



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:
Date of Submission:



# Vidyavardhini's College of Engineering & Technology

## Department of Computer Engineering

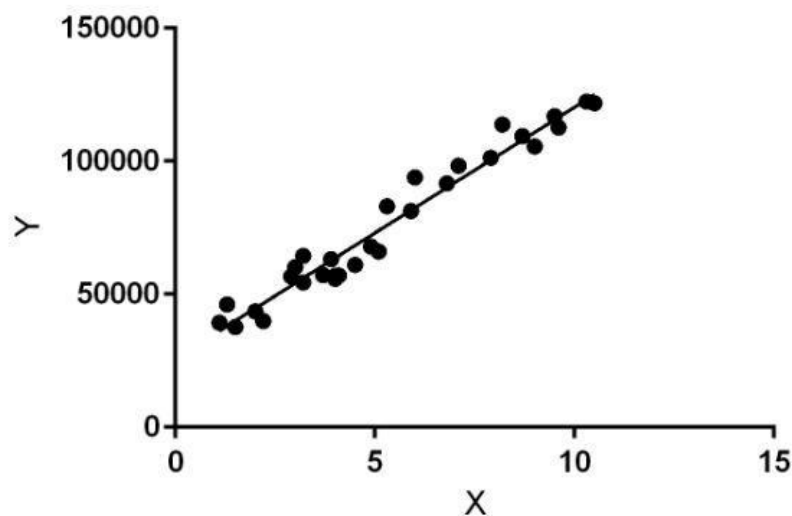
---

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

**Objective:** Ability to perform various feature engineering tasks, apply linear regression on the given dataset and minimize 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.



# Vidyavardhini's College of Engineering & Technology

## Department of Computer Engineering

---

### **Dataset:**

The Boston Housing Dataset

The Boston Housing Dataset is 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 centers

RAD - index of accessibility to radial highways

TAX - full-value property-tax rate per \$10,000

PTRATIO - pupil-teacher ratio by town

B -  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of blacks by town

LSTAT - % lower status of the population

MEDV - Median value of owner-occupied homes in \$1000's

```
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
```

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

```
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	
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	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.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2



#The price of the house indicated by the variable MEDV is the target variable and the rest are the independent variables based on which w

```
# Info of dataframe
df.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

# checking the number of rows and Columns in the data frame
df.shape

(506, 14)

# check for missing values
df.isnull().sum()

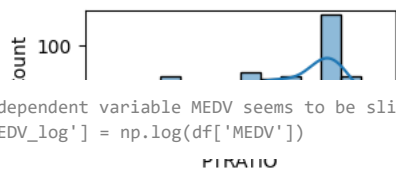
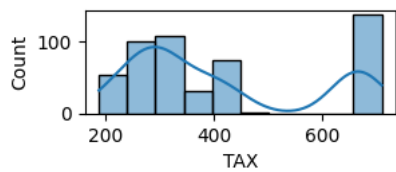
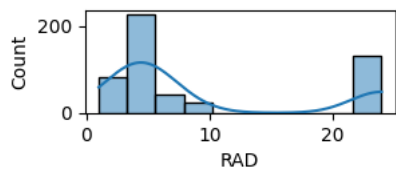
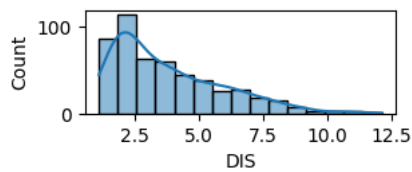
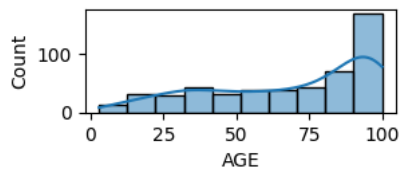
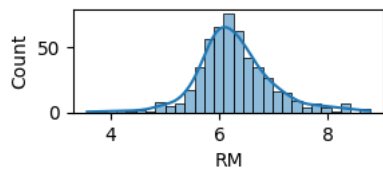
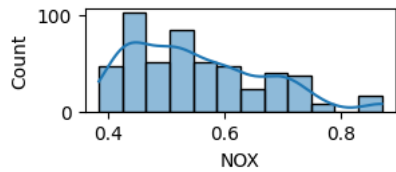
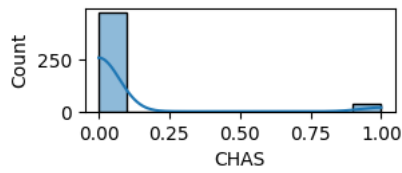
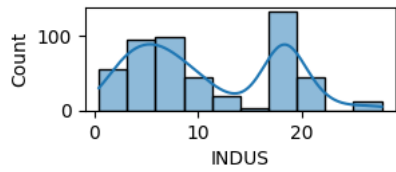
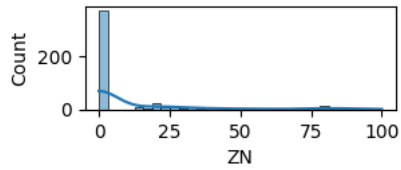
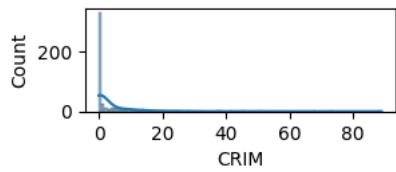
CRIM      0
ZN        0
INDUS     0
CHAS      0
NOX       0
RM        0
AGE       0
DIS       0
RAD       0
TAX       0
PTRATIO   0
B         0
LSTAT     0
MEDV      0
dtype: int64

# statistical measures of the dataset
df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.

```
correlation = df.corr()

#plot all the columns to look at their distributions
for i in df.columns:
    plt.figure(figsize=(3, 1))
    sns.histplot(data=df, x=i, kde = True)
    plt.show()
```

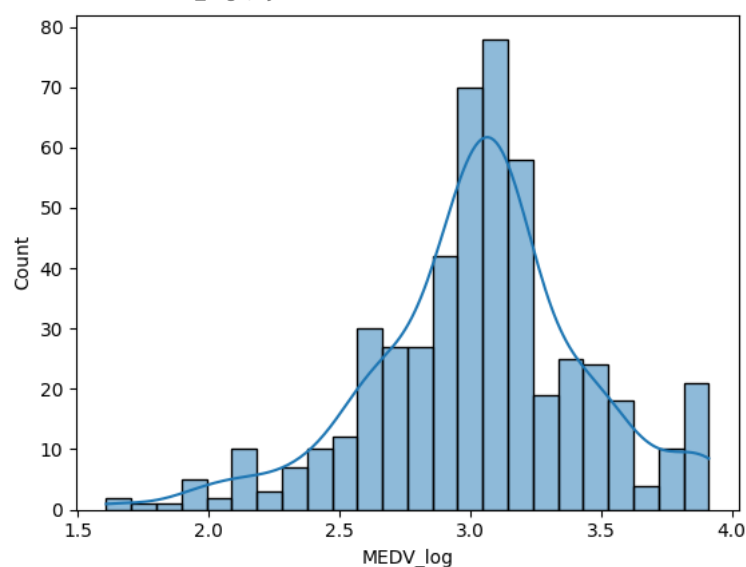


#The dependent variable MEDV seems to be slightly right skewed, apply a log transformation on the 'MEDV' column and check the distribution  
`df['MEDV_log'] = np.log(df['MEDV'])`

PTRATIO

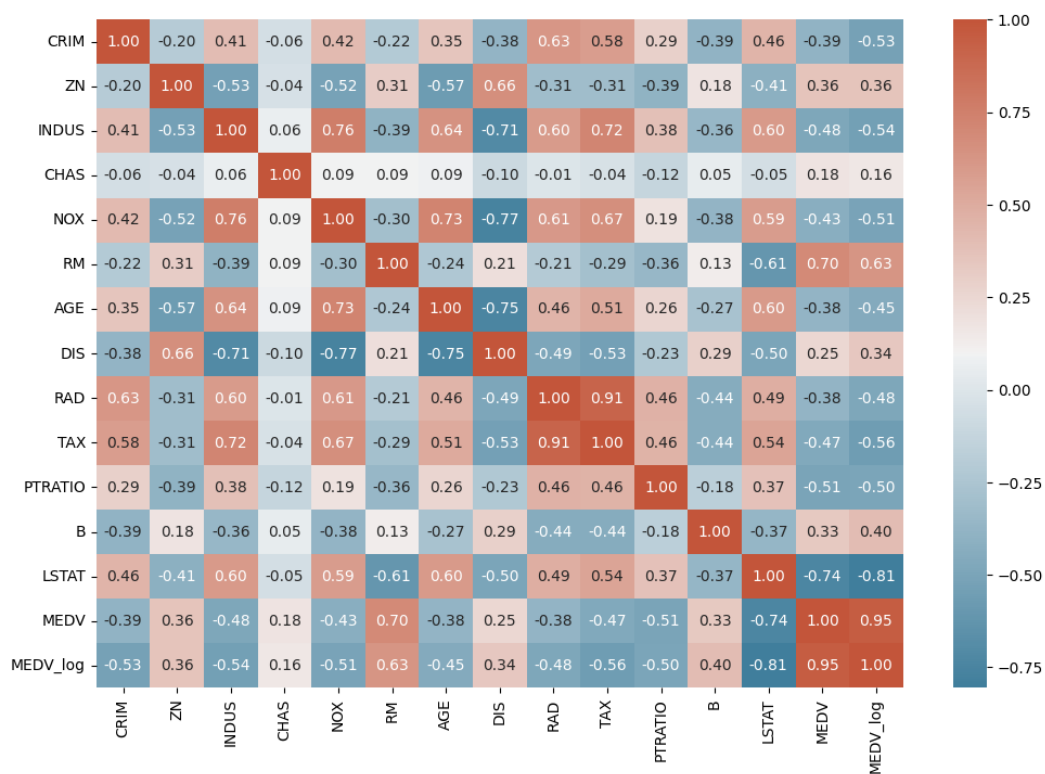
```
sns.histplot(data=df, x='MEDV_log', kde = True)
```

```
<Axes: xlabel='MEDV_log', ylabel='Count'>
```



The log-transformed variable (MEDV\_log) appears to have a nearly normal distribution without skew, and hence we can proceed.

```
plt.figure(figsize=(12,8))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(df.corr(),annot=True,fmt='.2f',cmap=cmap )
plt.show()
```



```
# separate the dependent and independent variable
Y = df['MEDV_log']
X = df.drop(columns = {'MEDV', 'MEDV_log'})

# add the intercept term
X = sm.add_constant(X)
```

```
# splitting the data in 70:30 ratio of train to test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30 , random_state=1)

#Check for Multicollinearity
#use the Variance Inflation Factor (VIF), to check if there is multicollinearity in the data.
#Features having a VIF score > 5 will be dropped/treated till all the features have a VIF score < 5

from statsmodels.stats.outliers_influence import variance_inflation_factor

# function to check VIF
def checking_vif(train):
    vif = pd.DataFrame()
    vif["feature"] = train.columns

    # calculating VIF for each feature
    vif["VIF"] = [
        variance_inflation_factor(train.values, i) for i in range(len(train.columns))
    ]
    return vif

print(checking_vif(X_train))
```

	feature	VIF
0	const	585.099960
1	CRIM	1.993439
2	ZN	2.743911
3	INDUS	4.004462
4	CHAS	1.078490
5	NOX	4.430555
6	RM	1.879494
7	AGE	3.155351
8	DIS	4.361514
9	RAD	8.369185
10	TAX	10.194047
11	PTRATIO	1.948555
12	B	1.385213
13	LSTAT	2.926462

There are two variables with a high VIF - RAD and TAX. Remove TAX as it has the highest VIF values and check the multicollinearity again.

```
# create the model after dropping TAX
X_train = X_train.drop(['TAX'],1)

# check for VIF
print(checking_vif(X_train))
```

	feature	VIF
0	const	581.372515
1	CRIM	1.992236
2	ZN	2.483521
3	INDUS	3.277778
4	CHAS	1.052841
5	NOX	4.397232
6	RM	1.876243
7	AGE	3.154114
8	DIS	4.339453
9	RAD	2.978247
10	PTRATIO	1.914523
11	B	1.384927
12	LSTAT	2.924524

<ipython-input-17-31a12e8753ff>:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu  
X\_train = X\_train.drop(['TAX'],1)

```
#create the linear regression model using statsmodels OLS and print the model summary.
model1 = sm.OLS(y_train, X_train).fit()

# get the model summary
model1.summary()
```



OLS Regression Results

Dep. Variable:	MEDV_log	R-squared:	0.771
Model:	OLS	Adj. R-squared:	0.763
Method:	Least Squares	F-statistic:	95.56
Date:	Tue, 01 Aug 2023	Prob (F-statistic):	2.97e-101
Time:	01:45:20	Log-Likelihood:	78.262
No. Observations:	354	AIC:	-130.5
Df Residuals:	341	BIC:	-80.22
Df Model:	12		
Covariance Type: nonrobust			

	coef	std err	t	P> t	[0.025	0.975]
const	4.4999	0.253	17.767	0.000	4.002	4.998
CRIM	-0.0122	0.002	-7.005	0.000	-0.016	-0.009
ZN	0.0010	0.001	1.417	0.157	-0.000	0.002
INDUS	-0.0002	0.003	-0.066	0.947	-0.006	0.005
CHAS	0.1164	0.039	3.008	0.003	0.040	0.193
NOX	-1.0297	0.187	-5.509	0.000	-1.397	-0.662
RM	0.0569	0.021	2.734	0.007	0.016	0.098
AGE	0.0003	0.001	0.390	0.697	-0.001	0.002
DIS	-0.0496	0.010	-4.841	0.000	-0.070	-0.029
RAD	0.0080	0.002	3.885	0.000	0.004	0.012
PTRATIO	-0.0458	0.007	-6.762	0.000	-0.059	-0.033
B	0.0002	0.000	1.796	0.073	-2.35e-05	0.001

Independent variables (ZN, AGE, and INDUS) have a high p-value and low t, which implies a minimum significance. Drop insignificant variables from the above model and create the regression model again

```
Skew: 0.387 Prob(JB): 1.34e-22

# create the model after dropping TAX
Y = df['MEDV_log']

X = df.drop(columns = {'MEDV', 'MEDV_log', 'ZN', 'AGE', 'INDUS', 'TAX'})
X = sm.add_constant(X)

#splitting the data in 70:30 ratio of train to test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30 , random_state=1)

# create the model
model2 = sm.OLS(y_train, X_train).fit()

# get the model summary
model2.summary()
```

OLS Regression Results

Dep. Variable:	MEDV_log	R-squared:	0.769
Model:	OLS	Adj. R-squared:	0.763
Method:	Least Squares	F-statistic:	127.5
Date:	Tue, 01 Aug 2023	Prob (F-statistic):	6.21e-104
Time:	01:45:24	Log-Likelihood:	77.190
No. Observations:	354	AIC:	-134.4
Df Residuals:	344	BIC:	-95.69
Df Model:	9		
Covariance Type: nonrobust			

	coef	std err	t	P> t	[0.025	0.975]
const	4.5147	0.252	17.925	0.000	4.019	5.010
CRIM	-0.0119	0.002	-6.909	0.000	-0.015	-0.009
CHAS	0.1165	0.039	3.016	0.003	0.041	0.192
NOX	-1.0234	0.168	-6.086	0.000	-1.354	-0.693
RM	0.0622	0.020	3.089	0.002	0.023	0.102
DIS	-0.0434	0.008	-5.488	0.000	-0.059	-0.028
RAD	0.0083	0.002	4.092	0.000	0.004	0.012
PTRATIO	-0.0490	0.006	-7.936	0.000	-0.061	-0.037
B	0.0002	0.000	1.824	0.069	-1.95e-05	0.001
LSTAT	-0.0287	0.002	-12.577	0.000	-0.033	-0.024

Omnibus: 35.608 Durbin-Watson: 1.927

Prob(Omnibus): 0.000 Jarque-Bera (JB): 104.246

Skew: 0.425 Prob(JB): 2.31e-23

Kurtosis: 5.519 Cond. No. 9.76e+03

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

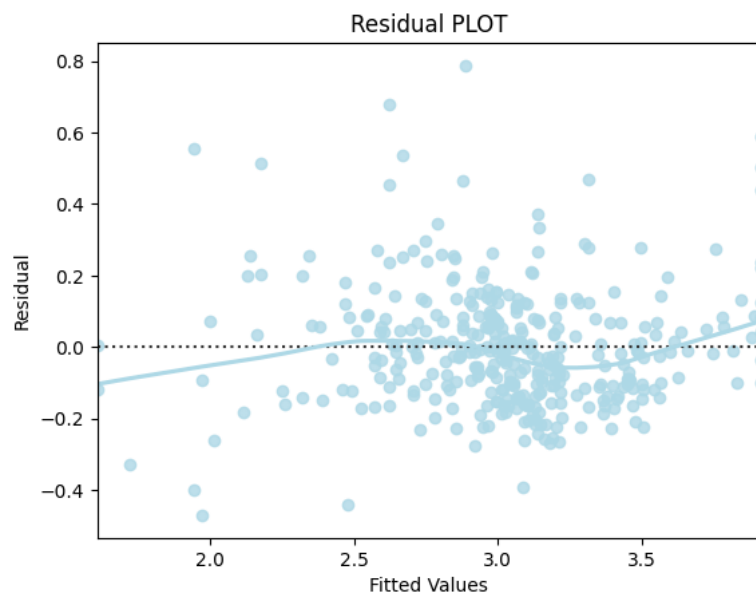
[2] The condition number is large, 9.76e+03. This might indicate that there are strong multicollinearity or other numerical problems.

```
residuals = model2.resid
residuals.mean()
```

```
8.154180406851432e-17
```

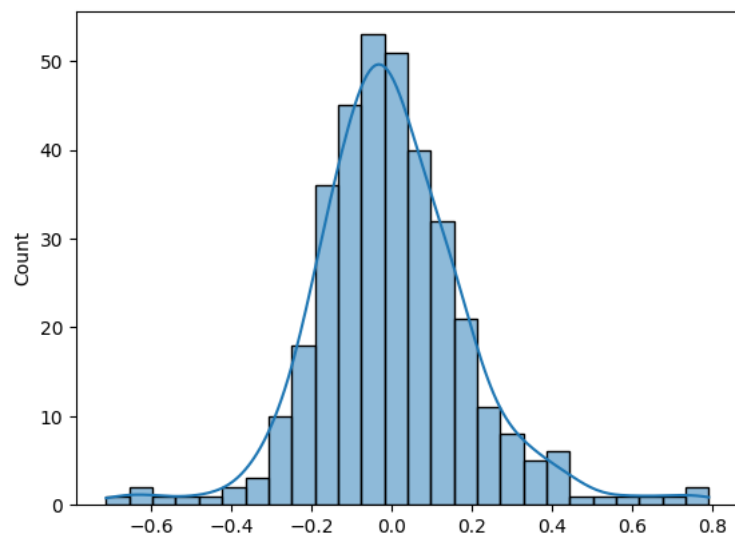
```
# predicted values
fitted = model2.fittedvalues
```

```
#sns.set_style("whitegrid")
sns.residplot(x = y_train, y = residuals , color="lightblue", lowess=True)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
plt.title("Residual PLOT")
plt.show()
```



```
# Plot histogram of residuals
sns.histplot(residuals, kde=True)
```

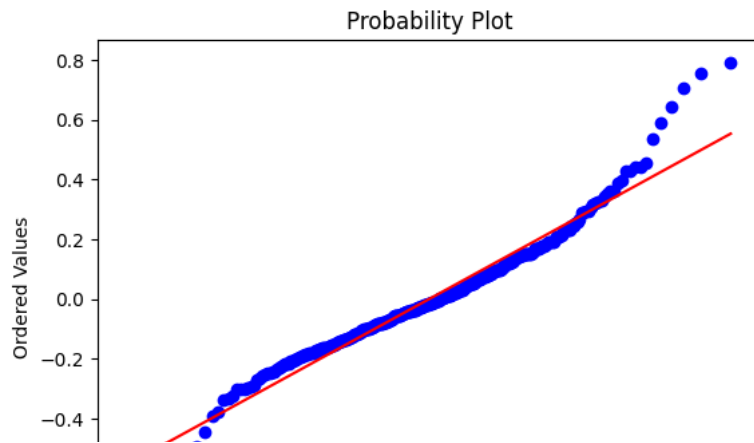
```
<Axes: ylabel='Count'>
```



```
# Plot q-q plot of residuals
```

```
import pylab
import scipy.stats as stats
```

```
stats.probplot(residuals, dist="norm", plot=pylab)
plt.show()
```



```
# RMSE
def rmse(predictions, targets):
    return np.sqrt(((targets - predictions) ** 2).mean())

# MAPE
def mape(predictions, targets):
    return np.mean(np.abs((targets - predictions)) / targets) * 100

# MAE
def mae(predictions, targets):
    return np.mean(np.abs((targets - predictions)))

# Model Performance on test and train data
def model_perf(olsmodel, x_train, x_test):

    # Insample Prediction
    y_pred_train = olsmodel.predict(x_train)
    y_observed_train = y_train

    # Prediction on test data
    y_pred_test = olsmodel.predict(x_test)
    y_observed_test = y_test

    print(
        pd.DataFrame(
            {
                "Data": ["Train", "Test"],
                "RMSE": [
                    rmse(y_pred_train, y_observed_train),
                    rmse(y_pred_test, y_observed_test),
                ],
                "MAE": [
                    mae(y_pred_train, y_observed_train),
                    mae(y_pred_test, y_observed_test),
                ],
                "MAPE": [
                    mape(y_pred_train, y_observed_train),
                    mape(y_pred_test, y_observed_test),
                ],
            }
        )
    )

# Checking model performance
model_perf(model2, X_train, X_test)
```

	Data	RMSE	MAE	MAPE
0	Train	0.194565	0.141729	4.919107
1	Test	0.191732	0.146199	5.069304

The errors have increased slightly on the test data. This suggested further investigation to improve the performance on general data.

```
# import the required function

from sklearn.model_selection import cross_val_score

# build the regression model and
linearregression = LinearRegression()
```

```
cv_Score11 = cross_val_score(linearregression, X_train, y_train, cv = 10)
cv_Score12 = cross_val_score(linearregression, X_train, y_train, cv = 10, scoring = 'neg_mean_squared_error')
```

```
print("RSquared: %0.3f (+/- %0.3f)" % (cv_Score11.mean(), cv_Score11.std() * 2))
print("Mean Squared Error: %0.3f (+/- %0.3f)" % (-1*cv_Score12.mean(), cv_Score12.std() * 2))

RSquared: 0.726 (+/- 0.251)
Mean Squared Error: 0.041 (+/- 0.024)
```

Get model Coefficients in a pandas dataframe with column 'Feature' having all the features and column 'Coefs' with all the corresponding Coefs. Write the regression equation.

```
coef = pd.Series(index = X_train.columns, data = model12.params.values)

coef_df = pd.DataFrame(data = {'Coefs': model12.params.values }, index = X_train.columns)
coef_df
```

	Coefs
const	4.514720
CRIM	-0.011919
CHAS	0.116497
NOX	-1.023431
RM	0.062203
DIS	-0.043391
RAD	0.008288
PTRATIO	-0.049038
B	0.000249
LSTAT	-0.028659

```
#Write the equation of the fit
Equation = "log (Price) ="
print(Equation, end='\t')
for i in range(len(coef)):
    print('(', coef[i], ') * ', coef.index[i], '+', end = ' ')

log (Price) =  ( 4.514720483568433 ) *  const + ( -0.011918775173037938 ) *  CRIM + ( 0.11649715902151694 ) *  CHAS + ( -1.0234312
```



# Vidyavardhini's College of Engineering & Technology

## Department of Computer Engineering

---

### Conclusion:

Selection of some important features:

1. CRIM: The crime rate can affect housing prices; areas with higher crime rates may have lower property values.
2. CHAS: Proximity to the Charles River can be an attractive feature, potentially increasing the house price.
3. NOX: Air pollution (nitric oxides concentration) can influence property value; areas with higher pollution levels may have lower prices.
4. RM: The number of rooms in a dwelling is positively related to the house price; larger homes tend to have higher values.
5. DIS: Shorter distances to employment centers are often preferred, leading to higher demand and potentially higher prices.
6. RAD: Better accessibility to radial highways can be desirable, affecting the housing demand and, consequently, prices.
7. PTRATIO: A lower pupil-teacher ratio is often considered desirable, indicating better educational resources in the area.
8. B: The proportion of Black residents can influence housing prices in certain locations due to historical segregation patterns.
9. LSTAT: The percentage of lower-status population may be indicative of the overall economic condition of the area, affecting property values.

The chosen features appear to be relevant and meaningful in estimating the price of a house.

- Mean Squared Error (MSE): 0.041 (+/- 0.024)
- A lower MSE indicates better model performance, as it means that the model's predictions are closer to the actual house prices.
- The MSE value of 0.041 indicates that, on average, the squared difference between the predicted house prices and the actual house prices is 0.041. This value is relatively low, which suggests that the model is performing well and producing accurate predictions for house prices.
- The standard deviation of the MSE is given as +/- 0.024. A low standard deviation indicates that the model's performance is consistent across the folds, which is a positive sign.
- Mean Squared Error of 0.041 is indicative of a well-performing linear regression model for the given data.