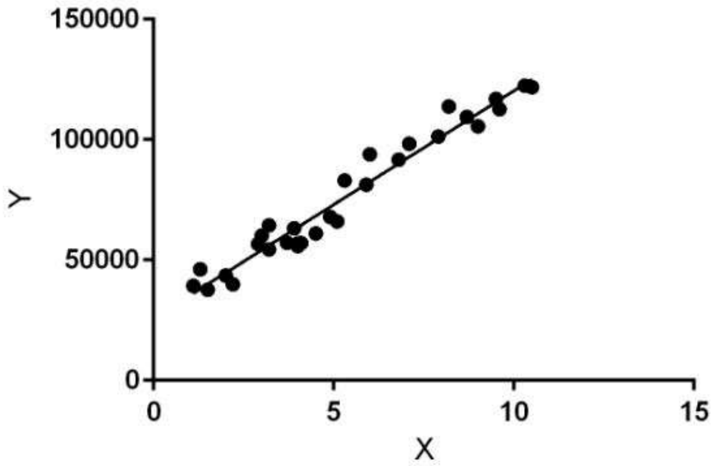
Experiment No. 1
Analyze the Boston Housing dataset and apply appropriate Regression Technique
Date of Performance:
Date of Submission:

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.

   The MEDV column is the target variable as it can predict the median values of homes. To fit the linear regression model we select features which have high corelation with our target column MEDV.

   The features chosen to estimate house prices include the attributes RM, TAX, RAD, CRIM, ZN.

   These columns contain the attributes of town such as average no. of room per dwelling, tax rate, accessibility to highways , per capita crime rate and proportion of residential land.

2. Comment on the Mean Squared Error calculated.

   Mean Squared Error (MSE) is a commonly used in linear regression that quantifies the average squared difference between the predicted values and the actual values. It provides a measure of the model's overall performance, with lower MSE values indicating better fit and predictive accuracy. A higher MSE (like 28.849) suggests higher prediction errors.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.gofplots import ProbPlot
import sklearn.datasets
from sklearn.model_selection import train_test_split
from statsmodels.formula.api import ols
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
data=pd.read_csv('HousingData.csv')
print(data)
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273

     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    NaN  36.2
..       ...     ...    ...   ...
501     21.0  391.99    NaN  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]
```

```
print(np.shape(data))
```

```
(506, 14)
```

```
print(data.describe())
```

```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  486.000000  486.000000  486.000000  486.000000  506.000000  506.000000
mean     3.611874   11.211934   11.083992    0.069959    0.554695    6.284634
std      8.720192   23.388876    6.835896    0.255340    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.081900    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.253715    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.560263   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  486.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.518519    3.795043    9.549407  408.237154   18.455534  356.674032
std     27.999513    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.175000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     76.800000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     93.975000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  486.000000  506.000000
mean    12.715432   22.532806
std      7.155871    9.197104
min      1.730000    5.000000
25%      7.125000   17.025000
50%     11.430000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

```
data.isnull().sum()
```

```
CRIM       20
ZN         20
INDUS      20
CHAS       20
```

```
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

```
data.duplicated().sum()
```

```
0
```

```
data = data.dropna()
data.isnull().sum()
```

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

```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

```
x = data.drop(['MEDV'], axis=1)
y = data['MEDV']
```

```
X = pd.DataFrame(np.c_[data['RM'] ,data['TAX'], data['RAD'],data['CRIM'],data['ZN']], columns = ['RM', 'TAX', 'RAD','CRIM','ZN'])
Y = data['MEDV']
```

Double-click (or enter) to edit

```
print(X)
```

```
        RM    TAX  RAD     CRIM    ZN
0    6.575  296.0  1.0  0.00632  18.0
1    6.421  242.0  2.0  0.02731   0.0
2    7.185  242.0  2.0  0.02729   0.0
3    6.998  222.0  3.0  0.03237   0.0
4    6.430  222.0  3.0  0.02985   0.0
..     ...    ...  ...      ...   ...
389  5.569  391.0  6.0  0.17783   0.0
390  6.027  391.0  6.0  0.22438   0.0
391  6.120  273.0  1.0  0.04527   0.0
392  6.976  273.0  1.0  0.06076   0.0
393  6.794  273.0  1.0  0.10959   0.0

[394 rows x 5 columns]
```

```
print(Y)
```

```
0      24.0
1      21.6
2      34.7
3      33.4
5      28.7
       ...
499    17.5
500    16.8
502    20.6
503    23.9
504    22.0
Name: MEDV, Length: 394, dtype: float64
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size =0.2, random_state = 0)
```

```
print("xtrain shape : ", xtrain.shape)
print("xtest shape : ", xtest.shape)
print("ytrain shape : ", ytrain.shape)
print("ytest shape : ", ytest.shape)
```

```
xtrain shape :  (315, 13)
xtest shape :  (79, 13)
ytrain shape :  (315,)
ytest shape :  (79,)
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(xtrain, ytrain)

y_pred = regressor.predict(xtest)
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(ytest, y_pred)
mae = mean_absolute_error(ytest,y_pred)
print("Mean Square Error : ", mse)
print("Mean Absolute Error : ", mae)
```

```
Mean Square Error :  28.84987277716687
Mean Absolute Error :  3.484356255630224
```