In [1]:
```python
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
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE

import warnings

# Filter warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
data = pd.read_csv(r"C:\Users\Tanmayee\OneDrive\Documents\Personal\Inifinte Solutions\ID 3399\Data
```

In [3]:
```python
data
```

Out[3]:

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	.
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	.
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	.
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	.
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	.
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	.
...	...	...	...	...	...	...	...	...	...	...	.
253675	0.0	1.0	1.0	1.0	45.0	0.0	0.0	0.0	0.0	1.0	.
253676	2.0	1.0	1.0	1.0	18.0	0.0	0.0	0.0	0.0	0.0	.
253677	0.0	0.0	0.0	1.0	28.0	0.0	0.0	0.0	1.0	1.0	.
253678	0.0	1.0	0.0	1.0	23.0	0.0	0.0	0.0	0.0	1.0	.
253679	2.0	1.0	1.0	1.0	25.0	0.0	0.0	1.0	1.0	1.0	.

253680 rows × 22 columns

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_012          253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

In [5]:
```
# Check for null values in the entire dataset
null_values = data.isnull().sum()

# Display the count of null values for each column
print("Null Values in Each Column:")
print(null_values)
```

```
Null Values in Each Column:
Diabetes_012            0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

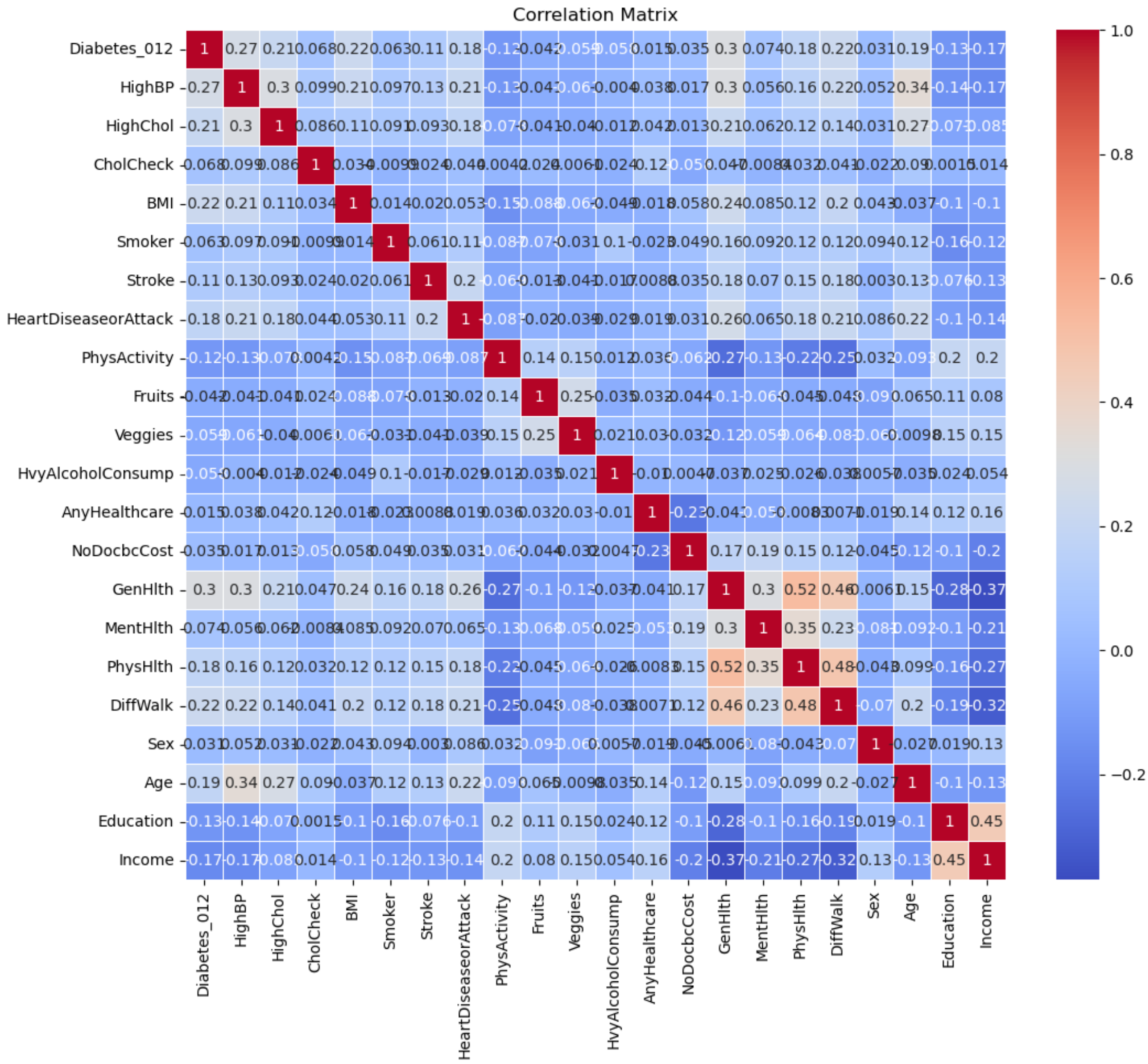
In [6]:
```
# Check if there are any null values in the dataset
if data.isnull().values.any():
    print("\nThere are null values in the dataset.")
else:
    print("\nThere are no null values in the dataset.")
```

```

There are no null values in the dataset.
```

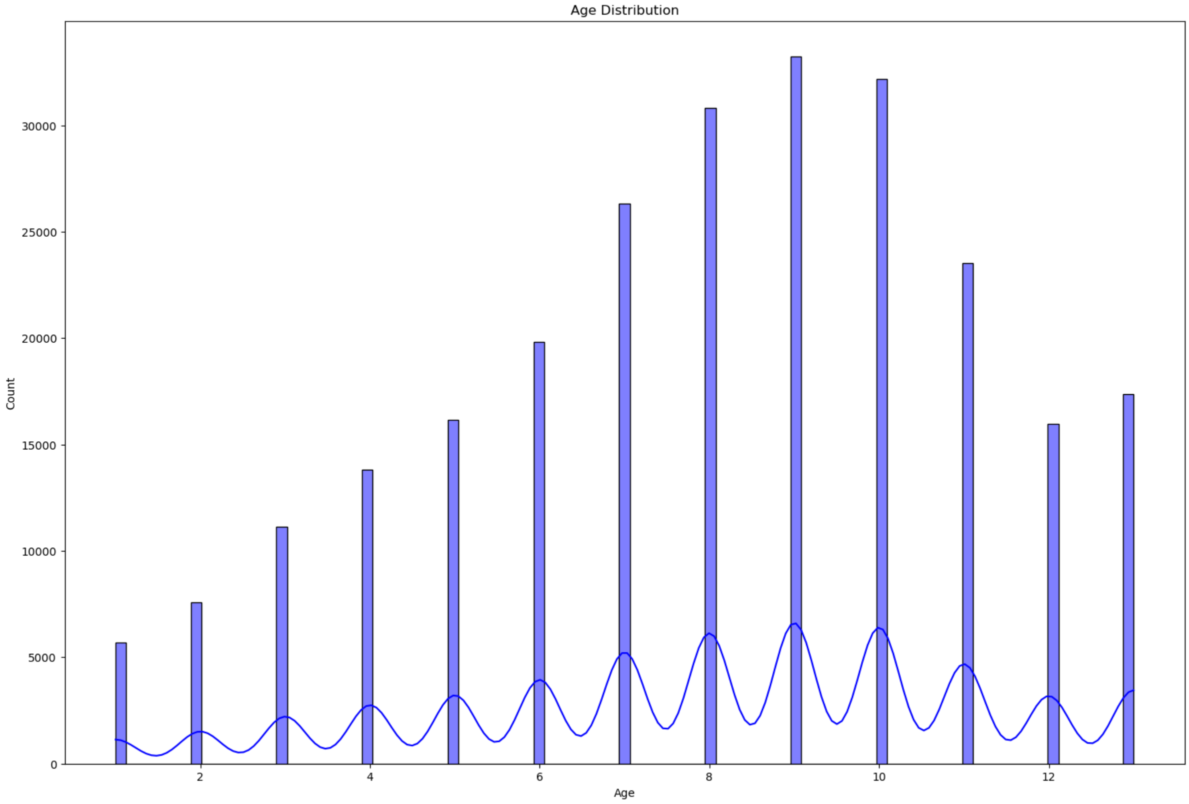
In [7]:

```python
# Correlation matrix
correlation_matrix = data.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title("Correlation Matrix")
plt.show()
```



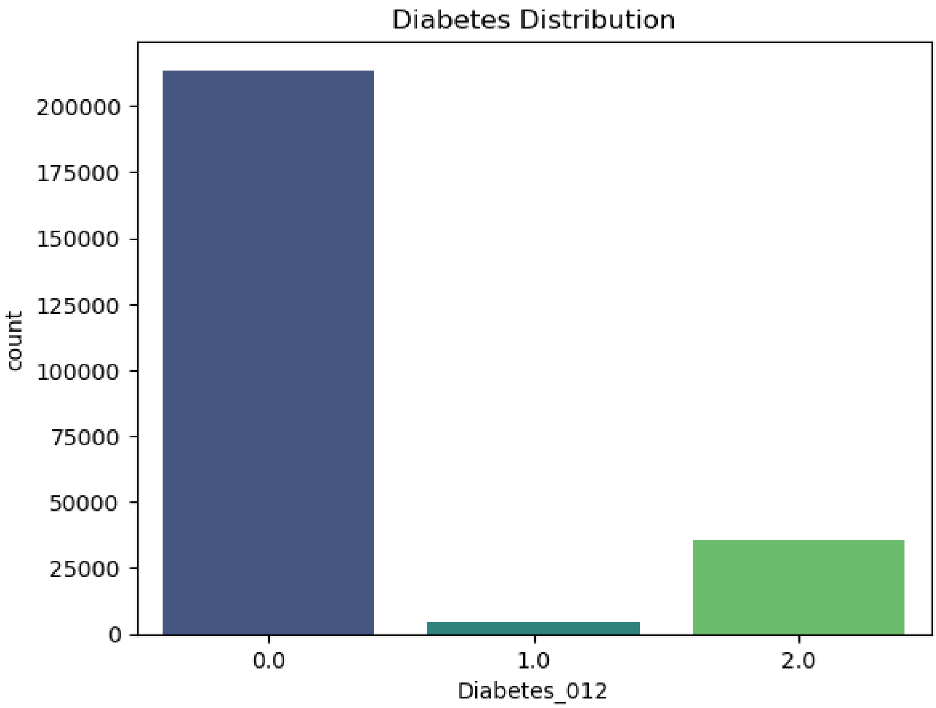
In [8]:
```python
# Distribution of some key features
plt.figure(figsize=(18, 12))
sns.histplot(data['Age'], kde=True, color='blue')
plt.title('Age Distribution')
```

Out[8]: Text(0.5, 1.0, 'Age Distribution')



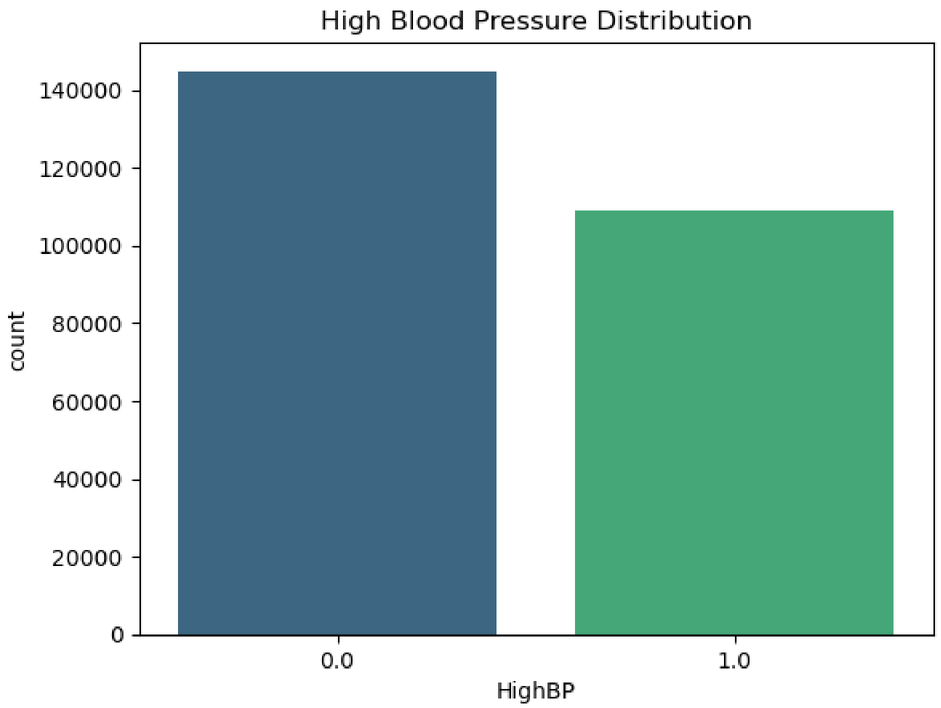
In [9]:
```python
sns.countplot(x='Diabetes_012', data=data, palette='viridis')
plt.title('Diabetes Distribution')
```

Out[9]: Text(0.5, 1.0, 'Diabetes Distribution')



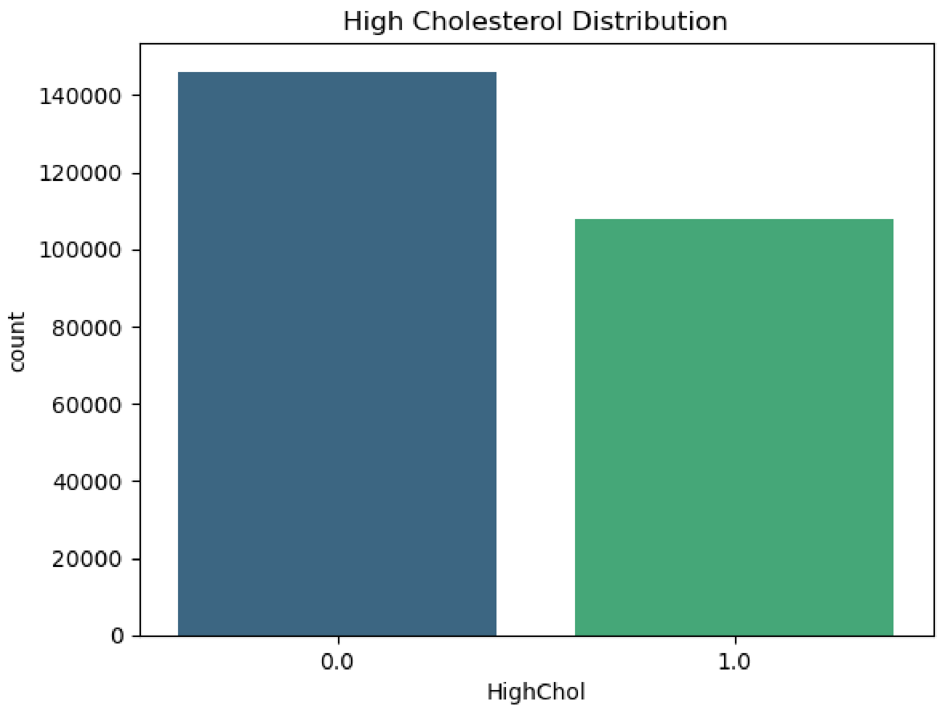
In [10]:
```python
sns.countplot(x='HighBP', data=data, palette='viridis')
plt.title('High Blood Pressure Distribution')
```

Out[10]: Text(0.5, 1.0, 'High Blood Pressure Distribution')



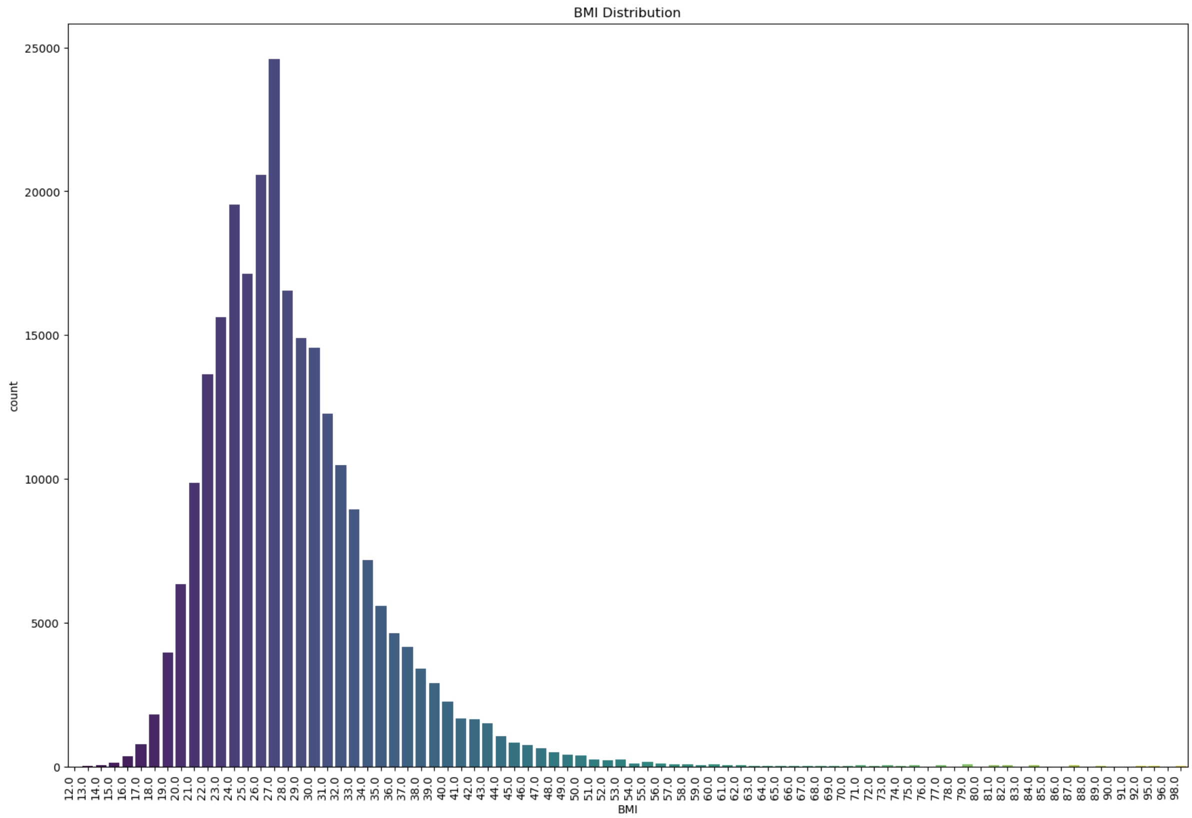
In [11]:
```python
sns.countplot(x='HighChol', data=data, palette='viridis')
plt.title('High Cholesterol Distribution')
```

Out[11]: Text(0.5, 1.0, 'High Cholesterol Distribution')



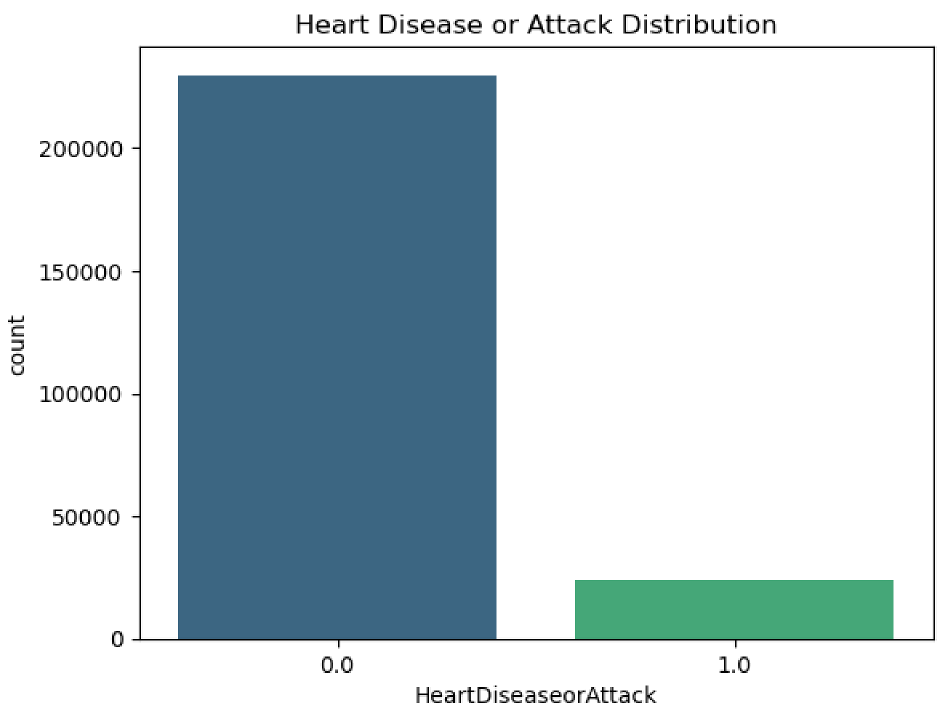
In [12]:
```
plt.figure(figsize=(18, 12))
sns.countplot(x='BMI', data=data, palette='viridis')
plt.title('BMI Distribution')
# Rotate x-axis labels at a 45-degree angle
plt.xticks(rotation=90, ha='right')

plt.show()
```



In [13]:
```
sns.countplot(x='HeartDiseaseorAttack', data=data, palette='viridis')
plt.title('Heart Disease or Attack Distribution')
```

Out[13]: Text(0.5, 1.0, 'Heart Disease or Attack Distribution')



Develop a predictive model using classification techniques like k-Nearest Neighbors (kNN), Naive Bayes, or Logistic Regression to classify individuals into three diabetes types (0 for no diabetes or only during pregnancy, 1 for

In [14]:
```python
X = data.drop('Diabetes_012', axis=1)
y = data['Diabetes_012']
```

In [15]:
```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [16]:
```python
# Initialize the Random Forest Classifier
model = RandomForestClassifier(random_state=42)

# Train the model
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 84.12%
```

In [17]:
```python
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

         0.0       0.86      0.97      0.91     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.47      0.20      0.28      6997

    accuracy                           0.84     50736
   macro avg       0.44      0.39      0.40     50736
weighted avg       0.79      0.84      0.81     50736
```

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

# Assuming you have your features in X and labels in y
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
dt_model.fit(X_train, y_train)

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

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
report_dt = classification_report(y_test, y_pred_dt)

# Print the results
print(f"Decision Tree Accuracy: {accuracy_dt * 100:.2f}%")
print("Decision Tree Classification Report:")
print(report_dt)
```

```
Decision Tree Accuracy: 76.73%
Decision Tree Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.86      0.87     42795
         1.0       0.04      0.05      0.04       944
         2.0       0.29      0.33      0.31      6997

    accuracy                           0.77     50736
   macro avg       0.40      0.41      0.41     50736
weighted avg       0.78      0.77      0.77     50736
```

In [19]:
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [20]:
```
# Initialize k-Nearest Neighbors classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the classifier
knn_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.82
```

In [21]:
```python
# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.93      0.90     42795
         1.0       0.07      0.01      0.01       944
         2.0       0.36      0.24      0.29      6997

    accuracy                           0.82     50736
   macro avg       0.43      0.39      0.40     50736
weighted avg       0.78      0.82      0.80     50736
```

In [22]:
```python
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Initialize Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier
dt_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_dt = dt_classifier.predict(X_test_scaled)

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {accuracy_dt:.2f}")
```

```
Decision Tree Accuracy: 0.77
```

In [23]:
```python
# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_dt))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.86      0.87     42795
         1.0       0.04      0.05      0.04       944
         2.0       0.29      0.33      0.31      6997

    accuracy                           0.77     50736
   macro avg       0.40      0.41      0.41     50736
weighted avg       0.78      0.77      0.77     50736
```

In [24]:
```python
# Logistic Regression Model
model = LogisticRegression(max_iter=1000)

# Train the model
model.fit(X_train_scaled, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display the results
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 84.83%
```

In [25]:
```python
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[41755     0  1040]
 [  871     0    73]
 [ 5714     0  1283]]
```

In [26]:
```python
print("Classification Report:")
print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.86      0.98      0.92     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.54      0.18      0.27      6997

    accuracy                           0.85     50736
   macro avg       0.47      0.39      0.40     50736
weighted avg       0.80      0.85      0.81     50736
```

Create a predictive model to classify individuals as either having no diabetes (0) or having prediabetes/diabetes (1) using the balanced Diabetes_binary 50-50 split dataset.

In [27]:
```python
# Select features and target variable
X = data.drop('Diabetes_012', axis=1)
y = data['Diabetes_012']
```

In [28]:
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42, stratify
```

In [29]:
```python
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [30]:
```python
# Create a logistic regression model
model = LogisticRegression(random_state=42)

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Display results
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.85
```

In [31]:
```python
print('Confusion Matrix:')
print(conf_matrix)
```

```
Confusion Matrix:
[[104092      0   2760]
 [  2087      0    228]
 [ 14479      0   3194]]
```

In [32]:
```
print('Classification Report:')
print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.86      0.97      0.92    106852
         1.0       0.00      0.00      0.00      2315
         2.0       0.52      0.18      0.27     17673

    accuracy                           0.85    126840
   macro avg       0.46      0.38      0.39    126840
weighted avg       0.80      0.85      0.81    126840
```

In [33]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming you have your features in X and labels in y
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features (optional but often recommended)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Decision Tree model
tree_model = DecisionTreeClassifier(random_state=42)

# Train the model
tree_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred_tree = tree_model.predict(X_test_scaled)

# Evaluate the Decision Tree model
accuracy_tree = accuracy_score(y_test, y_pred_tree)
conf_matrix_tree = confusion_matrix(y_test, y_pred_tree)
classification_rep_tree = classification_report(y_test, y_pred_tree)

# Display results
print(f'Decision Tree Accuracy: {accuracy_tree:.2f}')
print(f'Confusion Matrix:\n{conf_matrix_tree}')
print(f'Classification Report:\n{classification_rep_tree}')
```

```
Decision Tree Accuracy: 0.77
Confusion Matrix:
[[36637   875  5283]
 [  655    45   244]
 [ 4435   285  2277]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.86      0.87     42795
         1.0       0.04      0.05      0.04       944
         2.0       0.29      0.33      0.31      6997

    accuracy                           0.77     50736
   macro avg       0.40      0.41      0.41     50736
weighted avg       0.78      0.77      0.77     50736
```

Create a predictive model using classification techniques like k-Nearest Neighbors (kNN), Naive Bayes, or Logistic Regression to classify individuals as either having no diabetes (0) or having prediabetes/diabetes (1) using the balanced Diabetes_binary 50-50 split dataset.

In [34]:
```python
# Create k-Nearest Neighbors (kNN) model
knn_model = KNeighborsClassifier(n_neighbors=5)

# Train the model
knn_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = knn_model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display the results
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 83.03%
```

In [35]:
```python
print("\nConfusion Matrix:\n", conf_matrix)
```

```

Confusion Matrix:
 [[40659    24  2112]
 [  814     1   129]
 [ 5517    14  1466]]
```

In [36]:
```python
print("\nClassification Report:\n", class_report)
```

```

Classification Report:
               precision    recall  f1-score   support

         0.0       0.87      0.95      0.91     42795
         1.0       0.03      0.00      0.00       944
         2.0       0.40      0.21      0.27      6997

    accuracy                           0.83     50736
   macro avg       0.43      0.39      0.39     50736
weighted avg       0.78      0.83      0.80     50736
```

In [37]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming you have your features in X and labels in y
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features (if needed)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
dt_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = dt_model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display the results
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 76.79%
Confusion Matrix:
[[36637   875  5283]
 [  655    45   244]
 [ 4435   285  2277]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.86      0.87     42795
         1.0       0.04      0.05      0.04       944
         2.0       0.29      0.33      0.31      6997

    accuracy                           0.77     50736
   macro avg       0.40      0.41      0.41     50736
weighted avg       0.78      0.77      0.77     50736
```

In [38]:

```
# Initialize the Logistic Regression model
logreg_model = LogisticRegression(random_state=42)

# Train the model on the training set
logreg_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = logreg_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Display the results
print(f"Accuracy: {accuracy*100:.2f}%")
```

```
Accuracy: 84.49%
```

In [39]:
```python
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[41785     0  1010]
 [  878     0    66]
 [ 5913     0  1084]]
```

In [40]:
```python
print("\nClassification Report:")
print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.86      0.98      0.91     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.50      0.15      0.24      6997

    accuracy                           0.84     50736
   macro avg       0.45      0.38      0.38     50736
weighted avg       0.79      0.84      0.80     50736
```

In [41]:
```python
# Apply SMOTE to address class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

In [42]:
```python
# Build a predictive model (Random Forest in this example)
model = RandomForestClassifier(random_state=42)
model.fit(X_train_resampled, y_train_resampled)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

In [43]:
```python
# Print the evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.84
```

In [44]:
```python
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[40936    45  1814]
 [  806     0   138]
 [ 5367    11  1619]]
```

In [45]:
```python
print("\nClassification Report:")
print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.96      0.91     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.45      0.23      0.31      6997

    accuracy                           0.84     50736
   macro avg       0.44      0.40      0.41     50736
weighted avg       0.80      0.84      0.81     50736
```

In [46]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Assuming you have a dataset with features X and target variable y
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build a Decision Tree model
model = DecisionTreeClassifier(random_state=42)

# Train the model on the training data
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.7673446862188584
Confusion Matrix:
 [[36608   884  5303]
 [  652    45   247]
 [ 4439   279  2279]]
Classification Report:
               precision    recall  f1-score   support

         0.0       0.88      0.86      0.87     42795
         1.0       0.04      0.05      0.04       944
         2.0       0.29      0.33      0.31      6997

    accuracy                           0.77     50736
   macro avg       0.40      0.41      0.41     50736
weighted avg       0.78      0.77      0.77     50736
```

Explore classification techniques such as k-Nearest Neighbors (kNN), Naive Bayes, or Logistic Regression while addressing the challenge of class imbalance in the imbalanced Diabetes_binary dataset. Techniques like oversampling, undersampling, or advanced algorithms can be applied to handle imbalanced datasets.

In [47]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

In [48]:
```
# Assuming 'data' is your DataFrame with the provided columns
X = data.drop('Diabetes_012', axis=1)
y = data['Diabetes_012']
```

In [49]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [50]:
```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [51]:
```
# Handle class imbalance using RandomOverSampler
oversampler = RandomOverSampler(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)
```

In [52]:
```
# Create pipelines for each classification technique
knn_model = Pipeline([
    ('classifier', KNeighborsClassifier(n_neighbors=5))
])
```

In [53]:
```
nb_model = Pipeline([
    ('classifier', GaussianNB())
])
```

In [54]:
```
lr_model = Pipeline([
    ('classifier', LogisticRegression())
])
```

In [55]:
```
# Create a Decision Tree model and a pipeline
dt_model = Pipeline([
    ('classifier', DecisionTreeClassifier(random_state=42))
])
```

In [56]:
```
# Train models on resampled data
knn_model.fit(X_train_resampled, y_train_resampled)
nb_model.fit(X_train_resampled, y_train_resampled)
lr_model.fit(X_train_resampled, y_train_resampled)
dt_model.fit(X_train_resampled, y_train_resampled)
```

Out[56]:
```
Pipeline(steps=[('classifier', DecisionTreeClassifier(random_state=42))])
```

In [57]:
```
# Evaluate models on the original test set
y_pred_knn = knn_model.predict(X_test)
y_pred_nb = nb_model.predict(X_test)
y_pred_lr = lr_model.predict(X_test)
y_pred_dt = dt_model.predict(X_test)
```

In [58]:
```
# Print classification reports and confusion matrices
print("k-Nearest Neighbors:")
print(classification_report(y_test, y_pred_knn))
```

```
k-Nearest Neighbors:
              precision    recall  f1-score   support

         0.0       0.91      0.71      0.80     42795
         1.0       0.03      0.07      0.04       944
         2.0       0.27      0.57      0.36      6997

    accuracy                           0.68     50736
   macro avg       0.40      0.45      0.40     50736
weighted avg       0.81      0.68      0.72     50736
```

In [59]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_knn))
```

```
Confusion Matrix:
[[30417  1740 10638]
 [  455    64   425]
 [ 2531   463  4003]]
```

In [60]:
```
print("\nNaive Bayes:")
print(classification_report(y_test, y_pred_nb))
```

```
Naive Bayes:
              precision    recall  f1-score   support

         0.0       0.93      0.70      0.80     42795
         1.0       0.03      0.12      0.04       944
         2.0       0.31      0.63      0.42      6997

    accuracy                           0.68     50736
   macro avg       0.42      0.49      0.42     50736
weighted avg       0.83      0.68      0.73     50736
```

In [61]:
```
print("\nDecision Tree:")
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree:
              precision    recall  f1-score   support

         0.0       0.87      0.86      0.86     42795
         1.0       0.02      0.03      0.03       944
         2.0       0.28      0.31      0.29      6997

    accuracy                           0.76     50736
   macro avg       0.39      0.40      0.39     50736
weighted avg       0.78      0.76      0.77     50736
```

In [62]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))
```

```
Confusion Matrix:
[[29858  3675  9262]
 [  392   117   435]
 [ 1906   651  4440]]
```

In [63]:
```
print("\nLogistic Regression:")
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression:
              precision    recall  f1-score   support

         0.0       0.95      0.66      0.78     42795
         1.0       0.03      0.31      0.06       944
         2.0       0.35      0.59      0.44      6997

    accuracy                           0.65     50736
   macro avg       0.45      0.52      0.43     50736
weighted avg       0.85      0.65      0.72     50736
```

In [64]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_lr))
```

```
Confusion Matrix:
[[28390  7173  7232]
 [  269   295   380]
 [ 1124  1729  4144]]
```

Identify and select the most influential features in predicting diabetes outcomes from the 21 available features.

In [65]:
```
# Create a Random Forest classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

Out[65]:
```
RandomForestClassifier(random_state=42)
```

In [66]:
```
# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame with feature names and their importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort features by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
```

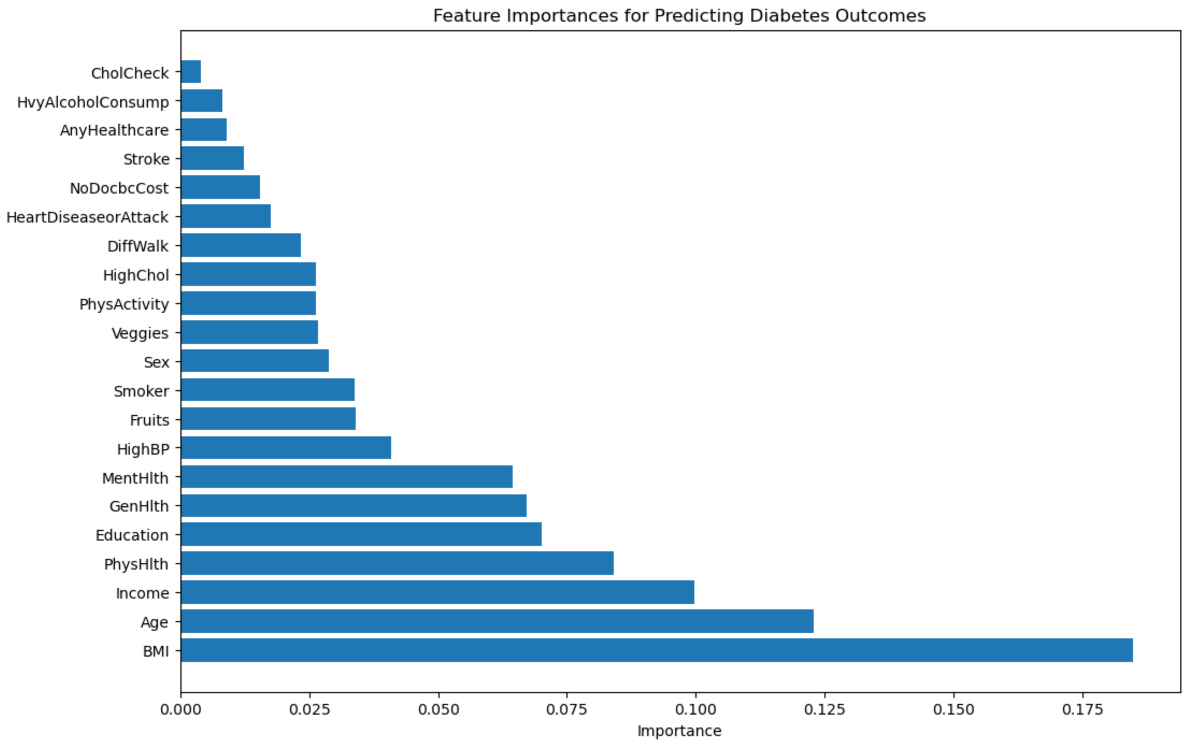
In [67]:
```
N = 10

# Select the top N most important features
top_features = feature_importance_df.head(N)['Feature'].tolist()
```

In [68]:
```
# Print and visualize the feature importances
print("Top Features:")
print(top_features)
```

```
Top Features:
['BMI', 'Age', 'Income', 'PhysHlth', 'Education', 'GenHlth', 'MentHlth', 'HighBP', 'Fruits', 'Smo
ker']
```

In [69]:
```
# Plot the feature importances
plt.figure(figsize=(12, 8))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Importance')
plt.title('Feature Importances for Predicting Diabetes Outcomes')
plt.show()
```



Use classification techniques like Logistic Regression, which inherently provides feature importance, or employ techniques like Recursive Feature Elimination (RFE) with k-Nearest Neighbors (kNN) or Naive Bayes to identify and select the most influential features in predicting diabetes outcomes from the 21 available features.

In [70]:
```
# Use RFE to select the top 10 features
rfe = RFE(model, n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test)
```

In [71]:
```
# Create pipelines for other classification techniques (k-Nearest Neighbors and Naive Bayes)
knn_model = KNeighborsClassifier(n_neighbors=5)
nb_model = GaussianNB()
```

In [72]:
```
# Train models on the original data
knn_model.fit(X_train, y_train)
nb_model.fit(X_train, y_train)

# Train models on the RFE selected features
knn_model_rfe = KNeighborsClassifier(n_neighbors=5)
nb_model_rfe = GaussianNB()

knn_model_rfe.fit(X_train_rfe, y_train)
nb_model_rfe.fit(X_train_rfe, y_train)
```

Out[72]:
```
GaussianNB()
```

In [73]:
```
# Evaluate models on the original test set
y_pred_knn = knn_model.predict(X_test)
y_pred_nb = nb_model.predict(X_test)

# Evaluate models on the RFE selected features
y_pred_knn_rfe = knn_model_rfe.predict(X_test_rfe)
y_pred_nb_rfe = nb_model_rfe.predict(X_test_rfe)
```

In [74]:
```
# Print classification reports and confusion matrices for original and RFE-selected features
print("Original Features:")
print("\nOriginal k-Nearest Neighbors:")
print(classification_report(y_test, y_pred_knn))
```

```
Original Features:

Original k-Nearest Neighbors:
              precision    recall  f1-score   support

         0.0       0.87      0.95      0.91     42795
         1.0       0.03      0.00      0.00       944
         2.0       0.40      0.21      0.27      6997

    accuracy                           0.83     50736
   macro avg       0.43      0.39      0.39     50736
weighted avg       0.78      0.83      0.80     50736
```

In [75]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_knn))
```

```
Confusion Matrix:
[[40659    24  2112]
 [  814     1   129]
 [ 5517    14  1466]]
```

In [76]:
```
print("\nOriginal Naive Bayes:")
print(classification_report(y_test, y_pred_nb))
```

```

Original Naive Bayes:
              precision    recall  f1-score   support

         0.0       0.91      0.80      0.85     42795
         1.0       0.04      0.02      0.03       944
         2.0       0.32      0.56      0.40      6997

    accuracy                           0.75     50736
   macro avg       0.42      0.46      0.43     50736
weighted avg       0.81      0.75      0.77     50736
```

In [77]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))
```

```
Confusion Matrix:
[[34298   414  8083]
 [  542    21   381]
 [ 2960   115  3922]]
```

In [78]:
```
print("\nRFE Selected Features:")
print("\nRFE k-Nearest Neighbors:")
print(classification_report(y_test, y_pred_knn_rfe))
```

```

RFE Selected Features:

RFE k-Nearest Neighbors:
              precision    recall  f1-score   support

         0.0       0.86      0.95      0.91     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.39      0.20      0.27      6997

    accuracy                           0.83     50736
   macro avg       0.42      0.38      0.39     50736
weighted avg       0.78      0.83      0.80     50736
```

In [79]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_knn_rfe))
```

```
Confusion Matrix:
[[40683    32  2080]
 [  825     0   119]
 [ 5554    15  1428]]
```

In [80]:
```
print("\nRFE Naive Bayes:")
print(classification_report(y_test, y_pred_nb_rfe))
```

```

RFE Naive Bayes:
              precision    recall  f1-score   support

         0.0       0.89      0.87      0.88     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.34      0.43      0.38      6997

    accuracy                           0.79     50736
   macro avg       0.41      0.43      0.42     50736
weighted avg       0.80      0.79      0.79     50736
```

In [81]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb_rfe))
```

```
Confusion Matrix:
[[37233     1  5561]
 [  666     0   278]
 [ 3992     0  3005]]
```

In [82]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming you have your data and target variable (X and y) ready
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
dt_model.fit(X_train, y_train)

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

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree Accuracy: {accuracy}")
```

```
Decision Tree Accuracy: 0.7678768527278461
```

Implement techniques to make machine learning models more interpretable and explainable, especially for healthcare applications.

In [83]:
```
!pip install shap
```

```
Requirement already satisfied: shap in c:\users\tanmayee\anaconda3\lib\site-packages (0.43.0)
Requirement already satisfied: pandas in c:\users\tanmayee\anaconda3\lib\site-packages (from sha
p) (1.4.4)
Requirement already satisfied: numpy in c:\users\tanmayee\anaconda3\lib\site-packages (from shap)
(1.21.6)
Requirement already satisfied: packaging>20.9 in c:\users\tanmayee\anaconda3\lib\site-packages (f
rom shap) (21.3)
Requirement already satisfied: cloudpickle in c:\users\tanmayee\anaconda3\lib\site-packages (from
shap) (2.0.0)
Requirement already satisfied: scipy in c:\users\tanmayee\anaconda3\lib\site-packages (from shap)
(1.9.1)
Requirement already satisfied: numba in c:\users\tanmayee\anaconda3\lib\site-packages (from shap)
(0.55.1)
Requirement already satisfied: slicer==0.0.7 in c:\users\tanmayee\anaconda3\lib\site-packages (fr
om shap) (0.0.7)
Requirement already satisfied: tqdm>=4.27.0 in c:\users\tanmayee\anaconda3\lib\site-packages (fro
m shap) (4.64.1)
Requirement already satisfied: scikit-learn in c:\users\tanmayee\anaconda3\lib\site-packages (fro
m shap) (1.0.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\tanmayee\anaconda3\lib\site-p
ackages (from packaging>20.9->shap) (3.0.9)
Requirement already satisfied: colorama in c:\users\tanmayee\anaconda3\lib\site-packages (from tq
dm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: setuptools in c:\users\tanmayee\anaconda3\lib\site-packages (from
numba->shap) (63.4.1)
Requirement already satisfied: llvmlite<0.39,>=0.38.0rc1 in c:\users\tanmayee\anaconda3\lib\site-
packages (from numba->shap) (0.38.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\tanmayee\anaconda3\lib\site-pac
kages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\tanmayee\anaconda3\lib\site-packages (fro
m pandas->shap) (2023.3)
Requirement already satisfied: joblib>=0.11 in c:\users\tanmayee\anaconda3\lib\site-packages (fro
m scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\tanmayee\anaconda3\lib\site-packa
ges (from scikit-learn->shap) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\tanmayee\anaconda3\lib\site-packages (from py
thon-dateutil>=2.8.1->pandas->shap) (1.16.0)

WARNING: Ignoring invalid distribution -rotobuf (c:\users\tanmayee\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\tanmayee\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\tanmayee\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\tanmayee\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\tanmayee\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\tanmayee\anaconda3\lib\site-packages)
```

In [84]:
```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
```

In [85]:
```
# Assuming 'data' is your DataFrame with the provided columns
X = data.drop('Diabetes_012', axis=1)
y = data['Diabetes_012']
```

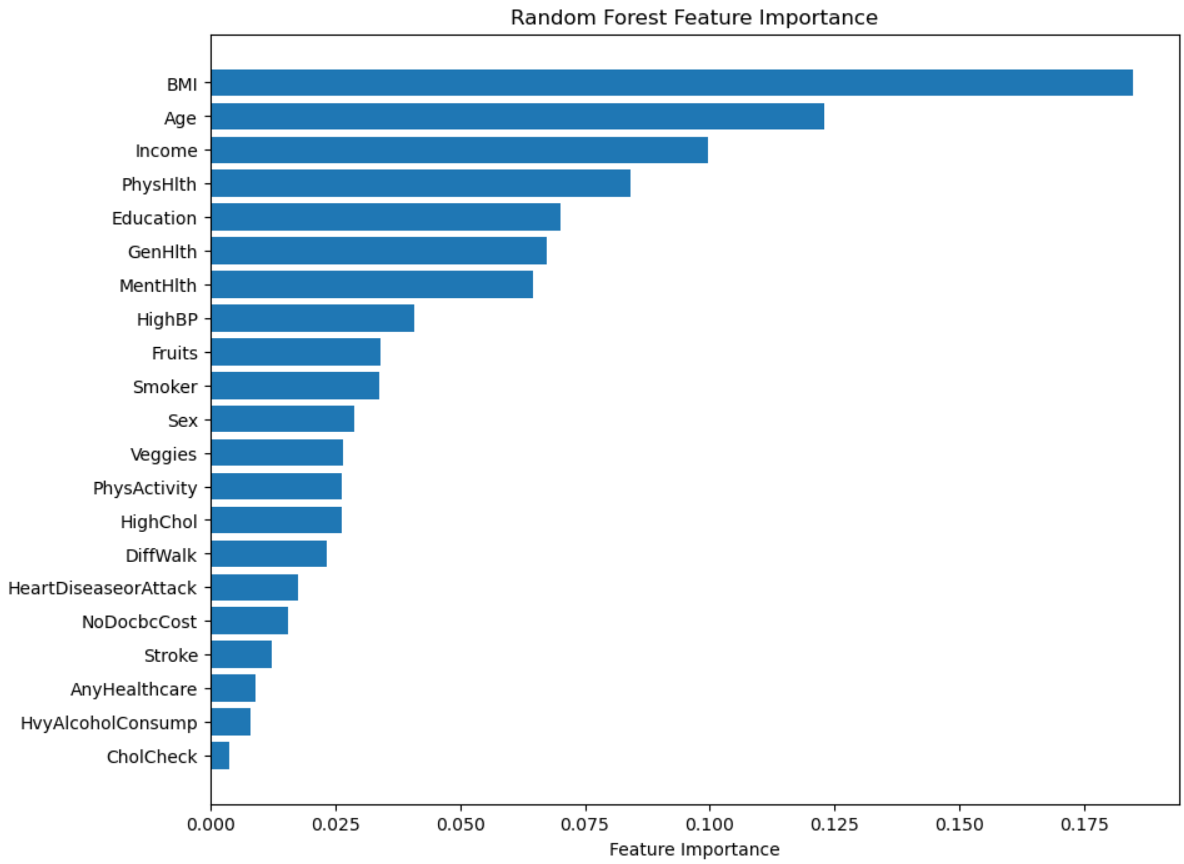
In [86]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [87]:
```
# Train a RandomForestClassifier (you can replace this with your model)
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Out[87]:
```
RandomForestClassifier(random_state=42)
```

In [88]:
```
# Feature Importance Plot
feature_importances = model.feature_importances_
feature_names = X.columns
sorted_idx = feature_importances.argsort()
```

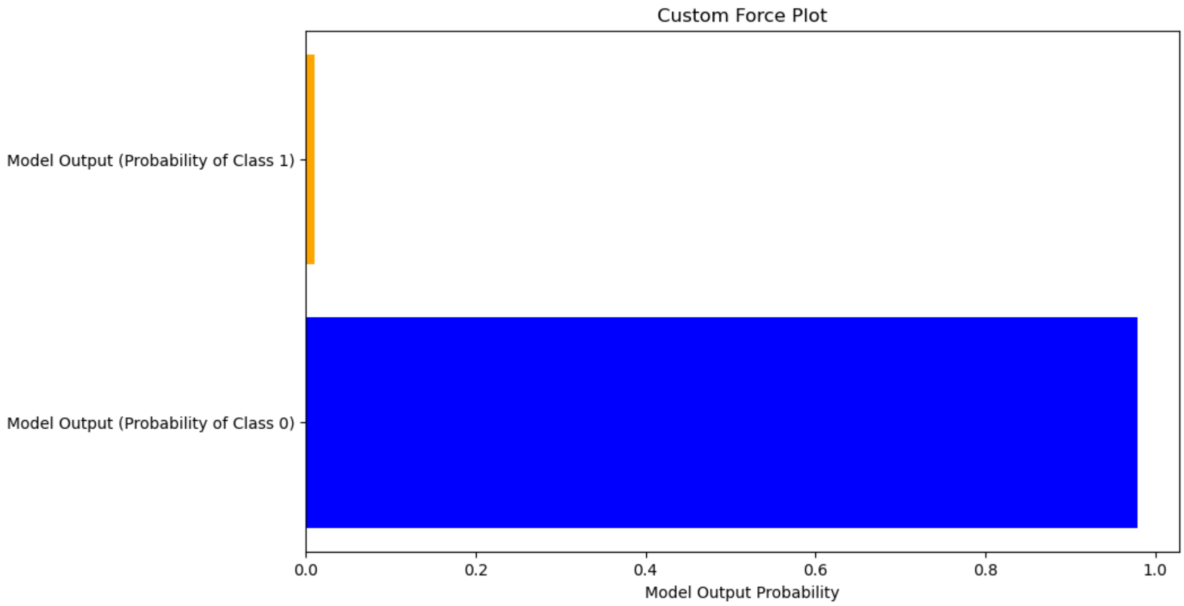
In [89]:
```
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.show()
```



In [90]:
```python
def custom_force_plot(model, sample, feature_names):
    base_value = model.predict_proba([sample])[0][0]
    output_value = model.predict_proba([sample])[0][1]

    plt.figure(figsize=(10, 6))
    plt.barh(['Model Output (Probability of Class 0)', 'Model Output (Probability of Class 1)'],
             [base_value, output_value], color=['blue', 'orange'])
    plt.xlabel('Model Output Probability')
    plt.title('Custom Force Plot')
    plt.show()

# Custom Force Plot for the first test sample
custom_force_plot(model, X_test.iloc[0, :], feature_names)
```



Implement classification techniques like Logistic Regression, which is inherently more interpretable, or use techniques like SHAP (SHapley Additive exPlanations) values with k-Nearest Neighbors (kNN) or Naive Bayes to enhance the interpretability and explainability of the models.

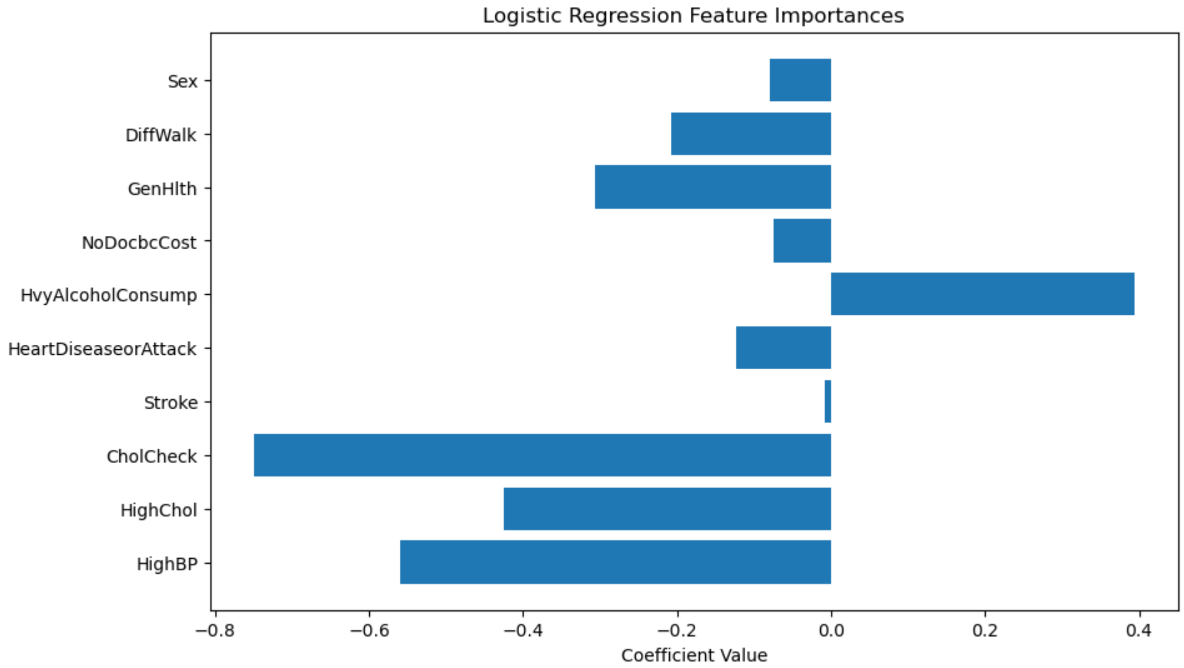
In [91]:
```python
# Use RFE to select the top N features
N = 10  # Choose the desired number of top features
rfe = RFE(logreg_model, n_features_to_select=N)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test)
```

In [92]:
```python
# Train the model on the selected features
logreg_model.fit(X_train_rfe, y_train)

# Get the feature names corresponding to the selected features
selected_feature_names = np.array(X.columns)[rfe.support_]
```

In [93]:
```python
# Ensure N is not greater than the number of selected features
N = min(N, len(selected_feature_names))
```

In [94]:
```
# Visualize feature importances
coefficients = logreg_model.coef_.flatten()[:N]  # Take only N coefficients
plt.figure(figsize=(10, 6))
plt.barh(selected_feature_names[:N], coefficients)
plt.xlabel('Coefficient Value')
plt.title('Logistic Regression Feature Importances')
plt.show()
```



In [95]:
```
# Evaluate the model
y_pred = logreg_model.predict(X_test_rfe)
```

In [96]:
```
print("Logistic Regression:")
print(classification_report(y_test, y_pred))
```

```
Logistic Regression:
              precision    recall  f1-score   support

         0.0       0.86      0.98      0.91     42795
         1.0       0.00      0.00      0.00       944
         2.0       0.49      0.14      0.22      6997

    accuracy                           0.84     50736
   macro avg       0.45      0.37      0.38     50736
weighted avg       0.79      0.84      0.80     50736
```

In [97]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[41855     0   940]
 [  876     0    68]
 [ 6028     0   969]]
```