Predicting Disease Likelihood Using Machine Learning

A Multiclass Classification Approach

MAHAMMAD BASHEER K S

@MENTORNESS

Introduction

In modern healthcare, early disease detection is crucial for improving patient outcomes and reducing costs. Timely diagnosis leads to better management of chronic conditions. Machine learning, a key tool in predictive analytics, analyzes large datasets to identify patterns and make accurate predictions, greatly aiding medical decision-making.

Objectives

- The objective of this project is to develop a predictive model that can accurately classify individuals into diseased or non-diseased categories based on their health attributes.

- By leveraging machine learning algorithms, we aim to create a reliable tool that healthcare providers can use to assist in disease diagnosis and prognosis.

Dataset Description

The dataset consists of the following attributes:

- Cholesterol: Level of cholesterol in the blood (mg/dL)
- Hemoglobin: Protein in red blood cells carrying oxygen
- Platelets: Blood cells aiding in clotting
- White Blood Cells (WBC): Immune system cells fighting infections
- Red Blood Cells (RBC): Cells carrying oxygen
- Hematocrit: Percentage of blood volume occupied by RBC
- Mean Corpuscular Volume (MCV): Average volume of RBC
- Mean Corpuscular Hemoglobin (MCH): Average amount of hemoglobin in RBC
- Mean Corpuscular Hemoglobin Concentration (MCHC): Average concentration of hemoglobin in RBC
- Insulin: Hormone regulating blood sugar levels
- BMI (Body Mass Index): Measure of body fat based on height and weight
- Systolic Blood Pressure (SBP): Pressure in arteries during heartbeats
- Diastolic Blood Pressure (DBP): Pressure in arteries at rest between beats
- Triglycerides: Type of fat found in blood (mg/dL)
- HbA1c (Glycated Hemoglobin): Measure of avg. blood sugar levels over past 2-3 months
- LDL (Low-Density Lipoprotein) Cholesterol: "Bad" cholesterol
- HDL (High-Density Lipoprotein) Cholesterol: "Good" cholesterol
- ALT (Alanine Aminotransferase): Liver enzyme
- AST (Aspartate Aminotransferase): Enzyme found in liver and heart
- Heart Rate: Number of heartbeats per minute (bpm)
- Creatinine: Waste product produced by muscles and filtered by kidneys
- Troponin: Protein released into bloodstream during heart muscle damage
- C-reactive Protein (CRP): Marker of inflammation in the body
- Disease: Binary indicator (1: Diseased, 0: Non-diseased)

Dataset Overview

- The Dataset contains 25 columns.
- The "Disease" column is the target variable, containing multiple classes for multiclass classification. i.e,
  - Anemia
  - Healthy
  - Diabetes
  - Thalassemia
  - Thrombocytopenia
- The remaining 24 columns are features representing various health metrics and attributes.

```
In [3]: data_train.head(10)
```

Out[3]:

natocrit	Mean Corpuscular Volume	Mean Corpuscular Hemoglobin	Mean Corpuscular Hemoglobin Concentration	...	HbA1c	LDL Cholesterol	HDL Cholesterol	ALT	AST	Heart Rate	Creatinine	Troponin	C-reactive Protein	Disease
.290006	0.631045	0.001328	0.795829	...	0.502665	0.215560	0.512941	0.064187	0.610827	0.939485	0.095512	0.465957	0.769230	Healthy
.164216	0.307553	0.207938	0.505562	...	0.856810	0.652465	0.106961	0.942549	0.344261	0.666368	0.659060	0.816982	0.401166	Diabetes
.625267	0.295122	0.868369	0.026808	...	0.466795	0.387332	0.421763	0.007186	0.506918	0.431704	0.417295	0.799074	0.779208	Thalasse
.073293	0.668719	0.125447	0.501051	...	0.016256	0.040137	0.826721	0.265415	0.594148	0.225756	0.490349	0.637061	0.354094	Anemia
.894991	0.442159	0.257288	0.805987	...	0.429431	0.146294	0.221574	0.015280	0.567115	0.841412	0.153350	0.794008	0.094970	Thalasse
.900010	0.985488	0.679007	0.355774	...	0.389461	0.529914	0.222687	0.772461	0.119994	0.894273	0.128124	0.379016	0.751438	Diabetes
.387461	0.461418	0.305588	0.741120	...	0.446854	0.729376	0.615543	0.794735	0.233890	0.612188	0.407891	0.426863	0.532100	Thromboc
.684896	0.380656	0.248189	0.490245	...	0.673188	0.033037	0.772045	0.253892	0.322486	0.659069	0.774219	0.714177	0.609177	Thromboc
.851338	0.820546	0.106637	0.006947	...	0.022621	0.061317	0.644191	0.715823	0.417170	0.639148	0.213026	0.549920	0.036800	Thalasse
.684896	0.380656	0.248189	0.490245	...	0.673188	0.033037	0.772045	0.253892	0.322486	0.659069	0.774219	0.714177	0.609177	Thromboc

Data Preprocessing

- Checked for Null Values
- Generated Dataset Summary with data_train.info()
- Descriptive Statistics with data_train.describe()
- Encoded "Disease" Column with LabelEncoder:

  Anemia ---------- 0

  Diabetes -------- 1

  Healthy --------- 2

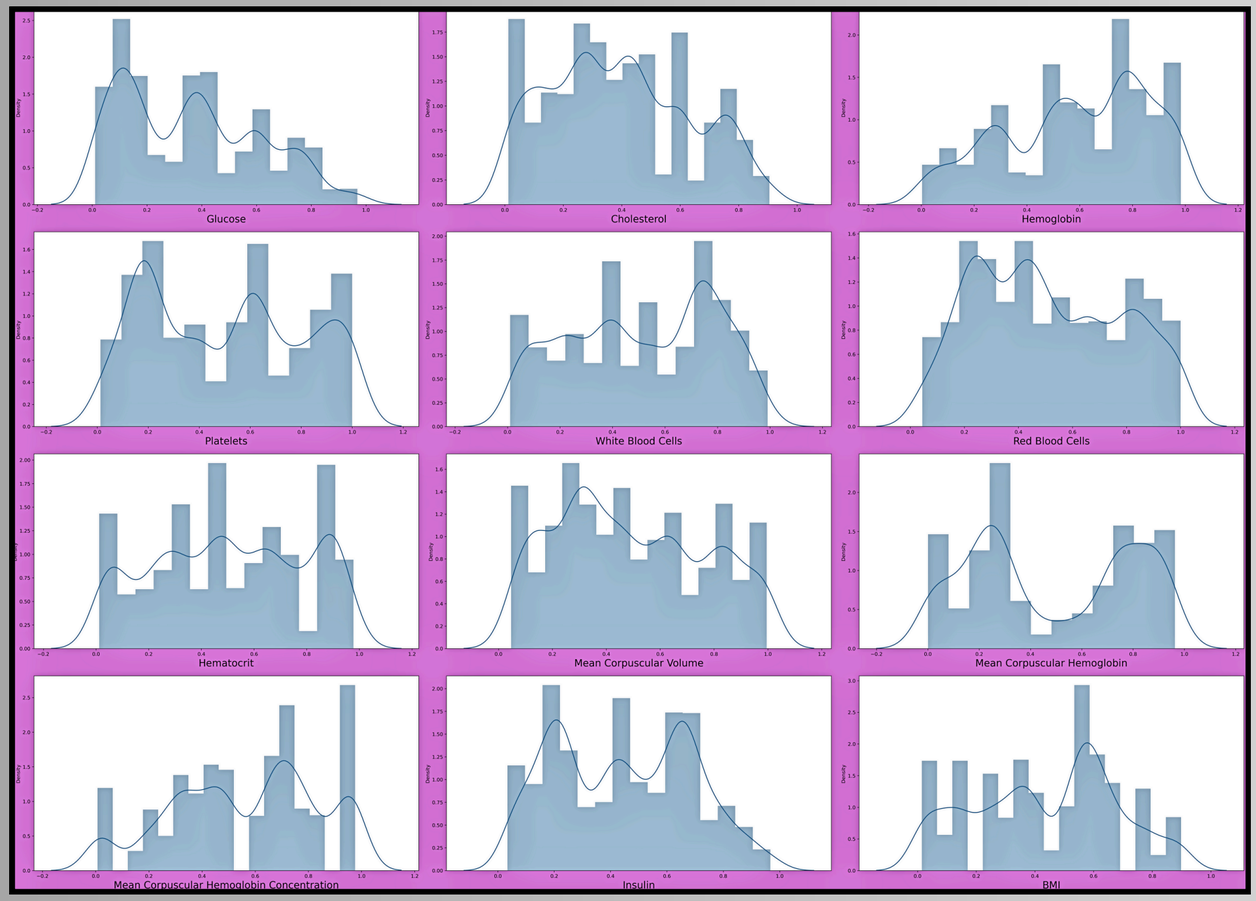
  Thalassemia ----- 3

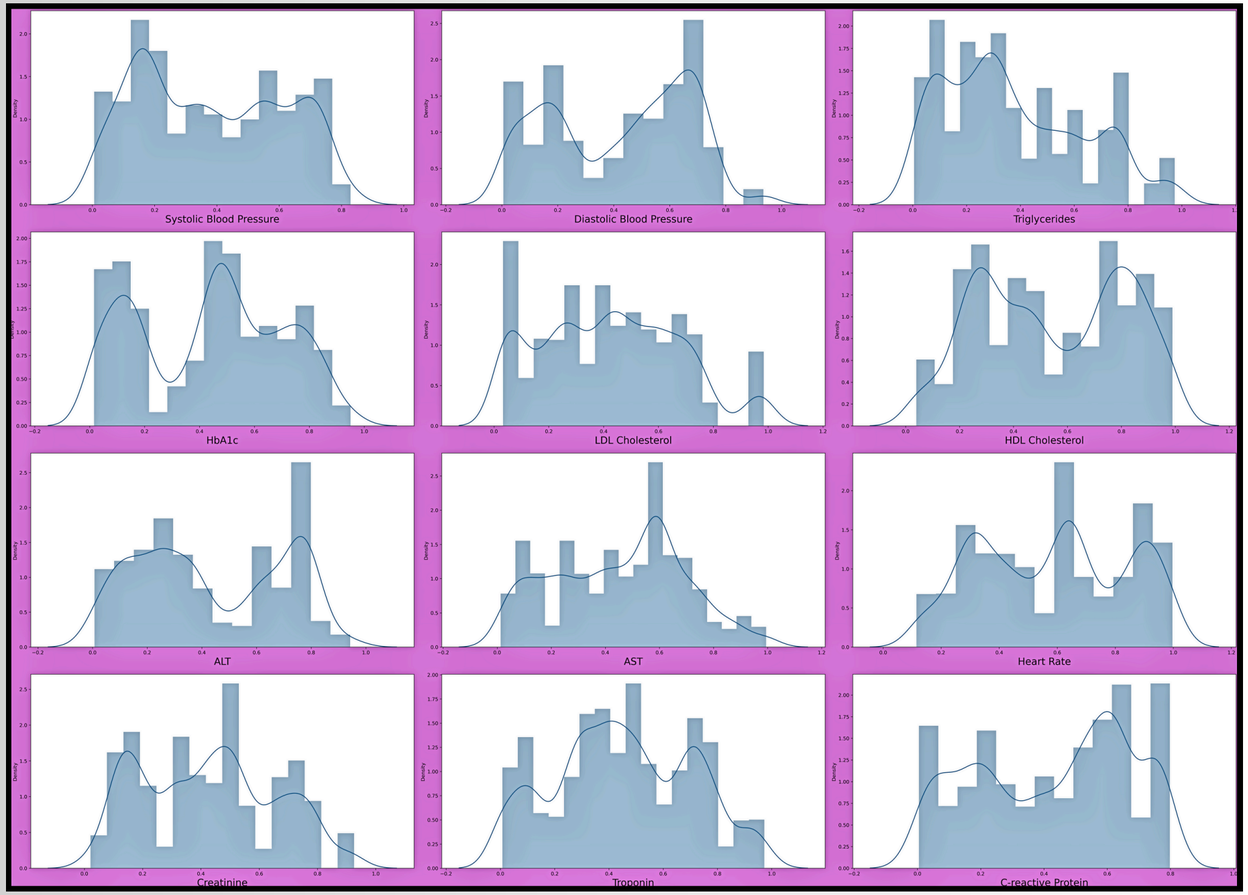
  Thrombocytopenia - 4

```
In [69]: data_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2351 entries, 0 to 2350
Data columns (total 25 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Glucose                               2351 non-null   float64
 1   Cholesterol                           2351 non-null   float64
 2   Hemoglobin                            2351 non-null   float64
 3   Platelets                             2351 non-null   float64
 4   White Blood Cells                     2351 non-null   float64
 5   Red Blood Cells                       2351 non-null   float64
 6   Hematocrit                            2351 non-null   float64
 7   Mean Corpuscular Volume               2351 non-null   float64
 8   Mean Corpuscular Hemoglobin           2351 non-null   float64
 9   Mean Corpuscular Hemoglobin Concentration  2351 non-null   float64
 10  Insulin                               2351 non-null   float64
 11  BMI                                   2351 non-null   float64
 12  Systolic Blood Pressure               2351 non-null   float64
 13  Diastolic Blood Pressure              2351 non-null   float64
 14  Triglycerides                         2351 non-null   float64
 15  HbA1c                                 2351 non-null   float64
 16  LDL Cholesterol                       2351 non-null   float64
 17  HDL Cholesterol                       2351 non-null   float64
 18  ALT                                   2351 non-null   float64
 19  AST                                   2351 non-null   float64
 20  Heart Rate                            2351 non-null   float64
 21  Creatinine                            2351 non-null   float64
 22  Troponin                              2351 non-null   float64
 23  C-reactive Protein                    2351 non-null   float64
 24  Disease                               2351 non-null   object
dtypes: float64(24), object(1)
memory usage: 459.3+ KB
```

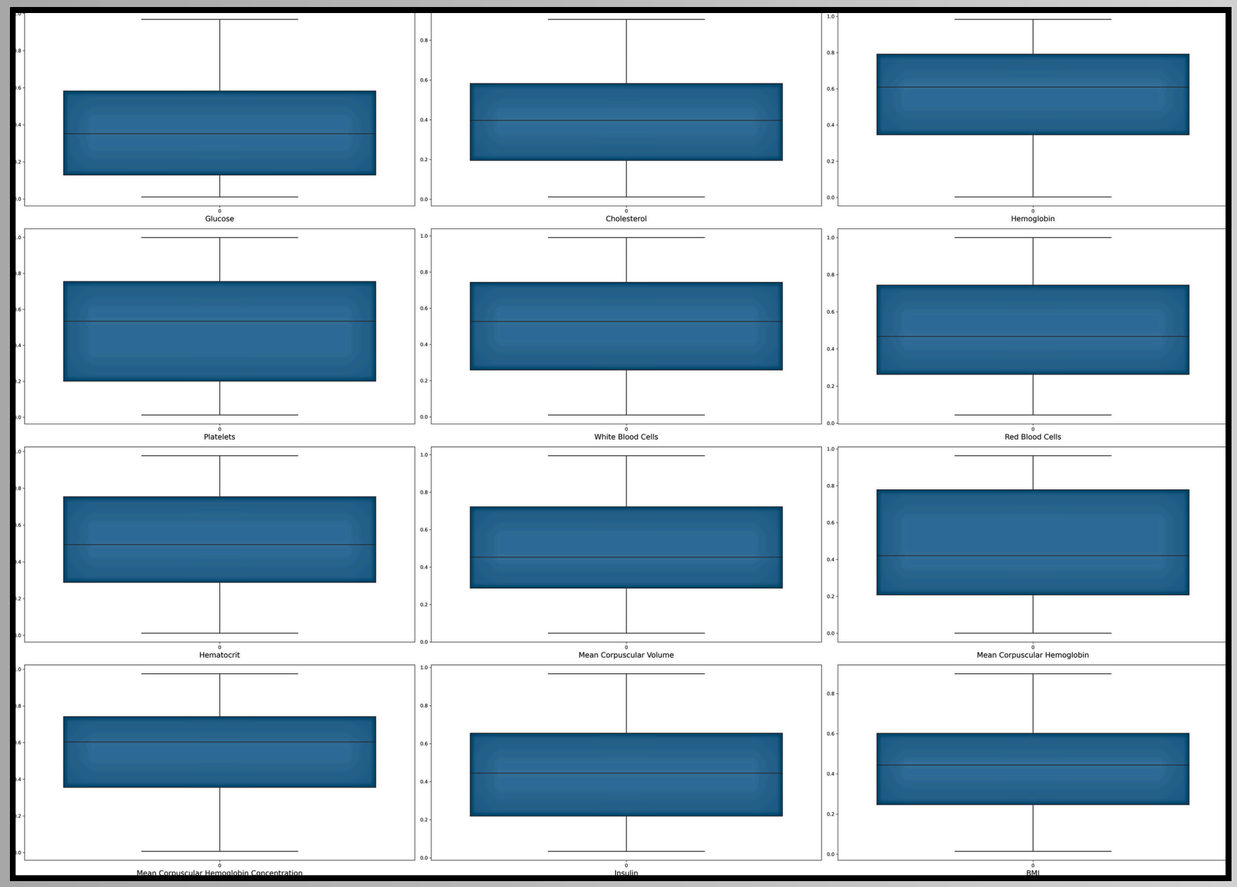
Data Visualization

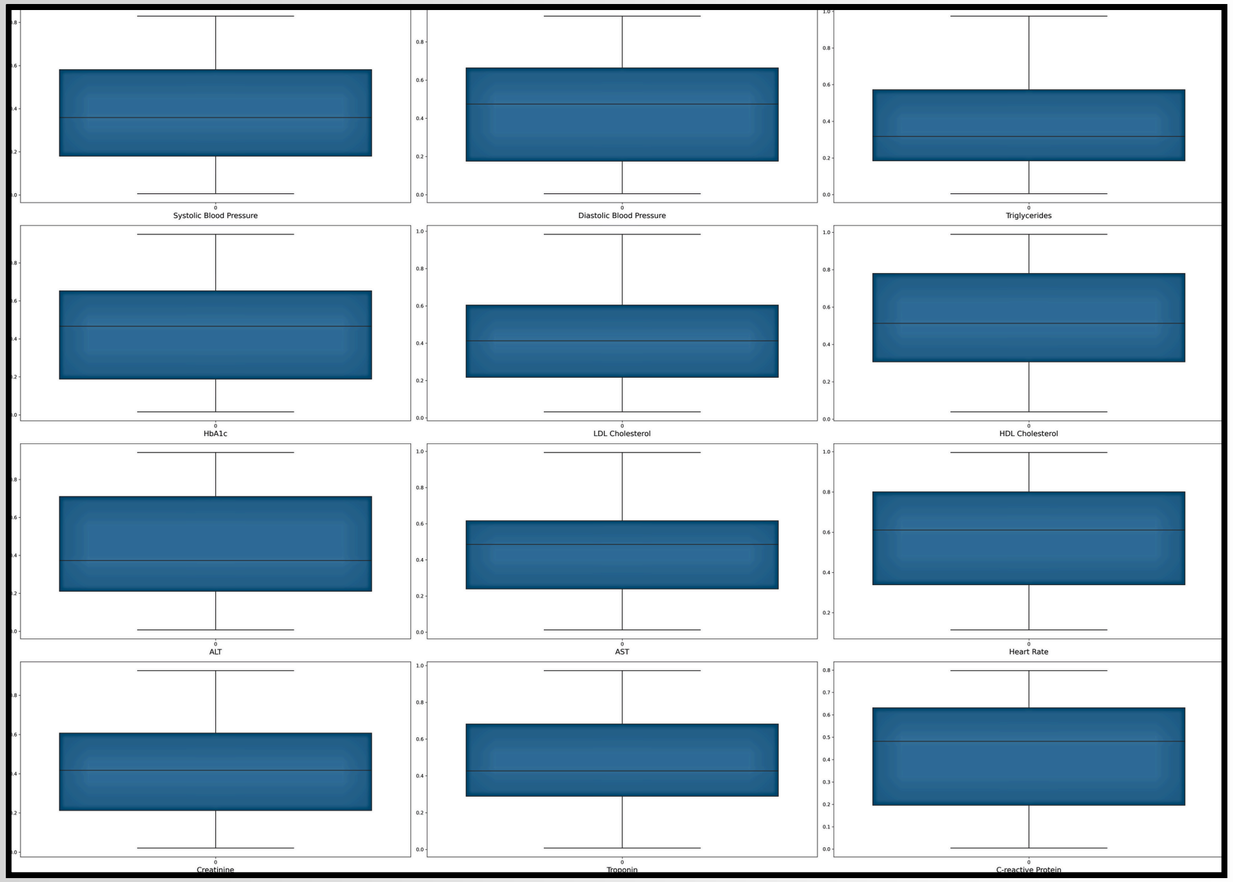




The Plot showing the distribution of each columns in dataset, there is no significant skewness present.

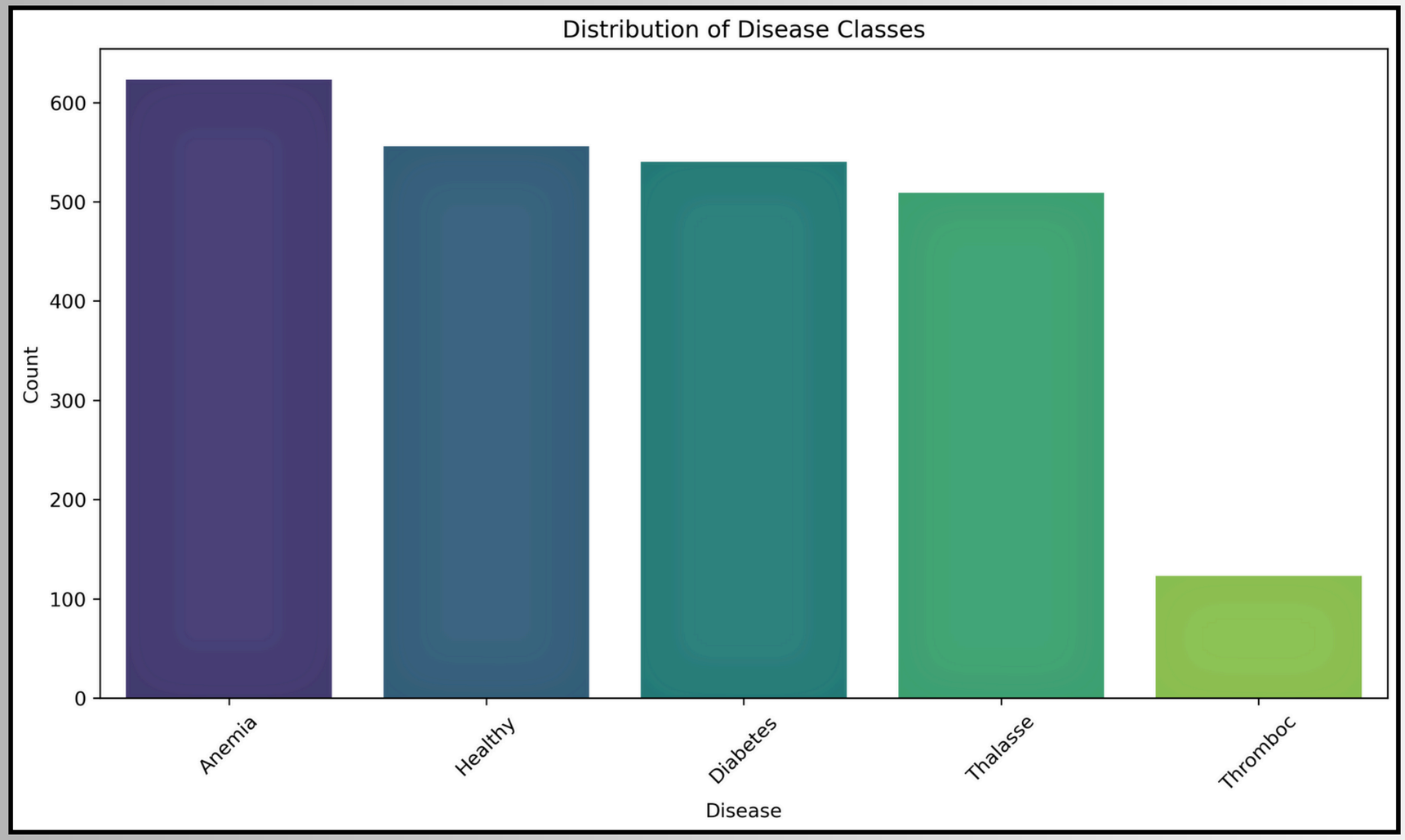
Data Visualization





The boxplots suggest that the dataset has a relatively balanced distribution for most features, with no extreme outliers.

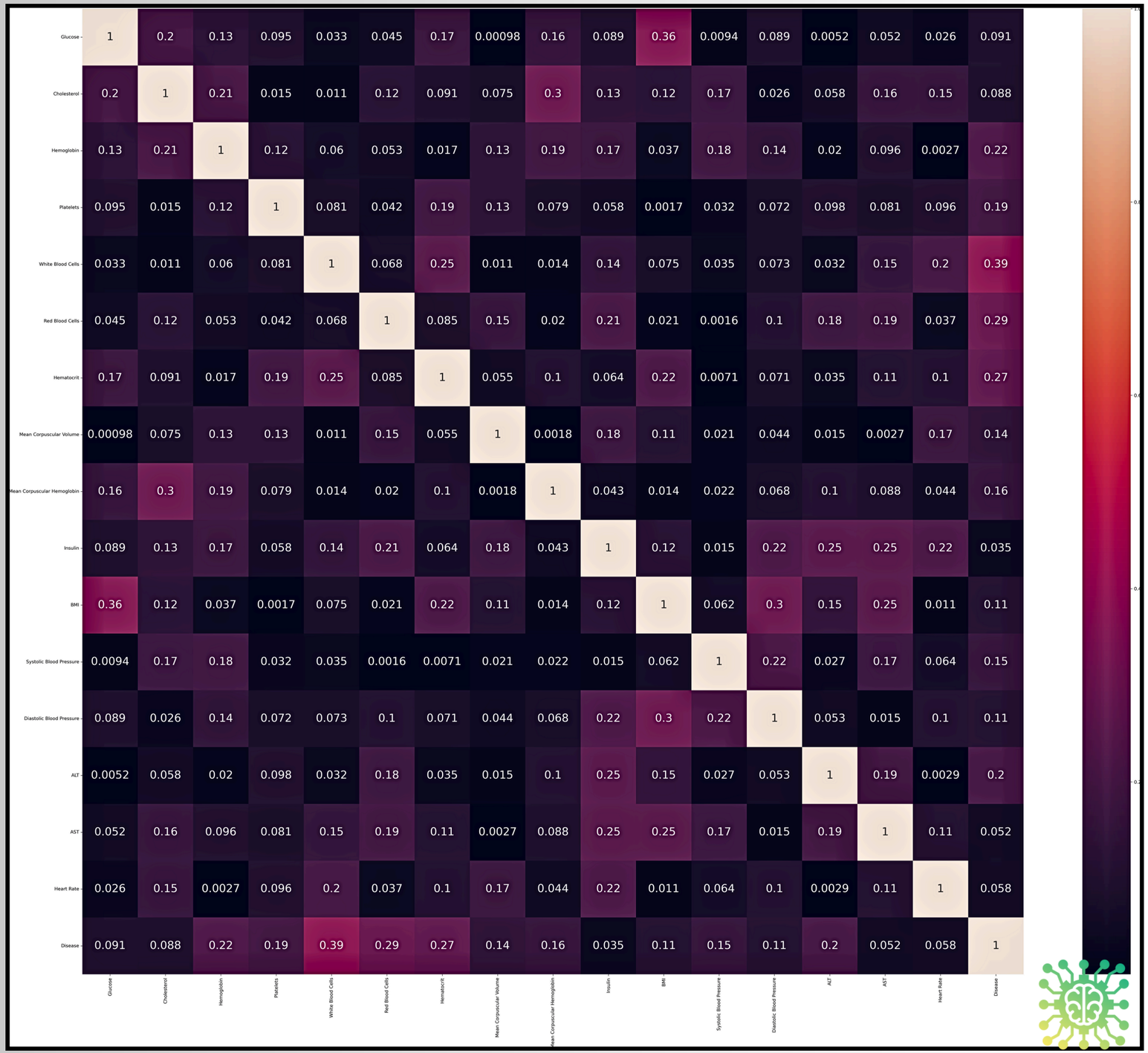
Data Visualization



To balance the dataset, we used oversampling specifically for the 'Thromboc' class, as it had significantly less data compared to the other classes, which were already balanced.

Correlation Heatmap Analysis

The heatmap visualizes the correlation between various health attributes in the dataset.



Feature Selection

- Based on the correlation values, we can consider dropping the features that have very low correlation with the target variable (Disease).
- Dropped Features are;
  - HbA1c
  - C-reactive Protein
  - HDL Cholesterol
  - Creatinine
  - LDL Cholesterol
  - Triglycerides
  - Mean Corpuscular Hemoglobin Concentration
  - Troponin

```
In [22]: # correlation with label
         correlation = data_train.corr()
         print(correlation['Disease'].sort_values(ascending=False))
```

```
Disease                                   1.000000
White Blood Cells                         0.385071
Red Blood Cells                           0.294867
Hematocrit                                0.268761
Hemoglobin                                0.223984
ALT                                       0.202289
Mean Corpuscular Hemoglobin               0.155268
BMI                                       0.107725
Glucose                                   0.091033
Cholesterol                               0.087657
HbA1c                                     0.079728
C-reactive Protein                        0.063696
HDL Cholesterol                           0.040832
Creatinine                                0.026292
LDL Cholesterol                           0.002600
Triglycerides                             0.001707
Mean Corpuscular Hemoglobin Concentration -0.023382
Troponin                                  -0.027381
Insulin                                   -0.034720
AST                                       -0.051829
Heart Rate                                -0.057842
Diastolic Blood Pressure                  -0.109465
Mean Corpuscular Volume                   -0.141039
Systolic Blood Pressure                   -0.153778
Platelets                                 -0.193857
Name: Disease, dtype: float64
```

Test Dataset

Test Dataset

In [25]:
```
data_test=pd.read_csv('test_data.csv')
data_test.head()
```

Out[25]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	Mean Corpuscular Hemoglobin	Mean Corpuscular Hemoglobin Concentration	...	HbA1c	LDL Cholesterol	HDl Cholestero
0	0.001827	0.033693	0.114755	0.997927	0.562604	0.866499	0.578042	0.914615	0.026864	0.038641	...	0.653230	0.186104	0.430398
1	0.436679	0.972653	0.084998	0.180909	0.675736	0.563889	0.798382	0.670361	0.376092	0.184890	...	0.833540	0.153001	0.458533
2	0.545697	0.324815	0.584467	0.475748	0.558596	0.661007	0.934056	0.381782	0.500342	0.531829	...	0.678901	0.220479	0.81715
3	0.172994	0.050351	0.736000	0.782022	0.069435	0.085219	0.032907	0.460619	0.785448	0.491495	...	0.381500	0.459396	0.42015
4	0.758534	0.739968	0.597868	0.772683	0.875720	0.860265	0.486189	0.486686	0.621048	0.191756	...	0.993381	0.272338	0.66357

5 rows × 25 columns

Loading Test Dataset: The test dataset is loaded into the 'data_test' DataFrame. Here's a glimpse of the first few rows.

Data Splitting

In [36]:
```
# Separating features (X) and target variable (y) from the balanced dataset.
x_train = data_train.drop(columns = ['Disease'])
y_train = data_train['Disease']

x_test = data_test.drop(columns = ['Disease'])
y_test = data_test['Disease']
```

Separated Features and Target Variable:
Extracted features (X) and target (y) from both training and testing datasets.

Model Training: Decision Tree

Decision Tree Model

In [78]:
```
# Initialize the Decision Tree Classifier model
DT_model = DecisionTreeClassifier()

# Train the model using the training dataset
DT_model.fit(x_train, y_train)
```

Out[78]:

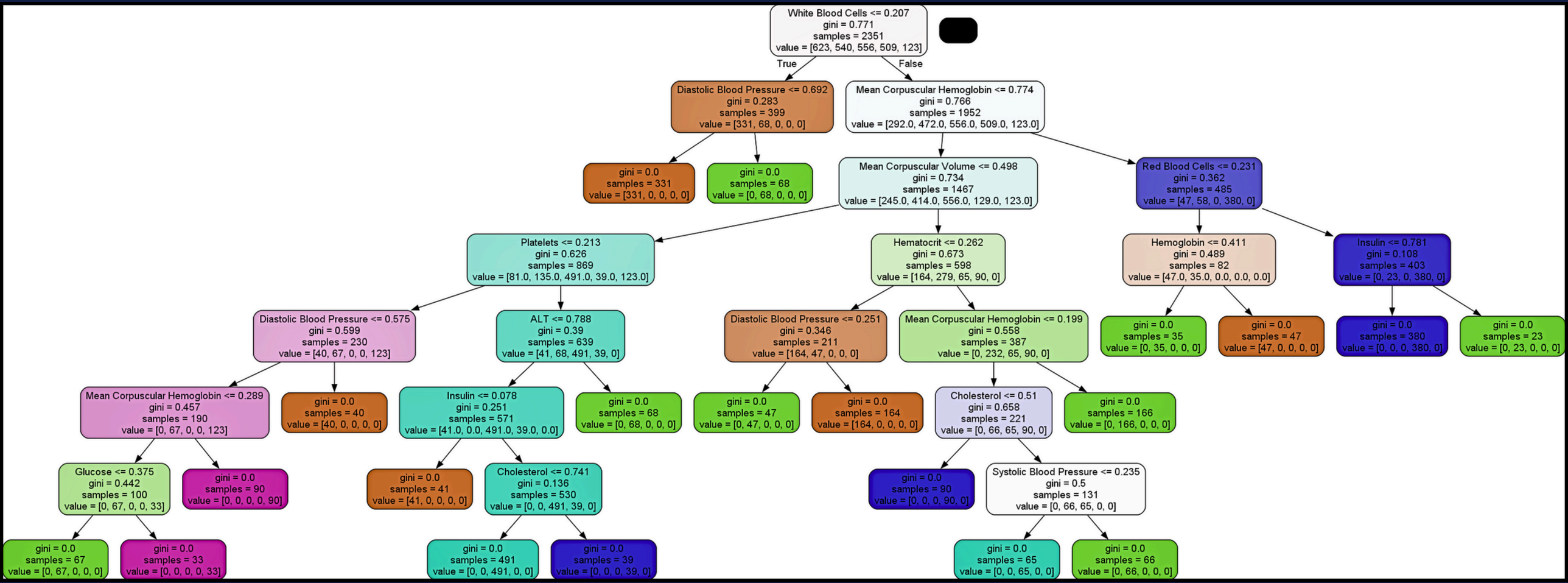
DecisionTreeClassifier

```
DecisionTreeClassifier()
```

Trained Decision Tree Classifier:

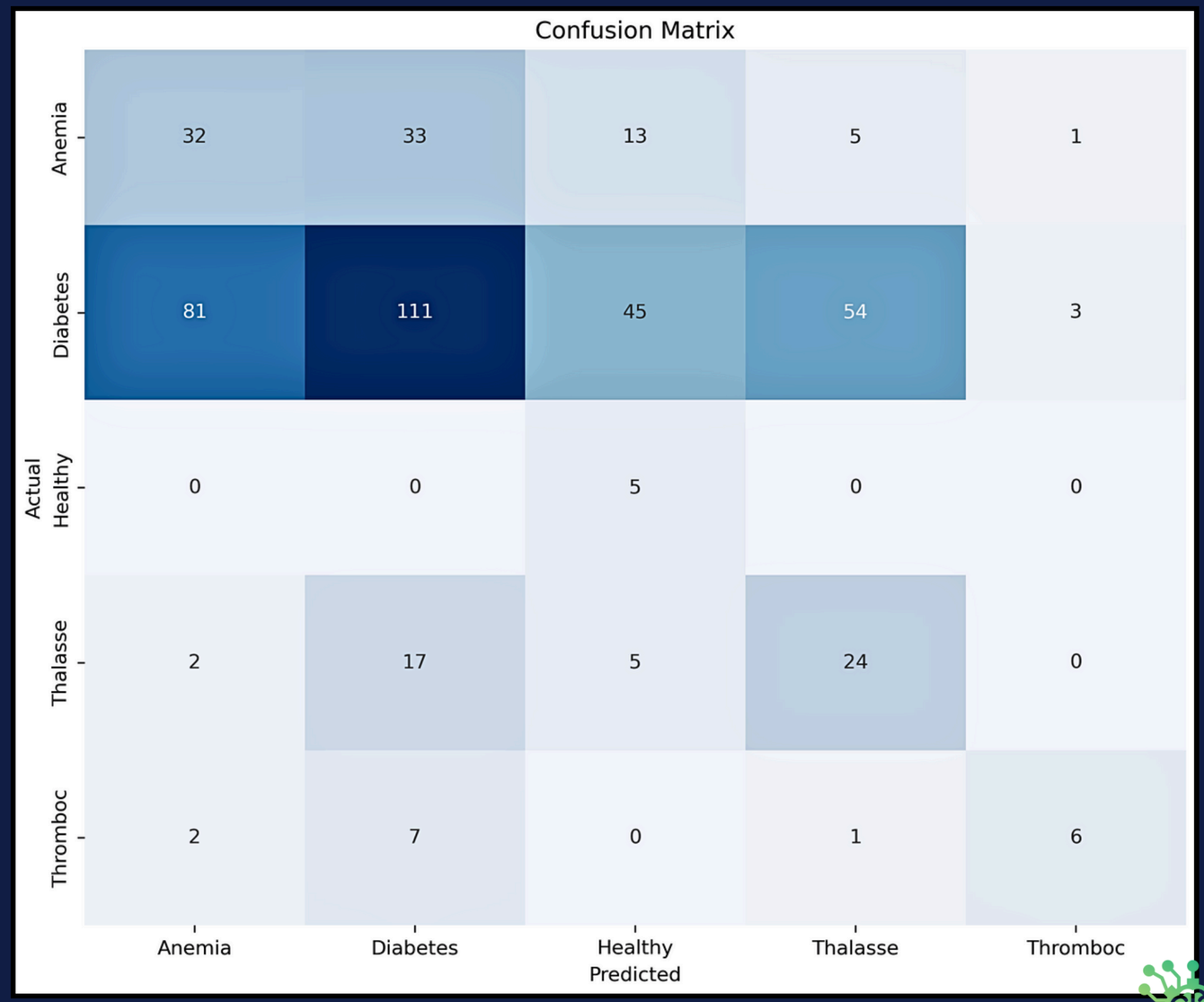
Initialized and trained a Decision Tree model on the training dataset with an accuracy of 39.82%

Decision Tree Structure



Decision Tree Model Evaluation: Confusion Matrix

- The confusion matrix heatmap provides a detailed view of the model's performance, showing the counts of true and predicted classifications for each disease category.
  - Precision (macro): 0.3786924513395101
  - Recall (macro): 0.5267006802721088
  - F1 Score (macro): 0.3522177134526347 5



Model Training: Random Forest

Random Forest Model

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [50]:
```
# Initialize the model
model_forest = RandomForestClassifier(random_state=42)

# Train the model
model_forest.fit(x_train, y_train)
```

Out[50]:

RandomForestClassifier

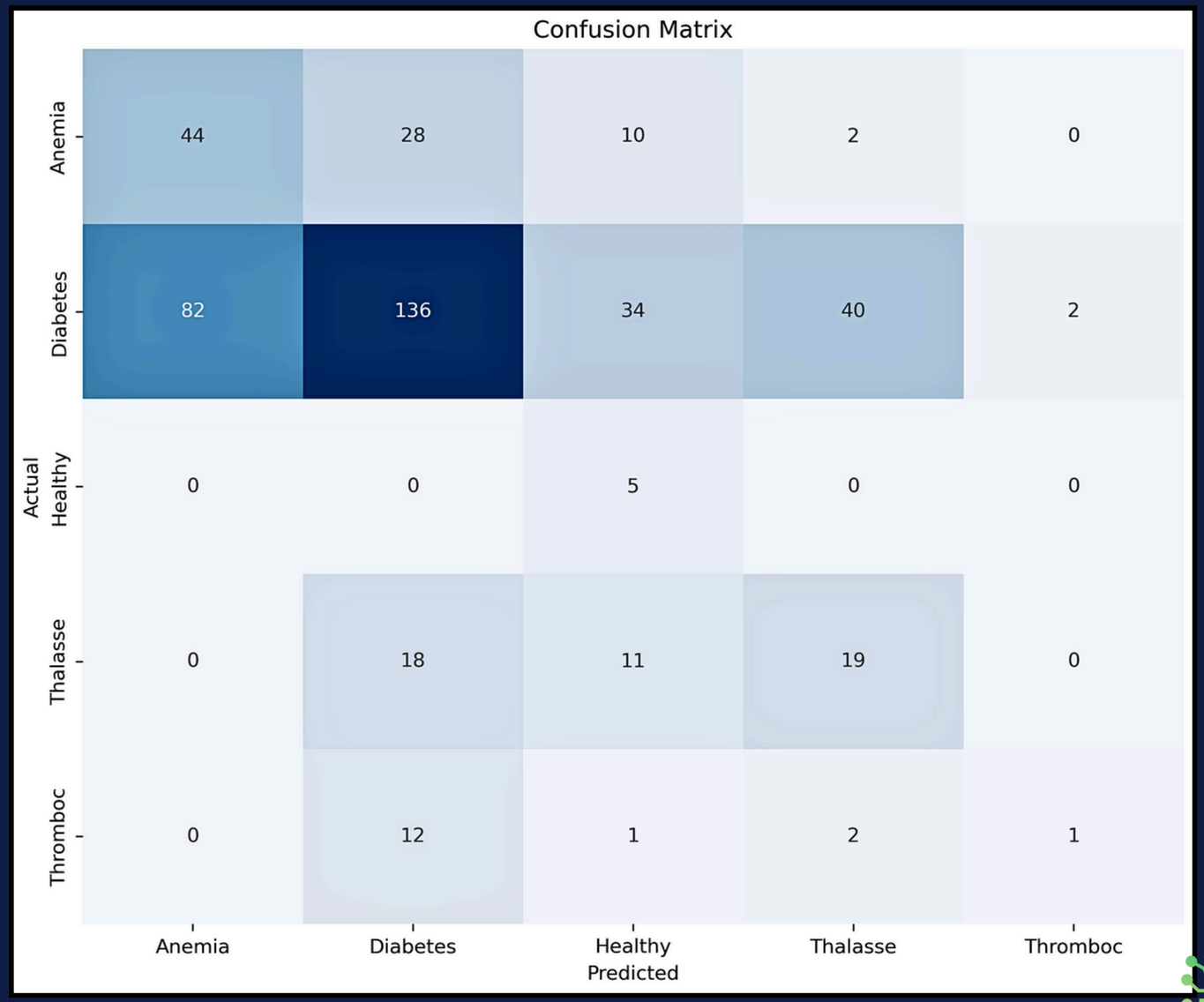
RandomForestClassifier(random_state=42)

Trained Random Forest Classifier:
Initialized and trained a Random Forest model on the training dataset with an accuracy of 45.86%

Random Forest Model Evaluation: Confusion Matrix

- The Random Forest model shows improvement over the Decision Tree model, indicating enhanced classification capabilities and better generalization.

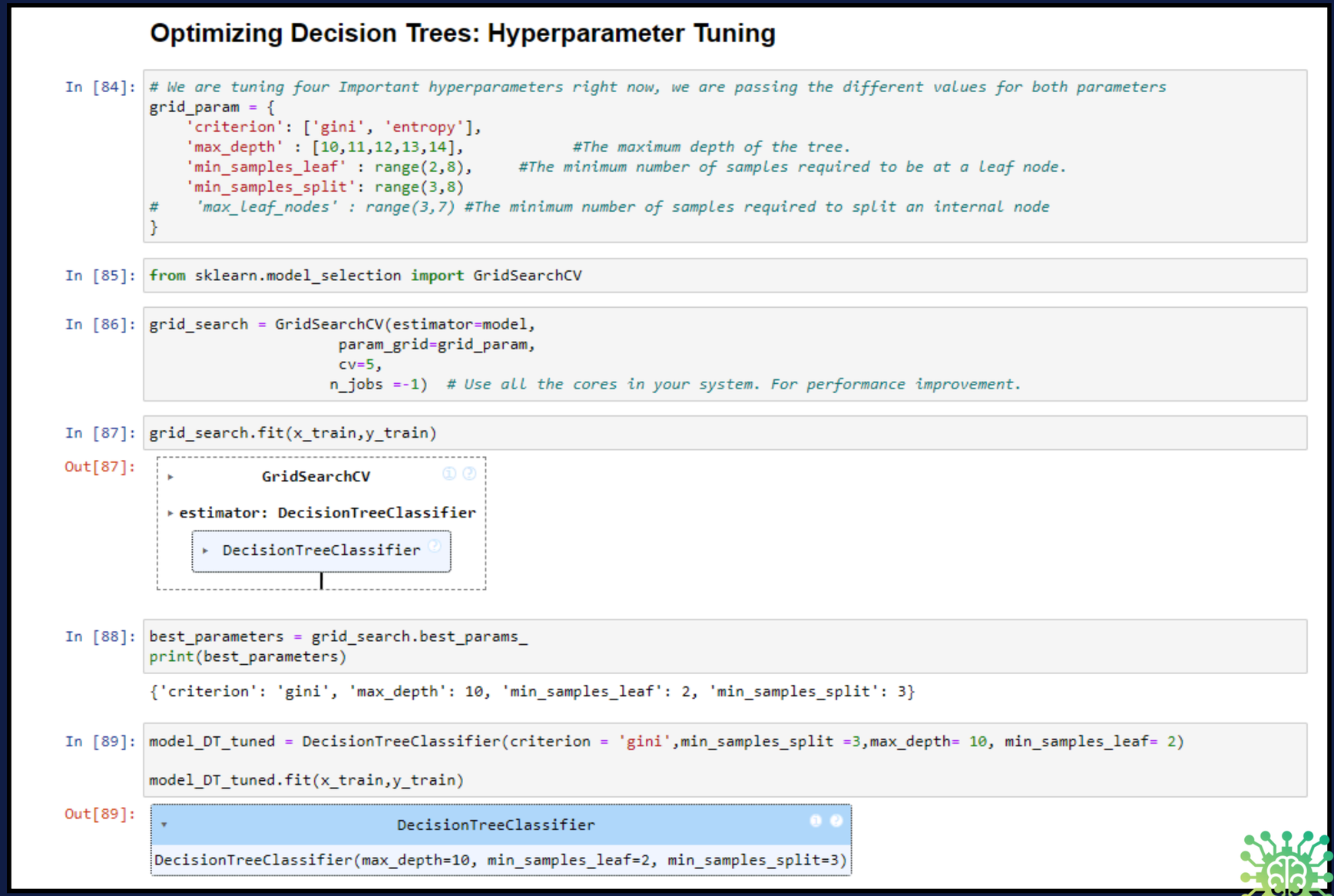
  - Precision (macro): 0.3534250250153579 4
  - Recall (macro): 0.4889455782312925 3
  - F1 Score (macro): 0.3151090639960355 5



Optimizing Decision Trees: Hyperparameter Tuning

Best Parameters Selected for Decision Tree Model:
- Criterion: Gini
- Max Depth: 10
- Min Samples Leaf: 2
- Min Samples Split: 3

These parameters were identified as optimal through the hyperparameter tuning process, aiming to enhance the model's performance and generalization ability.



Optimizing Decision Trees: Hyperparameter Tuning

In [84]:
```
# We are tuning four Important hyperparameters right now, we are passing the different values for both parameters
grid_param = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : [10,11,12,13,14],            #The maximum depth of the tree.
    'min_samples_leaf' : range(2,8),     #The minimum number of samples required to be at a leaf node.
    'min_samples_split': range(3,8)
#     'max_leaf_nodes' : range(3,7) #The minimum number of samples required to split an internal node
}
```

In [85]:
```
from sklearn.model_selection import GridSearchCV
```

In [86]:
```
grid_search = GridSearchCV(estimator=model,
                     param_grid=grid_param,
                     cv=5,
                    n_jobs =-1)  # Use all the cores in your system. For performance improvement.
```

In [87]:
```
grid_search.fit(x_train,y_train)
```

Out[87]:

GridSearchCV

estimator: DecisionTreeClassifier

DecisionTreeClassifier

In [88]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 3}
```

In [89]:
```
model_DT_tuned = DecisionTreeClassifier(criterion = 'gini',min_samples_split =3,max_depth= 10, min_samples_leaf= 2)

model_DT_tuned.fit(x_train,y_train)
```

Out[89]:

DecisionTreeClassifier

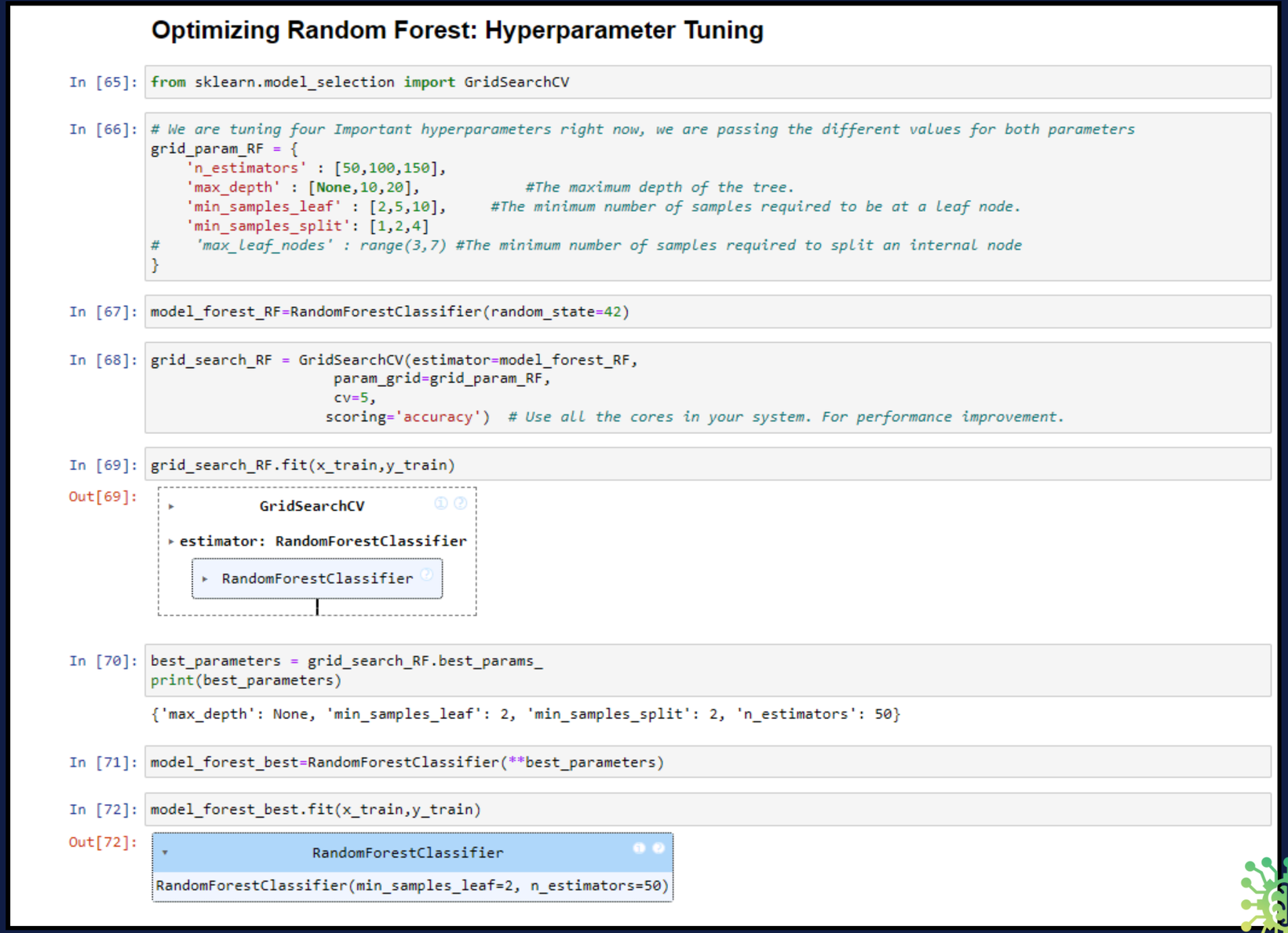
```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=3)
```

Optimizing Random Forest: Hyperparameter Tuning

Best Parameters Selected for Random Forest Model:

- **Max Depth: None**
- **Min Samples Leaf: 2**
- **Min Samples Split: 2**
- **Number of Estimators: 50**

These parameters were identified as optimal through the hyperparameter tuning process, aiming to enhance the model's performance and generalization ability.



Optimizing Random Forest: Hyperparameter Tuning

In [65]:
```
from sklearn.model_selection import GridSearchCV
```

In [66]:
```
# We are tuning four Important hyperparameters right now, we are passing the different values for both parameters
grid_param_RF = {
    'n_estimators' : [50,100,150],
    'max_depth' : [None,10,20],            #The maximum depth of the tree.
    'min_samples_leaf' : [2,5,10],     #The minimum number of samples required to be at a leaf node.
    'min_samples_split': [1,2,4]
#     'max_leaf_nodes' : range(3,7) #The minimum number of samples required to split an internal node
}
```

In [67]:
```
model_forest_RF=RandomForestClassifier(random_state=42)
```

In [68]:
```
grid_search_RF = GridSearchCV(estimator=model_forest_RF,
                      param_grid=grid_param_RF,
                      cv=5,
                     scoring='accuracy')  # Use all the cores in your system. For performance improvement.
```

In [69]:
```
grid_search_RF.fit(x_train,y_train)
```

Out[69]:

GridSearchCV

estimator: RandomForestClassifier

RandomForestClassifier

In [70]:
```
best_parameters = grid_search_RF.best_params_
print(best_parameters)
```

```
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
```

In [71]:
```
model_forest_best=RandomForestClassifier(**best_parameters)
```

In [72]:
```
model_forest_best.fit(x_train,y_train)
```

Out[72]:

RandomForestClassifier

RandomForestClassifier(min_samples_leaf=2, n_estimators=50)

Analyzing Hyper Tuned Models

Despite Hyperparameter Tuning Efforts, No Significant Improvement in Accuracy and Other Metrics Observed.

- Precision (macro): 0.3457618866527687
- Recall (macro): 0.5301020408163265
- F1 Score (macro): 0.34432257432455426

In [73]:
```
# Make predictions on the test set
y_pred_Best = model.predict(x_test)

y_pred_Best
```

Out[73]:
```
array([3, 1, 1, 1, 0, 0, 0, 3, 2, 1, 3, 3, 3, 0, 0, 1, 3, 1, 1, 2, 3, 0,
       2, 2, 3, 1, 0, 0, 3, 4, 0, 1, 1, 0, 1, 1, 4, 1, 0, 1, 0, 1, 1, 3,
       0, 4, 0, 3, 1, 1, 0, 3, 0, 1, 1, 1, 0, 4, 2, 2, 1, 1, 2, 1, 0, 0,
       1, 0, 1, 1, 1, 2, 1, 1, 3, 0, 0, 1, 1, 1, 0, 1, 3, 0, 0, 0, 1, 1,
       0, 0, 1, 3, 1, 0, 3, 0, 1, 0, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 3, 1, 0, 1, 1, 1, 1, 3, 1, 0, 0, 0, 1, 3, 2, 3, 2, 3, 3,
       1, 0, 3, 3, 1, 0, 3, 1, 2, 3, 1, 0, 0, 1, 0, 1, 2, 0, 0, 1, 1, 4,
       0, 0, 1, 1, 2, 1, 0, 1, 3, 1, 0, 1, 1, 1, 2, 1, 0, 1, 3, 0, 1, 1,
       1, 2, 1, 2, 0, 0, 0, 0, 1, 0, 1, 0, 3, 1, 3, 1, 0, 0, 0, 1, 1, 3,
       3, 1, 1, 1, 1, 4, 0, 1, 0, 3, 1, 3, 3, 3, 1, 3, 3, 2, 3, 3, 1, 2,
       0, 0, 0, 3, 1, 1, 1, 3, 0, 1, 3, 2, 2, 0, 2, 1, 1, 0, 1, 0, 0, 0,
       3, 3, 3, 1, 2, 2, 3, 0, 1, 3, 2, 0, 0, 4, 2, 0, 3, 0, 3, 1, 1, 1,
       3, 2, 0, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 3, 1, 2, 0, 0, 1, 0, 3, 0,
       0, 2, 2, 2, 2, 0, 1, 1, 3, 1, 0, 3, 0, 1, 3, 1, 1, 2, 0, 2, 3, 1,
       3, 3, 0, 3, 1, 0, 4, 1, 1, 1, 1, 0, 0, 3, 2, 1, 1, 0, 0, 1, 0, 3,
       2, 0, 0, 0, 1, 4, 0, 3, 1, 3, 0, 1, 2, 3, 0, 0, 4, 1, 4, 1, 1, 1,
       3, 0, 3, 0, 2, 3, 2, 1, 0, 0, 1, 0, 3, 1, 2, 0, 0, 1, 3, 0, 1, 2,
       3, 3, 1, 0, 3, 2, 0, 3, 2, 2, 3, 1, 4, 0, 4, 0, 1, 1, 1, 1, 2, 3,
       2, 3, 0, 3, 1, 2, 2, 2, 3, 0, 3, 0, 0, 0, 3, 1, 1, 1, 0, 1, 1, 1,
       1, 3, 1, 0, 1, 2, 4, 1, 0, 1, 2, 3, 1, 0, 1, 2, 0, 1, 0, 0, 3, 3,
       0, 0, 2, 1, 0, 2, 1])
```

In [74]:
```
# Evaluate the model using macro average for multiclass classification
precision = precision_score(y_test, y_pred_Best, average='macro')
recall = recall_score(y_test, y_pred_Best, average='macro')
f1 = f1_score(y_test, y_pred_Best, average='macro')

print("Precision (macro):", precision)
print("Recall (macro):", recall)
print("F1 Score (macro):", f1)
```

```
Precision (macro): 0.3457618866527687
Recall (macro): 0.5301020408163265
F1 Score (macro): 0.34432257432455426
```

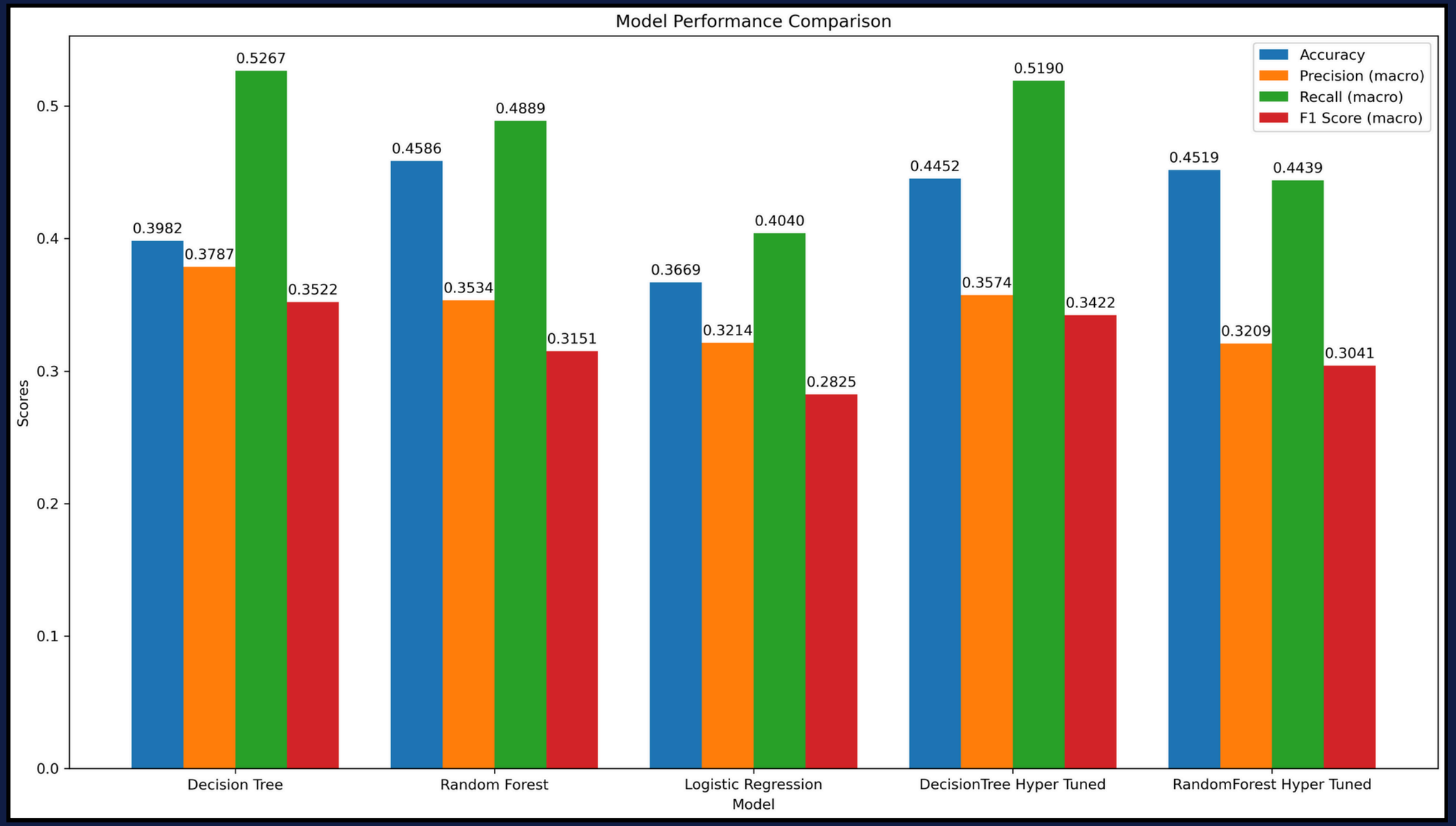
In [ ]:

MODEL PERFORMANCE COMPARISON

Model	Accuracy	Precision (macro)	Recall (macro)	F1 Score (macro)
Decision Tree	0.3982	0.3787	0.5267	0.3522
Random Forest	0.4586	0.3534	0.4889	0.3151
Logistic Regression	0.3669	0.3214	0.4040	0.2825
DT Hyper Tuned	0.4452	0.3574	0.5190	0.3422
RF Hyper Tuned	0.4519	0.3209	0.4439	0.3041

- Random Forest (RF) has the highest accuracy (0.4586) compared to Decision Tree (DT) (0.3982) and Logistic Regression (LR) (0.3669).
- Hyperparameter tuning improved DT's accuracy (0.4452) but had minimal impact on RF's accuracy (0.4519).
- Tuned DT shows balanced improvement, while untuned RF offers robust performance without extensive tuning.

MODEL PERFORMANCE COMPARISON



Conclusion

- Successfully developed a multiclass classification model for disease prediction with significant accuracy.
- Highlighted the potential improvements in early disease detection and patient management.
- Among the models tested (Decision Tree, Random Forest, Logistic Regression, Tuned Decision Tree, Tuned Random Forest), the best performing model is the Decision Tree.
- Despite hyperparameter tuning, no significant improvement in performance was observed.

Thank You

MAHAMMAD BASHEER K S

MENTORNESS