Regression Project: Boston House Price Prediction

Welcome to the project on regression. We will use the **Atlanta house price dataset** for this project.

Objective

The problem at hand is to **predict the housing prices of a town or a suburb based on the features of the locality provided to us**. In the process, we need to **identify the most important features affecting the price of the house**. We need to employ techniques of data preprocessing and build a linear regression model that predicts the prices for the unseen data.

Dataset

Each record in the database describes a house in Atlanta suburb or town. The data was drawn from the Atlanta Standard Metropolitan Statistical Area (SMSA) in 1970. Detailed attribute information can be found below:

Attribute Information:

- **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 Oxide concentration (parts per 10 million)
- **RM:** The average number of rooms per dwelling
- **AGE:** Proportion of owner-occupied units built before 1940
- **DIS:** Weighted distances to five Atlanta employment centers
- **RAD:** Index of accessibility to radial highways
- **TAX:** Full-value property-tax rate per 10,000 dollars
- **PTRATIO:** Pupil-teacher ratio by town
- **LSTAT:** % lower status of the population
- **MEDV:** Median value of owner-occupied homes in 1000 dollars

Importing the necessary libraries and overview of the dataset

In [1]:

```
# Import libraries for data manipulation
import pandas as pd

import numpy as np

# Import libraries for data visualization
import matplotlib.pyplot as plt

import seaborn as sns

from statsmodels.graphics.gofplots import ProbPlot

# Import libraries for building linear regression model
from statsmodels.formula.api import ols

import statsmodels.api as sm

from sklearn.linear_model import LinearRegression

# Import library for preparing data
from sklearn.model_selection import train_test_split

# Import library for data preprocessing
from sklearn.preprocessing import MinMaxScaler

import warnings
warnings.filterwarnings("ignore")
```

Loading the data

In [2]:

```
df = pd.read_csv("Boston.csv")

df.head()
```

Out[2]:

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

Observation:

- The price of the house indicated by the variable MEDV is the target variable and the rest of the variables are independent variables based on which we will predict the house price (MEDV).

Checking the info of the data

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 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  LSTAT    506 non-null    float64
 12  MEDV     506 non-null    float64
dtypes: float64(10), int64(3)
memory usage: 51.5 KB
```

Observations:

- There are a total of **506 non-null observations in each of the columns**. This indicates that there are **no missing values** in the data.
- There are **13 columns** in the dataset and **every column is of numeric data type**.

Exploratory Data Analysis and Data Preprocessing

Summary Statistics of this Dataset

In [4]:

```
# Write your code here
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

**Observations:

-- Per capita crime rate by town (CRIM) and Proportion of residential land zoned for lots over 25,000 sq.ft.(ZN) have high standard deviations above mean indicating large outlier values to the right.

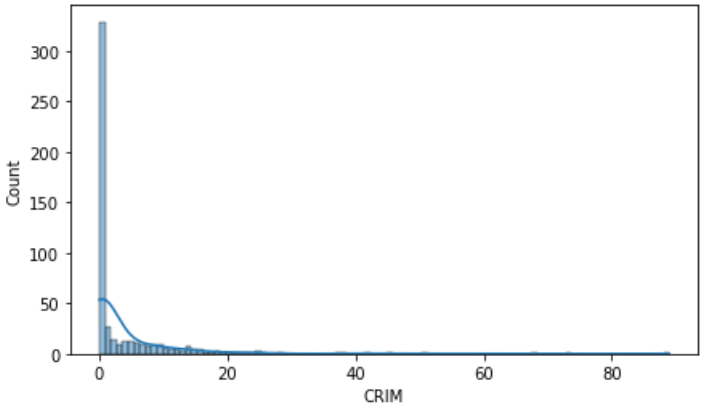
-- The average % lower status of the population(LSTAT) is 12.6% and maximum is approx. 38%

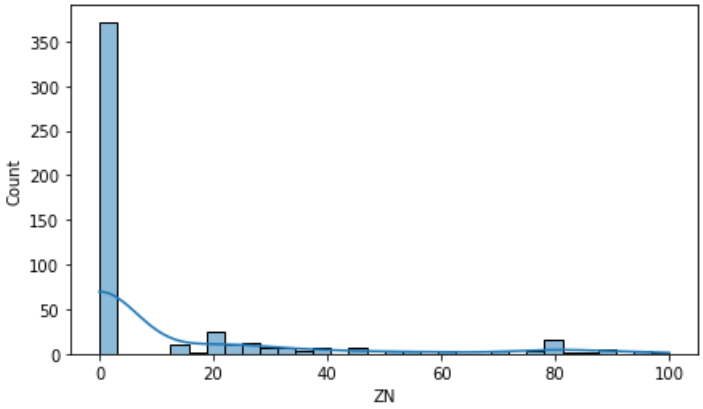
Univariate Analysis

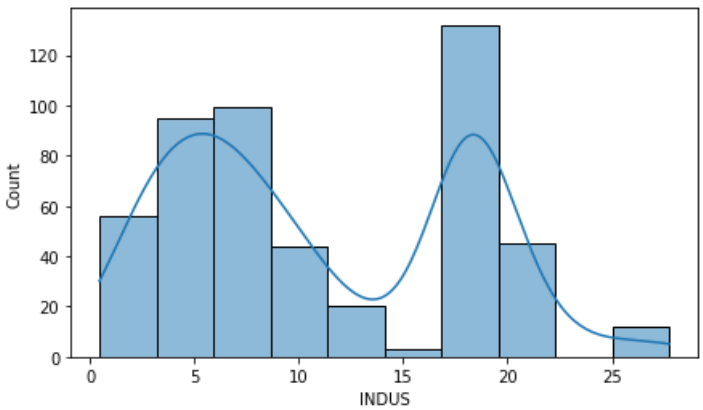
Check the distribution of the variables

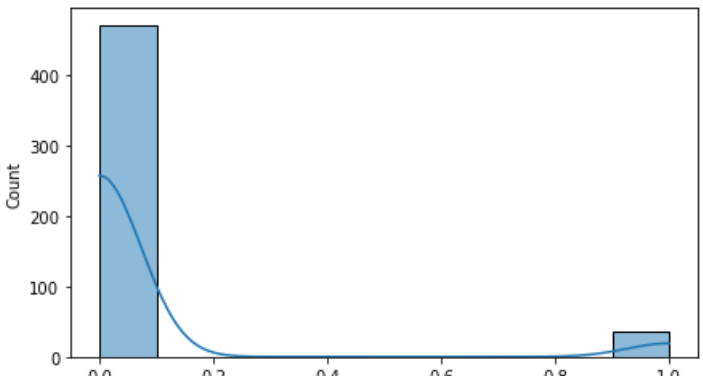
In [5]:

```
# Plotting all the columns to look at their distributions
for i in df.columns:

    plt.figure(figsize = (7, 4))

    sns.histplot(data = df, x = i, kde = True)

    plt.show()
```

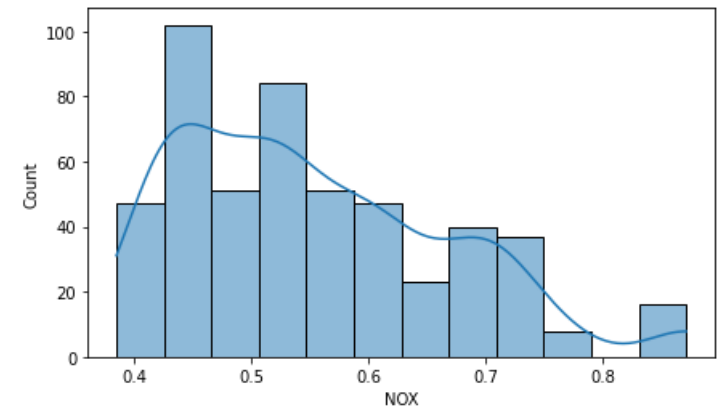


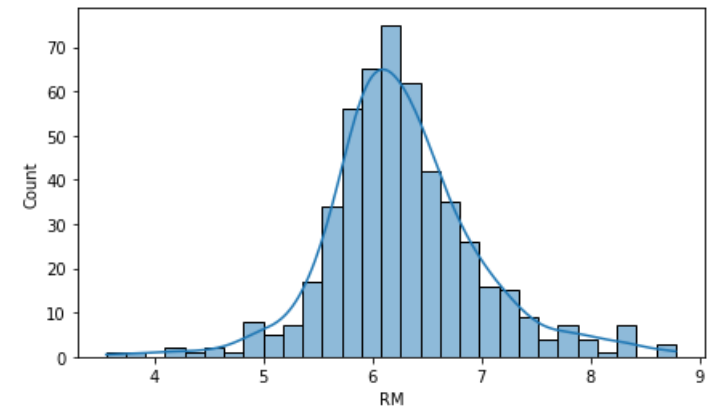


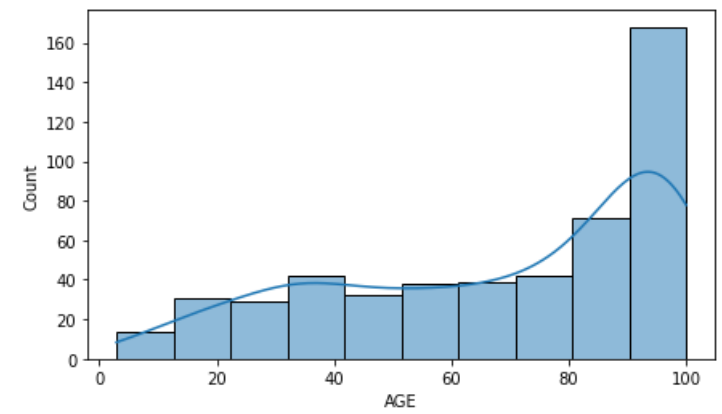


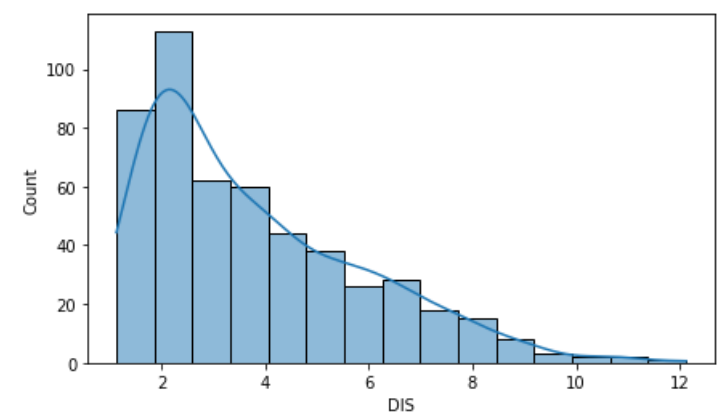


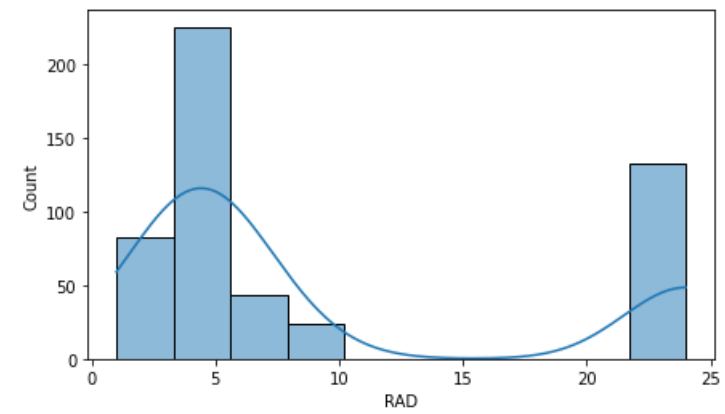

0.0
0.2
0.4
0.6
0.8
1.0
CHAS

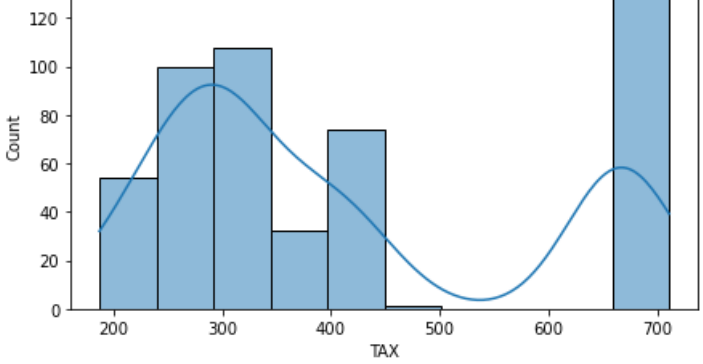

Count
NOX

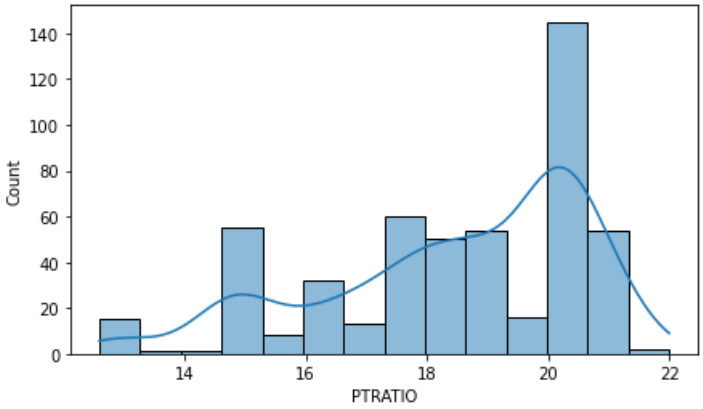

Count
RM

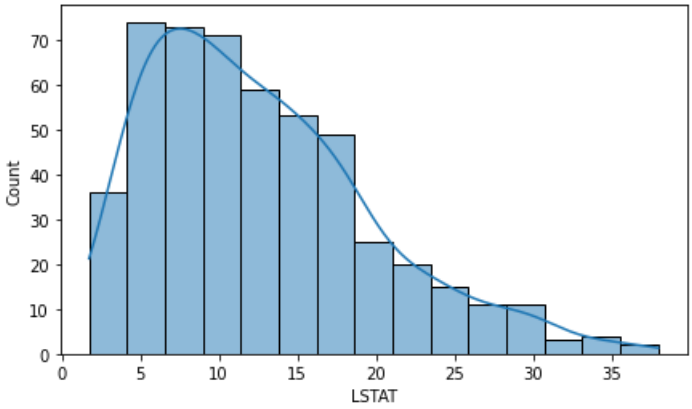

Count
AGE

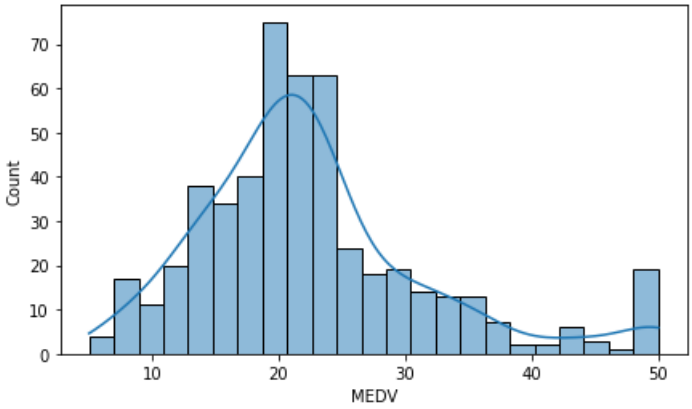

Count
DIS


Count
RAD









Observations:

- **The variables CRIM and ZN are positively skewed.** This suggests that most of the areas have lower crime rates and most residential plots are under the area of 25,000 sq.ft.
- **The variable CHAS, with only 2 possible values 0 and 1, follows a binomial distribution** , and the majority of the houses are away from Charles river (CHAS = 0).
- The distribution of the variable AGE suggests that many of the owner-occupied houses were built before 1940.
- **The variable DIS** (average distances to five Boston employment centers) **has a nearly exponential distribution**, which indicates that most of the houses are closer to these employment centers.
- **The variables TAX and RAD have a bimodal distribution.** , indicating that the tax rate is possibly higher for some properties which have a high index of accessibility to radial highways.
- The dependent variable MEDV seems to be slightly right skewed.

As the dependent variable is sightly skewed, apply a **log transformation on the 'MEDV' column** and check the distribution of the transformed column.

In [6]:

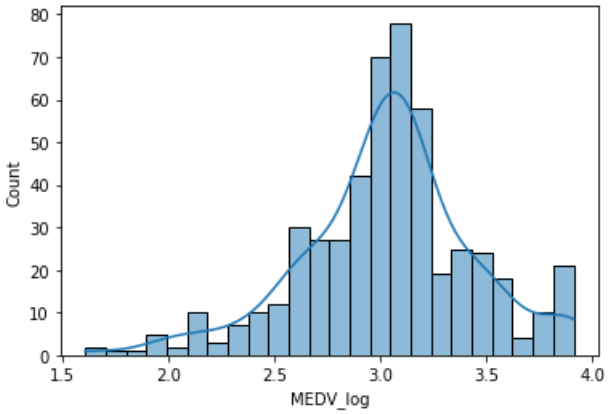
```
df['MEDV_log'] = np.log(df['MEDV'])
```

In [7]:

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

Out[7]:

<AxesSubplot:xlabel='MEDV_log', ylabel='Count'>



Observation:

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

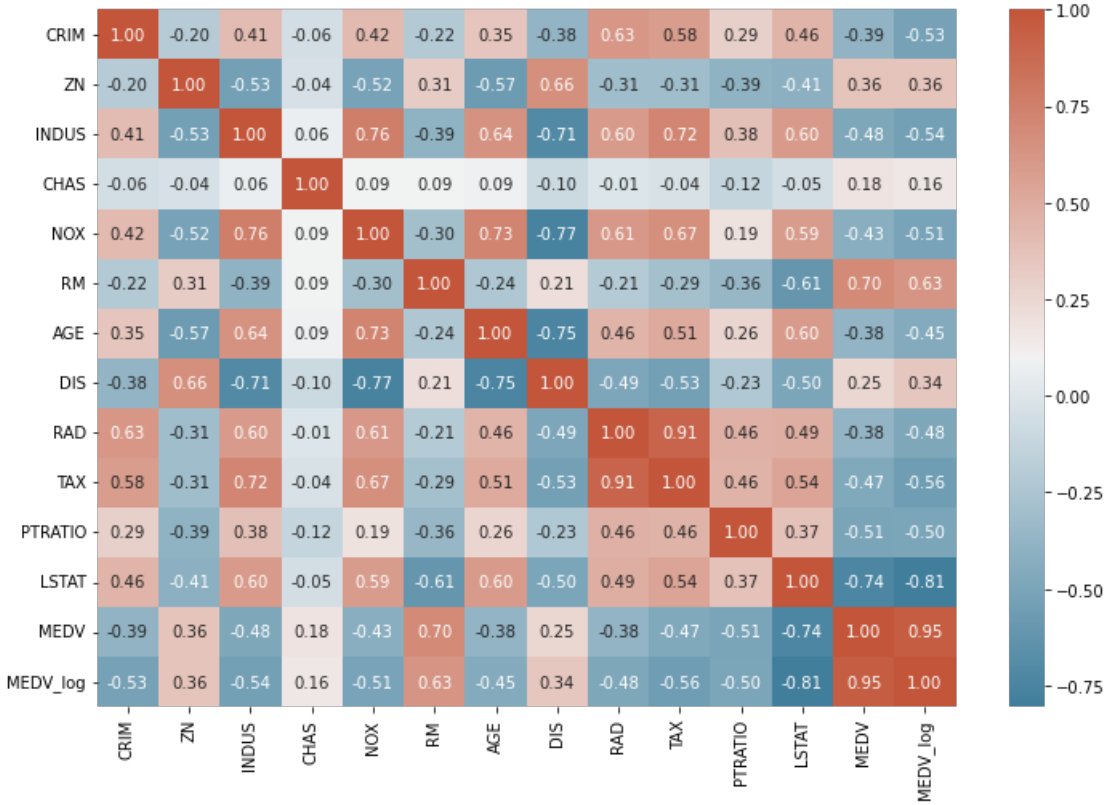
Before creating the linear regression model, it is important to check the bivariate relationship between the variables. Check the same using the heatmap and scatterplot.

Bivariate Analysis

Check the correlation using the heatmap

In [8]:

```
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()
```



**Observations:

--CHAS has very low positive correlation and some low negative correlation with other independent variables

--High postive correlation is seen between AGE vs NOX, TAX vs INDUS, NOX vs INDUS, ZN vs DIS, RAD vs TAX, RM vs MEDV, TAX vs NOX... i.e as Nitric Oxide concerntration increases, Tax, Proportion of non-retail business acres per town and Proportion of owner-occupied old houses also increases.

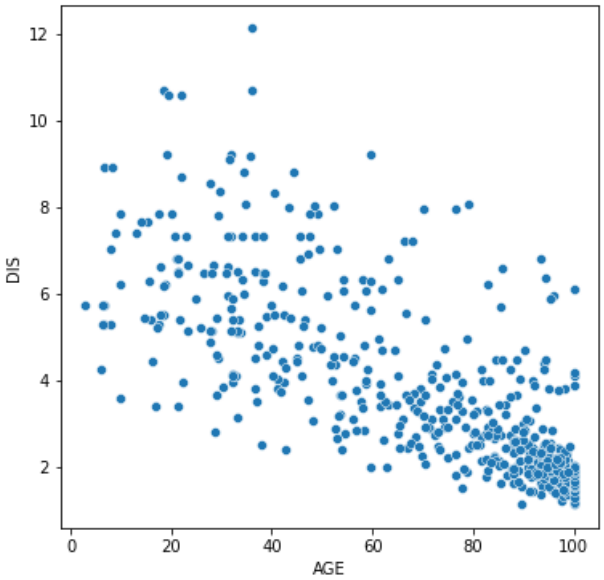
--High negative correlation between DIS and NOX, DIS and INDUS, LSTAT and MEDV, DIS and AGE

Visualize the relationship between the pairs of features having significant correlations.

Visualizing the relationship between the features having significant correlations (>= 0.7 or <= -0.7)

In [9]:

```
# Scatterplot to visualize the relationship between AGE and DIS
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'AGE', y = 'DIS', data = df)

plt.show()
```

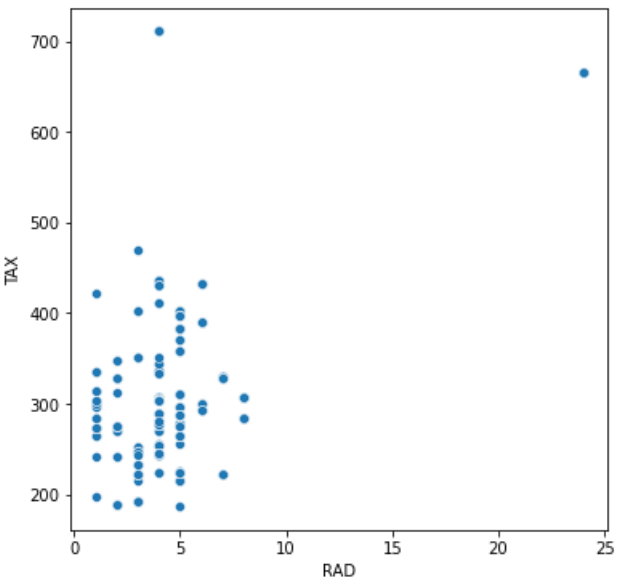


Observations:

- The distance of the houses to the Atlanta employment centers appears to decrease moderately as the the proportion of the old houses increase in the town. It is possible that the Atlanta employment centers are located in the established towns where proportion of owner-occupied units built prior to 1940 is comparatively high.

In [10]:

```
# Scatterplot to visulaize the relationship between RAD and TAX
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'RAD', y = 'TAX', data = df)

plt.show()
```



Observations:

- The correlation between RAD and TAX is very high. But, no trend is visible between the two variables.
- The strong correlation might be due to outliers.

Check the correlation after removing the outliers.

In [11]:

```
# Remove the data corresponding to high tax rate
df1 = df[df['TAX'] < 600]

# Import the required function
from scipy.stats import pearsonr

# Calculate the correlation
print('The correlation between TAX and RAD is', pearsonr(df1['TAX'], df1['RAD'])[0])
```

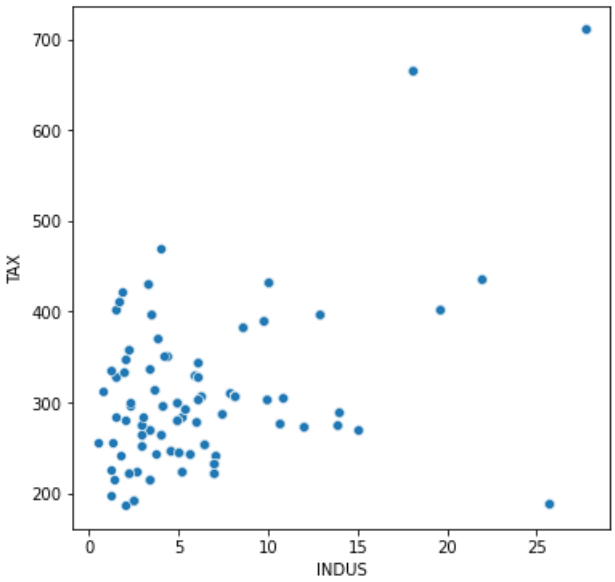
The correlation between TAX and RAD is 0.24975731331429196

Observation:

- The high correlation between TAX and RAD is due to the outliers. The tax rate for some properties might be higher due to some other reason.

In [12]:

```
# Scatterplot to visualize the relationship between INDUS and TAX
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'INDUS', y = 'TAX', data = df)

plt.show()
```

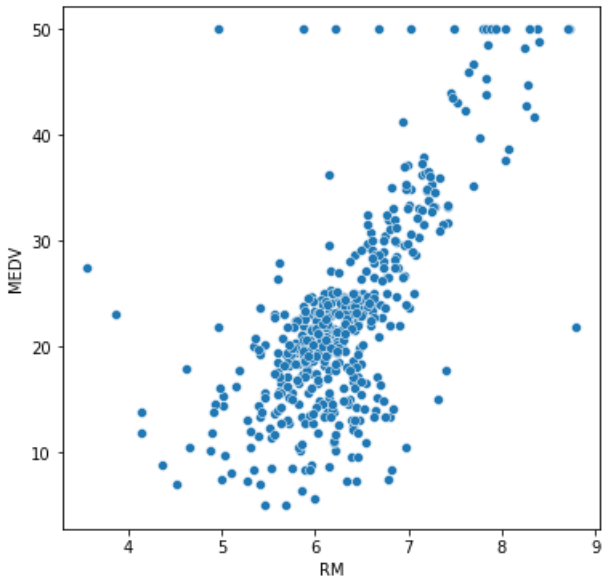


Observations:

- The tax rate appears to increase with an increase in the proportion of non-retail business acres per town. This might be due to the reason that the variables TAX and INDUS are related with a third variable.

In [13]:

```
# Scatterplot to visulaize the relationship between RM and MEDV
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'RM', y = 'MEDV', data = df)

plt.show()
```



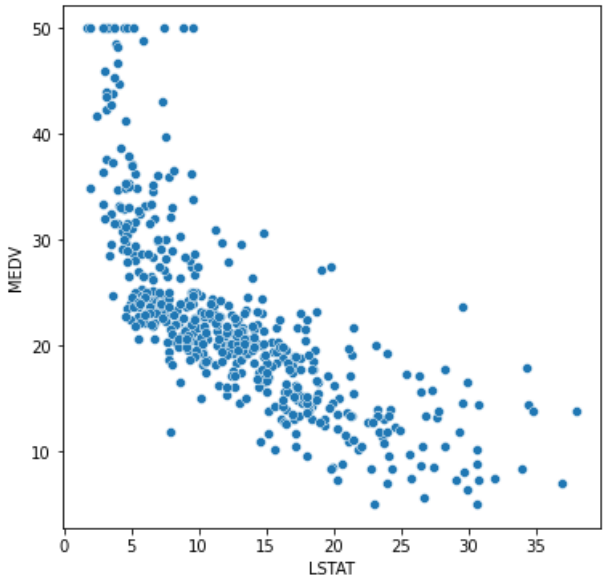
Observations:

- The price of the house seems to increase as the value of RM increases. This is expected as the price is generally higher for more rooms.
- There are a few outliers in a horizontal line as the MEDV value seems to be capped at 50.

In [14]:

```
# Scatterplot to visulaize the relationship between LSTAT and MEDV
plt.figure(figsize = (6, 6))
```

```
sns.scatterplot(x = 'LSTAT', y = 'MEDV', data = df)

plt.show()
```



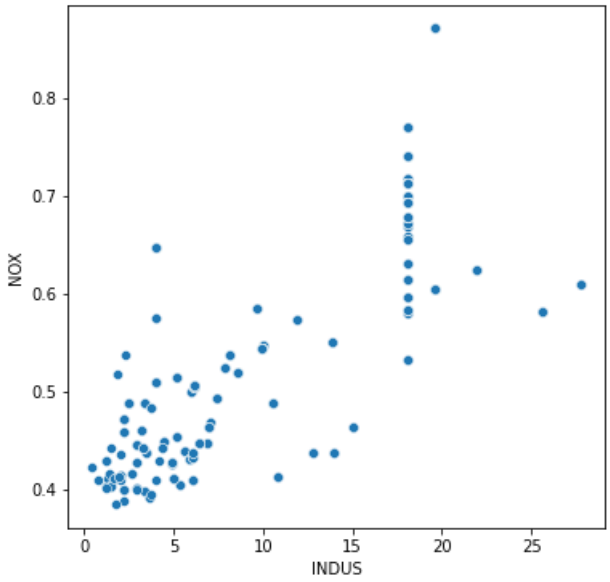
Observations:

- The price of the house tends to decrease with an increase in LSTAT. This is also possible as the house price is lower in areas where lower status people live.
- There are few outliers and the data seems to be capped at 50.

- **Create a scatter plot to visualize the relationship between the remaining features having significant correlations (>= 0.7 or <= -0.7)**
  - INDUS and NOX
  - AGE and NOX
  - DIS and NOX

In [15]:

```
# Scatterplot to visualize the relationship between INDUS and NOX
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'INDUS', y = 'NOX', data = df)

plt.show()
```

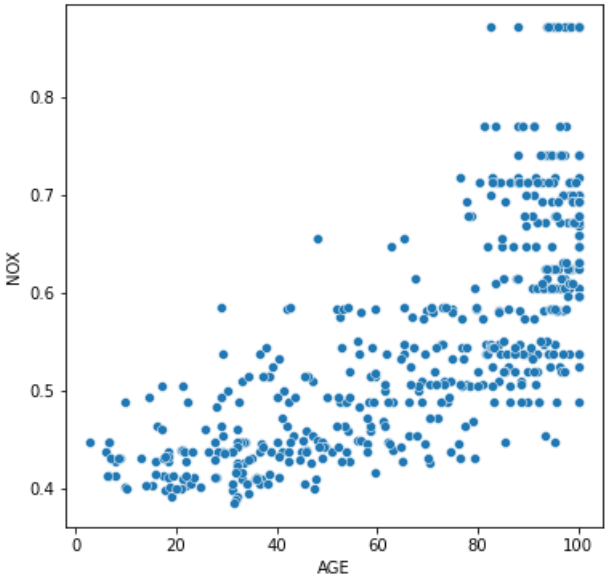


**Observations:

--as proportion of non-retail business acres per town increases, nitric oxide concentration increases... this could be due to presence of big factories/manufacturing buildings that emit Nitric oxide.There are outliers in the vertical line of NOX

In [16]:

```
# Scatterplot to visualize the relationship between AGE and NOX
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'AGE', y = 'NOX', data = df)

plt.show()
```

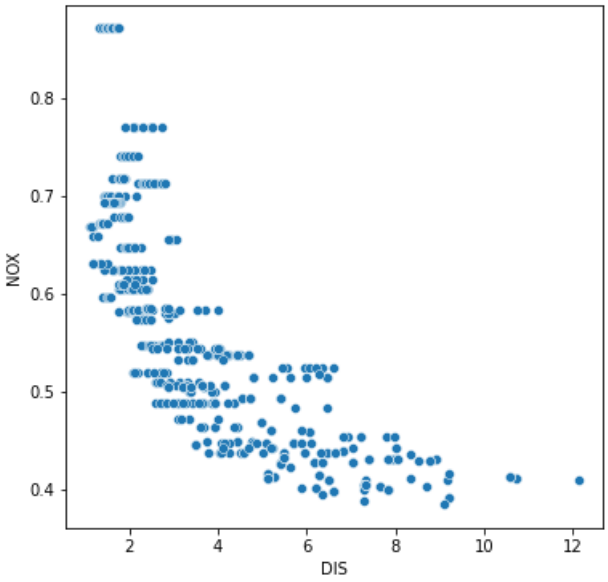


**Observations:

--positive trend between AGE and NOX which indicates as the number of older houses increases, the concentration of nitric acid also increases. These old buildings could be in industrial areas. Age of old buildings are capped at 100

In [17]:

```
# Scatterplot to visualize the relationship between DIS and NOX
plt.figure(figsize = (6, 6))

sns.scatterplot(x = 'DIS', y = 'NOX', data = df)

plt.show()
```



**Observations:

--a negative correlation between DIS and NOX which indicates as distance of the houses to the Atlanta employment centers increases, the concentration of Nitric oxide decreases, this also could be that the farther away we are from industrial areas the lesser air pollution.

We have seen that the variables LSTAT and RM have a linear relationship with the dependent variable MEDV. Also, there are significant relationships among few independent variables, which is not desirable for a linear regression model.

Split the dataset

Split the data into the dependent and independent variables and further split it into train and test set in a ratio of 70:30 for train and test sets.

In [18]:

```
# Separate the dependent variable and indepedent variables
Y = df['MEDV_log']

X = df.drop(columns = {'MEDV', 'MEDV_log'})

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

In [19]:

```
# 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

In [20]:

```
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  535.372593
1      CRIM    1.924114
2        ZN    2.743574
3     INDUS    3.999538
4      CHAS    1.076564
5       NOX    4.396157
6        RM    1.860950
7       AGE    3.150170
8       DIS    4.355469
9       RAD    8.345247
10      TAX   10.191941
11  PTRATIO    1.943409
12    LSTAT    2.861881
```

Observations:

- There are two variables with a high VIF - RAD and TAX (greater than 5).
- Remove TAX as it has the highest VIF values and check the multicollinearity again.

Drop the column 'TAX' from the training data and check if multicollinearity is removed

In [21]:

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

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

```
    feature         VIF
0     const  532.025529
1      CRIM    1.923159
2        ZN    2.483399
3     INDUS    3.270983
4      CHAS    1.050708
5       NOX    4.361847
6        RM    1.857918
7       AGE    3.149005
8       DIS    4.333734
9       RAD    2.942862
10  PTRATIO    1.909750
11    LSTAT    2.860251
```

Create the linear regression model as the VIF is less than 5 for all the independent variables, and we can assume that multicollinearity has been removed between the variables.

create the linear regression model and print the model summary.

Use the sm.OLS() model on the training data

In [22]:

```
# Create the model
model1 = sm.OLS(y_train, X_train)
model1 = model1.fit()

# Get the model summary
```

```
model1.summary()
```

Out[22]:

OLS Regression Results

Dep. Variable:	MEDV_log	**R-squared:**	0.769
Model:	OLS	**Adj. R-squared:**	0.761
Method:	Least Squares	**F-statistic:**	103.3
Date:	Mon, 25 Apr 2022	**Prob (F-statistic):**	1.40e-101
Time:	17:58:27	**Log-Likelihood:**	76.596
No. Observations:	354	**AIC:**	-129.2
Df Residuals:	342	**BIC:**	-82.76
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.6324	0.243	19.057	0.000	4.154	5.111
CRIM	-0.0128	0.002	-7.445	0.000	-0.016	-0.009
ZN	0.0010	0.001	1.425	0.155	-0.000	0.002
INDUS	-0.0004	0.003	-0.148	0.883	-0.006	0.005
CHAS	0.1196	0.039	3.082	0.002	0.043	0.196
NOX	-1.0598	0.187	-5.675	0.000	-1.427	-0.692
RM	0.0532	0.021	2.560	0.011	0.012	0.094
AGE	0.0003	0.001	0.461	0.645	-0.001	0.002
DIS	-0.0503	0.010	-4.894	0.000	-0.071	-0.030
RAD	0.0076	0.002	3.699	0.000	0.004	0.012
PTRATIO	-0.0452	0.007	-6.659	0.000	-0.059	-0.032
LSTAT	-0.0298	0.002	-12.134	0.000	-0.035	-0.025

Omnibus:	30.699	**Durbin-Watson:**	1.923
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	83.718
Skew:	0.372	**Prob(JB):**	6.62e-19
Kurtosis:	5.263	**Cond. No.**	2.09e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.09e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
**Observations:

--We can see that the R-squared for the model is approx. 0.77 and adjusted R-squared is 0.76 suggesting the regression model passed evaluation metric.

--Not all the variables are statistically significant to predict the outcome variable. To check which variables are statistically significant or have predictive power to predict the target variable, we need to check the p-value against all the independent variables.

--From the model summary, we can see most of the variables have a p-value lower than 0.05.

Drop insignificant variables (variables with p-value > 0.05) from the above model and create the regression model again.

Examining the significance of the model

It is not enough to fit a multiple regression model to the data, it is necessary to check whether all the regression coefficients are significant or not. Significance here means whether the population regression parameters are significantly different from zero.

From the above it may be noted that the regression coefficients corresponding to ZN, AGE, and INDUS are not statistically significant at level α = 0.05. The regression coefficients corresponding to these three are not significantly different from 0 in the population; we will eliminate the three features and create a new model.

In [23]:

```
# Create the model after dropping columns 'MEDV', 'MEDV_log', 'TAX', 'ZN', 'AGE', 'INDUS' from df DataFrame
Y = df['MEDV_log']

X = df.drop(columns=['MEDV', 'MEDV_log', 'TAX', 'ZN', 'AGE', 'INDUS'])

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()
```

Out[23]:

OLS Regression Results

Dep. Variable:	MEDV_log	**R-squared:**	0.767
Model:	OLS	**Adj. R-squared:**	0.762
Method:	Least Squares	**F-statistic:**	142.1
Date:	Mon, 25 Apr 2022	**Prob (F-statistic):**	2.61e-104
Time:	17:58:30	**Log-Likelihood:**	75.486
No. Observations:	354	**AIC:**	-133.0
Df Residuals:	345	**BIC:**	-98.15
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.6494	0.242	19.242	0.000	4.174	5.125
CRIM	-0.0125	0.002	-7.349	0.000	-0.016	-0.009
CHAS	0.1198	0.039	3.093	0.002	0.044	0.196
NOX	-1.0562	0.168	-6.296	0.000	-1.386	-0.726
RM	0.0589	0.020	2.928	0.004	0.019	0.098
DIS	-0.0441	0.008	-5.561	0.000	-0.060	-0.028
RAD	0.0078	0.002	3.890	0.000	0.004	0.012
PTRATIO	-0.0485	0.006	-7.832	0.000	-0.061	-0.036
LSTAT	-0.0293	0.002	-12.949	0.000	-0.034	-0.025

Omnibus:	32.514	**Durbin-Watson:**	1.925
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	87.354
Skew:	0.408	**Prob(JB):**	1.07e-19
Kurtosis:	5.293	**Cond. No.**	690.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Check the linear regression assumptions.

linear regression assumptions

1. **Mean of residuals should be 0**
2. **No Heteroscedasticity**
3. **Linearity of variables**
4. **Normality of error terms**

1. Check for mean residuals

In [24]:

```
residuals = model2.resid

np.mean(residuals)
```

Out[24]:

-1.549921521948453e-15
**Observations:

--The mean of residuals is very close to 0. Hence, the corresponding assumption is satisfied.

2. Check for homoscedasticity

- Homoscedasticity - If the residuals are symmetrically distributed across the regression line, then the data is said to be homoscedastic.
- Heteroscedasticity- - If the residuals are not symmetrically distributed across the regression line, then the data is said to be heteroscedastic. In this case, the residuals can form a funnel shape or any other non-symmetrical shape.
- We'll use Goldfeldquandt Test to test the following hypothesis with alpha = 0.05:
  - Null hypothesis: Residuals are homoscedastic
  - Alternate hypothesis: Residuals have heteroscedastic

In [25]:

```
from statsmodels.stats.diagnostic import het_white

from statsmodels.compat import lzip

import statsmodels.stats.api as sms
```

In [26]:

```
name = ["F statistic", "p-value"]

test = sms.het_goldfeldquandt(y_train, X_train)

lzip(name, test)
```

Out[26]:

```
[('F statistic', 1.0835082923425285), ('p-value', 0.3019012006766869)]
```

**Observations:

--the p-value is greater than 0.05, so we fail to reject the null-hypothesis. That means the residuals are homoscedastic.

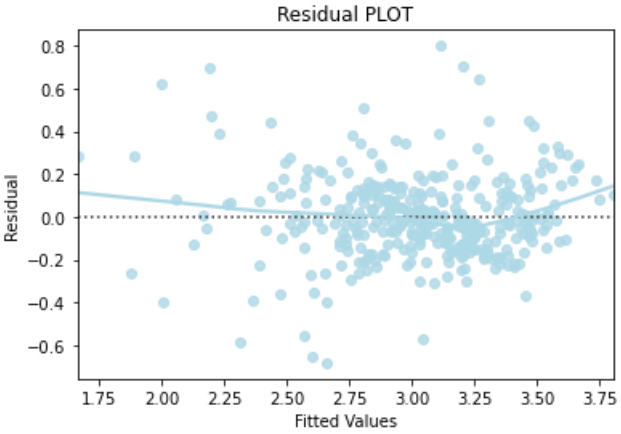
3. Linearity of variables

It states that the predictor variables must have a linear relation with the dependent variable.

To test the assumption, we'll plot residuals and the fitted values on a plot and ensure that residuals do not form a strong pattern. They should be randomly and uniformly scattered on the x-axis.

In [27]:

```
# Predicted values
fitted = model2.fittedvalues

# sns.set_style("whitegrid")
sns.residplot(x = fitted, y = residuals, color = "lightblue", lowess = True)

plt.xlabel("Fitted Values")

plt.ylabel("Residual")

plt.title("Residual PLOT")

plt.show()
```



**Observations:

--there is no pattern in fitted values and residuals, i.e., the residuals are randomly distributed.

4. Normality of error terms

The residuals should be normally distributed.

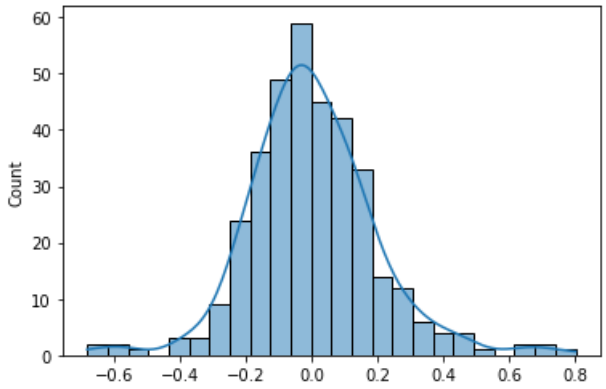
In [28]:

```
# Plot histogram of residuals
```

```
sns.histplot(residuals, kde = True)
```

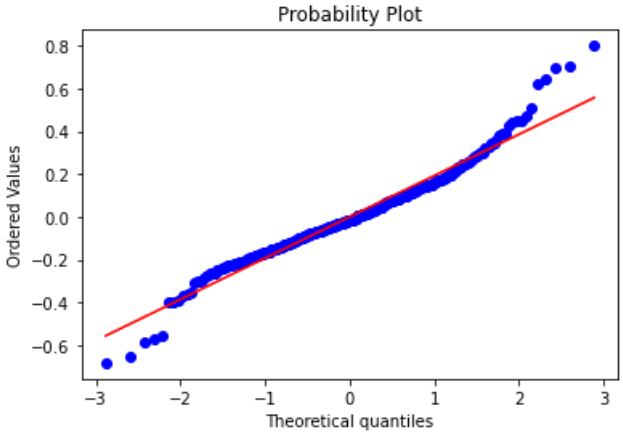
Out[28]:

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



In [29]:

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

import scipy.stats as stats

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

plt.show()
```



**Observations:

--the error terms are normally distributed. The assumption of normality is satisfied.

Check the performance of the model on the train and test data set

comparing model performance of train and test dataset

In [30]:

```
# 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_pref(olsmodel, x_train, x_test):

    # In-sample 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_pref(model2, X_train, X_test)

    Data      RMSE       MAE      MAPE
0  Train  0.195504  0.143686  4.981813
1   Test  0.198045  0.151284  5.257965
```
**Observations:

--The RMSE on the test data is approx 0.198 which is almost similar to the RMSE on the training dataset.

--The MAE on test is approx 0.15 which is slightly similar to the MAE on the training dataset.

----The MAPE on test is approx 5.26 which is somewhat similar to the MAE on the training dataset.

It seems like that our model is just right fit, which gives a generalized performance.

Apply cross validation to improve the model and evaluate it using different evaluation metrics

In [31]:

```
 # Import the required function

 from sklearn.model_selection import cross_val_score

 # Build the regression model and cross-validate
 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: 0.729 (+/- 0.232)
Mean Squared Error: 0.041 (+/- 0.023)
```

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

In [32]:

```
 coef = model2.params

 pd.DataFrame({'Feature' : coef.index, 'Coefs' : coef.values})
```

Out[32]:

	Feature	Coefs
0	const	4.649386
1	CRIM	-0.012500
2	CHAS	0.119773
3	NOX	-1.056225
4	RM	0.058907
5	DIS	-0.044069
6	RAD	0.007848
7	PTRATIO	-0.048504
8	LSTAT	-0.029277

In [33]:

```
# Let us 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) = (4.649385823266638) * const + (-0.012500455079103887) * CRIM + (0.1197731907701965) * CHAS + (-1.0562253516683255) * NOX + (0.058906575109279824) * RM + (-0.044068890799405055) * DIS + (0.00784847460624381) * RAD + (-0.048503620794999) * PTRATIO + (-0.029277040479796866) * LSTAT +

Note: There might be slight variation in the coefficients depending on the library version you are using. There will be no deducting in marks for that as long as your observations are aligned with the output. In case, the coefficients vary too much, please make sure your code is correct.

Write the conclusions and business recommendations derived from the model

Conclusions:

--We performed EDA, univariate and bivariate analysis, on all the variables in the dataset.

--Split the data into training and testing set; We removed multicollinearity from the data using VIF and analyzed the model summary report to drop insignificant features.

--We started the model building process with relevant features

--We checked for different assumptions of linear regression and fixed the model iteratively if any assumptions did not hold true.

--Finally, we compared model performance on training and test sets and did cross validation to improve the model

Recommendations :

--After applying linear regression, we can say CRIM has the most effect on the house prices among all features; no one would want to buy a house around areas were the crime rate is high or in a environment affected by NOX, the increase in these features will affect the prices of houses in Boston negatively. In conclusion, predicting Boston house prices are affected more by the crime rate and the concerntration of Nitric Oxide in the air.

In [ ]: