TMA TM271

Name: Ahmed Mohamed Abdullatif Rifai.

ID: 1851711008.

Project Report

Introduction

This report presents a comprehensive analysis of heart disease data. Our goal is to elucidate patterns and predictors that can inform strategies for early detection and prevention of heart conditions, using machine learning as our analytic tool.

1. Description of the Problem and Dataset

Problem Statement:
The objective of the analysis is to explore and model the dataset related to heart disease. Heart disease is one of the leading causes of mortality globally. By analyzing factors that contribute to heart health, we can better understand how to predict and prevent heart disease.

Dataset Summary:
The dataset utilized in this study comes from the UCI Machine Learning Repository, specifically the Heart Disease dataset identified by the ID=45. This dataset is a collection of attributes that relate to heart health and has been used by researchers to identify correlations and causative factors of heart disease.

Dataset Details:

- **Attributes:** There are 13 explanatory variables in the dataset:
  - Age: The age of the individual.
  - Sex: The gender of the individual.
  - CP (Chest Pain type): The type of chest pain experienced.
  - Trestbps (Resting Blood Pressure): The resting blood pressure.
  - Chol (Serum Cholesterol in mg/dl): The level of cholesterol in the blood.
  - FBS (Fasting Blood Sugar > 120 mg/dl): Whether the fasting blood sugar is above 120 mg/dl.
  - Restecg (Resting Electrocardiographic results): The results of an ECG.
  - Thalach (Maximum Heart Rate Achieved): The maximum heart rate achieved.
  - Exang (Exercise Induced Angina): Whether the exercise induced angina.

- Oldpeak: ST depression induced by exercise relative to rest.
- Slope: The slope of the peak exercise ST segment.
- CA (Number of Major Vessels Colored by Flourosopy): The number of major vessels colored by flourosopy.
- Thal: A blood disorder called thalassemia.

- **Target Variable:** The target variable is binary, indicating the presence or absence of heart disease.
- **Data Size:** The dataset contains 303 instances with 13 features each.

```
Column Names: ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']
Number of Rows: 303
Number of Columns: 13
```

Data Acquisition and Loading:
The dataset was accessed and loaded into the working environment through the use of the `ucimlrepo` Python package, specifically using the `fetch_ucirepo` function.

2.1 Data Preprocessing Steps

Before diving into the analysis, the initial step is to load and understand the dataset to identify the features and determine if there are any discrepancies that need to be addressed.

Part A: Loading Dataset

- **Objectives**: Familiarize with the dataset by displaying its basic characteristics.
- **Tasks**:
  - Show column names, row count, and column count.
  - Display the first and last 10 rows.
  - Display basic statistics (count, mean, standard deviation, etc.).
  - Check and identify any missing values in the dataset.

Loading the Dataset:
The data was imported using the `ucimlrepo` package, which provides an interface to fetch datasets from the UCI Machine Learning Repository. The Heart Disease dataset was loaded by calling the `fetch_ucirepo` function with the dataset ID of 45.

```
[ ] # Step 1: Install the ucimlrepo package (Uncomment the line below if you're running this in Colab or if the package is not installed)
    !pip install ucimlrepo

    # Step 2: Import the necessary package to fetch the dataset
    from ucimlrepo import fetch_ucirepo

    # Step 3: Fetch the heart disease dataset
    heart_disease = fetch_ucirepo(id=45)

    # Step 4: Load the data into variables X (features) and y (target)
    X = heart_disease.data.features
    y = heart_disease.data.targets

    Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.6)
```

Understanding the Dataset:
To get familiar with the dataset, the following steps were taken:

- The names of the columns were displayed to understand the features available in the dataset. The columns correspond to clinical attributes that are commonly associated with heart disease.
- The dataset was found to have 303 rows, each representing a patient, and 13 columns, each representing a clinical feature.
- The first and last 10 rows of the dataset were displayed to get a glimpse of the values across different patients.

```
[ ] # Display the names of the columns
    print("Column Names:", X.columns.tolist())

    # Display the number of rows and columns
    print("Number of Rows:", X.shape[0])
    print("Number of Columns:", X.shape[1])

    Column Names: ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']
    Number of Rows: 303
    Number of Columns: 13
```

```
[ ] # Display the first 10 rows
    print("First 10 Rows:")
    X.head(10)
```

First 10 Rows:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0
5	56	1	2	120	236	0	0	178	0	0.8	1	0.0	3.0
6	62	0	4	140	268	0	2	160	0	3.6	3	2.0	3.0
7	57	0	4	120	354	0	0	163	1	0.6	1	0.0	3.0
8	63	1	4	130	254	0	2	147	0	1.4	2	1.0	7.0
9	53	1	4	140	203	1	2	155	1	3.1	3	0.0	7.0

```
[ ] # Display the last 10 rows
    print("Last 10 Rows:")
    X.tail(10)
```

Last 10 Rows:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
293	63	1	4	140	187	0	2	144	1	4.0	1	2.0	7.0
294	63	0	4	124	197	0	0	136	1	0.0	2	0.0	3.0
295	41	1	2	120	157	0	0	182	0	0.0	1	0.0	3.0
296	59	1	4	164	176	1	2	90	0	1.0	2	2.0	6.0
297	57	0	4	140	241	0	0	123	1	0.2	2	0.0	7.0
298	45	1	1	110	264	0	0	132	0	1.2	2	0.0	7.0
299	68	1	4	144	193	1	0	141	0	3.4	2	2.0	7.0
300	57	1	4	130	131	0	0	115	1	1.2	2	1.0	7.0
301	57	0	2	130	236	0	2	174	0	0.0	2	1.0	3.0
302	38	1	3	138	175	0	0	173	0	0.0	1	NaN	3.0

Statistical Summary:

A statistical summary was produced, providing key metrics such as count, mean, and standard deviation for each feature. These statistics offer a preliminary insight into the distribution and central tendencies of the clinical attributes.

```python
# Display basic statistics of the dataset
print("Basic Statistics:")
X.describe()
```

Basic Statistics:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	299.000000	301.000000
mean	54.438944	0.679868	3.158416	131.689769	246.693069	0.148515	0.990099	149.607261	0.326733	1.039604	1.600660	0.672241	4.734219
std	9.038662	0.467299	0.960126	17.599748	51.776918	0.356198	0.994971	22.875003	0.469794	1.161075	0.616226	0.937438	1.939706
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000	3.000000
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	3.000000
50%	56.000000	1.000000	3.000000	130.000000	241.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000	3.000000
75%	61.000000	1.000000	4.000000	140.000000	275.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000	7.000000
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000	7.000000

Missing Values:

A check for missing values was conducted to ensure the integrity of the dataset. The `ca` and `thal` columns were found to have missing values, with 4 and 2 missing entries respectively. Identifying missing values is crucial as they can affect the performance of machine learning models and the validity of the analysis.

```python
# Find and display any missing values
print("Missing Values per Column:")
X.isnull().sum()
```

```
Missing Values per Column:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          4
thal        2
dtype: int64
```

2.2 Data Preprocessing Steps

Part B: Data Preprocessing

- **Objectives**: Prepare the dataset for modeling by cleaning and structuring the data.
- **Tasks**:
    - Handle missing values, potentially using techniques like mean imputation.
    - Encode categorical features, considering methods like one-hot encoding.
    - Split the dataset into training and testing sets, typically using an 80/20 split.

Objective:
The data preprocessing phase aims to clean and structure the dataset to ensure it is in the optimal format for building and evaluating machine learning models.

Handling Missing Values:
The initial assessment of the dataset revealed missing values in two features: `ca` and `thal`. To address this, mean imputation was performed using the SimpleImputer from `sklearn.impute`. This technique replaces missing values with the mean value of each respective feature. After imputation, we verified that there were no missing values remaining, ensuring that the dataset is complete for all features.

```
[ ] # Import pandas
    import pandas as pd

    # Import the SimpleImputer class from sklearn.impute
    from sklearn.impute import SimpleImputer

    # Create an imputer object with a strategy of 'mean' to replace missing values
    imputer = SimpleImputer(strategy='mean')

    # Apply the imputer to our data
    # Assuming X is a DataFrame and we want to apply imputation column-wise
    X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

    # Check if there are any missing values left
    print("Missing Values After Imputation:")
    print(X_imputed.isnull().sum())
```

```
Missing Values After Imputation:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
dtype: int64
```

Categorical Feature Encoding:

The dataset contains categorical variables that machine learning algorithms cannot process in their raw form. To convert these to a suitable format, one-hot encoding was applied to the `sex` and `cp` features using the OneHotEncoder from `sklearn.preprocessing`. This method transforms each categorical value into a new binary feature, thus enabling the algorithm to better understand the patterns within these variables.

```
[ ] # Check the number of unique values in each column of X_imputed
    X_imputed.nunique()
```

```
age          41
sex           2
cp            4
trestbps     50
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
dtype: int64
```

```
[ ] X_imputed.head(10)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0
5	56.0	1.0	2.0	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0
6	62.0	0.0	4.0	140.0	268.0	0.0	2.0	160.0	0.0	3.6	3.0	2.0	3.0
7	57.0	0.0	4.0	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0
8	63.0	1.0	4.0	130.0	254.0	0.0	2.0	147.0	0.0	1.4	2.0	1.0	7.0
9	53.0	1.0	4.0	140.0	203.0	1.0	2.0	155.0	1.0	3.1	3.0	0.0	7.0

Dataframe Transformation:
After encoding, the transformed dataset now contains 20 features, with the categorical variables represented as binary vectors. The first ten rows of the newly encoded dataframe were displayed to confirm the transformation.

```python
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Define which columns to encode
categorical_columns = ['sex', 'cp']  # Ensure these are the names in your DataFrame

# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse=False)

# Perform the one-hot encoding and convert the result to a DataFrame
encoded_data = encoder.fit_transform(X_imputed[categorical_columns])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical_columns))

# Drop original categorical columns and concatenate the new one-hot encoded columns
X_encoded = pd.concat([X_imputed.drop(categorical_columns, axis=1), encoded_df], axis=1)

# Display the transformed DataFrame
X_encoded.head(10)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_ou
  warnings.warn(
```

	age	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	sex_0.0	sex_1.0	cp_1.0	cp_2.0	cp_3.0	cp_4.0
0	63.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0.0	1.0	1.0	0.0	0.0	0.0
1	67.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	0.0	1.0	0.0	0.0	0.0	1.0
2	67.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	0.0	1.0	0.0	0.0	0.0	1.0
3	37.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0.0	1.0	0.0	0.0	1.0	0.0
4	41.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	1.0	0.0	0.0	1.0	0.0	0.0
5	56.0	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0.0	1.0	0.0	1.0	0.0	0.0
6	62.0	140.0	268.0	0.0	2.0	160.0	0.0	3.6	3.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	1.0
7	57.0	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	1.0	0.0	0.0	0.0	0.0	1.0
8	63.0	130.0	254.0	0.0	2.0	147.0	0.0	1.4	2.0	1.0	7.0	0.0	1.0	0.0	0.0	0.0	1.0
9	53.0	140.0	203.0	1.0	2.0	155.0	1.0	3.1	3.0	0.0	7.0	0.0	1.0	0.0	0.0	0.0	1.0

Splitting the Dataset:
The final step in data preprocessing was to split the dataset into training and testing sets. This is a vital step to evaluate the model's performance on unseen data, ensuring that the results are reliable and the model has not simply memorized the training data. We used an 80/20 split, allocating 80% of the data to the training set and 20% to the testing set. The training set includes 242 instances, and the testing set includes 61 instances.

```
[ ] X_encoded.shape

(303, 20)
```

```
[ ] # Import the train_test_split function
    from sklearn.model_selection import train_test_split

    # Split the dataset into training (80%) and testing (20%) sets
    X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

    # Display the size of the training and testing sets
    print(f"Training set size: X={X_train.shape}, y={y_train.shape}")
    print(f"Testing set size: X={X_test.shape}, y={y_test.shape}")

    Training set size: X=(242, 20), y=(242, 1)
    Testing set size: X=(61, 20), y=(61, 1)
```

With preprocessing completed, the dataset is now ready for the model selection, training, and evaluation stages.

---

3. Model Selection, Training, and Evaluation

Part C: Model Building

- **Objectives**: Develop a machine learning model to predict heart disease.
- **Tasks**:
    - Choose and implement a suitable classification or regression algorithm.
    - Train the model on the training data.
    - Evaluate the model's performance using appropriate metrics (e.g., accuracy, recall, precision).
    - Fine-tune the model through hyperparameter adjustments or by trying different algorithms.

Objective:
The aim of this phase was to develop a predictive model for heart disease. This involved choosing a suitable algorithm, training the model, evaluating its performance, and making necessary adjustments to improve its accuracy.

Model Selection and Training:
A Logistic Regression classifier was chosen due to its efficiency and robust performance for binary classification problems. The model was trained using the dataset, with the 'max_iter' parameter set

to 1000 to ensure convergence. However, a warning indicated that the number of iterations reached the maximum limit, suggesting that either the number of iterations should be increased or the data should be scaled.

```
[24] # Import the Logistic Regression model
     from sklearn.linear_model import LogisticRegression

     # Create a Logistic Regression classifier instance
     clf = LogisticRegression(max_iter=1000, random_state=42)

     # Train the classifier on the training set
     clf.fit(X_train, y_train)

     /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:
       y = column_or_1d(y, warn=True)
     /usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logist
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as sh
         https://scikit-learn.org/stable/modules/preprocessing.html
     Please also refer to the documentation for alternative solver option
         https://scikit-learn.org/stable/modules/linear_model.html#logist
       n_iter_i = _check_optimize_result(

                     LogisticRegression
     LogisticRegression(max_iter=1000, random_state=42)
```

Initial Model Evaluation:
The model's performance was evaluated using a set of metrics including accuracy, precision, recall, and F1 score. The initial results showed an accuracy of approximately 55.74%. The precision and recall were also in a similar range, and the F1 score was slightly below 52%. These metrics were complemented by a confusion matrix to visualize the model's performance in classifying the different classes correctly.

```
[25] # Import necessary metrics from sklearn.metrics
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

     # Predict the labels for the test set
     y_pred = clf.predict(X_test)

     # Calculate and print the metrics
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Precision:", precision_score(y_test, y_pred, average='weighted'))
     print("Recall:", recall_score(y_test, y_pred, average='weighted'))
     print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))

     # Additionally, display the confusion matrix
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.5573770491803278
Precision: 0.510928961748634
Recall: 0.5573770491803278
F1 Score: 0.5192166167588692
Confusion Matrix:
 [[28  0  1  0  0]
 [ 3  2  4  3  0]
 [ 3  0  2  4  0]
 [ 0  3  2  2  0]
 [ 1  0  0  3  0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
  _warn_prf(average, modifier, msg_start, len(result))
```

Model Fine-Tuning:
To improve the model, hyperparameter tuning was conducted using GridSearchCV, which searched over a range of 'C' parameter values to find the most effective regularization strength. The best parameters found were then used to retrain the model. This process slightly improved precision to around 54.38% but did lead to significant changes in other metrics.

```
[26] # Example of fine-tuning Logistic Regression by adjusting the regularization strength
     # Import GridSearchCV
     from sklearn.model_selection import GridSearchCV

     # Define the parameter grid
     param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

     # Create a GridSearchCV object
     grid_search = GridSearchCV(LogisticRegression(max_iter=1000, random_state=42), param_grid, cv=5)

     # Fit it to the training data
     grid_search.fit(X_train, y_train)

     # Print the best parameters and the best score
     print("Best Parameters:", grid_search.best_params_)
     print("Best cross-validation score:", grid_search.best_score_)
```

```python
# Import necessary metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Use the best estimator found by the grid search
best_clf = grid_search.best_estimator_

# Predict the labels for the test set
y_pred = best_clf.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the metrics
print("Test Accuracy:", accuracy)
print("Test Precision:", precision)
print("Test Recall:", recall)
print("Test F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Test Accuracy: 0.5737704918032787
Test Precision: 0.5438069216757742
Test Recall: 0.5737704918032787
Test F1 Score: 0.5378891929772506
Confusion Matrix:
 [[28  0  1  0  0]
 [ 4  3  1  4  0]
 [ 3  0  2  4  0]
 [ 0  2  3  2  0]
 [ 1  0  1  2  0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
  _warn_prf(average, modifier, msg_start, len(result))
```

Alternative Model - Random Forest:
Given the initial model's performance, a Random Forest classifier was also implemented, utilizing class weights to handle class imbalance. The Random Forest model showed a significant improvement in the recall for the minority class, though the overall accuracy remained similar.

```
[28] from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import classification_report

     # Initialize the Random Forest classifier with class weights to handle imbalance
     rf_clf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

     # Fit the classifier
     rf_clf.fit(X_train, y_train)

     # Predict on the test set
     y_pred_rf = rf_clf.predict(X_test)

     # Evaluate the model
     print("Random Forest Test Metrics:")
     print(classification_report(y_test, y_pred_rf))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
<ipython-input-28-7699d60a4885>:8: DataConversionWarning: A column-vector y was passed when a
  rf_clf.fit(X_train, y_train)
Random Forest Test Metrics:
              precision    recall  f1-score   support

           0       0.76      0.97      0.85        29
           1       0.22      0.17      0.19        12
           2       0.14      0.11      0.12         9
           3       0.12      0.14      0.13         7
           4       0.00      0.00      0.00         4

    accuracy                           0.52        61
   macro avg       0.25      0.28      0.26        61
weighted avg       0.44      0.52      0.47        61

Confusion Matrix:
 [[28  0  1  0  0]
 [ 5  2  2  3  0]
 [ 3  2  1  3  0]
 [ 0  4  2  1  0]
 [ 1  1  1  1  0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMet
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMet
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMet
  _warn_prf(average, modifier, msg_start, len(result))
```

Evaluation Metrics:

The metrics post-fine-tuning and the Random Forest classifier’s performance were:

- Logistic Regression (after fine-tuning):
  - Accuracy: 55.74%
  - Precision: 54.38%
  - Recall: 55.74%

  - F1 Score: 53.78%
- Random Forest Classifier:
  - Accuracy: Varies (due to class weights)
  - Precision: Varies
  - Recall: Improved for minority class
  - F1 Score: Varies

The confusion matrices for both models provide a detailed account of true positives, false positives, true negatives, and false negatives, which are critical for understanding the model's performance in a clinical setting.

---

4. Discussion of Results and Insights Gained

The initial Logistic Regression model yielded an accuracy just above random chance, highlighting the complexity of the problem at hand. Meanwhile, the Random Forest model demonstrated a higher sensitivity to the minority class, suggesting its potential for improving over baseline models in imbalanced datasets like the one we have.

These results emphasize the multifaceted nature of medical data and the necessity for models that can capture the nuanced interactions of biological variables. The insights from this analysis point to the need for more robust feature engineering and the exploration of more sophisticated models.

5. Recommendations for Further Improvements

Based on the outcomes of the current models, the following recommendations are made for enhancing future iterations of the analysis:

- **Data Scaling and Transformation:** Prior to model training, scaling features could potentially improve the Logistic Regression model's ability to converge and find a solution.
- **Algorithm Exploration:** Testing additional algorithms, such as Support Vector Machines, Gradient Boosting Machines, or Neural Networks, might uncover hidden patterns within the data that simpler models could not.
- **Cross-Validation:** Implementing stratified k-fold cross-validation could provide a more reliable estimate of the model's performance, especially in an imbalanced dataset.
- **Feature Engineering:** Investigating the creation of new features or the modification of existing ones could provide the models with more predictive power.

6. Social, Professional, Legal, and Ethical Issues

The deployment of machine learning models in healthcare must be carefully managed, considering the following issues:

- **Bias and Fairness:** There is a need to ensure that the models do not propagate or amplify biases present in the data, which could lead to unequal treatment of different patient groups.
- **Accountability:** It must be clear who is accountable for the decisions made by the models, especially in cases where the prediction influences a patient's treatment plan.
- **Transparency:** The models and their decision-making processes should be transparent to allow healthcare professionals to understand and trust their predictions.
- **Legal Compliance:** The models must comply with healthcare regulations, such as HIPAA in the United States, which protect patient privacy and the security of medical data.