In [1]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\smegh\anaconda3\lib\site-packages
(1.6.2)
Requirement already satisfied: scipy in c:\users\smegh\anaconda3\lib\site-packages
(from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\smegh\anaconda3\lib\site-packages
(from xgboost) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
import numpy as np #used for making arrays
import pandas as pd #used for making dataframe
import matplotlib.pyplot as plt #used for plotting graphs
import seaborn as sns
import sklearn.datasets #its a machine learning library from which we can get few da
from sklearn.model_selection import train_test_split #this function is used to our o
from xgboost import XGBRegressor
from sklearn import metrics #used for evaluating our model
```

Importing the boston house price dataset

In [3]:

```
house_price_dataset=sklearn.datasets.load_boston()
```

```
C:\Users\smegh\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWa
rning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and wi
ll be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

In [4]:
```
house_price_dataset
```

Out[4]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
```

```
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n---------------------------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506 \n\n    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.\n\n    :Attribute Information (in order):\n        - CRIM     per capita crime rate by town\n        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS    proportion of non-retail business acres per town\n        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX      nitric oxides concentration (parts per 10 million)\n        - RM       average number of rooms per dwelling\n        - AGE      proportion of owner-occupied units built prior to 1940\n        - DIS      weighted distances to five Boston employment centres\n        - RAD      index of accessibility to radial highways\n        - TAX      full-value property-tax rate per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town\n        - LSTAT    % lower status of the population\n        - MEDV     Median value of owner-occupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are used in the table on\npages 244-261 of the latter.\n\nThe Boston house-price data has been used in many machine learning papers that address regression\nproblems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.\n",
 'filename': 'boston_house_prices.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [5]:

```
#Loading the dataset to a pandas dataframe
house_price_dataframe=pd.DataFrame(house_price_dataset.data,columns=house_price_data
```

In [6]:

```
#print first five rows of dataframe
house_price_dataframe.head()
```

Out[6]:

	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

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 10,000usd, 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.

In [7]:

```
#add the target(price) column to the dataframe
house_price_dataframe['price']=house_price_dataset.target
```

In [8]:

```
house_price_dataframe.head()
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	p
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 [9]:

```
#checking the number of rows and columns in the dataframe
house_price_dataframe.shape
```

Out[9]:

```
(506, 14)
```

In [10]:

```
#check for missing values
house_price_dataframe.isnull().sum()
```

Out[10]:

```
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 [11]:

```
#statistical measures of the dataset
house_price_dataframe.describe()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	D
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.7950
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.1057
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.1296
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.1001
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.2074
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.1884

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	D
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.1265

Understanding the correlation between various features in the dataset

1.Postive Correlation

2.Negative Correlation

In [12]:

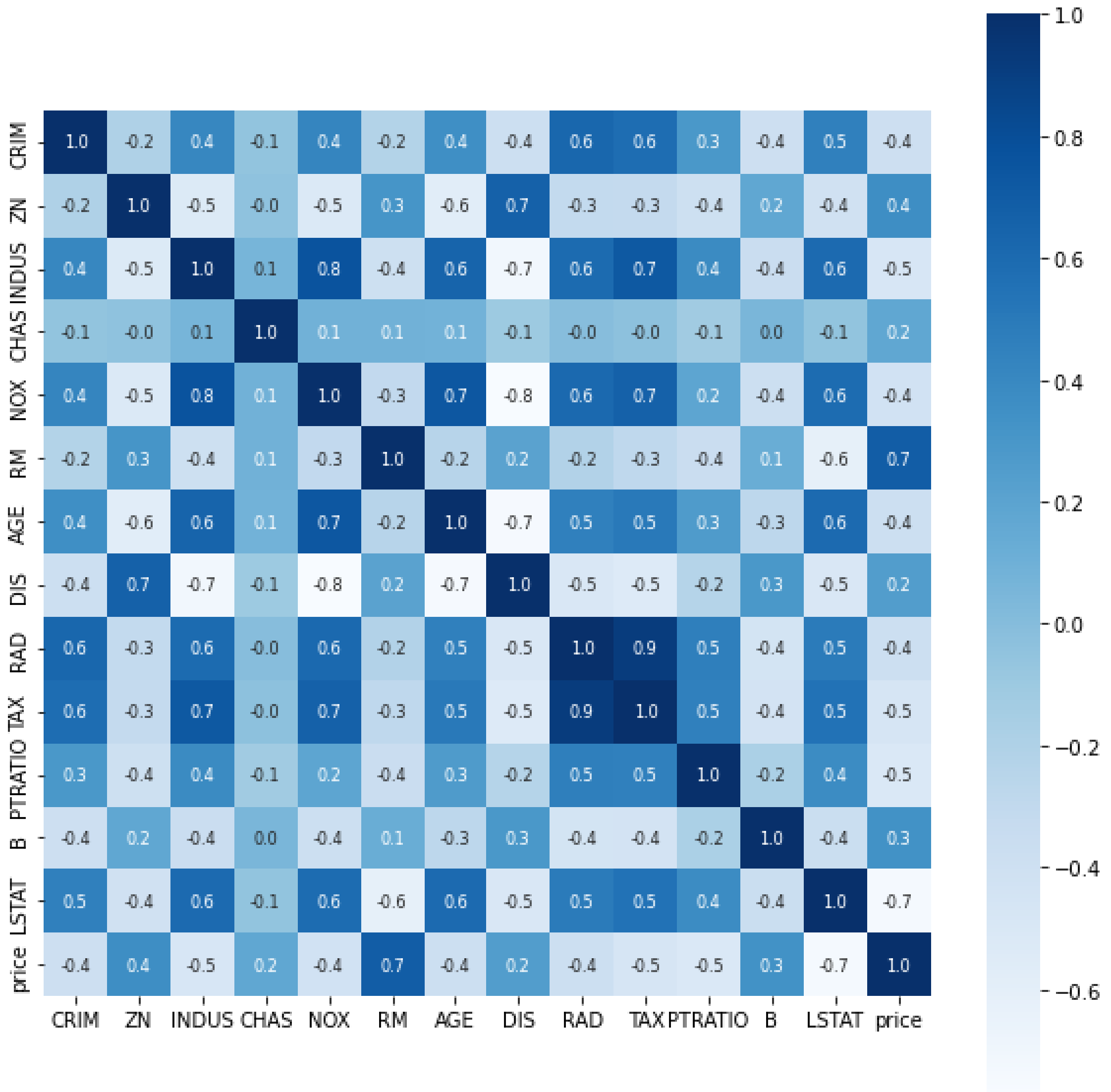
```
#correlation basically represents the relationship between two variables
correlation=house_price_dataframe.corr()
```

In [13]:

```
#constructing a heatmap to understand the correlation
plt.figure(figsize=(10,10))
sns.heatmap(correlation,cbar=True,square=True,fmt='.1f',annot=True,annot_kws={'size'
```

Out[13]:

```
<AxesSubplot:>
```



Splitting the data into data and Targer(Price)

In [14]:

```
X=house_price_dataframe.drop(['price'],axis=1)
Y=house_price_dataframe['price']
```

In [15]:
```
print(X)
print(Y)
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
..       ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     PTRATIO       B  LSTAT
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: price, Length: 506, dtype: float64
```

Splitting the data into training data and test data

In [16]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=4)
```

In [17]:
```
print(X.shape,X_train.shape,X_test.shape)
```

```
(506, 13) (354, 13) (152, 13)
```

Model Training

XGBoost Regressor

In [18]:
```
#Loading the model
model=XGBRegressor()
```

In [19]:

```
#Training the model with X_train
model.fit(X_train,Y_train)
```

Out[19]:

```
XGBRegressor

XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwis
e',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weigh
t=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_
jobs=0,
```

Evaluation

Prediction on Training data

In [20]:

```
#accuracy for prediction on training data
training_data_prediction =model.predict(X_train)
```

In [21]:

```
print(training_data_prediction)
```

```
[23.898668  18.202156  21.698345  13.493743  49.992077  23.10208
 48.794533  13.817799  20.102333  50.00149   34.924053   8.39999
 15.187474  22.976278  24.694382  25.28971   17.211199  50.002148
 22.878271  20.199478  17.412027  19.499348  18.501293  14.001987
 22.610323  14.109741  15.607283  46.006325  20.496004  13.502242
 10.402703  21.431185  21.598164  23.185183  23.025291  17.60395
 16.11072    5.00106    8.298136  27.49878   18.691305  21.697563
 30.692205   5.008037  11.308817   7.000945  32.904667  14.601417
 11.990232  28.090174  17.998945   5.5982914 23.606403  24.698915
 22.4925    17.7013    13.096569  23.114618  25.0129    14.903409
  9.713751  22.810575  22.009037  23.61215   14.306128  18.795181
 19.897982  13.618455  19.405636  16.81838   20.00098   43.11957
 27.88593   20.115192  18.981453  19.216972  21.709469  33.10091
 49.99492   33.200787  20.118906  21.10423    8.802685  12.2546835
 14.4984255 23.788445  18.701365  21.803246  21.895859  21.698557
 17.102743  23.09697   36.097076  28.195421  11.524114  19.017113
 22.012339  10.486594  21.41922   16.49911   20.59537   23.30626
 23.50774   15.000558  26.490255  50.000847  10.492561  17.519302
 13.59443   17.19163   19.098984  16.397259  20.590748  20.906923
 30.069551  20.705257  22.199791  24.594826  25.205332  37.89925
 20.085173  29.596458  18.694838  22.986837  22.894016  24.586397
 24.80737   20.80593   22.403542  18.20351   14.401529  23.185047
 13.000709  19.696535  21.17987   21.703268  23.983181  22.002363
 20.60496   11.898764  24.276354  23.779114  22.80477   13.328387
 24.9945    20.989483  20.401754  33.09654   48.301083  14.49227
 36.00746   22.592964  18.392752  18.927902  12.6190405 15.213818
 24.094107  29.901505  23.910872  31.594862  11.701875  20.29879
 16.601126  22.176989  26.601843  36.191845  28.413687  20.82073
 15.398618  49.999863  18.09718   23.073902  21.493528  13.083476
 21.795036   8.502242  15.604793  26.208536  32.198586   9.606724
 31.602957  17.798496  34.697876  19.993362  21.002974  22.694841
 28.680223  23.875227  35.41479   13.195639  18.289051  13.100803
 23.106459  20.59922    7.000341  13.384457  24.104977  30.103985
 20.304886  15.612457  26.613838  15.00681   37.20695   27.093132
```

```
 24.397884  17.802233  19.806568  10.210543  23.113098  37.294025
 23.170357  19.073616  19.654306  38.7006    25.031815  23.711258
 22.794123  16.200487  20.325508  24.296154  21.201744  19.326782
 20.595406  21.384968  14.406331  19.91124   16.19812   22.480814
 19.128166  17.818674  30.0999    14.803343  35.199852  29.001917
 25.095686  21.505173   8.2950325 21.97281   44.80051   24.48276
 34.89165   17.199     33.80461   19.603312  14.0736685  8.423438
 33.316612  23.398811  21.405186  18.891907  21.190443   7.2160635
 27.092018  14.507033  10.389338  21.398642  14.091925  10.196625
 24.285624  18.603714  18.90239   10.909892  24.393059  19.293148
 24.998196  36.489086  20.514498  20.396557  19.599285  27.896692
 21.098457  26.594328  10.784633  36.180676  34.88086   31.495077
 31.708479  34.577793  17.793695  29.80721   35.100258  17.095772
 13.396441  36.98654   15.213398  27.510576  18.50712   19.58253
 23.203495  31.976273  23.401077  28.692957  21.999323  13.794538
 19.694435  20.905571  17.09005   28.394444  43.800564  22.482336
 50.00451   49.99332   33.421295  17.89427   25.002327  22.3129
 50.00089    9.503665  10.206892  23.712202  23.793251   7.500893
 23.905233  18.388336  20.41871   19.397469  17.395752  12.699183
 13.792526  22.00778   29.095022  24.695938  20.797657  24.094175
 15.395751  19.554085  32.495182  24.008078   7.401533  25.019457
 15.699164  21.704689  21.206905  11.691455  22.68644   16.846464
 21.598713  23.889149  22.111568  20.584314  19.391888  22.591293
 29.591736  23.302452  13.795169  33.40695   12.712214  22.213974
 25.005445   7.2029343 30.30505   12.809058  22.581894  20.50012  ]
```

In [22]:

```python
#R sqaured error
score_1=metrics.r2_score(Y_train,training_data_prediction)

#Mean Absolute error
score_2=metrics.mean_absolute_error(Y_train,training_data_prediction)

print("R sqaured error:",score_1)
print("Mean Absolute error:",score_2)
```

```
R sqaured error: 0.9999980912185324
Mean Absolute error: 0.008653184923075066
```

Visaulizing the actual Prices and predicted prices

In [23]:

```python
plt.scatter(Y_train,training_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Price vs Predicted Price")
plt.show()
```

Prediction on Test Data

In [24]:
```
#accuracy for prediction on test data
test_data_prediction =model.predict(X_test)
```

In [25]:
```
 #R sqaured error
score_1=metrics.r2_score(Y_test,test_data_prediction)

#Mean Absolute error
score_2=metrics.mean_absolute_error(Y_test,test_data_prediction)

print("R sqaured error:",score_1)
print("Mean Absolute error",score_2)
```

```
R sqaured error: 0.8579951986672496
Mean Absolute error 2.5309582503218397
```

In [ ]: