Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

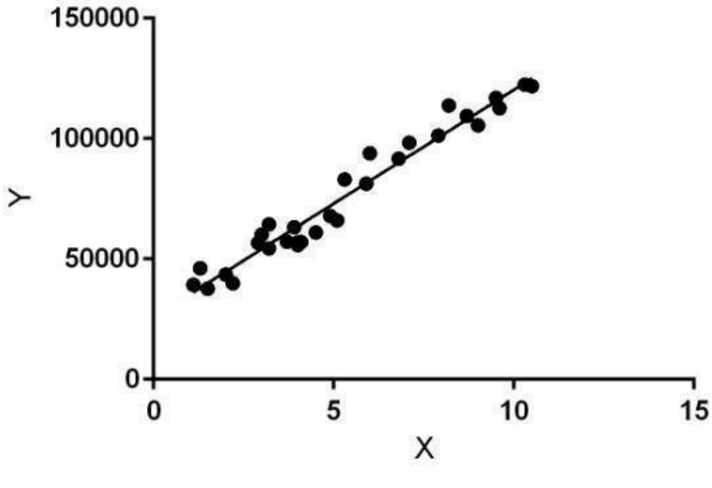
Experiment No. 1
Analyze the Boston Housing dataset and apply appropriate Regression Technique
Date of Performance: 24—07—23
Date of Submission: 07—08—23

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:

A number of factors that capture various facets of the towns that can affect the median home value are part of the features chosen for predicting house prices. The following characteristics are among them: CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, and LSTAT.

Numerous variables affect home prices in various neighbourhoods, including crime rate, percentage of residential land, nitric oxide concentration, average number of rooms, accessibility to highways, property tax rate, etc. Our prediction model is centred on the goal variable MEDV (Median Home Value), and it uses attributes that can forecast median home values.

```
import numpy as np import pandas as pd 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
```

```
data=pd.read_csv('/content/BostonHousing (1).csv')
```

```
pd.read_csv('/content/BostonHousing (1).csv')
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

506 rows × 14 columns

```
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
```

```
print(data.head(5))
```

```
       crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7   4
   0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv  0
396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

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

```
(506, 14)
```

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

```
             crim          zn       indus        chas         nox          rm  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
```

```
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   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  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    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  506.000000  506.000000
mean    12.653063   22.532806  std
7.141062    9.197104  min
1.730000    5.000000  25%
6.950000   17.025000
50%     11.360000   21.200000  75%
16.955000   25.000000  max
37.970000   50.000000
```

```
data.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       506 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: 55.5 KB
```

```
linr = LinearRegression()
```

```
data['medv'] = np.log1p(data['medv'])
```

```
X = data.drop(['medv','b'], axis=1)
Y = data['medv']
```

```
print(X)
```

```
        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
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222   ..
     ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273   505
     0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio  lstat  0
15.3   4.98
1       17.8   9.14
2       17.8   4.03
3       18.7   2.94
```

```
4       18.7   5.33  ..       ...    ...
501     21.0   9.67
502     21.0   9.08
503     21.0   5.64
504     21.0   6.48
505     21.0   7.88

506     rows x 12 columns]
```

```
print(Y)
```

```
0      3.218876
1      3.117950
2      3.575151
3      3.538057
4      3.616309          ...
501    3.152736
502    3.072693
503    3.214868
504    3.135494
505    2.557227
Name: medv, Length: 506, dtype: float64
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=42, test_size=0.3)
```

```
print("x_train shape:",x_train.shape)
print("x_test shape:",x_test.shape)
print("y_train shape:",x_train.shape)
print("y_train shape:",x_test.shape)
```

```
x_train shape: (354, 12)
x_test shape: (152, 12)
y_train shape: (354, 12)
y_train shape: (152, 12)
```

```
linr.fit(x_train, y_train)
```

```
▾LinearRegression
LinearRegression()
```

```
y_pred = linr.predict(x_test)
```

```
print(mean_squared_error(y_test, y_pred))
```

```
0.03112933398095344
```

```
plt.scatter(y_test,y_pred,c ='blue') plt.xlabel("value")
plt.ylabel("Predicted value") plt.title("True value vs
predicted value : Linear Regression") plt.show()
```