Boston_house_price_prediction_model

May 22, 2025

1 Regression Project: Boston House Price Prediction

Author: Sepehr Safari

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

1.1 Importing the necessary libraries

[ ]:
```python
# Libraries for data exploration
import pandas as pd
import numpy as np

# Libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Add libraries for linear regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Add StateModels libraries
from statsmodels.formula.api import ols
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.diagnostic import het_white
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

# Importing libraries for scaling the data
from sklearn.preprocessing import MinMaxScaler

# Library for uploading dataset
from google.colab import files

# To suppress scientific notations for a dataframe
pd.set_option("display.float_format", lambda x: "%.2f" % x)

# To suppress warnings
import warnings

warnings.filterwarnings("ignore")
```

1.1.1 Loading the dataset

[ ]:
```python
# let colab access my google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

1.2 Data Overview

- Observations
- Sanity checks

[ ]:
```python
#Build the dataframe from the uploaded file
df = pd.read_csv('/content/drive/MyDrive/Boston.csv')

# First find out the number of rows and columns in the data
df.shape
```

[ ]:
```
(506, 13)
```

The original dataset has:

- 509 rows
- 13 columns

1.2.1 Data Sanity checks

View the first 5 rows and 5 last rows of dataframe

[ ]:
```python
# First 5 rows
df.head()
```

[ ]:
```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD  TAX  PTRATIO  LSTAT  \
0  0.01 18.00   2.31     0 0.54 6.58 65.20 4.09    1  296    15.30   4.98
1  0.03  0.00   7.07     0 0.47 6.42 78.90 4.97    2  242    17.80   9.14
2  0.03  0.00   7.07     0 0.47 7.18 61.10 4.97    2  242    17.80   4.03
3  0.03  0.00   2.18     0 0.46 7.00 45.80 6.06    3  222    18.70   2.94
4  0.07  0.00   2.18     0 0.46 7.15 54.20 6.06    3  222    18.70   5.33

   MEDV
0 24.00
1 21.60
2 34.70
3 33.40
4 36.20
```

[ ]:
```python
# Last 5 rows
df.tail()
```

[ ]:
```
     CRIM   ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD  TAX  PTRATIO  LSTAT  \
501  0.06 0.00  11.93     0 0.57 6.59 69.10 2.48    1  273    21.00   9.67
502  0.05 0.00  11.93     0 0.57 6.12 76.70 2.29    1  273    21.00   9.08
503  0.06 0.00  11.93     0 0.57 6.98 91.00 2.17    1  273    21.00   5.64
504  0.11 0.00  11.93     0 0.57 6.79 89.30 2.39    1  273    21.00   6.48
505  0.05 0.00  11.93     0 0.57 6.03 80.80 2.50    1  273    21.00   7.88

     MEDV
```

```
501 22.40
502 20.60
503 23.90
504 22.00
505 11.90
```

Find duplicate values in the data

[ ]:
```python
print(df.duplicated().sum())
```

```
0
```

No duplicated data observed

Find missing values

[ ]:
```python
print(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
LSTAT      0
MEDV       0
dtype: int64
```

No missing value observed

[ ]:
```python
print(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
None
```

- All 506 rows have non-null which confirms no missing values observed before
- There is no non-numeric (Object) type values

Check statistical information

[ ]:
```
df.describe().T
```

[ ]:

	count	mean	std	min	25%	50%	75%	max
CRIM	506.00	3.61	8.60	0.01	0.08	0.26	3.68	88.98
ZN	506.00	11.36	23.32	0.00	0.00	0.00	12.50	100.00
INDUS	506.00	11.14	6.86	0.46	5.19	9.69	18.10	27.74
CHAS	506.00	0.07	0.25	0.00	0.00	0.00	0.00	1.00
NOX	506.00	0.55	0.12	0.39	0.45	0.54	0.62	0.87
RM	506.00	6.28	0.70	3.56	5.89	6.21	6.62	8.78
AGE	506.00	68.57	28.15	2.90	45.02	77.50	94.07	100.00
DIS	506.00	3.80	2.11	1.13	2.10	3.21	5.19	12.13
RAD	506.00	9.55	8.71	1.00	4.00	5.00	24.00	24.00
TAX	506.00	408.24	168.54	187.00	279.00	330.00	666.00	711.00
PTRATIO	506.00	18.46	2.16	12.60	17.40	19.05	20.20	22.00
LSTAT	506.00	12.65	7.14	1.73	6.95	11.36	16.96	37.97
MEDV	506.00	22.53	9.20	5.00	17.02	21.20	25.00	50.00

Some observations:

- Low crime rate in 50% of towns.
- Around 50% of towns have no residential land zoned for lots over 25,000 sq.ft
- 75% of houses have more than 5 rooms.
- Most of the house are not on riverside.
- Almost 75% of the houses are older than 50 years.

1.3 Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

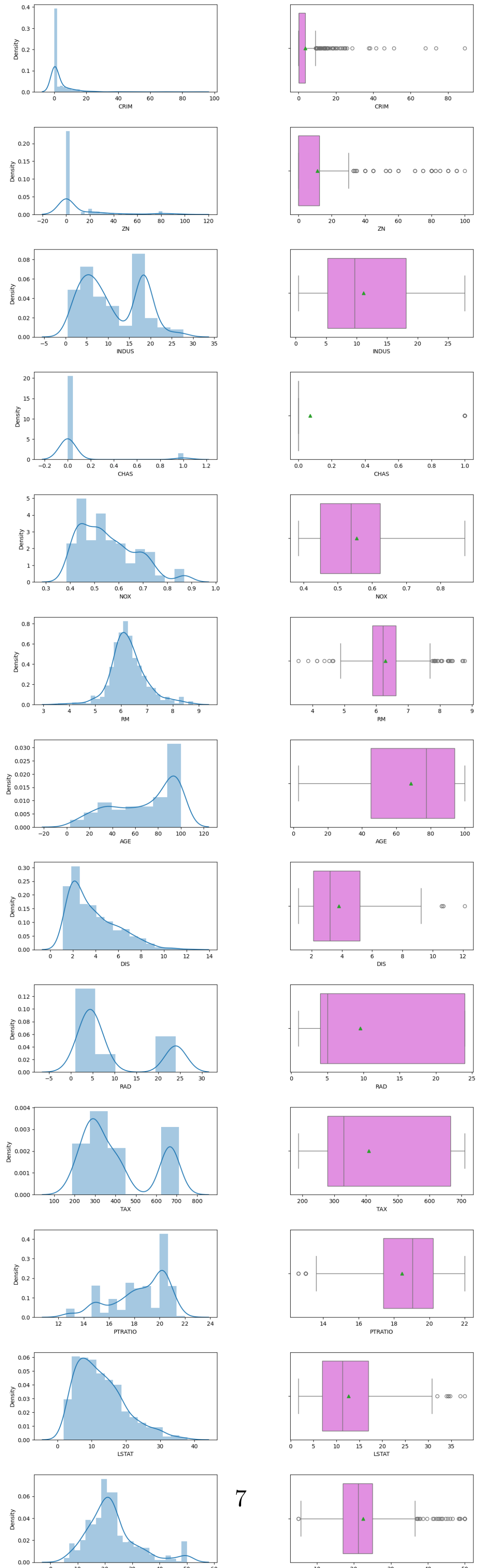
Questions:

1. What does the distribution of 'MEDV' look like?

2. What can we infer form the correlation heatmap? Is there correlation between the dependent and independent variables?
3. What are all the inferences that can be found by doing univariate analysis for different variables?
4. Do bivariate analysis to visualize the relationship between the features having significant correlations (>= 0.7 or <= -0.7)

1.3.1 Univariate analysis

[ ]:
```
columns = df.columns.tolist()
num_columns = len(columns)
fig = plt.figure(figsize=(14,50))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
i = 0
for j, variable in enumerate(columns):
  i=i+1;
  plt.subplot(num_columns,2,i)
  sns.distplot(df[variable])
  i=i+1
  plt.subplot(num_columns,2,i)
  sns.boxplot(df, x=variable, showmeans=True, color="violet")
```


Density
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
LSTAT
MEDV

Univariant observations

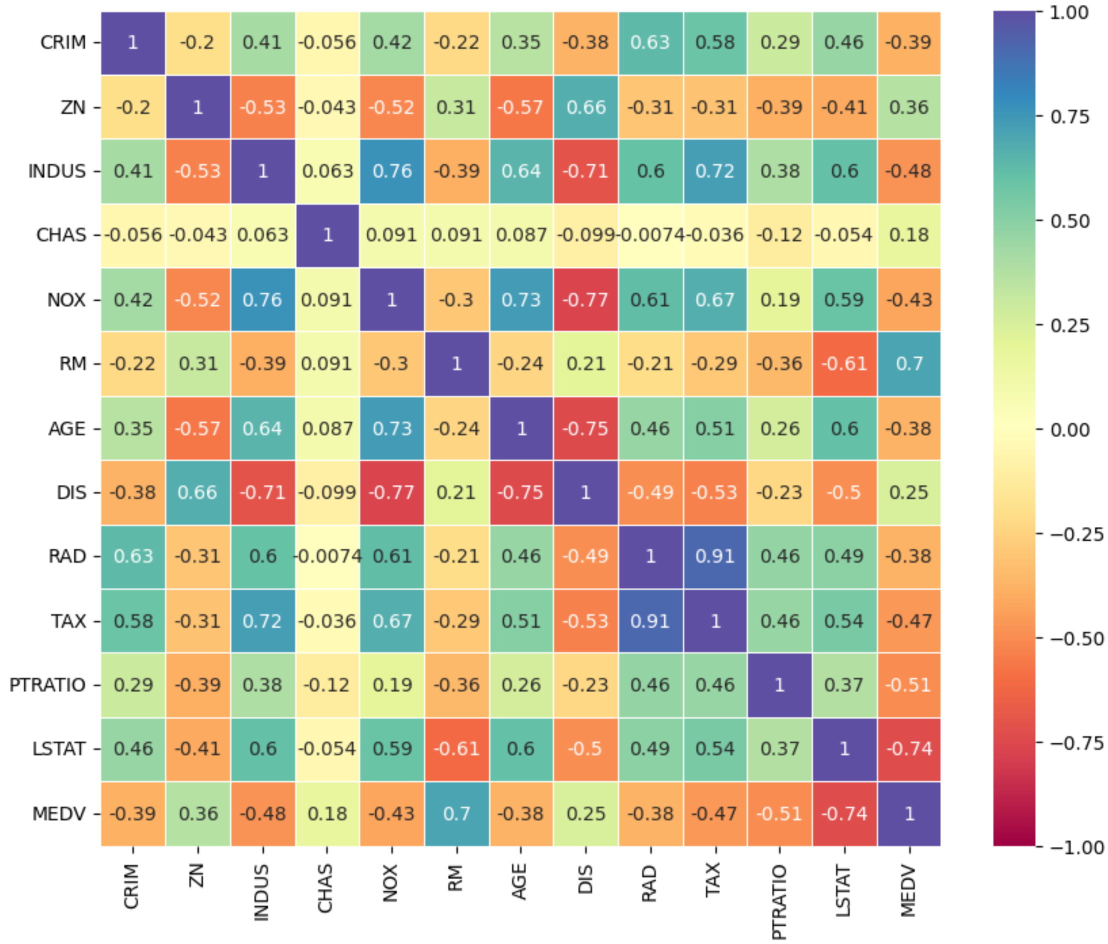
- **CRIM** It shows that most of the areas have lower crime rates with some outlier forcing the destibution sligtly skewed to the right.
- **ZN** It shows that most residential are under the area of 25,000 sq. ft. with outlier to right skewed.
- **CHAS** follows a binomial distribution, and the majority of the houses are outside of riverside.
- **RD** has more a normal distribution with its mean value around 6 rooms
- **AGE** shows that many of the owner-occupied houses were built before 1940.
- **DIS** looks more an exponential distribution, with most of the houses close to the employment centers.
- **RAD** with binomial shape shows more houses have lower index of accessibility to radial highways
- **TAX** and **RAD** follows the same pattern where the tax rate is lower for most with lower RAD index of accessibility to radial highways, and higher for those with higher RAD index.
- **MEDV** Most house prices are around 20000 with a right skewe due to some outliers with jump in number of houses at 50000.

1.3.2 Bivariant Analysis

Checking the relationship between the variables.

Lets check the correlations with the heatmap as well

[ ]:
```
# Correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, linewidths=0.5,␣
 ↪cmap="Spectral",vmin=-1,vmax=1)
plt.yticks(rotation=0)
plt.xticks(rotation = 90);
plt.show()
```



1.3.3 Observation

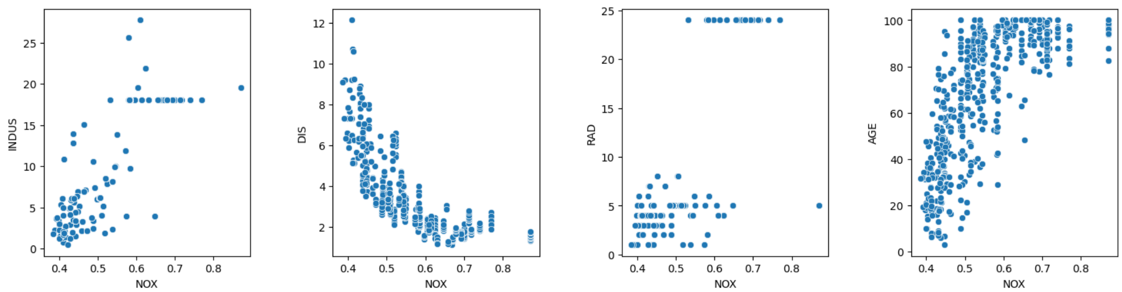
Checking for collerations > 0.7:

- There is a strong colleration between MEDV and RM. Price will increase with number of rooms in same neighborhoods.
- There is colleration 0.91 between TAX and RAD, which doesn't look normal for tax increase when house closer to roads. It can be due to some outlier we need to address this later when we create our model.
- There is 0.72 colleration between TAX and INDUS as well which need to be analyzed as well.
- There are 0.76 correlation between NOX and INDUS plus 0.73 between NOX and AGE. We can be explained by looking at the previous plots older houses are closer to INDUS and normal to have hight NOX.

First let check the correlation of NOX with some of the variables :

If there is no significant dependancies we may can drop NOX.

[ ]:
```python
# Using scatterplot to visualize the relationship
fig = plt.figure(figsize = (18,10))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

ax = fig.add_subplot(2, 4, 1)
sns.scatterplot(data=df, x="NOX", y="INDUS")

ax = fig.add_subplot(2, 4, 2)
sns.scatterplot(data=df, x="NOX", y="DIS")

ax = fig.add_subplot(2, 4, 3)
sns.scatterplot(data=df, x="NOX", y="RAD");

ax = fig.add_subplot(2, 4, 4)
sns.scatterplot(data=df, x="NOX", y="AGE");
```



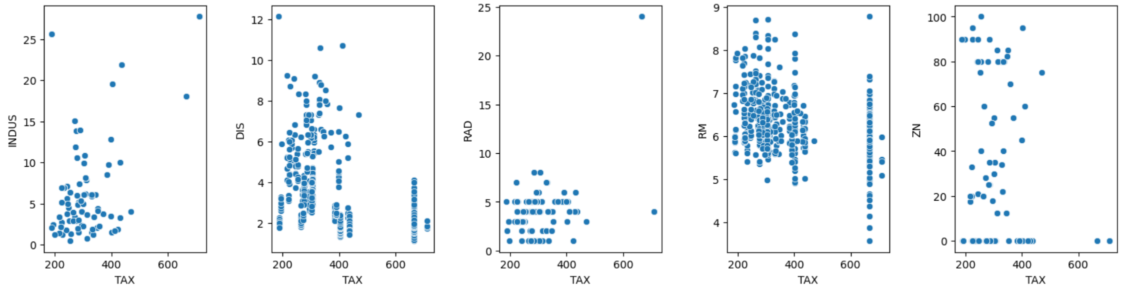
Observation and Analysis

- We can't find a significant correlation between NOX and INDUS from first graph
- In 2ed plot we can observe more distance the house are from employment centers there is less NOX.
- In 3rd plot there is no high correlation between NOX and distance to the freeway maybe due to outlier but further than roads there is less NOX
- In 4th plot we can see with age increase NOX increase more since a lot of houses are older than 50 years, the older one must be closer to DIS or RAD and younger houses are further.

We can check the TAX variation based on other variables

[ ]:
```python
# Using scatterplot to visualize the relationship
fig = plt.figure(figsize = (18,10))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

ax = fig.add_subplot(2, 5, 1)
sns.scatterplot(data=df, x="TAX", y="INDUS")

ax = fig.add_subplot(2, 5, 2)
sns.scatterplot(data=df, x="TAX", y="DIS")
```

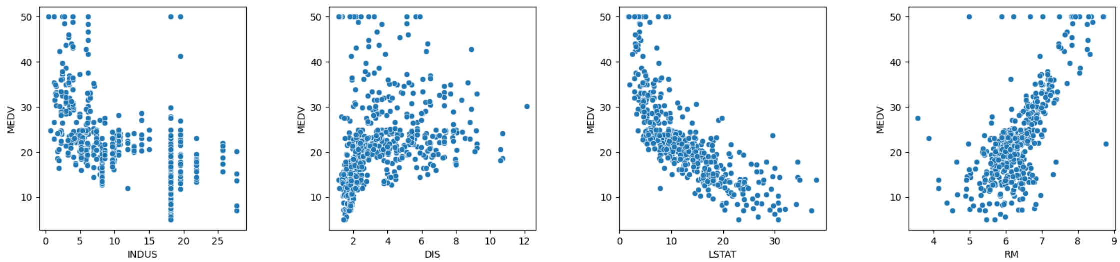
```
ax = fig.add_subplot(2, 5, 3)
sns.scatterplot(data=df, x="TAX", y="RAD");

ax = fig.add_subplot(2, 5, 4)
sns.scatterplot(data=df, x="TAX", y="RM");

ax = fig.add_subplot(2, 5, 5)
sns.scatterplot(data=df, x="TAX", y="ZN");
```



Now lets check all depending variable related to the independent variable MEDV

MEDV is the independent variable and we will check its correlation with other variables.

[ ]:
```
# Using scatterplot to visualize the relationship
fig = plt.figure(figsize = (20,10))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

ax = fig.add_subplot(2, 4, 1)
sns.scatterplot(data=df, x="INDUS", y="MEDV")

ax = fig.add_subplot(2, 4, 2)
sns.scatterplot(data=df, x="DIS", y="MEDV")

ax = fig.add_subplot(2, 4, 3)
sns.scatterplot(data=df, x="LSTAT", y="MEDV");

ax = fig.add_subplot(2, 4, 4)
sns.scatterplot(data=df, x="RM", y="MEDV");
```



The take away from these plots are:

- In first plot we can when INDUS index decrease the price increases too
- In 2ed plot we can observe that with DIS increas price increase slightly as well, but not significant relation.
- In 3rd plot we can see more the location is lower status of the population the house price gets lower.
- In 4th plot there is increase in house price with increase of rooms. There are some outlier can be observed as well.

1.4 Data Preprocessing

- Missing value treatment
- Log transformation of dependent variable if skewed
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

IQR Based Filtering for finding outlier in some of skewed variables

[ ]:
```
#keep a new copy of our data
new_df = df.copy()
```

Log transformation of dependent variable

From above MEDV plot graph we can observer that it is skewed to the right

[ ]:
```
print("Highest allowed",new_df['MEDV'].mean() + 3*new_df['MEDV'].std())
print("Lowest allowed",new_df['MEDV'].mean() - 3*new_df['MEDV'].std())
```

```
Highest allowed 50.124118586250134
Lowest allowed -5.058505938028784
```

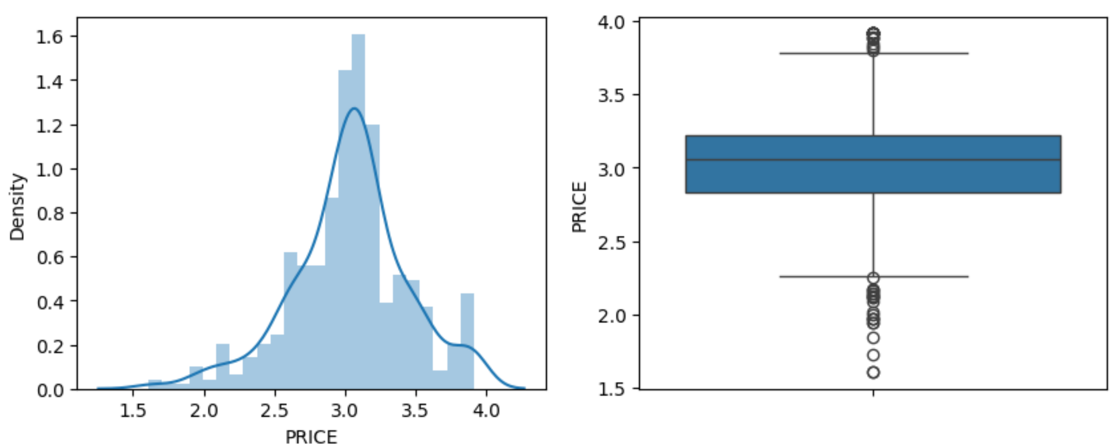
[ ]:
```
new_df['PRICE'] = np.log(new_df['MEDV'])
new_df.head()
```

[ ]:
```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD  TAX  PTRATIO  LSTAT  \
0  0.01 18.00   2.31     0 0.54 6.58 65.20 4.09    1  296    15.30   4.98
1  0.03  0.00   7.07     0 0.47 6.42 78.90 4.97    2  242    17.80   9.14
```

```
2  0.03  0.00   7.07     0 0.47 7.18 61.10 4.97    2  242    17.80   4.03
3  0.03  0.00   2.18     0 0.46 7.00 45.80 6.06    3  222    18.70   2.94
4  0.07  0.00   2.18     0 0.46 7.15 54.20 6.06    3  222    18.70   5.33

   MEDV  PRICE
0 24.00   3.18
1 21.60   3.07
2 34.70   3.55
3 33.40   3.51
4 36.20   3.59
```

[ ]:
```
plt.figure(figsize=(10,8))
plt.subplot(2,2,1)
sns.distplot(new_df['PRICE'])
plt.subplot(2,2,2)
sns.boxplot(new_df['PRICE'])
```

[ ]: `<Axes: ylabel='PRICE'>`



After this transformation the house prices nearly close to a normal distribution.

No need to remove the outlier and we can take care of them by using log tramsformation.

Seperate dependent and independent variables

[ ]:
```
Y = new_df['PRICE']
X = new_df.drop(columns = ['MEDV','PRICE'], axis=1)

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

Splitting data to train and test

```
[ ]: # splitting the data in 80:20 ratio of train to test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20 ,␣
 ↪random_state=1)
```

```
[ ]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(404, 13) (102, 13) (404,) (102,)
```

1.5 Model Building - Linear Regression

```
[ ]: # create the first model
ols_model_1 = sm.OLS(y_train, X_train).fit()

# get the first model summary
print(ols_model_1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  PRICE   R-squared:                       0.783
Model:                            OLS   Adj. R-squared:                  0.777
Method:                 Least Squares   F-statistic:                     117.8
Date:                Sun, 30 Mar 2025   Prob (F-statistic):          1.02e-121
Time:                        15:53:46   Log-Likelihood:                 101.65
No. Observations:                 404   AIC:                            -177.3
Df Residuals:                     391   BIC:                            -125.3
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.4875      0.222     20.194      0.000       4.051       4.924
CRIM          -0.0109      0.001     -7.966      0.000      -0.014      -0.008
ZN             0.0015      0.001      2.334      0.020       0.000       0.003
INDUS          0.0019      0.003      0.717      0.474      -0.003       0.007
CHAS           0.0934      0.036      2.559      0.011       0.022       0.165
NOX           -0.9213      0.171     -5.376      0.000      -1.258      -0.584
RM             0.0680      0.019      3.548      0.000       0.030       0.106
AGE            0.0005      0.001      0.829      0.408      -0.001       0.002
DIS           -0.0506      0.009     -5.493      0.000      -0.069      -0.032
RAD            0.0133      0.003      4.256      0.000       0.007       0.019
TAX           -0.0005      0.000     -3.084      0.002      -0.001      -0.000
PTRATIO       -0.0399      0.006     -6.670      0.000      -0.052      -0.028
LSTAT         -0.0301      0.002    -13.182      0.000      -0.035      -0.026
==============================================================================
Omnibus:                       42.023   Durbin-Watson:                   1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              130.632
Skew:                           0.438   Prob(JB):                     4.30e-29
Kurtosis:                       5.645   Cond. No.                     1.17e+04
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.17e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Observations:

- R square and adjusted R square values are large which gives a good level of confidence about the model.
- Independent variables (AGE and INDUS) have a high p-value and low t, which implies a minimum significance.

[ ]:
```
vif_series = pd.Series(
    [variance_inflation_factor(X_train.values, i) for i in range(X_train.
 ↪shape[1])],
    index = X_train.columns,
    dtype = float)

print("VIF Scores: \n\n{}\n".format(vif_series))
```

```
VIF Scores:

const     545.47
CRIM        1.73
ZN          2.52
INDUS       3.81
CHAS        1.07
NOX         4.42
RM          1.93
AGE         3.23
DIS         4.21
RAD         8.08
TAX         9.81
PTRATIO     1.86
LSTAT       2.99
dtype: float64
```

VIF scores for RAD and TAX is higher than 5. Lets drop one of them and check how it improve the model

[ ]:
```
Y2 = new_df['PRICE']

#Here we drop TAX which has higher VIF score
X2 = new_df.drop(columns = ['MEDV','PRICE', 'TAX'], axis=1)
```

```
# add the intercept
X2 =sm.add_constant(X2)
```

[ ]:
```
# splitting the data in 80:20 ratio of train to test data
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X2, Y2, test_size=0.
 ↪20 , random_state=1)
```

[ ]:
```
print(X_train_2.shape, X_test_2.shape, y_train_2.shape, y_test_2.shape)
```

```
(404, 12) (102, 12) (404,) (102,)
```

[ ]:
```
# create new model2 after dropping TAX column
ols_model_2 = sm.OLS(y_train_2, X_train_2).fit()

# get the model2 summary
print(ols_model_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  PRICE   R-squared:                       0.778
Model:                            OLS   Adj. R-squared:                  0.772
Method:                 Least Squares   F-statistic:                     125.0
Date:                Sun, 30 Mar 2025   Prob (F-statistic):          9.46e-121
Time:                        15:53:47   Log-Likelihood:                 96.797
No. Observations:                 404   AIC:                            -169.6
Df Residuals:                     392   BIC:                            -121.6
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.4201      0.224     19.774      0.000       3.981       4.860
CRIM          -0.0108      0.001     -7.845      0.000      -0.014      -0.008
ZN             0.0010      0.001      1.570      0.117      -0.000       0.002
INDUS         -0.0015      0.002     -0.621      0.535      -0.006       0.003
CHAS           0.1077      0.037      2.944      0.003       0.036       0.180
NOX           -0.9681      0.173     -5.611      0.000      -1.307      -0.629
RM             0.0717      0.019      3.705      0.000       0.034       0.110
AGE            0.0004      0.001      0.729      0.466      -0.001       0.002
DIS           -0.0497      0.009     -5.345      0.000      -0.068      -0.031
RAD            0.0055      0.002      2.966      0.003       0.002       0.009
PTRATIO       -0.0418      0.006     -6.933      0.000      -0.054      -0.030
LSTAT         -0.0299      0.002    -12.972      0.000      -0.034      -0.025
==============================================================================
Omnibus:                       37.811   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              121.737
Skew:                           0.364   Prob(JB):                     3.67e-27
Kurtosis:                       5.589   Cond. No.                     2.08e+03
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.08e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- Almost 10% improvement in condition number after dropping TAX
- Regression coefficients corresponding to ZN, AGE, and INDUS are not statistically significant. So we can drop them alongside of TAX.

[ ]:
```python
# create the new model3 after dropping new columns.
Y3 = new_df['PRICE']
X3 = new_df.drop(columns = {'MEDV', 'PRICE', 'ZN', 'AGE', 'INDUS', 'TAX'})
X3 = sm.add_constant(X3)

#splitting the new dataframe
X_train_3, X_test_3, y_train_3, y_test_3 = train_test_split(X3, Y3, test_size=0.
↪20 , random_state=1)

# create the model3
ols_model_3 = sm.OLS(y_train_3, X_train_3).fit()
print(ols_model_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  PRICE   R-squared:                       0.776
Model:                            OLS   Adj. R-squared:                  0.772
Method:                 Least Squares   F-statistic:                     171.3
Date:                Sun, 30 Mar 2025   Prob (F-statistic):          2.30e-123
Time:                        15:53:47   Log-Likelihood:                 95.125
No. Observations:                 404   AIC:                            -172.2
Df Residuals:                     395   BIC:                            -136.2
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.4259      0.223     19.864      0.000       3.988       4.864
CRIM          -0.0106      0.001     -7.726      0.000      -0.013      -0.008
CHAS           0.1074      0.037      2.939      0.003       0.036       0.179
NOX           -0.9808      0.156     -6.296      0.000      -1.287      -0.675
RM             0.0801      0.019      4.294      0.000       0.043       0.117
DIS           -0.0436      0.007     -6.042      0.000      -0.058      -0.029
RAD            0.0056      0.002      3.058      0.002       0.002       0.009
PTRATIO       -0.0450      0.006     -8.076      0.000      -0.056      -0.034
LSTAT         -0.0293      0.002    -13.808      0.000      -0.033      -0.025
```

```
==============================================================================
Omnibus:                       40.722   Durbin-Watson:                   1.936
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              132.395
Skew:                           0.402   Prob(JB):                     1.78e-29
Kurtosis:                       5.687   Cond. No.                         691.
==============================================================================

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

- We can see a big improvement in condition number
- Small decrease in R-squared and Adj. R-squared due to dropping those columns but still at good confidence level.

1.6 Model Performance Check

1. How does the model is performing? Check using Rsquared, RSME, MAE, MAPE
2. Is there multicollinearity? Check using VIF
3. How does the model is performing after cross validation?

Lets check our 3rd model performance

Check for multicollinearity using VIF

```
[ ]: vif_series = pd.Series(
         [variance_inflation_factor(X_train_3.values, i) for i in range(X_train_3.
      ↪shape[1])],
         index = X_train_3.columns,
         dtype = float)

     print("VIF Scores: \n\n{}\n".format(vif_series))
```

```
VIF Scores:

const      536.35
CRIM         1.71
CHAS         1.05
NOX          3.57
RM           1.78
DIS          2.53
RAD          2.68
PTRATIO      1.57
LSTAT        2.53
dtype: float64
```

We can see all scores are below 5 which is a good indicator that we have removed the multicollinearity.

Check using Rsquared, RSME, MAE, MAPE and MSE

[ ]:
```
y_train_pred = ols_model_3.predict(X_train_3)
y_test_pred = ols_model_3.predict(X_test_3)

performance = {}
performance['data'] = ["Train", 'Test']
performance['RMSE'] = [np.sqrt(((y_train_pred - y_train_3) ** 2).mean()), np.
↪sqrt(((y_test_pred - y_test_3) ** 2).mean())]
performance['MAE'] = [np.mean(np.abs((y_train_pred - y_train_3))), np.mean(np.
↪abs((y_test_pred - y_test_3)))]
performance['MAPE'] = [np.mean(np.abs((y_train_pred - y_train_3)) /␣
↪y_train_pred) * 100, np.mean(np.abs((y_test_pred - y_test_3)) / y_test_pred)␣
↪* 100]
performance['R2'] = [r2_score(y_train_3, y_train_pred), r2_score(y_test_3,␣
↪y_test_pred)]

print(pd.DataFrame(performance))
print("\nMean Square Error (MSE): %0.2f" % ols_model_3.mse_resid)
```

```
    data  RMSE  MAE  MAPE   R2
0  Train  0.19 0.14  4.80 0.78
1   Test  0.21 0.16  5.34 0.76

Mean Square Error (MSE): 0.04
```

Observation

- These numbers give a good impression about the model accurancy base their low values
- R-squared is high enough to show the accurancy of the modle
- The error numbers on the test data are slightly higher probably can be tuned by revisiting data processing for tuning.

The **cross-validation score** to identify if the model that we have built is **underfitted, overfitted** or just right fit model.

[ ]:
```
lrg = LinearRegression()

cv_Score11 = cross_val_score(lrg, X_train_3, y_train_3, cv = 10)
cv_Score12 = cross_val_score(lrg, X_train_3, y_train_3, 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: 0.733 (+/- 0.201)
Mean Squared Error: 0.040 (+/- 0.018)
```

Observations:

- The R-Squared on the cross-validation is 0.733 which is almost similar to the R-Squared on

the training dataset.
- The MSE on cross-validation is 0.041 which is almost similar to the MSE on the training dataset.

It seems like that our model is just right fit. It is giving a generalized performance.

1.7 Checking Linear Regression Assumptions

- In order to make statistical inferences from a linear regression model, it is important to ensure that the assumptions of linear regression are satisfied.

Linear regression assumptions are:

- Mean of residuals should be 0
- Normality of error terms
- Linearity of variables
- No Heteroscedasticity

Mean of residuals should be 0

[ ]:
```
# Residuals
residual = ols_model_3.resid
print(residual.mean())
```

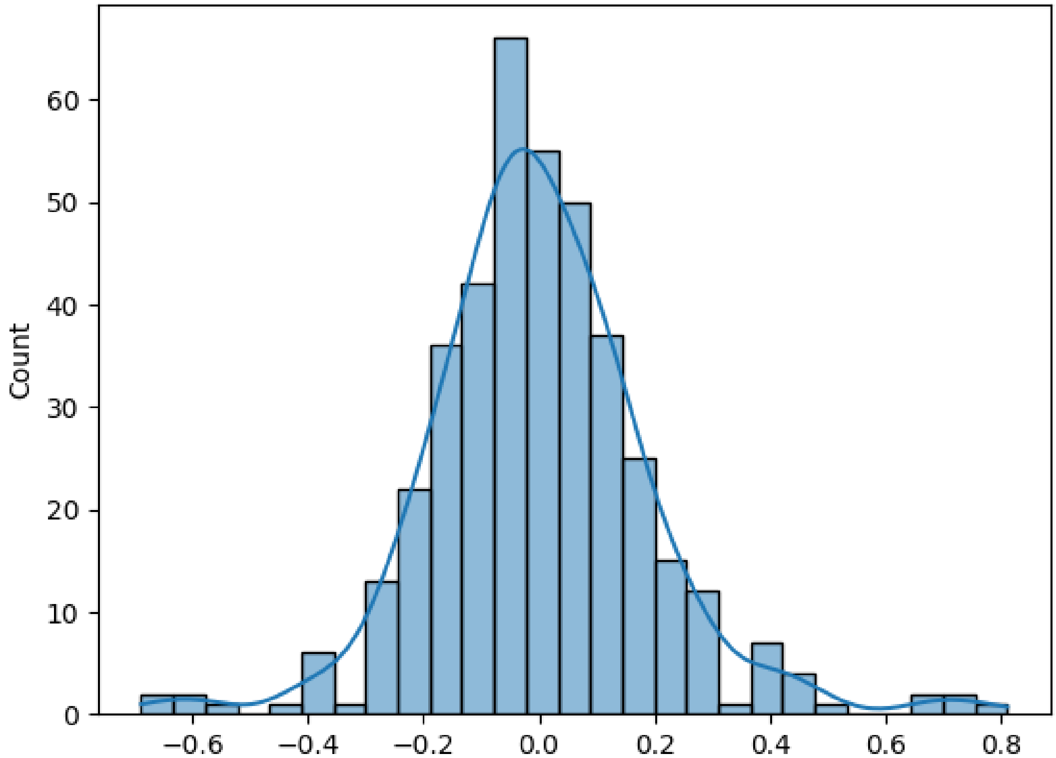
```
7.936445779993693e-16
```

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

Normality of error terms

[ ]:
```
# Plot histogram of residuals
sns.histplot(residual, kde = True)
```

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

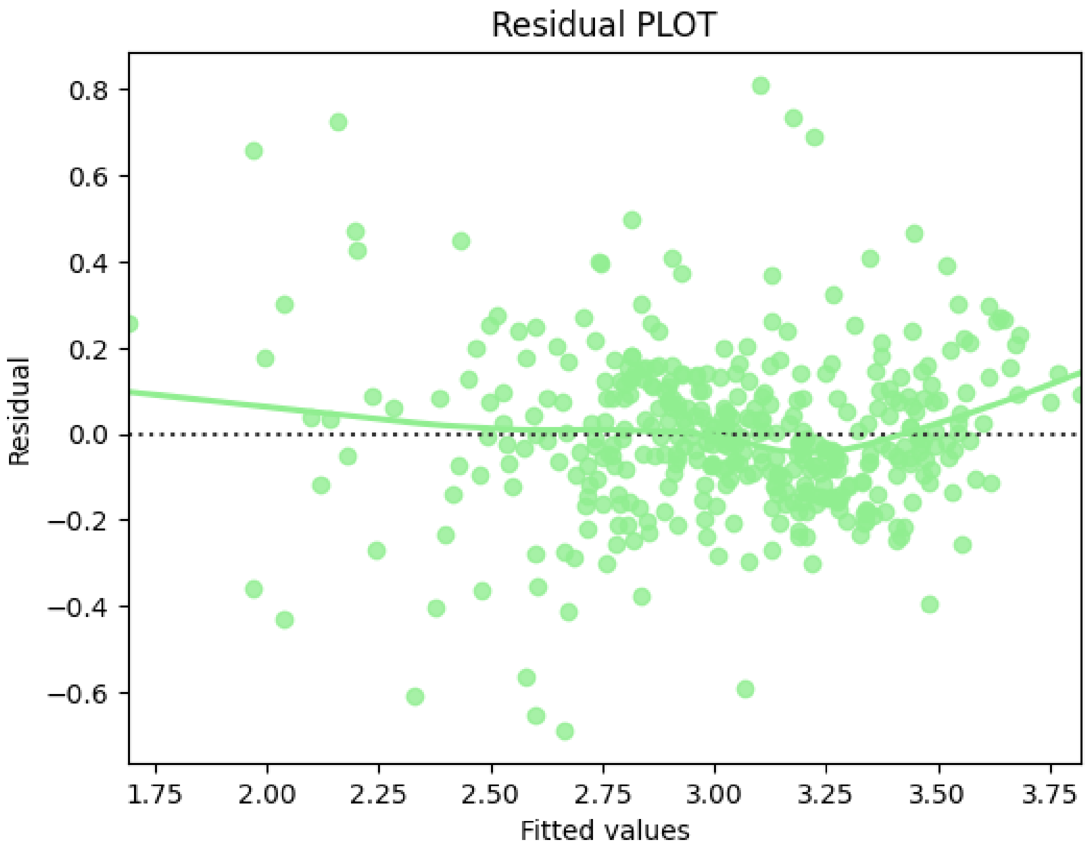


We can see that the error terms are normally distributed. The assumption of normality is satisfied.

Linearity of Variables We must check the linear relation of the predictor variables with the dependent variable. And that must follow a linear pattern.

One way to verify that it is to check disparity of the residuals and the fitted values and check that residuals do not form a pattern and randomly scattered on the x-axis.

[ ]:
```
fitted = ols_model_3.fittedvalues

sns.residplot(x = fitted, y = residual , color="lightgreen", lowess=True)
plt.xlabel("Fitted values")
plt.ylabel("Residual")
plt.title("Residual PLOT")
plt.show()
```



Observations:

We can see that there is no pattern in the residuals vs fitted values scatter plot and we have our linearity satisfied.

No Heteroscedasticity Test to check homoscedasticity:

- Null hypothesis : Residuals are homoscedastic
- Alternate hypothesis : Residuals are hetroscedastic

[ ]:
```python
name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(y_train_3, X_train_3)
print(lzip(name, test))
```

```
[('F statistic', np.float64(1.0430733582484752)), ('p-value',
np.float64(0.384935885603375))]
```

Observations:

We can see that the p-value is greater than 0.05, so we fail to reject the null hypothesis which is the indication that the residuals have homoscedastic. And we conclued that Residuals are **hetroscedastic**

1.8 Final Model

[ ]:
```
coef = pd.Series(index = X_train_3.columns, data = ols_model_3.params.values)

coef_df = pd.DataFrame(data = {'Coefs': ols_model_3.params.values }, index = 
 ↪X_train_3.columns)
print(coef_df)
```

```
         Coefs
const     4.43
CRIM     -0.01
CHAS      0.11
NOX      -0.98
RM        0.08
DIS      -0.04
RAD       0.01
PTRATIO  -0.04
LSTAT    -0.03
```

[ ]:
```
Equation = "log (Price) = "+ str(coef[0])+ " + "
coef.drop(index = 'const', inplace = True)
print(Equation)
for i in range(len(coef)):
  print('(', coef[i], ') * ', coef.index[i], ' + ')
```

```
log (Price) = 4.425887057311069 +
( -0.010629267079143807 ) *  CRIM  +
( 0.10739439482589186 ) *  CHAS  +
( -0.98082752607359 ) *  NOX  +
( 0.08007338538404643 ) *  RM  +
( -0.04361885474272929 ) *  DIS  +
( 0.005551720676718903 ) *  RAD  +
( -0.04498032296642056 ) *  PTRATIO  +
( -0.02931894906026002 ) *  LSTAT  +
```

1.9 Predictions on the Test Dataset with our final Model

Now it is time to check the prediction of our model in action.

We need to transform the output of the our model from log scale back to its original scale by doing the inverse of log transformation in order to see how the prediction and true values are compared below.

Lets first compare the output with actual data in a table

[ ]:
```
# These test predictions are on a log scale
test_predictions = ols_model_3.predict(X_test_3)
```

```
# converting the log scale predictions to its original scale
test_predictions_inverse = np.exp(test_predictions)
dfp = pd.DataFrame({'Actual': np.exp(y_test_3), 'Predicted':␣
↪test_predictions_inverse})
dfp['Residuals'] = dfp['Actual'] - dfp['Predicted']
dfp['Differences%'] = np.abs(dfp['Residuals']/dfp['Actual'] )*100
dfp.head(20)
```

[ ]:

	Actual	Predicted	Residuals	Differences%
307	28.20	28.86	-0.66	2.34
343	23.90	26.12	-2.22	9.28
47	16.60	17.80	-1.20	7.21
67	22.00	23.24	-1.24	5.64
362	20.80	18.08	2.72	13.06
132	23.00	19.28	3.72	16.19
292	27.90	28.53	-0.63	2.25
31	14.50	17.87	-3.37	23.25
218	21.50	22.37	-0.87	4.06
90	22.60	26.49	-3.89	17.22
481	23.70	25.44	-1.74	7.34
344	31.20	27.67	3.53	11.33
119	19.30	21.33	-2.03	10.51
66	19.40	24.73	-5.33	27.45
312	19.40	22.18	-2.78	14.33
407	27.90	18.44	9.46	33.91
376	13.90	13.77	0.13	0.96
225	50.00	37.50	12.50	25.00
201	24.10	29.09	-4.99	20.71
147	14.60	10.78	3.82	26.15

[ ]:
```
dfp.describe().T
```

[ ]:

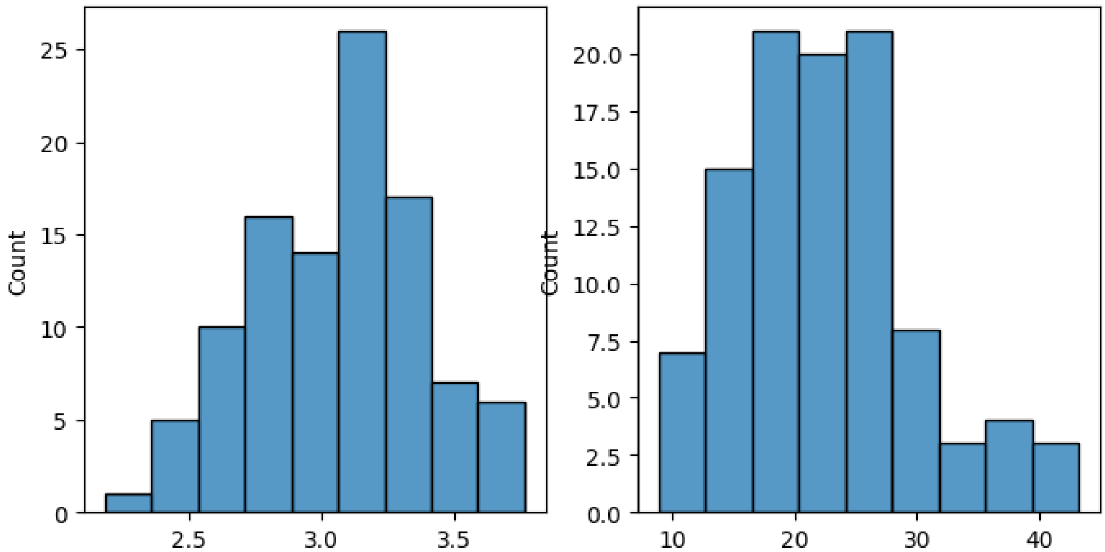
	count	mean	std	min	25%	50%	75%	max
Actual	102.00	22.57	9.99	6.30	16.27	21.85	24.10	50.00
Predicted	102.00	22.41	7.25	8.88	16.75	22.02	26.44	43.22
Residuals	102.00	0.17	4.75	-8.15	-2.82	-0.77	1.65	17.15
Differences%	102.00	16.51	15.50	0.17	5.06	13.75	22.99	74.04

Observation

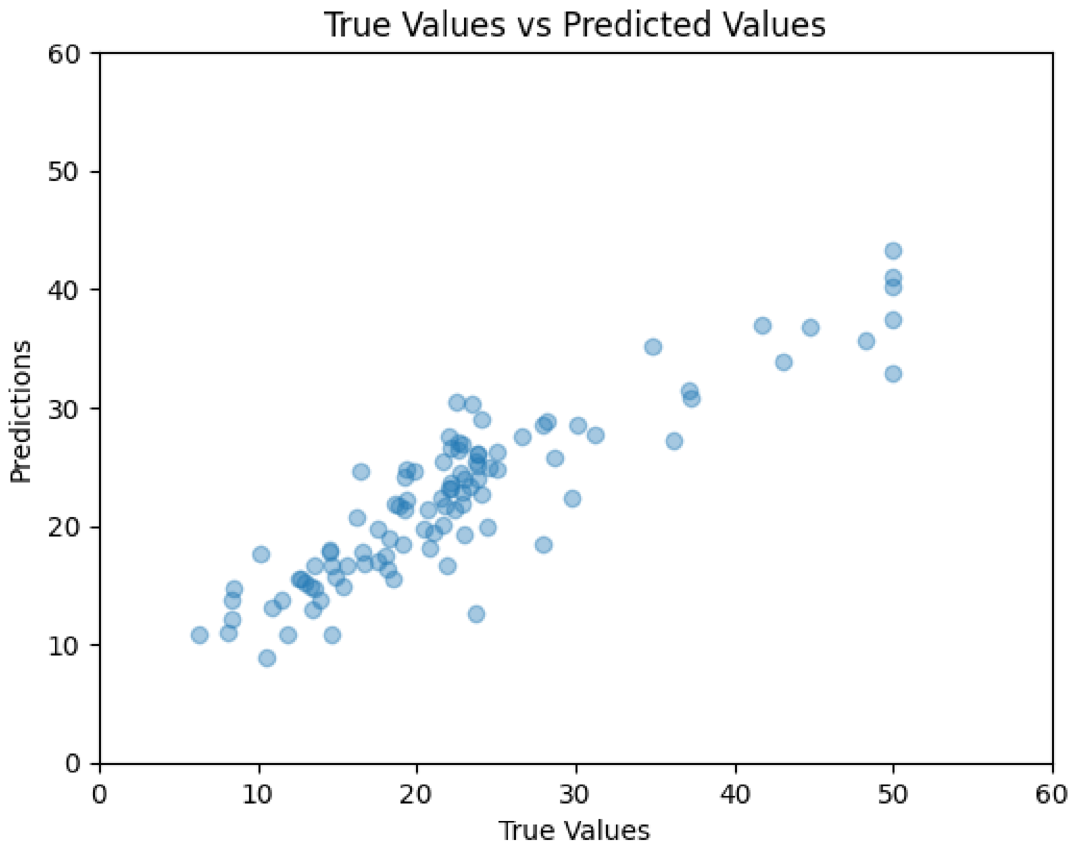
- We have create a dataframe to compare our prediction with actual testing set.
- We have added columns for examining the residuals of the predcition values and actual test values.
- We have added the difference in % to evaluate the ouputs and the targets.
- And the results shows we had close prediction of the prices and our model does relatively good.

Lets check our prediction using the graphs

[ ]:
```
fig, ax = plt.subplots(1, 2, figsize = (8, 4))

sns.histplot(test_predictions, ax = ax[0]);
sns.histplot(test_predictions_inverse, ax = ax[1]);
```



[ ]:
```
plt.scatter(np.exp(y_test_3), test_predictions_inverse, alpha=0.4)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs Predicted Values')
plt.xlim(0, 60)
plt.ylim(0, 60)
plt.show()
```



Conclusion:

Our models does relatively well on lower prices < 50 and it is slightly off on higher prices than 40. This might be more related to the few outliers.

1.10 Actionable Insights and Recommendations

- We performed EDA, univariate and bivariate analysis, on all the variables in the dataset.
- We have checked for missing values and duplicat values to treat which was no needed.
- We checked univariant observations for finding data densities and outlier for all features one by one.
- We check the correlation between features using a heatmap matrix to find more correlated features.
- We studied bivariant data which have the highest correlation number from the heatmap.
- We started the data processes by removing some of the outliers from LSTAT and DIS features.
- We started the model building process with all the features.
- We used statemodels libraries for building our models.
- We kept the most influenciale features by removing features with high p-values and improved the R values.
- That has removed multicollinearity from the data.
- We checked for different assumptions of linear regression.

- Finally, we evaluated the model using different evaluation metrics.

In general our model prediction is good. It deosn't mean that this model cannot be improved. Removing outlier from some of the feature maybe helps to inprove R2 but it can errod the output of the model as well. Therefor, we need to test it more over time and find out other influencing feature and add them to the model in order to get better output.

In conclusion, we can get insight from model equation to see which features have the highest positive or negative influence on the house prices.

Model Equation:

log (Price) = 4.2309720361810506 + (-0.010523068517828627) * CRIM + (0.11255171018658489) * CHAS + (-0.9244406042291291) * NOX + (0.1025739618126921) * RM + (-0.046532450120617665) * DIS + (0.005551720676718903) * RAD + (-0.04197927057814633) * PTRATIO + (-0.03006149859820082) * LSTAT

- The houses prices are more positively affect by their location close to Charles river and the number of rooms they have.
- The prices are more negatively affected by the oxide rate elevation which is mostly high close to employement centers.

Buisiness Recommandation:

As we have evalueted the model equation we can recommends buyers to buy a house near Charles river and mostly away from employement center where price will appreciate more by the time.
