Risk of Cardiovascular Disease Prediction

The data set was collected by telephone surveys about U.S residents on their health risk behaviours. The data was already cleaned and so will just require preprocessing.

Hypothesis

Heart disease is caused by multiple factors such as Diabetes, Exercise, BMI, Diet, Height*(cm), Weight*(kg), BMI, Smoking_History, Alcohol_Consumption and sex. So we will be investigating the role of these factors in Heart Disease causation.

Data attributes

In [ ]:
```
#import data and interogate the data structure
import pandas as pd

df = pd.read_csv(r"C:\Users\ka416\OneDrive - University of Exeter\Documents\data\ds-ml
```

In [2]:
```
df.head()
```

Out[2]:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diabetes
0	Poor	Within the past 2 years	No	No	No	No	No	No
1	Very Good	Within the past year	No	Yes	No	No	No	Yes
2	Very Good	Within the past year	Yes	No	No	No	No	Yes
3	Poor	Within the past year	Yes	Yes	No	No	No	Yes
4	Good	Within the past year	No	No	No	No	No	No

columns include: General_Health, Checkup, Exercise,Heart_Disease, Skin_Cancer, Other_Cancer, Depression, Diabetes, Arthritis, Sex, Age*Category, Height*(cm), Weight_(kg), BMI, Smoking_History, Alcohol_Consumption, Fruit_Consumption, Green_Vegetables_Consumption, FriedPotato_Consumption

importing required libraries

In [38]:
```
#
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from xgboost import XGBClassifier
import plotly.express as px

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, roc_
from sklearn.linear_model import Lasso, Ridge
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

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

Preprocessing and Exploratory Data Analysis

In [4]:
```
df.info()
df.shape
df.isnull().sum() #There are no null values if you use isnull.count(..)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308854 entries, 0 to 308853
Data columns (total 19 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   General_Health                308854 non-null  object 
 1   Checkup                       308854 non-null  object 
 2   Exercise                      308854 non-null  object 
 3   Heart_Disease                 308854 non-null  object 
 4   Skin_Cancer                   308854 non-null  object 
 5   Other_Cancer                  308854 non-null  object 
 6   Depression                    308854 non-null  object 
 7   Diabetes                      308854 non-null  object 
 8   Arthritis                     308854 non-null  object 
 9   Sex                           308854 non-null  object 
 10  Age_Category                  308854 non-null  object 
 11  Height_(cm)                   308854 non-null  float64
 12  Weight_(kg)                   308854 non-null  float64
 13  BMI                           308854 non-null  float64
 14  Smoking_History               308854 non-null  object 
 15  Alcohol_Consumption           308854 non-null  float64
 16  Fruit_Consumption             308854 non-null  float64
 17  Green_Vegetables_Consumption  308854 non-null  float64
 18  FriedPotato_Consumption       308854 non-null  float64
dtypes: float64(7), object(12)
memory usage: 44.8+ MB
```

Out[4]:

```
General_Health                  0
Checkup                         0
Exercise                        0
Heart_Disease                   0
Skin_Cancer                     0
Other_Cancer                    0
Depression                      0
Diabetes                        0
Arthritis                       0
Sex                             0
Age_Category                    0
Height_(cm)                     0
Weight_(kg)                     0
BMI                             0
Smoking_History                 0
Alcohol_Consumption             0
Fruit_Consumption               0
Green_Vegetables_Consumption    0
FriedPotato_Consumption         0
dtype: int64
```

In [5]:

```
df.describe()
```

Out[5]:

	Height_(cm)	Weight_(kg)	BMI	Alcohol_Consumption	Fruit_Consumption	Green_
count	308854.000000	308854.000000	308854.000000	308854.000000	308854.000000	
mean	170.615249	83.588655	28.626211	5.096366	29.835200	
std	10.658026	21.343210	6.522323	8.199763	24.875735	
min	91.000000	24.950000	12.020000	0.000000	0.000000	
25%	163.000000	68.040000	24.210000	0.000000	12.000000	
50%	170.000000	81.650000	27.440000	1.000000	30.000000	
75%	178.000000	95.250000	31.850000	6.000000	30.000000	
max	241.000000	293.020000	99.330000	30.000000	120.000000	

check for duplicates

In [6]:

```
df.duplicated().sum()
```

Out[6]:

```
80
```

In [7]:

```
#drop any duplicates
df.drop_duplicates(inplace = True)
df.shape #check if the shape changed with the drop
```

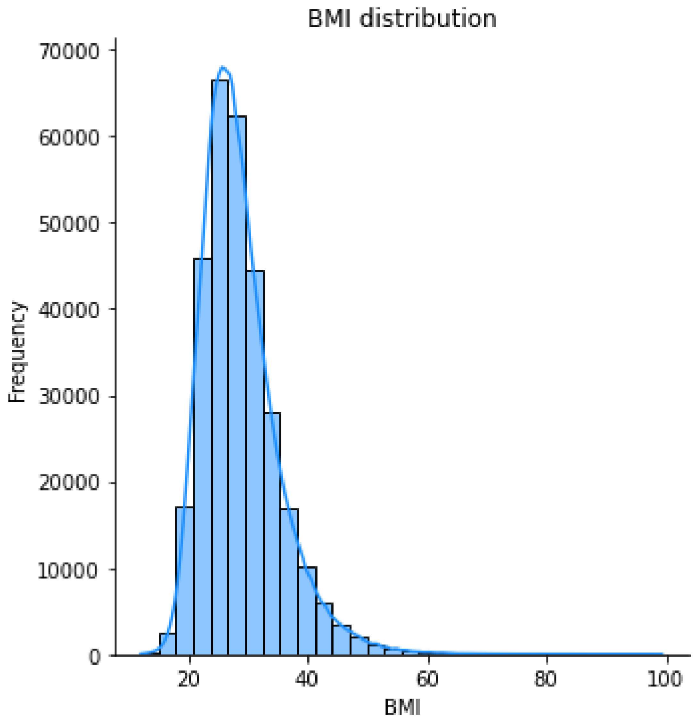
Out[7]:

```
(308774, 19)
```

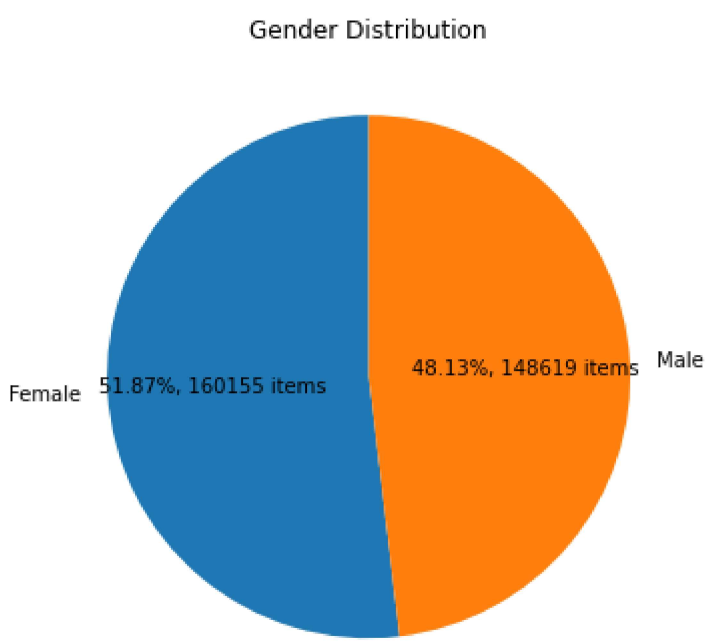
Visualisation of the data

In [8]:

```
#Distribution of BMI

sns.displot(df['BMI'], color="dodgerblue", label="Compact", bins = 30, kde = True)
plt.title('BMI distribution')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.show()
```



In [9]:
```
# Gender Distribution

gender_counts = df['Sex'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(gender_counts, labels=gender_counts.index, autopct=lambda p:f'{p:.2f}%, {p*sum
plt.title('Gender Distribution')
plt.show()
```



In [10]:
```
#fig=plt.figure(figsize=(12,12))

#ax=fig.add_subplot(221)
#sns.countplot(df["Age_Category"], color="red", label="Age_Category", kde=True, ax=ax)
#ax.set_title('Age_Category', fontsize=16)
```

```
#ax=fig.add_subplot(222)
#sns.histplot(df["Heart_Disease"], color="green", label="Heart_Disease", kde=True, ax=
#ax.set_title('Heart_Disease', fontsize=16)

#ax=fig.add_subplot(223)
#sns.histplot(df["Smoking_History"], color="blue", label="Smoking_History", kde=True,
#ax.set_title('Smoking_History', fontsize=16)

#ax=fig.add_subplot(224)
#sns.histplot(df["FriedPotato_Consumption"], color="blue", label="FriedPotato_Consumpt
#ax.set_title('FriedPotato_Consumption', fontsize=16)


#plt.show()
```

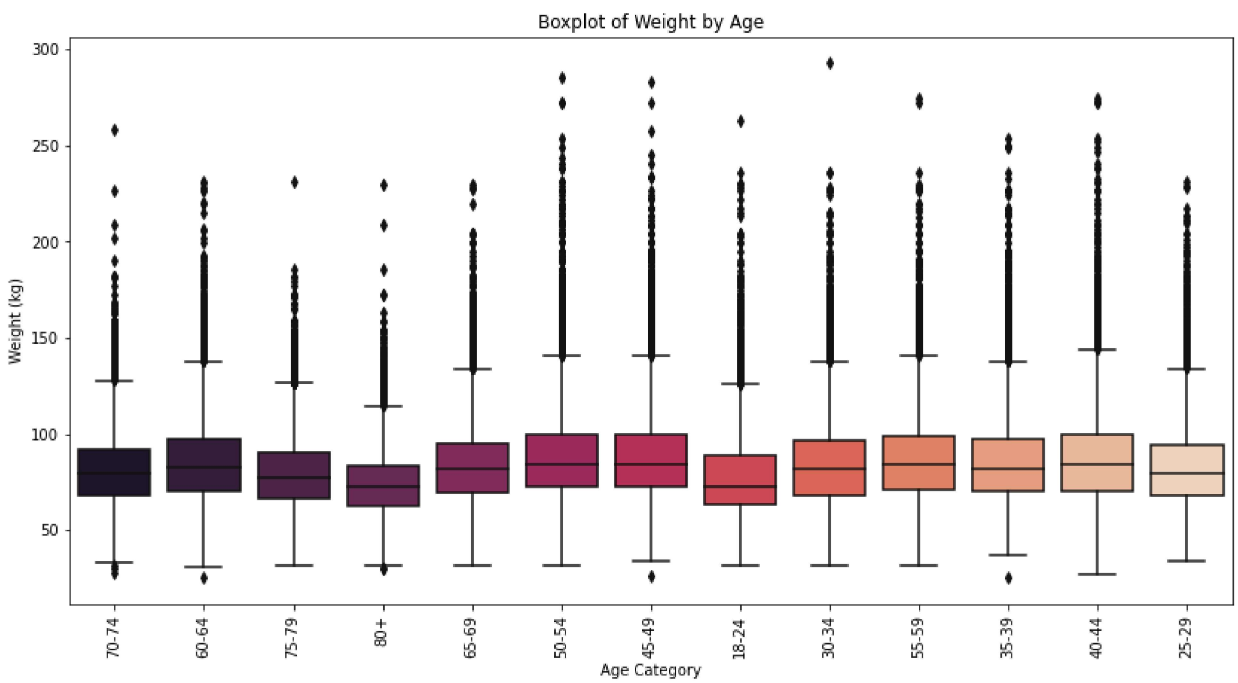
In [ ]:

In [11]:

```
# observing outliers and distribution of Age category and weight in kg
```

In [12]:

```
plt.figure(figsize=(14, 7))
sns.boxplot(x='Age_Category', y='Weight_(kg)', data=df, palette='rocket')
plt.title('Boxplot of Weight by Age')
plt.xlabel('Age Category')
plt.ylabel('Weight (kg)')
plt.xticks(rotation=90)
plt.show()
```

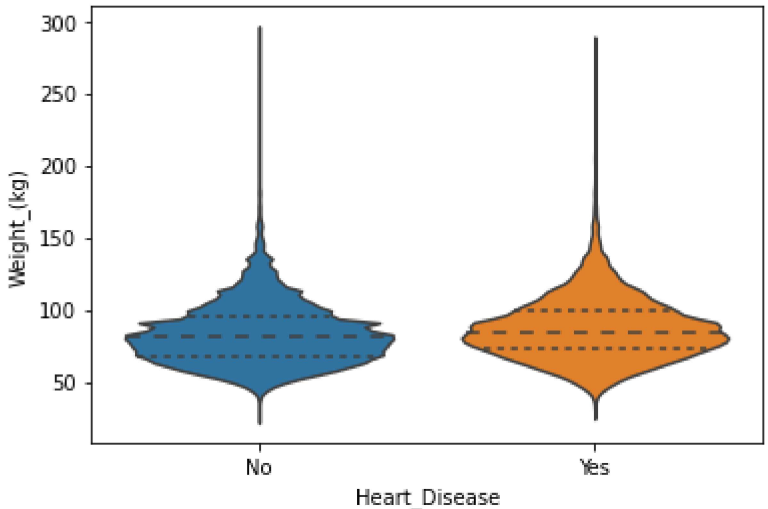


In [13]:

```
#Heart disease vs Weight
sns.violinplot( x=df["Heart_Disease"], y=df["Weight_(kg)"], split=True, gap=.1, inner=
```

Out[13]:

```
<AxesSubplot:xlabel='Heart_Disease', ylabel='Weight_(kg)'>
```



In [14]:
```
#!pip install statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from statsmodels.graphics.factorplots import interaction_plot
import matplotlib.pyplot as plt
from scipy import stats as ss
```

In [16]:
```
ss.ks_2samp(df['Heart_Disease'], df['Sex']) #determine where there is a significant di
```

Out[16]:
```
KstestResult(statistic=1.0, pvalue=0.0)
```

In [17]:
```
# Categorize age based on the specified ranges
def categorize_age(Age_Category):
    if Age_Category in ['18-24']:
        return 'Young'
    if Age_Category in ['25-29', '30-34', '35-39']:
        return 'Adult'
    elif Age_Category in ['40-44', '45-49', '50-54']:
        return 'Mid-Aged'
    elif Age_Category in ['55-59', '60-64','65-69']:
        return 'Senior-Adult'
    elif Age_Category in ['70-74', '75-79','80+']:
        return 'Elderly'
# Apply the function to the Age_Category column
df['Age_Group'] = df['Age_Category'].apply(categorize_age)

# Create a mapping from Age_Group to its combined string of name and range
age_mapping = {
    'Young': 'Young (18-24)',
    'Adult': 'Adult (25-39)',
    'Mid-Aged': 'Mid-Aged (40-54)',
    'Senior-Adult': 'Senior-Adult (55-69)',
    'Elderly': 'Elderly (70+)'
}

# Map the Age_Group to its detailed label
df['Age_Label'] = df['Age_Group'].map(age_mapping)

# Create a box plot visualizing BMI across age details and colored by heart disease st
fig = px.box(df,
             x="Age_Label",
```

```
             y="BMI",
             color="Heart_Disease",
             title="BMI Distribution across Age Groups based on Heart Disease Status",
             category_orders={"Age_Label": list(age_mapping.values())})

# Update layout to make it more readable
fig.update_layout(xaxis_title="Age Group",
                  yaxis_title="BMI",
                  legend_title="Heart Disease")

# Display the plot
fig.show()
```

In [ ]:

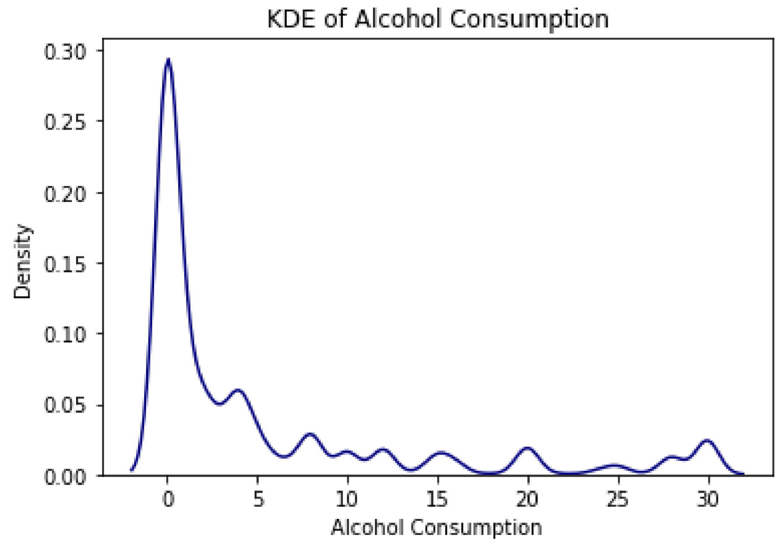
```

```

In [ ]:

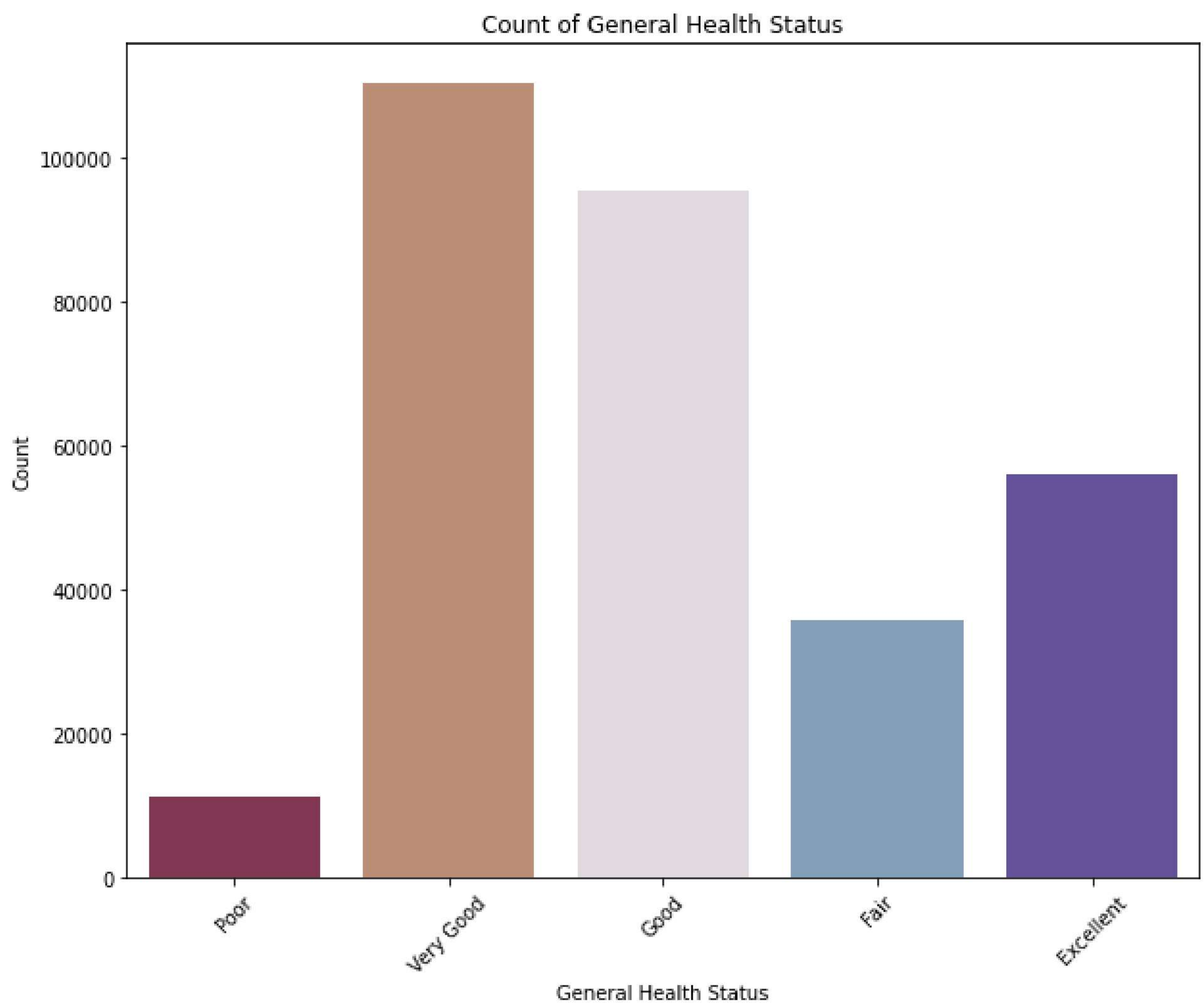
```

```

In [18]:

```
sns.kdeplot(df['Alcohol_Consumption'], color="navy")
plt.title('KDE of Alcohol Consumption')
plt.xlabel('Alcohol Consumption')
plt.ylabel('Density')
plt.show()
```



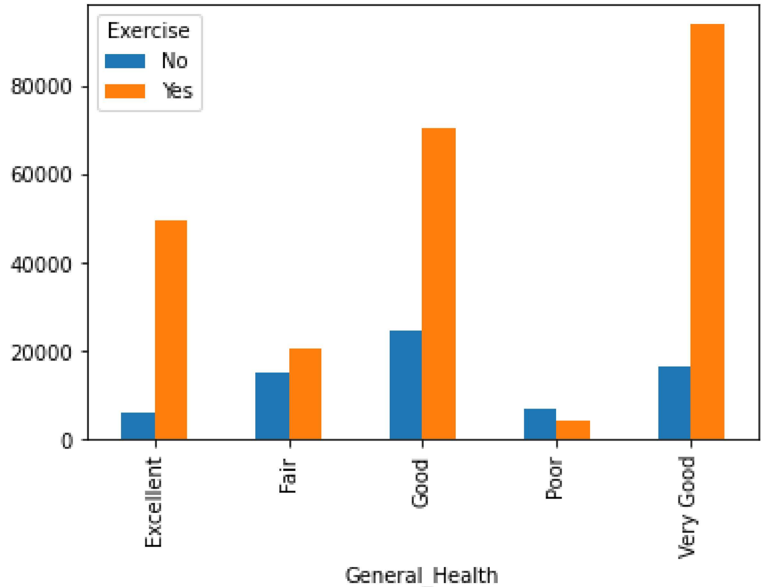
In [19]:
```
# counts for General Health Status
plt.figure(figsize=(10, 8))
sns.countplot(data=df, x='General_Health', palette = 'twilight_shifted_r')
plt.title('Count of General Health Status')
plt.xlabel('General Health Status')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



In [20]:
```python
#General Health vs Exercise
plt.figure(figsize=(10, 8))
df1= df.groupby(['General_Health', 'Exercise']).size()
df1 = df1.unstack()
df1.plot(kind='bar')
```

Out[20]: <AxesSubplot:xlabel='General_Health'>

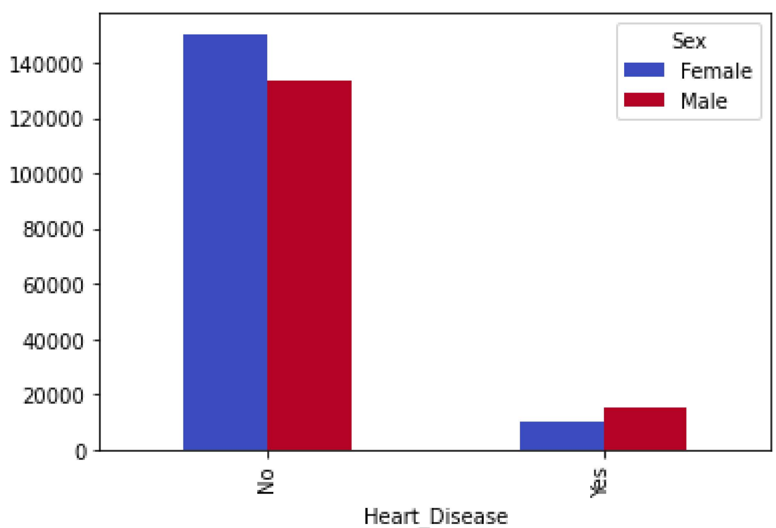
<Figure size 720x576 with 0 Axes>



In [21]:
```python
# Heart Disease vs Sex
plt.figure(figsize=(10, 8))
df2= df.groupby(['Heart_Disease', 'Sex']).size().unstack()
df2.plot(kind='bar', colormap = 'coolwarm')
```

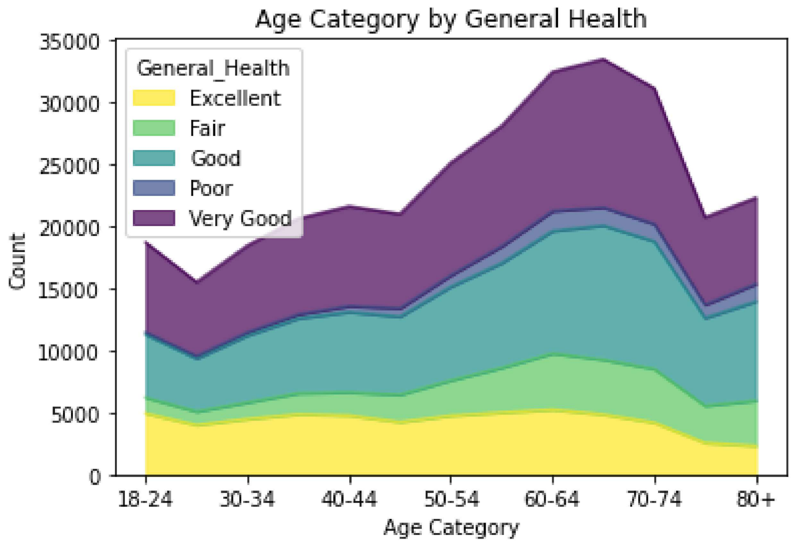
Out[21]: <AxesSubplot:xlabel='Heart_Disease'>

<Figure size 720x576 with 0 Axes>



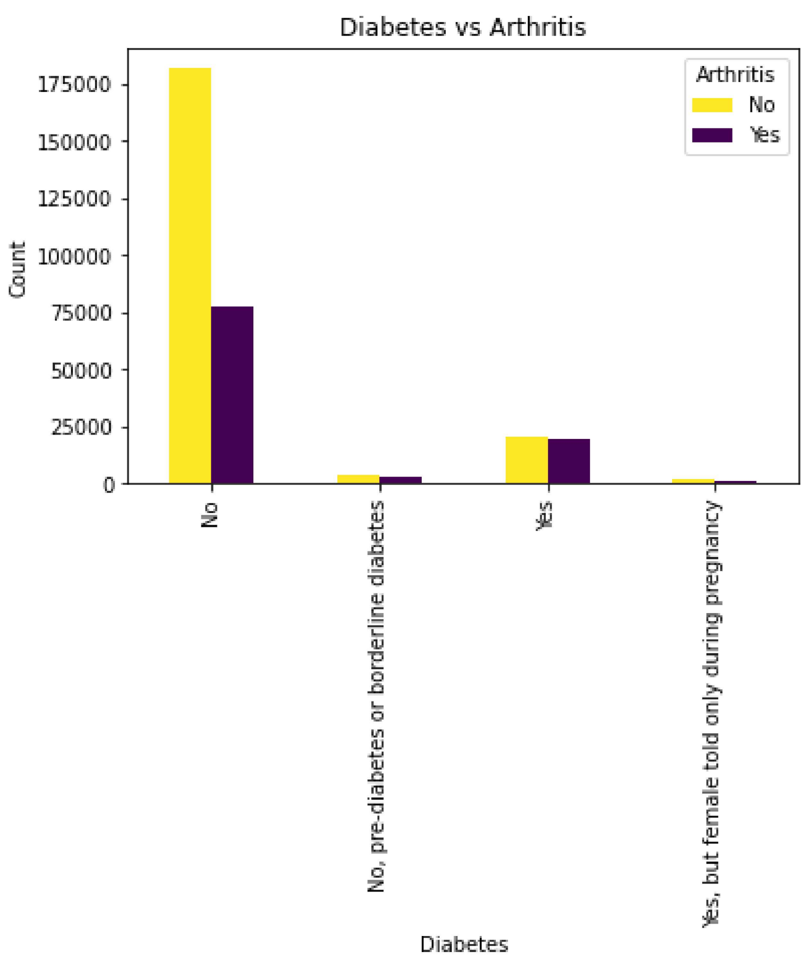
In [22]:
```python
#comparing general health and Age category

crosstab = pd.crosstab(df['Age_Category'], df['General_Health'])
crosstab.plot(kind='area', colormap='viridis_r', alpha=0.7, stacked=True)
plt.title( 'Age Category by General Health')
```

```
plt.xlabel('Age Category')
plt.ylabel('Count')
plt.show()
```



In [23]:

```
#Diabetes and Arthritis
crosstab1 = pd.crosstab(df['Diabetes'], df['Arthritis'])
crosstab1.plot(kind='bar', colormap='viridis_r')
plt.title( 'Diabetes vs Arthritis')
plt.xlabel('Diabetes')
plt.ylabel('Count')
plt.show()
```



In [24]:
```python
#Perform a correlation matrix to visulise the data
correlation_matrix = df.corr()
correlation_matrix

#need to encode the data

#plt.figure(figsize=(9,9))
#sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='Blues')
#plt.show()
```

Out[24]:

	Height_(cm)	Weight_(kg)	BMI	Alcohol_Consumption	Fruit_Cons
Height_(cm)	1.000000	0.472175	-0.027413	0.128850	-
Weight_(kg)	0.472175	1.000000	0.859702	-0.032427	-
BMI	-0.027413	0.859702	1.000000	-0.108750	-
Alcohol_Consumption	0.128850	-0.032427	-0.108750	1.000000	-
Fruit_Consumption	-0.045925	-0.090611	-0.076603	-0.012542	
Green_Vegetables_Consumption	-0.030153	-0.075895	-0.070629	0.060088	
FriedPotato_Consumption	0.108790	0.096327	0.048343	0.020503	-

In [25]:
```python
# Create a copy of the DataFrame to avoid modifying the original
df_encoded = df.copy()

# Create a label encoder object
label_encoder = LabelEncoder()

# Iterate through each object column and encode its values
for column in df_encoded.select_dtypes(include='object'):
    df_encoded[column] = label_encoder.fit_transform(df_encoded[column])

# Now, df_encoded contains the label-encoded categorical columns
df_encoded.head()
```

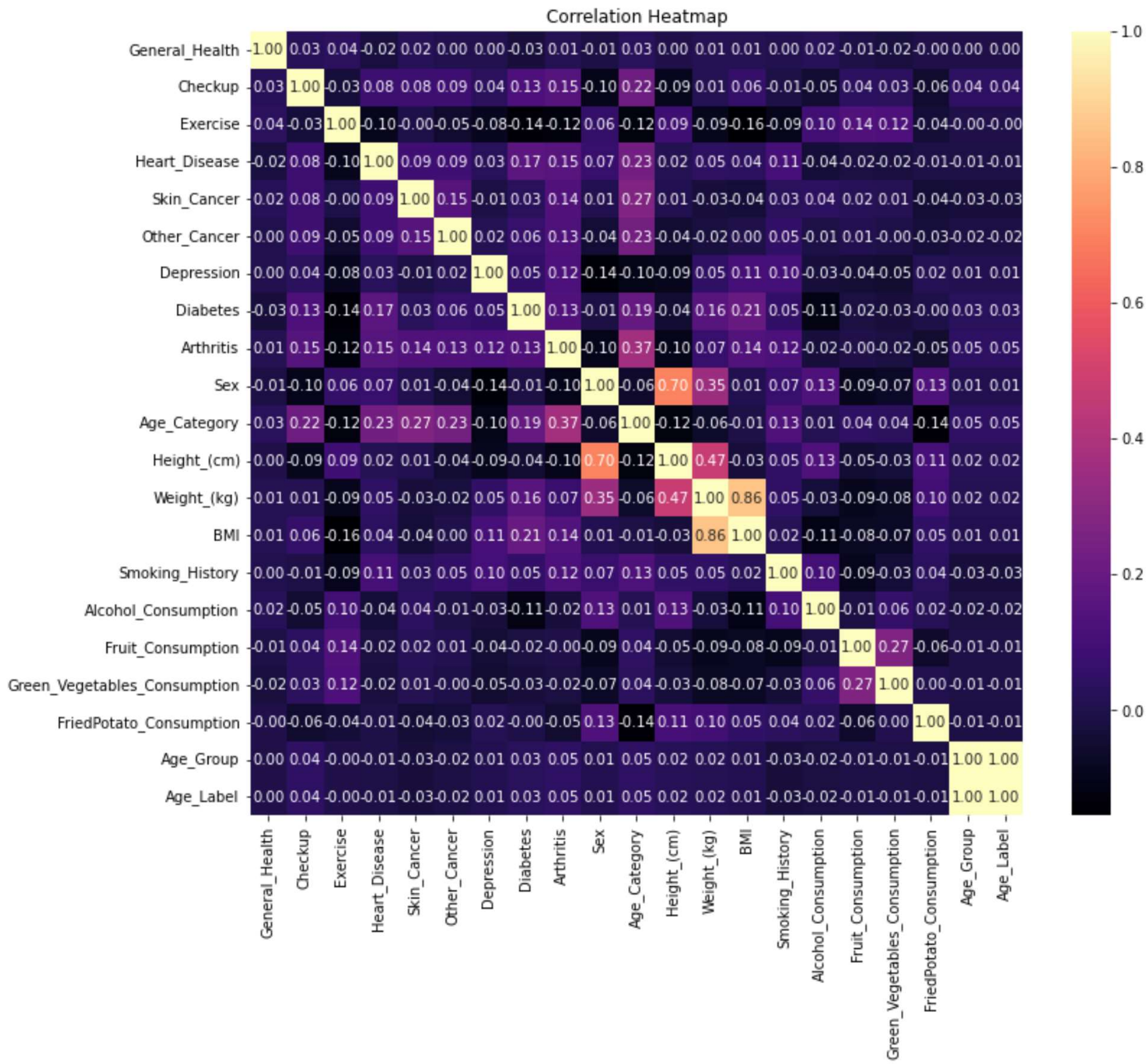
Out[25]:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diabete
0	3	2	0	0	0	0	0	0
1	4	4	0	1	0	0	0	2
2	4	4	1	0	0	0	0	2
3	3	4	1	1	0	0	0	2
4	2	4	0	0	0	0	0	0

5 rows × 21 columns

In [26]:
```python
correlation_matrix1 = df_encoded.corr()

# Create a heatmap
plt.figure(figsize=(12, 10))
```

```
sns.heatmap(correlation_matrix1, annot=True, cmap='magma', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



checking for differences in the target class distribution yes vs no

In [27]:
```
df_encoded['Heart_Disease'].value_counts()
```

Out[27]:
```
0    283803
1     24971
Name: Heart_Disease, dtype: int64
```

In [28]:
```
X = df_encoded.drop("Heart_Disease", axis = 1)
y = df_encoded['Heart_Disease']
```

In [29]:
```
#using the randomundersampler to balance the target
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(sampling_strategy={0:24971})
X_rus, y_rus = rus.fit_resample(X, y)
```

```
print(y.value_counts())

print(y_rus.value_counts())
```

```
0    283803
1     24971
Name: Heart_Disease, dtype: int64
0    24971
1    24971
Name: Heart_Disease, dtype: int64
```

In [30]:

```
# Step 1: Define features and target variable
X = df_encoded.drop("Heart_Disease", axis=1)  # Features (all columns except 'Heart_Di
y = df_encoded["Heart_Disease"]  # Target variable

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_rus, y_rus, test_size=0.2, randc
```

In [ ]:

Linear Regression

In [34]:

```
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
print(f"Linear Regression Mean Squared Error: {mse:.2f}")
print(f"Linear Regression Mean Absolute Error: {mae:.2f}")
coef = model.coef_
print(coef)
```

```
Linear Regression Mean Squared Error: 0.18
Linear Regression Mean Absolute Error: 0.37
[-1.09327862e-02  2.63233574e-02 -5.94127013e-02  1.88667195e-02
  5.84017106e-02  8.69843136e-02  7.73129957e-02  9.23756181e-02
  1.69683194e-01  5.15747279e-02 -2.15760900e-03  4.57702645e-04
  9.73845086e-07  1.02232212e-01 -3.10025725e-03 -4.58127284e-05
 -3.06039984e-04  4.29925623e-04  5.52578173e-04  5.52578173e-04]
```

In [75]:

```
# PCA

from sklearn.decomposition import PCA

pca = PCA(n_components=20)
pca.fit(X)
evr = pca.explained_variance_ratio_
[i*100 for i in evr]
```

Out[75]:

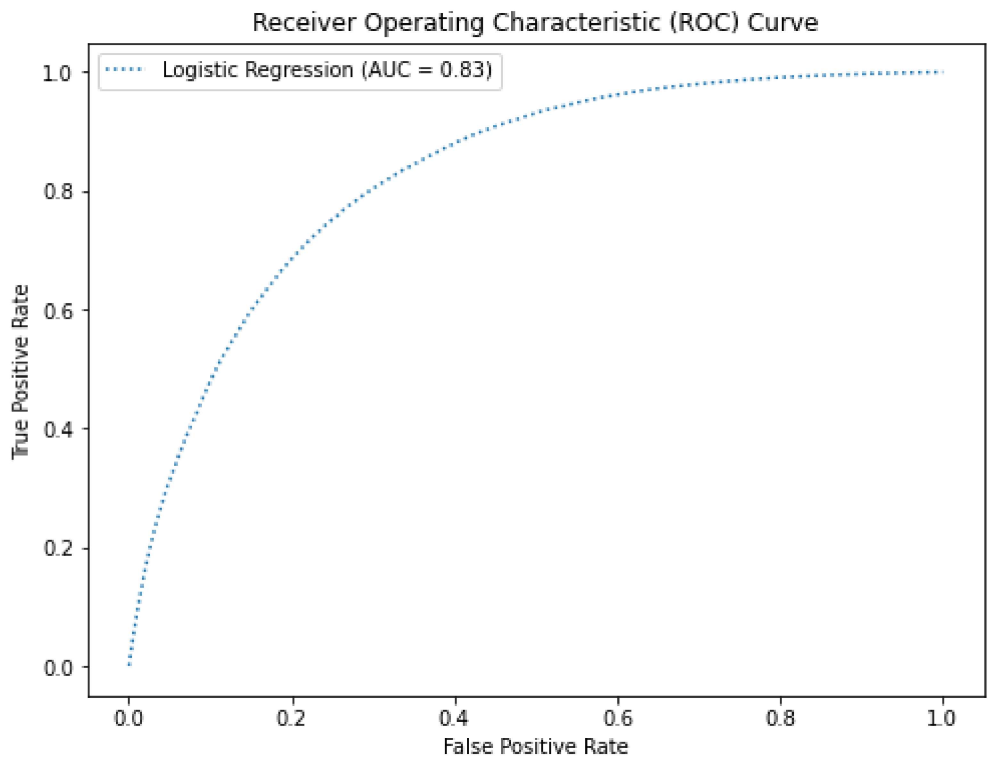
```
[41.145944153635114,
 30.64011979522403,
 12.318467979873672,
 6.369700042306446,
 4.44249079010112,
 3.823888228655301,
 0.7472729217141375,
 0.1806211753602524,
 0.13798067757468227,
 0.06049144685377348,
 0.041068919109884264,
 0.029288620665604143,
 0.015391436562210958,
 0.011697681929583046,
 0.00962065758382213,
 0.008763575119451217,
 0.007167957191756336,
 0.005461836466016722,
 0.004562104073156001,
 9.103845055198117e-33]
```

Logistic Regression

In [132…

```
model1 = LogisticRegression()
model1.fit(X_train, y_train)

# Make predictions on the test set
predictions = model1.predict(X_test)
proba = model1.predict_proba(X_test)[:,1]
# Calculate AUC
logistic_auc = roc_auc_score(y_test, proba)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, proba)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, predictions)
print(f"Logistic Regression Accuracy: {accuracy:.2f}")
print("Logistic Regression Classification Report:")
print(classification_report(y_test, predictions))

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='Logistic Regression (AUC = %0.2f)' % logistic
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
Logistic Regression Accuracy: 0.75
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.74      0.75     84921
           1       0.75      0.76      0.75     85361

    accuracy                           0.75    170282
   macro avg       0.75      0.75      0.75    170282
weighted avg       0.75      0.75      0.75    170282
```



In [35]:
```
model1.coef_
```

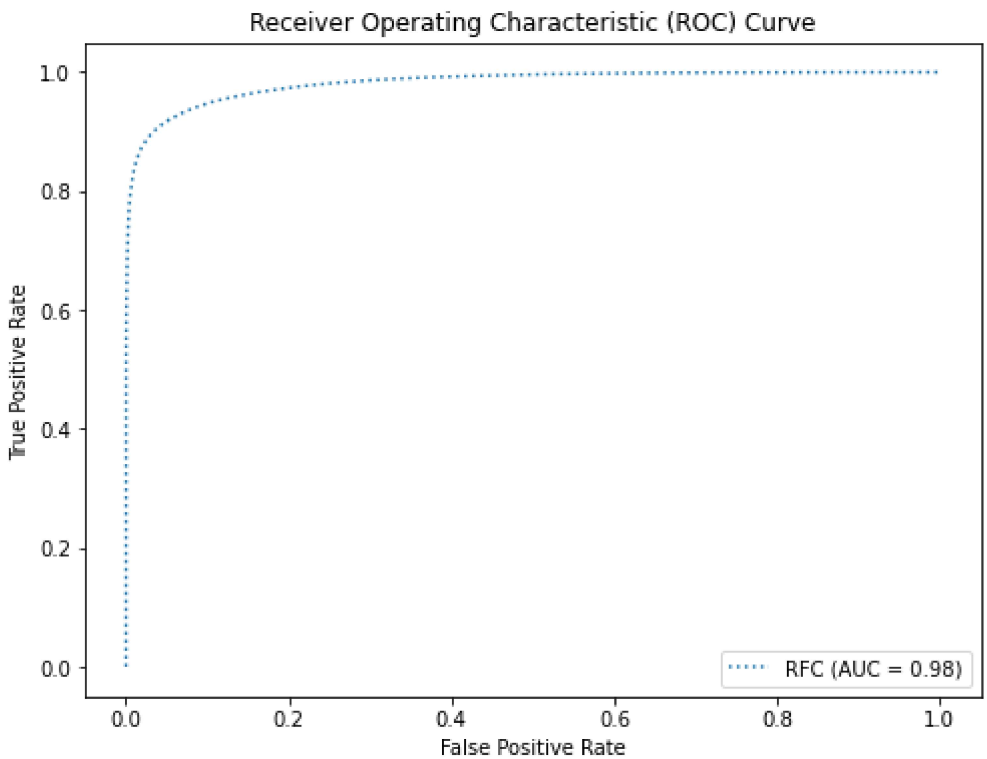
Out[35]:
```
array([[-0.06178045,  0.19635345, -0.23997771,  0.0585695 ,  0.14014567,
         0.30310519,  0.57445957,  0.35859651,  0.3573496 ,  0.28935773,
        -0.02319466,  0.03084421, -0.07771226,  0.48983031, -0.00900975,
        -0.00116978, -0.00082081,  0.001325  , -0.0010353 , -0.0010353 ]])
```

RandomForestClassifier

In [131…
```
model_rf = RandomForestClassifier(random_state=42)


model_rf.fit(X_train, y_train)

rf_pred = model_rf.predict(X_test)
rf_proba = model_rf.predict_proba(X_test)[:,1]

roc_auc = roc_auc_score(y_test, model_rf.predict(X_test))

print(roc_auc)

rf = model_rf.predict(X_test)
```

```python
# Calculate AUC
rf_auc = roc_auc_score(y_test, rf_proba)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, rf_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='RFC (AUC = %0.2f)' % rf_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
0.9320132200278138
```



In [ ]:

```python

```

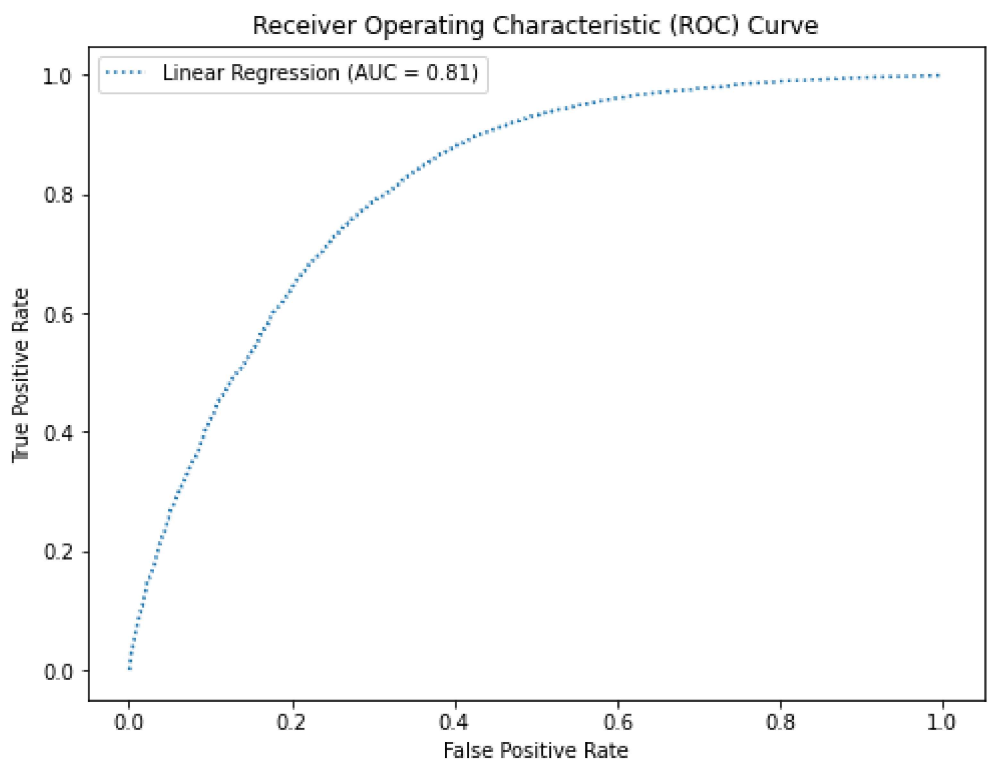
In [39]:

```python
from sklearn.neural_network import MLPClassifier

model_nn = MLPClassifier()
model_nn.fit(X_train, y_train)

y_pred = model_nn.predict(X_test)
y_pred_proba = model_nn.predict_proba(X_test)[:,1]

print (roc_auc_score(y_test, y_pred_proba))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
0.8138246542393266
              precision    recall  f1-score   support

           0       0.79      0.67      0.72      4989
           1       0.71      0.82      0.76      5000

    accuracy                           0.74      9989
   macro avg       0.75      0.74      0.74      9989
weighted avg       0.75      0.74      0.74      9989

[[3318 1671]
 [ 889 4111]]
```

In [40]:

```
# Make predictions on the test set
predictions = model_nn.predict(X_test)

# Calculate AUC
nn_auc = roc_auc_score(y_test, y_pred_proba)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='Linear Regression (AUC = %0.2f)' % nn_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



Decision Tree

In [59]:

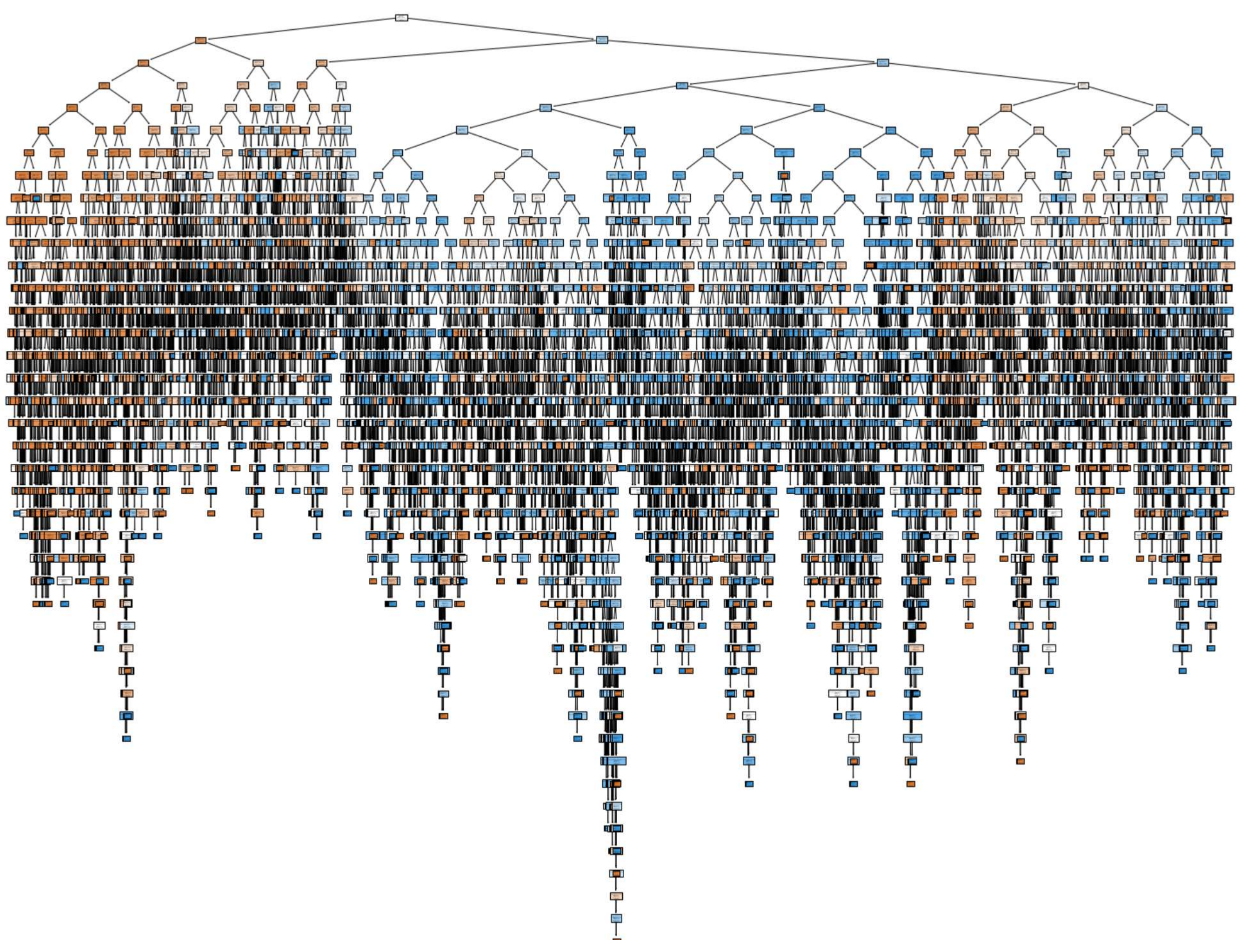
```
model_dt = DecisionTreeClassifier(random_state =42)
```

```
model_dt.fit(X_train, y_train)

y_pred = model_dt.predict(X_test)
y_pred_proba = model_dt.predict_proba(X_test)[:,1]

print(roc_auc_score(y_test, y_pred_proba))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
0.6567863299258369
              precision    recall  f1-score   support

           0       0.66      0.66      0.66      4989
           1       0.66      0.66      0.66      5000

    accuracy                           0.66      9989
   macro avg       0.66      0.66      0.66      9989
weighted avg       0.66      0.66      0.66      9989

[[3268 1721]
 [1707 3293]]
```

In [129…

```
from sklearn.tree import plot_tree
fig = plt.figure(figsize=(25,20))
_ = plot_tree(model_dt,
                   feature_names=df.columns,
                   class_names=['0','1'],
                   filled=True)
```



In [ ]:

```

```

In [ ]:

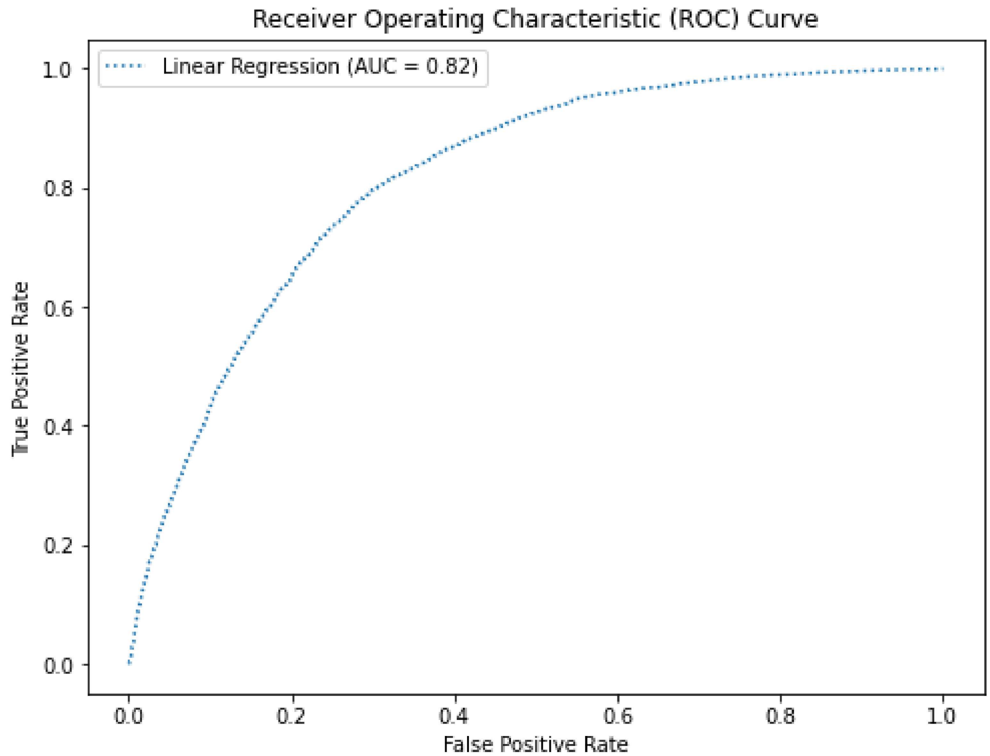
```

```

xgboost

In [47]:

```
import xgboost as xgb

model_xgb = xgb.XGBClassifier()
model_xgb.fit (X_train, y_train)


y_pred = model_xgb.predict(X_test)
y_pred_proba = model_xgb.predict_proba(X_test)[:,1]

print (roc_auc_score(y_test, y_pred_proba))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

predictions = model_xgb.predict(X_test)

# Calculate AUC
xgb_auc = roc_auc_score(y_test, y_pred_proba)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='Linear Regression (AUC = %0.2f)' % xgb_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
0.8160481860092204
              precision    recall  f1-score   support

           0       0.77      0.70      0.74      4989
           1       0.73      0.80      0.76      5000

    accuracy                           0.75      9989
   macro avg       0.75      0.75      0.75      9989
weighted avg       0.75      0.75      0.75      9989

[[3500 1489]
 [1024 3976]]
```



In [50]:
```
model_xgb.feature_importances_
```

Out[50]:
```
array([0.12992813, 0.02657313, 0.01794893, 0.02094171, 0.02627362,
       0.02574706, 0.09767201, 0.09923099, 0.10965686, 0.2356074 ,
       0.01574115, 0.01717101, 0.01772181, 0.0717163 , 0.02136608,
       0.01699585, 0.01604828, 0.01746664, 0.01619304, 0.        ],
      dtype=float32)
```

In [ ]:
```
model_xgb.feature_names_in_
```

Hyperparameter Turning

In [63]:
```
from scipy.stats import randint
from sklearn.model_selection import RandomizedSearchCV
```

In [69]:
```
model_ht = DecisionTreeClassifier()

parameters = {'class_weight':['balanced', None]
              , 'max_depth': randint(1,20)
              , 'min_samples_split': randint(2,11)
             }

rs = RandomizedSearchCV(model_ht, parameters, cv=8)
search = rs.fit(X_train, y_train)
```

In [70]:
```
search.best_params_
```

Out[70]:
```
{'class_weight': 'balanced', 'max_depth': 7, 'min_samples_split': 3}
```

In [71]:
```
search.cv_results_
```

Out[71]:
```
{'mean_fit_time': array([0.24538472, 0.17750892, 0.08599845, 0.02212727, 0.22524071,
        0.1797469 , 0.19337451, 0.12425086, 0.11075252, 0.15824923]),
 'std_fit_time': array([0.02589179, 0.0125098 , 0.00281916, 0.00078986, 0.0079667 ,
        0.0117652 , 0.00431645, 0.00534841, 0.00371983, 0.01567381]),
 'mean_score_time': array([0.00386548, 0.0028739 , 0.00212613, 0.0018779 , 0.0030024
1,
        0.0028761 , 0.0027521 , 0.00237539, 0.00288367, 0.00287592]),
 'std_score_time': array([0.00059608, 0.00032511, 0.00033012, 0.00033141, 0.00049991,
        0.00033092, 0.00082743, 0.00048724, 0.000604  , 0.00060087]),
 'param_class_weight': masked_array(data=['balanced', 'balanced', None, None, 'balanc
ed', None,
                    'balanced', None, 'balanced', 'balanced'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_max_depth': masked_array(data=[16, 11, 6, 1, 18, 13, 14, 9, 7, 9],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[3, 7, 7, 6, 3, 9, 6, 5, 3, 2],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'class_weight': 'balanced',
   'max_depth': 16,
   'min_samples_split': 3},
  {'class_weight': 'balanced', 'max_depth': 11, 'min_samples_split': 7},
  {'class_weight': None, 'max_depth': 6, 'min_samples_split': 7},
  {'class_weight': None, 'max_depth': 1, 'min_samples_split': 6},
  {'class_weight': 'balanced', 'max_depth': 18, 'min_samples_split': 3},
  {'class_weight': None, 'max_depth': 13, 'min_samples_split': 9},
  {'class_weight': 'balanced', 'max_depth': 14, 'min_samples_split': 6},
  {'class_weight': None, 'max_depth': 9, 'min_samples_split': 5},
  {'class_weight': 'balanced', 'max_depth': 7, 'min_samples_split': 3},
  {'class_weight': 'balanced', 'max_depth': 9, 'min_samples_split': 2}],
 'split0_test_score': array([0.7035035 , 0.73653654, 0.75255255, 0.68308308, 0.694894
89,
        0.72192192, 0.71971972, 0.75295295, 0.76076076, 0.75375375]),
 'split1_test_score': array([0.69763716, 0.72887465, 0.73848618, 0.68602323, 0.684821
79,
        0.71766119, 0.71405687, 0.73928714, 0.74289147, 0.73928714]),
 'split2_test_score': array([0.70424509, 0.74229075, 0.75110132, 0.6824189 , 0.682619
14,
        0.73528234, 0.71826191, 0.75570685, 0.75670805, 0.75470565]),
 'split3_test_score': array([0.69823789, 0.73428114, 0.7505006 , 0.69223068, 0.690228
27,
        0.71786143, 0.71465759, 0.74889868, 0.75270324, 0.74989988]),
 'split4_test_score': array([0.68702443, 0.73548258, 0.74128955, 0.67561073, 0.674209
05,
        0.7112535 , 0.70344413, 0.74229075, 0.74369243, 0.74249099]),
 'split5_test_score': array([0.69743692, 0.72807369, 0.74309171, 0.6784141 , 0.680016
02,
        0.71005206, 0.70544654, 0.73748498, 0.74209051, 0.73688426]),
 'split6_test_score': array([0.70224269, 0.72527032, 0.73848618, 0.67961554, 0.683219
86,
        0.71425711, 0.70744894, 0.74309171, 0.74449339, 0.74349219]),
 'split7_test_score': array([0.69743692, 0.73007609, 0.7484982 , 0.67881458, 0.680216
```

```
26,
        0.71585903, 0.7112535 , 0.74329195, 0.75190228, 0.74209051]),
 'mean_test_score': array([0.69847058, 0.73261072, 0.74550079, 0.68202635, 0.6837781
6,
        0.71801857, 0.71178615, 0.74537563, 0.74940527, 0.74532555]),
 'std_test_score': array([0.00508434, 0.00518413, 0.00544316, 0.00489528, 0.00598267,
        0.00742858, 0.00555825, 0.00607215, 0.00663408, 0.00622017]),
 'rank_test_score': array([ 8,  5,  2, 10,  9,  6,  7,  3,  1,  4])}
```

In [72]:

```
search.best_score_
```

Out[72]:

```
0.7494052673017431
```

In [ ]:

In [51]:

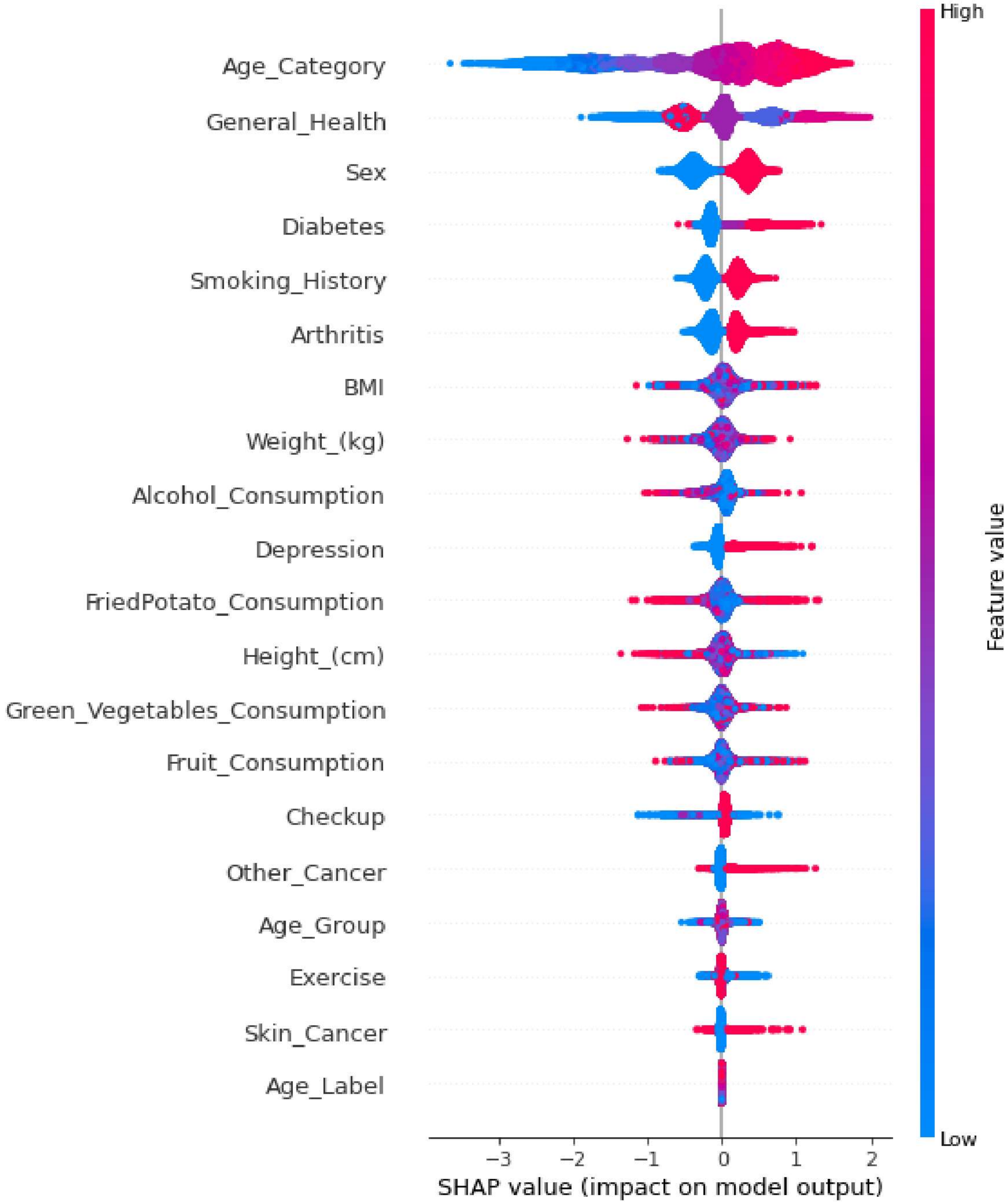
```
import shap
```

In [52]:

```
shap.initjs()
```

In [57]:

```
explainer = shap.TreeExplainer(model_xgb)
shap_values = explainer(X_train)
shap.summary_plot(shap_values, X_train)
```



In [ ]:
```
# rather than  use the randomUnder sampler we will use fit_resample
```

In [77]:
```
smote = SMOTE(random_state = 42)
X_balanced, y_balanced = smote.fit_resample(X, y)

print(y.value_counts())

print(y_balanced.value_counts())
```

```
0    283803
1     24971
Name: Heart_Disease, dtype: int64
0    283803
1    283803
Name: Heart_Disease, dtype: int64
```

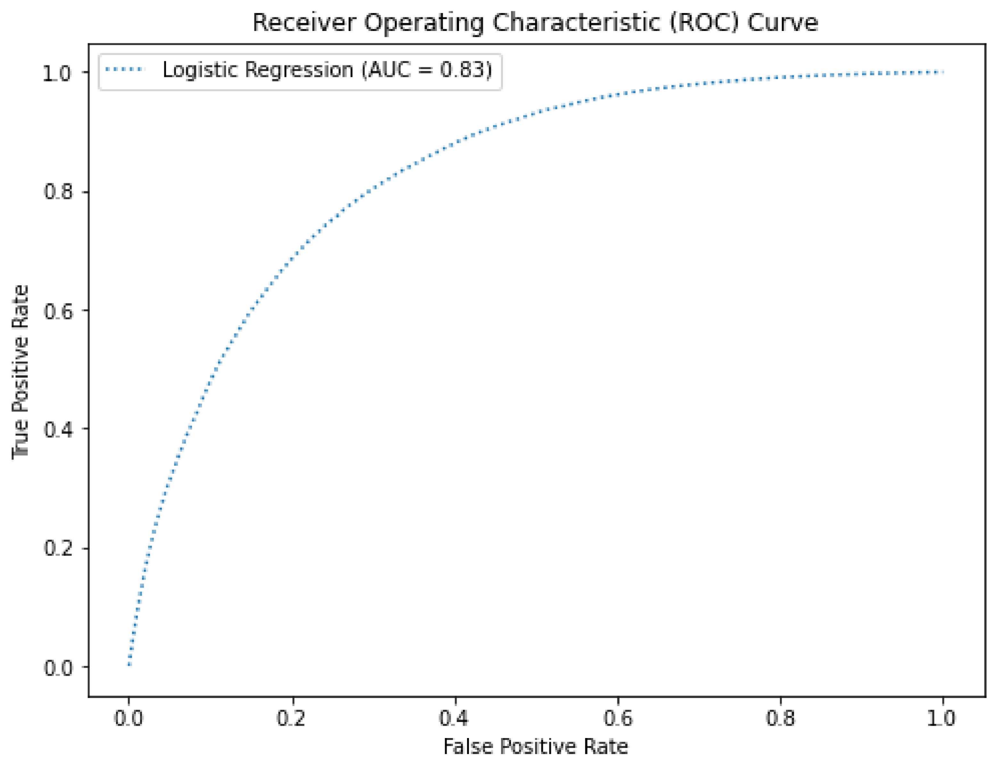
In [85]:
```
#training data and testing data set split
X = df_encoded.drop("Heart_Disease", axis=1)  # Features (all columns except 'Heart_Di
y = df_encoded["Heart_Disease"]  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=
```

In [86]:
```
models = LinearRegression()
models.fit(X_train, y_train)

# Make predictions on the test set
predictions = models.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
print(f"Linear Regression Mean Squared Error: {mse:.2f}")
print(f"Linear Regression Mean Absolute Error: {mae:.2f}")
coef = model.coef_
print(coef)
```

```
Linear Regression Mean Squared Error: 0.17
Linear Regression Mean Absolute Error: 0.35
[-1.09327862e-02  2.63233574e-02 -5.94127013e-02  1.88667195e-02
  5.84017106e-02  8.69843136e-02  7.73129957e-02  9.23756181e-02
  1.69683194e-01  5.15747279e-02 -2.15760900e-03  4.57702645e-04
  9.73845086e-07  1.02232212e-01 -3.10025725e-03 -4.58127284e-05
 -3.06039984e-04  4.29925623e-04  5.52578173e-04  5.52578173e-04]
```

In [133…
```
model1s = LogisticRegression()
model1s.fit(X_train, y_train)

# Make predictions on the test set
predictions = model1s.predict(X_test)
proba1s = model1s.predict_proba(X_test)[:,1]
# Calculate AUC
logistic_aucs = roc_auc_score(y_test, proba1s)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, proba1s)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, predictions)
print(f"Logistic Regression Accuracy: {accuracy:.2f}")
print("Logistic Regression Classification Report:")
print(classification_report(y_test, predictions))

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='Logistic Regression (AUC = %0.2f)' % logistic
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

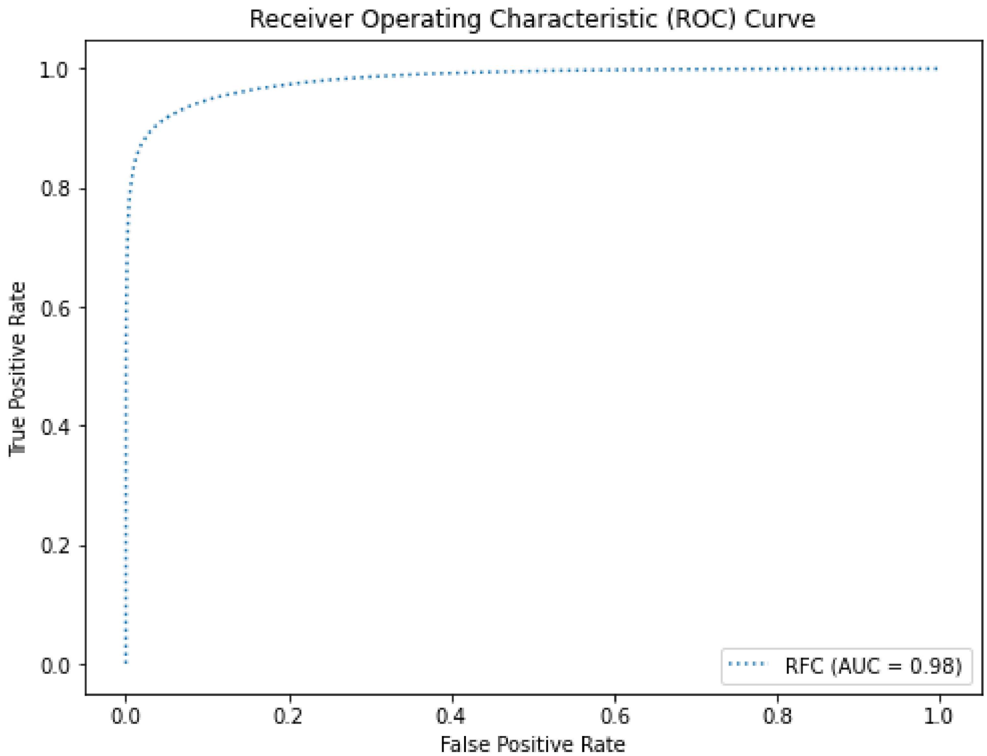
```
Logistic Regression Accuracy: 0.75
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.74      0.75     84921
           1       0.75      0.76      0.75     85361

    accuracy                           0.75    170282
   macro avg       0.75      0.75      0.75    170282
weighted avg       0.75      0.75      0.75    170282
```



In [135…

```python
model_rfs = RandomForestClassifier(random_state=42)


model_rfs.fit(X_train, y_train)


print(roc_auc)

rfs = model_rfs.predict(X_test)
rfs_proba = model_rfs.predict_proba(X_test)[:,1]
# Calculate AUC
rf_aucs = roc_auc_score(y_test, rfs_proba)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, rfs_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='RFC (AUC = %0.2f)' % rf_aucs)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
0.9320132200278138
```



In [98]:
```
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=
scaler_d = StandardScaler()
X_train_scaled = scaler_d.fit_transform(X_train)
X_test_scaled = scaler_d.transform(X_test)
```

In [112…
```
lr = LogisticRegression()
ra = RandomForestClassifier()
```

In [113…
```
lrm = lr.fit(X_train_scaled, y_train)
rfc = ra.fit(X_train_scaled, y_train)
```

In [114…
```
lrd = lr.predict(X_test_scaled)
rfd = ra.predict(X_test_scaled)
```

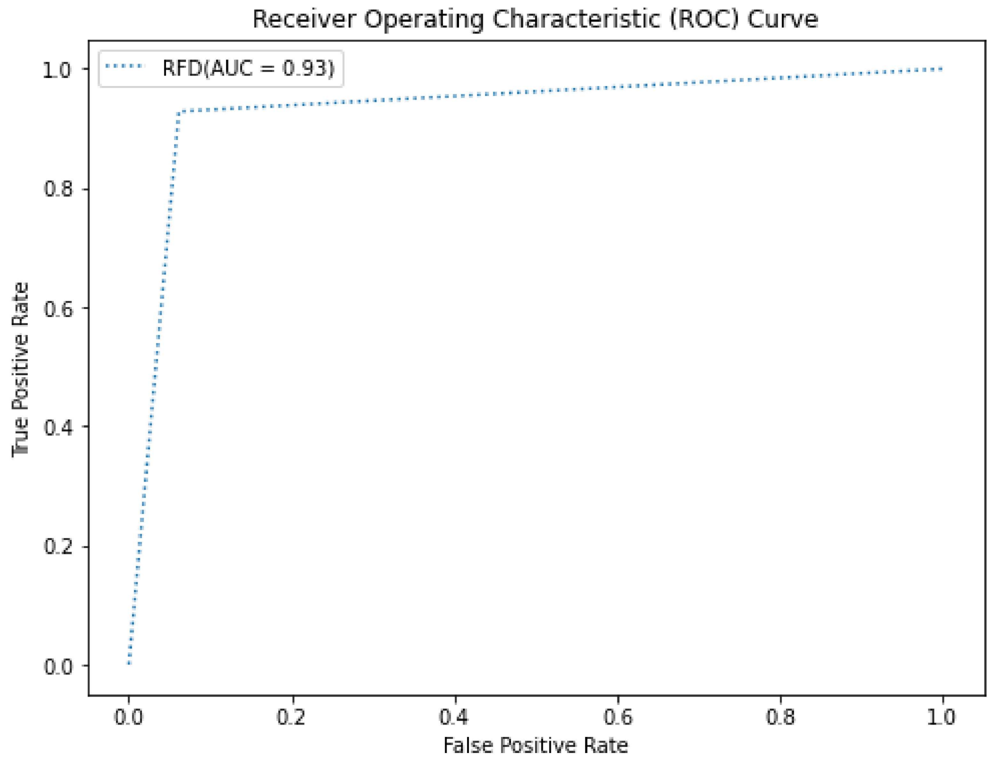
In [116…
```
lrd_report = classification_report(y_test, lrd)
rfd_report = classification_report(y_test, rfd)

print("="*40, "Logistic regression report:", "="*45, '\n')
print(lrd_report)


print("="*40, "Random forest report:", "="*45, '\n')
print(rfd_report)
```

```
======================================== Logistic regression report: ================
=============================

              precision    recall  f1-score   support

           0       0.76      0.74      0.75     84921
           1       0.75      0.76      0.76     85361

    accuracy                           0.75    170282
   macro avg       0.75      0.75      0.75    170282
weighted avg       0.75      0.75      0.75    170282

======================================== Random forest report: ======================
=======================

              precision    recall  f1-score   support

           0       0.93      0.94      0.93     84921
           1       0.94      0.93      0.93     85361

    accuracy                           0.93    170282
   macro avg       0.93      0.93      0.93    170282
weighted avg       0.93      0.93      0.93    170282
```

In [121…

```
ra = RandomForestClassifier()
rfc = ra.fit(X_train_scaled, y_train)

rfd = ra.predict(X_test_scaled)

roc_auc = roc_auc_score(y_test, rfd)
print(roc_auc)



# Calculate AUC
rfc_aucs = roc_auc_score(y_test, rfd)

# Generate ROC curve
fpr, tpr, _ = roc_curve(y_test, rfd)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linestyle=':', label='RFD(AUC = %0.2f)' % rfc_aucs)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
0.9334178002036487
```


Receiver Operating Characteristic (ROC) Curve
RFD(AUC = 0.93)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]:

In [ ]: