

Importing the Dependencies

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

Importing the Boston House Price Dataset

In [2]:

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

In [3]:

```
print(house_price_dataset)
```

```
{'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, 2
  7.1, 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. , 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_databse
t:\n\nBoston house prices dataset\n-----\n**Data Set Charact
eristics:** \n\n :Number of Instances: 506 \n\n :Number of Attributes: 13 num
eric/categorical predictive. Median Value (attribute 14) is usually the target.\n\n
:Attribute Information (in order):\n      - CRIM      per capita crime rate by town
      - ZN      proportion of residential land zoned for lots over 25,000 sq.f
      - INDUS     proportion of non-retail business acres per town\n      - C
HAS      Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n
      - NOX      nitric oxides concentration (parts per 10 million)\n      - RM      av
erage number of rooms per dwelling\n      - AGE      proportion of owner-occupied
units built prior to 1940\n      - DIS      weighted distances to five Boston empl
oyment centres\n      - RAD      index of accessibility to radial highways\n
      - TAX      full-value property-tax rate per $10,000\n      - PTRATIO      pupil-teache
r 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 V
alues: None\n\n :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of U
CI ML housing dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/ho
using/\n\nThis dataset was taken from the StatLib library which is maintained at C
arnegie Mellon University.\n\nThe Boston house-price data of Harrison, D. and Rubinf
eld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Economics & Ma
nagement,\nvol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnos
tics\n...', Wiley, 1980. N.B. Various transformations are used in the table on\npa
ges 244-261 of the latter.\n\nThe Boston house-price data has been used in many mach
ine learning papers that address regression\nproblems. \n      .. topic:: Referen
ces\n\n      - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.\n      - Quinlan,R. (1993). Co
mbining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Interna
tional Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. M
organ Kaufmann.\n", 'filename': 'C:\\Users\\ANKITA DAS\\anaconda3\\lib\\site-pac
kages\\sklearn\\datasets\\data\\boston_house_prices.csv'}
```

In [4]:

```
# Loading the dataset to a Pandas DataFrame
house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns = house_price
```

In [5]:

```
# Print First 5 rows of our DataFrame
house_price_dataframe.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
--	------	----	-------	------	-----	----	-----	-----	-----	-----	---------	---	-------

	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 [6]:

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

In [7]:

```
house price dataframe.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	F
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]:

```
# checking the number of rows and Columns in the data frame  
house_price_dataframe.shape
```

Out[81]:

(506, 14)

Tn [9]:

check for missing value

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

2 / 503

```
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]:

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

Out[10]:

CRIM **ZN** **INDUS** **CHAS** **NOX** **RM** **AGE** **D**

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	D
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
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
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.1265



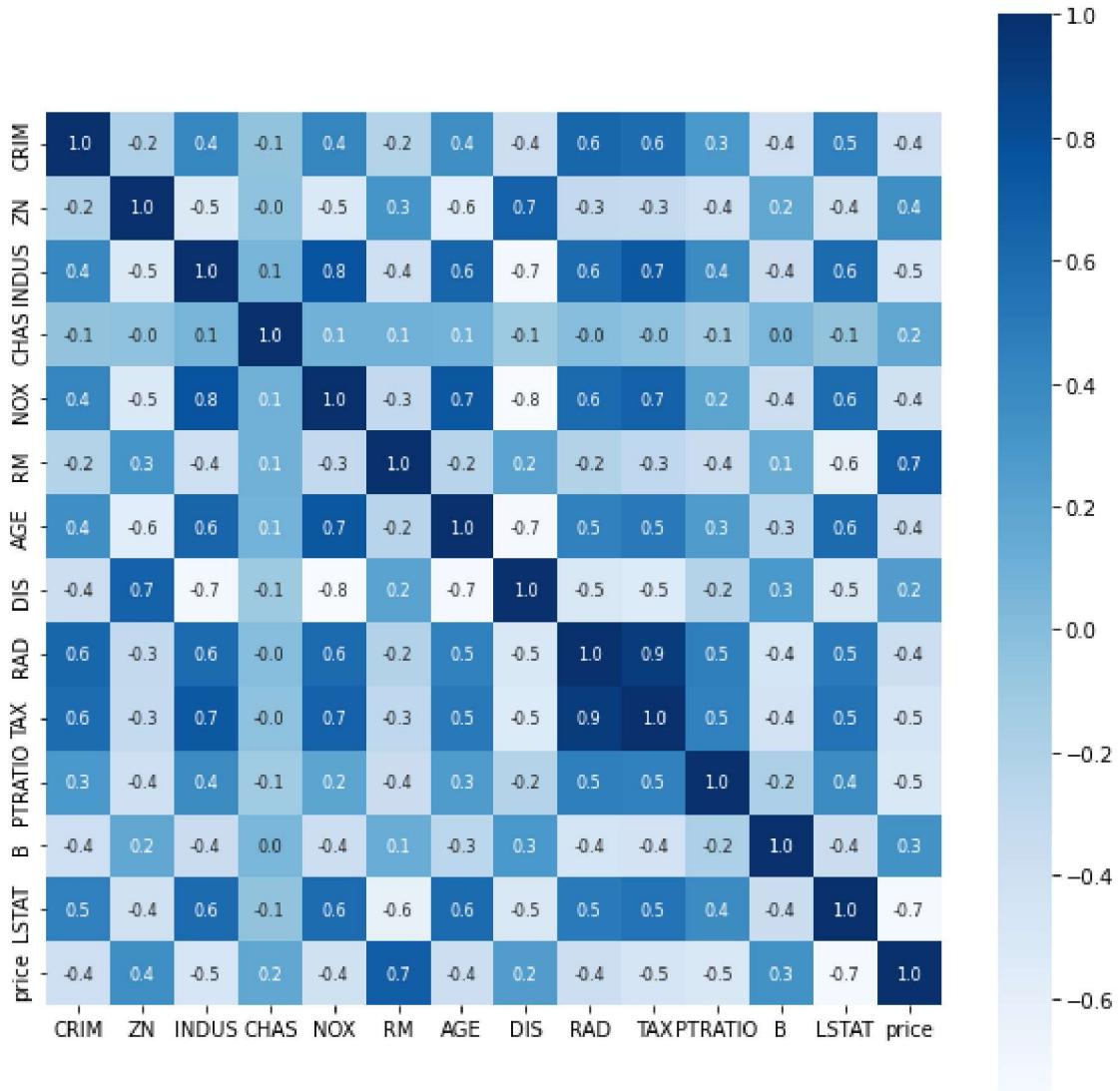
1. Positive Correlation

2. Negative Correlation

```
In [11]: correlation = house_price_dataframe.corr()
```

```
In [12]: # 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={
```

```
Out[12]: <AxesSubplot:>
```



Splitting the data and Target

```
In [13]: X = house_price_dataframe.drop(['price'], axis=1)
        Y = house_price_dataframe['price']
```

In [14]:

	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 [15]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_st
```



```
In [16]: print(X.shape, X_train.shape, X_test.shape)
```



```
(506, 13) (404, 13) (102, 13)
```

Model Training

XGBoost Regressor

```
In [17]: # Loading the model
model = XGBRegressor()
```



```
In [18]: # training the model with X_train
model.fit(X_train, Y_train)
```



```
Out[18]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                      colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
                      gamma=0, gpu_id=-1, importance_type=None,
                      interaction_constraints='', learning_rate=0.300000012,
                      max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
                      monotone_constraints='()', n_estimators=100, n_jobs=4,
                      num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
                      reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
                      validate_parameters=1, verbosity=None)
```

Evaluation

Prediction on training data

```
In [19]: # accuracy for prediction on training data
training_data_prediction = model.predict(X_train)
```

In [20]:

```
print(training_data_prediction)
```

23.147501	20.99463	20.090284	34.69053	13.903663	13.510157
21.998634	15.1940975	10.899711	22.709627	13.832816	5.592794
29.810236	49.99096	34.89215	20.607384	23.351097	19.23555
32.695698	19.641418	26.991022	8.401829	46.00729	21.708961
27.062933	19.321356	19.288303	24.809872	22.61626	31.70493
18.542515	8.697379	17.395294	23.700663	13.304856	10.492197
12.688369	25.016556	19.67495	14.902088	24.193798	25.007143
14.900281	16.995798	15.6009035	12.699232	24.51537	14.999952
50.00104	17.525454	21.184624	31.998049	15.613355	22.89754
19.325378	18.717896	23.301125	37.222923	30.09486	33.102703
21.00072	49.999332	13.405827	5.0280113	16.492886	8.405072
28.64328	19.499939	20.586452	45.402164	39.79833	33.407326
19.83506	33.406372	25.271482	50.001534	12.521657	17.457413
18.61758	22.602625	50.002117	23.801117	23.317268	23.087355
41.700035	16.119293	31.620516	36.069206	7.0022025	20.3827
19.996452	11.986318	25.023014	49.970123	37.881588	23.123034
41.292133	17.596548	16.305374	30.034231	22.860699	19.810343
17.098848	18.898268	18.96717	22.606049	23.141363	33.183487
15.010934	11.693824	18.78828	20.80524	17.99983	19.68991
50.00332	17.207317	16.404053	17.520426	14.593481	33.110855
14.508482	43.821655	34.939106	20.381636	14.655634	8.094332
11.7662115	11.846876	18.69599	6.314154	23.983706	13.084503
19.603905	49.989143	22.300608	18.930315	31.197134	20.69645
32.21111	36.15102	14.240763	15.698188	49.99381	20.423601
16.184978	13.409128	50.01321	31.602146	12.271495	19.219482
29.794909	31.536846	22.798779	10.189648	24.08648	23.710463
21.991894	13.802495	28.420696	33.181534	13.105958	18.988266
26.576572	36.967175	30.794083	22.77071	10.201246	22.213818
24.483162	36.178806	23.09194	20.097307	19.470194	10.786644
22.671095	19.502405	20.109184	9.611871	42.799637	48.794792
13.097208	20.28583	24.793974	14.110478	21.701134	22.217012
33.003544	21.11041	25.00658	19.122992	32.398567	13.605098
15.1145315	23.088867	27.474783	19.364998	26.487135	27.499458
28.697094	21.21718	18.703201	26.775208	14.010719	21.692347
18.372562	43.11582	29.081839	20.289959	23.680176	18.308306
17.204844	18.320065	24.393475	26.396057	19.094141	13.3019905
22.15311	22.185797	8.516214	18.894428	21.792608	19.331121
18.197924	7.5006843	22.406403	20.004215	14.412416	22.503702
28.53306	21.591028	13.810223	20.497831	21.898977	23.104464
49.99585	16.242056	30.294561	50.001595	17.771557	19.053703
10.399217	20.378187	16.49973	17.183376	16.70228	19.495337
30.507633	28.98067	19.528809	23.148346	24.391027	9.521643
23.886024	49.995125	21.167099	22.597813	19.965279	13.4072275
19.948694	17.087479	12.738807	23.00453	15.222122	20.604322
26.207253	18.09243	24.090246	14.105	21.689667	20.08065
25.010437	27.874954	22.92366	18.509727	22.190847	24.004797
14.788686	19.89675	24.39812	17.796036	24.556297	31.970308
17.774675	23.356768	16.134794	13.009915	10.98219	24.28906
15.56895	35.209793	19.605724	42.301712	8.797891	24.400295
14.086652	15.408639	17.301126	22.127419	23.09363	44.79579
17.776684	31.50014	22.835577	16.888603	23.925127	12.097476
38.685944	21.388391	15.98878	23.912495	11.909485	24.960499
7.2018585	24.696215	18.201897	22.489008	23.03332	24.260433
17.101519	17.805563	13.493165	27.105328	13.311978	21.913465
20.00738	15.405392	16.595737	22.301016	24.708412	21.422579
22.878702	29.606575	21.877811	19.900253	29.605219	23.407152
13.781474	24.454706	11.897682	7.2203646	20.521074	9.725295
48.30087	25.19501	11.688618	17.404732	14.480284	28.618876
19.397131	22.468653	7.0117908	20.602013	22.970919	19.719397
23.693787	25.048244	27.977154	13.393578	14.513882	20.309145
19.306028	24.095829	14.894031	26.382381	33.298378	23.61644

```
24.591206 18.514652 20.900269 10.406055 23.303423 13.092017  
24.675085 22.582184 20.502762 16.820635 10.220605 33.81239  
18.608067 49.999187 23.775583 23.909609 21.192276 18.805798  
8.502987 21.50807 23.204473 21.012218 16.611097 28.100965  
21.193024 28.419638 14.294126 49.99958 30.988504 24.991066  
21.433628 18.975573 28.991457 15.206939 22.817244 21.765755  
19.915497 23.7961 ]
```

In [21]:

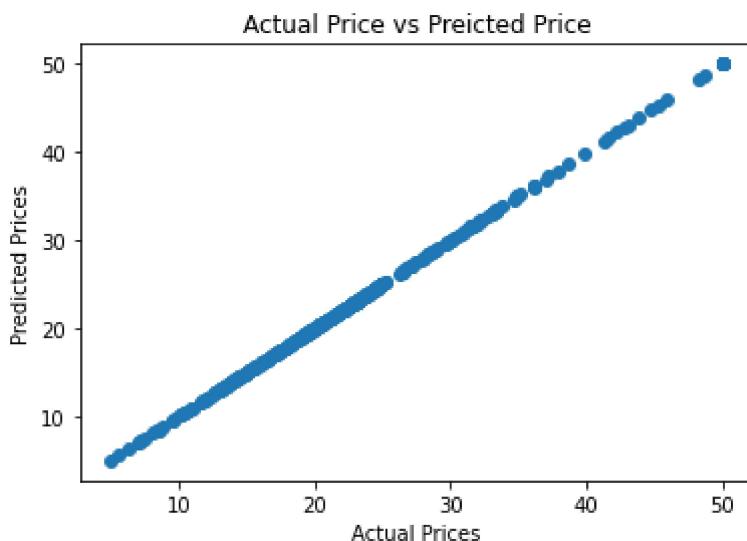
```
# R squared error  
score_1 = metrics.r2_score(Y_train, training_data_prediction)  
  
# Mean Absolute Error  
score_2 = metrics.mean_absolute_error(Y_train, training_data_prediction)  
  
print("R squared error : ", score_1)  
print('Mean Absolute Error : ', score_2)
```

```
R squared error : 0.9999948236320982  
Mean Absolute Error : 0.0145848437110976
```

Visualizing the actual Prices and predicted prices

In [22]:

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



Prediction on Test Data

In [23]:

```
# accuracy for prediction on test data  
test_data_prediction = model.predict(X_test)
```

In [24]:

```
# R squared 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 squared error : ", score_1)
print('Mean Absolute Error : ', score_2)
```

```
R squared error : 0.8711660369151691
Mean Absolute Error : 2.2834744154238233
```