Features Description

CRIM - per capita crime rate by town

ZN - proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS - proportion of non-retail business acres per town.

CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)

NOX - nitric oxides concentration (parts per 10 million)

RM - average number of rooms per dwelling

AGE - proportion of owner-occupied units built prior to 1940

DIS - weighted distances to five Boston employment centres

RAD - index of accessibility to radial highways

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

PTRATIO - pupil-teacher ratio by town

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

LSTAT - % lower status of the population

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

Import Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import pylab
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.preprocessing import PolynomialFeatures
```

Load Dataset

In [2]:

```python
raw_data = pd.read_csv('Boston_Housing.csv')
data = raw_data.copy()
print(data.shape)
data.head()
```

(506, 14)

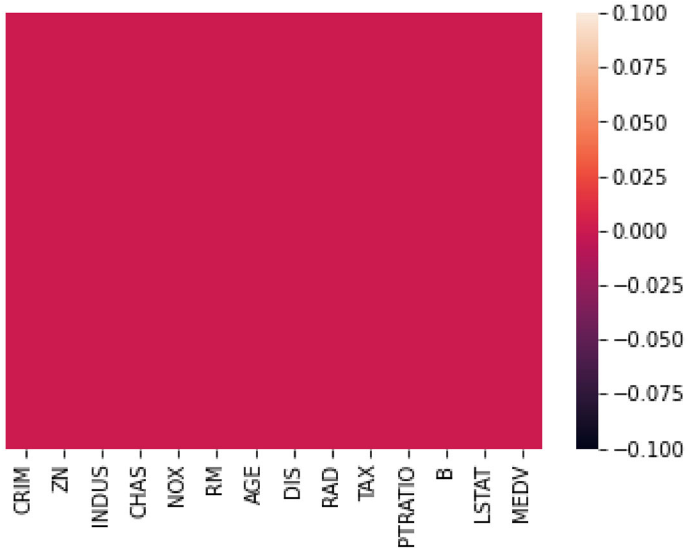
Out[2]:

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

Check Missing values

In [3]:

```python
sns.heatmap(data.isnull(),yticklabels=False)
plt.show()
```

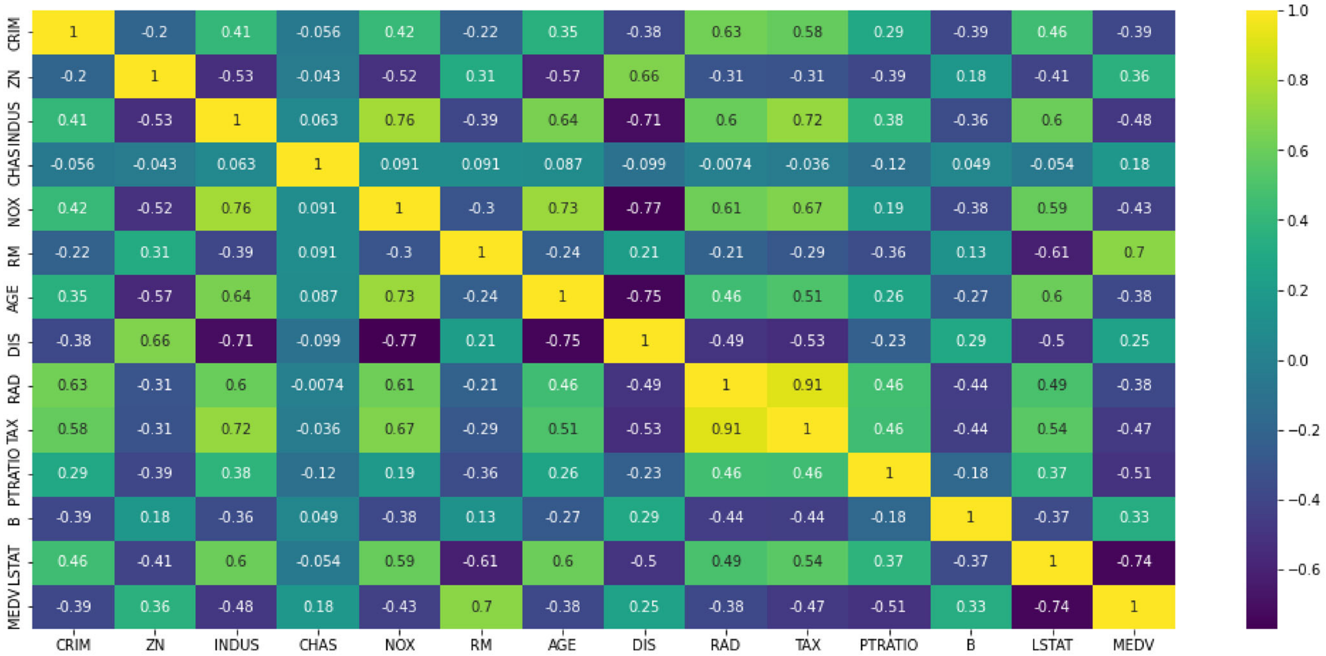


- **No missing value present**

Check important features

In [4]:

```
plt.figure(figsize=(18,8))
sns.heatmap(data.corr(),annot=True,cmap='viridis')
plt.show()
```



- **As LSTAT, PTRATIO & RM are highly correlated with MEDV therefore select these 3 feature for model**

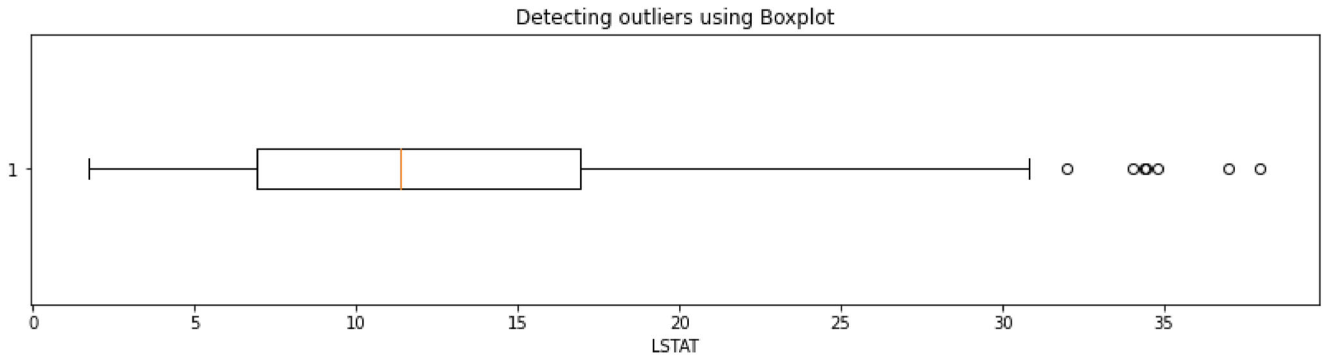
In [5]:

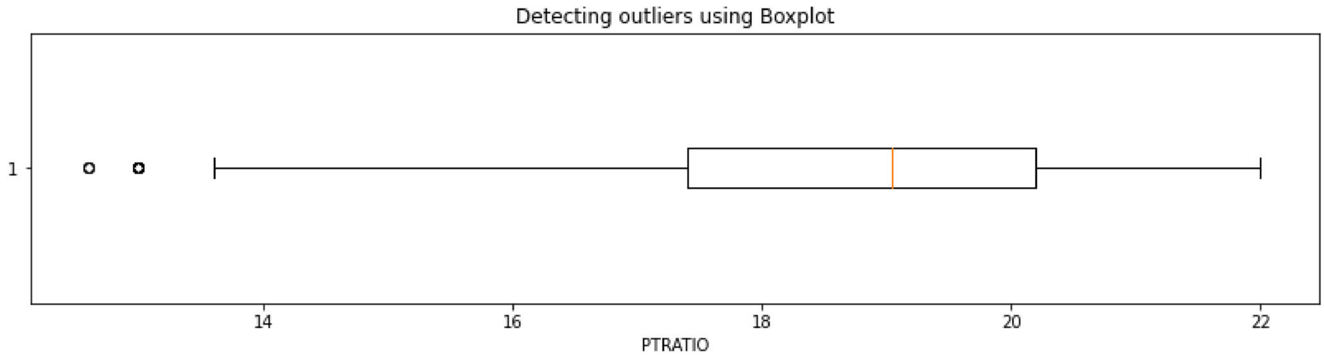
```
data = data[['LSTAT','PTRATIO','RM','MEDV']]
```

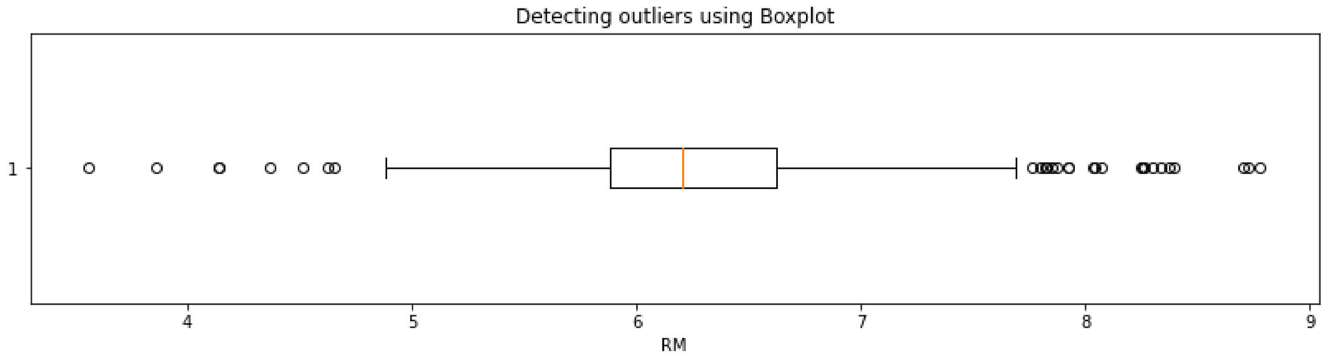
Check presence of outliers using boxplot

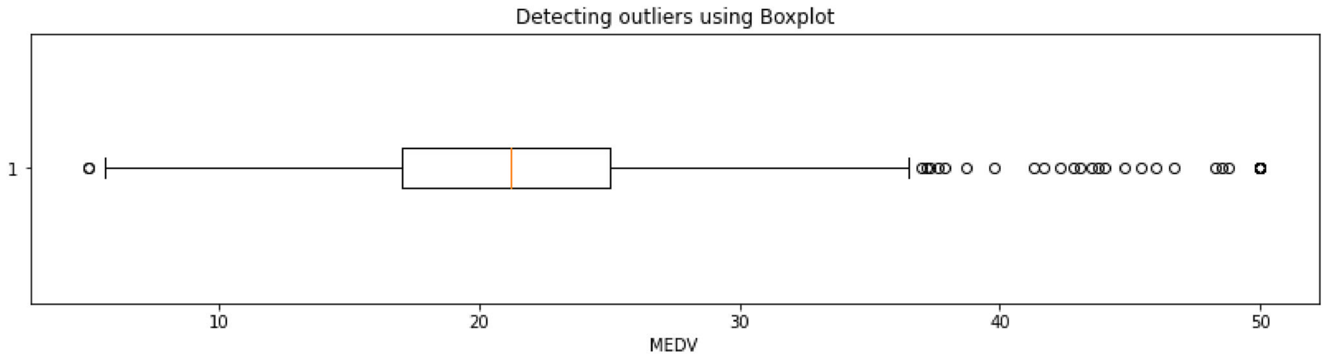
In [6]:

```python
for i in data.columns:
    plt.figure(figsize=(14,3))
    plt.boxplot(data[i], vert=False)
    plt.title("Detecting outliers using Boxplot")
    plt.xlabel(i)
    plt.show()
```









- **In MEDV value 50 repeating 16 times, this is outlier therefore removed**

In [7]:
```
data['MEDV'].value_counts().sort_index()
```
Out[7]:
```
5.0      2
5.6      1
6.3      1
7.0      2
7.2      3
        ..
46.7     1
48.3     1
48.5     1
48.8     1
50.0    16
Name: MEDV, Length: 229, dtype: int64
```

In [8]:
```
data = data[data['MEDV']!=50]
```

- **In LSTAT last 2 values are outliers therefore removed**

In [9]:
```
data['LSTAT'].value_counts().sort_index()
```
Out[9]:
```
1.98     1
2.47     1
2.87     1
2.94     1
2.98     1
        ..
34.37    1
34.41    1
34.77    1
36.98    1
37.97    1
Name: LSTAT, Length: 442, dtype: int64
```

In [10]:
```
data = data[data['LSTAT']<=36]
```

- **In RM last value is outlier therefore removed**

In [11]:
```
data['RM'].value_counts().sort_index()
```
Out[11]:
```
3.561    1
3.863    1
4.138    1
4.368    1
4.628    1
        ..
8.259    1
8.266    1
8.337    1
8.398    1
8.780    1
Name: RM, Length: 430, dtype: int64
```

In [12]:
```
data = data[data['RM']<=8.5]
```

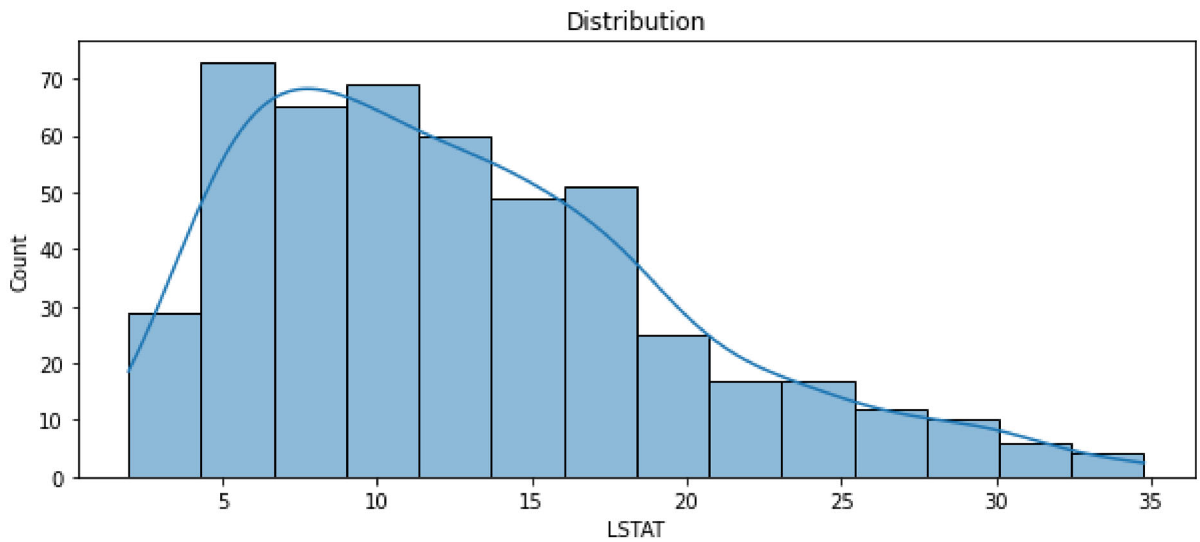
In [13]:
```
data.shape
```
Out[13]:
```
(487, 4)
```

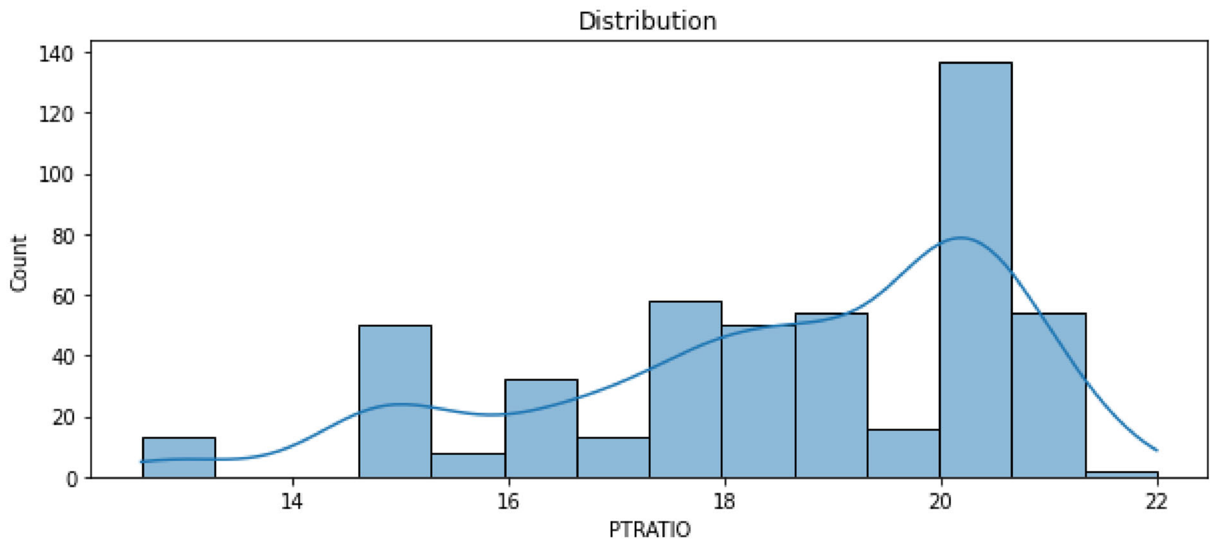
- **Total 19 data points are removed due to outliers**

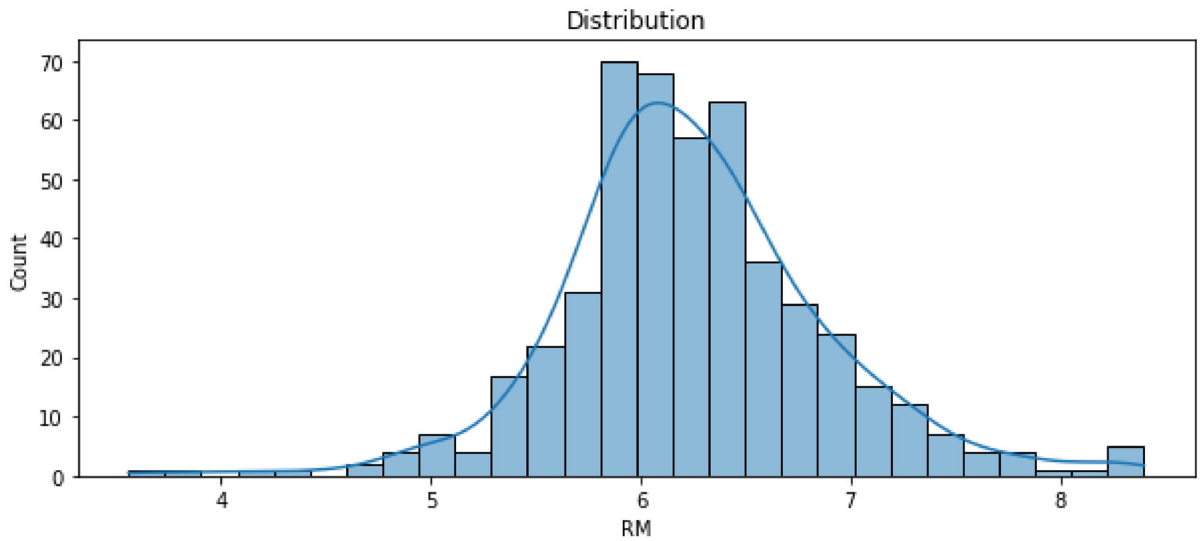
Now check whether features are uniformly distributed or not using histogram

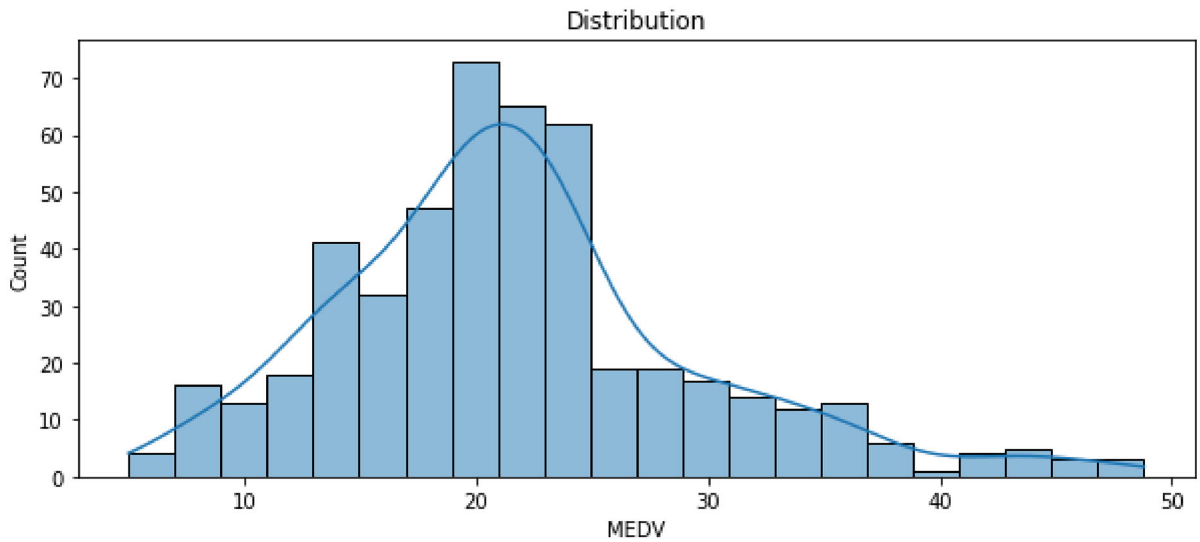
In [14]:

```
for i in data.columns:
    plt.figure(figsize=(10,4))
    sns.histplot(data[i],kde=True)
    plt.title("Distribution")
    plt.xlabel(i)
    plt.show()
```









In [15]:
```
data.skew()
```

Out[15]:
```
LSTAT       0.834479
PTRATIO    -0.814366
RM          0.165935
MEDV        0.781963
dtype: float64
```

- **From graph and above values, it is concluded that**
- **LSTAT and MEDV is right skewed**
- **PTRAIO is left skewed**
- **RM is normally distributed**

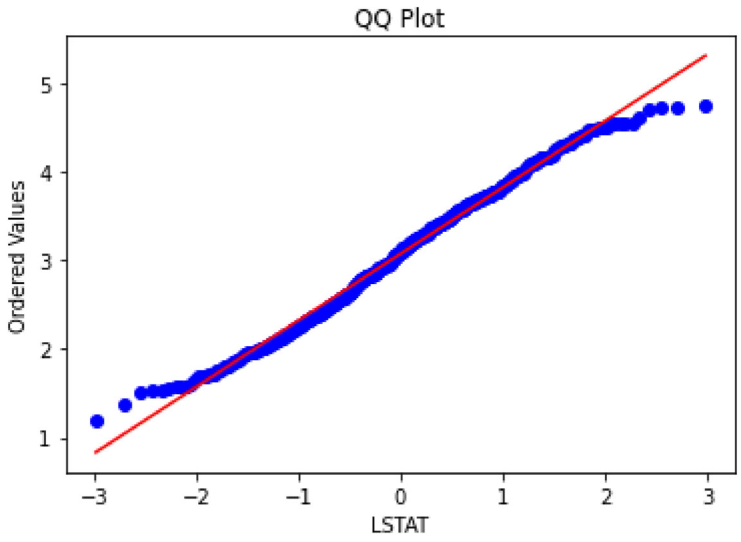
Use yeojohnson power tranformer to normalize data

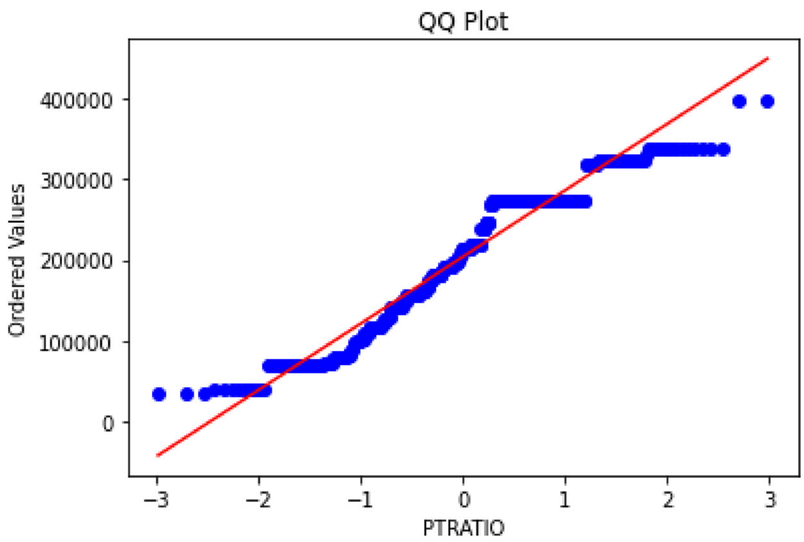
In [16]:
```
data['PTRATIO'],parameters=stats.yeojohnson(data['PTRATIO'])
data['LSTAT'],parameters=stats.yeojohnson(data['LSTAT'])
data['MEDV'],parameters=stats.yeojohnson(data['MEDV'])
```

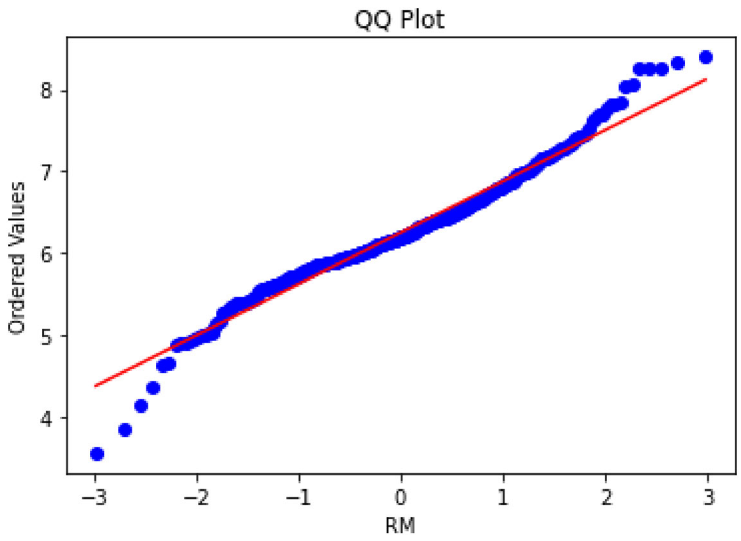
Check normalize data using QQ Plot

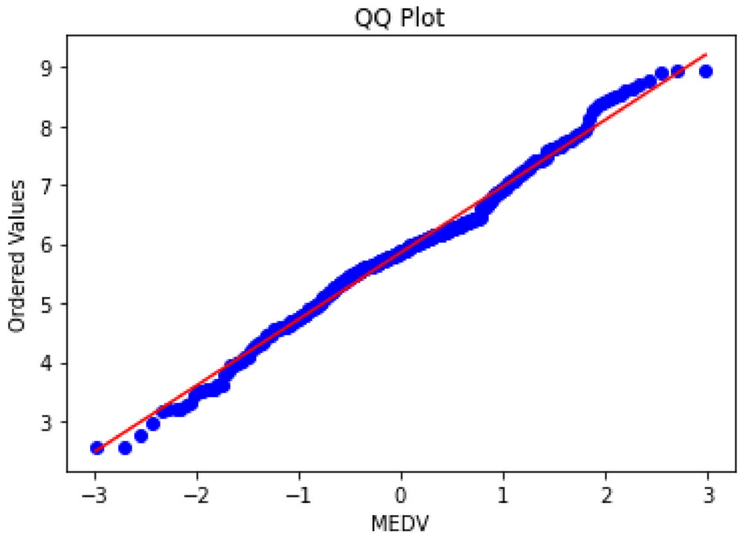
In [17]:

```
for i in data.columns:
    stats.probplot(data[i],plot=pylab)
    plt.title("QQ Plot")
    plt.xlabel(i)
    plt.show()
```









In [18]:
```
data.skew()
```

Out[18]:
```
LSTAT      -0.017960
PTRATIO    -0.209212
RM          0.165935
MEDV        0.018101
dtype: float64
```

- **Data is normally distributed**

Seperate and splitting features and label

In [19]:
```
X =data[['RM','PTRATIO','LSTAT']]
y=data['MEDV']
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=
```

Use standardization technique to scale features before applying linear regression

In [21]:
```
X_train_stand = X_train.copy()
X_test_stand = X_test.copy()

# numerical features
num_cols = ['RM','PTRATIO','LSTAT']

# apply standardization on numerical features
for i in num_cols:

    # fit on training data column
    scale = StandardScaler().fit(X_train_stand[[i]])

    # transform the training data column
    X_train_stand[i] = scale.transform(X_train_stand[[i]])

    # transform the testing data column
    X_test_stand[i] = scale.transform(X_test_stand[[i]])
```

Apply linear regression model

In [22]:
```python
lr_model = LinearRegression()
lr_model.fit(X_train_stand,y_train)
y_train_pred = lr_model.predict(X_train_stand)
y_test_pred = lr_model.predict(X_test_stand)
s1 = mean_squared_error(y_train,y_train_pred)
print("Mean Squared error of training set :%2f"%s1)
s2 = mean_squared_error(y_test,y_test_pred)
print("Mean squared error of testing set: %.2f"%s2)

```

```
Mean Squared error of training set :0.304072
Mean squared error of testing set: 0.40
```

In [23]:
```python
s = r2_score(y_train, y_train_pred)
print('R2 variance score of training set: %.2f' %s )
s = r2_score(y_test,y_test_pred)
print("R2 variance score of testing set: %.2f" %s)
N = y_test.size
p = X_train_stand.shape[1]
adjr2score = 1 - ((1-r2_score(y_test, y_test_pred))*(N - 1))/ (N - p - 1)
print("Adjusted R^2 Score %.2f" % adjr2score)
```

```
R2 variance score of training set: 0.75
R2 variance score of testing set: 0.74
Adjusted R^2 Score 0.73
```

To impove r2 score ,Polynomial regression is applied

In [24]:
```python
poly_reg = PolynomialFeatures(degree = 2)
X_train_poly = poly_reg.fit_transform(X_train_stand)
X_test_poly = poly_reg.fit_transform(X_test_stand)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_train_poly,y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```python
#predicting on training data set
y_train_predict = lin_reg_2.predict(X_train_poly)
#predicting on testing data set
y_test_predict = lin_reg_2.predict(X_test_poly)
mse_train = mean_squared_error(y_train,y_train_predict)
r2_train = r2_score(y_train,y_train_predict)
print("The model performance of training set")
print("---------------------------------------------")
print("MSE of training set is {}".format(round(mse_train,2)))
print("R2 score of training set is {}".format(round(r2_train,2)))
```

```
The model performance of training set
---------------------------------------------
MSE of training set is 0.24
R2 score of training set is 0.8
```

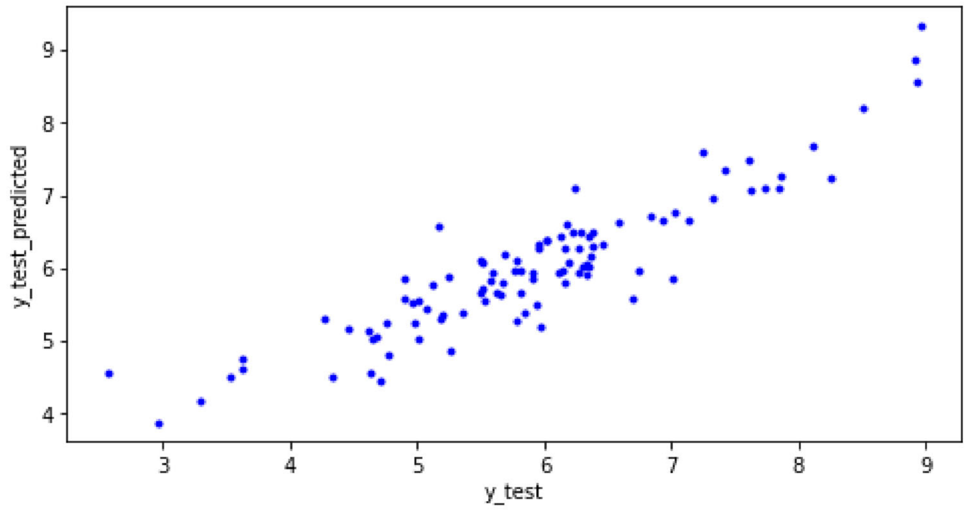
In [26]:
```python
mse_test = mean_squared_error(y_test,y_test_predict)
r2_test = r2_score(y_test,y_test_predict)

print("The model performance of training set")
print("--------------------------------------------")
print("MSE of testing set is {}".format(round(mse_test,2)))
print("R2 score of testing set is {}".format(round(r2_test,2)))
```

```
The model performance of training set
--------------------------------------------
MSE of testing set is 0.29
R2 score of testing set is 0.81
```

Plot of actual values vs predicted

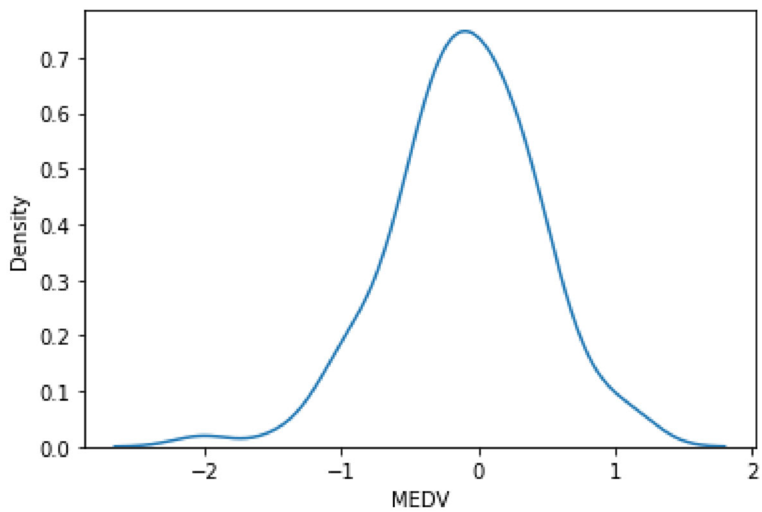
In [27]:
```python
plt.figure(figsize=(8,4))
plt.plot(y_test, y_test_predict, 'b.')
plt.xlabel("y_test")
plt.ylabel("y_test_predicted")
plt.show()
```



Difference of actual values and predicted is less

In [28]:
```python
sns.kdeplot(y_test- y_test_predict)
```

Out[28]: <AxesSubplot:xlabel='MEDV', ylabel='Density'>



Accuracy of model is 81% & Mean square error =0.29

Create pickle file

In [42]:
```
import pickle
file = open('model.pkl','wb')
pickle.dump(lr_model,file)
```

In [ ]:
```

```