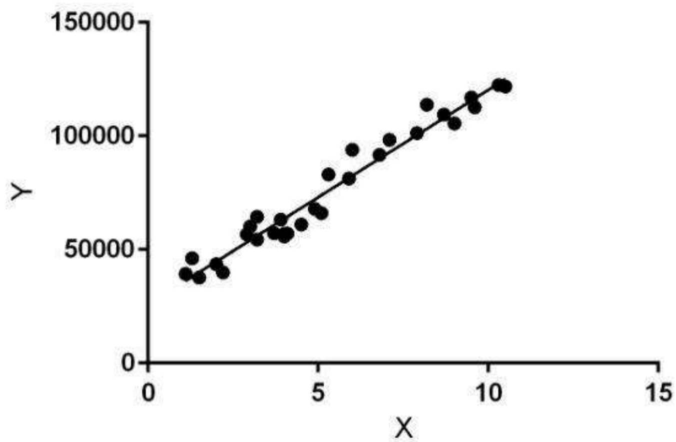
Experiment No. 1
Analyze the Boston Housing dataset and Apply appropriate Regression Technique
Date of Performance: 15/07/2024
Date of Submission: 05/08/2024

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

Conclusion:

The RMSE of 4.896 indicates that, on average, your model's predictions for housing prices deviate by approximately $4,896 from the actual prices.

The R-squared value of 0.69 suggests that your model is moderately effective in explaining the variability in housing prices based on the features included in the dataset.

```
import seaborn as sns
from scipy import st
import numpy as np
```



```
import pandas as pd
import matplotlib.pypl
```

```
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

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

```
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237154	18.455534	35
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.707259	168.537116	2.164946	9
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	[illegible]
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.000000	279.000000	17.400000	37
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.000000	330.000000	19.050000	39
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.000000	666.000000	20.200000	39
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	39

```
missing_values = data.isna().sum() #for each column this stores the number of na in an array
print(missing_values)
```

```
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
```

```
# List of columns with NaN values
na_columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'AGE', 'LSTAT']

# Fill NaN values in these columns with the mean of the column
data[na_columns] = data[na_columns].fillna(data.mean())

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  65.200000  4.0900    1  296
1   0.02731   0.0   7.07   0.0  0.469  6.421  78.900000  4.9671    2  242
2   0.02729   0.0   7.07   0.0  0.469  7.185  61.100000  4.9671    2  242
3   0.03237   0.0   2.18   0.0  0.458  6.998  45.800000  6.0622    3  222
4   0.06905   0.0   2.18   0.0  0.458  7.147  54.200000  6.0622    3  222
..      ...   ...    ...   ...    ...    ...        ...     ...  ...  ...
```

```
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.100000  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.700000  2.2875    1  273
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.000000  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.300000  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030  68.518519  2.5050    1  273

     PTRATIO       B      LSTAT  MEDV
0       15.3  396.90   4.980000  24.0
1       17.8  396.90   9.140000  21.6
2       17.8  392.83   4.030000  34.7
3       18.7  394.63   2.940000  33.4
4       18.7  396.90  12.715432  36.2
..       ...     ...        ...   ...
501     21.0  391.99  12.715432  22.4
502     21.0  396.90   9.080000  20.6
503     21.0  396.90   5.640000  23.9
504     21.0  393.45   6.480000  22.0
505     21.0  396.90   7.880000  11.9

[506 rows x 14 columns]
```

```
new_missing_values = data.isna().sum() #for each column this stores the number of na in an arra
print("Earlier Missing Values\n",missing_values,"New missing v",new_missing_values)
```

```
Earlier Missing Values
 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 New missing v 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
```

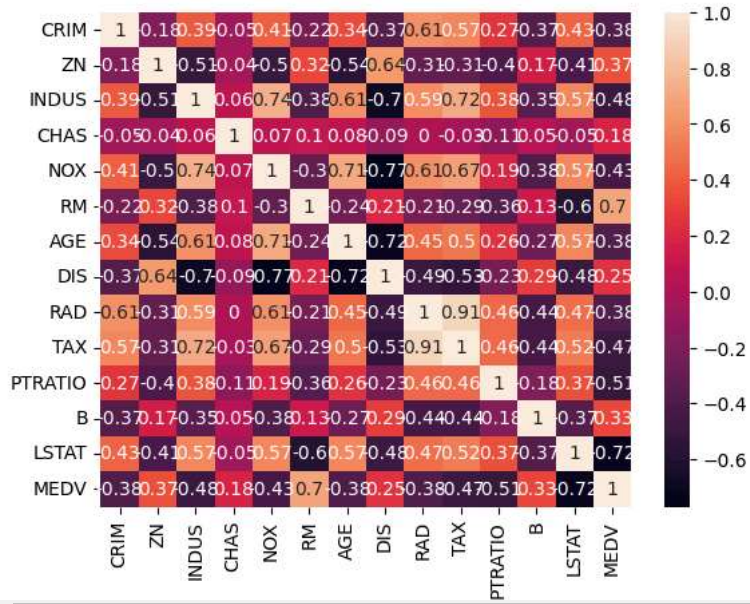
```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.980000	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.140000	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.030000	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.940000	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	12.715432	36.2

```
correlation_matrix = data.corr().round(2)
```

```
sns.heatmap(data=correlation_matrix, annot=True)
```

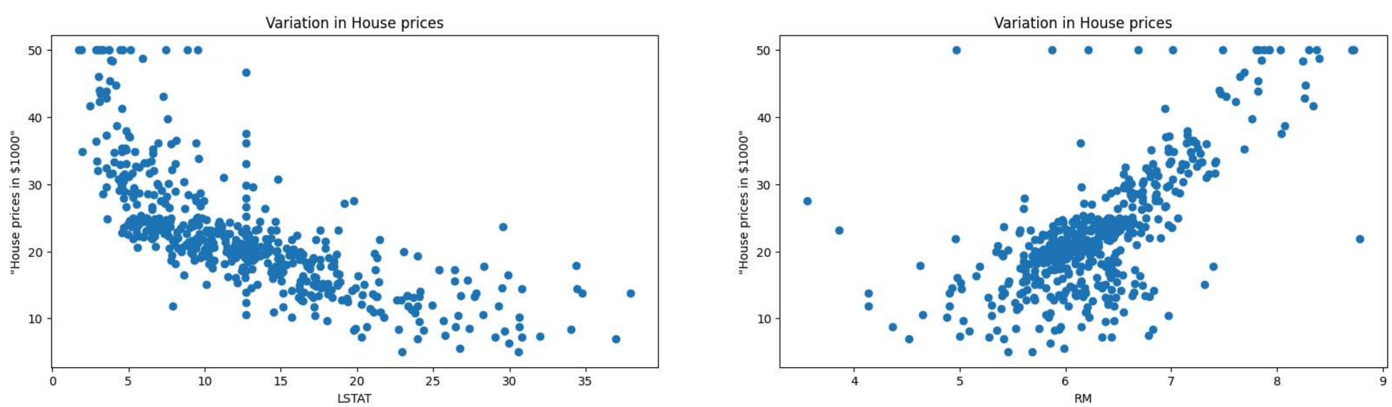
<Axes: >




```python
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = data['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = data[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```



```
X = data.RM #all attributes except the dependent
y = data.MEDV #dependent attribute
X = np.array(X).reshape(-1,1)
y = np.array(y).reshape(-1,1)

print(X.shape)
print(y.shape)
```

```
(506, 1)
(506, 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)

```

```
reg_1 = LinearRegression()
reg_1.fit(X_train, y_train)

```

```
▾ LinearRegression
LinearRegression()
```

```
y_train_predict_1 = reg_1.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict_1)))
r2 = round(reg_1.score(X_train, y_train),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

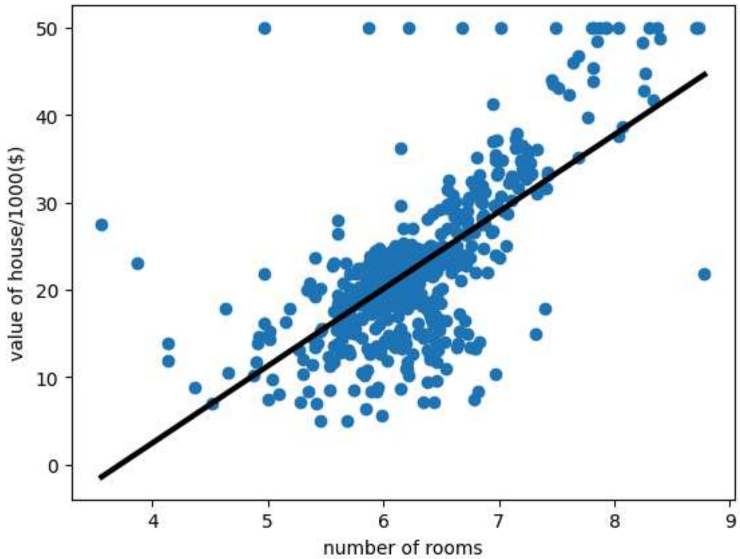
```
The model performance for training set
--------------------------------------
RMSE is 6.972277149440585
R2 score is 0.43
```

```
y_pred_1 = reg_1.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred_1)))
r2 = round(reg_1.score(X_test, y_test),2)

print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.895963186952216
R^2: 0.69
```

```
prediction_space = np.linspace(min(X), max(X)).reshape(-1,1)
plt.scatter(X,y)
plt.plot(prediction_space, reg_1.predict(prediction_space), color = 'black', linewidth = 3)
plt.ylabel('value of house/1000($)')
plt.xlabel('number of rooms')
plt.show()
```



50
40
30
20
10
0
value of house/1000($)
4
5
6
7
8
9
number of rooms