Dragon Real Estate-Price Predictor

In [1]:
```
import pandas as pd
```

In [2]:
```
housing=pd.read_csv('data.csv')
```

In [3]:
```
#housing.head()
```

In [4]:
```
#housing.info()
```

In [5]:
```
#housing['CHAS'].value_counts()
```

In [6]:
```
#housing.describe()
```

In [7]:
```
#%matplotlib inline
```

In [8]:
```
#import matplotlib.pyplot as plt
#housing.hist(bins=50,figsize=(20,15))
```

Train-Test Splitting

In [9]:
```
#import numpy as np
#def split_train_test(data,test_ratio):
    #np.random.seed()
    #shuffled=np.random.permutation(len(data))
    #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 [10]:
```
#train_ser,test_set=split_train_test(housing,0.2)
```

In [11]:
```
from sklearn.model_selection import train_test_split
train_set,test_set=train_test_split(housing,test_size=0.2,random_state=
42)
```

In [12]:
```
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['CHAS']):
    strat_train_set=housing.loc[train_index]
    strat_test_set=housing.loc[test_index]
```

In [13]:
```
#strat_test_set['CHAS'].value_counts()
housing=strat_train_set.copy()
```

Looking for Correlations

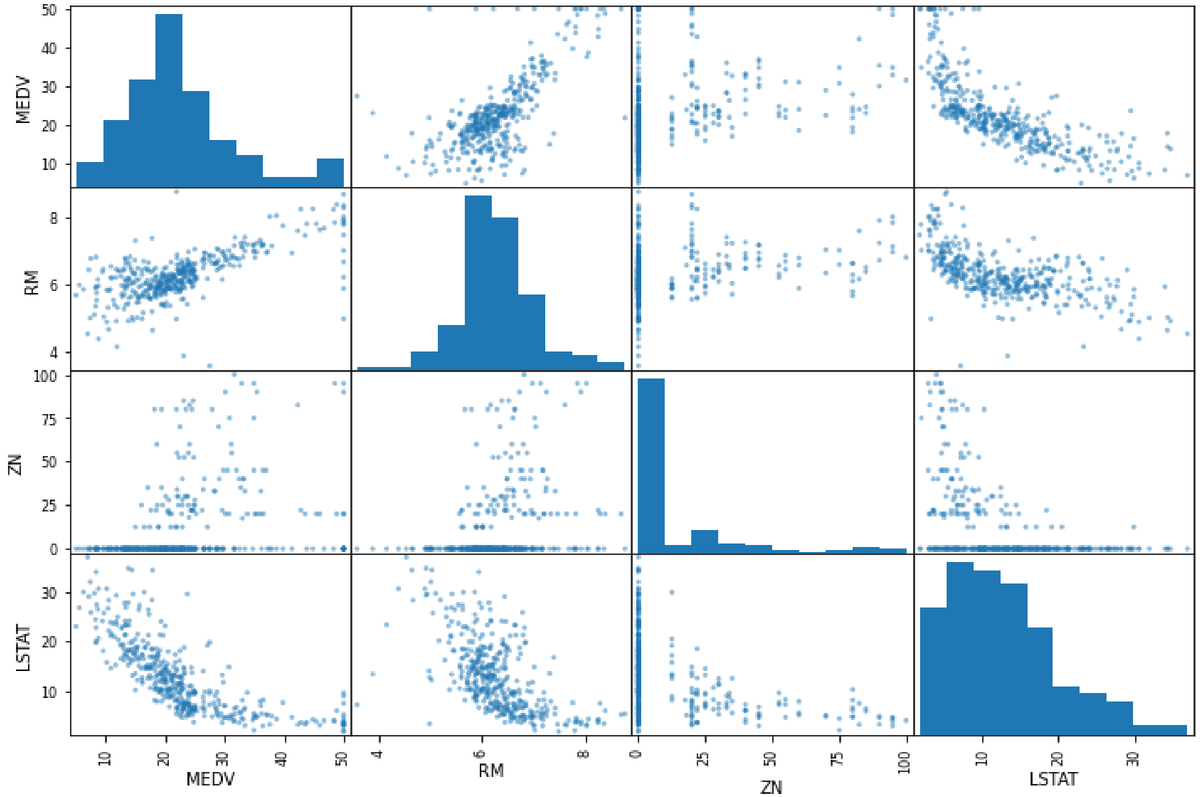
In [14]:
```
corr_matrix=housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[14]:
```
MEDV       1.000000
RM         0.680857
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
```

```
PTRATIO   -0.493534
LSTAT     -0.740494
Name: MEDV, dtype: float64
```

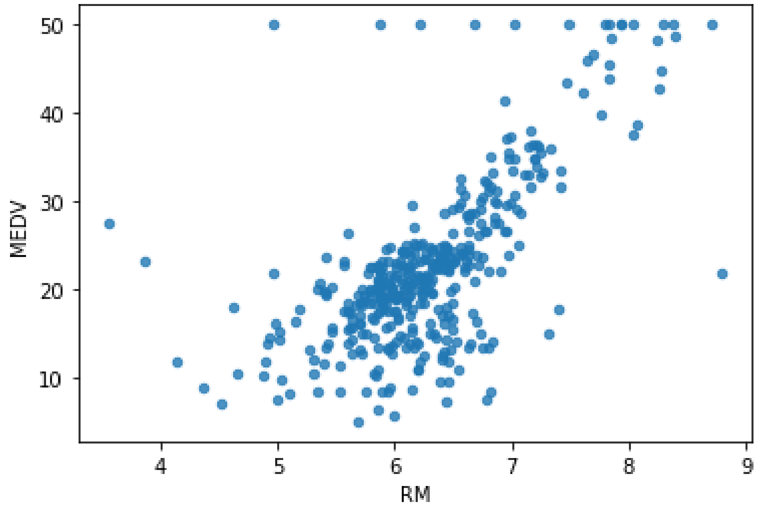
In [15]:
```
from pandas.plotting import scatter_matrix
attributes=['MEDV','RM','ZN','LSTAT']
scatter_matrix(housing[attributes],figsize=(12,8))
```

Out[15]:
```
array([[<AxesSubplot:xlabel='MEDV', ylabel='MEDV'>,
        <AxesSubplot:xlabel='RM', ylabel='MEDV'>,
        <AxesSubplot:xlabel='ZN', ylabel='MEDV'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='MEDV'>],
       [<AxesSubplot:xlabel='MEDV', ylabel='RM'>,
        <AxesSubplot:xlabel='RM', ylabel='RM'>,
        <AxesSubplot:xlabel='ZN', ylabel='RM'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='RM'>],
       [<AxesSubplot:xlabel='MEDV', ylabel='ZN'>,
        <AxesSubplot:xlabel='RM', ylabel='ZN'>,
        <AxesSubplot:xlabel='ZN', ylabel='ZN'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='ZN'>],
       [<AxesSubplot:xlabel='MEDV', ylabel='LSTAT'>,
        <AxesSubplot:xlabel='RM', ylabel='LSTAT'>,
        <AxesSubplot:xlabel='ZN', ylabel='LSTAT'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='LSTAT'>]], dtype=object)
```



In [16]:
```
housing.plot(kind='scatter',x='RM',y='MEDV',alpha=0.8)
```

Out[16]: <AxesSubplot:xlabel='RM', ylabel='MEDV'>



Trying out Attribute Combinations

In [17]:
```
housing['TAXRM']=housing['TAX']/housing['RM']
```

In [18]:
```
housing['TAXRM']
```

Out[18]:
```
254     51.571709
348     42.200452
476    102.714374
321     45.012547
326     45.468948
          ...
155     65.507152
423    109.126659
98      35.294118
455    102.068966
216     46.875000
Name: TAXRM, Length: 404, dtype: float64
```

In [19]:
```
housing.head()
```

Out[19]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6

In [20]:

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

Out[20]:

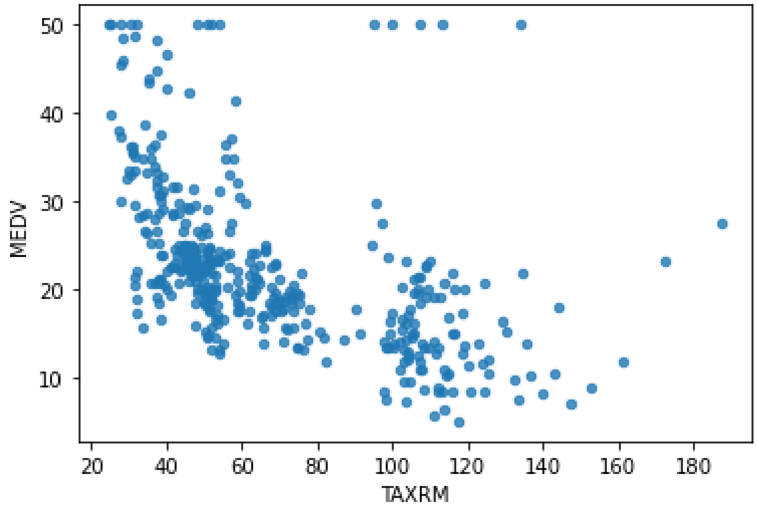
```
MEDV       1.000000
RM         0.680857
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
PTRATIO   -0.493534
TAXRM     -0.528626
LSTAT     -0.740494
Name: MEDV, dtype: float64
```

In [21]:

```
housing.plot(kind='scatter',x='TAXRM',y='MEDV',alpha=0.8)
```

Out[21]:

```
<AxesSubplot:xlabel='TAXRM', ylabel='MEDV'>
```



In [22]:
```
housing=strat_train_set.drop('MEDV',axis=1)
housing_labels=strat_train_set['MEDV'].copy()
```

Missing Attributes

In [23]:
```
#To take care of missing attributes, you have three options
#1.Get rid of the missing data points
#2.Get rid of the whole attribute
#3.Set the value to some value(zero,mean,median)
```

In [24]:
```
a=housing.dropna(subset=["RM"]) #option 1
a.shape
```

Out[24]: (399, 13)

In [25]:
```
housing.drop('RM',axis=1) #option 2
#Note there is no RM column
```

Out[25]:

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

	CRIM	ZN	INDUS	CHAS	NOX	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
254	0.04819	80.0	3.64	0	0.392	32.0	9.2203	1	315	16.4	392.89	6.57
348	0.01501	80.0	2.01	0	0.435	29.7	8.3440	4	280	17.0	390.94	5.99
476	4.87141	0.0	18.10	0	0.614	93.6	2.3053	24	666	20.2	396.21	18.68
321	0.18159	0.0	7.38	0	0.493	54.3	4.5404	5	287	19.6	396.90	6.87
326	0.30347	0.0	7.38	0	0.493	28.9	5.4159	5	287	19.6	396.90	6.15
...	...	...	...	...	...	...	...	...	...	...	...	...
155	3.53501	0.0	19.58	1	0.871	82.6	1.7455	5	403	14.7	88.01	15.02
423	7.05042	0.0	18.10	0	0.614	85.1	2.0218	24	666	20.2	2.52	23.29
98	0.08187	0.0	2.89	0	0.445	36.9	3.4952	2	276	18.0	393.53	3.57
455	4.75237	0.0	18.10	0	0.713	86.5	2.4358	24	666	20.2	50.92	18.13
216	0.04560	0.0	13.89	1	0.550	56.0	3.1121	5	276	16.4	392.80	13.51

404 rows × 12 columns

In [26]:
```
median=housing['RM'].median() #Compute median for option three
```

In [27]:
```
housing['RM'].fillna(median)
```

Out[27]:
```
254    6.108
348    6.635
476    6.484
321    6.376
326    6.312
       ...
155    6.152
423    6.103
98     7.820
455    6.525
216    5.888
Name: RM, Length: 404, dtype: float64
```

In [28]:
```
housing.shape
```
Out[28]: (404, 13)

In [29]:
```
housing.describe()
```
Out[29]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	404.000000	404.000000	404.000000	404.000000	404.000000	399.000000	404.000000	404.(
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.279481	69.039851	3.:
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.716784	28.258248	2.(
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.'
25%	0.086963	0.000000	5.190000	0.000000	0.453000	5.876500	44.850000	2.(
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.209000	78.200000	3.'
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630500	94.100000	5.'
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.'

In [30]:
```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(strategy='median')
imputer.fit(housing)
```
Out[30]: SimpleImputer(strategy='median')

In [31]:
```
imputer.statistics_.shape
```
Out[31]: (13,)

In [32]:
```
X=imputer.transform(housing)
```

In [33]:
```
housing_tr=pd.DataFrame(X,columns=housing.columns)
```

In [34]:
```
housing_tr.describe()
```

Out[34]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.(
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.278609	69.039851	3.:
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.712366	28.258248	2.(
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.:
25%	0.086963	0.000000	5.190000	0.000000	0.453000	5.878750	44.850000	2.(
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.209000	78.200000	3.:
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630000	94.100000	5.:
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.:

Scikit-learn Design

Primarily, three types of objects 1. Estimators-It estimates some parameter based on dataset. Eg. imputer It has a fit method and transform method. Fit method- fits the dataset and calculates internal parameters 2. Transformers-Transform method takes input and returns output based on the learnings from fit() It also has a covenience function called fit_transform() which fits and then transforms. 3. Predictors- LinearRegression model is an example of predictor. fit and predict are the two common functions. It also gives score function which will evaluate the predictions.

Feature Scaling

Primarily, two types of scaling methods: 1.Min-max scaling(Normalization) (value-min)/(max-min) Sklearn provides a class called MinMaxScaler for this 2.Standardization (value-mean)/std where std is Standard Deviation Sklearn provides a class called Standard Scaler for this

Creating a Pipeline

In [35]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
my_pipeline=Pipeline([
    ('imputer',SimpleImputer(strategy='median')),
    #...add as many as you want
    ('std_scaler',StandardScaler())
])
```

In [36]:
```
housing_num_tr=my_pipeline.fit_transform(housing)
```

In [37]:
```
housing_num_tr.shape
```

Out[37]: (404, 13)

Selecting a desired model for Dragon Real Estates

In [38]:
```
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,housing_labels)
```

Out[38]: RandomForestRegressor()

In [39]:
```
 some_data=housing.iloc[:5]
```

In [40]:
```
some_data
```

Out[40]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6

In [41]:
```
some_labels=housing_labels.iloc[:5]
```

In [42]:
```
prepared_data=my_pipeline.transform(some_data)
```

In [43]:
```
model.predict(prepared_data)
```

Out[43]:
```
array([22.234, 25.156, 16.681, 23.242, 23.605])
```

In [44]:
```
list(some_labels)
```

Out[44]:
```
[21.9, 24.5, 16.7, 23.1, 23.0]
```

Evaluating the model

In [45]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
housing_predictions=model.predict(housing_num_tr)
mse=mean_squared_error(housing_labels,housing_predictions)
rmse=np.sqrt(mse)
```

In [46]:
```
rmse
```

Out[46]:
```
1.1890649739926897
```

Using better evaluation technique-Cross Validation

In [47]:
```
from sklearn.model_selection import cross_val_score
scores=cross_val_score(model,housing_num_tr,housing_labels,scoring="neg
```

```
_mean_squared_error",cv=10)
rmse_scores=np.sqrt(-scores)
```

In [48]:
```
rmse_scores
```

Out[48]:
```
array([2.88966988, 2.8194387 , 4.51906724, 2.56478567, 3.42599734,
       2.73919311, 4.68853915, 3.31364106, 3.19134686, 3.20368707])
```

In [49]:
```
def print_scores(scores):
    print('Scores:',scores)
    print('Mean:',scores.mean())
    print('Standard Deviation:',scores.std())
```

In [50]:
```
print_scores(rmse_scores)
```

```
Scores: [2.88966988 2.8194387  4.51906724 2.56478567 3.42599734 2.73919
311
 4.68853915 3.31364106 3.19134686 3.20368707]
Mean: 3.335536608109737
Standard Deviation: 0.6850572941919849
```

saving the model

In [52]:
```
from joblib import dump,load
dump(model,'Dragon.joblib')
```

Out[52]:
```
['Dragon.joblib']
```

Testing the Model on test data

In [53]:
```
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_predictions=model.predict(X_test_prepared)
```

```
final_mse=mean_squared_error(Y_test,final_predictions)
final_rmse=np.sqrt(final_mse)
```

In [54]:
```
final_rmse
```

Out[54]: 2.8963498123144786

In [ ]: