In [11]:
```python
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.feature_selection import SelectKBest, f_classif
```

In [13]:
```python
#Load Dataset
df = pd.read_csv('heart_disease.csv')
df.head()
```

Out[13]:

	Age	Gender	Blood Pressure	Cholesterol Level	Exercise Habits	Smoking	Family Heart Disease	Diabetes	BMI
0	56.0	Male	153.0	155.0	High	Yes	Yes	No	24.991591
1	69.0	Female	146.0	286.0	High	No	Yes	Yes	25.221799
2	46.0	Male	126.0	216.0	Low	No	No	No	29.855447
3	32.0	Female	122.0	293.0	High	Yes	Yes	No	24.130477
4	60.0	Male	166.0	242.0	Low	Yes	Yes	Yes	20.486289

5 rows × 21 columns

In [15]:
```python
# List of categorical columns to encode (already confirmed as int types)
categorical_cols = [
    'Gender', 'Exercise Habits', 'Smoking', 'Family Heart Disease',
    'Diabetes', 'High Blood Pressure', 'Low HDL Cholesterol',
    'High LDL Cholesterol', 'Alcohol Consumption', 'Stress Level',
    'Sugar Consumption', 'Heart Disease Status'
]

# Initialize LabelEncoder
le = LabelEncoder()

# Apply LabelEncoder to each categorical column
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

# Check data types and confirm no missing values
print("\n🧾 Data types after encoding:\n")
print(df.info())
```

```
print("\n❓ Missing values after encoding:\n")
print(df.isnull().sum())
```

🧾 Data types after encoding:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Age                   9971 non-null   float64
 1   Gender                10000 non-null  int32  
 2   Blood Pressure        9981 non-null   float64
 3   Cholesterol Level     9970 non-null   float64
 4   Exercise Habits       10000 non-null  int32  
 5   Smoking               10000 non-null  int32  
 6   Family Heart Disease  10000 non-null  int32  
 7   Diabetes              10000 non-null  int32  
 8   BMI                   9978 non-null   float64
 9   High Blood Pressure   10000 non-null  int32  
 10  Low HDL Cholesterol   10000 non-null  int32  
 11  High LDL Cholesterol  10000 non-null  int32  
 12  Alcohol Consumption   10000 non-null  int32  
 13  Stress Level          10000 non-null  int32  
 14  Sleep Hours           9975 non-null   float64
 15  Sugar Consumption     10000 non-null  int32  
 16  Triglyceride Level    9974 non-null   float64
 17  Fasting Blood Sugar   9978 non-null   float64
 18  CRP Level             9974 non-null   float64
 19  Homocysteine Level    9980 non-null   float64
 20  Heart Disease Status  10000 non-null  int32  
dtypes: float64(9), int32(12)
memory usage: 1.1 MB
None
```

❓ Missing values after encoding:

```
Age                     29
Gender                   0
Blood Pressure          19
Cholesterol Level       30
Exercise Habits          0
Smoking                  0
Family Heart Disease     0
Diabetes                 0
BMI                     22
High Blood Pressure      0
Low HDL Cholesterol      0
High LDL Cholesterol     0
Alcohol Consumption      0
Stress Level             0
Sleep Hours             25
Sugar Consumption        0
Triglyceride Level      26
Fasting Blood Sugar     22
CRP Level               26
Homocysteine Level      20
Heart Disease Status     0
dtype: int64
```

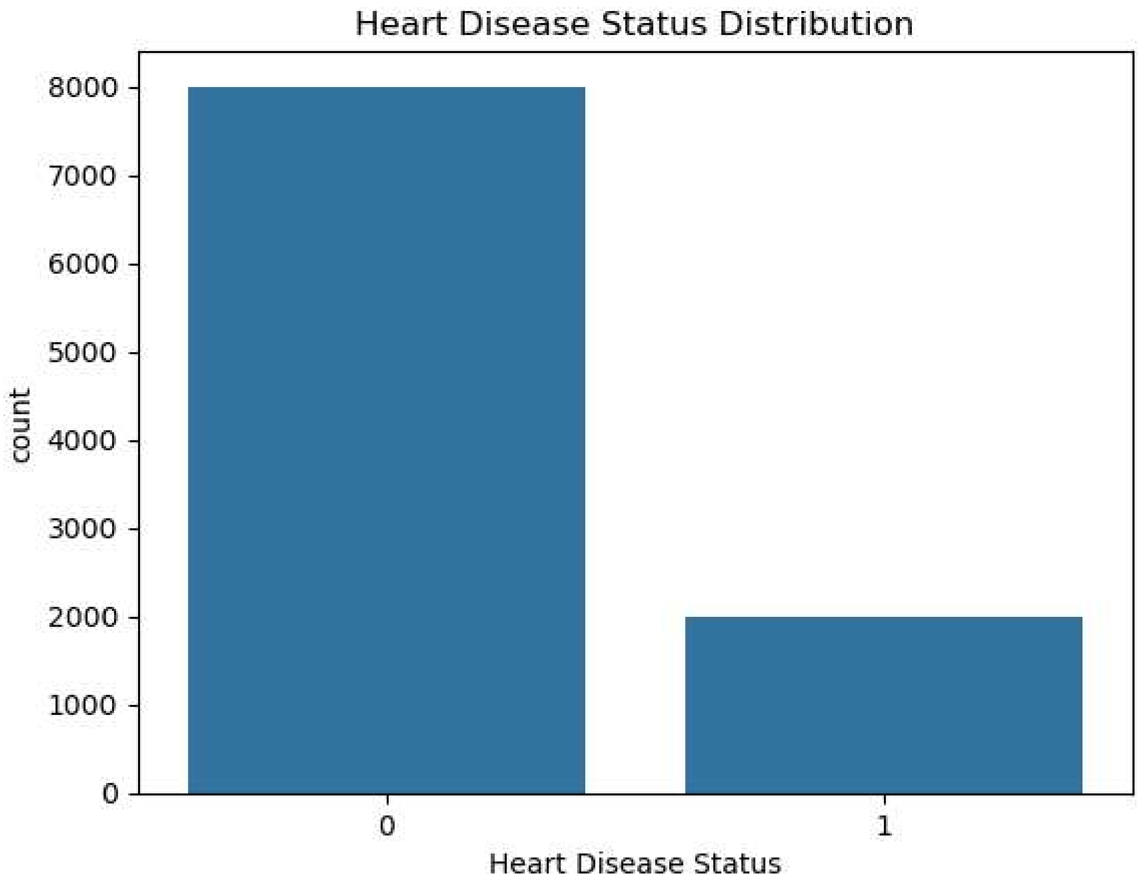
In [19]:

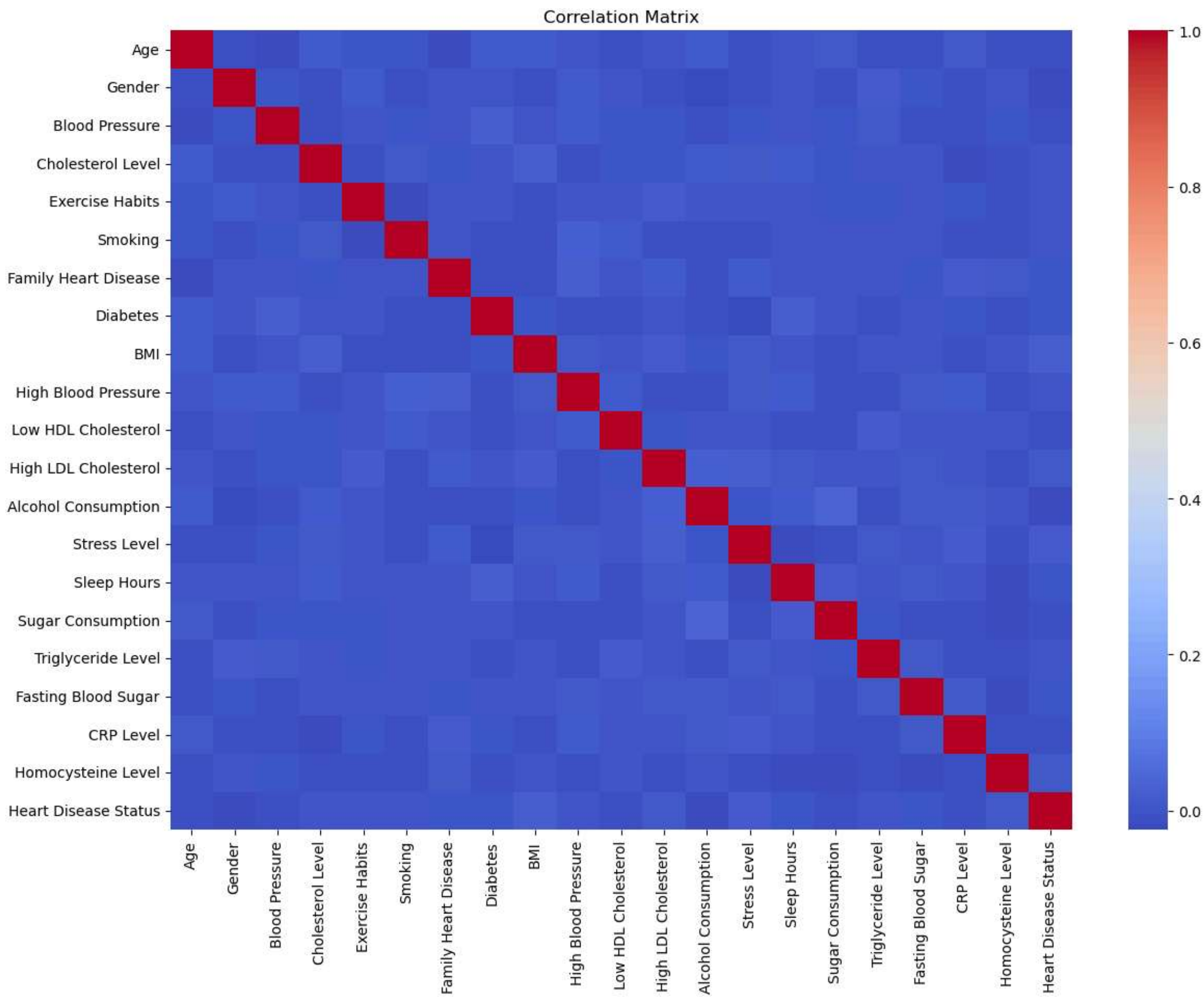
```
# Identify numeric columns (float64)
numeric_cols = df.select_dtypes(include=['float64']).columns

# Initialize imputer
imputer = SimpleImputer(strategy='mean')

# Apply imputer to only numeric columns
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

# Final check to confirm no missing values remain
print("\n✅ Missing values after imputation:\n")
print(df.isnull().sum().sum())
```

```
✅ Missing values after imputation:

0
```

In [7]:

```
# Exploratory Data Analysis

# Target variable distribution
sns.countplot(x='Heart Disease Status', data=df)
plt.title("Heart Disease Status Distribution")
plt.show()

# Correlation heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(df.corr(), cmap='coolwarm', annot=False)
plt.title("Correlation Matrix")
plt.show()
```


Heart Disease Status Distribution
8000
7000
6000
5000
4000
3000
2000
1000
0
count
0
1
Heart Disease Status



In [21]:
```
# Feature Selection

X = df.drop('Heart Disease Status', axis=1)
y = df['Heart Disease Status']

# Use SelectKBest to choose top 10 features
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features.tolist())

X = df[selected_features]
```

```
Selected Features: ['Age', 'Gender', 'Blood Pressure', 'BMI', 'Low HDL Cholesterol',
'High LDL Cholesterol', 'Alcohol Consumption', 'Stress Level', 'Sugar Consumption',
'Homocysteine Level']
```

In [23]:
```
# 6. Train-Test Split and Scaling

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [25]:
```python
# 7. Model Training

model = RandomForestClassifier(random_state=42)
model.fit(X_train_scaled, y_train)
```

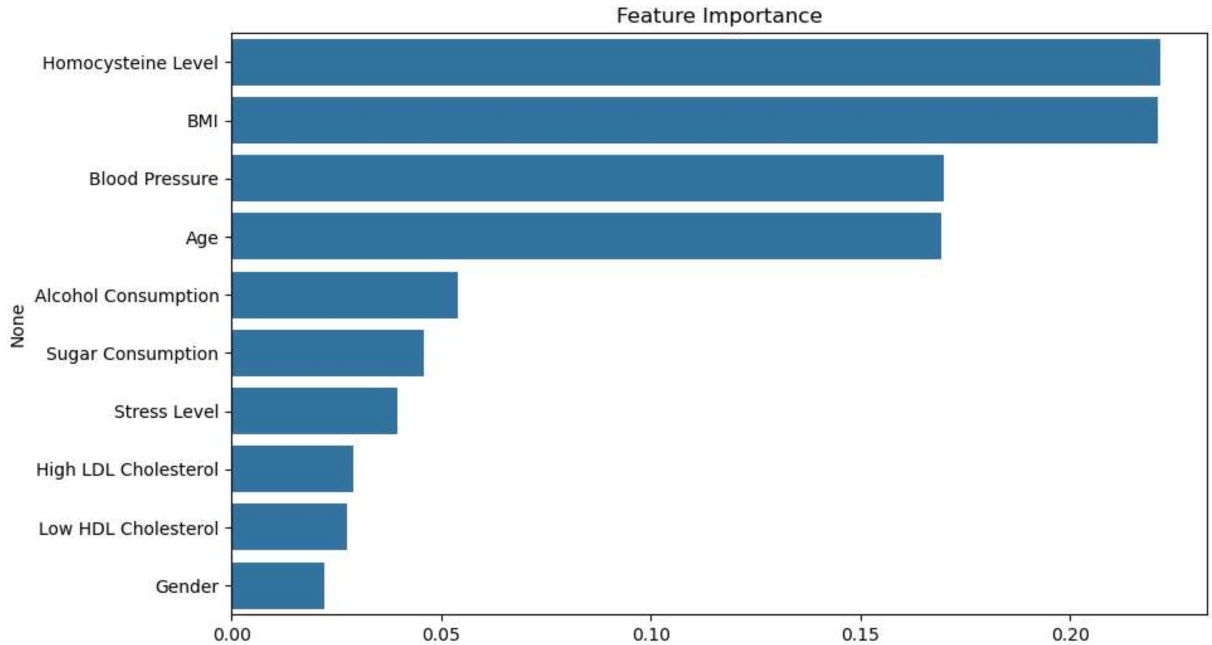
Out[25]:

```
RandomForestClassifier

RandomForestClassifier(random_state=42)
```

In [27]:
```python
# 8. Model Evaluation

y_pred = model.predict(X_test_scaled)

print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.805

Classification Report:
               precision    recall  f1-score   support

           0       0.81      1.00      0.89      1613
           1       0.00      0.00      0.00       387

    accuracy                           0.81      2000
   macro avg       0.40      0.50      0.45      2000
weighted avg       0.65      0.81      0.72      2000

Confusion Matrix:
 [[1610    3]
 [ 387    0]]
```

In [29]:
```python
# 9. Feature Importance

importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=X.columns[indices])
plt.title("Feature Importance")
plt.show()
```



Feature Importance
Homocysteine Level
BMI
Blood Pressure
Age
Alcohol Consumption
Sugar Consumption
Stress Level
High LDL Cholesterol
Low HDL Cholesterol
Gender
None
0.00
0.05
0.10
0.15
0.20