```
import numpy as np  # For numerical operations
import pandas as pd  # For handling datasets
import matplotlib.pyplot as plt  # For plotting graphs
import seaborn as sns  # For advanced visualizations
from sklearn.model_selection import train_test_split  # To split data into training & testing
from sklearn.preprocessing import StandardScaler  # For feature scaling
from sklearn.feature_selection import mutual_info_classif
from imblearn.over_sampling import SMOTE  # For handling imbalanced data
from imblearn.over_sampling import SMOTE, ADASYN  # Oversampling techniques
from imblearn.under_sampling import RandomUnderSampler  # Undersampling technique
from imblearn.combine import SMOTETomek  # Hybrid approach (SMOTE + Tomek Links)
from sklearn.tree import DecisionTreeClassifier  # Decision Tree Model
from sklearn.svm import SVC  # Support Vector Machine
from sklearn.ensemble import RandomForestClassifier  # Random Forest Model
from sklearn.linear_model import LogisticRegression  # Logistic Regression
from xgboost import XGBClassifier  # XGBoost Model
import tensorflow as tf  # TensorFlow for deep learning
from tensorflow import keras  # Keras API for ANN
from tensorflow.keras.models import Sequential  # Sequential model for ANN
from tensorflow.keras.layers import Dense, Dropout  # ANN layers
from tensorflow.keras.optimizers import Adam, SGD, RMSprop  # Optimizers for tuning
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_auc_score
```

```
df=pd.read_csv('/content/framingham_expanded_v2.csv')
```

```
df.head()
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	...	sleepHours	stress
0	0	35	2.0	0	0.0	0.0	0	0	0	248.0	...	7.311960	
1	1	39	2.0	1	10.0	0.0	0	0	0	215.0	...	7.014648	
2	0	60	2.0	0	0.0	0.0	0	1	0	298.0	...	4.000000	
3	0	57	3.0	1	15.0	0.0	0	0	0	250.0	...	6.024542	
4	0	36	1.0	1	5.0	0.0	0	1	0	222.0	...	8.245791	

5 rows × 30 columns

```
df.shape
```

```
(15000, 30)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 30 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   male                 15000 non-null  int64  
 1   age                  15000 non-null  int64  
 2   education            14611 non-null  float64
 3   currentSmoker        15000 non-null  int64  
 4   cigsPerDay           14898 non-null  float64
 5   BPMeds               14786 non-null  float64
 6   prevalentStroke      15000 non-null  int64  
 7   prevalentHyp         15000 non-null  int64  
 8   diabetes             15000 non-null  int64  
 9   totChol              14824 non-null  float64
 10  sysBP                15000 non-null  float64
 11  diaBP                15000 non-null  float64
 12  BMI                  14937 non-null  float64
 13  heartRate            14994 non-null  float64
 14  glucose              13630 non-null  float64
 15  TenYearCHD           15000 non-null  int64  
 16  physicalActivity     15000 non-null  int64  
 17  familyHistory        15000 non-null  int64  
 18  diet                 15000 non-null  int64  
 19  cholesterolRatio     14824 non-null  float64
 20  sleepHours           15000 non-null  float64
 21  stressLevel          15000 non-null  int64  
 22  waistHipRatio        15000 non-null  float64
 23  restingHeartRate     15000 non-null  float64
 24  alcoholConsumption   15000 non-null  int64  
 25  exerciseFrequency    15000 non-null  int64  
 26  sodiumIntake         15000 non-null  float64
 27  mentalHealthIndex    15000 non-null  int64  
 28  airPollutionExposure 15000 non-null  int64  
```

```
 29  medicationAdherence   15000 non-null  int64
dtypes: float64(14), int64(16)
memory usage: 3.4 MB
```

```
df.isnull().sum() #check missing values
```

	0
male	0
age	0
education	389
currentSmoker	0
cigsPerDay	102
BPMeds	214
prevalentStroke	0
prevalentHyp	0
diabetes	0
totChol	176
sysBP	0
diaBP	0
BMI	63
heartRate	6
glucose	1370
TenYearCHD	0
physicalActivity	0
familyHistory	0
diet	0
cholesterolRatio	176
sleepHours	0
stressLevel	0
waistHipRatio	0
restingHeartRate	0
alcoholConsumption	0
exerciseFrequency	0
sodiumIntake	0
mentalHealthIndex	0
airPollutionExposure	0
medicationAdherence	0

dtype: int64

```
df['TenYearCHD'].value_counts(normalize=True) * 100
#check for class imbaalnce
```

TenYearCHD	proportion
0	84.346667
1	15.653333

dtype: float64

```
df.fillna(df.median(), inplace=True)  # Replace missing values with median (numerical)
df.fillna(df.mode().iloc[0], inplace=True)  # Replace with most frequent value (categorical)
```

```
df.isnull().sum()  # Should now show all zeros
```

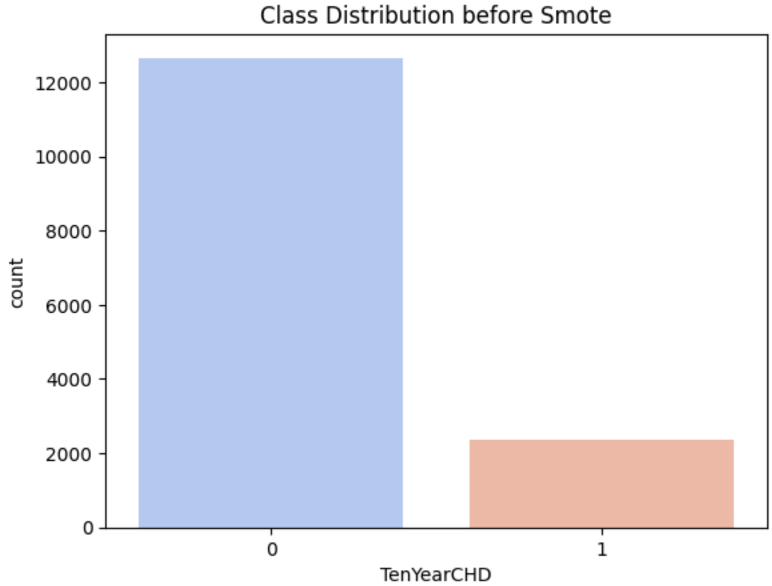
	0
male	0
age	0
education	0
currentSmoker	0
cigsPerDay	0
BPMeds	0
prevalentStroke	0
prevalentHyp	0
diabetes	0
totChol	0
sysBP	0
diaBP	0
BMI	0
heartRate	0
glucose	0
TenYearCHD	0
physicalActivity	0
familyHistory	0
diet	0
cholesterolRatio	0
sleepHours	0
stressLevel	0
waistHipRatio	0
restingHeartRate	0
alcoholConsumption	0
exerciseFrequency	0
sodiumIntake	0
mentalHealthIndex	0
airPollutionExposure	0
medicationAdherence	0

dtype: int64

EDA

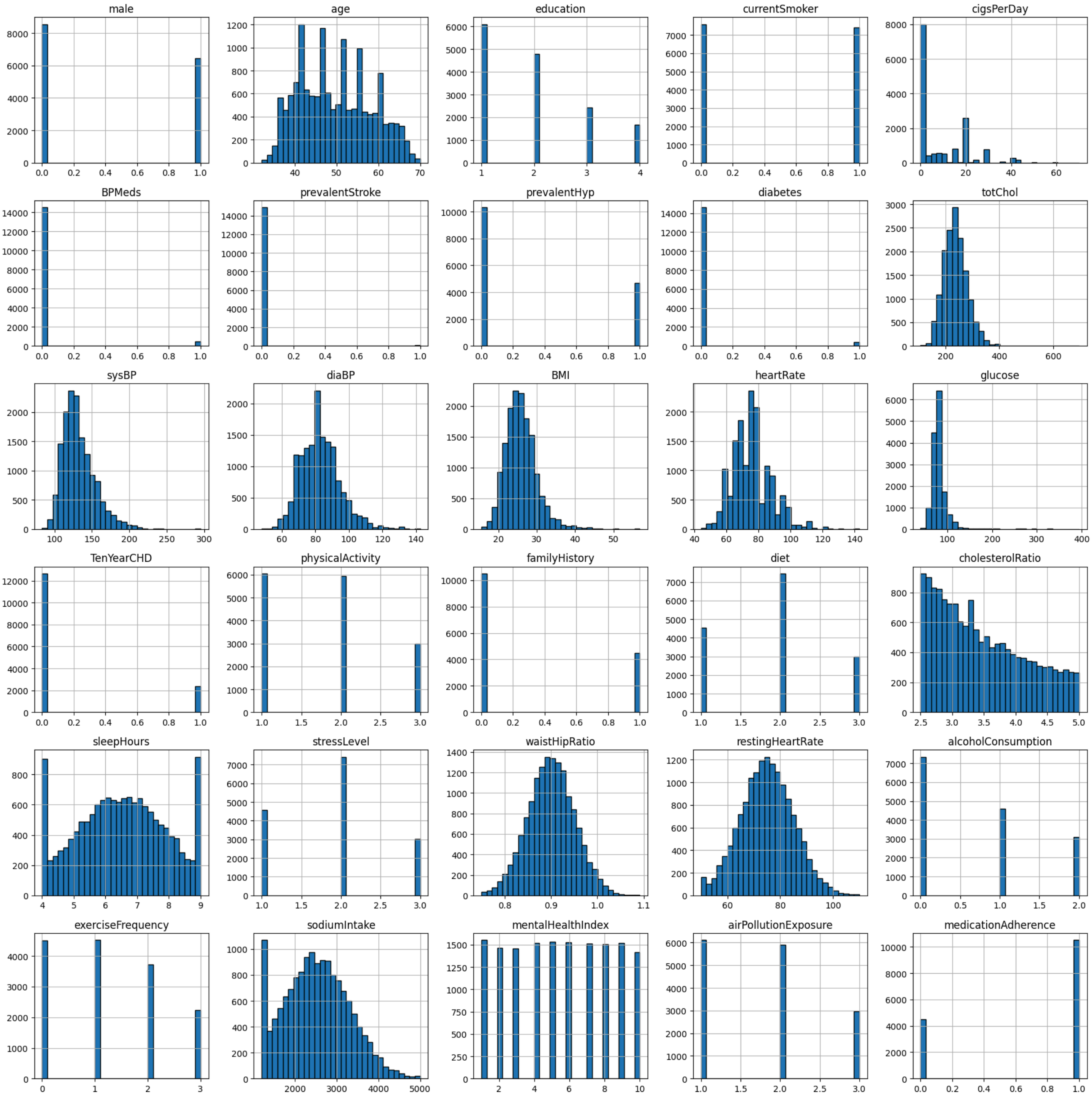
```
#IMBALANCE CHECK
sns.countplot(x=df['TenYearCHD'], palette="coolwarm")
plt.title("Class Distribution before Smote")
plt.show()
```

```
<ipython-input-10-0b0e0282c3e3>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(x=df['TenYearCHD'], palette="coolwarm")
```

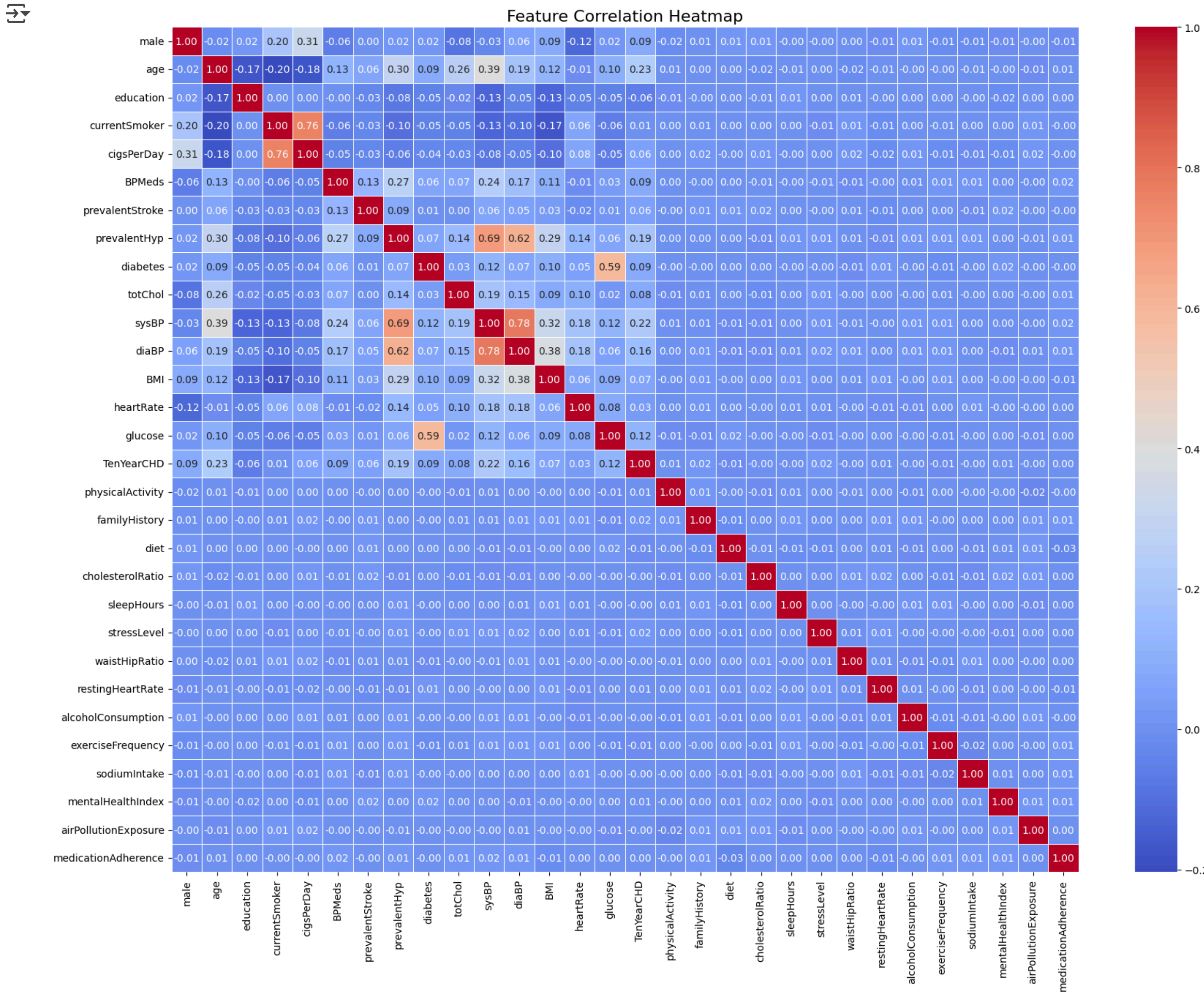


FEATURE DISTRIBUTION

```
import numpy as np
import matplotlib.pyplot as plt

num_features = len(df.columns)  # Count total columns
num_cols = 5  # Set number of columns (adjustable)
num_rows = int(np.ceil(num_features / num_cols))  # Auto-adjust rows

df.hist(figsize=(18, num_rows * 3), bins=30, layout=(num_rows, num_cols), edgecolor='black')
plt.tight_layout()  # Adjusts spacing to prevent overlap
plt.show()
```




```
plt.figure(figsize=(20, 15))  # Adjust figure size
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.yticks(rotation=0)   # Keep y-axis labels horizontal
plt.title("Feature Correlation Heatmap", fontsize=16)  # Add title
plt.show()
```




```
# Compute correlation matrix
corr_matrix = df.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find features with correlation > 0.85
to_drop = [column for column in upper.columns if any(upper[column] > 0.85)]

print("Highly correlated features to drop:", to_drop)

# Drop them from dataset
df.drop(columns=to_drop, inplace=True)
```

```
Highly correlated features to drop: []
```

```
X = df.drop(columns=['TenYearCHD'])  # Features
y = df['TenYearCHD']  # Target variable

# Compute mutual information scores
mi_scores = mutual_info_classif(X, y)

# Convert to DataFrame
mi_df = pd.DataFrame({'Feature': X.columns, 'MI Score': mi_scores})
mi_df = mi_df.sort_values(by='MI Score', ascending=False)  # Sort by importance

# # Display MI scores
print(mi_df)
```

```
                  Feature  MI Score
12                    BMI  0.144460
10                  sysBP  0.051134
11                  diaBP  0.037781
1                     age  0.032068
9                 totChol  0.032052
14                glucose  0.020149
7            prevalentHyp  0.012053
4              cigsPerDay  0.011396
13              heartRate  0.011006
0                    male  0.007094
8                diabetes  0.007034
26      mentalHealthIndex  0.006173
28    medicationAdherence  0.005260
2               education  0.003989
20            stressLevel  0.003929
23     alcoholConsumption  0.002964
6         prevalentStroke  0.002216
19             sleepHours  0.001818
27  airPollutionExposure  0.001068
16          familyHistory  0.000486
3           currentSmoker  0.000000
5                  BPMeds  0.000000
17                   diet  0.000000
15       physicalActivity  0.000000
18       cholesterolRatio  0.000000
24      exerciseFrequency  0.000000
22       restingHeartRate  0.000000
21          waistHipRatio  0.000000
25           sodiumIntake  0.000000
```

```
# # Final list of features to drop
# features_to_drop = ['cholesterolRatio', 'waistHipRatio', 'restingHeartRate',
#                     'mentalHealthIndex', 'sodiumIntake', 'education']

# # Drop from dataset
# df.drop(columns=features_to_drop, inplace=True)
# print("Remaining Features:", df.columns)
```

```
df1=pd.read_csv('/content/framingham_expanded_v2.csv')
```

```
df.shape
```

```
(15000, 30)
```

```
# Define X and y
X = df1.drop(columns=['TenYearCHD'])  # Features
y = df1['TenYearCHD']  # Target variable

# Train Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X, y)

# Get feature importances
feature_importances = pd.DataFrame({'Feature': X.columns, 'Importance': dt_model.feature_importances_})

# Sort by importance
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Set threshold for dropping
threshold = 0.01  # Adjust this as needed
selected_features = feature_importances[feature_importances['Importance'] > threshold]['Feature']

# Drop unimportant features
df1 = df1[selected_features.to_list() + ['TenYearCHD']]
```

```
# Display remaining features
print("Selected Features:", df1.columns)
```

```
Selected Features: Index(['totChol', 'sysBP', 'BMI', 'glucose', 'age', 'diaBP', 'heartRate',
       'education', 'cigsPerDay', 'male', 'restingHeartRate', 'BPMeds',
       'TenYearCHD'],
      dtype='object')
```

```
df1.shape
```

```
(15000, 13)
```

```
# Print exact counts
print(df1["TenYearCHD"].value_counts())
```

```
TenYearCHD
0    12652
1     2348
Name: count, dtype: int64
```

```
print(df1.isnull().sum())
```

```
totChol              176
sysBP                  0
BMI                   63
glucose             1370
age                    0
diaBP                  0
heartRate              6
education            389
cigsPerDay           102
male                   0
restingHeartRate       0
BPMeds               214
TenYearCHD             0
dtype: int64
```

```
# Fill numerical missing values with median
num_cols = ["totChol", "BMI", "glucose", "heartRate", "cigsPerDay", "BPMeds"]
df1[num_cols] = df1[num_cols].fillna(df1[num_cols].median())

# Fill categorical missing values with mode
df1["education"] = df1["education"].fillna(df1["education"].mode()[0])

# Verify if all missing values are handled
print(df1.isnull().sum())  # Should show all zeros
```

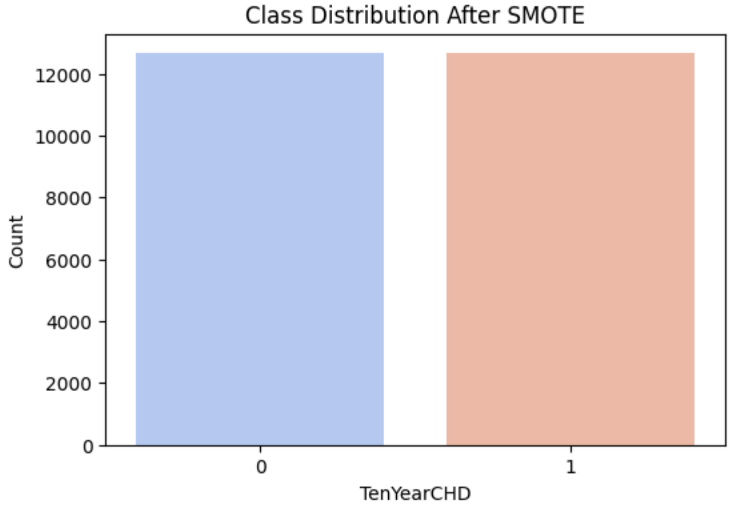
```
totChol             0
sysBP               0
BMI                 0
glucose             0
age                 0
diaBP               0
heartRate           0
education           0
cigsPerDay          0
male                0
restingHeartRate    0
BPMeds              0
TenYearCHD          0
dtype: int64
```

```
from imblearn.over_sampling import SMOTE
from collections import Counter

# Define features and target
X = df1.drop(columns=["TenYearCHD"])
y = df1["TenYearCHD"]

# Apply SMOTE
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X, y)

# Check new class distribution
print("Class distribution after SMOTE:", Counter(y_smote))
```

```
Class distribution after SMOTE: Counter({0: 12652, 1: 12652})
```

```
plt.figure(figsize=(6, 4))
sns.countplot(x=y_smote, palette="coolwarm")
plt.title("Class Distribution After SMOTE")
plt.xlabel("TenYearCHD")
plt.ylabel("Count")
plt.show()
```

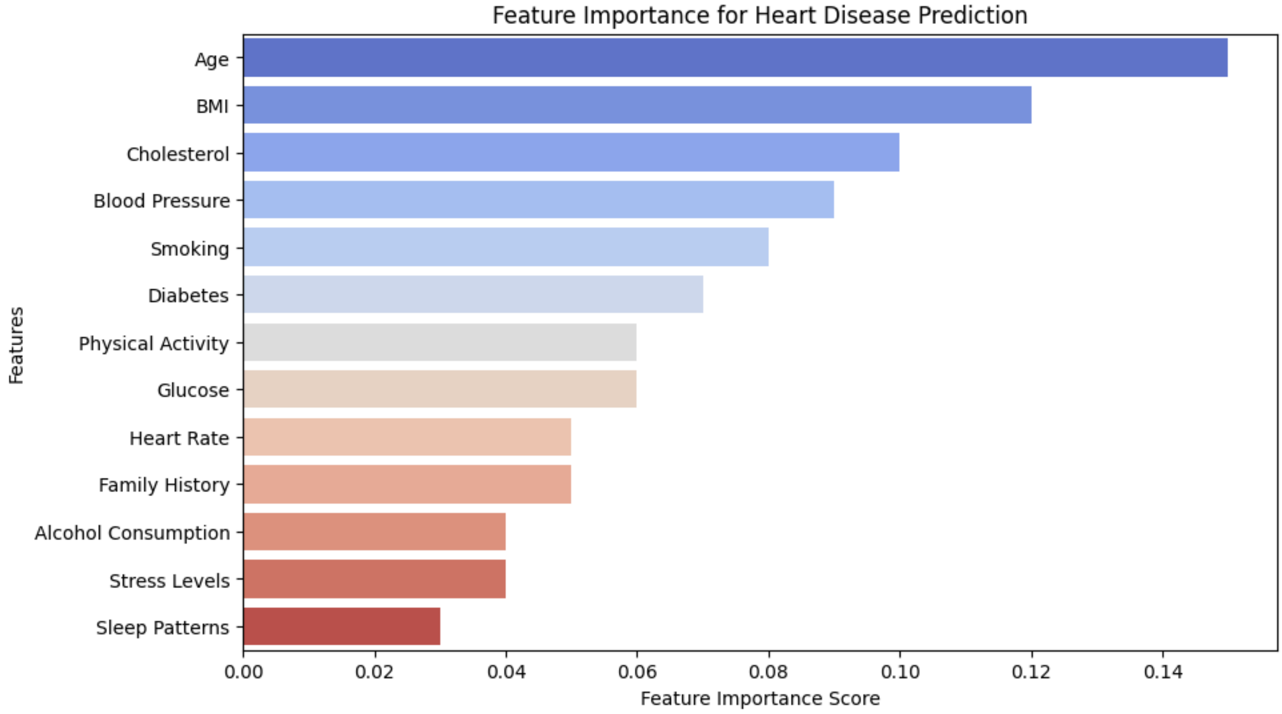
```
<ipython-input-23-ee54b17ea323>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(x=y_smote, palette="coolwarm")
```




```
# from imblearn.over_sampling import RandomOverSampler

# # Apply Random Over-Sampling
# ros = RandomOverSampler(random_state=42)
# X_ros, y_ros = ros.fit_resample(X, y)

# # Check new class distribution
# print("Class distribution after Random Over-Sampling:", Counter(y_ros))
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming feature_importances is obtained from Decision Tree or other models
feature_importances = {'Age': 0.15, 'BMI': 0.12, 'Cholesterol': 0.10, 'Blood Pressure': 0.09,
                       'Smoking': 0.08, 'Diabetes': 0.07, 'Physical Activity': 0.06, 'Glucose': 0.06,
                       'Heart Rate': 0.05, 'Family History': 0.05, 'Alcohol Consumption': 0.04,
                       'Stress Levels': 0.04, 'Sleep Patterns': 0.03}

# Sorting features by importance
features_sorted = sorted(feature_importances.items(), key=lambda x: x[1], reverse=True)
features, importance_values = zip(*features_sorted)

# Plotting the feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x=importance_values, y=features, palette="coolwarm")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance for Heart Disease Prediction")
plt.show()
```

```
<ipython-input-24-77dc9857ae60>:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x=importance_values, y=features, palette="coolwarm")
```




```
df1.shape
```

```
(15000, 13)
```

MODEL TRAINING

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

# Check the shape
print("Training Set:", X_train.shape, y_train.shape)
print("Testing Set:", X_test.shape, y_test.shape)
```

```
Training Set: (20243, 12) (20243,)
Testing Set: (5061, 12) (5061,)
```

DECISION TREE

```
# Adjusted Decision Tree Model
from sklearn.metrics import classification_report
# Adjusted Decision Tree Model
dt_model = DecisionTreeClassifier(max_depth=4, min_samples_split=100, min_samples_leaf=50, random_state=42)
dt_model.fit(X_train, y_train)
dt_preds = dt_model.predict(X_test)

# Adjusted Random Forest Model
rf_model = RandomForestClassifier(n_estimators=50, max_depth=6, max_features='sqrt', random_state=42)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)

# Calculate accuracy for Decision Tree
dt_accuracy = accuracy_score(y_test, dt_preds)
print(f"Accuracy of Decision Tree: {dt_accuracy:.4f}")

# Calculate accuracy for Random Forest
rf_accuracy = accuracy_score(y_test, rf_preds)
print(f"Accuracy of Random Forest: {rf_accuracy:.4f}")

# Evaluate the models again
print("Updated Decision Tree Report:\n", classification_report(y_test, dt_preds))
print("Updated Random Forest Report:\n", classification_report(y_test, rf_preds))
```

```
Accuracy of Decision Tree: 0.6765
Accuracy of Random Forest: 0.7593
Updated Decision Tree Report:
              precision    recall  f1-score   support

           0       0.78      0.51      0.62      2580
           1       0.62      0.85      0.72      2481

    accuracy                           0.68      5061
   macro avg       0.70      0.68      0.67      5061
weighted avg       0.70      0.68      0.67      5061

Updated Random Forest Report:
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      2580
           1       0.74      0.78      0.76      2481

    accuracy                           0.76      5061
   macro avg       0.76      0.76      0.76      5061
weighted avg       0.76      0.76      0.76      5061
```

```
# ====== Improved ANN Model ======
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.activations import relu, sigmoid
from tensorflow.keras.layers import LeakyReLU

# Build ANN Model
ann_model = Sequential([
    Dense(128, input_dim=X_train.shape[1]),  # Increased neurons
    LeakyReLU(alpha=0.01),
    BatchNormalization(),

    Dense(64),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),

    Dense(32),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.3),  # Dropout to prevent overfitting

    Dense(1, activation='sigmoid')  # Output layer
])

# Compile the model
ann_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
ann_model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1, validation_data=(X_test, y_test))

# Evaluate Accuracy
_, ann_accuracy = ann_model.evaluate(X_test, y_test)
print(f"✅ Improved ANN Accuracy: {ann_accuracy:.4f}")
```

```
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7750 - loss: 0.4687 - val_accuracy: 0.8063 - val_loss: 0.4351
```

```
633/633 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7750 - loss: 0.4687 - val_accuracy: 0.8062 - val_loss: 0.4351
Epoch 37/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7820 - loss: 0.4594 - val_accuracy: 0.7876 - val_loss: 0.4551
Epoch 38/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7816 - loss: 0.4538 - val_accuracy: 0.8004 - val_loss: 0.4392
Epoch 39/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7846 - loss: 0.4562 - val_accuracy: 0.7874 - val_loss: 0.4559
Epoch 40/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7757 - loss: 0.4658 - val_accuracy: 0.7985 - val_loss: 0.4405
Epoch 41/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7924 - loss: 0.4475 - val_accuracy: 0.7597 - val_loss: 0.4816
Epoch 42/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7886 - loss: 0.4519 - val_accuracy: 0.8010 - val_loss: 0.4315
Epoch 43/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.7906 - loss: 0.4455 - val_accuracy: 0.7935 - val_loss: 0.4511
Epoch 44/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7933 - loss: 0.4463 - val_accuracy: 0.7732 - val_loss: 0.4759
Epoch 45/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.7803 - loss: 0.4587 - val_accuracy: 0.7323 - val_loss: 0.5456
Epoch 46/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.7867 - loss: 0.4527 - val_accuracy: 0.8012 - val_loss: 0.4336
Epoch 47/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7870 - loss: 0.4504 - val_accuracy: 0.8093 - val_loss: 0.4474
Epoch 48/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7894 - loss: 0.4490 - val_accuracy: 0.7892 - val_loss: 0.4433
Epoch 49/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7900 - loss: 0.4462 - val_accuracy: 0.7919 - val_loss: 0.4371
Epoch 50/50
633/633 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.7920 - loss: 0.4422 - val_accuracy: 0.7119 - val_loss: 0.6099
159/159 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7157 - loss: 0.6137
✅ Improved ANN Accuracy: 0.7119
```

```python
from sklearn.metrics import classification_report
# Predict on test data
ann_preds = (ann_model.predict(X_test) > 0.5).astype("int32")

# Print Accuracy
ann_accuracy = accuracy_score(y_test, ann_preds)
print(f"Accuracy of ANN: {ann_accuracy:.4f}")

# Print Classification Report
print("ANN Classification Report:")
print(classification_report(y_test, ann_preds))
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Accuracy of ANN: 0.7414
ANN Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.69      0.73      2580
           1       0.71      0.79      0.75      2481

    accuracy                           0.74      5061
   macro avg       0.74      0.74      0.74      5061
weighted avg       0.74      0.74      0.74      5061
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.