ORCHID INTERNATIONAL College

(Affiliated to Tribhuvan University)

A
Lab Report
On

Predicting Heart Disease Using Machine Learning

Submitted By:

Name: Bivek Chaudhary

Faculty: BSc. CSIT

Roll No: 23871/076

Batch: 2076

Year: IV

Semester: VIII

Submitted To:

Department of CSIT

Orchid International College

This report tells discusses various Python-based ML and Data Science libraries in an attempt to build a Machine Learning model capable of predicting whether or not someone has heart disease based on their medical attributes. We're going to take the following approach:

1. Problem Definition
2. Understanding Features
3. Data Preparation and its tools
4. Exploratory Data Analysis
5. Modelling
6. Model Evaluation

Problem Definition

Given clinical parameters about a patient, can we predict whether or not they have heart disease? Our aim is to reach the model accuracy of more than 85% . If the model scores better than 85%, we will select the model.

Understanding Features

a. Heart_disease: It is the target variable indicating the presence of heart disease (object type) into the value of:
   0 = No
   1 = Yes
b. BMI: It Stands for Body Mass Index, a measure of body fat based on height and weight (float) which is in numeric values.
c. Smoking: It indicates if the individual smokes (object) or not categorized into:
   0=No
   1= Yes
d. Alcohol_drinking: It indicates if the individual consumes alcohol (object) or not categorized into:
   0=No
   1= Yes
e. Stroke: It indicates if the individual has had a stroke (object)) or not categorized into:
   0=No
   1= Yes
f. Physical_health: Number of days the individual experienced poor physical health in the past 30 days (float).

g. Mental_health: Number of days the individual experienced poor mental health in the past 30 days (float).
h. Diff_walking: It indicates if the individual has difficulty walking (object) whose format:
   0=No
   1= Yes
i. Sex: Gender of the individual (object)
   0=Female
   1= Male
j.
k. Age_category: It displays the age category of the individual (object) whose format:
   18-24': 21,
   '25-29': 27,
   '30-34': 32,
   '35-39': 37,
   '40-44': 42,
   '45-49': 47,
   '50-54': 52,
   '55-59': 57,
   '60-64': 62,
   '65-69': 67,
   '70-74': 72,
   '75-79': 77,
   '80 or older': 85
l. Race: It displays the race of the individual (object), categorized into:
   'White': 1,
   'Black': 2,
   'Asian': 3,
   'American Indian/Alaskan Native': 4,
   'Other': 5,
   'Hispanic': 6
m. Diabetic: It indicates if the individual has diabetes (object) whose format:
   0=No
   1= Yes

n. Physical_activity: Indicates if the individual engages in physical activity (object).
o. General_health: Self-assessed general health status (object).
p. Sleep_time: Average hours of sleep per night (float).
q. Asthma: Indicates if the individual has asthma (object) whose format:
   0=No
   1= Yes
r. Kidney_disease: Indicates if the individual has kidney disease (object) whose format:
   0=No
   1= Yes
s. Skin_cancer: Indicates if the individual has skin cancer (object) whose format:
   0=No
   1= Yes

This dataset includes a mix of categorical and numerical features, capturing various health-related attributes and behaviors that can influence heart disease risk.

Data preparation and its tools

a. Pandas & NumPy: Pandas offers powerful data structures like Series and DataFrame for structured data manipulation, while NumPy provides support for numerical computing with multi-dimensional arrays and mathematical functions.
b. Matplotlib & Seaborn: Matplotlib is a versatile library for creating static, interactive, and animated visualizations, while Seaborn simplifies the process of generating complex statistical graphics on top of Matplotlib.
c. Scikit-Learn: Scikit-Learn is a comprehensive machine learning library with various algorithms for supervised and unsupervised learning tasks, along with utilities for model evaluation and selection.

In [1]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Load the dataset
df = pd.read_csv('heart_disease.csv')
```

In [3]:
```
# Display the first few rows of the dataframe
df.head()
```

Out[3]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	Ge
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Yes	V
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Yes	V
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Yes	
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	No	
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Yes	V

Remaining the columns

In [4]:
```
df.rename(columns = {'HeartDisease':'Heart_disease','AlcoholDrinking':'Alcohol_drinking','PhysicalHealth':'Physical_health',
```

In [5]:
```
# View of the Renamed Dataframe
df.head()
```

Out[5]:

	Heart_disease	BMI	Smoking	Alcohol_drinking	Stroke	Physical_health	Mental_health	Diff_walking	Sex	Age_category	Race	Diabetic	Physical_activi
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Y
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Y
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Y
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	N
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Y

In [6]:
```
# Replace "Very good" with "Very_good" in General_health
df['General_health'] = df['General_health'].replace('Very good', 'Very_good')

# Replace "No, borderline diabetes" with "No" in Diabetic
df['Diabetic'] = df['Diabetic'].replace('No, borderline diabetes', 'No')

# Replace "Yes (during pregnancy)" with "Yes" in Diabetic
df['Diabetic'] = df['Diabetic'].replace('Yes (during pregnancy)', 'Yes')
```

Exploratory Data Analysis

In [7]:
```
# Exploratory Data Analysis (EDA)
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Heart_disease      319795 non-null  object 
 1   BMI                319789 non-null  float64
 2   Smoking            319721 non-null  object 
 3   Alcohol_drinking   319680 non-null  object 
 4   Stroke             319774 non-null  object 
 5   Physical_health    319778 non-null  float64
 6   Mental_health      319781 non-null  float64
 7   Diff_walking       319762 non-null  object 
 8   Sex                319482 non-null  object 
 9   Age_category       319751 non-null  object 
 10  Race               319756 non-null  object 
 11  Diabetic           319762 non-null  object 
 12  Physical_activity  319779 non-null  object 
 13  General_health     319302 non-null  object 
 14  Sleep_time         319576 non-null  float64
 15  Asthma             319795 non-null  object 
 16  Kidney_disease     319778 non-null  object 
 17  Skin_cancer        319795 non-null  object 
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

Out[7]:

```
Heart_disease          0
BMI                    6
Smoking               74
Alcohol_drinking     115
Stroke                21
Physical_health       17
Mental_health         14
Diff_walking          33
Sex                  313
Age_category          44
Race                  39
Diabetic              33
Physical_activity     16
General_health       493
Sleep_time           219
Asthma                 0
Kidney_disease        17
Skin_cancer            0
dtype: int64
```

Handling the missing data using SimpleImputer

In [8]:
```
# Numerical columns to impute
from sklearn.impute import SimpleImputer
numerical_cols = ['BMI', 'Physical_health', 'Mental_health', 'Sleep_time']
imputer = SimpleImputer(strategy='median')
df[numerical_cols] = imputer.fit_transform(df[numerical_cols])
```

In [9]:
```
# Categorical columns to impute
categorical_cols = ['Smoking', 'Alcohol_drinking', 'Stroke', 'Diff_walking',
                    'Sex', 'Age_category', 'Race', 'Diabetic', 'Physical_activity',
                    'General_health', 'Kidney_disease']
imputer = SimpleImputer(strategy='most_frequent')
df[categorical_cols] = imputer.fit_transform(df[categorical_cols])
```

In [10]:
```
# Confirm all missing values are handled
print(df.isnull().sum())
```

```
Heart_disease        0
BMI                  0
Smoking              0
Alcohol_drinking     0
Stroke               0
Physical_health      0
Mental_health        0
Diff_walking         0
Sex                  0
Age_category         0
Race                 0
Diabetic             0
Physical_activity    0
General_health       0
Sleep_time           0
Asthma               0
Kidney_disease       0
Skin_cancer          0
dtype: int64
```

In [11]:
```
df.head()
```

Out[11]:

	Heart_disease	BMI	Smoking	Alcohol_drinking	Stroke	Physical_health	Mental_health	Diff_walking	Sex	Age_category	Race	Diabetic	Physical_activi
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Y
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Y
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Y
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	N
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Y

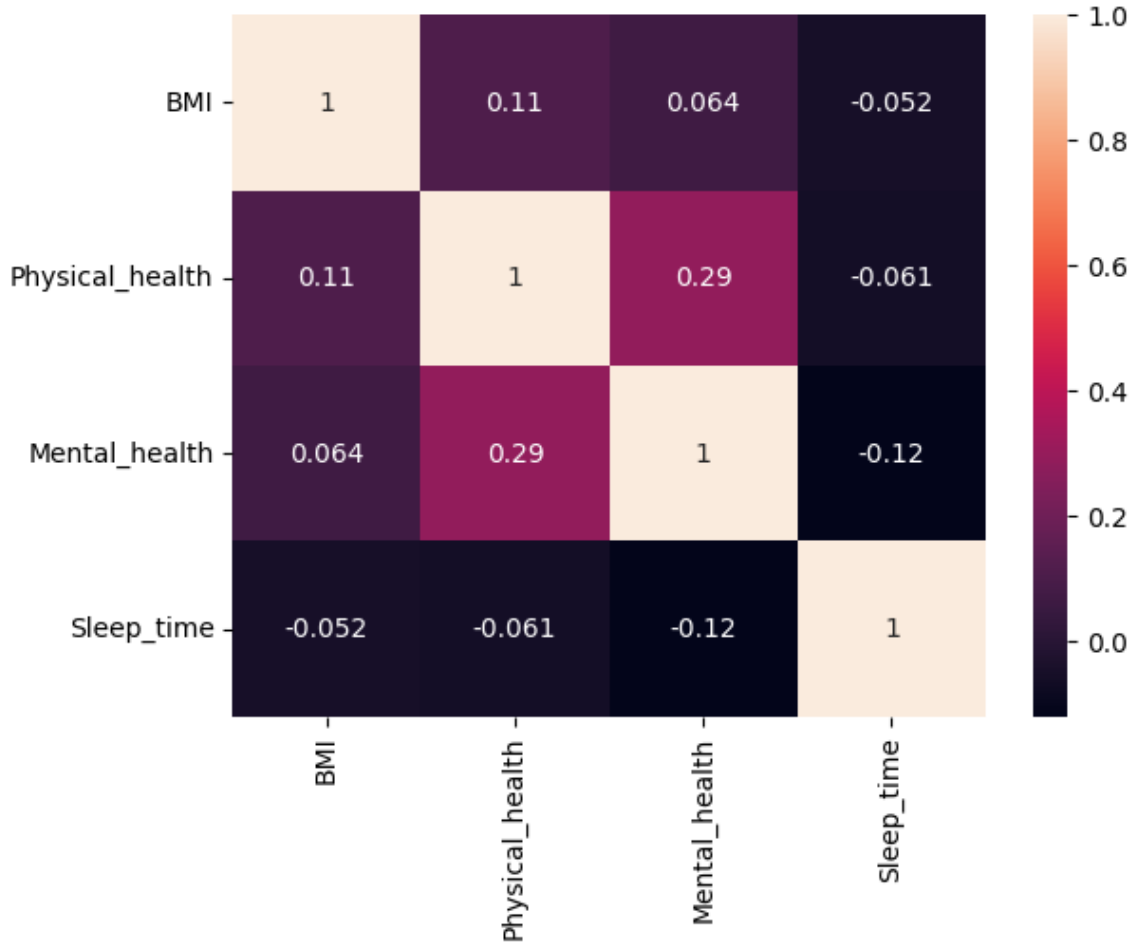
Correlation matrix & Matrix Visualisation

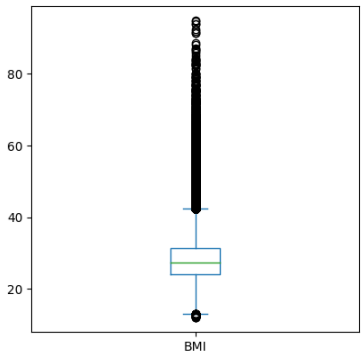
In [12]:
```
df.corr(numeric_only=True)
```

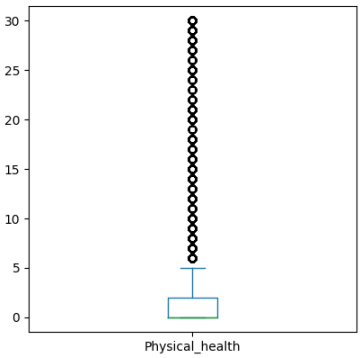
Out[12]:

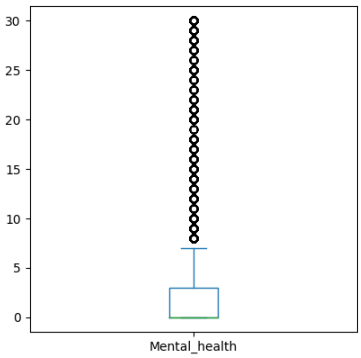
	BMI	Physical_health	Mental_health	Sleep_time
BMI	1.000000	0.109787	0.064127	-0.051801
Physical_health	0.109787	1.000000	0.287948	-0.061384
Mental_health	0.064127	0.287948	1.000000	-0.119692
Sleep_time	-0.051801	-0.061384	-0.119692	1.000000

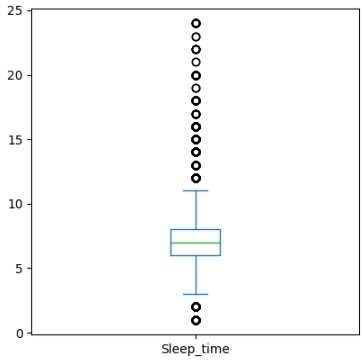
In [13]:
```
# Visualizing the correlation matrix
# Create the correlation heatmap
sns.heatmap(df.corr(numeric_only=True), annot = True);
```

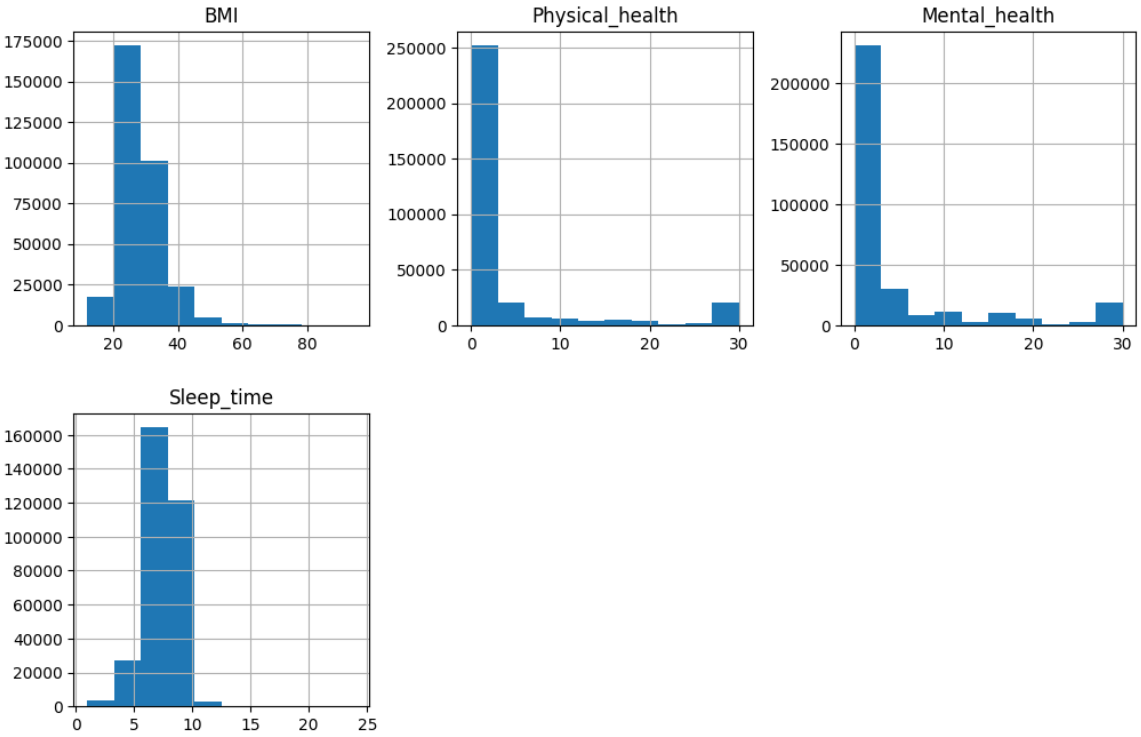

BMI
Physical_health
Mental_health
Sleep_time
1
0.11
0.064
-0.052
0.11
1
0.29
-0.061
0.064
0.29
1
-0.12
-0.052
-0.061
-0.12
1
1.0
0.8
0.6
0.4
0.2
0.0

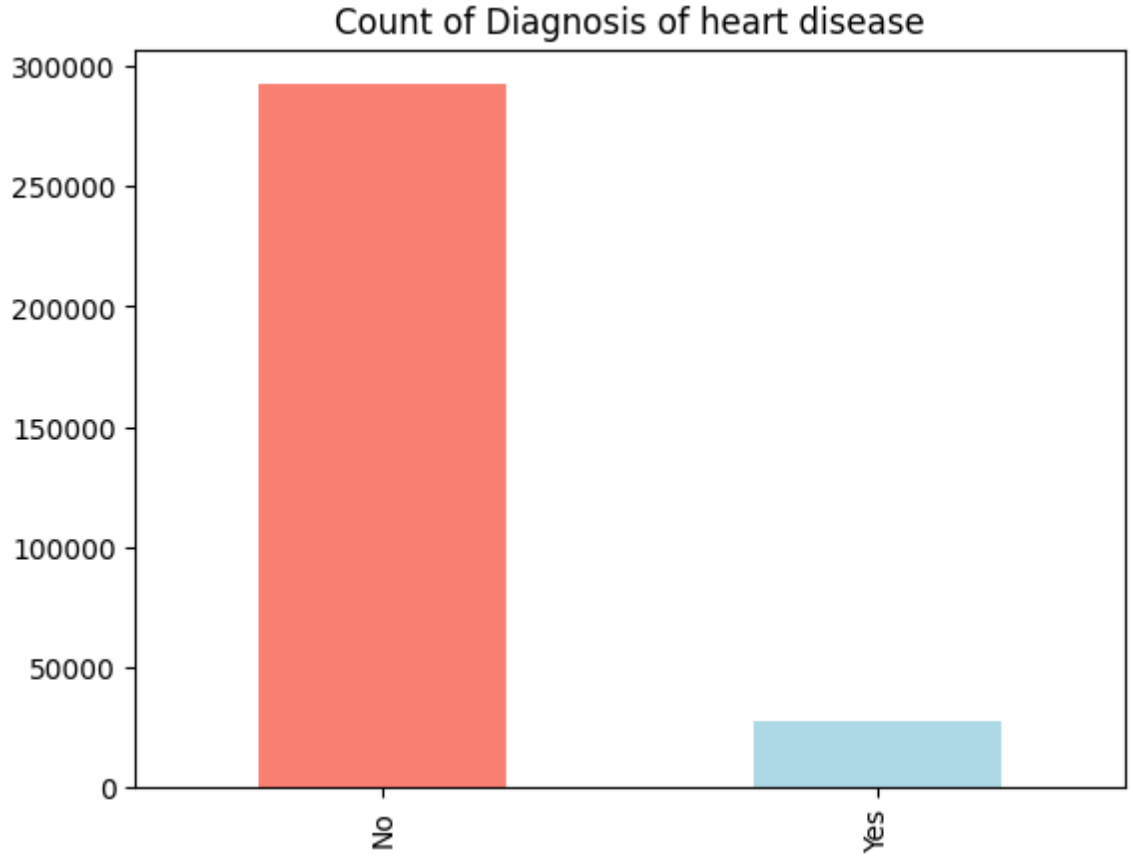

BMI


Physical_health


Mental_health


Sleep_time





Interpretation:

The above plot shows the count of population having disease and not having disease where the count of having disease is low and the count of not having disease is high

Heart Disease vs Sex

In [17]:
```
# Male and Female
df['Sex'].value_counts()
```

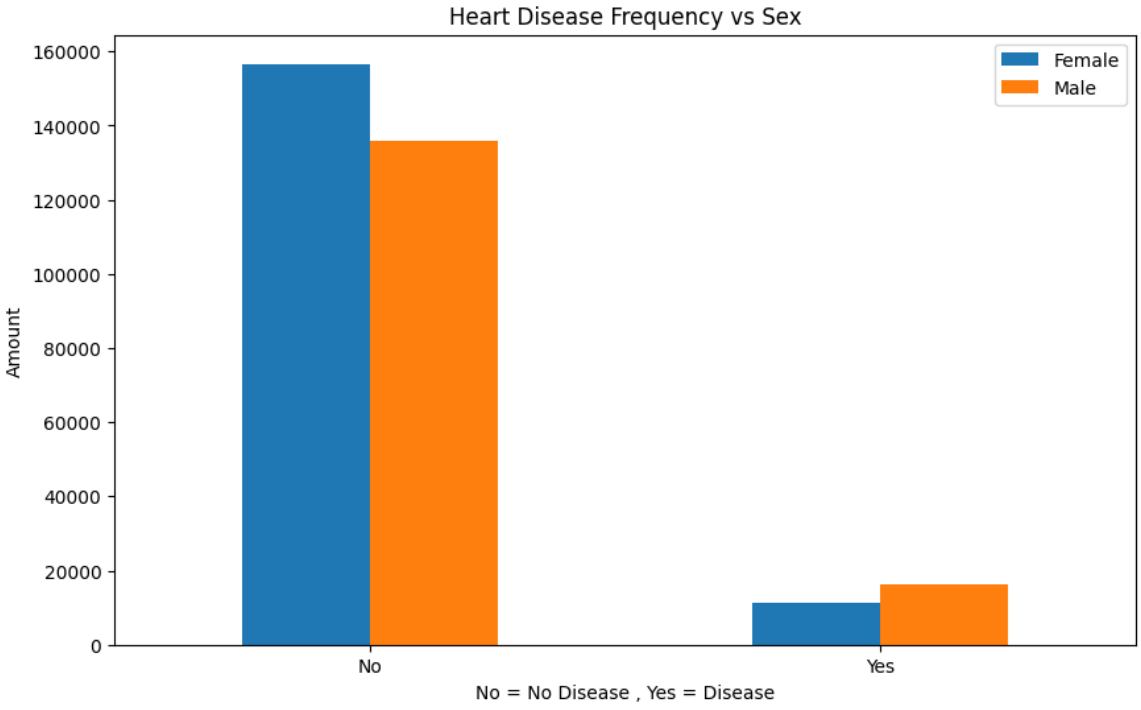
Out[17]:
```
Female    167904
Male      151891
Name: Sex, dtype: int64
```

In [18]:
```
#Compare Target column i.e. Heart Disease with Sex Column
pd.crosstab(df['Heart_disease'],df['Sex'])
```

Out[18]:

Sex	Female	Male
Heart_disease		
No	156648	135774
Yes	11256	16117

In [19]:
```
#Create a plot for crosstab
pd.crosstab(df['Heart_disease'],df['Sex']).plot(kind="bar",figsize=(10,6));
plt.title("Heart Disease Frequency vs Sex")
plt.xlabel("No = No Disease , Yes = Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])
plt.xticks(rotation=0)
```

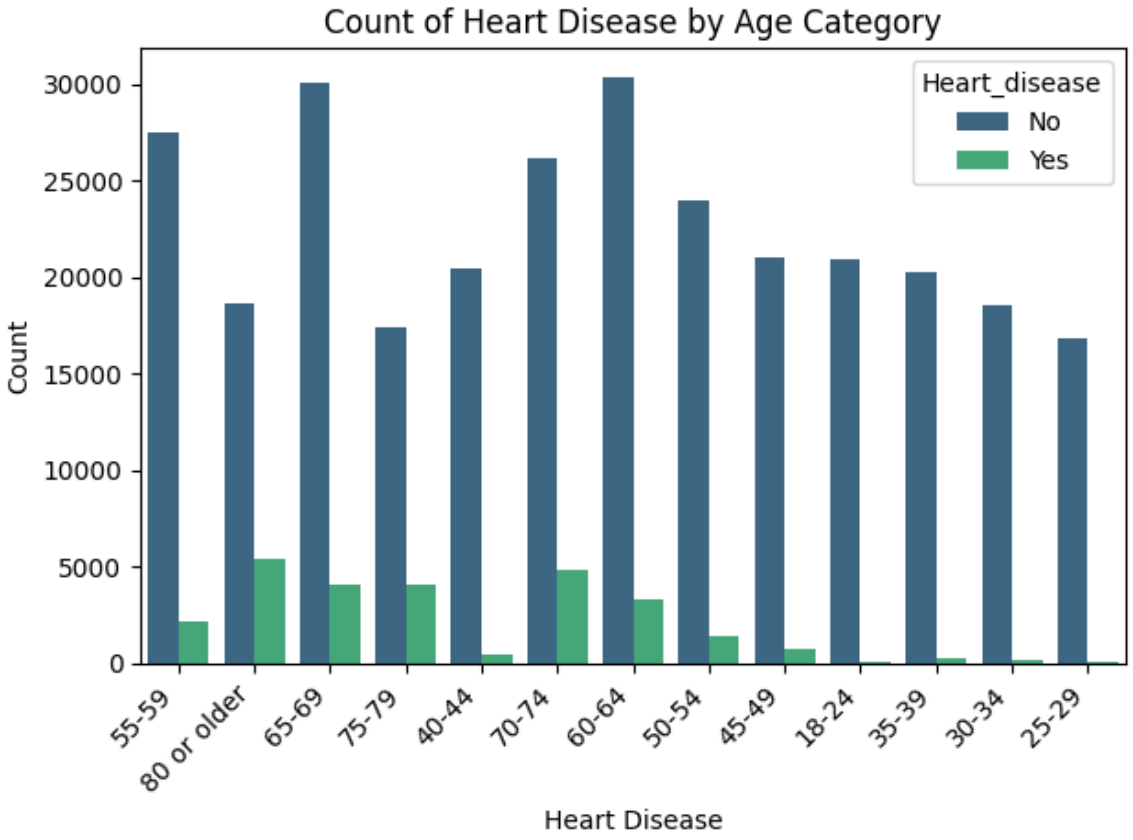


Interpretation:

The above plot shows the ratio of population having disease and not having disease wrt sex

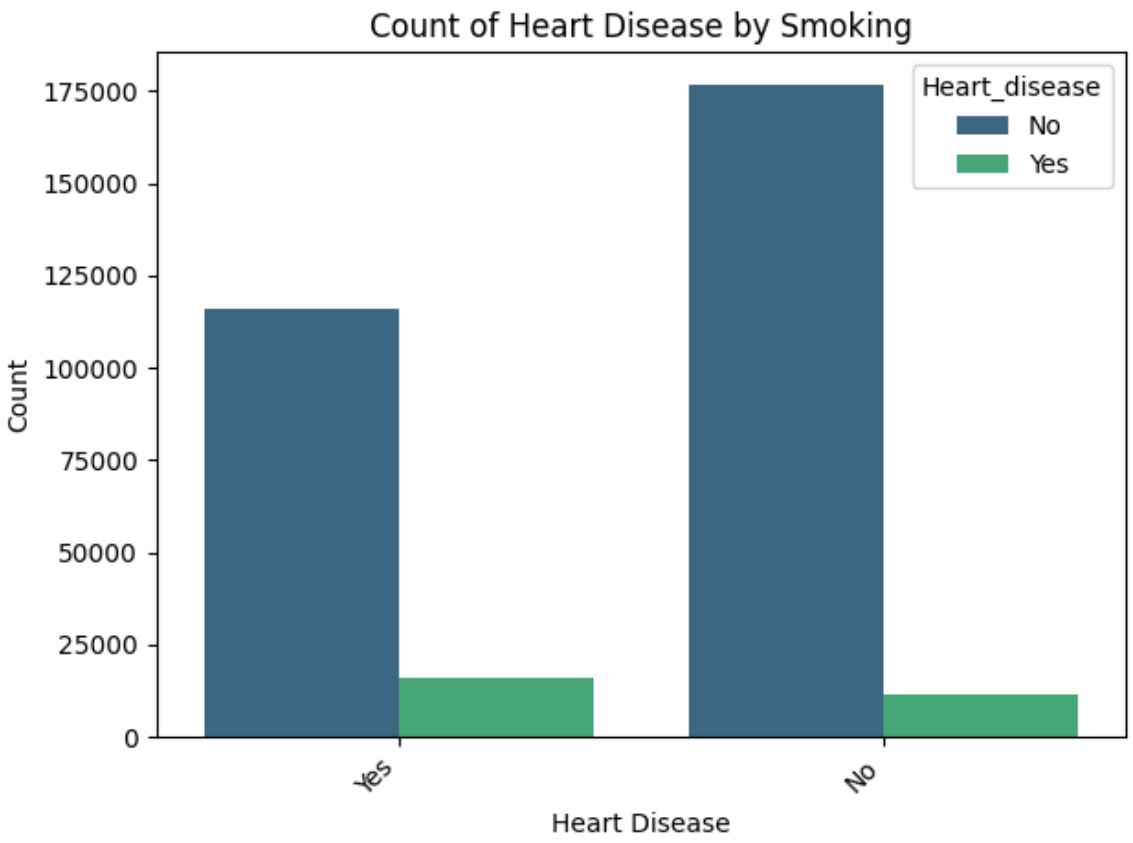
Heart Disease vs Age

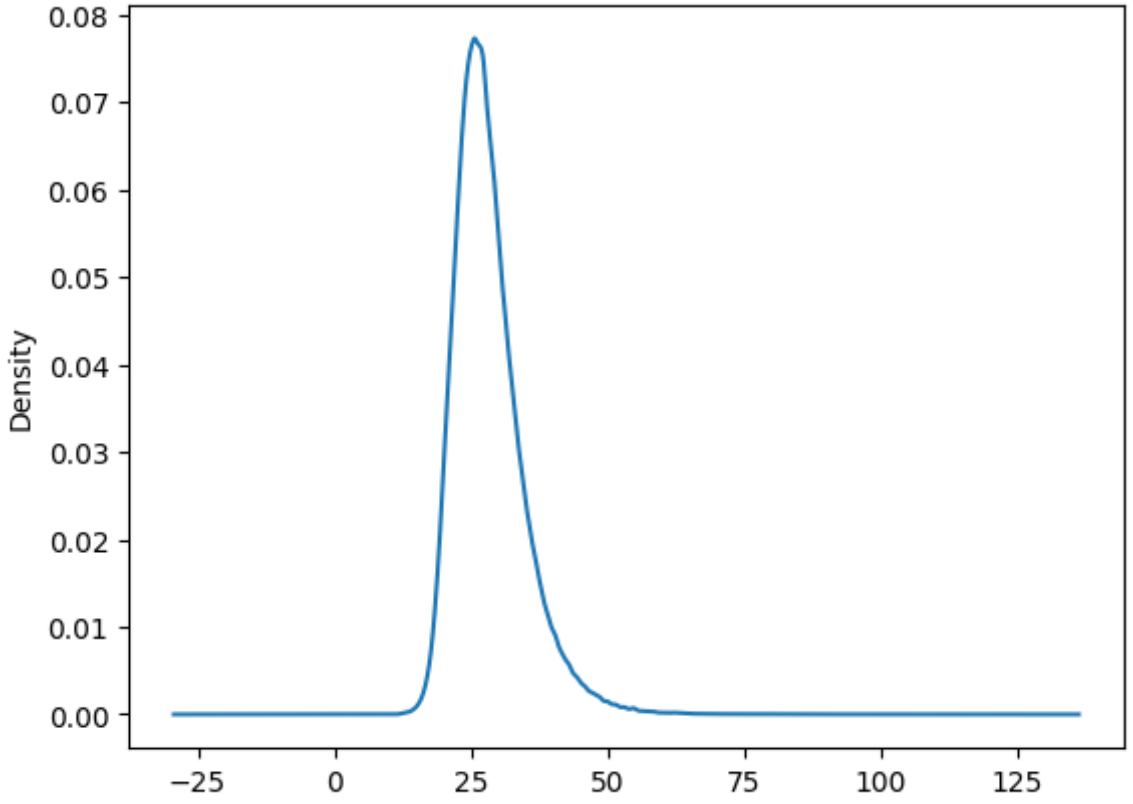
In [20]:
```
sns.countplot(x='Age_category', hue='Heart_disease', palette='viridis', data=df)

# Display the plot
plt.xlabel('Heart Disease')
plt.ylabel('Count')
plt.title('Count of Heart Disease by Age Category')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Count plot of Heart Disease vs Smoking

In [21]:
```
# Create a plot for countplot of Heart_disease vs Smoking
sns.countplot(x='Smoking', hue='Heart_disease', palette='viridis', data=df)

# Display the plot
plt.xlabel('Heart Disease')
plt.ylabel('Count')
plt.title('Count of Heart Disease by Smoking')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```





Interpretation:
The above plot shows the density of BMI and the maximum density is around 23-26

In [23]:

```python
# Define the mapping for age categories to continuous values
age_mapping = {
    '18-24': 21,
    '25-29': 27,
    '30-34': 32,
    '35-39': 37,
    '40-44': 42,
    '45-49': 47,
    '50-54': 52,
    '55-59': 57,
    '60-64': 62,
    '65-69': 67,
    '70-74': 72,
    '75-79': 77,
    '80 or older': 85
}

# Apply the mapping to the Age_category column
df['Age_category'] = df['Age_category'].map(age_mapping)
```

In [24]:

```python
print('Heart_disease:', df['Heart_disease'].nunique(),df['Heart_disease'].unique())
print('Smoking:', df['Smoking'].nunique(),df['Smoking'].unique())
print('Alcohol_drinking:',df['Alcohol_drinking'].nunique(),df['Alcohol_drinking'].unique())
print('Stroke:',df['Stroke'].nunique(),df['Stroke'].unique())
print('Diff_waliking:',df['Diff_walking'].nunique(),df['Diff_walking'].unique())
print('Race:',df['Race'].nunique(),df['Race'].unique())
print('Diabetic:',df['Diabetic'].nunique(),df['Diabetic'].unique())
print('Physical_health:',df['Physical_health'].nunique(),df['Physical_health'].unique())
print('General_health:',df['General_health'].nunique(),df['General_health'].unique())
print('Asthma:',df['Asthma'].nunique(),df['Asthma'].unique())
print('Kidney_disease:',df['Kidney_disease'].nunique(),df['Kidney_disease'].unique())
print('Skin_cancer:',df['Skin_cancer'].nunique(),df['Skin_cancer'].unique())
```

```
Heart_disease: 2 ['No' 'Yes']
Smoking: 2 ['Yes' 'No']
Alcohol_drinking: 2 ['No' 'Yes']
Stroke: 2 ['No' 'Yes']
Diff_waliking: 2 ['No' 'Yes']
Race: 6 ['White' 'Black' 'Asian' 'American Indian/Alaskan Native' 'Other'
 'Hispanic']
Diabetic: 2 ['Yes' 'No']
Physical_health: 31 [ 3.  0. 20. 28.  6. 15.  5. 30.  7.  1.  2. 21.  4. 10. 14. 18.  8. 25.
 16. 29. 27. 17. 24. 12. 23. 26. 22. 19.  9. 13. 11.]
General_health: 5 ['Very_good' 'Fair' 'Good' 'Poor' 'Excellent']
Asthma: 2 ['Yes' 'No']
Kidney_disease: 2 ['No' 'Yes']
Skin_cancer: 2 ['Yes' 'No']
```

In [25]:

```
# Convert categorical columns to numerical values
df['Heart_disease'] = df['Heart_disease'].map({'No': 0, 'Yes': 1})
df['Smoking'] = df['Smoking'].map({'No': 0, 'Yes': 1})
df['Alcohol_drinking'] = df['Alcohol_drinking'].map({'No': 0, 'Yes': 1})
df['Stroke'] = df['Stroke'].map({'No': 0, 'Yes': 1})
df['Diff_walking'] = df['Diff_walking'].map({'No': 0, 'Yes': 1})
df['Sex'] = df['Sex'].map({'Male': 0, 'Female': 1})
race_map = {
    'White': 1,
    'Black': 2,
    'Asian': 3,
    'American Indian/Alaskan Native': 4,
    'Other': 5,
    'Hispanic': 6
}

df['Race'] = df['Race'].map(race_map)
df['Asthma'] = df['Asthma'].map({'No': 0, 'Yes': 1})
df['Kidney_disease'] = df['Kidney_disease'].map({'No': 0, 'Yes': 1})
df['Skin_cancer'] = df['Skin_cancer'].map({'No': 0, 'Yes': 1})

# Map General_health to numerical values
health_map = {
    'Excellent': 1,
    'Very_good': 2,
    'Good': 3,
    'Fair': 4,
    'Poor': 5
}
df['General_health'] = df['General_health'].map(health_map)

# Map Diabetic to numerical values
df['Diabetic'] = df['Diabetic'].map({'No': 0, 'Yes': 1})
df['Physical_activity'] = df['Physical_activity'].map({'No': 0, 'Yes': 1})

# Display the unique values after mapping
print('Heart_disease:', df['Heart_disease'].nunique(), df['Heart_disease'].unique())
print('Smoking:', df['Smoking'].nunique(), df['Smoking'].unique())
print('Alcohol_drinking:', df['Alcohol_drinking'].nunique(), df['Alcohol_drinking'].unique())
print('Stroke:', df['Stroke'].nunique(), df['Stroke'].unique())
print('Diff_walking:', df['Diff_walking'].nunique(), df['Diff_walking'].unique())
print('Race:', df['Race'].nunique(), df['Race'].unique())
print('Diabetic:', df['Diabetic'].nunique(), df['Diabetic'].unique())
print('Physical_health:', df['Physical_health'].nunique(), df['Physical_health'].unique())
print('General_health:', df['General_health'].nunique(), df['General_health'].unique())
print('Asthma:', df['Asthma'].nunique(), df['Asthma'].unique())
print('Kidney_disease:', df['Kidney_disease'].nunique(), df['Kidney_disease'].unique())
print('Skin_cancer:', df['Skin_cancer'].nunique(), df['Skin_cancer'].unique())
```

```
Heart_disease: 2 [0 1]
Smoking: 2 [1 0]
Alcohol_drinking: 2 [0 1]
Stroke: 2 [0 1]
Diff_walking: 2 [0 1]
Race: 6 [1 2 3 4 5 6]
Diabetic: 2 [1 0]
Physical_health: 31 [ 3.  0. 20. 28.  6. 15.  5. 30.  7.  1.  2. 21.  4. 10. 14. 18.  8. 25.
 16. 29. 27. 17. 24. 12. 23. 26. 22. 19.  9. 13. 11.]
General_health: 5 [2 4 3 5 1]
Asthma: 2 [1 0]
Kidney_disease: 2 [0 1]
Skin_cancer: 2 [1 0]
```

In [26]:

```
df['Heart_disease'].value_counts()
```

Out[26]:

```
0    292422
1     27373
Name: Heart_disease, dtype: int64
```

In [27]:

```
#Representation of Target variable in Percentage

countNoDisease = len(df[df.Heart_disease == 0])
countHaveDisease = len(df[df.Heart_disease == 1])
print("Percentage of Patients Haven't Heart Disease: {:.2f}%".format((countNoDisease / (len(df.Heart_disease))*100)))
print("Percentage of Patients Have Heart Disease: {:.2f}%".format((countHaveDisease / (len(df.Heart_disease))*100)))
```

```
Percentage of Patients Haven't Heart Disease: 91.44%
Percentage of Patients Have Heart Disease: 8.56%
```

Modelling

We will experiment with the models, trying the model and getting the results from them. Since in the target variable, Percentage of Patients Haven't Heart Disease: 91.44% Percentage of Patients Have Heart Disease: 8.56% so to imbalance the ratio we use the SMOTETomek which helps in balancing the data by importing Synthetic Minority Oversampling Technique by doing both downsampling and upsampling of class.

In [28]:
```
# Splitting the X and y values we called X as independent variables and y as target variable
X = df.drop('Heart_disease',axis=1)
y = df['Heart_disease']
```

In [30]:
```
#balancing the data by importing Synthetic Minority Oversampling Technique(SMOTE)
from imblearn.combine import SMOTETomek
#smotetomek does both downsampling and upsampling of class
```

In [31]:
```
# Instantiate SMOTETomek with the desired sampling strategy (float between 0 and 1)
# As specified as 0.9 means minprity has 90% of majority
smote_tomek = SMOTETomek(sampling_strategy=0.9)
```

In [32]:
```
# Fit and transform data
X_resampled, y_resampled = smote_tomek.fit_resample(X, y)
```

In [33]:
```
df.head()
```

Out[33]:

	Heart_disease	BMI	Smoking	Alcohol_drinking	Stroke	Physical_health	Mental_health	Diff_walking	Sex	Age_category	Race	Diabetic	Physical_activity
0	0	16.60	1	0	0	3.0	30.0	0	1	57	1	1	1
1	0	20.34	0	0	1	0.0	0.0	0	1	85	1	0	1
2	0	26.58	1	0	0	20.0	30.0	0	0	67	1	1	1
3	0	24.21	0	0	0	0.0	0.0	0	1	77	1	0	0
4	0	23.71	0	0	0	28.0	0.0	1	1	42	1	0	1

In [34]:
```
# Checking the number of majority and minority values
from collections import Counter
Counter(y_resampled)
```

Out[34]:
```
Counter({0: 289909, 1: 260666})
```

In [35]:
```
# Splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size = 0.2, random_state = 0)

# Display the shapes of the training and testing sets
print(f"\nX_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

# Save training and testing sets to CSV files
X_train.to_csv('train.csv', index=False)
X_test.to_csv('test.csv', index=False)
y_train.to_csv('train_data_labels.csv', index=False, header=True)
y_test.to_csv('test_data_labels.csv', index=False, header=True)
```

Now we have got our data split into training and test sets, it is time to build a Machine Learning model. We will train it (find the patterns) on the training set. And we will test it (use the patterns) on the test set.

Building Model

In [37]:
```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classi
```

In [38]:
```
# Random Forest
print("Random Forest")
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

```
Random Forest
```

Out[38]:
```
RandomForestClassifier(random_state=42)
```

Model Evaluation

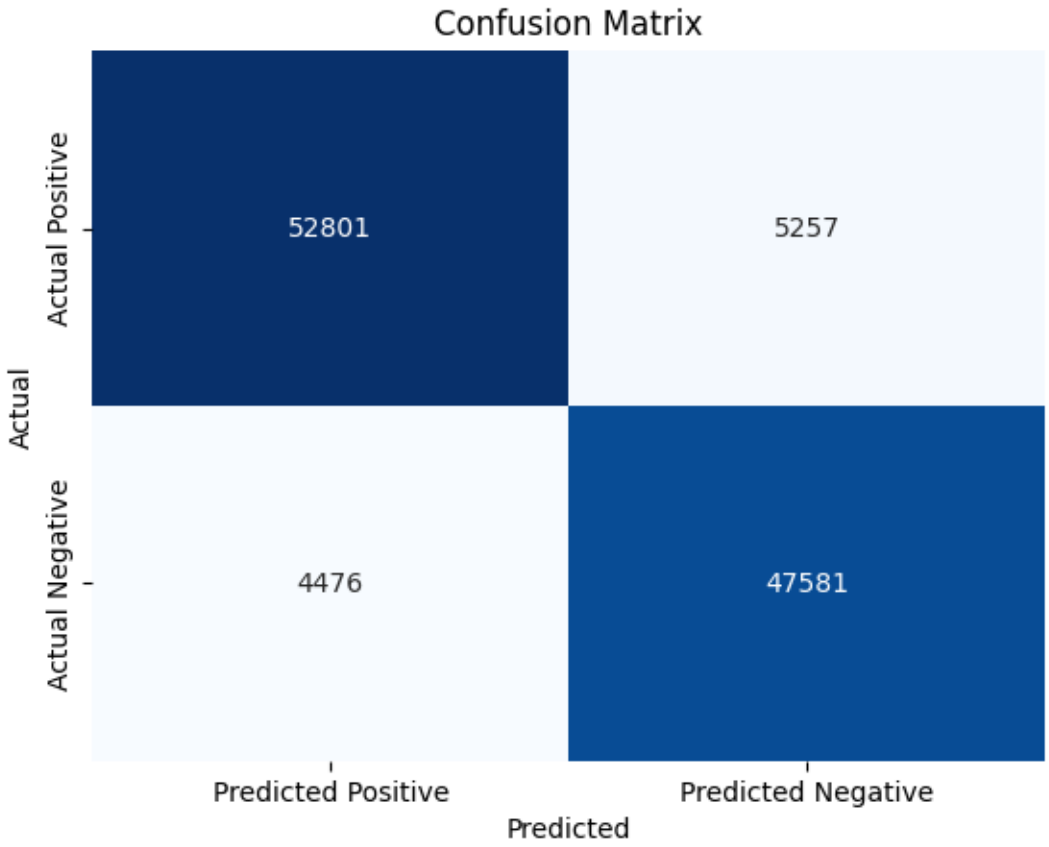
In [52]:

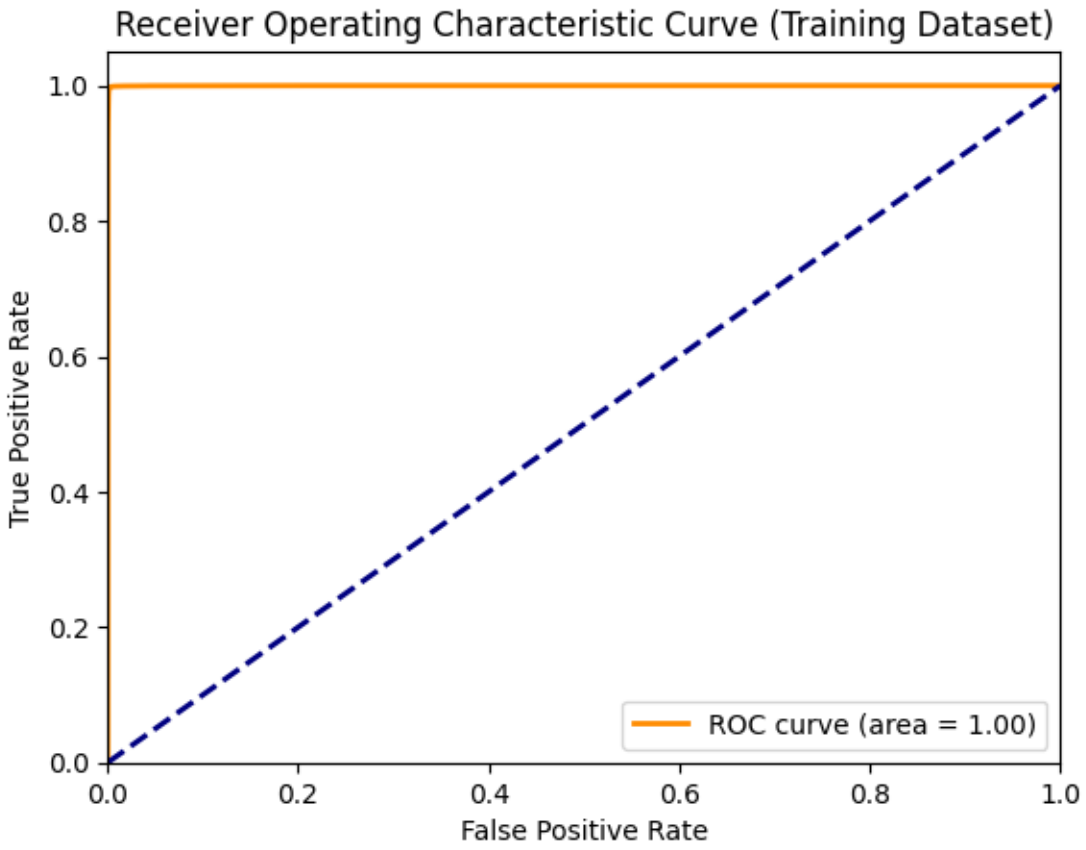
```python
# Function to evaluate and print the performance of a model
def evaluate_model(model,X_train, y_train, X_test, y_test):
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    # Predictions on training set for ROC curve
    y_pred_train_proba = model.predict_proba(X_train)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1 Score: {f1:.4f}')
    print(f'ROC AUC Score: {roc_auc:.4f}')


    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Plot the confusion matrix using seaborn heatmap
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
                xticklabels=["Predicted Positive", "Predicted Negative"],
                yticklabels=["Actual Positive", "Actual Negative"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

    # ROC Curve for Training Dataset
    fpr_train, tpr_train, _ = roc_curve(y_train, y_pred_train_proba)
    roc_auc_train = auc(fpr_train, tpr_train)

    plt.figure()
    lw = 2
    plt.plot(fpr_train, tpr_train, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_train)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve (Training Dataset)')
    plt.legend(loc="lower right")
    plt.show()

    # ROC Curve for Testing Dataset
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend(loc="lower right")
    plt.show()
```

In [53]:

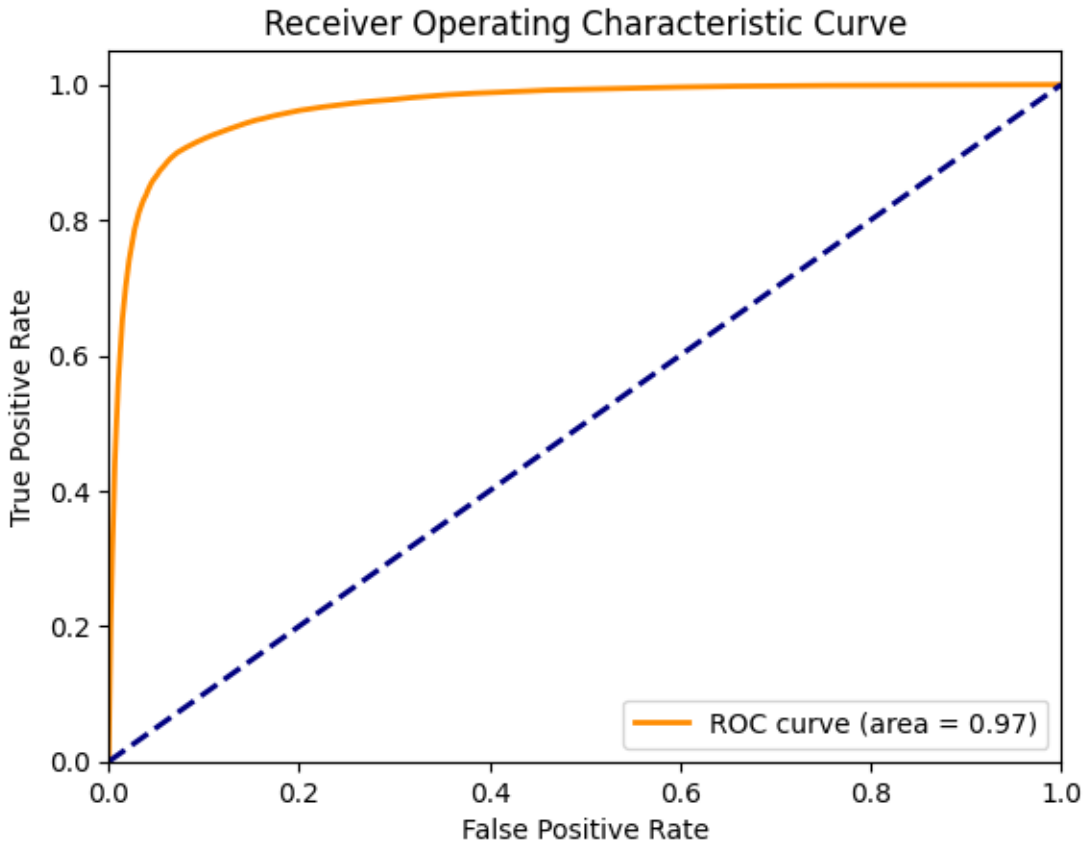
```python
evaluate_model(rf_model, X_train, y_train, X_test, y_test)
```

```
Accuracy: 0.9116
Precision: 0.9005
Recall: 0.9140
F1 Score: 0.9072
ROC AUC Score: 0.9117

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.92     58058
           1       0.90      0.91      0.91     52057

    accuracy                           0.91    110115
   macro avg       0.91      0.91      0.91    110115
weighted avg       0.91      0.91      0.91    110115
```


Confusion Matrix
Actual Positive
Actual Negative
Actual
52801
5257
4476
47581
Predicted Positive
Predicted Negative
Predicted

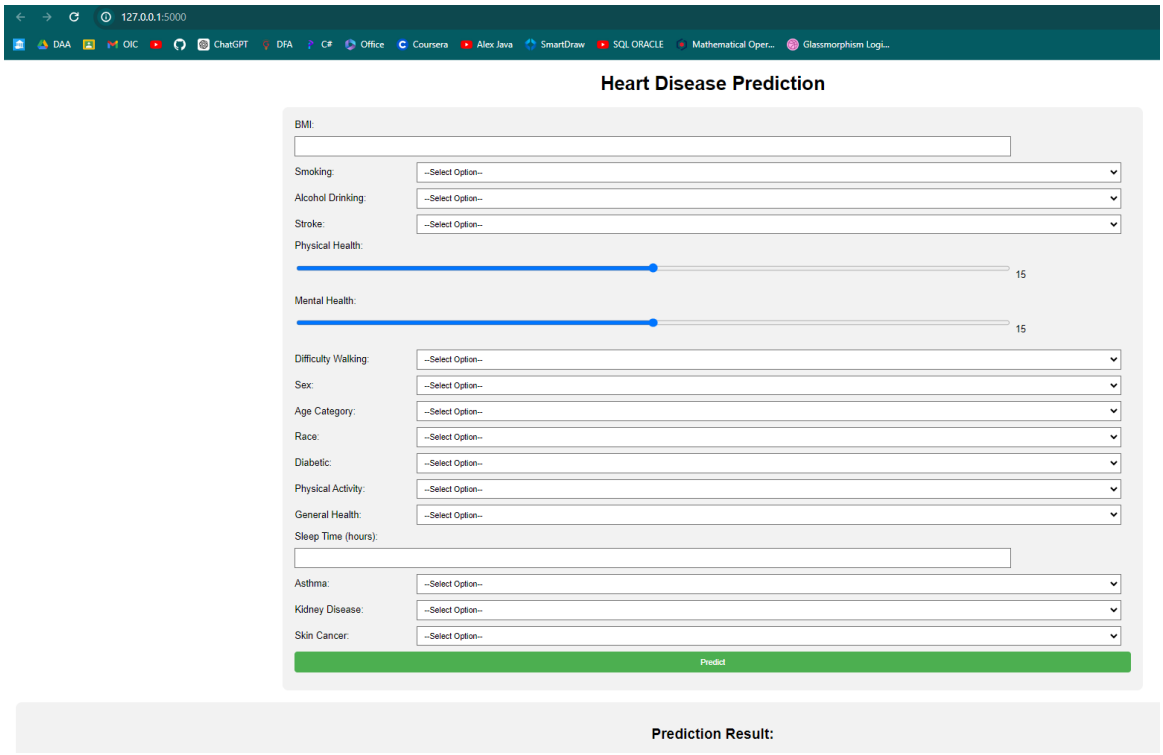

Receiver Operating Characteristic Curve (Training Dataset)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)

In training dataset. ROC Curve shows that area = 1 which is equals to the AUC of 1 that indicates perfect performance.

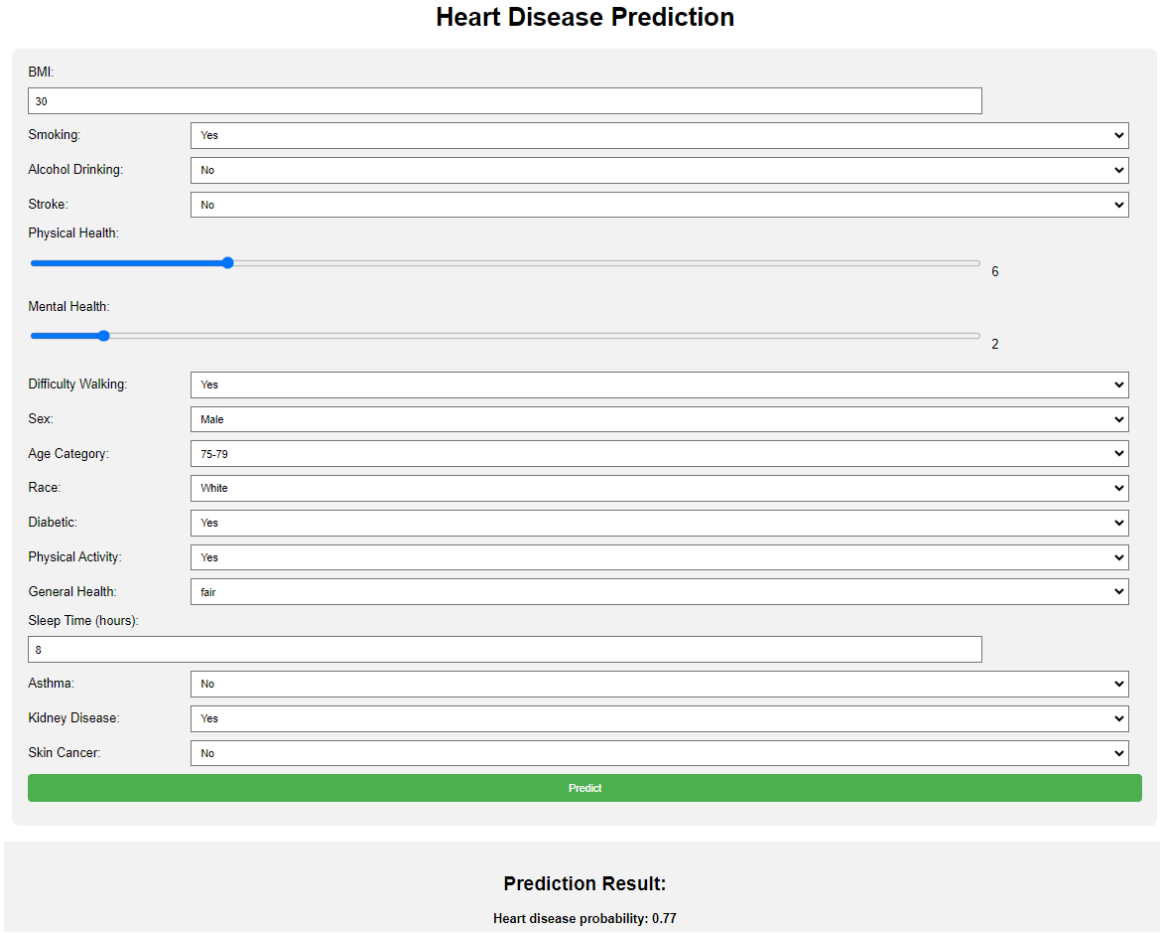


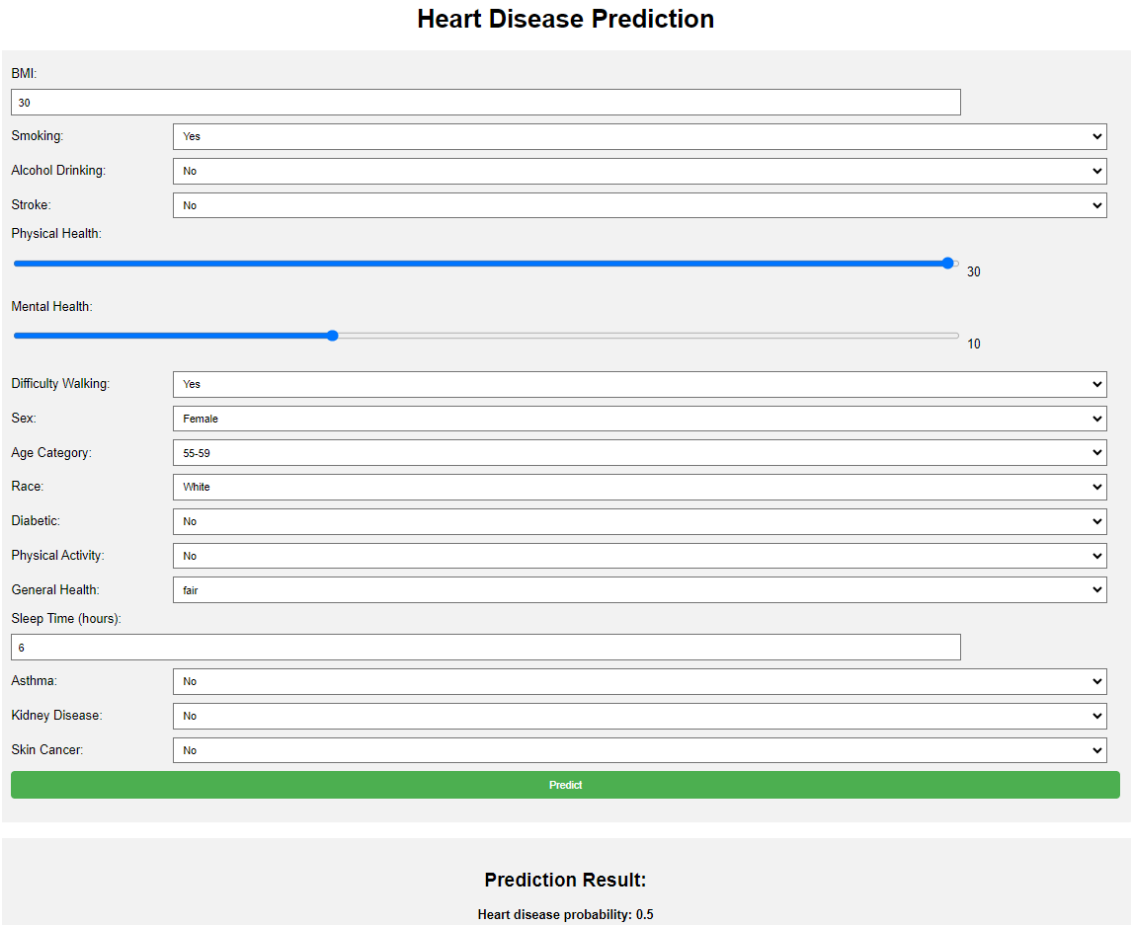
In testing dataset, ROC Curve shows that area = **0.97** which means nearer to the AUC of 1 that indicates better performance.

Form For the Heart Disease Prediction



Result:





Conclusion

From the above prediction analysis, we were able to predict the probability of the heart disease or not as in the result there is the heart disease probability is **0.77** which is suffering from heart disease where as there is also the heart disease probability is **0.5** which is not suffering from heart disease using Random Forest Classifier Model.