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

1 Loading the Data ¶

In [2]:
```
BOSTON_HOUSING_PATH="/home/bhaveshnarayan19905535/housing.data"
housing = pd.read_csv(BOSTON_HOUSING_PATH,delim_whitespace=True,header = No
housing.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	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	242.0	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	242.0	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	222.0	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	222.0	18.7	396.90	5.33	36.2

In [3]:
```
columns_names = ['CRIM', 'ZN' , 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
```

In [4]:
```
housing.columns = columns_names
```

In [5]:
```
housing.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L:
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	

2 Analyzing the Data

In [6]:
```
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       506 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    float64
 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(12), int64(2)
memory usage: 55.5 KB
```

In [7]:
```
housing.describe()
```

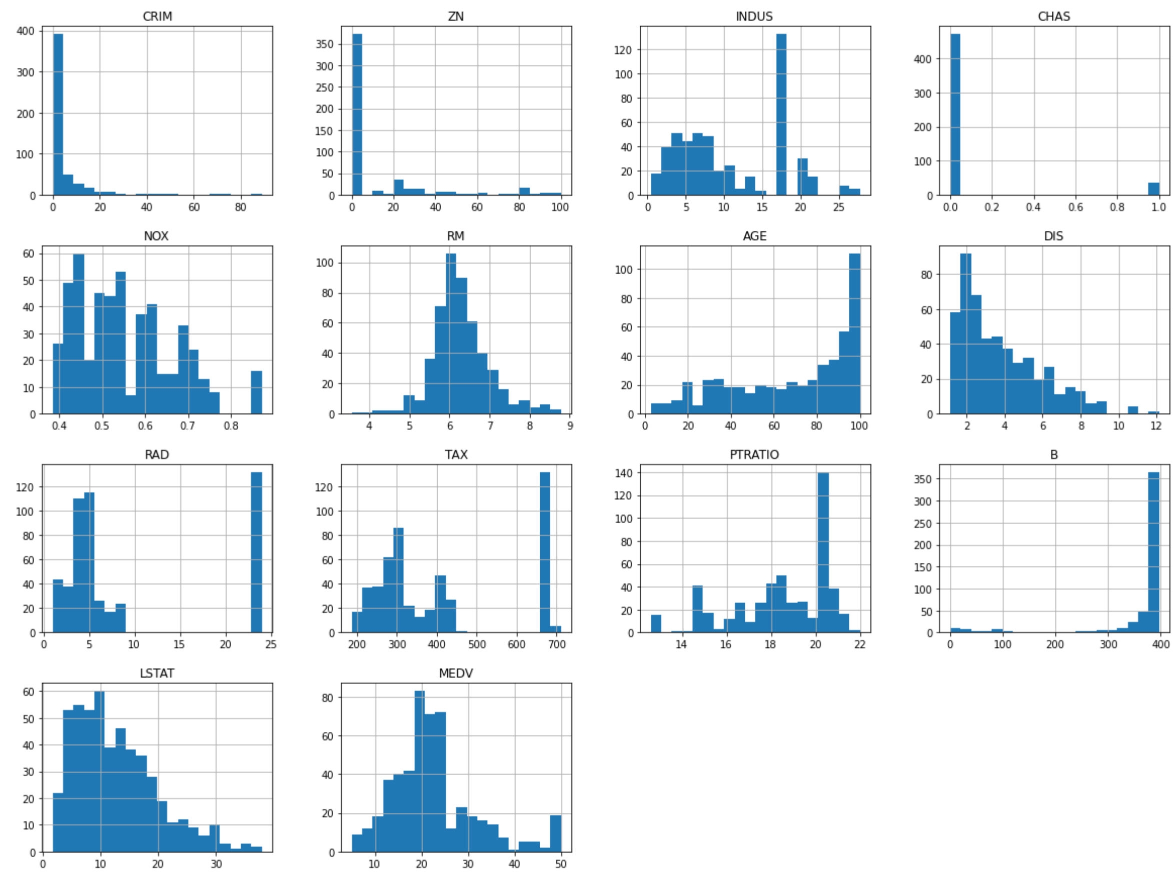
Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

3 PLotting the Data

In [8]:
```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

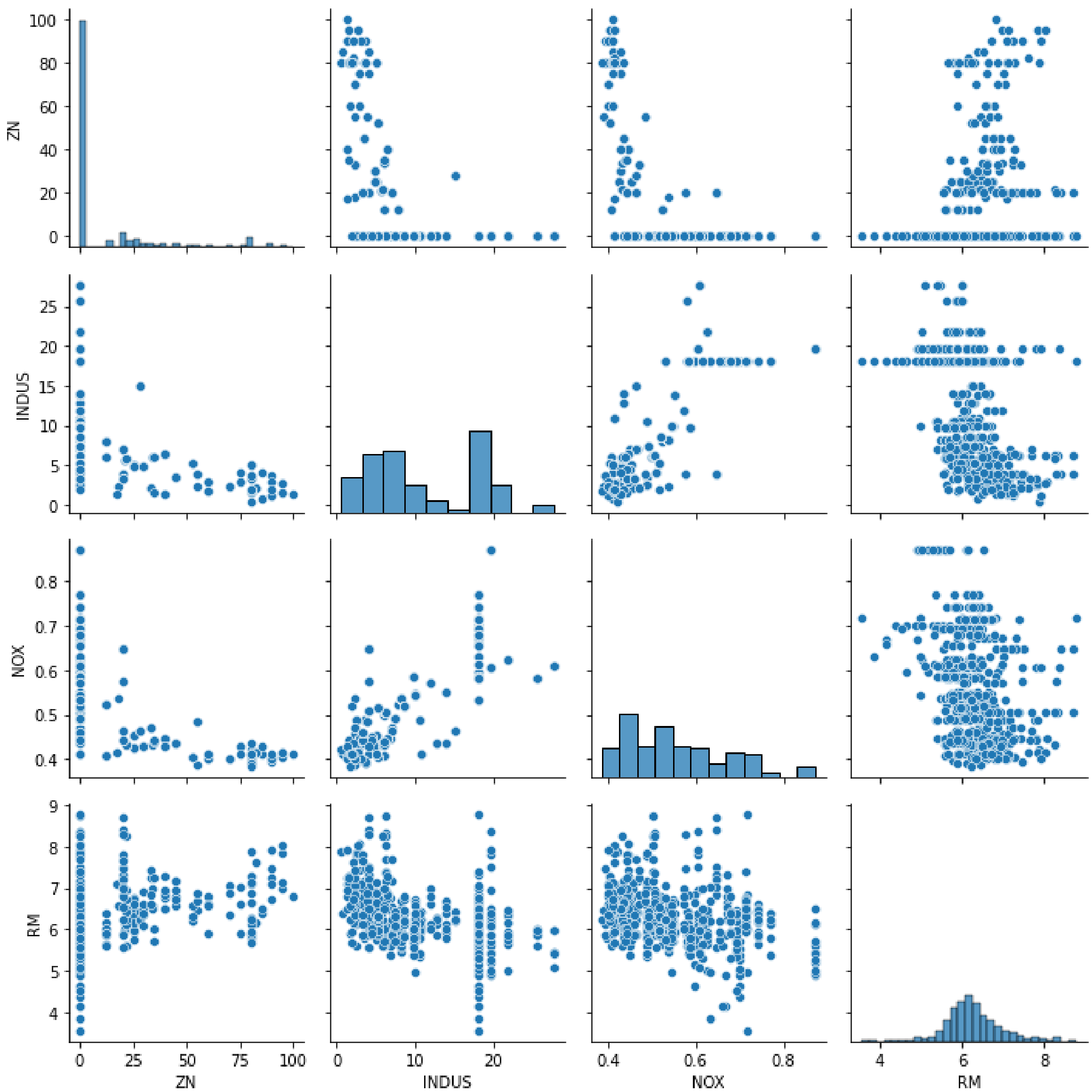
In [9]:
```
housing.hist(bins=20, figsize=(20,15))
plt.show()
```



In [10]:

```
import seaborn as sns
column_analysis = ['ZN', 'INDUS','NOX', 'RM']
sns.pairplot(housing[column_analysis],size = (2.5))
plt.tight_layout()
plt.show()
```

```
/usr/local/anaconda/lib/python3.6/site-packages/seaborn/axisgrid.py:2076: U
serWarning: The `size` parameter has been renamed to `height`; please updat
e your code.
  warnings.warn(msg, UserWarning)
```



4 Calculating the Outliers percentage

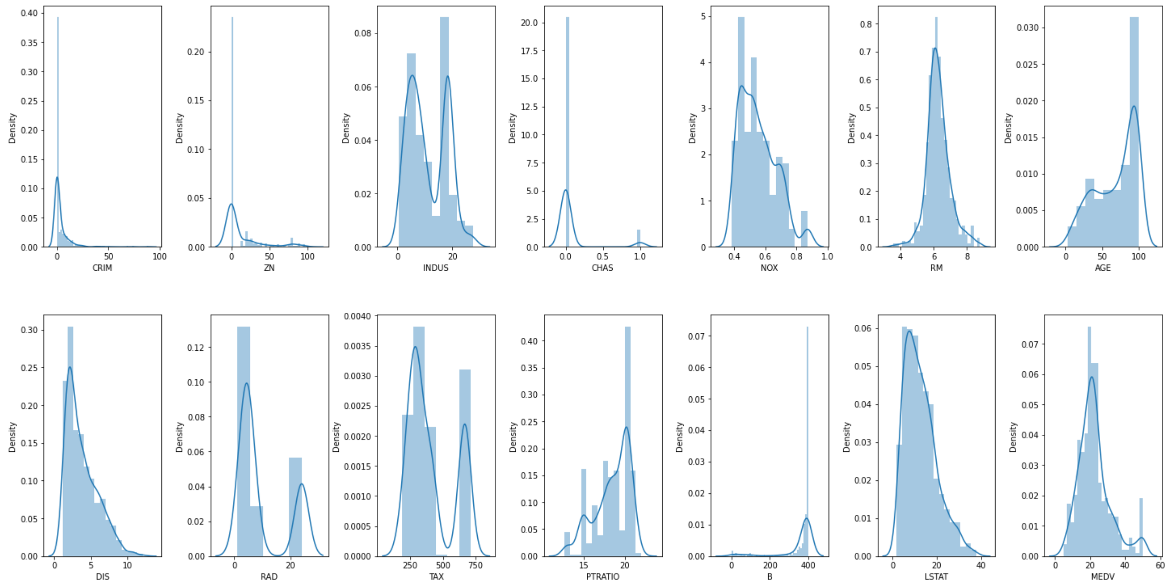
In [11]:

```
for k, v in housing.items():
    q1 = v.quantile(0.25)
    q3 = v.quantile(0.75)
    irq = q3 - q1
    v_col = v[(v <= q1 - 1.5 * irq) | (v >= q3 + 1.5 * irq)]
    perc = np.shape(v_col)[0] * 100.0 / np.shape(housing)[0]
    print("Column %s outliers = %.2f%%" % (k, perc))
```

```
Column CRIM outliers = 13.04%
Column ZN outliers = 13.44%
Column INDUS outliers = 0.00%
Column CHAS outliers = 100.00%
Column NOX outliers = 0.00%
Column RM outliers = 5.93%
Column AGE outliers = 0.00%
Column DIS outliers = 0.99%
Column RAD outliers = 0.00%
Column TAX outliers = 0.00%
Column PTRATIO outliers = 2.96%
Column B outliers = 15.22%
Column LSTAT outliers = 1.38%
Column MEDV outliers = 7.91%
```

In [12]:

```
import warnings
warnings.filterwarnings(action="ignore")
fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in housing.items():
    sns.distplot(v, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



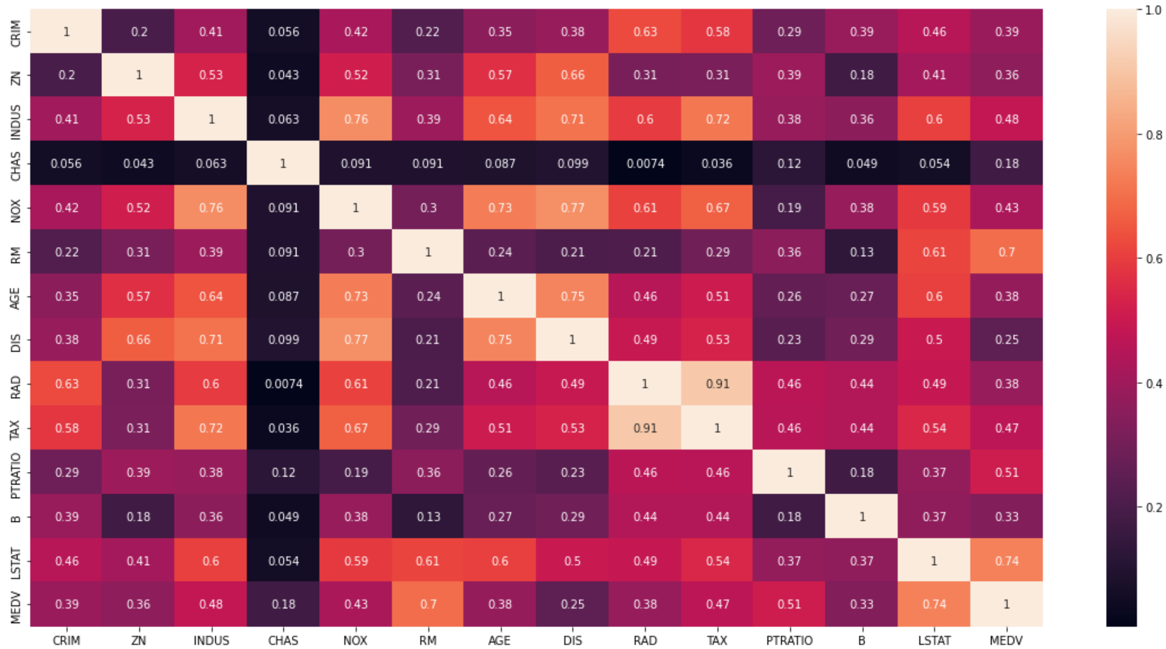
In [13]:

```
##CRIM, ZN, B has highly skewed distributions and CHAS  is categorical disc
```

In [14]:
```
##he pairwise correlation on data.

plt.figure(figsize=(20, 10))
sns.heatmap(housing.corr().abs(),  annot=True)
```

Out[14]: <AxesSubplot:>

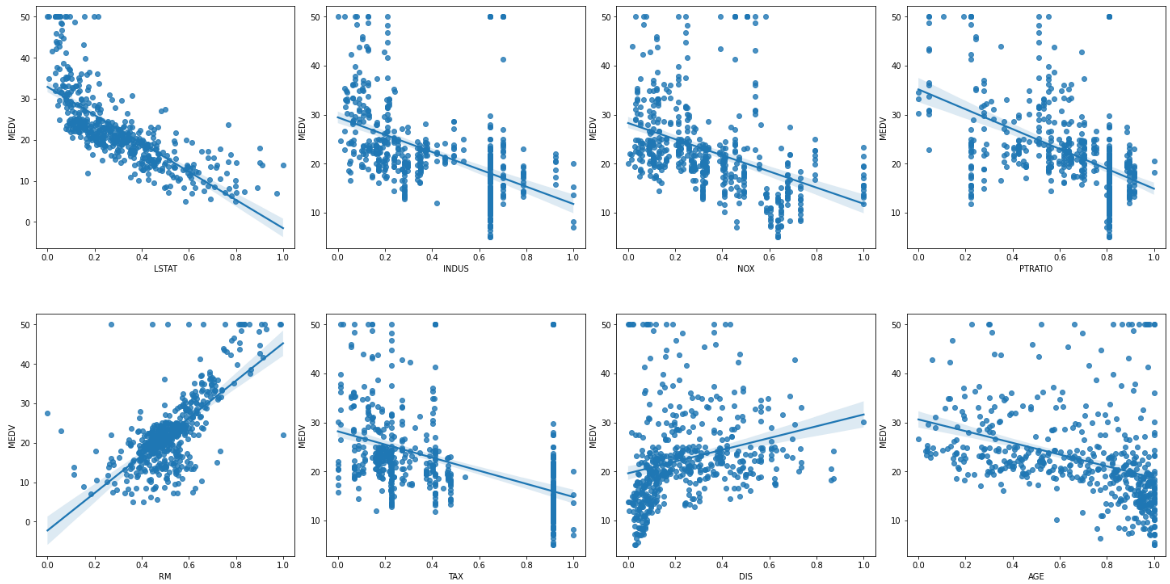


5 From correlation matrix, we see TAX and RAD are highly correlated features. The columns LSTAT, INDUS, RM, TAX, NOX, PTRAIO has a correlation score above 0.5 with MEDV. Let's plot these columns against MEDV.

In [15]:
```
from sklearn import preprocessing
# Let's scale the columns before plotting them against MEDV
min_max_scaler = preprocessing.MinMaxScaler()
column_sels = ['LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE
X = housing.loc[:,column_sels]
Y= housing['MEDV']
```

In [16]:
```
X = pd.DataFrame(data=min_max_scaler.fit_transform(X), columns=column_sels)
```

In [17]:
```
fig, axs = plt.subplots(ncols=4, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for i, k in enumerate(column_sels):
    sns.regplot(y=Y, x=X[k], ax=axs[i])
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



6 Lets Split the Data into Training and Test Set

In [18]:
```
from sklearn.model_selection import train_test_split, cross_val_score, Gric
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,rar
```

In [19]:
```
print(len(X_train))
print(len(Y_test))
```

```
404
102
```

7 Linear Regression

In [20]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, Y_train)
```

Out[20]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [21]:
```
y_pred = lr.predict(X_train)
```

In [22]:
```
lr.score(X_train, Y_train)
```

Out[22]:
```
0.687512913072722
```

In [23]:
```
lr.score(X_test, Y_test)
```

Out[23]:
```
0.7781051528631586
```

8 DecisionTreeRegressor

In [24]:
```
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(X_train, Y_train)
```

Out[24]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

9 RandoMForest Search

In [25]:
```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor(n_estimators=30, random_state=42)
forest_reg.fit(X_train, Y_train)
housing_predictions = forest_reg.predict(X_train)
```

In [26]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [27]:
```
from sklearn.model_selection import cross_val_score
```

In [28]:
```
scores = cross_val_score(tree_reg, X_train, Y_train,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
display_scores(tree_rmse_scores)
```

```
Scores: [4.14216863 3.25295363 5.13911352 5.01684966 3.03141881 3.60385211
 5.19882198 4.49849975 4.17031174 3.81945677]
Mean: 4.1873446596549275
Standard deviation: 0.7349308537515244
```

In [29]:
```
forest_scores = cross_val_score(forest_reg, X_train, Y_train,scoring="neg_m
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [2.90621906 2.43973331 3.36182692 5.20251875 2.97215588 2.72032555
 3.54428578 3.45646114 2.65981615 2.72456893]
Mean: 3.1987911460475944
Standard deviation: 0.7539699060832402
```

10 Evaluation of Model

In [30]:
```
from sklearn.metrics import mean_squared_error
housing_predictions = tree_reg.predict(X_train)
```

In [31]:
```
tree_mse = mean_squared_error(Y_train, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[31]: 0.0

In [32]:
```
housing_predictions = lr.predict(X_train)
lr_mse = mean_squared_error(Y_train, housing_predictions)
lr_rmse = np.sqrt(lr_mse)
lr_rmse
```

Out[32]: 5.097360209306243

In [33]:
```
housing_predictions = forest_reg.predict(X_train)
forest_mse = mean_squared_error(Y_train, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[33]: 1.2818768577081934

11 Evaluation of Model using Cross Validation

In [34]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [35]:
```
from sklearn.model_selection import cross_val_score
```

In [36]:
```
scores = cross_val_score(tree_reg, X_train, Y_train,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
display_scores(tree_rmse_scores)
```

```
Scores: [4.14216863 3.25295363 5.13911352 5.01684966 3.03141881 3.60385211
 5.19882198 4.49849975 4.17031174 3.81945677]
Mean: 4.1873446596549275
Standard deviation: 0.7349308537515244
```

In [37]:
```
forest_scores = cross_val_score(forest_reg, X_train, Y_train,scoring="neg_m
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [2.90621906 2.43973331 3.36182692 5.20251875 2.97215588 2.72032555
 3.54428578 3.45646114 2.65981615 2.72456893]
Mean: 3.1987911460475944
Standard deviation: 0.7539699060832402
```

In [38]:
```
lr_scores = cross_val_score(lr, X_train, Y_train,scoring="neg_mean_squared_
lr_rmse_scores = np.sqrt(-lr_scores)
display_scores(lr_rmse_scores)
```

```
Scores: [5.35864187 4.15512896 4.24465783 8.07236404 4.61776717 4.63800544
 6.35487368 5.21052691 4.99756821 4.11310168]
Mean: 5.1762635800355685
Standard deviation: 1.1612685288843354
```

12 Pipeline

In [39]:
```
from sklearn.pipeline import Pipeline
```

In [40]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

In [41]:

```
# We will crete a pipeline here and add Imputer and scaling into pipeline
my_pipeline=Pipeline([
('imputer',SimpleImputer(strategy="median")),#We can add as many process as
('std_scaler',StandardScaler())
])

# fit and transform the pipeline on features
#we are just giving a new name to the features as X_train and labels to Y_t
X_train=my_pipeline.fit_transform(X_train)
Y_train=Y_train.to_numpy()

# this below step is very important because if our training feature have pa
# pass through pipeline
X_test=my_pipeline.fit_transform(X_test)
```

In [42]:

```
X_test=my_pipeline.fit_transform(X_test)
```