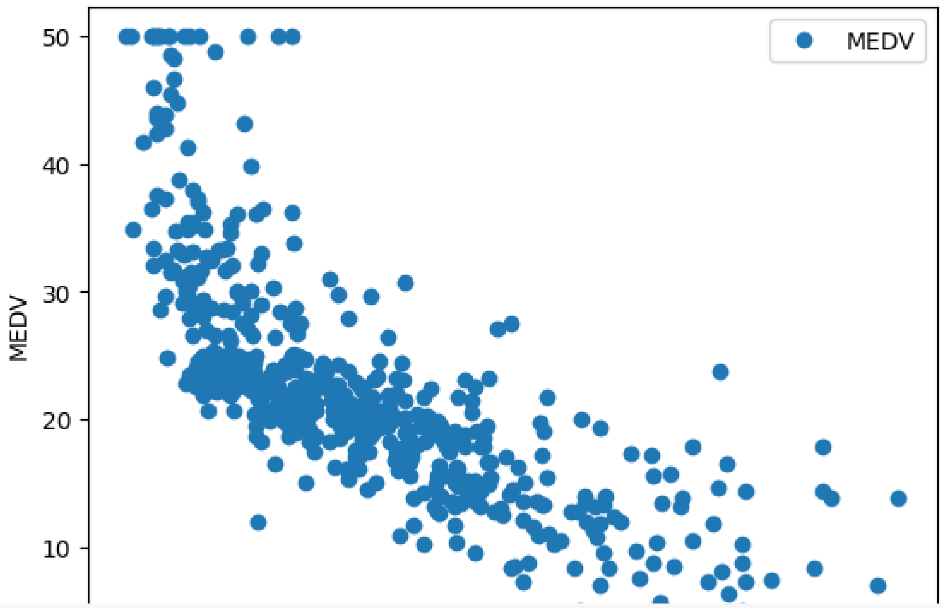
```
plt.figure(figsize=(15,10))
sns.heatmap(df.select_dtypes(include=['int','float']).corr(),annot=True,center = 2)
plt.show()
```




```
df= df.loc[:,['LSTAT','MEDV']]
df.head(10)
```

	LSTAT	MEDV
0	4.98	24.0
1	9.14	21.6
2	4.03	34.7
3	2.94	33.4
4	5.33	36.2
5	5.21	28.7
6	12.43	22.9
7	19.15	27.1
8	29.93	16.5
9	17.10	18.9

```
df.plot(x='LSTAT',y='MEDV',style='o')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
plt.show()
```




```
X=pd.DataFrame(df['LSTAT'])
y=pd.DataFrame(df['MEDV'])
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size=0.2, random_state=1)
```

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

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

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = lin_model.predict(X_train)
```

```
from sklearn import metrics
print('R^2:',metrics.r2_score(y_train, y_pred))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.5495280791456811
MAE: 4.38437515332486
MSE: 36.389745631141025
RMSE: 6.032391369195223
```

```
y_test_pred = lin_model.predict(X_test)
```

```
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.5244757432765152
MAE: 5.078127727696937
MSE: 46.994820919547124
RMSE: 6.855276866731724
```