📌 Project Title: Predicting Heart Attacks Using Machine Learning

Overview:

This project aims to predict the likelihood of a heart attack based on patient data using machine learning techniques. The goal is to identify patterns in medical features that may indicate increased risk, thereby supporting early intervention and prevention.

Key Steps:

- Exploratory Data Analysis (EDA)
- Handling missing values and data imbalance
- Feature engineering and encoding
- Training models (Random Forest, XGBoost, etc.)
- Evaluating performance using classification metrics (Precision, Recall, ROC-AUC)

In [1]:
```
from sklearn.model_selection import train_test_split
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/personal-key-indicators-of-heart-disease/2020/heart_2
020_cleaned.csv
/kaggle/input/personal-key-indicators-of-heart-disease/2022/heart_2
022_with_nans.csv
/kaggle/input/personal-key-indicators-of-heart-disease/2022/heart_2
022_no_nans.csv
```

In [2]:
```
data = pd.read_csv("/kaggle/input/personal-key-indicators-of-heart-disease/2022/heart_2022_with_nans.csv")
print(" , ".join(data.columns))
```

```
State , Sex , GeneralHealth , PhysicalHealthDays , MentalHealthDays , LastCheckupTime , PhysicalActivities , SleepHours , RemovedTeeth , HadHeartAttack , HadAngina , HadStroke , HadAsthma , HadSkinCancer , HadCOPD , HadDepressiveDisorder , HadKidneyDisease , HadArthritis , HadDiabetes , DeafOrHardOfHearing , BlindOrVisionDifficulty , DifficultyConcentrating , DifficultyWalking , DifficultyDressingBathing , DifficultyErrands , SmokerStatus , ECigaretteUsage , ChestScan , RaceEthnicityCategory , AgeCategory , HeightInMeters , WeightInKilograms , BMI , AlcoholDrinkers , HIVTesting , FluVaxLast12 , PneumoVaxEver , TetanusLast10Tdap , HighRiskLastYear , CovidPos
```

Checking Null Values And Delete Unnecessary Columns

Step 1: The line `data.isnull().sum().sort_values()` checks for any missing values in the dataset by calculating the sum of null values for each column and sorting them in ascending order. This helps in identifying which columns have missing data and need to be handled.

Step 2: The line `data = data.drop('State', axis=1)` removes the *'State'* column from the dataset. This step is performed because the 'State' feature may not provide valuable information for model predictions and could potentially introduce noise.

In [3]:

```
data = data.drop('State', axis=1)
data = data.dropna(subset=['HadHeartAttack'])
data.isnull().sum().sort_values()
```

Out[3]:

Sex	0
HadHeartAttack	0
HadDiabetes	813
PhysicalActivities	972
HadStroke	1070
GeneralHealth	1095
HadAsthma	1437
HadKidneyDisease	1614
HadCOPD	1838
HadArthritis	2313
HadDepressiveDisorder	2421
HadSkinCancer	2764
HadAngina	3588
SleepHours	5196
LastCheckupTime	8041
MentalHealthDays	8792
AgeCategory	8807
PhysicalHealthDays	10597
RemovedTeeth	11010
RaceEthnicityCategory	13698
DeafOrHardOfHearing	20278
BlindOrVisionDifficulty	21190
DifficultyDressingBathing	23513
DifficultyWalking	23604
DifficultyConcentrating	23795
DifficultyErrands	25231
HeightInMeters	28141
SmokerStatus	34941
ECigaretteUsage	35143
WeightInKilograms	41466
AlcoholDrinkers	45937
FluVaxLast12	46477
BMI	48110
HighRiskLastYear	49985
CovidPos	50127
ChestScan	55288
HIVTesting	65310
PneumoVaxEver	76210
TetanusLast10Tdap	81574

dtype: int64

In [4]:
```
pd.set_option('display.max_columns', None)
data.head()
```

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

Out[4]:

	Sex	GeneralHealth	PhysicalHealthDays	MentalHealthDays	LastCheckupTime	PhysicalActi
0	Female	Very good	0.0	0.0	Within past year (anytime less than 12 months ...	No
1	Female	Excellent	0.0	0.0	NaN	No
2	Female	Very good	2.0	3.0	Within past year (anytime less than 12 months ...	Yes
3	Female	Excellent	0.0	0.0	Within past year (anytime less than 12 months ...	Yes
4	Female	Fair	2.0	0.0	Within past year (anytime less than 12 months ...	Yes

Handling Missing Values

In this step, we address missing values in the dataset. For categorical columns, missing values are filled with the most frequent value (mode), and any remaining nulls are set to 'Unknown'. For numerical columns, missing values are filled based on their distribution: if the data is normally distributed, the median is used, otherwise, the mean is used. This ensures that the dataset is clean and ready for modeling.

In [5]:

```
categorical_cols = data.select_dtypes(include=["object","category"]).columns
numerical_cols = data.select_dtypes(include=['float64']).columns
mode_fill_cols = ['HighRiskLastYear', 'CovidPos', 'ChestScan', 'HIVTesting' , 'PneumoVaxEver', 'TetanusLast10Tdap']  # Listeyi oluşturduktan sonra doldurulacaklar
for cols in mode_fill_cols:
    data[cols] = data[cols].fillna(data[cols].mode()[0])

for cols in categorical_cols:
    data[cols] = data[cols].fillna(data[cols].mode()[0])
    data[cols] = data[cols].fillna('Unknown')

for cols in numerical_cols:
    if(-0.5 < data[cols].skew() < 0.5):
        data[cols] = data[cols].fillna(data[cols].median())
    else:
        data[cols] = data[cols].fillna(data[cols].mean())
```

Encoding Categorical Features

In this step, we apply encoding techniques to transform categorical variables into numerical values for machine learning. Some columns, such as 'GeneralHealth', 'SmokerStatus', 'ECigaretteUsage', 'AgeCategory', and 'RemovedTeeth', are mapped to predefined numerical scales based on their order. For other categorical variables like gender, physical activity status, health conditions, and test results, we use one-hot encoding to convert them into binary columns, making them suitable for modeling. This ensures that all categorical data is properly formatted for the machine learning algorithms.

In [6]:

```
from sklearn.preprocessing import LabelEncoder
health_order = {
    'Excellent':4,
    'Very Good':3,
    'Good':2,
    'Fair':1,
    'Poor':0,
}
Smoke_order = {
    'Current smoker - now smokes every day':3,
    'Current smoker - now smokes some days':2,
    'Former smoker':1,
    'Never smoked':0,
}
ESig_order = {
    'Use them every day':3,
    'Use them some days':2,
    'Not at all (right now)':1,
    'Never used e-cigarettes in my entire life':0,
}
Age_order = {
'Age 80 or older':12,
'Age 55 to 59':7,
'Age 40 to 44':4,
'Age 75 to 79':11,
'Age 70 to 74':10,
'Age 65 to 69':9,
'Age 60 to 64':8,
'Age 50 to 54':6,
'Age 45 to 49':5,
'Age 35 to 39':3,
'Age 30 to 34':2,
'Age 25 to 29':1,
'Age 18 to 24':0
}
Teeth_order = {
'All':3,
'6 or more, but not all':2,
'1 to 5':1,
'None of them':0,
}
data['RemovedTeeth'] = data['RemovedTeeth'].map(Teeth_order)
data['GeneralHealth'] = data['GeneralHealth'].map(health_order)
```

```
data['SmokerStatus'] = data['SmokerStatus'].map(Smoke_order)
data['ECigaretteUsage'] = data['ECigaretteUsage'].map(ESig_order)
data['AgeCategory'] = data['AgeCategory'].map(Age_order)
Label_Encoders = ['GeneralHealth' , 'SmokerStatus' , 'ECigaretteUsage',
'AgeCategory','RemovedTeeth']

label_encoder = LabelEncoder()
for col in Label_Encoders:
    data[col] = label_encoder.fit_transform(data[col])

One_Hot_Encoders = ['Sex','PhysicalActivities','LastCheckupTime','HadAn
gina' , 'HadStroke' , 'HadAsthma' , 'HadSkinCancer' , 'HadCOPD' , 'HadD
epressiveDisorder','HadKidneyDisease', 'HadArthritis','HadDiabetes' ,
'DeafOrHardOfHearing', 'BlindOrVisionDifficulty', 'DifficultyConcentrat
ing', 'DifficultyWalking','DifficultyDressingBathing', 'DifficultyErran
ds','ChestScan', 'RaceEthnicityCategory', 'AlcoholDrinkers', 'HIVTestin
g', 'FluVaxLast12', 'PneumoVaxEver','TetanusLast10Tdap', 'HighRiskLastY
ear', 'CovidPos']
for cols in One_Hot_Encoders:
    data = pd.get_dummies(data, columns=[cols], drop_first=True)
```

Standardizing Numerical Features

In this step, we standardize the numerical features using the `StandardScaler`. Standardization is essential for machine learning models that are sensitive to the scale of the data, such as logistic regression or k-nearest neighbors. By scaling the numerical features, we ensure that they all have a mean of 0 and a standard deviation of 1, which improves model performance and convergence speed.

In [7]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])
```

Preparing Data for Model Training

In this step, we prepare the data for model training:

- The target variable `y` is defined as the `HadHeartAttack` column, which we aim to predict.
- The features `X` are the rest of the columns, excluding the target variable.
- The dataset is split into training and test sets using `train_test_split` with a ratio of 80% for training and 20% for testing. Stratification is applied to maintain the same distribution of target classes in both sets.
- `LabelEncoder` is used to convert the target variable into numerical format for the model.

In [8]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

# Target column
y = data['HadHeartAttack']

# Features
X = data.drop(columns=['HadHeartAttack'])

# Split into training and test set (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
2, random_state=42,stratify=y)

le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

Handling Imbalanced Data with SMOTE

To handle the class imbalance in the dataset, we apply the `SMOTE` (Synthetic Minority Over-sampling Technique) algorithm, which generates synthetic examples for the minority class to balance the dataset.

- `SMOTE` is imported from the `imbalanced-learn` library, specifically targeting the training data `X_train` and the target labels `y_train`.
- We use the `fit_resample` method to create a resampled dataset with more balanced class distribution in `X_resampled` and `y_resampled`.

In [9]:

```
!pip install imbalanced-learn==0.12.2
from imblearn.over_sampling import SMOTE

# SMOTE'yi veri setine uygulamak
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

```
Collecting imbalanced-learn==0.12.2
  Downloading imbalanced_learn-0.12.2-py3-none-any.whl.metadata (8.
2 kB)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/pyth
on3.11/dist-packages (from imbalanced-learn==0.12.2) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/pytho
n3.11/dist-packages (from imbalanced-learn==0.12.2) (1.15.2)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/li
b/python3.11/dist-packages (from imbalanced-learn==0.12.2) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/pyth
on3.11/dist-packages (from imbalanced-learn==0.12.2) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/l
ib/python3.11/dist-packages (from imbalanced-learn==0.12.2) (3.6.0)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.1
1/dist-packages (from numpy>=1.17.3->imbalanced-learn==0.12.2) (1.
3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python
3.11/dist-packages (from numpy>=1.17.3->imbalanced-learn==0.12.2)
(1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.
11/dist-packages (from numpy>=1.17.3->imbalanced-learn==0.12.2) (0.
1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dis
t-packages (from numpy>=1.17.3->imbalanced-learn==0.12.2) (2025.1.
0)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.11/
dist-packages (from numpy>=1.17.3->imbalanced-learn==0.12.2) (2022.
1.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python
3.11/dist-packages (from numpy>=1.17.3->imbalanced-learn==0.12.2)
(2.4.1)
Requirement already satisfied: intel-openmp<2026,>=2024 in /usr/loc
al/lib/python3.11/dist-packages (from mkl->numpy>=1.17.3->imbalance
d-learn==0.12.2) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python
3.11/dist-packages (from mkl->numpy>=1.17.3->imbalanced-learn==0.1
2.2) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python
3.11/dist-packages (from tbb==2022.*->mkl->numpy>=1.17.3->imbalance
d-learn==0.12.2) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/li
b/python3.11/dist-packages (from mkl_umath->numpy>=1.17.3->imbalanc
ed-learn==0.12.2) (2024.2.0)
```

```
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /us
r/local/lib/python3.11/dist-packages (from intel-openmp<2026,>=2024
->mkl->numpy>=1.17.3->imbalanced-learn==0.12.2) (2024.2.0)
Downloading imbalanced_learn-0.12.2-py3-none-any.whl (257 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 258.0/258.0 kB 4.5 MB/s eta
0:00:00
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.13.0
    Uninstalling imbalanced-learn-0.13.0:
      Successfully uninstalled imbalanced-learn-0.13.0
Successfully installed imbalanced-learn-0.12.2
```

Model Evaluation with XGBoost

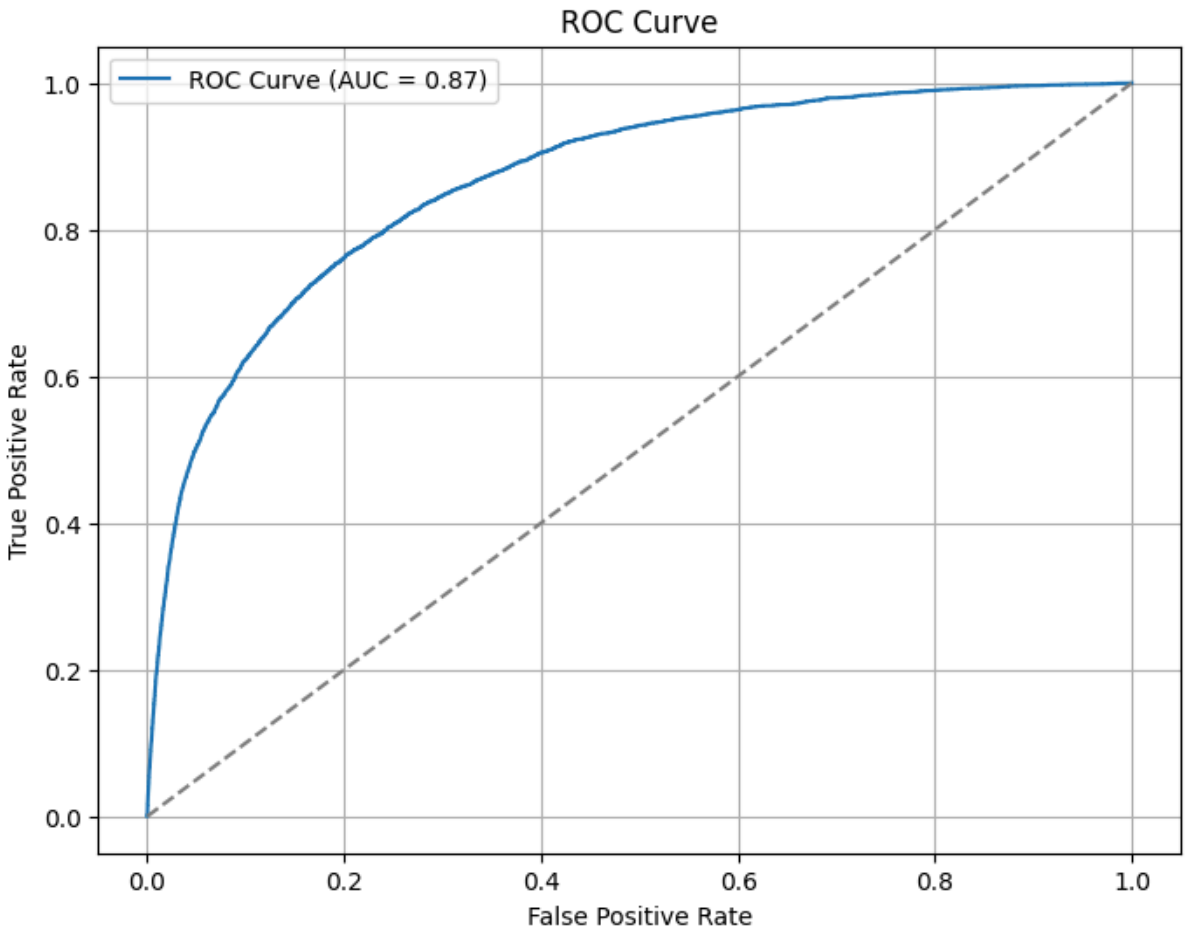
In this step, we train an XGBoost classifier and evaluate its performance using various metrics.

- The `XGBClassifier` from `xgboost` is used to create the model, with `mlogloss` as the evaluation metric.
- We train the model on the resampled training data `X_resampled` and `y_resampled`.
- The model's predictions are made using the `predict` method for class labels and `predict_proba` to get probabilities for calculating the ROC-AUC score.
- The performance of the model is evaluated using:
  - **Classification Report** that includes precision, recall, f1-score, and support for each class.
  - **ROC-AUC Score** to assess the model's ability to discriminate between the classes.
- We also plot the **ROC Curve** to visualize the model's performance across various thresholds.

In [10]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import xgboost as xgb

# Create the Model
# XGBoost Model
xg_clf = xgb.XGBClassifier(eval_metric='mlogloss', use_label_encoder=Fa
lse)
xg_clf.fit(X_resampled, y_resampled)

# Tahmin ve değerlendirme
y_pred = xg_clf.predict(X_test)
y_proba = xg_clf.predict_proba(X_test)[:, 1]

# Print the results
print("Classification Report:\n", classification_report(y_test, y_pre
d))

# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_proba)
print("ROC-AUC Skoru:", roc_auc)

# Drawing ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```

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

           0       0.96      0.98      0.97     83392
           1       0.48      0.36      0.41      5022

    accuracy                           0.94     88414
   macro avg       0.72      0.67      0.69     88414
weighted avg       0.93      0.94      0.94     88414

ROC-AUC Skoru: 0.8660508270994424
```



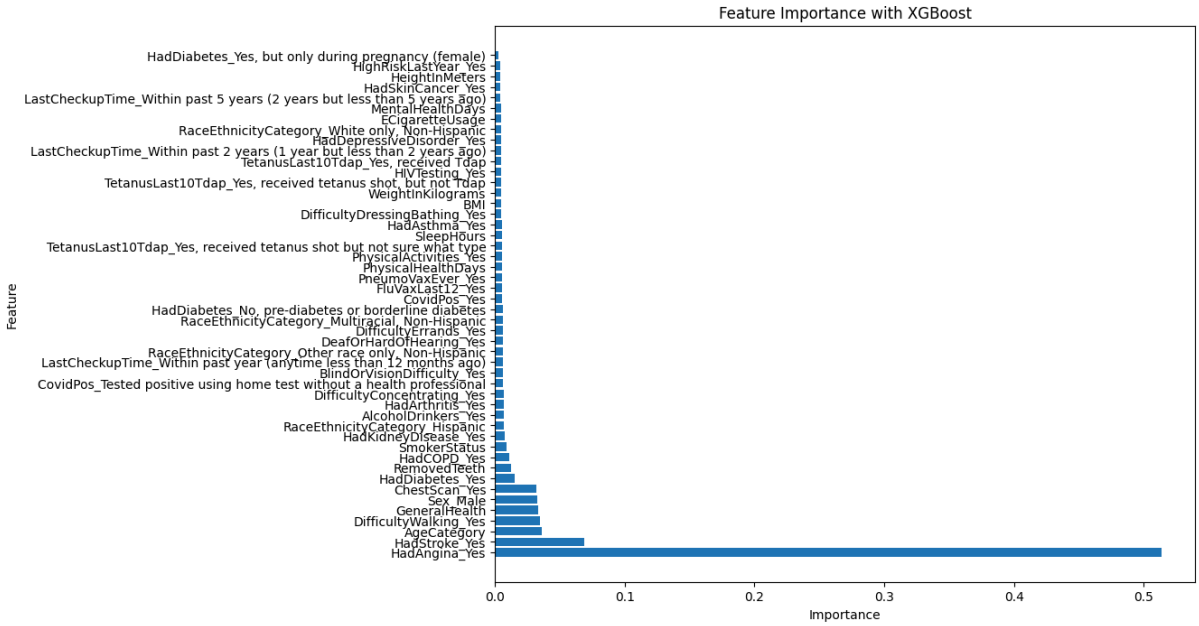
Feature Importance with XGBoost

In this step, we evaluate the importance of each feature in the XGBoost model.

- The `XGBClassifier` is used to fit the model on the training data `X_train` and `y_train`.
- After training, we extract the feature importances using the `feature_importances_` attribute of the model.
- A DataFrame is created containing the features and their corresponding importance values.
- The features are sorted in descending order based on their importance.
- A horizontal bar plot is generated to visualize the importance of each feature. The longer the bar, the more important the feature is in predicting the target variable.

In [11]:

```
import xgboost as xgb
import pandas as pd
import matplotlib.pyplot as plt

xg_clf = xgb.XGBClassifier(eval_metric='mlogloss', use_label_encoder=False)
xg_clf.fit(X_train, y_train)

importances = xg_clf.feature_importances_

feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance': importances})
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 8))
plt.barh(feature_importance['Feature'], feature_importance['Importance'])
plt.title('Feature Importance with XGBoost')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Conclusion

In this project, I created a model to predict the likelihood of someone having a heart attack based on their health data. By analyzing medical history, lifestyle, and other health factors, the model can help identify people who may be at risk. 💔

I used a powerful machine learning technique called **XGBoost**, and the model gave a **93% accuracy** in predicting heart attack risk. This means the model is pretty good at identifying those who need attention. The model works well for both healthy individuals and those who may be at higher risk. 📊

I also applied a method called **SMOTE** to help improve the model's accuracy by balancing the data. While the current model performs well, there's always room for improvement—like adding more health information or trying other methods in the future. 🔧

This model could be very useful for healthcare providers in identifying high-risk patients early on, allowing them to take preventive actions. 🏥