INTRODUCTION

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. Our objective: Predict the House prices (MEDV) based on given features using Linear Regression.

DATA DESCRIPTION

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centers
9. RAD: index of accessibility to radial highways 10.TAX: full-value property-tax rate per 10,000 USD
10. PTRATIO: pupil-teacher ratio by town
11. Black: $1000(Bk — 0.63)^2$ where Bk is the proportion of blacks by town
12. LSTAT: % lower status of the population

APPROACH

My approach to the solution of this project is beginning with checking for null values in the dataset, then getting the info or description of the dataset i.e, whether the variables are categorical or numerical. This step is also known as Preprocessing. Following is how I went about it.

In [41]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn

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

Preprocessing for Train and Test data

In [3]:

```
#Loading the datasets

boston_train = pd.read_csv("Boston_Train.csv", index_col=0)
boston_test = pd.read_csv("Boston_Test.csv", index_col=0)
```

In [4]:

```
#Sample of train data

boston_train.head()
```

Out[4]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	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

In [5]:

```
#Sample of test data

boston_test.head()
```

Out[5]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
351	0.07950	60	1.69	0	0.411	6.579	35.9	10.7103	4	411	18.3	370.78	5.49	24.1
352	0.07244	60	1.69	0	0.411	5.884	18.5	10.7103	4	411	18.3	392.33	7.79	18.6
353	0.01709	90	2.02	0	0.410	6.728	36.1	12.1265	5	187	17.0	384.46	4.50	30.1
354	0.04301	80	1.91	0	0.413	5.663	21.9	10.5857	4	334	22.0	382.80	8.05	18.2
355	0.10659	80	1.91	0	0.413	5.936	19.5	10.5857	4	334	22.0	376.04	5.57	20.6

In [6]:

```
#Checking for null values in our train data

boston_train.isna().sum()
```

Out[6]:

```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

In [7]:
```python
#Checking for null values in our test data

boston_test.isna().sum()
```

Out[7]:
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

This implies that there are no null values for both train and test datasets.

In [8]:
```python
#Shape of our training dataset
print("Shape of train data:", boston_train.shape)

#Shape of our test dataset
print("Shape of test data:", boston_test.shape)
```

```
Shape of train data: (351, 14)
Shape of test data: (155, 14)
```

In [9]:
```python
#Information about the train dataset features

boston_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 351 entries, 0 to 350
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     351 non-null    float64
 1   zn       351 non-null    float64
 2   indus    351 non-null    float64
 3   chas     351 non-null    int64  
 4   nox      351 non-null    float64
 5   rm       351 non-null    float64
 6   age      351 non-null    float64
 7   dis      351 non-null    float64
 8   rad      351 non-null    int64  
 9   tax      351 non-null    int64  
 10  ptratio  351 non-null    float64
 11  black    351 non-null    float64
 12  lstat    351 non-null    float64
 13  medv     351 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 41.1 KB
```

In [10]:

```python
#Information about the test dataset features

boston_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 155 entries, 351 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     155 non-null    float64
 1   zn       155 non-null    int64  
 2   indus    155 non-null    float64
 3   chas     155 non-null    int64  
 4   nox      155 non-null    float64
 5   rm       155 non-null    float64
 6   age      155 non-null    float64
 7   dis      155 non-null    float64
 8   rad      155 non-null    int64  
 9   tax      155 non-null    int64  
 10  ptratio  155 non-null    float64
 11  black    155 non-null    float64
 12  lstat    155 non-null    float64
 13  medv     155 non-null    float64
dtypes: float64(10), int64(4)
memory usage: 18.2 KB
```

In [11]:

```python
boston_train.describe()
```

Out[11]:

	crim	zn	indus	chas	nox	rm	age	d
count	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.0000
mean	0.401659	15.327635	8.435670	0.076923	0.510737	6.403900	60.817949	4.4208
std	0.641716	25.605040	6.088947	0.266850	0.102256	0.676424	28.393094	1.9686
min	0.006320	0.000000	0.460000	0.000000	0.385000	4.903000	2.900000	1.3216
25%	0.057845	0.000000	4.025000	0.000000	0.437450	5.949500	36.150000	2.7685
50%	0.132620	0.000000	6.200000	0.000000	0.493000	6.266000	62.000000	4.0952
75%	0.404865	22.000000	10.010000	0.000000	0.544000	6.733000	88.450000	5.8718
max	4.097400	100.000000	25.650000	1.000000	0.871000	8.725000	100.000000	9.2229

In [12]:

```python
boston_test.describe()
```

Out[12]:

	crim	zn	indus	chas	nox	rm	age	d
count	155.000000	155.000000	155.000000	155.000000	155.000000	155.000000	155.000000	155.0000
mean	10.886843	2.387097	17.253484	0.051613	0.654239	6.014555	86.140645	2.3778
std	12.842318	13.294070	3.973223	0.221961	0.076748	0.687848	17.844278	1.6786

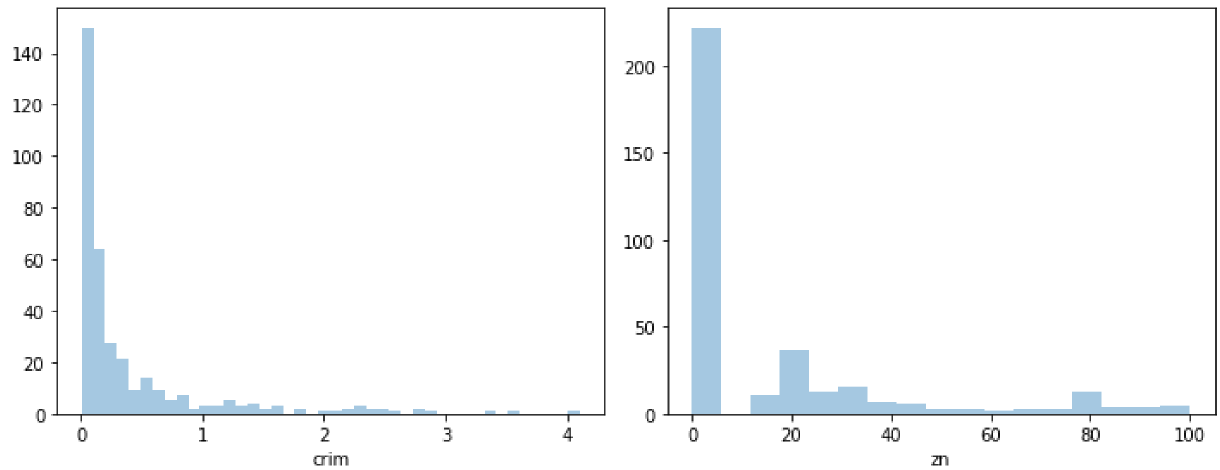
	crim	zn	indus	chas	nox	rm	age	d
min	0.017090	0.000000	1.690000	0.000000	0.410000	3.561000	18.500000	1.1296(
25%	4.385535	0.000000	18.100000	0.000000	0.591000	5.695000	81.900000	1.6432!
50%	7.839320	0.000000	18.100000	0.000000	0.679000	6.112000	92.600000	2.0048(
75%	13.441000	0.000000	18.100000	0.000000	0.713000	6.414000	98.250000	2.5016(
max	88.976200	90.000000	27.740000	1.000000	0.770000	8.780000	100.000000	12.1265(

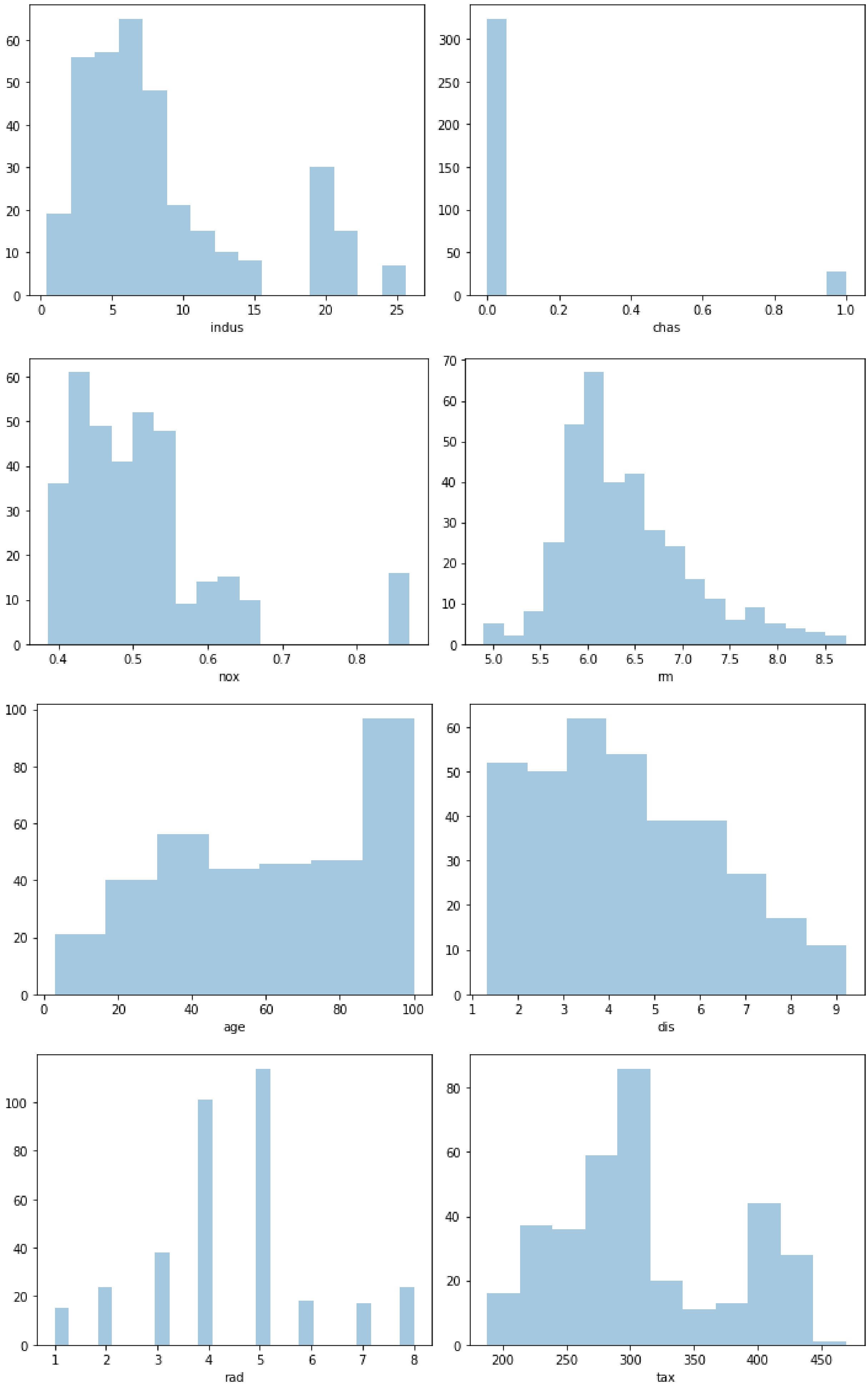
VISUALIZATION

Further I plotted histograms for all the variables, and found our response variable to possibly follow normal distribution. This step is also known as EDA. This was done for both test and train data.

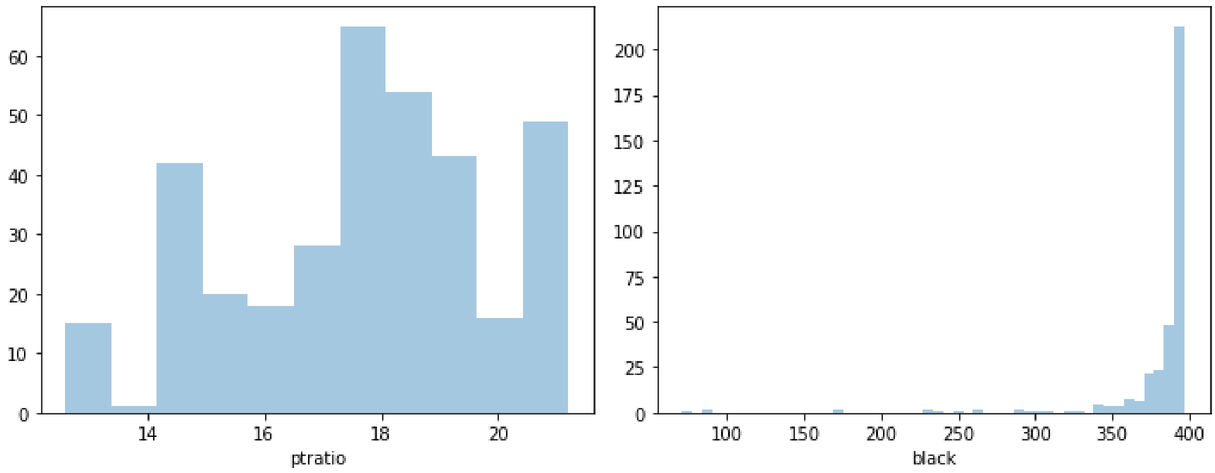
In [42]:

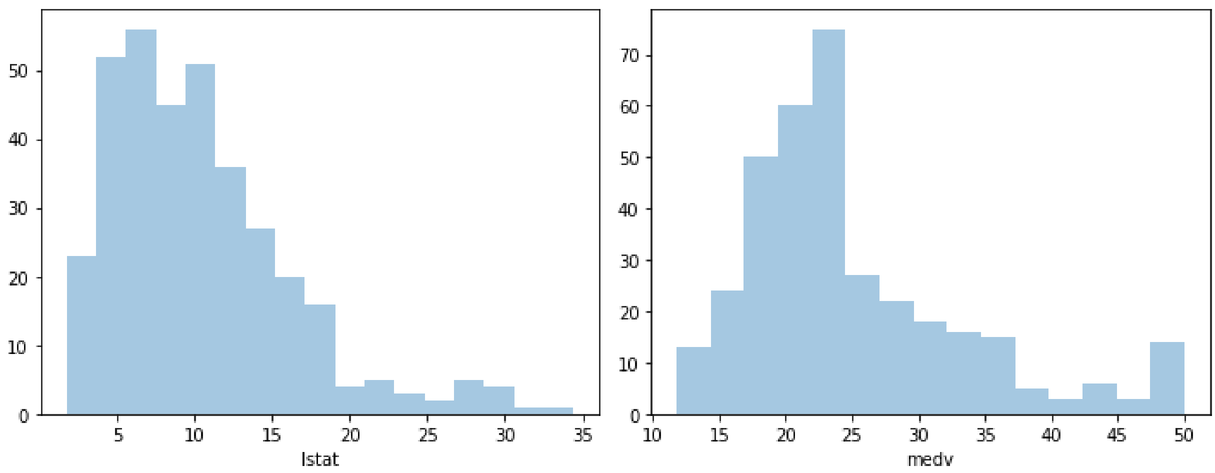
```
def plot_graph(boston_train):
    num_col = boston_train.select_dtypes(include=['number']).columns.tolist()
    boston_train = boston_train[num_col]

    for i in range(0,len(num_col),2):
        if len(num_col) > i+1:
            plt.figure(figsize=(10,4))
            plt.subplot(121)
            sns.distplot(boston_train[num_col[i]], kde=False)
            plt.subplot(122)
            sns.distplot(boston_train[num_col[i+1]], kde=False)
            plt.tight_layout()
            plt.show()

        else:
            sns.distplot(boston_train[num_col[i]], kde=False)
plot_graph(boston_train)
plt.show()
```





0 10 20 30 40 50 60
0 5 10 15 20 25
indus
0 50 100 150 200 250 300
0.0 0.2 0.4 0.6 0.8 1.0
chas
0 10 20 30 40 50 60
0.4 0.5 0.6 0.7 0.8
nox
0 10 20 30 40 50 60 70
5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
rm
0 20 40 60 80 100
0 20 40 60 80 100
age
0 10 20 30 40 50 60
1 2 3 4 5 6 7 8 9
dis
0 20 40 60 80 100
1 2 3 4 5 6 7 8
rad
0 20 40 60 80
200 250 300 350 400 450
tax

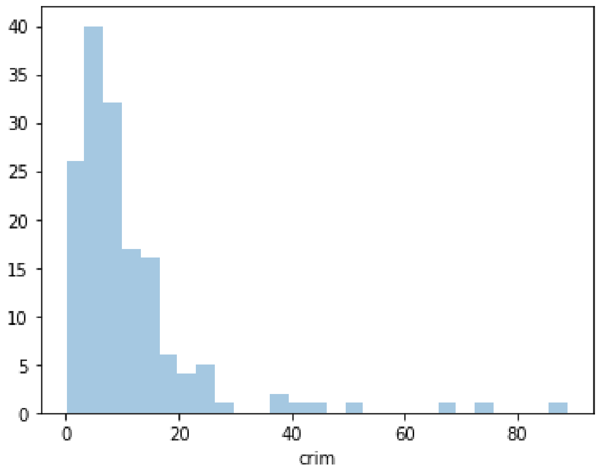


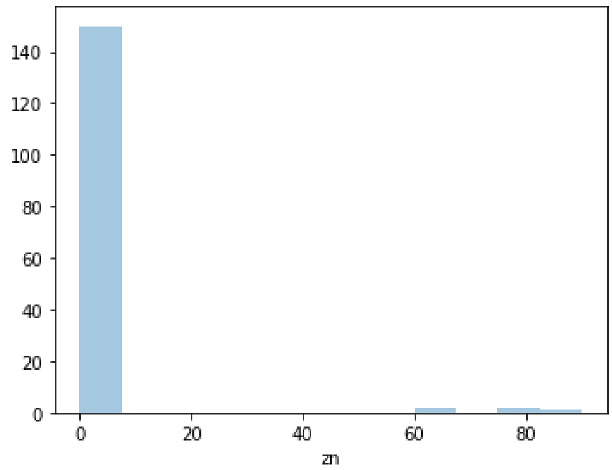


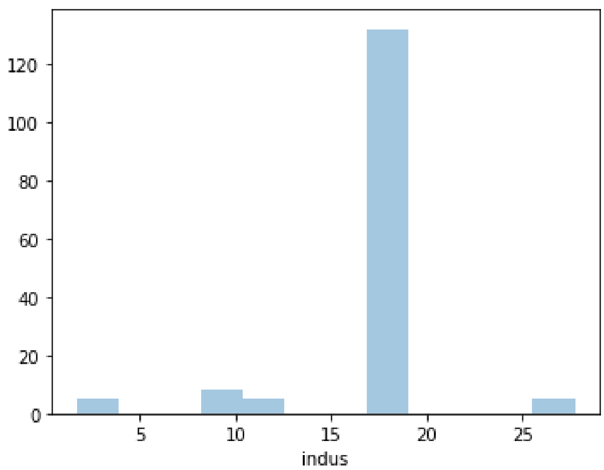
From this graph we can see that medv, which is our response variable, may be normally distributed.

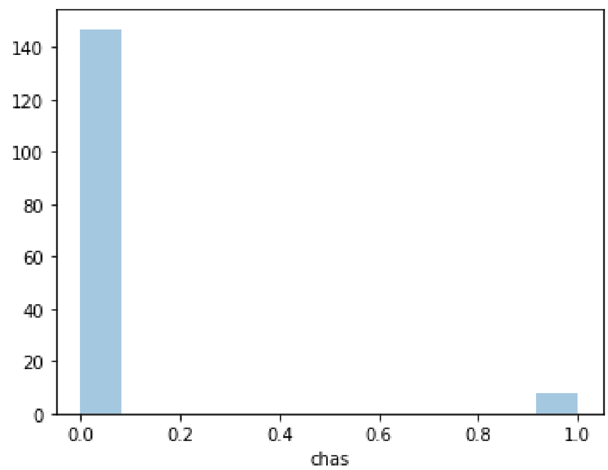
In [14]:

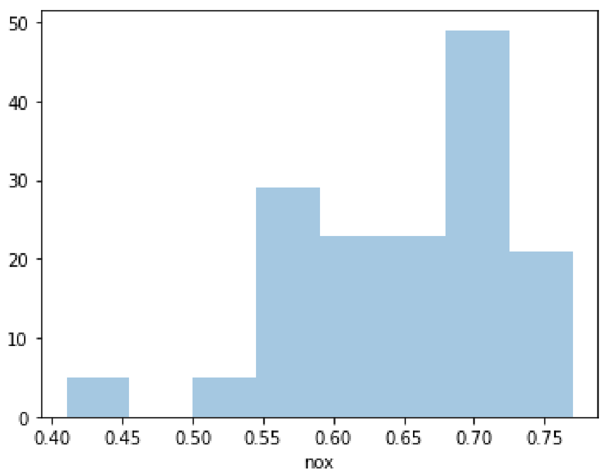
```
def plot_graph(boston_test):
    num_col = boston_test.select_dtypes(include=['number']).columns.tolist()
    boston_test = boston_test[num_col]

    for i in range(0,len(num_col),2):
        if len(num_col) > i+1:
            plt.figure(figsize=(10,4))
            plt.subplot(121)
            sns.distplot(boston_test[num_col[i]], kde=False)
            plt.subplot(122)
            sns.distplot(boston_test[num_col[i+1]], kde=False)
            plt.tight_layout()
            plt.show()

        else:
            sns.distplot(boston_test[num_col[i]], kde=False)
plot_graph(boston_test)
plt.show()
```

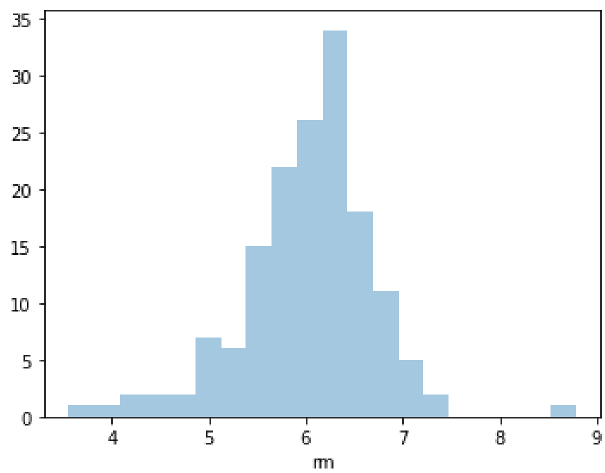

crim

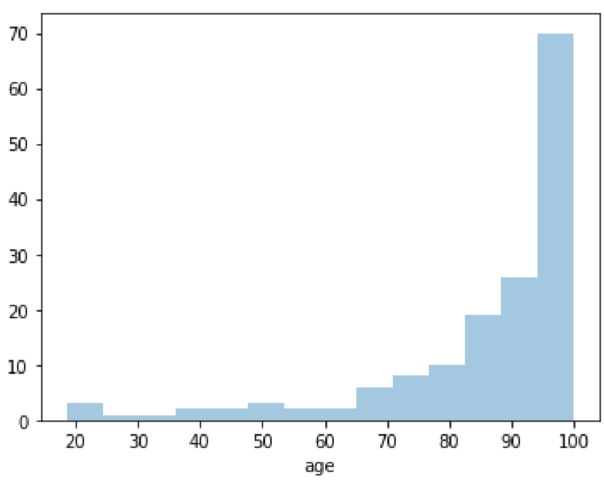

zn

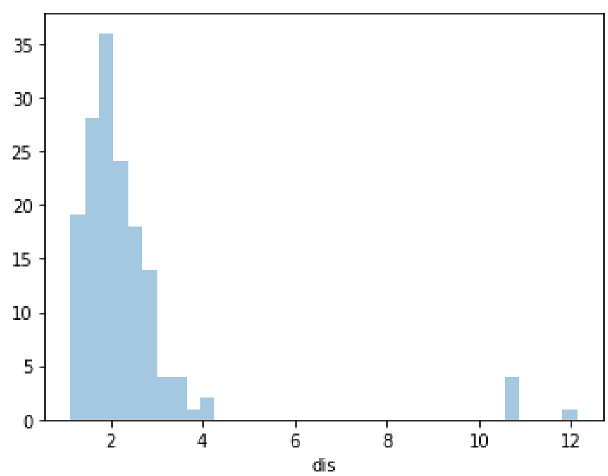

indus

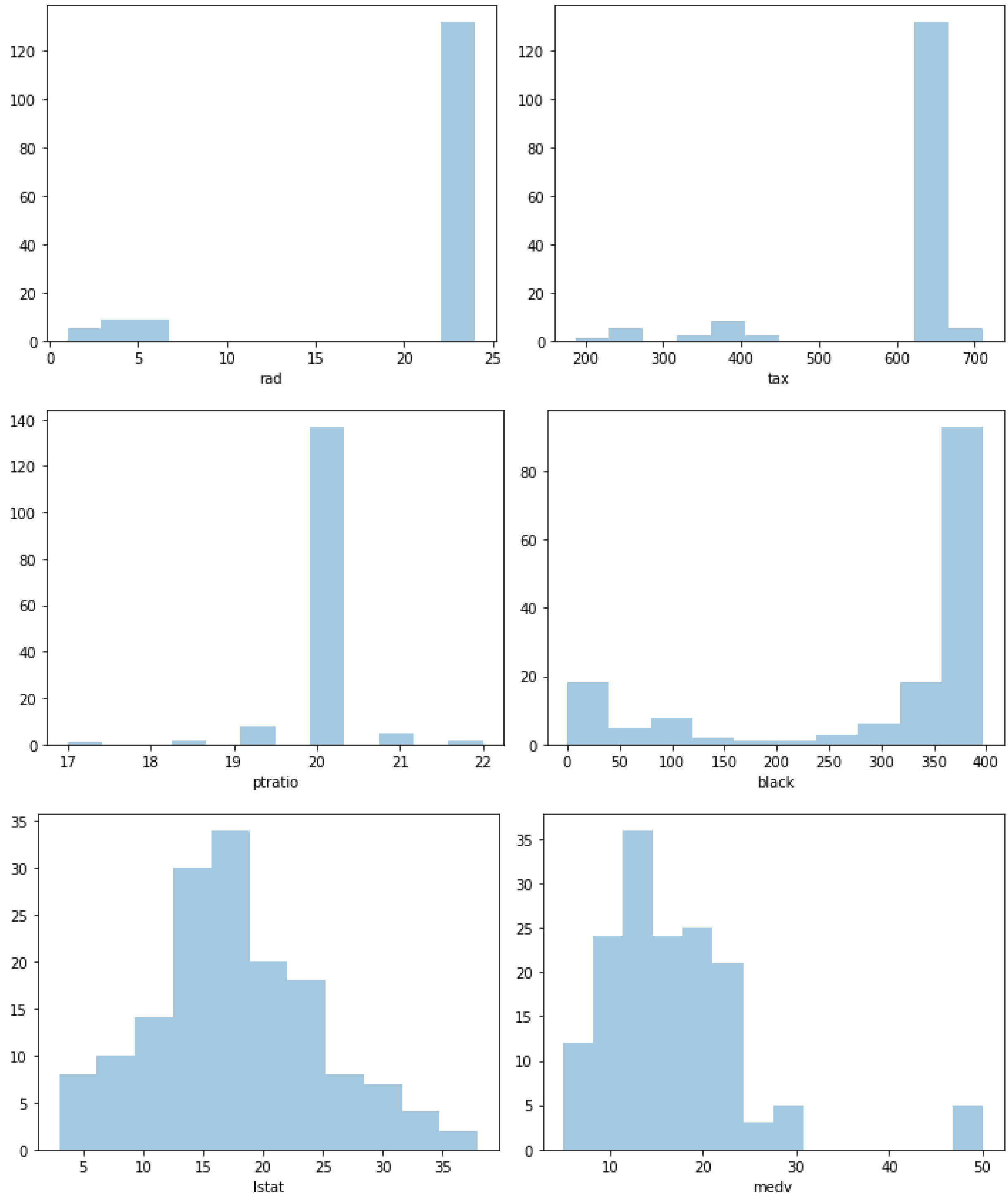

chas


nox


rm


age

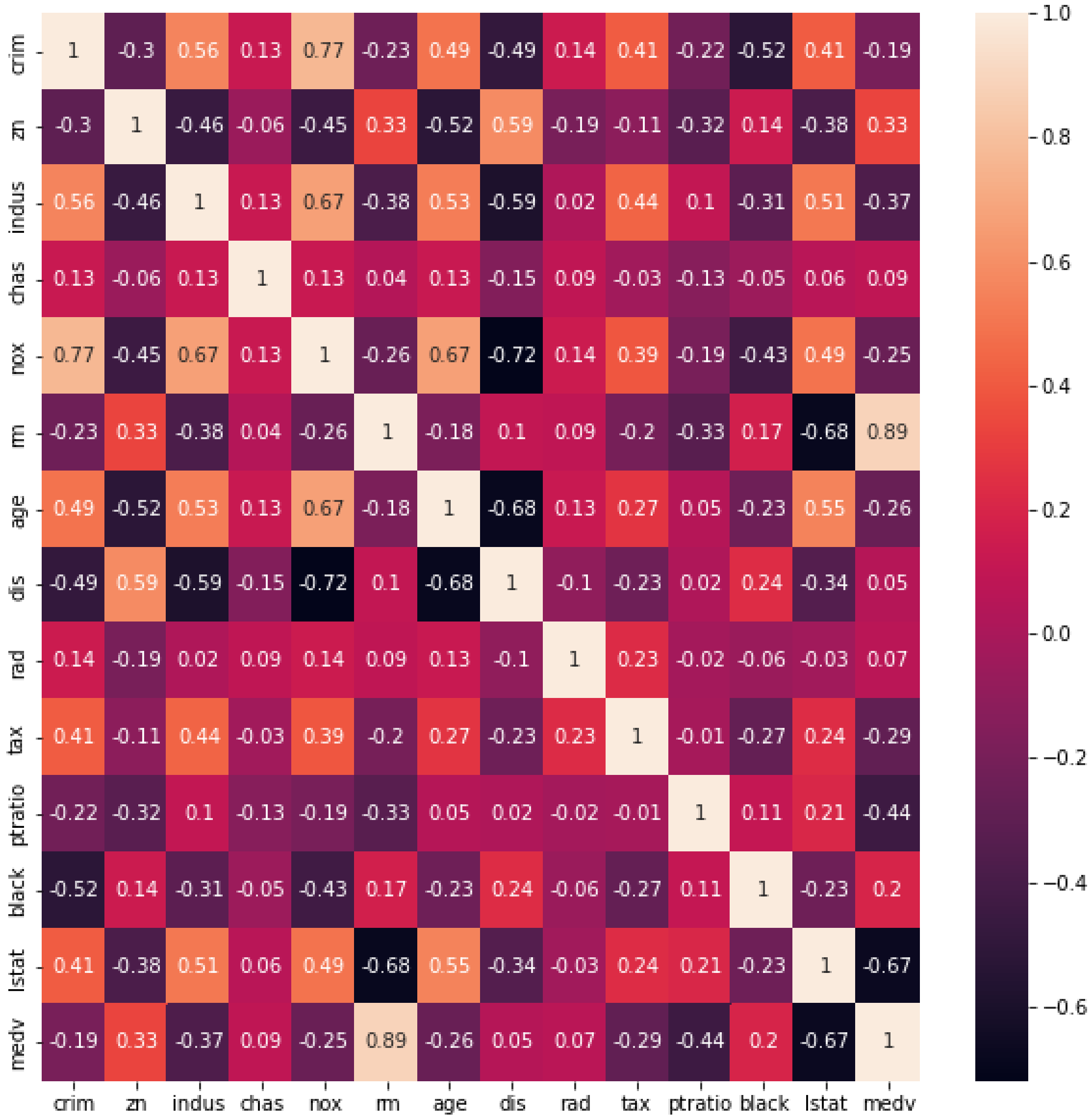

dis



Heat maps

In [15]:

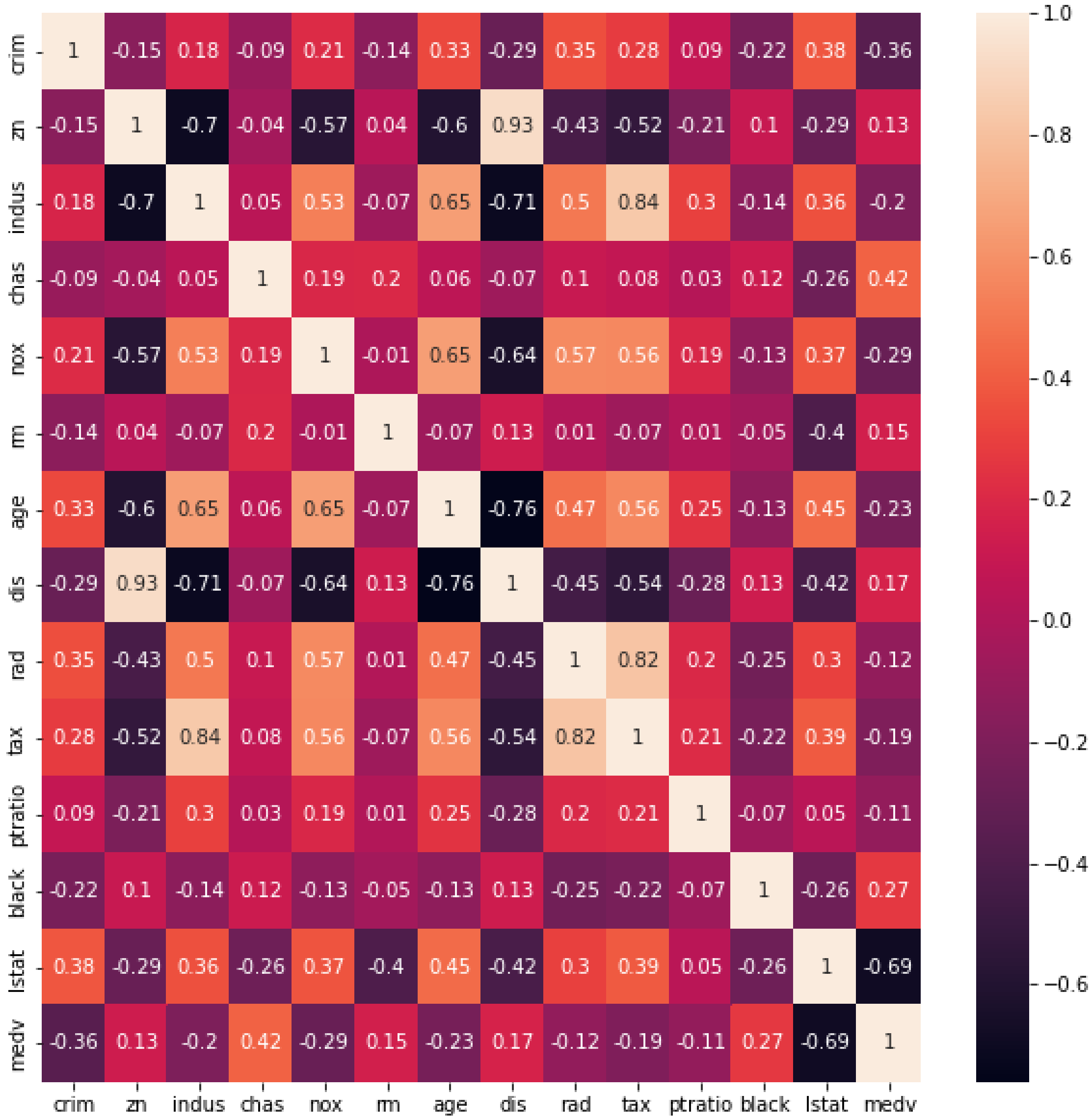
```
#Heatmap to find correlation between our target variable and all the other variables

plt.figure(figsize = (10,10))
sns.heatmap(boston_train.corr().round(2), annot = True)
plt.show()
```



Looking at the heatmap, we can derive the conclusion that there is a strong positive correlation between average number of rooms per dwelling and our response variable medv i.e, Median value of owner-occupied homes in $1000's, and a high negative correlation between percentage of lower status of the population and medv (target variable).

In [16]:

```
#Heatmap to find correlation between our target variable and all the other variables

plt.figure(figsize = (10,10))
sns.heatmap(boston_test.corr().round(2), annot = True)
plt.show()
```

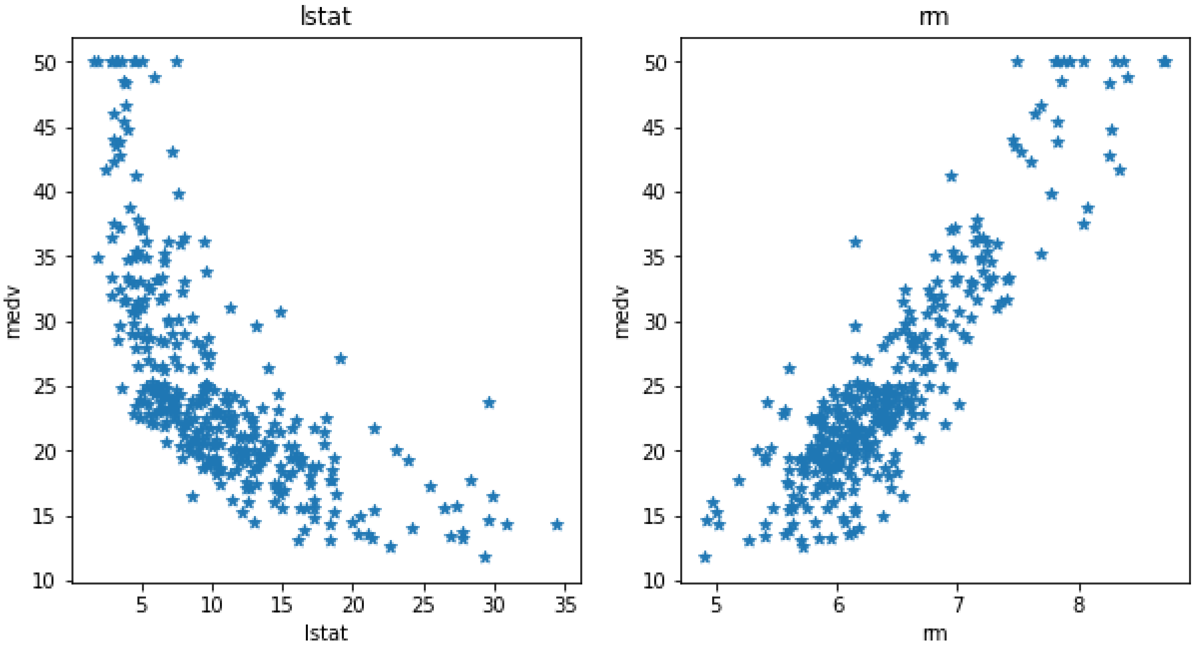


Looking at the heatmap, we can derive the conclusion that there is a high negative correlation between percentage of lower status of the population and medv which is our target variable.

Since there was a positive and negative correlation between our target variable and rooms per dwelling and lower status of the population respectively, we can visually represent these variables and derive our observations from them.

In [17]:

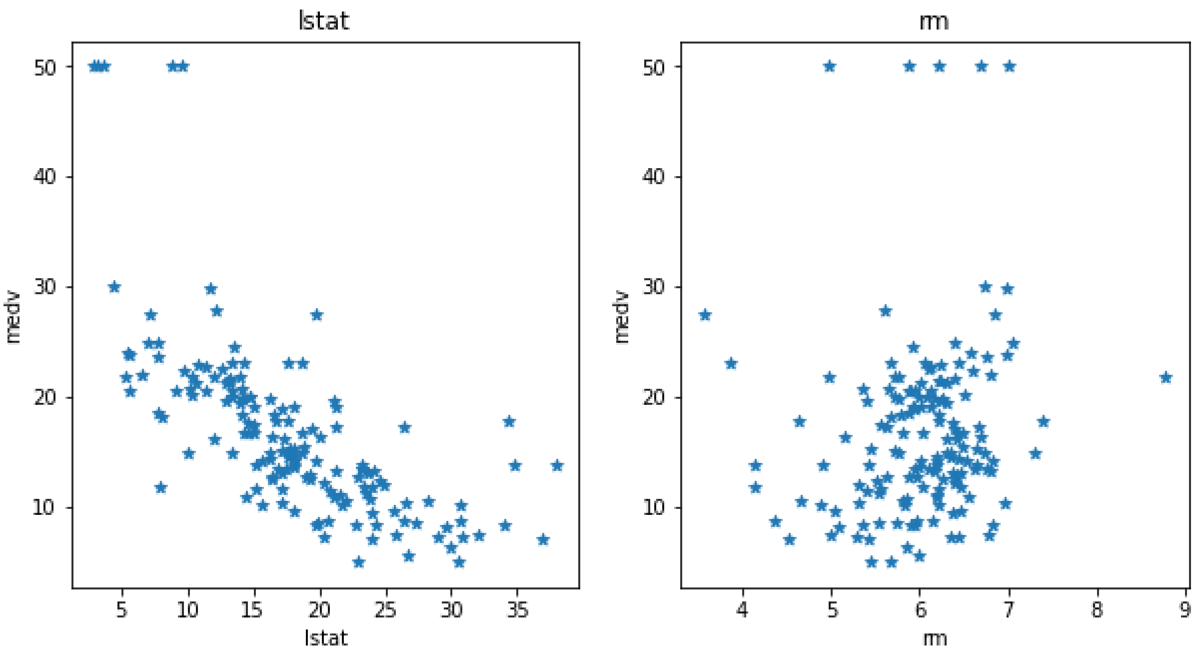
```
#Plotting of graphs for train data

plt.figure(figsize=(10, 5))
attributes = ['lstat', 'rm']
response_var = boston_train['medv']

for i, col in enumerate(attributes):
    plt.subplot(1, len(attributes) , i+1)
    x = boston_train[col]
    y = response_var
    plt.scatter(x, y, marker='*')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('medv')
```



Here, we observe that as the median price increases, the rm value increases as well, implying a direct relationship. Whereas, with the increase in the value of lstat, median price decreases, implying an inverse relationship.

In [18]:

```
#Plotting of graphs for test data

plt.figure(figsize=(10, 5))
attributes = ['lstat', 'rm']
response_var = boston_test['medv']

for i, col in enumerate(attributes):
    plt.subplot(1, len(attributes) , i+1)
    x = boston_test[col]
    y = response_var
    plt.scatter(x, y, marker='*')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('medv')
```



Here, we observe that the graphs looks inconclusive for rm, but we can say that lstat is inversely proportional to medv with some outliers.

ALGORITHMS

The major aim of in this project is to predict the house prices based on the features. Therefore the algorithm used is Linear Regression.

In [65]:
```
X = pd.DataFrame(np.c_[boston_train['lstat'], boston_train['rm']], columns = ['lstat
Y = boston_train['medv']
```

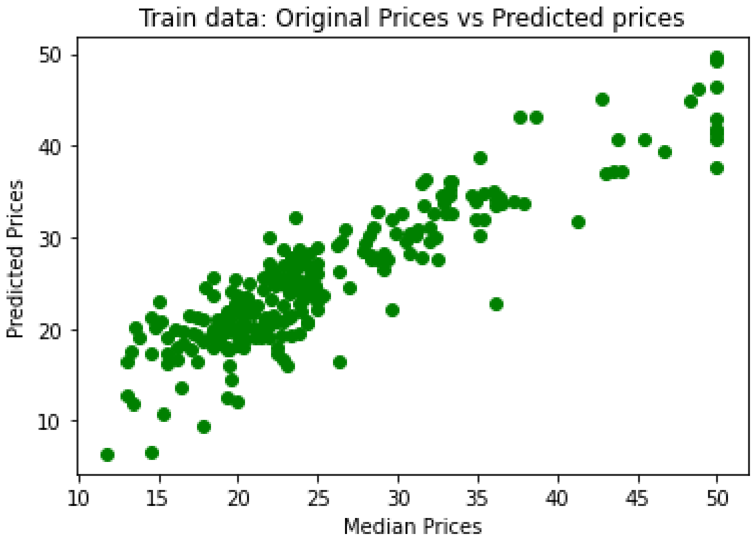
In [66]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33)
```

In [67]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()
model.fit(X_train, Y_train)
```

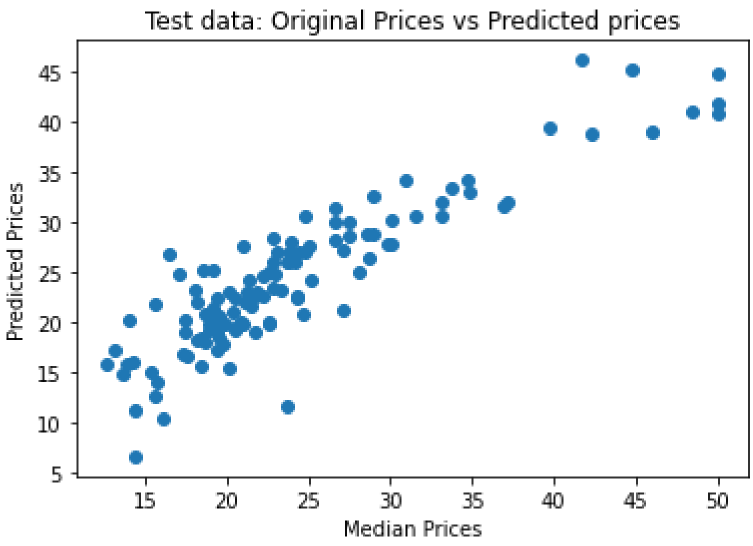
Out[67]:
```
LinearRegression()
```

In [68]:
```
y_test_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
```

In [74]:
```
plt.scatter(Y_train, y_train_pred, c = 'g')
plt.xlabel('Median Prices')
plt.ylabel('Predicted Prices')
plt.title('Train data: Original Prices vs Predicted prices')
plt.show()
```



In [75]:
```
plt.scatter(Y_test, y_test_pred)
plt.xlabel('Median Prices')
plt.ylabel('Predicted Prices')
plt.title('Test data: Original Prices vs Predicted prices')
plt.show()
```



MODEL EVALUATION

In [69]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

#Train data
rmse_train = (np.sqrt(mean_squared_error(Y_train, y_train_pred)))
absolute_train = (mean_absolute_error(Y_train, y_train_pred))
r2_train = r2_score(Y_train, y_train_pred)

print("Results:")
print('RMSE is {}'.format(rmse_train))
print('Absolute error score is {}'.format(absolute_train))
print('R2 score is {}'.format(r2_train))
```

```
Results:
RMSE is 3.729476096711187
Absolute error score is 2.9000280329180623
R2 score is 0.8086363195008512
```

In [70]:
```
#Test data
rmse_test = (np.sqrt(mean_squared_error(Y_test, y_test_pred)))
absolute_test = (mean_absolute_error(Y_test, y_test_pred))
r2_test = r2_score(Y_test, y_test_pred)

print("Results:")
print('RMSE is {}'.format(rmse_test))
print('Absolute error score is {}'.format(absolute_test))
print('R2 score is {}'.format(r2_test))
```

```
Results:
RMSE is 3.636796462435612
Absolute error score is 2.746963433637365
R2 score is 0.8042978653000581
```

In [71]:
```
#Train data
print('Model performance for train data:')
print('Accuracy for this model is', r2_train*100, '%')

#Test data
print('\nModel performance for test data:')
print('Accuracy for this model is', r2_test*100, '%')
```

```
Model performance for train data:
```

```
Accuracy for this model is 80.86363195008512 %

Model performance for test data:
Accuracy for this model is 80.42978653000581 %
```

CONCLUSION

Since R2 is nearer to 1 and there is not much difference between R2 value of Train set and Test set, we can say model is not overfitted.

FUTURE WORK

We can use multiple models and then compare the results to derive more accurate results.

REFERENCES

1) https://www.w3schools.com/python/python_ml_linear_regression.asp
2) https://www.geeksforgeeks.org/linear-regression-python-implementation/