Importing Libraries

```
# Importing data processing and Linear Algebra libraries
import pandas as pd
import numpy as np

# Importing data visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading our Dataset

```
# Readin data from CSV file and loading into df varaible
df = pd.read_csv('/content/heart.csv')
df.head(10)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerci
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	
5	39	M	NAP	120	339	0	Normal	170	
6	45	F	ATA	130	237	0	Normal	170	
7	54	M	ATA	110	208	0	Normal	142	
8	37	M	ASY	140	207	0	Normal	130	
9	48	F	ATA	120	284	0	Normal	120	

Preprocessing the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
```

```
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

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

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG
count	746.000000	746	746	746.000000	746.000000	746.000000	746
unique	NaN	2	4	NaN	NaN	NaN	3
top	NaN	M	ASY	NaN	NaN	NaN	Normal
freq	NaN	564	370	NaN	NaN	NaN	445
mean	52.882038	NaN	NaN	133.022788	244.635389	0.167560	NaN
std	9.505888	NaN	NaN	17.282750	59.153524	0.373726	NaN
min	28.000000	NaN	NaN	92.000000	85.000000	0.000000	NaN
25%	46.000000	NaN	NaN	120.000000	207.250000	0.000000	NaN
50%	54.000000	NaN	NaN	130.000000	237.000000	0.000000	NaN
75%	59.000000	NaN	NaN	140.000000	275.000000	0.000000	NaN
max	77.000000	NaN	NaN	200.000000	603.000000	1.000000	NaN

```
# Finding any missing values
missing_data = df.isnull().sum()
missing_data
```

```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
```

```
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

```
# Finding the count of unique values in each category
df.nunique()
```

```
Age                50
Sex                 2
ChestPainType       4
RestingBP          67
Cholesterol       222
FastingBS           2
RestingECG          3
MaxHR             119
ExerciseAngina      2
Oldpeak            53
ST_Slope            3
HeartDisease        2
dtype: int64
```

```
# categorical data
catg_lst = df.select_dtypes(include='object').columns
# numerical data
num_lst = df.select_dtypes(include=['int64','float64']).columns
```

```
#IQR for checking outliers
low_range = df[num_lst].quantile(0.25) #Q1
high_range = df[num_lst].quantile(0.75) #Q3
low_range
```

```
Age              47.00
RestingBP       120.00
Cholesterol     173.25
FastingBS         0.00
MaxHR           120.00
Oldpeak           0.00
HeartDisease      0.00
Name: 0.25, dtype: float64
```

```
high_range
```

```
Age              60.0
RestingBP       140.0
Cholesterol     267.0
FastingBS         0.0
MaxHR           156.0
Oldpeak           1.5
HeartDisease      1.0
Name: 0.75, dtype: float64
```

```
#
iqr = high_range - low_range
iqr
```

```
Age             13.00
RestingBP       20.00
Cholesterol     93.75
FastingBS        0.00
MaxHR           36.00
Oldpeak          1.50
HeartDisease     1.00
dtype: float64
```

```
lower_limit = low_range - 1.5*iqr
```

```
lower_limit
```

```
Age             27.500
RestingBP       90.000
Cholesterol     32.625
FastingBS        0.000
MaxHR           66.000
Oldpeak         -2.250
HeartDisease    -1.500
dtype: float64
```

```
upper_limit = high_range + 1.5*iqr
```

```
upper_limit
```

```
Age              79.500
RestingBP       170.000
Cholesterol     407.625
FastingBS         0.000
MaxHR           210.000
Oldpeak           3.750
HeartDisease      2.500
dtype: float64
```

Skewness of the Data

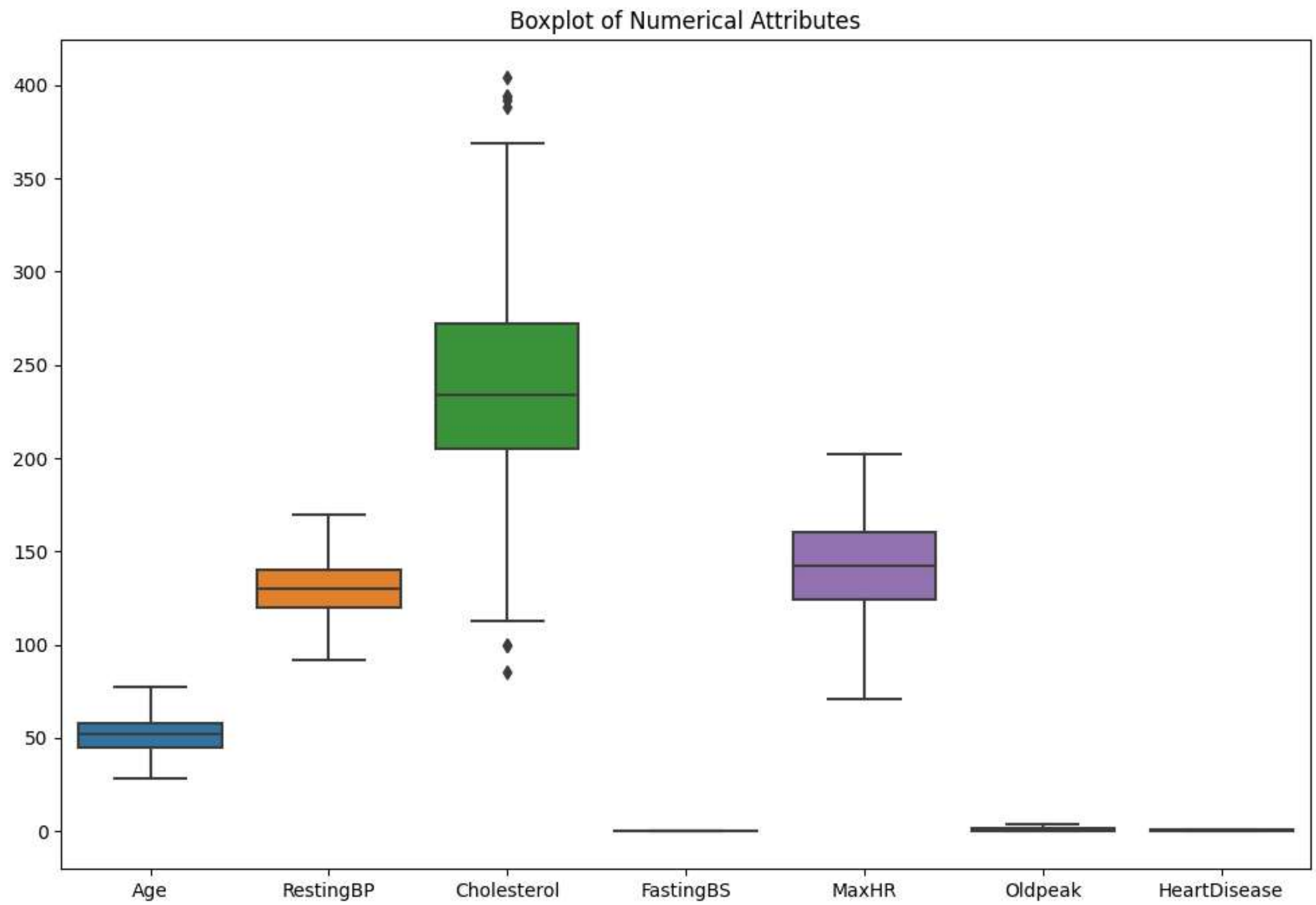
```
df_cleaned = df[~((df[num_lst] < lower_limit) | (df[num_lst] > upper_limit)).any(axis=1)]

skewness = df_cleaned.skew()
```

```
<ipython-input-19-c422d9007445>:3: FutureWarning: The default value of numeric_only in D
  skewness = df_cleaned.skew()
```

```
skewness

     Age             -0.008090
     RestingBP        0.221929
     Cholesterol      0.283054
     FastingBS        0.000000
     MaxHR           -0.177518
     Oldpeak          0.953711
     HeartDisease     0.296454
     dtype: float64
```

Using Boxplot to check outliers

```
num_lst = df_cleaned.select_dtypes(include=['int64', 'float64'])
plt.figure(figsize=(12, 8))

sns.boxplot(data=num_lst)

plt.title("Boxplot of Numerical Attributes")
plt.xticks(rotation=0)
plt.show()
```



Removing OUtliers

```
#to remove the outliers
(df[['Cholesterol','RestingBP']]==0).sum()
```

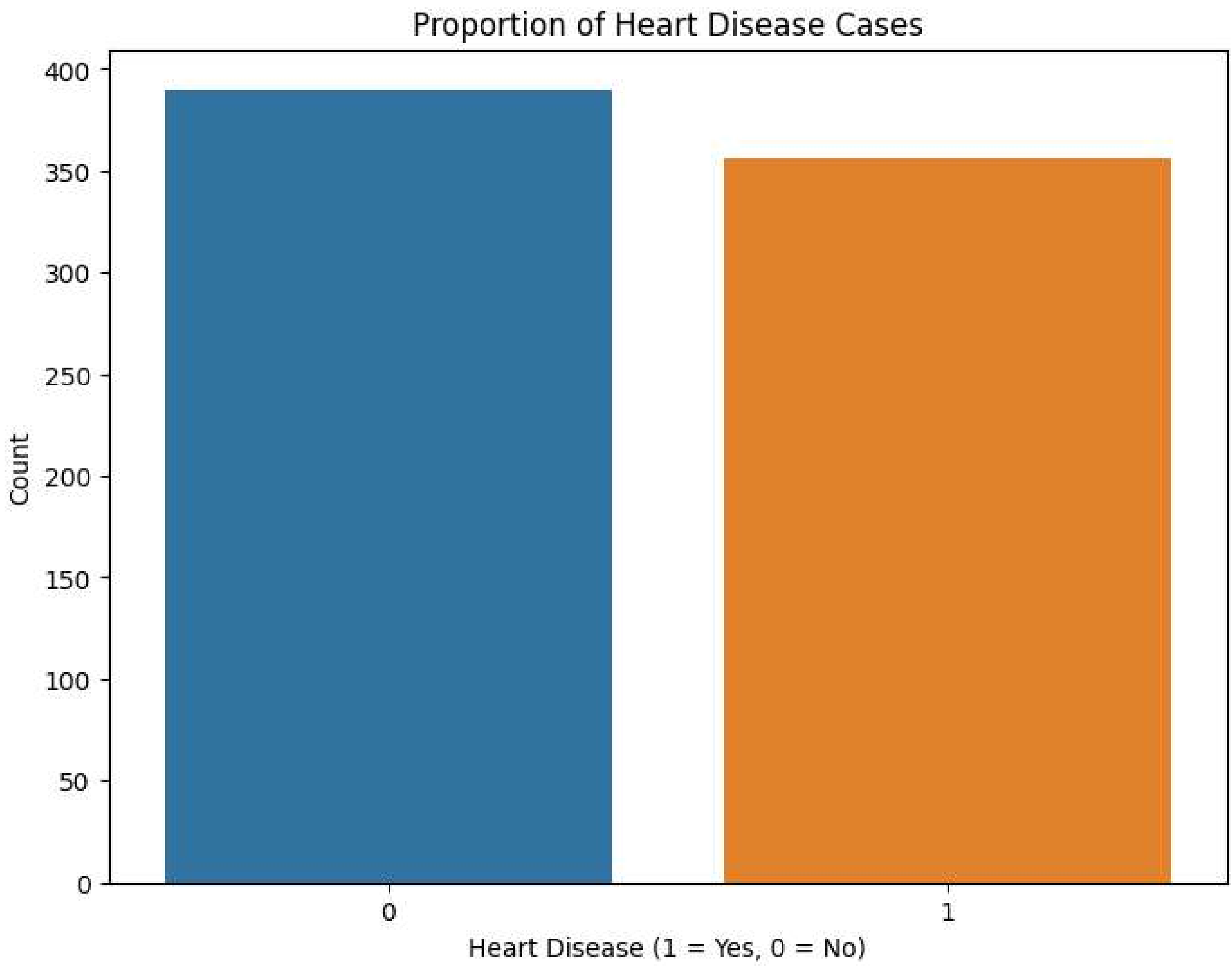
```
Cholesterol    172
RestingBP        1
dtype: int64
```

```
# removing the unnecessary datapoints from dataset
old_data = df.copy()
df = df[ (df['Cholesterol']!=0) & (df['RestingBP']!=0) ]
print('Old Data Shape: ',old_data.shape)
print('New Data Shape: ',df.shape)
```

```
Old Data Shape:  (918, 12)
New Data Shape:  (746, 12)
```

Checking the distribution of patients who have heart disease in our dataset

```
#Proportion of patients who have heart disease in our dataset
plt.figure(figsize=(8, 6))
ax = sns.countplot(data=df, x='HeartDisease')

# Set labels and title
plt.xlabel("Heart Disease (1 = Yes, 0 = No)")
plt.ylabel("Count")
plt.title("Proportion of Heart Disease Cases")

# Show the plot
plt.show()
```



Converting Categorical to Numerical Values

```
#data preprocessing
# one hot encoding
df_scalled = pd.get_dummies(df,drop_first = True)
df_scalled.head()
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	Sex_M	ChestPai
0	40	140	289	0	172	0.0	0	1	
1	49	160	180	0	156	1.0	1	0	
2	37	130	283	0	98	0.0	0	1	
3	48	138	214	0	108	1.5	1	0	
4	54	150	195	0	122	0.0	0	1	

```
x = df_scalled.drop(['HeartDisease'],axis=1)
y = df_scalled['HeartDisease']
```

Splitting the dataset into 2 partitions - Test & Training Set

```
# splitting the data
# Importing data spliting data library for training and testing from given dataframe
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.2)


# Scaling the data
# Module for Scaling the data
from sklearn.preprocessing import StandardScaler


# Normalize the data
sc = StandardScaler()
X_train = sc.fit_transform(x_train)
X_test = sc.transform(x_test)


X_train = pd.DataFrame(X_train, columns=x.columns)
X_test = pd.DataFrame(X_test, columns=x.columns)

display(X_train.head())
display(X_test.head())
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	Sex_M	ChestPain
0	0.244090	0.390043	0.392867	-0.446313	-0.491015	0.524183	0.543739	-
1	1.723166	1.543717	-0.191002	2.240581	-0.370634	-0.755075	0.543739	-
2	-0.495448	1.543717	0.392867	-0.446313	-1.494184	0.067305	0.543739	-
3	-0.918041	-0.763632	-0.465764	-0.446313	-1.012663	-0.846450	-1.839117	-
4	-0.706744	-1.340469	-0.087966	-0.446313	-0.009493	-0.846450	0.543739	-

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	Sex_M	ChestPainTy
0	0.138442	-0.071427	0.736319	2.240581	0.752916	-0.846450	-1.839117	1
1	0.561035	0.390043	-1.135496	-0.446313	0.793043	-0.846450	0.543739	-0
2	0.561035	-1.917306	-0.191002	-0.446313	0.632536	-0.755075	0.543739	-0
3	0.244090	1.543717	0.753492	-0.446313	0.191141	-0.115446	0.543739	-0
4	0.561035	-0.475213	0.942391	-0.446313	1.234438	-0.846450	0.543739	-0

Logistic Regression

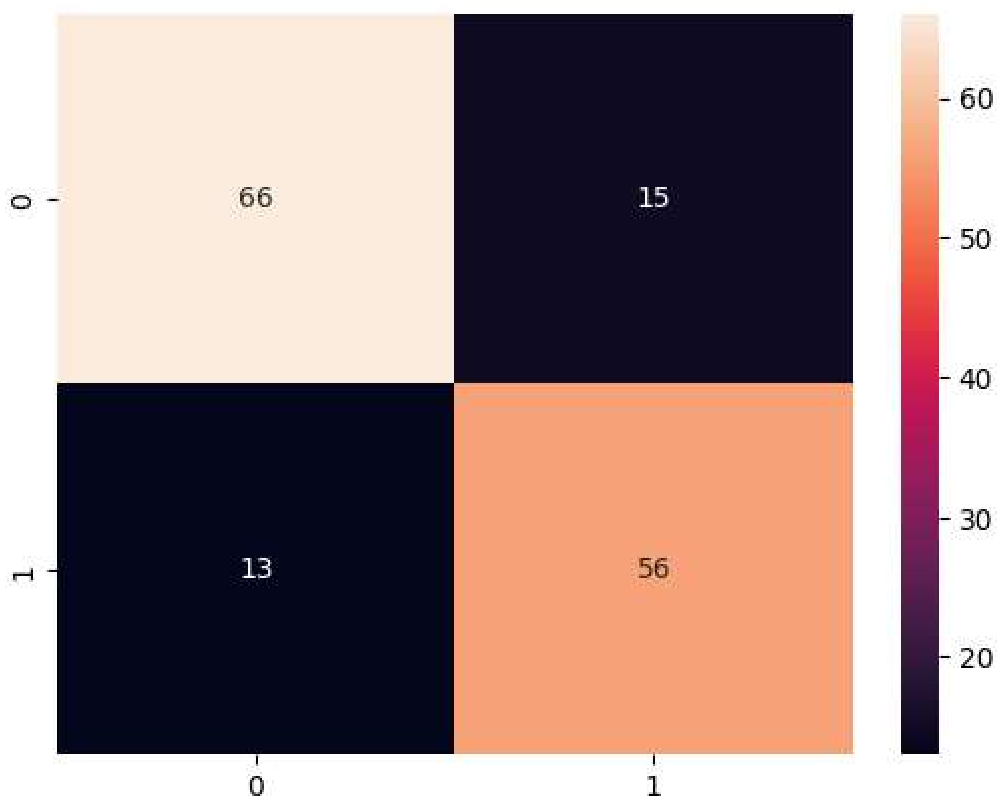
```
# Logistic Regression Model
from sklearn.linear_model import LogisticRegression

# Library module to check performance evaluation measures.
from sklearn.metrics import accuracy_score, roc_curve ,confusion_matrix,classification_repor
log_reg = LogisticRegression()

# Training the data
log_reg.fit(x_train,y_train)

# Testing the data
y_pred = log_reg.predict(x_test)

# Model Scores
log_train_accuracy = round(log_reg.score(x_train, y_train) * 100, 2)
log_accuracy = round(accuracy_score(y_pred, y_test) * 100, 2)
log_f1_score = round(f1_score(y_pred, y_test) * 100, 2)
print("Training Accuracy    :",log_train_accuracy,"%")
print("Model Accuracy Score :",log_accuracy,"%")
print("Classification_Report: \n",classification_report(y_test,y_pred))
```

```
Training Accuracy    : 87.25 %
Model Accuracy Score : 81.33 %
Classification_Report:
```

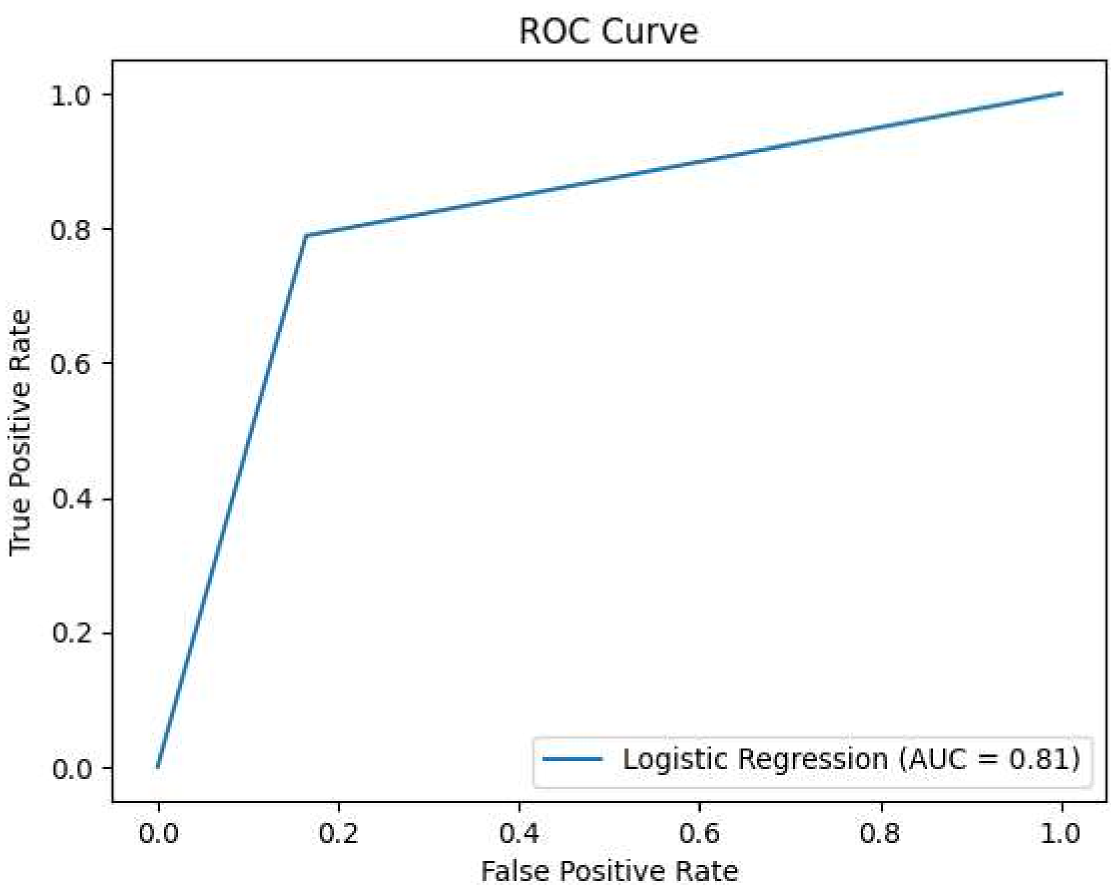
```
              precision    recall  f1-score   support

           0       0.81      0.84      0.82        79
           1       0.81      0.79      0.80        71

    accuracy                           0.81       150
   macro avg       0.81      0.81      0.81       150
weighted avg       0.81      0.81      0.81       150

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
#Plotting of Confusion Matrix
cm=confusion_matrix(y_pred, y_test)
conf_matrix =sns.heatmap(cm,annot=True)
print("Confusion Matrix:\n")
plt.show()
```

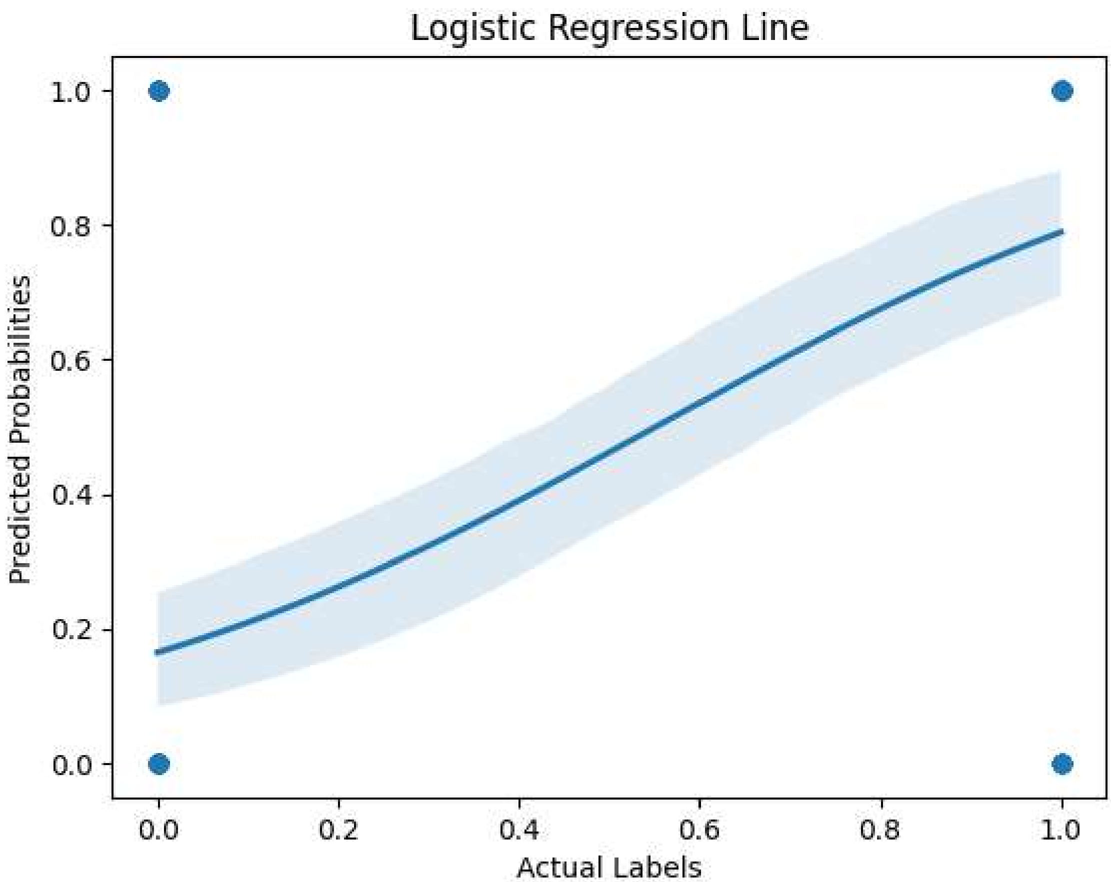
```
Confusion Matrix:
```



```
from sklearn.metrics import roc_curve, roc_auc_score

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_pred)

# Plot the ROC curve
plt.plot(fpr, tpr, label='Logistic Regression (AUC = %0.2f)' % auc_score)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```




```
sns.regplot(x=y_test, y=y_pred, logistic=True)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Probabilities')
plt.title('Logistic Regression Line')
plt.show()
```



KNN Algorithm

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
KNN_model = KNeighborsClassifier(n_neighbors=7)
KNN_model.fit(X_train, y_train)
```

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
y_pred_k = KNN_model.predict(X_test)
```

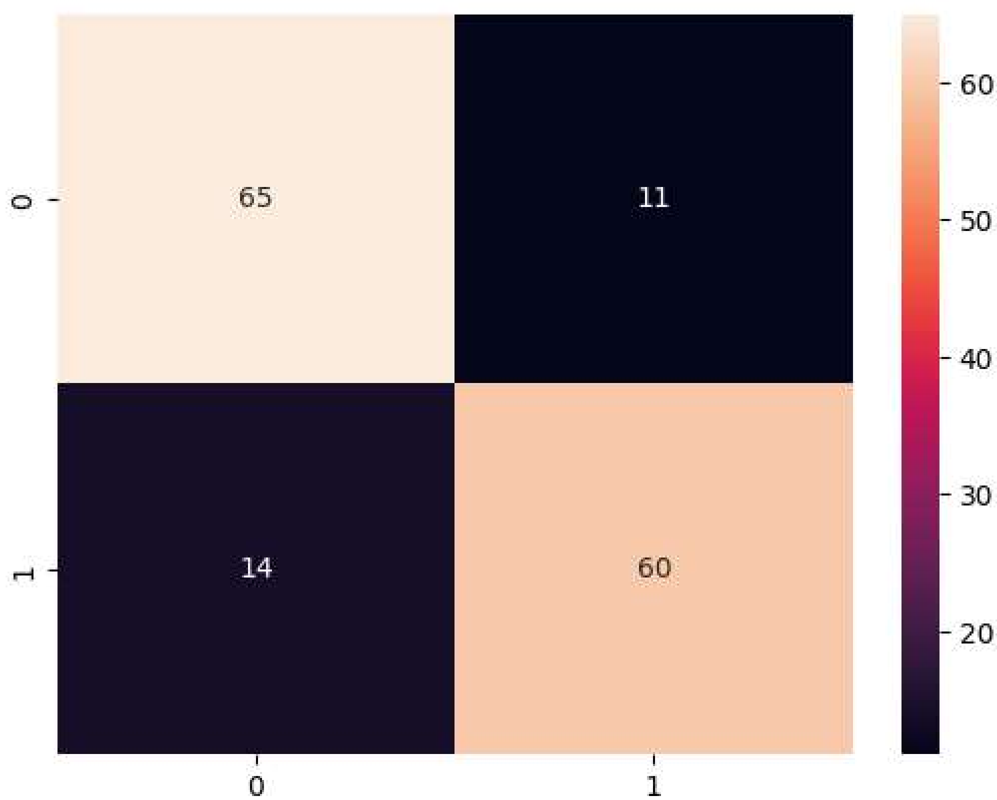
```
# Model Scores
knn_train_accuracy = round(KNN_model.score(x_train, y_train) * 100, 2)
knn_accuracy = round(accuracy_score(y_pred_k, y_test) * 100, 2)
knn_f1_score = round(f1_score(y_pred_k, y_test) * 100, 2)
print("Training Accuracy    :",knn_train_accuracy,"%")
print("Model Accuracy Score :",knn_accuracy,"%")
print("Classification_Report: \n",classification_report(y_test,y_pred_k))
```

```
Training Accuracy    : 54.7 %
Model Accuracy Score : 83.33 %
Classification_Report:
               precision    recall  f1-score   support

           0       0.86      0.82      0.84        79
           1       0.81      0.85      0.83        71

    accuracy                           0.83       150
   macro avg       0.83      0.83      0.83       150
weighted avg       0.83      0.83      0.83       150
```

Confusion Matrix for KNN

```
cm=confusion_matrix(y_pred_k, y_test)
conf_matrix =sns.heatmap(cm,annot=True)
print("Confusion Matrix:\n")
plt.show()
```

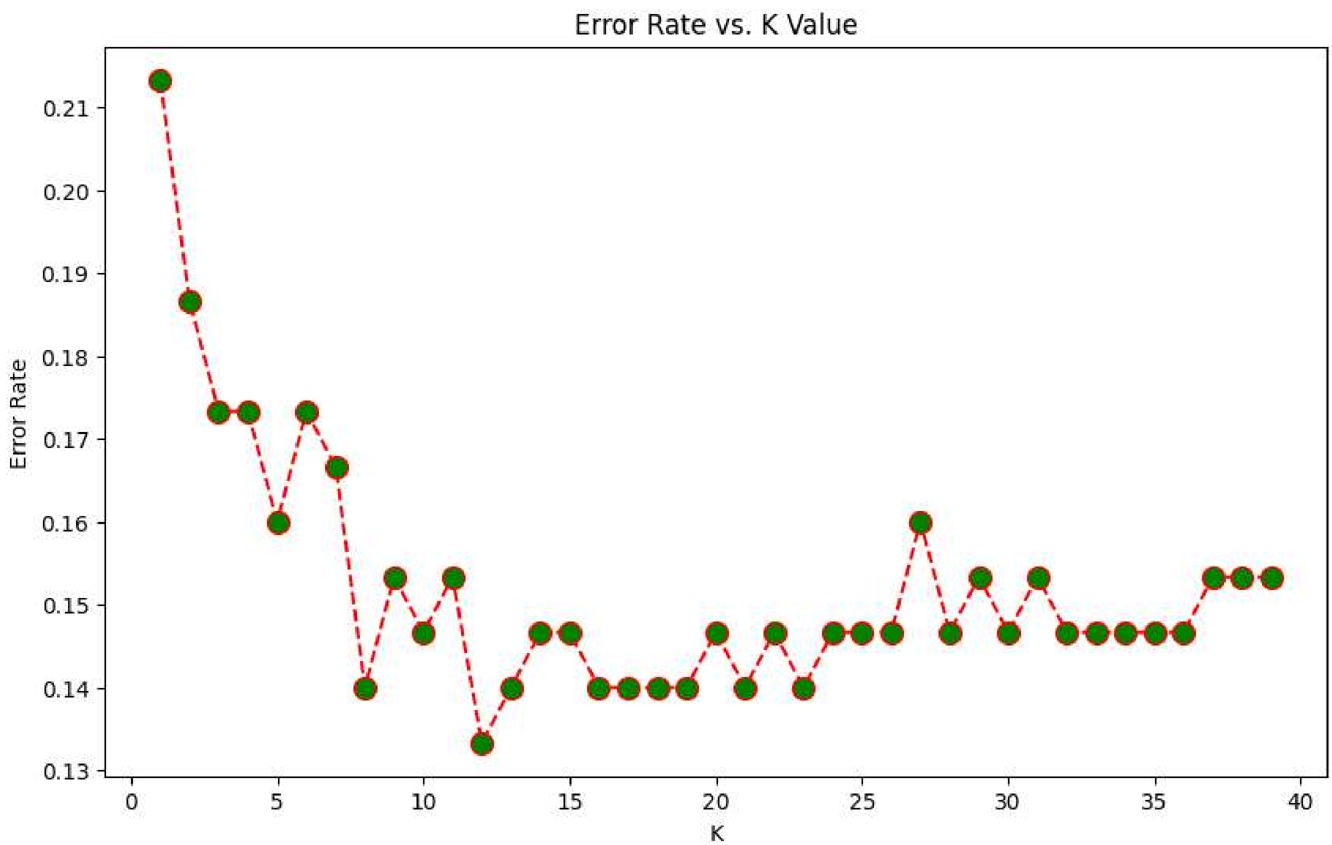
```
Confusion Matrix:
```



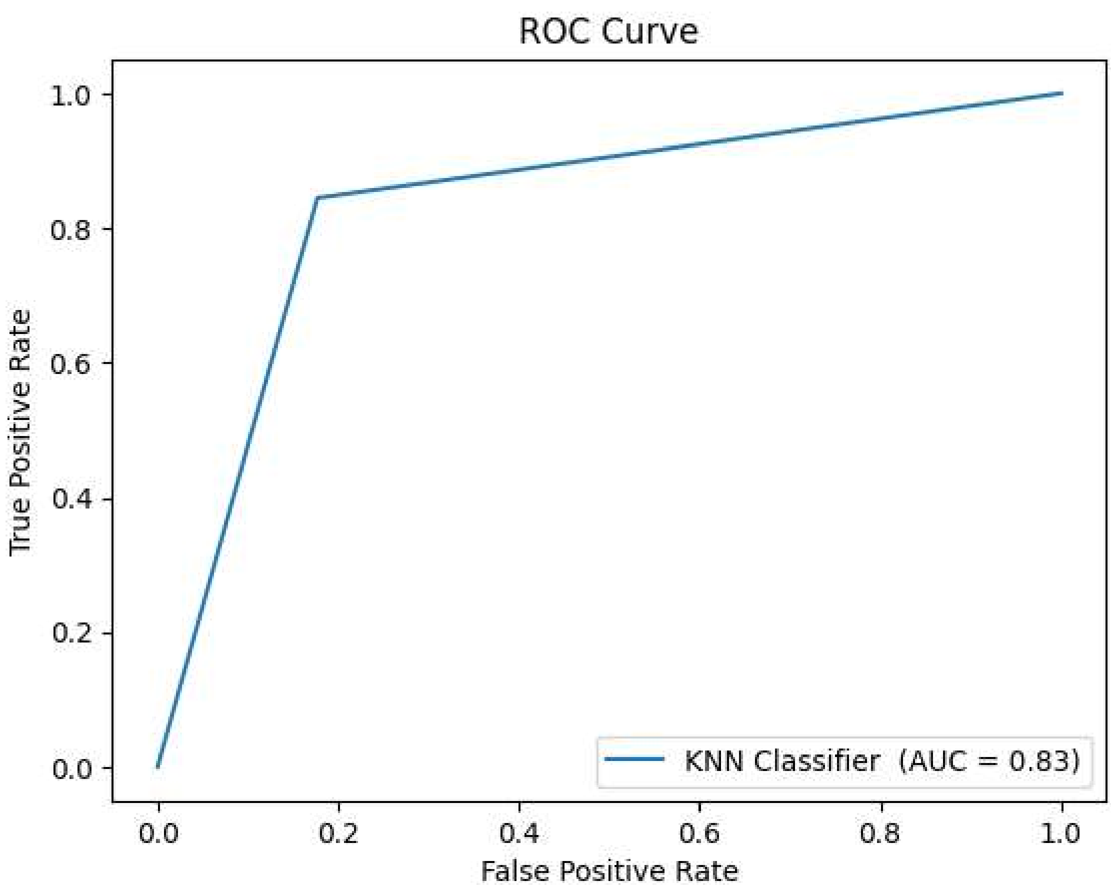
Selection of K value using Elbow Method

```
#For selecting K value
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))


import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='red', linestyle='dashed', marker='o',
         markerfacecolor='green', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



```
from sklearn.metrics import roc_curve, roc_auc_score

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_k)
# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_pred_k)

# Plot the ROC curve
plt.plot(fpr, tpr, label='KNN Classifier  (AUC = %0.2f)' % auc_score)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



Decision Tree Classification

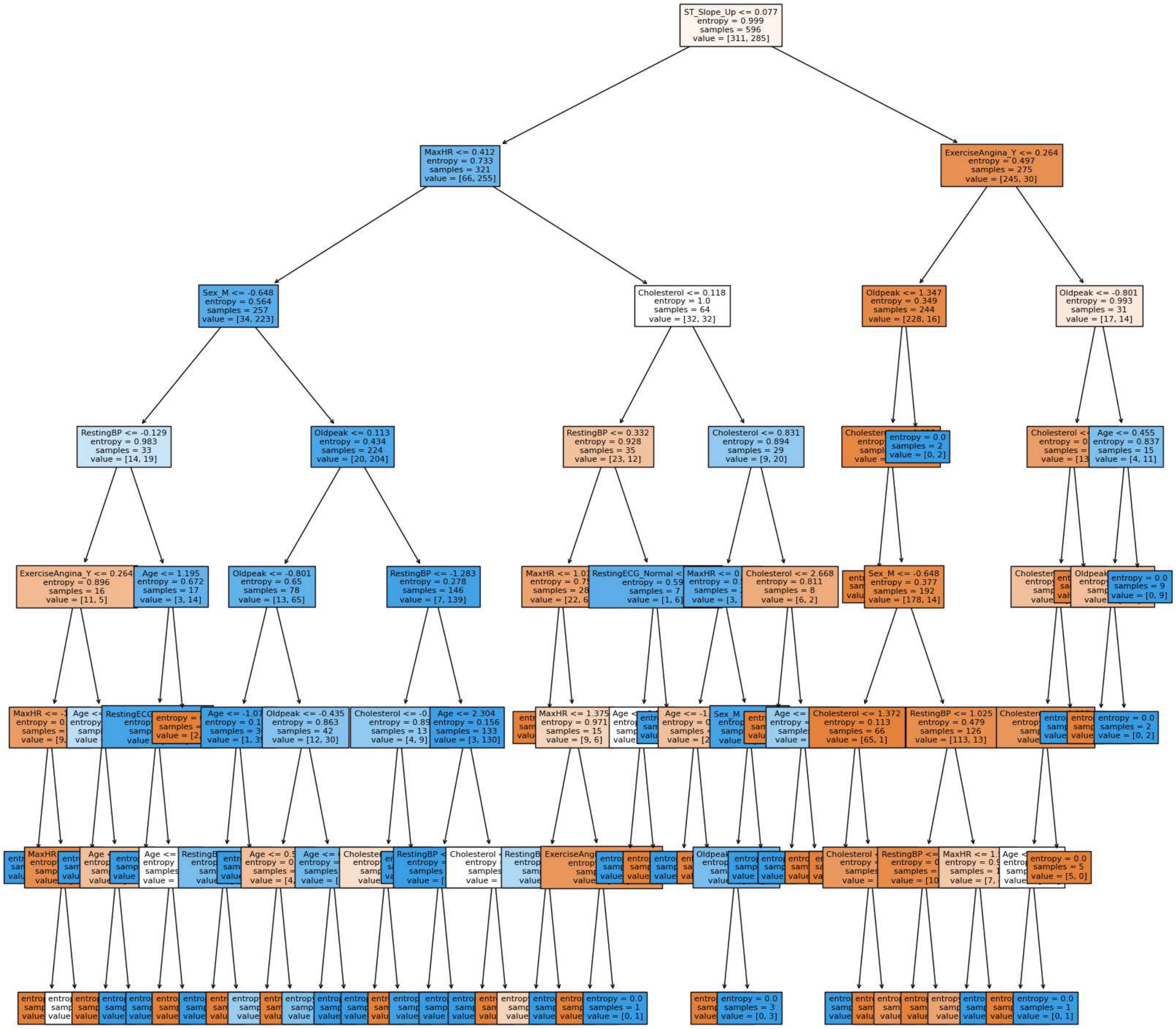
```
#decision tree classifier
from sklearn.tree import DecisionTreeClassifier
Decision_model = DecisionTreeClassifier()
Decision_model = DecisionTreeClassifier(criterion="entropy",max_depth=7)
Decision_model.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=7)
```

```
y_pred_d = Decision_model.predict(X_test)
```

```
#increase accuracy by using k folds cross-validation

from sklearn.model_selection import cross_val_score
scores = cross_val_score(Decision_model, X_train, y_train, cv=5)
```

```
from sklearn.tree import plot_tree, export_text

text_rep = export_text(Decision_model)

fig = plt.figure(figsize=(20,20))
graph = plot_tree(Decision_model, feature_names=list(x.columns), filled=True, fontsize=8)
```

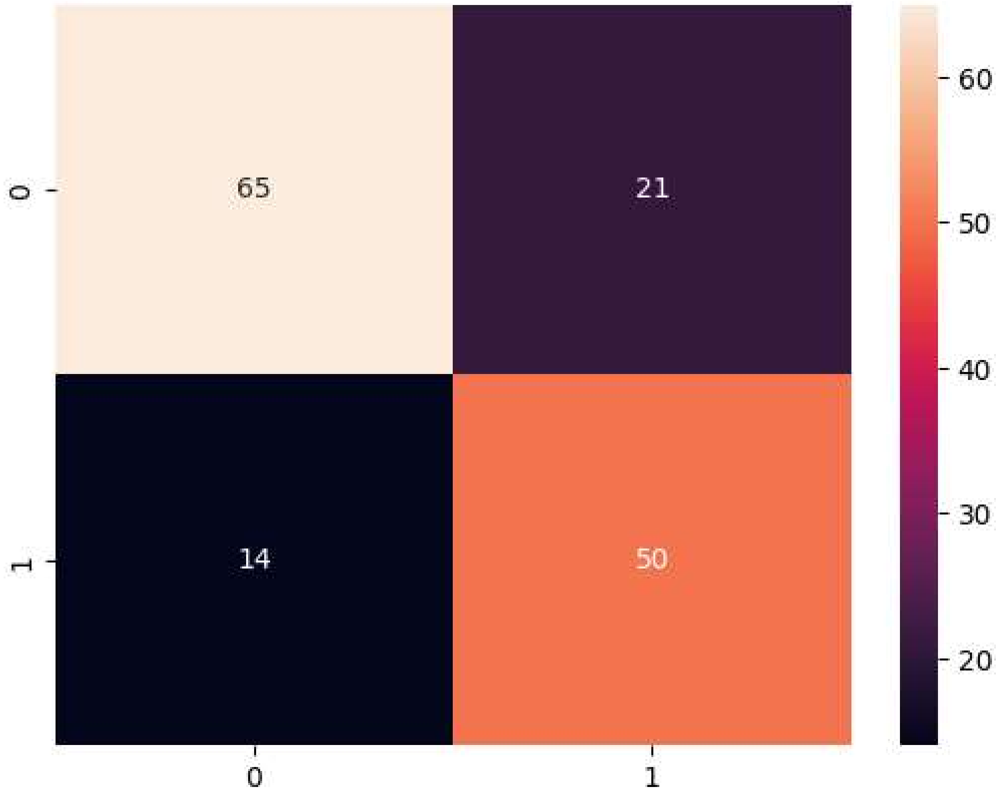

ST_Slope_Up <= 0.077
entropy = 0.999
samples = 596
value = [311, 285]
MaxHR <= 0.412
entropy = 0.733
samples = 321
value = [66, 255]
ExerciseAngina_Y <= 0.264
entropy = 0.497
samples = 275
value = [245, 30]
Sex_M <= -0.648
entropy = 0.564
samples = 257
value = [34, 223]
Cholesterol <= 0.118
entropy = 1.0
samples = 64
value = [32, 32]
Oldpeak <= 1.347
entropy = 0.349
samples = 244
value = [228, 16]
Oldpeak <= -0.801
entropy = 0.993
samples = 31
value = [17, 14]
RestingBP <= -0.129
entropy = 0.983
samples = 33
value = [14, 19]
Oldpeak <= 0.113
entropy = 0.434
samples = 224
value = [20, 204]
RestingBP <= 0.332
entropy = 0.928
samples = 35
value = [23, 12]
Cholesterol <= 0.831
entropy = 0.894
samples = 29
value = [9, 20]
entropy = 0.0
samples = 2
value = [0, 2]
Age <= 0.455
entropy = 0.837
samples = 15
value = [4, 11]
ExerciseAngina_Y <= 0.264
entropy = 0.896
samples = 16
value = [11, 5]
Age <= 1.195
entropy = 0.672
samples = 17
value = [3, 14]
Oldpeak <= -0.801
entropy = 0.65
samples = 78
value = [13, 65]
RestingBP <= -1.283
entropy = 0.278
samples = 146
value = [7, 139]
Cholesterol <= 2.668
entropy = 0.811
samples = 8
value = [6, 2]
Sex_M <= -0.648
entropy = 0.377
samples = 192
value = [178, 14]
entropy = 0.0
samples = 9
value = [0, 9]
Oldpeak <= -0.435
entropy = 0.863
samples = 42
value = [12, 30]
Age <= 2.304
entropy = 0.156
samples = 133
value = [3, 130]
MaxHR <= 1.375
entropy = 0.971
samples = 15
value = [9, 6]
Cholesterol <= 1.372
entropy = 0.113
samples = 66
value = [65, 1]
RestingBP <= 1.025
entropy = 0.479
samples = 126
value = [113, 13]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 5
value = [5, 0]
entropy = 0.0
samples = 1
value = [0, 1]
entropy = 0.0
samples = 3
value = [0, 3]
entropy = 0.0
samples = 1
value = [0, 1]

```
# Model Scores
from sklearn.metrics import accuracy_score, roc_curve ,confusion_matrix,classification_repor
dec_train_accuracy = round(Decision_model.score(x_train, y_train) * 100, 2)
dec_accuracy = round(accuracy_score(y_pred_d, y_test) * 100, 2)
dec_f1_score = round(f1_score(y_pred_d, y_test) * 100, 2)
print("Training Accuracy    :",dec_train_accuracy,"%")
print("Model Accuracy Score :",dec_accuracy,"%")
print("Classification_Report: \n",classification_report(y_test,y_pred_d))
```

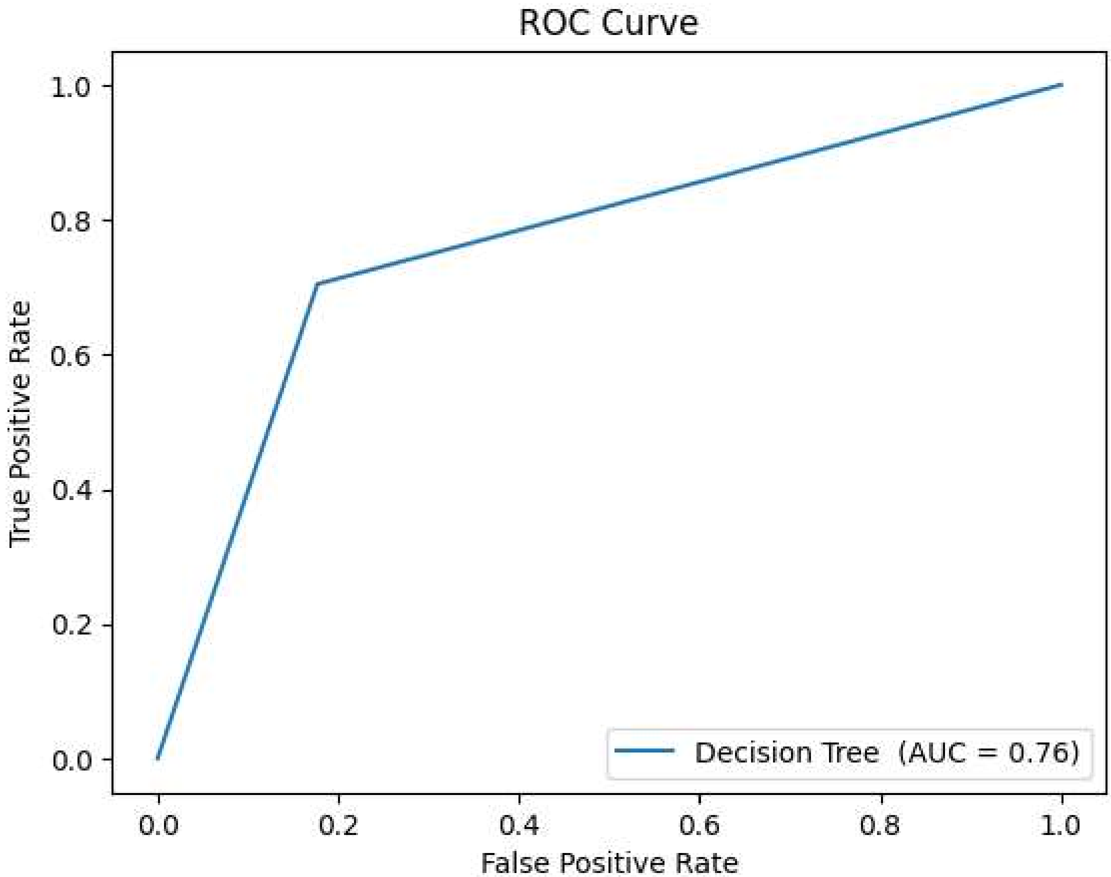
```
Training Accuracy    : 16.11 %
Model Accuracy Score : 76.67 %
Classification_Report:
               precision    recall  f1-score   support

           0       0.76      0.82      0.79        79
           1       0.78      0.70      0.74        71

    accuracy                           0.77       150
   macro avg       0.77      0.76      0.76       150
weighted avg       0.77      0.77      0.77       150
```

```
cm=confusion_matrix(y_pred_d, y_test)
conf_matrix =sns.heatmap(cm,annot=True)
print("Confusion Matrix:\n")
plt.show()
```

```
Confusion Matrix:
```




```
from sklearn.metrics import roc_curve, roc_auc_score

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_d)
# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_pred_d)

# Plot the ROC curve
plt.plot(fpr, tpr, label='Decision Tree  (AUC = %0.2f)' % auc_score)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



Random Forest Classifier

```
# Create a Random Forest model
from sklearn.ensemble import RandomForestClassifier
RandomForest_model = RandomForestClassifier(n_estimators=50, criterion='entropy', random_sta
```

```
# Train the model
RandomForest_model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=50, random_state=40)
```

```
# Make predictions on the test set
y_pred_r = RandomForest_model.predict(X_test)
```

```
# Visualize individual trees
for i in range(min(4, len(RandomForest_model.estimators_))):
    plt.figure(figsize=(10, 7))
    plot_tree(RandomForest_model.estimators_[i], filled=True, feature_names=[f'feature_{i}'
    plt.title(f"Decision Tree {i+1}")
    plt.show()
```