In [1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
plt.figure(figsize=(8,6))
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

```
<Figure size 800x600 with 0 Axes>
```

In [ ]:
```
```

In [3]:
```
df=pd.read_csv("D:\Downloads\HousingData.csv")
```

In [4]:
```
df.head()
```

Out[4]:

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

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 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    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

In [7]:
```
df['CRIM'].fillna(value=df['CRIM'].mean(),inplace=True)
df['ZN'].fillna(value=df['ZN'].mean(),inplace=True)
df['INDUS'].fillna(value=df['INDUS'].mean(),inplace=True)
df['CHAS'].fillna(value=df['CHAS'].mean(),inplace=True)
df['AGE'].fillna(value=df['AGE'].mean(),inplace=True)
df['LSTAT'].fillna(value=df['LSTAT'].mean(),inplace=True)
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
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
```

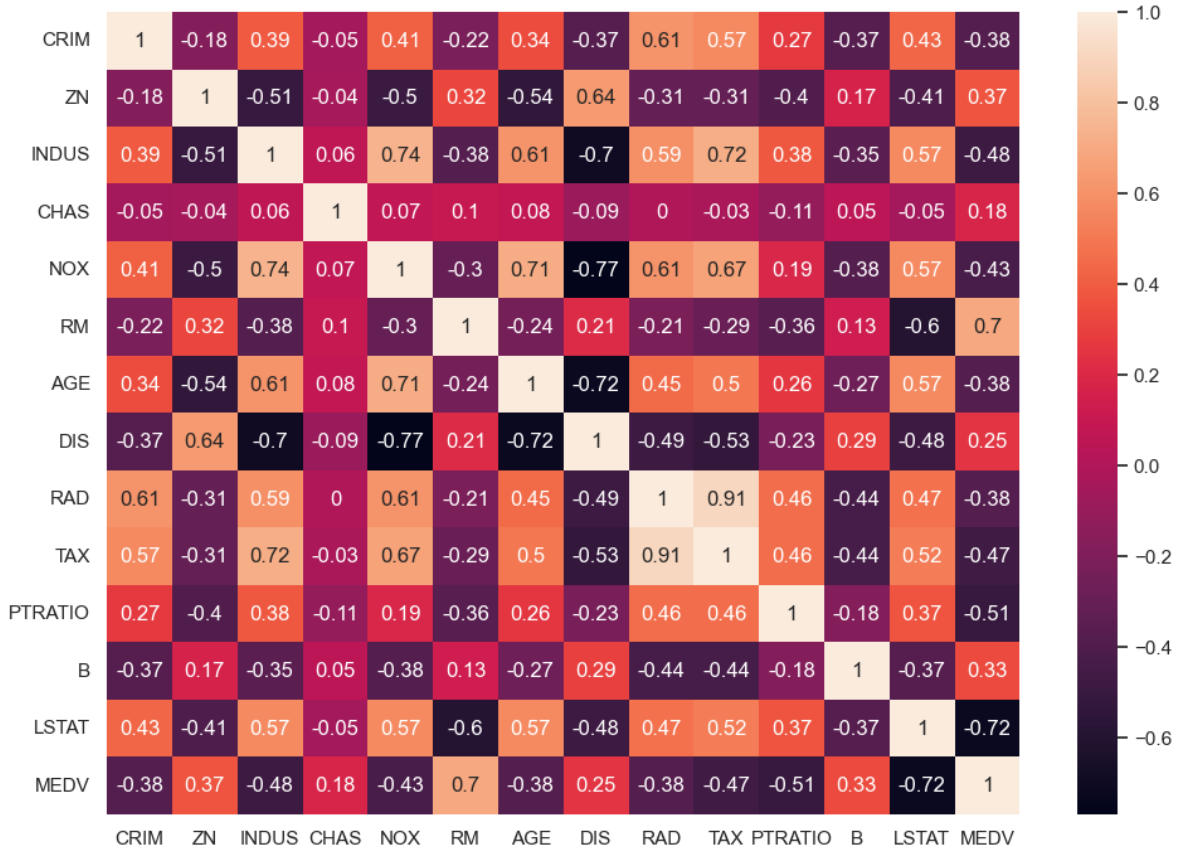
In [9]:
```
df.describe()
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3
std	8.545770	22.921051	6.699165	0.250233	0.115878	0.702617	27.439466	2
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1
25%	0.083235	0.000000	5.190000	0.000000	0.449000	5.885500	45.925000	2
50%	0.290250	0.000000	9.900000	0.000000	0.538000	6.208500	74.450000	3
75%	3.611874	11.211934	18.100000	0.000000	0.624000	6.623500	93.575000	5
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12

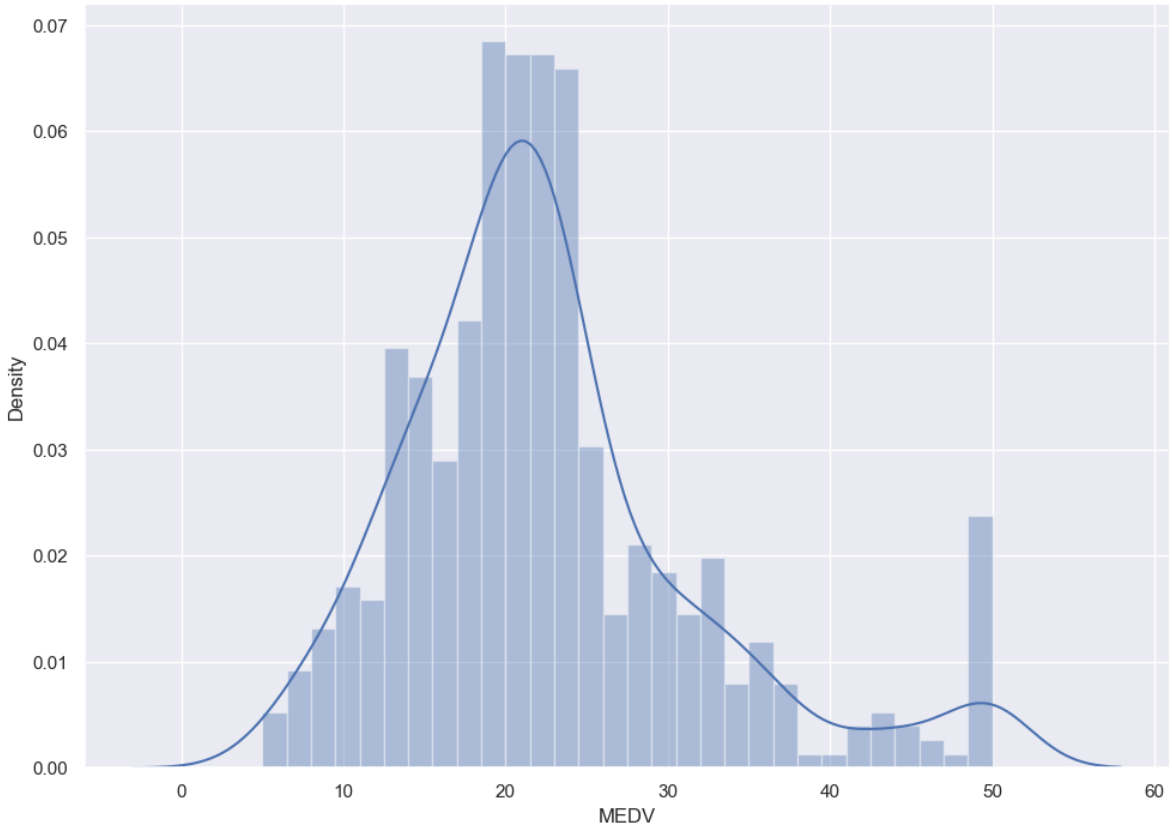
In [10]:
```
correlation_matrix=df.corr().round(2)
sns.heatmap(data=correlation_matrix,annot=True)
```

Out[10]: <Axes: >

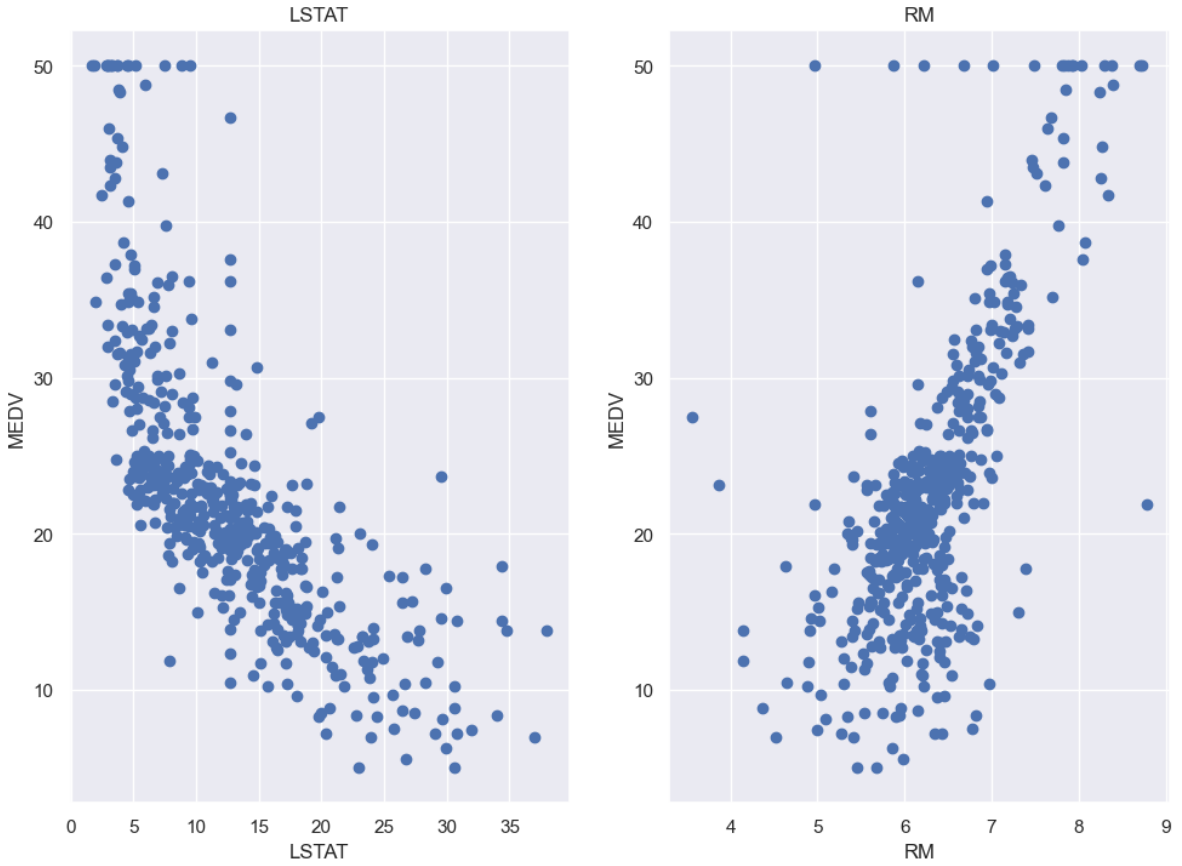


In [12]:
```
sns.distplot(df['MEDV'], bins=30)
plt.show()
```

```
C:\Users\shris\AppData\Local\Temp\ipykernel_12736\1549546113.py:1: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gis
t.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['MEDV'], bins=30)
```



In [13]:
```
features= ['LSTAT', 'RM']
for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = df[col]
    y = df['MEDV']
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



In [14]:
```
from sklearn.model_selection import train_test_split

X = df.loc[:, df.columns != 'MEDV']
y = df.loc[:, df.columns == 'MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

In [15]:
```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
mms.fit(X_train)
X_train = mms.transform(X_train)
X_test = mms.transform(X_test)
```

In [16]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(512, input_shape=(13, ), activation='relu', name='dense_1'))
model.add(Dense(256, activation='relu', name='dense__layer_1'))
model.add(Dense(128, activation='relu', name='dense__layer_2'))
model.add(Dense(64, activation='relu', name='dense_layer_3'))
model.add(Dense(1, activation='relu', name='dense_output'))

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 512)               7168

 dense__layer_1 (Dense)      (None, 256)               131328

 dense__layer_2 (Dense)      (None, 128)               32896

 dense_layer_3 (Dense)       (None, 64)                8256

 dense_output (Dense)        (None, 1)                 65

=================================================================
Total params: 179713 (702.00 KB)
Trainable params: 179713 (702.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [17]:
```
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05, verbo
```

```
Epoch 1/100
11/11 [==============================] - 1s 23ms/step - loss: 305.2480 - m
ae: 13.9452 - val_loss: 103.4119 - val_mae: 7.1613
Epoch 2/100
11/11 [==============================] - 0s 7ms/step - loss: 77.6854 - ma
e: 6.2926 - val_loss: 66.7409 - val_mae: 5.9860
Epoch 3/100
11/11 [==============================] - 0s 8ms/step - loss: 56.0574 - ma
e: 5.3036 - val_loss: 59.0851 - val_mae: 6.6060
Epoch 4/100
11/11 [==============================] - 0s 8ms/step - loss: 44.3206 - ma
e: 4.6928 - val_loss: 49.6580 - val_mae: 5.0873
Epoch 5/100
11/11 [==============================] - 0s 9ms/step - loss: 41.3859 - ma
e: 4.6344 - val_loss: 53.8902 - val_mae: 4.5734
Epoch 6/100
11/11 [==============================] - 0s 9ms/step - loss: 32.0731 - ma
e: 4.0102 - val_loss: 49.9080 - val_mae: 4.5274
Epoch 7/100
```

In [18]:
```
mse_nn, mae_nn = model.evaluate(X_test, y_test)

print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)
```

```
5/5 [==============================] - 0s 4ms/step - loss: 23.4769 - mae: 3.3
117
Mean squared error on test data:  23.47692108154297
Mean absolute error on test data:  3.3117029666900635
```

In [ ]:

In [ ]:

In [ ]: