Regression Project: Boston House Price Prediction

Welcome to the project on regression. We will use the **Boston 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 Boston suburb or town. The data was drawn from the Boston 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 Boston 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

In [1]:
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from statsmodels.formula.api import ols

import statsmodels.api as sm

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.metrics import r2_score, mean_absolute_percentage_error, mean_a

from sklearn.model_selection import train_test_split

from statsmodels.stats.diagnostic import het_white

from statsmodels.compat import lzip

import statsmodels.stats.api as sms

import pylab

import scipy.stats as stats

from sklearn.model_selection import KFold

import sklearn

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

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Loading the dataset

In [3]:
```
data = pd.read_csv('/content/drive/MyDrive/MIT course/Elective Project/Bosto
```

Data Overview

- Observations
- Sanity checks

In [4]:
```
data.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LS
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4
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	4
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5

In [5]:
```
# Check data dimensions
data.shape
```

Out[5]: (506, 13)

In [6]:
```
data.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
```

In [7]:
```
# Checking for duplicate values in the data
data.duplicated().sum()
```

Out[7]: 0

In [8]:
```
# Checking the descriptive statistics of the columns
data.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.6770
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.5000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.1000
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.6240
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.6235
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.0750
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.1884
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.2000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.9550
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.0000

Observations:

- There are no Null values within the data.
- All the variables are numeric.
- The dependent variable in this scenario is MEDV
- There are outliers in certain variables like in ZN.

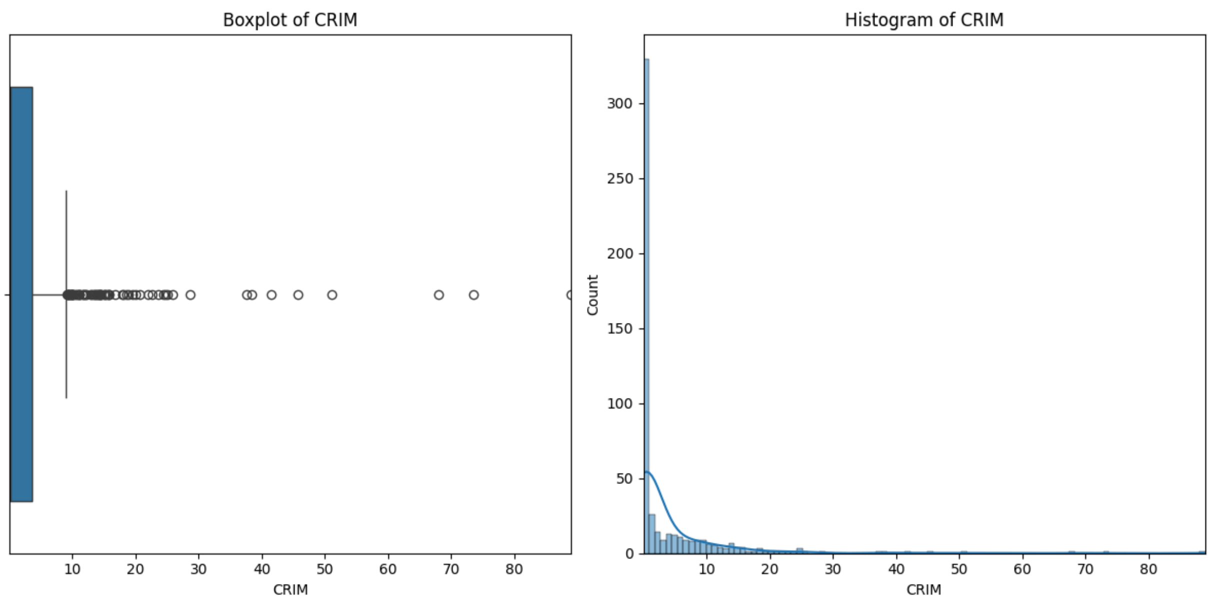
Exploratory Data Analysis (EDA)

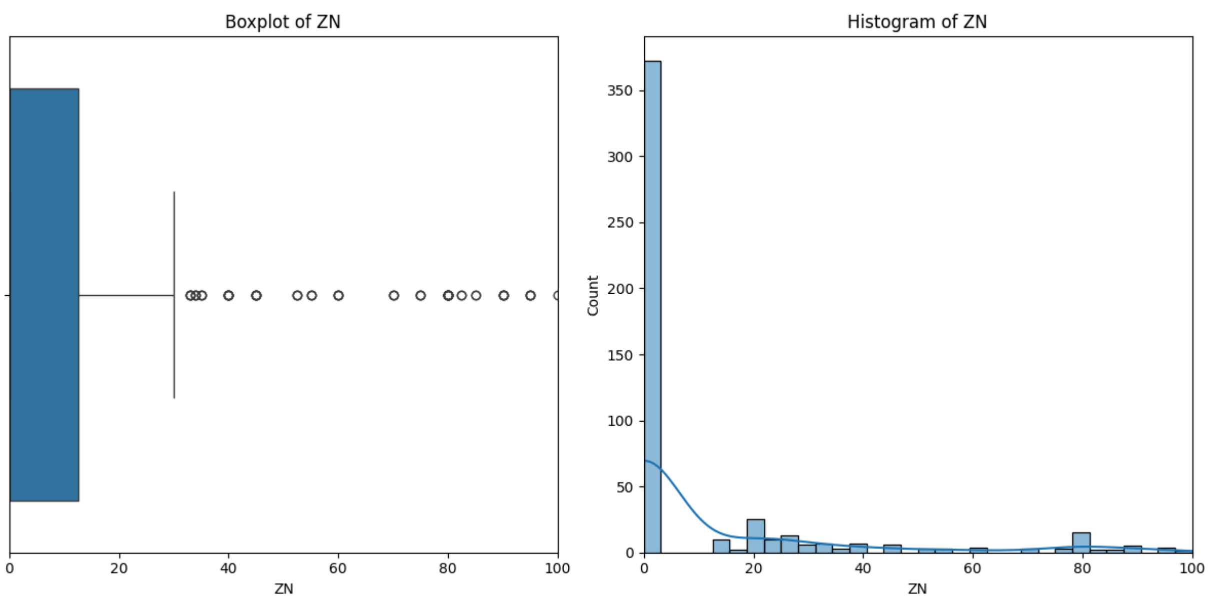
In [9]:

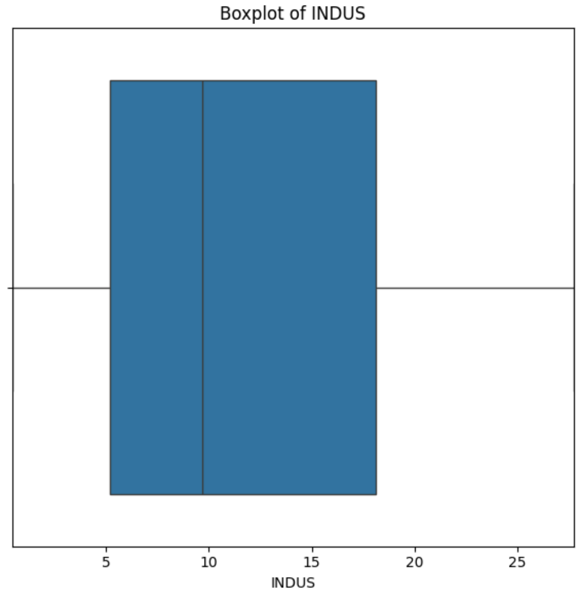
```
# Function to plot a boxplot and a histogram along the same scale
def plot_box_hist(data, feature):
    """
    Plots a boxplot and a histogram of the given feature along the same scal

    Parameters:
    data (DataFrame): The dataset containing the features.
    feature (str): The name of the feature to plot.
    """
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

    # Boxplot
    sns.boxplot(data=data, x=feature, ax=axes[0])
    axes[0].set_title(f'Boxplot of {feature}')


    # Histogram
    sns.histplot(data=data, x=feature, kde=True, ax=axes[1])
```

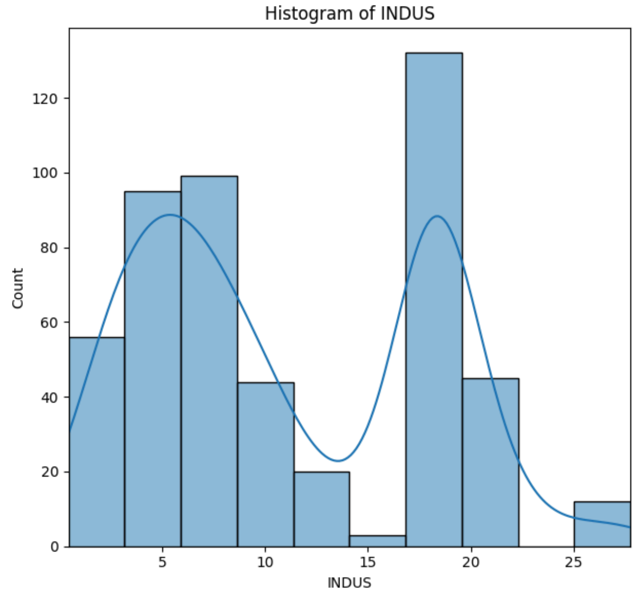
```
    axes[1].set_title(f'Histogram of {feature}')

    # Set the same x-axis limits for both plots
    xmin = min(data[feature])
    xmax = max(data[feature])
    axes[0].set_xlim([xmin, xmax])
    axes[1].set_xlim([xmin, xmax])

    plt.tight_layout()
    plt.show()
```

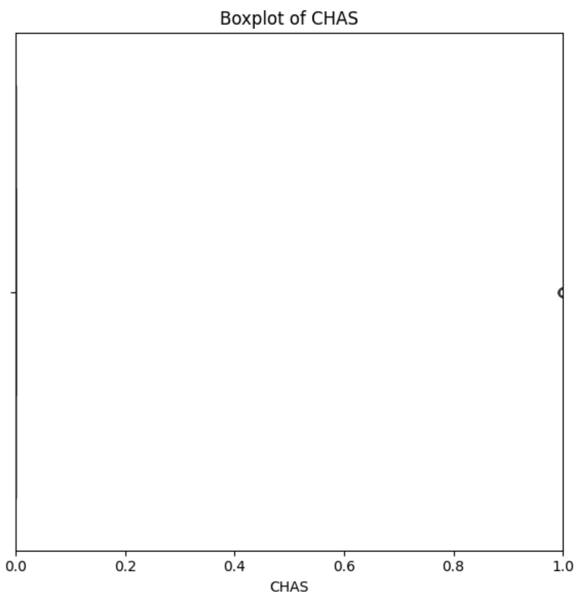
In [10]:
```
# Plot all the distributions of the variables
for feature in data.columns:
  plot_box_hist(data, feature)
```

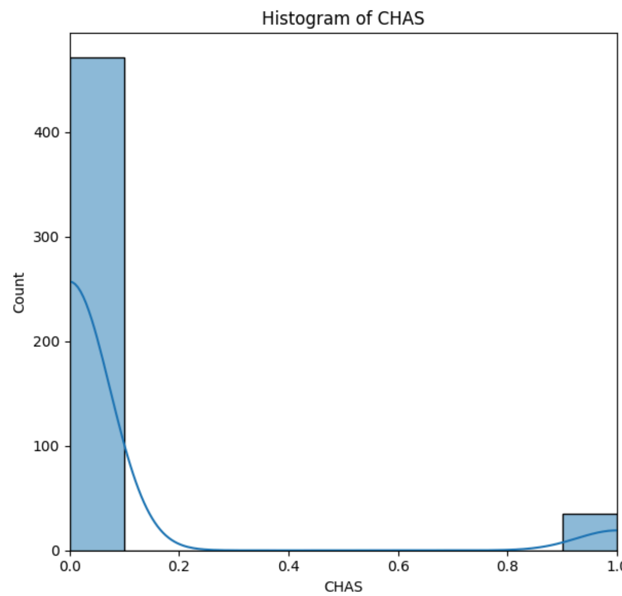


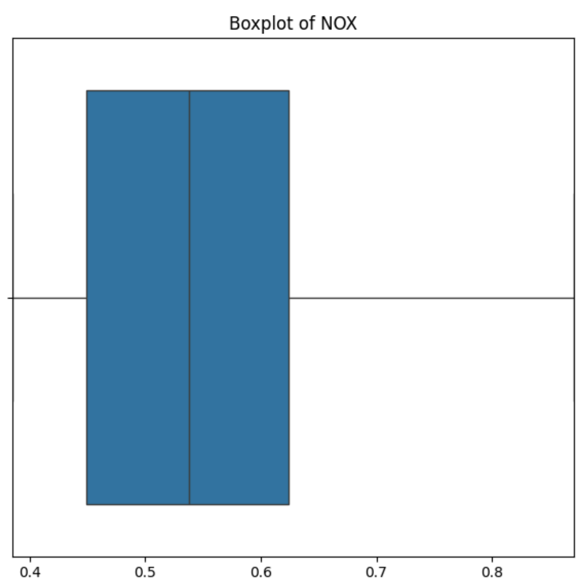


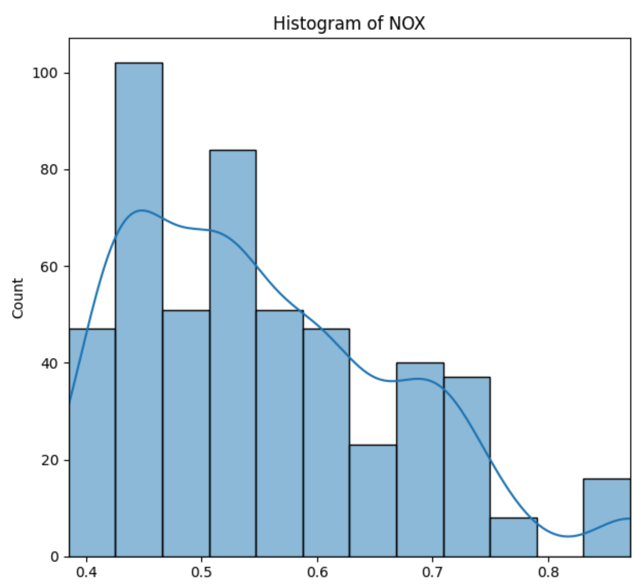

Boxplot of INDUS
5
10
15
20
25
INDUS

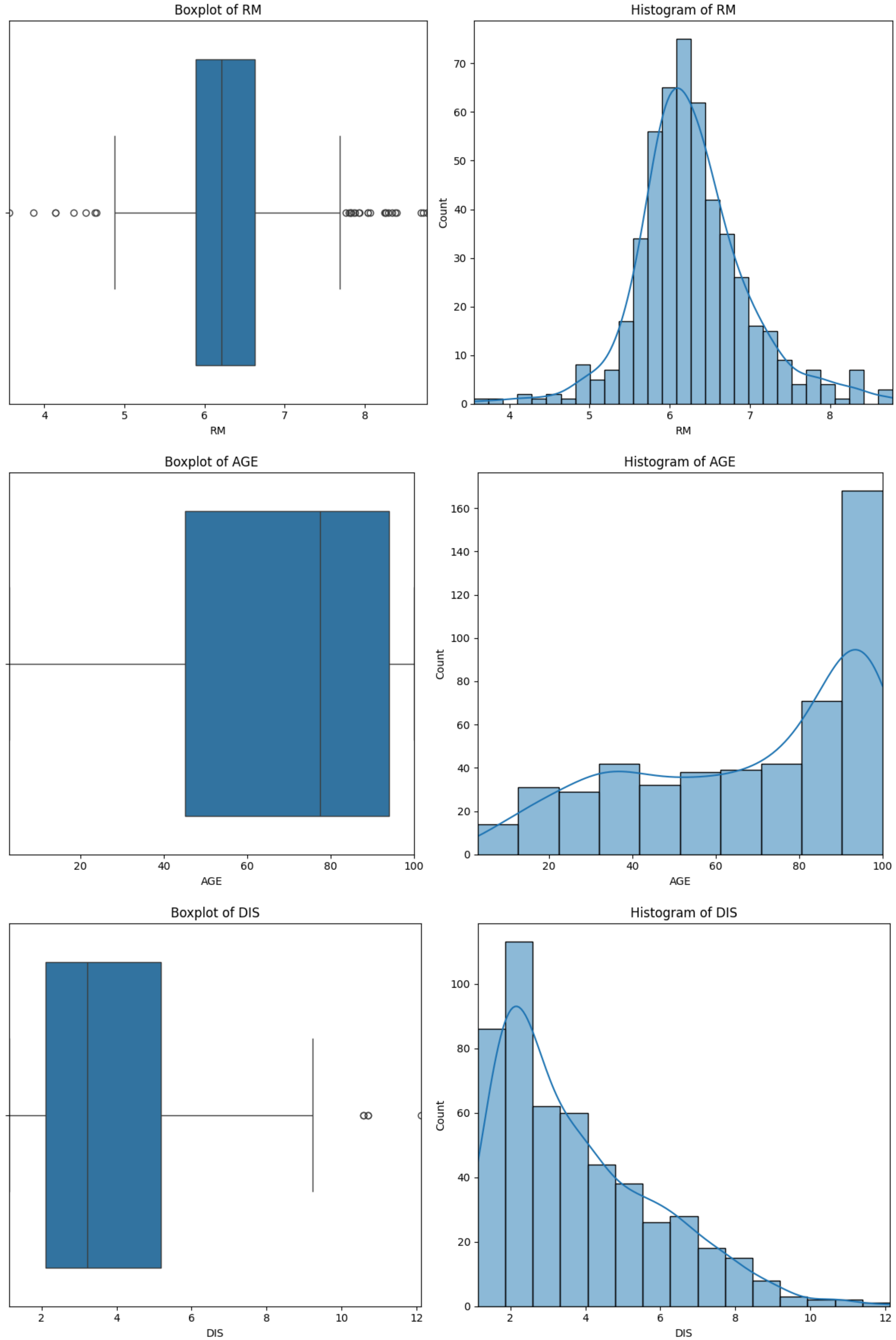

Histogram of INDUS
Count
0
20
40
60
80
100
120
5
10
15
20
25
INDUS

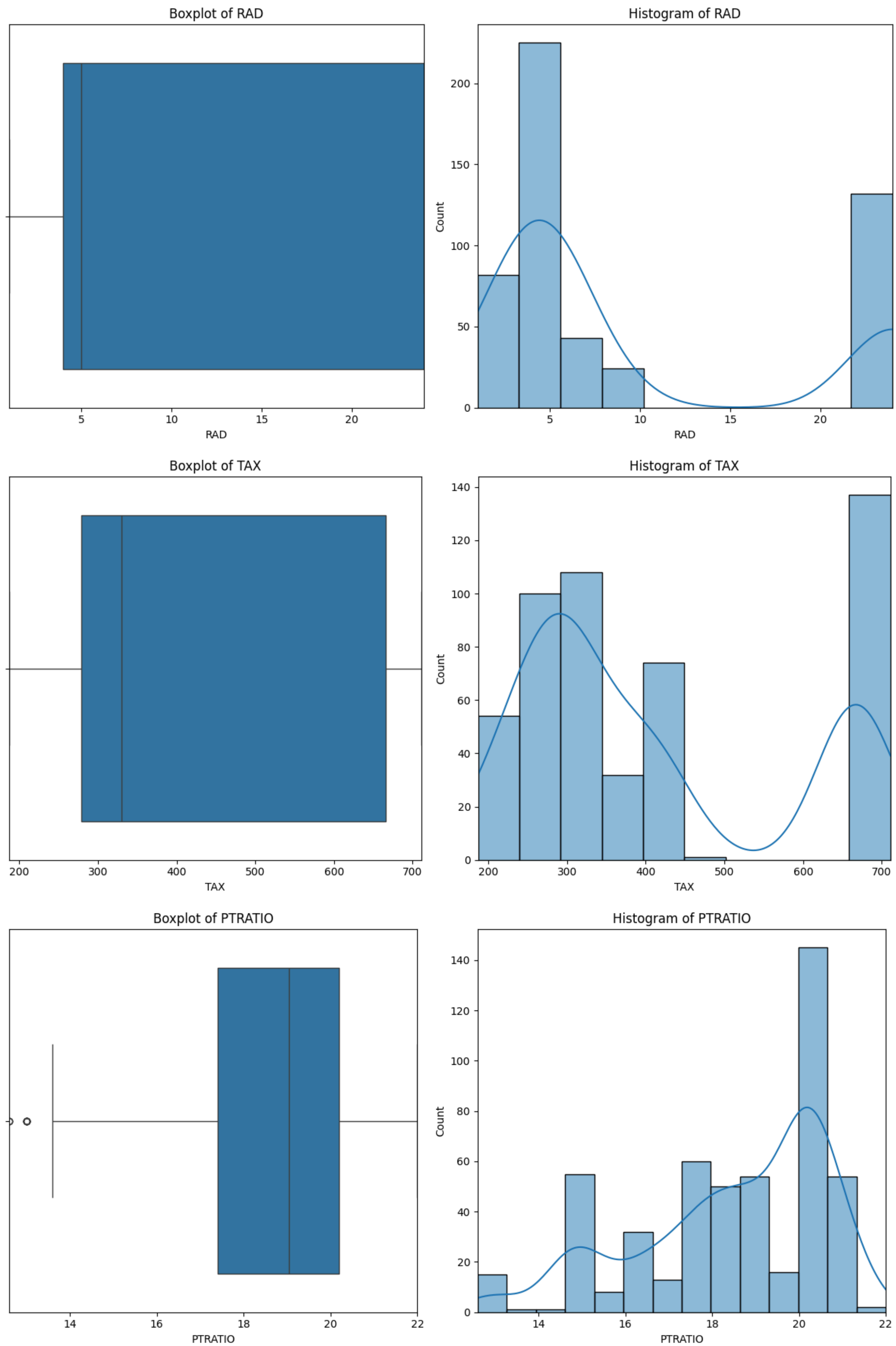

Boxplot of CHAS
0.0
0.2
0.4
0.6
0.8
1.0
CHAS

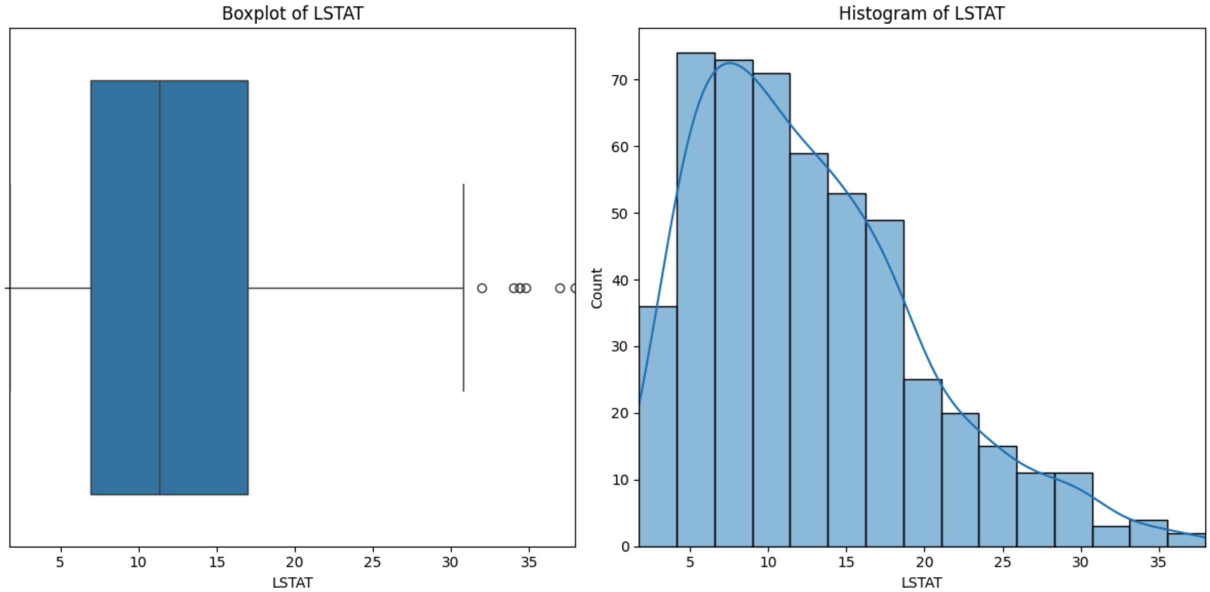

Histogram of CHAS
Count
0
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0
CHAS

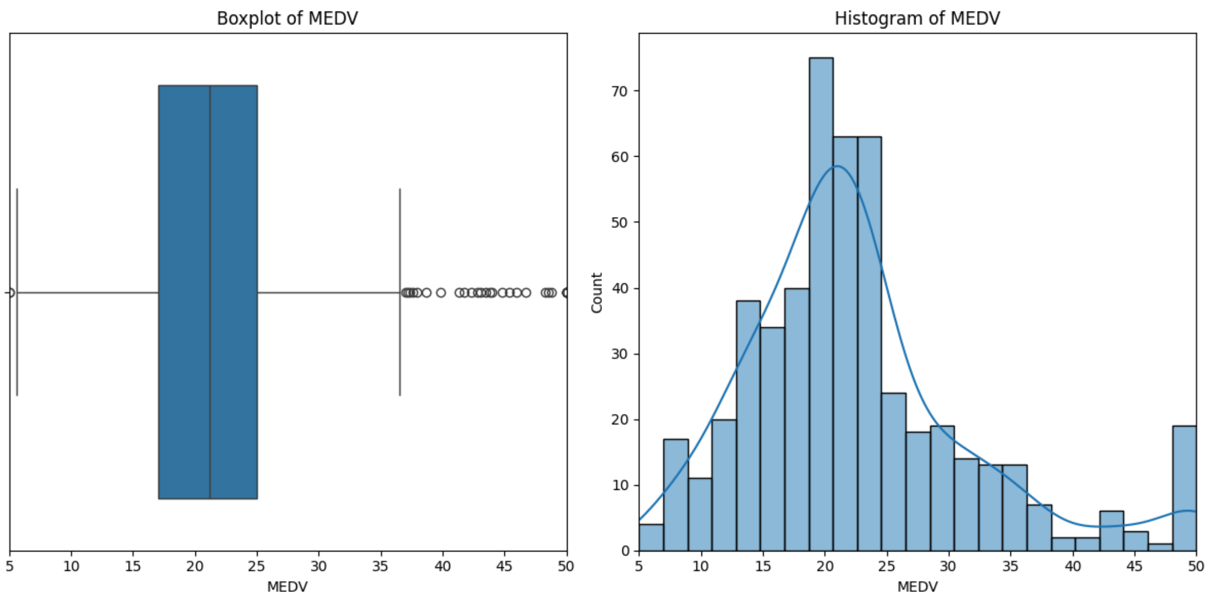

Boxplot of NOX
0.4
0.5
0.6
0.7
0.8
NOX


Histogram of NOX
Count
0
20
40
60
80
100
0.4
0.5
0.6
0.7
0.8
NOX


Boxplot of RM
RM
Histogram of RM
Count
RM
Boxplot of AGE
AGE
Histogram of AGE
Count
AGE
Boxplot of DIS
DIS
Histogram of DIS
Count
DIS


Boxplot of RAD
5
10
15
20
RAD
Histogram of RAD
Count
0
50
100
150
200
5
10
15
20
RAD
Boxplot of TAX
200
300
400
500
600
700
TAX
Histogram of TAX
Count
0
20
40
60
80
100
120
140
200
300
400
500
600
700
TAX
Boxplot of PTRATIO
14
16
18
20
22
PTRATIO
Histogram of PTRATIO
Count
0
20
40
60
80
100
120
140
14
16
18
20
22
PTRATIO





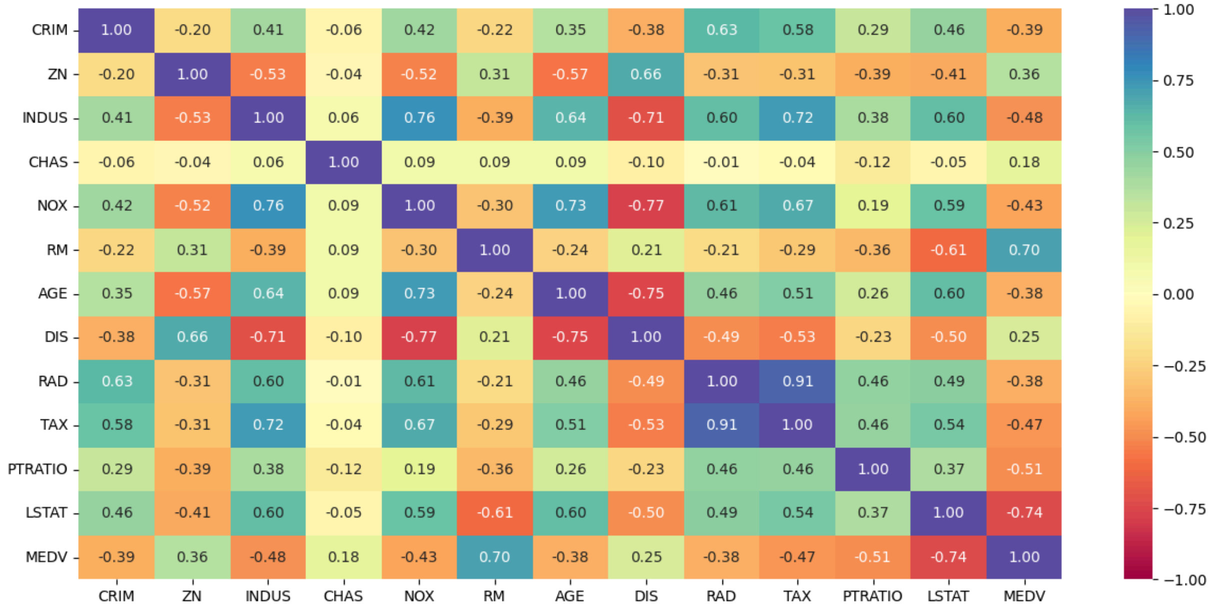
Observations:

- Almost all the variables don't have a normal distribution, they are also very skewed.
- In the case of the variable CHAS, there is no distribution since it is binary, however, most houses are not close to the Charles River.
- TAX distribution has a distinct drop after 500 and then another spike at 700. Perhaps this is due to a correlation with another variable. Something similar happens with the variable RAD.
- The dependent variable, MEDV, is slightly skewed to the right, so computing a log transform might be useful to achieve normality.

Bivariate analysis

In [11]:
```
# Finding the correlation between various columns of the dataset
plt.figure(figsize = (15,7))
sns.heatmap(data.corr(), annot = True, vmin = -1, vmax = 1, fmt = ".2f", cma
```

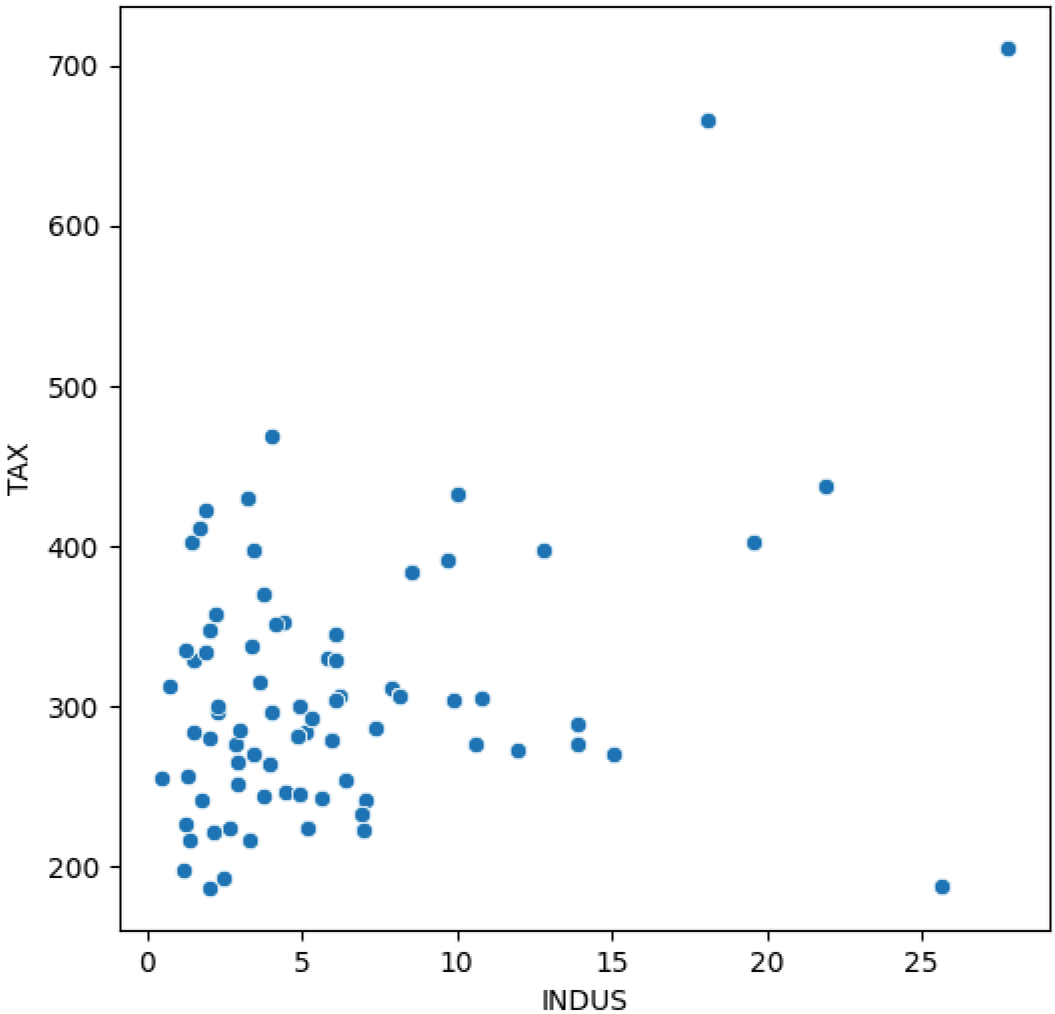
Out[11]: <Axes: >



Observations:

- There is significant correlation between some variables:
- TAX and INDUS of 0.72
- NOX and INDUS of 0.76
- DIS and INDUS of -0.71
- AGE and NOX of 0.73
- DIS and NOX of -0.77
- MEDV and RM of 0.7
- DIS and AGE of -0.75
- TAX and RAD of 0.91
- LSTAT and MEDV of -0.74
- The dependent variable only has a correlation of more than |0.7| with two variables, RM and LSTAT

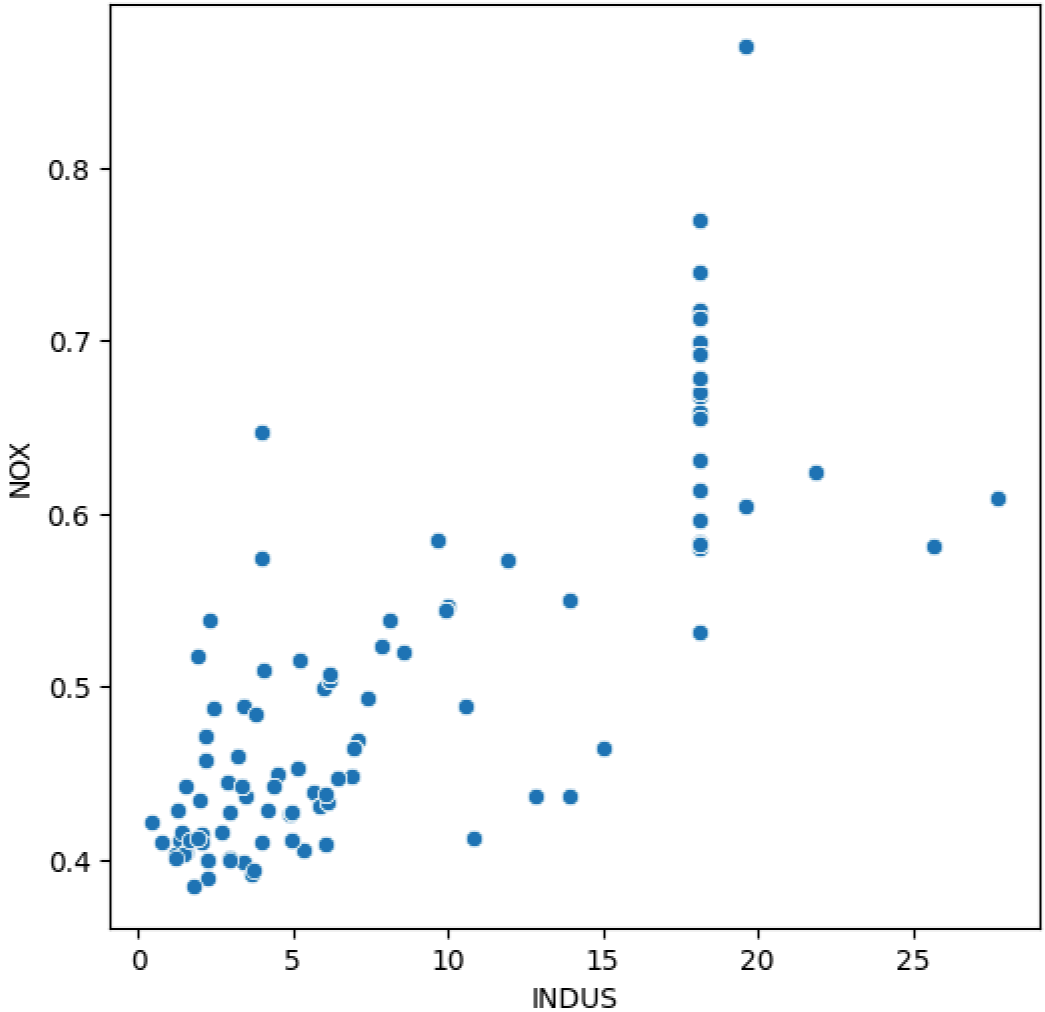
In [12]:
```
# Scatterplot for INDUS and TAX
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'INDUS', y = 'TAX', data = data)
plt.show()
```



Observations:

- As INDUS increases, TAX increases, this can be due that there is also correlation between these two variables and NOX, which is the Nitric Oxide concentration, so it makes sense that when the non-retail businesses increase, contamination increases and so does taxes.

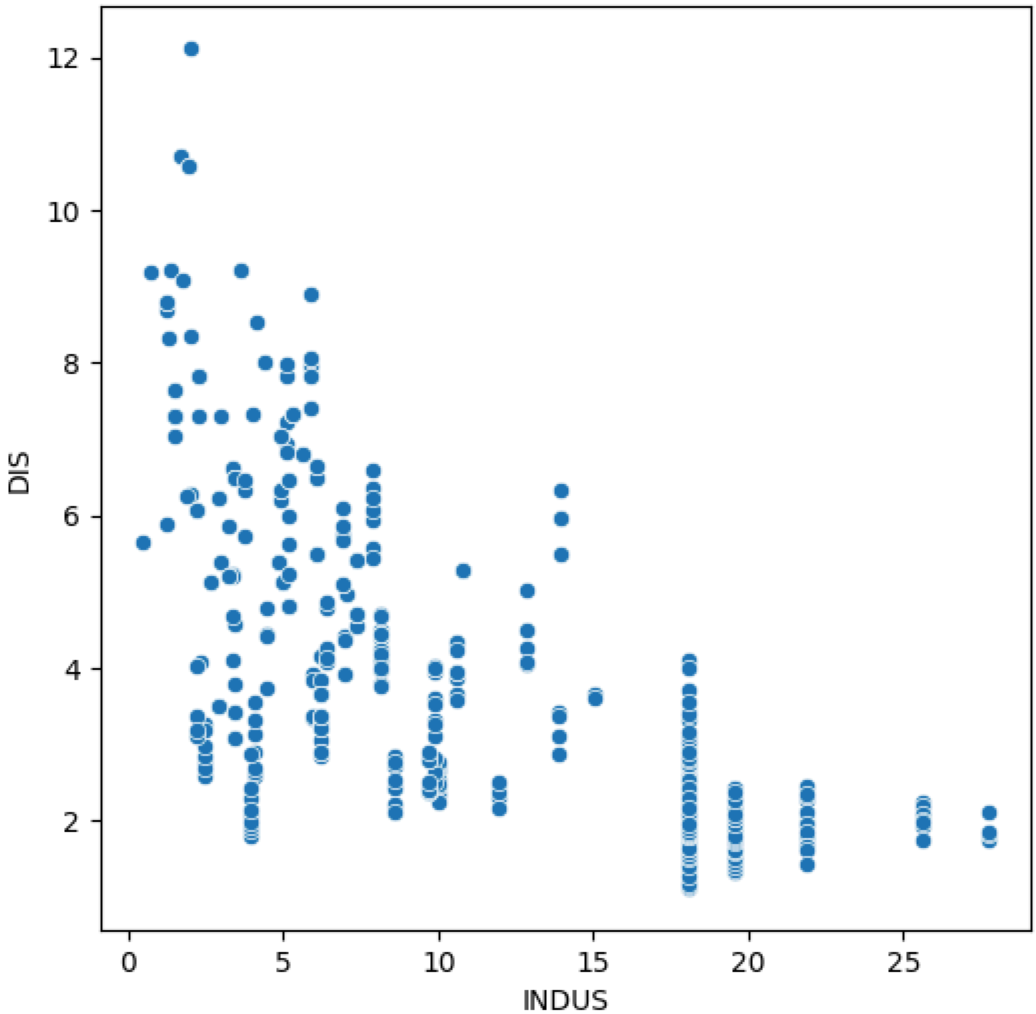
In [13]:
```
# Scatterplot for INDUS and NOX
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'INDUS', y = 'NOX', data = data)
plt.show()
```



Observations:

- As industrial activities increase, the amount of air pollution due to Nitric Oxide increases

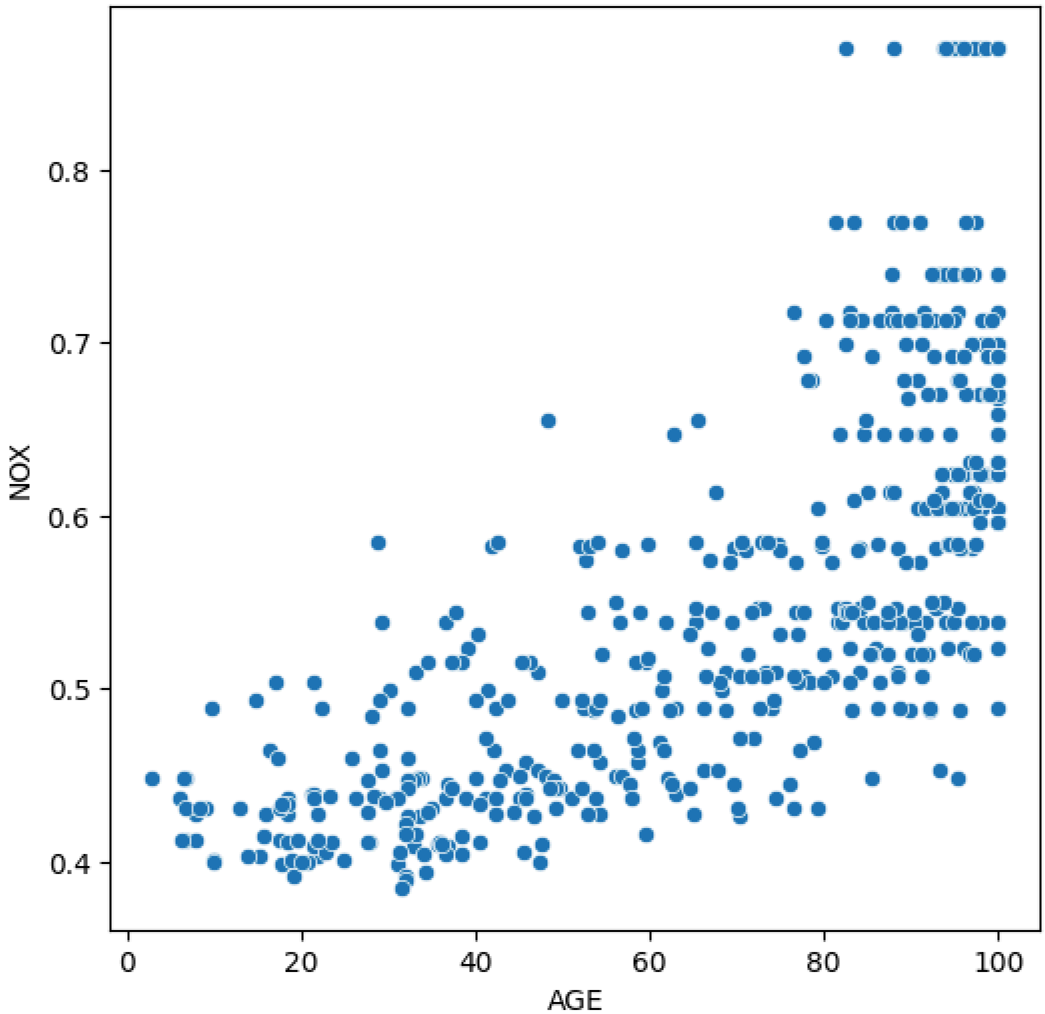
In [14]:
```
# Scatterplot for DIS and INDUS
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'INDUS', y = 'DIS', data = data)
plt.show()
```



Observations:

- When the proportion of non-retail businesses increase, the distance between employment centers and towns decrease

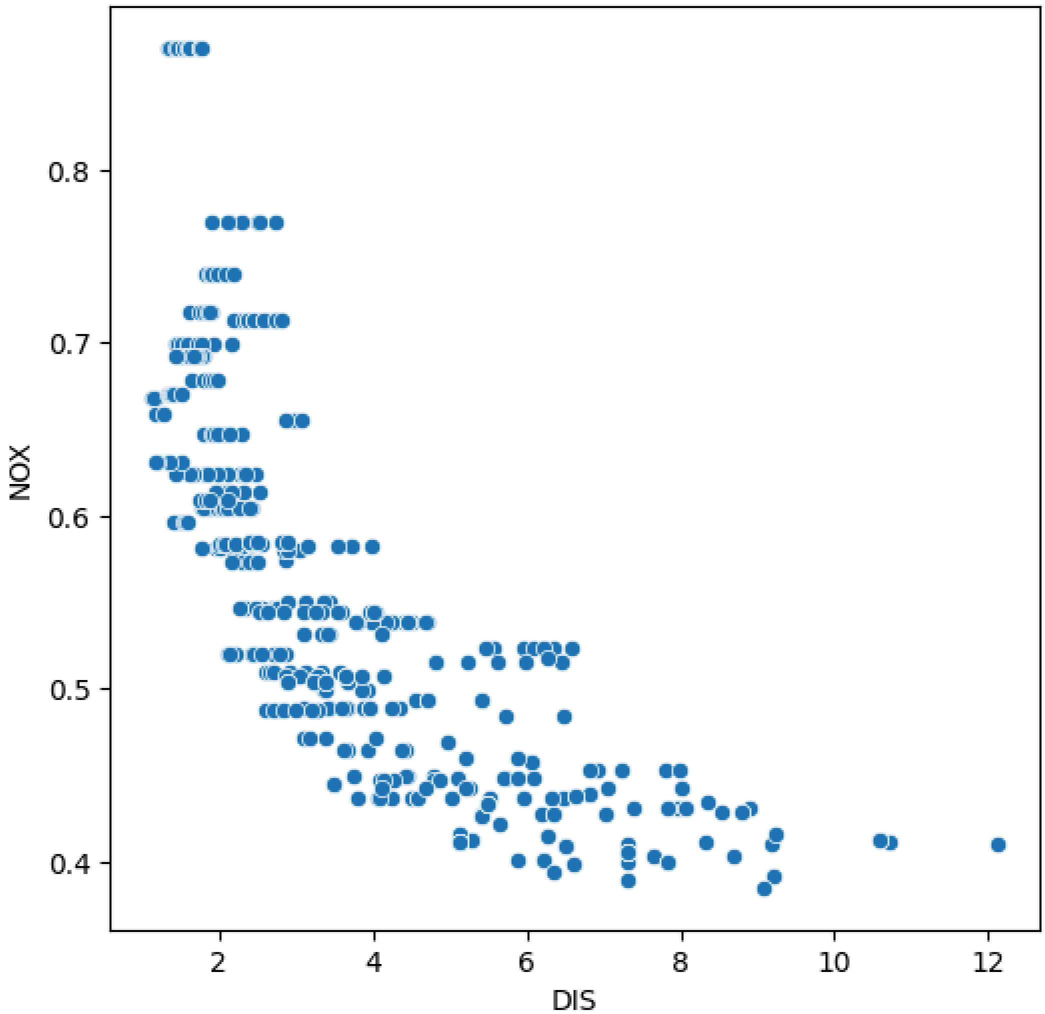
In [15]:
```
# Scatterplot for AGE and NOX
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'AGE', y = 'NOX', data = data)
plt.show()
```



Observations:

- As age increases, Nitric Oxide increases, this makes little sense by its own, however, these two variables have significant correlation with DIS, so this would explain it. Older buildings are closer to industrial sites, so, the NOX variable is higher.

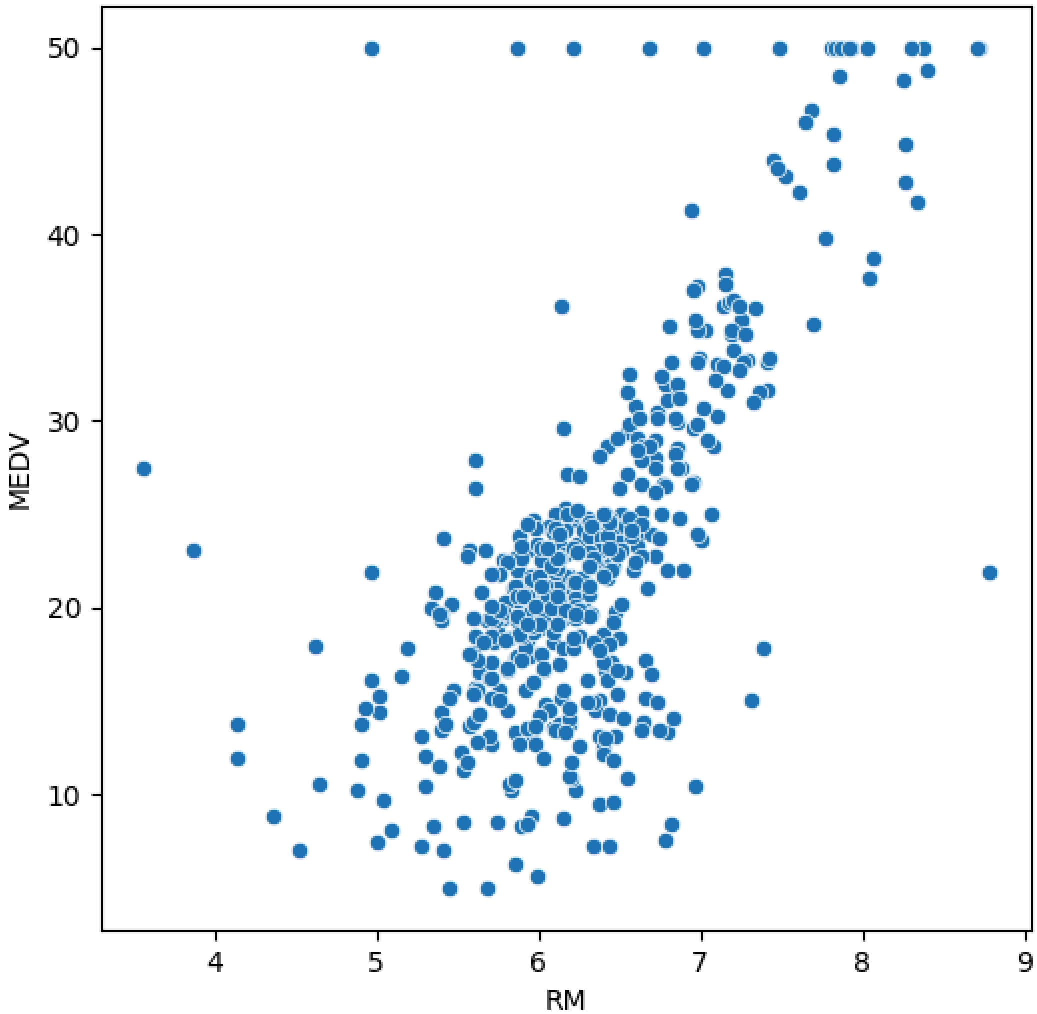
In [16]:
```
# Scatterplot for DIS and NOX
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'DIS', y = 'NOX', data = data)
plt.show()
```



Observations:

- The lower the distance between a town and industrial sites, the higher the air pollution.

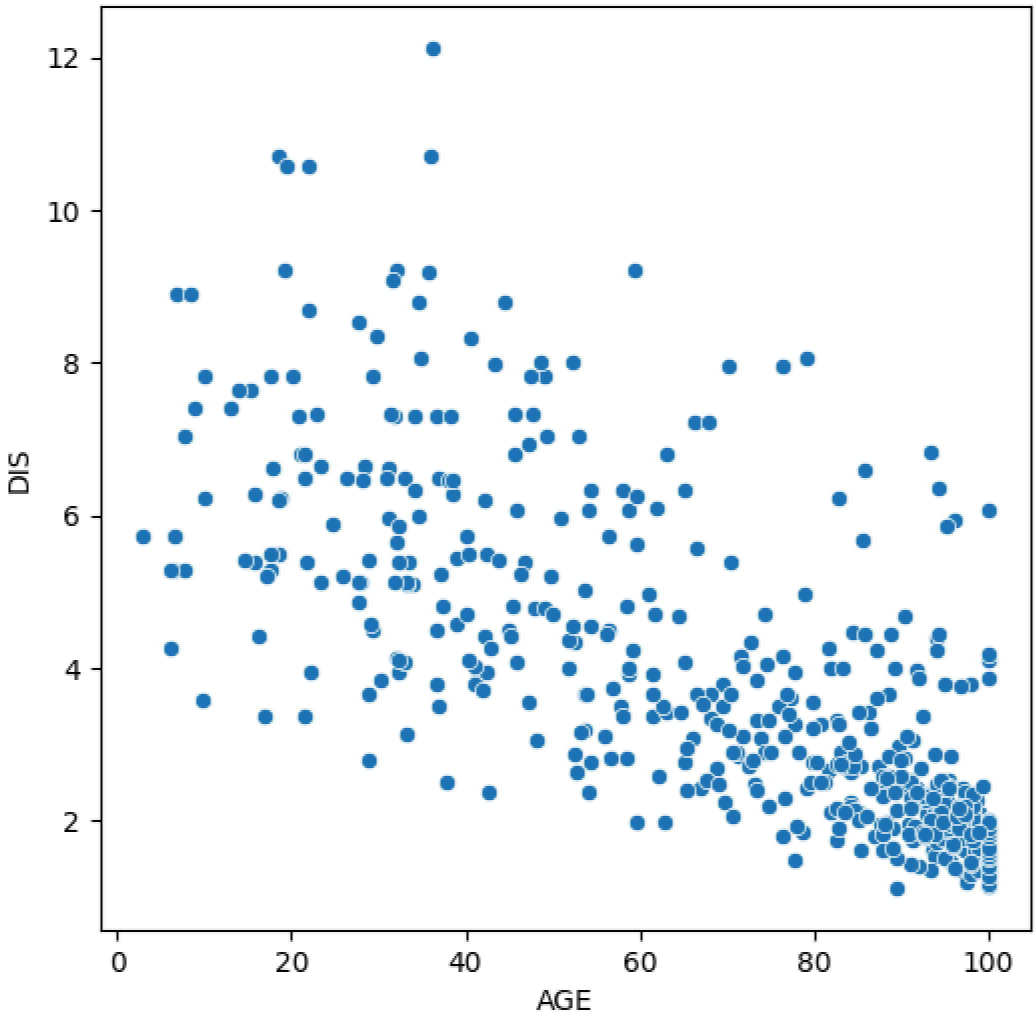
In [17]:
```
# Scatterplot for MEDV and RM
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'RM', y = 'MEDV', data = data)
plt.show()
```



Observations:

- This positive correlation makes sense, the more rooms per dwelling, the higher the price.

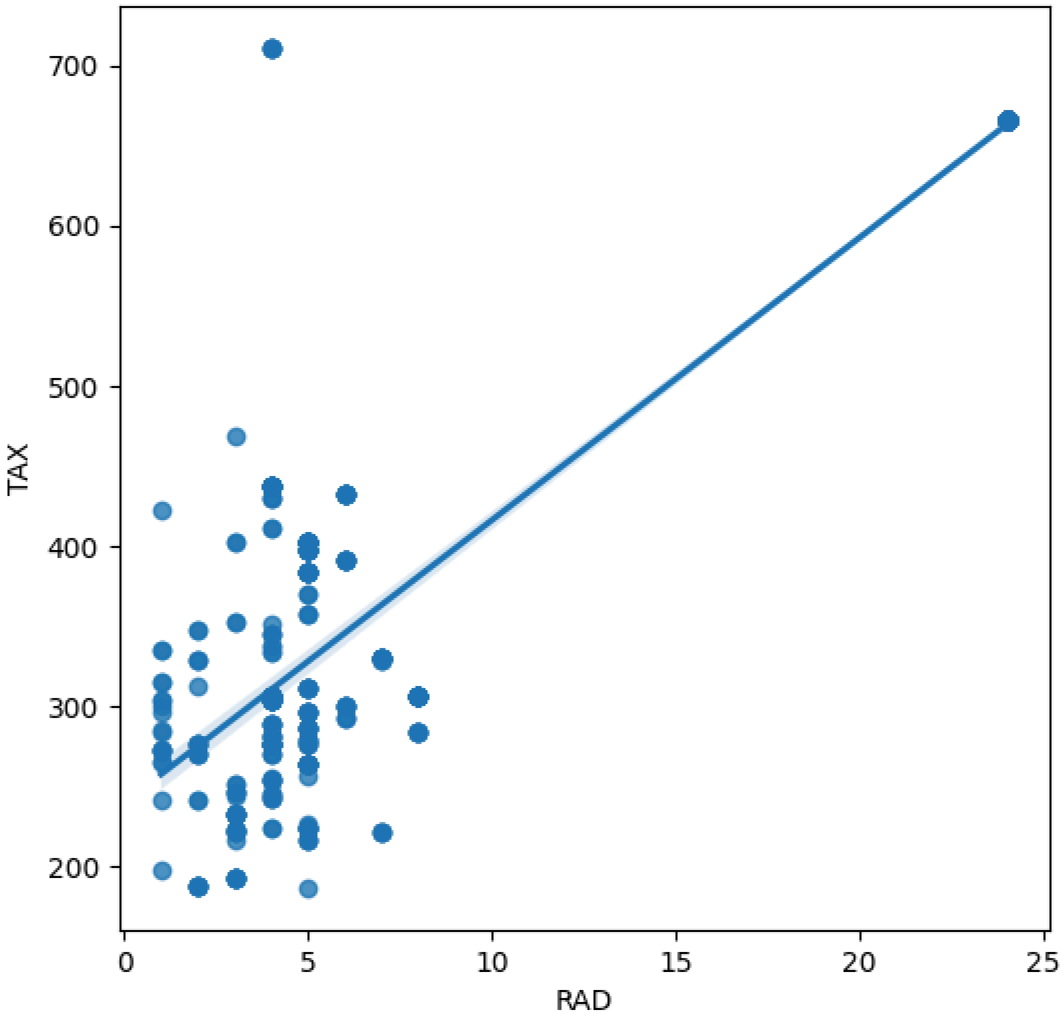
In [18]:
```
# Scatterplot for DIS and AGE
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'AGE', y = 'DIS', data = data)
plt.show()
```



Observations:

- Towns with older buildings are closer to employment centers.

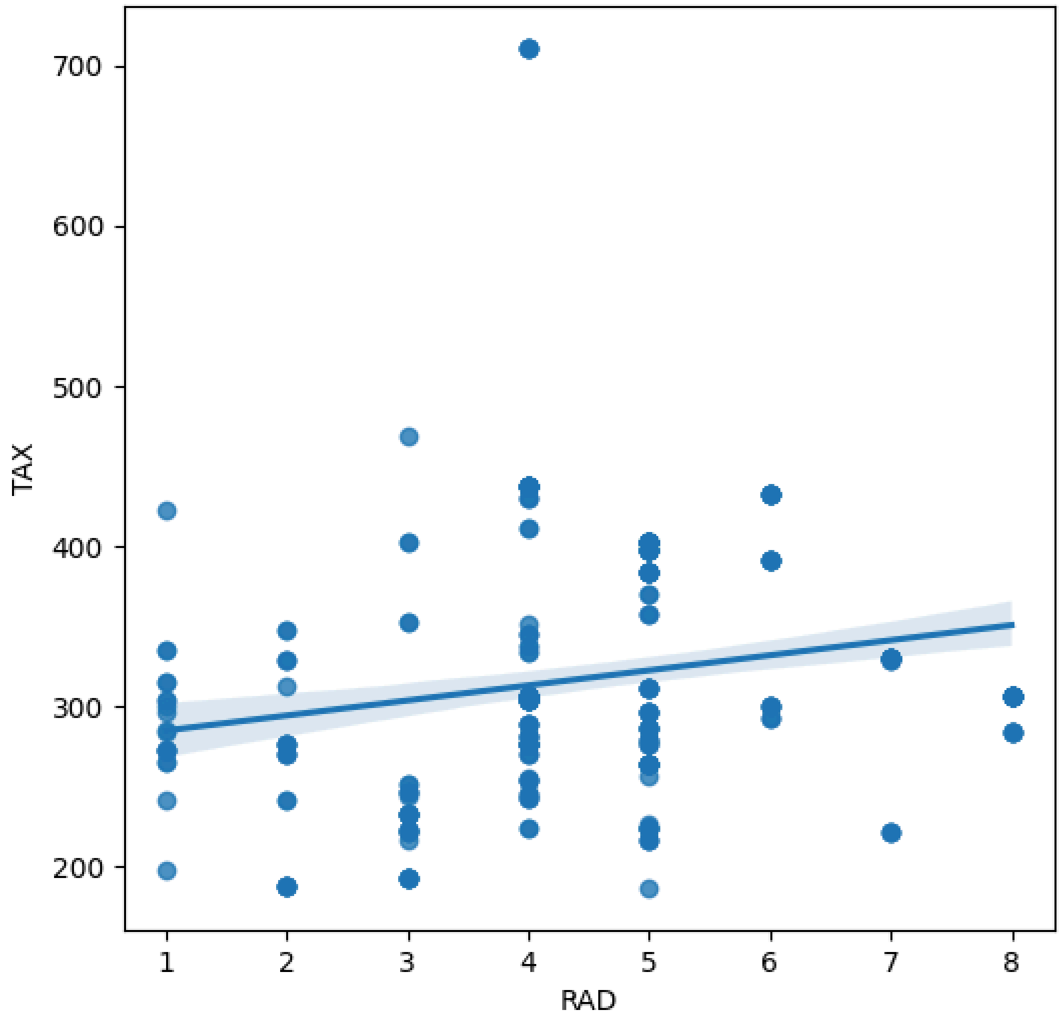
In [19]:
```
# Scatterplot for TAX and RAD
plt.figure(figsize = (6, 6))
sns.regplot(x = 'RAD', y = 'TAX', data = data)
plt.show()
```



Observations:

- This plot does not follow any trend, perhaps the correlation is due to the outliers

In [20]:
```
test_df = data[data['RAD'] < 10]
plt.figure(figsize = (6, 6))
sns.regplot(x = 'RAD', y = 'TAX', data = test_df)
plt.show()
```



In [21]:
```
test_df.corr()
```

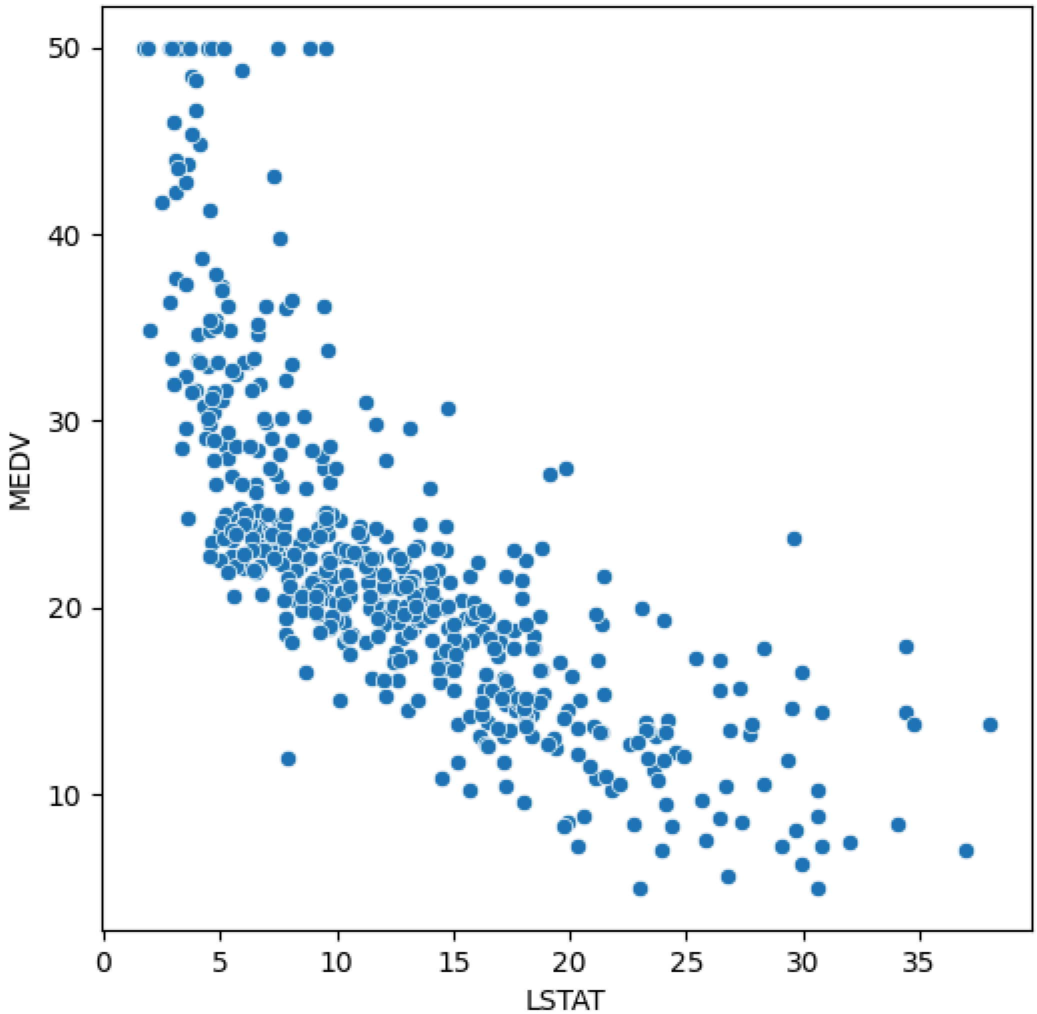
Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AG
CRIM	1.000000	-0.288726	0.500636	0.135539	0.746568	-0.212319	0.46838
ZN	-0.288726	1.000000	-0.470373	-0.057796	-0.475555	0.313877	-0.53152
INDUS	0.500636	-0.470373	1.000000	0.107334	0.666886	-0.391950	0.54295
CHAS	0.135539	-0.057796	0.107334	1.000000	0.121360	0.047722	0.12270
NOX	0.746568	-0.475555	0.666886	0.121360	1.000000	-0.274146	0.67178
RM	-0.212319	0.313877	-0.391950	0.047722	-0.274146	1.000000	-0.17884
AGE	0.468386	-0.531523	0.542954	0.122706	0.671789	-0.178846	1.00000
DIS	-0.446093	0.635556	-0.602261	-0.136626	-0.716061	0.106284	-0.67870
RAD	0.145992	-0.168458	-0.004461	0.087884	0.125672	0.070815	0.09600
TAX	0.295886	-0.128271	0.518329	-0.047599	0.380057	-0.262646	0.27430
PTRATIO	-0.237853	-0.300973	0.134352	-0.145399	-0.152672	-0.344239	0.06143
LSTAT	0.384030	-0.393447	0.534288	0.053330	0.498434	-0.685705	0.54985
MEDV	-0.171697	0.326101	-0.389719	0.095457	-0.263013	0.890823	-0.26820

Observations:

- If outliers are taken out, the correlation between these two variables is of ~0.19, so it is not of significance.

In [22]:
```
# Scatterplot for LSTAT and MEDV
plt.figure(figsize = (6, 6))
sns.scatterplot(x = 'LSTAT', y = 'MEDV', data = data)
plt.show()
```



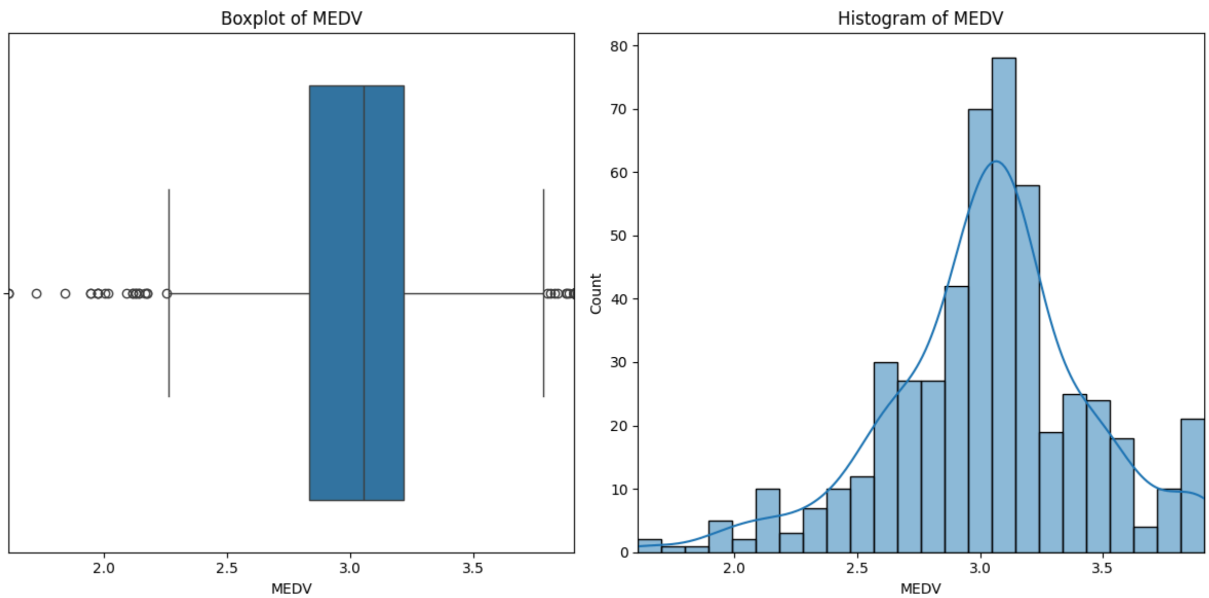
Observations:

- As the amount of people with low socioeconomic status increases, house prices drop. This makes sense since most people would not be able to afford housing if princes were higher.

Data Preprocessing

In [23]:
```
# Apply log transformation to the dependent variable
data['MEDV'] = np.log(data['MEDV'])
```

In [24]:
```
plot_box_hist(data, 'MEDV')
```



Observation:

- With the log transformation, the dependent variable seems a little more normally distributed and less skewed.
- There are no null values to be treated.
- The outliers don't need to be treated since they are proper values.

Model Building - Linear Regression

In [25]:

```
# Take dependent and independent variables
y = data['MEDV']
x = data.drop(columns = 'MEDV')

# Add constant
x = sm.add_constant(x)
```

In [26]:

```
# Split the data 15% for testing, 85% for training

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.15,
```

In [27]:

```
# Check for multicollinearity

# 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.
    ]
    return vif
```

```
print(checking_vif(x_train))
```

```
    feature         VIF
0     const  535.200091
1      CRIM    1.739110
2        ZN    2.504154
3     INDUS    3.901991
4      CHAS    1.069708
5       NOX    4.378652
6        RM    1.927099
7       AGE    3.256620
8       DIS    4.152061
9       RAD    7.639489
10      TAX    9.318876
11  PTRATIO    1.816933
12    LSTAT    2.980971
```

In [28]:
```
# Remove TAX since it has the highest collinearity
x_train.drop(columns = 'TAX', inplace = True)
x_test.drop(columns = 'TAX', inplace = True)
```

In [29]:
```
x_train.head()
```

Out[29]:

	const	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	PTRATIO
370	1.0	6.53876	0.0	18.10	1	0.631	7.016	97.5	1.2024	24	20.2
285	1.0	0.01096	55.0	2.25	0	0.389	6.453	31.9	7.3073	1	15.3
159	1.0	1.42502	0.0	19.58	0	0.871	6.510	100.0	1.7659	5	14.7
291	1.0	0.07886	80.0	4.95	0	0.411	7.148	27.7	5.1167	4	19.2
128	1.0	0.32543	0.0	21.89	0	0.624	6.431	98.8	1.8125	4	21.2

In [30]:
```
print(checking_vif(x_train))
```

```
    feature         VIF
0     const  529.939355
1      CRIM    1.738776
2        ZN    2.338177
3     INDUS    3.175019
4      CHAS    1.052718
5       NOX    4.352911
6        RM    1.918919
7       AGE    3.250590
8       DIS    4.150677
9       RAD    2.702143
10  PTRATIO    1.799632
11    LSTAT    2.978939
```

Now all of the variables have a VIF of less than 5. There is no longer multicollinearity between variables.

In [31]:
```
# Model Performance on test and train data
def model_performance(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": [
                    np.sqrt(mean_squared_error(y_observed_train, y_pred_trai
                    np.sqrt(mean_squared_error(y_observed_test,y_pred_test))
                ],
                "MAE": [
                    mean_absolute_error(y_observed_train, y_pred_train),
                    mean_absolute_error(y_observed_test,y_pred_test),
                ],
                "MAPE": [
                    mean_absolute_percentage_error(y_observed_train, y_pred_
                    mean_absolute_percentage_error(y_observed_test,y_pred_te
                ],

                'r2': [
                    r2_score(y_observed_train, y_pred_train),
                    r2_score(y_observed_test,y_pred_test),
                ],
            }
        )
    )
```

In [32]:
```
# Create model
model_1 = sm.OLS(y_train, x_train).fit()

# Check model summary
model_1.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.780
Model:	OLS	**Adj. R-squared:**	0.774
Method:	Least Squares	**F-statistic:**	134.4
Date:	Mon, 27 May 2024	**Prob (F-statistic):**	1.04e-129
Time:	20:16:10	**Log-Likelihood:**	103.59
No. Observations:	430	**AIC:**	-183.2
Df Residuals:	418	**BIC:**	-134.4
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.2916	0.214	20.043	0.000	3.871	4.712
CRIM	-0.0110	0.001	-8.144	0.000	-0.014	-0.008
ZN	0.0010	0.001	1.581	0.115	-0.000	0.002
INDUS	-0.0024	0.002	-0.988	0.324	-0.007	0.002
CHAS	0.1210	0.036	3.374	0.001	0.050	0.191
NOX	-0.9178	0.166	-5.524	0.000	-1.244	-0.591
RM	0.0814	0.019	4.365	0.000	0.045	0.118
AGE	0.0005	0.001	0.881	0.379	-0.001	0.002
DIS	-0.0502	0.009	-5.617	0.000	-0.068	-0.033
RAD	0.0052	0.002	2.936	0.004	0.002	0.009
PTRATIO	-0.0392	0.006	-6.876	0.000	-0.050	-0.028
LSTAT	-0.0300	0.002	-13.317	0.000	-0.034	-0.026

Omnibus:	39.523	**Durbin-Watson:**	1.976
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	130.953
Skew:	0.349	**Prob(JB):**	3.66e-29
Kurtosis:	5.612	**Cond. No.**	2.06e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.06e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Model Performance Check

In [33]:
```
# Evaluate model's performance
model_performance(model_1, x_train, x_test)
```

	Data	RMSE	MAE	MAPE	r2
0	Train	0.190167	0.137504	0.047765	0.779549
1	Test	0.212692	0.161786	0.057654	0.750711

The metrics differ between testing and training data, there may be slight overfitting happening. Changing the ratio of training/testing data might be helpful.

In [34]:
```
# Split the data with a new ratio
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.35,

# Drop TAX column, as it has a high VIF value
x_train.drop(columns = 'TAX', inplace = True)
x_test.drop(columns = 'TAX', inplace = True)

# Check that all VIF values are < 5
print(checking_vif(x_train))
```

```
    feature         VIF
0     const  571.184053
1      CRIM    1.895501
2        ZN    2.522618
3     INDUS    3.187149
4      CHAS    1.046378
5       NOX    4.255035
6        RM    2.045091
7       AGE    3.181671
8       DIS    4.333145
9       RAD    2.906020
10  PTRATIO    1.978790
11    LSTAT    3.080552
```

In [35]:
```
# Create model 2
model_2 = sm.OLS(y_train, x_train).fit()

# Check summary
model_2.summary()
```

Out[35]:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.772
Model:	OLS	**Adj. R-squared:**	0.764
Method:	Least Squares	**F-statistic:**	97.04
Date:	Mon, 27 May 2024	**Prob (F-statistic):**	2.42e-94
Time:	20:16:10	**Log-Likelihood:**	66.666
No. Observations:	328	**AIC:**	-109.3
Df Residuals:	316	**BIC:**	-63.82
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.4664	0.265	16.824	0.000	3.944	4.989
CRIM	-0.0129	0.002	-7.331	0.000	-0.016	-0.009
ZN	0.0010	0.001	1.340	0.181	-0.000	0.003
INDUS	-7.105e-05	0.003	-0.025	0.980	-0.006	0.006
CHAS	0.1337	0.041	3.235	0.001	0.052	0.215
NOX	-1.0378	0.193	-5.363	0.000	-1.418	-0.657
RM	0.0711	0.024	3.025	0.003	0.025	0.117
AGE	0.0004	0.001	0.597	0.551	-0.001	0.002
DIS	-0.0479	0.011	-4.418	0.000	-0.069	-0.027
RAD	0.0074	0.002	3.408	0.001	0.003	0.012
PTRATIO	-0.0446	0.007	-6.215	0.000	-0.059	-0.031
LSTAT	-0.0289	0.003	-10.934	0.000	-0.034	-0.024

Omnibus:	31.315	**Durbin-Watson:**	1.867
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	91.283
Skew:	0.388	**Prob(JB):**	1.51e-20
Kurtosis:	5.465	**Cond. No.**	2.14e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.14e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [36]:
```
# Check model's performance
model_performance(model_2, x_train, x_test)
```

	Data	RMSE	MAE	MAPE	r2
0	Train	0.197466	0.143189	0.050054	0.771587
1	Test	0.191158	0.143477	0.049691	0.769924

Model performance may be improved by removing unnecesary variables (p-value > 0.05)

In [37]:
```
# Remove unnecesary variables on both training and testing sets
x_train_2 = x_train.drop(columns = ['INDUS', 'ZN', 'AGE'])
x_test_2 = x_test.drop(columns = ['INDUS', 'ZN', 'AGE'])

# Check VIF values
print(checking_vif(x_train_2))
```

```
   feature         VIF
0    const  564.784511
1     CRIM    1.862987
2     CHAS    1.044680
3      NOX    3.465392
4       RM    1.884616
5      DIS    2.575430
6      RAD    2.802494
7  PTRATIO    1.636511
8    LSTAT    2.603805
```

In [38]:
```
# Build model 3
model_3 = sm.OLS(y_train, x_train_2).fit()

# Check model summary
model_3.summary()
```

Out[38]:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.770
Model:	OLS	**Adj. R-squared:**	0.764
Method:	Least Squares	**F-statistic:**	133.6
Date:	Mon, 27 May 2024	**Prob (F-statistic):**	4.58e-97
Time:	20:16:11	**Log-Likelihood:**	65.633
No. Observations:	328	**AIC:**	-113.3
Df Residuals:	319	**BIC:**	-79.13
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.4696	0.264	16.957	0.000	3.951	4.988
CRIM	-0.0126	0.002	-7.243	0.000	-0.016	-0.009
CHAS	0.1356	0.041	3.288	0.001	0.054	0.217
NOX	-1.0211	0.174	-5.857	0.000	-1.364	-0.678
RM	0.0789	0.023	3.499	0.001	0.035	0.123
DIS	-0.0427	0.008	-5.122	0.000	-0.059	-0.026
RAD	0.0076	0.002	3.570	0.000	0.003	0.012
PTRATIO	-0.0476	0.007	-7.293	0.000	-0.060	-0.035
LSTAT	-0.0281	0.002	-11.598	0.000	-0.033	-0.023

Omnibus:	34.074	**Durbin-Watson:**	1.877
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	98.776
Skew:	0.437	**Prob(JB):**	3.56e-22
Kurtosis:	5.542	**Cond. No.**	706.

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

In [39]:

```
# Check model's performance
model_performance(model_3, x_train_2, x_test_2)
```

```
    Data      RMSE       MAE      MAPE        r2
0  Train  0.198089  0.143639  0.050181  0.770145
1   Test  0.190653  0.144641  0.050073  0.771138
```

This seems to be the best model out of the 3. It is able to explain ~77% of the variance in the target variable.

Checking Linear Regression Assumptions

1. Mean of residuals = 0

In [40]:
```
# Checking mean residuals = 0

residuals = model_3.resid
np.mean(residuals)
```

Out[40]:
```
1.2693098604708345e-15
```

Mean of residuals is very close to 0, so the first condition is satisfied.

2. Homoscedasticity

- Residuals must be symetrically distributed across the regresion line.
- Goldfelquandt test with alpha = 0.05
- Null hypotheses: Residuals are homoscedastic.
- Alternate hypotheses: Residuals are heteroscedastic.

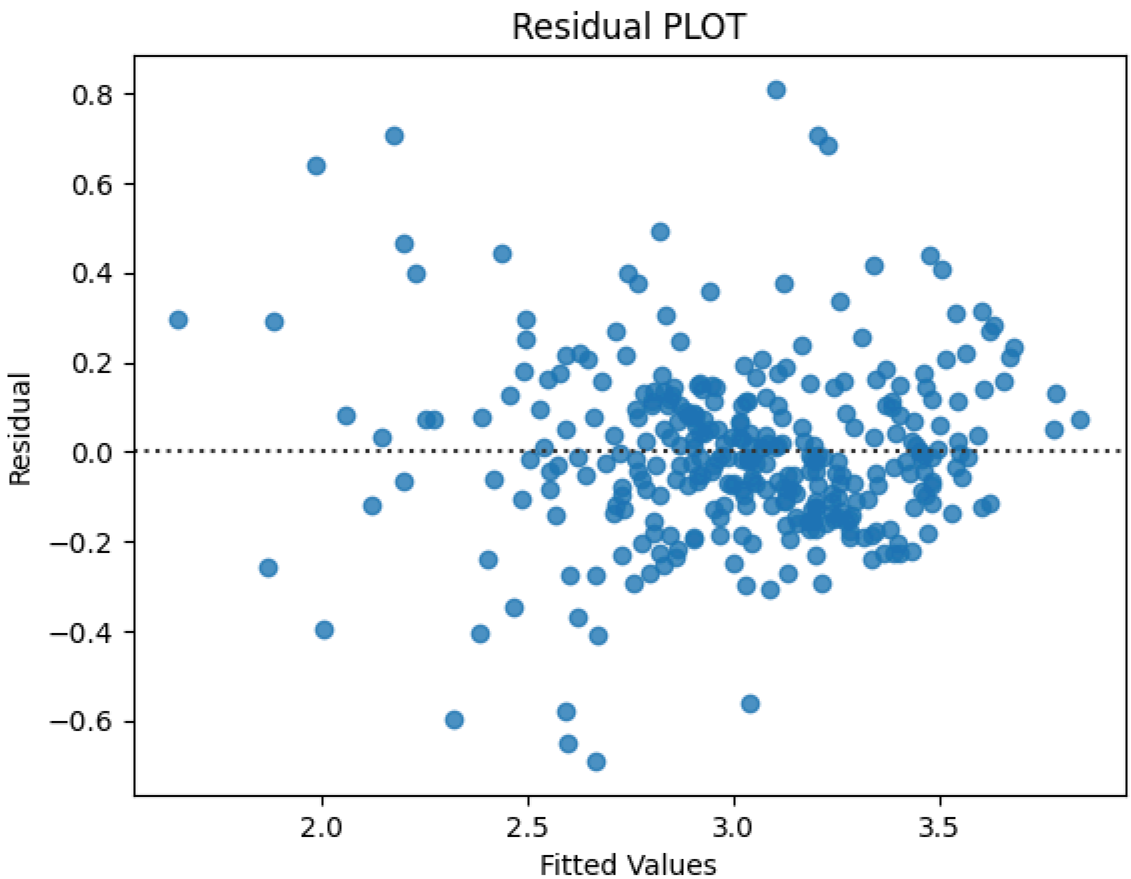
In [41]:
```
# Perform test and display results
name = ["F statistic", "p-value"]

test = sms.het_goldfeldquandt(y_train, x_train_2)

lzip(name, test)
```

Out[41]:
```
[('F statistic', 0.8787596043806704), ('p-value', 0.7890014860698799)]
```

p-value is greater than 0.05 so Null Hypothesis that the residuals are homoscedastic can not be rejected.

3. Linearity of variables

In [42]:
```
# Predicted values
fitted = model_3.fittedvalues

sns.residplot(x = fitted, y = residuals)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
plt.title("Residual PLOT")

plt.show()
```

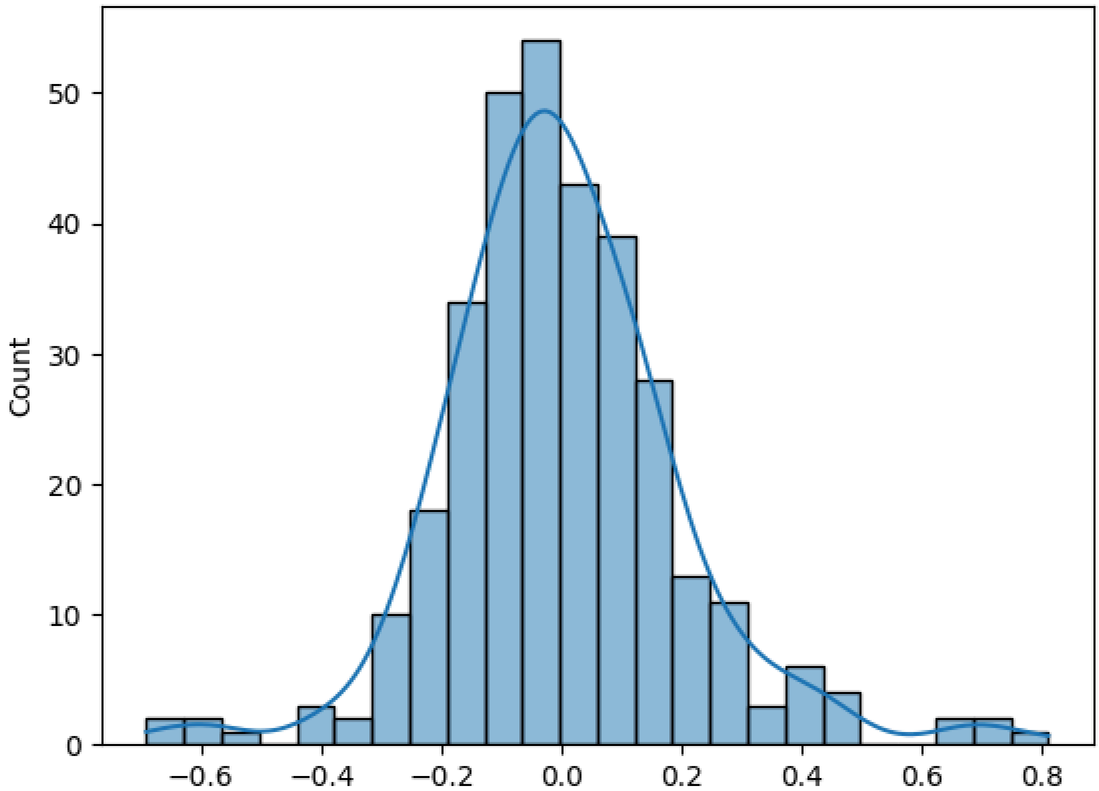


The residuals are randomly and uniformely scattered along the x axis, they do not form any pattern or follow any trend.

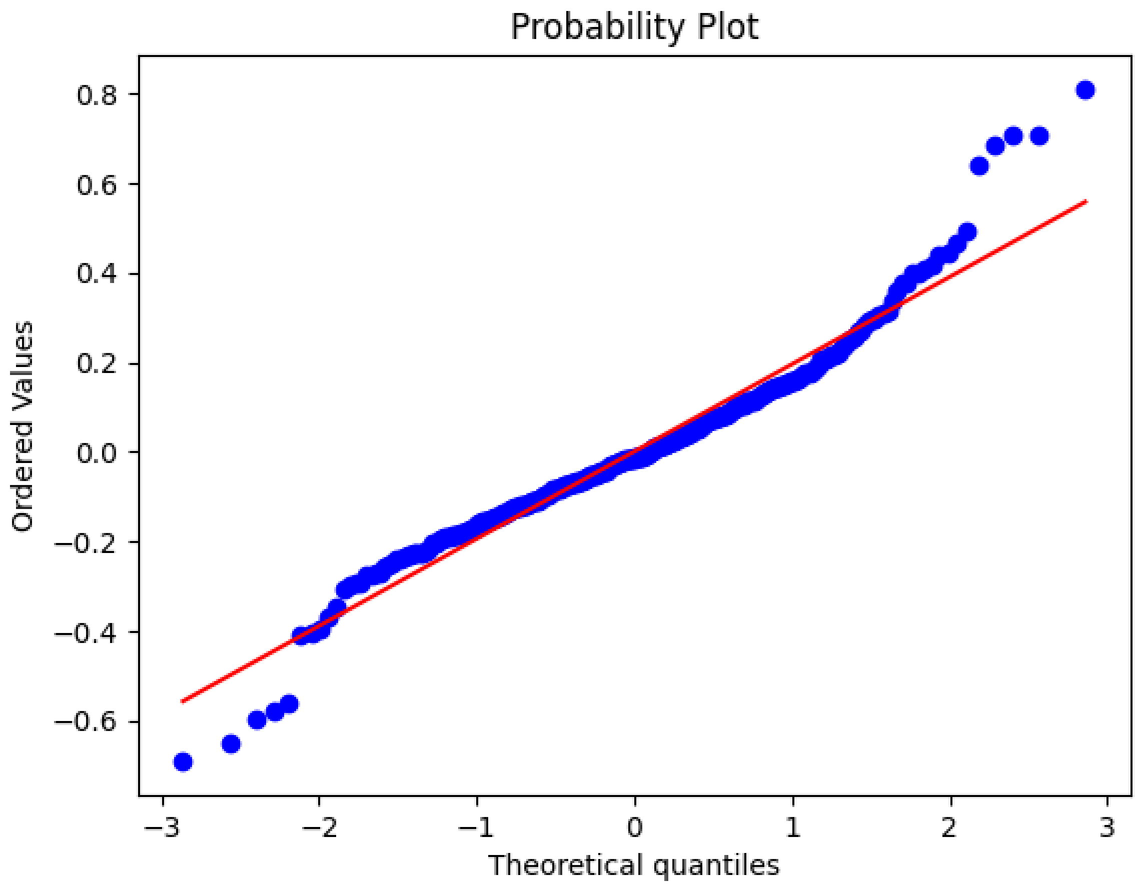
4. Normality of error terms

In [43]:
```
# Plot histogram to see distribution of residuals
sns.histplot(residuals, kde = True)
```

Out[43]: <Axes: ylabel='Count'>



In [44]:
```
# Q-Q plot to confirm normality
stats.probplot(residuals, dist = "norm", plot = pylab)

plt.show()
```



The residuals follow the diagonal, meaning they are normally distributed, also, the histogram looks fairly normal as well.

Cross Validation

In [45]:
```
# Fucntion to perform cross validation
def cross_validate_sm_ols(X, y, k = 10):

    "Perform cross validation, takes the values of the dependent variables a

    kf = KFold(n_splits=k, shuffle=True, random_state=1)
    r2_scores = []
    mape_scores = []

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        model = sm.OLS(y_train, X_train).fit()
        y_pred = model.predict(X_test)

        r2_scores.append(r2_score(y_test, y_pred))
        mape_scores.append(mean_absolute_percentage_error(y_test, y_pred))

    mean_r2 = np.mean(r2_scores)
    std_r2 = np.std(r2_scores)
```

```
    mean_mape = np.mean(mape_scores)
    std_mape = np.std(mape_scores)

    return mean_r2, std_r2, mean_mape, std_mape
```

In [46]:
```
x_k = x.drop(columns = ['INDUS', 'ZN', 'AGE', 'TAX']) # Discard irrelevant v
results = []

# Use cross validation function for different values of k
for k in range(2, 16):
    mean_r2, std_r2, mean_mape, std_mape = cross_validate_sm_ols(x_k.values,
    results.append((k, mean_r2, 2 * std_r2, mean_mape, 2 * std_mape))

results_df = pd.DataFrame(results, columns=['k', 'R-squared', ' +/-', 'MAPE'
print(results_df)
```

```
     k  R-squared       +/-      MAPE       +/-
0    2   0.761020  0.001353  0.050294  0.003408
1    3   0.753610  0.053949  0.050856  0.002963
2    4   0.756429  0.057054  0.050685  0.011851
3    5   0.758859  0.055010  0.050164  0.006461
4    6   0.758235  0.065670  0.050351  0.012033
5    7   0.744151  0.170680  0.050682  0.012941
6    8   0.754371  0.091982  0.050322  0.013811
7    9   0.738565  0.175791  0.050489  0.013341
8   10   0.752921  0.179383  0.050380  0.014381
9   11   0.746399  0.163482  0.050318  0.011263
10  12   0.749910  0.175206  0.050410  0.014480
11  13   0.744375  0.174850  0.050201  0.016079
12  14   0.749295  0.184597  0.050273  0.016705
13  15   0.751714  0.194131  0.050286  0.020729
```

Splitting the data a different way does not significantly improve the model's performance, therefore "model_3" performs well.

Final Model

In [47]:
```
# The final model would be model_3. Check and display parameters and equatic
coef = model_3.params
print(coef)
Equation = "MEDV="
print(Equation, end='\t')
for i in range(len(coef)):
    print('(', coef[i], ') * ', coef.index[i], '+', end = ' ')
```

```
const      4.469577
CRIM      -0.012602
CHAS       0.135595
NOX       -1.021076
RM         0.078866
DIS       -0.042741
RAD        0.007575
PTRATIO   -0.047572
LSTAT     -0.028128
dtype: float64
MEDV=   ( 4.469576588292422 ) *  const + ( -0.012601642030266917 ) *  CRIM +
( 0.13559503227071162 ) *  CHAS + ( -1.0210758898499361 ) *  NOX + ( 0.07886
5575928663 ) *  RM + ( -0.04274141946759229 ) *  DIS + ( 0.00757475722352162
3 ) *  RAD + ( -0.04757248778481292 ) *  PTRATIO + ( -0.02812793143138356 )
*  LSTAT +
```

- That would be the final regression equation obtained by the model.

Actionable Insights and Recommendations

- This predicition model will be useful to predict the price of a house.
- The model explains ~77% of the variation in the data.
- In order to predict the price of a house, it is important to consider that a non linear (logarithmic) transformation was made to the dependent variable, so additional steps are required to get the desired result.
- The model's performance was evaluated using different metrics (R^2, MAPE, RMSE, MAE) as well as a cross validation tecnique (k-folds) and parameters were adjusted accordingly to achieve the best results.
- All 4 assumptions of linear regression were checked and met.
- As per capite crime rate increases, the house price decreases.
- There is a slight increment in the price of a house when it is close to the Charles River.
- Air pollution caused by nitric oxide decreases the price.
- If the house has more rooms, the price will increase accordingly.
- Price will decrease if the house is closer to Boston employment centers.
- Price will slightly increase if there is more accessibility to radial highways.
- If the pupil teacher ratio increases, house price decreases.
- As the percentage of lower class population increases, price decreases.
- There is additional information that may be of use to provide a more accurate model, such as distance to schools, shopping centers or supermarkets.
