Boston Housing with Linear Regression ** With this data our objective is create a model using linear regression to predict the houses price **

The data contains the following columns:

> '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': nitrogen 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 mean of distances to five Boston employment centres.
> 'rad': index of accessibility to radial highways.
> 'tax': full-value property-tax rate perb$10,000$.
> 'ptratio': pupil-teacher ratio by town
> 'black': 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
> 'lstat': lower status of the population (percent).
> 'medv': median value of owner-occupied homes in $$1000s

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

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

In [3]:
```
from sklearn.datasets import load_boston
boston =load_boston()
```

In [6]:
```
data = pd.DataFrame(boston.data)
data
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [7]:
```
data.columns = boston.feature_names

data.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [8]:
```
#Adding target variable to dataframe
data['PRICE'] = boston.target
```

In [11]:
```
data
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [9]:
```
data.isnull().sum()
```

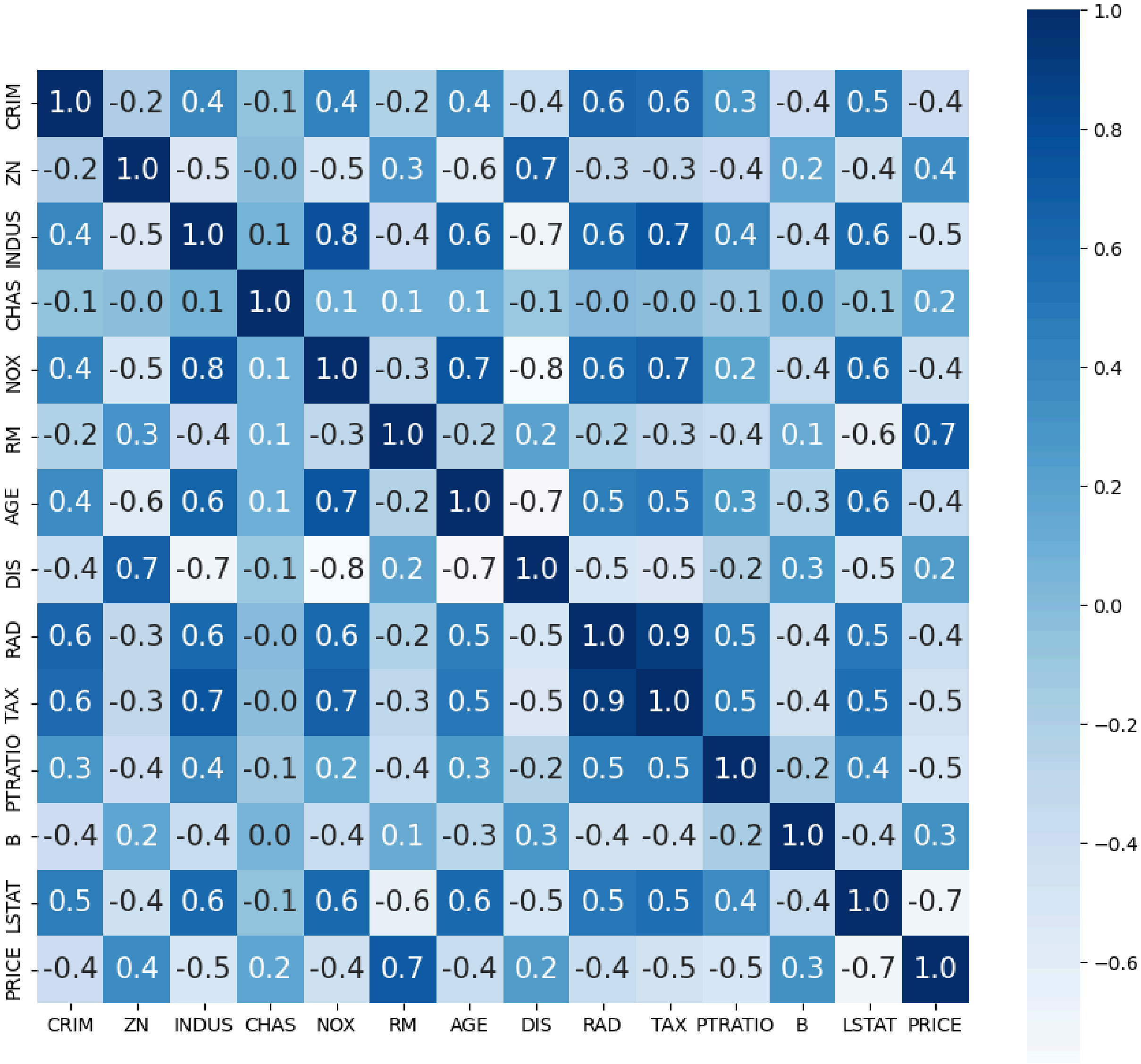
Out[9]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

In [10]:
```
# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

Out[10]:
```
(14, 14)
```

In [15]:
```python
# Plotting the heatmap of correlation between features
plt.figure(figsize=(11,10))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'size':15}, cmap='Blue
```

Out[15]: <AxesSubplot:>



In [11]:
```python
#Split dependent variable and independent variables
x = data.drop(['PRICE'], axis = 1)
y = data['PRICE']
```

In [12]:
```python
#splitting data to training and testing dataset.
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest =train_test_split(x, y, test_size =0.2,random_state = 0)
```

In [13]:
```python
#Use linear regression( Train the Machine ) to Create Model
import sklearn
from sklearn.linear_model import LinearRegression
# Create a Linear regressor
lm = LinearRegression()
# Train the model using the training sets
model=lm.fit(xtrain, ytrain)
```

In [14]:
```
xtrain
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
220	0.35809	0.0	6.20	1.0	0.507	6.951	88.5	2.8617	8.0	307.0	17.4	391.70	9.71
71	0.15876	0.0	10.81	0.0	0.413	5.961	17.5	5.2873	4.0	305.0	19.2	376.94	9.88
240	0.11329	30.0	4.93	0.0	0.428	6.897	54.3	6.3361	6.0	300.0	16.6	391.25	11.38
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
417	25.94060	0.0	18.10	0.0	0.679	5.304	89.1	1.6475	24.0	666.0	20.2	127.36	26.64
...	...	...	...	...	...	...	...	...	...	...	...	...	...
323	0.28392	0.0	7.38	0.0	0.493	5.708	74.3	4.7211	5.0	287.0	19.6	391.13	11.74
192	0.08664	45.0	3.44	0.0	0.437	7.178	26.3	6.4798	5.0	398.0	15.2	390.49	2.87
117	0.15098	0.0	10.01	0.0	0.547	6.021	82.6	2.7474	6.0	432.0	17.8	394.51	10.30
47	0.22927	0.0	6.91	0.0	0.448	6.030	85.5	5.6894	3.0	233.0	17.9	392.74	18.80
172	0.13914	0.0	4.05	0.0	0.510	5.572	88.5	2.5961	5.0	296.0	16.6	396.90	14.69

404 rows × 13 columns

In [20]:
```
#Predict the y_pred for all values of train_x and test_x
ytrain_pred = lm.predict(xtrain)
ytest_pred = lm.predict(xtest)
```

In [21]:
```
#Evaluate the performance of Model for train_y and test_y
df1=pd.DataFrame(ytrain_pred,ytrain)
df2=pd.DataFrame(ytest_pred,ytest)
df1
```

Out[21]:

	0
PRICE	
26.7	32.556927
21.7	21.927095
22.0	27.543826
22.9	23.603188
10.4	6.571910
...	...
18.5	19.494951
36.4	33.326364
19.2	23.796208
16.6	18.458353
23.1	23.249181

404 rows × 1 columns

In [22]:
```
df2
```

Out[22]:

	0
PRICE	
22.6	24.889638
50.0	23.721411
23.0	29.364999
8.3	12.122386
21.2	21.443823
...	...
24.7	25.442171
14.1	15.571783
18.7	17.937195
28.1	25.305888
19.8	22.373233

102 rows × 1 columns

Model Evaluation

R^2 : It is a measure of the linear relationship between X and Y. It is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.

Adjusted R^2 :The adjusted R-squared compares the explanatory power of regression models that contain different numbers of predictors.

MAE : It is the mean of the absolute value of the errors. It measures the difference between two continuous variables, here actual and predicted values of y.

MSE: The mean square error (MSE) is just like the MAE, but squares the difference before summing them all instead of using the absolute value.

RMSE: The mean square error (MSE) is just like the MAE, but squares the difference before summing them all instead of using the absolute value.

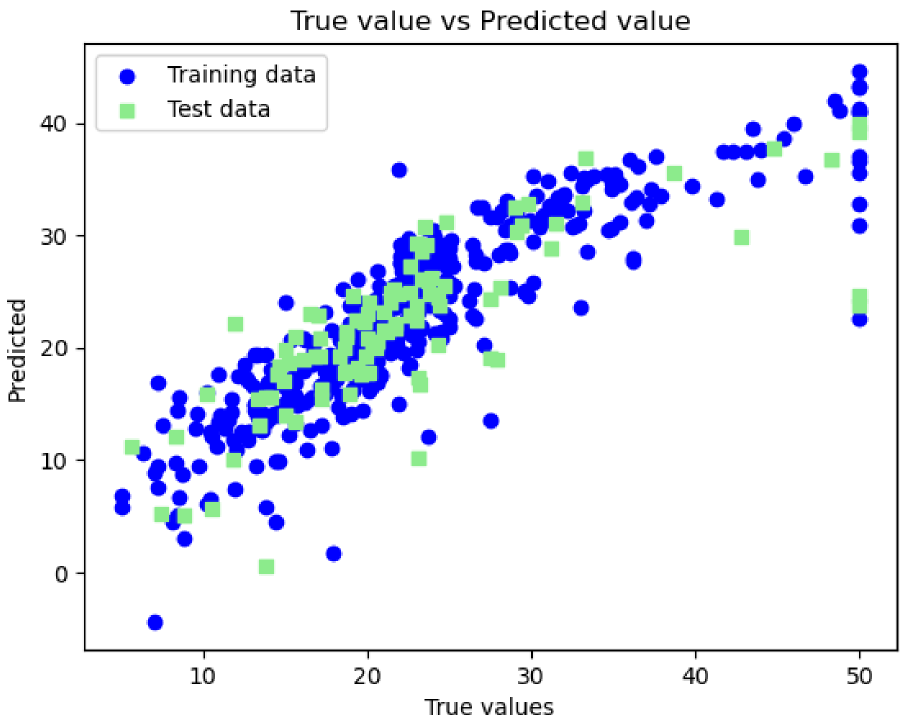
In [23]:
```
#Calculate Mean Square error for train_y and test_y
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(ytest, ytest_pred)
print('MSE on test data:',mse)
mse1 = mean_squared_error(ytrain_pred,ytrain)
print('MSE on training data:',mse1)
```

```
MSE on test data: 33.44897999767649
MSE on training data: 19.326470203585725
```

In [24]:
```
#from sklearn.metrics import mean_squared_error
#def linear_metrics():
r2 = lm.score(xtest, ytest)
rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
print('r-squared: {}'.format(r2))
print('---------------------------------------')
print('root mean squared error: {}'.format(rmse))
```

```
r-squared: 0.5892223849182514
---------------------------------------
root mean squared error: 5.783509315085132
```

In [25]:
```python
#Plotting the linear regression model
plt.scatter(ytrain ,ytrain_pred,c='blue',marker='o',label='Training data')
plt.scatter(ytest,ytest_pred ,c='lightgreen',marker='s',label='Test data')
plt.xlabel('True values')
plt.ylabel('Predicted')
plt.title("True value vs Predicted value")
plt.legend(loc= 'upper left') #plt.hlines(y=0,xmin=0,xmax=50)
plt.plot()
plt.show()
```



In [26]:
```python
testdata=[[0.00632,18.0,2.31,0.0,0.538,6.575,65.2,4.0900,1.0,296.0,15.3,396.90,4.98]]
```

In [27]:
```python
test_pred = lm.predict(testdata)
test_pred
```

Out[27]:
```
array([30.49949836])
```