ProjectMilestone_3_PuppalaSucharitha

November 6, 2022

0.1 Term Project

0.2 Project MileStone 1: Data Selection and EDA

0.3 Puppala Sucharitha

0.4 Data Science, Bellevue University

0.5 DSC550-T301 Data Mining (2231-1)

0.6 Professor : Dr. Brett Werner

0.7 Date : 10/03/2022

0.8 Introduction :

Heart disease describes a range of conditions that affect your heart. Diseases under the heart disease umbrella include blood vessel diseases, such as coronary artery disease, heart rhythm problems (arrhythmias) and heart defects you're born with (congenital heart defects), among others.

The term "heart disease" is often used interchangeably with the term "cardiovascular disease". Cardiovascular disease generally refers to conditions that involve narrowed or blocked blood vessels that can lead to a heart attack, chest pain (angina) or stroke. Other heart conditions, such as those that affect your heart's muscle, valves or rhythm, also are considered forms of heart disease.

Heart disease is one of the biggest causes of morbidity and mortality among the population of the world. Prediction of cardiovascular disease is regarded as one of the most important subjects in the section of clinical data analysis. The amount of data in the healthcare industry is huge. Data mining turns the large collection of raw healthcare data into information that can help to make informed decisions and predictions.

According to a news article, heart disease proves to be the leading cause of death for both women and men. About 610,000 people die of heart disease in the United States every year–that's 1 in every 4 deaths.Heart disease is the leading cause of death for both men and women. More than half of the deaths due to heart disease in 2009 were in men.Coronary Heart Disease(CHD) is the most common type of heart disease, killing over 370,000 people annually.

This makes heart disease a major concern to be dealt with. But it is difficult to identify heart disease because of several contributory risk factors such as diabetes, high blood pressure, high cholesterol, abnormal pulse rate, and many other factors. Due to such constraints, scientists have turned towards modern approaches like Data Mining and Machine Learning for predicting the disease.

Machine learning (ML) proves to be effective in assisting in making decisions and predictions from the large quantity of data produced by the healthcare industry.

This project focuses on to classify / predict whether a patient is prone to heart disease depending on multiple attributes.

In this project the following steps are involved :

- Overview of the Dataset.
- Exploratory Data Analysis.
- Data Preparation.
- Identifying the features that are most influential for having Heart Disease.
- Build different models to predict Heart Disease.

This dataset contains 11 features that can be used to predict a possible heart disease.

1. Age : age of the patient [years]
2. Sex : sex of the patient [M: Male, F: Female]
3. ChestPainType : chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP : resting blood pressure [mm Hg]
5. Cholesterol : serum cholesterol [mm/dl]
6. FastingBS : fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. 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]
8. MaxHR : maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina : exercise-induced angina [Y: Yes, N: No]
10. Oldpeak : oldpeak = ST [Numeric value measured in depression]
11. ST_Slope : the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
12. HeartDisease : output class [1: heart disease, 0: Normal]

The Heart Failure Prediction dataset is collected from Kaggle.com and the link to the dataset is given below.

https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

References :

- Shubhankar Rawat. "HeartDisease Prediction." Medium,Towards Data Science, 10 Aug 2019, https://medium.com/towards-data-science/heart-disease-prediction-73468d630cfc

- fedesoriano. (September 2021). Heart Failure Prediction Dataset. Retrieved [Date Retrieved] from https://www.kaggle.com/fedesoriano/heart-failure-prediction.

0.9 Milestone -1 : Data Selection and Exploratory Data Analysis(EDA)

In the Milestone-1, I would like to get an overview of the data set collected and the Exploratory Data Analysis of the the dataset.

In this Milestone-1, I would like to know the following points:

- Distribution of the different features in the data set on the HeartDisease.
- To know the counts of the categorical features available in the dataset.
- Are there any duplicated, missing values and outliers in the dataset.

[1]:
```
# Importing all the necessary libraries.
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.stats import skew
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Loading the heart dataset which is named heart.csv
heartdf = pd.read_csv('heart.csv')
```

[3]:
```
# Getting the first 5 rows of the heart dataset.
heartdf.head()
```

[3]:
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
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

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
```

```
3              Y     1.5    Flat           1
4              N     0.0      Up           0
```

[4]:
```python
# Getting the shape of the heart dataset
heartdf.shape
```

[4]: (918, 12)

[5]:
```python
# Getting the size of the dataset
heartdf.size
```

[5]: 11016

[6]:
```python
# Information about the dataset variables.
heartdf.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
```

[7]:
```python
# Checking for null values
heartdf.isna().sum()
```

[7]:
```
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
```

[8]:
```python
# Checking if any duplicated values present in the data set.
heartdf.duplicated()
```

[8]:
```
0      False
1      False
2      False
3      False
4      False
       …
913    False
914    False
915    False
916    False
917    False
Length: 918, dtype: bool
```

[9]:
```python
# Getting the number of unique columns in the data set
heartdf.nunique()
```

[9]:
```
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
```

Observations of the dataset:

- The dataset has 918 rows and 12 columns.
- We can see that the dataset contains both numerical and categorical data.
- In the initial review of the dataset we can see there are no null values.
- Extracted the number of unique values for each column.
- Number of unique values for each column is extracted above and will be reviewed during the EDA section.

0.9.1 Exploratory Data Analysis.

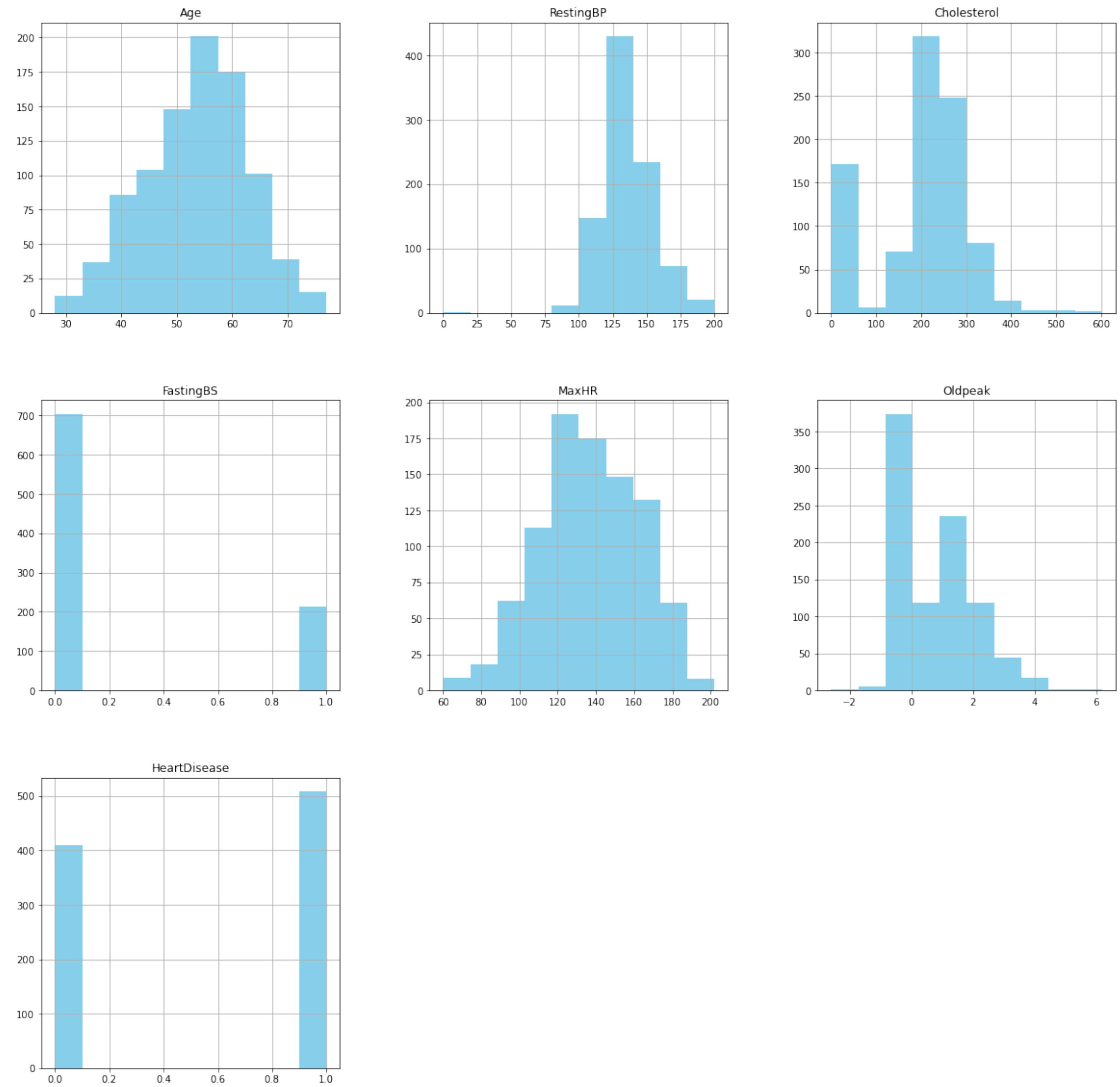
[10]:
```
# Initially lets's use the describe() to get an idea on the dataset.
heartdf.describe()
```

[10]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR
count	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368
std	9.432617	18.514154	109.384145	0.423046	25.460334
min	28.000000	0.000000	0.000000	0.000000	60.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000

	Oldpeak	HeartDisease
count	918.000000	918.000000
mean	0.887364	0.553377
std	1.066570	0.497414
min	-2.600000	0.000000
25%	0.000000	0.000000
50%	0.600000	1.000000
75%	1.500000	1.000000
max	6.200000	1.000000

[11]:
```
# plot histograms for each numerical variable
heartdf.hist(figsize = (20, 20),color='skyblue')
plt.show()
```



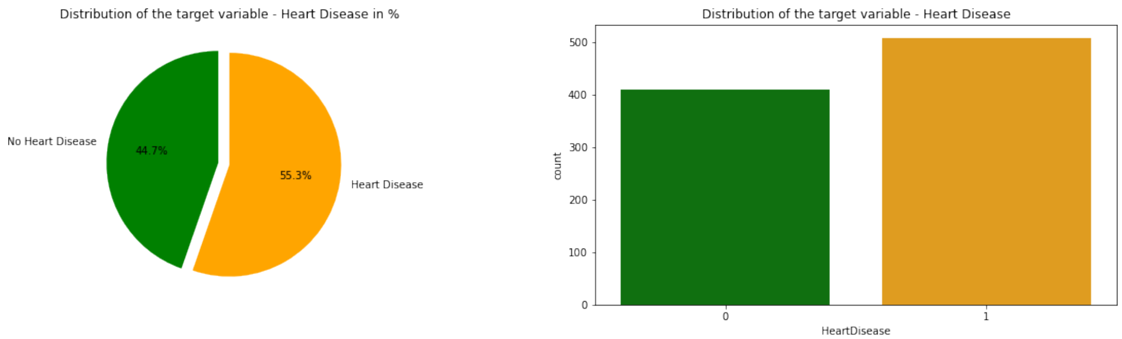
Observations from the histogram plots of the dataset.

- From the above plots we can see some features are skewed.

0.9.2 Visualization of the target variable

[12]:
```
# Let's plot a pie plot and count plot of the target variable 'HeartDisease'
l = list(heartdf['HeartDisease'].value_counts())
circle = [l[1] / sum(l) * 100,l[0] / sum(l) * 100]
# Setting the colors of the plots.
colors = ['green','orange']

# PLotting the pie plot.
fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (20,5))
```

```
plt.subplot(1,2,1)
plt.pie(circle,labels = ['No Heart Disease','Heart Disease'],autopct='%1.
↪1f%%',startangle = 90,explode = (0.1,0),colors = colors)
plt.title('Distribution of the target variable - Heart Disease in %'); # Title␣
↪of the plot.

# Plotting the count plot.
plt.subplot(1,2,2)
sns.countplot('HeartDisease',data = heartdf,palette = colors)
plt.title('Distribution of the target variable - Heart Disease');# Title of the␣
↪plot
plt.show()
```



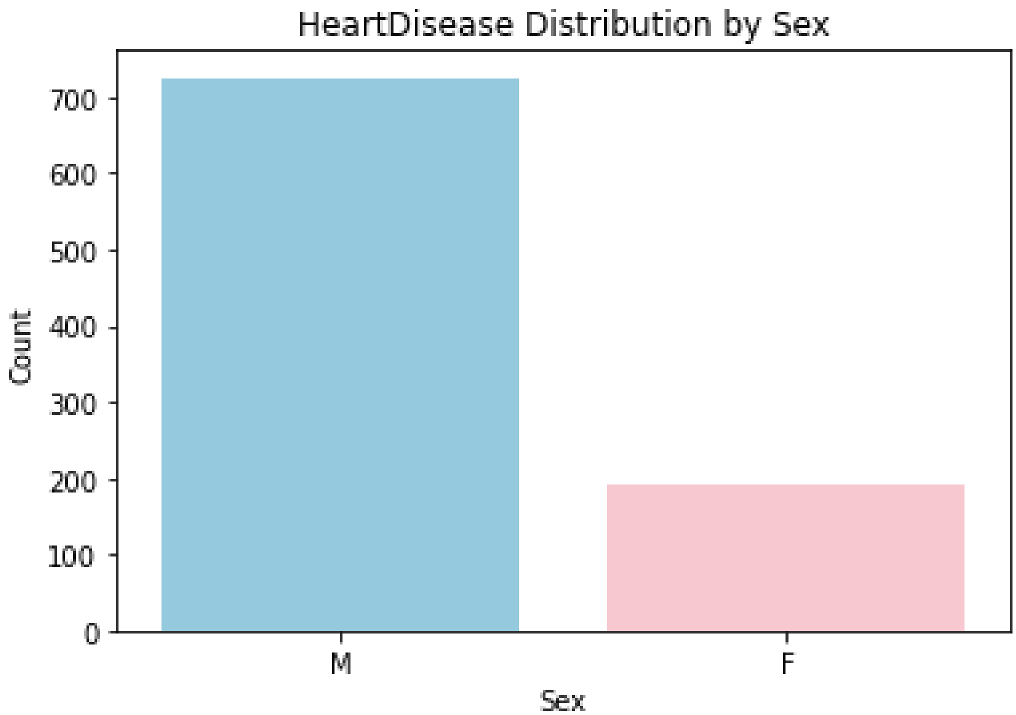
Observations from the target variable pie plot and count plot.

- From the above visualizations we can see that the dataset is evenly balanced.
- We can see from the pie plot that 55.3% are having Heart Disease and 44.7% are not having any Heart Disease.

0.9.3 Visualization of the target variable with sex.

[13]:
```
#Let's cheeck for the distribution of HeartDisease by sex.
sns.countplot(data=heartdf,x='Sex' , palette=['skyblue','pink']);
#Title of the plot
plt.title('HeartDisease Distribution by Sex')
# X- label
plt.xlabel('Sex')
# Y- label
plt.ylabel('Count')
```

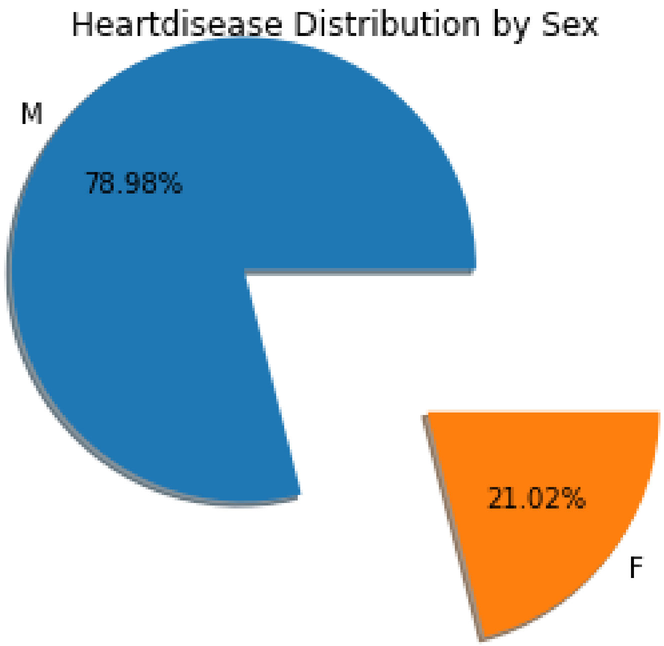
[13]: `Text(0, 0.5, 'Count')`



Observations of the target variable by sex.

- Here we can see that males are highly proned to have HeartDisease.

[14]:
```
# Let's plot a pie plot for the distribution of heart disease by sex.
plt.rcParams.update({'font.size': 10})
ax=heartdf['Sex'].value_counts().plot.pie(explode=[0.5,0.5],autopct='%1.
 ↪2f%%',shadow=True );
# Plot title.
ax.set_title(label = "Heartdisease Distribution by Sex");
plt.axis('off');
```



Observations of the pie plot :

- From the above pie plot we see that about 78.98% of males have heart disease and 21.02% of females have HeartDisease.
- We can say that Males are approximately 3 times more likely to have HeartDisease than females.

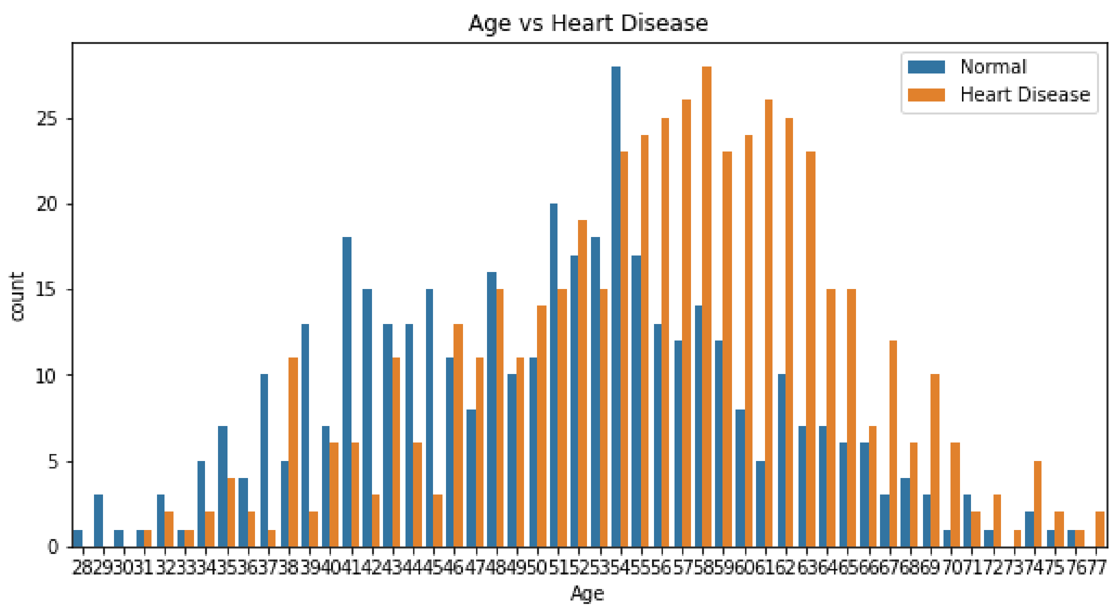
0.9.4 Dividing the Categorical and Numerical variables from the dataset.

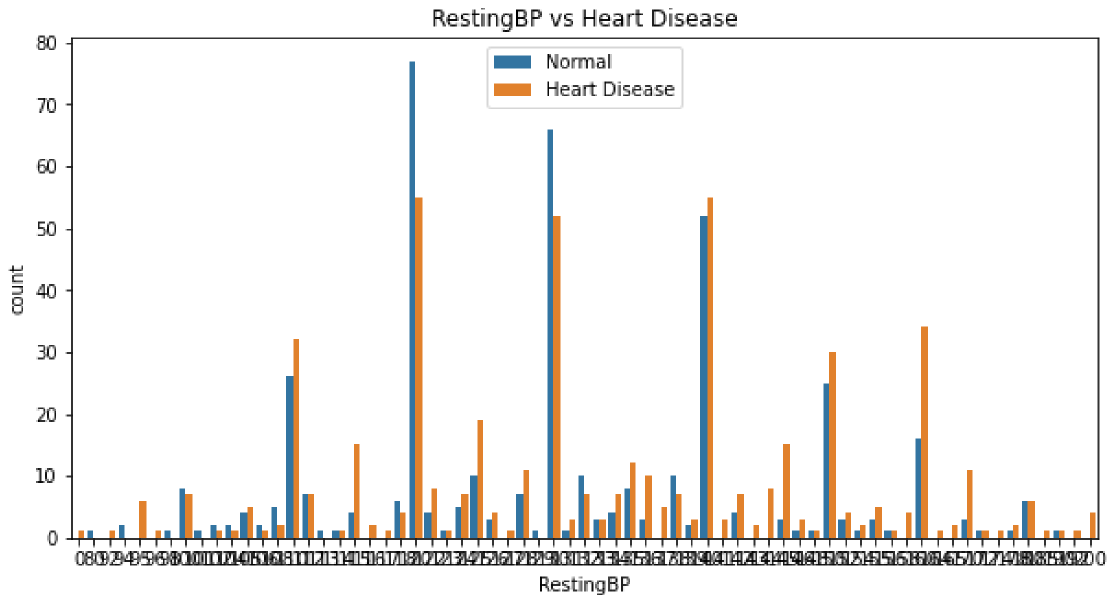
[15]:
```
# Let's divide the categorical and numerical variables from the dataset.
col = list(heartdf.columns)
categ_features = []
numer_features = []
for i in col:
    if heartdf[i].dtype == np.object:
        categ_features.append(i)
    else:
        numer_features.append(i)

print('Categorical Features of the heart dataset :',*categ_features)
print('Numerical Features of the heart dataset :',*numer_features)
```

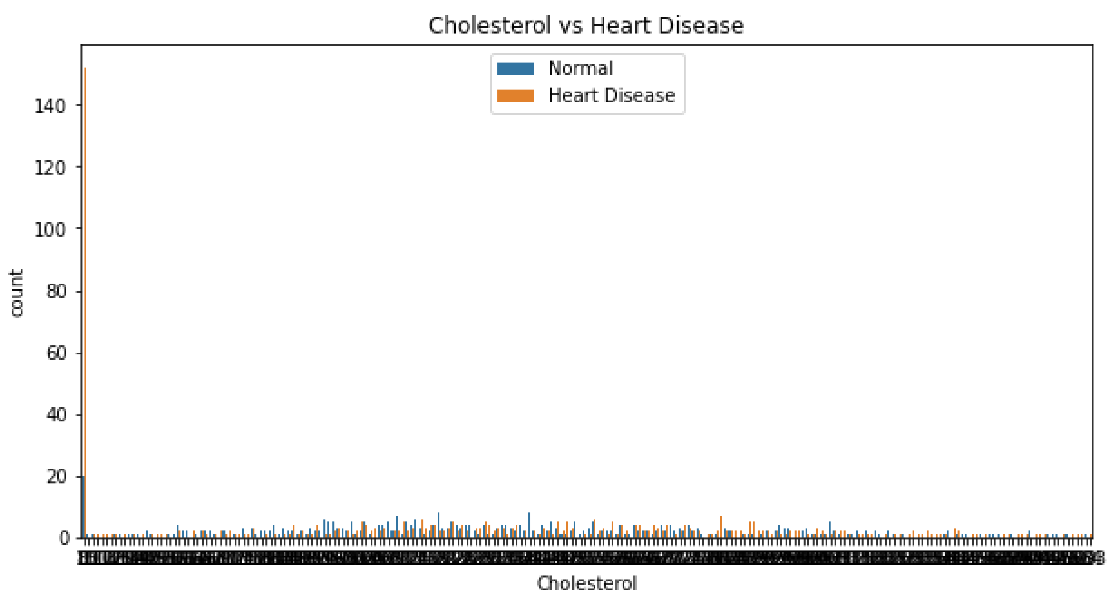
```
Categorical Features of the heart dataset : Sex ChestPainType RestingECG
ExerciseAngina ST_Slope
Numerical Features of the heart dataset : Age RestingBP Cholesterol FastingBS
MaxHR Oldpeak HeartDisease
```

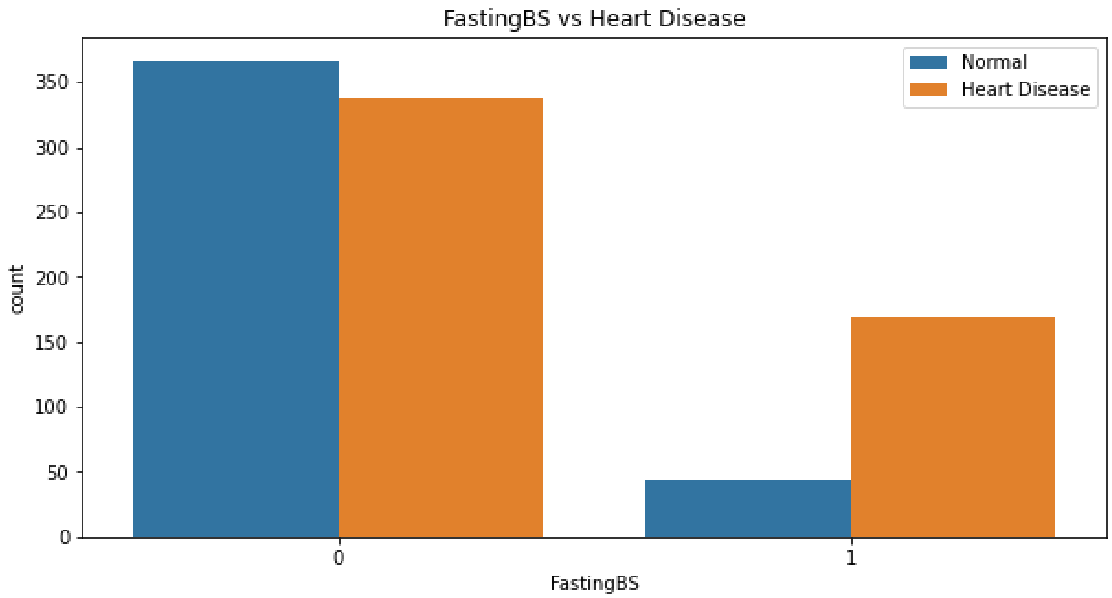
0.9.5 Plotting Numerical features with Target Variable i.e HeartDisease.

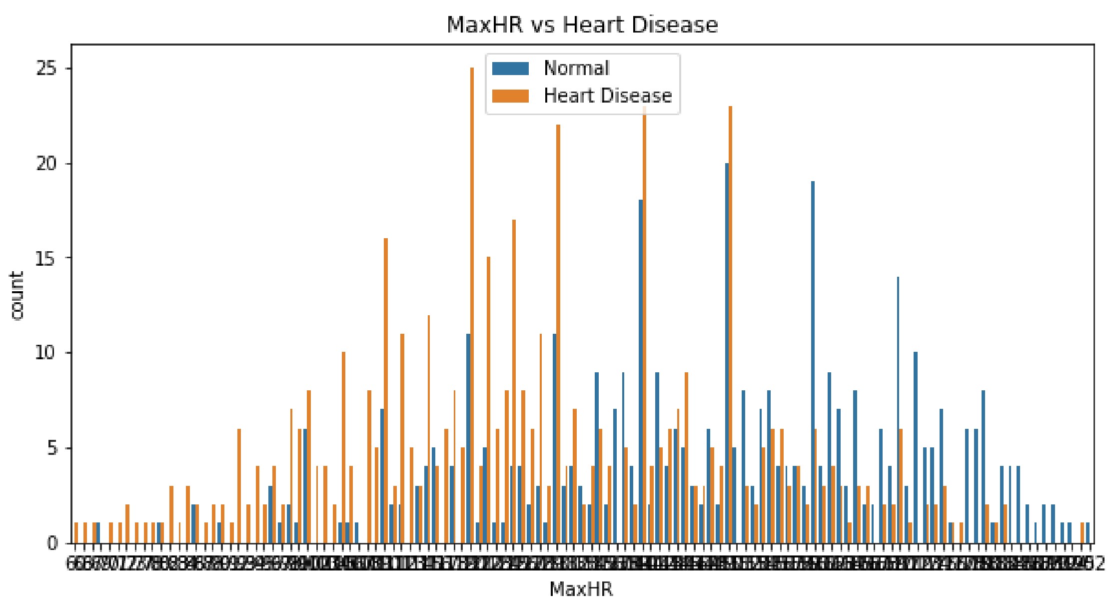
[16]:
```python
# Ploting numerical features with target variable.
# With these plots we compare between our target and our numerical features.
#colors = ['green','orange']
for i in numer_features:
    plt.figure(figsize=(10,5))
    sns.countplot(x=i, data=heartdf, hue='HeartDisease')
    plt.legend(['Normal', 'Heart Disease'])
    plt.title(i + " vs Heart Disease")# Plot title
    plt.show()
```

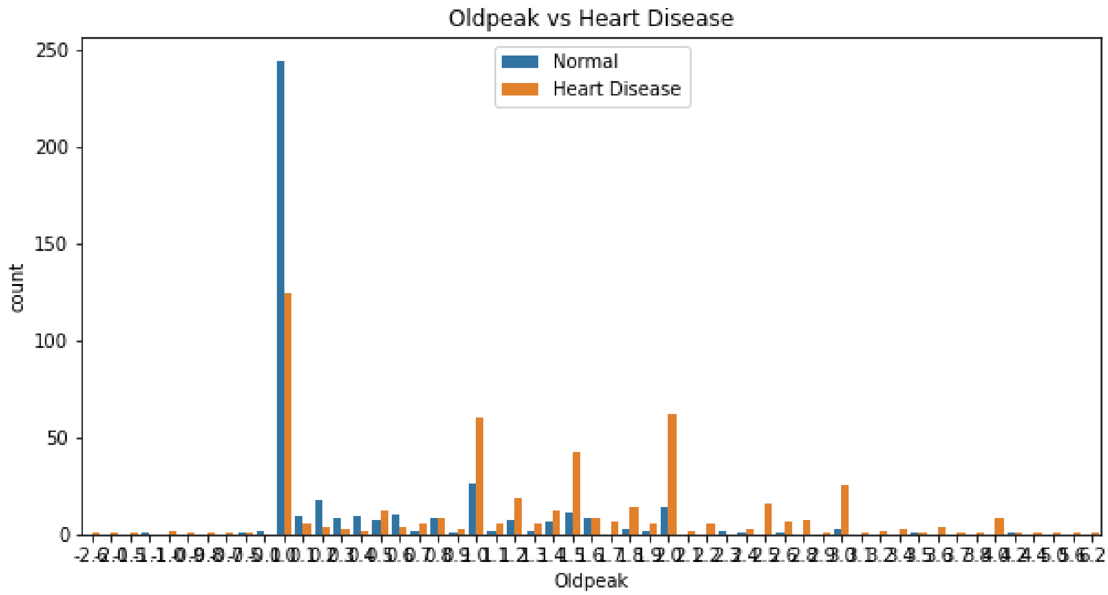


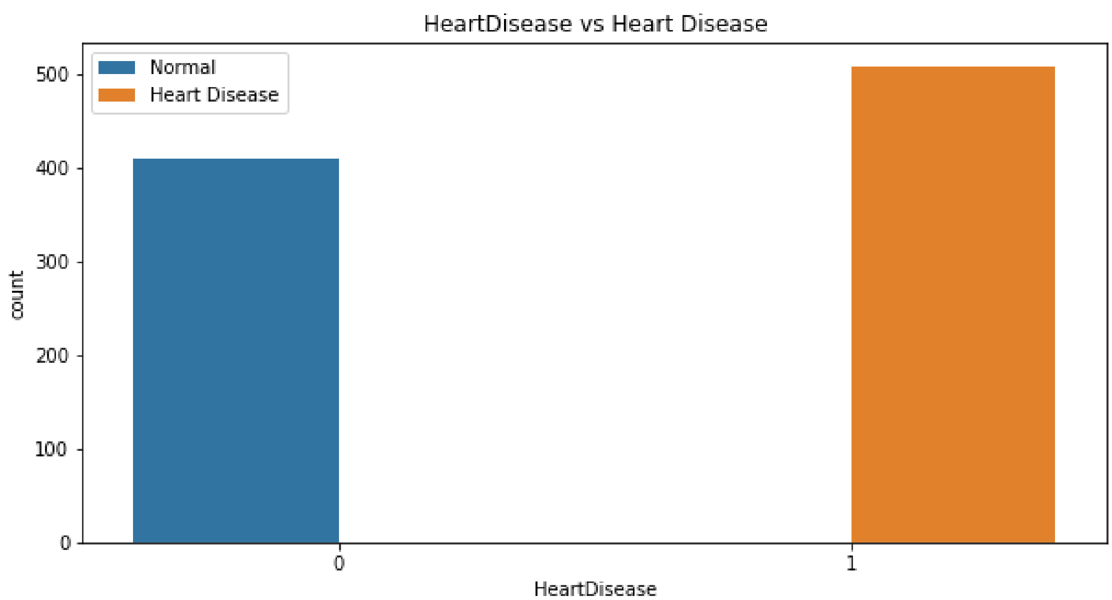

RestingBP vs Heart Disease
Normal
Heart Disease
count
RestingBP


Cholesterol vs Heart Disease
Normal
Heart Disease
count
Cholesterol


FastingBS vs Heart Disease
Normal
Heart Disease
count
FastingBS
0
1
0
50
100
150
200
250
300
350


MaxHR vs Heart Disease
Normal
Heart Disease
count
MaxHR
0
5
10
15
20
25



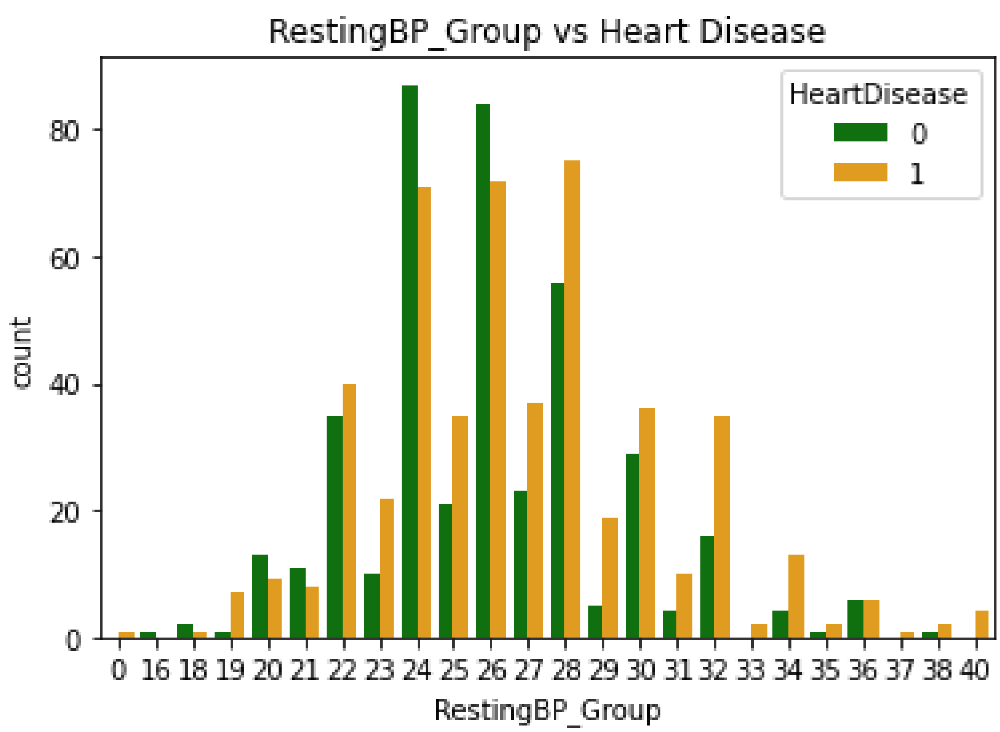


Observations:

- From the above count plot of Age vs HeartDisease most of the heart disease Patients have age between 55 and 65
- From the above count plot of Fasting Blood Sugar vs HeartDisease , we see that the persons having FastingBS and positive HeartDisease couts upto 150.

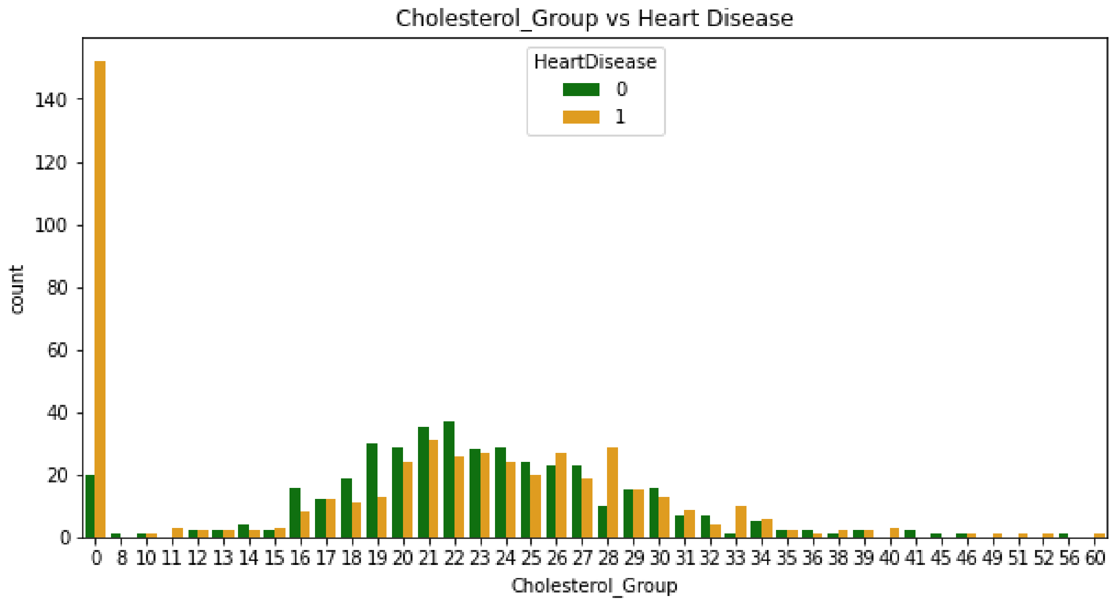
- From the above graphs we can see that the distribution of the features RestingBP,Cholesterol,MaxHR and Oldpeak are not clearly understanble because of the presence of many unique data points in the features.
- For more clear understanding let us scale the individual values of these features which brings the data points to a constant value representing a range of values.
- Here I have divided the data points of the numerical features by 5 or 10 and assigned the quotient value as the representative of the constant data point.

[17]:
```
# Scaling the 'RestingBP'
colors = ['green','orange']
heartdf['RestingBP_Group'] = [ int(i / 5) for i in heartdf['RestingBP']]
restingBP_group = heartdf['RestingBP_Group']
sns.countplot(restingBP_group,data = heartdf,hue = "HeartDisease",palette =␣
↪colors)
title = 'RestingBP_Group vs Heart Disease' # Plot title
plt.title(title);
```

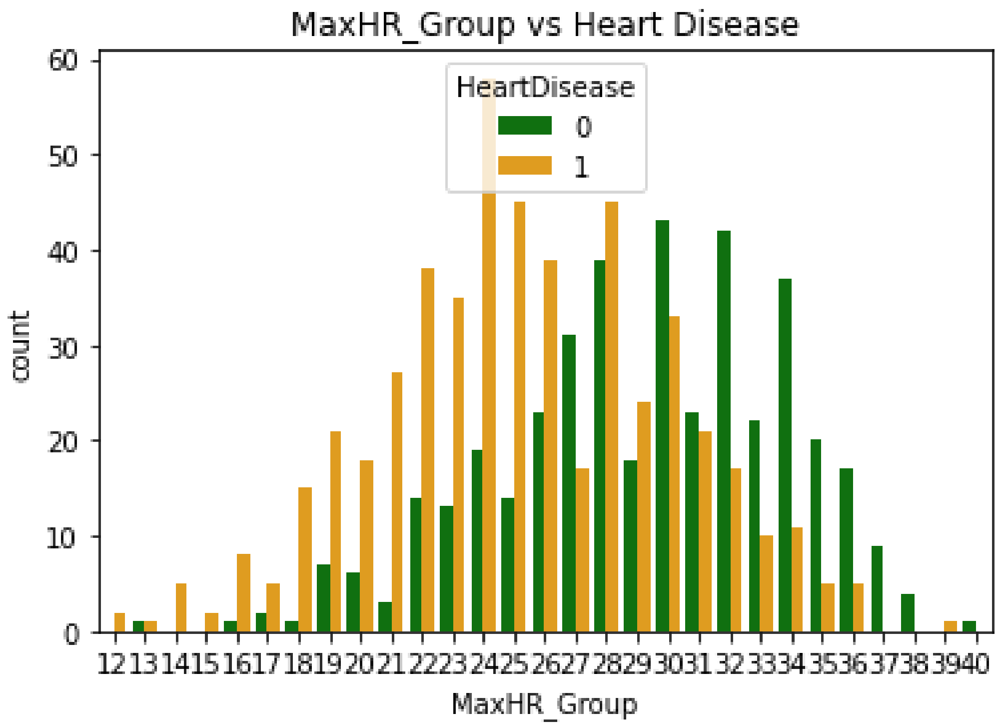


[18]:
```
# Scaling of 'Cholesterol'
heartdf['Cholesterol_Group'] = [ int(i / 10) for i in heartdf['Cholesterol']]
cholestrol_group = heartdf['Cholesterol_Group']
plt.figure(figsize=(10,5))
```

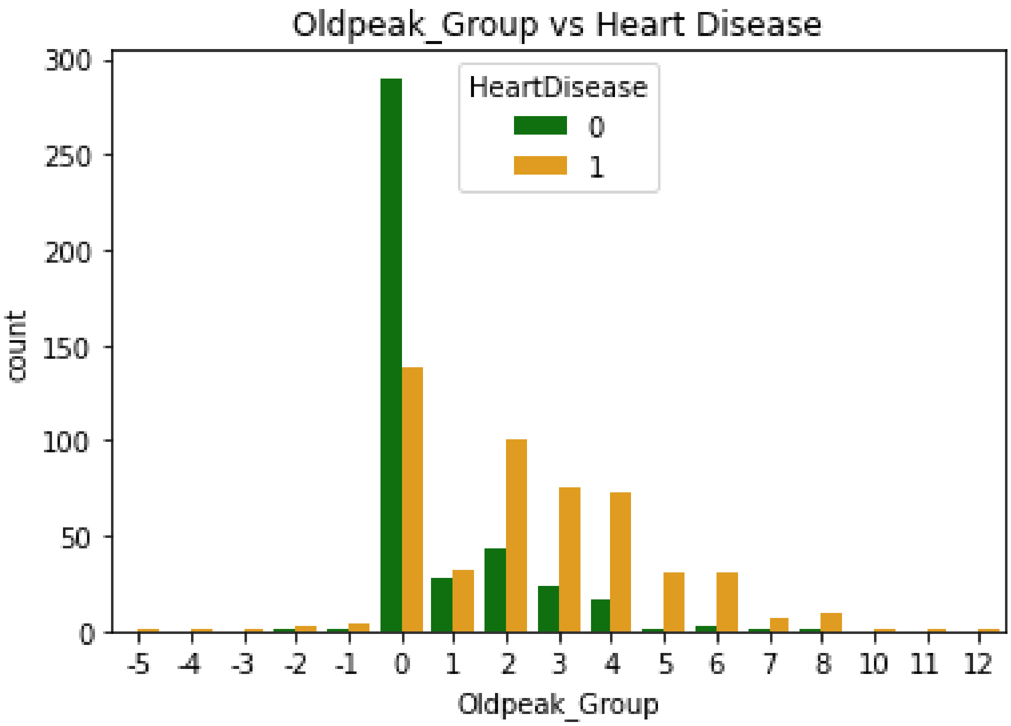
```
sns.countplot(cholestrol_group,data = heartdf,hue = "HeartDisease",palette =␣
↪colors)
title =  'Cholesterol_Group vs Heart Disease'# Plot Title.
plt.title(title);
```



[19]:
```
# Scaling of 'MaxHR'
heartdf['MaxHR_Group'] = [ int(i / 5) for i in heartdf['MaxHR']]
maxhr_group = heartdf['MaxHR_Group']
sns.countplot(maxhr_group,data = heartdf,hue = "HeartDisease",palette = colors)
title = 'MaxHR_Group vs Heart Disease' # Plot Title
plt.title(title);
```



[20]:
```
# Scaling of 'Oldpeak'
heartdf['Oldpeak_Group'] = [ int( (i*10) / 5) for i in heartdf['Oldpeak']]
oldpeak_group = heartdf['Oldpeak_Group']
sns.countplot(oldpeak_group,data = heartdf,hue = "HeartDisease",palette =␣
 ↪colors)
title = 'Oldpeak_Group vs Heart Disease'# Plot title.
plt.title(title);
```



Observations of count plots of :

- From the graph of scaled RestingBP, we can say that the values 95 to 170 ((195)- (345))are mostly proned to have HeartDisease.
- From the graph of scaled Cholesterol, we can say that the values 160 to 340 ((1610)- (3410))are proned to have HeartDisease.
- From the graph of scaled MaxHR, we can say that the values 70 to 180 ((145) - (365)) are mostly proned to have HeartDisease.
- From the graph of sccaled Oldpeak, we can say that the values 0 to 8 i.e 0 to 4((05/10) - (85/10)) are mostly proned to have HeartDisease.

[21]:

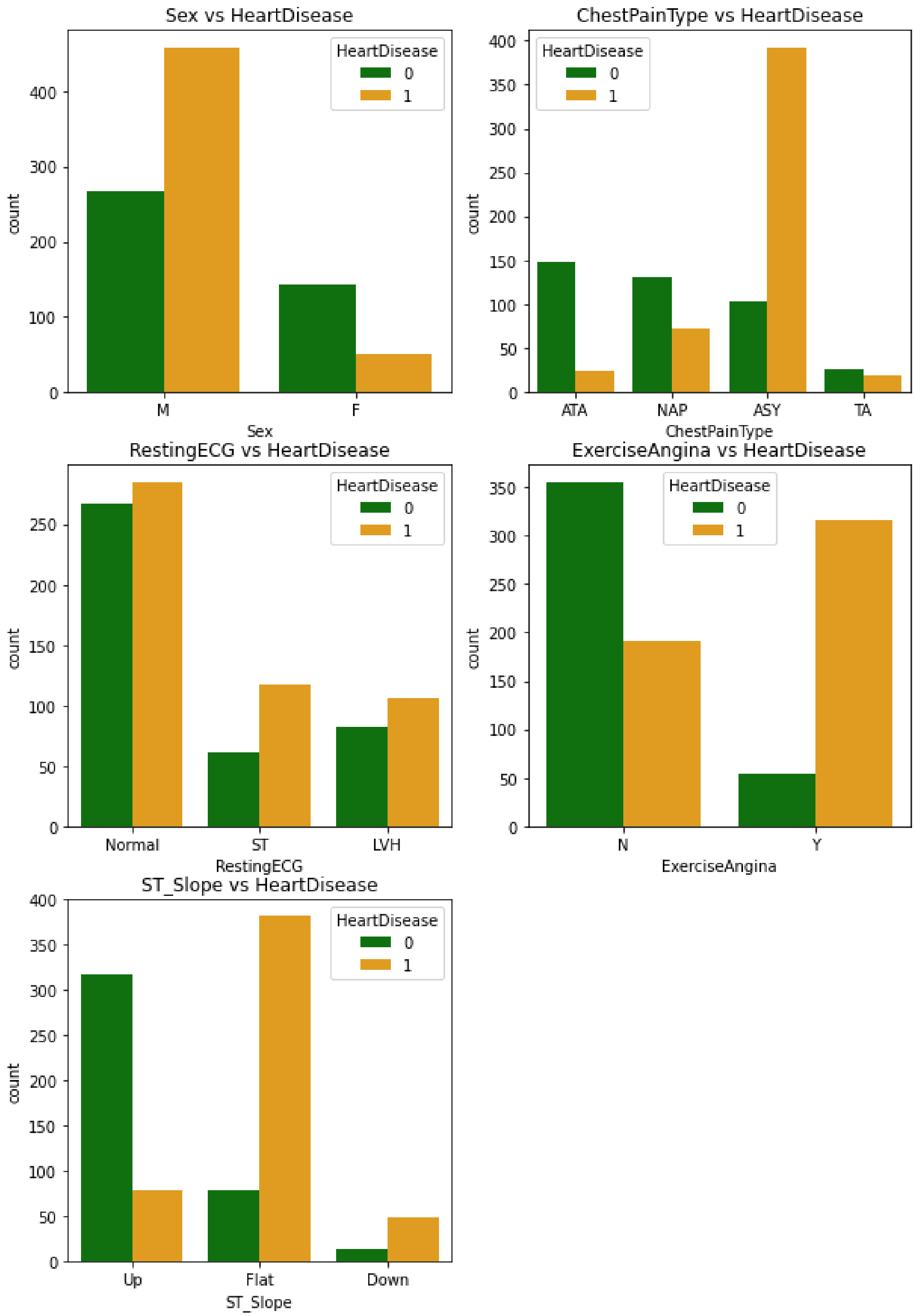
```
# Drop the grouped columns that are created.
heartdf = heartdf.drop(columns =␣
↪["Oldpeak_Group","RestingBP_Group","Cholesterol_Group","MaxHR_Group"])
```

0.9.6 Plotting Categorical Features with the Target Variable i.e. Heart Disease.

[22]:

```
# Plotting categorical features with the target variable.
# here we compare between target and Categorical features
colors = ['green','orange']
fig, ax = plt.subplots(nrows = 3,ncols = 1,figsize = (10,15))
for i in range(len(categ_features)):
```

```
    plt.subplot(3,2,i+1)
    sns.countplot(categ_features[i],data = heartdf,hue = "HeartDisease",palette␣
↪= colors)
    title = categ_features[i] + ' vs HeartDisease'
    plt.title(title);
```


Sex vs HeartDisease
count
Sex
M
F
ChestPainType vs HeartDisease
ChestPainType
ATA
NAP
ASY
TA
RestingECG vs HeartDisease
RestingECG
Normal
ST
LVH
ExerciseAngina vs HeartDisease
ExerciseAngina
N
Y
ST_Slope vs HeartDisease
ST_Slope
Up
Flat
Down
HeartDisease
0
1

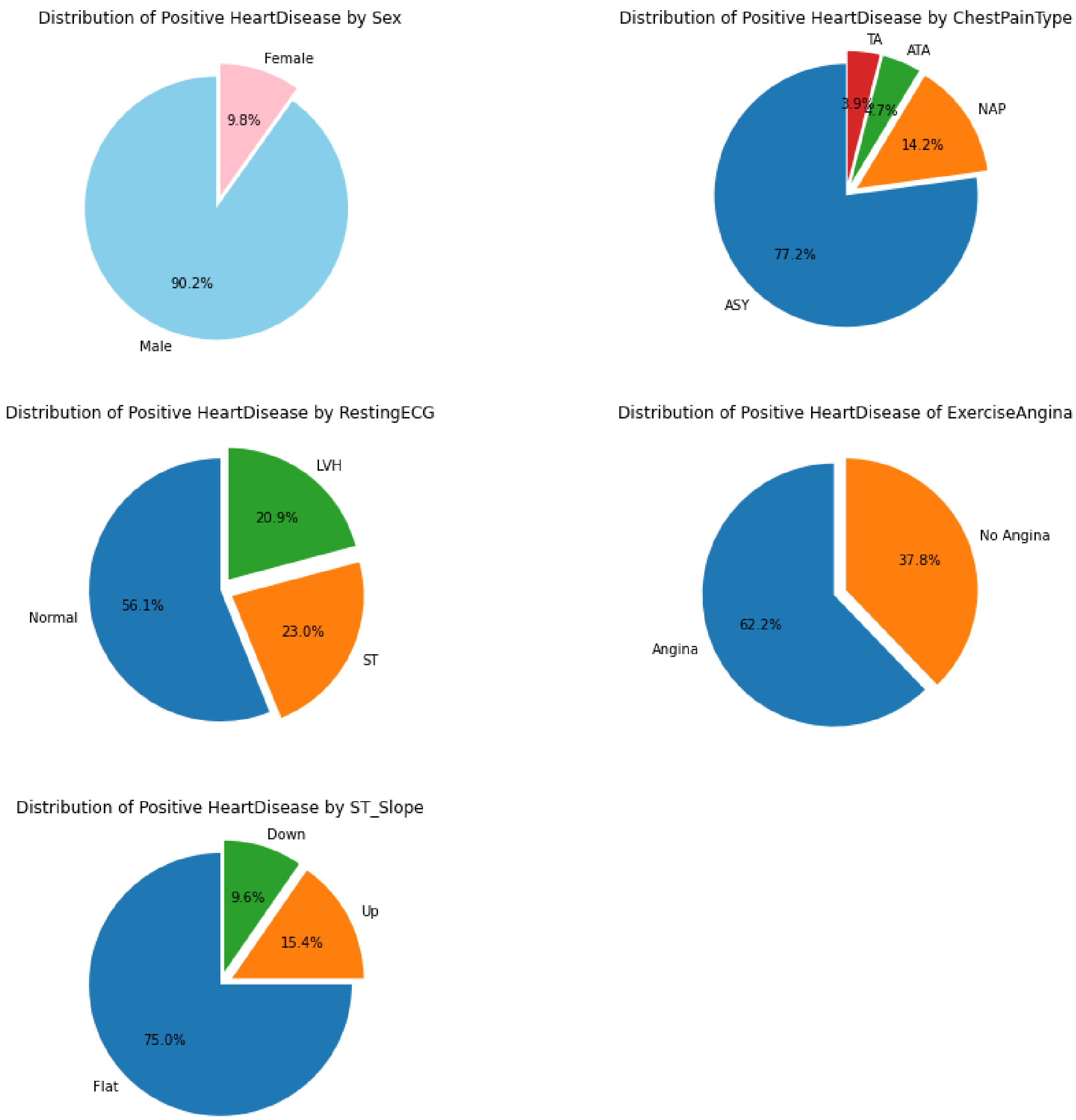
Observations of Count plots: The following observations are made from the above plots.

- From the above 'Sex vs HeartDisease' plot we can say that the Male population has more proned to HeartDisease and Female population are less proned to HeartDisease.
- The 'ChestPainType vs HeartDisease' plot has four types they are - TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic among these four ASY type has more positive HeartDisease cases.
- From the 'RestingECG vs HeartDisease' plot we see that all the three types are having Heart Disease.
- From the 'ExerciseAngina vs HeartDisease' plot we see that the persons having ExerciseAngina are more proned to have HeartDisease.
- From the 'ST_Slope vs HeartDisease'plot we can see that the persons having flat slop are more proned to have HeartDisease.

0.9.7 Visualizations of Categorical Features vs Positive Heart Disease Cases :

[23]:
```
# Getting the postive heart disease data for the categorial features.

# Sex feature
sex = heartdf[heartdf['HeartDisease'] == 1]['Sex'].value_counts()
sex = [sex[0] / sum(sex) * 100, sex[1] / sum(sex) * 100]

# ChestPainType
chestpain = heartdf[heartdf['HeartDisease'] == 1]['ChestPainType'].
 ↪value_counts()
chestpain = [chestpain[0] / sum(chestpain) * 100,chestpain[1] / sum(chestpain)␣
 ↪* 100,chestpain[2] / sum(chestpain) * 100,chestpain[3] / sum(chestpain) *␣
 ↪100]

# RestingECG
restingecg = heartdf[heartdf['HeartDisease'] == 1]['RestingECG'].value_counts()
restingecg = [restingecg[0] / sum(restingecg) * 100,restingecg[1] /␣
 ↪sum(restingecg) * 100,restingecg[2] / sum(restingecg) * 100]

# ExerciseAngina
exerangina = heartdf[heartdf['HeartDisease'] == 1]['ExerciseAngina'].
 ↪value_counts()
exerangina = [exerangina[0] / sum(exerangina) * 100,exerangina[1] /␣
 ↪sum(exerangina) * 100]

# ST_Slope
stslope = heartdf[heartdf['HeartDisease'] == 1]['ST_Slope'].value_counts()
stslope = [stslope[0] / sum(stslope) * 100,stslope[1] / sum(stslope) *␣
 ↪100,stslope[2] / sum(stslope) * 100]
```

[24]:
```
# Plotting Pie plots for the Categorical Variables with positive HeartDisease.

ax,fig = plt.subplots(nrows = 3,ncols = 1,figsize = (15,15))
colors = ['skyblue','pink']

# Pie plot of HeartDisease by Sex.
plt.subplot(3,2,1)
plt.pie(sex,labels = ['Male','Female'],autopct='%1.1f%%',startangle =␣
↪90,explode = (0.1,0),colors = colors)
plt.title('Distribution of Positive HeartDisease by Sex');# Plot title

# Pie plot of  HeartDisease by ChestPainType.
plt.subplot(3,2,2)
plt.pie(chestpain,labels = ['ASY', 'NAP', 'ATA', 'TA'],autopct='%1.
↪1f%%',startangle = 90,explode = (0,0.1,0.1,0.1))
plt.title('Distribution of Positive HeartDisease by ChestPainType'); # Plot␣
↪title

# Pie plot of HeartDisease by RestingECG.
plt.subplot(3,2,3)
plt.pie(restingecg,labels = ['Normal','ST','LVH'],autopct='%1.1f%%',startangle␣
↪= 90,explode = (0,0.1,0.1))
plt.title('Distribution of Positive HeartDisease by RestingECG'); #Plot Title

#Pie plot of HeartDisease by ExerciseAngina.
plt.subplot(3,2,4)
plt.pie(exerangina,labels = ['Angina','No Angina'],autopct='%1.1f%%',startangle␣
↪= 90,explode = (0.1,0))
plt.title('Distribution of Positive HeartDisease of ExerciseAngina'); # Plot␣
↪Title

#Pie plot of HeartDisease by ST_Slope.
plt.subplot(3,2,5)
plt.pie(stslope,labels = ['Flat','Up','Down'],autopct='%1.1f%%',startangle =␣
↪90,explode = (0,0.1,0.1))
plt.title('Distribution of Positive HeartDisease by ST_Slope'); # Plot Tile
```



Observations of Pie plots : The above plots are plotted for more clear idea about the percentage of impact of the categorical feature on positive HeartDisease

- From the pie plot of 'Sex Vs Positive HeartDisease' we see that about 90% of the patients are male.
- From the pie plot of 'ChestPainType vs Positive HeartDisease' we see that about 77.2% of the patient with HeartDisease are having ASY type of Chest Pain.
- From the pie plot 'RestingECG vs Positive HeartDisease' we see that the patients having HeartDisease have RestingECG normal level which is 56.1%
- From the pie plot 'ExerciseAngina' we see that the patients having HeartDisease , have the

problem of Exercise Induced Angina of about 62.2%

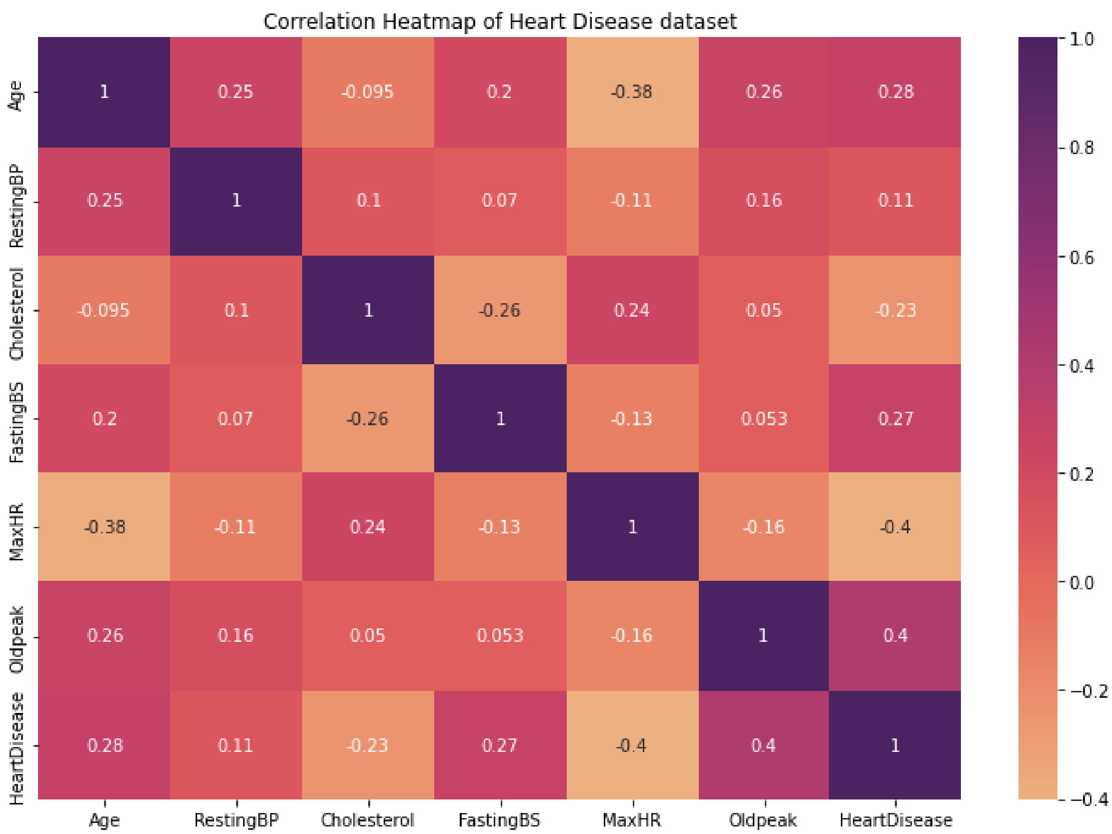
- From the pie plot 'ST_Slope vs Postivie HeartDisease' we see that about 75% of patients having HeartDisease have a flat ST_Slope.

[25]:
```
# Let's use pairplot() function from seaborn to understand the relationship␣
↪between all features.

sns.pairplot(heartdf,hue="HeartDisease");
```

[26]:
```
# Let's plot a correlation heatmap of the heart disease dataset.

# Calculate the correlation coefficient with corr().
corr_number = heartdf.corr()

# Create the heatmap for the correlation coefficients calculated above.
```

```python
fig, ax = plt.subplots(1, 1, figsize=(10,7), tight_layout = True)
sns.heatmap(corr_number, annot = True, cmap = 'flare')

# Title of the plot
plt.title('Correlation Heatmap of Heart Disease dataset')
```

[26]: Text(0.5, 1.0, 'Correlation Heatmap of Heart Disease dataset')



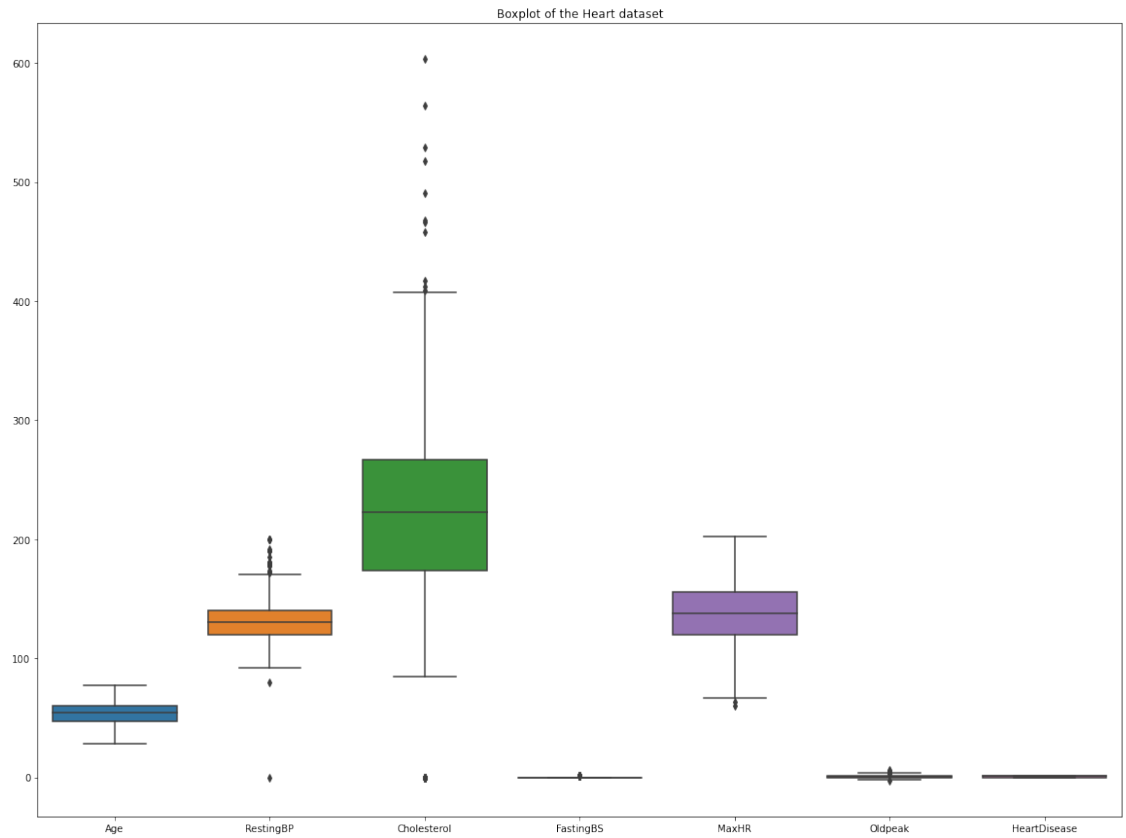
Observations of Pairplot and Correlation Heat Map:

- From the pair plot above we can see that some features are correlated with the target variable. In individuals who are older and having high RestingBP are more prone to HeartDisease.
- From the above correlation heat map we see some features are positively correlated with HeartDisease and some features are negatively correlated, but the correlation seems to be very low.
- It is better to create dummy variables for the categorical varibales of the dataset.
- As per the above correlation heat map we can say that Age, RestingBP are having good correlation with the HeartDisease.

0.9.8 Box plot to identify Outliers :

[27]:
```
# let's plot a boxplot for the heart dataset.
plt.figure(figsize=(20,15))
sns.boxplot(data = heartdf)
# Plot title
plt.title('Boxplot of the Heart dataset')
```

[27]: Text(0.5, 1.0, 'Boxplot of the Heart dataset')



Observations of boxplot :

- From the above boxplot we see that there are outliers present in the features Age, RestingBP, Cholesterol, Oldpeak and MaxHR.

0.9.9 Conclusions from Milestone -1 : Data Selection and EDA.

- The dataset has 918 rows and 12 columns.
- The target variable "HeartDisease" is well distributed.
- There are no missing values and duplicated data in the dataset.

- The observations from the plots of patients having positive HeartDisease with different variables in the dataset says that the males are more in number, ChestPainType ASY is more dominant, FastingBS is less, RestingECG is low, ST_Slope is flat.
- The people of age above 50 years, having RestingBP between 95 - 170 , having Cholesterol ranging between 160 - 340 are more prone to HeartDisease.
- We can see some features having a good correlation with the HeartDisease like Age, Sex, but the correlation strength is very low.
- In the data preparation step it is recommended to create dummy variables for the categorical variables.
- Ethical Consederations - The ethical concerns that are to be considered in handling the data are privacy, confidentiality, honesty and fairness. While handling the data I have taken care that the data is not biased towards any factors causing heartdisease. All the data is validated to ensure that the facts are not misrepresented in the visualizations.

0.10 Milestone 2: Data Preparation

- In the Milestone 2 I would like to discuss about Data Preparation and identifying the features that are most influential for having Heart Disease.
- In the data preparation I would like to describe the handling of outliers, creating dummy variables for the categorical features,elimination of the non essential features if any,and finally splitting the data into training(80%) and test(20%) data sets.

[28]:
```
#Let's define a function for detecting outliers using IQR method.

def outliers_IQR(df):

   q1 = df.quantile(0.25)

   q3 = df.quantile(0.75)

   IQR = q3-q1

   outlier = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   return outlier
```

[29]:
```
# Now let's check for the outliers in features : Age.
outlier = outliers_IQR(heartdf['Age'])

print('Total number of outliers in feature Age : '+ str(len(outlier)))

print('Max. outlier value : '+ str(outlier.max()))

print('Min. outlier value : '+ str(outlier.min()))
```

```
Total number of outliers in feature Age : 0
Max. outlier value : nan
Min. outlier value : nan
```

[30]:
```python
# Outliers in -  RestingBP

outlier = outliers_IQR(heartdf['RestingBP'])

print('Total number of outliers in feature RestingBP: '+ str(len(outlier)))

print('Max. outlier value : '+ str(outlier.max()))

print('Min. outlier value : '+ str(outlier.min()))
```

```
Total number of outliers in feature RestingBP: 28
Max. outlier value : 200
Min. outlier value : 0
```

[31]:
```python
# Outliers in  Cholesterol
outlier = outliers_IQR(heartdf['Cholesterol'])

print('Total number of outliers in feature Cholesterol : '+ str(len(outlier)))

print('Max. outlier value : '+ str(outlier.max()))

print('Min. outlier value : '+ str(outlier.min()))
```

```
Total number of outliers in feature Cholesterol : 183
Max. outlier value : 603
Min. outlier value : 0
```

[32]:
```python
# Outliers in Oldpeak.

outlier = outliers_IQR(heartdf['Oldpeak'])

print('Total number of outliers in feature Oldpeak : '+ str(len(outlier)))

print('Max. outlier value : '+ str(outlier.max()))

print('Min. outlier value : '+ str(outlier.min()))
```

```
Total number of outliers in feature Oldpeak : 16
Max. outlier value : 6.2
Min. outlier value : -2.6
```

[33]:
```python
# Outliers in MaxHR.
outlier = outliers_IQR(heartdf['MaxHR'])

print('Total number of outliers in feature MaxHR : '+ str(len(outlier)))
```

```
print('Max. outlier value : '+ str(outlier.max()))

print('Min. outlier value : '+ str(outlier.min()))
```

```
Total number of outliers in feature MaxHR : 2
Max. outlier value : 63
Min. outlier value : 60
```

Observations:

- From the above we see that we have more outliers in features “RestingBP” and “Cholesterol” where we see minimum outlier value as ‘0’

[34]:
```
# Let's calculate the median of the 'Cholesterol' column so as to replace the␣
↪zeros with the median.
median = round(heartdf['Cholesterol'].median(),0)
median
```

[34]:
```
223.0
```

[35]:
```
# Let's replace the zeros from Cholesterol with the median.

heartdf['Cholesterol'].replace(0, heartdf['Cholesterol'].median(),inplace=True)
heartdf['Cholesterol']
```

[35]:
```
0      289
1      180
2      283
3      214
4      195
      …
913    264
914    193
915    131
916    236
917    175
Name: Cholesterol, Length: 918, dtype: int64
```

[36]:
```
# Let's calculate the median of the 'RestingBP' column so as to replace the␣
↪zeros with the median.
median = round(heartdf['RestingBP'].median(),0)
median
```

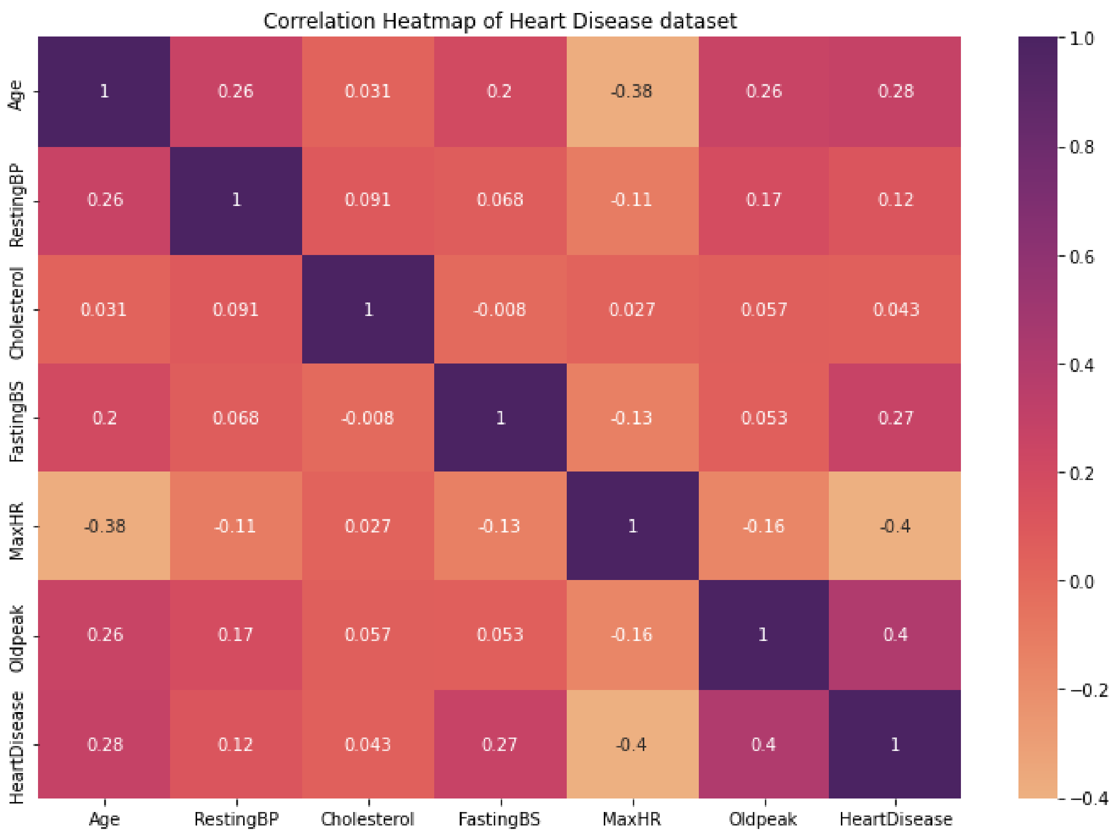
[36]:
```
130.0
```

[37]:
```
# Now let's replace the zeros from RestingBP with the median.
```

```
heartdf['RestingBP'].replace(0, heartdf['RestingBP'].median(),inplace=True)
```

[38]:
```
# Let's plot a correlation heatmap of the heart disease dataset.

# Calculate the correlation coefficient with corr().
corr_number = heartdf.corr()

# Create the heatmap for the correlation coefficients calculated above.
fig, ax = plt.subplots(1, 1, figsize=(10,7), tight_layout = True)
sns.heatmap(corr_number, annot = True, cmap = 'flare')

# Title of the plot
plt.title('Correlation Heatmap of Heart Disease dataset')
```

[38]: `Text(0.5, 1.0, 'Correlation Heatmap of Heart Disease dataset')`



Observations:

- From the above correlation map, we can see the correlation between HeartDisease and the remaining features has been improved after the removal of outliers.

- We can see that Age, RestingBP,Oldpeak are having high correlation with the target "HeartDisease."

[39]:
```
# Now that we have replaced the zeros
# Let's create dummy variables for the categorical features.
# Consider the categorical variables.
categ_col = heartdf.select_dtypes(include=object).columns

# Creating dummy variables using pd.get_dummies()
heartdf = pd.get_dummies(heartdf,columns=categ_col, drop_first=True)

# Print the first five rows of the dataset after creating the dummy variables␣
↪for the categorical features.
heartdf.head()
```

[39]:
```
   Age  RestingBP  Cholesterol  FastingBS  MaxHR  Oldpeak  HeartDisease  \
0   40        140          289          0    172      0.0             0
1   49        160          180          0    156      1.0             1
2   37        130          283          0     98      0.0             0
3   48        138          214          0    108      1.5             1
4   54        150          195          0    122      0.0             0

   Sex_M  ChestPainType_ATA  ChestPainType_NAP  ChestPainType_TA  \
0      1                  1                  0                 0
1      0                  0                  1                 0
2      1                  1                  0                 0
3      0                  0                  0                 0
4      1                  0                  1                 0

   RestingECG_Normal  RestingECG_ST  ExerciseAngina_Y  ST_Slope_Flat  \
0                  1              0                 0              0
1                  1              0                 0              1
2                  0              1                 0              0
3                  1              0                 1              1
4                  1              0                 0              0

   ST_Slope_Up
0            1
1            0
2            1
3            0
4            1
```

[40]:
```
# Getting the shape of the heart dataset
heartdf.shape
```

[40]:
```
(918, 16)
```

[41]:
```
# Checking for any nulll values after creating dummy variables.
heartdf.isnull().sum().sum()
```

[41]: 0

[42]:
```
# Let's define the features and target variables X and y respectively.

X = heartdf.drop('HeartDisease', axis = 1)
y = heartdf['HeartDisease']
```

[43]:
```
# Let's split the dataset into 80% train and 20% test datasets using␣
↪train_test_split().
# Test size is 0.2
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
↪random_state = 42)
```

[44]:
```
# Let's get the size of the test and train datasets.

print("X_train shape : {} rows and {} columns.".format(X_train.shape[0],X_train.
↪shape[1]))
print("y_train shape : {} rows.".format(y_train.shape[0]))
print("X_test shape : {} rows and {} columns.".format(X_test.shape[0],X_test.
↪shape[1]))
print("y_test shape : {} rows.".format(y_test.shape[0]))
```

```
X_train shape : 734 rows and 15 columns.
y_train shape : 734 rows.
X_test shape : 184 rows and 15 columns.
y_test shape : 184 rows.
```

Observations :

- After creating the dummy varibales the size of the dataset if 918 rows and 16 columns, dropped the first column while creating the dummy varibales.
- The following are the size of the test and train data sets shape.
  a. X_train shape : 734 rows and 15 columns.
  b. y_train shape : 734 rows.
  c. X_test shape : 184 rows and 15 columns.
  d. y_test shape : 184 rows.

[45]:
```
# Using the standard scaler on X_train and X_test datasets.
# Fit the transform.

standscaler = StandardScaler()
X_train_std = standscaler.fit_transform(X_train)
X_test_std = standscaler.transform(X_test)
```

[46]:
```
# Let's create a PCA that will retain 90% of the variance
pcah = PCA(n_components=0.90, whiten = True)
```

[47]:
```
# Conduct PCA

X_train_pcah = pcah.fit_transform(X_train_std)
```

[48]:
```
# Print the number of features of the train dataset.
print('Original number of features of the train dataset:', X_train.shape[1])
# Print the features of the PCA transformed train dataset.
print('Reduced number of features of the train dataset :', X_train_pcah.
 ↪shape[1])
```

```
Original number of features of the train dataset: 15
Reduced number of features of the train dataset : 11
```

[49]:
```
# Let's use the transform() method on the test features with PCA retaining 90%␣
 ↪of the variance but not fit the transform.

X_test_pcah = pcah.transform(X_test_std)
```

Observations:

- The heart dataset is split into 80% training and 20% test datasets. The features considered here are 15.
- I have used the Principal Component Analysis(PCA) and the train test data sets have been split, with this the features have been reduced from 15 to 11.

0.11 Summary from Milestone 2 : Data Preparation

- Removed outliers identified in the dataset using the interquartile range method.
- The columns 'Cholesterol' and 'RestingBP'are having zeros, the zeros are replaced with the median value of the respective columns.
- Correlation heat map is plotted and identified the features that are having good correlation to the target varibale.
- Dummy variables are created for the categorical columns and the data set is split into 80% test and 20% train datasets.
- Principal Component Analysis(PCA) is applied and second set of train and test data sets are created, with this the features have been reduced from 15 to 11.
- The data is now prepared for performing different model building and evaluation.

0.12 Milestone 3: Model Building and Evaluation

- Here in this Milestone 3 I would like to build four models for the test and train datasets and the PCA applied test and train datasets. For this Heart Disease prediction dataset, I would

like to build 4 models, they are Logistic Regression model, K- Nearest Neighbour Classifier, Decision Tree Classifier, Support Vector Machine(SVM).
- All the models will be created,trained and fit with the test and train datasets and PCA applied test and train datasets respectively.
- For the selection of the best model that fits our data will be evaluated based on the metrics Accuracy, Precision, Recall and F1 Score.
- I would like to use the Confusion Matrix that summarizes the performance of the model build. After observing the evaluation metrics calculated for each model with the test dataset and the PCA applied test and train datasets, the best model that fits the dataset, is selected. A short conclusion of the results obtained will be summarized at last.

0.12.1 Logistic Regression Model :

[50]:
```
# Create a Logistic Regression Model.

lgreg = LogisticRegression()

# Fitting the Logistic Regression Model.
lgreg = lgreg.fit(X_train, y_train)

# Let's get the predictions using the test dataset.
lgreg_pred = lgreg.predict(X_test)

# Let's get the predictions using the train dataset.
lgreg_pred_train = lgreg.predict(X_train)
```

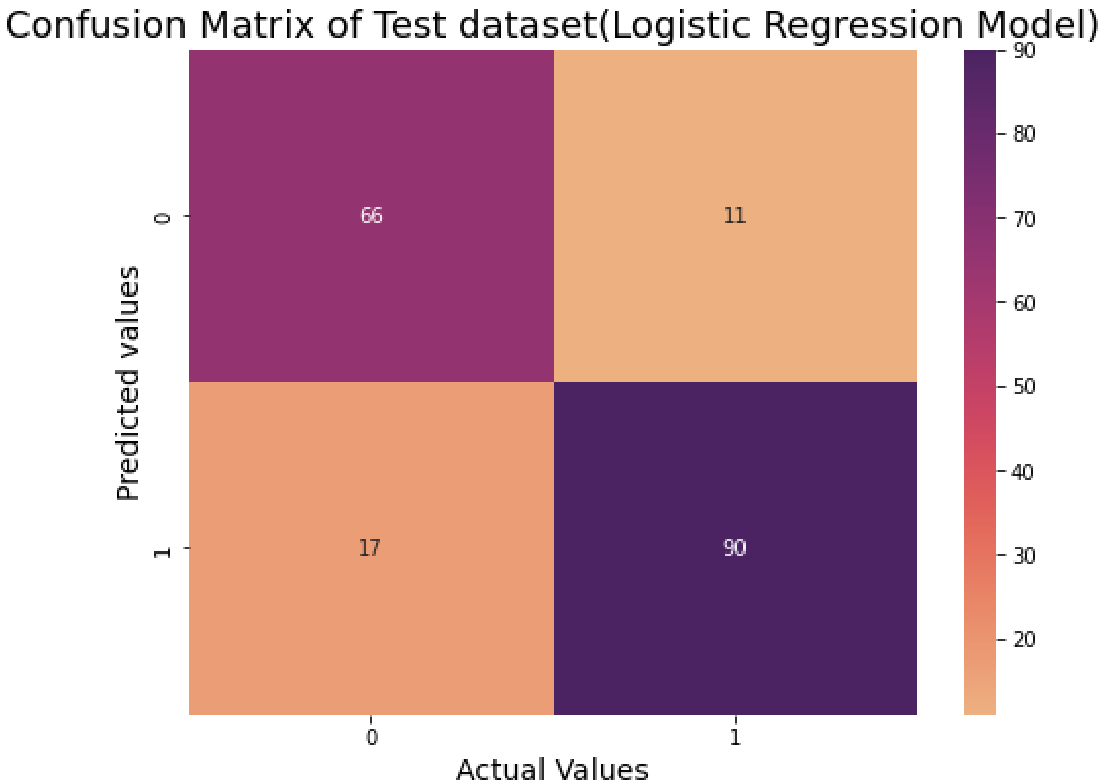
[51]:
```
# Let's create a Confusion Matrix for the test set predictions.
conmatrix = confusion_matrix(y_test, lgreg_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(Logistic Regression Model) :␣
↪\n', conmatrix)
```

```
Confusion Matrix of Test set predictions(Logistic Regression Model) :
 [[66 11]
 [17 90]]
```

[52]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(conmatrix, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test dataset(Logistic Regression Model)',␣
↪fontsize = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
```

```
plt.ylabel('Predicted values', fontsize = 14)
```

[52]: Text(51.0, 0.5, 'Predicted values')



[53]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the logistic␣
↪regression model.
# Getting the accuracy score of the test dataset.
lgreg_accuracy = metrics.accuracy_score(y_test, lgreg_pred)

# Getting the accuracy score of the train dataset
lgreg_accu_train = metrics.accuracy_score(y_train, lgreg_pred_train)

# Getting the precision score.
lgreg_precision = round(precision_score(y_test, lgreg_pred),3)

# Getting the Recall Score.
lgreg_recall = round(recall_score(y_test, lgreg_pred),3)

# Getting the F1 Score.
lgreg_f1score = round(f1_score(y_test, lgreg_pred),3)
```

```python
# Printing the accuracy of the model.
print('The accuracy score of the Logistic Regression Model on test dataset: {}␣
↪'.format(lgreg_accuracy))
print('The accuracy score of the Logistic Regression Model on train dataset :␣
↪{} '.format(lgreg_accu_train))
print('Precision Score of the test set for the Logistic Regression Model : {}'.
↪format(lgreg_precision))
print('Recall Score of the test set for the Logistic Regression Model : {}'.
↪format(lgreg_recall))
print('F1 Score of the test set for the Logistic Regression Model : {}'.
↪format(lgreg_f1score))
```

```
The accuracy score of the Logistic Regression Model on test dataset:
0.8478260869565217
The accuracy score of the Logistic Regression Model on train dataset :
0.8746594005449592
Precision Score of the test set for the Logistic Regression Model : 0.891
Recall Score of the test set for the Logistic Regression Model : 0.841
F1 Score of the test set for the Logistic Regression Model : 0.865
```

0.12.2 Logistic Regression Model (PCA):

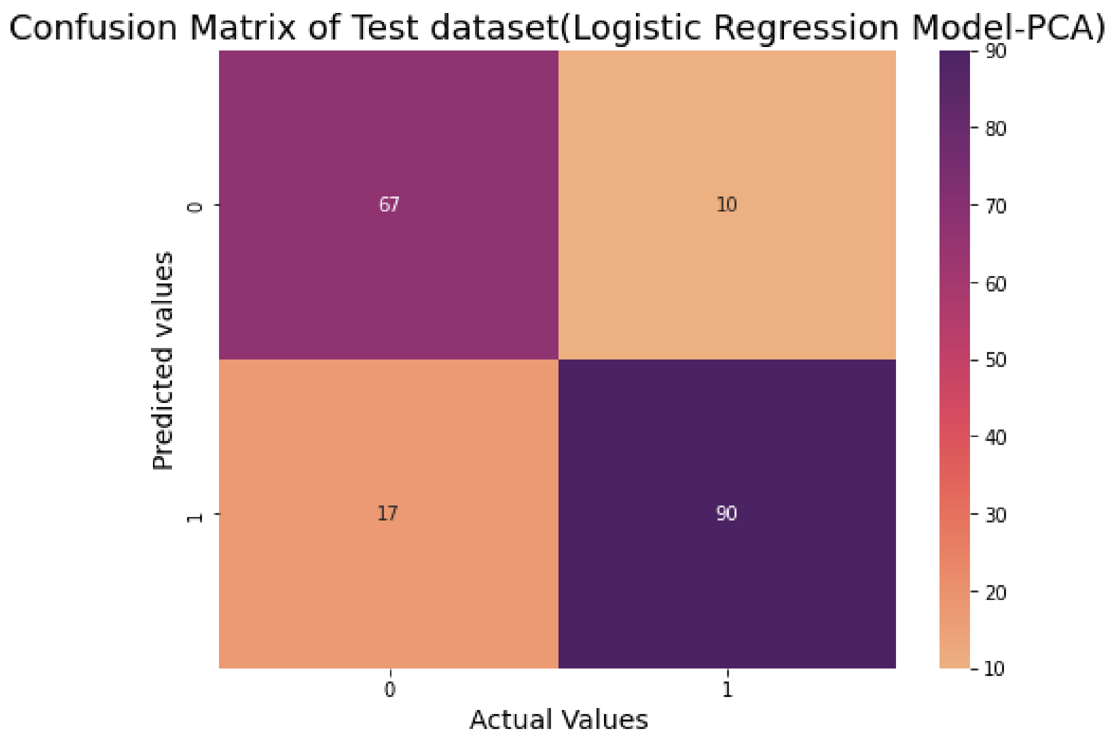
[54]:
```python
# Create a Logistic Regression Model(PCA)
lgreg_pcah = LogisticRegression()

# Fit the model  to  train datasets(PCA)
lgreg_pcah.fit(X_train_pcah, y_train)

# Create prediction of the model using the test data(PCA)
lgreg_pcah_pred = lgreg_pcah.predict(X_test_pcah)

# Create prediction of the model using the train data(PCA)
lgreg_pcah_pred_train = lgreg_pcah.predict(X_train_pcah)
```

[55]:
```python
# Let's create a Confusion Matrix for the test set predictions.
conmatrix_pca = confusion_matrix(y_test, lgreg_pcah_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(Logistic Regression Model-PCA) :
↪ \n', conmatrix_pca)
```

```
Confusion Matrix of Test set predictions(Logistic Regression Model-PCA) :
 [[67 10]
 [17 90]]
```

[56]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(conmatrix_pca, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test dataset(Logistic Regression Model-PCA)',␣
↪fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values',fontsize = 14)
```

[56]:
```
Text(51.0, 0.5, 'Predicted values')
```



[57]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the logistic␣
↪regression model(PCA).
# Getting the accuracy score of the test dataset.
lgreg_pca_accuracy = metrics.accuracy_score(y_test, lgreg_pcah_pred)

# Getting the accuracy score of the train dataset
lgreg_pca_accu_train = metrics.accuracy_score(y_train, lgreg_pcah_pred_train)
```

```
# Getting the precision score.
lgreg_pca_precision = round(precision_score(y_test, lgreg_pcah_pred),3)

# Getting the Recall Score.
lgreg_pca_recall = round(recall_score(y_test, lgreg_pcah_pred),3)

# Getting the F1 Score.
lgreg_pca_f1score = round(f1_score(y_test, lgreg_pcah_pred),3)


# Printing the accuracy of the model.
print('The accuracy score of the Logistic Regression Model (PCA) on test␣
↪dataset: {} '.format(lgreg_pca_accuracy))
print('The accuracy score of the Logistic Regression Model (PCA) on train␣
↪dataset : {} '.format(lgreg_pca_accu_train))
print('Precision Score of the test set for the Logistic Regression Model (PCA):␣
↪{}'.format(lgreg_pca_precision))
print('Recall Score of the test set for the Logistic Regression Model (PCA):␣
↪{}'.format(lgreg_pca_recall))
print('F1 Score of the test set for the Logistic Regression Model (PCA): {}'.
↪format(lgreg_pca_f1score))
```

```
The accuracy score of the Logistic Regression Model (PCA) on test dataset:
0.8532608695652174
The accuracy score of the Logistic Regression Model (PCA) on train dataset :
0.8678474114441417
Precision Score of the test set for the Logistic Regression Model (PCA): 0.9
Recall Score of the test set for the Logistic Regression Model (PCA): 0.841
F1 Score of the test set for the Logistic Regression Model (PCA): 0.87
```

0.12.3 K - Nearest Neighbour Classifier Model :

[58]:
```
# Create KNN classifier
knnclass = KNeighborsClassifier()

# Fit the model to train datasets.
knnclass = knnclass.fit(X_train, y_train )

# Create prediction of the model using the test data.
knnclass_pred = knnclass.predict(X_test)

# Create prediction of the model using the train data.
knnclass_pred_train = knnclass.predict(X_train)
```

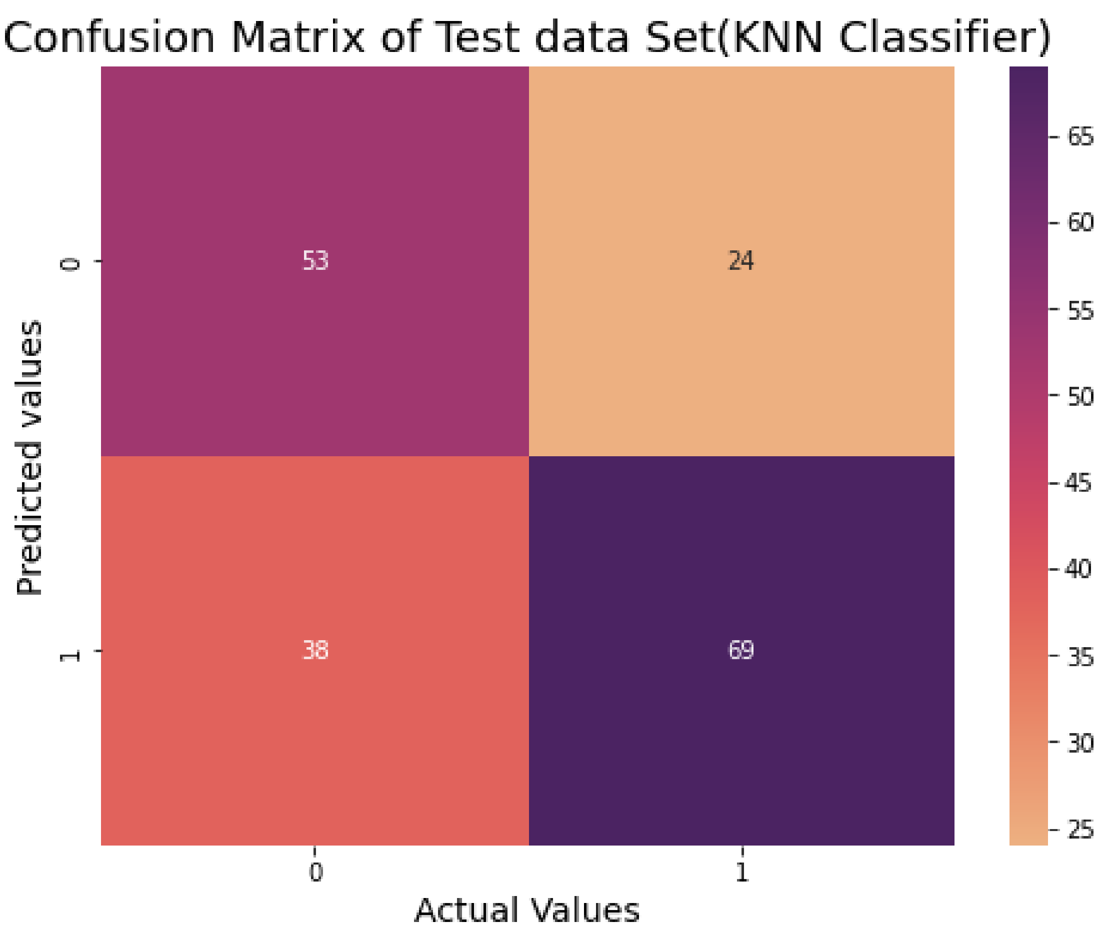
[59]:
```
# Let's create a Confusion Matrix for the test set predictions.
knnconmatrix = confusion_matrix(y_test, knnclass_pred)
```

```
# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(KNN Classifier) : \n',␣
↪knnconmatrix)
```

```
Confusion Matrix of Test set predictions(KNN Classifier) :
 [[53 24]
 [38 69]]
```

[60]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(knnconmatrix, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(KNN Classifier)', fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[60]:
```
Text(51.0, 0.5, 'Predicted values')
```



[61]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the KNN␣
↪Classifier.
# Getting the accuracy score of the test dataset.
knnclass_accuracy = metrics.accuracy_score(y_test, knnclass_pred)

# Getting the accuracy score of the train dataset
knnclass_accu_train = metrics.accuracy_score(y_train, knnclass_pred_train)

# Getting the precision score.
knnclass_precision = round(precision_score(y_test, knnclass_pred),3)

# Getting the Recall Score.
knnclass_recall = round(recall_score(y_test, knnclass_pred),3)

# Getting the F1 Score.
knnclass_f1score = round(f1_score(y_test, knnclass_pred),3)


# Printing the accuracy of the model.
print('The accuracy score of the KNN Classifier on test dataset: {} '.
↪format(knnclass_accuracy))
print('The accuracy score of the KNN Classifier on train dataset : {} '.
↪format(knnclass_accu_train))
print('Precision Score of the test set for the KNN Classifier : {}'.
↪format(knnclass_precision))
print('Recall Score of the test set for the KNN Classifier : {}'.
↪format(knnclass_recall))
print('F1 Score of the test set for the KNN Classifier : {}'.
↪format(knnclass_f1score))
```

```
The accuracy score of the KNN Classifier on test dataset: 0.6630434782608695
The accuracy score of the KNN Classifier on train dataset : 0.779291553133515
Precision Score of the test set for the KNN Classifier : 0.742
Recall Score of the test set for the KNN Classifier : 0.645
F1 Score of the test set for the KNN Classifier : 0.69
```

0.12.4 K - Nearest Neighbour Classifier Model (PCA) :

[62]:
```
# Create KNN classifier
knnclass_pca = KNeighborsClassifier()

# Fit the model to train datasets.
knnclass_pca = knnclass_pca.fit(X_train_pcah, y_train )
```

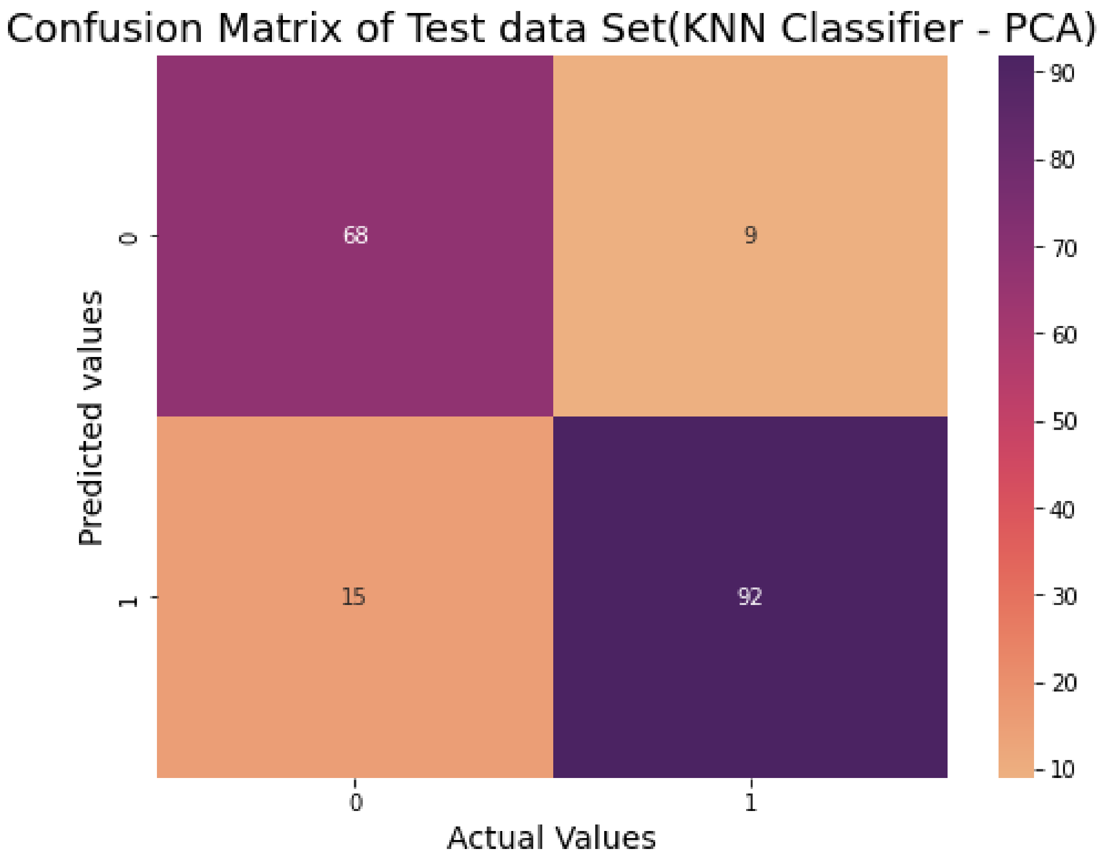
```
# Create prediction of the model using the test data.
knnclass_pca_pred = knnclass_pca.predict(X_test_pcah)

# Create prediction of the model using the train data.
knnclass_pca_pred_train = knnclass_pca.predict(X_train_pcah)
```

[63]:
```
# Let's create a Confusion Matrix for the test set predictions.
knnconmatrix_pca = confusion_matrix(y_test, knnclass_pca_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(KNN Classifier - PCA) : \n',␣
↪knnconmatrix_pca)
```

```
Confusion Matrix of Test set predictions(KNN Classifier - PCA) :
 [[68  9]
 [15 92]]
```

[64]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(knnconmatrix_pca, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(KNN Classifier - PCA)', fontsize ␣
↪= 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[64]:
```
Text(51.0, 0.5, 'Predicted values')
```




```
[65]: # Let's get the accuracy score, Precision, Recall and F1 Score of the KNN␣
      ↪Classifier(PCA).
      # Getting the accuracy score of the test dataset.
      knnclass_pca_accuracy = metrics.accuracy_score(y_test, knnclass_pca_pred)

      # Getting the accuracy score of the train dataset
      knnclass_pca_accu_train = metrics.accuracy_score(y_train,␣
      ↪knnclass_pca_pred_train)

      # Getting the precision score.
      knnclass_pca_precision = round(precision_score(y_test, knnclass_pca_pred),3)

      # Getting the Recall Score.
      knnclass_pca_recall = round(recall_score(y_test, knnclass_pca_pred),3)

      # Getting the F1 Score.
      knnclass_pca_f1score = round(f1_score(y_test, knnclass_pca_pred),3)
```

```
# Printing the accuracy of the model.
print('The accuracy score of the KNN Classifier(PCA) on test dataset: {} '.
 ↪format(knnclass_pca_accuracy))
print('The accuracy score of the KNN Classifier(PCA) on train dataset : {} '.
 ↪format(knnclass_pca_accu_train))
print('Precision Score of the test set for the KNN Classifier(PCA) : {}'.
 ↪format(knnclass_pca_precision))
print('Recall Score of the test set for the KNN Classifier(PCA) : {}'.
 ↪format(knnclass_pca_recall))
print('F1 Score of the test set for the KNN Classifier(PCA) : {}'.
 ↪format(knnclass_pca_f1score))
```

```
The accuracy score of the KNN Classifier(PCA) on test dataset:
0.8695652173913043
The accuracy score of the KNN Classifier(PCA) on train dataset :
0.8773841961852861
Precision Score of the test set for the KNN Classifier(PCA) : 0.911
Recall Score of the test set for the KNN Classifier(PCA) : 0.86
F1 Score of the test set for the KNN Classifier(PCA) : 0.885
```

0.12.5 Decision Tree Classifier Model :

[66]:
```
# Create Decision Tree classifier
dtclass = DecisionTreeClassifier()

# Fit the model to train datasets.
dtclass = dtclass.fit(X_train, y_train )

# Create prediction of the model using the test data.
dtclass_pred = dtclass.predict(X_test)

# Create prediction of the model using the train data.
dtclass_pred_train = dtclass.predict(X_train)
```

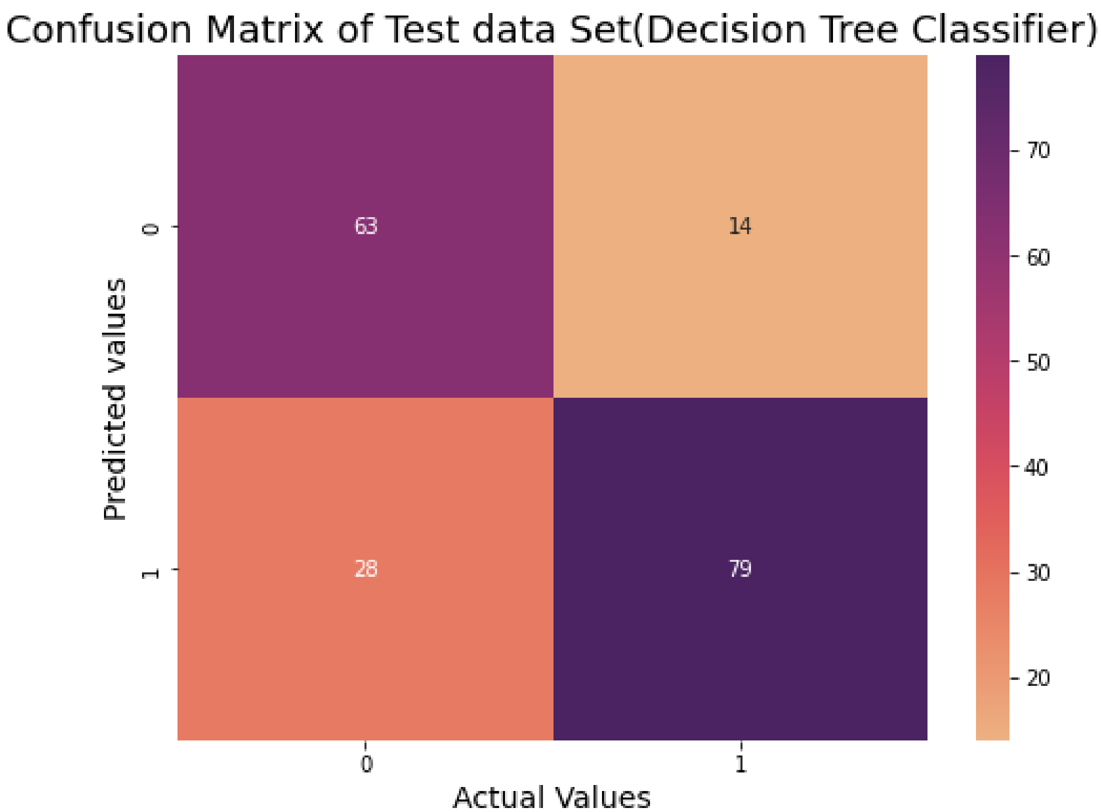
[67]:
```
# Let's create a Confusion Matrix for the test set predictions.
dtconmatrix = confusion_matrix(y_test, dtclass_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(Decision Tree Classifier): \n',␣
 ↪dtconmatrix)
```

```
Confusion Matrix of Test set predictions(Decision Tree Classifier):
 [[63 14]
 [28 79]]
```

[68]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
```

```
#  Confusion matrix heat map.
sns.heatmap(dtconmatrix, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(Decision Tree Classifier)',␣
↪fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[68]: Text(51.0, 0.5, 'Predicted values')



[69]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the Decision␣
↪Tree Classifier.
# Getting the accuracy score of the test dataset.
dtclass_accuracy = metrics.accuracy_score(y_test, dtclass_pred)

# Getting the accuracy score of the train dataset
dtclass_accu_train = metrics.accuracy_score(y_train, dtclass_pred_train)

# Getting the precision score.
```

```
dtclass_precision = round(precision_score(y_test, dtclass_pred),3)

# Getting the Recall Score.
dtclass_recall = round(recall_score(y_test, dtclass_pred),3)

# Getting the F1 Score.
dtclass_f1score = round(f1_score(y_test, dtclass_pred),3)


# Printing the accuracy of the model.
print('The accuracy score of the Decision Tree Classifier on test dataset: {} '.
 ↪format(dtclass_accuracy))
print('The accuracy score of the Decision Tree Classifier on train dataset : {}␣
 ↪'.format(dtclass_accu_train))
print('Precision Score of the test set for the Decision Tree Classifier : {}'.
 ↪format(dtclass_precision))
print('Recall Score of the test set for the Decision Tree Classifier : {}'.
 ↪format(dtclass_recall))
print('F1 Score of the test set for the Decision Tree Classifier : {}'.
 ↪format(dtclass_f1score))
```

```
The accuracy score of the Decision Tree Classifier on test dataset:
0.7717391304347826
The accuracy score of the Decision Tree Classifier on train dataset : 1.0
Precision Score of the test set for the Decision Tree Classifier : 0.849
Recall Score of the test set for the Decision Tree Classifier : 0.738
F1 Score of the test set for the Decision Tree Classifier : 0.79
```

0.12.6 Decision Tree Classifier Model (PCA) :

[70]:
```
# Create Decision Tree classifier
dtclass_pca = DecisionTreeClassifier()

# Fit the model to train datasets.
dtclass_pca = dtclass_pca.fit(X_train_pcah, y_train )

# Create prediction of the model using the test data.
dtclass_pca_pred = dtclass_pca.predict(X_test_pcah)

# Create prediction of the model using the train data.
dtclass_pca_pred_train = dtclass_pca.predict(X_train_pcah)
```

[71]:
```
# Let's create a Confusion Matrix for the test set predictions.
dtconmatrix_pca = confusion_matrix(y_test, dtclass_pca_pred)

# Print the Confusion matrix.
```

```
print('Confusion Matrix of Test set predictions(Decision Tree Classifier- PCA):
↪\n', dtconmatrix_pca)
```

```
Confusion Matrix of Test set predictions(Decision Tree Classifier- PCA):
 [[64 13]
 [22 85]]
```

[72]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(dtconmatrix_pca, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(Decision Tree Classifier - PCA)',
↪fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[72]:
```
Text(51.0, 0.5, 'Predicted values')
```

[73]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the Decision␣
↪Tree Classifier(PCA).
# Getting the accuracy score of the test dataset.
dtclass_pca_accuracy = metrics.accuracy_score(y_test, dtclass_pca_pred)

# Getting the accuracy score of the train dataset
dtclass_pca_accu_train = metrics.accuracy_score(y_train, dtclass_pca_pred_train)

# Getting the precision score.
dtclass_pca_precision = round(precision_score(y_test, dtclass_pca_pred),3)

# Getting the Recall Score.
dtclass_pca_recall = round(recall_score(y_test, dtclass_pca_pred),3)

# Getting the F1 Score.
dtclass_pca_f1score = round(f1_score(y_test, dtclass_pca_pred),3)


# Printing the accuracy of the model.
print('The accuracy score of the Decision Tree Classifier (PCA)on test dataset:␣
↪{} '.format(dtclass_pca_accuracy))
print('The accuracy score of the Decision Tree Classifier (PCA)on train dataset␣
↪: {} '.format(dtclass_pca_accu_train))
print('Precision Score of the test set for the Decision Tree Classifier(PCA) :␣
↪{}'.format(dtclass_pca_precision))
print('Recall Score of the test set for the Decision Tree Classifier (PCA): {}'.
↪format(dtclass_pca_recall))
print('F1 Score of the test set for the Decision Tree Classifier (PCA) : {}'.
↪format(dtclass_pca_f1score))
```

```
The accuracy score of the Decision Tree Classifier (PCA)on test dataset:
0.8097826086956522
The accuracy score of the Decision Tree Classifier (PCA)on train dataset : 1.0
Precision Score of the test set for the Decision Tree Classifier(PCA) : 0.867
Recall Score of the test set for the Decision Tree Classifier (PCA): 0.794
F1 Score of the test set for the Decision Tree Classifier (PCA) : 0.829
```

0.12.7 Support Vector Machine Model :

[74]:
```
# Create Support Vector Machine Classifier

svmclass = SVC(kernel='linear', C=1)

# Fit the model to train datasets.
svmclass = svmclass.fit(X_train, y_train )

# Create prediction of the model using the test data.
```

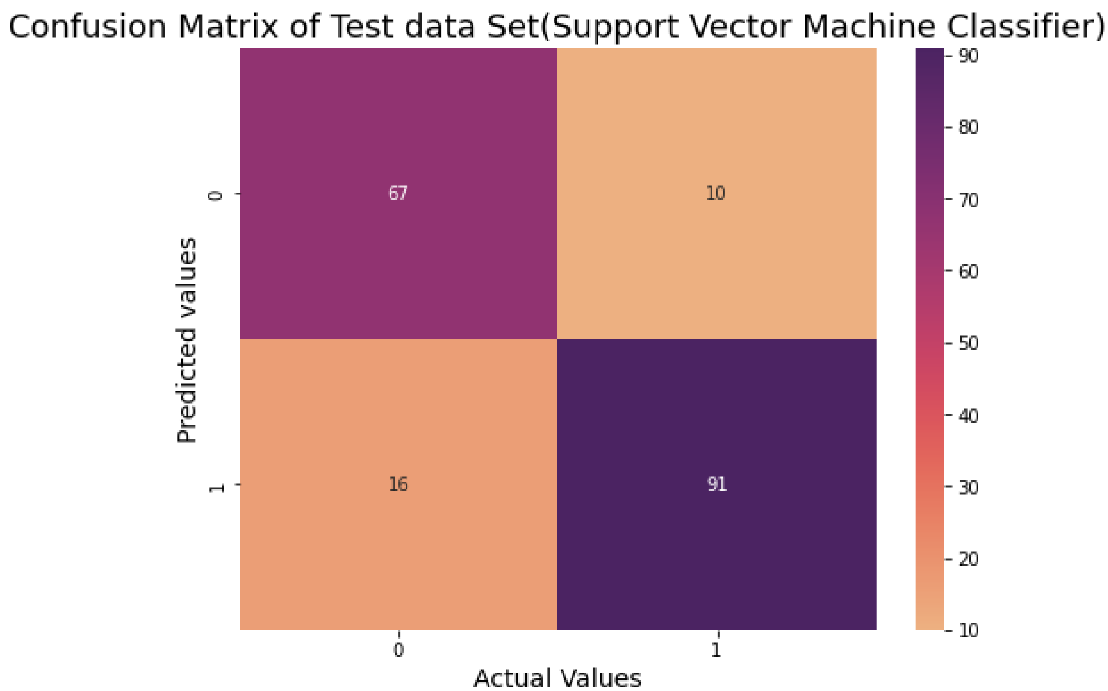
```
svmclass_pred = svmclass.predict(X_test)

# Create prediction of the model using the train data.
svmclass_pred_train = svmclass.predict(X_train)
```

[75]:
```
# Let's create a Confusion Matrix for the test set predictions.
svmconmatrix = confusion_matrix(y_test, svmclass_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(Support Vector Machine␣
↪Classifier): \n', svmconmatrix)
```

```
Confusion Matrix of Test set predictions(Support Vector Machine Classifier):
 [[67 10]
 [16 91]]
```

[76]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(svmconmatrix, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(Support Vector Machine␣
↪Classifier)', fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[76]:
```
Text(51.0, 0.5, 'Predicted values')
```



[77]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the Support␣
↪Vector Machine Classifier.
# Getting the accuracy score of the test dataset.
svmclass_accuracy = metrics.accuracy_score(y_test, svmclass_pred)

# Getting the accuracy score of the train dataset
svmclass_accu_train = metrics.accuracy_score(y_train, svmclass_pred_train)

# Getting the precision score.
svmclass_precision = round(precision_score(y_test, svmclass_pred),3)

# Getting the Recall Score.
svmclass_recall = round(recall_score(y_test, svmclass_pred),3)

# Getting the F1 Score.
svmclass_f1score = round(f1_score(y_test, svmclass_pred),3)


# Printing the accuracy of the model.
print('The accuracy score of the SVM Classifier on test dataset: {} '.
↪format(svmclass_accuracy))
print('The accuracy score of the SVM Classifier on train dataset : {} '.
↪format(svmclass_accu_train))
```

```
print('Precision Score of the test set for the SVM Classifier : {}'.
 ↪format(svmclass_precision))
print('Recall Score of the test set for the SVM Classifier : {}'.
 ↪format(svmclass_recall))
print('F1 Score of the test set for the SVM Classifier : {}'.
 ↪format(svmclass_f1score))
```

```
The accuracy score of the SVM Classifier on test dataset: 0.8586956521739131
The accuracy score of the SVM Classifier on train dataset : 0.8705722070844687
Precision Score of the test set for the SVM Classifier : 0.901
Recall Score of the test set for the SVM Classifier : 0.85
F1 Score of the test set for the SVM Classifier : 0.875
```

0.12.8 Support Vector Machine Model (PCA):

[78]:
```
# Create Support Vector Machine Classifier

svmclass_pca = SVC(kernel='linear', C=1)

# Fit the model to train datasets.
svmclass_pca = svmclass_pca.fit(X_train_pcah, y_train )

# Create prediction of the model using the test data.
svmclass_pca_pred = svmclass_pca.predict(X_test_pcah)

# Create prediction of the model using the train data.
svmclass_pca_pred_train = svmclass_pca.predict(X_train_pcah)
```

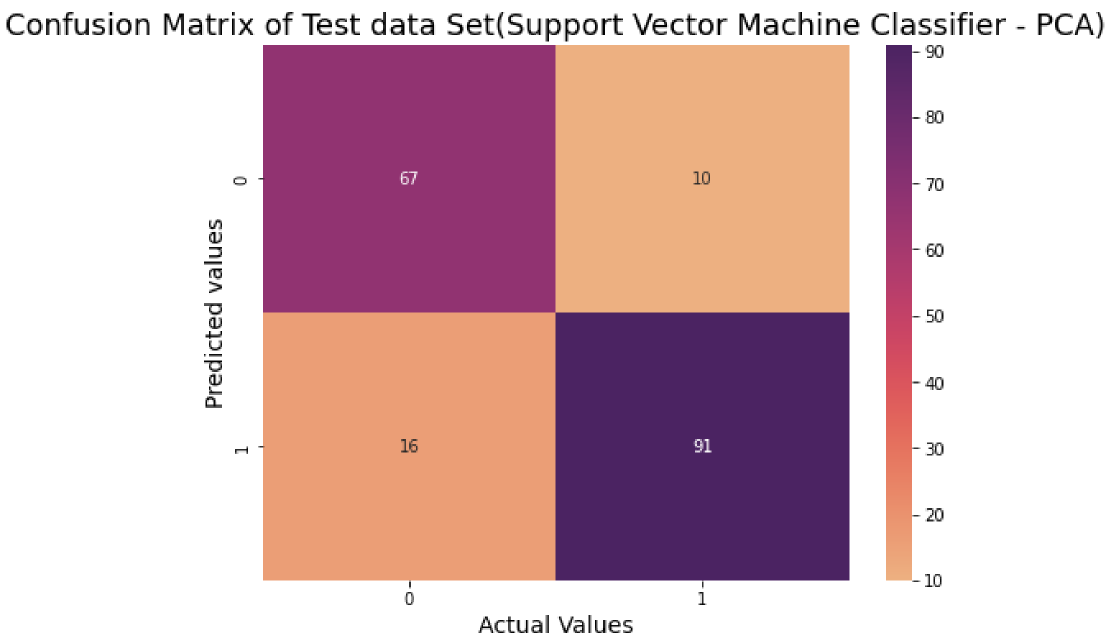
[79]:
```
# Let's create a Confusion Matrix for the test set predictions.
svmconmatrix_pca = confusion_matrix(y_test, svmclass_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(Support Vector Machine␣
 ↪Classifier - PCA): \n', svmconmatrix_pca)
```

```
Confusion Matrix of Test set predictions(Support Vector Machine Classifier -
PCA):
 [[67 10]
 [16 91]]
```

[80]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(svmconmatrix_pca, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(Support Vector Machine Classifier␣
 ↪- PCA)', fontsize  = 18)
```

```
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[80]: Text(51.0, 0.5, 'Predicted values')



[81]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the Support␣
↪Vector Machine Classifier(PCA).
# Getting the accuracy score of the test dataset.
svmclass_pca_accuracy = metrics.accuracy_score(y_test, svmclass_pca_pred)

# Getting the accuracy score of the train dataset
svmclass_pca_accu_train = metrics.accuracy_score(y_train,␣
↪svmclass_pca_pred_train)

# Getting the precision score.
svmclass_pca_precision = round(precision_score(y_test, svmclass_pca_pred),3)

# Getting the Recall Score.
svmclass_pca_recall = round(recall_score(y_test, svmclass_pca_pred),3)

# Getting the F1 Score.
svmclass_pca_f1score = round(f1_score(y_test, svmclass_pca_pred),3)
```

```
# Printing the accuracy of the model.
print('The accuracy score of the SVM Classifier(PCA) on test dataset: {} '.
 ↪format(svmclass_pca_accuracy))
print('The accuracy score of the SVM Classifier(PCA) on train dataset : {} '.
 ↪format(svmclass_pca_accu_train))
print('Precision Score of the test set for the SVM Classifier(PCA) : {}'.
 ↪format(svmclass_pca_precision))
print('Recall Score of the test set for the SVM Classifier(PCA) : {}'.
 ↪format(svmclass_pca_recall))
print('F1 Score of the test set for the SVM Classifier(PCA) : {}'.
 ↪format(svmclass_pca_f1score))
```

```
The accuracy score of the SVM Classifier(PCA) on test dataset:
0.8532608695652174
The accuracy score of the SVM Classifier(PCA) on train dataset :
0.8692098092643051
Precision Score of the test set for the SVM Classifier(PCA) : 0.892
Recall Score of the test set for the SVM Classifier(PCA) : 0.85
F1 Score of the test set for the SVM Classifier(PCA) : 0.871
```

0.12.9 Summary of the evaluation metrics calculated using trained and test datasets for the 4 models

[82]:
```
# Let's form arrays for the calculated accuracy score for the train and test␣
 ↪datasets, Precision,
# Recall and F1 Score for the above three models.

logistic_reg = {'Model':'Logistic Regression',
                'Accuracy (test)':lgreg_accuracy,
               'Accuracy (train)':lgreg_accu_train,
               'Precision score' :lgreg_precision,
               'Recall score'    :lgreg_recall,
                'F1 Score':lgreg_f1score,}

KNN_classifier = {'Model':'KNN Classifier',
                  'Accuracy (test)':knnclass_accuracy,
               'Accuracy (train)':knnclass_accu_train,
               'Precision score' :knnclass_precision,
               'Recall score'    :knnclass_recall,
                'F1 Score':knnclass_f1score,}

DecisionTree_classifier = {'Model':'Decision Tree Classifier',
                          'Accuracy (test)':dtclass_accuracy,
                           'Accuracy (train)':dtclass_accu_train,
                          'Precision score' :dtclass_precision,
                          'Recall score'    :dtclass_recall,
                           'F1 Score':dtclass_f1score,}
```

```
SVM_classifier = {'Model':'Support Vector Machine Model',
               'Accuracy (test)':svmclass_accuracy,
              'Accuracy (train)':svmclass_accu_train,
              'Precision score' :svmclass_precision,
              'Recall score'    :svmclass_recall,
               'F1 Score':svmclass_f1score,}
```

[83]:
```
# Let's group the results of the three models using the pd.series()
models_evalmetrics = pd.DataFrame({'Logistic Regression Model':pd.
 ↪Series(logistic_reg),
                        'KNN Classifier': pd.Series(KNN_classifier),
                       'Decision Tree Classifier Model':pd.
 ↪Series(DecisionTree_classifier),
                       'Support Vector Machine Model':pd.Series(SVM_classifier),
                      })

models_evalmetrics
```

[83]:
```
                 Logistic Regression Model  KNN Classifier  \
Model                  Logistic Regression  KNN Classifier
Accuracy (test)                   0.847826        0.663043
Accuracy (train)                  0.874659        0.779292
Precision score                      0.891           0.742
Recall score                         0.841           0.645
F1 Score                             0.865            0.69

                 Decision Tree Classifier Model  Support Vector Machine Model
Model                  Decision Tree Classifier  Support Vector Machine Model
Accuracy (test)                        0.771739                      0.858696
Accuracy (train)                            1.0                      0.870572
Precision score                           0.849                         0.901
Recall score                              0.738                          0.85
F1 Score                                   0.79                         0.875
```

0.12.10 Observations:

- From the above values of the metrics for all the four models build, we can say that the Support Vector Machine classifier Model performed well for the selected data set for Heart Disease Prediction.

0.12.11 Summary of the evaluation metrics calculated using PCA applied trained and test datasets for the 4 Models.

[84]:
```
logistic_reg = {'Model':'Logistic Regression(PCA)',
               'Accuracy (test)':lgreg_pca_accuracy,
              'Accuracy (train)':lgreg_pca_accu_train,
              'Precision score' :lgreg_pca_precision,
```

```
                'Recall score'    :lgreg_pca_recall,
                 'F1 Score':lgreg_pca_f1score,}

KNN_classifier = {'Model':'KNN Classifier(PCA)',
                   'Accuracy (test)':knnclass_pca_accuracy,
                'Accuracy (train)':knnclass_pca_accu_train,
                'Precision score' :knnclass_pca_precision,
                'Recall score'    :knnclass_pca_recall,
                 'F1 Score':knnclass_pca_f1score,}

DecisionTree_classifier = {'Model':'Decision Tree Classifier(PCA)',
                          'Accuracy (test)':dtclass_pca_accuracy,
                          'Accuracy (train)':dtclass_pca_accu_train,
                          'Precision score' :dtclass_pca_precision,
                          'Recall score'    :dtclass_pca_recall,
                           'F1 Score':dtclass_pca_f1score,}

SVM_classifier = {'Model':'Support Vector Machine Model(PCA)',
                  'Accuracy (test)':svmclass_pca_accuracy,
                'Accuracy (train)':svmclass_pca_accu_train,
                'Precision score' :svmclass_pca_precision,
                'Recall score'    :svmclass_pca_recall,
                 'F1 Score':svmclass_pca_f1score,}
```

[85]:
```
# Let's group the results of the three models using the pd.series()
models_evalmetrics = pd.DataFrame({'Logistic Regression Model':pd.
 ↪Series(logistic_reg),
                        'KNN Classifier': pd.Series(KNN_classifier),
                       'Decision Tree Classifier Model':pd.
 ↪Series(DecisionTree_classifier),
                       'Support Vector Machine Model':pd.Series(SVM_classifier),
                      })

models_evalmetrics
```

[85]:
```
                  Logistic Regression Model       KNN Classifier  \
Model              Logistic Regression(PCA)  KNN Classifier(PCA)
Accuracy (test)                    0.853261             0.869565
Accuracy (train)                   0.867847             0.877384
Precision score                         0.9                0.911
Recall score                          0.841                 0.86
F1 Score                               0.87                0.885

                  Decision Tree Classifier Model  \
Model              Decision Tree Classifier(PCA)
Accuracy (test)                         0.809783
Accuracy (train)                             1.0
```

```
Precision score                                  0.867
Recall score                                     0.794
F1 Score                                         0.829

                       Support Vector Machine Model
Model             Support Vector Machine Model(PCA)
Accuracy (test)                               0.853261
Accuracy (train)                               0.86921
Precision score                                  0.892
Recall score                                      0.85
F1 Score                                         0.871
```

0.12.12 Observations:

- Here the evaluation metrics values are obtained using the PCA trained and test datasets.
- From the above we can see that the Support Vector Machine Model, and the Logistic Regression Model performed well for the selected data set.
- The accuracy score and the evaluation metrics values are good for the Logistic Regression Model and the SUpport Vector Machine Model.

0.13 Summary From Milestone 3 : Model Building and Evaluation:

- I have selected four Machine Learning models to build, they are Logistic Regression model, KNN (K - Nearest Neighbour)Classifier, Decision Tree classifier, Support Vector Machine Model.
- All the four models are created and trained using the training dataset and PCA applied train datasets.
- For evaluating the models I have generated the Confusion Matrix,and computed Accuracy Score, Precision, Recall and F1 score for all the four models respectively.
- All the four models are evaluated using the test dataset and the PCA applied test datasets respectively.
- The summary of the metrcis calculated for the models are summarized above.
- When observed the summary metrics calculated using the actual train and test datasets, the Support Vector Machine Classifier Model performed best.The accuracy score obtained is 0.858(~86%).
- The second best model is the Logistic Regression Model with accuracy score of 0.847(~85%).
- The KNN classifier performed with low accuracy score and the Decision Tree classifier accuracy score is 0.77 but the accuracy of the trained dataset is 1.0 indicating there is somee overfit with the trained dataset.
- When used the PCA applied training and test datasets the KNN classifier is best performed model, which improved the performance with accuracy score 0.869(~87%) when compared to the performance of the actual trained and test datasets.

- When observed the summary metrics calculated using the PCA applied trian and test datasets,the Logistic Regression Model with accuracy score 0.853(85%)and Support Vector Machine Model with accuracy score 0.853(85%) performed second best.
- The Decision Tree classifier model performance didn't change even after using the PCA applied trained dataset.
- Both the KNN Classifier, Logistic Regression and Support Vector Machine models performed good with good accuracy scores when the features are reduced from the 15 to 11 with the PCA application.
- For future recommedations for this project I would like to perform the Hyperparameter tuning on the models and check the model performance,and for evaluation Cross Validation can be included for model evaluation.

[ ]:
```

```