In [15]:
```
#5. Data Analytics I:
#Create a Linear Regression Model using Python/R to predict home prices using
#(https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contair
#Boston through different parameters. There are 506 samples and 14 feature var
#The objective is to predict the value of prices of the house using the given

#1st try..boston.csv nnnnn
#xerox wala printout
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston

boston_dataset=load_boston()
boston_dataset.keys()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function load_boston is deprecated; `load_boston` is deprecat
ed in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer
to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of th
is
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Out[2]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_mod
ule'])
```

In [3]:
```
boston=pd.DataFrame(boston_dataset.data,
                    columns=boston_dataset.feature_names)
boston.head()
```

Out[3]:

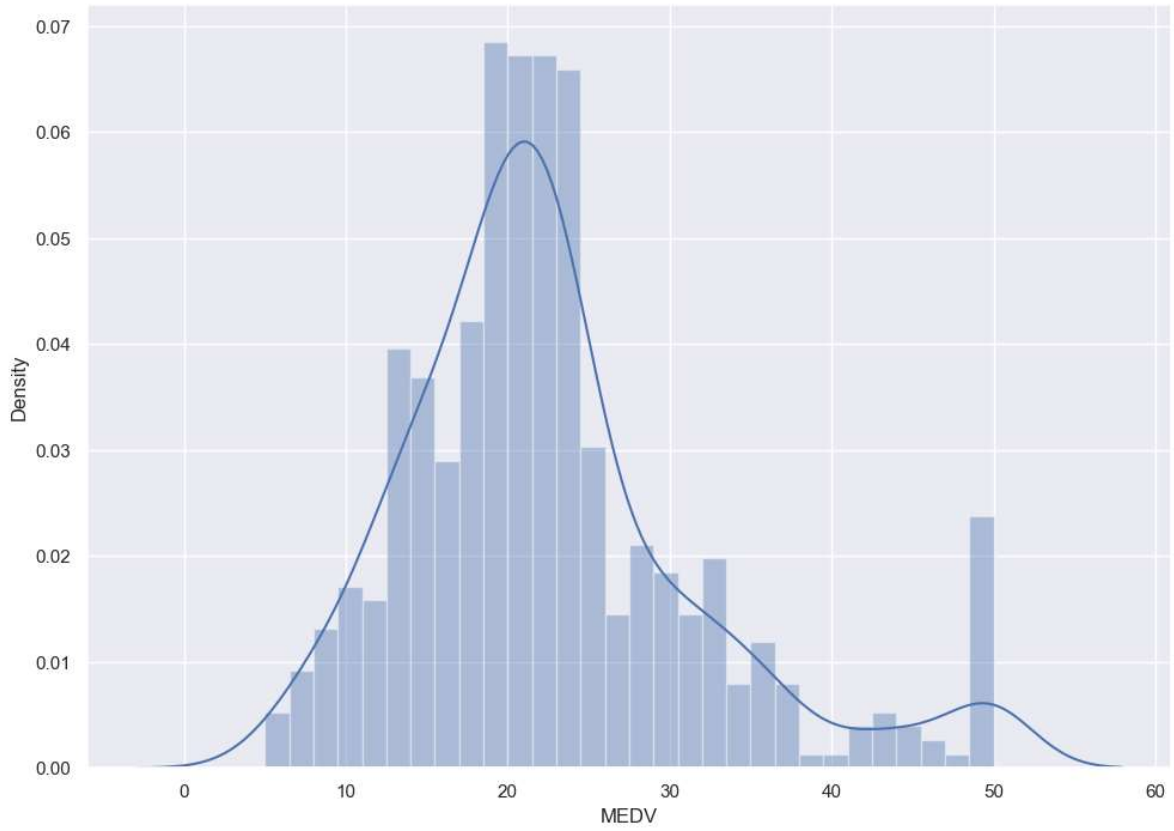
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
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.

In [4]:
```
boston['MEDV']=boston_dataset.target
boston.isnull().sum()
```

Out[4]:
```
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
```

In [5]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(boston['MEDV'],bins=30)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



In [6]:
```
correlation_matrix=boston.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[6]: <AxesSubplot:>

In [7]:
```
x=pd.DataFrame(np.c_[boston['LSTAT'],
                     boston['RM']],
               columns=['LSTAT','RM'])
y=boston['MEDV']
```

In [8]:
```
from sklearn.model_selection
import train_test_split
X_train, X_test, Y_train,
Y_test=train_test_split(x,y, test_size=0.2, random_state=0)
```

In [9]:
```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

In [10]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
lin_model=LinearRegression()
lin_model.fit(X_train,Y_train)
```

Out[10]:
```
LinearRegression()
```

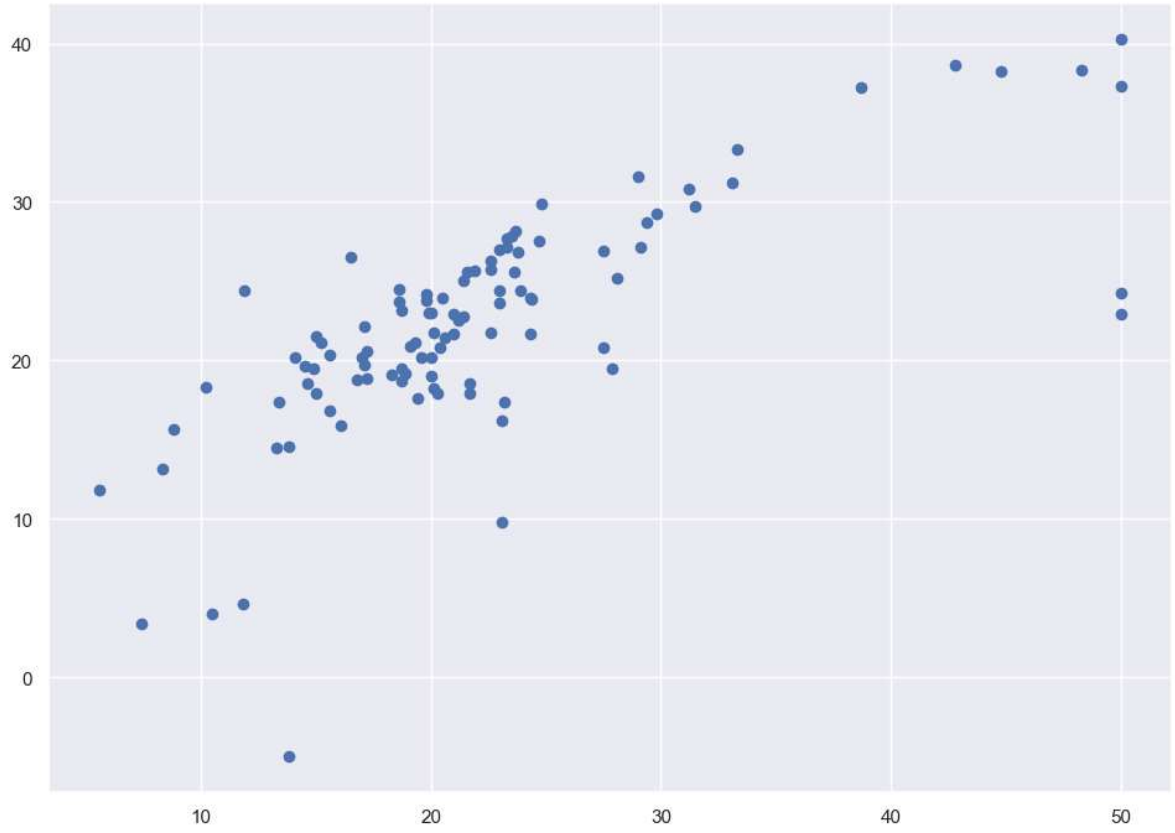
In [11]:
```
y_train_predict=lin_model.predict(X_train)
rmse=(np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2=r2_score(Y_train, y_train_predict)
print("The model performance of training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 is {}'.format(r2))
print('\n')
```

```
The model performance of training set
--------------------------------------
RMSE is 5.3656571342244215
R2 is 0.6618625964841895
```

In [13]:
```
y_test_predict=lin_model.predict(X_test)
rmse=(np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2=r2_score(Y_test, y_test_predict)
print("The model performance of testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 is {}'.format(r2))
print('\n')
```

```
The model performance of testing set
--------------------------------------
RMSE is 6.114172522817783
R2 is 0.5409084827186417
```

In [14]:
```
plt.scatter(Y_test, y_test_predict)
plt.show()
```



In [ ]:
```

```