Importing the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, classification_report
```

Import the dataset

```
df = pd.read_csv("indian_liver_patient.csv")
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminot
0	65	Female	0.7	0.1	187	
1	62	Male	10.9	5.5	699	
2	62	Male	7.3	4.1	490	
3	58	Male	1.0	0.4	182	
4	72	Male	3.9	2.0	195	

```
df.shape
```

```
(583, 11)
```

```
df.columns
```

```
Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

```
#there is 1 object data that needs to be converted
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
 5   Alamine_Aminotransferase    583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

```
df.isna().sum()
```

```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphotase          0
Alamine_Aminotransferase      0
Aspartate_Aminotransferase    0
Total_Protiens                0
Albumin                       0
Albumin_and_Globulin_Ratio    4
Dataset                       0
dtype: int64
```

```
df[df['Albumin_and_Globulin_Ratio'].isna()]
df.dropna(inplace=True)
```

```
df[df.duplicated()]
df.drop_duplicates(inplace=True)
df.reset_index(drop=True, inplace=True)
```

```
df.shape
```

```
(566, 11)
```

Descriptive Data Analysis

```
df.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amir
count	566.000000	566.000000	566.000000	566.000000	
mean	44.886926	3.338869	1.505830	292.567138	
std	16.274893	6.286728	2.841485	245.936559	
min	4.000000	0.400000	0.100000	63.000000	
25%	33.000000	0.800000	0.200000	176.000000	
50%	45.000000	1.000000	0.300000	208.000000	
75%	58.000000	2.600000	1.300000	298.000000	
max	90.000000	75.000000	19.700000	2110.000000	

```
df.describe(include ='object')
```

	Gender
count	566
unique	2
top	Male
freq	428

```
df.rename(columns={'Dataset':'target'},inplace=True)
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminot
0	65	Female	0.7	0.1	187	
1	62	Male	10.9	5.5	699	
2	62	Male	7.3	4.1	490	
3	58	Male	1.0	0.4	182	
4	72	Male	3.9	2.0	195	

```
# let's look on target variable - classes imbalanced?
df['target'].value_counts()
```

```
1    404
2    162
Name: target, dtype: int64
```

Encoding of categorical values using Label Encoder

```
from sklearn.preprocessing import LabelEncoder
cols = ['Gender']
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminot
0	65	0	0.7	0.1	187	
1	62	1	10.9	5.5	699	
2	62	1	7.3	4.1	490	
3	58	1	1.0	0.4	182	
4	72	1	3.9	2.0	195	

Data Preprocessing

Extract the independent and dependent variables

```
X = df.drop('target',axis=1)
y = df['target'] # allocating the output to dependent variable which we want to predict
```

```
print('Shape of Feature-set : ', X.shape)
print('Shape of Target-set : ', y.shape)
```

```
Shape of Feature-set :  (566, 10)
Shape of Target-set :  (566,)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25,random_state = 23)
```

```
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(424, 10) (424,)
(142, 10) (142,)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
print(X_train.shape, X_test.shape)
```

```
(424, 10) (142, 10)
```

Decision Tree Classifier

```
classifier = DecisionTreeClassifier(criterion = 'entropy',random_state = 23)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
[[83 28]
 [23  8]]
Accuracy of our model is equal 64.08 %.
```

```
from sklearn import tree
text_representation = tree.export_text(classifier)
print(text_representation)
```

```
|--- feature_3 <= -0.11
|   |--- feature_4 <= -0.34
|   |   |--- feature_6 <= 0.07
|   |   |   |--- feature_4 <= -0.75
|   |   |   |   |--- class: 1
|   |   |   |--- feature_4 >  -0.75
|   |   |   |   |--- feature_8 <= -1.51
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_8 >  -1.51
|   |   |   |   |   |--- feature_2 <= -0.25
|   |   |   |   |   |   |--- feature_8 <= -1.25
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- feature_8 >  -1.25
|   |   |   |   |   |   |   |--- feature_6 <= -0.10
|   |   |   |   |   |   |   |   |--- feature_5 <= -0.37
|   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |--- feature_5 >  -0.37
```

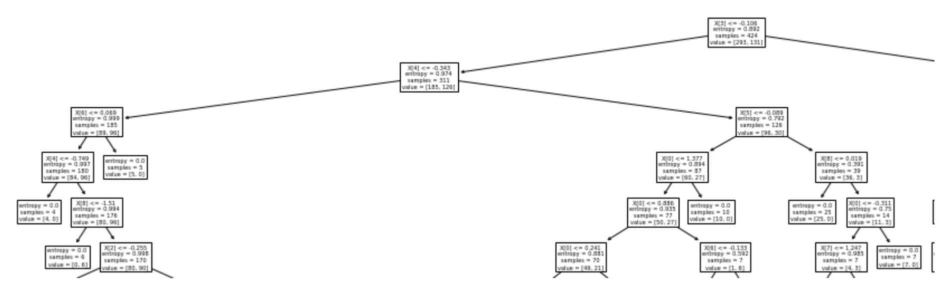
```
|   |   |   |   |   |   |   |   |   |--- feature_6 <= -0.29
|   |   |   |   |   |   |   |   |   |   |--- feature_4 <= -0.43
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |   |--- feature_4 >  -0.43
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |--- feature_6 >  -0.29
|   |   |   |   |   |   |   |   |   |   |--- feature_9 <= 0.61
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 16
|   |   |   |   |   |   |   |   |   |   |--- feature_9 >  0.61
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 6
|   |   |   |   |   |   |   |--- feature_6 >  -0.10
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_2 >  -0.25
|   |   |   |   |   |   |--- feature_7 <= -0.58
|   |   |   |   |   |   |   |--- feature_5 <= -0.21
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_5 >  -0.21
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_7 >  -0.58
|   |   |   |   |   |   |   |--- feature_7 <= 1.53
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_7 >  1.53
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |--- feature_6 >  0.07
|   |   |   |--- class: 1
|   |--- feature_4 >  -0.34
|   |   |--- feature_5 <= -0.09
|   |   |   |--- feature_0 <= 1.38
|   |   |   |   |--- feature_0 <= 0.89
|   |   |   |   |   |--- feature_0 <= 0.24
|   |   |   |   |   |   |--- feature_6 <= -0.09
|   |   |   |   |   |   |   |--- feature_7 <= -1.00
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |--- feature_7 >  -1.00
|   |   |   |   |   |   |   |   |--- feature_0 <= -1.29
|   |   |   |   |   |   |   |   |   |--- feature_4 <= -0.09
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_4 >  -0.09
|   |   |   |   |   |   |   |   |   |   |--- feature_7 <= 0.83
|   |   |   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |   |   |--- feature 7 >  0.83
```

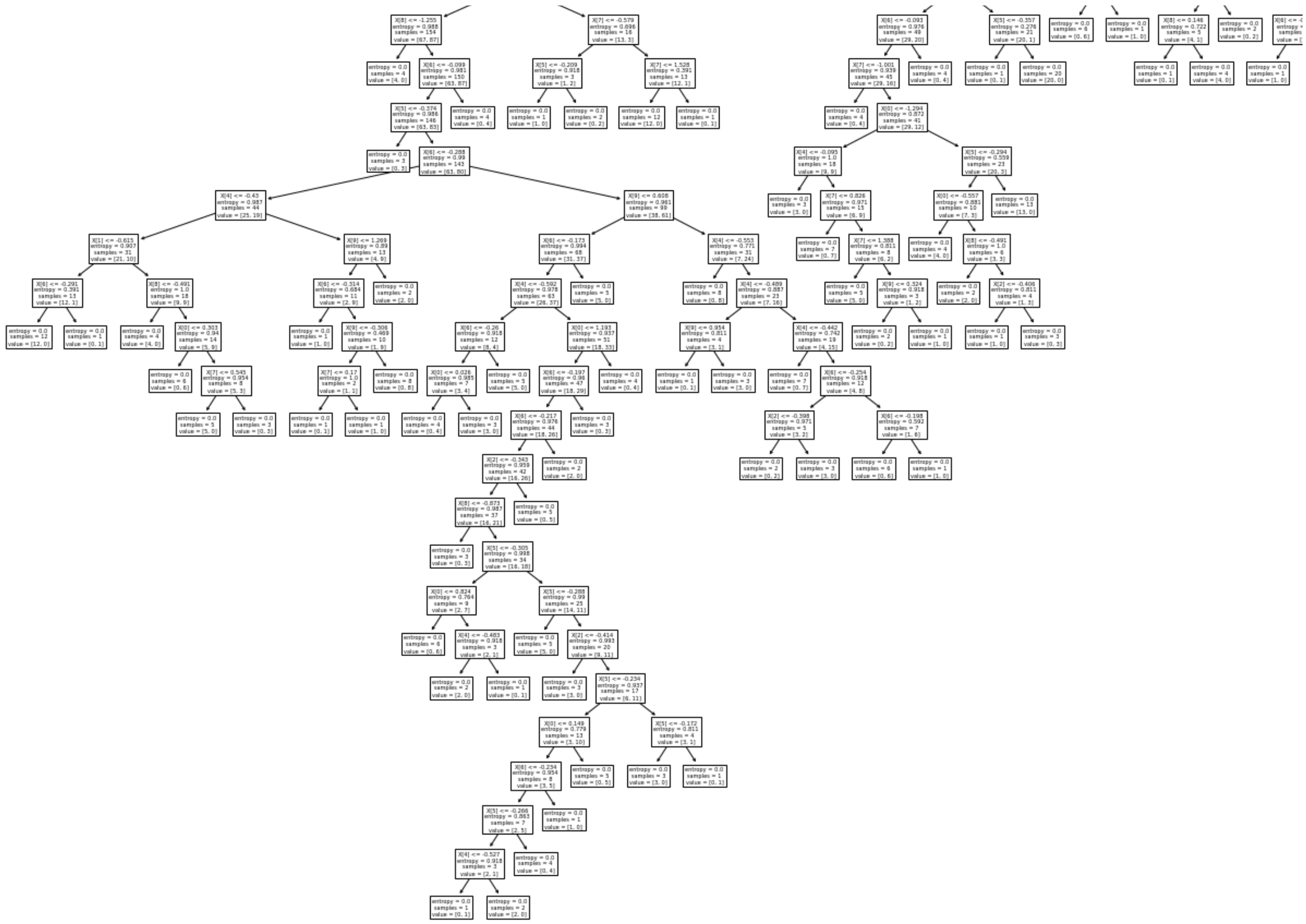
```
fig = plt.figure(figsize=(25,20))
tree.plot_tree(classifier)
```

```
[Text(0.7291666666666666, 0.9814814814814815, 'X[3] <= -0.106\nentropy = 0.892\nsamples
 Text(0.5324074074074074, 0.9444444444444444, 'X[4] <= -0.343\nentropy = 0.974\nsamples
 Text(0.3194444444444444, 0.9074074074074074, 'X[6] <= 0.069\nentropy = 0.999\nsamples =
 Text(0.30092592592592593, 0.8703703703703703, 'X[4] <= -0.749\nentropy = 0.997\nsamples
 Text(0.2824074074074074, 0.8333333333333334, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0
 Text(0.3194444444444444, 0.8333333333333334, 'X[8] <= -1.51\nentropy = 0.994\nsamples =
 Text(0.30092592592592593, 0.7962962962962963, 'entropy = 0.0\nsamples = 6\nvalue = [0,
 Text(0.33796296296296297, 0.7962962962962963, 'X[2] <= -0.255\nentropy = 0.998\nsamples
 Text(0.27314814814814814, 0.7592592592592593, 'X[8] <= -1.255\nentropy = 0.988\nsamples
 Text(0.25462962962962965, 0.7222222222222222, 'entropy = 0.0\nsamples = 4\nvalue = [4,
 Text(0.2916666666666667, 0.7222222222222222, 'X[6] <= -0.099\nentropy = 0.981\nsamples
 Text(0.27314814814814814, 0.6851851851851852, 'X[5] <= -0.374\nentropy = 0.986\nsamples
 Text(0.25462962962962965, 0.6481481481481481, 'entropy = 0.0\nsamples = 3\nvalue = [0,
 Text(0.2916666666666667, 0.6481481481481481, 'X[6] <= -0.288\nentropy = 0.99\nsamples =
 Text(0.1574074074074074, 0.6111111111111112, 'X[4] <= -0.43\nentropy = 0.987\nsamples =
 Text(0.07407407407407407, 0.5740740740740741, 'X[1] <= -0.615\nentropy = 0.907\nsamples
 Text(0.037037037037037035, 0.5370370370370371, 'X[6] <= -0.291\nentropy = 0.391\nsample
 Text(0.018518518518518517, 0.5, 'entropy = 0.0\nsamples = 12\nvalue = [12, 0]'),
 Text(0.05555555555555555, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.1111111111111111, 0.5370370370370371, 'X[8] <= -0.491\nentropy = 1.0\nsamples =
 Text(0.09259259259259259, 0.5, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.12962962962962962, 0.5, 'X[0] <= 0.303\nentropy = 0.94\nsamples = 14\nvalue = [5
 Text(0.1111111111111111, 0.46296296296296297, 'entropy = 0.0\nsamples = 6\nvalue = [0,
 Text(0.14814814814814814, 0.46296296296296297, 'X[7] <= 0.545\nentropy = 0.954\nsamples
 Text(0.12962962962962962, 0.42592592592592593, 'entropy = 0.0\nsamples = 5\nvalue = [5,
 Text(0.16666666666666666, 0.42592592592592593, 'entropy = 0.0\nsamples = 3\nvalue = [0,
 Text(0.24074074074074073, 0.5740740740740741, 'X[9] <= 1.269\nentropy = 0.89\nsamples =
 Text(0.2222222222222222, 0.5370370370370371, 'X[6] <= -0.314\nentropy = 0.684\nsamples
 Text(0.2037037037037037, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.24074074074074073, 0.5, 'X[9] <= -0.306\nentropy = 0.469\nsamples = 10\nvalue =
 Text(0.2222222222222222, 0.46296296296296297, 'X[7] <= 0.17\nentropy = 1.0\nsamples = 2
 Text(0.2037037037037037, 0.42592592592592593, 'entropy = 0.0\nsamples = 1\nvalue = [0,
 Text(0.24074074074074073, 0.42592592592592593, 'entropy = 0.0\nsamples = 1\nvalue = [1,
 Text(0.25925925925925924, 0.46296296296296297, 'entropy = 0.0\nsamples = 8\nvalue = [0,
 Text(0.25925925925925924, 0.5370370370370371, 'entropy = 0.0\nsamples = 2\nvalue = [2,
 Text(0.42592592592592593, 0.6111111111111112, 'X[9] <= 0.608\nentropy = 0.961\nsamples
 Text(0.37037037037037035, 0.5740740740740741, 'X[6] <= -0.173\nentropy = 0.994\nsamples
 Text(0.35185185185185186, 0.5370370370370371, 'X[4] <= -0.592\nentropy = 0.978\nsamples
 Text(0.3148148148148148, 0.5, 'X[6] <= -0.26\nentropy = 0.918\nsamples = 12\nvalue = [8
 Text(0.2962962962962963, 0.46296296296296297, 'X[0] <= 0.026\nentropy = 0.985\nsamples
 Text(0.2777777777777778, 0.42592592592592593, 'entropy = 0.0\nsamples = 4\nvalue = [0,
 Text(0.3148148148148148, 0.42592592592592593, 'entropy = 0.0\nsamples = 3\nvalue = [3,
 Text(0.3333333333333333, 0.46296296296296297, 'entropy = 0.0\nsamples = 5\nvalue = [5,
 Text(0.3888888888888889, 0.5, 'X[0] <= 1.193\nentropy = 0.937\nsamples = 51\nvalue = [1
 Text(0.37037037037037035, 0.46296296296296297, 'X[6] <= -0.197\nentropy = 0.96\nsamples
 Text(0.35185185185185186, 0.42592592592592593, 'X[6] <= -0.217\nentropy = 0.976\nsample
 Text(0.3333333333333333, 0.3888888888888889, 'X[2] <= -0.343\nentropy = 0.959\nsamples
 Text(0.3148148148148148, 0.35185185185185186, 'X[8] <= -0.873\nentropy = 0.987\nsamples
 Text(0.2962962962962963, 0.3148148148148148, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3
 Text(0.3333333333333333, 0.3148148148148148, 'X[5] <= -0.305\nentropy = 0.998\nsamples
 Text(0.2962962962962963, 0.2777777777777778, 'X[0] <= 0.824\nentropy = 0.764\nsamples =
 Text(0.2777777777777778, 0.24074074074074073, 'entropy = 0.0\nsamples = 6\nvalue = [0,
 Text(0.3148148148148148, 0.24074074074074073, 'X[4] <= -0.483\nentropy = 0.918\nsamples
 Text(0.2962962962962963, 0.2037037037037037, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0
 Text(0.3333333333333333, 0.2037037037037037, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1
 Text(0.37037037037037035, 0.2777777777777778, 'X[5] <= -0.288\nentropy = 0.99\nsamples
```

```
Text(0.35185185185185186, 0.24074074074074073, 'entropy = 0.0\nsamples = 5\nvalue = [5,
Text(0.3888888888888889, 0.24074074074074073, 'X[2] <= -0.414\nentropy = 0.993\nsamples
Text(0.37037037037037035, 0.2037037037037037, 'entropy = 0.0\nsamples = 3\nvalue = [3,
Text(0.4074074074074074, 0.2037037037037037, 'X[5] <= -0.234\nentropy = 0.937\nsamples
Text(0.37037037037037035, 0.16666666666666666, 'X[0] <= 0.149\nentropy = 0.779\nsamples
Text(0.35185185185185186, 0.12962962962962962, 'X[6] <= -0.234\nentropy = 0.954\nsample
Text(0.3333333333333333, 0.09259259259259259, 'X[5] <= -0.266\nentropy = 0.863\nsamples
Text(0.3148148148148148, 0.05555555555555555, 'X[4] <= -0.527\nentropy = 0.918\nsamples
Text(0.2962962962962963, 0.018518518518518517, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.3333333333333333, 0.018518518518518517, 'entropy = 0.0\nsamples = 2\nvalue = [2,
Text(0.35185185185185186, 0.05555555555555555, 'entropy = 0.0\nsamples = 4\nvalue = [0,
Text(0.37037037037037035, 0.09259259259259259, 'entropy = 0.0\nsamples = 1\nvalue = [1,
Text(0.3888888888888889, 0.12962962962962962, 'entropy = 0.0\nsamples = 5\nvalue = [0,
Text(0.4444444444444444, 0.16666666666666666, 'X[5] <= -0.172\nentropy = 0.811\nsamples
Text(0.42592592592592593, 0.12962962962962962, 'entropy = 0.0\nsamples = 3\nvalue = [3,
Text(0.46296296296296297, 0.12962962962962962, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.35185185185185186, 0.35185185185185186, 'entropy = 0.0\nsamples = 5\nvalue = [0,
Text(0.37037037037037035, 0.3888888888888889, 'entropy = 0.0\nsamples = 2\nvalue = [2,
Text(0.3888888888888889, 0.42592592592592593, 'entropy = 0.0\nsamples = 3\nvalue = [0,
Text(0.4074074074074074, 0.46296296296296297, 'entropy = 0.0\nsamples = 4\nvalue = [0,
Text(0.3888888888888889, 0.5370370370370371, 'entropy = 0.0\nsamples = 5\nvalue = [5, 0
Text(0.48148148148148145, 0.5740740740740741, 'X[4] <= -0.553\nentropy = 0.771\nsamples
Text(0.46296296296296297, 0.5370370370370371, 'entropy = 0.0\nsamples = 8\nvalue = [0,
Text(0.5, 0.5370370370370371, 'X[4] <= -0.489\nentropy = 0.887\nsamples = 23\nvalue =
Text(0.46296296296296297, 0.5, 'X[9] <= 0.954\nentropy = 0.811\nsamples = 4\nvalue = [3
Text(0.4444444444444444, 0.46296296296296297, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.48148148148148145, 0.46296296296296297, 'entropy = 0.0\nsamples = 3\nvalue = [3,
Text(0.5370370370370371, 0.5, 'X[4] <= -0.442\nentropy = 0.742\nsamples = 19\nvalue =
Text(0.5185185185185185, 0.46296296296296297, 'entropy = 0.0\nsamples = 7\nvalue = [0,
Text(0.5555555555555556, 0.46296296296296297, 'X[6] <= -0.254\nentropy = 0.918\nsamples
Text(0.5185185185185185, 0.42592592592592593, 'X[2] <= -0.398\nentropy = 0.971\nsamples
Text(0.5, 0.3888888888888889, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.5370370370370371, 0.3888888888888889, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0
Text(0.5925925925925926, 0.42592592592592593, 'X[6] <= -0.198\nentropy = 0.592\nsamples
Text(0.5740740740740741, 0.3888888888888889, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6
Text(0.6111111111111112, 0.3888888888888889, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0
Text(0.3101851851851852, 0.6851851851851852, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4
Text(0.4027777777777778, 0.7592592592592593, 'X[7] <= -0.579\nentropy = 0.696\nsamples
Text(0.36574074074074076, 0.7222222222222222, 'X[5] <= -0.209\nentropy = 0.918\nsamples
Text(0.3472222222222222, 0.6851851851851852, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0
Text(0.38425925925925924, 0.6851851851851852, 'entropy = 0.0\nsamples = 2\nvalue = [0,
Text(0.4398148148148148, 0.7222222222222222, 'X[7] <= 1.528\nentropy = 0.391\nsamples =
Text(0.4212962962962963, 0.6851851851851852, 'entropy = 0.0\nsamples = 12\nvalue = [12,
Text(0.4583333333333333, 0.6851851851851852, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1
Text(0.33796296296296297, 0.8703703703703703, 'entropy = 0.0\nsamples = 5\nvalue = [5,
Text(0.7453703703703703, 0.9074074074074074, 'X[5] <= -0.089\nentropy = 0.792\nsamples
Text(0.6944444444444444, 0.8703703703703703, 'X[0] <= 1.377\nentropy = 0.894\nsamples =
Text(0.6759259259259259, 0.8333333333333334, 'X[0] <= 0.886\nentropy = 0.935\nsamples =
Text(0.6296296296296297, 0.7962962962962963, 'X[0] <= 0.241\nentropy = 0.881\nsamples =
Text(0.5925925925925926, 0.7592592592592593, 'X[6] <= -0.093\nentropy = 0.976\nsamples
Text(0.5740740740740741, 0.7222222222222222, 'X[7] <= -1.001\nentropy = 0.939\nsamples
Text(0.5555555555555556, 0.6851851851851852, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4
Text(0.5925925925925926, 0.6851851851851852, 'X[0] <= -1.294\nentropy = 0.872\nsamples
Text(0.5370370370370371, 0.6481481481481481, 'X[4] <= -0.095\nentropy = 1.0\nsamples =
Text(0.5185185185185185, 0.6111111111111112, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0
Text(0.5555555555555556, 0.6111111111111112, 'X[7] <= 0.826\nentropy = 0.971\nsamples =
Text(0.5370370370370371, 0.5740740740740741, 'entropy = 0.0\nsamples = 7\nvalue = [0, 1
```

```
Text(0.5370370370370371, 0.5740740740740741, 'entropy = 0.0\nsamples = 7\nvalue = [0,
Text(0.5740740740740741, 0.5740740740740741, 'X[7] <= 1.388\nentropy = 0.811\nsamples =
Text(0.5555555555555556, 0.5370370370370371, 'entropy = 0.0\nsamples = 5\nvalue = [5,
Text(0.5925925925925926, 0.5370370370370371, 'X[9] <= 0.324\nentropy = 0.918\nsamples =
Text(0.5740740740740741, 0.5, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.6111111111111112, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.6481481481481481, 0.6481481481481481, 'X[5] <= -0.294\nentropy = 0.559\nsamples
Text(0.6296296296296297, 0.6111111111111112, 'X[0] <= -0.557\nentropy = 0.881\nsamples
Text(0.6111111111111112, 0.5740740740740741, 'entropy = 0.0\nsamples = 4\nvalue = [4,
Text(0.6481481481481481, 0.5740740740740741, 'X[8] <= -0.491\nentropy = 1.0\nsamples =
Text(0.6296296296296297, 0.5370370370370371, 'entropy = 0.0\nsamples = 2\nvalue = [2,
Text(0.6666666666666666, 0.5370370370370371, 'X[2] <= -0.406\nentropy = 0.811\nsamples
Text(0.6481481481481481, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.6851851851851852, 0.5, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.6666666666666666, 0.6111111111111112, 'entropy = 0.0\nsamples = 13\nvalue = [13,
Text(0.6111111111111112, 0.7222222222222222, 'entropy = 0.0\nsamples = 4\nvalue = [0,
Text(0.6666666666666666, 0.7592592592592593, 'X[5] <= -0.357\nentropy = 0.276\nsamples
Text(0.6481481481481481, 0.7222222222222222, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.6851851851851852, 0.7222222222222222, 'entropy = 0.0\nsamples = 20\nvalue = [20,
Text(0.7222222222222222, 0.7962962962962963, 'X[6] <= -0.133\nentropy = 0.592\nsamples
Text(0.7037037037037037, 0.7592592592592593, 'entropy = 0.0\nsamples = 6\nvalue = [0,
Text(0.7407407407407407, 0.7592592592592593, 'entropy = 0.0\nsamples = 1\nvalue = [1,
Text(0.7129629629629629, 0.8333333333333334, 'entropy = 0.0\nsamples = 10\nvalue = [10,
Text(0.7962962962962963, 0.8703703703703703, 'X[8] <= 0.019\nentropy = 0.391\nsamples =
Text(0.7777777777777778, 0.8333333333333334, 'entropy = 0.0\nsamples = 25\nvalue = [25,
Text(0.8148148148148148, 0.8333333333333334, 'X[0] <= -0.311\nentropy = 0.75\nsamples =
Text(0.7962962962962963, 0.7962962962962963, 'X[7] <= 1.247\nentropy = 0.985\nsamples =
Text(0.7777777777777778, 0.7592592592592593, 'X[8] <= 0.146\nentropy = 0.722\nsamples =
Text(0.7592592592592593, 0.7222222222222222, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.7962962962962963, 0.7222222222222222, 'entropy = 0.0\nsamples = 4\nvalue = [4,
Text(0.8148148148148148, 0.7592592592592593, 'entropy = 0.0\nsamples = 2\nvalue = [0,
Text(0.8333333333333334, 0.7962962962962963, 'entropy = 0.0\nsamples = 7\nvalue = [7,
Text(0.9259259259259259, 0.9444444444444444, 'X[8] <= 1.293\nentropy = 0.261\nsamples =
Text(0.9074074074074074, 0.9074074074074074, 'X[7] <= -0.298\nentropy = 0.222\nsamples
Text(0.8888888888888888, 0.8703703703703703, 'X[8] <= -1.127\nentropy = 0.42\nsamples =
Text(0.8703703703703703, 0.8333333333333334, 'entropy = 0.0\nsamples = 25\nvalue = [25,
Text(0.9074074074074074, 0.8333333333333334, 'X[4] <= 0.364\nentropy = 0.684\nsamples =
Text(0.8703703703703703, 0.7962962962962963, 'X[7] <= -1.141\nentropy = 0.323\nsamples
Text(0.8518518518518519, 0.7592592592592593, 'X[6] <= -0.108\nentropy = 1.0\nsamples =
Text(0.8333333333333334, 0.7222222222222222, 'entropy = 0.0\nsamples = 1\nvalue = [1,
Text(0.8703703703703703, 0.7222222222222222, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.8888888888888888, 0.7592592592592593, 'entropy = 0.0\nsamples = 15\nvalue = [15,
Text(0.9444444444444444, 0.7962962962962963, 'X[6] <= -0.173\nentropy = 0.971\nsamples
Text(0.9259259259259259, 0.7592592592592593, 'entropy = 0.0\nsamples = 2\nvalue = [0,
Text(0.9629629629629629, 0.7592592592592593, 'X[7] <= -0.579\nentropy = 0.918\nsamples
Text(0.9444444444444444, 0.7222222222222222, 'entropy = 0.0\nsamples = 1\nvalue = [0,
Text(0.9814814814814815, 0.7222222222222222, 'entropy = 0.0\nsamples = 2\nvalue = [2,
Text(0.9259259259259259, 0.8703703703703703, 'entropy = 0.0\nsamples = 65\nvalue = [65,
Text(0.9444444444444444, 0.9074074074074074, 'entropy = 0.0\nsamples = 1\nvalue = [0,
```





```
from sklearn.tree import DecisionTreeClassifier
Live_Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
Live_Tree
Live_Tree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
predTree = Live_Tree.predict(X_test)
print (predTree [0:5])
```

```
print (y_test [0:5])
```

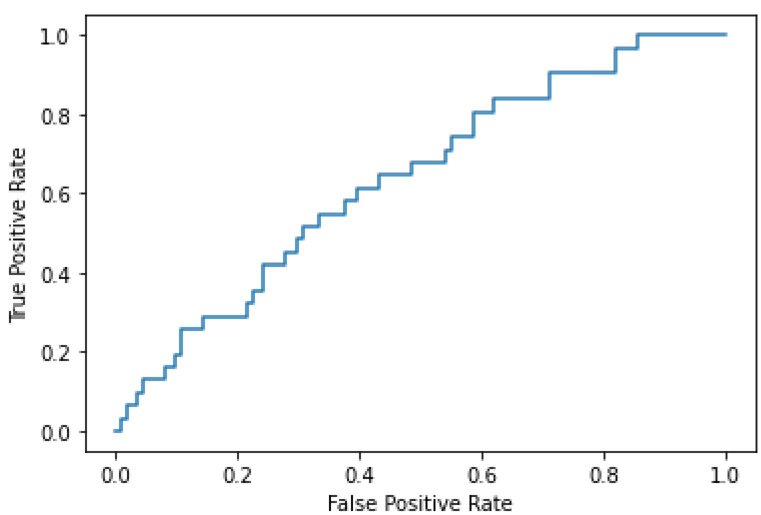
```
[1 1 2 1 2]
486    1
158    1
452    1
397    2
485    1
Name: target, dtype: int64
```

```
from sklearn import metrics
print("The accuracy of (Loan_ tree) DecisionTrees's {:.2f} ".format(metrics.accuracy_score(y_
print('The jaccard_score of the (Loan_ tree) DecisionTrees classifier on train data is {:.2f}
print('The F1-score of the (Loan_ tree) DecisionTrees classifier on train data is {:.2f}'.for
```

```
The accuracy of (Loan_ tree) DecisionTrees's 0.63
The jaccard_score of the (Loan_ tree) DecisionTrees classifier on train data is 0.58
The F1-score of the (Loan_ tree) DecisionTrees classifier on train data is 0.66
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay

y_score = clf.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_score, pos_label=clf.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
  "X does not have valid feature names, but"
```



Splitting the dataset into the Training and Testing sets

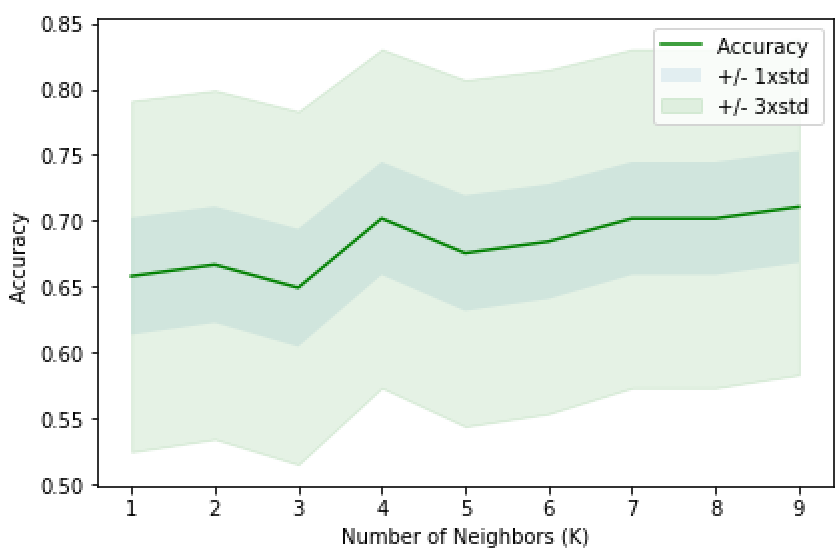
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
```

```
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (452, 10) (452,)
Test set: (114, 10) (114,)
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):
    knn1 = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=knn1.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)

    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,color=
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```




```
The best accuracy was with 0.7105263157894737 with k= 9
```

```
classifier = KNeighborsClassifier(n_neighbors=10,p=2,metric='minkowski')
```

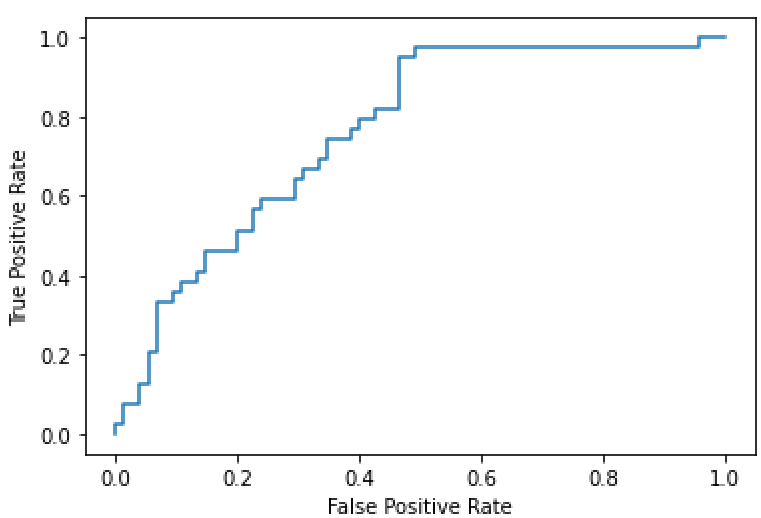
```
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
[[71  4]
 [34  5]]
Accuracy of our model is equal 66.67 %.
```

```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
knn1= KNeighborsClassifier(n_neighbors = 7).fit(X_train,y_train)
print('The jaccard_score of the KNN for k = 7 classifier on train data is {:.2f}'.format(jacc
print('The F1-score of the KNN for k = 7 classifier on train data is {:.2f}'.format(f1_score(
```

```
The jaccard_score of the KNN for k = 7 classifier on train data is 0.68
The F1-score of the KNN for k = 7 classifier on train data is 0.67
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay

y_score = clf.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_score, pos_label=clf.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```



SVM Classifier

```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

```
SVC()
```

```
yhat = clf.predict(X_test)
yhat [0:5]
```

```
array([1, 1, 1, 1, 1])
```

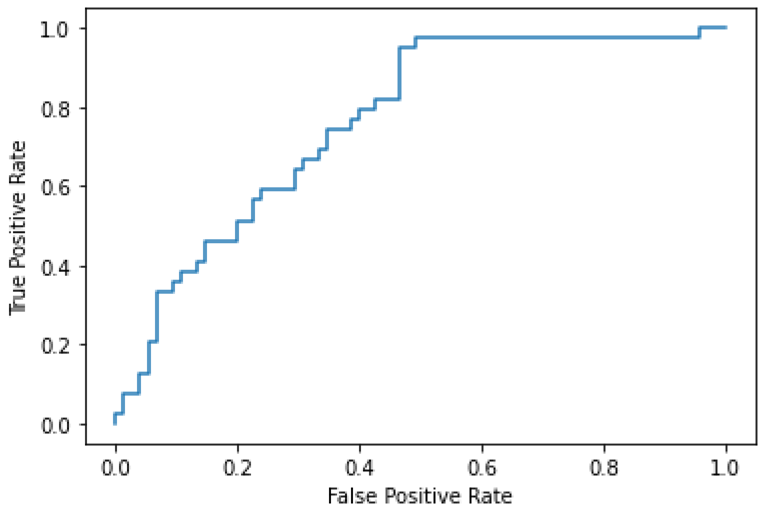
```
svm = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

print('The accuracy of the svm classifier on training data is {:.2f} out of 1'.format(svm.sco
print('The accuracy of the svm classifier on test data is {:.2f} out of 1'.format(svm.score(X
print('The jaccard_score of the SVM classifier on train data is {:.2f}'.format(jaccard_score(
print('The F1-score of the SVM classifier on train data is {:.2f}'.format(f1_score(y_test,yha
```

```
The accuracy of the svm classifier on training data is 0.99 out of 1
The accuracy of the svm classifier on test data is 0.66 out of 1
The jaccard_score of the SVM classifier on train data is 0.66
The F1-score of the SVM classifier on train data is 0.52
```

```
cm = confusion_matrix(y_test,y_pred)
print(cm)

accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
[[75  0]
 [39  0]]
Accuracy of our model is equal 65.79 %.
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay

y_score = clf.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_score, pos_label=clf.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```



1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate