In [725]:

```
import pandas as pd
```

In [726]:

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

In [727]:

```
housing.head
```

Out[727]:

```
<bound method NDFrame.head of         CRIM     ZN  INDUS  CHAS    NOX     RM   AGE     DIS
RAD  TAX  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900     1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671     2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671     2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622     3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622     3  222
..       ...   ...    ...   ...    ...    ...   ...     ...   ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786     1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875     1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675     1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889     1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050     1  273

     PTRATIO      B   LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       ...     ...    ...   ...
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]>
```

In [728]:

```
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
```

In [729]:

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

Out[729]:

```
0    471
1     35
Name: CHAS, dtype: int64
```

In [730]:

```
housing.describe()
```

Out[730]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000	506.000000	506.000000	506.0000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.286557	68.574901	3.795043	9.549407	408.2371
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.704436	28.148861	2.105710	8.707259	168.5371
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.0000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.888000	45.025000	2.100175	4.000000	279.0000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.209000	77.500000	3.207450	5.000000	330.0000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.625000	94.075000	5.188425	24.000000	666.0000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0000

In [731]:

```
%matplotlib inline
```

In [732]:

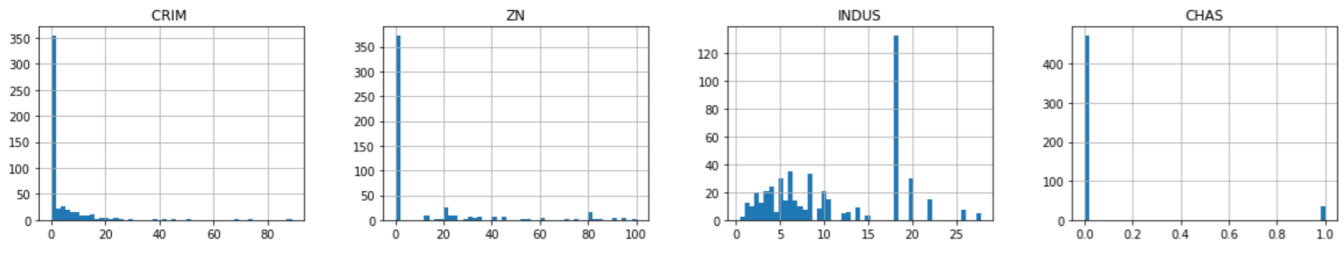
```
import matplotlib.pyplot as plt
```

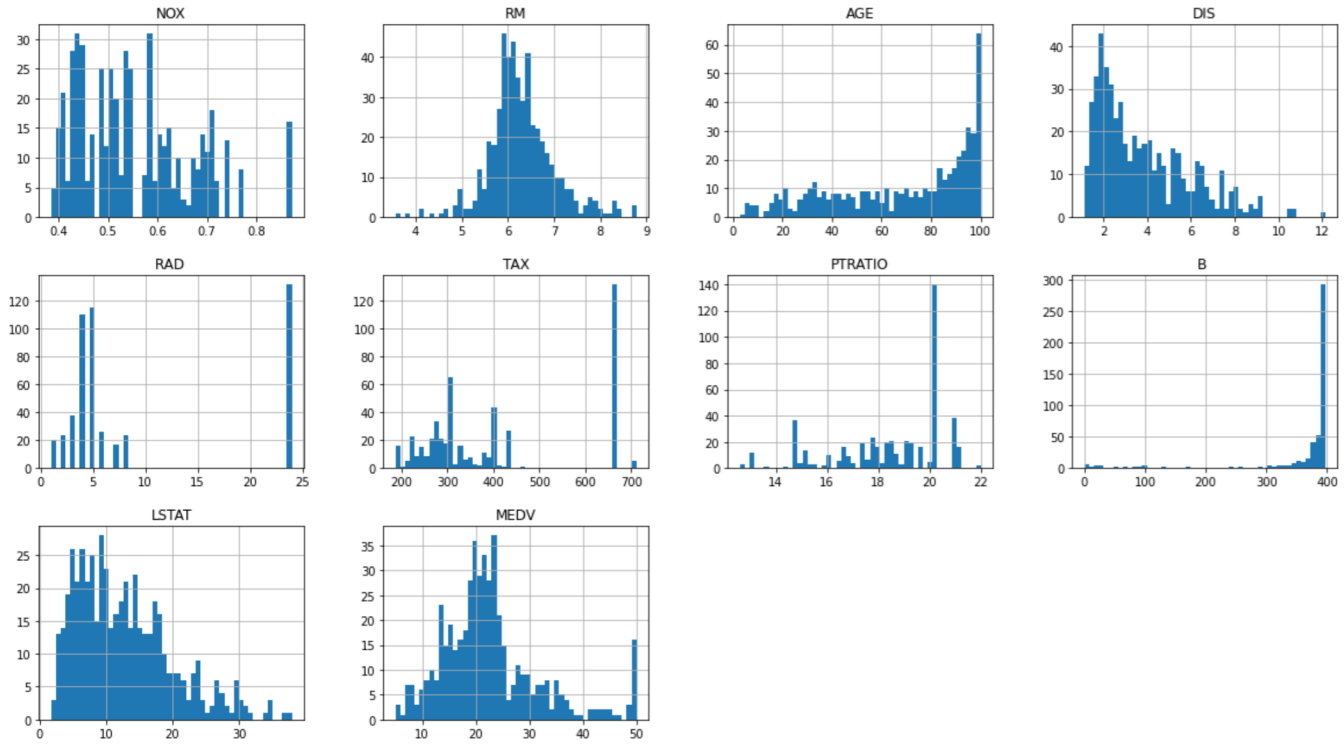
In [733]:

```
#plotting histogram
housing.hist(bins=50, figsize=(20,15))
```

Out[733]:

```
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)
```





Train-Test Splitting

In [734]:

```
# For learning purpose
import numpy as np
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]
```

In [735]:

```
# train_set, test_set = split_train_test(housing,0.2)
```

In [736]:

```
# print(f"Rows in train set: {len(train_set)}\n Rows in test set: {len(test_set)}\n")
```

In [737]:

```
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)}\n Rows in test set: {len(test_set)}\n")
```

```
Rows in train set: 404
 Rows in test set: 102
```

In [738]:

```
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 [739]:

```
strat_test_set['CHAS'].value_counts()
```

Out[739]:

```
0    95
1     7
Name: CHAS, dtype: int64
```

In [740]:

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

Out[740]:

```
0    376
1     28
Name: CHAS, dtype: int64
```

In [741]:

```
# 95/7
```

In [742]:

```
# 376/28
```

In [743]:

```
housing = strat_train_set.copy()
```

Looking for Correlations

In [744]:

```
corr_matrix = housing.corr()
```

In [745]:

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

Out[745]:

```
MEDV       1.000000
RM         0.680393
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
```

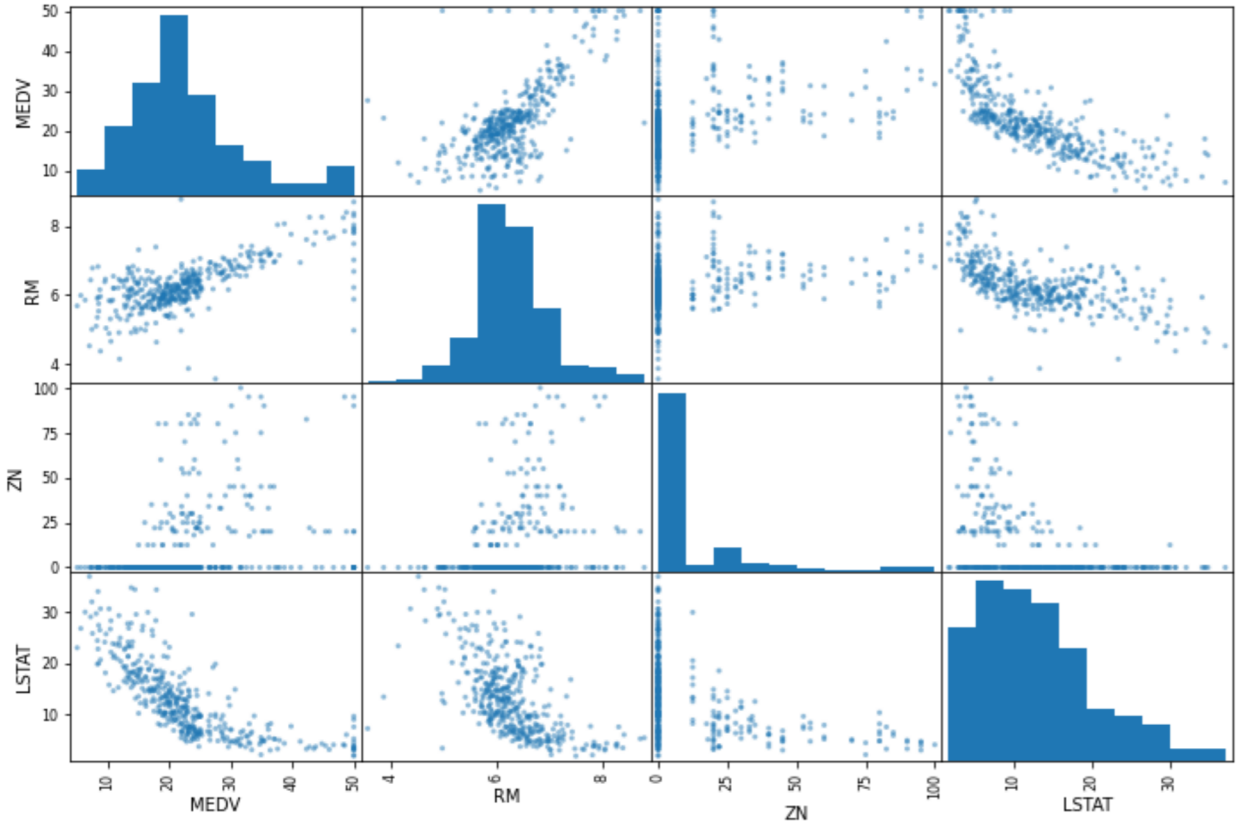
In [746]:

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

Out[746]:

```
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)
```

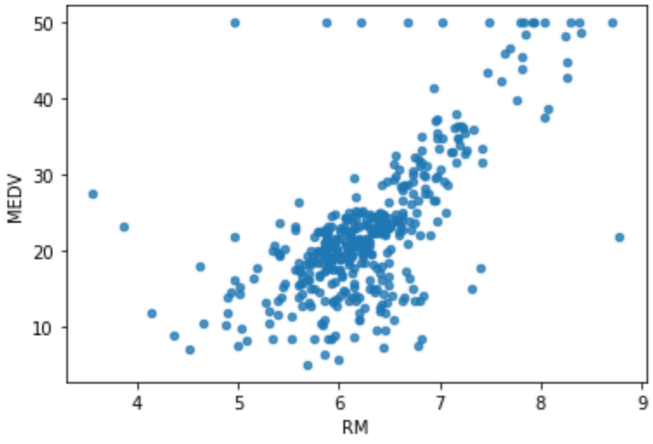


In [747]:

```
#remove outliers
housing.plot(kind="scatter",x = "RM",y="MEDV",alpha = 0.8)
```

Out[747]:

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



Trying out attribute combinations

In [748]:

```
housing["TAXRM"] = housing['TAX']/housing['RM']
```

In [749]:

```
housing["TAXRM"]
```

Out[749]:

```
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 [750]:

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

Out[750]:

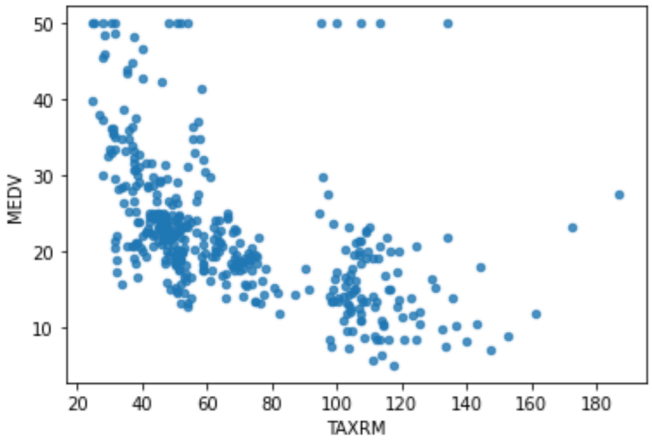
```
MEDV       1.000000
RM         0.680393
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
PTRATIO   -0.493534
TAXRM     -0.525076
LSTAT     -0.740494
Name: MEDV, dtype: float64
```

In [751]:

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

Out[751]:

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



In [752]:

```
housing = strat_train_set.drop("MEDV",axis = 1)
```

```
housing_labels =  strat_train_set["MEDV"].copy()
```

Missing attributes

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(0,mean or median)**

In [753]:

```
housing.dropna(subset=["RM"]).shape #option 1
# Note that the original housing dataframe will remain unchanged
```

Out[753]:

```
(402, 13)
```

In [754]:

```
housing.drop("RM",axis=1).shape #option 2
# Note there is no RM coloumn and also not that the original housing will remain unchange
d
```

Out[754]:

```
(404, 12)
```

In [755]:

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

In [756]:

```
median
```

Out[756]:

```
6.209
```

In [757]:

```
housing["RM"].fillna(median) # option 3
# Note that the original housing dataframe will remain unchanged
```

Out[757]:

```
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 [758]:

```
housing.shape
```

Out[758]:

```
(404, 13)
```

In [759]:

```
housing.describe() # before we started imputer
```

Out[759]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	T/
count	404.000000	404.000000	404.000000	404.000000	404.000000	402.000000	404.000000	404.000000	404.000000	404.0000
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.278256	69.039851	3.746210	9.735149	412.3415
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.714285	28.258248	2.099057	8.731259	168.6726
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.0000
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.878250	44.850000	2.035975	4.000000	284.0000
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.209000	78.200000	3.122200	5.000000	337.0000
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630000	94.100000	5.100400	24.000000	666.0000
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0000

In [760]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = "median")
imputer.fit(housing)
```

Out[760]:

```
SimpleImputer(strategy='median')
```

In [761]:

```
imputer.statistics_
```

Out[761]:

```
array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
       6.20900e+00, 7.82000e+01, 3.12220e+00, 5.00000e+00, 3.37000e+02,
       1.90000e+01, 3.90955e+02, 1.15700e+01])
```

In [762]:

```
x = imputer.transform(housing)
```

In [763]:

```
housing_tr = pd.DataFrame(x, columns=housing.columns)
```

In [764]:

```
housing_tr.describe()
```

Out[764]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	T/
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.0000
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.277913	69.039851	3.746210	9.735149	412.3415
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.712527	28.258248	2.099057	8.731259	168.6726
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.0000
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.878750	44.850000	2.035975	4.000000	284.0000
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.209000	78.200000	3.122200	5.000000	337.0000
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630000	94.100000	5.100400	24.000000	666.0000
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0000

Scikit-learn 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. Fit method - Fits the datasets and calculate internal parameters.**
2. **Transformers - transform method takes input and return output based on the learnings from fit(). It also has a convenience function called fit_transform() which fits and then transforms.**
3. **Predictors - LinearRegression model is an example of predictor. fit() and predict() are two common functions. It also gives score() function which will evaluate the predictions.**

Feature Scaling

Primarily, two types of feature scaling methods:

1. **Min-max scaling (Normalization) (value - min)/(max - min) Sklearn provides a class called MinMaxScaler for this**
2. **Standardization (value - mean)/std sklearn provide a class called Standard Scaler for this**

Creating a Pipeline

In [765]:

```
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()),
])
```

In [766]:

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

In [767]:

```
housing_num_tr.shape
```

Out[767]:

```
(404, 13)
```

Selecting a desired model for Dragon Real Estates

In [768]:

```
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[768]:

```
RandomForestRegressor()
```

In [769]:

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

In [770]:

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

In [771]:

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

In [772]:

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

Out[772]:

```
array([22.539, 25.574, 16.374, 23.337, 23.643])
```

In [773]:

```
list(some_labels)
```

Out[773]:

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

Evaluating the model

In [774]:

```
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 [775]:

```
rmse
```

Out[775]:

```
1.223576085954123
```

Using better evaluation technique - Cross Validation

In [776]:

```
# 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_square
d_error",cv=10)
rmse_scores = np.sqrt(-scores)
```

In [777]:

```
rmse_scores
```

Out[777]:

```
array([2.83172784, 2.73578287, 4.25124286, 2.5206622 , 3.4308568 ,
       2.61920586, 4.29618675, 3.32136184, 2.88694743, 3.24739945])
```

In [778]:

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

In [779]:

```
print_scores(rmse_scores)
```

```
Scores: [2.83172784 2.73578287 4.25124286 2.5206622  3.4308568  2.61920586
 4.29618675 3.32136184 2.88694743 3.24739945]
Mean: 3.2141373880353696
Standard deviation:  0.6019098004737453
```

Saving the model

In [780]:

```
from joblib import dump,load
dump(model,"Dragon.joblib")
```

Out[780]:

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

Testing the model on test data

In [783]:

```
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))
```

```
[24.881 11.743 25.651 22.366 18.739 14.72  20.487 15.087 31.417 41.324
 19.574 11.938 23.642 28.75  19.519 11.477 31.842 14.396 23.58  18.443
 19.9   17.183 15.317 21.575 19.051 32.149 15.852 33.171  8.716 33.025
 23.722 21.307 23.264 11.318 21.208 11.526 42.699 24.968 23.737 41.966
 24.232 29.723 20.144 20.648 18.729 33.486 44.638 19.998 20.512 21.752
 21.357 15.289 21.904 15.168 25.231 33.334 41.998 29.008 20.073 20.509
 46.572  9.605 19.05  25.033 14.865 33.113 19.94  18.27  19.189 34.733
 25.819 23.096 21.205 22.188 34.314 12.726 15.835 22.558 20.657 21.372
 22.79  21.1   14.196 23.577 20.995 21.148 13.677 21.253 21.573 23.446
 18.427 27.32   7.703 26.123 18.58  29.457 20.152 31.075 14.439 26.763
 21.996 20.423] [16.5, 10.2, 30.1, 23.0, 14.4, 15.6, 19.4, 14.1, 30.3, 35.2, 23.1, 13.8,
25.0, 27.9, 19.5, 12.3, 32.2, 13.5, 23.8, 21.7, 19.2, 19.5, 10.4, 23.2, 18.6, 28.5, 15.2,
32.0, 7.2, 34.6, 20.1, 20.6, 23.6, 13.1, 23.8, 12.7, 43.1, 24.7, 22.2, 44.0, 28.1, 31.0,
21.7, 23.4, 19.5, 33.1, 41.7, 18.7, 19.9, 20.6, 21.2, 13.6, 20.3, 17.8, 27.1, 31.5, 50.0,
29.1, 18.9, 20.4, 50.0, 7.2, 17.2, 36.2, 14.6, 33.2, 23.8, 19.9, 21.5, 37.3, 27.0, 22.0,
24.3, 19.8, 33.3, 7.0, 19.4, 20.9, 21.1, 20.4, 22.2, 11.9, 11.7, 21.6, 19.7, 23.0, 16.7,
21.7, 20.6, 23.3, 19.6, 28.0, 5.0, 24.4, 20.8, 24.8, 21.8, 23.6, 19.0, 25.0, 20.3, 21.5]
```

In [782]:

```
final_rmse
```

Out[782]:

```
2.9237002776833134
```

In [784]:

```
prepared_data[0]
```

Out[784]:

```
array([-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.23876152, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.41164221, -0.86091034])
```

In [ ]: