Applied ML Assignment 1

Submission by Akhil S - 2021MT12054

(This notebook along with it's PDF version can be found in this Github Repo)

Heart Attack Analaysis

Introduction

A heart attack occurs when an artery supplying your heart with blood and oxygen becomes blocked. A blood clot can form and block your arteries, causing a heart attack. This Heart Attack Analysis helps to understand the chance of attack occurrence in persons based on varied health conditions.

Dataset

The dataset is `Heart_Attack_Analysis_Data.csv`. It has been uploaded to elearn. This dataset contains data about some hundreds of patients mentioning:

- Age
- Sex
- Exercise Include Angina(1=YES, 0=NO)
- CP_Type (Chest Pain) (Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic)
- ECG Results
- Blood Pressure
- Cholesterol
- Blood Sugar
- Family History (Number of persons affected in the family)
- Maximum Heart Rate
- Target (0 = LESS CHANCE , 1 = MORE CHANCE)

Aim

- Building a Predictive Model using Naïve Bayesian Approach (Which features decide heart attack?)
- Comment on the performance of this model using AUC-ROC, Precision, Recall, F_score, Accuracy

You need to

1. Preprocess the data to enhance quality
2. Carry out descriptive summarization of data and make observations
3. Identify relevant, irrelevant attributes for building model.
4. Use data visualization tools and make observations
5. Carry out the chosen analytic task. Show results including intermediate results, as needed
6. Evaluate the solution

Following are some points for you to take note of, while doing the assignment in Jupyter Notebook:

- State all your assumptions clearly
- List all intermediate steps and learnings
- Mention your observations/findings

My Objectives

The following analysis will be done to the dataset that is provided:

1. Verify the datatypes of the values given in the dataset and validate with the information given in the document.
2. Check for invalid values based on domain knowledge by checking if values are present in humanly possible ranges.
3. Figure out which columns are numeric and categorical based on the unique values each column has and based on information given in the assignment document.
4. Check for trends among numerical features and among categorical features to see if feature reduction can be done (via pairplots etc)
5. Scale numerical attributes with a standard scaler.
6. Check for outliers via boxplots and remove them with IQR method.
7. One hot encode categorical data.

This notebook submission compares accuracy, Precision, Recall, F-Score and AOC-ROC of three models trained with data in different stages of data preprocessing:

1. **MODEL 1:** Model fit with data where only scaling is done to numerical data.
2. **MODEL 2:** Model fit with data where scaling is done to numerical data, and outliers are removed using the IQR method.
3. **MODEL 3:** Model fit with data where scaling is done to numerical data, outliers are removed using the IQR method and one hot encoding is done to categorical data

My Solution

Importing necessary packages

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sb
import warnings

from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score,
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings('ignore')
```

Data Cleaning, Preprocessing

Check the shape of the dataframe loaded into memory from the CSV file and see the datatypes used in the dataset given

In [2]:
```
df = pd.read_csv("./Heart_Attack_Analysis_Data.csv")
print("Dataframe Shape: {}".format(df.shape))
print("----------------------------------\n")
print("With following data types:\n")
df.info()
print("----------------------------------\n")
print("First 5 rows of Dataframe:")
df.head()
```

```
Dataframe Shape: (303, 11)
----------------------------------

With following data types:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             303 non-null    int64
 1   Sex             303 non-null    int64
 2   CP_Type         303 non-null    int64
 3   BloodPressure   303 non-null    int64
 4   Cholestrol      303 non-null    int64
 5   BloodSugar      303 non-null    int64
 6   ECG             303 non-null    int64
 7   MaxHeartRate    303 non-null    int64
 8   ExerciseAngina  303 non-null    int64
 9   FamilyHistory   303 non-null    int64
 10  Target          303 non-null    int64
dtypes: int64(11)
memory usage: 26.2 KB
----------------------------------

First 5 rows of Dataframe:
```

Out[2]:

	Age	Sex	CP_Type	BloodPressure	Cholestrol	BloodSugar	ECG	MaxHeartRate	ExerciseAngina	Fami
0	63	1	3	145	233	1	0	150	0	
1	37	1	2	130	250	0	1	187	0	
2	41	0	1	130	204	0	0	172	0	
3	56	1	1	120	236	0	1	178	0	
4	57	0	0	120	354	0	1	163	1	

All columns are `integer` type and all columns have a value for all rows (303 not null), which means that all values are filled and there are no missing values.

Now to check certain columns with humanly possible ranges based on domain knowledge:

1. 0 < Age <= 100 years
2. 90 <= BloodPressure <= 200
3. 60 <= MaxHeartRate <= 220

In [3]:

```
print("\nMinimum Age = {}".format(df["Age"].min()))
print("Maximum Age = {}".format(df["Age"].max()))

print("\nMinimum Blood Pressure = {}".format(df["BloodPressure"].min()))
print("Maximum Blood Pressure = {}".format(df["BloodPressure"].max()))

print("\nMinimum Heart Rate = {}".format(df["MaxHeartRate"].min()))
print("Maximum Heart Rate = {}".format(df["MaxHeartRate"].max()))
```

```
Minimum Age = 29
Maximum Age = 77
```

```
Minimum Blood Pressure = 94
Maximum Blood Pressure = 200

Minimum Heart Rate = 71
Maximum Heart Rate = 202
```

All of the mentioned columns have values within acceptable ranges.

Checking number of unique values for each column

`Column Name -> Unique Number count`

In [4]:
```
for column in list(df.columns):
    print("{} -> {}".format(column, df[column].value_counts().shape[0]))
```

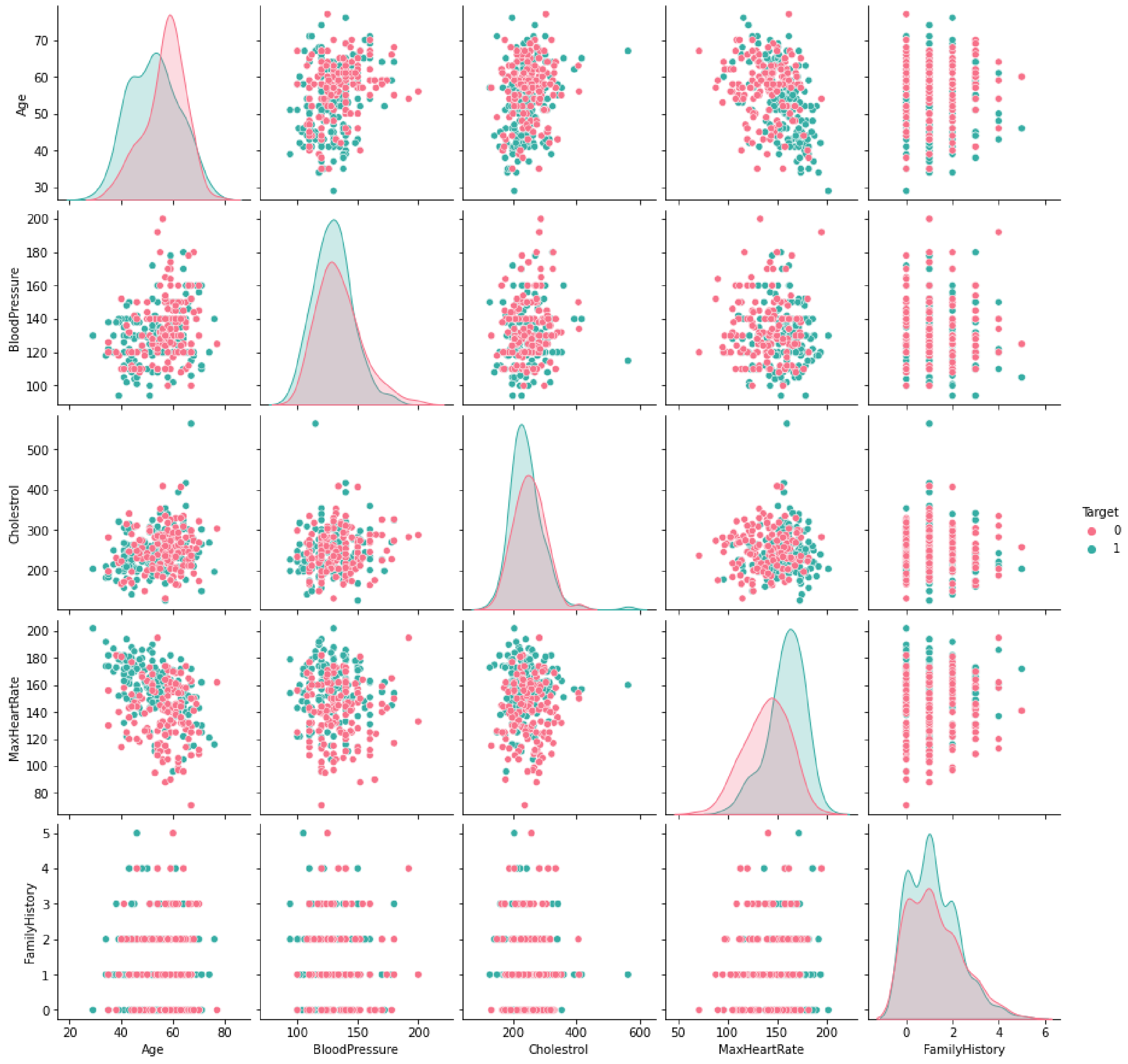
```
Age -> 41
Sex -> 2
CP_Type -> 4
BloodPressure -> 49
Cholestrol -> 152
BloodSugar -> 2
ECG -> 3
MaxHeartRate -> 91
ExerciseAngina -> 2
FamilyHistory -> 6
Target -> 2
```

Taking columns that have a maximum of 4 unique values (And based on column values mentioned in assignment document) as categorical and the rest as numeric:

In [5]:
```
category_list = ["Sex", "CP_Type", "BloodSugar", "ECG", "ExerciseAngina"]
numeric_list = ["Age", "BloodPressure", "Cholestrol", "MaxHeartRate", "FamilyHistory'
```

Pair Plotting numeric features

In [6]:
```
df_number = df.loc[:, numeric_list]
df_number["Target"] = df["Target"]
sb.pairplot(df_number, hue = "Target", palette="husl")
plt.show()
```

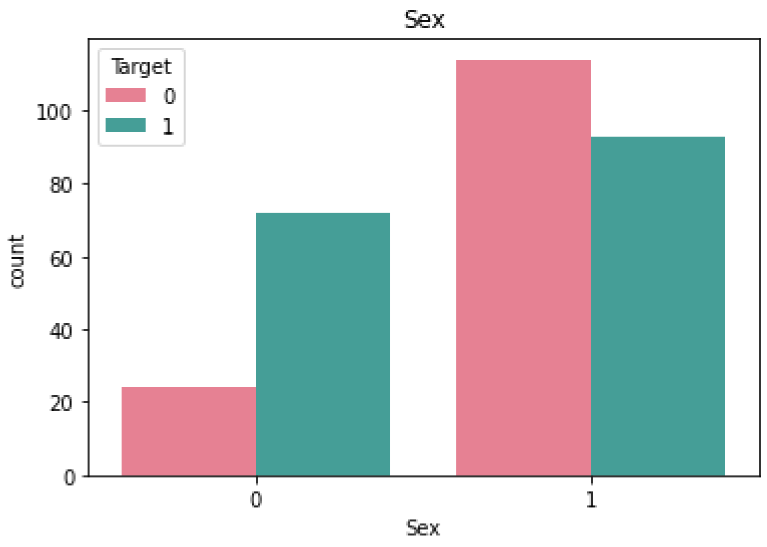


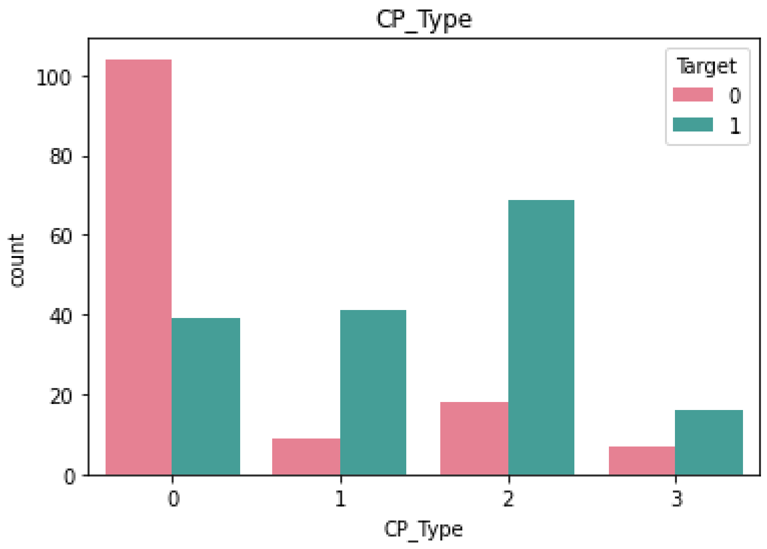
The pair plots do not show any particular trends that can be used to reduce numerical features. We do see a slight relation between age and max heart rate but the plot is scattered enough to not relate them together.

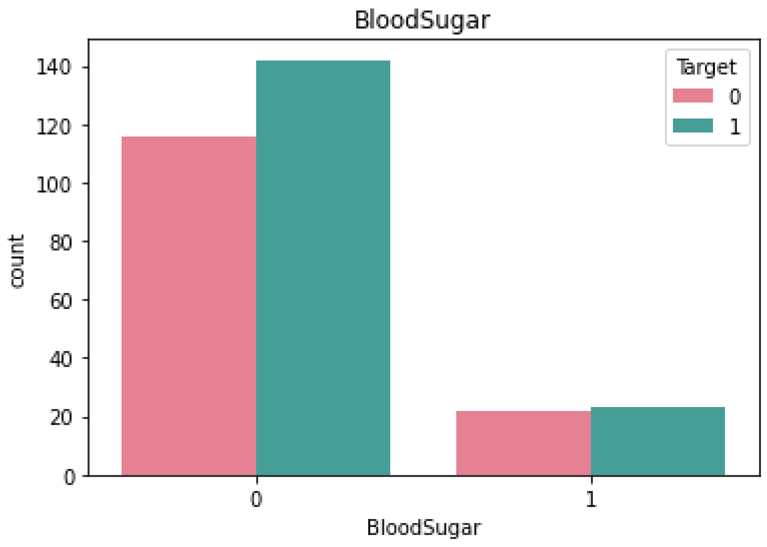
Checking frequncy of each categorical feature wrt target column to check how well it is balanced

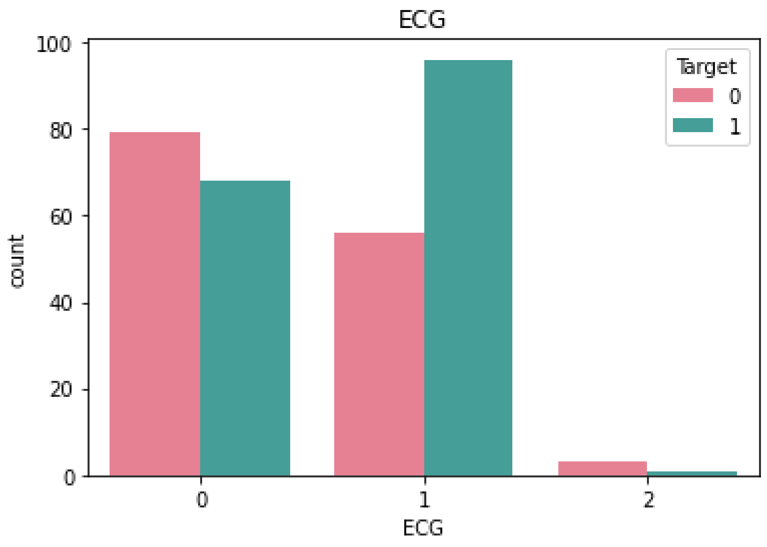
In [7]:

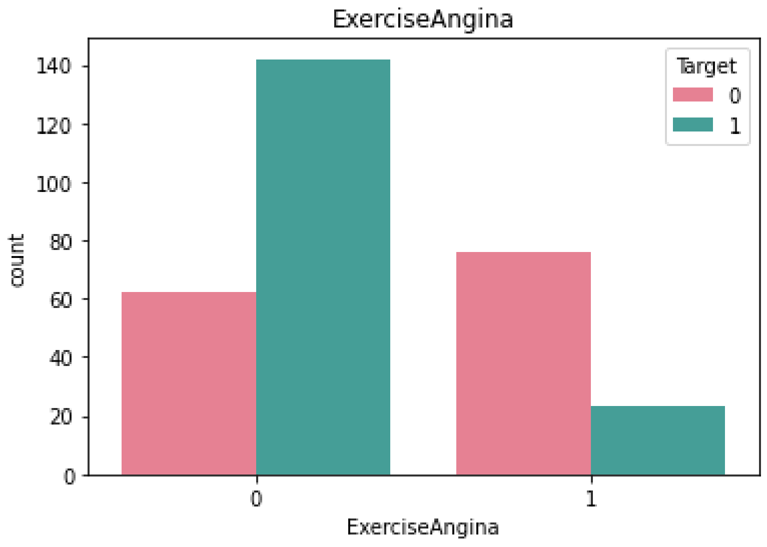
```
df_category = df.loc[:, category_list]
df_category["Target"] = df["Target"]
for i in category_list:
    plt.figure()
    sb.countplot(x = i, data = df_category, hue = "Target", palette="husl")
    plt.title(i)
```


Sex
Target
0
1
count
0
20
40
60
80
100
0
1
Sex


CP_Type
Target
0
1
count
0
20
40
60
80
100
0
1
2
3
CP_Type


BloodSugar
Target
0
1
count
0
20
40
60
80
100
120
140
0
1
BloodSugar





We see that there is less data for some categorical values (Like for ECG=2 and BloodSugar=1).

Scaling the numeric attributes in the dataframe with a standard scaler

In [8]:
```
scaler = StandardScaler()
df[numeric_list] = scaler.fit_transform(df[numeric_list])
df[numeric_list].head()
```

Out[8]:

	Age	BloodPressure	Cholestrol	MaxHeartRate	FamilyHistory
0	0.952197	0.763956	-0.256334	0.015443	0.726365
1	-1.915313	-0.092738	0.072199	1.633471	-0.186866
2	-1.474158	-0.092738	-0.816773	0.977514	-1.100096
3	0.180175	-0.663867	-0.198357	1.239897	-0.186866
4	0.290464	-0.663867	2.082050	0.583939	-1.100096

Model 1

Training a Guassian Naive Bayes model with the data (With scaling done to numerical features)

In [9]:
```
df1 = df.copy()
X = df1.drop(["Target"], axis = 1)
y = df1[["Target"]]
```

Split X and y to training and test data

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_st
print("X_train: {}".format(X_train.shape))
print("y_train: {}".format(y_train.shape))
print("X_test: {}".format(X_test.shape))
print("y_test: {}".format(y_test.shape))
```

```
X_train: (257, 10)
y_train: (257, 1)
X_test: (46, 10)
y_test: (46, 1)
```

Prediction Analysis:

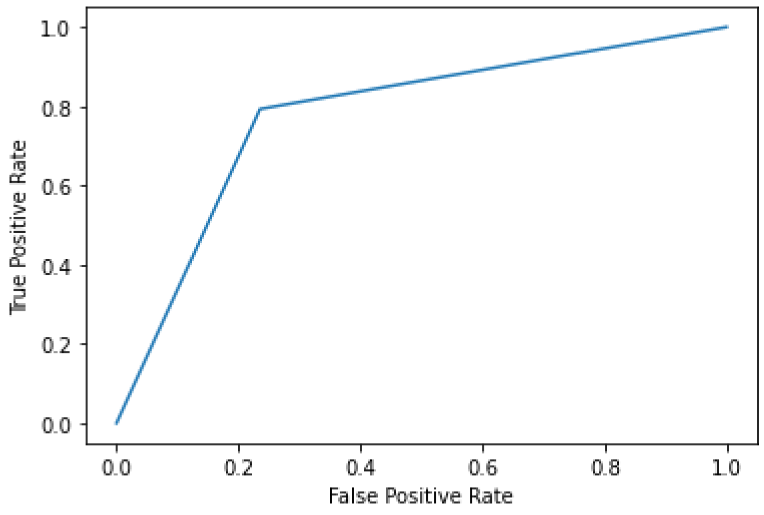
The following is the analysis for a naive bayes model that was trained with the following done on the data:

1. Scale the numerical features with a standard scaler
2. Split the data into training and test data with a 15% test data

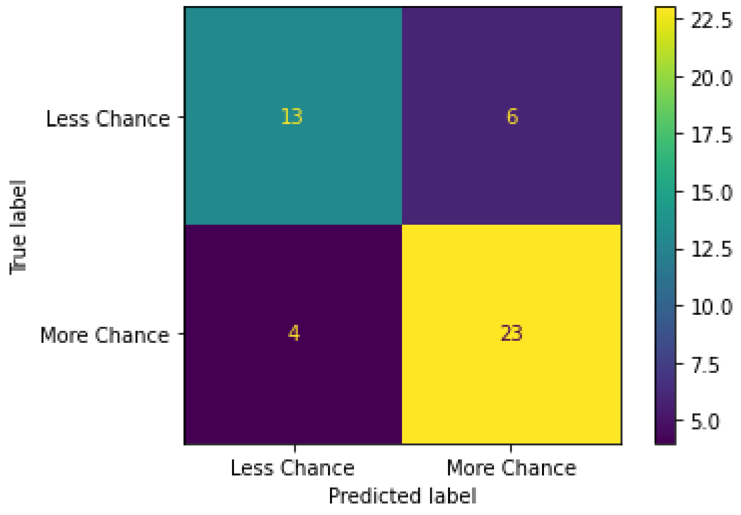
In [11]:

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print("Model 1 Results:")
print("----------------------------------\n")
print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, y_pred) * 100))
print("Precision Score: {0:0.2f}%".format(precision_score(y_test,y_pred) * 100))
print("Recall Score: {0:0.2f}%".format(recall_score(y_test,y_pred) * 100))
print("F Score: {0:0.2f}%".format(f1_score(y_test,y_pred) * 100))
print("Accuracy Score: {0:0.2f}%".format(accuracy_score(y_test,y_pred) * 100))

fpr, tpr, temp = roc_curve(y_test, y_pred)

print("\n----------------------------------\n")
print("ROC Curve:")
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

print("\n----------------------------------\n")
print("Confusion matrix:")
cm = confusion_matrix(y_pred,y_test)
ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
plt.show()
```

```
Model 1 Results:
----------------------------------

AUC-ROC Score: 77.89%
Precision Score: 85.19%
Recall Score: 79.31%
F Score: 82.14%
Accuracy Score: 78.26%

----------------------------------

ROC Curve:
```



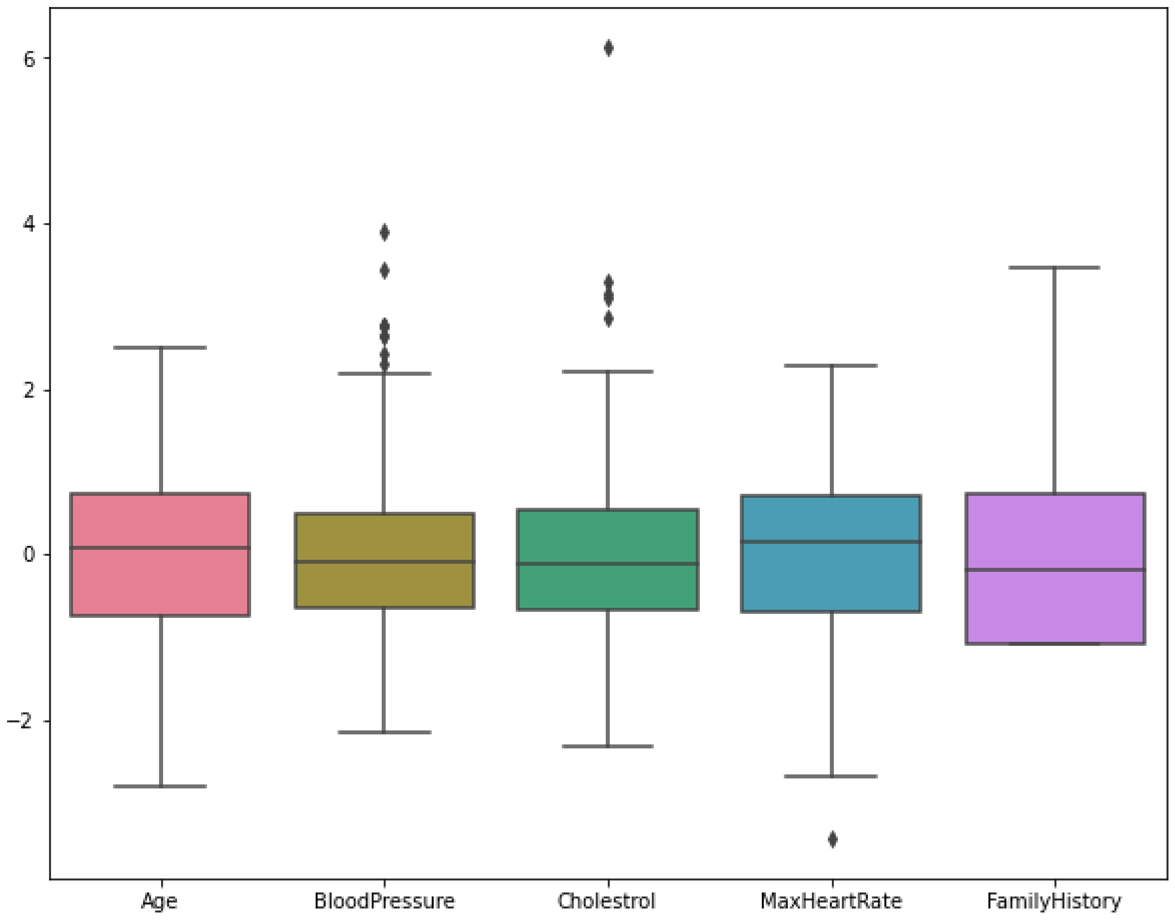
```
---------------------------------

Confusion matrix:
```



Investigating dataset for outliers

Analyzing the boxplot for scaled numeric attributes to check for outliers

In [12]:
```
plt.figure(figsize=(10,8))
sb.boxplot(data=df[numeric_list], palette="husl")
plt.show()
print("\n----------------------------------\n")
```



```
----------------------------------
```

We can see some outlier values for blood pressure, cholestrol and max heart rate. we can drop these outliers using the IQR method.

Dropping outliers with IQR method

Going with $+/-1.6 \times IQR$ to accomodate data upto 3σ from the mean to remove the outliers.

In [13]:

```
print("Original shape of dataframe: {}".format(df.shape))
for i in numeric_list:
    Q25 = np.percentile(df.loc[:, i],25)
    Q75 = np.percentile(df.loc[:, i],75)
    IQR = Q75 - Q25
    upper_bound = np.where(df.loc[:, i] >= (Q75 + 1.6*IQR))
    lower_bound = np.where(df.loc[:, i] <= (Q25 - 1.6*IQR))
    df.drop(upper_bound[0], inplace = True)
    df.drop(lower_bound[0], inplace = True)
print("Shape of dataframe after dropping outliers: {}".format(df.shape))
```

```
Original shape of dataframe: (303, 11)
Shape of dataframe after dropping outliers: (288, 11)
```

Model 2

Training a Guassian Naive Bayes model with the data (With scaling done to numerical features and removing outliers)

In [14]:

```
df2 = df.copy()
X = df2.drop(["Target"], axis = 1)
y = df2[["Target"]]
```

Split X and y to training and test data

```
In [15]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_st
         print("X_train: {}".format(X_train.shape))
         print("y_train: {}".format(y_train.shape))
         print("X_test: {}".format(X_test.shape))
         print("y_test: {}".format(y_test.shape))
```

```
X_train: (244, 10)
y_train: (244, 1)
X_test: (44, 10)
y_test: (44, 1)
```

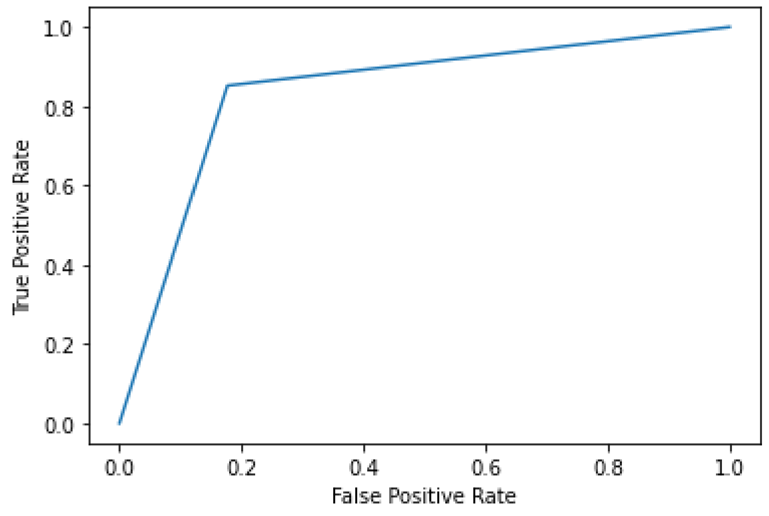
Prediction Analysis:

The following is the analysis for a naive bayes model that was trained with the following done on the data:

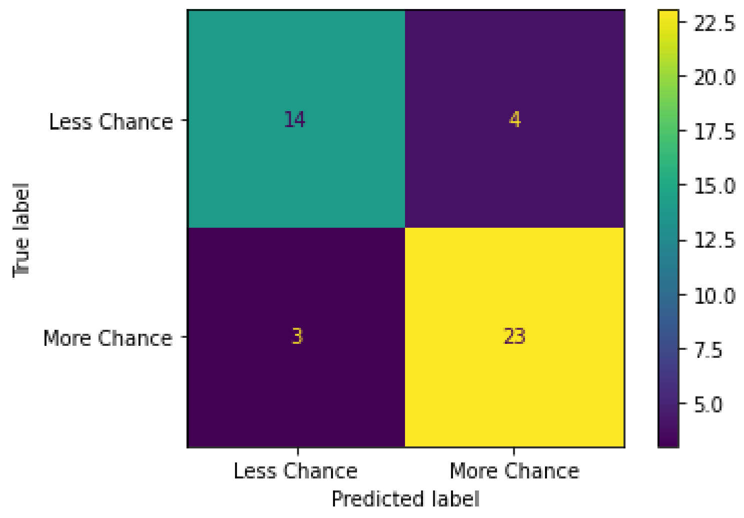
1. Scale the numerical features with a standard scaler
2. Remove outliers 1.6 times IQR below and above the Q1 and Q3 respectively
3. Split the data into training and test data with a 15% test data

```
In [16]: gnb = GaussianNB()
         gnb.fit(X_train, y_train)
         y_pred = gnb.predict(X_test)
         print("Model 2 Results:")
         print("----------------------------------\n")
         print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, y_pred) * 100))
         print("Precision Score: {0:0.2f}%".format(precision_score(y_test,y_pred) * 100))
         print("Recall Score: {0:0.2f}%".format(recall_score(y_test,y_pred) * 100))
         print("F Score: {0:0.2f}%".format(f1_score(y_test,y_pred) * 100))
         print("Accuracy Score: {0:0.2f}%".format(accuracy_score(y_test,y_pred) * 100))

         fpr, tpr, temp = roc_curve(y_test, y_pred)

         print("\n----------------------------------\n")
         print("ROC Curve:")
         plt.plot(fpr, tpr)
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.show()

         print("\n----------------------------------\n")
         print("Confusion matrix:")
         cm = confusion_matrix(y_pred,y_test)
         ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
         plt.show()
```

```
Model 2 Results:
----------------------------------

AUC-ROC Score: 83.77%
Precision Score: 88.46%
Recall Score: 85.19%
F Score: 86.79%
Accuracy Score: 84.09%

----------------------------------

ROC Curve:
```



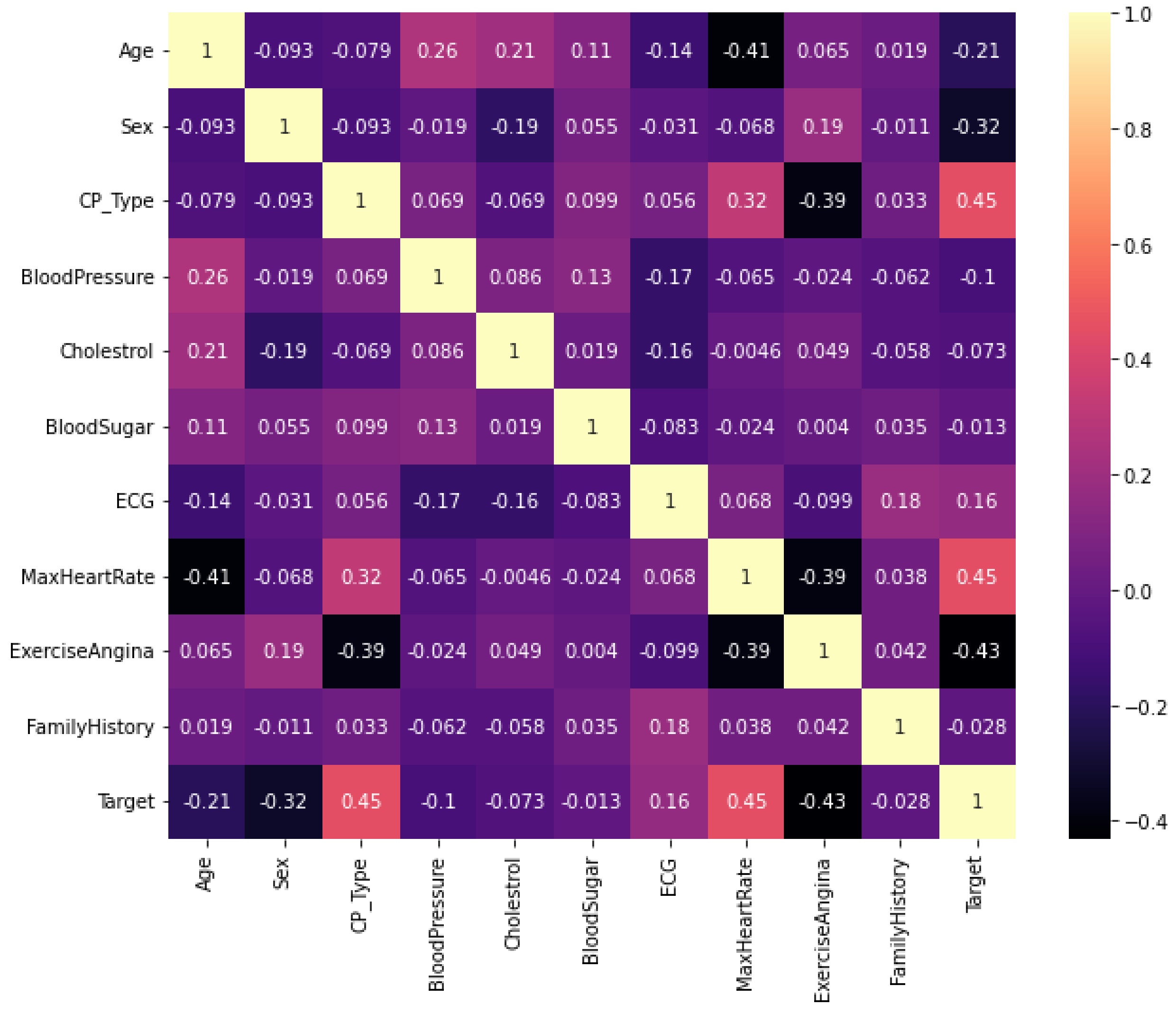
```
--------------------------------------------

Confusion matrix:
```



Finding correlation between features through a heatmap

In [17]:

```python
corr_features = set()
corr_matrix = df.corr()
plt.figure(figsize = (10,8))
sb.heatmap(corr_matrix, annot = True, cmap="magma")
plt.show()

for i in range(len(corr_matrix .columns)):
    for j in range(i):
        if abs(corr_matrix.iloc[i, j]) > 0.5:
            colname = corr_matrix.columns[i]
            corr_features.add(colname)
print("\n-----------------------------------\n")
print("The number of correlating features: {}".format(len(corr_features)))
print("The correlating features are: {}".format(corr_features))
print("\n-----------------------------------\n")
```



```
----------------------------------

The number of correlating features: 0
The correlating features are: set()

----------------------------------
```

None of the features seem to correlate with each other and hence we cannot do feature reduction

Model 3

Training a Guassian Naive Bayes model with the data (With scaling done to numerical features, outliers removed and one hot encoding categorical features)

Original dataframe:

In [18]:

```
df3 = df.copy()
df3.head()
```

Out[18]:

	Age	Sex	CP_Type	BloodPressure	Cholestrol	BloodSugar	ECG	MaxHeartRate	ExerciseAngina
0	0.952197	1	3	0.763956	-0.256334	1	0	0.015443	0
1	-1.915313	1	2	-0.092738	0.072199	0	1	1.633471	0
2	-1.474158	0	1	-0.092738	-0.816773	0	0	0.977514	0
3	0.180175	1	1	-0.663867	-0.198357	0	1	1.239897	0
4	0.290464	0	0	-0.663867	2.082050	0	1	0.583939	1

One hot encoded dataframe:

```
In [19]:
df3 = pd.get_dummies(df2, columns = category_list, drop_first = True)
df3.head()
```

Out[19]:

	Age	BloodPressure	Cholestrol	MaxHeartRate	FamilyHistory	Target	Sex_1	CP_Type_1	CP_Type_
0	0.952197	0.763956	-0.256334	0.015443	0.726365	1	1	0	
1	-1.915313	-0.092738	0.072199	1.633471	-0.186866	1	1	0	
2	-1.474158	-0.092738	-0.816773	0.977514	-1.100096	1	0	1	
3	0.180175	-0.663867	-0.198357	1.239897	-0.186866	1	1	1	
4	0.290464	-0.663867	2.082050	0.583939	-1.100096	1	0	0	

```
In [20]:
X = df3.drop(["Target"], axis = 1)
y = df3[["Target"]]
```

Split X and y to training and test data

```
In [21]:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_st
print("X_train: {}".format(X_train.shape))
print("y_train: {}".format(y_train.shape))
print("X_test: {}".format(X_test.shape))
print("y_test: {}".format(y_test.shape))
```

```
X_train: (244, 13)
y_train: (244, 1)
X_test: (44, 13)
y_test: (44, 1)
```

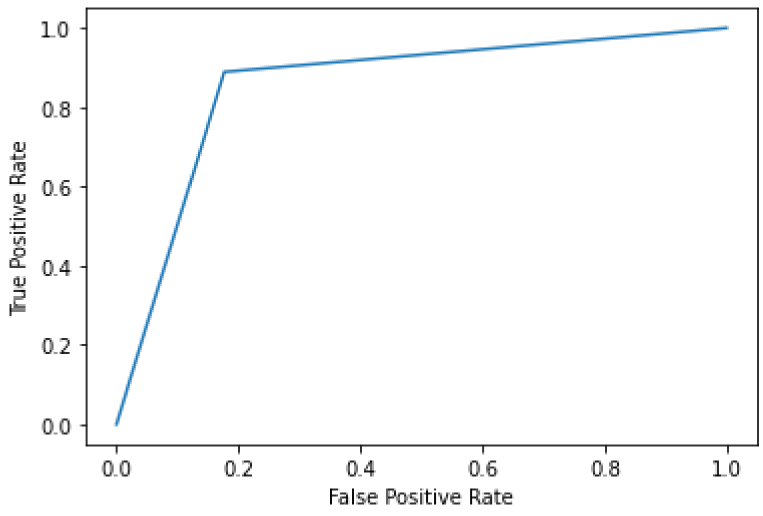
Prediction Analysis:

The following is the analysis for a naive bayes model that was trained with the following done on the data:

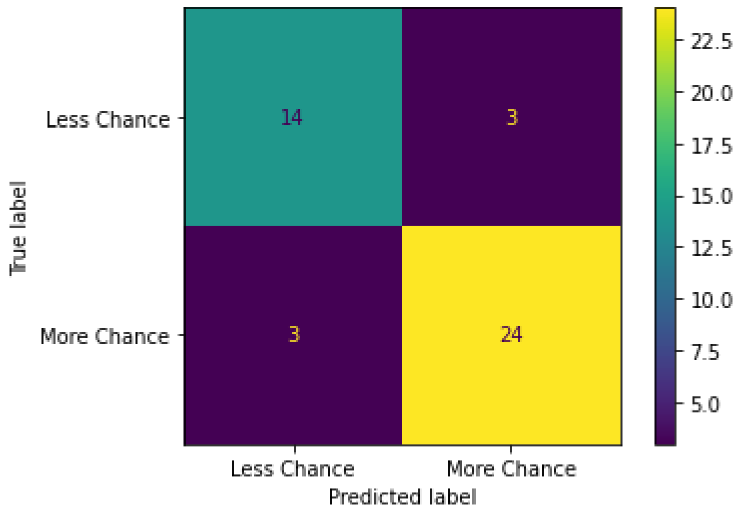
1. One hot encode categorical features
2. Scale the numerical features with a standard scaler
3. Remove outliers 1.6 times IQR below and above the Q1 and Q3 respectively
4. Split the data into training and test data with 15% test data

```
In [22]:
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

print("Model 3 Results:")
print("-----------------------------------\n")
print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, y_pred) * 100))
print("Precision Score: {0:0.2f}%".format(precision_score(y_test,y_pred) * 100))
print("Recall Score: {0:0.2f}%".format(recall_score(y_test,y_pred) * 100))
print("F Score: {0:0.2f}%".format(f1_score(y_test,y_pred) * 100))
print("Accuracy Score: {0:0.2f}%".format(accuracy_score(y_test,y_pred) * 100))

fpr, tpr, temp = roc_curve(y_test, y_pred)

print("\n-----------------------------------\n")
print("ROC Curve:")
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

print("\n-----------------------------------\n")
print("Confusion matrix:")
cm = confusion_matrix(y_pred,y_test)
ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
plt.show()
```

```
Model 3 Results:
----------------------------------

AUC-ROC Score: 85.62%
Precision Score: 88.89%
Recall Score: 88.89%
F Score: 88.89%
Accuracy Score: 86.36%

----------------------------------

ROC Curve:
```



```
----------------------------------

Confusion matrix:
```



Comparing the results of the three models:

From the results obtained from above prediction analysis, we can tabulate them together below:

Model/Metric	AUC-ROC	Precision	Recall	F_Score	Accuracy
Model 1	77.89%	85.19%	79.31%	82.14%	78.26%
Model 2	83.77%	88.46%	85.19%	86.79%	84.09%
Model 3	85.62%	88.89%	88.89%	88.89%	86.36%

This shows that the **maximum Accuracy is obtained when scaling is applied on numerical attributes with outliers being dropped and one hot encoding is done to the categorical features**. (If categorical features are untouched, it still does better than a model where outliers are not removed.)

We know that:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

and

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Upon viewing the confusion matrix for each model it is clear that **lesser false negatives and false positives are recorded in Model 3** in comparison to Model 1 and Model 2 which attributes to the **better Precision and Recall scores in Model 3**.

Finally, we know that:

$$F = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

This goes to show why Model 3 has better F score since its recall and precision is better than the other two models.

This notebook along with it's PDF version can be found in this Github Repo