Kaul Real Esate - Price Predictor

In [1]:

```
import pandas as pd
```

In [2]:

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

In [3]:

```
housing.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [4]:

```
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: 55.4 KB
```

In [5]:

```
housing['CRIM']
```

Out[5]:

```
0      0.00632
1      0.02731
2      0.02729
3      0.03237
4      0.06905
        ...
501    0.06263
502    0.04527
503    0.06076
```

```
504     0.10959
505     0.04741
Name: CRIM, Length: 506, dtype: float64
```

In [6]:

```python
housing['CRIM']
```

Out[6]:

```
0       0.00632
1       0.02731
2       0.02729
3       0.03237
4       0.06905
          ...
501     0.06263
502     0.04527
503     0.06076
504     0.10959
505     0.04741
Name: CRIM, Length: 506, dtype: float64
```

In [7]:

```python
housing.describe()
```

Out[7]:

	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	6.288563	68.574901	3.795043	9.549407	408.2371
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.704689	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	6.211000	77.500000	3.207450	5.000000	330.0000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.629000	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 [8]:

```python
housing['CRIM'].value_counts()
```

Out[8]:

```
0.01501     2
14.33370    2
0.57834     1
0.06127     1
0.03548     1
           ..
0.25356     1
0.10469     1
0.22876     1
0.34109     1
0.26363     1
Name: CRIM, Length: 504, dtype: int64
```

In [9]:

```python
%matplotlib inline
```

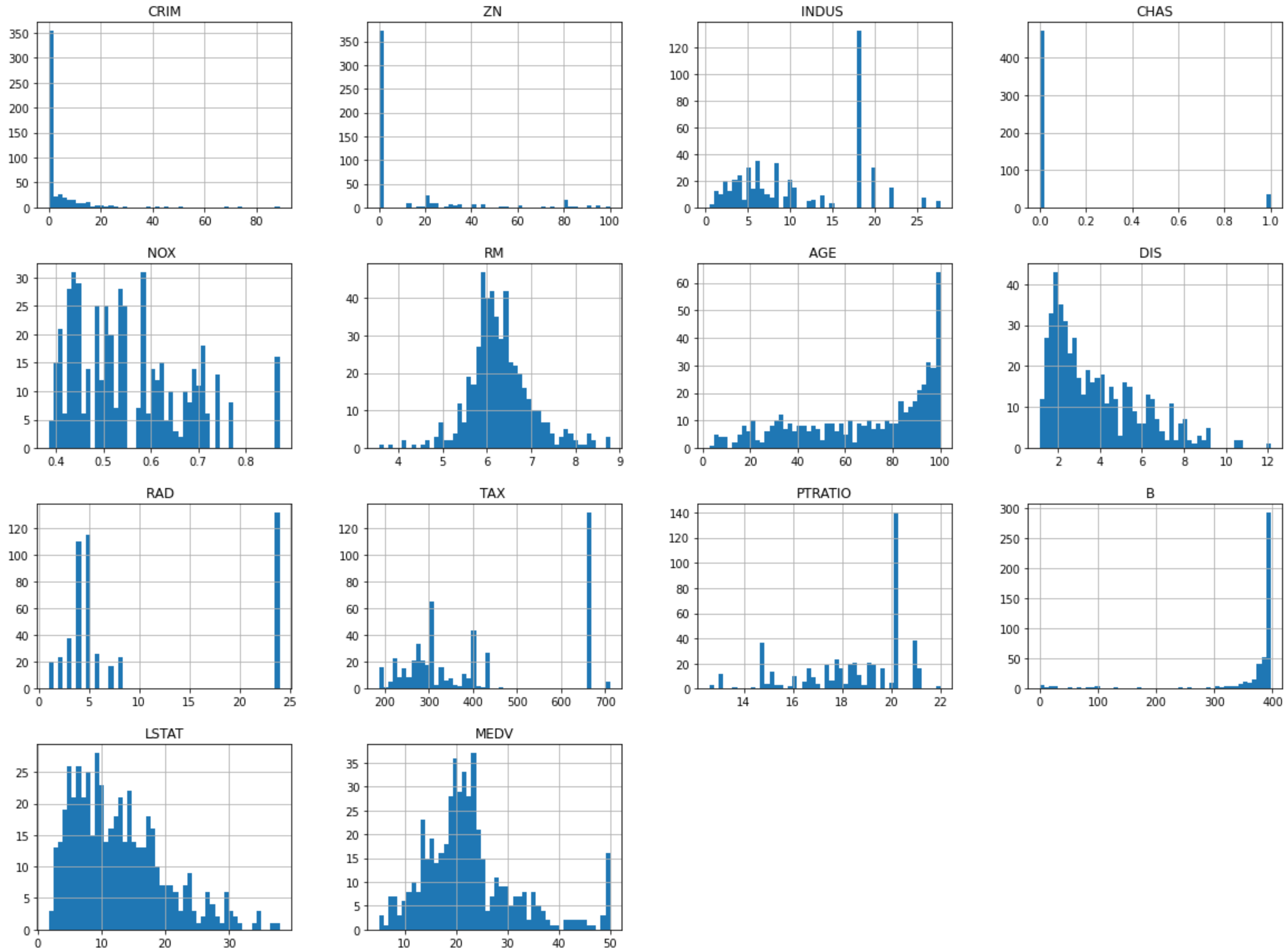
In [10]:

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

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

Out[10]:

```
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 [11]:

```python
# for learning purpose
import numpy as np
def split_train_test(data , test_ratio):
    np.random.seed(42)
    shuffled = np.random.permutation(len(data))
    print(shuffled)
    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 [12]:

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

```
[173 274 491  72 452  76 316 140 471 500 218   9 414  78 323 473 124 388
 195 448 271 278  30 501 421 474  79 454 210 497 172 320 375 362 467 153
   2 336 208  73 496 307 204  68  90 390  33  70 470   0  11 281  22 101
 268 485 442 290  84 245  63  55 229  18 351 209 395  82  39 456  46 481
 444 355  77 398 104 203 381 489  69 408 255 392 312 234 460 324  93 137
 176 417 131 346 365 132 371 412 436 411  86  75 477  15 332 423  19 325
 335  56 437 409 334 181 227 434 180  25 493 238 244 250 418 117  42 322
 347 182 155 280 126 329  31 113 148 432 338  57 194  24  17 298  66 211
 404  94 154 441  23 225 433 447   5 116  45  16 468 360   3 405 185  60
 110 321 265  29 262 478  26   7 492 108  37 157 472 118 114 175 192 272
 144 373 383 356 277 220 450 141 369  67 361 168 499 394 400 193 249 109
 420 145  92 152 222 304  83 248 165 163 199 231  74 311 455 253 119 284
 302 483 357 403 228 261 237 386 476  36 196 139 368 247 287 378  59 111
  89 266   6 364 503 341 158 150 177 397 184 318  10 384 103  81  38 317
 167 475 299 296 198 377 146 396 147 428 289 123 490  96 143 239 275  97
 353 122 183 202 246 484 301 354 410 399 286 125 305 223 422 219 129 424
 291 331 380 480 358 297 294 370 438 112 179 310 342 333 487 457 233 314
 164 136 197 258 232 115 120 352 224 406 340 127 285 415 107 374 449 133
 367  44 495  65 283  85 242 186 425 159  12  35  28 170 142 402 349 221
  95  51 240 376 382 178  41 440 391 206 282 254 416   4 256 453 100 226
 431 213 426 171  98 292 215  61  47  32 267 327 200 451  27 393 230 260
 288 162 429 138  62 135 128 482   8 326 469  64 300  14 156  40 379 465
 407 216 279 439 504 337 236 207 212 295 462 251 494 464 303 350 269 201
 161  43 217 401 190 309 259 105  53 389   1 446 488  49 419  80 205  34
 430 263 427 366  91 339 479  52 345 264 241  13 315  88 387 273 166 328
 498 134 306 486 319 243  54 363  50 461 174 445 189 502 463 187 169  58
  48 344 235 252  21 313 459 160 276 443 191 385 293 413 343 257 308 149
 130 151 359  99 372  87 458 330 214 466 121 505  20 188  71 106 270 348
 435 102]
```

In [13]:

```
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
```

In [14]:

```
from sklearn.model_selection import train_test_split
train_set , test_set = train_test_split(housing , test_size = 0.2, random_state = 42)
```

In [15]:

```
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 [16]:

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

Out[16]:

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

In [17]:

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

Out[17]:

```
0    376
1     28
```

```
Name: CHAS, dtype: int64
```

In [18]:

```
#95/7
```

In [19]:

```
#376/28
```

In [20]:

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

Looking for corelations

In [21]:

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

Out[21]:

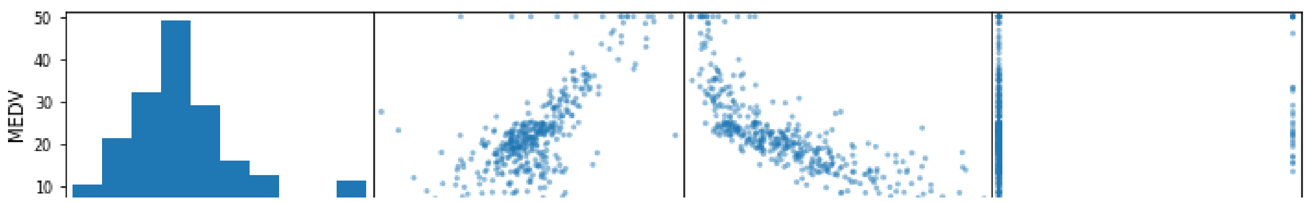
```
MEDV       1.000000
RM         0.679042
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
PTRATIO   -0.493534
LSTAT     -0.740494
Name: MEDV, dtype: float64
```

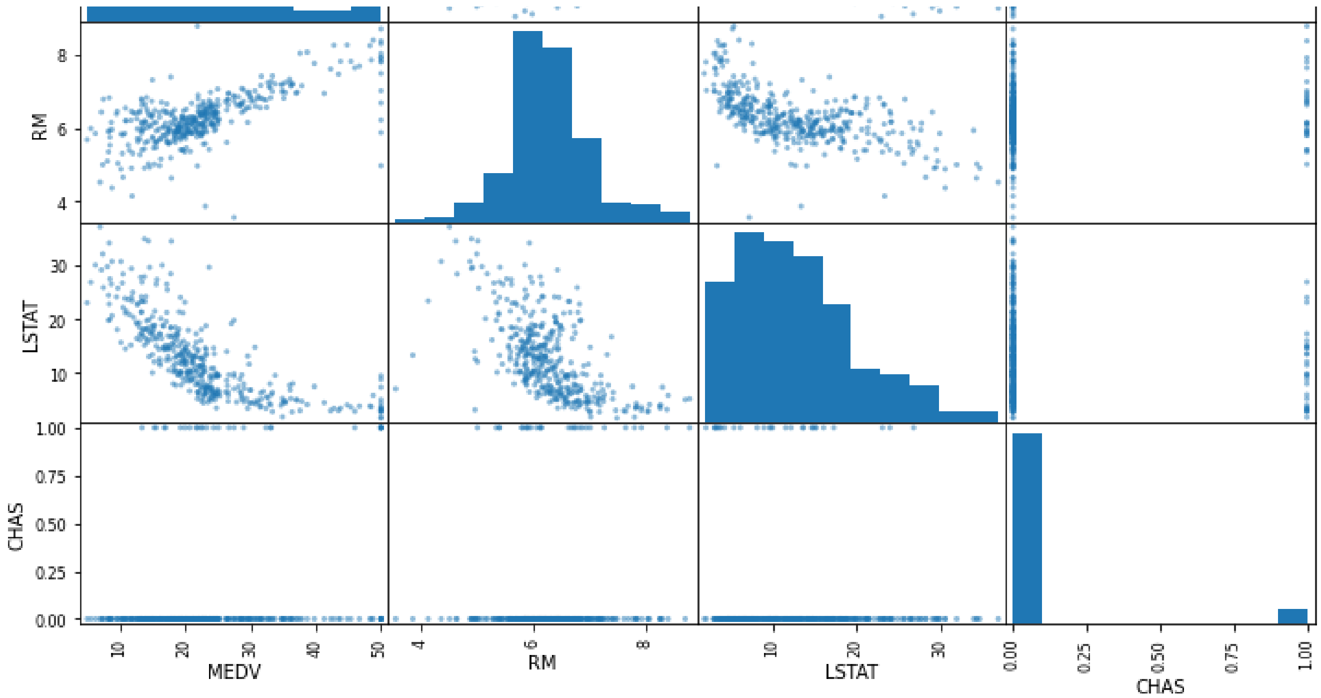
In [22]:

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

Out[22]:

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



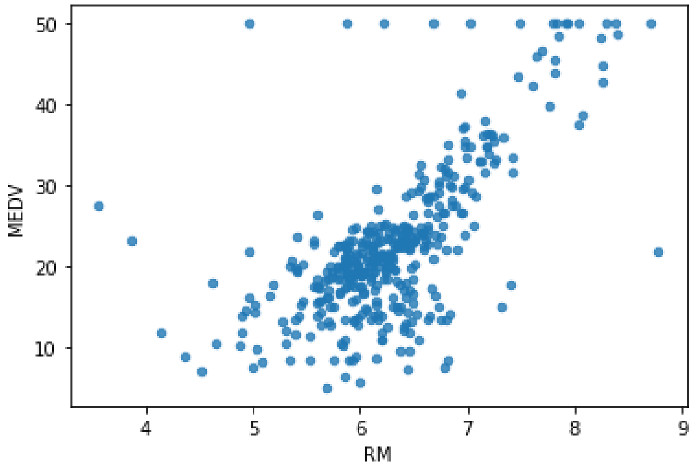


In [23]:

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

Out[23]:

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



Trying out attribute combination

In [24]:

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

In [25]:

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

Out[25]:

```
254     1.075639
348     0.902788
476     2.880938
321     1.077478
326     0.974335
          ...
155     2.441482
423     3.816156
98      0.456522
455     2.778544
216     2.294497
```

```
Name: TAXRM, Length: 404, dtype: float64
```

In [26]:

```
housing.head()
```

Out[26]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	TAXRM
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57	21.9	1.075639
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99	24.5	0.902788
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68	16.7	2.880938
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6.87	23.1	1.077478
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6.15	23.0	0.974335

In [27]:

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

Out[27]:

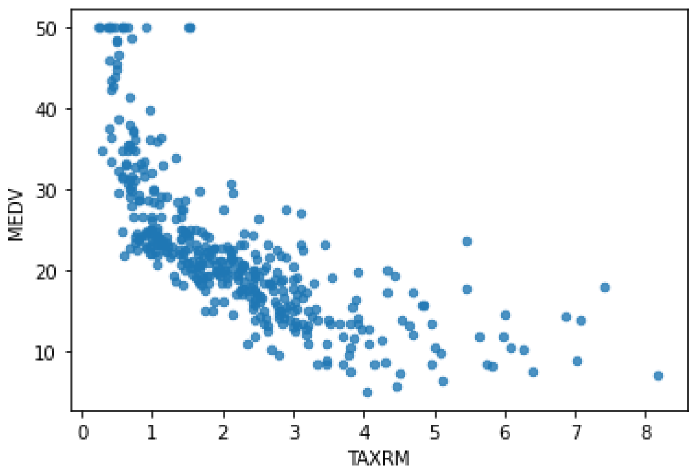
```
MEDV        1.000000
RM          0.679042
B           0.361761
ZN          0.339741
DIS         0.240451
CHAS        0.205066
AGE        -0.364596
RAD        -0.374693
CRIM       -0.393715
NOX        -0.422873
TAX        -0.456657
INDUS      -0.473516
PTRATIO    -0.493534
TAXRM      -0.714723
LSTAT      -0.740494
Name: MEDV, dtype: float64
```

In [28]:

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

Out[28]:

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



In [29]:

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

Missing attributes

In [30]:

```
#To take care of missing attributes , you have three options:
#     1. Get rid of the missing data points
#     2. Get rid of the whole attributes
#     3. set the value to some values(0, mean or median )
```

In [31]:

```
a=housing.dropna(subset=["RM"])  # option1
a.shape
# note that the original housing dataframe will remain unchanged
```

Out[31]:

```
(399, 13)
```

In [32]:

```
housing.drop("RM" , axis=1).shape # option2
# note that there is no RM column and also note that the original housing dataframe will
remain unchanged
```

Out[32]:

```
(404, 12)
```

In [33]:

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

In [34]:

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

Out[34]:

```
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 [35]:

```
housing.shape
```

Out[35]:

```
(404, 13)
```

In [36]:

```
housing.describe()  # before we started filling missing attributes
```

Out[36]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
count	404.000000	404.000000	404.000000	404.000000	404.000000	399.000000	404.000000	404.000000	404.000000	404.0000
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.284782	69.039851	3.746210	9.735149	412.3415

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.284782	69.039851	3.746210	9.735149	412.3415
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.715706	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	5.882000	44.850000	2.035975	4.000000	284.0000
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.219000	78.200000	3.122200	5.000000	337.0000
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.633000	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 [37]:

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

Out[37]:

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

In [38]:

```
imputer.statistics_
```

Out[38]:

```
array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
       6.21900e+00, 7.82000e+01, 3.12220e+00, 5.00000e+00, 3.37000e+02,
       1.90000e+01, 3.90955e+02, 1.15700e+01])
```

In [39]:

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

In [40]:

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

In [41]:

```
housing_tr.describe()
```

Out[41]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
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	6.283968	69.039851	3.746210	9.735149	412.3415
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.711289	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	5.884750	44.850000	2.035975	4.000000	284.0000
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.219000	78.200000	3.122200	5.000000	337.0000
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630250	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 [ ]:

scikit-learn Design

In [42]:

```
#Primarily /three types of objects
#1.Estimators - it estimates some parameter base on dataset . eg. imputer
#it has a fit method and transform method fit method - fits the dataset and calculates in
ternal parameters

#2.Transformers - trasform method takes input and returns output based on the learnings f
rom fit() it also has a convience
#function called fit_transform() which fits and then transforms .

#3.Predictor - linearRegression model is an example of predictor . fit() and predict() ar
e two common functions . it also gives
#score function which will evaluate the predictions.
```

Feature Scaling

In [43]:

```
# 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 provides a class called standard scaler for this
```

Creating a Pipeline

In [44]:

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

In [45]:

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

In [46]:

```
housing_num_tr.shape
```

Out[46]:

```
(404, 13)
```

Selecting a desired model for Kaul Real Estates

In [47]:

```
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[47]:

```
RandomForestRegressor()
```

In [48]:

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

In [49]:

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

In [50]:

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

In [51]:

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

Out[51]:

```
array([22.553, 25.569, 16.286, 23.437, 23.435])
```

In [52]:

```
list(some_labels)
```

Out[52]:

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

Evaluating the model

In [53]:

```
from sklearn.metrics import mean_squared_error
housing_prediction = model.predict(housing_num_tr)
mse = mean_squared_error(housing_labels, housing_prediction)
rmse = np.sqrt(mse)
```

In [54]:

```
rmse
```

Out[54]:

```
1.153499199594584
```

Using better evaluating technique - Cross Validation

In [55]:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model,housing_num_tr,housing_labels,scoring="neg_mean_squared_er
ror",cv = 10)

rmse_scores = np.sqrt(-scores)
```

In [56]:

```
rmse_scores
```

Out[56]:

```
array([2.75811139, 2.74687783, 4.38782954, 2.57153331, 3.33705963,
       2.5925675 , 4.79503724, 3.27699087, 3.34623819, 3.20906954])
```

In [57]:

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

In [58]:

```
print_scores(rmse_scores)
```

```
Scores: [2.75811139 2.74687783 4.38782954 2.57153331 3.33705963 2.5925675
 4.79503724 3.27699087 3.34623819 3.20906954]
Mean: 3.3021315034267005
Standard deviation: 0.7114411460695924
```

saving the model

In [61]:

```
from joblib import dump , load
dump(model, 'Kaul.joblib')
```

Out[61]:

```
['Kaul.joblib']
```

Testing the model on test data

In [72]:

```
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))
```

In [70]:

```
final_rmse
```

Out[70]:

```
2.916184264868504
```

In [74]:

```
prepared_data[0]
```

Out[74]:

```
array([-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.24769958, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.41164221, -0.86091034])
```

Using the model

In [75]:

```
from joblib import dump, load
import numpy as np
model = load('Kaul.joblib')
features = np.array([[-0.43942006,  10.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.14769958, -99.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.41164221, -0.86091034]])
model.predict(features)
```

Out[75]:

```
array([23.051])
```

In [ ]: