In [1]:
```
import pandas as pd
!pip install imblearn
```

```
Requirement already satisfied: imblearn in /opt/anaconda3/lib/python3.12/site-p
ackages (0.0)
Requirement already satisfied: imbalanced-learn in /opt/anaconda3/lib/python3.1
2/site-packages (from imblearn) (0.12.3)
Requirement already satisfied: numpy>=1.17.3 in /opt/anaconda3/lib/python3.12/s
ite-packages (from imbalanced-learn->imblearn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /opt/anaconda3/lib/python3.12/si
te-packages (from imbalanced-learn->imblearn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /opt/anaconda3/lib/python
3.12/site-packages (from imbalanced-learn->imblearn) (1.5.1)
Requirement already satisfied: joblib>=1.1.1 in /opt/anaconda3/lib/python3.12/s
ite-packages (from imbalanced-learn->imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/pytho
n3.12/site-packages (from imbalanced-learn->imblearn) (3.5.0)
```

In [2]:
```
heart=pd.read_csv("/Users/ronak/Library/Containers/com.microsoft.Excel/Data/Do
```

In [3]:
```
heart.isnull().sum()
```

Out[3]:
```
General_Health                  0
Checkup                         0
Exercise                        0
Heart_Disease                   0
Skin_Cancer                     0
Other_Cancer                    0
Depression                      0
Diabetes                        0
Arthritis                       0
Sex                             0
Age_Category                    0
Height_(cm)                     0
Weight_(kg)                     0
BMI                             0
Smoking_History                 0
Alcohol_Consumption             0
Fruit_Consumption               0
Green_Vegetables_Consumption    0
FriedPotato_Consumption         0
dtype: int64
```

In [4]:
```
# Convering the column names into lower case and replacing the space with an u
heart.columns = heart.columns.str.lower().str.replace(" ", "_")

#Changing the name of a big column

heart.rename(columns = {'height_(cm)' : 'height', 'weight_(kg)' : 'weight', 'g
```

In [5]:
```
heart['checkup'] = heart['checkup'].replace('Within the past 2 years', 'Past 2
heart['checkup'] = heart['checkup'].replace('Within the past year', 'Past 1 ye
heart['checkup'] = heart['checkup'].replace('Within the past 5 years', 'Past 5
```

```python
heart['checkup'] = heart['checkup'].replace('5 or more years ago', 'More than


heart['diabetes'] = heart['diabetes'].replace('No, pre-diabetes or borderline
heart['diabetes'] = heart['diabetes'].replace('Yes, but female told only durir

heart['age_category'] = heart['age_category'].replace('18-24', 'Young')
heart['age_category'] = heart['age_category'].replace('25-29', 'Adult')
heart['age_category'] = heart['age_category'].replace('30-34', 'Adult')
heart['age_category'] = heart['age_category'].replace('35-39', 'Adult')
heart['age_category'] = heart['age_category'].replace('40-44', 'Mid-Aged')
heart['age_category'] = heart['age_category'].replace('45-49', 'Mid-Aged')
heart['age_category'] = heart['age_category'].replace('50-54', 'Mid-Aged')
heart['age_category'] = heart['age_category'].replace('55-59', 'Senior-Adult')
heart['age_category'] = heart['age_category'].replace('60-64', 'Senior-Adult')
heart['age_category'] = heart['age_category'].replace('65-69', 'Elderly')
heart['age_category'] = heart['age_category'].replace('70-74', 'Elderly')
heart['age_category'] = heart['age_category'].replace('75-79', 'Elderly')
heart['age_category'] = heart['age_category'].replace('80+', 'Elderly')
```

In [6]:
```python
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308854 entries, 0 to 308853
Data columns (total 19 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   general_health          308854 non-null  object 
 1   checkup                 308854 non-null  object 
 2   exercise                308854 non-null  object 
 3   heart_disease           308854 non-null  object 
 4   skin_cancer             308854 non-null  object 
 5   other_cancer            308854 non-null  object 
 6   depression              308854 non-null  object 
 7   diabetes                308854 non-null  object 
 8   arthritis               308854 non-null  object 
 9   sex                     308854 non-null  object 
 10  age_category            308854 non-null  object 
 11  height                  308854 non-null  float64
 12  weight                  308854 non-null  float64
 13  bmi                     308854 non-null  float64
 14  smoking_history         308854 non-null  object 
 15  alcohol_consumption     308854 non-null  float64
 16  fruit_consumption       308854 non-null  float64
 17  vegetables_consumption  308854 non-null  float64
 18  potato_consumption      308854 non-null  float64
dtypes: float64(7), object(12)
memory usage: 44.8+ MB
```

In [7]:
```python
col = ['alcohol_consumption', 'fruit_consumption', 'vegetables_consumption', '

for i in col:
    heart[i] = heart[i].astype(int)
```

In [8]:

```
# Define BMI ranges and labels for each group
bmi_bins = [12.02, 18.3, 26.85, 31.58, 37.8, 100]
bmi_labels = ['Underweight', 'Normal weight', 'Overweight', 'Obese I', 'Obese
heart['bmi_group'] = pd.cut(heart['bmi'], bins=bmi_bins, labels=bmi_labels, ri
```

In [9]:

```
column_to_move = heart.pop('bmi_group')
heart.insert(14, 'bmi_group', column_to_move)
```

In [10]:

```
heart['bmi_group'] = heart['bmi_group'].astype('object')
```

In [11]:

```
# Import the OneHotEncoder class from scikit-learn
from sklearn.preprocessing import OneHotEncoder
heart['heart_disease'] = heart['heart_disease'].map({'Yes':1, 'No':0})
cat=['sex', 'smoking_history']

OH_Encoder = OneHotEncoder(handle_unknown='ignore',  sparse_output=False)
OH = OH_Encoder.fit_transform(heart[cat])
cols = OH_Encoder.get_feature_names_out(cat)
OH = pd.DataFrame(OH, columns=cols)
heart = heart.drop(cat,axis=1)
heart = pd.concat([heart, OH], axis =1)
```

In [12]:

```
from sklearn.preprocessing import LabelEncoder
categorical_columns = ['general_health', 'checkup', 'exercise', 'skin_cancer',

# Initialize LabelEncoder

label_encoder = LabelEncoder()

# Apply label encoding to each ordinal categorical column

for col in categorical_columns:
    heart[col] = label_encoder.fit_transform(heart[col])
```

In [13]:

```
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308854 entries, 0 to 308853
Data columns (total 22 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   general_health          308854 non-null  int64  
 1   checkup                 308854 non-null  int64  
 2   exercise                308854 non-null  int64  
 3   heart_disease           308854 non-null  int64  
 4   skin_cancer             308854 non-null  int64  
 5   other_cancer            308854 non-null  int64  
 6   depression              308854 non-null  int64  
 7   diabetes                308854 non-null  int64  
 8   arthritis               308854 non-null  int64  
 9   age_category            308854 non-null  int64  
 10  height                  308854 non-null  float64
 11  weight                  308854 non-null  float64
 12  bmi                     308854 non-null  float64
 13  bmi_group               308854 non-null  int64  
 14  alcohol_consumption     308854 non-null  int64  
 15  fruit_consumption       308854 non-null  int64  
 16  vegetables_consumption  308854 non-null  int64  
 17  potato_consumption      308854 non-null  int64  
 18  sex_Female              308854 non-null  float64
 19  sex_Male                308854 non-null  float64
 20  smoking_history_No      308854 non-null  float64
 21  smoking_history_Yes     308854 non-null  float64
dtypes: float64(7), int64(15)
memory usage: 51.8 MB
```

In [14]:
```python
heart["heart_disease"].value_counts()
```

Out[14]:
```
heart_disease
0    283883
1     24971
Name: count, dtype: int64
```

In [15]:
```python
X = heart.drop("heart_disease", axis = 1)
y = heart['heart_disease']
```

In [16]:
```python
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)
```

In [17]:
```python
from collections import Counter
print("Before SMOTE:", Counter(y))

# After SMOTE
print("After SMOTE:", Counter(y_balanced))

# Convert to DataFrame for better visualization
before = pd.Series(y).value_counts()
after = pd.Series(y_balanced).value_counts()
```

```python

print("\nClass distribution before SMOTE:\n", before)
print("\nClass distribution after SMOTE:\n", after)
```

```
Before SMOTE: Counter({0: 283883, 1: 24971})
After SMOTE: Counter({0: 283883, 1: 283883})

Class distribution before SMOTE:
 heart_disease
0    283883
1     24971
Name: count, dtype: int64

Class distribution after SMOTE:
 heart_disease
0    283883
1    283883
Name: count, dtype: int64
```

In [18]:
```python
from sklearn.model_selection import train_test_split
# Splitting the data into training and testing sets for diabetes balanced

X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, te
```

In [19]:
```python
from sklearn.preprocessing import StandardScaler
scaler_d = StandardScaler()
X_train_scaled = scaler_d.fit_transform(X_train)
X_test_scaled = scaler_d.transform(X_test)
```

In [20]:
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# Build Improved ANN
ann = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),

    Dense(32, activation='relu'),
    Dropout(0.2),

    Dense(1, activation='sigmoid')
])
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:92: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [21]:

```python
# Compile model
ann.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
            loss='binary_crossentropy',
            metrics=['accuracy'])
```

In [22]:

```python

```

In [24]:

```python
# Train model
history = ann.fit(X_train_scaled, y_train,
                  validation_split=0.2,
                  epochs=30,
                  batch_size=32,
                  verbose=1)
```

```
Epoch 1/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 45s 5ms/step - accuracy: 0.8699 - loss: 0.2916 -
val_accuracy: 0.8757 - val_loss: 0.2750
Epoch 2/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 86s 5ms/step - accuracy: 0.8704 - loss: 0.2906 -
val_accuracy: 0.8757 - val_loss: 0.2750
Epoch 3/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 80s 5ms/step - accuracy: 0.8700 - loss: 0.2910 -
val_accuracy: 0.8757 - val_loss: 0.2750
Epoch 4/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 50s 5ms/step - accuracy: 0.8704 - loss: 0.2915 -
val_accuracy: 0.8758 - val_loss: 0.2750
Epoch 5/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 81s 5ms/step - accuracy: 0.8699 - loss: 0.2908 -
val_accuracy: 0.8759 - val_loss: 0.2748
Epoch 6/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 51s 5ms/step - accuracy: 0.8702 - loss: 0.2912 -
val_accuracy: 0.8756 - val_loss: 0.2749
Epoch 7/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 91s 6ms/step - accuracy: 0.8700 - loss: 0.2912 -
val_accuracy: 0.8758 - val_loss: 0.2747
Epoch 8/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 74s 5ms/step - accuracy: 0.8701 - loss: 0.2906 -
val_accuracy: 0.8758 - val_loss: 0.2748
Epoch 9/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 78s 5ms/step - accuracy: 0.8701 - loss: 0.2912 -
val_accuracy: 0.8758 - val_loss: 0.2748
Epoch 10/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 53s 5ms/step - accuracy: 0.8698 - loss: 0.2914 -
val_accuracy: 0.8756 - val_loss: 0.2749
Epoch 11/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 50s 5ms/step - accuracy: 0.8699 - loss: 0.2913 -
val_accuracy: 0.8756 - val_loss: 0.2751
Epoch 12/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 82s 5ms/step - accuracy: 0.8698 - loss: 0.2910 -
val_accuracy: 0.8757 - val_loss: 0.2748
Epoch 13/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 85s 5ms/step - accuracy: 0.8701 - loss: 0.2913 -
val_accuracy: 0.8757 - val_loss: 0.2750
Epoch 14/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 52s 5ms/step - accuracy: 0.8697 - loss: 0.2911 -
val_accuracy: 0.8758 - val_loss: 0.2751
Epoch 15/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 54s 5ms/step - accuracy: 0.8705 - loss: 0.2906 -
val_accuracy: 0.8758 - val_loss: 0.2747
Epoch 16/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 81s 5ms/step - accuracy: 0.8705 - loss: 0.2908 -
val_accuracy: 0.8758 - val_loss: 0.2747
Epoch 17/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 53s 5ms/step - accuracy: 0.8698 - loss: 0.2912 -
val_accuracy: 0.8756 - val_loss: 0.2750
Epoch 18/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 89s 6ms/step - accuracy: 0.8699 - loss: 0.2908 -
val_accuracy: 0.8757 - val_loss: 0.2748
```

```
Epoch 19/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 55s 5ms/step - accuracy: 0.8707 - loss: 0.2904 -
val_accuracy: 0.8759 - val_loss: 0.2749
Epoch 20/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 82s 6ms/step - accuracy: 0.8701 - loss: 0.2910 -
val_accuracy: 0.8758 - val_loss: 0.2750
Epoch 21/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 76s 5ms/step - accuracy: 0.8702 - loss: 0.2906 -
val_accuracy: 0.8758 - val_loss: 0.2747
Epoch 22/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 49s 5ms/step - accuracy: 0.8700 - loss: 0.2909 -
val_accuracy: 0.8758 - val_loss: 0.2746
Epoch 23/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 79s 5ms/step - accuracy: 0.8701 - loss: 0.2908 -
val_accuracy: 0.8757 - val_loss: 0.2747
Epoch 24/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 79s 4ms/step - accuracy: 0.8700 - loss: 0.2913 -
val_accuracy: 0.8756 - val_loss: 0.2748
Epoch 25/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 82s 4ms/step - accuracy: 0.8702 - loss: 0.2903 -
val_accuracy: 0.8756 - val_loss: 0.2746
Epoch 26/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 81s 4ms/step - accuracy: 0.8701 - loss: 0.2905 -
val_accuracy: 0.8757 - val_loss: 0.2748
Epoch 27/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 45s 5ms/step - accuracy: 0.8704 - loss: 0.2905 -
val_accuracy: 0.8759 - val_loss: 0.2747
Epoch 28/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 45s 5ms/step - accuracy: 0.8707 - loss: 0.2908 -
val_accuracy: 0.8757 - val_loss: 0.2746
Epoch 29/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 45s 5ms/step - accuracy: 0.8704 - loss: 0.2912 -
val_accuracy: 0.8759 - val_loss: 0.2747
Epoch 30/30
9936/9936 ━━━━━━━━━━━━━━━━━━━━ 849s 82ms/step - accuracy: 0.8704 - loss: 0.2908
- val_accuracy: 0.8758 - val_loss: 0.2746
```

In [25]:

```python
# Predictions
y_pred_prob = ann.predict(X_test_scaled).ravel()
y_pred = (y_pred_prob > 0.5).astype(int)

# Evaluation
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_prob)

print("=== Improved Artificial Neural Network (ANN) ===")
print(f"Accuracy: {acc:.4f}")
print(f"F1-score: {f1:.4f}")
print(f"ROC-AUC: {auc:.4f}")
```

```
5323/5323 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step
=== Improved Artificial Neural Network (ANN) ===
Accuracy: 0.8776
F1-score: 0.8754
ROC-AUC: 0.9530
```