TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY
ACADEMIC YEAR 2023/2024 Session 202301

BACS3013 DATA SCIENCE: Assignment Documentation

Title:Heart Failure Prediction

Programme and Tutorial Class: RDS2S2G3 & RDS3S1G3
Tutor Name:Ms. Noor Aida Binti Husaini
Submission Date:7 May 2023
Team Member:

 **Student Name:Ong Weng Kai** **Student ID:22WMR03309** **Module-in-Charge:Part D(2 model)**	 **Student Name:Cho Wei Bin** **Student ID:22WMR04086** **Module-in-Charge:part E, F, H**
 **Student Name:Chong Yao Liang** **Student ID:22WMR01810** **Module-in-Charge:part c and**	 **Student Name:Tang Sharren** **Student ID:21WMR01086** **Module-in-Charge:Part 1,2, 3.1,3.2**

D(2model)	

Table of Content

1.0 Business Understanding

1.1 Heart disease description

Heart failure is a condition in which the heart cannot pump enough blood to meet the body's needs. It can be caused by various factors, such as coronary artery disease, high blood pressure, diabetes, infections, or congenital heart defects. Heart failure can lead to symptoms such as shortness of breath, fatigue, swelling in the legs or abdomen, coughing, or irregular heartbeat. Heart failure can be diagnosed by tests such as electrocardiogram, echocardiogram, blood tests, or chest X-ray. Heart failure can be treated by medications, lifestyle changes, surgery, or devices that help the heart pump better. Heart failure is a serious and chronic condition that requires regular monitoring and follow-up with a doctor.

1.2 Project Objectives

This project is to identify patients who are at risk of developing heart failure, a condition where the heart cannot pump enough blood to meet the body's needs. The objectives are to use historical data from patients with heart failure and other relevant features to build a predictive model that can classify new patients as high or low risk. The expected outcomes help us to have an early detection and management of heart disease in order to improve the quality of care, reduce hospitalizations and mortality, and optimize resource allocation for patients with heart failure.

1.3 Using Four Major Models

1. K-Nearest Neighbour Classifier (KNN)

The K-Nearest Neighbor (KNN) Classifier is a versatile supervised learning algorithm that can solve both classification and regression problems. However, it is more commonly used for classification tasks in the industry. When evaluating any technique, there are three key aspects to consider: the ease of interpreting the output, the calculation time, and the predictive ability. In the case of KNN, the output is relatively easy to interpret because the output is the predicted class of a new data point based on k nearest neighbours in the training data. The calculation time is an important consideration when dealing with large datasets because it requires calculating all the distance between the training data point and the new data point which could slow down the calculation time. However, the calculation can be sped up by using optimization techniques such as tree-based algorithms or dimensionality reduction techniques. The most important consideration for KNN would be predictive ability because KNN has been shown good predictive ability in many cases however performance can be influenced by the distance metric

used and the choice of k. Despite being one of the most basic classification algorithms, KNN can produce highly competitive results. It can also apply to regression problems.

2. Support Vector Machines (SVM)

A support vector machine (SVM) is a supervised machine learning algorithm that can be used for classification or regression problems. It works by finding the best hyperplane that separates the data points of different classes or predicts the output value for an input. SVMs are based on the idea of maximizing the margin, which is the distance between the hyperplane and the closest data points of each class.SVM try to find the decision boundary that can maximise the margin while correctly classifying the training data. Another advantage is it would be less likely to occur over lifting than other classification algorithms because of the margin maximization criterion it would tend to select a simpler model that can generalize well with new data. However SVM is more sensitive toward the choice of hyperparameters such as regularization parameters and kernel function, the performance of the model would be largely influenced by it. Lastly, SVMs can also handle nonlinear problems by using kernel functions that map the data to a higher-dimensional space where a linear hyperplane can be found.

3. Decision Tree

Decision tree algorithms are a type of supervised learning method that can be used for classification or regression problems. They work by splitting the data into smaller subsets based on certain criteria, such as the value of a feature or the outcome of a test. The result is a tree-like structure where each node represents a decision or a prediction. Decision tree algorithms are easy to interpret and can handle both numerical and categorical data.

4. Random Forest

Random Forest is a classification algorithm that constructs multiple decision trees to classify data. During the creation of each tree, the algorithm employs techniques such as bagging and feature randomization to generate an uncorrelated ensemble of trees. The combined prediction of this forest of trees is more accurate than that of any individual tree. Random Forest can produce accurate predictions even for large datasets, such as one with 382154 rows and 12 columns. Additionally, the presence of missing data does not affect the accuracy of the algorithm’s predictions, as Random Forest can estimate missing values while maintaining accuracy. This makes it a suitable choice for datasets with null values.

1.4 Target features of the dataset

[Heart_Failure_Predict_8_Classification_Techniques | Kaggle]

The 'HeartDisease' column is the targeted feature of the dataset. The 'HeartDisease' determines if they diagnosed a patient with heart disease or not. It is important to find out the factors that will affect the chance of getting heart disease so that doctors can help the patient reduce his/her risk an early detection and management of heart disease would be very helpful.

2.0 Data Understanding

In this project, we are going to predict whether a patient would be diagnosed with heart disease. The target feature or attribute in this dataset is the 'HeartDisease' column.

2.1 Reading the dataset in dataframe

The diagram below shows the data frame of our dataset using the read method from pandas library.

```
heart = pd.read_csv('heart.csv')
heart
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
5	39	M	NAP	120	339	0	Normal	170	N	0.0	Up	0
6	45	F	ATA	130	237	0	Normal	170	N	0.0	Up	0
7	54	M	ATA	110	208	0	Normal	142	N	0.0	Up	0
8	37	M	ASY	140	207	0	Normal	130	Y	1.5	Flat	1
9	48	F	ATA	120	284	0	Normal	120	N	0.0	Up	0
10	37	F	NAP	130	211	0	Normal	142	N	0.0	Up	0
11	58	M	ATA	136	164	0	ST	99	Y	2.0	Flat	1

By using the shape method, we know that there are 918 rows and 12 columns.

```
heart.shape
```

```
(918, 12)
```

2.2 Data description(Data types & its value)

By using dtypes method we get to know the data types of each columns.

```
Age                int64
Sex               object
ChestPainType     object
RestingBP          int64
Cholesterol        int64
FastingBS          int64
RestingECG        object
MaxHR              int64
ExerciseAngina    object
Oldpeak          float64
ST_Slope          object
HeartDisease       int64
dtype: object
```

There are some string data saved as the object type so we will convert them into string type to identify the categorical and numerical data later.

```
objectData = heart.select_dtypes(include="object").columns
heart[objectData]=heart[objectData].astype("string")
```

We have to identify the categorical and numerical columns so that we can perform label encoding during the data preparation step.

```
string_col=heart.select_dtypes("string").columns.to_list()
string_col
```

By using this method, we know that 'Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope' are the categorical data

```
num_col=heart.columns.to_list()
#remove columns with string data
for col in string_col:
    num_col.remove(col)
num_col.remove("HeartDisease")
print(num_col)
```

We exclude the target feature which is ‘HeartDisease’ and numerical data are 'Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak'

The unique method helps us to understand the values in each columns so that we can further confirm that a column is categorical or numerical.

For categorical columns,

```
heart.Sex.unique()
```

```
<StringArray>
['M', 'F']
Length: 2, dtype: string
```

```
heart.ChestPainType.unique()
```

```
<StringArray>
['ATA', 'NAP', 'ASY', 'TA']
Length: 4, dtype: string
```

```
heart.RestingECG.unique()
```

```
<StringArray>
['Normal', 'ST', 'LVH']
Length: 3, dtype: string
```

```
heart.ExerciseAngina.unique()
```

```
<StringArray>
['N', 'Y']
Length: 2, dtype: string
```

```
heart.ST_Slope.unique()
```

```
<StringArray>
['Up', 'Flat', 'Down']
Length: 3, dtype: string
```

For numerical columns,

```
heart.FastingBS.unique()
```

```
array([0, 1], dtype=int64)
```

FastingBS should be a categorical variable from the result here

```
heart.Age.unique()
```

```
array([40, 49, 37, 48, 54, 39, 45, 58, 42, 38, 43, 60, 36, 44, 53, 52, 51,
       56, 41, 32, 65, 35, 59, 50, 47, 31, 46, 57, 55, 63, 66, 34, 33, 61,
       29, 62, 28, 30, 74, 68, 72, 64, 69, 67, 73, 70, 77, 75, 76, 71],
      dtype=int64)
```

```
heart.MaxHR.unique()
```

```
array([172, 156,  98, 108, 122, 170, 142, 130, 120,  99, 145, 140, 137,
       150, 166, 165, 125, 160, 164, 138, 178, 112, 118, 127, 114, 154,
       155,  87, 148, 100, 168, 184, 121, 153, 134,  96, 174, 175, 144,
        82, 135, 115, 128, 116,  94, 110,  92, 180, 152, 124, 106, 185,
       139, 190, 146, 158, 132, 176, 119, 188, 162, 105,  90, 136, 167,
       129, 102, 143, 103,  91, 126,  93, 131, 149, 123, 182, 141,  77,
       109, 133, 179, 113, 104,  95,  72,  97, 117,  86,  63, 157,  83,
        60,  70, 163,  67,  78,  84, 111,  80, 107, 161,  69,  88,  73,
       159, 151, 181, 186, 177, 173, 169, 171, 147,  71, 192, 195, 194,
       187, 202], dtype=int64)
```

```
heart.Oldpeak.unique()
```

```
array([ 0. ,  1. ,  1.5,  2. ,  3. ,  4. ,  0.5,  2.5,  5. ,  0.8,  0.7,
        1.4,  2.1,  0.4,  0.2,  1.7,  2.2,  0.1,  1.6,  1.3,  0.3,  1.8,
        2.6, -0.9,  2.8, -2.6, -1.5, -0.1,  0.9,  1.1,  2.4, -1. , -1.1,
       -0.7, -0.8,  3.7,  1.2, -0.5, -2. ,  1.9,  3.5,  0.6,  3.1,  2.3,
        3.4,  3.6,  4.2,  3.2,  5.6,  3.8,  2.9,  6.2,  4.4])
```

```
heart.RestingBP.unique()
```

```
array([140, 160, 130, 138, 150, 120, 110, 136, 115, 100, 124, 113, 125,
       145, 112, 132, 118, 170, 142, 190, 135, 180, 108, 155, 128, 106,
        92, 200, 122,  98, 105, 133,  95,  80, 137, 185, 165, 126, 152,
       116,   0, 144, 154, 134, 104, 139, 131, 141, 178, 146, 158, 123,
       102,  96, 143, 172, 156, 114, 127, 101, 174,  94, 148, 117, 192,
       129, 164], dtype=int64)
```

```
heart.Cholesterol.unique()
```

```
array([289, 180, 283, 214, 195, 339, 237, 208, 207, 284, 211, 164, 204,
       234, 273, 196, 201, 248, 267, 223, 184, 288, 215, 209, 260, 468,
       188, 518, 167, 224, 172, 186, 254, 306, 250, 177, 227, 230, 294,
       264, 259, 175, 318, 216, 340, 233, 205, 245, 194, 270, 213, 365,
       342, 253, 277, 202, 297, 225, 246, 412, 265, 182, 218, 268, 163,
       529, 100, 206, 238, 139, 263, 291, 229, 307, 210, 329, 147,  85,
       269, 275, 179, 392, 466, 129, 241, 255, 276, 282, 338, 160, 156,
       272, 240, 393, 161, 228, 292, 388, 166, 247, 331, 341, 243, 279,
       198, 249, 168, 603, 159, 190, 185, 290, 212, 231, 222, 235, 320,
       187, 266, 287, 404, 312, 251, 328, 285, 280, 192, 193, 308, 219,
       257, 132, 226, 217, 303, 298, 256, 117, 295, 173, 315, 281, 309,
       200, 336, 355, 326, 171, 491, 271, 274, 394, 221, 126, 305, 220,
       242, 347, 344, 358, 169, 181,   0, 236, 203, 153, 316, 311, 252,
       458, 384, 258, 349, 142, 197, 113, 261, 310, 232, 110, 123, 170,
       369, 152, 244, 165, 337, 300, 333, 385, 322, 564, 239, 293, 407,
       149, 199, 417, 178, 319, 354, 330, 302, 313, 141, 327, 304, 286,
       360, 262, 325, 299, 409, 174, 183, 321, 353, 335, 278, 157, 176,
       131], dtype=int64)
```

There is zero values in RestingBP and Cholesterol which is impossible, these zero values are considered as outliers and will be treated during data preparation step

Identify null value in dataset

```
heart.isnull().sum()
```

```
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
```

This indicates that there is no null value present in the dataset.

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

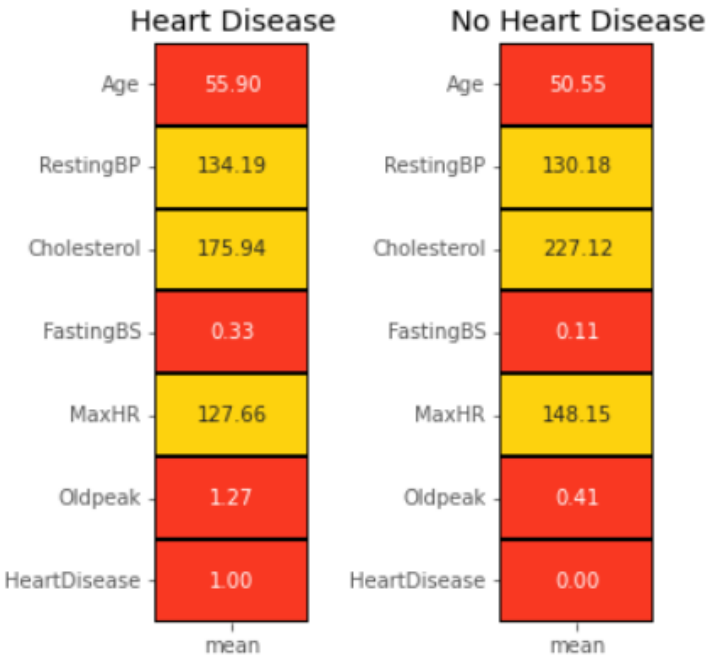
Using the describe() method, we can list out all the attributes of this dataset

- Age: age of the patient [years]
- Sex: sex of the patient [M: Male, F: Female]
- ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- RestingBP: resting blood pressure [mm Hg]
- Cholesterol: serum cholesterol [mm/dl]
- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
- ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
- Oldpeak: oldpeak = ST [Numeric value measured in depression]
- ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
- HeartDisease: output class [1: heart disease, 0: Normal]

2.3 EDA Exploratory Data Analysis

2.3.1 Using Heatmap

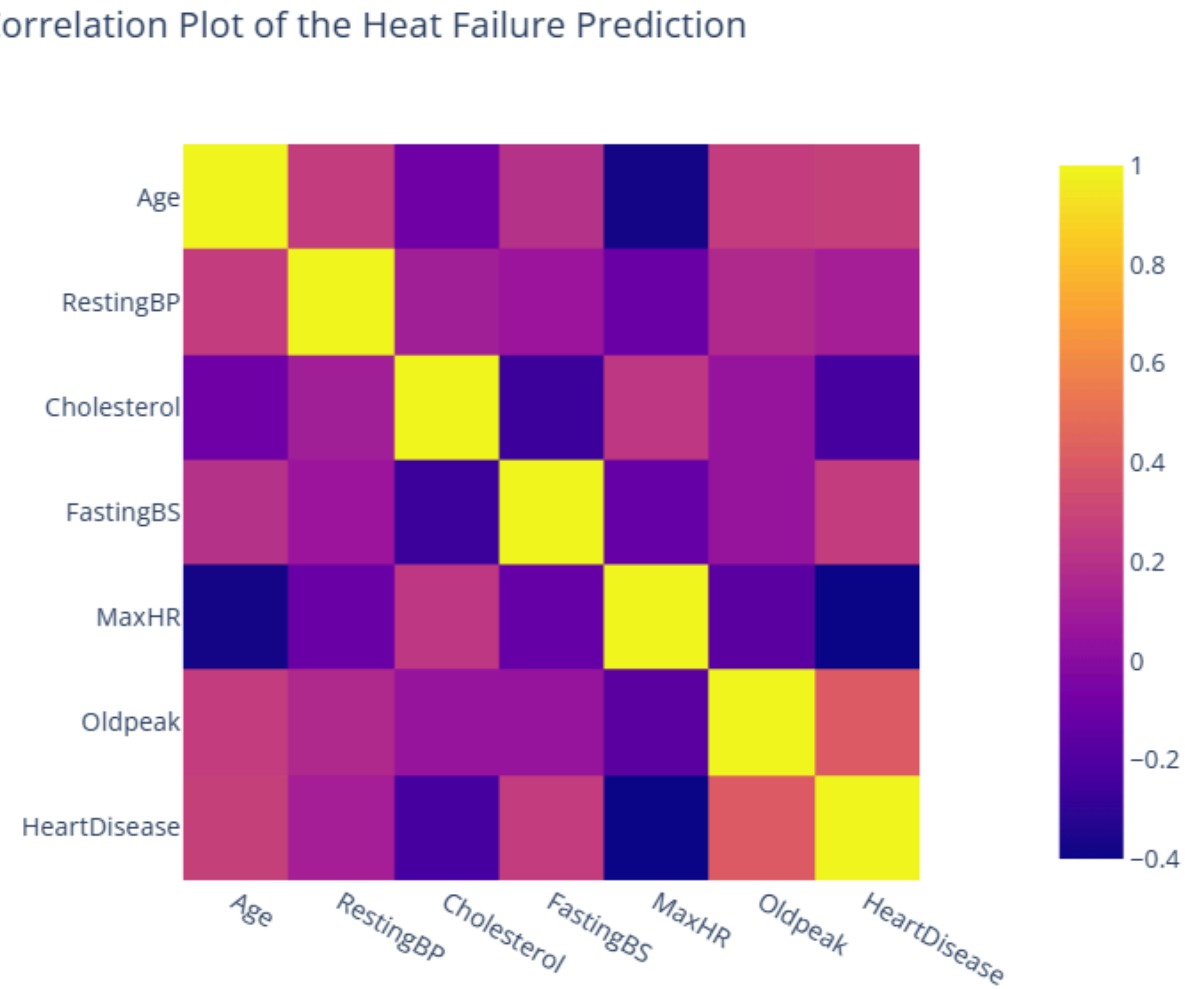
1. Visualizing average values of all the attributes for cases of with heart disease and without heart disease



2. **Correlation Matrix**

If the coefficient is close to -1 or 1, then there is a strong relationship between the two variables. If it is close to 0, then there is no relationship between the two variables. The sign of the coefficient indicates the direction of the relationship. Therefore, we can say that

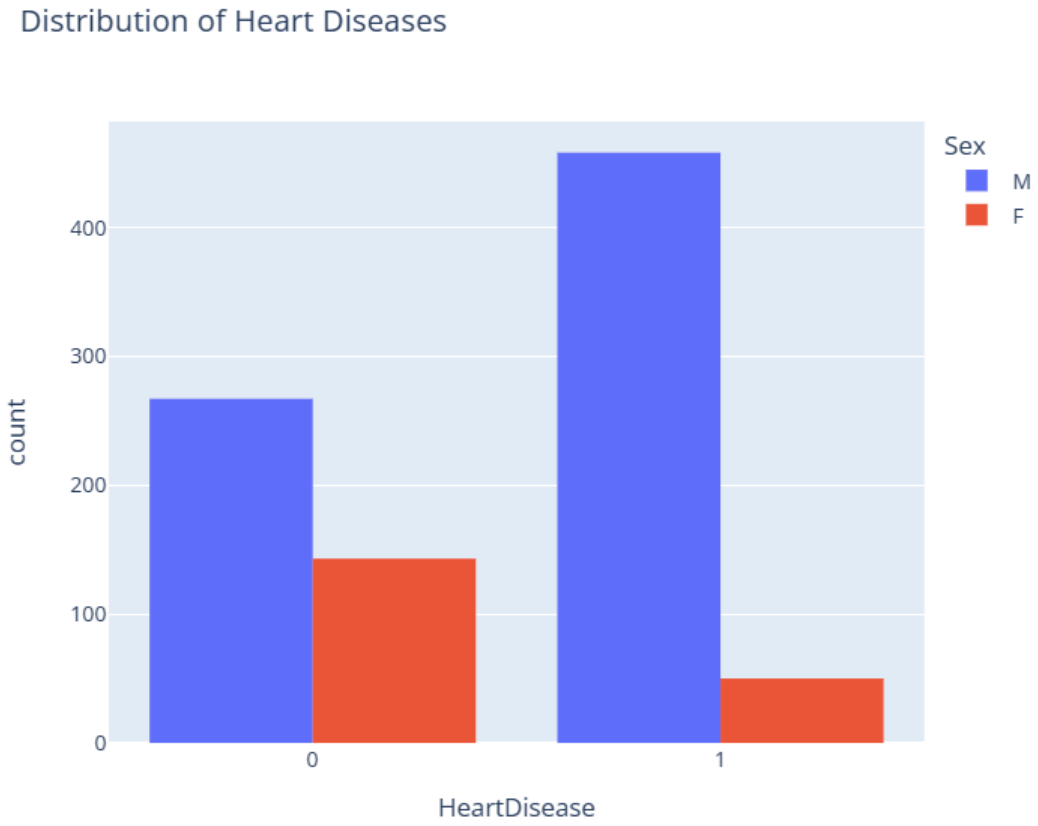
- Lighter shades represents positive correlation
- Darker shades represents negative correlation



Here we can see Heart Disease has a high negative correlation with "MaxHR" and somewhat negative correlation with "Cholesterol", where as here positive correlation with "Oldpeak","FastingBS" and "RestingBP

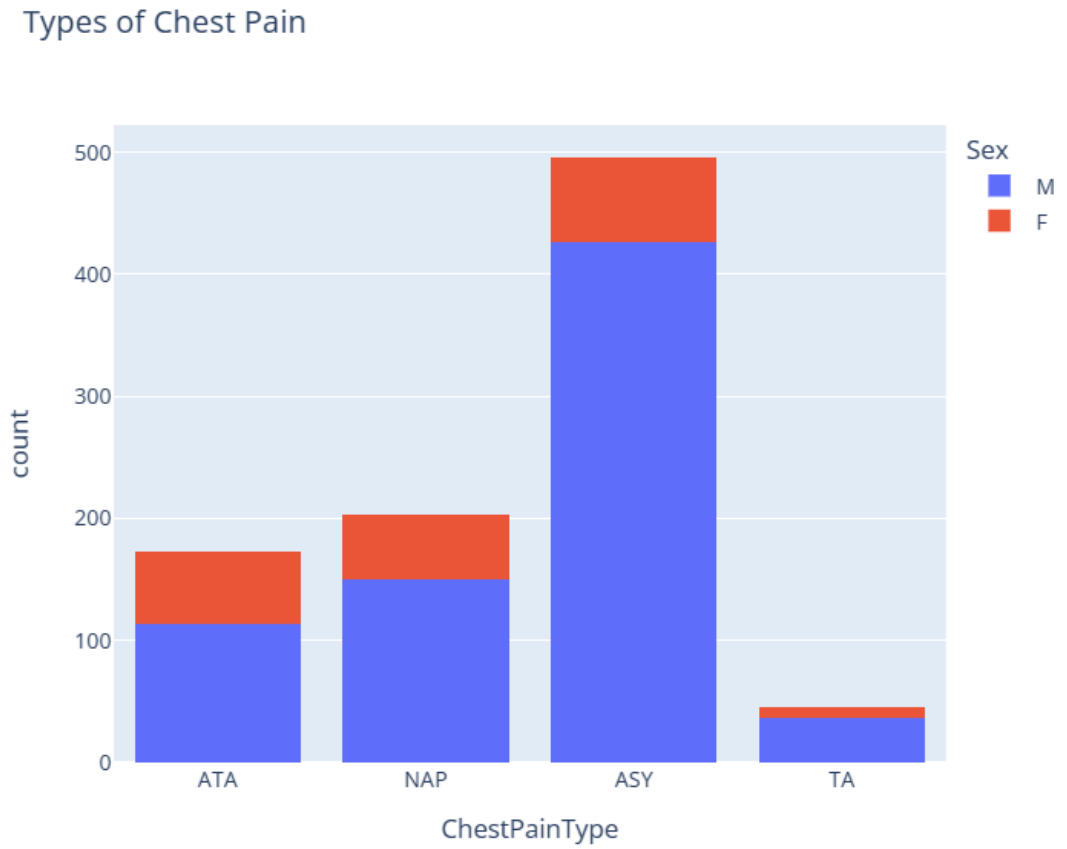
2.3.2 Descriptive Statistics - Histogram

1.Distribution of Heart Disease among genders



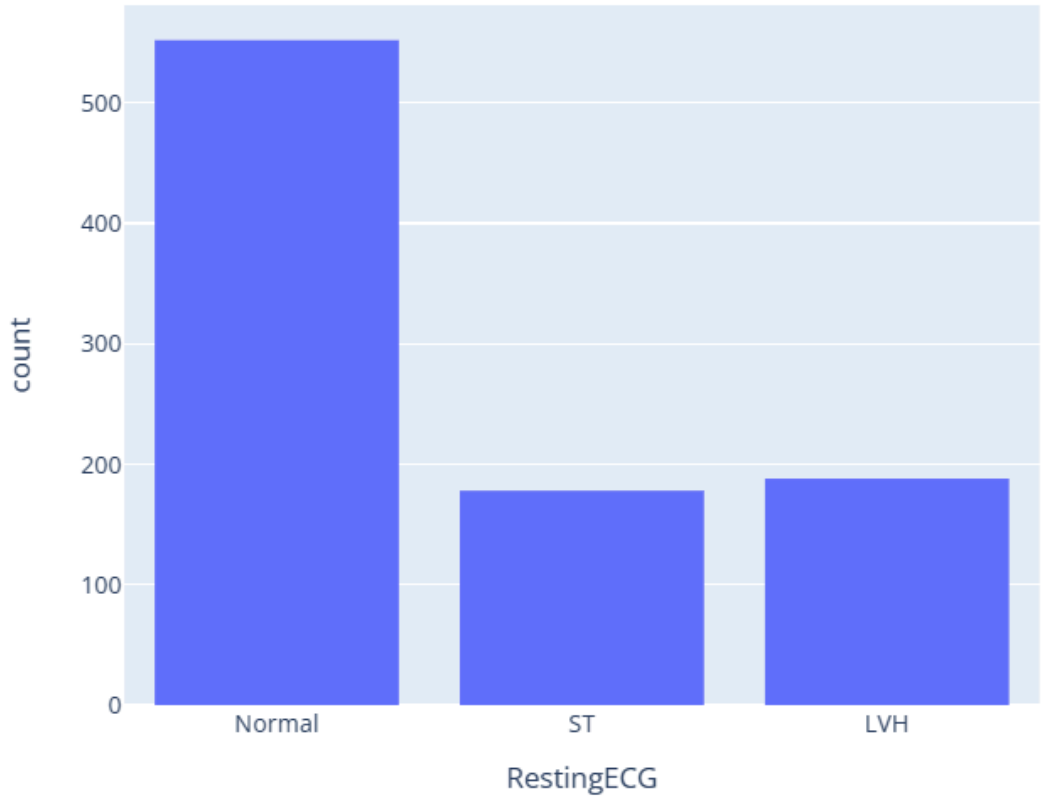
So, male have a higher chance of getting heart disease when compared to female

2.Distribution of Chest Pain Type among genders

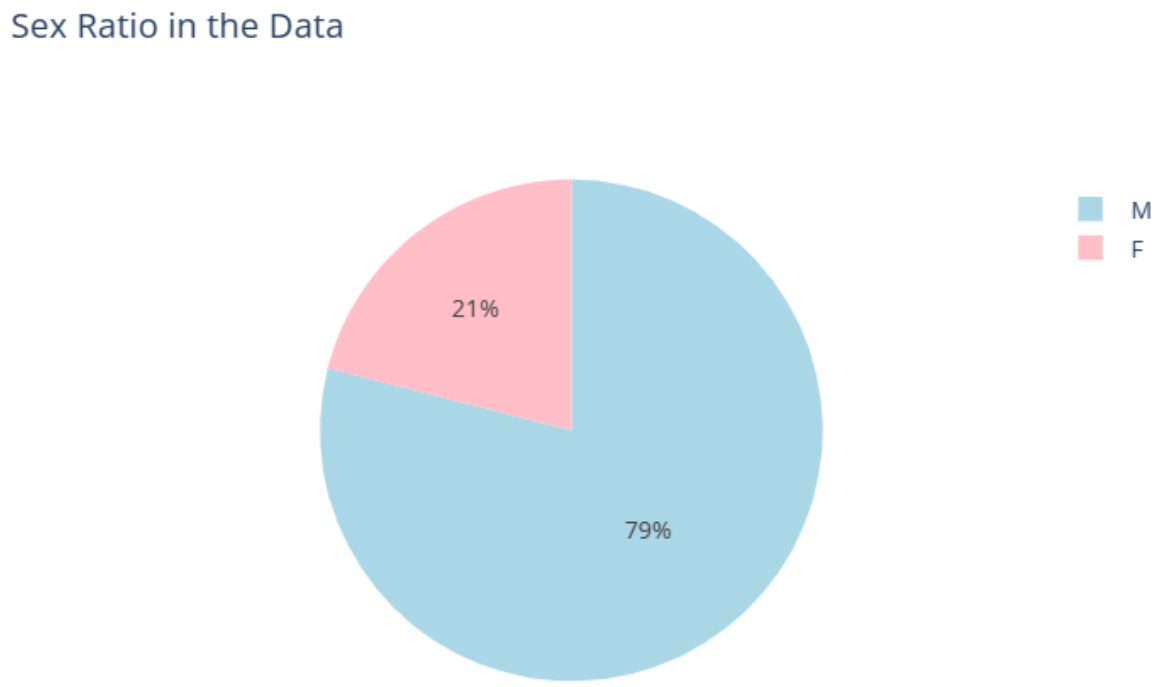


Most of the male is having 'ASY' (Asymptomatic) chest pain and most female is having 'ATA' (Atypical Angina) chest pain

3.Distribution of Resting ECG

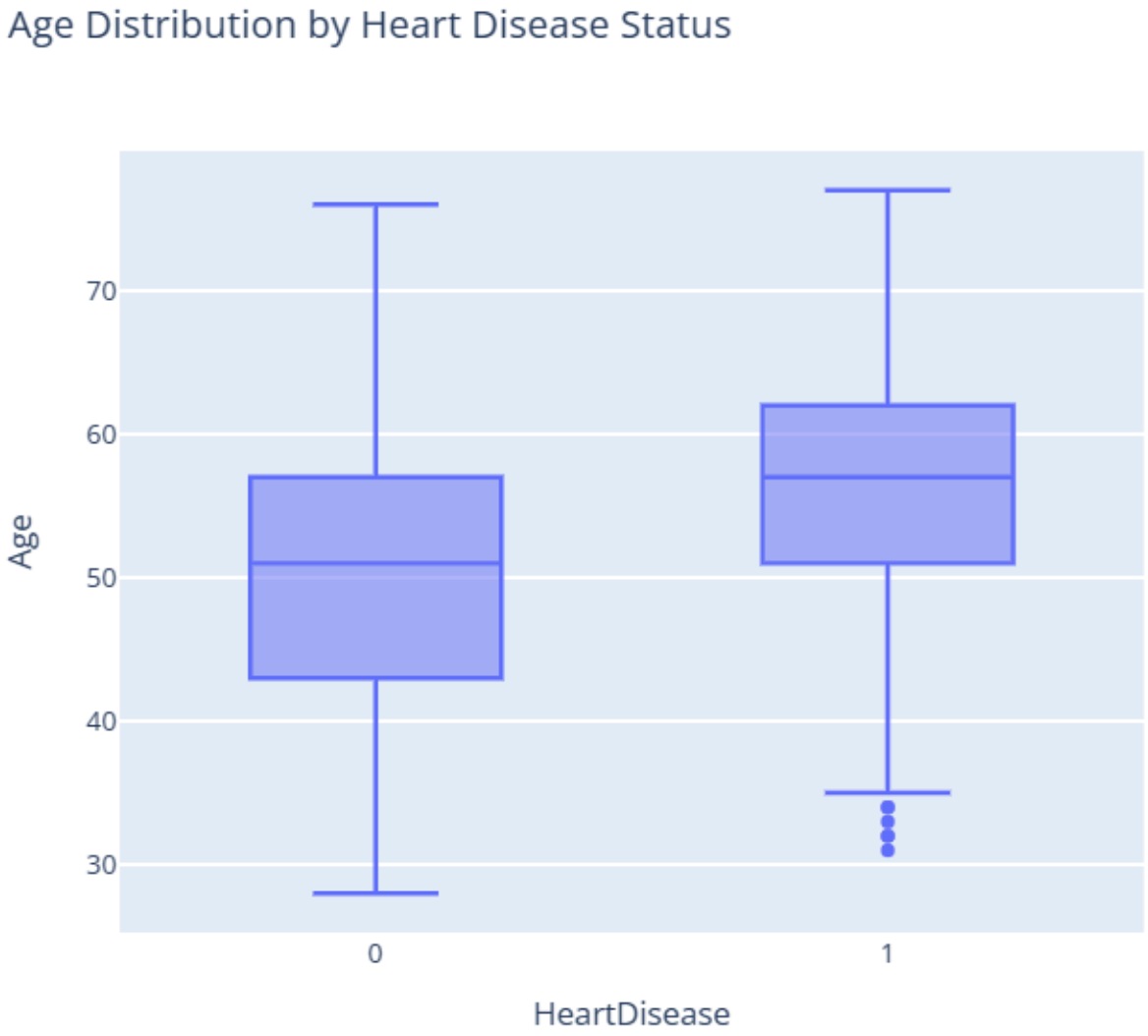


4.Gender ratio in the dataset with pie chart



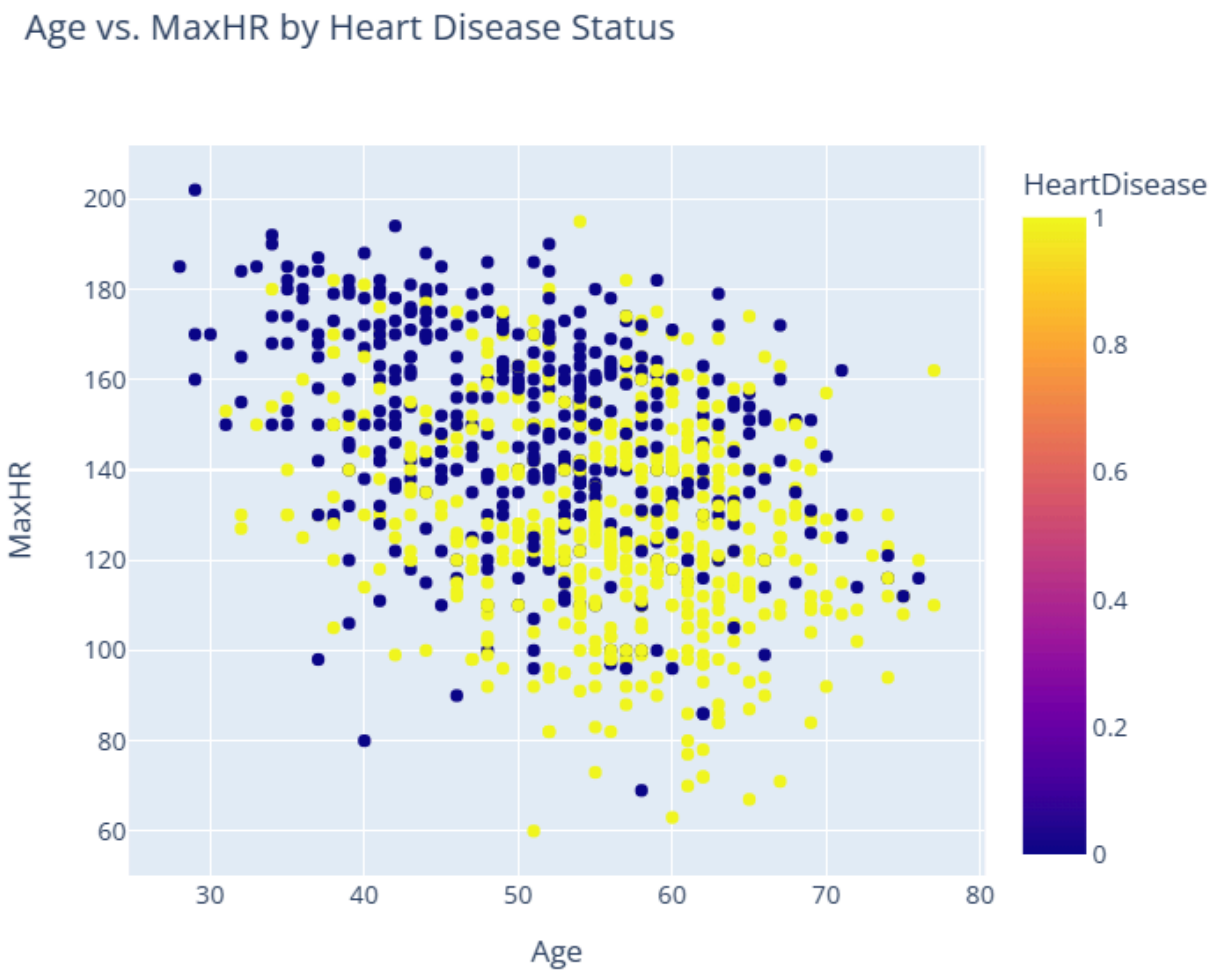
Most of the patients in the dataset are males

5. Distribution of Age for patients with and without heart disease using boxplots



From here, we can suggest that heart disease occurs fewer in younger people.

6.Relationship between Age and MaxHR with presence of HeartDisease using scatter plot



3.0 Data Preparation

3.1 Data Cleaning

3.1.1 Null Value

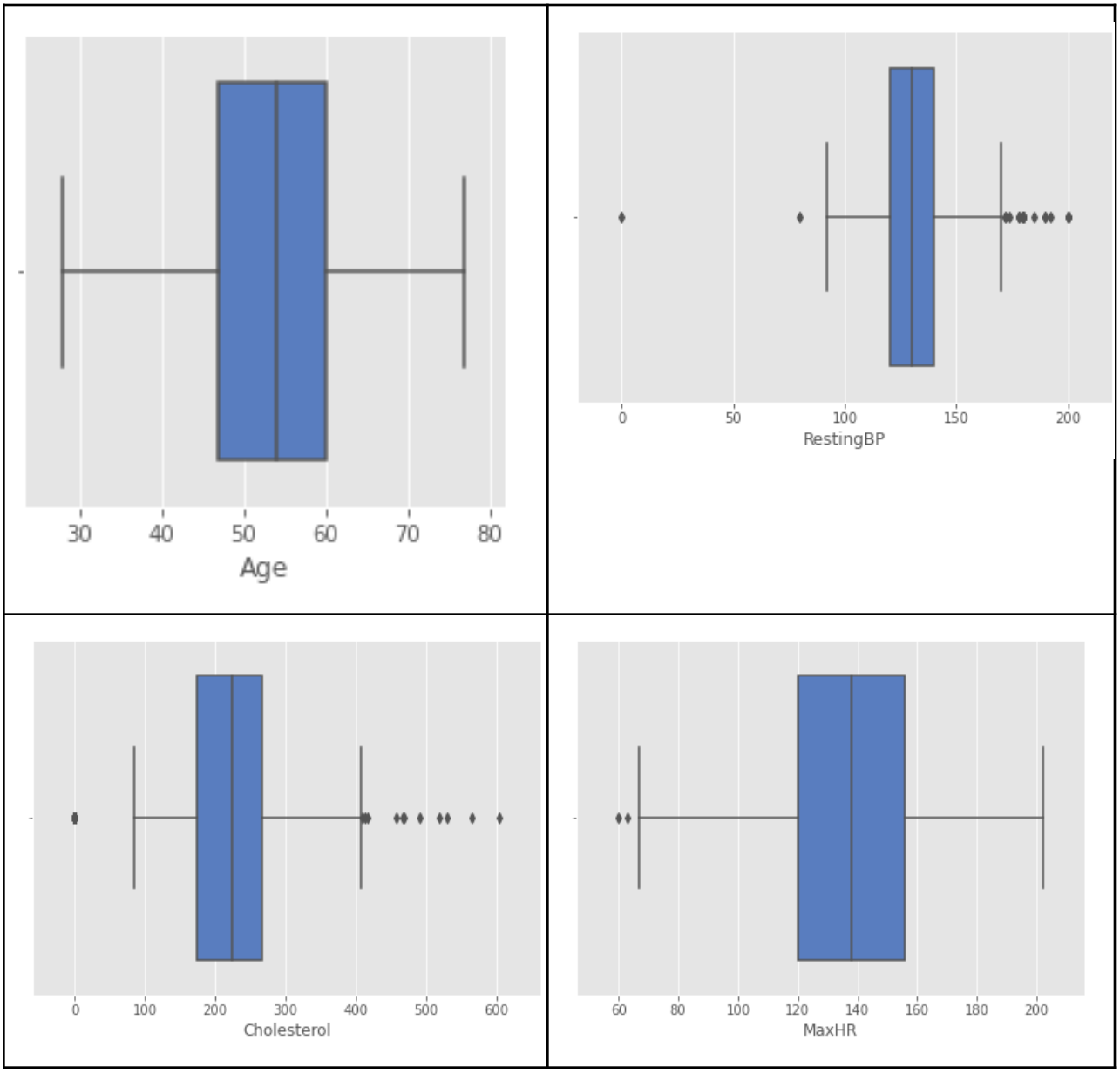
```
heart.isnull().sum()
```

```
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
```

There are no null values present in the dataset so no need to remove it.

3.2 Outliers

In the previous section, we know that numerical data are 'Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak'. Here, we will detect outliers for these numerical data using a box plot.



Outliers exist in all these numerical columns but it is not necessary for us to remove all outliers as they may contain valuable information and should not be removed. Therefore, depending on the context of heart failure and our analysis purpose, we will impute zero values with their median.

```
heart['RestingBP'].describe()
```

```
count    918.000000
mean     132.396514
std       18.514154
min        0.000000
25%      120.000000
50%      130.000000
75%      140.000000
max      200.000000
Name: RestingBP, dtype: float64
```

```
heart['Cholesterol'].describe()
```

```
count    918.000000
mean     198.799564
std      109.384145
min        0.000000
25%      173.250000
50%      223.000000
75%      267.000000
max      603.000000
Name: Cholesterol, dtype: float64
```

```
RestingBP = heart[heart['RestingBP'] == 0]
RestingBP.shape
```

```
(1, 12)
```

There is only 1 row is having the RestingBP of 0

```
Cholesterol = heart[heart['Cholesterol'] == 0]
Cholesterol.shape
```

```
(171, 12)
```

There are 171 rows having the Cholesterol of 0.

It is impossible to have 0 Cholesterol and 0 RestingBP, we impute the zeros with median using SimpleImputer

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=0, strategy='median')
imputer = imputer.fit(heart[['Cholesterol']])
heart['Cholesterol'] = imputer.transform(heart[['Cholesterol']])
```

From here, we can confirm that median value have replaced all zeros

```
heart['Cholesterol'].max
```

```
<bound method NDFrame._add_numeric_operations.<locals>.max of
1      180.0
2      283.0
3      214.0
4      195.0
5      339.0
6      237.0
7      208.0
8      207.0
9      284.0
10     211.0
11     164.0
12     204.0
13     234.0
14     211.0
15     273.0
16     196.0
17     201.0
18     248.0
```

3.3 Label Encoding to handle categorical variables

To handle categorical data including ordinal and nominal data
- One - Hot Encoding is suitable for nominal data with a small no of unique values [For working with non-tree based algortihms]
- Label Encoding is suitable for ordinal data with a small no of unique values [For working with non-tree based algortihms]

In this project, we will use both tree-based and non-tree based algorithms.
Therefore, we will apply both label encoding methods and use the encoded data set accordingly.

3.3.1 One-Hot Encoding for non-tree based algorithms

```
df_nontree=pd.get_dummies(heart,columns=string_col,drop_first=False)
df_nontree.head()
```

ChestPainType_TA	RestingECG_LVH	RestingECG_Normal	RestingECG_ST	ExerciseAngina_N	ExerciseAngina_Y	ST_Slope_Down	ST_Slope_Flat	ST_Slope_Up
0	0	1	0	1	0	0	0	1
0	0	1	0	1	0	0	1	0
0	0	0	1	1	0	0	0	1
0	0	1	0	0	1	0	1	0
0	0	1	0	1	0	0	0	1

```
# Getting the target column at the end
target="HeartDisease"
y=df_nontree[target].values
df_nontree.drop("HeartDisease",axis=1,inplace=True)
df_nontree=pd.concat([df_nontree,heart[target]],axis=1)
df_nontree.head()
```

Let column ‘HeartDisease’ go to the end

3.3.2 Label Encoding for tree based algorithms

```
df_tree = heart.copy()
encoder = LabelEncoder()
for col in string_col:
    df_tree[col] = encoder.fit_transform(df_tree[col])
df_tree.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	1	140	289.0	0	1	172	0	0.0	2	0
1	49	0	2	160	180.0	0	1	156	0	1.0	1	1
2	37	1	1	130	283.0	0	2	98	0	0.0	2	0
3	48	0	0	138	214.0	0	1	108	1	1.5	1	1
4	54	1	2	150	195.0	0	1	122	0	0.0	2	0

3.4 Data Splitting

Before we perform data modeling, we will need to split the dataset into train and test data.

Label encoded data (for tree-based algorithms)

```
# feature selection - drop our target feature (Response) - our x input
df_tree_without_target_col = np.array (df_tree.drop('HeartDisease', axis = 1))

# create our targeted feature(Response) array - our y output
tree_trainHeartDiseaseData = np.array (df_tree['HeartDisease'], dtype = 'int64')


tree_trainData = df_tree_without_target_col
```

we import the **train_test_split** function and start to split the datasets. The train set is set to 80% size whereas the test set is set to 20% size. Datasets are split into their respective train and test datasets. The **random_state** is fixed to ensure every outcome is the same.

```
X_tree = tree_trainData
y_tree = tree_trainHeartDiseaseData

from sklearn.model_selection import train_test_split

X_train_tree, X_test_tree, y_train_tree, y_test_tree = train_test_split(X_tree,
                                                                        y_tree,
                                                                        test_size = 0.2,
                                                                        random_state = 40)
#using the random state 40, we split the data into 80:20 for training:test
```

One-Hot Encoded data (for non-tree based algorithms)

```
# feature selection - drop our target feature (Response) - our x input
df_nontree_without_target_col = np.array (df_nontree.drop('HeartDisease', axis = 1))

# create our targeted feature(Response) array - our y output
nontree_trainHeartDiseaseData = np.array (df_nontree['HeartDisease'], dtype = 'int64')


nontree_trainData = df_nontree_without_target_col
```

```
X_nontree = nontree_trainData
y_nontree = nontree_trainHeartDiseaseData

from sklearn.model_selection import train_test_split

X_train_nontree, X_test_nontree, y_train_nontree, y_test_nontree = train_test_split(X_nontree,
                                                                                    y_nontree,
                                                                                    test_size = 0.2,
                                                                                    random_state = 40)
#using the random state 40, we split the data into 80:20 for training:test
```

4.0 Modeling

In this modeling stage, we will be testing the dataset with four major models:

1. KNN (K-Nearest Neighbour Classifier) *requires feature scaling*
2. SVM *requires feature scaling*
3. Decision Tree
4. Random Forest

```
#import libraries for model evaluation
from sklearn.metrics import plot_confusion_matrix,roc_auc_score, roc_curve, f1_score, accuracy_score
from sklearn.metrics import make_scorer, precision_score, precision_recall_curve, plot_precision_recall_curve
from sklearn.metrics import recall_score, plot_roc_curve

import warnings
warnings.filterwarnings('ignore')
```

```
plt.rcParams['figure.figsize'] = (14,8)
plt.rcParams['figure.facecolor'] = '#F0F8FF'
plt.rcParams['figure.titlesize'] = 'medium'
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.edgecolor'] = 'green'
plt.rcParams['figure.frameon'] = True

plt.rcParams["figure.autolayout"] = True

plt.rcParams['axes.facecolor'] = '#F5F5DC'
plt.rcParams['axes.titlesize'] = 25
plt.rcParams["axes.titleweight"] = 'normal'
plt.rcParams["axes.titlecolor"] = 'Olive'
plt.rcParams['axes.edgecolor'] = 'pink'
plt.rcParams["axes.linewidth"] = 2
plt.rcParams["axes.grid"] = True
plt.rcParams['axes.titlelocation'] = 'center'
plt.rcParams["axes.labelsize"] = 20
plt.rcParams["axes.labelpad"] = 2
plt.rcParams['axes.labelweight'] = 1
plt.rcParams["axes.labelcolor"] = 'Olive'
plt.rcParams["axes.axisbelow"] = False
plt.rcParams['axes.xmargin'] = .2
plt.rcParams["axes.ymargin"] = .2


plt.rcParams["xtick.bottom"] = True
plt.rcParams['xtick.color'] = '#A52A2A'
plt.rcParams["ytick.left"] = True
plt.rcParams['ytick.color'] = '#A52A2A'

plt.rcParams['axes.grid'] = True
plt.rcParams['grid.color'] = 'green'
plt.rcParams['grid.linestyle'] = '--'
plt.rcParams['grid.linewidth'] = .5
plt.rcParams['grid.alpha'] = .3
```

```
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['legend.facecolor'] =  'NavajoWhite'
plt.rcParams['legend.edgecolor'] = 'pink'
plt.rcParams['legend.shadow'] = True
plt.rcParams['legend.fontsize'] = 20


plt.rcParams['font.family'] = 'Lucida Calligraphy'
plt.rcParams['font.size'] = 14

plt.rcParams['figure.dpi'] = 200
plt.rcParams['figure.edgecolor'] = 'Blue'
```

Perform feature scaling for KNN and SVM models

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_X_train_nontree = scaler.fit_transform(X_train_nontree)
scaled_X_test_nontree = scaler.fit_transform(X_test_nontree)
```

The main reason feature scaling is used for KNN and SVM is to ensure all features are in the same scale, no single feature dominates the calculation of the distance. For KNN classification if the data is varied then the differences between features of the data point would be bigger which would result in the dominance of distance calculation. For example, if one feature has a range of 0 to 1 and another feature has a range of 0 to 10000. The latter would dominate in the distance calculation to address this issues feature scaling is used so every feature is contributed equally to the distance calculation. In addition as the dataset get larger the performance of KNN become worse, and the distance between the data point becomes less meaningful thus feature scaling is used so the curse of dimensionality can be addressed. SVM is an optimization-based algorithm where it would find the best hyperplane that separates the data points of different classes with maximum margin. Without the scaling feature, SVM would take a longer time to converge which would result in longer training time and poor performance. In addition, scaling each feature in the common range can increase the interpretability as each feature in a different scale is hard to reflect the important of each feature clearly.

1. KNN (K-Nearest Neighbour Classifier)

Without feature scaling

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train_nontree, y_train_nontree)
y_pred_knn = knn.predict(X_test_nontree)

knn_train = round(knn.score(X_train_nontree, y_train_nontree) * 100, 2)
knn_accuracy = round(accuracy_score(y_pred_knn, y_test_nontree) * 100, 2)
knn_f1 = round(f1_score(y_pred_knn, y_test_nontree) * 100, 2)

print("Training Accuracy     :",knn_train,"%")
print("Model Accuracy Score  :",knn_accuracy,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test_nontree,y_pred_knn))
```

```
Training Accuracy     : 81.17 %
Model Accuracy Score  : 67.39 %
----------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.61      0.61      0.61        76
           1       0.72      0.72      0.72       108

    accuracy                           0.67       184
   macro avg       0.66      0.66      0.66       184
weighted avg       0.67      0.67      0.67       184
```

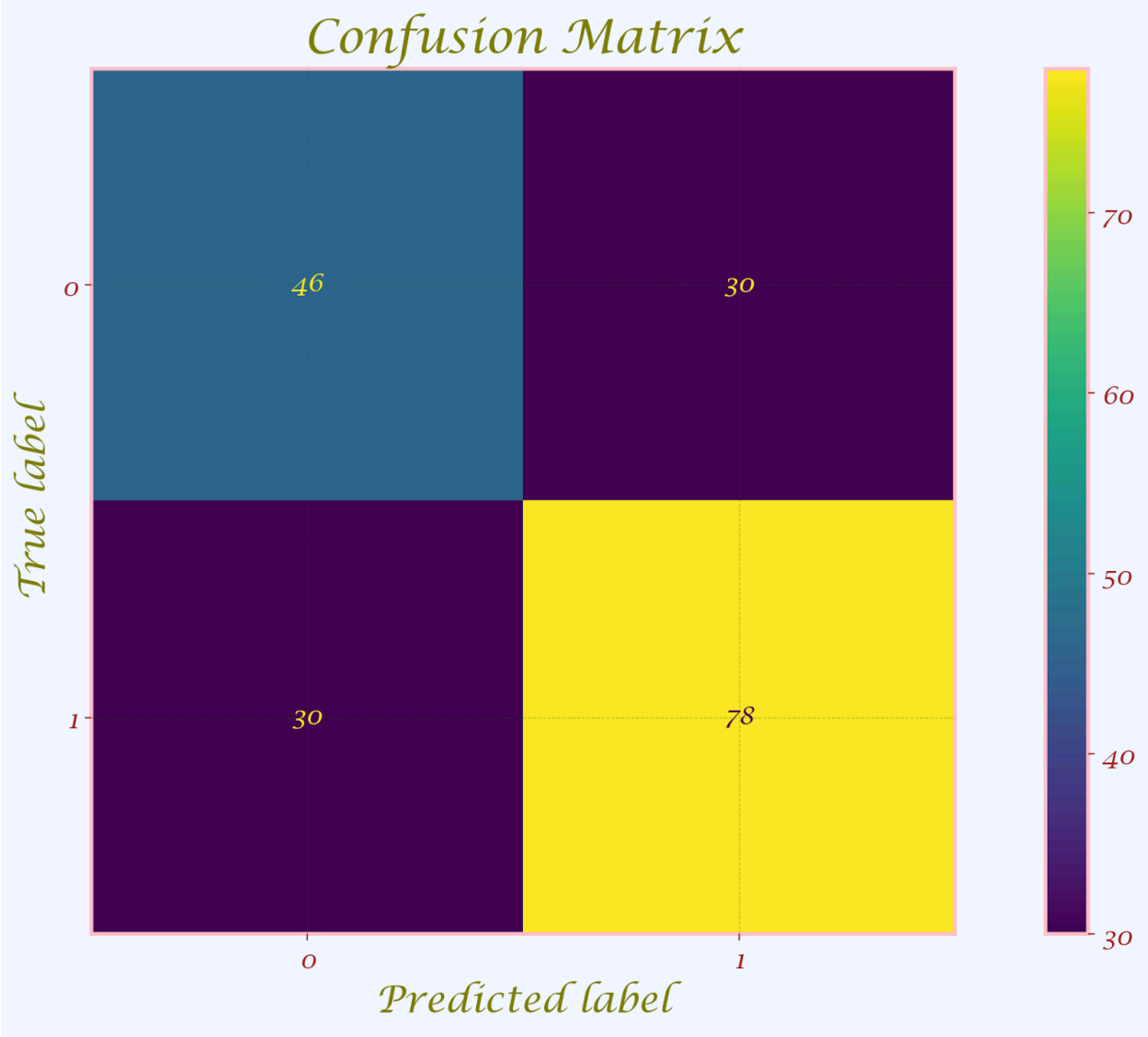
Class 0 indicates no heart disease, class 1 indicates heart disease.
Training accuracy measures how well the data fit into the training data by comparing the true label of the training data with the prediction made by the model on training data. Model accuracy means how well the model generalizes to unseen, new data. The occurrence of over lifting would be shown on the high training accuracy and a very low model accuracy score, this means that the model failed to learn the underlying pattern and relationship it just fit all the noise and memorise the data. This also indicates that the model does not generalize well to new, unseen data.
The precision means the ratio of true positives out of all positive predictions made by the model. Precision= true positive/ (true positive and false positive). The false negative indicates the model is predicted as positive but it actually is negative on the other hand true positive means that it correctly predicts it as positive. So high precision indicates the model makes an accurate true positive prediction and fewer false positive cases. In this case, the precision of 0.72 indicates that the model predicts 100 people would have heart disease and only 72 of them have heart disease. When no false positive occurred then the precision would be 1.0. Recall indicates the proportion

of actual positives which were correctly identified. 0.0 means worse possible performance and 1.0 means the model can perfectly predict all the patients with heart disease. So 0.61 recall value of the no heart disease prediction indicates patients who have heart disease are being incorrectly classified as not having heart disease by the model. This means that the model would cause patients to

not receive appropriate medical intervention or treatment that could improve their health outcomes. The F1 score is a combination of precision and recall, the perfect score in this model would be 1.0. The formula for the f1 score is 2*((precision * recall)/(precision +recall)) so for f1 score for no heart disease is 0.61 this indicates that the model is moderately accurate in predicting patients with no heart disease. So this indicates that the model is performing moderately well in recall and precision. For this model, the importance of recall value is greater compared with the precision and f1 score this is because the cost of failing to identify a patient who has heart disease is greater compared to identifying a patient with no heart disease as having heart disease. The failure to identify patients who have heart disease would cause serious consequences such as delayed or missed diagnosis which would cause the health condition worsening or even death. So in this case optimizing recall value would be more desirable compared with the F1 score and precision.

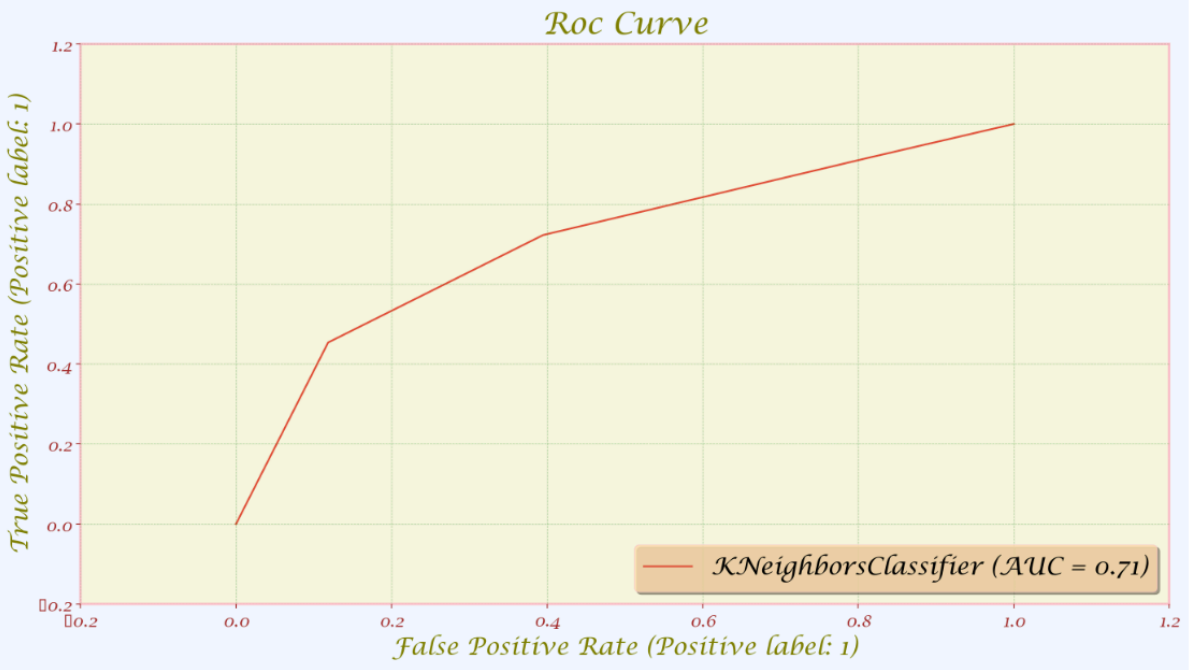
Support the mean number of datasets that belong to each class. So in this case it would calculate 76 samples with the class of 0 (no heart disease). So class imbalance didn't occur as it is relatively the same amount between class 0 and class 1. The accuracy means it calculates the ratio of correctly predicted samples in the dataset, this value would give an overall view of the performance of the model. This model is useful when the class size is roughly the same and there is no significant class imbalance. In this model 0.67 means that it accurately predicts 67 out of 100 samples. The macro avg is used to calculate the average of precision, f1 and recall across all classes in the dataset, so each class is calculated equally regardless of the size. So the average score for f1, precision and recall would be 0.66,0.66,0.66. The weighted avg is a similar concept to the macro avg but the contribution of each class to the average is proportional to its size in the dataset, so it would favour a majority class for example if one class has more samples it might outperform others and get a higher value. In this case, the weighted average for f1, precision and recall is the same which is 0.67, 0.67,0.67. Generally weighted average and macro average the higher the better however if the differences between the weighted average and macro average are too varied where the weighted average is extremely high and the macro average is extremely low this might indicate the occurrence of class imbalance because the weighted average would favour the class with more sample. In addition for the macro average, the smaller class would have a great impact on the outcome because all classes are treated equally.

```
plot_confusion_matrix(knn, X_test_nontree, y_test_nontree);
plt.title('Confusion Matrix');
```



A confusion matrix is relatively easy to understand and visualize the performance of the model. This shows the number of true positives, true negatives, false positives (Type I Error) and false negatives (Type II Error) and all this information can be used to calculate precision, recall and F1 score. Based on the confusion matrix it indicates that the occurrence of Type I Error and Type II Error is 30 for each. This is considered relatively high which means that this model is not performing well in predicting the classes. False positives would be cases of unnecessary action taken and false negatives would result in delayed action or missed opportunity.

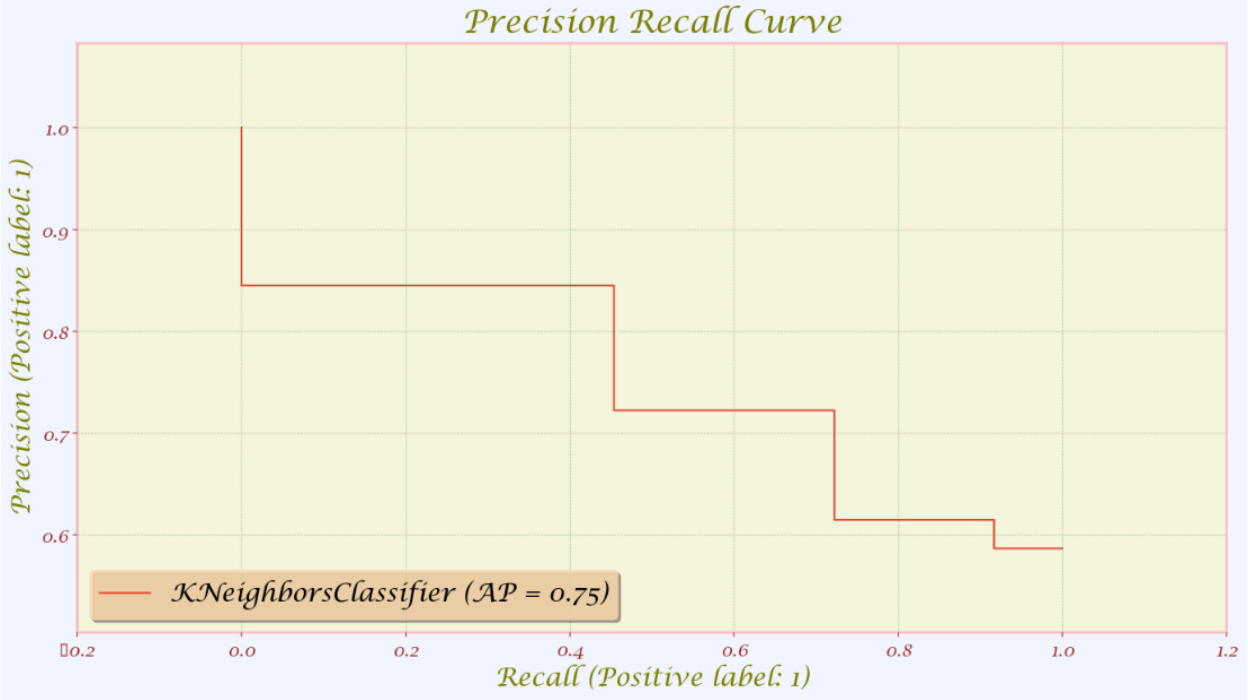
```
plot_roc_curve(knn, X_test_nontree, y_test_nontree);
plt.title('Roc Curve');
```



The ROC curve tells us how well our classifier is classifying between term deposit subscriptions (True Positives) and non-term deposit subscriptions. The X-axis is represented by False positive rates (Specificity) and the Y-axis is represented by the True Positive Rate (Sensitivity.) As the line moves the threshold of the classification changes giving us different values. The closer is the line to our top left corner the better is our model separating both classes.In summary, the ROC curve is used to evaluate binary classifiers in situations where the cost of false positives and false negatives is roughly the same and the class distribution is balanced. It allows us to compare different classifiers and select the best one based on the trade-off between TPR and FPR, and it provides a more informative view of classifier performance than simple accuracy measures.

Area under the ROC (Receiver Operating Characteristic) curve : AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease

```
plot_precision_recall_curve(knn, X_test_nontree, y_test_nontree)
plt.title('Precision Recall Curve');
```



A Precision-Recall curve is simply a graph with Precision values on the y-axis and Recall values on the x-axis. In other words, the PR curve contains TP/(TP+FN) on the y-axis and TP/(TP+FP) on the x-axis.

It is desired that the algorithm should have both high precision, and high recall. However, most machine learning algorithms often involve a trade-off between the two. A good PR curve has greater AUC (area under curve).In summary, precision is used in the PR curve because it is a useful metric for evaluating models in situations where the cost of false positives is high or when the class distribution is imbalanced. PR curves provide a more nuanced view of a model's performance, particularly in situations where high precision is important.

The precision recall curve is considered as relatively bad because it does not go from top left corner horizontally to top right corner and straight down to right bottom corner. This indicate that the model has failed to achieve high precision with high recall rate, it also mean that the model is failed to make positive predictions while correctly identify the most of the positive instances.

```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=knn, X = X_train_nontree, y=y_train_nontree, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 65.62 %
Std. Dev: 4.03 %
```

The model accuracy score is 65.62% which means that the predict is correct for approximately 2 out of 3 every sample. Standard deviation is used to test the reliability of the model, if the standard deviation is low means that the accuracy score is tightly clustered around the mean. So it indicates the performance of the model is more consistent, and the performance is less likely to differ greatly from each other. In this case a standard deviation of 4.03% indicates that the accuracy is slightly different from run to run but does not fluctuate too much until the model becomes unreliable or unstable.

With feature scaling

```
# from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(scaled_X_train_nontree, y_train_nontree)
y_pred_knn = knn.predict(scaled_X_test_nontree)

knn_train = round(knn.score(scaled_X_train_nontree, y_train_nontree) * 100, 2)
knn_accuracy = round(accuracy_score(y_pred_knn, y_test_nontree) * 100, 2)
knn_f1 = round(f1_score(y_pred_knn, y_test_nontree) * 100, 2)

print("Training Accuracy     :",knn_train,"%")
print("Model Accuracy Score  :",knn_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test_nontree,y_pred_knn))
```

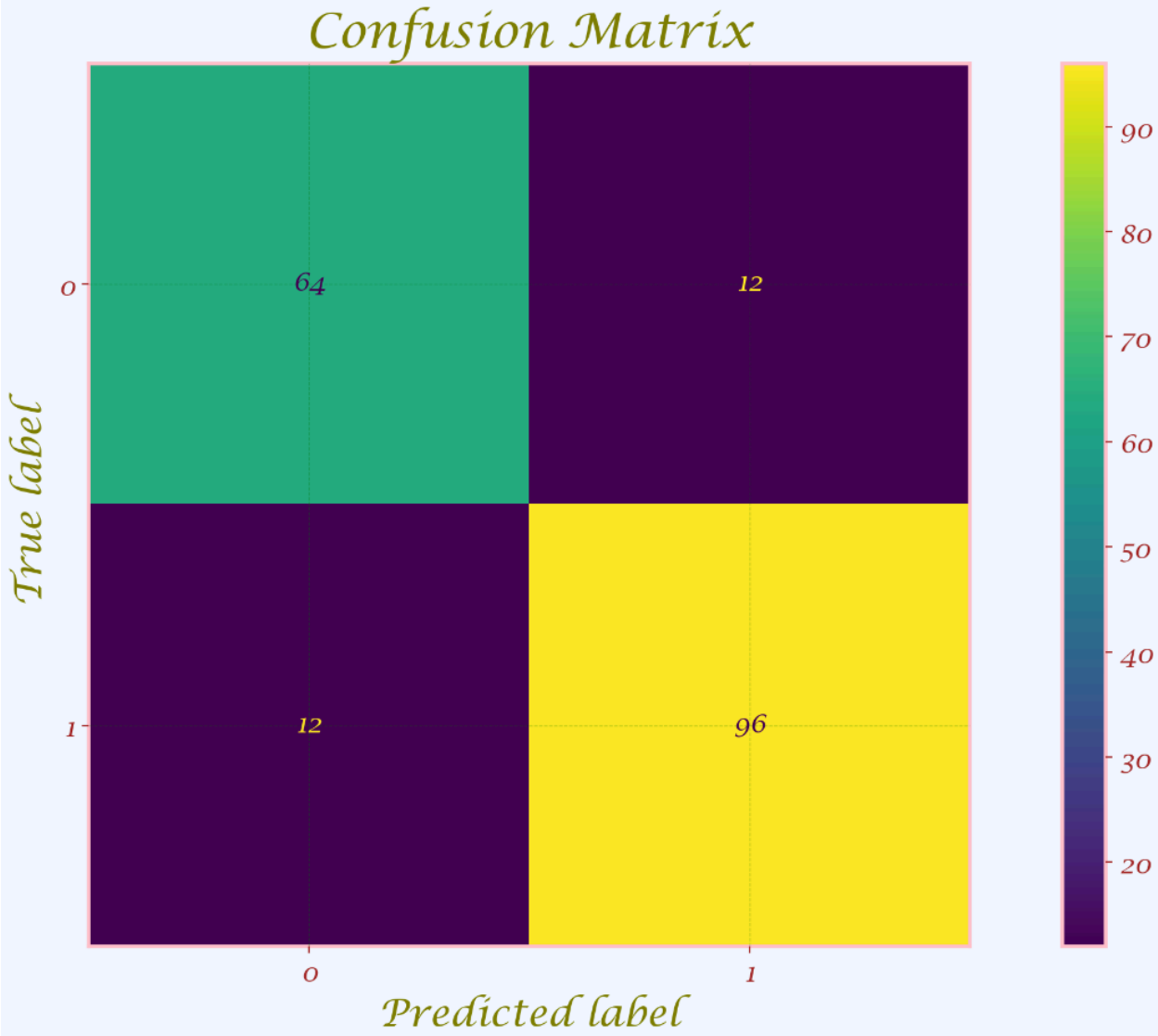
```
Training Accuracy     : 90.59 %
Model Accuracy Score  : 86.96 %
-------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.84      0.84      0.84        76
           1       0.89      0.89      0.89       108

    accuracy                           0.87       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.87      0.87      0.87       184
```

The training accuracy and model accuracy score is relatively the same indicating that it is not over-lifting or under-lifting, it is generalized well with new and unseen data. In addition, the high value in training accuracy and model accuracy score indicates the model performs well in many scenarios.

The precision in this case is 0.89 for heart disease. It indicates when the model predicts a patient has heart disease it is most likely to be correct, it indicates that 89% of the model prediction to be positive is actually true positive only 11% is false positive. the 0.89 recall value for heart disease prediction indicates this model is good at identifying a patient who actually has heart disease, it means that out of all the heart disease patients, this model is able to identify 89% of them. The f1-score of 0.89 indicates that this model is performing well in both precision and recall, it achieves a good balance of precision and recall with minimizing the false positive and false negative. This model is able to accurately identify a patient who has heart disease so necessary medical treatment is able to be provided to the patient, this also indicates this model is useful in real-life applications. This model has a 0.87 accurate prediction meaning it performs relatively well in classification accuracy, it is able to correctly predict 87% of both positive and negative heart disease, and it is only 13% away from the best model prediction. The macro average and

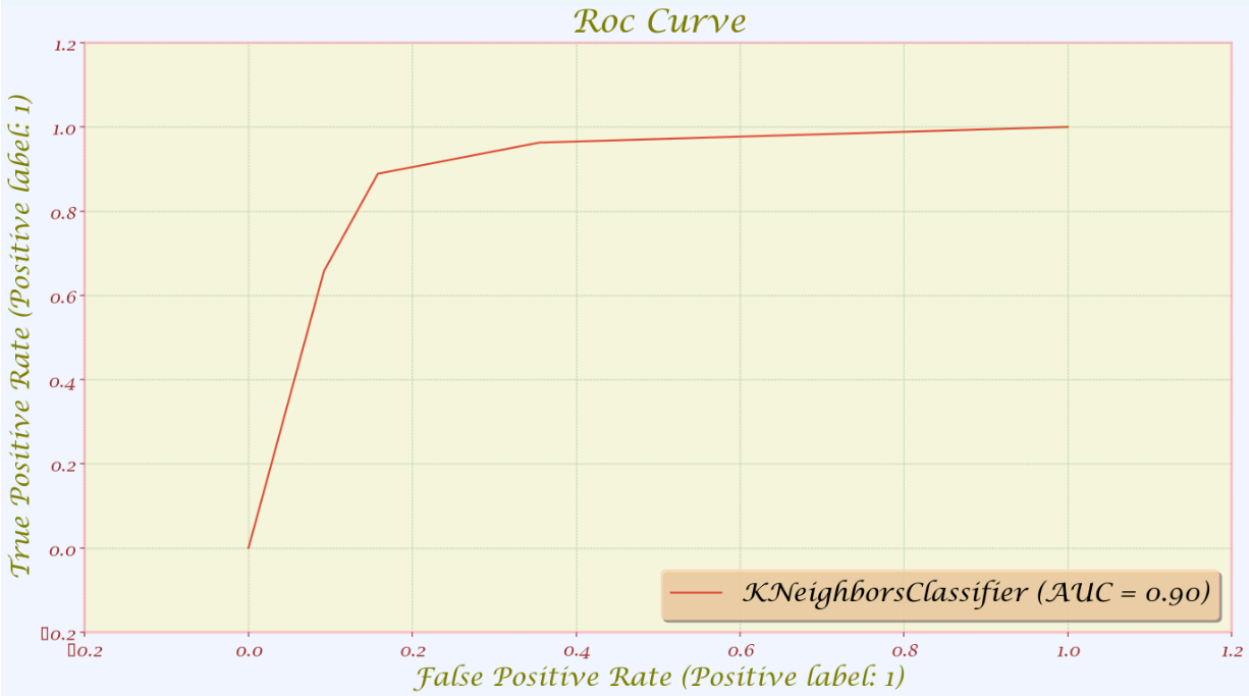
weighted avg of 0.87 indicate it performs relatively well across all classes, and it is able to correctly predict both positive and negative heart disease. It is able to predict the absence or presence of heart disease in a relatively high percentage of cases; this indicates this model is able to perform well in making heart disease predictions.

```
plot_confusion_matrix(knn, scaled_X_test_nontree, y_test_nontree);
plt.title('Confusion Matrix');
```



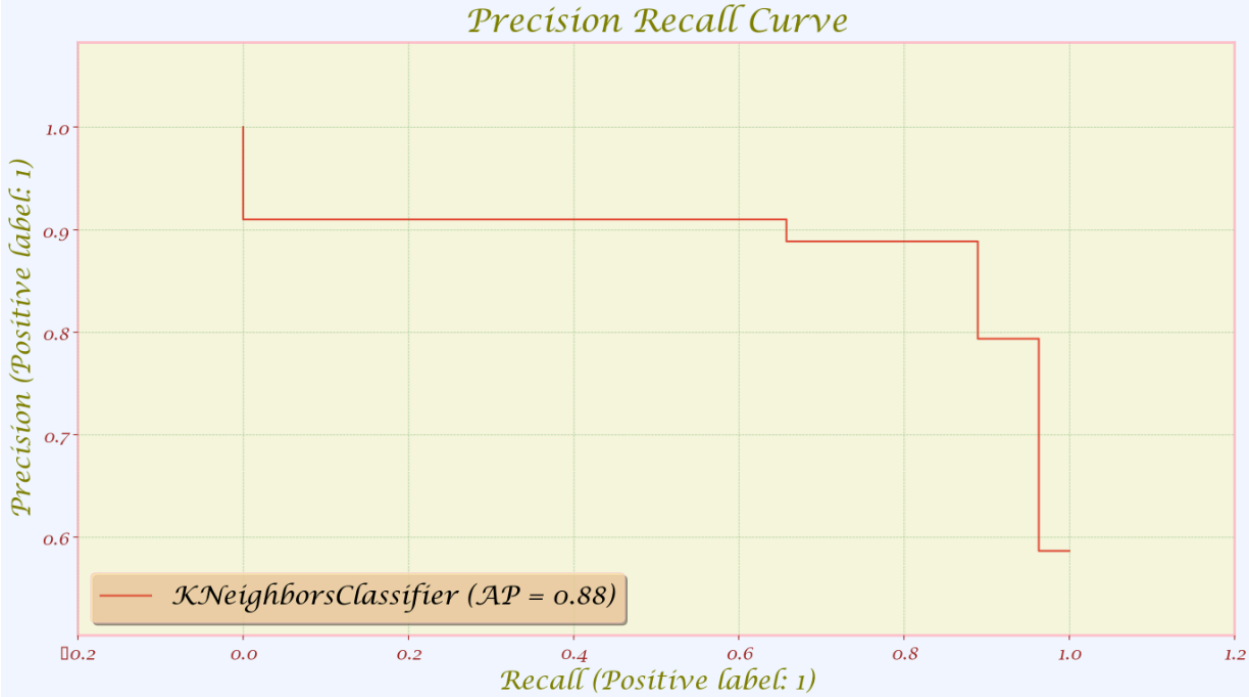
Based on the confusion matrix, the occurrence of false positives (Type I Error) and false negatives (Type II Error) is reduced greatly which dropped from 30 to 12. This indicates feature scaling make this model more reliable and accurate and it helps to reduce the occurrence of false negative and false positives.

```
plot_roc_curve(knn, scaled_X_test_nontree, y_test_nontree);
plt.title('Roc Curve');
```



As the line is moving more toward the top left corner this indicate that feature scaling help the model to separate both classes with and without heart disease.

```
plot_precision_recall_curve(knn, scaled_X_test_nontree, y_test_nontree)
plt.title('Precision Recall Curve');
```



This precision recall curve is considered as relatively good because it goes from top left corner horizontally to top right corner and straight down to right bottom corner. This indicates that the model is able to achieve high precision with high recall rate, it also means that the model is able to make positive prediction while correctly identify the most of the positive instances this also mean that the feature are able to increase the high precision without sacrifice too much recall rate.

```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=knn, X = scaled_X_train_nontree, y=y_train_nontree, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 83.91 %
Std. Dev: 3.78 %
```

When the data is modelled with feature scaling applied for KNN, it shows a better accuracy percentage. The model accuracy score of modelling with feature scaling(83.91 %) is much higher than modelling without feature scaling(65.62 %) and the standard deviation is decreased from 4.03% to 3.78% indicating that feature scaling is able to help me model to be more consistent and stable and the performance become less varies from run to run.

2. SVM

Without feature scaling

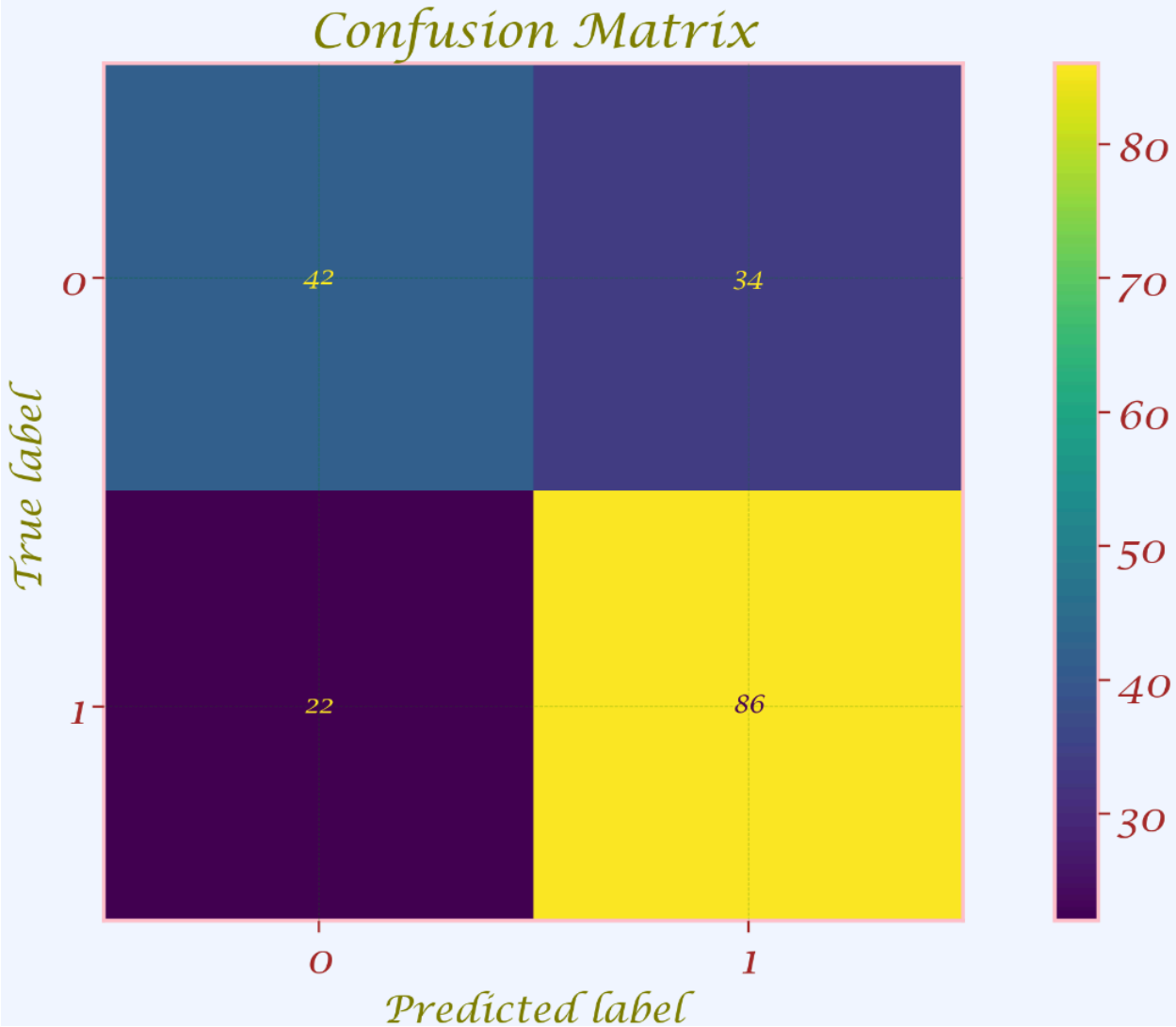
```
# Support Vector Machines
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train_nontree, y_train_nontree)
y_pred_svc = svc.predict(X_test_nontree)

svc_train = round(svc.score(X_train_nontree, y_train_nontree) * 100, 2)
svc_accuracy = round(accuracy_score(y_pred_svc, y_test_nontree) * 100, 2)
svc_f1 = round(f1_score(y_pred_svc, y_test_nontree) * 100, 2)

print("Training Accuracy     :",svc_train,"%")
print("Model Accuracy Score  :",svc_accuracy,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test_nontree,y_pred_svc))
```

```
Training Accuracy     : 69.03 %
Model Accuracy Score  : 69.57 %
----------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.66      0.55      0.60        76
           1       0.72      0.80      0.75       108

    accuracy                           0.70       184
   macro avg       0.69      0.67      0.68       184
weighted avg       0.69      0.70      0.69       184
```

The training accuracy and model accuracy score is relatively the same indicating that it is not over-lifting or under-lifting, it is generalized well with new and unseen data. However, this model prediction is correct for roughly 2 out of 3 every sample. This indicates that this model is somewhat accurate only. The precision for this model is 0.66 for no heart disease this means that 66% of the prediction of the patient who has no heart disease only 66% is correct. This indicates that this model has a relatively high false positive rate. This means it incorrectly identifies some patients with heart disease as healthy. The recall value of 0.55 means that the model is only able to identify 55% of the actual case of no heart disease. This indicates the model produces a relatively high false negative rate for the no heart disease class, so many people with no heart disease are wrongly classified as having heart disease. The f1- score for no heart disease indicates it performs moderately well for both true positive and true negative. The support for class no heart disease 0 and with heart disease 1 is roughly the same thus the class imbalance

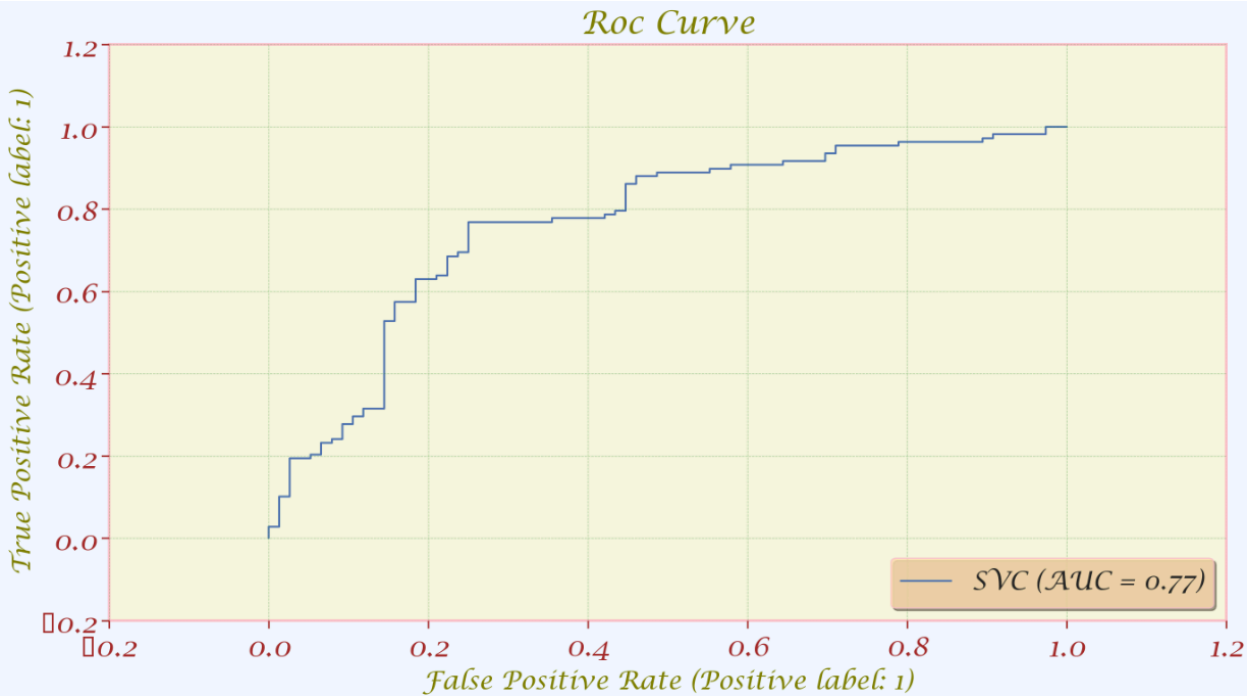
does not occur in this case. The accuracy is 0.7 meaning that this model correctly predicts 70% for both patients with and without heart disease. The macro average and weighted avg performance in this model is relatively good both with and without heart disease. As the class imbalance does not occur so the value of the macro average and weighted avg is not varied, if class imbalance occurred generally it would cause a high value in the weighted avg and a lower in the macro avg as the weighted avg is more favorable to a class with more samples. The performance of this model is moderately accurate.

```
plot_confusion_matrix(svc, X_test_nontree, y_test_nontree);
plt.title('Confusion Matrix');
```



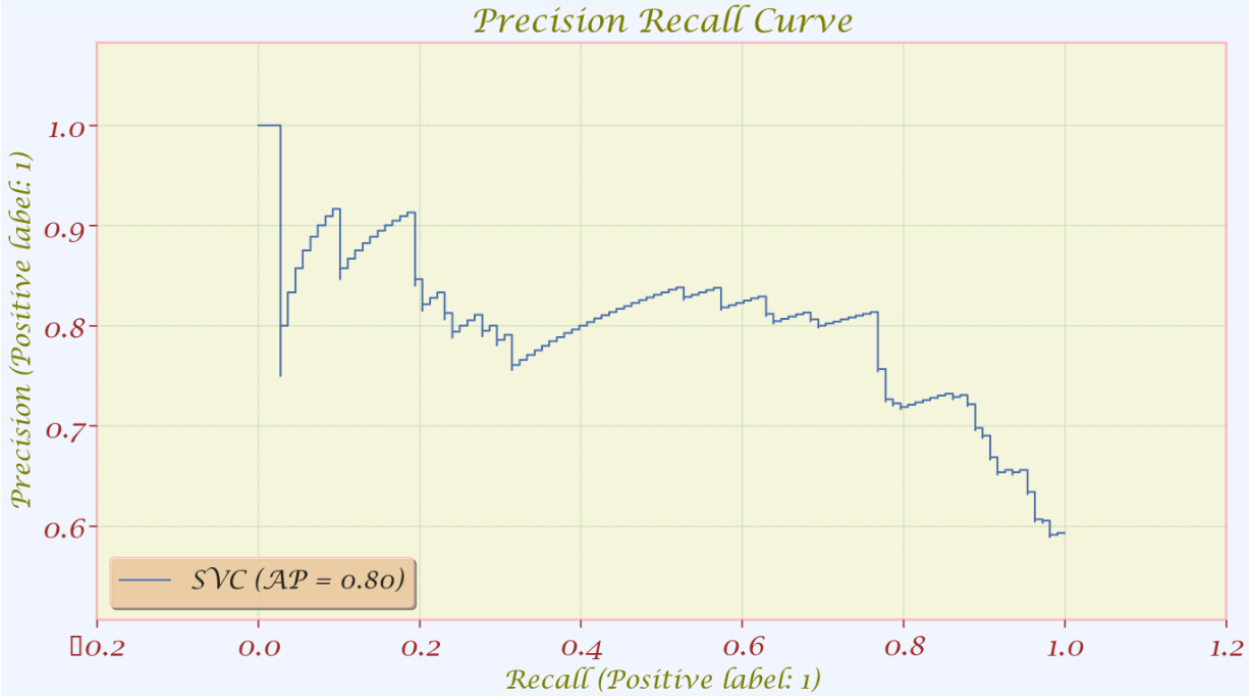
Based on the confusion matrix it indicates that the occurrence of Type I Error and Type II Error is
34 and 22. This indicate that this model is not performing well, type I error occurred quite often this indicate that this model is making a lot of false positive prediction where the result indicate it have heart disease but it actually don't have it.

```
plot_roc_curve(svc, X_test_nontree, y_test_nontree);
plt.title('Roc Curve');
```



As this model is perform moderately good at separating classes which have heart disease and without heart disease.

```
plot_precision_recall_curve(svc,  X_test_nontree, y_test_nontree)
plt.title('Precision Recall Curve');
```



This precision recall curve is considered as relatively bad because it does not goes from top left corner horizontally to top right corner and straight down to right bottom corner. This indicare that the model is failed to achieve high precision without sacrificing recall rate.

```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=svc, X = X_train_nontree, y=y_train_nontree, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 68.76 %
Std. Dev: 3.54 %
```

The model accuracy score is 68.76% this means that this model is roughly accurate to predict patient with and without heart disease. The standard deviation of 3.54 % indicates that this model prediction is only slightly different from run to run.

With feature scaling

```
# Support Vector Machines
from sklearn.svm import SVC
svc = SVC()
svc.fit(scaled_X_train_nontree, y_train_nontree)
y_pred_svc = svc.predict(scaled_X_test_nontree)

svc_train = round(svc.score(scaled_X_train_nontree, y_train_nontree) * 100, 2)
svc_accuracy = round(accuracy_score(y_pred_svc, y_test_nontree) * 100, 2)
svc_f1 = round(f1_score(y_pred_svc, y_test_nontree) * 100, 2)

print("Training Accuracy     :",svc_train,"%")
print("Model Accuracy Score  :",svc_accuracy,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test_nontree,y_pred_svc))
```

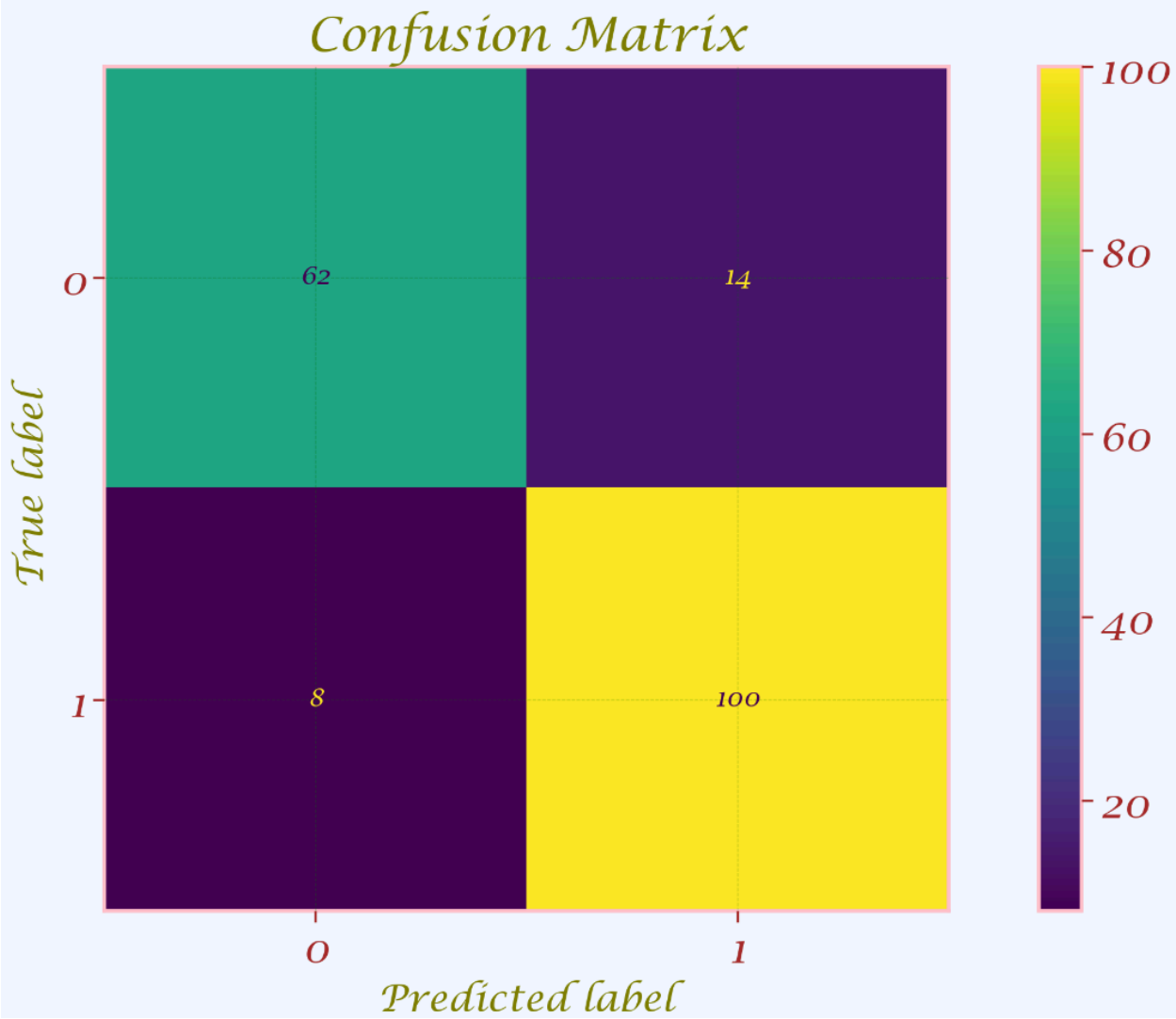
```
Training Accuracy     : 90.31 %
Model Accuracy Score  : 88.04 %
----------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.89      0.82      0.85        76
           1       0.88      0.93      0.90       108

    accuracy                           0.88       184
   macro avg       0.88      0.87      0.88       184
weighted avg       0.88      0.88      0.88       184
```

The training accuracy and model accuracy score is relatively the same indicating that it is not over-lifting or under-lifting, it is generalized well with new and unseen data. In addition, the high value in training accuracy and model accuracy score indicates the model performs well in many scenarios.

The precision in this case is 0.89 for heart disease. It indicates when the model predicts a patient has heart disease it is most likely to be correct, it indicates that 89% of the model prediction to be positive is actually true positive only 11% is false positive. 0.93 recall value for heart disease prediction indicates this model is good at identifying a patient who actually has heart disease, it means that out of all the heart disease patients, this model is able to identify 93% of them. The f1-score of 0.90 indicates that this model is performing well in both precision and recall, it achieves a good balance of precision and recall with minimizing the false positive and false negative. This model is able to mostly correctly identify the patient with heart disease this indicates that this model is useful in real life application. The accuracy of 0.88 means that this model is able to correctly identify patients with and without heart disease with 88% accuracy.

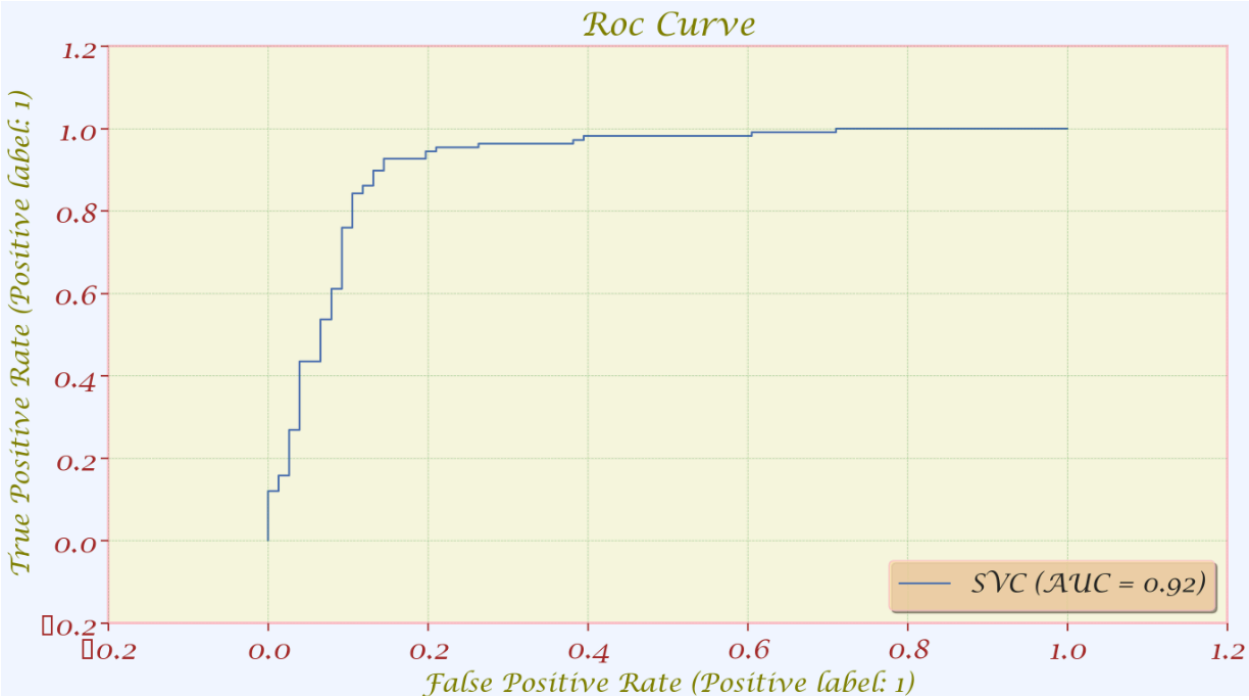
This model is only 12% away from the perfect model. The high macro and weighted avg indicate that this model is performing well across all classes, it is able to predict the patient with and without heart disease correctly. The model is able to get a high value in macro average and weighted avg indicate this model are able to predict the absence and present of heart disease correctly.

```
plot_confusion_matrix(svc, scaled_X_test_nontree, y_test_nontree);
plt.title('Confusion Matrix');
```



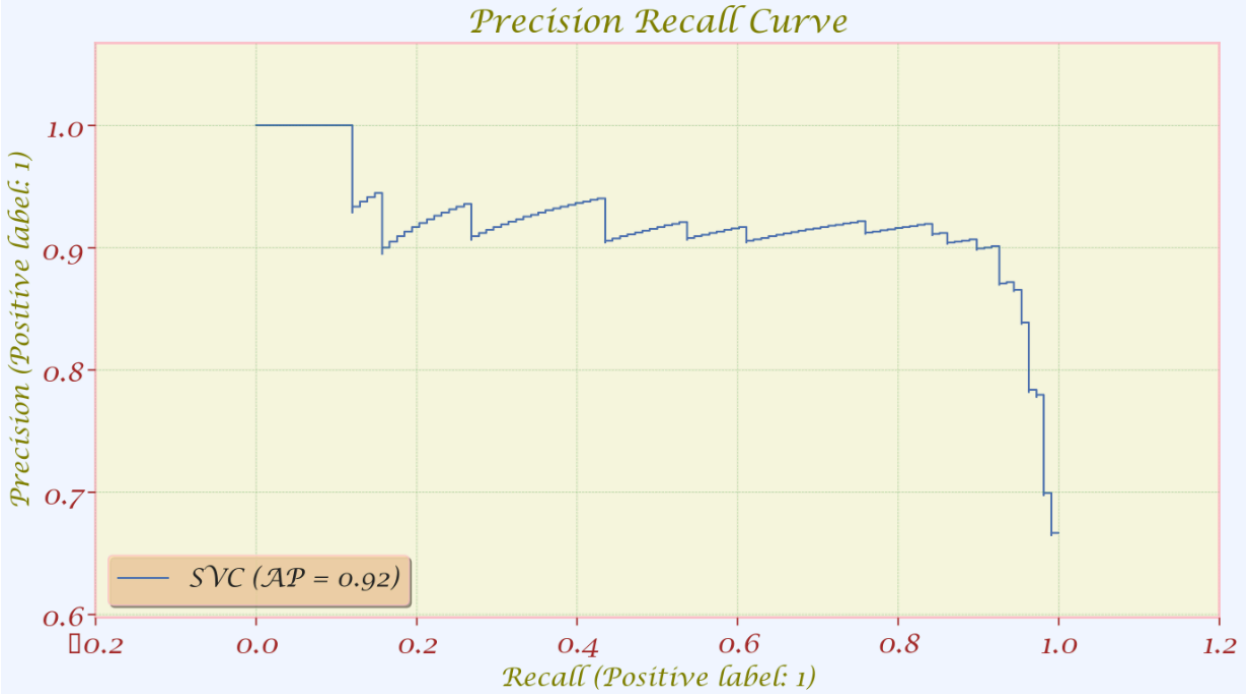
The type I error is reduced from 34 to 14 and type II error is able to reduced from 22 to 8 this indicates that feature scaling are able to reduce the occurrence of false positive and false negative.

```
plot_roc_curve(svc, scaled_X_test_nontree, y_test_nontree);
plt.title('Roc Curve');
```



The line is moving toward the top left corner more after the feature scaling is used this indicate that feature scaling are able to reduce the occurrence of false positive cases.

```
plot_precision_recall_curve(svc,  scaled_X_test_nontree, y_test_nontree)
plt.title('Precision Recall Curve');
```



This precision recall curve is considered as relatively good because it goes from top left corner horizontally to top right corner and straight down to right bottom corner. This indicate that the model is able to achieve high precision with high recall rate, it also mean that the model is able to make positive prediction while correctly identify the most of the positive instances.

```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=svc, X = scaled_X_train_nontree, y=y_train_nontree, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 84.73 %
Std. Dev: 2.59 %
```

It is very obvious that feature scaling is necessary for Support Vector Machines as the model accuracy score of modeling with feature scaling(84.73 %) is much higher than modeling without feature scaling(68.76 %). The standard deviation also dropped from 3.54 % to 2.59 % this indicates that feature scaling is able to make the model performance to become more consistent.

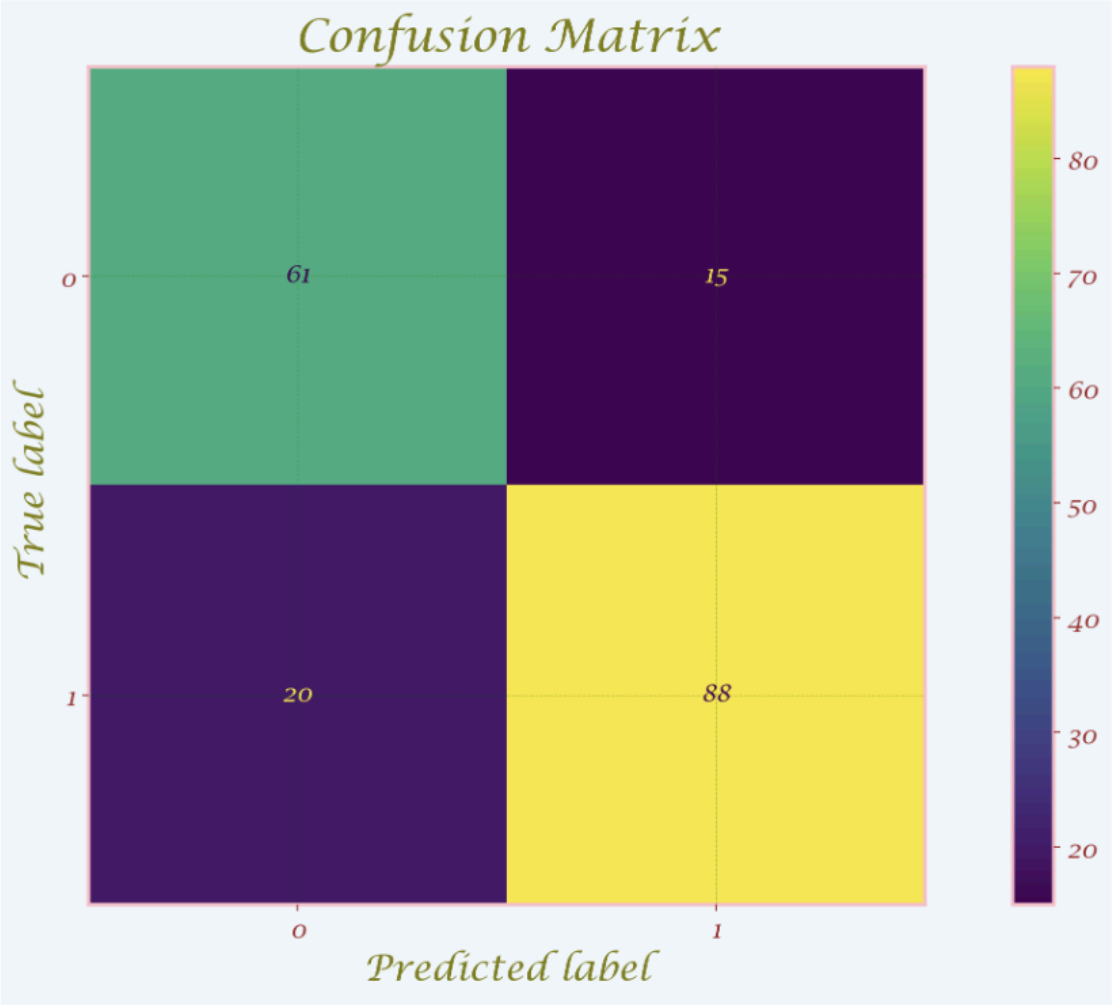
3. Decision Tree

In [430]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision = DecisionTreeClassifier()
decision.fit(X_train_tree, y_train_tree)
y_pred_Decision = decision.predict(X_test_tree)

decision_train = round(decision.score(X_train_tree, y_train_tree) * 100, 2)
decision_accuracy = round(accuracy_score(y_pred_Decision, y_test_tree) * 100, 2)
decision_f1 = round(f1_score(y_pred_Decision, y_test_tree) * 100, 2)

print("Training Accuracy     :",decision_train,"%")
print("Model Accuracy Score  :",decision_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test_tree,y_pred_Decision))
```

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 80.98 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.75      0.80      0.78        76
           1       0.85      0.81      0.83       108

    accuracy                           0.81       184
   macro avg       0.80      0.81      0.81       184
weighted avg       0.81      0.81      0.81       184
```

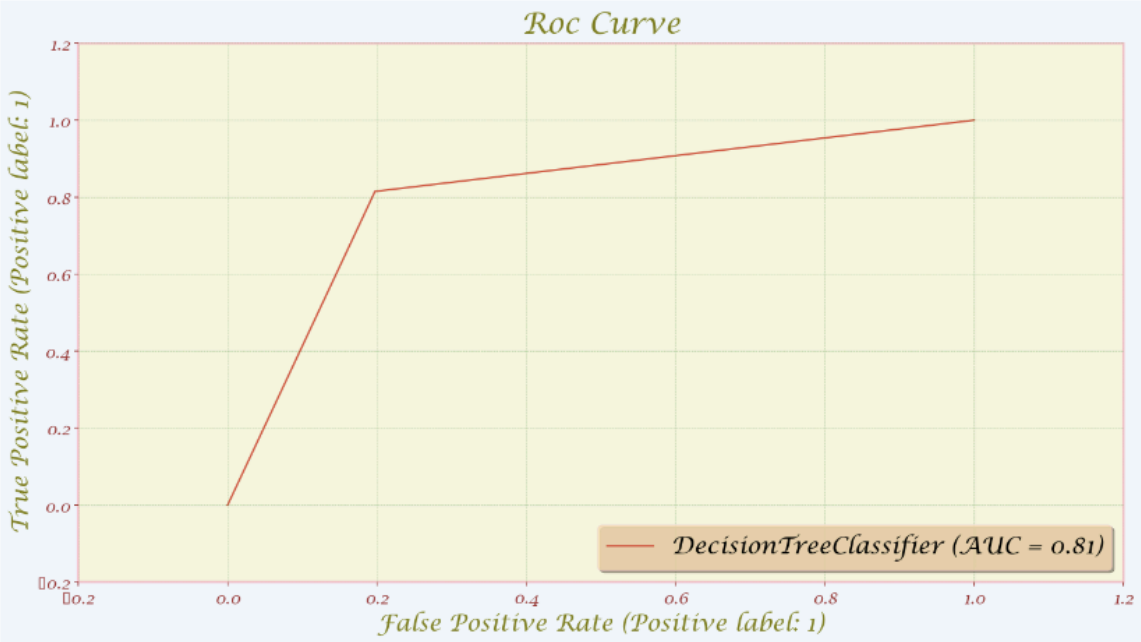
The training accuracy of 100% and model accuracy score of 80.98% may indicate the occurrence of over lifting where the model fits the training data too closely as a result it performs poorly on unseen or new data. The precision of this model for heart disease is 0.85 which indicates that 85% of the model prediction is true positive and only 15 % is false positive. This indicates that this model is able to predict heart disease mostly accurately. The recall value for this model of heart disease is 0.81 this indicates the false negative occurrence is relatively low and this model is able to predict the heart disease at a relatively high accuracy. The f1-score in this model is 0.83 which indicates that the overall performance of the heart disease in precision and recall is relatively good. The macro average and weighted average is high this indicates the model are able to correctly identify patient with and without heart disease correctly.

In [431]:
```
plot_confusion_matrix(decision, X_test_tree, y_test_tree);
plt.title('Confusion Matrix');
```



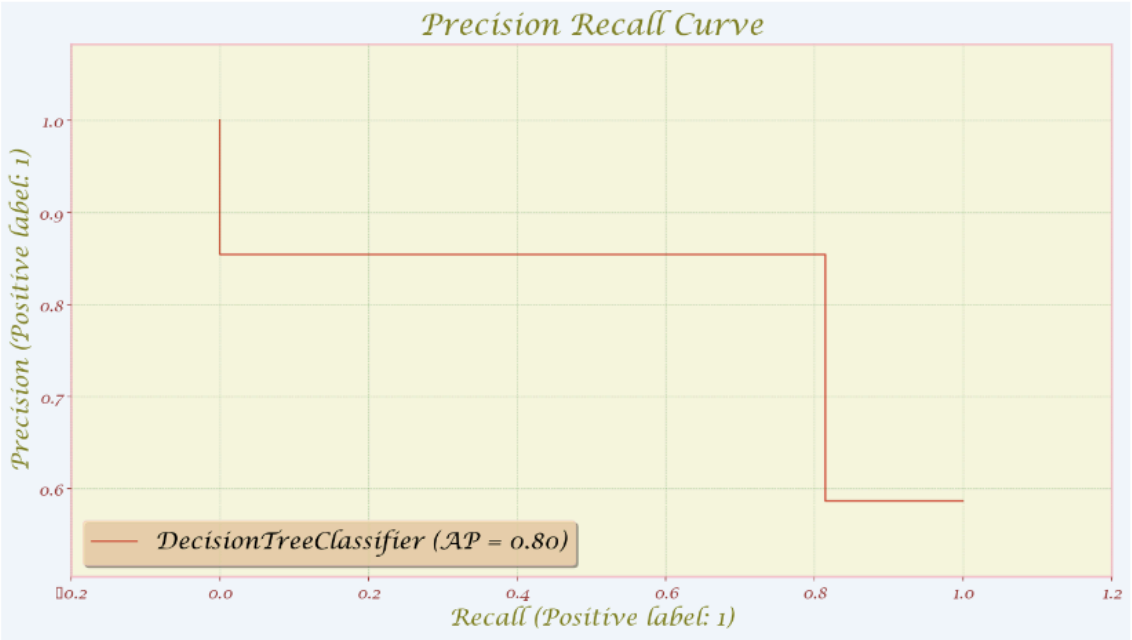
Confusion matrix for error I and Error II is relatively low, meaning that this model generate less error and able to perform well in predict heart disease.

```
In [432]: plot_roc_curve(decision, X_test_tree, y_test_tree);
          plt.title('Roc Curve');
```



The false positive rate is relatively low and the true positive is high in this case, this mean the model generates less false positives and is able to predict correctly.

```
In [433]: plot_precision_recall_curve(decision, X_test_tree, y_test_tree)
          plt.title('Precision Recall Curve');
```



This precision recall curve is considered as relatively good because it goes from top left corner horizontally to top right corner and straight down to right bottom corner. This indicates that the model is able to achieve high precision with high recall rate, it also mean that the model is able to make positive prediction while correctly identify the most of the positive instances.

```
In [434]: from sklearn.model_selection import cross_val_score
          val_score = cross_val_score(estimator=decision, X = X_train_tree, y=y_train_tree, cv=10)
          print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
          print("Std. Dev: {:.2f} %".format(val_score.std()*100))

          Model Accuracy Score: 79.11 %
          Std. Dev: 5.40 %
```

The model accuracy is 79.11% meaning that this model is able to predict the patient with and without heart disease at a relatively high rate. The standard deviation of 5.40% indicates that this model performance is less consistent and the performance is more likely to differ from each other greatly. Although the standard deviation is relatively high, the standard deviation is considered as acceptable where it does not fluctuate too much until the models become unreliable.

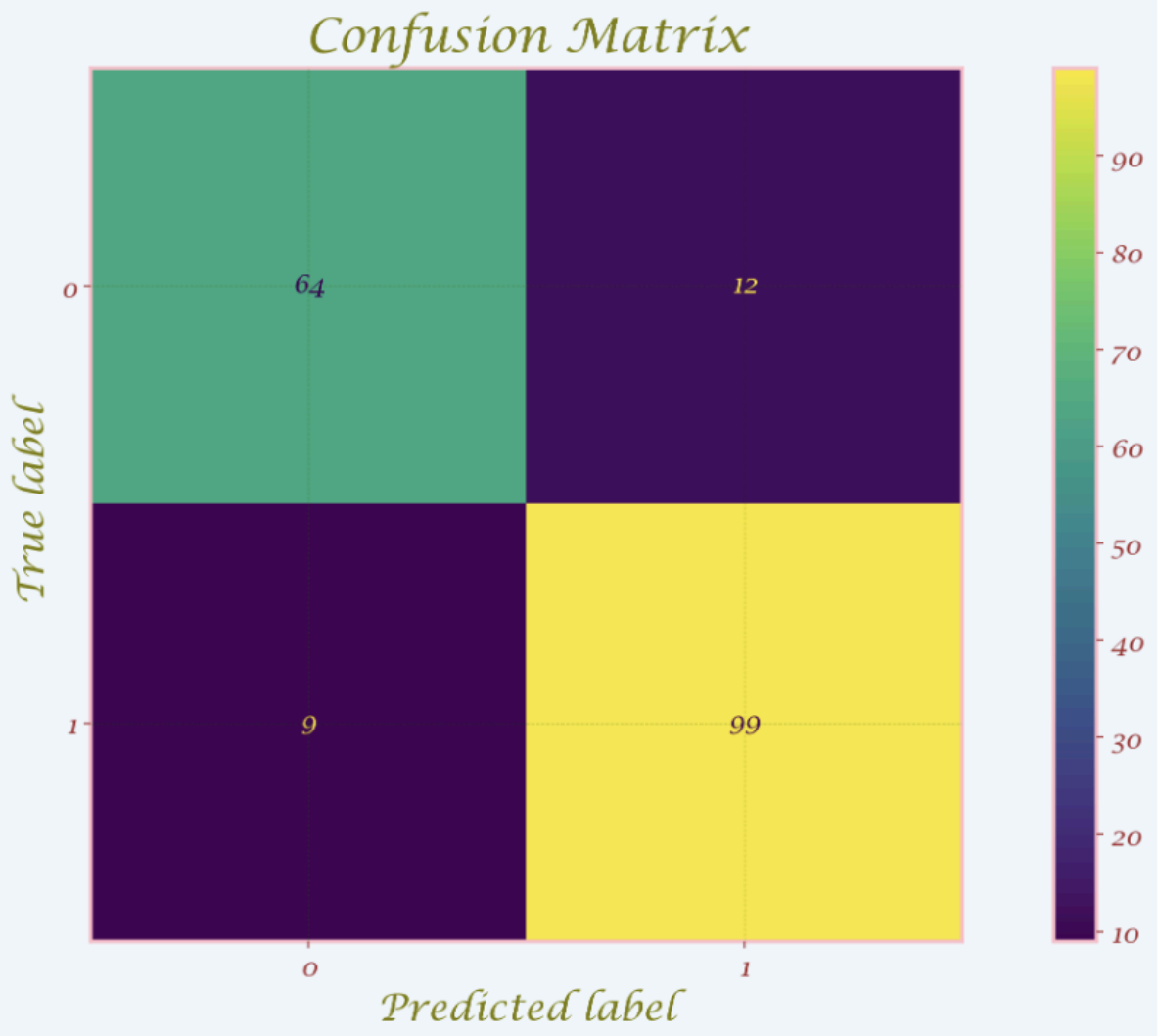
4.Random Forest

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train_tree, y_train_tree)
y_pred_random = random_forest.predict(X_test_tree)
random_forest.score(X_train_tree, y_train_tree)

random_forest_train = round(random_forest.score(X_train_tree, y_train_tree) * 100, 2)
random_forest_accuracy = round(accuracy_score(y_pred_random, y_test_tree) * 100, 2)
random_forest_f1 = round(f1_score(y_pred_random, y_test_tree) * 100, 2)

print("Training Accuracy     :",random_forest_train,"%")
print("Model Accuracy Score  :",random_forest_accuracy,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test_tree,y_pred_random))
```

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 88.59 %
----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.88      0.84      0.86        76
           1       0.89      0.92      0.90       108

    accuracy                           0.89       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.89      0.89      0.89       184
```

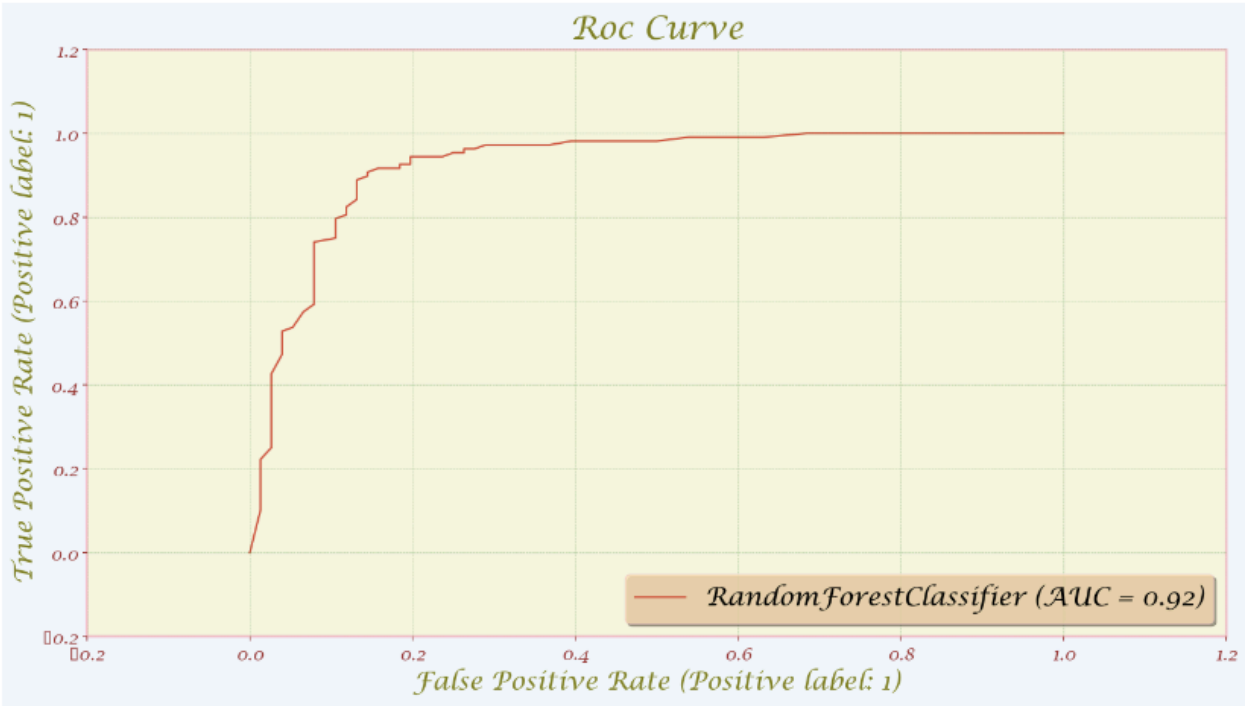
The model accuracy score of 88.59% indicates the model is able to deal with unforeseen and new data. The precision, recall, f-1 score is performed relatively well, this indicates that the overall performance of precision and recall is good, the high value in precision and recall score indicate that the occurrence of false positive and false negative is low in this model. The macro average and weighted average is high this indicate the model are able to correctly identify patient with and without heart disease correctly.

```
plot_confusion_matrix(random_forest, X_test_tree, y_test_tree);
plt.title('Confusion Matrix');
```



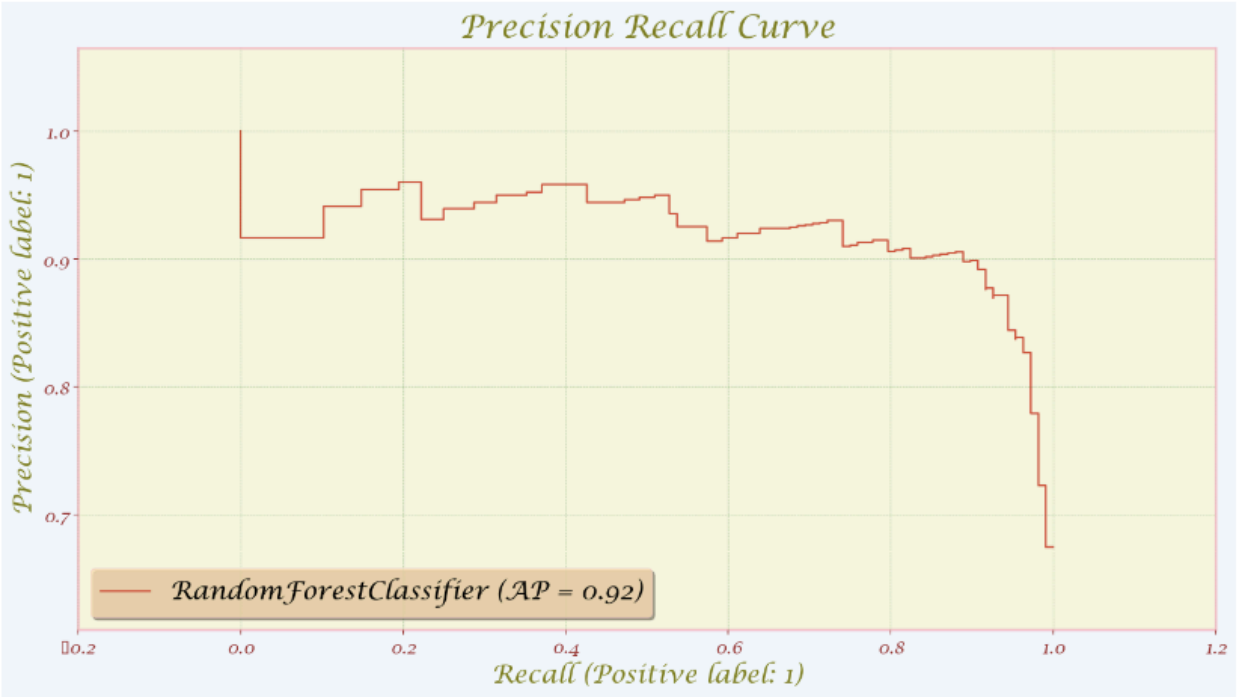
Based on the confusion matrix the occurrence of false positive and false negative is relatively low, this indicates that the model can be used for real life application.

```
plot_roc_curve(random_forest, X_test_tree, y_test_tree);
plt.title('Roc Curve');
```



The false positive in this model is relatively low with a high true positive rate.

```
plot_precision_recall_curve(random_forest, X_test_tree, y_test_tree)
plt.title('Precision Recall Curve');
```



This precision recall curve is considered as relatively good because it goes from top left corner horizontally to top right corner and straight down to right bottom corner. This indicates that the model is able to achieve high precision with high recall rate, it also mean that the model is able to make positive prediction while correctly identify the most of the positive instances.

```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=random_forest, X = X_train_tree, y=y_train_tree, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 86.77 %
Std. Dev: 2.41 %
```

The model accuracy is relatively high with a low standard deviation this indicates that this model is able to predict no heart disease and with heart disease accurately and the performance of this model is relatively stable, thus this model is considered as reliable.

5.0 Evaluation

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class, it is defined as the ratio of true positives to the sum of a true positive and false positive.
Precision = TP/(TP + FP)

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.
Recall = TP/(TP+FN)

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.
F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

5.1 Each classification report

KNN Classification (without feature scaling) report

```
Training Accuracy     : 81.17 %
Model Accuracy Score  : 67.39 %
-----------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.61      0.61      0.61        76
           1       0.72      0.72      0.72       108

    accuracy                           0.67       184
   macro avg       0.66      0.66      0.66       184
weighted avg       0.67      0.67      0.67       184
```

KNN classification (without feature scaling) report shows the accuracy (0.67), precision (0.67), recall (0.67) and f1-score (0.67).

KNN Classification (with feature scaling) report

```
Training Accuracy     : 90.59 %
Model Accuracy Score  : 86.96 %
-----------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.84      0.84      0.84        76
           1       0.89      0.89      0.89       108

    accuracy                           0.87       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.87      0.87      0.87       184
```

KNN classification (with feature scaling) report shows the accuracy (0.87), precision (0.87), recall (0.87) and f1-score (0.87). Has a 0.20 difference between KNN classification (without feature scaling) report.

SVM Classification (without feature scaling) report

```
Training Accuracy     : 69.03 %
Model Accuracy Score  : 69.57 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.66      0.55      0.60        76
           1       0.72      0.80      0.75       108

    accuracy                           0.70       184
   macro avg       0.69      0.67      0.68       184
weighted avg       0.69      0.70      0.69       184
```

SVM classification (with feature scaling) report shows the accuracy (0.70), precision (0.69), recall (0.70) and f1-score (0.69).

SVM Classification (with feature scaling) report

```
Training Accuracy     : 90.31 %
Model Accuracy Score  : 88.04 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.89      0.82      0.85        76
           1       0.88      0.93      0.90       108

    accuracy                           0.88       184
   macro avg       0.88      0.87      0.88       184
weighted avg       0.88      0.88      0.88       184
```

SVM classification (with feature scaling) report shows the accuracy (0.88), precision (0.88), recall (0.88) and f1-score (0.88). Has a 0.19 difference between SVM classification (without feature scaling) report.

Decision tree report

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 80.98 %
-------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.77      0.76      0.77        76
           1       0.83      0.84      0.84       108

    accuracy                           0.81       184
   macro avg       0.80      0.80      0.80       184
weighted avg       0.81      0.81      0.81       184
```

Decision Tree classification report shows the accuracy (0.81), precision (0.81), recall (0.81) and f1-score (0.81).

Random Forest report

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 85.87 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.85      0.80      0.82        76
           1       0.87      0.90      0.88       108

    accuracy                           0.86       184
   macro avg       0.86      0.85      0.85       184
weighted avg       0.86      0.86      0.86       184
```

Random Forest classification report shows the accuracy (0.86), precision (0.86), recall (0.86) and f1-score (0.86).

After comparing the result of modeling, it is found that **support vector machines** have the highest accuracy score, **88.04%** compared to the others classification models. It has the least training accuracy 90.31% among the other models but has the highest model f1 score 90.09 among the other models.

5.2 Comparing the model

Training accuracy refers to the accuracy of a machine learning model on the training data. It measures the degree to which the model correctly predicts the labels or classes of the training data. It measures how well the model fits the training data.

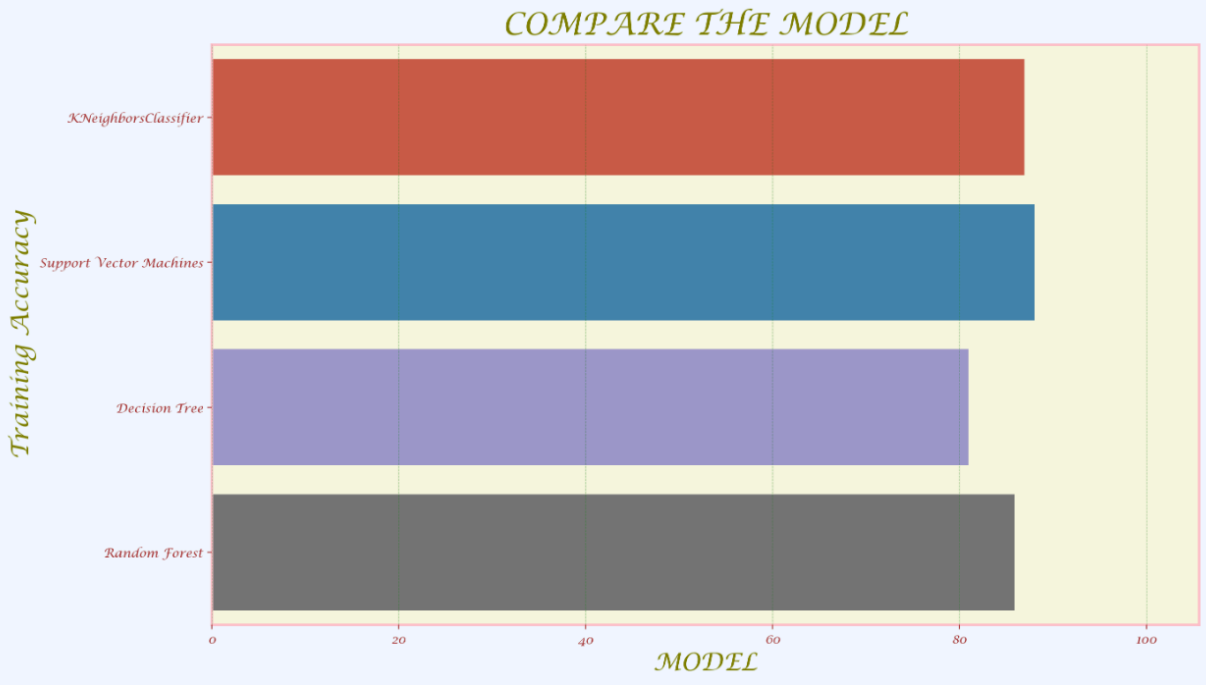
Model f1 score combines prediction and train test split from sklearn.model_selection, into a single score that provides a balanced measure of the model's performance. The formula is comparing the predicted labels to the actual labels then using accuracy_score to return value 0 and 1 to represent the proportion of correct predictions, multiplying by 100 to convert it to percentage format.

Model accuracy score uses the same formula as model f1 score and using the value 0 and 1 to calculate the ratio of the number correctly to the total number of instances in the dataset.

Model	Training Accuracy	Model f1 Score	Model Accuracy Score
Support Vector Machines	90.31000	90.090000	88.040000
KNeighborsClassifier	90.590000	88.890000	86.960000
Random Forest	100.000000	88.180000	85.570000
Decision Tree	100.000000	83.870000	80.980000

For final evaluation of how good our model, we are using accuracy_score from scikit learn instead of the mean score of cross_val_score

5.3 Result



We can see that SVM has higher training accuracy from the diagram above. Which makes it the best model classifier among the others.

6.0 Deployment

```
[1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 0 1 0 0 0 0 1 1 0
 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 1 0 1 1
 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 0
 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 1 1 0 0 1 0 0
 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1]
```

	Actual	Predicted
135	0	0
169	1	1
177	0	1
139	1	1
123	1	1
24	1	1
134	0	0
183	1	1
144	0	0
105	1	0
37	0	0
50	1	1
52	0	0
91	1	1
182	1	1

The diagram above is use to compare the actual data and predicted data by random forest by using values between 0 and 1. Some of the data between actual data and predicted data don't have the same color and same value because they are different. The predicted data is according to the data from the 11 column of data types and predicted by the result of each patient on getting heart disease but in fact some data was not predicted as same as actual data. For example, for

patient 135 was predicted correctly that is why the data value on predicted data column and actual data column are the same. On the other hand, the value of patient 177 for the 2 columns was not the same which means it was predicted incorrectly.

7.0 Conclusion

In this project, the selected dataset has been predicted with classification models. This dataset has been comprehended, validated, and cleansed in a way that improves the models' reliability and accuracy throughout training. The performance of a classification challenge is mostly determined by classification accuracy. To bypass this, we employed a confusion matrix as a performance metric for evaluation.

The four classification models are K-Nearest Neighbors, Support Vector Machines, Decision Tree and Random Forest have been utilized for this project. From the confusion matrix, we can see that **Support Vector Machines** has the highest accuracy score of **88.04%**. Support Vector Machines is one of the supervised Machine Learning methods used for classification, or predicting discrete valued outcomes. The inferences regarding the importance of each feature are based on the learned coefficients (weights) during the training process. The association's orientation, positive or negative, is also specified. As a result, Support Vector Machines is good at determining the relationship between the features. When the dataset comprises linearly separable features like gender, Support Vector Machines also appears to be quite efficient.

Next, **KNeighborsClassifier** also provides a good accuracy score of **86.96%**, but still lower than that of Support Vector Machines. While KNeighborsClassifier is a powerful algorithm for many classification tasks, its performance can degrade when dealing with high-dimensional feature spaces and complex datasets. In such cases, the cost of computing the distance between a new point and an old point can become prohibitively expensive, leading to reduced algorithm performance. However, the suitability of a model for a given task depends on various factors beyond just the size of the dataset, such as the number of input features, the nature of the target variable, and the degree of correlation between the features and the target variable.

Furthermore, **Random Forest** provided an accuracy score of **85.87%**, which is lower than that of Support Vector Machines and KNeighborsClassifier. Random Forest is an ensemble learning method that combines multiple decision trees to create a robust prediction model. Each tree in the forest is constructed using a random subset of the input features, which helps to reduce overfitting and improve the generalization of the model. Random Forest is a versatile algorithm that can be applied to a wide range of real-world scenarios and can handle both categorical and continuous input features

Lastly, **Decision Tree** has the least accuracy score of **80.98%**. Decision Tree "Decision Tree can be a useful algorithm for many classification and regression tasks, but it has some limitations. One limitation is that Decision Tree can be prone to overfitting and may not perform well on complex datasets with many features and classes.
In conclusion, Support Vector Machines is the best prediction model for this project which has an accuracy score of 88.04%.