Intelligent Anemia Classification System

Course

Machine Learning

Team Members

Mahmoud Ahmed Abd El_Hameed

Anwar Ayman Anwar

Yahia Mahmoud Farouk

Table of Contents

1. Introduction

Anemia is a common blood disorder that affects the blood’s ability to carry oxygen, which may lead to serious health complications if not diagnosed early. Diagnosis is usually based on manual analysis of Complete Blood Count (CBC) tests, which can be time-consuming and prone to human error. Therefore, applying machine learning techniques can help automate anemia diagnosis and improve accuracy.

This project focuses on classifying anemia types using CBC parameters. Several supervised learning algorithms can be used for this task, such as Logistic Regression, Decision Trees, and Random Forest. A suitable classification algorithm is selected to learn patterns from the data and predict the anemia type effectively.

1.1 Define Task

The task is a **supervised classification problem** where CBC parameters are used to predict the type of anemia.

- **Input**: CBC features (HGB, PLT, WBC, RBC, MCV, MCH, MCHC, PCT)
- **Output**: Anemia type (**Diagnosis**)

1.2 Dataset Description

The dataset used in this project consists of **Complete Blood Count (CBC) records** that have been **manually diagnosed by medical professionals**. Each record represents a single patient case and includes multiple hematological parameters along with a labeled anemia diagnosis.

Dataset Characteristics:

- **Source**: Collected from multiple CBC test results From Kaggle
- **Labeling**: Diagnosis of anemia type provided manually by specialists.
- **Data Type**: Structured tabular data.
- **Learning Type**: Supervised learning.

Features Description:

- **HGB (Hemoglobin)**: Measures the amount of hemoglobin in the blood, which is essential for oxygen transport.
- **PLT (Platelets)**: Indicates the number of platelets involved in blood clotting.
- **WBC (White Blood Cells)**: Represents immune system activity.
- **RBC (Red Blood Cells)**: Indicates the number of red blood cells responsible for carrying oxygen.
- **MCV (Mean Corpuscular Volume)**: Average size of red blood cells.
- **MCH (Mean Corpuscular Hemoglobin)**: Average amount of hemoglobin per red blood cell.
- **MCHC (Mean Corpuscular Hemoglobin Concentration)**: Average concentration of hemoglobin in red blood cells.
- **PCT (Procalcitonin)**: A biomarker that may indicate bacterial infection and systemic inflammation.

Target Variable

The target variable is Diagnosis, which represents the type of anemia determined based on the CBC parameters.

The dataset includes the following 7 diagnostic classes:

(Healthy, Normocytic hypochromic anemia , Normocytic normochromic anemia, Iron deficiency anemia, Thrombocytopenia, Other microcytic anemia, Leukemia, Macrocytic anemia, Leukemia with thrombocytopenia)

2. Methodology

This project follows a supervised machine learning approach to classify anemia types using CBC parameters. The methodology includes data preprocessing, feature scaling, model training, and performance evaluation. Three classification algorithms were implemented and compared to determine their effectiveness in solving the anemia classification problem.

The dataset was split into training and testing sets, and feature scaling was applied to ensure fair distance-based and gradient-based learning.

2.1 Algorithms Used

2.1.1 k-Nearest Neighbors (KNN)

KNN is a distance-based classification algorithm that assigns a class to a new instance based on the majority class among its *k* nearest neighbors.
In this project, Euclidean distance is used after feature standardization. KNN does not require a training phase, but prediction time increases with dataset size.

2.1.2 Logistic Regression

Logistic Regression is a probabilistic classification algorithm that models the relationship between input features and class labels using a logistic function. For multi-class classification, the One-vs-Rest strategy is applied. The algorithm is efficient and works well when features have a linear relationship with the target variable.

2.1.3 Decision Tree

Decision Tree is a rule-based algorithm that recursively splits the dataset based on feature values to maximize class separation. The model selects the best feature at each node using impurity measures such as Gini Index. Decision Trees can capture non-linear relationships and provide feature importance, but may overfit without proper constraints.

2.2 Data Preprocessing

Data preprocessing was performed to prepare the CBC dataset for machine learning models. The target variable (**Diagnosis**) was separated from the input features. All features are numerical CBC parameters. Feature scaling was applied using **StandardScaler**, which standardizes the data to have zero mean and unit variance. This step is essential to ensure fair learning for distance-based algorithms such as KNN and to improve convergence for Logistic Regression.

2.3 Data Splitting

The dataset was divided into training and testing sets using
the **train_test_split** function from Scikit-learn.
An **80% / 20%** split was applied, where 80% of the data was used for training
and 20% for testing.
To maintain the original class distribution across both sets, **stratified
sampling** was used based on the target variable. A fixed **random state
(42)** was applied to ensure reproducibility of results.

3. Experimental Simulation

3.1 Environment

- **Programming Language**: Python
- **Libraries**: Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn, pickle
- **Development Environment**: Jupyter Notebook

3.2 Methods

3.2.1 Logistic Regression Model

The Logistic Regression model was trained on the scaled training dataset evaluated using accuracy, precision, recall, F1-score, and confusion matrix.

3.2.2 KNN Classifier

The KNN classifier was trained using **K = 3** neighbors. Feature scaling was applied before training due to the distance-based nature of the algorithm. evaluated using accuracy, precision, recall, F1-score, and confusion matrix.

3.2.3 Decision Tree Classifier

The Decision Tree classifier was trained on the training dataset to learn decision rules based on CBC features and evaluated using standard classification metrics evaluated using accuracy, precision, recall, F1-score, and confusion matrix.

4. Results and Technical Discussion

This section presents the experimental results obtained from the CBC dataset and discusses the performance of the applied machine learning models.

4.1 Results

Three machine learning models were trained and evaluated using the CBC dataset:

Logistic Regression

K-Nearest Neighbors (KNN)

Decision Tree

The dataset was split into training and testing sets to assess generalization performance.

Model	Test Accuracy
Logistic Regression	~71%
KNN	~80%
Decision Tree	99.61%

The **Decision Tree model achieved the highest accuracy** and was selected as the final model. The model was saved and successfully tested on unseen samples, confirming its reliability.

4.2 Result Analysis

The results show that Logistic Regression performed poorly due to the non-linear nature of CBC data. KNN achieved better performance by capturing local patterns but remained sensitive to feature overlap.

The Decision Tree model outperformed all other models, achieving near-perfect classification. Its ability to model complex decision boundaries and provide feature importance makes it suitable for medical diagnostic tasks.

4.3 Error Analysis

Most errors occurred in Logistic Regression and KNN models due to overlapping class features and possible class imbalance. The Decision Tree model showed minimal errors; however, its very high accuracy suggests a risk of overfitting. Further validation on external datasets is recommended.

5. Deployment

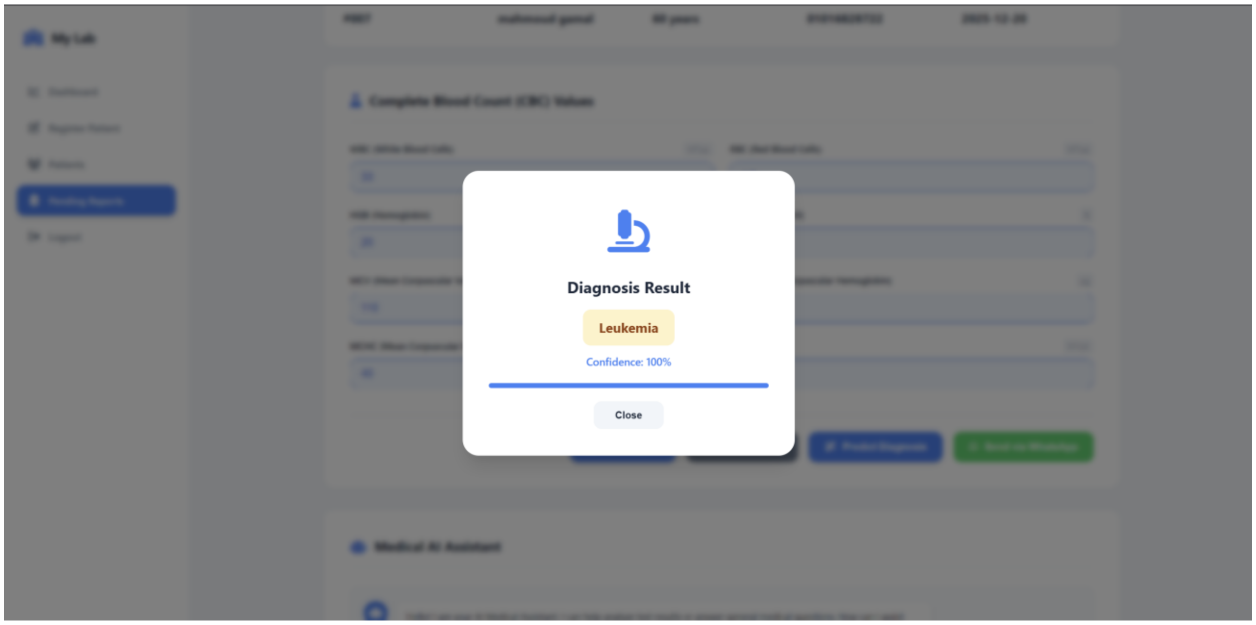
The trained Decision Tree model was saved and prepared for deployment. The deployment pipeline includes loading the model, preprocessing input CBC values, and generating predictions.

The model can be deployed as a web service or integrated into clinical decision support systems to provide fast and interpretable diagnostic assistance based on CBC test results.

We deploy this model into CBC Laboratory Management System that it can be **assist Doctors to diagnose** Anemia class for **patients**

"Laboratory Management System" is a web-based application built with a modern technology stack:

- **Frontend:** HTML, CSS, JavaScript (Vanilla).
- **Backend: FastAPI** (Python)
- **Database:** MySQL for robust data storage.
- **AI Integration:** Scikit-learn for ML model (Decision Tree) and Transformers for LLM-based chat**("SciReason-LFM2-2.6B")** model



6. Conclusions

This project demonstrated the effectiveness of supervised machine learning techniques in classifying anemia types using Complete Blood Count (CBC) parameters. Three models were evaluated: Logistic Regression, K-Nearest Neighbors (KNN), and Decision Tree.

Experimental results showed that the Decision Tree model achieved the highest performance with a test accuracy of 99.61%, making it the most suitable model for this task. Logistic Regression and KNN showed lower performance due to the non-linear nature of the data and overlapping class features.

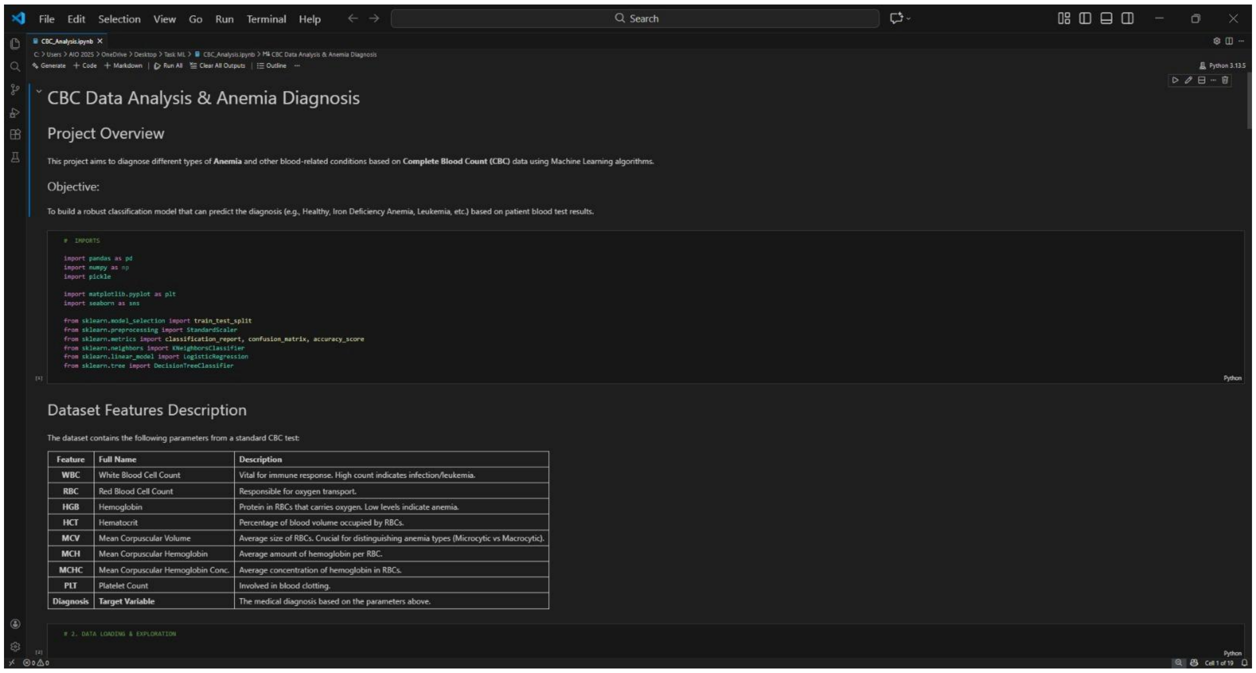
The trained Decision Tree model was saved and prepared for deployment, enabling fast and accurate anemia diagnosis. Future work includes validating the model on larger clinical datasets and improving robustness using advanced ensemble methods.

7. References

Kaggle **Anemia Types Classification DataSet** :

https://www.kaggle.com/datasets/ehababoelnaga/anemia-types-classification

8. Appendix


File Edit Selection View Go Run Terminal Help
Search
CBC_Analysis.ipynb
C: > Users > AIO 2025 > OneDrive > Desktop > Task ML > CBC_Analysis.ipynb > CBC Data Analysis & Anemia Diagnosis
Generate + Code + Markdown | Run All Clear All Outputs | Outline
Python 3.13.5

CBC Data Analysis & Anemia Diagnosis

Project Overview

This project aims to diagnose different types of **Anemia** and other blood-related conditions based on **Complete Blood Count (CBC)** data using Machine Learning algorithms.

Objective:

To build a robust classification model that can predict the diagnosis (e.g., Healthy, Iron Deficiency Anemia, Leukemia, etc.) based on patient blood test results.

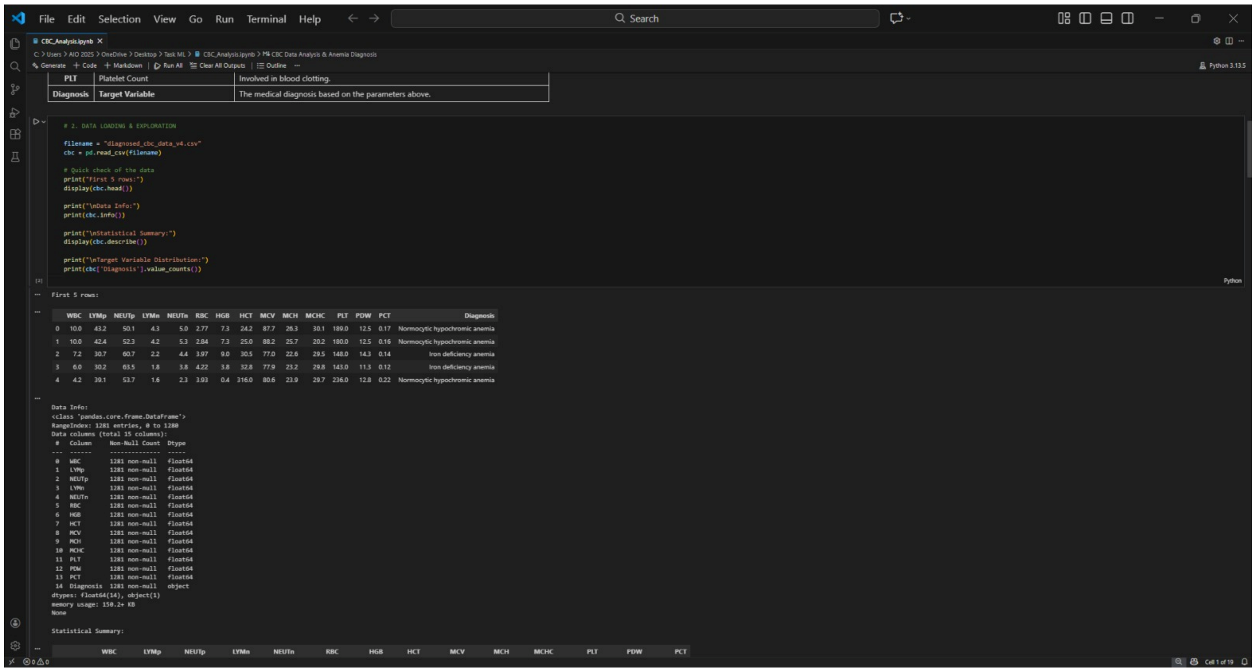
```python
#  IMPORTS

import pandas as pd
import numpy as np
import pickle

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

Dataset Features Description

The dataset contains the following parameters from a standard CBC test:

Feature	Full Name	Description
WBC	White Blood Cell Count	Vital for immune response. High count indicates infection/leukemia.
RBC	Red Blood Cell Count	Responsible for oxygen transport.
HGB	Hemoglobin	Protein in RBCs that carries oxygen. Low levels indicate anemia.
HCT	Hematocrit	Percentage of blood volume occupied by RBCs.
MCV	Mean Corpuscular Volume	Average size of RBCs. Crucial for distinguishing anemia types (Microcytic vs Macrocytic).
MCH	Mean Corpuscular Hemoglobin	Average amount of hemoglobin per RBC.
MCHC	Mean Corpuscular Hemoglobin Conc.	Average concentration of hemoglobin in RBCs.
PLT	Platelet Count	Involved in blood clotting.
Diagnosis	**Target Variable**	The medical diagnosis based on the parameters above.

```python
# 2. DATA LOADING & EXPLORATION
```


File Edit Selection View Go Run Terminal Help
Search
CBC_Analysis.ipynb
C: > Users > AIO 2025 > OneDrive > Desktop > Task ML > CBC_Analysis.ipynb > CBC Data Analysis & Anemia Diagnosis
Generate + Code + Markdown | Run All Clear All Outputs | Outline
Python 3.13.5

PLT	Platelet Count	Involved in blood clotting.
Diagnosis	**Target Variable**	The medical diagnosis based on the parameters above.

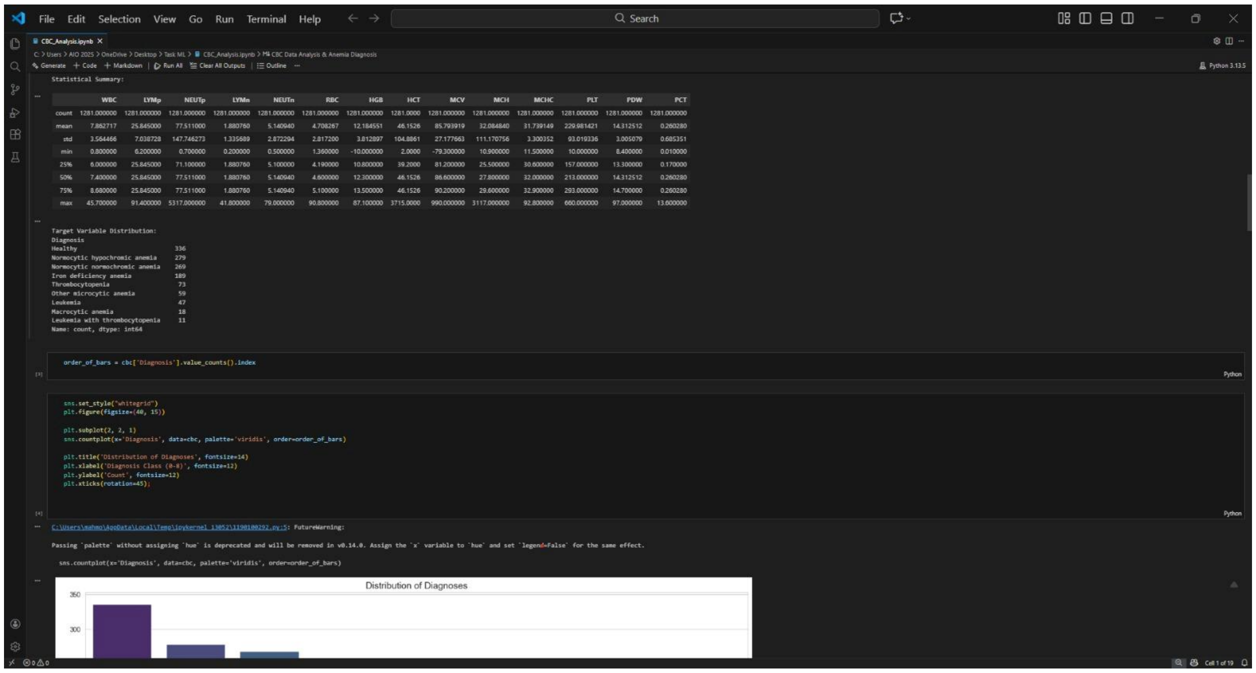
```python
# 2. DATA LOADING & EXPLORATION

filename = "diagnosed_cbc_data_v4.csv"
cbc = pd.read_csv(filename)

# Quick check of the data
print("First 5 rows:")
display(cbc.head())

print("\nData Info:")
print(cbc.info())

print("\nStatistical Summary:")
display(cbc.describe())

print("\nTarget Variable Distribution:")
print(cbc['Diagnosis'].value_counts())
```

```
First 5 rows:
```

	WBC	LYMp	NEUTp	LYMn	NEUTn	RBC	HGB	HCT	MCV	MCH	MCHC	PLT	PDW	PCT	Diagnosis
0	10.0	43.2	50.1	4.3	5.0	2.77	7.3	24.2	87.7	26.3	30.1	189.0	12.5	0.17	Normocytic hypochromic anemia
1	10.0	42.4	52.3	4.2	5.3	2.84	7.3	25.0	88.2	25.7	20.2	180.0	12.5	0.16	Normocytic hypochromic anemia
2	7.2	30.7	60.7	2.2	4.4	3.97	9.0	30.5	77.0	22.6	29.5	148.0	14.3	0.14	Iron deficiency anemia
3	6.0	30.2	63.5	1.8	3.8	4.22	3.8	32.8	77.9	23.2	29.8	143.0	11.3	0.12	Iron deficiency anemia
4	4.2	39.1	53.7	1.6	2.3	3.93	0.4	316.0	80.6	23.9	29.7	236.0	12.8	0.22	Normocytic hypochromic anemia

```
Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1281 entries, 0 to 1280
Data columns (total 15 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   WBC        1281 non-null   float64
 1   LYMp       1281 non-null   float64
 2   NEUTp      1281 non-null   float64
 3   LYMn       1281 non-null   float64
 4   NEUTn      1281 non-null   float64
 5   RBC        1281 non-null   float64
 6   HGB        1281 non-null   float64
 7   HCT        1281 non-null   float64
 8   MCV        1281 non-null   float64
 9   MCH        1281 non-null   float64
 10  MCHC       1281 non-null   float64
 11  PLT        1281 non-null   float64
 12  PDW        1281 non-null   float64
 13  PCT        1281 non-null   float64
 14  Diagnosis  1281 non-null   object 
dtypes: float64(14), object(1)
memory usage: 150.2+ KB
None

Statistical Summary:
```

	WBC	LYMp	NEUTp	LYMn	NEUTn	RBC	HGB	HCT	MCV	MCH	MCHC	PLT	PDW	PCT


File Edit Selection View Go Run Terminal Help
Search
CBC_Analysis.ipynb
C: > Users > AIO 2025 > OneDrive > Desktop > Task ML > CBC_Analysis.ipynb > CBC Data Analysis & Anemia Diagnosis
Generate + Code + Markdown Run All Clear All Outputs Outline
Python 3.13.5

```
Statistical Summary:
```

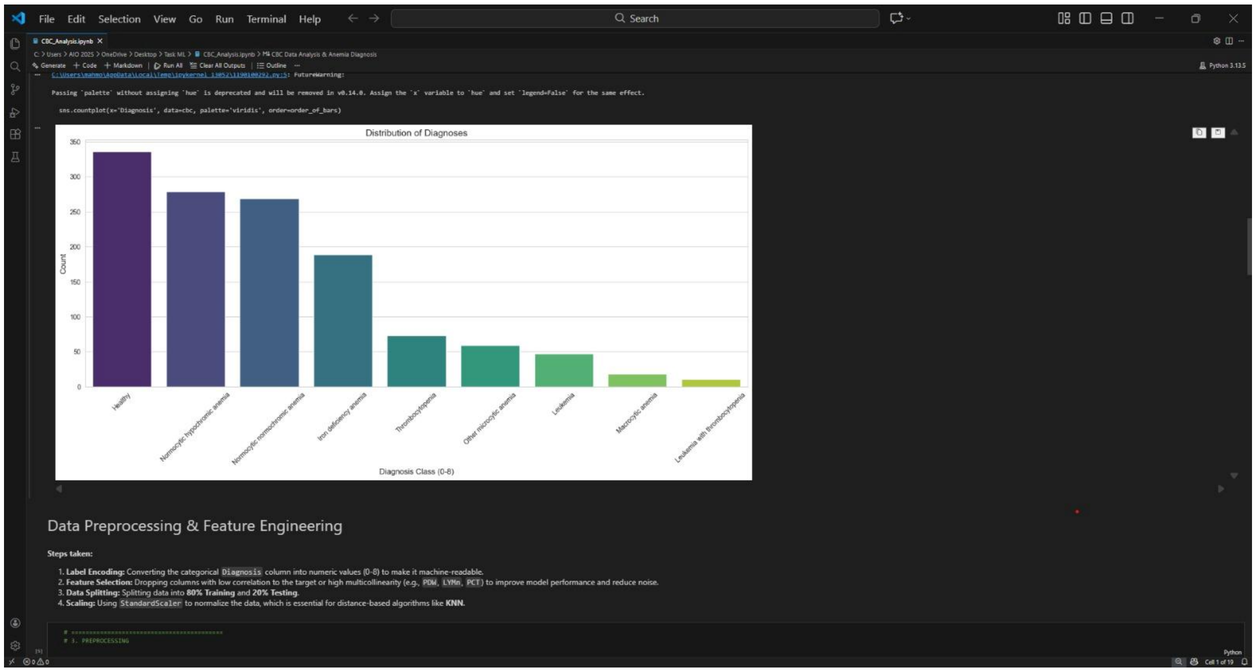
	WBC	LYMp	NEUTp	LYMn	NEUTn	RBC	HGB	HCT	MCV	MCH	MCHC	PLT	PDW	PCT
count	1281.000000	1281.000000	1281.000000	1281.000000	1281.000000	1281.000000	1281.000000	1281.0000	1281.000000	1281.000000	1281.000000	1281.000000	1281.000000	1281.000000
mean	7.862717	25.845000	77.511000	1.880760	5.140940	4.708267	12.184551	46.1526	85.793919	32.084840	31.739149	229.981421	14.312512	0.260280
std	3.564466	7.038728	147.746273	1.335689	2.872294	2.817200	3.812897	104.8861	27.177663	111.170756	3.300352	93.019336	3.005079	0.685351
min	0.800000	6.200000	0.700000	0.200000	0.500000	1.360000	-10.000000	2.0000	-79.300000	10.900000	11.500000	10.000000	8.400000	0.010000
25%	6.000000	25.845000	71.100000	1.880760	5.100000	4.190000	10.800000	39.2000	81.200000	25.500000	30.600000	157.000000	13.300000	0.170000
50%	7.400000	25.845000	77.511000	1.880760	5.140940	4.600000	12.300000	46.1526	86.600000	27.800000	32.000000	213.000000	14.312512	0.260280
75%	8.680000	25.845000	77.511000	1.880760	5.140940	5.100000	13.500000	46.1526	90.200000	29.600000	32.900000	293.000000	14.700000	0.260280
max	45.700000	91.400000	5317.000000	41.800000	79.000000	90.800000	87.100000	3715.0000	990.000000	3117.000000	92.800000	660.000000	97.000000	13.600000

```
Target Variable Distribution:
Diagnosis
Healthy                           336
Normocytic hypochromic anemia     279
Normocytic normochromic anemia    269
Iron deficiency anemia            189
Thrombocytopenia                   73
Other microcytic anemia            59
Leukemia                           47
Macrocytic anemia                  18
Leukemia with thrombocytopenia     11
Name: count, dtype: int64
```

```python
order_of_bars = cbc['Diagnosis'].value_counts().index
```

```python
sns.set_style("whitegrid")
plt.figure(figsize=(40, 15))

plt.subplot(2, 2, 1)
sns.countplot(x='Diagnosis', data=cbc, palette='viridis', order=order_of_bars)

plt.title('Distribution of Diagnoses', fontsize=14)
plt.xlabel('Diagnosis Class (0-8)', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=45);
```

```
C:\Users\mahmo\AppData\Local\Temp\ipykernel_13052\1190100292.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='Diagnosis', data=cbc, palette='viridis', order=order_of_bars)
```



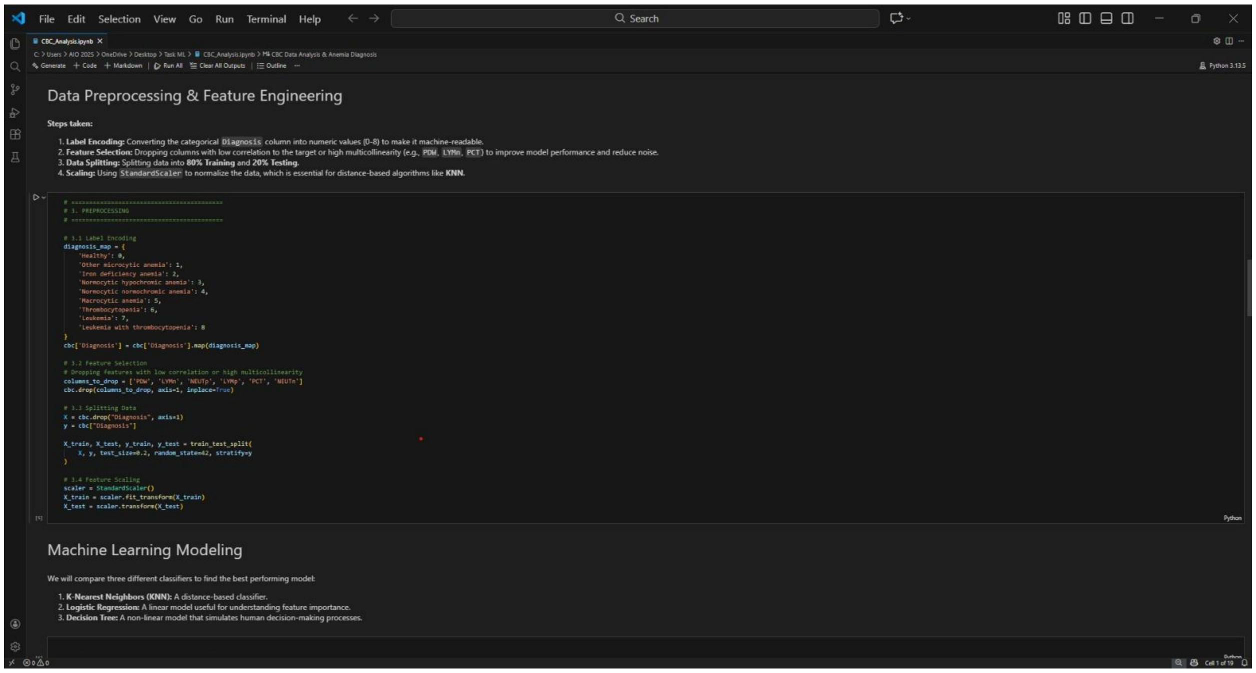
```
C:\Users\mahmo\AppData\Local\Temp\ipykernel_13052\1190100292.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='Diagnosis', data=cbc, palette='viridis', order=order_of_bars)
```

Data Preprocessing & Feature Engineering

Steps taken:

1. **Label Encoding:** Converting the categorical `Diagnosis` column into numeric values (0-8) to make it machine-readable.
2. **Feature Selection:** Dropping columns with low correlation to the target or high multicollinearity (e.g., `PDW`, `LYMn`, `PCT`) to improve model performance and reduce noise.
3. **Data Splitting:** Splitting data into **80% Training** and **20% Testing**.
4. **Scaling:** Using `StandardScaler` to normalize the data, which is essential for distance-based algorithms like **KNN**.

```python
# ==========================================
# 3. PREPROCESSING
```



Data Preprocessing & Feature Engineering

Steps taken:

1. **Label Encoding:** Converting the categorical `Diagnosis` column into numeric values (0-8) to make it machine-readable.
2. **Feature Selection:** Dropping columns with low correlation to the target or high multicollinearity (e.g., `PDW`, `LYMn`, `PCT`) to improve model performance and reduce noise.
3. **Data Splitting:** Splitting data into **80% Training** and **20% Testing**.
4. **Scaling:** Using `StandardScaler` to normalize the data, which is essential for distance-based algorithms like **KNN**.

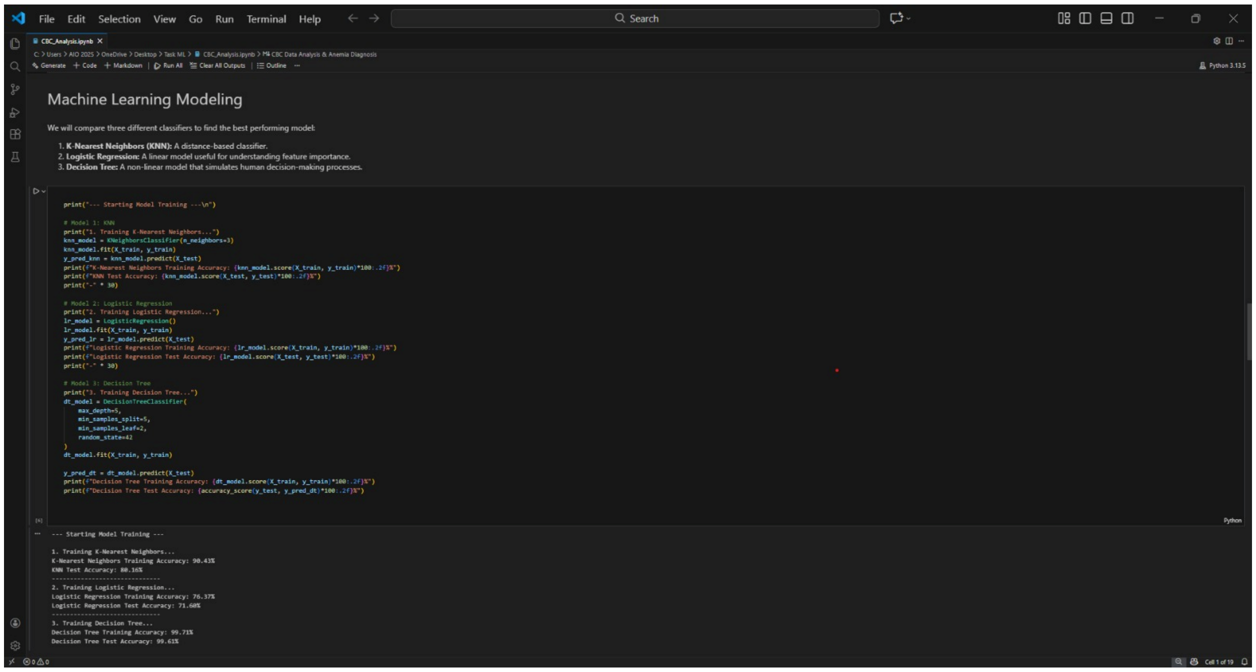
```python
# ==========================================
# 3. PREPROCESSING
# ==========================================

# 3.1 Label Encoding
diagnosis_map = {
    'Healthy': 0,
    'Other microcytic anemia': 1,
    'Iron deficiency anemia': 2,
    'Normocytic hypochromic anemia': 3,
    'Normocytic normochromic anemia': 4,
    'Macrocytic anemia': 5,
    'Thrombocytopenia': 6,
    'Leukemia': 7,
    'Leukemia with thrombocytopenia': 8
}
cbc['Diagnosis'] = cbc['Diagnosis'].map(diagnosis_map)

# 3.2 Feature Selection
# Dropping features with low correlation or high multicollinearity
columns_to_drop = ['PDW', 'LYMn', 'NEUTp', 'LYMp', 'PCT', 'NEUTn']
cbc.drop(columns_to_drop, axis=1, inplace=True)

# 3.3 Splitting Data
X = cbc.drop("Diagnosis", axis=1)
y = cbc["Diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 3.4 Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[5]

Machine Learning Modeling

We will compare three different classifiers to find the best performing model:

1. **K-Nearest Neighbors (KNN):** A distance-based classifier.
2. **Logistic Regression:** A linear model useful for understanding feature importance.
3. **Decision Tree:** A non-linear model that simulates human decision-making processes.



Machine Learning Modeling

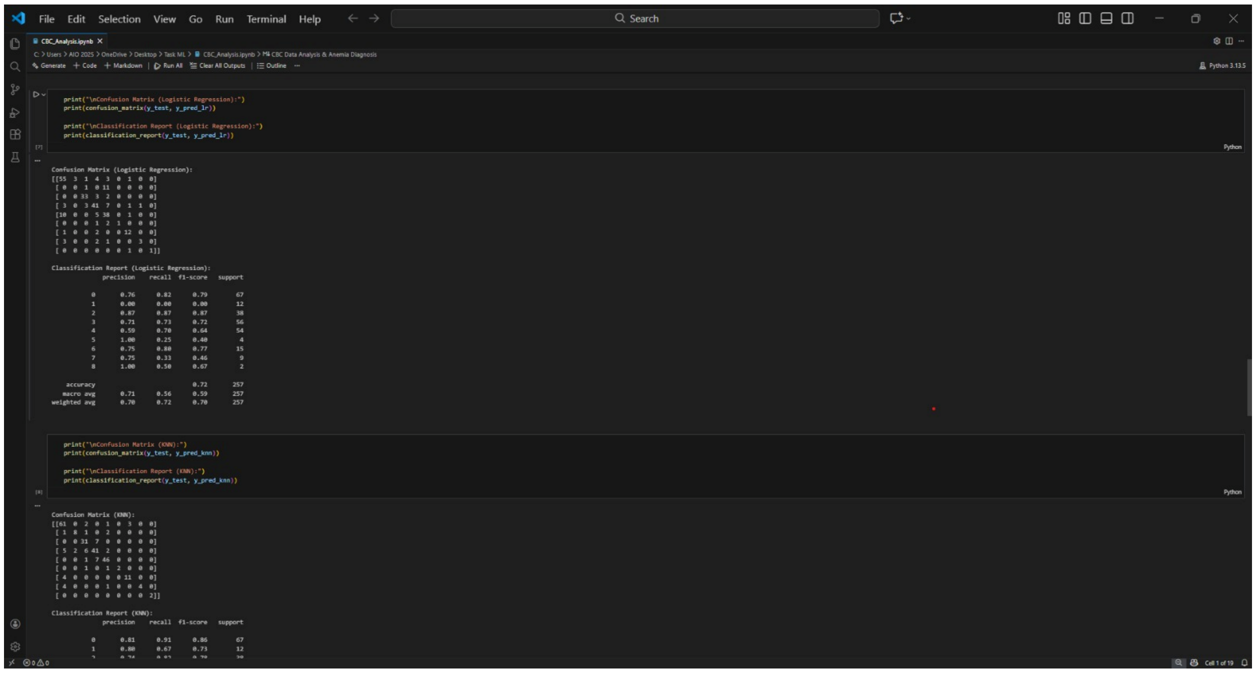
We will compare three different classifiers to find the best performing model:

1. **K-Nearest Neighbors (KNN):** A distance-based classifier.
2. **Logistic Regression:** A linear model useful for understanding feature importance.
3. **Decision Tree:** A non-linear model that simulates human decision-making processes.

```python
print("--- Starting Model Training ---\n")

# Model 1: KNN
print("1. Training K-Nearest Neighbors...")
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
print(f"K-Nearest Neighbors Training Accuracy: {knn_model.score(X_train, y_train)*100:.2f}%")
print(f"KNN Test Accuracy: {knn_model.score(X_test, y_test)*100:.2f}%")
print("-" * 30)

# Model 2: Logistic Regression
print("2. Training Logistic Regression...")
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
print(f"Logistic Regression Training Accuracy: {lr_model.score(X_train, y_train)*100:.2f}%")
print(f"Logistic Regression Test Accuracy: {lr_model.score(X_test, y_test)*100:.2f}%")
print("-" * 30)

# Model 3: Decision Tree
print("3. Training Decision Tree...")
dt_model = DecisionTreeClassifier(
    max_depth=5,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42
)
dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)
print(f"Decision Tree Training Accuracy: {dt_model.score(X_train, y_train)*100:.2f}%")
print(f"Decision Tree Test Accuracy: {accuracy_score(y_test, y_pred_dt)*100:.2f}%")
```

[6]

```
--- Starting Model Training ---

1. Training K-Nearest Neighbors...
K-Nearest Neighbors Training Accuracy: 90.43%
KNN Test Accuracy: 80.16%
------------------------------
2. Training Logistic Regression...
Logistic Regression Training Accuracy: 76.37%
Logistic Regression Test Accuracy: 71.60%
------------------------------
3. Training Decision Tree...
Decision Tree Training Accuracy: 99.71%
Decision Tree Test Accuracy: 99.61%
```

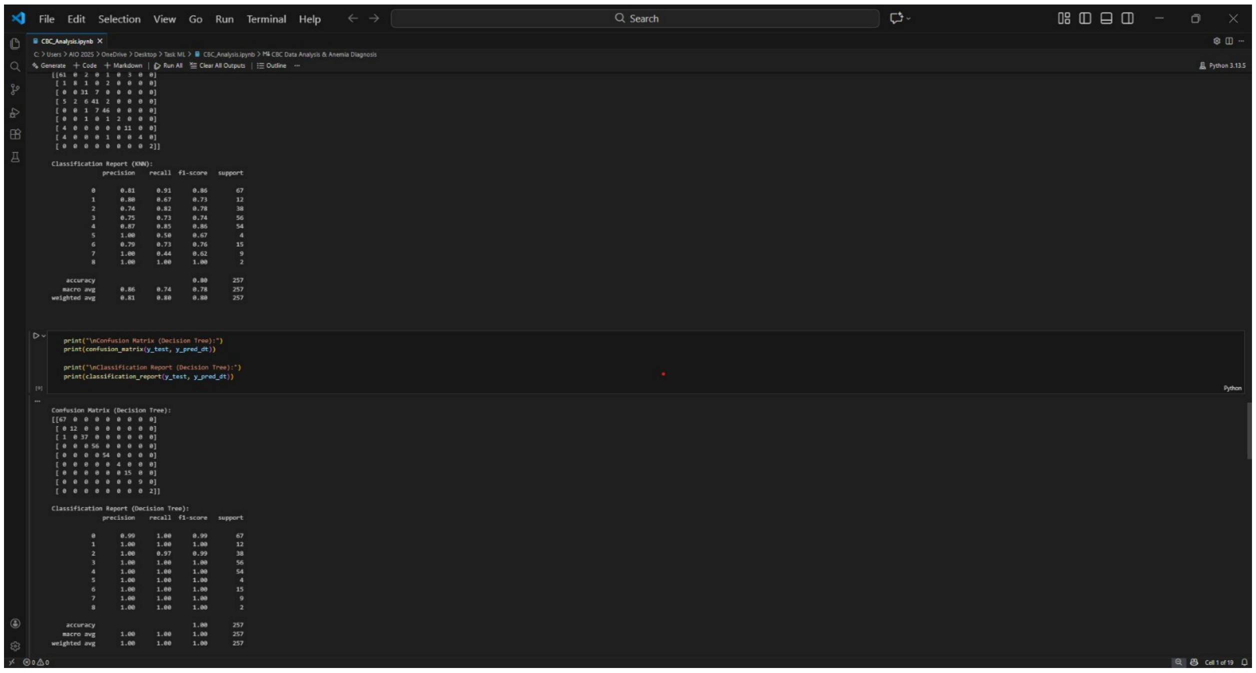


```
print("\nConfusion Matrix (Logistic Regression):")
print(confusion_matrix(y_test, y_pred_lr))

print("\nClassification Report (Logistic Regression):")
print(classification_report(y_test, y_pred_lr))
```

```
Confusion Matrix (Logistic Regression):
[[55  3  1  4  3  0  1  0  0]
 [ 0  0  1  0 11  0  0  0  0]
 [ 0  0 33  3  2  0  0  0  0]
 [ 3  0  3 41  7  0  1  1  0]
 [10  0  0  5 38  0  1  0  0]
 [ 0  0  0  1  2  1  0  0  0]
 [ 1  0  0  2  0  0 12  0  0]
 [ 3  0  0  2  1  0  0  3  0]
 [ 0  0  0  0  0  0  1  0  1]]

Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.76      0.82      0.79        67
           1       0.00      0.00      0.00        12
           2       0.87      0.87      0.87        38
           3       0.71      0.73      0.72        56
           4       0.59      0.70      0.64        54
           5       1.00      0.25      0.40         4
           6       0.75      0.80      0.77        15
           7       0.75      0.33      0.46         9
           8       1.00      0.50      0.67         2

    accuracy                           0.72       257
   macro avg       0.71      0.56      0.59       257
weighted avg       0.70      0.72      0.70       257
```

```
print("\nConfusion Matrix (KNN):")
print(confusion_matrix(y_test, y_pred_knn))

print("\nClassification Report (KNN):")
print(classification_report(y_test, y_pred_knn))
```

```
Confusion Matrix (KNN):
[[61  0  2  0  1  0  3  0  0]
 [ 1  8  1  0  2  0  0  0  0]
 [ 0  0 31  7  0  0  0  0  0]
 [ 5  2  6 41  2  0  0  0  0]
 [ 0  0  1  7 46  0  0  0  0]
 [ 0  0  1  0  1  2  0  0  0]
 [ 4  0  0  0  0  0 11  0  0]
 [ 4  0  0  0  1  0  0  4  0]
 [ 0  0  0  0  0  0  0  0  2]]

Classification Report (KNN):
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        67
           1       0.80      0.67      0.73        12
```

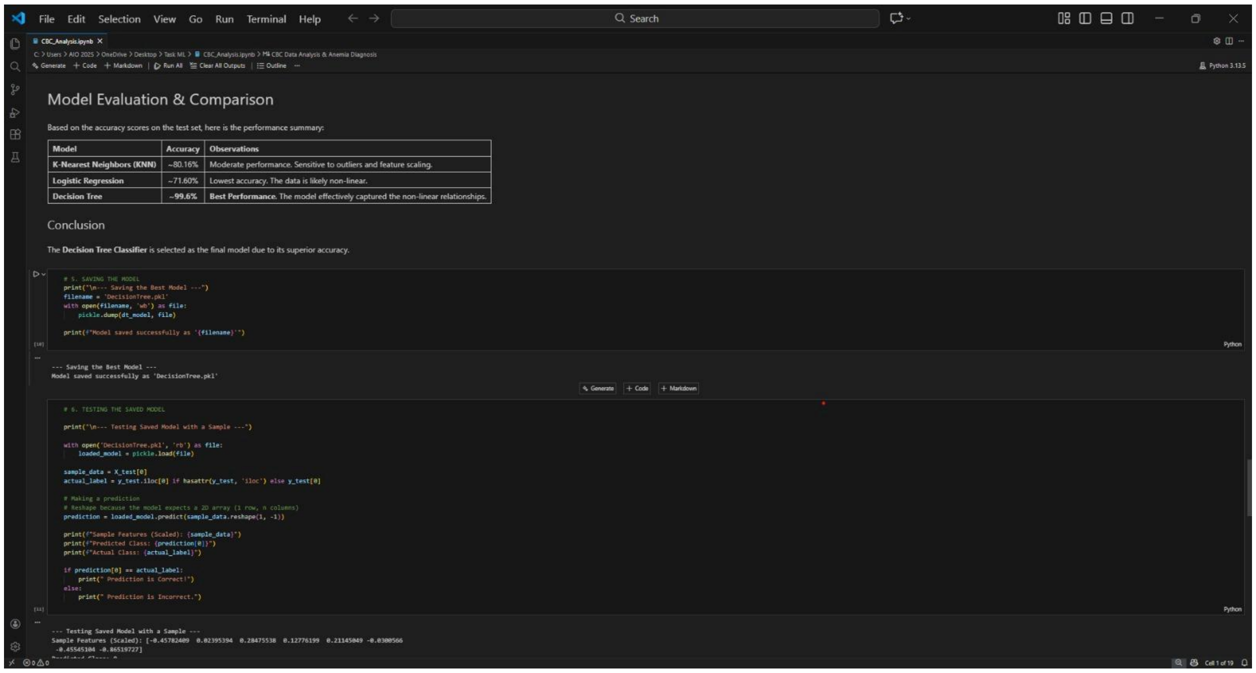


```
[[61  0  2  0  1  0  3  0  0]
 [ 1  8  1  0  2  0  0  0  0]
 [ 0  0 31  7  0  0  0  0  0]
 [ 5  2  6 41  2  0  0  0  0]
 [ 0  0  1  7 46  0  0  0  0]
 [ 0  0  1  0  1  2  0  0  0]
 [ 4  0  0  0  0  0 11  0  0]
 [ 4  0  0  0  1  0  0  4  0]
 [ 0  0  0  0  0  0  0  0  2]]
```

```
Classification Report (KNN):
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        67
           1       0.80      0.67      0.73        12
           2       0.74      0.82      0.78        38
           3       0.75      0.73      0.74        56
           4       0.87      0.85      0.86        54
           5       1.00      0.50      0.67         4
           6       0.79      0.73      0.76        15
           7       1.00      0.44      0.62         9
           8       1.00      1.00      1.00         2

    accuracy                           0.80       257
   macro avg       0.86      0.74      0.78       257
weighted avg       0.81      0.80      0.80       257
```

```
print("\nConfusion Matrix (Decision Tree):")
print(confusion_matrix(y_test, y_pred_dt))

print("\nClassification Report (Decision Tree):")
print(classification_report(y_test, y_pred_dt))
```

```
Confusion Matrix (Decision Tree):
[[67  0  0  0  0  0  0  0  0]
 [ 0 12  0  0  0  0  0  0  0]
 [ 1  0 37  0  0  0  0  0  0]
 [ 0  0  0 56  0  0  0  0  0]
 [ 0  0  0  0 54  0  0  0  0]
 [ 0  0  0  0  0  4  0  0  0]
 [ 0  0  0  0  0  0 15  0  0]
 [ 0  0  0  0  0  0  0  9  0]
 [ 0  0  0  0  0  0  0  0  2]]

Classification Report (Decision Tree):
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        67
           1       1.00      1.00      1.00        12
           2       1.00      0.97      0.99        38
           3       1.00      1.00      1.00        56
           4       1.00      1.00      1.00        54
           5       1.00      1.00      1.00         4
           6       1.00      1.00      1.00        15
           7       1.00      1.00      1.00         9
           8       1.00      1.00      1.00         2

    accuracy                           1.00       257
   macro avg       1.00      1.00      1.00       257
weighted avg       1.00      1.00      1.00       257
```



Model Evaluation & Comparison

Based on the accuracy scores on the test set, here is the performance summary:

Model	Accuracy	Observations
K-Nearest Neighbors (KNN)	~80.16%	Moderate performance. Sensitive to outliers and feature scaling.
Logistic Regression	~71.60%	Lowest accuracy. The data is likely non-linear.
Decision Tree	**~99.6%**	**Best Performance.** The model effectively captured the non-linear relationships.

Conclusion

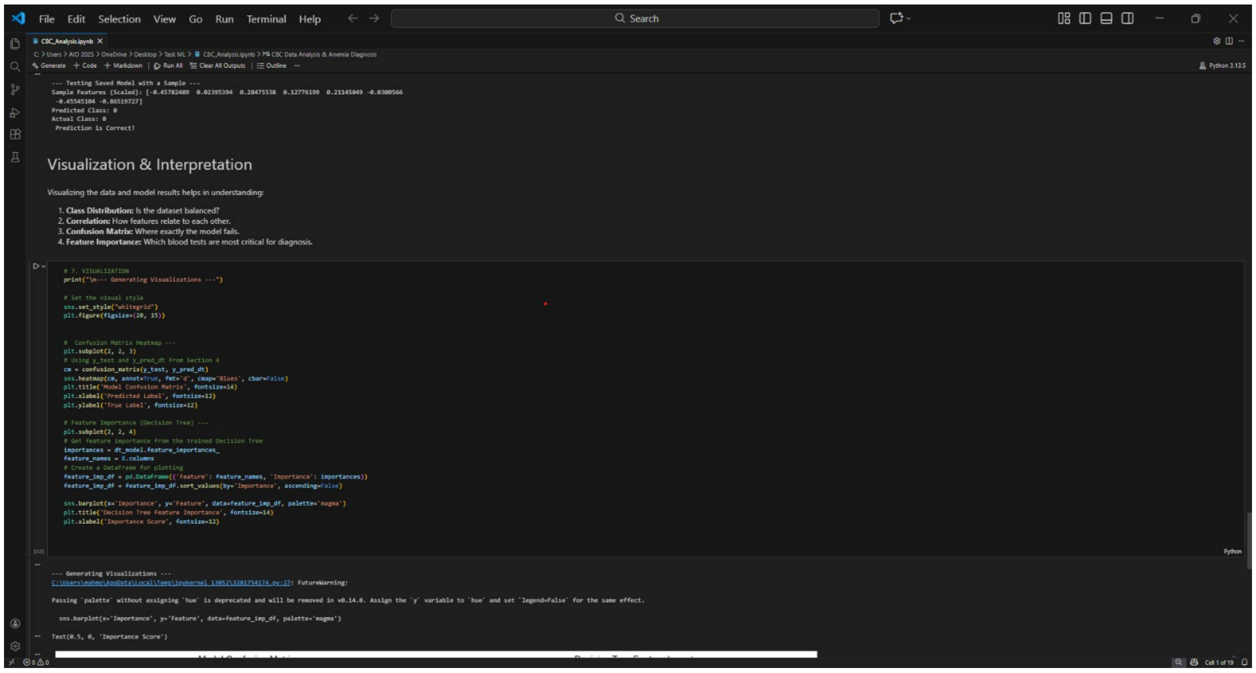
The **Decision Tree Classifier** is selected as the final model due to its superior accuracy.

```python
# 5. SAVING THE MODEL
print("\n--- Saving the Best Model ---")
filename = 'DecisionTree.pkl'
with open(filename, 'wb') as file:
    pickle.dump(dt_model, file)

print(f"Model saved successfully as '{filename}'")
```

```
--- Saving the Best Model ---
Model saved successfully as 'DecisionTree.pkl'
```

```python
# 6. TESTING THE SAVED MODEL

print("\n--- Testing Saved Model with a Sample ---")

with open('DecisionTree.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

sample_data = X_test[0]
actual_label = y_test.iloc[0] if hasattr(y_test, 'iloc') else y_test[0]

# Making a prediction
# Reshape because the model expects a 2D array (1 row, n columns)
prediction = loaded_model.predict(sample_data.reshape(1, -1))

print(f"Sample Features (Scaled): {sample_data}")
print(f"Predicted Class: {prediction[0]}")
print(f"Actual Class: {actual_label}")

if prediction[0] == actual_label:
    print(" Prediction is Correct!")
else:
    print(" Prediction is Incorrect.")
```

```
--- Testing Saved Model with a Sample ---
Sample Features (Scaled): [-0.45782409  0.02395394  0.28475538  0.12776199  0.21145049 -0.0300566
 -0.45545104 -0.86519727]
```



```
--- Testing Saved Model with a Sample ---
Sample Features (Scaled): [-0.45782409  0.02395394  0.28475538  0.12776199  0.21145049 -0.0300566
 -0.45545104 -0.86519727]
Predicted Class: 0
Actual Class: 0
 Prediction is Correct!
```

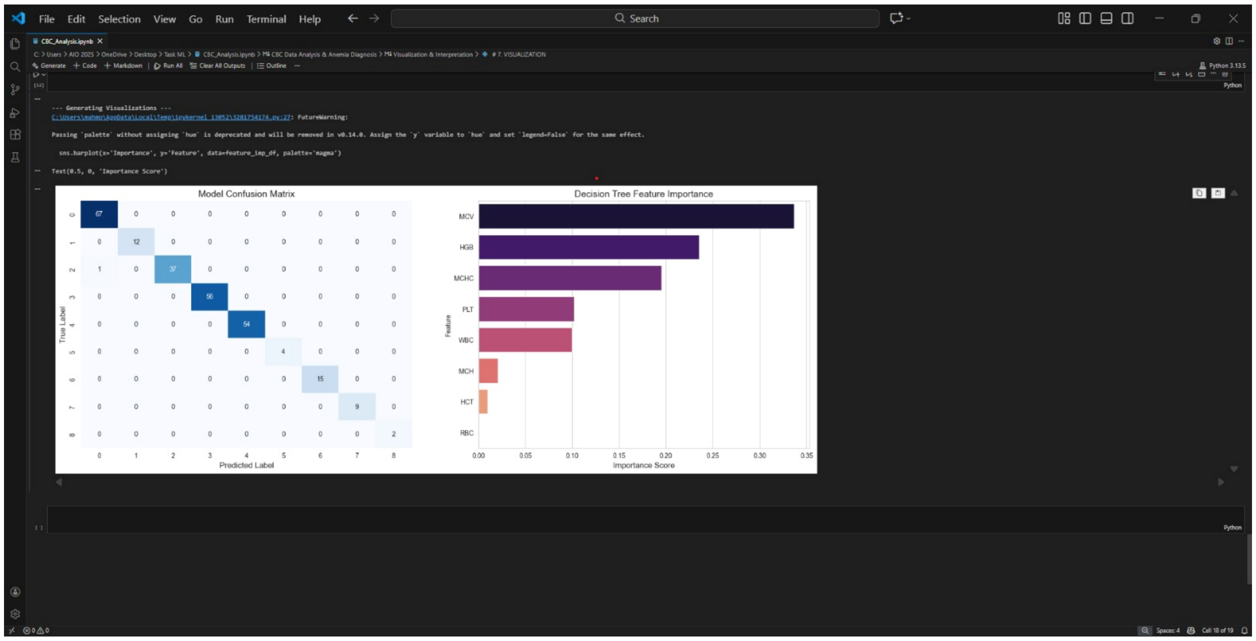
Visualization & Interpretation

Visualizing the data and model results helps in understanding:

1. **Class Distribution:** Is the dataset balanced?
2. **Correlation:** How features relate to each other.
3. **Confusion Matrix:** Where exactly the model fails.
4. **Feature Importance:** Which blood tests are most critical for diagnosis.

```python
# 7. VISUALIZATION
print("\n--- Generating Visualizations ---")

# Set the visual style
sns.set_style("whitegrid")
plt.figure(figsize=(20, 15))


#  Confusion Matrix Heatmap ---
plt.subplot(2, 2, 3)
# Using y_test and y_pred_dt from Section 4
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Model Confusion Matrix', fontsize=14)
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('True Label', fontsize=12)

# Feature Importance (Decision Tree) ---
plt.subplot(2, 2, 4)
# Get feature importance from the trained Decision Tree
importances = dt_model.feature_importances_
feature_names = X.columns
# Create a DataFrame for plotting
feature_imp_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_imp_df = feature_imp_df.sort_values(by='Importance', ascending=False)

sns.barplot(x='Importance', y='Feature', data=feature_imp_df, palette='magma')
plt.title('Decision Tree Feature Importance', fontsize=14)
plt.xlabel('Importance Score', fontsize=12)
```

```
--- Generating Visualizations ---
C:\Users\mahmo\AppData\Local\Temp\ipykernel_13052\3281754174.py:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Importance', y='Feature', data=feature_imp_df, palette='magma')

Text(0.5, 0, 'Importance Score')
```


File
Edit
Selection
View
Go
Run
Terminal
Help
Search
CBC_Analysis.ipynb
C: > Users > AIO 2025 > OneDrive > Desktop > Task ML > CBC_Analysis.ipynb > M↓ CBC Data Analysis & Anemia Diagnosis > M↓ Visualization & Interpretation > # 7. VISUALIZATION
Generate
Code
Markdown
Run All
Clear All Outputs
Outline
Python 3.13.5
Python

```
--- Generating Visualizations ---
C:\Users\mahmo\AppData\Local\Temp\ipykernel_13052\3281754174.py:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Importance', y='Feature', data=feature_imp_df, palette='magma')

Text(0.5, 0, 'Importance Score')
```

Model Confusion Matrix

True Label \ Predicted Label	0	1	2	3	4	5	6	7	8
0	67	0	0	0	0	0	0	0	0
1	0	12	0	0	0	0	0	0	0
2	1	0	37	0	0	0	0	0	0
3	0	0	0	56	0	0	0	0	0
4	0	0	0	0	54	0	0	0	0
5	0	0	0	0	0	4	0	0	0
6	0	0	0	0	0	0	15	0	0
7	0	0	0	0	0	0	0	9	0
8	0	0	0	0	0	0	0	0	2

GitHub Link : https://github.com/MahmOud-111ahmed/Intelligent-Anemia-Classification-System