Real Estate

June 23, 2022

0.1 Real Estate Price Predictor

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

[2]:
```python
housing = pd.read_csv("D:\Final Projects\Real Estate Price Prediction␣
↪Model\Dataset.csv")
```

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

[3]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        B  LSTAT  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       501 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
```

```
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

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

[5]:
```
0    471
1     35
Name: CHAS, dtype: int64
```

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

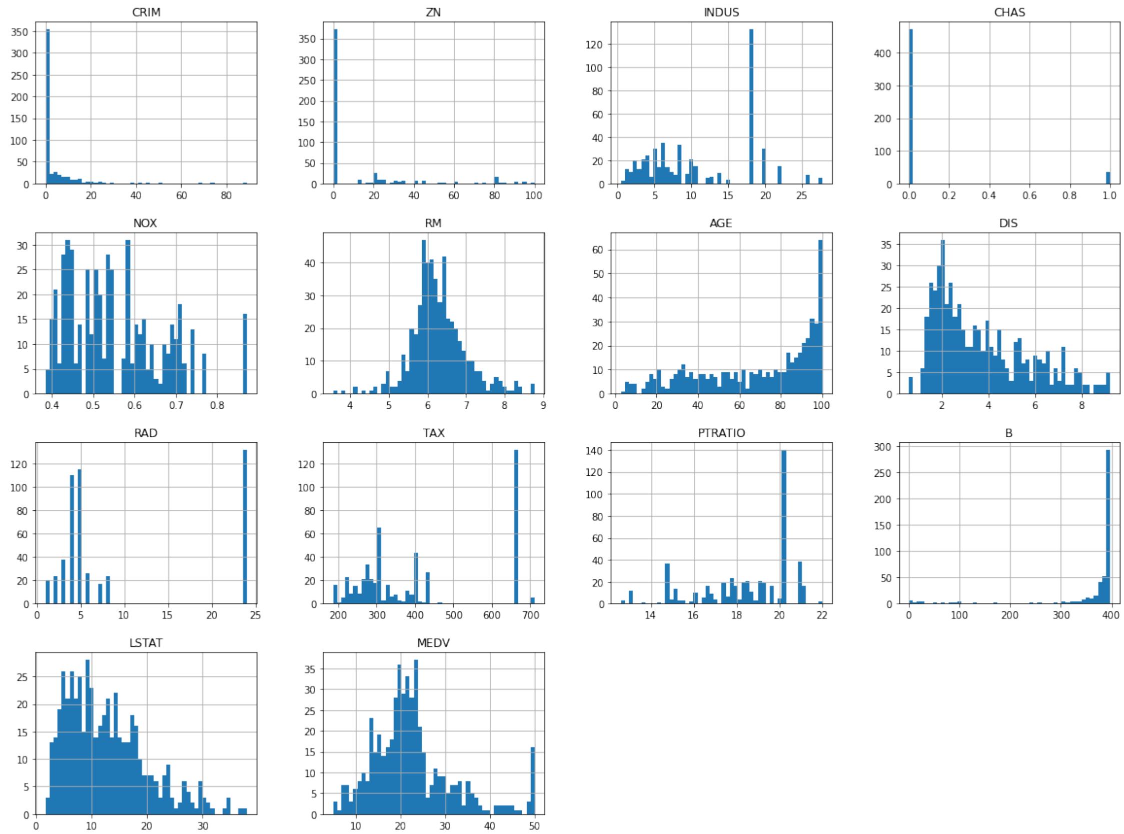
[6]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  501.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.286385
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.705648
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885000
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.209000
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.629000
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.696228    9.549407  408.237154   18.455534  356.674032
std     28.148861    1.999689    8.707259  168.537116    2.164946   91.294864
min      2.900000    0.585700    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.073700    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.107300    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.112625   24.000000  666.000000   20.200000  396.225000
max    100.000000    9.222900   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   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]:
```python
import numpy as np
def split_train_test(data,test_ratio):
    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]:
```python
train_set, test_set = split_train_test(housing, 0.2)
```

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

```
Rows in train set: 405
Rows in test set: 101
```

[13]:
```python
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: 102
```

[14]:
```python
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]:
```python
strat_test_set
```

[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
342	0.02498	0.0	1.89	0	0.518	6.540	59.7	6.2669	1	422	
379	17.86670	0.0	18.10	0	0.671	6.223	100.0	1.3861	24	666	
223	0.61470	0.0	6.20	0	0.507	6.618	80.8	3.2721	8	307	
219	0.11425	0.0	13.89	1	0.550	6.373	92.4	3.3633	5	276	
48	0.25387	0.0	6.91	0	0.448	5.399	95.3	5.8700	3	233	
..	…	…	…	…	…	…	…	…	…	…	
88	0.05660	0.0	3.41	0	0.489	7.007	86.3	3.4217	2	270	
466	3.77498	0.0	18.10	0	0.655	5.952	84.7	2.8715	24	666	
52	0.05360	21.0	5.64	0	0.439	6.511	21.1	6.8147	4	243	
121	0.07165	0.0	25.65	0	0.581	6.004	84.1	2.1974	2	188	
218	0.11069	0.0	13.89	1	0.550	5.951	93.8	2.8893	5	276	

```
     PTRATIO       B  LSTAT  MEDV
342     15.9  389.96   8.65  16.5
379     20.2  393.74  21.78  10.2
223     17.4  396.90   7.60  30.1
219     16.4  393.74  10.50  23.0
48      17.9  396.90  30.81  14.4
..       …       …     …     …
88      17.8  396.90   5.50  23.6
466     20.2   22.01  17.15  19.0
52      16.8  396.90   5.28  25.0
121     19.1  377.67  14.27  20.3
218     16.4  396.90  17.92  21.5

[102 rows x 14 columns]
```

[16]:
```
housing = strat_train_set.copy()
```

Correlations

[17]:
```
corr_matrix = housing.corr()
```

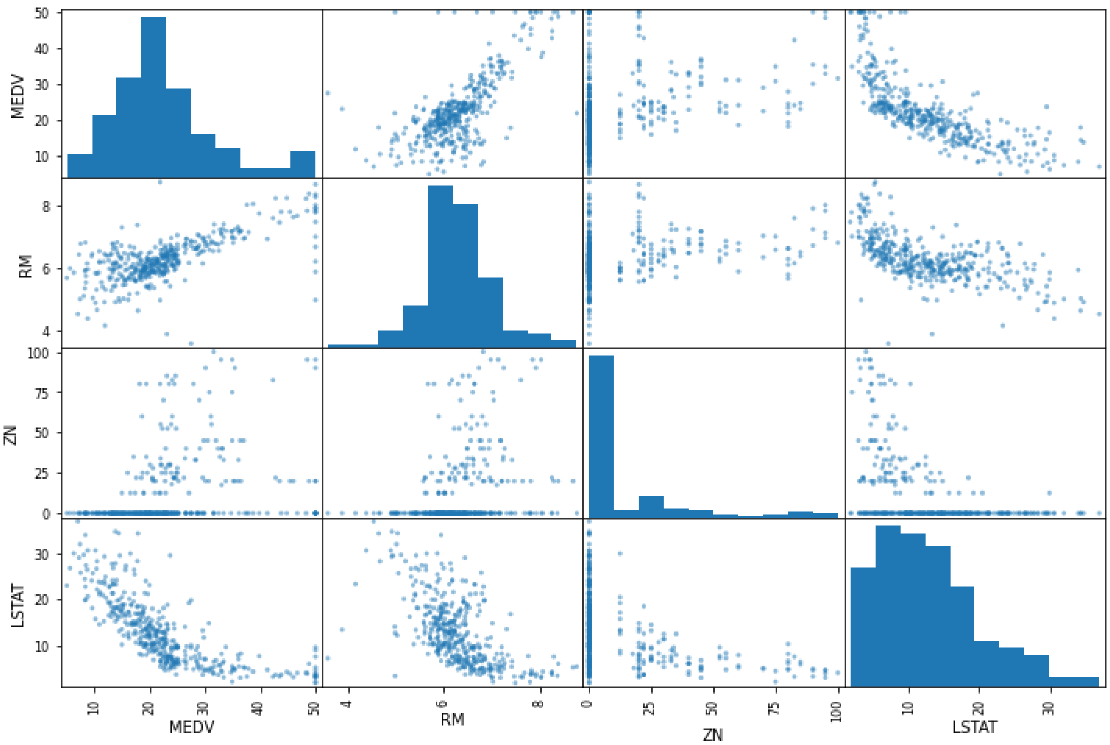
[18]:
```
corr_matrix['MEDV'].sort_values(ascending=False)
```

[18]:
```
MEDV       1.000000
RM         0.680898
B          0.361761
ZN         0.339741
DIS        0.252921
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
```

[19]:
```
from pandas.plotting import scatter_matrix
attributes = ["MEDV", "RM", "ZN", "LSTAT"]
scatter_matrix(housing[attributes], figsize=(12,8))
```

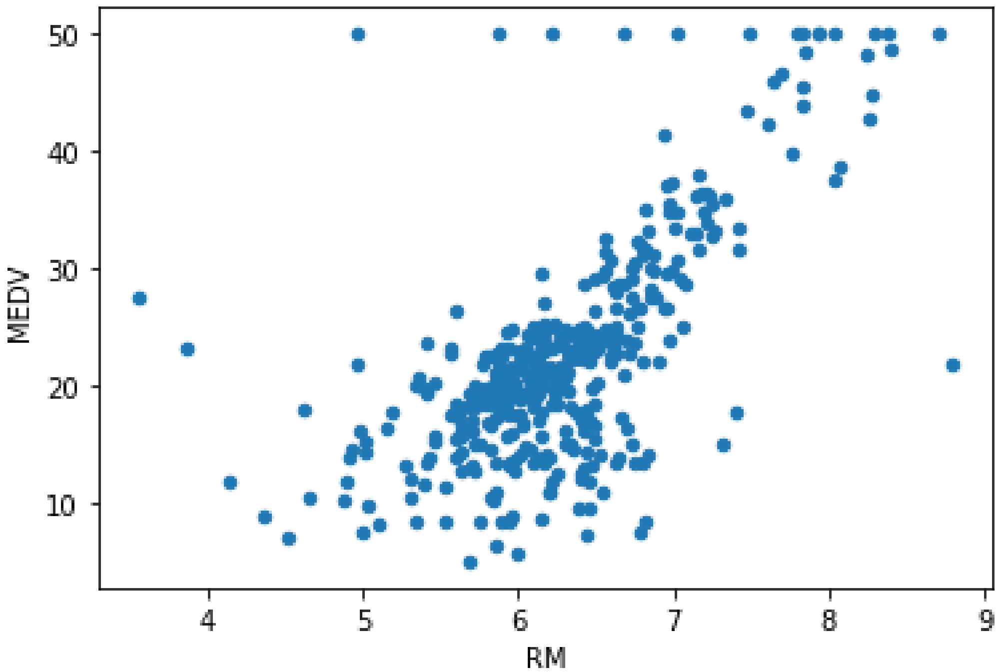
[19]:
```
array([[<AxesSubplot:xlabel='MEDV', ylabel='MEDV'>,
        <AxesSubplot:xlabel='RM', ylabel='MEDV'>,
        <AxesSubplot:xlabel='ZN', ylabel='MEDV'>,
```

```
 <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)
```



[20]:
```
housing.plot(kind="scatter", x="RM", y="MEDV", alpha=1)
```

[20]:
```
<AxesSubplot:xlabel='RM', ylabel='MEDV'>
```



0.3 Attribute Combinations

[21]:
```
housing["TPM"] = housing["TAX"]/housing["RM"]
```

[22]:
```
housing.head()
```

[22]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	

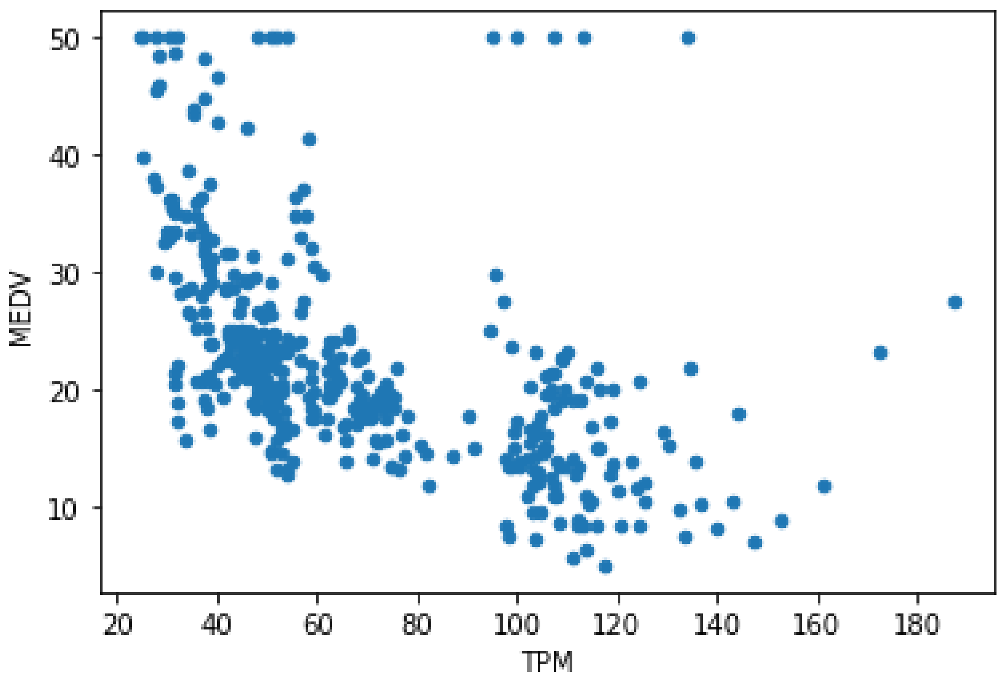
	PTRATIO	B	LSTAT	MEDV	TPM
254	16.4	392.89	6.57	21.9	51.571709
348	17.0	390.94	5.99	24.5	42.200452
476	20.2	396.21	18.68	16.7	102.714374
321	19.6	396.90	6.87	23.1	45.012547
326	19.6	396.90	6.15	23.0	45.468948

[23]:
```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

[23]:
```
MEDV        1.000000
RM          0.680898
B           0.361761
ZN          0.339741
DIS         0.252921
CHAS        0.205066
AGE        -0.364596
RAD        -0.374693
CRIM       -0.393715
NOX        -0.422873
TAX        -0.456657
INDUS      -0.473516
PTRATIO    -0.493534
TPM        -0.529820
LSTAT      -0.740494
Name: MEDV, dtype: float64
```

[24]:
```
housing.plot(kind="scatter", x="TPM", y="MEDV", alpha=1)
```

[24]: <AxesSubplot:xlabel='TPM', ylabel='MEDV'>



[25]:
```
housing = strat_train_set.drop("MEDV", axis=1)
housing_labels = strat_train_set["MEDV"].copy()
```

0.4 Missing Attributes

[26]:
```
#Set the value to some value (0, mean, median)
median = housing["RM"].median()
housing["RM"].fillna(median)
```

[26]:
```
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
```

[27]:
```
housing.describe() #Before Imputer
```

[27]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  404.000000  404.000000  404.000000  404.000000  404.000000  401.000000
mean     3.602814   10.836634   11.344950    0.069307    0.558064    6.281242
std      8.099383   22.150636    6.877817    0.254290    0.116875    0.715175
min      0.006320    0.000000    0.740000    0.000000    0.389000    3.561000
25%      0.086962    0.000000    5.190000    0.000000    0.453000    5.879000
50%      0.286735    0.000000    9.900000    0.000000    0.538000    6.211000
75%      3.731923   12.500000   18.100000    0.000000    0.631000    6.631000
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    69.039851    3.647200    9.735149  412.341584   18.473267  353.392822
std     28.258248    1.985503    8.731259  168.672623    2.129243   96.069235
min      2.900000    0.585700    1.000000  187.000000   13.000000    0.320000
25%     44.850000    2.005925    4.000000  284.000000   17.400000  374.617500
50%     78.200000    3.095750    5.000000  337.000000   19.000000  390.955000
75%     94.100000    4.824850   24.000000  666.000000   20.200000  395.630000
max    100.000000    9.222900   24.000000  711.000000   22.000000  396.900000

            LSTAT
count  404.000000
mean    12.791609
std      7.235740
min      1.730000
25%      6.847500
```

```
50%      11.570000
75%      17.102500
max      36.980000
```

[28]:
```python
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = "median")
imputer.fit(housing)
```

[28]:
```
SimpleImputer(strategy='median')
```

[29]:
```python
imputer.statistics_
```

[29]:
```
array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
       6.21100e+00, 7.82000e+01, 3.09575e+00, 5.00000e+00, 3.37000e+02,
       1.90000e+01, 3.90955e+02, 1.15700e+01])
```

[30]:
```python
imputer.statistics_.shape
```

[30]:
```
(13,)
```

[31]:
```python
X = imputer.transform(housing)
```

[32]:
```python
housing_tr = pd.DataFrame(X, columns=housing.columns)
```

[33]:
```python
housing_tr.describe()
```

[33]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  404.000000  404.000000  404.000000  404.000000  404.000000  404.000000
mean     3.602814   10.836634   11.344950    0.069307    0.558064    6.280720
std      8.099383   22.150636    6.877817    0.254290    0.116875    0.712533
min      0.006320    0.000000    0.740000    0.000000    0.389000    3.561000
25%      0.086962    0.000000    5.190000    0.000000    0.453000    5.879750
50%      0.286735    0.000000    9.900000    0.000000    0.538000    6.211000
75%      3.731923   12.500000   18.100000    0.000000    0.631000    6.630250
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    69.039851    3.647200    9.735149  412.341584   18.473267  353.392822
std     28.258248    1.985503    8.731259  168.672623    2.129243   96.069235
min      2.900000    0.585700    1.000000  187.000000   13.000000    0.320000
25%     44.850000    2.005925    4.000000  284.000000   17.400000  374.617500
50%     78.200000    3.095750    5.000000  337.000000   19.000000  390.955000
75%     94.100000    4.824850   24.000000  666.000000   20.200000  395.630000
max    100.000000    9.222900   24.000000  711.000000   22.000000  396.900000

            LSTAT
```

```
count    404.000000
mean      12.791609
std        7.235740
min        1.730000
25%        6.847500
50%       11.570000
75%       17.102500
max       36.980000
```

0.5 Scikit-learn Design

Three Types of objects: 1. Estimators – Estimates some parameters based on the dataset Fit method() – Fits the dataset and calculates the parameters 2. Transformers – Transform method takes input and returns output based on the learning from fit. It also has a convenience funtion called fit_transform(). 3. Predictors – Linear regression – Two common funtions are fit and predict

0.6 Feature Scaling

Two types of scaling: 1. Min-max scaling (Normalization) (value-min)/(max-min) – MinMaxScaler class by scikit-learn 2. Standardization (value-mean)/standard deviation – StandardScaler Class

0.7 Pipeline

[34]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scalar', StandardScaler()),
])
```

[35]:
```
housing_num_tr = my_pipeline.fit_transform(housing)
```

[36]:
```
housing_num_tr
```

[36]:
```
array([[-0.43942006,  3.12628155, -1.12165014, …, -0.97491834,
         0.41164221, -0.86091034],
       [-0.44352175,  3.12628155, -1.35893781, …, -0.69277865,
         0.39131918, -0.94116739],
       [ 0.15682292, -0.4898311 ,  0.98336806, …,  0.81196637,
         0.44624347,  0.81480158],
       …,
       [-0.43525657, -0.4898311 , -1.23083158, …, -0.22254583,
         0.41831233, -1.27603303],
       [ 0.14210728, -0.4898311 ,  0.98336806, …,  0.81196637,
        -3.15239177,  0.73869575],
       [-0.43974024, -0.4898311 ,  0.37049623, …, -0.97491834,
         0.41070422,  0.09940681]])
```

0.8 Desired Model For Real Estate

[37]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
#model = LinearRegression()
#model = DecisionTreeRegressor()
model.fit(housing_num_tr, housing_labels)
```

[37]:
```
RandomForestRegressor()
```

[38]:
```
some_data = housing.iloc[:5]
```

[39]:
```
some_labels = housing_labels.iloc[:5]
```

[40]:
```
prepare_data = my_pipeline.transform(some_data)
```

[41]:
```
model.predict(prepare_data)
```

[41]:
```
array([22.583, 25.132, 16.456, 23.316, 23.63 ])
```

[42]:
```
list(some_labels)
```

[42]:
```
[21.9, 24.5, 16.7, 23.1, 23.0]
```

0.9 Evauating the Model

[43]:
```
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)
```

[44]:
```
rmse
```

[44]:
```
1.3781245524908905
```

Better Evaluation Technique – Cross Validation

[45]:
```
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)
```

[46]:
```
rmse_scores
```

[46]: array([2.73790182, 2.6746119 , 4.59749578, 2.69350463, 3.71691875,
       2.78667253, 5.50491667, 3.44077699, 4.23587071, 4.07443288])

[47]:
```
def print_scores(scores):
    print("Scores: ", scores)
    print("Mean: ", scores.mean())
    print("Standard Deviation: ", scores.std())
```

[48]:
```
print_scores(rmse_scores)
```

```
Scores:  [2.73790182 2.6746119  4.59749578 2.69350463 3.71691875 2.78667253
 5.50491667 3.44077699 4.23587071 4.07443288]
Mean:  3.646310266391668
Standard Deviation:  0.9139743359391987
```

0.10 Saving the Model

[49]:
```
from joblib import dump,load
dump(model, 'Real_estate.joblib')
```

[49]: ['Real_estate.joblib']

0.11 Testing

[50]:
```
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepare = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepare)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
#print(final_predictions, list(Y_test))
```

[51]:
```
final_rmse
```

[51]: 3.0561793798729777

[52]:
```
prepare_data[0]
```

[52]: array([-0.43942006, 3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.24270365, -1.31238772, 2.81037668, -1.0016859 , -0.5778192 ,
       -0.97491834, 0.41164221, -0.86091034])

0.12 Using The Model

[53]:
```
from joblib import dump,load
import numpy as np
model = load('Real_estate.joblib')
features = np.array([[-0.43942006,  10.12628155, 5.12165014, -0.27288841, -2.
↪42262747,
       -2.54270365, -1.31238772,  2.81037668, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.41164221, -0.86091034]])
model.predict(features)
```

[53]:
```
array([22.414])
```