Heart Disease Prediction using Machine Learning

Part 1: Load Data and Explore

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For modeling
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classification_report, roc_curve
```

```
# Load dataset
df = pd.read_csv('HeartDisease[1].csv')
print("First 5 rows of data:")
display(df.head())
```

```
First 5 rows of data:
```

	HeartDiseaseorAttack	HighBP	HighChol	BMI	Smoker	Diabetes	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	MentHlth	PhysHl
0	0.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	1.0	0.0	18.0	15
1	0.0	0.0	0.0	25.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0
2	0.0	1.0	1.0	28.0	0.0	0.0	0.0	1.0	0.0	0.0	30.0	30
3	0.0	1.0	0.0	27.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0
4	0.0	1.0	1.0	24.0	0.0	0.0	1.0	1.0	1.0	0.0	3.0	0

```
print(df.head())
```

```
   HeartDiseaseorAttack  HighBP  HighChol   BMI  Smoker  Diabetes  \
0                   0.0     1.0       1.0  40.0     1.0       0.0
1                   0.0     0.0       0.0  25.0     1.0       0.0
2                   0.0     1.0       1.0  28.0     0.0       0.0
3                   0.0     1.0       0.0  27.0     0.0       0.0
4                   0.0     1.0       1.0  24.0     0.0       0.0

   PhysActivity  Fruits  Veggies  HvyAlcoholConsump  MentHlth  PhysHlth  Sex  \
0           0.0     0.0      1.0                0.0      18.0      15.0  0.0
1           1.0     0.0      0.0                0.0       0.0       0.0  0.0
2           0.0     1.0      0.0                0.0      30.0      30.0  0.0
3           1.0     1.0      1.0                0.0       0.0       0.0  0.0
4           1.0     1.0      1.0                0.0       3.0       0.0  0.0

    Age  Education  Income
0   9.0        4.0     3.0
1   7.0        6.0     1.0
2   9.0        4.0     8.0
3  11.0        3.0     6.0
4  11.0        5.0     4.0
```

```
# Check missing values
print("\nMissing values:")
print(df.isnull().sum())
```

```

Missing values:
HeartDiseaseorAttack    0
HighBP                  0
HighChol                0
BMI                     0
Smoker                  1
Diabetes                1
PhysActivity            1
Fruits                  1
Veggies                 1
HvyAlcoholConsump       1
MentHlth                1
PhysHlth                1
Sex                     1
Age                     1
Education               1
Income                  1
dtype: int64
```

```
# Check target distribution
print("\nTarget class distribution:")
print(df['HeartDiseaseorAttack'].value_counts())
```

```
Target class distribution:
HeartDiseaseorAttack
0.0    14259
1.0     1698
Name: count, dtype: int64
```

This shows a class imbalance — only about 9.4% of the samples have heart disease. We'll handle this when building models.

```
print("\nData summary:")
display(df.describe())
```

Data summary:

	HeartDiseaseorAttack	HighBP	HighChol	BMI	Smoker	Diabetes	PhysActivity	Fruits	Veg
count	15957.000000	15957.000000	15957.000000	15957.000000	15956.000000	15956.000000	15956.000000	15956.000000	15956.00
mean	0.106411	0.467757	0.449583	28.502601	0.447543	0.340248	0.739095	0.608862	0.80
std	0.308373	0.498975	0.497467	6.372932	0.497256	0.738340	0.439142	0.488021	0.39
min	0.000000	0.000000	0.000000	14.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	24.000000	0.000000	0.000000	0.000000	0.000000	1.00
50%	0.000000	0.000000	0.000000	27.000000	0.000000	0.000000	1.000000	1.000000	1.00
75%	0.000000	1.000000	1.000000	32.000000	1.000000	0.000000	1.000000	1.000000	1.00
max	1.000000	1.000000	1.000000	85.000000	1.000000	2.000000	1.000000	1.000000	1.00

Part 2: Preprocessing

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Separate features and target
X = df.drop('HeartDiseaseorAttack', axis=1)
y = df['HeartDiseaseorAttack']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Impute missing values using mean strategy
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)
```

Dataset Split :

Training Set: 202,944 samples

- HeartDiseaseorAttack = 1: 19,114 (9.4%)
- HeartDiseaseorAttack = 0: 183,830 (90.6%)

Testing Set: 50,736 samples

- HeartDiseaseorAttack = 1: 4,779 (9.4%)
- HeartDiseaseorAttack = 0: 45,957 (90.6%)

This confirms that class balance is preserved via stratify=y during the split.

Part 3: Model Building and Comparison

```
# Import models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import xgboost as xgb

# Create a dictionary of models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "SVM": SVC(probability=True),
    "XGBoost": xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

# Evaluate each model
results = {}

for name, model in models.items():
    print(f"\nTraining: {name}")
    model.fit(X_train_scaled, y_train)  # ✅ Corrected from X_train_imputed to X_train_scaled
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:, 1]

    acc = accuracy_score(y_test, y_pred)
    roc = roc_auc_score(y_test, y_prob)
    print("Accuracy:", acc)
    print("ROC AUC:", roc)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

    results[name] = {'model': model, 'accuracy': acc, 'roc_auc': roc}
```

```
Training: Logistic Regression
Accuracy: 0.893796992481203
ROC AUC: 0.7886148007590134
Classification Report:
               precision    recall  f1-score   support

         0.0       0.90      0.99      0.94      2852
         1.0       0.51      0.06      0.11       340

    accuracy                           0.89      3192
   macro avg       0.71      0.53      0.53      3192
weighted avg       0.86      0.89      0.86      3192

Confusion Matrix:
 [[2831   21]
 [ 318   22]]

Training: Random Forest
Accuracy: 0.8906641604010025
ROC AUC: 0.7649858716277536
Classification Report:
               precision    recall  f1-score   support

         0.0       0.90      0.99      0.94      2852
         1.0       0.41      0.06      0.11       340

    accuracy                           0.89      3192
   macro avg       0.66      0.53      0.52      3192
weighted avg       0.85      0.89      0.85      3192

Confusion Matrix:
 [[2822   30]
 [ 319   21]]

Training: SVM
Accuracy: 0.893796992481203
ROC AUC: 0.6344639468690703
Classification Report:
               precision    recall  f1-score   support

         0.0       0.89      1.00      0.94      2852
         1.0       0.67      0.01      0.01       340

    accuracy                           0.89      3192
   macro avg       0.78      0.50      0.48      3192
weighted avg       0.87      0.89      0.84      3192

Confusion Matrix:
 [[2851    1]
```

```
 [ 338    2]]

Training: XGBoost
Accuracy: 0.8793859649122807
ROC AUC: 0.7472887963039353
Classification Report:
              precision    recall  f1-score   support
```

```
# Plot ROC curves
plt.figure(figsize=(10, 7))
for name, result in results.items():
    model = result['model']
    y_prob = model.predict_proba(X_test_scaled)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {result['roc_auc']:.2f})")

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.grid(True)
plt.show()
```

Part 4: Feature Importance & SHAP

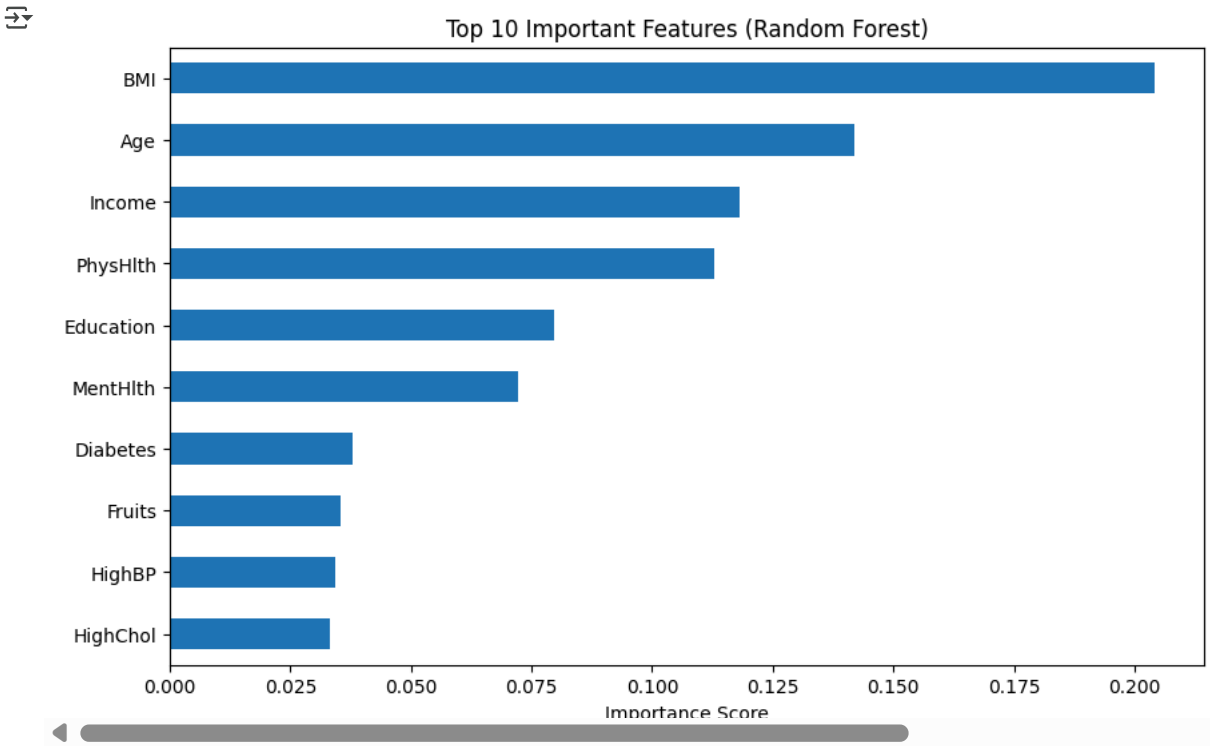
```
# Step 1: Feature Importance from Random Forest
# Get feature names
feature_names = X.columns

# Get trained Random Forest model
rf_model = results["Random Forest"]["model"]

# Get feature importances
importances = rf_model.feature_importances_

# Plot top 10 important features
important_features = pd.Series(importances, index=feature_names).sort_values(ascending=False)

plt.figure(figsize=(10, 6))
important_features.head(10).plot(kind='barh')
plt.title("Top 10 Important Features (Random Forest)")
plt.gca().invert_yaxis()  # highest at top
plt.xlabel("Importance Score")
plt.show()
```



```
# Step 2: Install SHAP
# Run this only once to install SHAP
!pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages (0.47.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from shap) (1.15.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from shap) (4.14.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->shap) (1.17
```

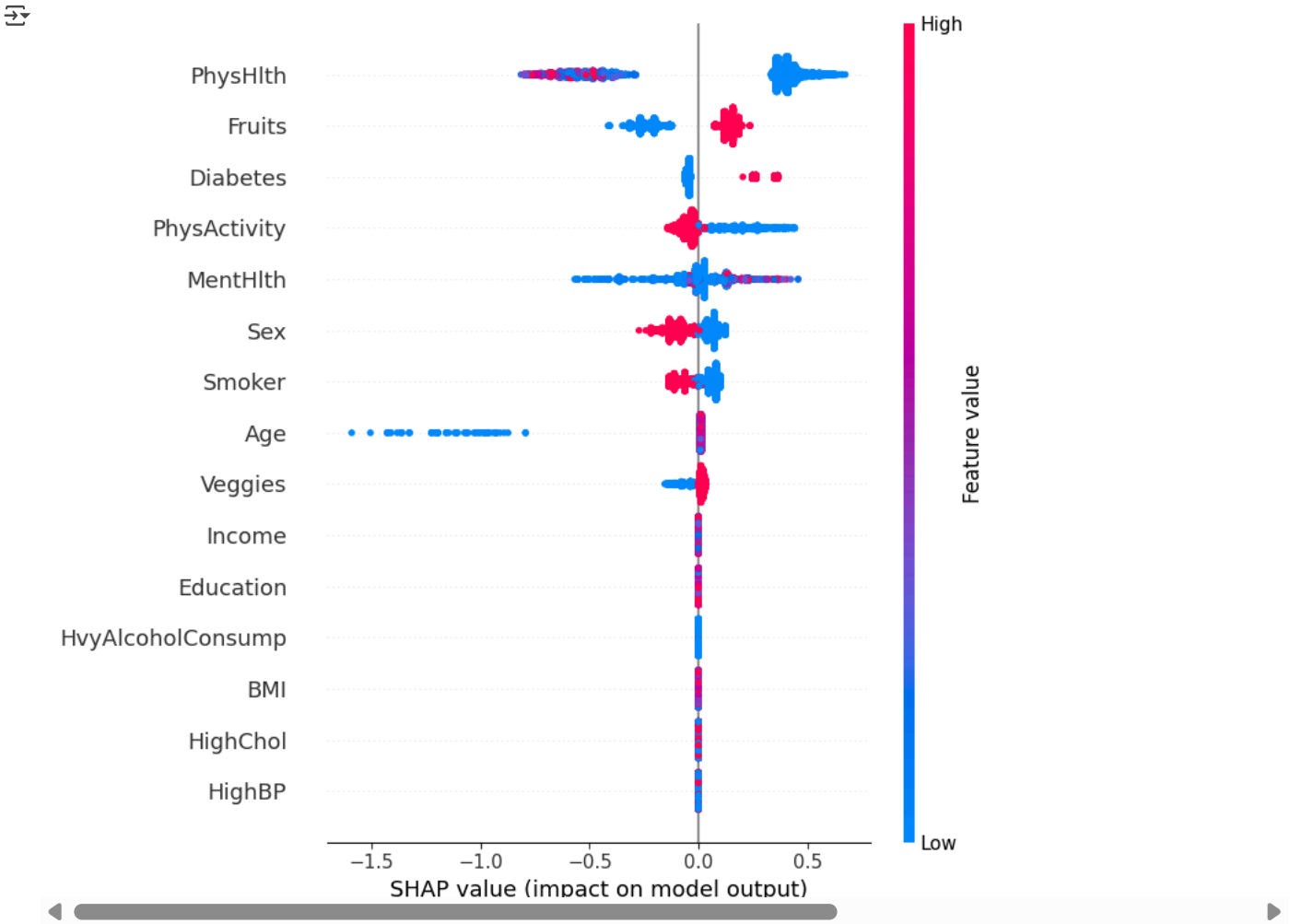
```
import shap

# Get XGBoost model
xgb_model = results["XGBoost"]["model"]

# Create SHAP explainer (auto type detection)
explainer = shap.Explainer(xgb_model, X_train_imputed)

# Calculate SHAP values for test data
shap_values = explainer(X_test_imputed)

# Summary Plot – this shows most important features and how they affect prediction
shap.summary_plot(shap_values, X_test_imputed, feature_names=feature_names)
```



Top features in predicting heart disease

Red dots = pushing prediction toward disease

Blue dots = pushing prediction toward no disease

Spread = impact of that feature across many patients

Part 5: Final Model Comparison & Save Best Model

```
# Compare all model scores
import pandas as pd

# Create a DataFrame from the results dictionary
model_scores = pd.DataFrame({
    model: {
        'Accuracy': round(metrics['accuracy'], 3),
        'ROC AUC': round(metrics['roc_auc'], 3)
    }
    for model, metrics in results.items()
}).T

# Sort by ROC AUC
model_scores = model_scores.sort_values(by='ROC AUC', ascending=False)
print("Model Performance Comparison:\n")
print(model_scores)
```

```
Model Performance Comparison:

                     Accuracy  ROC AUC
Logistic Regression     0.894    0.789
Random Forest           0.891    0.765
XGBoost                 0.879    0.747
SVM                     0.894    0.634
```

Step 2: Pick the Best Model

```
# Get the name of the best model (highest ROC AUC)
best_model_name = model_scores.index[0]
best_model = results[best_model_name]['model']

print(f"\nBest Performing Model: {best_model_name}")
```

```
Best Performing Model: Logistic Regression
```

Step 3: Save the Best Model Using joblib

```
import joblib

# Save the model
joblib.dump(best_model, f"{best_model_name.replace(' ', '_')}_model.pkl")
print(f"\nModel saved as {best_model_name.replace(' ', '_')}_model.pkl")
```

```
Model saved as Logistic_Regression_model.pkl
```