Selecting a desired model for Dragon Real Estates End to End Machine learning Project

import pandas as pd

In [6]:

```
housing=pd.read_csv("housing.csv")
```

In [7]:

```
housing.head()
```

Out[7]:

	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

In [8]:

```
housing.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: 15.3 KB
```

In [9]:

```
housing['RM'].value_counts()
```

Out[9]:

```
6.229    3
6.417    3
6.405    3
5.713    3
6.167    3
6.127    3
6.727    2
6.635    2
6.376    2
5.856    2
5.390    2
6.162    2
6.096    2
6.431    2
5.404    2
6.968    2
5.813    2
6.315    2
5.854    2
6.030    2
7.185    2
5.888    2
```

```
6.495    2
5.926    2
6.794    2
5.936    2
6.630    2
6.312    2
6.004    2
5.935    2
        ..
6.982    1
6.456    1
6.718    1
6.812    1
6.854    1
5.898    1
6.939    1
5.950    1
6.021    1
7.416    1
6.015    1
5.706    1
6.510    1
6.212    1
7.163    1
6.092    1
6.072    1
5.783    1
6.242    1
5.818    1
6.430    1
5.520    1
7.178    1
5.036    1
5.570    1
6.701    1
6.232    1
6.593    1
6.425    1
6.625    1
Name: RM, Length: 430, dtype: int64
```

In [10]:

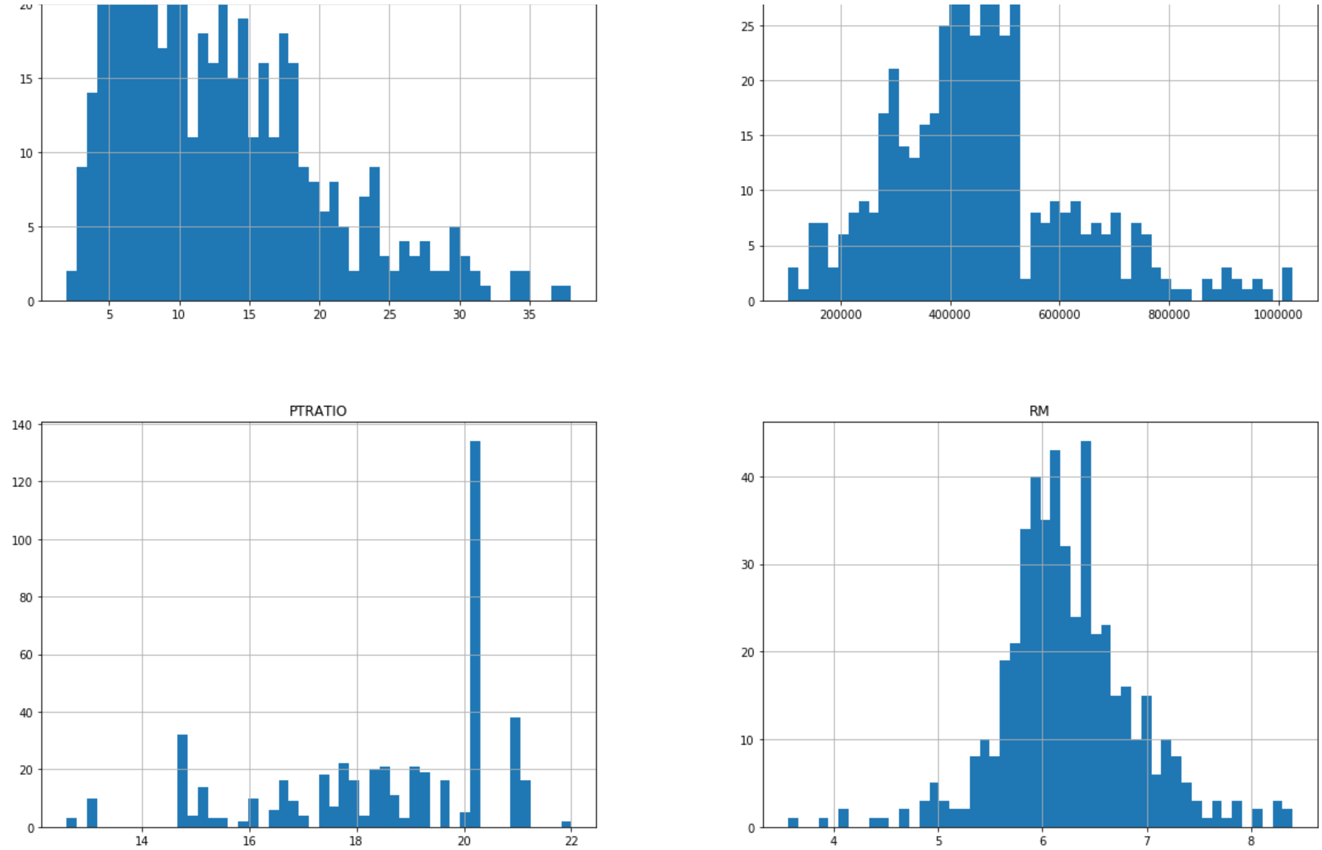
```
housing.describe()
```

Out[10]:

	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

In [12]:

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50,figsize=(20,15))
plt.show()
```



In [13]:

```
##Train-Test Splitting
##learning perpose for testing data and training data
import numpy as np
def split_train_test(data,test_ratio):
    np.random.seed(42)
    shuffled=np.random.permutation(len(data))
    ##print(shuffled)
    test_set_size=int(len(data)*test_ratio)
    test_indices=shuffled[:test_set_size]
    train_indices=shuffled[test_set_size:]
    return data.iloc[train_indices],data.iloc[test_indices]
```

In [14]:

```
train_set,test_set=split_train_test(housing,0.2)
print(f"Rows in train set:{len(train_set)}\nRows in test set:{len(test_set)}\n")
```

```
Rows in train set:392
Rows in test set:97
```

In [15]:

```
from sklearn.model_selection import train_test_split
train_set,test_set=train_test_split(housing,test_size=0.2,random_state=42)
print(f"Rows in train set:{len(train_set)}\nRows in test set:{len(test_set)}\n")
```

```
Rows in train set:391
Rows in test set:98
```

In [16]:

```
'''from sklearn.model_selection import StratifiedShuffleSplit
split=StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=42)
for train_index,test_index in split.split(housing,housing['MEDV']):
    strat_train_set=housing[train_index]
    strat_test_set=housing[test_index]
```

```
    strat_test_set=housing[test_index]
strat_test_set'''
```

Out[16]:

```
"from sklearn.model_selection import
StratifiedShuffleSplit\nsplit=StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=42)\nfor
train_index,test_index in split.split(housing,housing['MEDV']):\n
strat_train_set=housing[train_index]\n    strat_test_set=housing[test_index]\nstrat_test_set"
```

In [17]:

```
##Looking for Correlations
corr_matrix=housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[17]:

```
MEDV       1.000000
RM         0.697209
PTRATIO   -0.519034
LSTAT     -0.760670
Name: MEDV, dtype: float64
```

In [18]:

```
from pandas.plotting import scatter_matrix
attributes=['MEDV','RM','PTRATIO','LSTAT']
scatter_matrix(housing[attributes],figsize=(12,8))
```

Out[18]:

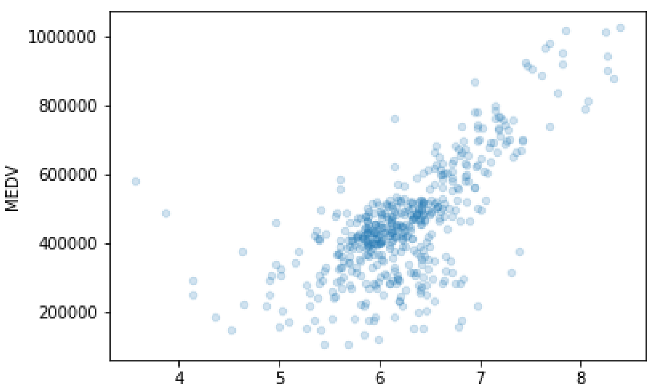
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0B134F70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F1CDEB0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F1DDDD0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0FA58E90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0FA73F50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0FA8CFF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0FAB06D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0FACA190>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0EFEF090>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F1AE510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F139E90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F14C5F0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0F01CCF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F034B90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F12F290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0F11CEB0>]],
      dtype=object)
```

In [19]:

```
housing.plot(kind='scatter',x='RM',y='MEDV',alpha=0.2)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xefa8cf0>
```



In [20]:

```
##Trying out attributes combinations
housing['MEDV']=housing['LSTAT']/housing['RM']
housing.head()
```

Out[20]:

	RM	LSTAT	PTRATIO	MEDV
0	6.575	4.98	15.3	0.757414
1	6.421	9.14	17.8	1.423454
2	7.185	4.03	17.8	0.560891
3	6.998	2.94	18.7	0.420120
4	7.147	5.33	18.7	0.745767

In [21]:

```
corr_matrix=housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[21]:

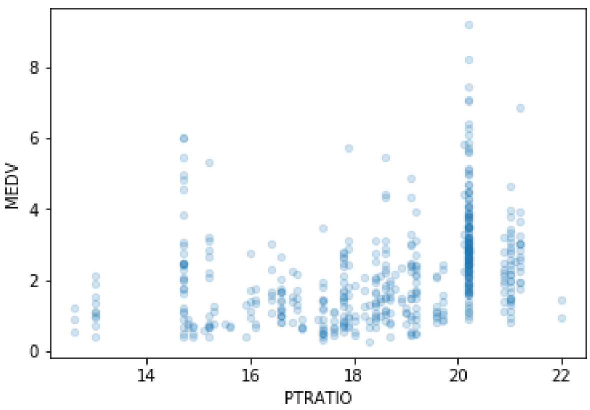
```
MEDV        1.000000
LSTAT       0.980709
PTRATIO     0.336583
RM         -0.695559
Name: MEDV, dtype: float64
```

In [22]:

```
housing.plot(kind='scatter',x='PTRATIO',y='MEDV',alpha=0.2)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xf1343f0>
```



In [23]:

```
##missing attributes
##1. get rid of the missing data points
##2.get rid of the whole attributes
##3.set the value to same value(0,mean or median)
```

In [24]:

```
a=housing.dropna(subset=['RM'])
a.shape
```

Out[24]:

```
(489, 4)
```

In [25]:

```
housing.drop("RM",axis=1).shape##option 2
#NOTE that there is no RM column also note that the original housing dataframe will remain unchang
es
```

Out[25]:

```
(489, 3)
```

In [26]:

```
median=housing["RM"].median()## option 3
housing["RM"].fillna(median)##option 3
```

Out[26]:

```
0      6.575
1      6.421
2      7.185
3      6.998
4      7.147
5      6.430
6      6.012
7      6.172
8      5.631
9      6.004
10     6.377
11     6.009
12     5.889
13     5.949
14     6.096
15     5.834
16     5.935
17     5.990
18     5.456
19     5.727
20     5.570
21     5.965
22     6.142
23     5.813
24     5.924
25     5.599
26     5.813
27     6.047
28     6.495
29     6.674
       ...
459    6.484
460    5.304
461    6.185
462    6.229
463    6.242
464    6.750
465    7.061
466    5.762
467    5.871
468    6.312
469    6.114
470    5.905
471    5.454
472    5.414
473    5.093
474    5.983
475    5.983
476    5.707
477    5.926
478    5.670
479    5.390
480    5.794
481    6.019
```

```
482    5.569
483    6.027
484    6.593
485    6.120
486    6.976
487    6.794
488    6.030
Name: RM, Length: 489, dtype: float64
```

In [27]:

```
housing.shape
##before we started filling missing attributes
housing.describe()
```

Out[27]:

	RM	LSTAT	PTRATIO	MEDV
count	489.000000	489.000000	489.000000	489.000000
mean	6.240288	12.939632	18.516564	2.172089
std	0.643650	7.081990	2.111268	1.376131
min	3.561000	1.980000	12.600000	0.281891
25%	5.880000	7.370000	17.400000	1.116355
50%	6.185000	11.690000	19.100000	1.945874
75%	6.575000	17.120000	20.200000	2.820874
max	8.398000	37.970000	22.000000	9.175930

In [28]:

```
##imputer to solve missing data
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(strategy="median")
imputer.fit(housing)
```

Out[28]:

```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [29]:

```
imputer.statistics_
```

Out[29]:

```
array([ 6.185     , 11.69      , 19.1       ,  1.94587415])
```

In [30]:

```
##fit for new values in dataframe
X=imputer.transform(housing)
housing_tr=pd.DataFrame(X,columns=housing.columns)
housing.describe()
```

Out[30]:

	RM	LSTAT	PTRATIO	MEDV
count	489.000000	489.000000	489.000000	489.000000
mean	6.240288	12.939632	18.516564	2.172089
std	0.643650	7.081990	2.111268	1.376131
min	3.561000	1.980000	12.600000	0.281891
25%	5.880000	7.370000	17.400000	1.116355

	RM	LSTAT	PTRATIO	MEDV
50%	6.185000	11.690000	19.100000	1.945874
75%	6.575000	17.120000	20.200000	2.820874
max	8.398000	37.970000	22.000000	9.175930

In [31]:

```
##Scikit-learn Design
```

In [32]:

```
'''
Primarily three types of objets
1)Esrimators->it estimates some parameters based on a dataset .eg.imputer it has a fit method and transform method .
Fit method - fits the datsets and calculates inetrnal parameters internal parameters
2)Tranforms->transform method takes input and returns output based on the learnings from fit().it also has a convenience function
called fit_transform() which fits and then transform.
3)Predictors->LinearRegression model is an example of predictor .fit() and predict() are two comman functions
its also gives score() function which  will evalute the predictions.
'''
```

Out[32]:

```
'\nPrimarily three types of objets\n1)Esrimators->it estimates some parameters based on a dataset .eg.imputer it has a fit method and transform method . \nFit method - fits the datsets and calculates inetrnal parameters internal parameters\n2)Tranforms->transform method takes input and returns output based on the learnings from fit().it also has a convenience function \ncalled fit_transform() which fits and then transform.\n3)Predictors->LinearRegression model is an example of predictor .fit() and predict() are two comman functions \nits also gives score() function which  will evalute the predictions.\n'
```

In [33]:

```
##Feature scaling
'''Primary two types of fearture scaling method:
1)Min-max scaling (Normalization)->(value - min)/(max - min)
    sklearn provired a class called minMaxscaler for this
2)Standardization-> (value-mean)/std
    sklearn provides a class called standardscaler for this
'''
```

Out[33]:

```
'Primary two types of fearture scaling method:\n1)Min-max scaling (Normalization)->(value - min)/(max - min)\n    sklearn provired a class called minMaxscaler for this\n2)Standardization-> (value-mean)/std\n    sklearn provides a class called standardscaler for this\n'
```

creating a pipeline

In [34]:

```
##Creating a Pipeline

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline =Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
                      ])
```

In [35]:

```
housing_num_tr=my_pipeline.fit_transform(housing_tr)
```

In [36]:

```
housing_num_tr.shape
```

Out[36]:

```
(489, 4)
```

In [64]:

```
##Selecting a desired model for Dargon real Estates
#housing=strat_train_set.drop("MEDV",axis=1)
#housing_lables=strat_train_set["MEDV"].copy()
from sklearn.linear_model import LinearRegression
from sklearn.tree import  DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
#model=LinearRegression()
model=DecisionTreeRegressor()
#model=RandomForestRegressor()
model.fit( housing_num_tr[1:4],housing_num_tr[1:4])
```

Out[64]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      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='best')
```

In [65]:

```
some_data=housing.iloc[:5]
some_lables=housing.iloc[:5]
```

In [66]:

```
prepared_data=my_pipeline.transform(some_data)
```

In [67]:

```
model.predict(prepared_data)
```

Out[67]:

```
array([[ 0.28104837, -0.53706982, -0.33974768, -0.54457113],
       [ 0.28104837, -0.53706982, -0.33974768, -0.54457113],
       [ 1.46924486, -1.25935736, -0.33974768, -1.17201633],
       [ 1.17841666, -1.41342652,  0.08697308, -1.27441563],
       [ 1.46924486, -1.25935736, -0.33974768, -1.17201633]])
```

In [68]:

```
list(some_lables)
```

Out[68]:

```
['RM', 'LSTAT', 'PTRATIO', 'MEDV']
```

In [69]:

```
##Evaluating the model
from sklearn.metrics import mean_squared_error
housing_predictions=model.predict(housing_num_tr)
lin_mse=mean_squared_error(housing_num_tr,housing_predictions)
line_rmse=np.sqrt(lin_mse)
lin_mse
```

Out[69]:

```
1.1419524637789777
```

In [70]:

```
##model doing overfitting
## cross validation using better evaluation technique->
```

In [71]:

```
from sklearn.model_selection import cross_val_score
scores=cross_val_score(model,housing_num_tr,housing,scoring="neg_mean_squared_error",cv=10)
rmse_scores=np.sqrt(-scores)
```

In [72]:

```
rmse_scores
```

Out[72]:

```
array([0.57183927, 0.34159461, 0.7749851 , 0.62028285, 0.51406886,
       0.53238782, 0.40513897, 0.58460316, 0.35741065, 0.41278805])
```

In [73]:

```
##decision Tree aglorithms
##

def print_scores(scores):
    print("scores:",scores)
    print("mean:",scores.mean())
    print("Standard:",scores.std())
```

In [74]:

```
print_scores(rmse_scores)
```

```
scores: [0.57183927 0.34159461 0.7749851  0.62028285 0.51406886 0.53238782
 0.40513897 0.58460316 0.35741065 0.41278805]
mean: 0.5115099363313181
Standard: 0.12820062056618592
```

In [50]:

```
## Linear  Regression
mean: 1.1455494716546406
Standard: 0.12482084643975724
```

In [63]:

```
##RandomForestRegreassion
mean: 0.39351936973647705
Standard: 0.1389670708389857
```

In [75]:

```
##Decision Tree
mean: 0.5115099363313181
Standard: 0.12820062056618592
```

In [78]:

```
#create a another file to save model model create.
from joblib import  dump ,load
dump(model,'Dragon.joblib')
```

Out[78]:

```
['Dragon.joblib']
```

In [80]:

```
##model testing on the test data
'''x_test=strat_test_set.drop("MEDV",axis=1)
y_test=strat_test_set["MEDV"].copy()
x_test_prepared=my_pipeline.transform(x_test)
final_preditions=model.predict(x_test_prepared)
final_mse=mean_squared_error(y_test,final_preditions)
final_rmse=np.sqrt(final_mse)
final_rmse
'''
```

Out[80]:

```
'x_test=strat_test_set.drop("MEDV",axis=1)\ny_test=strat_test_set["MEDV"].copy()\nx_test_prepared=m
peline.transform(x_test)\nfinal_preditions=model.predict(x_test_prepared)\nfinal_mse=mean_squared_e
(y_test,final_preditions)\nfinal_rmse=np.sqrt(final_mse)\nfinal_rmse\n'
```

In [86]:

```
from joblib import  dump,load
import  numpy as np
model=load('Dragon.joblib')
features=np.array([[ 0.28104837, -0.53706982, -0.33974768, -0.54457113],
       [ 0.28104837, -0.53706982, -0.33974768, -0.54457113],
       [ 1.46924486, -1.25935736, -0.33974768, -1.17201633],
       [ 1.17841666, -1.41342652,  0.08697308, -1.27441563],
       [ 1.46924486, -1.25935736, -0.33974768, -1.17201633]])
model.predict(features)
```

Out[86]:

```
array([[ 0.28104837, -0.53706982, -0.33974768, -0.54457113],
       [ 0.28104837, -0.53706982, -0.33974768, -0.54457113],
       [ 1.46924486, -1.25935736, -0.33974768, -1.17201633],
       [ 1.17841666, -1.41342652,  0.08697308, -1.27441563],
       [ 1.46924486, -1.25935736, -0.33974768, -1.17201633]])
```

In [87]:

In [ ]: