Applied ML Assignment 2

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 the following approaches:
  - Naïve Bayesian Approach
  - Logistic Regression
  - Decision Tree
  - Ensemble Methods (any one of your choice)
  - K-Nearest Neighbour
- Compare the performances of each model/classifier considering the given dataset using different evaluation measures such as Precision, Recall, F1-Score, AUC-ROC. Show the comparison chart in Python notebook
- Identify the model which you think is the best amongst all the models you have built. Also, explain why you think this is the best model. Answer this question in the notebook itself.

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 models

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

Submission Plan

The following will be done in this notebook:

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. Check which numerical attributes are relevant and which are irrelevant and drop irrelevant ones (Using ANOVA).
6. Scale numerical attributes with a standard scaler.

7. Check for outliers via boxplots and remove them with IQR method.
8. Train a gnb model.
9. Train a logistic regression model.
10. Train a decision tree model.
11. Train a ensemble learning model (With random forest classifier).
12. Train a knn model.

We will then **compare the accuracy, Precision, Recall, F-Score and AOC-ROC of all the models trained** and choose the best one based on these metrics.

My Submission

Importing necessary packages

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sb
import warnings

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusion_matrix, ConfusionMatrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier

warnings.filterwarnings('ignore')
```

Viewing Data

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()
dfcp = df.copy()
```

```
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:
```

In [3]:

```
print("Stats on Dataframe:")
df.describe()
```

```
Stats on Dataframe:
```

Out[3]:

	Age	Sex	CP_Type	BloodPressure	Cholestrol	BloodSugar	ECG	MaxHeartRate	ExerciseAngina	FamilyHistory	Ta
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.204620	0.54
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.096825	0.49
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	0.00
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	1.000000	1.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	2.000000	1.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	5.000000	1.00

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.

Data Preprocessing

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

The following are the assumed ranges for these columns:

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

In [4]:
```python
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 [5]:
```python
for column in list(df.columns):
    print("{} -> {}".format(column, df[column].value_counts().shape[0]))
```

```
Age -> 41
Sex -> 2
CP_Type -> 4
BloodPressure -> 49
```

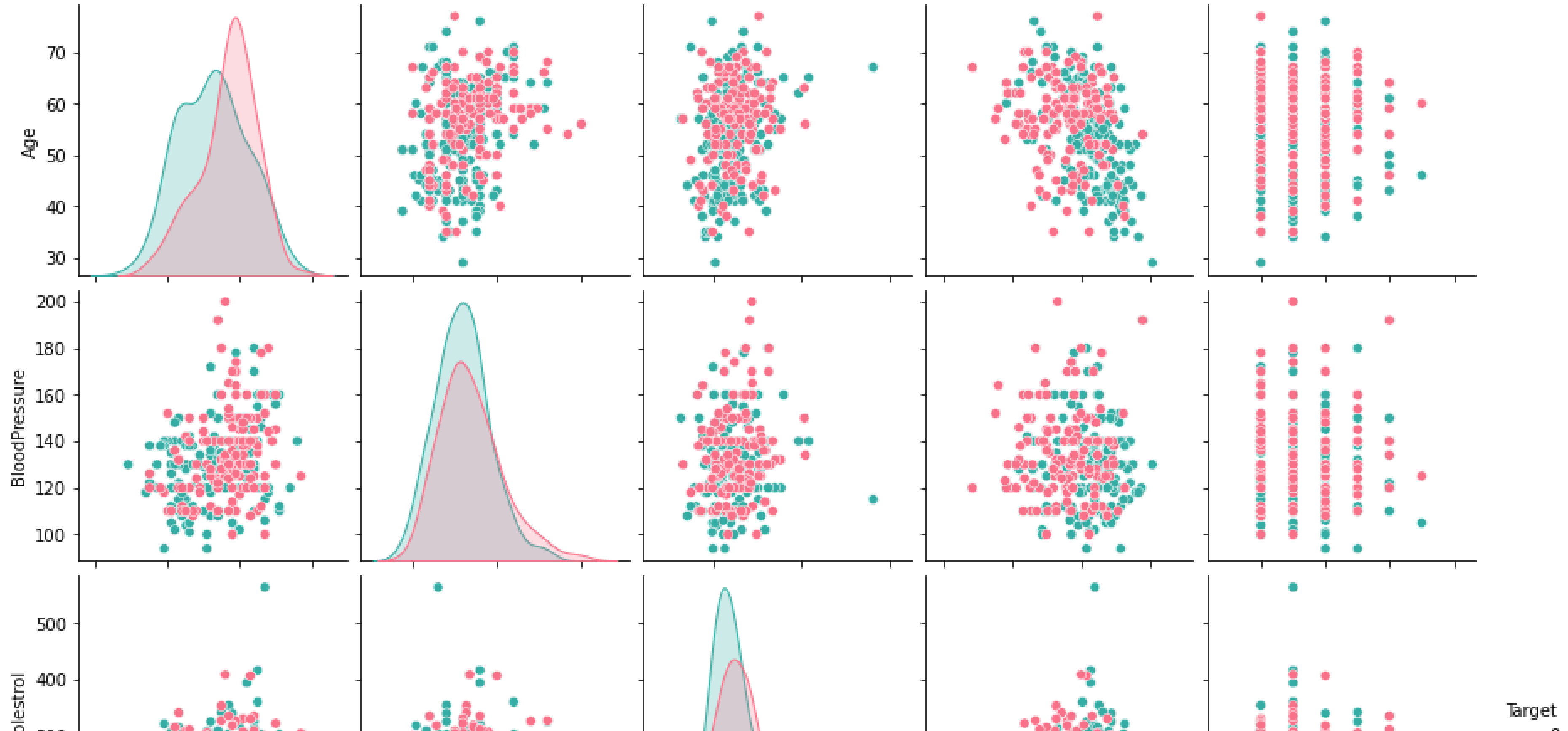
```
Cholestrol -> 152
BloodSugar -> 2
ECG -> 3
MaxHeartRate -> 91
ExerciseAngina -> 2
FamilyHistory -> 6
Target -> 2
```

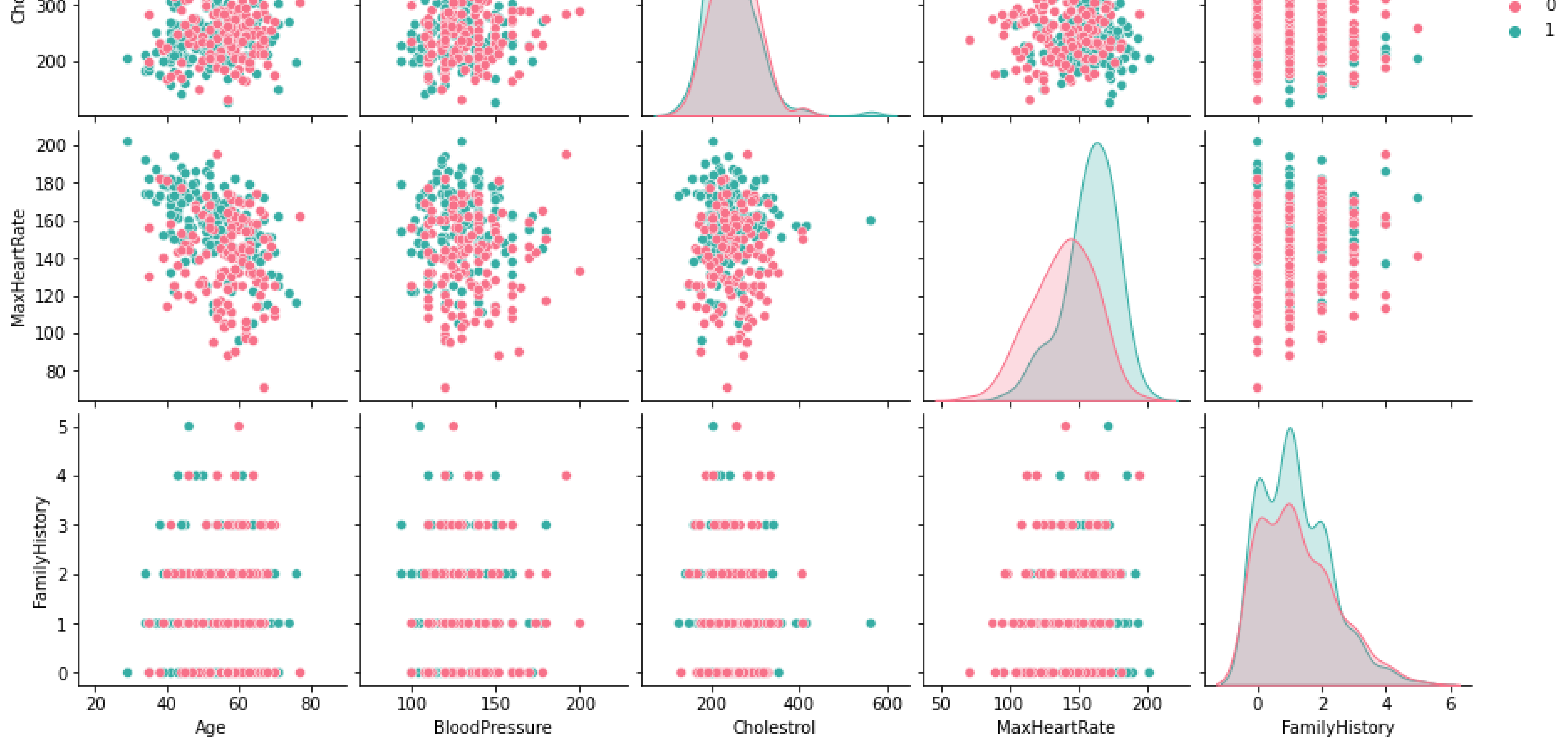
Taking columns that have a maximum of 4 unique values (And based on information given in assignment document) as categorical and the rest as numeric:

In [6]:
```
category_list = ["Sex", "CP_Type", "BloodSugar", "ECG", "ExerciseAngina"]
numeric_list = ["Age", "BloodPressure", "Cholestrol", "MaxHeartRate", "FamilyHistory"]
```

Checking for trends in numeric features through Pair Plots

In [7]:
```
df_number = df.loc[:, numeric_list]
df_number["Target"] = df["Target"]
sb.pairplot(df_number, hue = "Target", palette="husl")
plt.show()
```

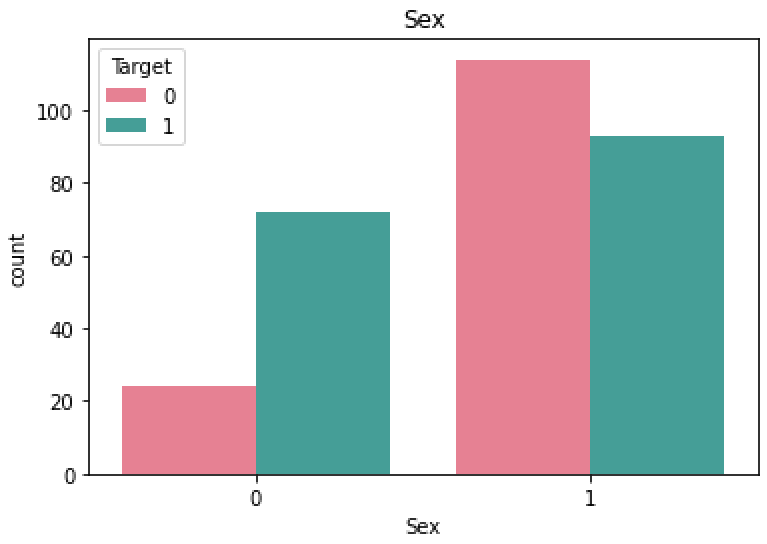


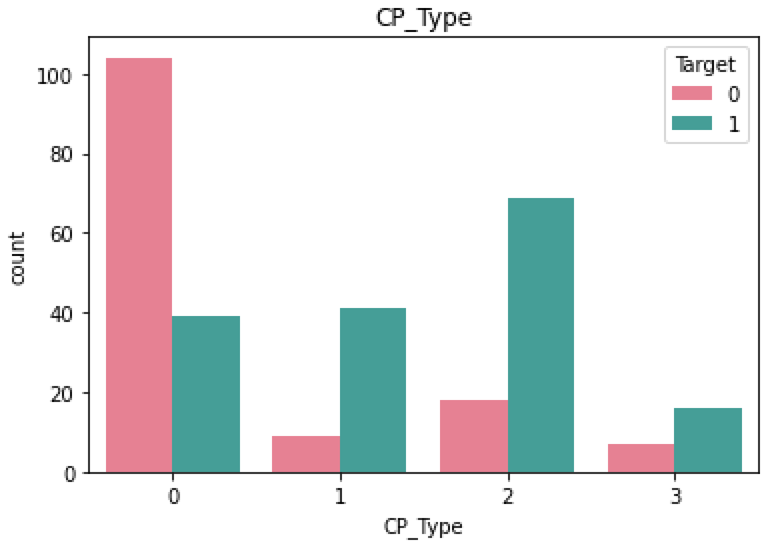


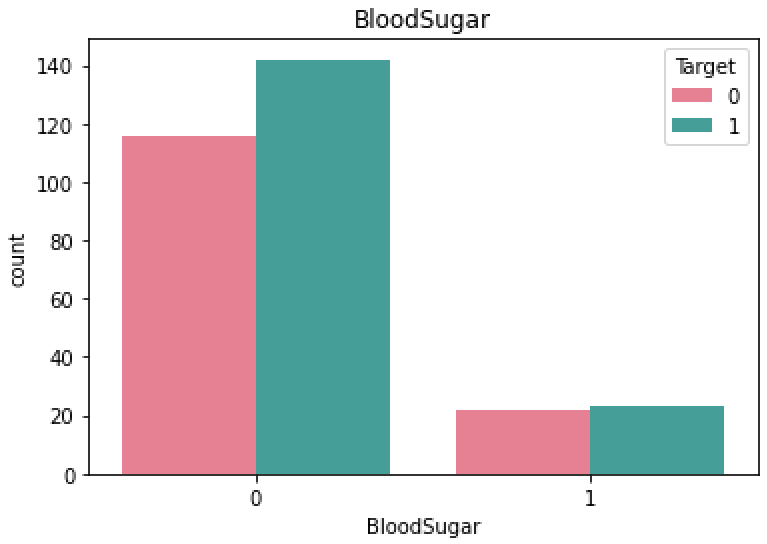
The pair plots do not show any particular trends that can be used to reduce numerical features. We do see a slight relation between age and max heart rate but the plot is scattered enough to not relate them together.

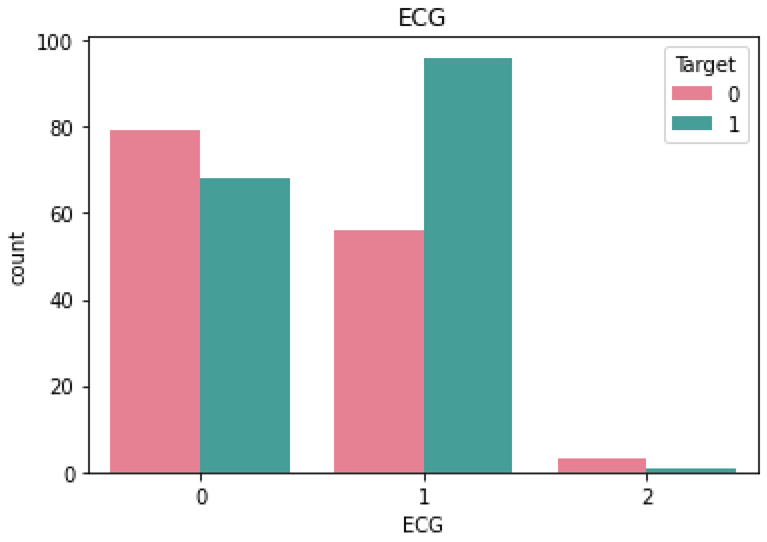
Checking frequncy of each categorical feature wrt target column to check how well it is balanced

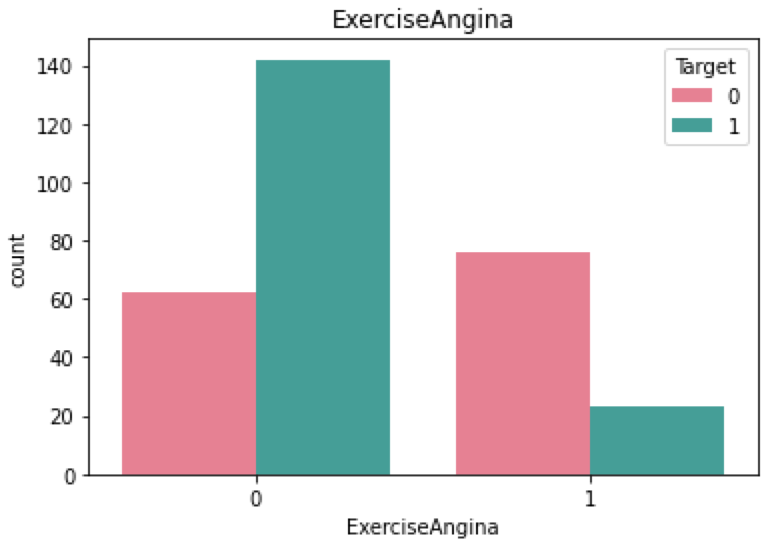
In [8]:
```
df_category = df.loc[:, category_list]
df_category["Target"] = df["Target"]
for i in category_list:
    plt.figure()
    sb.countplot(x = i, data = df_category, hue = "Target", palette="husl")
    plt.title(i)
```


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
140
120
100
80
60
40
20
0
0
1
BloodSugar


ECG
Target
0
1
count
100
80
60
40
20
0
0
1
2
ECG



Here we see that there is very little rows that have ECG value = 2 and similarly very little rows for BloodSugar value = 1.

Checking relevant numerical features

To do this we shall use the `f_oneway` function from `scipy.stats`. This function performs one-way ANOVA(Analysis of Variance) to test the null hypothesis that two groups of data have the same population mean.

A feature is only relevant if the sample from a particular feature for a target category is statistically very different from another sample from the same feature for another target category.

Checking relevance of Age

In [9]:
```
result = stats.f_oneway(df["Age"][df["Target"] == 0],
                        df["Age"][df["Target"] == 1])
result.pvalue
```

Out[9]:
```
7.5248013034422 68e-05
```

The pvalue is < 0.05. This shows that the means of the two distributions (One with Age wrt less chance of getting heart attack and the other with more chance of getting heart attack) are significantly different statistically, hence **Age is relevant**

Checking relevance of BloodPressure

In [10]:
```
result = stats.f_oneway(df["BloodPressure"][df["Target"] == 0],
                        df["BloodPressure"][df["Target"] == 1])
result.pvalue
```

Out[10]:
```
0.0115460592002333376
```

The pvalue is < 0.05. This shows that the means of the two distributions (One with BloodPressure wrt less chance of getting heart attack and the other with more chance of getting heart attack) are significantly different statistically, hence **BloodPressure is relevant**

Checking relevance of Cholestrol

In [11]:
```
result = stats.f_oneway(df["Cholestrol"][df["Target"] == 0],
               df["Cholestrol"][df["Target"] == 1])
result.pvalue
```

Out[11]:
```
0.1387903269560108
```

The pvalue is > 0.05. This shows that the means of the two distributions (One with Cholestrol wrt less chance of getting heart attack and the other with more chance of getting heart attack) are not significantly different statistically, hence **Cholestrol is irrelevant**

Checking relevance of MaxHeartRate

In [12]:
```
result = stats.f_oneway(df["MaxHeartRate"][df["Target"] == 0],
               df["MaxHeartRate"][df["Target"] == 1])
result.pvalue
```

Out[12]:
```
1.6973376386560805e-14
```

The pvalue is < 0.05. This shows that the means of the two distributions (One with MaxHeartRate wrt less chance of getting heart attack and the other with more chance of getting heart attack) are significantly different statistically, hence **MaxHeartRate is relevant**

Checking relevance of FamilyHistory

In [13]:
```
result = stats.f_oneway(df["FamilyHistory"][df["Target"] == 0],
               df["FamilyHistory"][df["Target"] == 1])
result.pvalue
```

Out[13]:
```
0.6172651404419242
```

The pvalue is > 0.05. This shows that the means of the two distributions (One with FamilyHistory wrt less chance of getting heart attack and the other with more chance of getting heart attack) are not significantly different statistically, hence **FamilyHistory is irrelevant**

Dropping the irrelevant features

In [14]:
```
df.drop(["Cholestrol"], axis = 1, inplace= True)
df.drop(["FamilyHistory"], axis = 1, inplace= True)
numeric_list.remove("Cholestrol")
numeric_list.remove("FamilyHistory")
df.head()
```

Out[14]:

	Age	Sex	CP_Type	BloodPressure	BloodSugar	ECG	MaxHeartRate	ExerciseAngina	Target
0	63	1	3	145	1	0	150	0	1
1	37	1	2	130	0	1	187	0	1
2	41	0	1	130	0	0	172	0	1
3	56	1	1	120	0	1	178	0	1
4	57	0	0	120	0	1	163	1	1

Scaling the numeric attributes in the dataframe with a standard scaler

In [15]:

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

Out[15]:

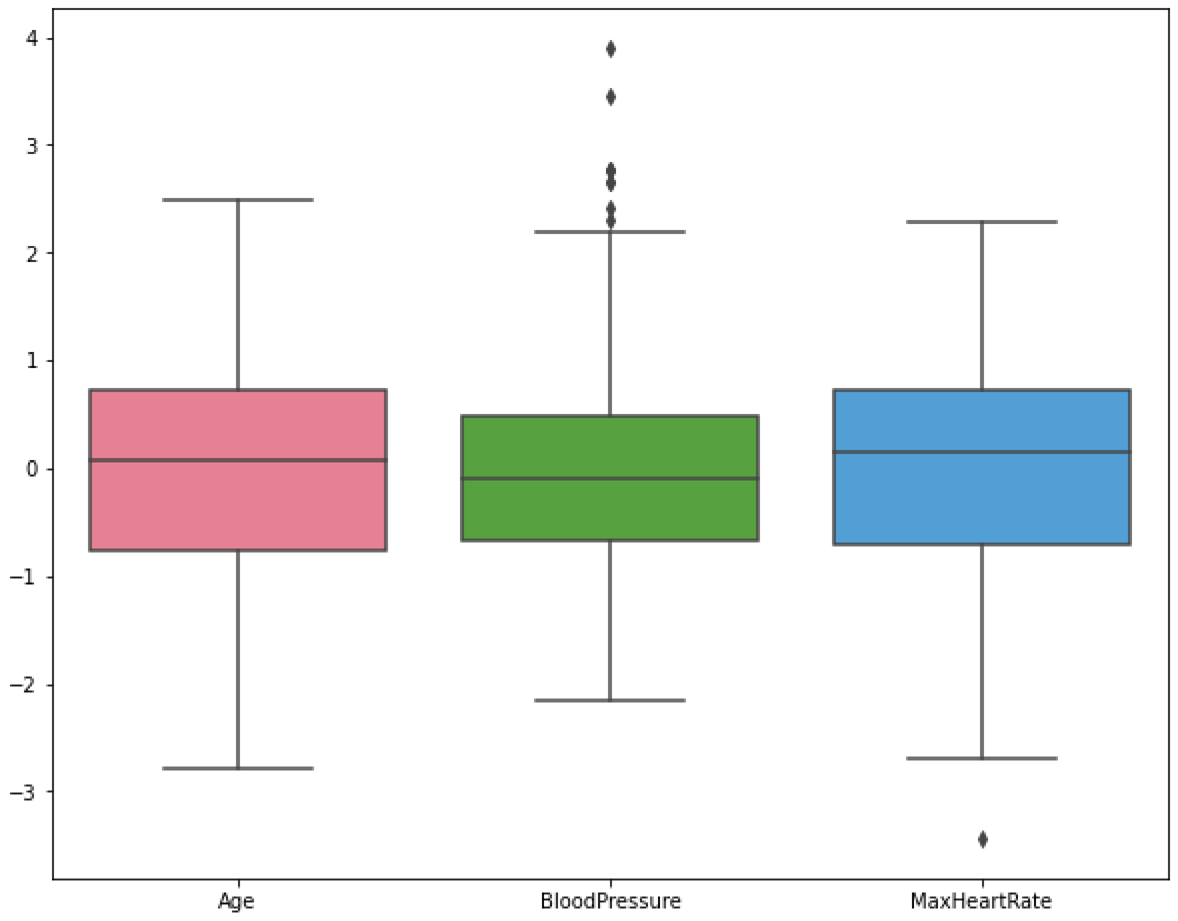
	Age	Sex	CP_Type	BloodPressure	BloodSugar	ECG	MaxHeartRate	ExerciseAngina	Target
0	0.952197	1	3	0.763956	1	0	0.015443	0	1
1	-1.915313	1	2	-0.092738	0	1	1.633471	0	1
2	-1.474158	0	1	-0.092738	0	0	0.977514	0	1
3	0.180175	1	1	-0.663867	0	1	1.239897	0	1
4	0.290464	0	0	-0.663867	0	1	0.583939	1	1

Investigating dataset for outliers

Analyzing the boxplot for scaled numeric attributes to check for outliers

In [16]:

```
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 and max heart rate. we can drop these outliers using the IQR method.

Dropping outliers with IQR method

In [17]:

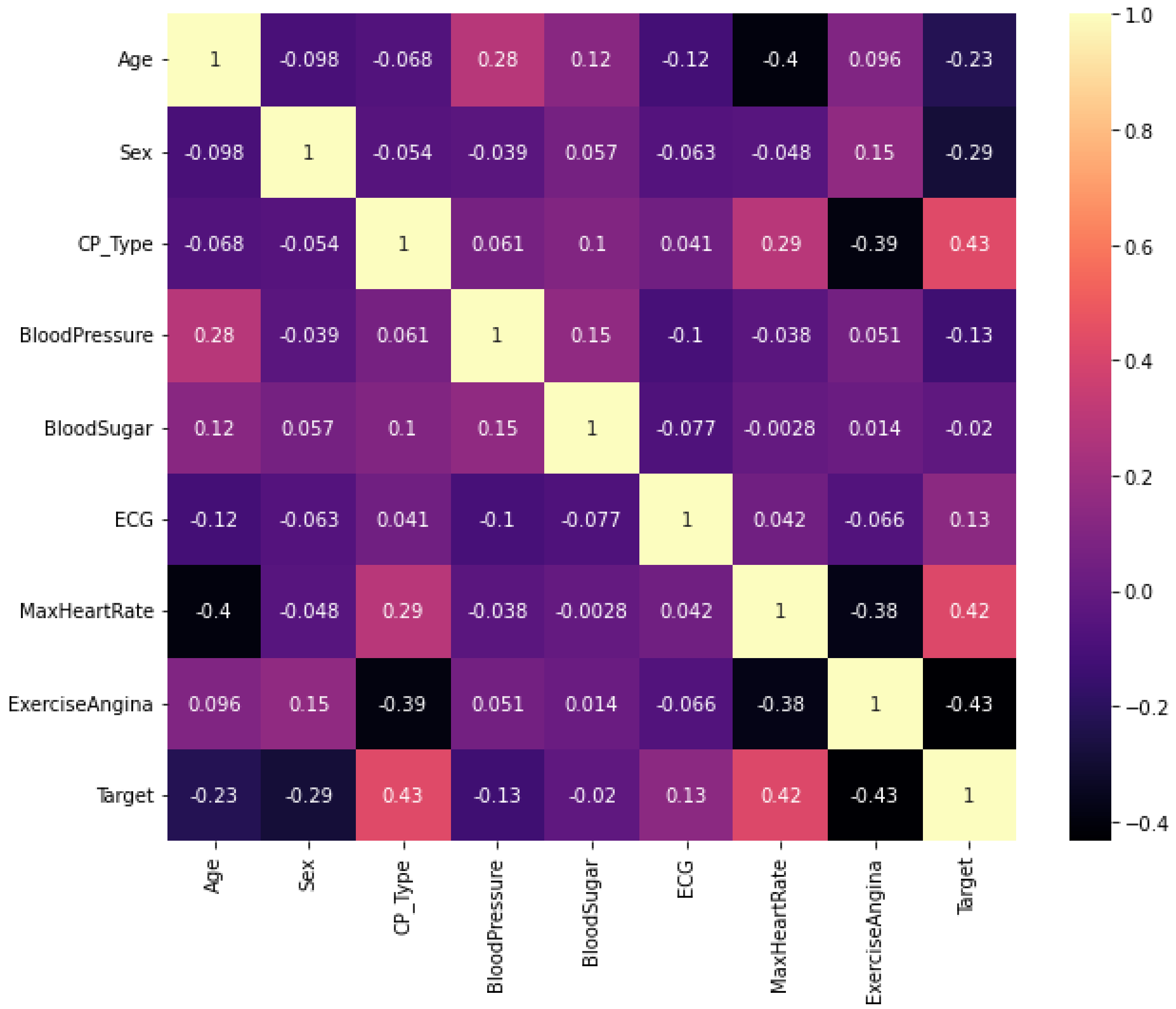
```
print("Original shape of dataframe: {}".format(df.shape))
for i in numeric_list:
    Q25 = np.percentile(df.loc[:, i],25)
    Q75 = np.percentile(df.loc[:, i],75)
    IQR = Q75 - Q25
    upper_bound = np.where(df.loc[:, i] >= (Q75 + 3*IQR))
    lower_bound = np.where(df.loc[:, i] <= (Q25 - 3*IQR))
    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, 9)
Shape of dataframe after dropping outliers: (302, 9)
```

Finding correlation between features through a heatmap

In [18]:
```
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")
```



	Age	Sex	CP_Type	BloodPressure	BloodSugar	ECG	MaxHeartRate	ExerciseAngina	Target
Age	1	-0.098	-0.068	0.28	0.12	-0.12	-0.4	0.096	-0.23
Sex	-0.098	1	-0.054	-0.039	0.057	-0.063	-0.048	0.15	-0.29
CP_Type	-0.068	-0.054	1	0.061	0.1	0.041	0.29	-0.39	0.43
BloodPressure	0.28	-0.039	0.061	1	0.15	-0.1	-0.038	0.051	-0.13
BloodSugar	0.12	0.057	0.1	0.15	1	-0.077	-0.0028	0.014	-0.02
ECG	-0.12	-0.063	0.041	-0.1	-0.077	1	0.042	-0.066	0.13
MaxHeartRate	-0.4	-0.048	0.29	-0.038	-0.0028	0.042	1	-0.38	0.42
ExerciseAngina	0.096	0.15	-0.39	0.051	0.014	-0.066	-0.38	1	-0.43
Target	-0.23	-0.29	0.43	-0.13	-0.02	0.13	0.42	-0.43	1

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

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

Splitting data set into X and y

In [19]:
```
X = df.drop(["Target"], axis = 1)
y = df[["Target"]]
```

Guassian Naive Bayes Model

Split X and y to training and test data

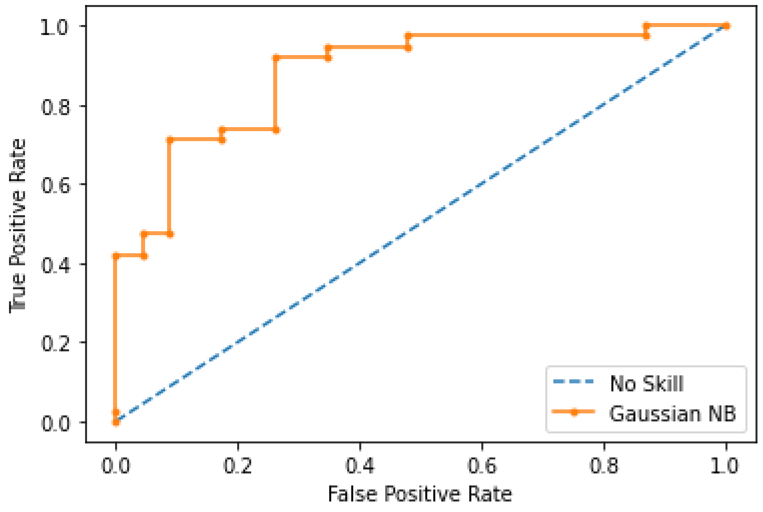
In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 3)
print("X_train: {}".format(X_train.shape))
print("y_train: {}".format(y_train.shape))
print("X_test: {}".format(X_test.shape))
print("y_test: {}".format(y_test.shape))
```

```
X_train: (241, 8)
y_train: (241, 1)
X_test: (61, 8)
y_test: (61, 1)
```

In [21]:
```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print("Guassian NB Results:")
print("----------------------------------\n")
ns_probs = [0 for _ in range(len(y_test))]
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

y_probs = gnb.predict_proba(X_test)
gnb_probs = y_probs[:, 1]
gnb_fpr, gnb_tpr, temp = roc_curve(y_test, gnb_probs)
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(gnb_fpr, gnb_tpr, marker='.', label='Gaussian NB')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
print("----------------------------------\n")
print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, gnb_probs) * 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))
print("\n----------------------------------\n")
print("Confusion matrix:")
cm = confusion_matrix(y_pred,y_test)
ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
plt.show()
```

```
Guassian NB Results:
----------------------------------
```

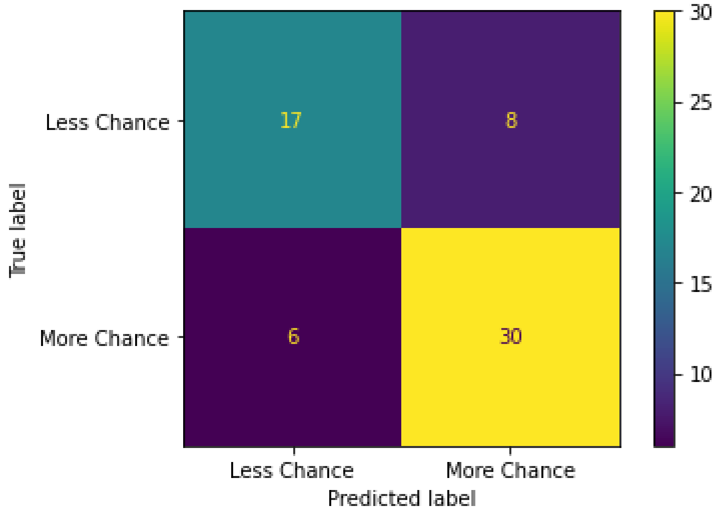


```
----------------------------------

AUC-ROC Score: 87.99%
Precision Score: 83.33%
Recall Score: 78.95%
F Score: 81.08%
Accuracy Score: 77.05%

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

Confusion matrix:
```



Logistic Regression Model

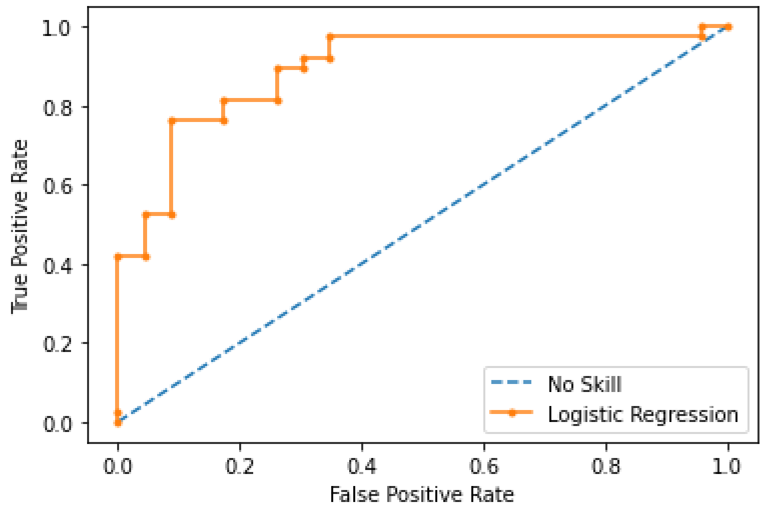
In [22]:

```
logreg = LogisticRegression(solver='newton-cg', penalty='l2', C=1.0)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print("Logistic Regression Results:")
print("----------------------------------\n")
ns_probs = [0 for _ in range(len(y_test))]
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

y_probs = logreg.predict_proba(X_test)
lr_probs = y_probs[:, 1]
lr_fpr, lr_tpr, temp = roc_curve(y_test, lr_probs)
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
print("----------------------------------\n")
print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, lr_probs) * 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))
print("\n----------------------------------\n")
print("Confusion matrix:")
cm = confusion_matrix(y_pred,y_test)
ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
plt.show()
```

```
Logistic Regression Results:
----------------------------------
```

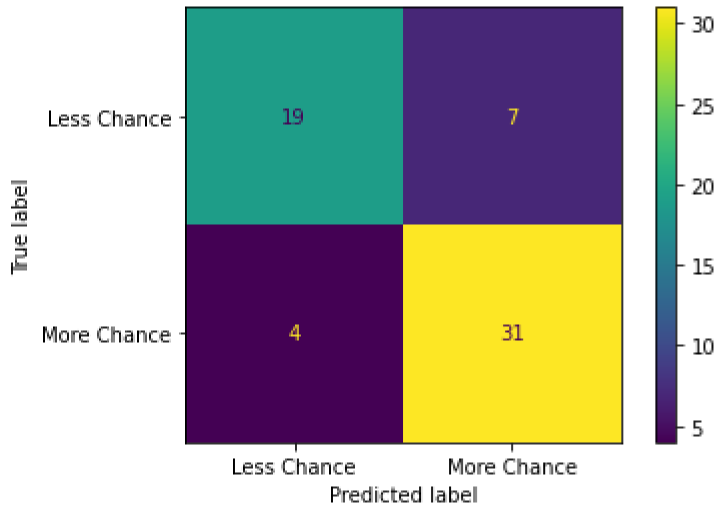


```
----------------------------------

AUC-ROC Score: 89.36%
Precision Score: 88.57%
Recall Score: 81.58%
F Score: 84.93%
Accuracy Score: 81.97%

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

Confusion matrix:
```



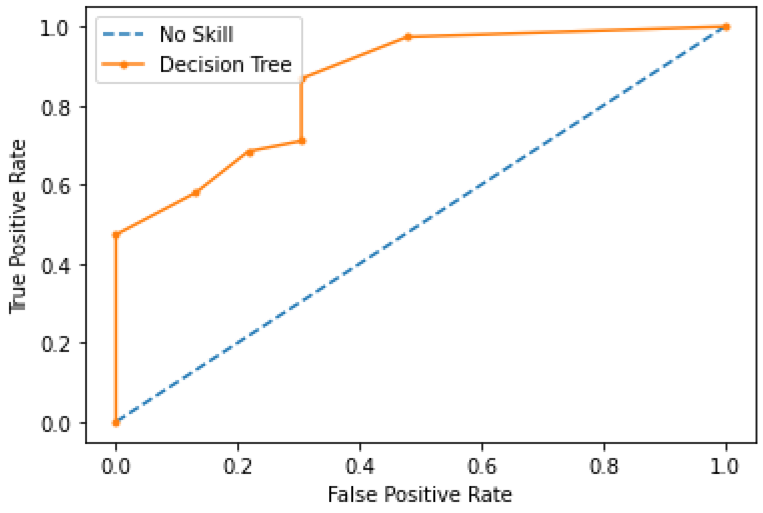
Decision Tree Model

```
In [23]: clf_model = DecisionTreeClassifier(criterion="gini", random_state=4,max_depth=3, min_samples_leaf=10)
         clf_model.fit(X_train,y_train)
         y_pred = clf_model.predict(X_test)

         print("Decision Tree Results:")
         print("----------------------------------\n")
         ns_probs = [0 for _ in range(len(y_test))]
         ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

         y_probs = clf_model.predict_proba(X_test)
         dt_probs = y_probs[:, 1]
         dt_fpr, dt_tpr, temp = roc_curve(y_test, dt_probs)
         plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
         plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree')
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend()
         plt.show()
         print("----------------------------------\n")
         print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, dt_probs) * 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))
         print("\n----------------------------------\n")
         print("Confusion matrix:")
         cm = confusion_matrix(y_pred,y_test)
         ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
         plt.show()
```

```
Decision Tree Results:
----------------------------------
```

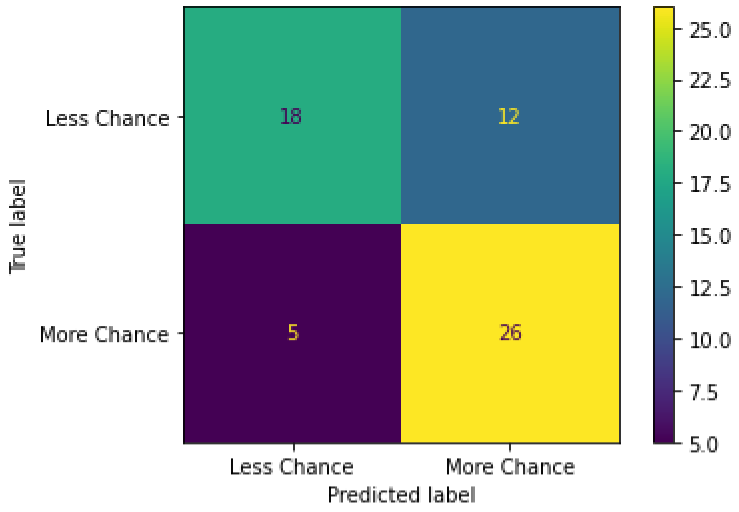


```
----------------------------------

AUC-ROC Score: 85.93%
Precision Score: 83.87%
Recall Score: 68.42%
F Score: 75.36%
Accuracy Score: 72.13%

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

Confusion matrix:
```



Ensemble Random Forest Classifier Model

In [24]:

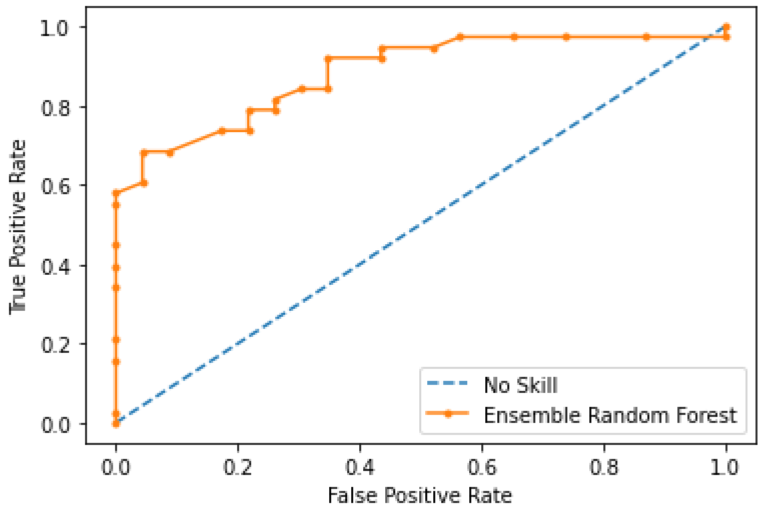
```
rfc = RandomForestClassifier(n_estimators = 100)
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)

print("Ensemble Random Forest Classifier Results:")
print("----------------------------------\n")
ns_probs = [0 for _ in range(len(y_test))]
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

y_probs = rfc.predict_proba(X_test)
rfc_probs = y_probs[:, 1]
rfc_fpr, rfc_tpr, temp = roc_curve(y_test, rfc_probs)
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(rfc_fpr, rfc_tpr, marker='.', label='Ensemble Random Forest')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
print("----------------------------------\n")
print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, rfc_probs) * 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))
print("\n----------------------------------\n")
print("Confusion matrix:")
cm = confusion_matrix(y_pred,y_test)
ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
plt.show()
```

```
Ensemble Random Forest Classifier Results:
----------------------------------
```

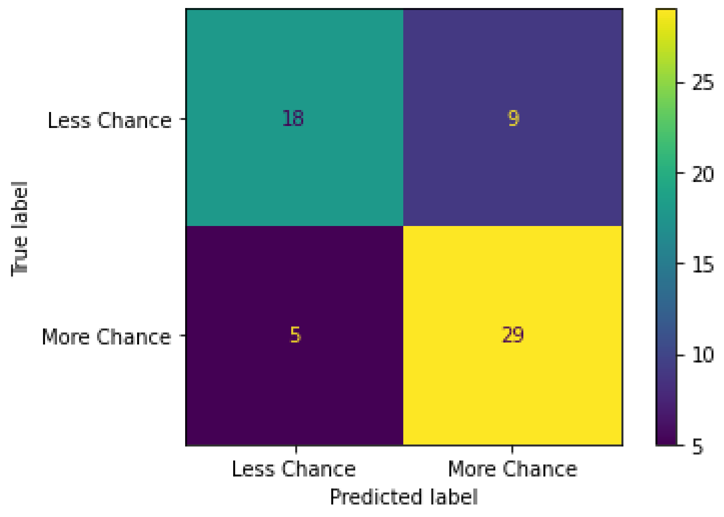


```
----------------------------------

AUC-ROC Score: 88.39%
Precision Score: 85.29%
Recall Score: 76.32%
F Score: 80.56%
Accuracy Score: 77.05%

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

Confusion matrix:
```



K Nearest Neighbor Model

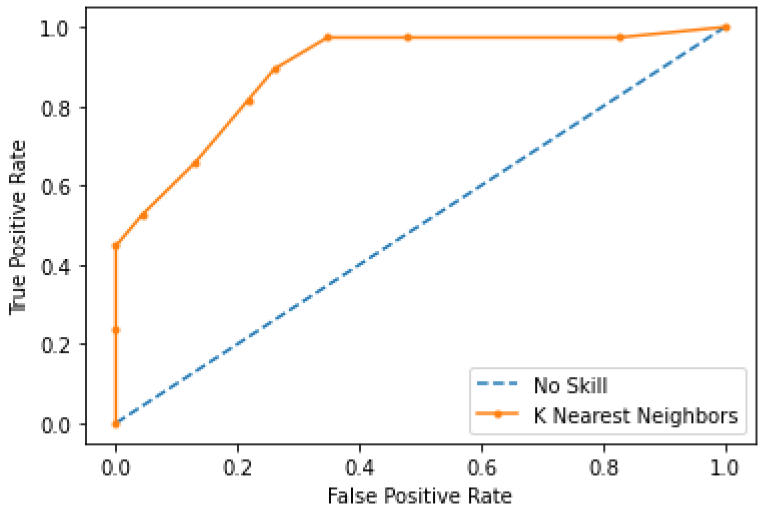
In [25]:
```
model = KNeighborsClassifier(algorithm='auto', leaf_size = 1, p=1 , n_neighbors=9)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

print("KNN Results:")
print("----------------------------------\n")
ns_probs = [0 for _ in range(len(y_test))]
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

y_probs = model.predict_proba(X_test)
knn_probs = y_probs[:, 1]
knn_fpr, knn_tpr, temp = roc_curve(y_test, knn_probs)
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(knn_fpr, knn_tpr, marker='.', label='K Nearest Neighbors')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
print("----------------------------------\n")
print("AUC-ROC Score: {0:0.2f}%".format(roc_auc_score(y_test, knn_probs) * 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))
print("\n----------------------------------\n")
print("Confusion matrix:")
cm = confusion_matrix(y_pred,y_test)
```

```
ConfusionMatrixDisplay(cm,display_labels =["Less Chance","More Chance"]).plot()
plt.show()
```

```
KNN Results:
----------------------------------
```

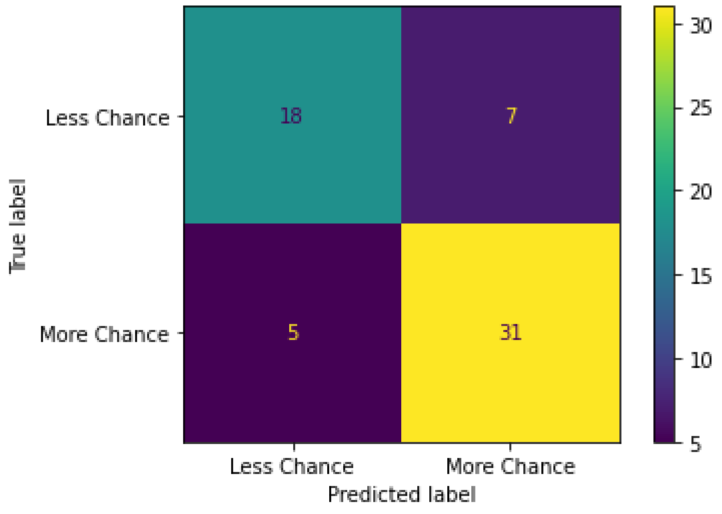


```
----------------------------------

AUC-ROC Score: 89.24%
Precision Score: 86.11%
Recall Score: 81.58%
F Score: 83.78%
Accuracy Score: 80.33%

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

Confusion matrix:
```



Comparing the results of these models:

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

Model/Metric	AUC-ROC	Precision	Recall	F_Score	Accuracy
Guassian NB	87.99%	83.33%	78.95%	81.08%	77.05%
Logistic Regression	89.36%	88.57%	81.58%	84.93%	81.97%
Decision Tree	85.93%	83.87%	68.42%	75.36%	72.13%
Ensemble Random Forest Classifier	88.39%	85.29%	76.32%	80.56%	77.05%
K Nearest Neighbors Classifier	89.24%	86.11%	81.58%	83.78%	80.33%

This shows that the **maximum Accuracy is obtained while performing logistic regression**.

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 contribute to the better Precision and Recall in the respective models**.

Finally, we know that:

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

This goes to show why **logistic regression has a better F score overall** since its **recall and precision is better** than the other models.

Since **logistic regression has better accuracy, precision and recall** we can say that **logistic regression is a better model at prediction for the given data** when comparing it to the rest.

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