Observing the given Dataset

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

In [2]:
```
housing = pd.read_csv("housing_dataset.csv")
```

In [3]:
```
housing.head() #it gives the first 5 rows of dataset
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	I
0	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
1	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
2	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
3	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	
4	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	

In [4]:
```
housing.tail() #it gives the last 5 rows of dataset
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
500	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
501	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
502	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
503	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
504	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

In [5]:
```
housing.info() #it gives description of dataset
```

```
<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       505 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
```

In [6]:
```
housing['CHAS'].value_counts() #it gives the value counts
```

Out[6]:
```
0    470
1     35
Name: CHAS, dtype: int64
```

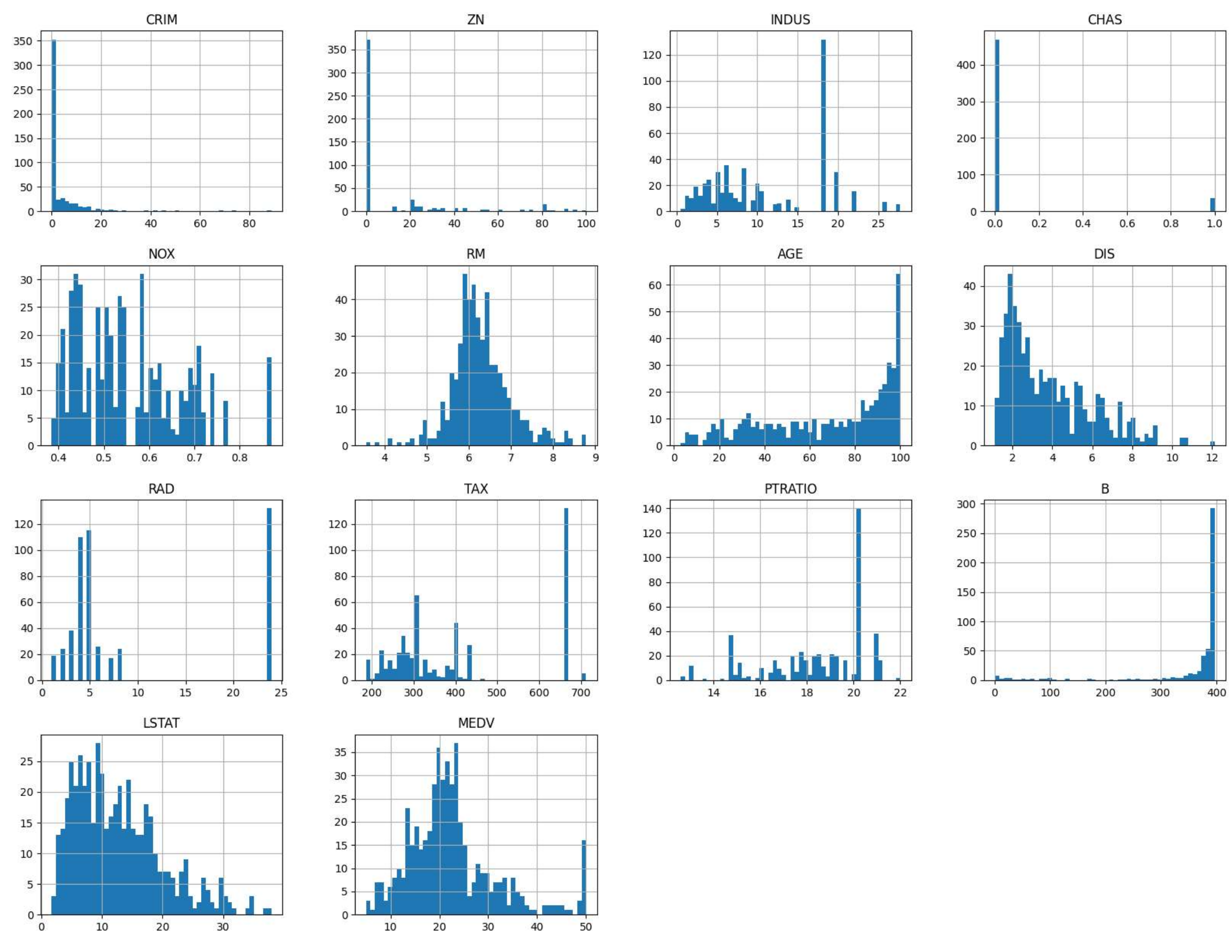
In [7]:
```
housing.describe()
#it gives count, mean, standard deviation, minimum, percentiles and maximum
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	505.00
mean	3.620667	11.350495	11.154257	0.069307	0.554728	6.284059	68.581584	3.79
std	8.608572	23.343704	6.855868	0.254227	0.115990	0.703195	28.176371	2.10
min	0.009060	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082210	0.000000	5.190000	0.000000	0.449000	5.885000	45.000000	2.10
50%	0.259150	0.000000	9.690000	0.000000	0.538000	6.208000	77.700000	3.19
75%	3.678220	12.500000	18.100000	0.000000	0.624000	6.625000	94.100000	5.21
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

In [8]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20, 15))
#for plotting histograms of all data columns
```

Out[8]:
```
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 [9]:
```
#for learning purpose only

import numpy as np

def split_train_test(data, test_ratio):

    np.random.seed(42)
    #this command is used so that it fixes the data
    #which comes under random function.
    #Otherwise the model would have seen all the data
    #points and it would have been resulted in
    #"overfitting".

    shuffled = np.random.permutation(len(data))

    test_set_size = int(len(data)*test_ratio)
    #taking out some percentage of data as test data

    train_indices = shuffled[test_set_size:]
    #slicing out training data

    test_indices = shuffled[:test_set_size]
    #slicing out testing data

    return data.iloc[train_indices], data.iloc[test_indices]
    #returning sliced out training and testing data

train_set, test_set = split_train_test(housing, 0.2)
#invoking the split_train_test method

print("Rows in training data set = ", len(train_set))
print("Rows in testing data set = ", len(test_set))
```

```
Rows in training data set =  404
Rows in testing data set =  101
```

In [10]:
```
from sklearn.model_selection import train_test_split as tts
#in built module for above written code snippet

train_set, test_set = tts(housing, test_size=0.2, random_state=42)
#invoking train_test_split which is named as "tts"

print("Rows in training data set = ", len(train_set))
print("Rows in testing data set = ", len(test_set))
```

```
Rows in training data set =  404
Rows in testing data set =  101
```

In [11]:
```
#actually if an attribute having boolean data, comes in our dataset
#then our problem is the equal
#distribution of both boolean values in training and testing
#dataset which is achieved by
#StratifiedShuffleSplit function

from sklearn.model_selection import StratifiedShuffleSplit as sss

split = sss(n_splits=1, test_size=0.2, random_state=42)

for strat_train_index,strat_test_index in split.split(housing, housing['CHAS']):
    strat_train_set = housing.loc[strat_train_index]
    strat_test_set = housing.loc[strat_test_index]

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

```
0    376
1     28
Name: CHAS, dtype: int64
0    94
1     7
Name: CHAS, dtype: int64
```

In [12]:
```
housing = strat_train_set.copy()
# now our strat_train_set has become the usable training dataset.
```

Looking for Correlations

Correlations suggest that on increasing the value of a particular column how the values of other columns change. here we have taken correlations with respect to 'MEDV', the label. the greater the value more is effect of change of the taken argument on the respective column. postive value suggests that, the particular column increases on increasing the taken argument. correlation values range from -1 to 1.

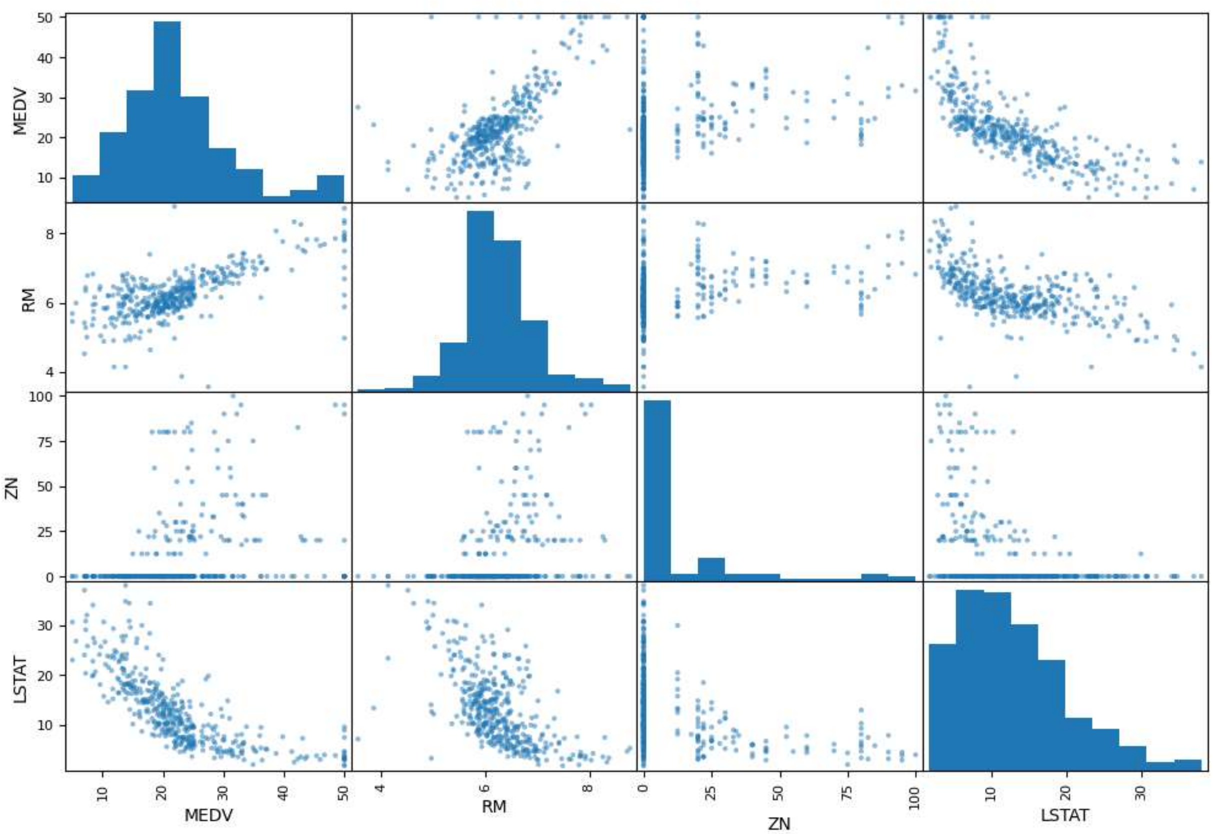
In [13]:
```
corr_matrix = housing.corr() # .corr() is a pandas function

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

Out[13]:
```
MEDV       1.000000
RM         0.660761
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
```

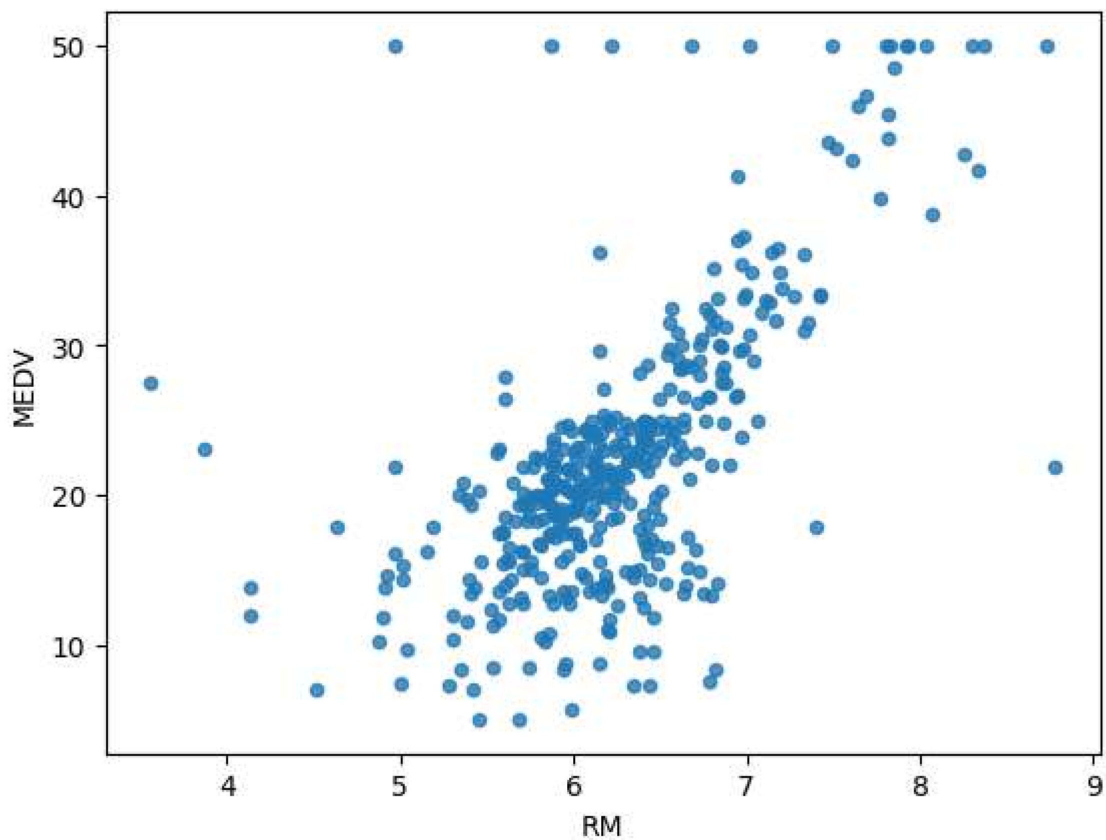
In [14]:
```
from pandas.plotting import scatter_matrix
attributes = ["MEDV", "RM", "ZN", "LSTAT"]
scatter_matrix(housing[attributes], figsize = (12,8))
```

Out[14]:
```
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 [15]:
```
housing.plot(kind="scatter", x="RM", y="MEDV", alpha=.8)
#here in this graph we are observing some points which are very far from
#the main cluster of points
#so these type of points will give more error in our model hence,
#better to remove these points
#we have drawn RM vs MEDV graph coz RM is the most important
#factor in determining the cost of house.
```

Out[15]: <AxesSubplot:xlabel='RM', ylabel='MEDV'>



Trying out attribute(column) combinations

Attribute combination is just, producing new attributes with the help of existing attributes by implementing mathematical operations on them.

In [16]:
```
#housing["TAXperRM"] = housing["TAX"]/housing["RM"]
#added a column TAXperRM(tax per room)
```

In [17]:
```
#housing.head()
```

In [18]:
```
#housing.plot(kind="scatter", x="TAXperRM", y="MEDV", alpha=.8)
#this graph is more proper insight of data
```

Techniques to avoid the presence of Missing attributes (if any)

Currently here in our housing dataset there is no missing attribute so the below mentioned code is for learning purpose

To take care of missing attributes we have three options:

1.get rid of the missing data points by removing the whole tuple. 2.get rid of whole attribute. 3.set the missing datapoints with some values like 0, median or mean.

In [19]:
```
# 1.
a = housing.dropna(subset = "RM")
# this function .dropna(), removes the tuple having the given attribute as null.
a.shape # this provides the order of dataset.

# note that the original dataset 'housing' remains unchanged



# 2.
b = housing.drop("RM", axis=1)
# this function .drop(), drops the whole 'RM' attribute from the dataset
b.shape # this provides the order of dataset.

# note that the original dataset 'housing' remains unchanged



# 3.
median = housing["RM"].median()
# this function calculates the median of particular attribute of a dataset.
c = housing["RM"].fillna(median)
# this function fills the null cells of particular attribute with the data given.
c.shape

# note that the original dataset 'housing' remains unchanged
# also note that the median must also be updated in the test set (if any).
```

Out[19]:
```
(404,)
```

let's automate this task

In [20]:
```
from sklearn.impute import SimpleImputer as SI
imputer = SI(strategy="median")
# creating a SimpleImputer array witth startegy as median.
imputer.fit(housing)
# taking out the medians of all attributes of housing dataset.
imputer.statistics_
# knowing the length of array of medians of all attributes.
X = imputer.transform(housing)
# giving the null cells of dataset, the respective
#value of median of that particular column.
# note here that the returned datatype of .transform() function
#is a numpy array so care must be
#taken to convert it into a pandas array.
housing_imputed = pd.DataFrame(X, columns = housing.columns)
housing_imputed.describe()
housing_imputed.head()
```

Out[20]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.03548	80.0	3.64	0.0	0.392	5.876	19.1	9.2203	1.0	315.0	16.4	395.18	9.2
1	0.02899	40.0	1.25	0.0	0.429	6.939	34.5	8.7921	1.0	335.0	19.7	389.85	5.8
2	15.02340	0.0	18.10	0.0	0.614	5.304	97.3	2.1007	24.0	666.0	20.2	349.48	24.9
3	0.35114	0.0	7.38	0.0	0.493	6.041	49.9	4.7211	5.0	287.0	19.6	396.90	7.7
4	0.24103	0.0	7.38	0.0	0.493	6.083	43.7	5.4159	5.0	287.0	19.6	396.90	12.7

SciKit learn- design

Primarily, there are three types of objects:

1. Estimators- It estimates some parameter based on the dataset. eg. IMPUTER. It has fit() function and transform() function. The fit() function calculates the internal parameters by fitting dataset
2. Transformers- It takes input from the learnings of the fit() function and fills the desired cells with the input. It also has a fit_transform() function wjich is better optimised for fitting and transforming tasks simultaneously.
3. Predictors- LinearRegressors, Classifiers. They have fit(), predict() and score() functions. The score() function evaluates the predictors.

Feature Scaling

Feature scaling refers to making all the attributes lie in the same range of numbers. For eg. one attribute is no_of_rooms which has values in the range 1-10, second an attribute crime_rate having values in the range 1-100. So to bring down the crime_rate values between 1-10 ranged equivalent values, we use feature scaling.

Primarily, there are two types of feature scalings of which standardization is better:

1. Min-Max scaling(normalization): (cell_value - min)/(max -min) For the above task SKlearn provides us a class "MinMaxScaler"
2. Standardization: (cell_value - mean)/standard_deviation For the above task sklearn provides us a class "StandardScaler"

Creating a PipeLine

PipeLine is a joint effort to do all the tasks like Imputer, FeatureScaling. So, what we have done before hand in our data, is all to be done in the pipeline itself. The code line marked as * takes a list of tasks to be performed on your dataframe, which is finally given to the code in line mentioned as **. This line returnes a numpy array which is to be used further in prediction tasks.

In [21]:
```
from sklearn.pipeline import Pipeline as PL
from sklearn.preprocessing import StandardScaler as SS
my_pipeline = PL([('imputer', SI(strategy="median")),             # *
                  # ...................add as many as you want
                  #('std_scaler', SS())
                  ])

housing_piped_imputed = my_pipeline.fit_transform(housing_imputed) # **
housing_piped_imputed.shape
# this should be the dimensions of your stratified_training_dataset
```

Out[21]:
```
(404, 14)
```

Separting Features and Labels.

Now, we have with us a housing_piped_imputed and a strat_test_set. But they have both features and labels entact. So, we will have to separate the features and labels.

*note that housing_piped_imputed is a numpy array. So, to separate features and labels, first make it as a pandas dataframe.

In [22]:
```
new = pd.DataFrame(housing_piped_imputed, columns = housing.columns)

train_features = new.drop("MEDV", axis=1)
# shape is (404,13). It has all except "MEDV".
train_labels = new["MEDV"].copy()
# shape is (404,1). It has "MEDV".

test_features = strat_test_set.drop("MEDV", axis=1)
# shape is (101,13). It has all except "MEDV".
test_labels = strat_test_set["MEDV"].copy()
# shape is (101,1). It has "MEDV".
```

Trying out different ML models for training

Now the steps to be followed are:

1. Training - training refers to the fitting of training data as features and labels in our created model. Here we use the model.fit() command.
2. Evaluating the model - Here we evaluate our model by predicting some of the data points taken from the training data itself. We use the techniques of MSE, RMSE etc. Thus, on the basis of the value of MSE or RMSE we decide wether our model is good or bad.

- note here do not use the testing data. As it would result in overfitting in the testing step.

3. Testing - The final step of our project is this. when our code requires no more editing and bug fixing then at last we take the test data and observe the predictions of our ML model.

In [23]:
```
from sklearn.linear_model import LinearRegression as LR
from sklearn.tree import DecisionTreeRegressor as DTR
from sklearn.ensemble import RandomForestRegressor as RFR
#model = LR()
#model = DTR()
model = RFR()
model.fit(train_features, train_labels) #model training.
```

Out[23]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

Evaluating Model

Here, we will take some of the entries of traning dataset and evaluate the model by manually onserving difference between predicted and actual labels. Also, we can calculate our RMSE.

In [24]:
```
some_features = train_features.iloc[:5]
actual_some_labels = train_labels.iloc[:5]
piped_some_features = my_pipeline.fit_transform(some_features)
predicted_some_labels = model.predict(piped_some_features)
print("actual labels: ", list(actual_some_labels)) # these are the actual labels
print("predicted labels: ", predicted_some_labels) # these are predicted values

# Evaluation of 5 entries in the training dataset.
import numpy as np
from sklearn.metrics import mean_squared_error
some_mse = mean_squared_error(actual_some_labels, predicted_some_labels)
some_rmse = np.sqrt(some_mse)
print("some mean squared error: ", some_mse)
print("some root mean squared error: ", some_rmse)

# Evaluationof all entries in training dataset.
predicted_train_labels = model.predict(train_features)
whole_mse = mean_squared_error(train_labels, predicted_train_labels)
whole_rmse = np.sqrt(whole_mse)
print("whole mean squared error: ", whole_mse)
print("whole root mean squared error: ", whole_rmse)
```

```
actual labels:  [20.9, 26.6, 12.0, 20.4, 22.2]
predicted labels:  [20.57  27.608 12.29  20.932 22.176]
some mean squared error:  0.2985327999999883
some root mean squared error:  0.5463815516651237
whole mean squared error:  1.3686446757425752
whole root mean squared error:  1.1698908819811253

C:\Users\prath\Desktop\Coding\Machine Learning\learning_ml\lib\site-packages\sk
learn\base.py:450: UserWarning: X does not have valid feature names, but Random
ForestRegressor was fitted with feature names
  warnings.warn(
```

Now in the above code we have observed that our model has learnt the noise of training data and not the trend. It has overfitted the training data. So, we will use a better evaluation technique - CROSS VALIDATION.

In [25]:
```
from sklearn.model_selection import cross_val_score
scores=cross_val_score(model,housing_piped_imputed,train_labels,scoring="neg_mear
rmse_scores = np.sqrt(-scores)
rmse_scores
```

Out[25]:
```
array([0.19291676, 0.35064133, 0.40037208, 0.11023169, 0.12931918,
       0.1481241 , 0.16827136, 0.32042039, 0.22581403, 0.19404329])
```

In [26]:
```
rmse_scores.mean()
```

Out[26]:
```
0.22401542113697293
```

In [27]:
```
rmse_scores.std()
```

Out[27]:
```
0.09438188899822719
```

Choosing the best ML model.

LinearRegression:

```
whole mean squared error:  22.283078267672217

whole root mean squared error:  4.7204955531884805

scores: array([2.30200374e-14, 3.13325354e-14, 1.79944030e-14, 1.4803045
9e-14,
                   1.57385617e-14, 1.53811373e-14, 2.54087340e-14, 3.974
85892e-14,
                       2.96563825e-14, 3.49972165e-14])

mean: 0.4529685254952523

standard deviation: 0.19672006304223297
```

Decision tree regression:

```
 whole mean squared error:  0.0 (over fitting)

 whole root mean squared error:  0.0 (over fitting)

 scores: array([0.53601966, 0.68875993, 0.74309833, 0.14055639, 0.280624
3 ,
                       0.13509256, 0.54977268, 0.43646306, 0.19937402,
 0.2403123 ])

 mean: 0.39500732202348576

 standard deviation: 0.21461609726758857
```

Random Forest Regressor:

```
 whole mean squared error:  1.3345383564356426

 whole root mean squared error:  1.1552222108476111

 scores: array([0.254613  , 0.37137142, 0.46742731, 0.11132254, 0.124603
57,
                   0.12256457, 0.17177864, 0.3624241 , 0.21361747, 0.251
40212])

 mean: 0.24511247416041795

 standard deviation: 0.11535465413441479

 actual labels:  [20.9, 26.6, 12.0, 20.4, 22.2]

 predicted labels:  [20.482 27.375 12.439 20.969 22.154]
```

So, our Random Forest Regressor is the best ML model. System Faad diya bande ne.... predictions dekho bhai ki...... gadar kat diya.

Saving the model using Joblib

Joblib is a very important tool of SciKitLearn. It is used to inherit a particular function from one notebook to other. It has two functions, load(), dump().

dump() - It saves the given function with the given name as a .joblib file. load() - It loads the given function with the given name from the saved .joblib file.

In [28]:

```
from joblib import dump,load

dump(model, "RANDOM_FOREST_REGRESSOR.joblib")
dump(my_pipeline, "my_pipeline.joblib")
dump(train_features, "tf.joblib")
```

Out[28]:

```
['tf.joblib']
```

Testing the model on Test Dataset

We have with us, test_features and test_labels. Now the process is to pass the test_features through the pipeline and use the model.predict() function to predict predicted_test_labels. Then we should evaluate the model by mse, rmse, scores, mean, std etc.

In [29]:
```
piped_test_features = my_pipeline.fit_transform(test_features)

predicted_test_labels = model.predict(piped_test_features)

testing_mse = mean_squared_error(test_labels, predicted_test_labels)

testing_rmse = np.sqrt(testing_mse)
```

```
C:\Users\prath\Desktop\Coding\Machine Learning\learning_ml\lib\site-packages\sk
learn\base.py:450: UserWarning: X does not have valid feature names, but Random
ForestRegressor was fitted with feature names
  warnings.warn(
```

In [30]:
```
testing_mse
```

Out[30]:
```
11.573636504950487
```

In [31]:
```
testing_rmse
```

Out[31]:
```
3.4020047773262294
```

In [32]:
```
print(list(test_labels))
```

```
[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, 1
6.1, 15.2, 10.5, 21.4, 23.2, 20.7, 21.7, 13.0, 22.3, 19.6, 21.2, 18.1, 50.0, 2
3.7, 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]
```

In [33]:
```
predicted_test_labels
```

Out[33]:
```
array([22.809, 22.382, 46.511, 32.727, 45.369, 34.634, 20.991, 23.462,
       32.855, 19.774, 19.453, 30.949, 21.832, 33.439, 20.51 , 21.548,
       12.385, 21.241, 28.22 , 19.569, 19.929, 45.359, 11.864, 19.153,
       26.105, 34.339, 16.486, 15.708,  6.531, 20.493, 23.543, 23.106,
       18.379, 15.256, 20.723, 18.901, 22.964, 17.403, 45.272, 17.427,
       21.352, 18.699, 19.489, 18.376, 33.167,  8.279, 24.915, 14.451,
       21.146, 21.339, 45.866, 23.831, 15.006, 21.478, 19.702, 46.907,
       33.415, 19.811, 34.957, 10.595, 23.705, 35.441, 33.253, 23.821,
       14.248, 20.886, 20.964, 15.693, 28.151, 24.322, 23.414, 32.196,
       19.341, 31.899, 10.897, 20.097, 42.607, 19.618, 19.812, 13.993,
       41.619,  9.048, 35.662, 22.9  , 28.754, 15.866, 23.224, 21.954,
       20.501, 16.101, 26.238,  9.887, 32.008, 12.701, 25.918, 20.456,
       33.375, 13.723, 21.146, 21.067, 20.837])
```

Hurray!!! our model has performed very well on a mere set of 404 training data entries.......