In [ ]:

```
#importing all the important libraries
```

In [2]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import sklearn
```

In [4]:

```
#loading the boston dataset from the sklearn library
from sklearn.datasets import load_boston
```

In [5]:

```
boston = load_boston()
```

In [6]:

```
bos = pd.DataFrame(boston.data)
```

In [7]:

```
bos.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [8]:

```
#Renaming the columns
bos.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
bos.head()
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [9]:

```
#As the target variable is missing we load it from the sklear libaray
bos['MEDV'] = boston.target
```

In [10]:

```
bos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
MEDV       506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

there is 14 columns with non-null values

In [11]:

```
bos.describe()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
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%	3.677083	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	1

In [12]:

```
bos.isnull().sum()
```

Out[12]:

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

there is no null value present in the whole dataset

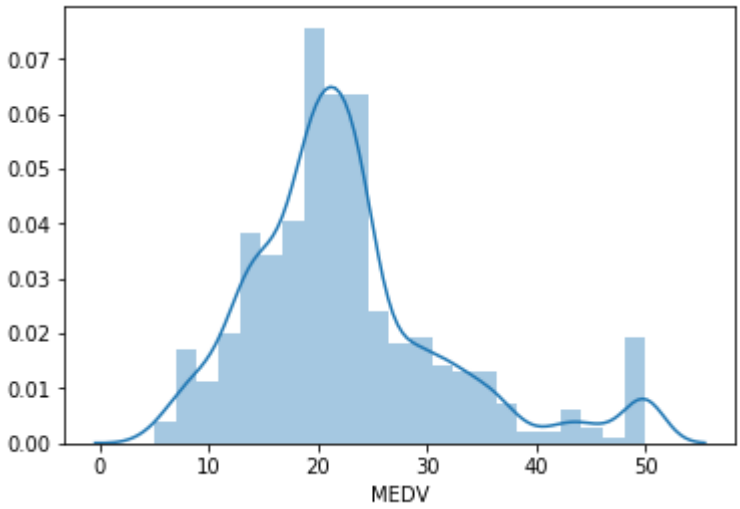
EDA

we will see the distribution of the target variable MEDV

In [13]:

```
sns.distplot(bos['MEDV'])
plt.show()
```

C:\Users\DELL\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will r esult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval



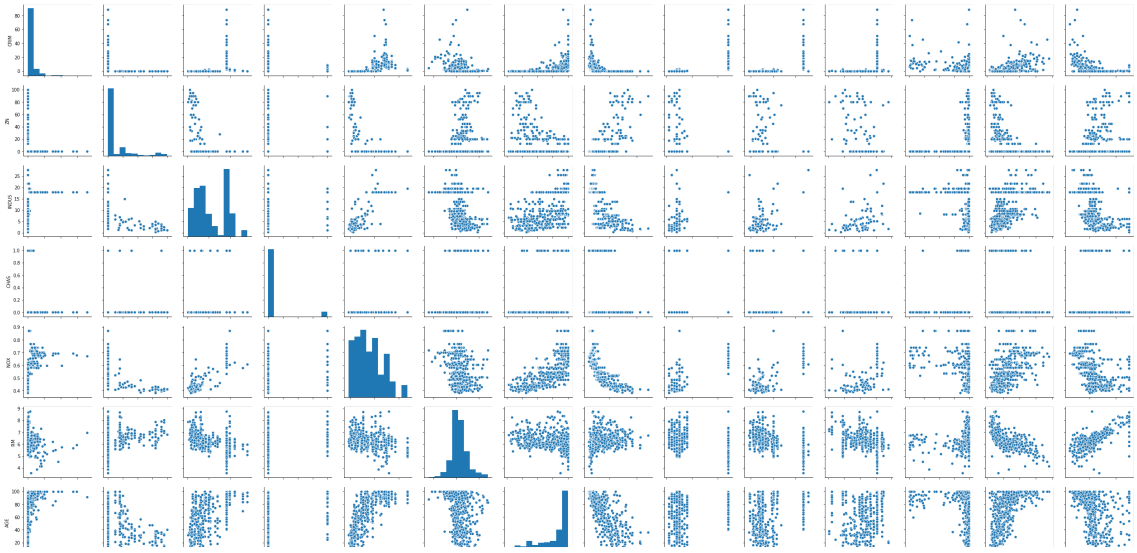
the values of MEDV is normally distributed with some of the outliers

In [14]:

```
sns.pairplot(bos)
```

Out[14]:

<seaborn.axisgrid.PairGrid at 0x2781a788128>

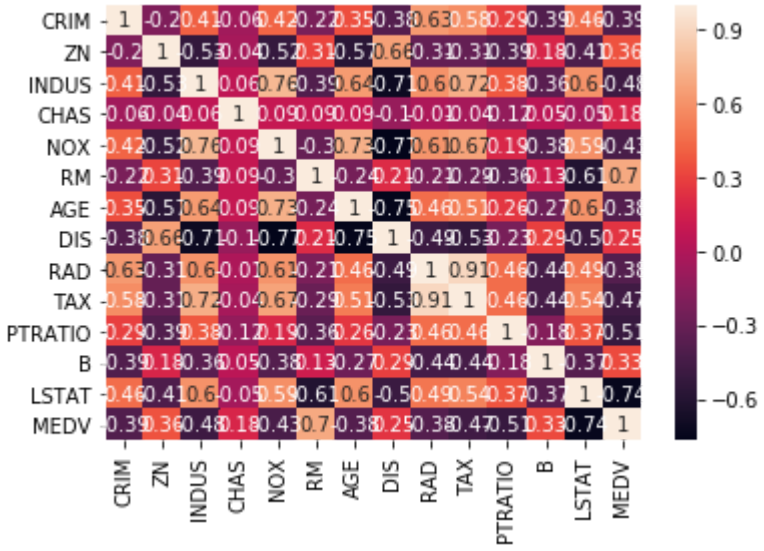


In [15]:

```
corr_mat = bos.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=corr_mat, annot=True)
```

Out[15]:

<matplotlib.axes._subplots.AxesSubplot at 0x27821f355f8>



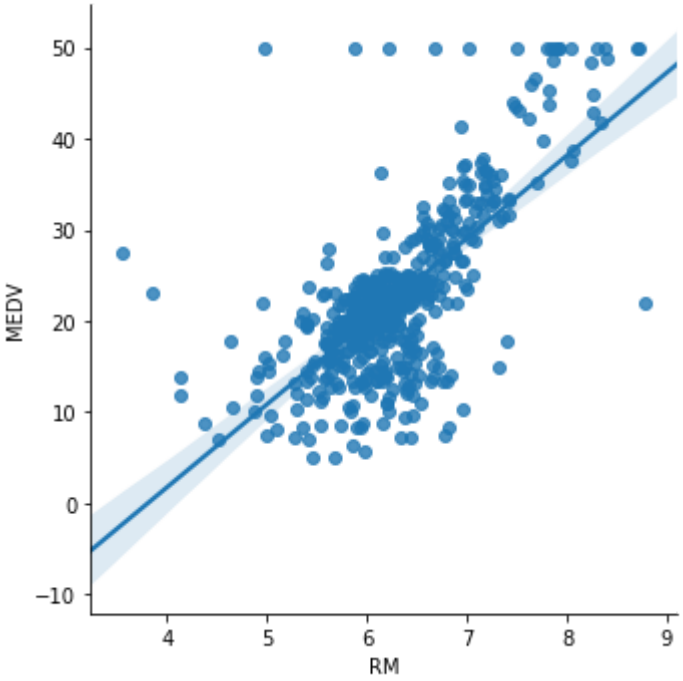
from the above two results we can see that the variable RM is positively related with MEDV

In [16]:

```
#we will now closely see the relationship between the two variable
sns.lmplot(x = 'RM', y = 'MEDV', data = bos)
```

Out[16]:

<seaborn.axisgrid.FacetGrid at 0x27823c58048>



Splitting the data into Training and Test data

In [52]:

```
bos.columns
```

Out[52]:

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

In [53]:

```
X = bos[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT']]
```

In [54]:

```
y = bos['MEDV']
```

In [57]:

```
from sklearn.model_selection import train_test_split
```

In [58]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 10)
```

Training the model

In [60]:

```
from sklearn.linear_model import LinearRegression
```

In [61]:

```
lm = LinearRegression()
```

In [62]:

```
lm.fit(X_train, y_train)
```

Out[62]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [64]:

```
df = pd.DataFrame(lm.coef_, X.columns)
df.columns = ['Coefficients']
df
```

Out[64]:

	Coefficients
CRIM	-0.128181
ZN	0.063198
INDUS	-0.007576
CHAS	1.974515
NOX	-16.271989
RM	3.108456
AGE	0.016292
DIS	-1.483014
RAD	0.303988
TAX	-0.012082
PTRATIO	-0.820306
B	0.011419
LSTAT	-0.581626

Prediction

In [65]:

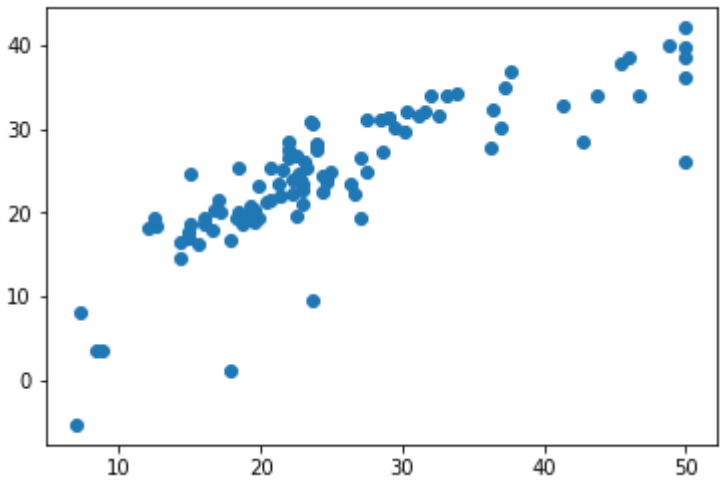
```
#making prediction on the test data
prediction = lm.predict(X_test)
```

In [66]:

```
#visualizing the actual and predicted value with scatter plot
plt.scatter(y_test, prediction)
```

Out[66]:

<matplotlib.collections.PathCollection at 0x1cffbe6df98>



In [67]:

```
df1 = pd.DataFrame({'Actual': y_test, 'Predicted':prediction})
df2 = df1.head(10)
df2
```

Out[67]:

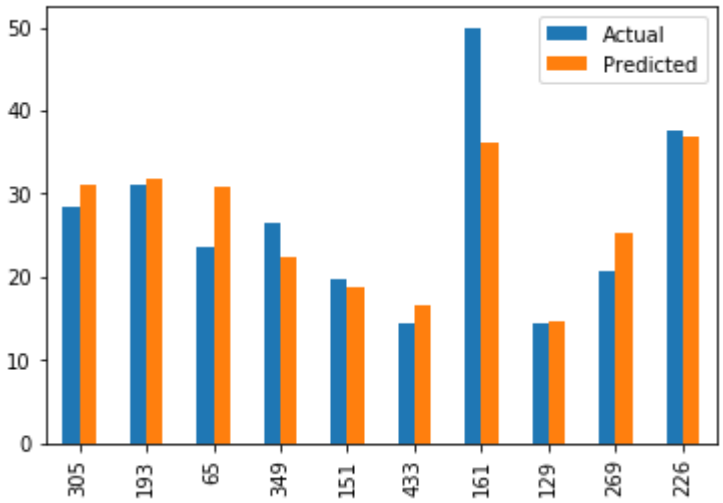
	Actual	Predicted
305	28.4	31.078964
193	31.1	31.721694
65	23.5	30.873149
349	26.6	22.282350
151	19.6	18.856061
433	14.3	16.471325
161	50.0	36.050042
129	14.3	14.640323
269	20.7	25.240786
226	37.6	36.920739

In [68]:

```
#visualizing the actual and predicted values using barplot
df2.plot(kind = 'bar')
```

Out[68]:

<matplotlib.axes._subplots.AxesSubplot at 0x1cffc71e128>



Model Evaluation

Evaluating the model by taking the Mean Absolute Error, Mean squared Error, Root Mean Squared Error and R-Squared Error

In [73]:

```
from sklearn import metrics
from sklearn.metrics import r2_score
```

In [75]:

```
print('MAE', metrics.mean_absolute_error(y_test, prediction))
print('MSE', metrics.mean_squared_error(y_test, prediction))
print('RMSE', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
print('R squared error', r2_score(y_test, prediction))
```

```
MAE 4.06141918295471
MSE 34.413968453138565
RMSE 5.866341999333023
R squared error 0.6709339839115628
```

In [ ]:

```

```