ICE-9

ANANYA SAMALA -11527196

In [171]:
```
import numpy as np
import pandas as pd

import os,sys
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

```

In [172]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [173]:
```
df = pd.read_csv("play_tennis.csv")
```

In [174]:
```
df.head()
```

Out[174]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes

In [175]:
```
df.tail()
```

Out[175]:

	day	outlook	temp	humidity	wind	play
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

1.Do an exploratory data analysis of your dataset and show a part of your dataset

In [176]:
```
df.shape
```
Out[176]:
```
(14, 6)
```

2.Show the datatypes of your features

In [177]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   day       14 non-null     object
 1   outlook   14 non-null     object
 2   temp      14 non-null     object
 3   humidity  14 non-null     object
 4   wind      14 non-null     object
 5   play      14 non-null     object
dtypes: object(6)
memory usage: 800.0+ bytes
```

In [178]:

```python
for col in df.columns:
    print(format(col))
    print(df[col].value_counts())
    print('\n')
```

```
day
D1     1
D2     1
D3     1
D4     1
D5     1
D6     1
D7     1
D8     1
D9     1
D10    1
D11    1
D12    1
D13    1
D14    1
Name: day, dtype: int64


outlook
Sunny       5
Rain        5
Overcast    4
Name: outlook, dtype: int64


temp
Mild    6
Hot     4
Cool    4
Name: temp, dtype: int64


humidity
High      7
Normal    7
Name: humidity, dtype: int64


wind
Weak      8
Strong    6
Name: wind, dtype: int64


play
Yes    9
No     5
Name: play, dtype: int64
```

In [179]:
```
df.describe()
```

Out[179]:

	day	outlook	temp	humidity	wind	play
count	14	14	14	14	14	14
unique	14	3	3	2	2	2
top	D1	Sunny	Mild	High	Weak	Yes
freq	1	5	6	7	8	9

3. Check if you have any missing values in your dataset.

In [180]:
```
#There are no missing values in the dataset
#df.['play'].isnull()
```

In [181]:
```
df["day"] = [float(str(i).replace("D", "")) for i in df["day"]]
df["day"]
```

Out[181]:
```
0      1.0
1      2.0
2      3.0
3      4.0
4      5.0
5      6.0
6      7.0
7      8.0
8      9.0
9     10.0
10    11.0
11    12.0
12    13.0
13    14.0
Name: day, dtype: float64
```

In [182]:
```
categorical_val = {"outlook": {'Rain': 0 ,'Overcast': 1 ,'Sunny' : 2},
                   "temp": {'Cool':0,'Mild':1,'Hot':2},
                   "humidity": {'Normal': 0,'High': 1},
                   "wind": {'Weak' : 0,'Strong': 1},
                   "play": {'No': 0, 'Yes': 1}}

df_obj = df.replace(categorical_val)

```

In [183]:
```
df_obj.head()
```
Out[183]:

	day	outlook	temp	humidity	wind	play
0	1.0	2	2	1	0	0
1	2.0	2	2	1	1	0
2	3.0	1	2	1	0	1
3	4.0	0	1	1	0	1
4	5.0	0	0	0	0	1

In [184]:
```
df_obj.tail()
```
Out[184]:

	day	outlook	temp	humidity	wind	play
9	10.0	0	1	0	0	1
10	11.0	2	1	0	1	1
11	12.0	1	1	1	1	1
12	13.0	1	2	0	0	1
13	14.0	0	1	1	1	0

In [185]:
```
df_obj
```
Out[185]:

	day	outlook	temp	humidity	wind	play
0	1.0	2	2	1	0	0
1	2.0	2	2	1	1	0
2	3.0	1	2	1	0	1
3	4.0	0	1	1	0	1
4	5.0	0	0	0	0	1
5	6.0	0	0	0	1	0
6	7.0	1	0	0	1	1
7	8.0	2	1	1	0	0
8	9.0	2	0	0	0	1
9	10.0	0	1	0	0	1
10	11.0	2	1	0	1	1
11	12.0	1	1	1	1	1
12	13.0	1	2	0	0	1
13	14.0	0	1	1	1	0

4.Split data into separate training and test set

In [186]:
```
X = df_obj.drop(['play'],axis=1)
y = df_obj['play']
```

In [187]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0

X_train.shape, X_test.shape
```

Out[187]:
```
((9, 5), (5, 5))
```

5.Build a Decision Tree Classifier with criterion “gini index” and add depth to it as well

In [188]:
```
# instantiate the DecisionTreeClassifier model with criterion gini inde
model = DecisionTreeClassifier(criterion = 'gini', max_depth = 2)
# fit the model
model.fit(X_train, y_train)
```

Out[188]:
```
DecisionTreeClassifier(max_depth=2)
```

6.Check accuracy score with criterion “gini index”

In [189]:
```
y_pred = model.predict(X_test)
```

In [190]:
```
y_test
```

Out[190]:
```
1     0
8     1
11    1
5     0
6     1
Name: play, dtype: int64
```

In [191]:
```
y_pred
```

Out[191]:
```
array([0, 1, 1, 1, 1])
```

In [192]:
```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Out[192]:
```
0.8
```

7. Show the train-set and test-set accuracy and compare them

In [193]:
```
print('Training set score' +str(accuracy_score(y_train,model.predict(X_
print('Test set score:' +str(accuracy_score(y_test,y_pred)))
```

```
Training set score0.8888888888888888
Test set score:0.8
```

In [194]:
```python
from sklearn.metrics import confusion_matrix
```

In [195]:
```python
y_true = (y_test)
```

In [196]:
```python
y_pred = (y_pred)
```

In [197]:
```python
cf = confusion_matrix(y_true, y_pred)
```

In [198]:
```python
print('CONFUSION MATRIX\n')
print('\t' + 'Yes' + '\t' + 'No')
print('Yes' + '\t' + str(cf[0,0]) + '\t' + str(cf[0,1]))
print('No' + '\t' + str(cf[1,0]) + '\t' + str(cf[1,1]))
```

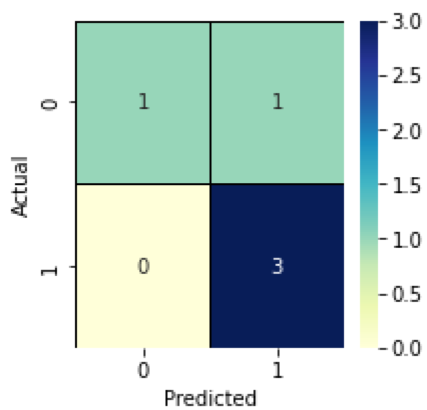
```
CONFUSION MATRIX

        Yes     No
Yes     1       1
No      0       3
```

8. Explain whether the model is overfitting or underfitting

In [ ]:
```python
#It is not overfitting and underfitting
```

9. Confusion matrix and a classification report

In [199]:
```python
from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(3,3))

sns.heatmap(confusion_matrix, annot=True, fmt='d', xticklabels=np.uniqu
            yticklabels=np.unique(y_test), linewidths = 0.7, linecolor
            cmap = 'YlGnBu')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [200]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.75      1.00      0.86         3

    accuracy                           0.80         5
   macro avg       0.88      0.75      0.76         5
weighted avg       0.85      0.80      0.78         5
```

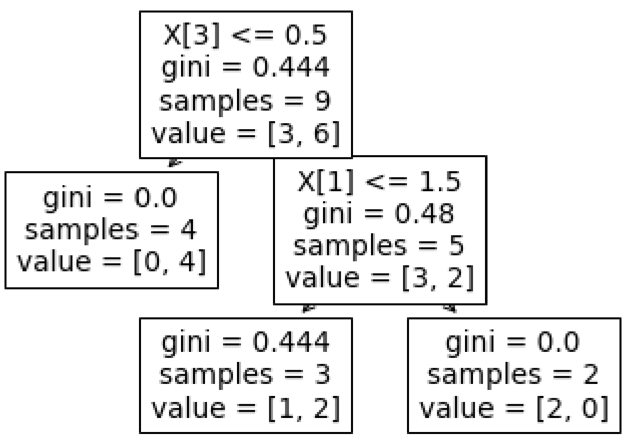
10. Show a visualization of your final decision tree

In [201]:

```
from sklearn import tree

tree.plot_tree(model)
```

Out[201]:

```
[Text(0.4, 0.8333333333333334, 'X[3] <= 0.5\ngini = 0.444\nsamples = 9\nv
alue = [3, 6]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.6, 0.5, 'X[1] <= 1.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.444\nsamples = 3\nvalue = [1,
2]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]')]
```



In [ ]:

```

```

Implement a decision tree by using the criterion “Entropy”

1) Do an exploratory data analysis of your dataset and show a part of your dataset.b

2.Show the datatypes of your features

3. Check if you have any missing values in your dataset.

4.Split data into separate training and test set

5. Build a Decision Tree Classifier with criterion “entropy” and add depth to it as well

6.Check accuracy score with criterion “entropy”

7. Show the train-set and test-set accuracy and compare them

In [202]:
```
model_entropy = DecisionTreeClassifier(criterion = 'entropy', max_depth
# fit the model
model_entropy.fit(X_train, y_train)
```
Out[202]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In [203]:
```
y_pred1 = model_entropy.predict(X_test)
```

In [204]:
```
y_test
```
Out[204]:
```
1     0
8     1
11    1
5     0
6     1
Name: play, dtype: int64
```

In [205]:
```
y_pred1
```
Out[205]:
```
array([1, 1, 0, 1, 1])
```

In [206]:
```
accuracy_score(y_test, y_pred1)
```
Out[206]:
```
0.4
```

In [207]:
```
print('Accuracy on Training Set :- '+ str(accuracy_score(y_train, model
```
```
Accuracy on Training Set :- 0.8888888888888888
```

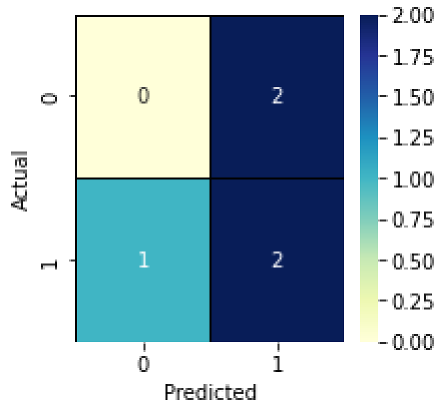
In [208]:
```
print('Accuracy on Testing Set :- ' + str(accuracy_score(y_test, y_pred
```
```
Accuracy on Testing Set :- 0.4
```

In [209]:
```
y_true = (y_test)
```

In [210]:
```
y_pred = (y_pred)
```

9. Confusion matrix and a classification report

In [211]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(y_test, y_pred1)

fig, ax = plt.subplots(figsize=(3,3))

sns.heatmap(confusion_matrix, annot=True, fmt='d', xticklabels=np.uniqu
            yticklabels=np.unique(y_test), linewidths = 0.7, linecolor
            cmap = 'YlGnBu')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



8. Explain whether the model is overfitting or underfitting

In [ ]:
```
#It is not overfitting and underfitting
```

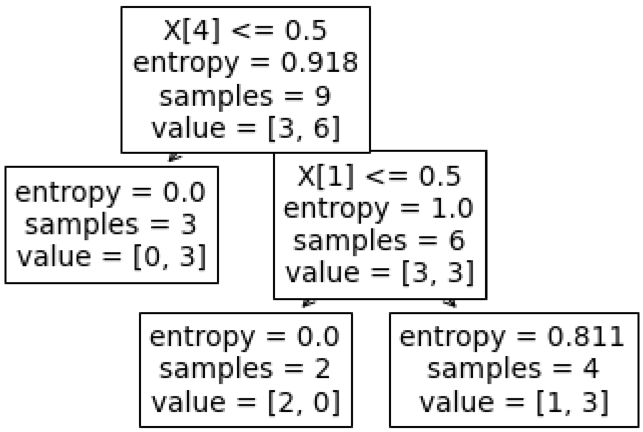
In [212]:
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.75      1.00      0.86         3

    accuracy                           0.80         5
   macro avg       0.88      0.75      0.76         5
weighted avg       0.85      0.80      0.78         5
```

10. Show a visualization of your final decision tree

In [97]:
```
1 tree.plot_tree(model_entropy)
```

Out[97]:
```
[Text(0.4, 0.8333333333333334, 'X[4] <= 0.5\nentropy = 0.918\nsamples = 9
\nvalue = [3, 6]'),
 Text(0.2, 0.5, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6, 0.5, 'X[1] <= 0.5\nentropy = 1.0\nsamples = 6\nvalue = [3,
3]'),
 Text(0.4, 0.16666666666666666, 'entropy = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.8, 0.16666666666666666, 'entropy = 0.811\nsamples = 4\nvalue =
[1, 3]')]
```



In [ ]:
```
1
```

In [ ]:
```
1
```

Linear Regression

Read and understand the data. Perform an Exploratory Data Analysis or EDA of your dataset, and show a part of your dataset as well.

In [26]:
```
1 LinReg = pd.read_csv('housingdata-1.csv')
```

In [27]:
```
LinReg.head
```

Out[27]:
```
<bound method NDFrame.head of          CRIM    ZN  INDUS  CHAS    NOX
RM   AGE     DIS  RAD  TAX  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
```

In [29]:
```
LinReg.isna().sum()
```

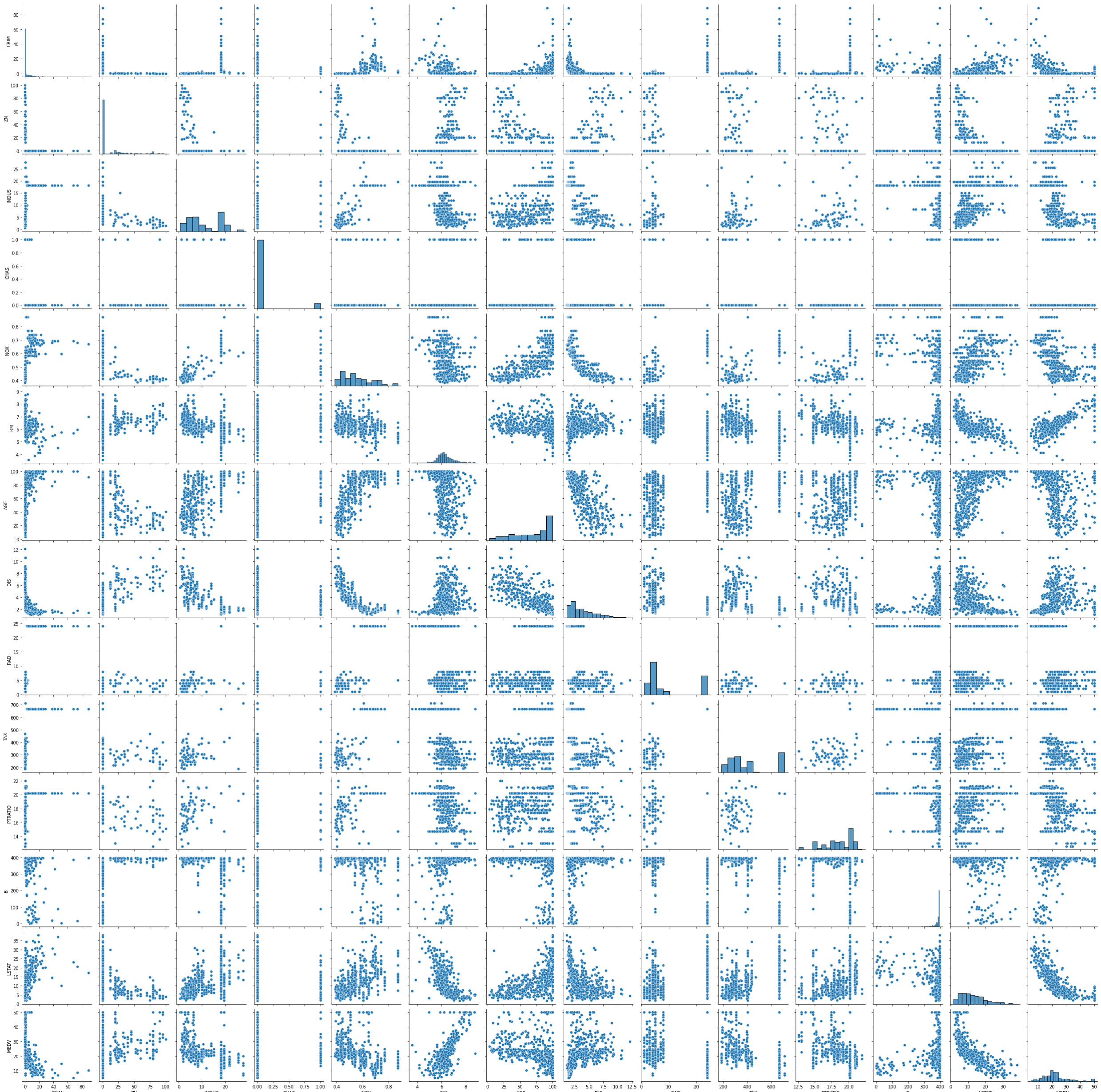
Out[29]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [30]:
```
LinReg.dtypes
```

Out[30]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX          int64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

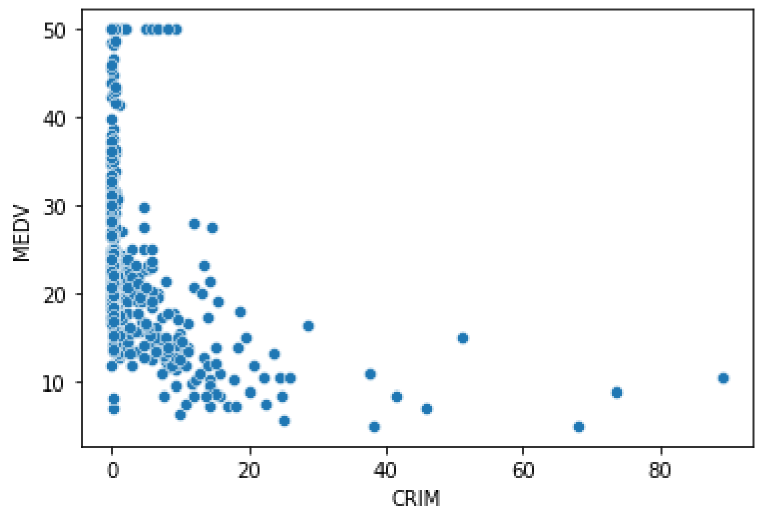
In [31]:
```
sns.pairplot(LinReg)
```

Out[31]: <seaborn.axisgrid.PairGrid at 0x7fa9d82e9fd0>



In [33]:

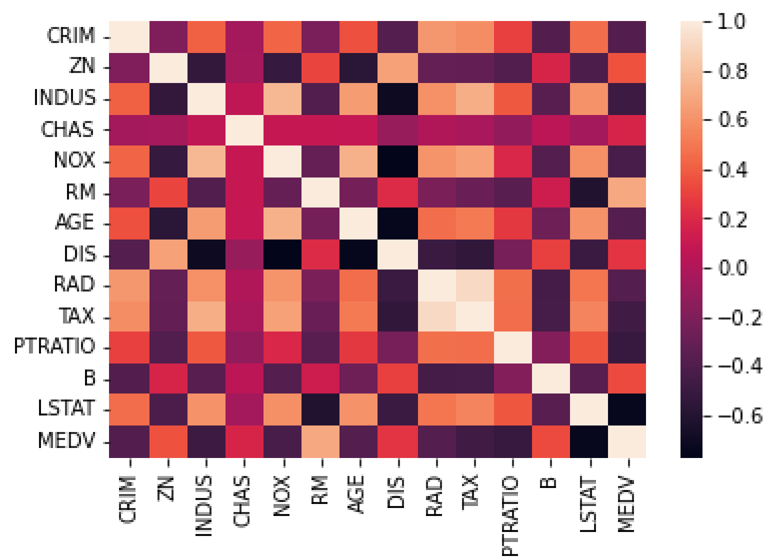
```
for i in list(LinReg.columns):
    plt.figure()
    sns.scatterplot(x = i, y = 'MEDV', data = LinReg)
```



In [34]:

```
sns.heatmap(LinReg.corr())
```

Out[34]: <AxesSubplot:>



Show some visualization of your data to find correlation between the features. For example, seaborn.pairplot() or any other methods to see which columns are the most correlated to “MEDV

In [36]:
```python
LinReg.corr()
```

Out[36]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.
MEDV	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.

Observe which features has a linear relationship with the Median House Value (MEDV) and predict MEDV taking only that feature as feature by using linear regression.

In [37]:
```python
X, y = LinReg['LSTAT'], LinReg['MEDV']
```

In [38]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
```

In [39]:

```
print('X Train :- \n\n' + str(X_train))
print('\nX Test :- \n\n' + str(X_test))
print('\nY Train :- \n\n' + str(np.array(y_train)))
print('\nY Test :- \n\n' + str(np.array(y_test)))
```

```
X Train :-

44      9.55
184    13.98
159     7.39
70      6.72
130    12.60
       ...
217     9.69
107    14.09
162     1.92
363    14.64
478    18.03
Name: LSTAT, Length: 354, dtype: float64

X Test :-

266    14.79
504     6.48
50     13.45
367    13.33
232     2.47
       ...
505     7.88
87      8.44
235    10.88
45     10.21
395    17.12
Name: LSTAT, Length: 152, dtype: float64

Y Train :-

[21.2 26.4 23.3 24.2 19.2 24.7 14.4 10.5  8.1 50.  23.7 43.5 44.  10.2
 27.5 13.9 20.3 28.  38.7 16.5 10.2 26.5 22.  34.9 34.9 19.3 24.4 12.3
 20.9 13.3 22.  23.7 18.7 15.  19.4 15.4 27.9 16.1 25.  20.4 50.  18.7
 15.2 13.4 15.2 16.1 19.6 19.6 28.2 31.5 20.1 20.3 18.9 11.7 22.8 21.6
 14.5 24.8 20.1 28.1 24.6 15.1 21.7 19.4 16.2  7.4 23.9 33.2  5.  13.1
 10.5 17.4 29.8 30.3 50.  21.4 50.  19.5 19.1 27.1 28.4 50.  22.6 18.8
 17.6 23.1 17.8 43.1 12.5 50.  22.5 14.5 23.4 22.4 21.7 44.8 20.4 42.8
 19.2 31.6 35.2 36.5 23.2 50.  23.  20.  32.5 18.3 17.4 24.3 24.8 23.7
 37.  21.  36.4 31.7 19.9 27.   8.8 20.6 17.2 29.  46.7 22.6 17.3 16.7
 13.8 32.7 19.9 24.6  6.3 20.5 23.3 23.1 27.1 10.2 14.3 19.4 20.5 25.
 20.  19.8 26.7 36.2 24.  21.7 13.2 33.4 16.8 10.9 23.1 18.5 32.4 23.8
  8.5 20.4 14.3 17.8 35.1 19.8 17.4 25.  14.1 16.5 29.6 19.3  8.7 22.3
 50.  21.6 22.5 19.1 22.7 19.1 24.4 26.2 36.1  5.  21.4 48.3 22.9 25.3
 18.1 19.5 16.6 22.  17.5 41.3 21.1 24.5 29.4 10.4 18.9 15.2 18.8 20.8
 26.6 21.7 15.7 19.  33.8 24.5  7.  19.7 30.1 28.5 19.6 10.4 34.9 22.6
 32.  21.4 21.7 20.2 14.1 22.  21.8 16.3  5.6 23.6 20.  22.8 11.5 24.1
 19.5 42.3 13.5 23.9 20.5 36.2 25.2 12.7 16.4 20.1 11.7 17.8 15.6 29.1
 14.8 24.3 36.  20.8 23.7 18.5 15.4 20.1 32.9 21.8 19.3 21.  24.5 28.4
 37.9 23.  23.2 13.3 20.1 22.9 24.1 20.6 22.9 20.  16.1 24.4 18.2  8.8
 21.2 19.6 11.8 22.6 21.4 23.5 30.1 18.3 20.6 23.9 27.5 22.  16.  21.7
```

```
 20.7 16.6 10.8 28.7 12.1 46.  30.1 23.1 12.6 15.3 48.5 21.5 18.9 33.
 12.8 20.6 22.2 23.2 25.  18.9 23.3 14.4 11.8 18.  13.8 21.2  7.2 37.2
  8.3 19.8 20.7 17.9 30.5 22.4 14.9 13.6 28.6 15.  21.9 29.8 25.   7.2
 34.6 12.7 23.8 31.5 21.7  7.  20.6 30.8 50.  17.1 37.3 13.1 13.4 22.6
 17.8 15.6 27.5 19.9 20.  13.  23.  14.9 37.6 14.1 23.2 48.8 23.9 28.7
 20.4 50.  16.8 14.6]

Y Test :-

[30.7 22.  19.7 23.1 41.7 13.8 23.1 17.8  7.2 21.9  8.3 12.  18.2 33.2
 24.8 19.4 31.   8.4 19.3 29.9 13.6 22.3 15.6 13.9 18.5 27.5 33.4 50.
 13.4 25.  23.6 20.8 17.7 17.5 24.3 50.  50.  14.5  9.6 19.4 22.2 45.4
 29.1 17.  13.8 21.5 31.2 34.7 18.6 25.  21.1 11.9 23.  29.6 17.1 12.7
 50.  22.2 18.4 13.3 13.8 28.7 35.4 26.6 18.4  8.4 22.8 26.4 18.6 14.9
 19.6 14.6 21.4 32.  31.1 24.4 18.7 11.  31.6 23.4 13.4 18.2 19.9 18.4
 24.7 17.2 19.  18.5 19.1 21.  33.1 11.3 20.9 23.3 14.2 35.4 15.6 22.1
 15.  22.7 22.2 13.1 50.  22.  16.2 20.3 24.8 15.6 23.8 43.8 26.6 32.2
 22.8 20.3 21.2 13.5  8.5 27.9 39.8 22.9 14.   9.5 23.8 16.7 22.5 17.2
 10.9 20.6 17.1 33.1 21.9 21.2 24.1 29.  17.5 20.2  9.7 23.9  7.5 23.1
 19.4 24.7 25.  25.1 19.5 50.  33.3 11.9 22.2 24.  19.3 13.1]
```

In [40]:
```python
from sklearn.linear_model import LinearRegression
```

In [41]:
```python
reg = LinearRegression(normalize = True)
```

In [42]:
```python
X_train = np.array(X_train).reshape(-1,1)
```

In [43]:
```python
X_test = np.array(X_test).reshape(-1,1)
```

In [44]:
```python
reg.fit(np.array(X_train).reshape(-1,1), y_train)
```

```
/Users/ananyasamala/opt/anaconda3/envs/Ananya/lib/python3.9/site-package
s/sklearn/linear_model/_base.py:141: FutureWarning: 'normalize' was depre
cated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a pr
eprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression
())

If you wish to pass a sample_weight parameter, you need to pass it as a f
it parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

Out[44]:
```
LinearRegression(normalize=True)
```

In [45]:

```
reg.coef_
```

Out[45]:

```
array([-0.96949635])
```

In [47]:

```
reg.intercept_
```

Out[47]:

```
34.816093590661914
```

In [49]:

```
y_pred_train = reg.predict(X_train)
```

In [50]:

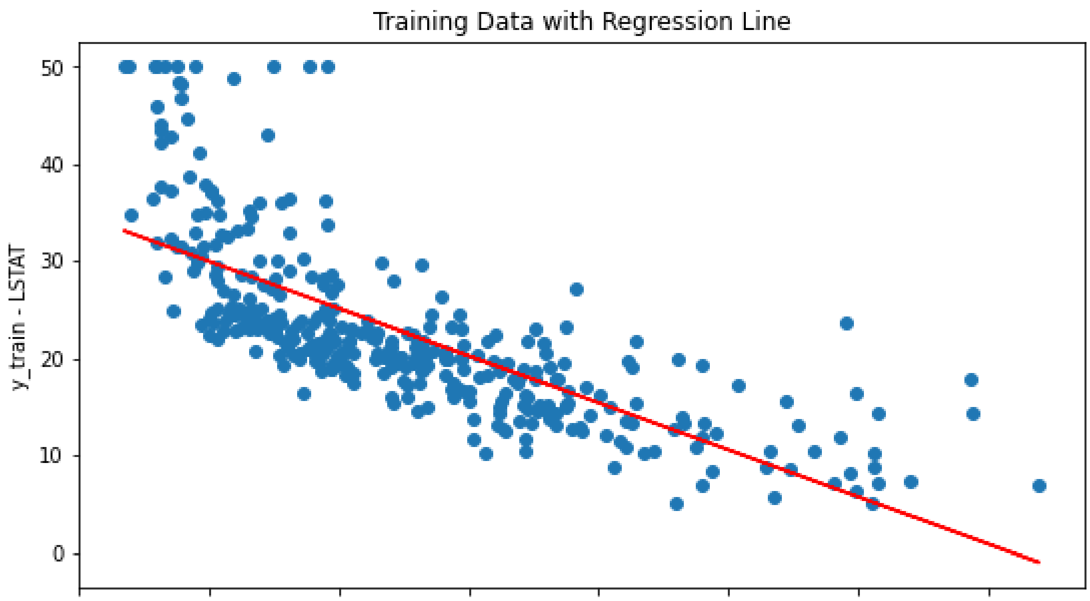
```
y_pred_test = reg.predict(X_test)
print(y_pred_test)
```

```
[20.47724251 28.53375721 21.77636762 21.89270719 32.4214376   1.10670535
 28.65009678  7.35995683 15.11592767 21.24314463 15.64915066 10.6659394
 27.01164794 28.81491116 28.93125072 19.50774616 23.9092596   1.83382761
 21.62124821 28.10717882 17.29729447 23.84139486 18.0341117  18.80970878
 24.68485669 15.6394557  31.96577431 31.22895708  8.81420137 28.79552123
 29.48386364 24.85936603 20.612972   20.59358207 23.52146106 31.75248511
 31.65553548 15.54250607 17.31668439 23.45359632 28.29138313 31.1707873
 27.8454148  20.15730871 -1.99568298 21.14619499 30.3467154  30.90902328
 24.51034734 27.26371699 22.21264098 12.18804868 24.40370274 31.39377146
 22.76525391 12.90547598 29.85227226 22.41623522 22.40654025 14.22399102
  7.86409494 29.28996437 30.36610532 28.4852824  20.24456338 12.76974649
 30.41458014 26.4105602  27.26371699 17.56875345 21.40795901  6.18686625
 23.82200493 28.33985794 29.93952693 24.18071858 18.24740089 13.95253204
 28.65009678 27.50609108 12.25591342 23.88017471 19.02299798 18.73214907
 24.99509552 14.24338095 18.18923111 16.91919089 20.2930382  22.474405
 30.10434131 11.93597963 26.29422064 28.1653486  19.59500083 30.15281613
  9.20199991 29.01850539 25.01448545 23.53115603 29.3093543  11.84872496
 31.93668942 20.88443098 23.66688552 20.98138061 28.30107809 18.67397929
 25.81916742 31.35499161 30.12373124 27.20554721 24.96601063 19.45927134
```

In [52]:

```python
plt.figure(figsize = (9,5))
plt.scatter(X_train, y_train)

plt.plot(X_train, y_pred_train, 'r')
plt.title('Training Data with Regression Line')
plt.xlabel('X_train - LSTAT')
plt.ylabel('y_train - LSTAT')
```

Out[52]: Text(0, 0.5, 'y_train - LSTAT')

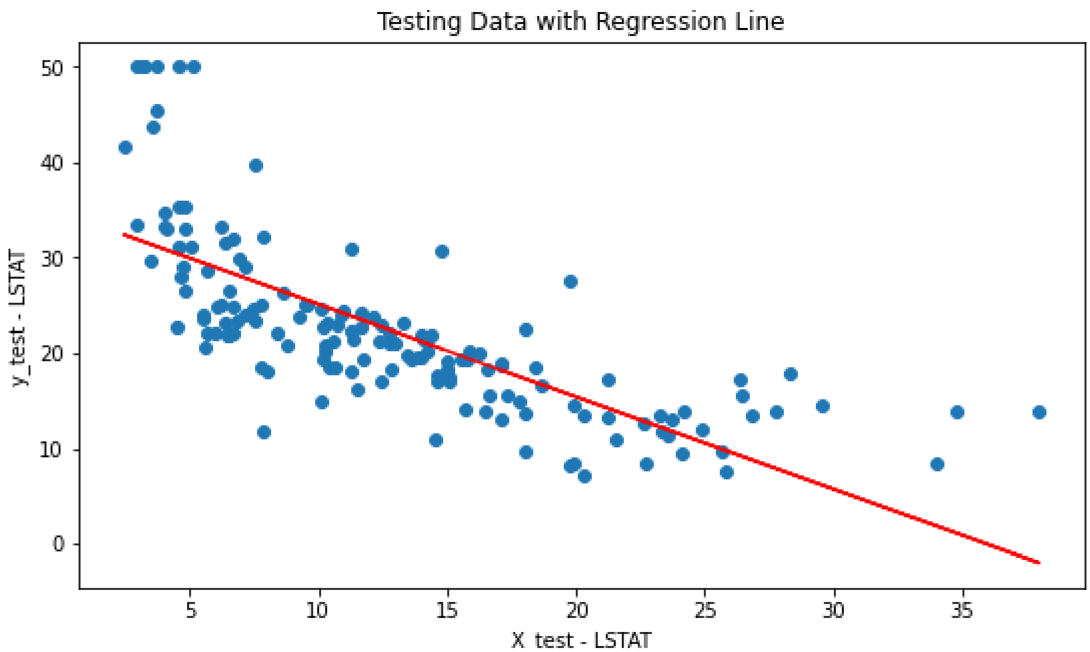


In [53]:

```
plt.figure(figsize = (9,5))
plt.scatter(X_test, y_test)

plt.plot(X_test, y_pred_test, 'r')
plt.title('Testing Data with Regression Line')
plt.xlabel('X_test - LSTAT')
plt.ylabel('y_test - LSTAT')
```

Out[53]:

```
Text(0, 0.5, 'y_test - LSTAT')
```



Do a residual analysis of the train data and the plot the histogram of the residuals and see whether it looks like normal distribution or not. One of the major assumptions of the linear regression model is the error terms are normally distributed.

In [54]:

```
from sklearn.metrics import mean_squared_error
```

In [55]:

```
mean_squared_error(y_test, y_pred_test)
```

Out[55]:

```
39.848251009505645
```

In [56]:

```
res = y_train - y_pred_train
```

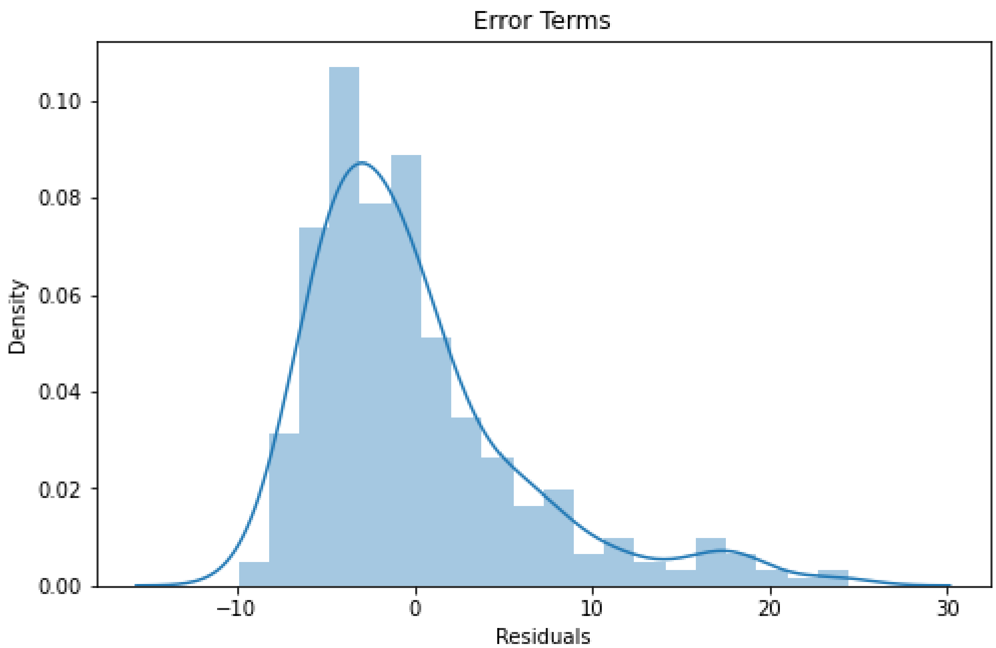
In [57]:

```
plt.figure(figsize = (8,5))
sns.distplot(res)
plt.title('Error Terms')
plt.xlabel('Residuals')
```

```
/Users/ananyasamala/opt/anaconda3/envs/Ananya/lib/python3.9/site-package
s/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecate
d function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[57]:

```
Text(0.5, 0, 'Residuals')
```



In [ ]:

```

```