HOUSING PRICE PREDICTIONS

The goal of this Project is to help us understand the relationship between house features and how these variables are used to predict house price.

Predict the house price

The data set is taken from https://www.kaggle.com/schirmerchad/bostonhoustingmlnd

IMPORTING THE LIBRARIES

In [52]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas.util.testing as tm
```

LOADING THE DATASET

In [53]:
```
dataset = pd.read_csv("C:/Users/HP/Desktop/coursera/project/housing_pri
ce/boston 2/housing.csv")
```

EDA

Size of the dataset

In [54]:
```
dataset.shape
```

Out[54]:
```
(489, 4)
```

View the top few rows

In [55]:
```
dataset.head()
```

Out[55]:

	RM	LSTAT	PTRATIO	MEDV
0	6.575	4.98	15.3	504000.0
1	6.421	9.14	17.8	453600.0
2	7.185	4.03	17.8	728700.0
3	6.998	2.94	18.7	701400.0
4	7.147	5.33	18.7	760200.0

View the bottom few rows

In [56]:
```
dataset.tail()
```

Out[56]:

	RM	LSTAT	PTRATIO	MEDV
484	6.593	9.67	21.0	470400.0
485	6.120	9.08	21.0	432600.0
486	6.976	5.64	21.0	501900.0
487	6.794	6.48	21.0	462000.0
488	6.030	7.88	21.0	249900.0

Check for null values

In [57]:
```
dataset.isnull().sum()
```

Out[57]:
```
RM         0
LSTAT      0
PTRATIO    0
MEDV       0
dtype: int64
```

We dont have any null values

The variable we have to predict is MEDV which will be stored as prices

In [58]:
```
prices = dataset['MEDV']
```

Get a overview of the statistical information of tthe dataset

In [59]:
```
dataset.describe()
```

Out[59]:

	RM	LSTAT	PTRATIO	MEDV
count	489.000000	489.000000	489.000000	4.890000e+02
mean	6.240288	12.939632	18.516564	4.543429e+05
std	0.643650	7.081990	2.111268	1.653403e+05
min	3.561000	1.980000	12.600000	1.050000e+05
25%	5.880000	7.370000	17.400000	3.507000e+05
50%	6.185000	11.690000	19.100000	4.389000e+05
75%	6.575000	17.120000	20.200000	5.187000e+05
max	8.398000	37.970000	22.000000	1.024800e+06

Get an overview of the type of variables in the dataset

In [60]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 489 entries, 0 to 488
Data columns (total 4 columns):
RM         489 non-null float64
LSTAT      489 non-null float64
PTRATIO    489 non-null float64
MEDV       489 non-null float64
dtypes: float64(4)
memory usage: 15.4 KB
```

In [61]:
```
X=dataset.drop('MEDV',axis=1)
```

In [62]:
```
y=dataset['MEDV']
```

In [63]:
```
X
```

Out[63]:

	RM	LSTAT	PTRATIO
0	6.575	4.98	15.3
1	6.421	9.14	17.8
2	7.185	4.03	17.8
3	6.998	2.94	18.7
4	7.147	5.33	18.7
...	...	...	...
484	6.593	9.67	21.0
485	6.120	9.08	21.0
486	6.976	5.64	21.0
487	6.794	6.48	21.0
488	6.030	7.88	21.0

489 rows × 3 columns

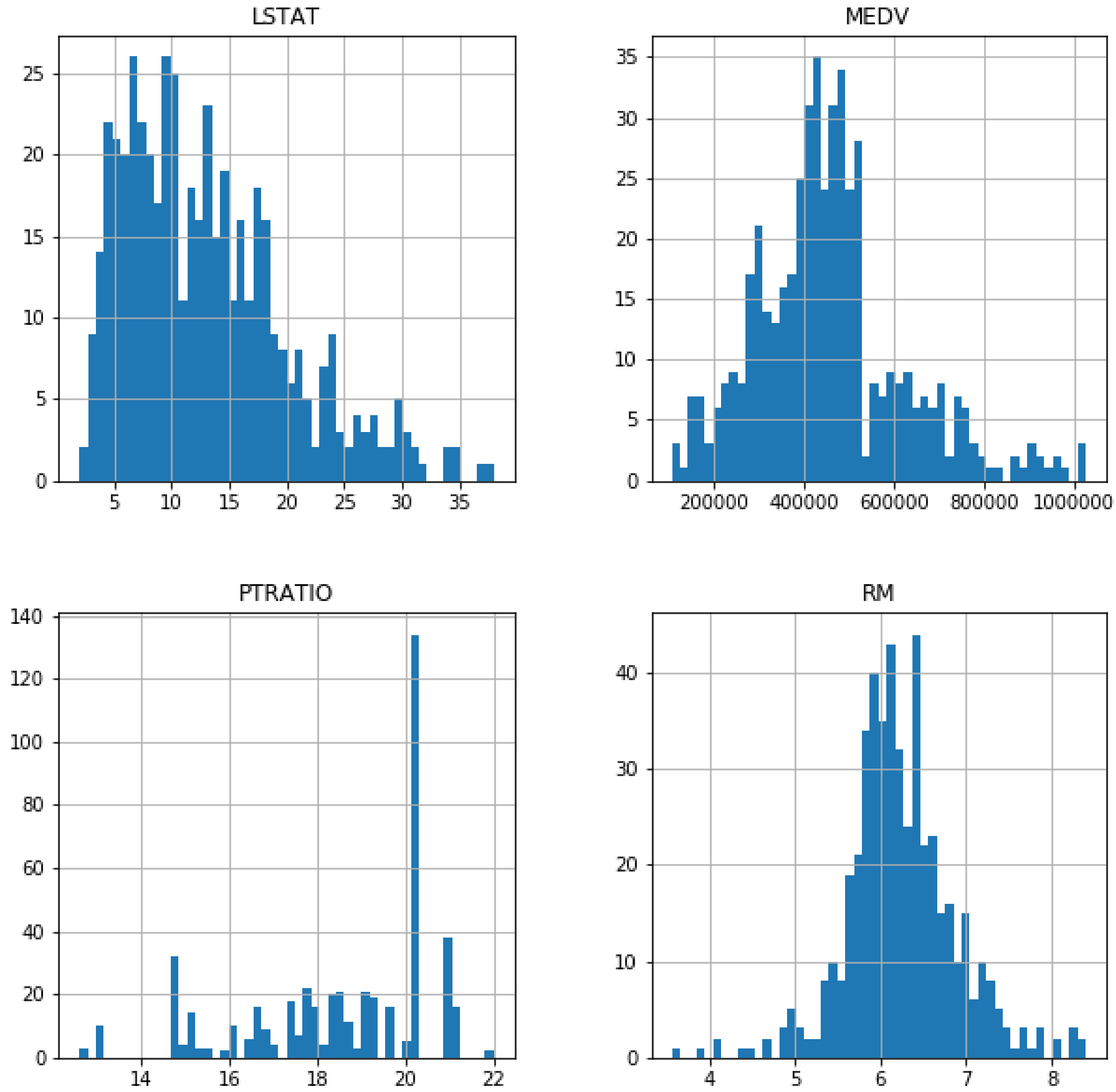
In [64]:
```
X.shape
```
Out[64]:
```
(489, 3)
```

In [65]:
```
y
```
Out[65]:
```
0      504000.0
1      453600.0
2      728700.0
3      701400.0
4      760200.0
         ...
484    470400.0
485    432600.0
486    501900.0
487    462000.0
488    249900.0
Name: MEDV, Length: 489, dtype: float64
```

In [66]:
```
y.shape
```
Out[66]:
```
(489,)
```

PLOTTING FEW GRAPHS

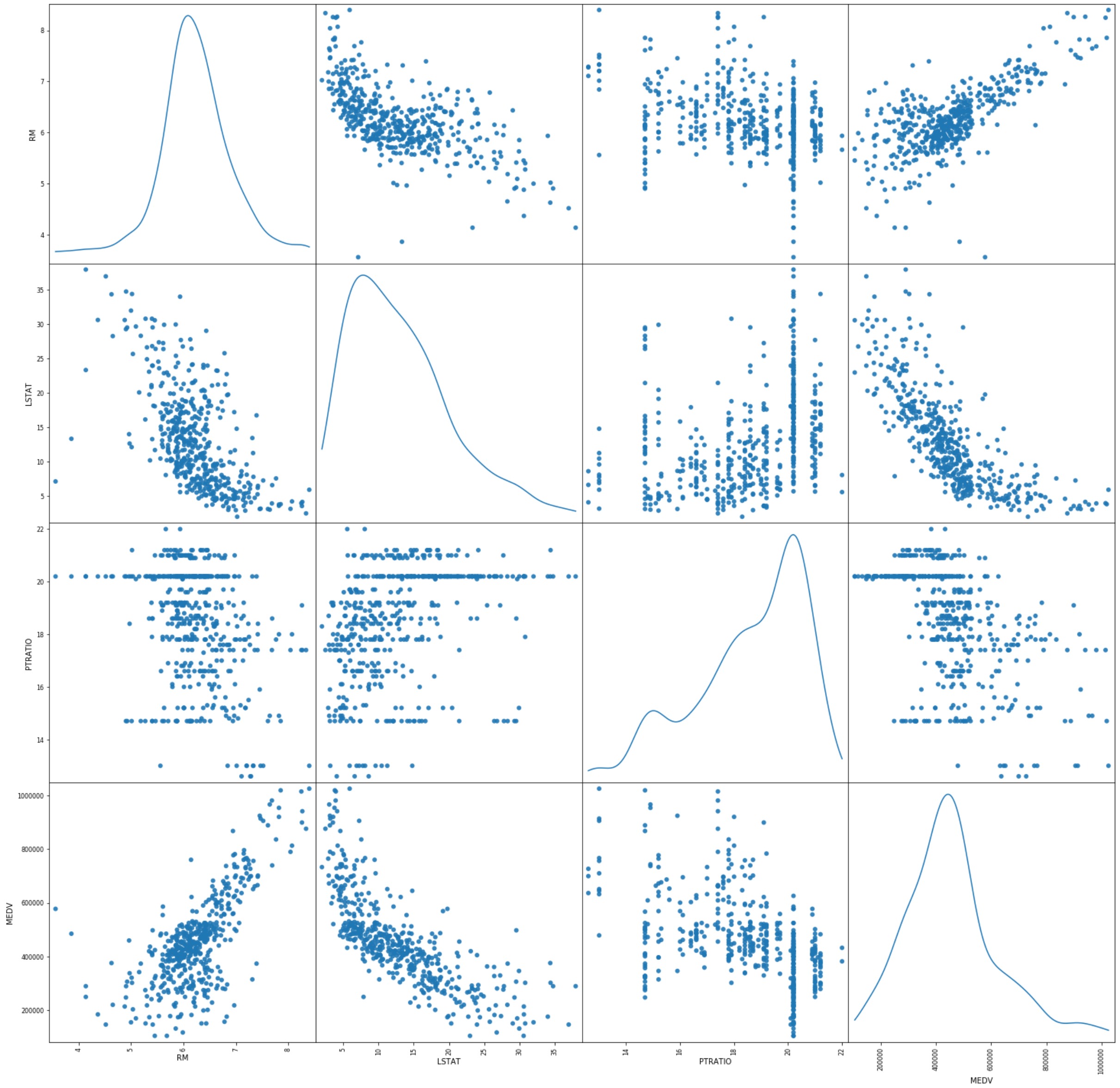
In [67]:
```
dataset.hist(bins=50,figsize=(10,10))
```
Out[67]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8C7D
7DC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8C7B
4CC8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8C76
AE88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8D89
CEC8>]],
      dtype=object)
```


LSTAT
MEDV
PTRATIO
RM

In [68]:
```python
from pandas.plotting import scatter_matrix
fig=plt.figure()
scatter_matrix(dataset,figsize=(25,25),alpha=0.9,diagonal="kde",marker=
"o")
```

Out[68]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DD3
42C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DE0
09C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DB1
23C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DB4
5DC8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DB7
F748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DBB
B348>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DBF
3408>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DC2
B508>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DC3
5608>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DC6
E7C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8DCD
6848>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8EF5
D908>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8EF9
7A48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8EFC
FB48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8F00
8C48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001BE8F03
FD88>]],
      dtype=object)

<Figure size 432x288 with 0 Axes>
```



RM
LSTAT
PTRATIO
MEDV

CORRELATION MATRIX

In [69]:
```
corr=dataset.corr()
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[69]:

	RM	LSTAT	PTRATIO	MEDV
RM	1	-0.61	-0.3	0.7
LSTAT	-0.61	1	0.36	-0.76
PTRATIO	-0.3	0.36	1	-0.52
MEDV	0.7	-0.76	-0.52	1

SPLITTING THE DATASET INTO TEST AND TRAIN SET

In [70]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
random_state = 42)
```

EDA OF TEST AND TRAIN SETS

In [71]:
```
X_train.shape
```

Out[71]:
```
(391, 3)
```

In [72]:
```
X_train
```

Out[72]:

	RM	LSTAT	PTRATIO
325	5.869	9.80	20.2
140	6.174	24.16	21.2
433	6.749	17.44	20.2
416	6.436	16.22	20.2
487	6.794	6.48	21.0
...	...	...	...
106	5.836	18.66	20.9
270	7.820	3.76	14.9
348	6.112	12.67	20.2
435	6.297	17.27	20.2
102	6.405	10.63	20.9

391 rows × 3 columns

In [73]:
```
y_train.shape
```

Out[73]:
```
(391,)
```

In [74]:
```
y_train
```

Out[74]:
```
325    409500.0
140    294000.0
```

```
433    281400.0
416    300300.0
487    462000.0
         ...
106    409500.0
270    953400.0
348    474600.0
435    338100.0
102    390600.0
Name: MEDV, Length: 391, dtype: float64
```

In [75]:
```
X_test.shape
```

Out[75]:
```
(98, 3)
```

In [76]:
```
X_test
```

Out[76]:

	RM	LSTAT	PTRATIO
451	5.926	18.13	20.2
84	6.389	9.62	18.5
434	6.655	17.73	20.2
472	5.414	23.97	20.1
428	6.459	23.98	20.2
...	...	...	...
317	5.868	9.97	16.9
376	6.193	15.17	20.2
56	6.383	5.77	17.3
275	6.230	12.93	18.2
398	6.434	29.05	20.2

98 rows × 3 columns

In [77]:
```
y_test.shape
```
Out[77]: (98,)

In [78]:
```
y_test
```
Out[78]:
```
451    401100.0
84     501900.0
434    319200.0
472    147000.0
428    247800.0
         ...
317    405300.0
376    289800.0
56     518700.0
275    422100.0
398    151200.0
Name: MEDV, Length: 98, dtype: float64
```

In [79]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

MODEL

Linear Regression

In [80]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
```
Out[80]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normaliz
e=False)
```

In [81]:
```
y_lr_pred=lr.predict(X_test)
```

In [82]:
```
y_lr_pred
```

Out[82]:
```
array([342593.79029768, 506257.0916297 , 410499.93166174, 237792.741153
7 ,
        327005.79653234, 403018.068531  , 261060.38389067, 701308.473745
97,
        362924.70496746, 585818.82333754, 456966.23009711, 365587.848577
13,
        266036.4241684 , 265799.92818911, 385359.28098829, 525974.874337
62,
        388922.38353646, 365210.2410349 , 365315.35425769, 420439.938351
04,
        459794.49010487, 461685.28906052, 369745.76216645, 644034.098405
83,
        467828.26948158, 473745.56661447, 498572.57258183, 634774.917352
29,
        679806.33028785, 168957.24703839, 514819.05350129, 239552.373203
21,
        536885.46626665, 508876.38428348, 305150.22603695, 502246.532716
74,
        633616.8915942 , 498079.88203251, 664064.07473373, 640154.943209
99,
        417975.24110305, 413013.84915423, 321372.49298713, 454781.420060
81,
        392252.56415048, 583126.90625531, 354489.73066978, 392557.517849
78,
        411096.16495751, 393192.83809688, 276193.78443298, 584399.725405
12,
        571878.35038497, 428494.3472316 , 501643.04699658, 507982.223442
14,
        366272.30476642, 532727.43571945, 440147.75671624, 276035.322008
52,
        475228.76356089, 383760.04325352, 516913.45770147, 475003.737089
5 ,
        570709.98844992, 755704.0690957 , 530831.9167639 , 537408.663465
   ,
         19532.43641433, 345192.65424488, 273964.90585283, 411711.050590
```

```
52,
        425191.62275258, 456485.9726244 , 350140.83517251, 462944.006110
73,
        468058.95069006, 424612.225707  , 519797.70298078, 420541.402488
99,
        513071.80427338, 571461.91574647, 378549.15343831, 283145.381135
78,
        480427.57453125, 570576.42448841, 439187.72257502, 500620.055972
04,
        382443.52148658, 421695.97579263, 461775.6500832 , 400911.234958
2 ,
        382508.50914966, 487964.05695847, 397345.90511691, 569811.904427
4 ,
        463015.8748299 , 270976.11025369])
```

In [83]:
```
np.mean((y_lr_pred - y_test)**2)**0.5
```

Out[83]:
```
82395.54332162565
```

Logistic Regression

In [84]:
```
from sklearn.linear_model import LogisticRegression
log_r=LinearRegression()
log_r.fit(X_train,y_train)
```

Out[84]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normaliz
e=False)
```

In [85]:
```
y_log_r_pred=lr.predict(X_test)
```

In [86]:
```
y_log_r_pred
```

Out[86]:
```
array([342593.79029768, 506257.0916297 , 410499.93166174, 237792.741153
7 ,
        327005.79653234, 403018.068531  , 261060.38389067, 701308.473745
97,
        362924.70496746, 585818.82333754, 456966.23009711, 365587.848577
```

```
13,
       266036.4241684 , 265799.92818911, 385359.28098829, 525974.874337
62,
       388922.38353646, 365210.2410349 , 365315.35425769, 420439.938351
04,
       459794.49010487, 461685.28906052, 369745.76216645, 644034.098405
83,
       467828.26948158, 473745.56661447, 498572.57258183, 634774.917352
29,
       679806.33028785, 168957.24703839, 514819.05350129, 239552.373203
21,
       536885.46626665, 508876.38428348, 305150.22603695, 502246.532716
74,
       633616.8915942 , 498079.88203251, 664064.07473373, 640154.943209
99,
       417975.24110305, 413013.84915423, 321372.49298713, 454781.420060
81,
       392252.56415048, 583126.90625531, 354489.73066978, 392557.517849
78,
       411096.16495751, 393192.83809688, 276193.78443298, 584399.725405
12,
       571878.35038497, 428494.3472316 , 501643.04699658, 507982.223442
14,
       366272.30476642, 532727.43571945, 440147.75671624, 276035.322008
52,
       475228.76356089, 383760.04325352, 516913.45770147, 475003.737089
5 ,
       570709.98844992, 755704.0690957 , 530831.9167639 , 537408.663465
  ,
        19532.43641433, 345192.65424488, 273964.90585283, 411711.050590
52,
       425191.62275258, 456485.9726244 , 350140.83517251, 462944.006110
73,
       468058.95069006, 424612.225707  , 519797.70298078, 420541.402488
99,
       513071.80427338, 571461.91574647, 378549.15343831, 283145.381135
78,
       480427.57453125, 570576.42448841, 439187.72257502, 500620.055972
04,
```

```
        382443.52148658, 421695.97579263, 461775.6500832 , 400911.234958
2 ,
        382508.50914966, 487964.05695847, 397345.90511691, 569811.904427
4 ,
        463015.8748299 , 270976.11025369])
```

In [87]:
```
np.mean((y_log_r_pred - y_test)**2)**0.5
```

Out[87]:
```
82395.54332162565
```

Decision Tree Regressor

In [88]:
```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()
dt.fit(X_train,y_train)
```

Out[88]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=Non
e,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.
0,
                      presort=False, random_state=None, splitter='bes
t')
```

In [89]:
```
y_dt_pred = dt.predict(X_test)
```

In [90]:
```
np.mean((y_dt_pred - y_test)**2)**0.5
```

Out[90]:
```
77323.79969970437
```

Random Forest Regressor

In [91]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
```

```
rf.fit(X_train,y_train)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[91]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=Non
e,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [92]:

```
y_rf_pred = rf.predict(X_test)
```

In [93]:

```
y_rf_pred
```

Out[93]:

```
array([324870., 503160., 295260., 237090., 244440., 421890., 244650.,
       891450., 399000., 598290., 381780., 426720., 270480., 413490.,
       305550., 500220., 376740., 342300., 279300., 408870., 447930.,
       432810., 265020., 726180., 446460., 453810., 416220., 689010.,
       661710., 155820., 517020., 421050., 533190., 479430., 318990.,
       494340., 635880., 449610., 692580., 681450., 357630., 450870.,
       315210., 352170., 445200., 487410., 383460., 359100., 453600.,
       427770., 228690., 535080., 494340., 438270., 413910., 441840.,
       335790., 499590., 426720., 288540., 494130., 355320., 546630.,
       396060., 546210., 895020., 472500., 428400., 315630., 389130.,
       260820., 429660., 432180., 435960., 337470., 438060., 480900.,
       408870., 492660., 421050., 499590., 548940., 439950., 382620.,
       435540., 548100., 446880., 503580., 323820., 414120., 420420.,
       286440., 387240., 437010., 313530., 517860., 433020., 154350.])
```

In [94]:

```
np.mean((y_rf_pred - y_test)**2)**0.5
```

Out[94]:

```
60801.42268072352
```

CALCULATE R SQUARED MATRIX

In [95]:
```
from sklearn.metrics import r2_score

def performance_metric(y_true, y_predict):
    score = r2_score(y_true, y_predict)
    return score
```

In [96]:
```
performance_metric(y_test,y_rf_pred)
```

Out[96]:
```
0.8317917757782279
```

HYPER PARAMETER TUNNING

In [97]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, n
um = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
```

```
                    'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 200
0], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50,
60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_
samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

In [98]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
model = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
model_random = RandomizedSearchCV(estimator = model, param_distribution
s = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jo
bs = -1)
# Fit the random search model
model_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    7.4s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   36.5s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  1.3min finished
```

Out[98]:
```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   min_impurity_decreas
e=0.0,
                                                   min_impurity_split=N
one,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_
```

```
leaf=0.0,
                                                                          n_estimators='warn',
                                                                          n_jobs=None, oob_sco
re=False,
                                                                          random_sta...
                           param_distributions={'bootstrap': [True, False],
                                                'max_depth': [10, 20, 30, 40, 5
0, 60,
                                                                70, 80, 90, 100,
110,
                                                                None],
                                                'max_features': ['auto', 'sqr
t'],
                                                'min_samples_leaf': [1, 2, 4],
                                                'min_samples_split': [2, 5, 1
0],
                                                'n_estimators': [200, 400, 600,
800,
                                                                   1000, 1200, 14
00, 1600,
                                                                   1800, 2000]},
                           pre_dispatch='2*n_jobs', random_state=42, refit=Tru
e,
                           return_train_score=False, scoring=None, verbose=2)
```

In [99]:
```
model_random.best_params_
```

Out[99]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 60,
 'bootstrap': True}
```

In [100]:
```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
```

```python
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
```

In [102]:

```python
best_random = model_random.best_estimator_
random_accuracy = evaluate(best_random, X_test, y_test)
```

```
Model Performance
Average Error: 44086.0017 degrees.
Accuracy = 86.78%.
```

In [103]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: