Dragon_Real_Estates

January 5, 2021

0.1 Dragon Real Estates - Price Predictor

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

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

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

[3]:
```
      CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.02731  0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
1  0.02729  0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
2  0.03237  0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
3  0.06905  0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7
4  0.02985  0.0   2.18     0  0.458  6.430  58.7  6.0622    3  222     18.7

        B  LSTAT  MEDV
0  396.90   9.14  21.6
1  392.83   4.03  34.7
2  394.63   2.94  33.4
3  396.90   5.33  36.2
4  394.12   5.21  28.7
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     505 non-null    float64
 1   ZN       505 non-null    float64
 2   INDUS    505 non-null    float64
 3   CHAS     505 non-null    int64
 4   NOX      505 non-null    float64
 5   RM       500 non-null    float64
 6   AGE      505 non-null    float64
 7   DIS      505 non-null    float64
 8   RAD      505 non-null    int64
 9   TAX      505 non-null    int64
```

```
 10  PTRATIO  505 non-null    float64
 11  B        505 non-null    float64
 12  LSTAT    505 non-null    float64
 13  MEDV     505 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.4 KB
```

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

[5]:
```
0    470
1     35
Name: CHAS, dtype: int64
```

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

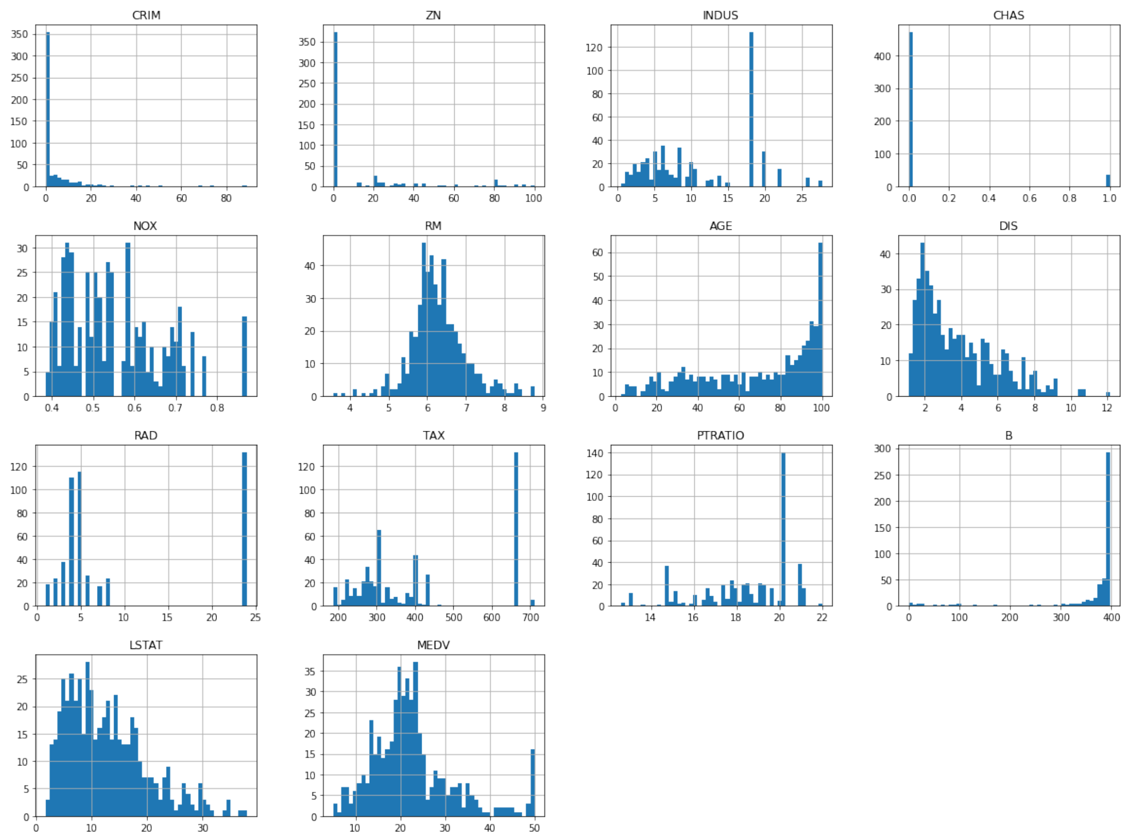
[6]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  505.000000  505.000000  505.000000  505.000000  505.000000  500.000000
mean     3.620667   11.350495   11.154257    0.069307    0.554728    6.285460
std      8.608572   23.343704    6.855868    0.254227    0.115990    0.706416
min      0.009060    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082210    0.000000    5.190000    0.000000    0.449000    5.883000
50%      0.259150    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.678220   12.500000   18.100000    0.000000    0.624000    6.629250
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  505.000000  505.000000  505.000000  505.000000  505.000000  505.000000
mean    68.581584    3.794459    9.566337  408.459406   18.461782  356.594376
std     28.176371    2.107757    8.707553  168.629992    2.162520   91.367787
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.000000    2.100000    4.000000  279.000000   17.400000  375.330000
50%     77.700000    3.199200    5.000000  330.000000   19.100000  391.430000
75%     94.100000    5.211900   24.000000  666.000000   20.200000  396.210000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  505.000000  505.000000
mean    12.668257   22.529901
std      7.139950    9.205991
min      1.730000    5.000000
25%      7.010000   17.000000
50%     11.380000   21.200000
75%     16.960000   25.000000
max     37.970000   50.000000
```

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

[8]:
```
import matplotlib.pyplot as plt
```

[9]:
```
housing.hist(bins=50, figsize=(20,15))
```

[9]:
```
array([[<AxesSubplot:title={'center':'CRIM'}>,
        <AxesSubplot:title={'center':'ZN'}>,
        <AxesSubplot:title={'center':'INDUS'}>,
        <AxesSubplot:title={'center':'CHAS'}>],
       [<AxesSubplot:title={'center':'NOX'}>,
        <AxesSubplot:title={'center':'RM'}>,
        <AxesSubplot:title={'center':'AGE'}>,
        <AxesSubplot:title={'center':'DIS'}>],
       [<AxesSubplot:title={'center':'RAD'}>,
        <AxesSubplot:title={'center':'TAX'}>,
        <AxesSubplot:title={'center':'PTRATIO'}>,
        <AxesSubplot:title={'center':'B'}>],
       [<AxesSubplot:title={'center':'LSTAT'}>,
        <AxesSubplot:title={'center':'MEDV'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



0.2 Train Test Splitting

[10]:
```
import numpy as np
# for learning purupose
def split_train_test(data, test_ratio):
    np.random.seed(42)
    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]
```

[11]:
```
train_set, test_set = split_train_test(housing, 0.2)
```

[12]:
```
# print(f"Rows in train set: {len(train_set)}\nRows in test set:␣
↪{len(test_set)}\n")
```

[13]:
```
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: 404
Rows in test set: 101
```

[14]:
```
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]
```

[15]:
```
strat_train_set['CHAS'].value_counts()
```

[15]:
```
0    376
1     28
Name: CHAS, dtype: int64
```

[16]:
```
# 376/28
```

[17]:
```
# strat_test_set['CHAS'].value_counts()
```

[18]:
```
# 94/7
```

[19]:
```
housing = strat_train_set.copy()
```

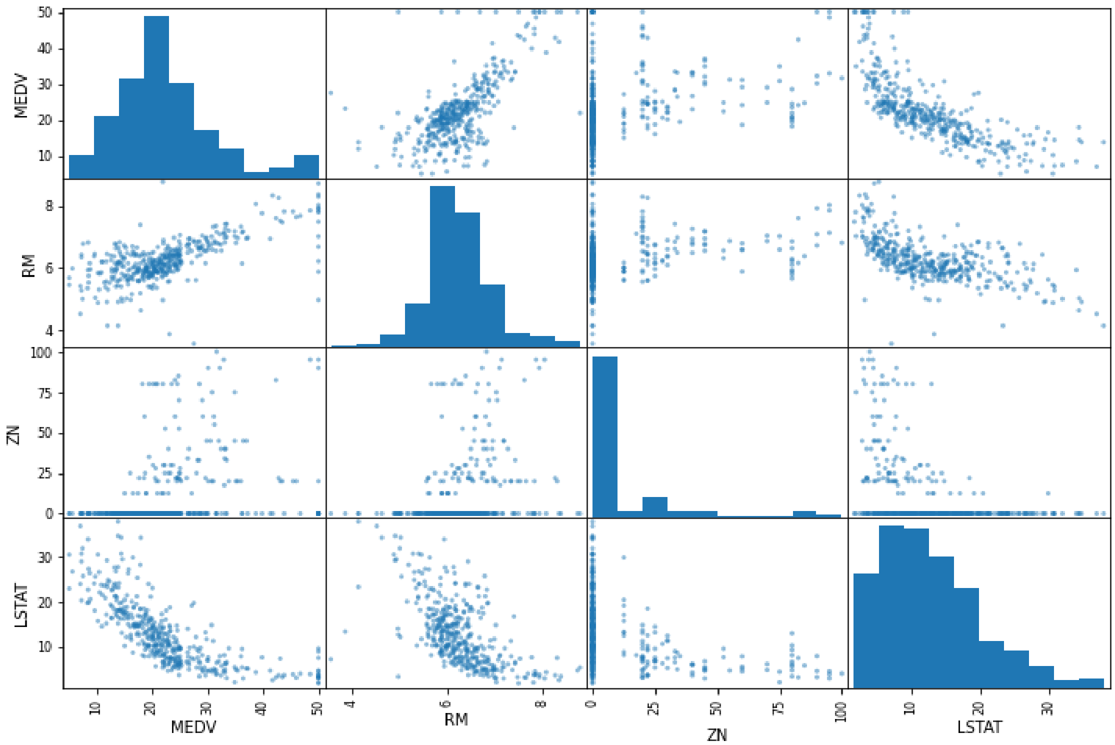
0.3 Looking for Correlations

[20]:
```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

[20]:
```
MEDV       1.000000
RM         0.661599
B          0.344609
ZN         0.329206
DIS        0.231680
CHAS       0.215042
RAD       -0.362619
AGE       -0.378913
CRIM      -0.397993
NOX       -0.421815
TAX       -0.441617
INDUS     -0.448303
PTRATIO   -0.486045
LSTAT     -0.739129
Name: MEDV, dtype: float64
```

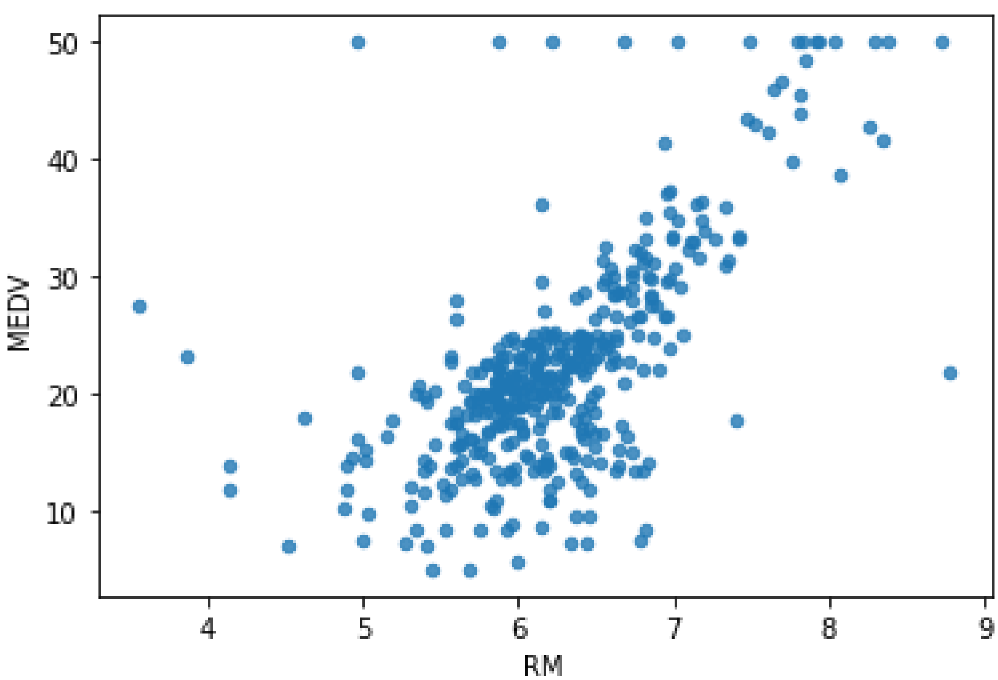
[21]:
```
from pandas.plotting import scatter_matrix
attributes = ['MEDV', 'RM', 'ZN', 'LSTAT']
scatter_matrix(housing[attributes], figsize = (12,8))
```

[21]:
```
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)
```



[22]:
```
housing.plot(kind='scatter', x='RM', y='MEDV', alpha=0.8)
```

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



1 Trying out Attribute Combinations

[23]:
```
housing['TAXRM'] = housing['TAX']/housing['RM']
```

[24]:
```
housing.head()
```

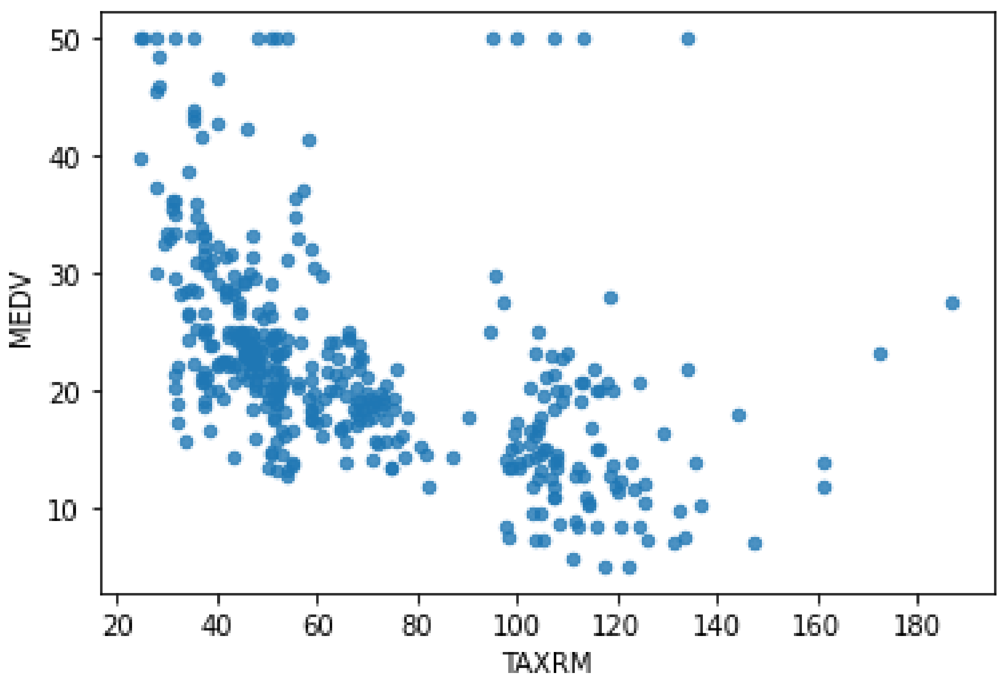
[24]:
```
          CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
254   0.03548  80.0   3.64     0  0.392  5.876  19.1  9.2203    1  315
348   0.02899  40.0   1.25     0  0.429  6.939  34.5  8.7921    1  335
476  15.02340   0.0  18.10     0  0.614  5.304  97.3  2.1007   24  666
321   0.35114   0.0   7.38     0  0.493  6.041  49.9  4.7211    5  287
326   0.24103   0.0   7.38     0  0.493  6.083  43.7  5.4159    5  287

     PTRATIO       B  LSTAT  MEDV       TAXRM
254     16.4  395.18   9.25  20.9   53.607897
348     19.7  389.85   5.89  26.6   48.277850
476     20.2  349.48  24.91  12.0  125.565611
321     19.6  396.90   7.70  20.4   47.508691
326     19.6  396.90  12.79  22.2   47.180667
```

[25]:
```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

[25]:
```
MEDV       1.000000
RM         0.661599
B          0.344609
ZN         0.329206
DIS        0.231680
CHAS       0.215042
RAD       -0.362619
AGE       -0.378913
CRIM      -0.397993
NOX       -0.421815
TAX       -0.441617
INDUS     -0.448303
PTRATIO   -0.486045
TAXRM     -0.509117
LSTAT     -0.739129
Name: MEDV, dtype: float64
```

[26]:
```
housing.plot(kind='scatter', x='TAXRM', y='MEDV', alpha=0.8)
```

[26]:
```
<AxesSubplot:xlabel='TAXRM', ylabel='MEDV'>
```




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

1.1 Missing Attributes

```
[28]: # to take care of missing attributes
      # 1. Get rid off the missing data points
      # 2. Get rid off the whole attribute
      # 3. Set the value to some value (0, mean or meadian)
```

```
[29]: a= housing.dropna(subset=['RM']) #option 1
      a.shape
      # note that the original housing df will remain unchanged
```

```
[29]: (401, 13)
```

```
[30]: housing.drop('RM', axis=1).shape #option 2
      # note that there is no RM column and also note that the original housing df␣
       ↪will remain unchanged
```

```
[30]: (404, 12)
```

```
[31]: median = housing['RM'].median() #compute median for option 3
```

[32]:
```python
housing['RM'].fillna(median) #option 3
# note that the original housing df will remain unchanged
```

[32]:
```
254    5.876
348    6.939
476    5.304
321    6.041
326    6.083
       …
154    6.152
423    5.565
98     7.416
455    5.976
215    5.888
Name: RM, Length: 404, dtype: float64
```

[33]:
```python
housing.shape
```

[33]:
```
(404, 13)
```

[34]:
```python
housing.describe() # before we started filling missing attributes
```

[34]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	\
count	404.000000	404.000000	404.000000	404.000000	404.000000	401.000000	
mean	3.680733	10.189356	11.305965	0.069307	0.557274	6.251918	
std	8.249705	21.930822	6.817698	0.254290	0.116503	0.691261	
min	0.009060	0.000000	0.740000	0.000000	0.385000	3.561000	
25%	0.090060	0.000000	5.190000	0.000000	0.452000	5.874000	
50%	0.290250	0.000000	9.900000	0.000000	0.538000	6.176000	
75%	3.694070	3.125000	18.100000	0.000000	0.625750	6.606000	
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	

	AGE	DIS	RAD	TAX	PTRATIO	B	\
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	
mean	68.548020	3.778549	9.702970	411.428218	18.502723	353.522649	
std	28.433028	2.125958	8.754489	168.237476	2.117437	95.111003	
min	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	
25%	44.850000	2.070275	4.000000	284.000000	17.400000	374.237500	
50%	77.500000	3.167500	5.000000	336.000000	19.050000	390.940000	
75%	94.600000	5.104475	24.000000	666.000000	20.200000	396.157500	
max	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	

	LSTAT
count	404.000000
mean	12.833292
std	7.199418
min	1.730000

```
25%       7.362500
50%      11.570000
75%      16.977500
max      37.970000
```

[35]:
```python
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = 'median')
imputer.fit(housing)
```

[35]:
```
SimpleImputer(strategy='median')
```

[36]:
```python
imputer.statistics_
```

[36]:
```
array([2.9025e-01, 0.0000e+00, 9.9000e+00, 0.0000e+00, 5.3800e-01,
       6.1760e+00, 7.7500e+01, 3.1675e+00, 5.0000e+00, 3.3600e+02,
       1.9050e+01, 3.9094e+02, 1.1570e+01])
```

[37]:
```python
x = imputer.transform(housing)
```

[38]:
```python
housing_tr = pd.DataFrame(x, columns= housing.columns)
```

[39]:
```python
housing_tr.describe() #now rm count will increase from 500 to 501
```

[39]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  404.000000  404.000000  404.000000  404.000000  404.000000  404.000000
mean     3.680733   10.189356   11.305965    0.069307    0.557274    6.251354
std      8.249705   21.930822    6.817698    0.254290    0.116503    0.688714
min      0.009060    0.000000    0.740000    0.000000    0.385000    3.561000
25%      0.090060    0.000000    5.190000    0.000000    0.452000    5.874750
50%      0.290250    0.000000    9.900000    0.000000    0.538000    6.176000
75%      3.694070    3.125000   18.100000    0.000000    0.625750    6.604500
max     73.534100  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  404.000000  404.000000  404.000000  404.000000  404.000000  404.000000
mean    68.548020    3.778549    9.702970  411.428218   18.502723  353.522649
std     28.433028    2.125958    8.754489  168.237476    2.117437   95.111003
min      2.900000    1.129600    1.000000  187.000000   13.000000    0.320000
25%     44.850000    2.070275    4.000000  284.000000   17.400000  374.237500
50%     77.500000    3.167500    5.000000  336.000000   19.050000  390.940000
75%     94.600000    5.104475   24.000000  666.000000   20.200000  396.157500
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT
count  404.000000
mean    12.833292
std      7.199418
min      1.730000
```

```
25%       7.362500
50%      11.570000
75%      16.977500
max      37.970000
```

1.2 ScikitLearn Design

Primarily, three types of objects 1. Estimators- It estimates some parameter based on a dataset. Eg imputer. It has a fit method and transform method.

1.3 Feature Scaling

Primarily, two types of feature scaling methods: 1. Min-Max scaling (Normalization) (value - min)/(max - min) ranges from 0 to 1 Sklearn proivdes a class MinMaxScaler for this

2. Standardization (value - mean)/std Sklearn provides a class called Standard Scaler for this

1.4 Creating a Pipeline

[40]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median')),
    #     ........ add as many as you want in your pipeline
    ('std_scaler', StandardScaler()),
])
```

[41]:
```
housing_num_tr = my_pipeline.fit_transform(housing)
```

[42]:
```
housing_num_tr.shape
```

[42]:
```
(404, 13)
```

1.5 Selecting a desired model for Dragon Real Estates

[44]:
```
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)
```

[44]:
```
RandomForestRegressor()
```

[45]:
```
some_data = housing.iloc[:5]
```

[46]:
```
some_labels = housing_labels.iloc[:5]
```

[47]:
```
prepared_data = my_pipeline.transform(some_data)
```

[48]:
```
model.predict(prepared_data)
```

[48]:
```
array([20.717, 27.434, 12.339, 21.093, 22.174])
```

[49]:
```
list(some_labels)
```

[49]:
```
[20.9, 26.6, 12.0, 20.4, 22.2]
```

1.6 Evaluating the model

[50]:
```
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)
```

[51]:
```
rmse
```

[51]:
```
1.1751382402353208
```

[52]:
```
#due to its high mean sq error we will discard this model and we'll use␣
↪decision tree regressor
# but decision tree regressor does overfiiting that's why mse is 0
```

1.7 Using Better Evaluation Technique- Cross Valdiation

[53]:
```
# 1 2 3 4 5 6 7 8 9 10
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)
```

[54]:
```
rmse_scores
```

[54]:
```
array([3.21732103, 2.53358929, 5.07545782, 2.7368161 , 2.69396689,
       2.53952812, 2.75901019, 2.99910886, 1.99235371, 4.29029414])
```

[55]:
```
def print_scores(scores):
    print('Scores: ', scores)
    print('Mean: ', scores.mean())
    print('Std Dev: ', scores.std())
```

[56]:
```
print_scores(rmse_scores)
```

```
Scores:  [3.21732103 2.53358929 5.07545782 2.7368161  2.69396689 2.53952812
 2.75901019 2.99910886 1.99235371 4.29029414]
```

```
Mean:  3.0837446166561895
Std Dev:  0.8726628328940665
```

1.8 Saving the model

[57]:
```
from joblib import dump, load
dump(model, 'Dragon.joblib')
```

[57]:
```
['Dragon.joblib']
```

1.9 Testing the model on test data

[61]:
```
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)
print(final_predictions, list(y_test))
```

```
[22.774 22.234 46.452 32.708 45.269 34.833 20.967 23.491 32.813 19.749
 19.401 30.646 22.038 33.504 20.447 21.997 12.351 21.29  28.141 19.577
 20.035 45.194 12.158 19.231 26.17  34.225 16.545 15.826  6.566 20.523
 23.558 23.228 18.153 15.227 20.715 18.828 22.893 17.532 45.359 17.284
 21.484 18.683 19.441 18.604 33.154  8.203 24.735 14.436 21.167 21.333
 45.969 23.799 14.994 21.599 19.767 46.899 33.329 20.024 35.048 10.564
 23.8   35.34  33.224 23.76  14.238 20.778 21.026 15.79  27.983 24.352
 23.375 32.209 19.282 31.91  11.048 20.14  42.456 19.75  19.804 14.03
 41.829  8.996 35.687 23.043 28.801 15.961 23.127 21.926 20.614 16.192
 26.234  9.877 32.011 12.73  25.99  20.553 33.358 13.687 21.429 21.372
 20.813] [24.6, 22.0, 44.8, 23.6, 48.8, 36.5, 19.7, 23.1, 34.6, 21.5, 23.1,
15.0, 23.0, 34.9, 18.5, 10.4, 10.2, 18.9, 23.9, 19.3, 19.4, 48.3, 10.9, 19.6,
27.5, 37.3, 16.1, 15.2, 10.5, 21.4, 23.2, 20.7, 21.7, 13.0, 22.3, 19.6, 21.2,
18.1, 50.0, 23.7, 22.6, 20.5, 18.9, 19.5, 32.7, 8.8, 29.1, 19.0, 22.6, 21.2,
50.0, 22.5, 17.8, 20.3, 20.4, 37.6, 35.4, 18.2, 33.3, 12.1, 23.1, 37.9, 36.1,
23.7, 13.1, 23.8, 19.6, 13.1, 27.9, 27.0, 22.9, 31.7, 17.1, 30.3, 8.1, 19.6,
44.0, 19.5, 18.5, 17.2, 35.2, 8.3, 34.7, 20.5, 23.7, 14.2, 22.8, 20.6, 19.6,
15.2, 23.9, 6.3, 32.0, 13.4, 22.0, 19.9, 28.7, 19.1, 23.4, 11.9, 21.7]
```

[59]:
```
final_rmse
```

[59]:
```
3.4284325619272673
```

[63]:
```
prepared_data[0]
```

[63]:
```
array([-0.44241248,  3.18716752, -1.12581552, -0.27288841, -1.42038605,
       -0.54568298, -1.7412613 ,  2.56284386, -0.99534776, -0.57387797,
       -0.99428207,  0.43852974, -0.49833679])
```

1.10 Using the model

[64]:
```
from joblib import dump, load
import numpy as np
model = load('Dragon.joblib')
features = np.array([[-0.44241248,  34.18716752, -1.12581552, -0.27288841, -1.
↪42038605,
       -99.54568298, -122.7412613 ,  9.56284386, -0.99534776, -0.57387797,
       -0.99428207,  0.43852974, -9.49833679]])
model.predict(features)
```

[64]:
```
array([23.91])
```

[ ]:
```
```