In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.gofplots import ProbPlot
from statsmodels.formula.api import ols
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
data = pd.read_csv('/home/praveen/Desktop/SEM/ML/housing.data.txt',header=None,deli
data.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	

In [3]:

```
data.dtypes
```

Out[3]:

```
CRIM        float64
ZN          float64
INDUS       float64
CHAS          int64
NOX         float64
RM          float64
AGE         float64
DIS         float64
RAD           int64
TAX         float64
PTRATIO     float64
B           float64
LSTAT       float64
MEDV        float64
dtype: object
```

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [5]:

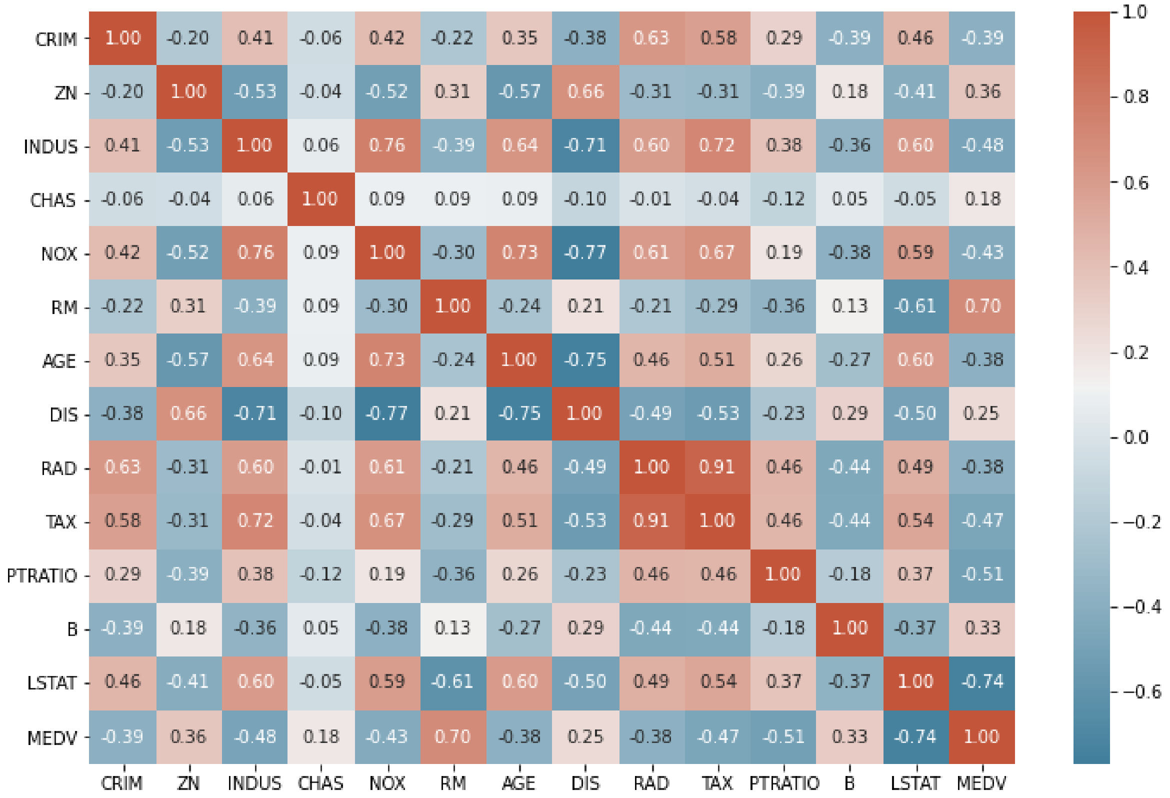
```
data.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	ma
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.97
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.00
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.74
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.00
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.87
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.78
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.00
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.12
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.00
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.00
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.00
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.90
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.97
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.00

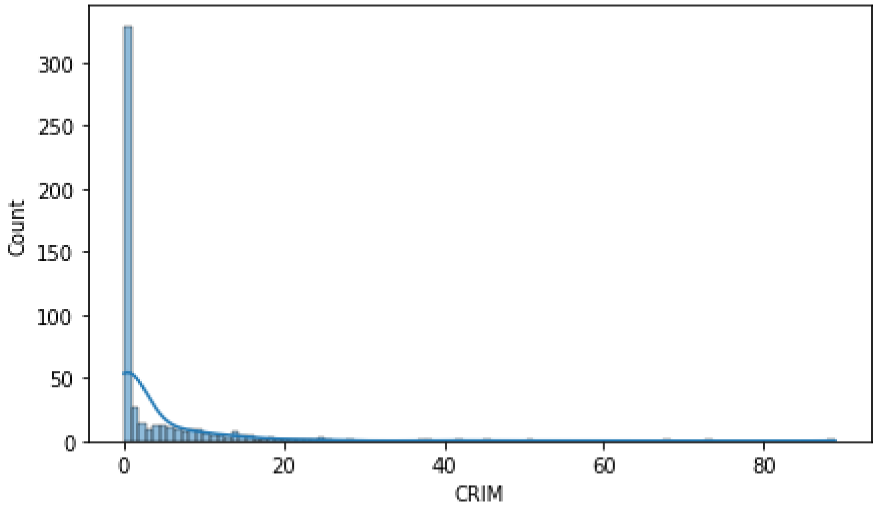
In [6]:

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



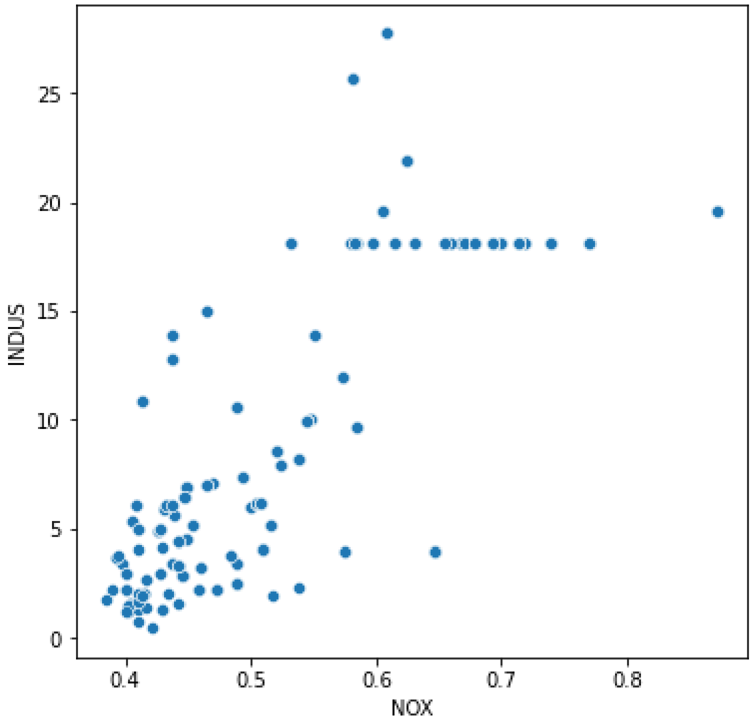
In [7]:

```
for i in data.columns:
    plt.figure(figsize=(7, 4))
    sns.histplot(data, x=i, kde = True)
    plt.show()
```



In [8]:

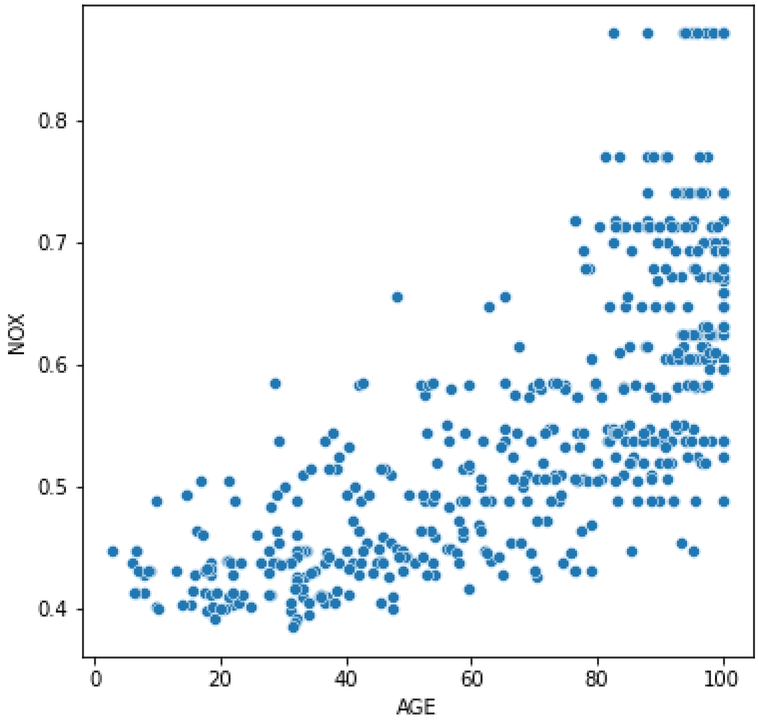
```
plt.figure(figsize=(6, 6))
sns.scatterplot(x=data['NOX'], y=data['INDUS'],data=data)
plt.show()
```



In [9]:

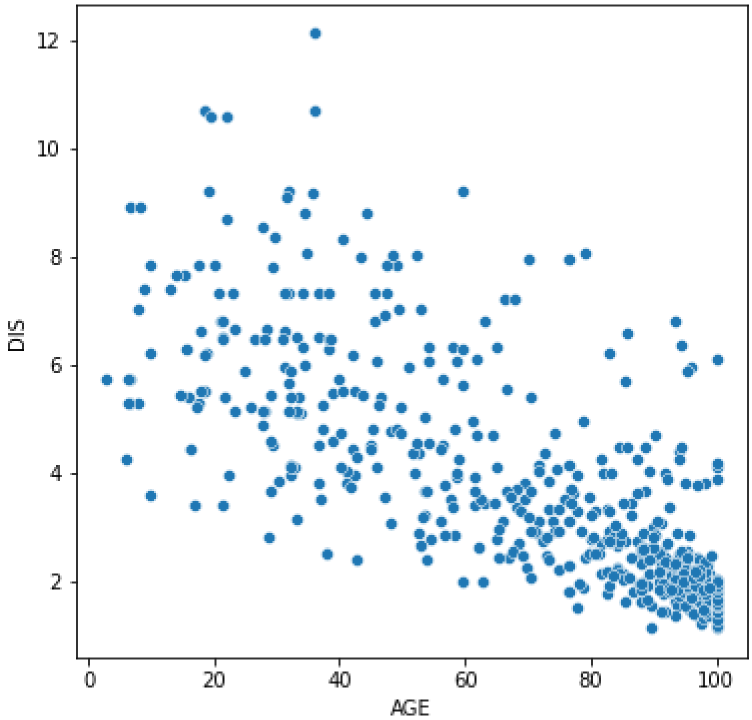
```
plt.figure(figsize=(6, 6))
sns.scatterplot(x=data['AGE'], y=data['NOX'], data=data)

plt.show()
```



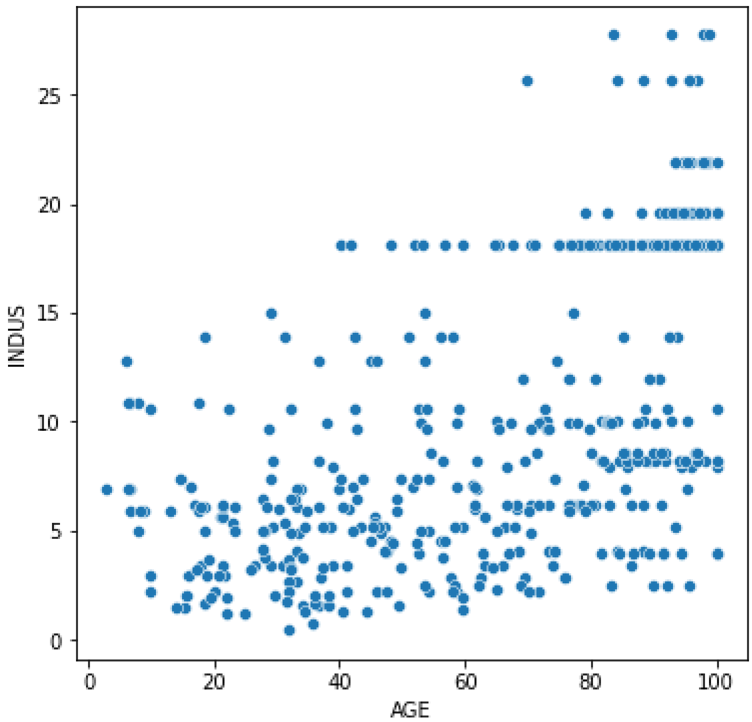
In [10]:

```
plt.figure(figsize=(6, 6))
sns.scatterplot(x = 'AGE', y = 'DIS', data = data)
plt.show()
```



In [11]:

```
plt.figure(figsize=(6, 6))
sns.scatterplot(x = 'AGE', y = 'INDUS', data = data)
plt.show()
```



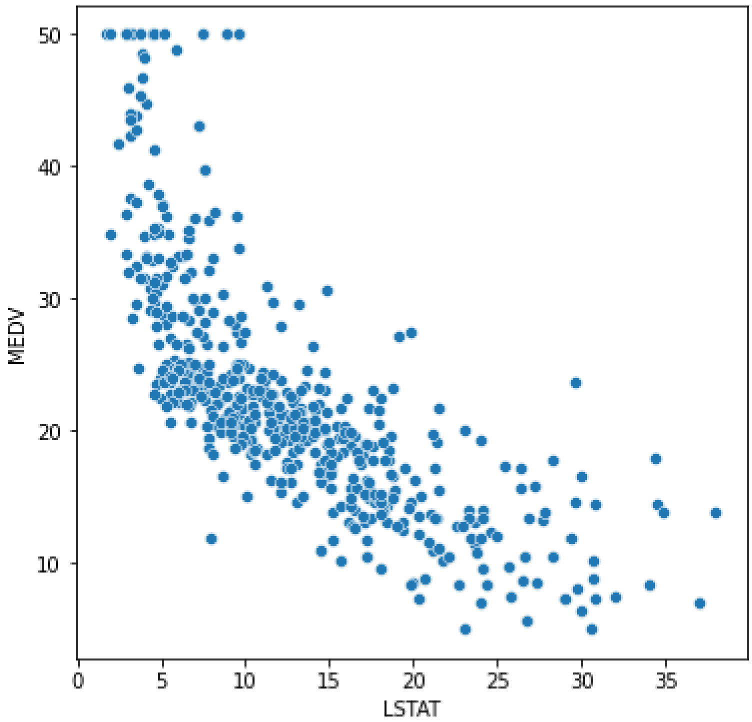
In [12]:

```
df1 = data[data['TAX'] < 600]
from scipy.stats import pearsonr
print('The correlation between TAX and RAD is', pearsonr(df1['TAX'], df1['RAD'])[0]
```

```
The correlation between TAX and RAD is 0.24975731331429196
```

In [13]:

```
plt.figure(figsize=(6, 6))
sns.scatterplot(x = 'LSTAT', y = 'MEDV', data = data)
plt.show()
```



In [14]:

```
data['MEDV_log'] = np.log(data['MEDV'])
Y = data['MEDV_log']
X = data.drop(columns = {'MEDV', 'MEDV_log'})

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

In [15]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30 , random_s
```

In [16]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def checking_vif(train):
    vif = pd.DataFrame()
    vif["feature"] = train.columns
    vif["VIF"] = [
        variance_inflation_factor(train.values, i) for i in range(len(train.columns
    ]
    return vif


print(checking_vif(X_train))
```

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

In [17]:

```
X_train = X_train.drop(['TAX'],1)
print(checking_vif(X_train))
```

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

In [18]:

```
model1 = sm.OLS(y_train, X_train).fit()
model1.summary()
```

Out[18]:

OLS Regression Results

Dep. Variable:	MEDV_log	**R-squared:**	0.771
Model:	OLS	**Adj. R-squared:**	0.763
Method:	Least Squares	**F-statistic:**	95.56
Date:	Sat, 11 Sep 2021	**Prob (F-statistic):**	2.97e-101
Time:	12:16:32	**Log-Likelihood:**	78.262
No. Observations:	354	**AIC:**	-130.5
Df Residuals:	341	**BIC:**	-80.22
Df Model:	12		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	4.4999	0.253	17.767	0.000	4.002	4.998
CRIM	-0.0122	0.002	-7.005	0.000	-0.016	-0.009
ZN	0.0010	0.001	1.417	0.157	-0.000	0.002
INDUS	-0.0002	0.003	-0.066	0.947	-0.006	0.005
CHAS	0.1164	0.039	3.008	0.003	0.040	0.193
NOX	-1.0297	0.187	-5.509	0.000	-1.397	-0.662
RM	0.0569	0.021	2.734	0.007	0.016	0.098
AGE	0.0003	0.001	0.390	0.697	-0.001	0.002
DIS	-0.0496	0.010	-4.841	0.000	-0.070	-0.029
RAD	0.0080	0.002	3.885	0.000	0.004	0.012
PTRATIO	-0.0458	0.007	-6.762	0.000	-0.059	-0.033
B	0.0002	0.000	1.796	0.073	-2.35e-05	0.001
LSTAT	-0.0291	0.002	-11.772	0.000	-0.034	-0.024

Omnibus:	33.707	**Durbin-Watson:**	1.924
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	100.726
Skew:	0.387	**Prob(JB):**	1.34e-22
Kurtosis:	5.496	**Cond. No.**	1.01e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.01e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [19]:

```
X = data.drop(columns = {'MEDV', 'MEDV_log', 'ZN', 'AGE', 'INDUS', 'TAX'})
X = sm.add_constant(X)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30 , random_s
model2 = sm.OLS(y_train, X_train).fit()
model2.summary()
```

Out[19]:

OLS Regression Results

Dep. Variable:	MEDV_log	**R-squared:**	0.769
Model:	OLS	**Adj. R-squared:**	0.763
Method:	Least Squares	**F-statistic:**	127.5
Date:	Sat, 11 Sep 2021	**Prob (F-statistic):**	6.21e-104
Time:	12:16:32	**Log-Likelihood:**	77.190
No. Observations:	354	**AIC:**	-134.4
Df Residuals:	344	**BIC:**	-95.69
Df Model:	9		
Covariance Type:	nonrobust		

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

Omnibus:	35.608	**Durbin-Watson:**	1.927
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	104.246
Skew:	0.425	**Prob(JB):**	2.31e-23
Kurtosis:	5.519	**Cond. No.**	9.76e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.76e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [20]:

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

Out[20]:

```
-2.8570993656314058e-15
```

In [21]:

```
from statsmodels.stats.diagnostic import het_white
from statsmodels.compat import lzip
import statsmodels.stats.api as sms
```

In [22]:

```
name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(y_train, X_train)
lzip(name, test)
```

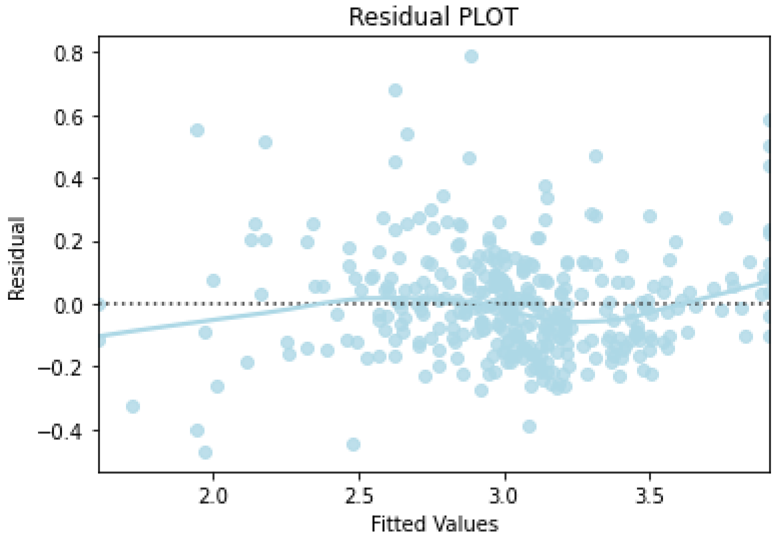
Out[22]:

```
[('F statistic', 1.0844138711700861), ('p-value', 0.3005648212246474
5)]
```

In [23]:

```
fitted = model2.fittedvalues

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

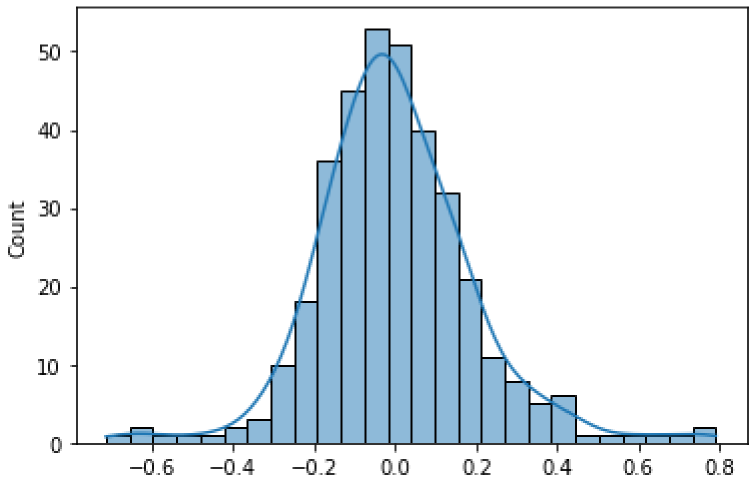


In [24]:

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

Out[24]:

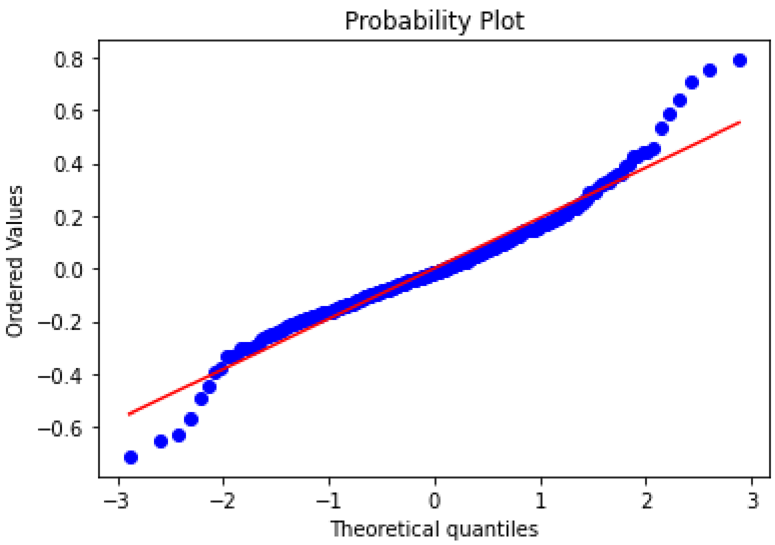
```
<AxesSubplot:ylabel='Count'>
```



In [25]:

```
import pylab
import scipy.stats as stats

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



In [26]:

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

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

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

def model_pref(olsmodel, x_train, x_test):

    y_pred_train = olsmodel.predict(x_train)
    y_observed_train = y_train

    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),
                ],
            }
        )
    )

model_pref(model2, X_train, X_test)
```

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

In [27]:

```
from sklearn.model_selection import cross_val_score

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 =

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.s
```

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

In [28]:

```
coef = pd.Series(index = X_train.columns, data = model2.params.values)

coef_df = pd.DataFrame(data = {'Coefs': model2.params.values }, index =  X_train.co
coef_df
```

Out[28]:

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

In [29]:

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

```
log (Price) =    ( 4.514720483568423 ) *  const + ( -0.0119187751730377
9 ) *  CRIM + ( 0.11649715902151608 ) *  CHAS + ( -1.0234312247045108
) *  NOX + ( 0.06220269133025548 ) *  RM + ( -0.04339113889561061 ) *
DIS + ( 0.008287691091705312 ) *  RAD + ( -0.049037903605757244 ) *  P
TRATIO + ( 0.00024900512380058866 ) *  B + ( -0.02865873169444097 ) *
LSTAT +
```

In [30]:

```
X = data.iloc[:, [0, 12]]
y = data.iloc[:, 13]
```

In [31]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)
```

In [32]:

```
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
```

In [33]:

```
scores = []
best_svr = SVR(kernel='rbf')
cv = KFold(n_splits=10, random_state=None, shuffle=False)
for train_index, test_index in cv.split(X):
    print("Train Index: ", train_index, "\n")
    print("Test Index: ", test_index)

    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_index
    best_svr.fit(X_train, y_train)
    scores.append(best_svr.score(X_test, y_test))
```

```
Train Index:  [ 51  52  53  54  55  56  57  58  59  60  61  62  63
64  65  66  67  68
  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
86
  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103
104
 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
122
 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140
 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
158
 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176
 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193
194
 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211
212
 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229
```

In [34]:

```
best_svr.fit(X_train, y_train)
scores.append(best_svr.score(X_test, y_test))
```

In [35]:

```
cross_val_score(best_svr, X, y, cv=10)
```

Out[35]:

```
array([ 0.71484256,  0.43145909,  0.46093183,  0.00835446,  0.2505539
,
       -0.20966503, -0.45867327,  0.50286329,  0.05559233,  0.2229886
4])
```

In [ ]: