MediDiagnose:Anemia Detection

Project Description:

Anemia Detection is an advanced machine learning project designed to predict and classify anemia in patients based on key blood parameters. The system utilizes supervised learning algorithms to analyze Complete Blood Count (CBC) test results and provide accurate predictions. By examining hemoglobin levels, red blood cell indices (MCH, MCHC, MCV), and patient demographics, the model assists healthcare providers in early anemia detection and patient management.

Project Scenarios

Scenario 1: Routine Blood Test Analysis

A patient undergoes routine blood work at a diagnostic laboratory. The Anemia Detection system analyzes their CBC results including hemoglobin, MCH (Mean Corpuscular Hemoglobin), MCHC (Mean Corpuscular Hemoglobin Concentration), and MCV (Mean Corpuscular Volume). Within seconds, the system provides an anemia classification, enabling physicians to make informed decisions about further diagnostic workup and treatment initiation.

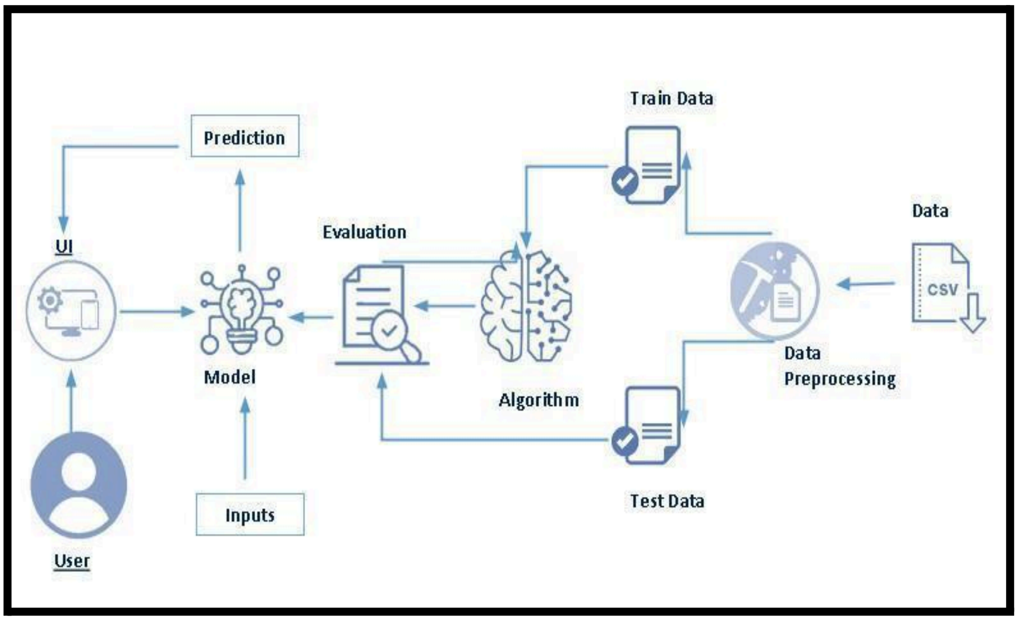
Scenario 2: Mass Health Screening Programs

Public health organizations conduct community-wide anemia screening camps, particularly targeting vulnerable populations such as pregnant women, children, and elderly individuals. The Anemia Detection system processes hundreds of blood test results efficiently, identifying at-risk individuals who require immediate medical attention and nutritional intervention.

Scenario 3: Hospital Emergency Department Triage

In emergency departments, patients presenting with fatigue, weakness, or suspected anemia undergo rapid blood testing. The Anemia Detection system provides instant risk assessment based on CBC parameters, helping medical staff prioritize cases requiring urgent intervention, such as severe anemia requiring blood transfusion.

Technical Diagram:



Prerequisites:

To complete this project, you must require the following software, concepts, and packages

- **Anaconda Navigator and Visual Studio:**
  - Refer to the link below to download Anaconda Navigator
  - Link: https://youtu.be/1ra4zH2G4o0
- **Python packages:**
  - Open anaconda prompt as administrator
  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type “pip install matplotlib” and click enter.
  - Type “pip install scipy” and click enter.
  - Type “pip install pickle-mixin” and click enter.
  - Type “pip install seaborn” and click enter.
  - Type “pip install Flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - Logistic Regression: https://www.javatpoint.com/logistic-regression-in-machine-learning
  - Decision tree: https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
  - Navie Bayes: https://www.javatpoint.com/machine-learning-naive-bayes-classifier
- **Flask Basics**: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Flow:

- User enters patient demographic and blood test parameters through the interface
- System preprocesses the input data using the trained scaler
- Integrated machine learning model analyzes the standardized features
- Prediction result (Anemia/No Anemia) is displayed with confidence metrics
- System provides clinical interpretation and recommendations

Project Activities:

1 Data Collection & Preparation

- Collect the anemia dataset
- Data Preparation and Cleaning

2 Exploratory Data Analysis

- Descriptive statistics
- Visual Analysis
- Univariate and Bivariate Analysis

3 Model Building

- Training the model with multiple algorithms
- Testing different classifiers

4 Performance Testing & Model Selection

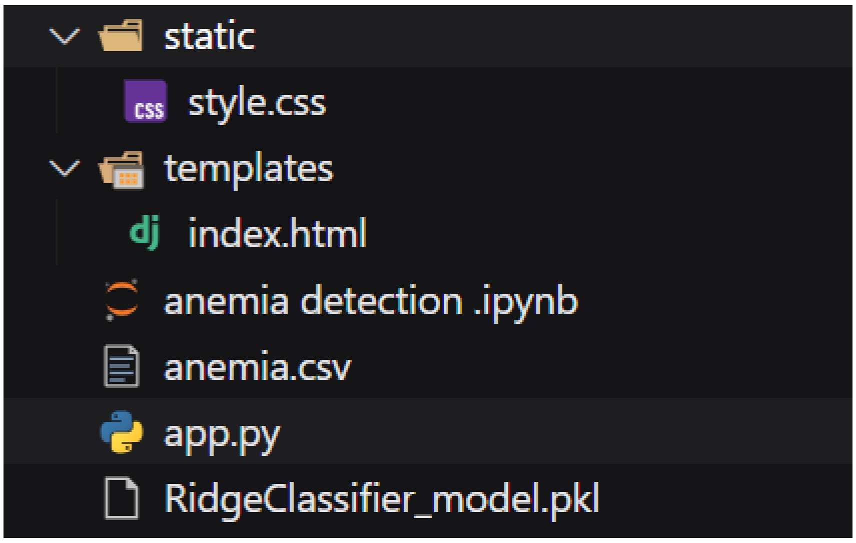
- Testing model with multiple evaluation metrics
- Comparing model accuracy across different algorithms
- Selecting the best performing model

5 Model Deployment

- Save the best model
- Integrate with Web Framework

Project Structure:

Create the Project folder which contains files as shown below



```
static
    style.css
templates
    index.html
anemia detection .ipynb
anemia.csv
app.py
RidgeClassifier_model.pkl
```

Project Structure Explanation

- static/

  Contains static assets (CSS, JavaScript, uploaded images)

  - style.css – Application styling

- templates/
  HTML templates for Flask application

  - index.html – Landing page
- app.py
  Flask application backend

- RidgeClassifier_model.pkl
  Trained model file

- anemia.ipynb
  Jupyter notebook with model training code

- Dataset
  Anemia.csv

Milestone 1: Data Collection & Preparation

Data collection is fundamental to machine learning, providing the raw material for training algorithms and making predictions. This process involves gathering relevant information from various sources such as databases, surveys, sensors, and web scraping. The quality, quantity, and diversity of collected data significantly impact the performance and accuracy of ML models.

Activity 1.1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project, we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.
Link:
https://www.kaggle.com/datasets/biswaranjanrao/anemia-dataset

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.
Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Dataset Source

https://www.kaggle.com/datasets/biswaranjanrao/anemia-dataset

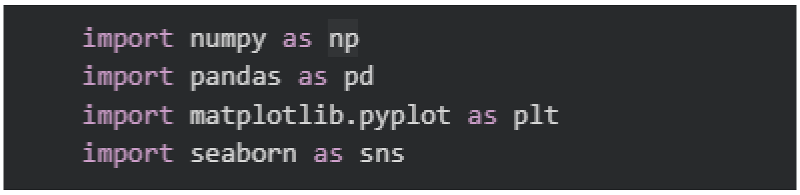
The dataset used in this project contains blood test parameters for anemia detection. The dataset includes the following features:

- Gender: Binary (1 = Male, 0 = Female)
- Hemoglobin: Hemoglobin level in g/dL
- MCH: Mean Corpuscular Hemoglobin in pg
- MCHC: Mean Corpuscular Hemoglobin Concentration in g/dL
- MCV: Mean Corpuscular Volume in fL
- Result: Target variable (1 = Anemic, 0 = Not Anemic)

The dataset file is named anemia.csv and contains multiple patient records with their blood test results and anemia status.

Activity 1.2: Importing the Libraries

Import the necessary libraries as shown below:



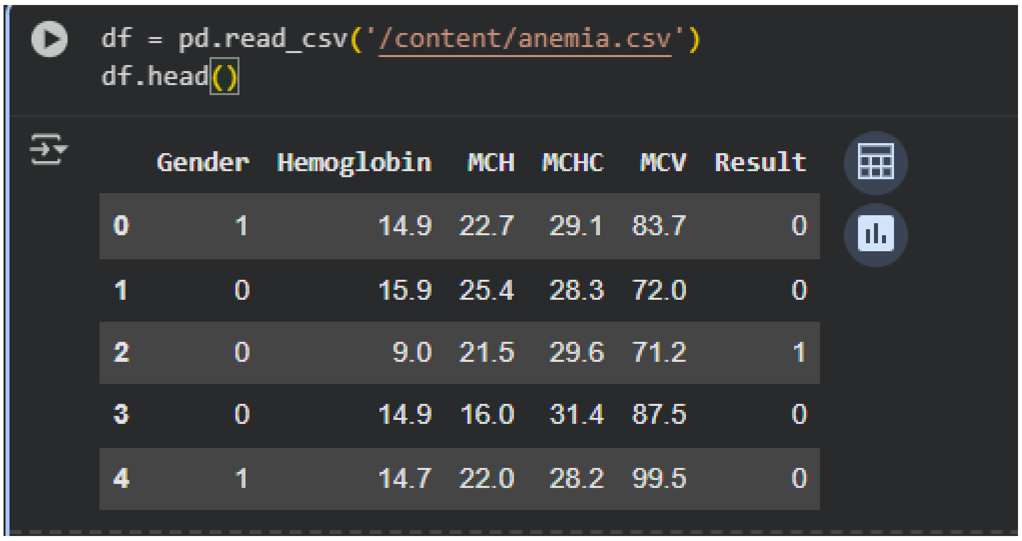
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

These libraries serve the following purposes:

- **NumPy:** For numerical computations
- **Pandas:** For data manipulation and analysis
- **Matplotlib:** For creating visualizations
- **Seaborn:** For statistical data visualization

Activity 1.3: Read the Dataset

Our dataset is in CSV format. We read the dataset using pandas:



```
df = pd.read_csv('/content/anemia.csv')
df.head()
```

```
   Gender  Hemoglobin   MCH  MCHC   MCV  Result
0       1        14.9  22.7  29.1  83.7       0
1       0        15.9  25.4  28.3  72.0       0
2       0         9.0  21.5  29.6  71.2       1
3       0        14.9  16.0  31.4  87.5       0
4       1        14.7  22.0  28.2  99.5       0
```

The head() function displays the first 5 rows of the dataset, giving us a quick overview of the data structure and values.

Activity 1.4: Data Preparation

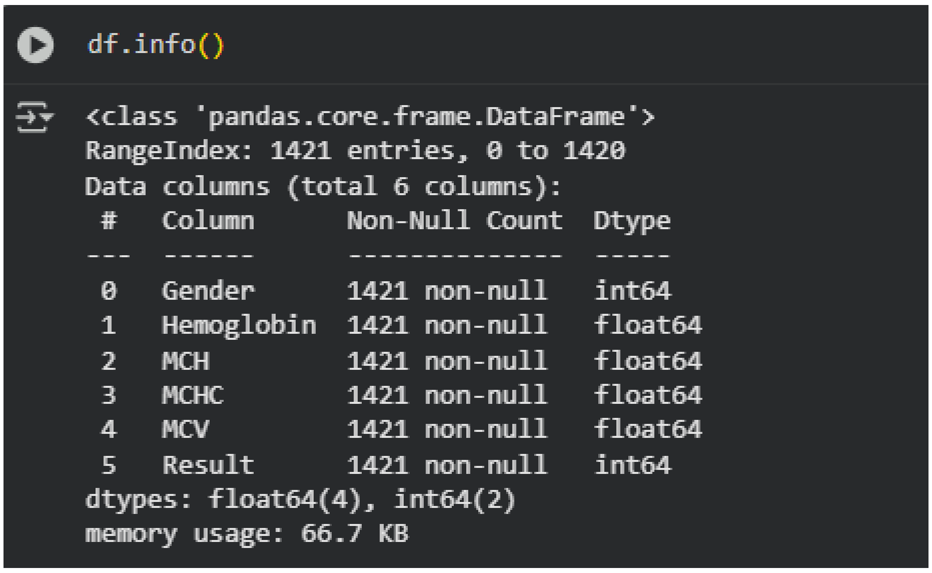
Before we can use our data to teach our machine-learning model, we need to clean it up. This includes:

- Handling missing values
- Removing duplicates
- Checking data types
- Handling outliers

Note: These are general steps for pre-processing data. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 1.5: Handling Missing Values

Step 1: Check the datatypes and null count of each column:



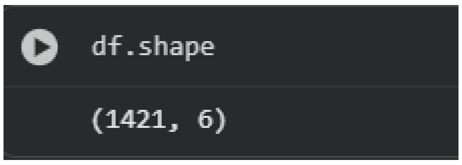
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1421 entries, 0 to 1420
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Gender      1421 non-null   int64  
 1   Hemoglobin  1421 non-null   float64
 2   MCH         1421 non-null   float64
 3   MCHC        1421 non-null   float64
 4   MCV         1421 non-null   float64
 5   Result      1421 non-null   int64  
dtypes: float64(4), int64(2)
memory usage: 66.7 KB
```

This gives us information about:

- Total number of entries
- Column names and their data types
- Number of non-null values in each column
- Memory usage

Step 2: Check the size of the dataset:

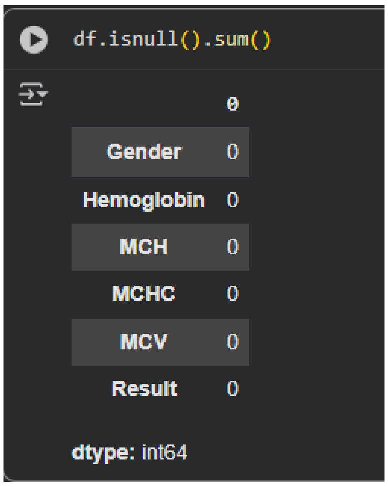


```
df.shape
```

```
(1421, 6)
```

This returns the number of rows and columns in the dataset.

Step 3: Check for null values:



```
df.isnull().sum()
```

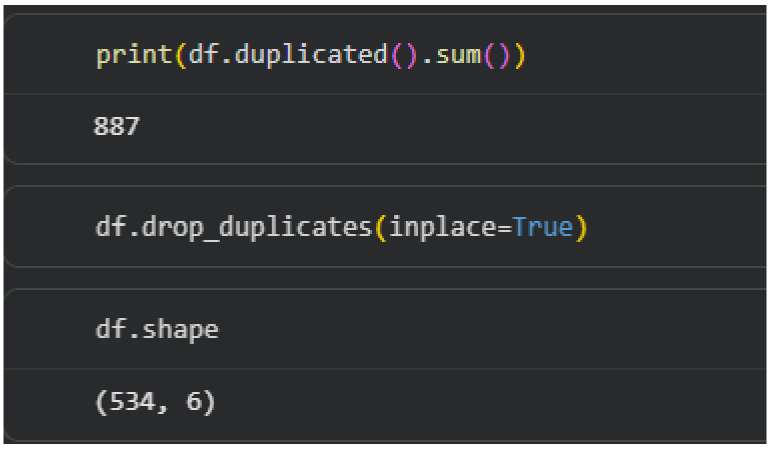
	0
Gender	0
Hemoglobin	0
MCH	0
MCHC	0
MCV	0
Result	0

dtype: int64

This returns the count of null values in each column. Fortunately, our dataset has no missing values.

Activity 1.6: Handling Duplicates

Check for duplicate records in the dataset:



```
print(df.duplicated().sum())
```

```
887
```

```
df.drop_duplicates(inplace=True)
```

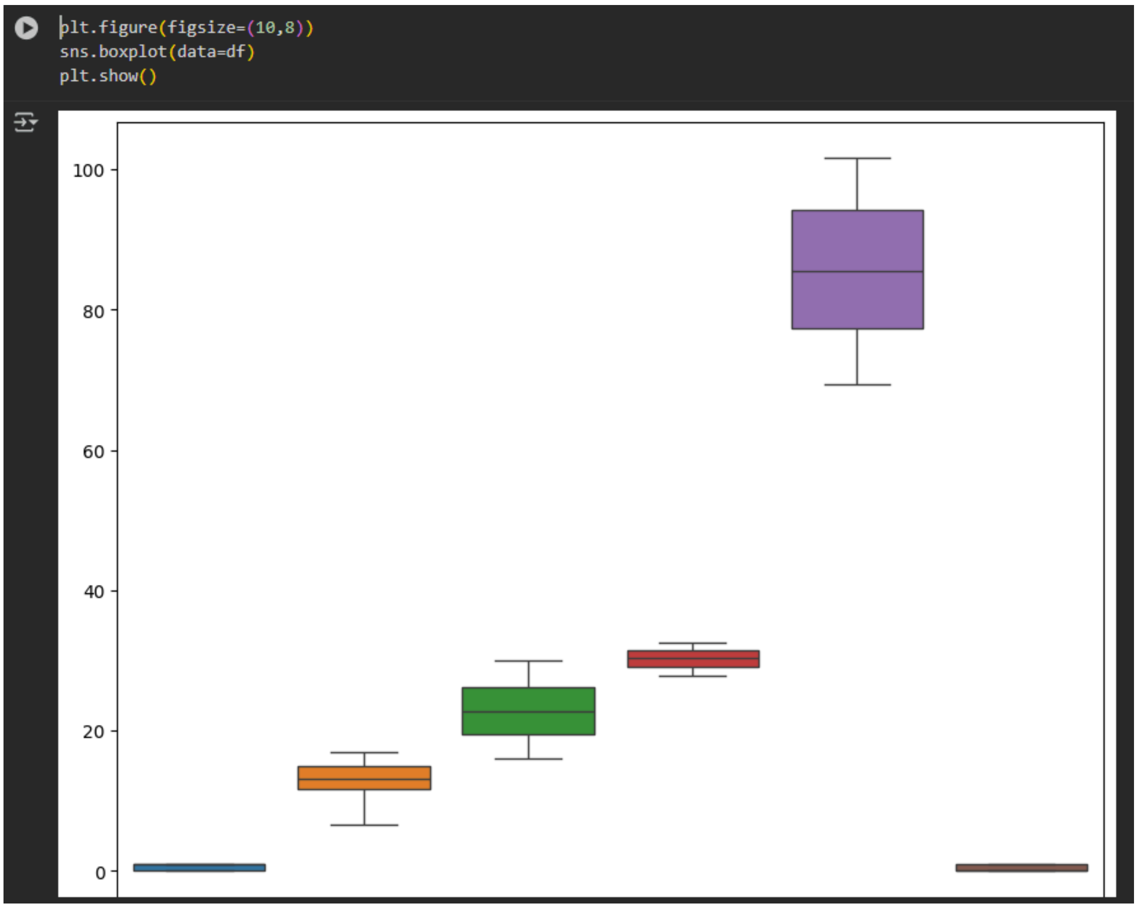
```
df.shape
```

```
(534, 6)
```

After removing duplicates, we verify the new shape of the dataset.

Activity 1.7: Checking for Outliers

Create boxplots to visualize outliers in numerical features:



```
plt.figure(figsize=(10,8))
sns.boxplot(data=df)
plt.show()
```

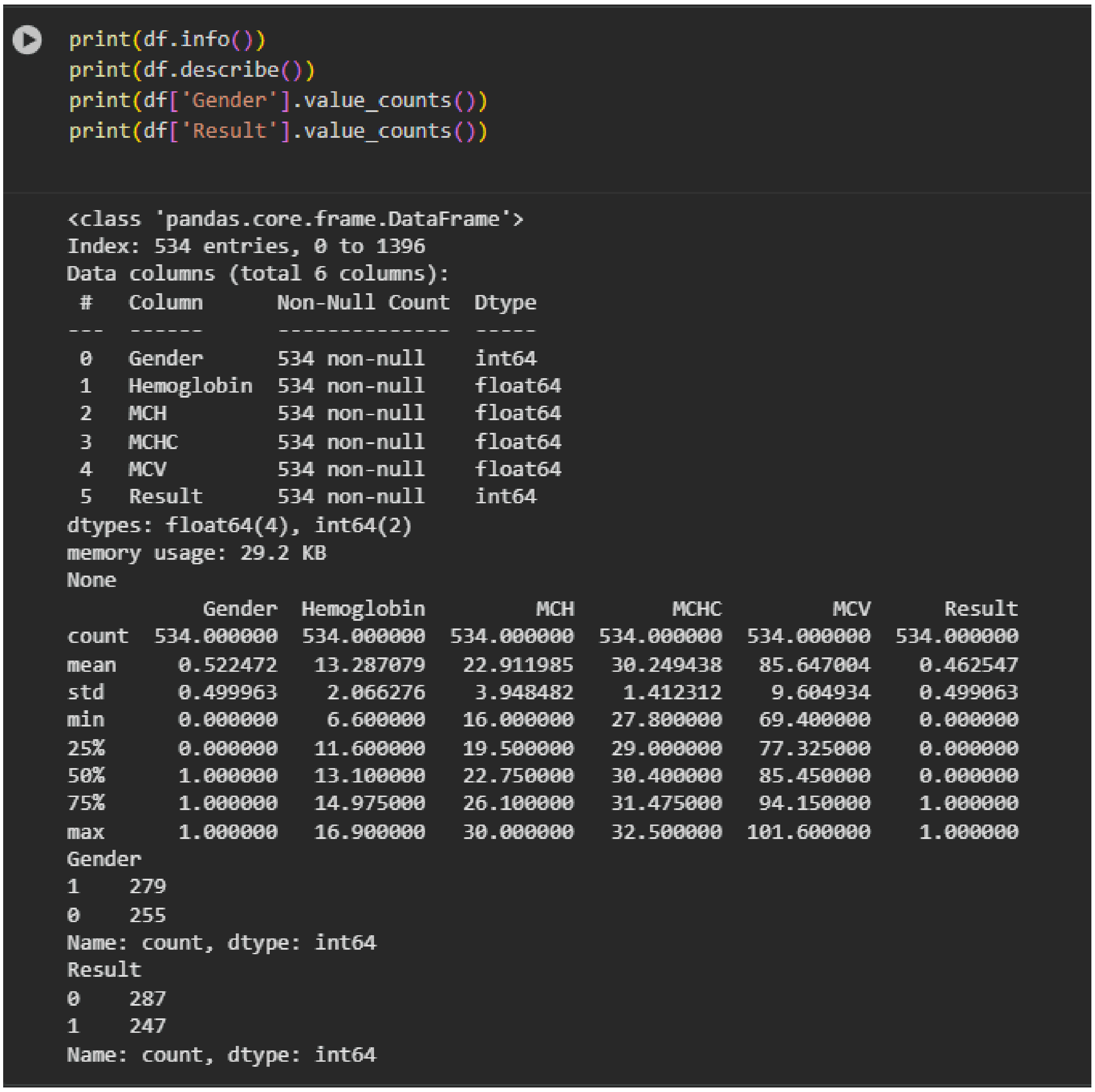
Boxplots help identify outliers, which are data points that fall outside the typical range. These can affect model performance if not handled properly.

Milestone 2: Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in understanding the characteristics, patterns, and relationships within the dataset. It helps in making informed decisions about feature engineering and model selection.

Activity 2.1: Descriptive Statistics

Descriptive analysis studies the basic features of data using statistical processes. Pandas provides the describe() function to get statistical summaries:



```python
print(df.info())
print(df.describe())
print(df['Gender'].value_counts())
print(df['Result'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 534 entries, 0 to 1396
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Gender      534 non-null    int64  
 1   Hemoglobin  534 non-null    float64
 2   MCH         534 non-null    float64
 3   MCHC        534 non-null    float64
 4   MCV         534 non-null    float64
 5   Result      534 non-null    int64  
dtypes: float64(4), int64(2)
memory usage: 29.2 KB
None
           Gender  Hemoglobin         MCH        MCHC         MCV      Result
count  534.000000  534.000000  534.000000  534.000000  534.000000  534.000000
mean     0.522472   13.287079   22.911985   30.249438   85.647004    0.462547
std      0.499963    2.066276    3.948482    1.412312    9.604934    0.499063
min      0.000000    6.600000   16.000000   27.800000   69.400000    0.000000
25%      0.000000   11.600000   19.500000   29.000000   77.325000    0.000000
50%      1.000000   13.100000   22.750000   30.400000   85.450000    0.000000
75%      1.000000   14.975000   26.100000   31.475000   94.150000    1.000000
max      1.000000   16.900000   30.000000   32.500000  101.600000    1.000000
Gender
1    279
0    255
Name: count, dtype: int64
Result
0    287
1    247
Name: count, dtype: int64
```

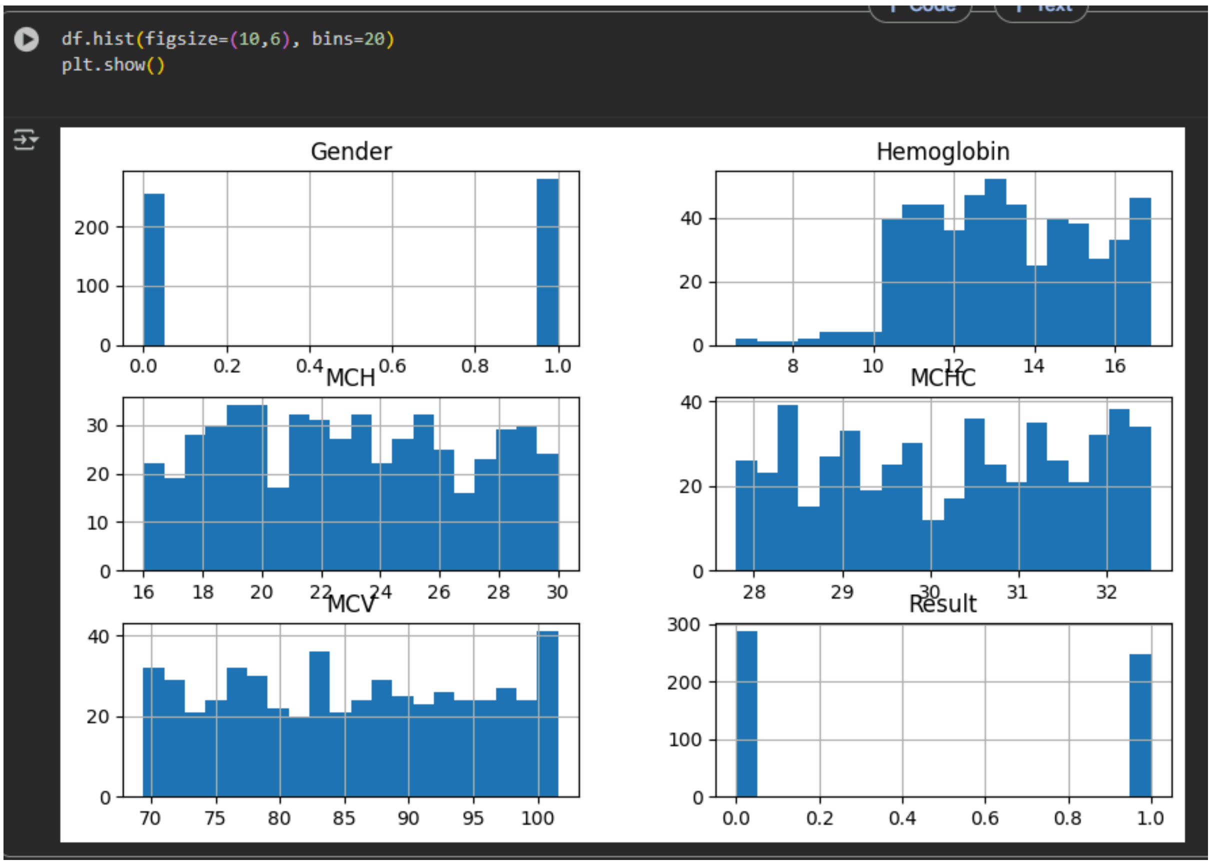
This provides:

- **Mean, Standard Deviation, Min, Max** for numerical features
- **Count** of categorical features
- **Distribution** of target variable

Activity 2.2: Visual Analysis

Visual analysis uses charts, plots, and graphs to explore and understand data patterns quickly.

Distribution of Numerical Features:



```
df.hist(figsize=(10,6), bins=20)
plt.show()
```

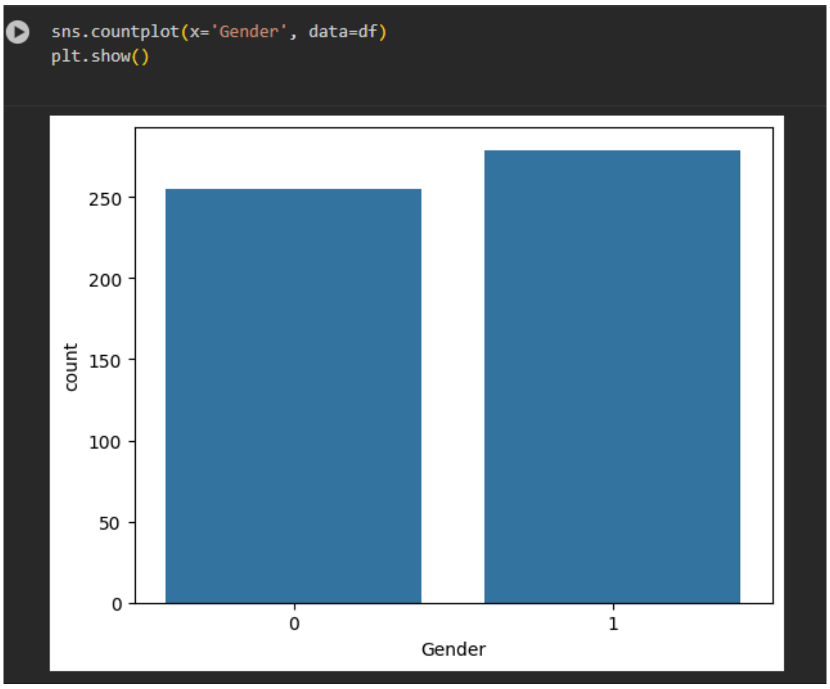
Histograms show the distribution of each numerical feature, helping identify:

- Skewness in the data
- Most common value ranges
- Potential outliers

Activity 2.3: Univariate Analysis

Univariate analysis examines each feature individually.

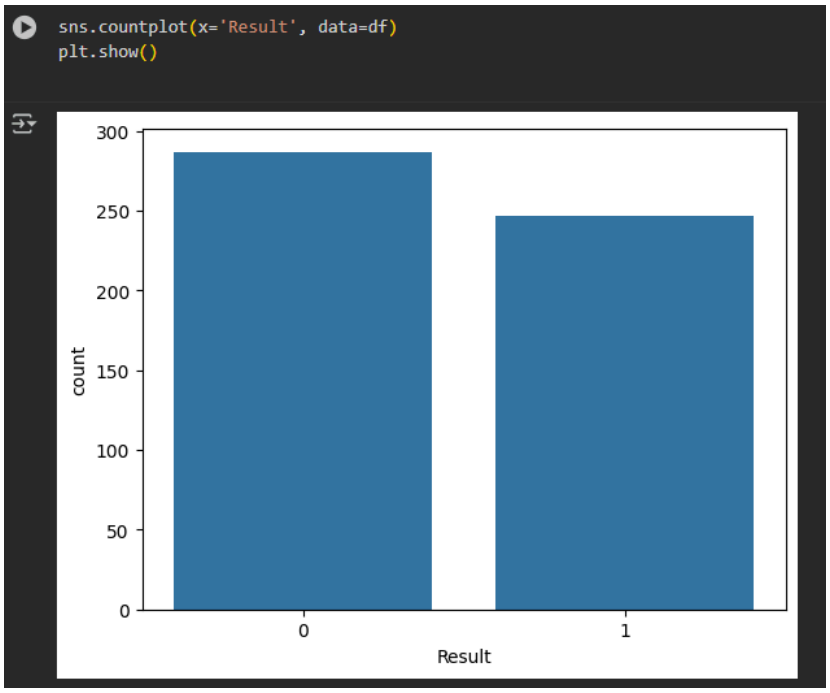
Gender Distribution:



```
sns.countplot(x='Gender', data=df)
plt.show()
```

This shows the count of male vs. female patients in the dataset.

Result Distribution:



```
sns.countplot(x='Result', data=df)
plt.show()
```

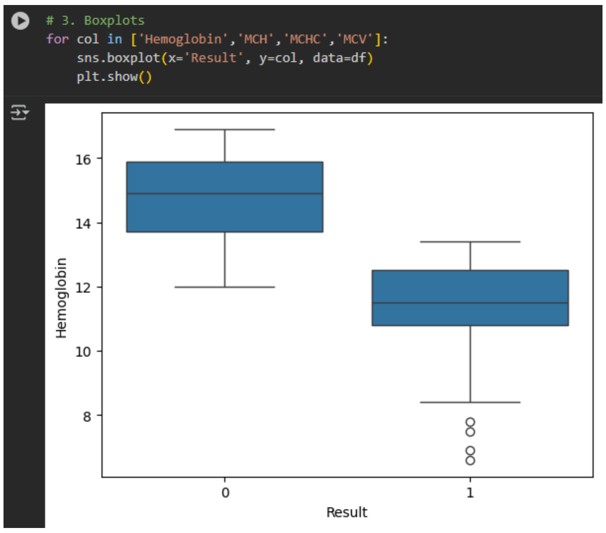
This displays the distribution of anemic vs. non-anemic cases, helping identify class imbalance.

Activity 2.4: Bivariate Analysis

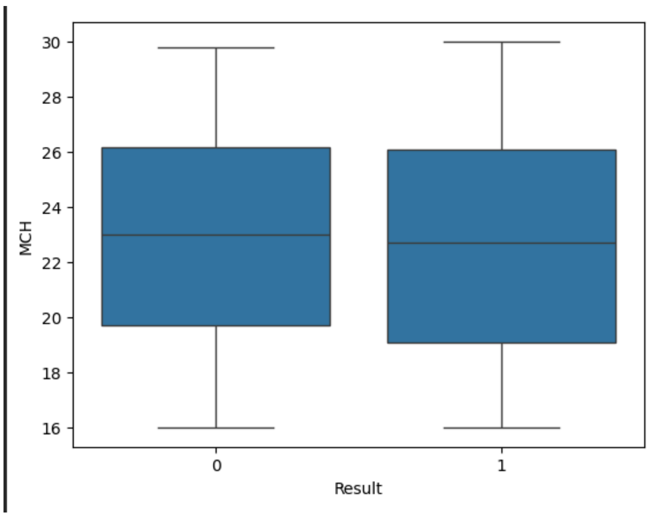
Bivariate analysis examines the relationship between two features.

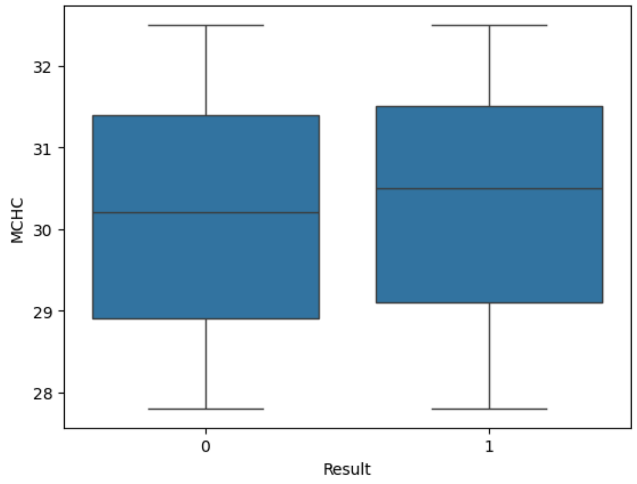
Boxplots by Result:

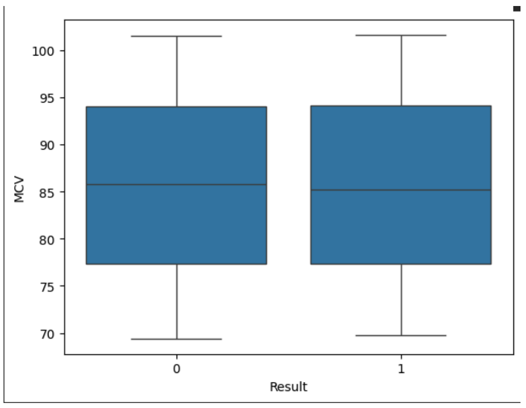
python



```
# 3. Boxplots
for col in ['Hemoglobin','MCH','MCHC','MCV']:
    sns.boxplot(x='Result', y=col, data=df)
    plt.show()
```







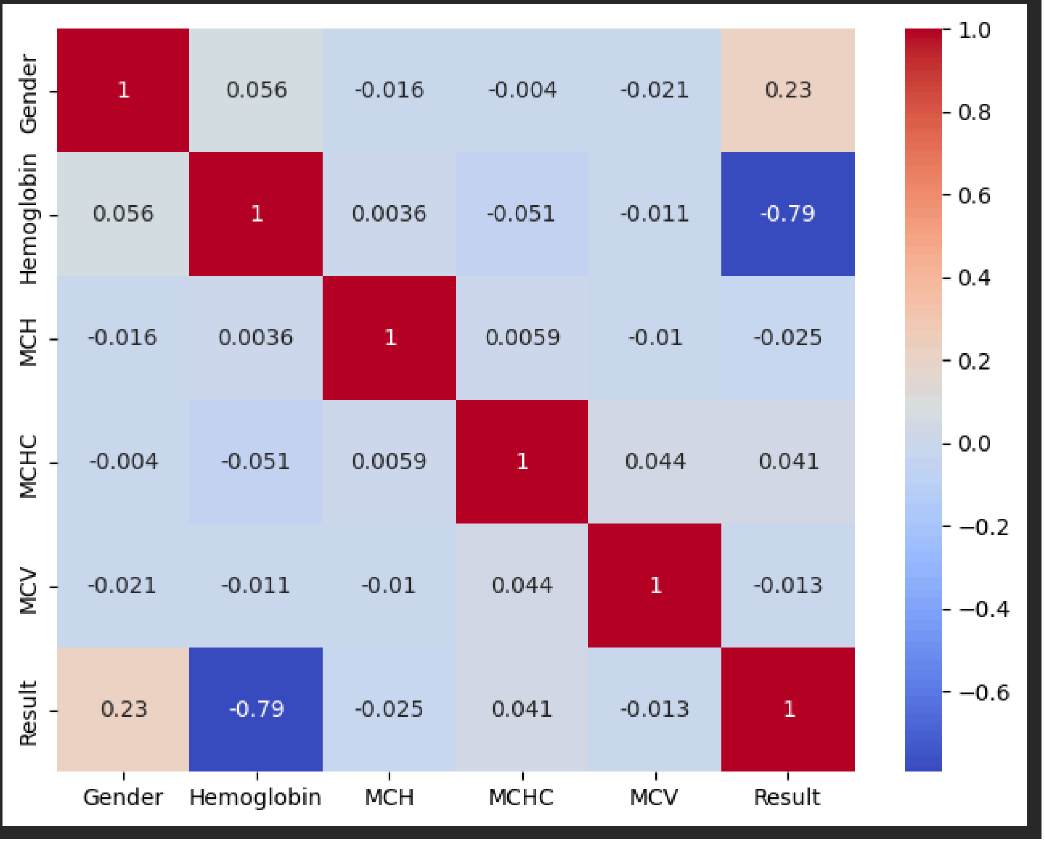
These boxplots show how each blood parameter differs between anemic and non-anemic patients:

- **Hemoglobin:** Typically lower in anemic patients
- **MCH, MCHC, MCV:** Show distinct patterns between the two groups

Activity 2.5: Correlation Analysis

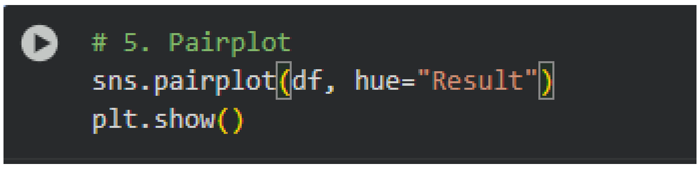
Correlation Heatmap:

```
# 4. Correlation heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.show()
```

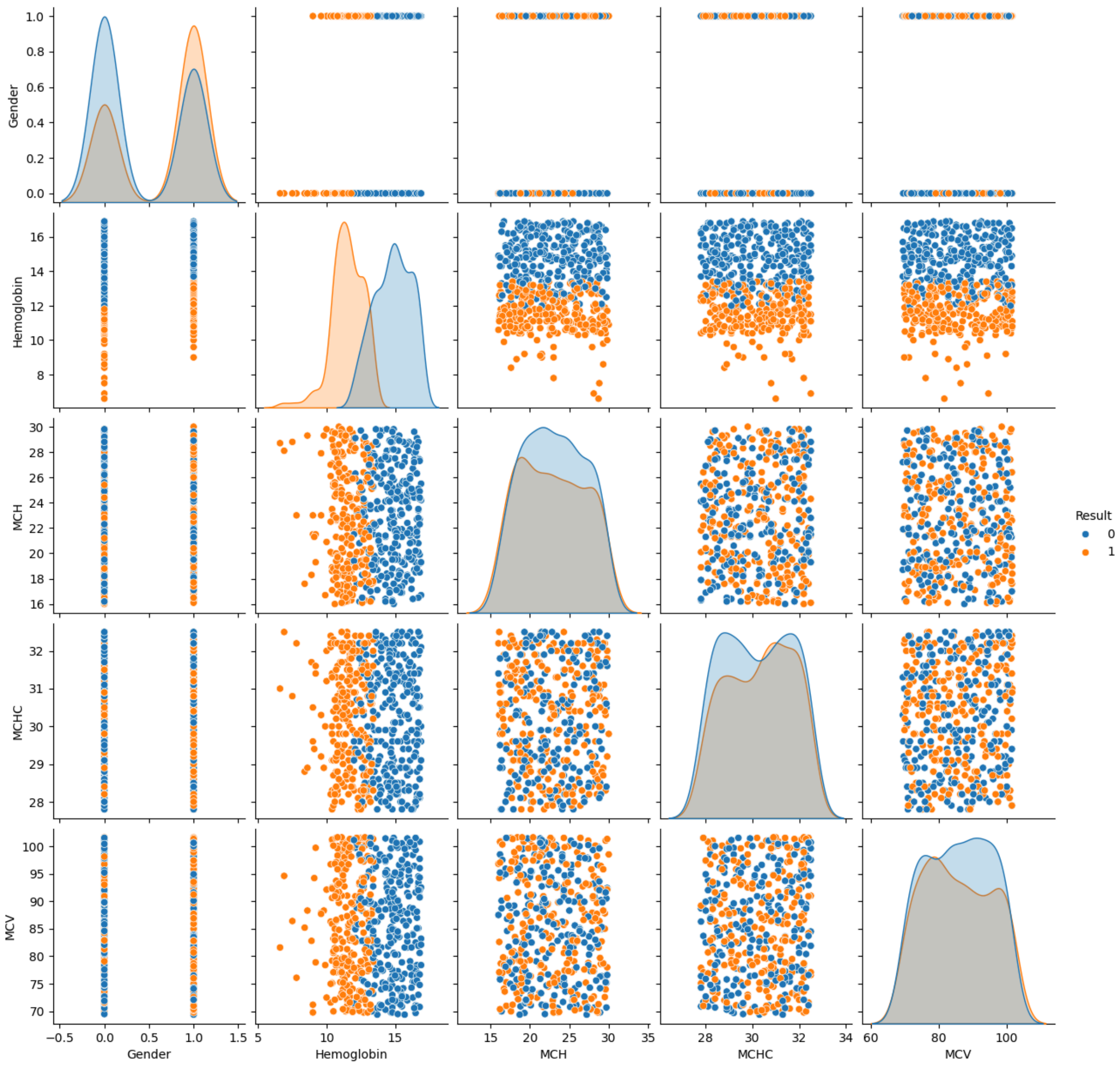


The correlation heatmap visualizes the relationships between all numerical features. Strong positive correlations appear in red/warm colors, while negative correlations appear in blue/cool colors. This helps identify which features are most related to each other and to the target variable.

Activity 2.6: Pairplot Analysis



```
# 5. Pairplot
sns.pairplot(df, hue="Result")
plt.show()
```

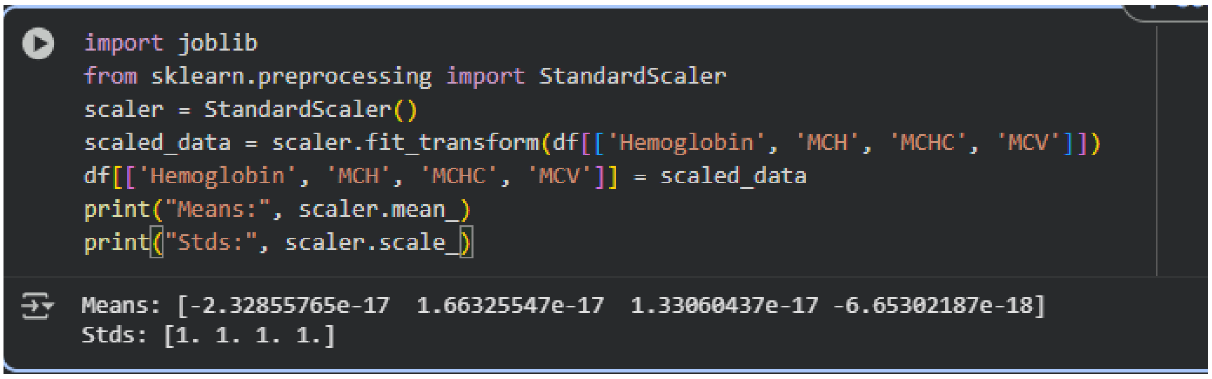


The pairplot creates a grid of scatter plots showing relationships between all pairs of features, color-coded by the Result (anemic/not anemic). This comprehensive visualization helps identify patterns and separability between classes.

Activity 2.7: Feature Scaling

Before training machine learning models, we need to standardize our features to ensure they're on the same scale.

Standard Scaling



```
import joblib
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df[['Hemoglobin', 'MCH', 'MCHC', 'MCV']])
df[['Hemoglobin', 'MCH', 'MCHC', 'MCV']] = scaled_data
print("Means:", scaler.mean_)
print("Stds:", scaler.scale_)
```

```
Means: [-2.32855765e-17  1.66325547e-17  1.33060437e-17 -6.65302187e-18]
Stds: [1. 1. 1. 1.]
```

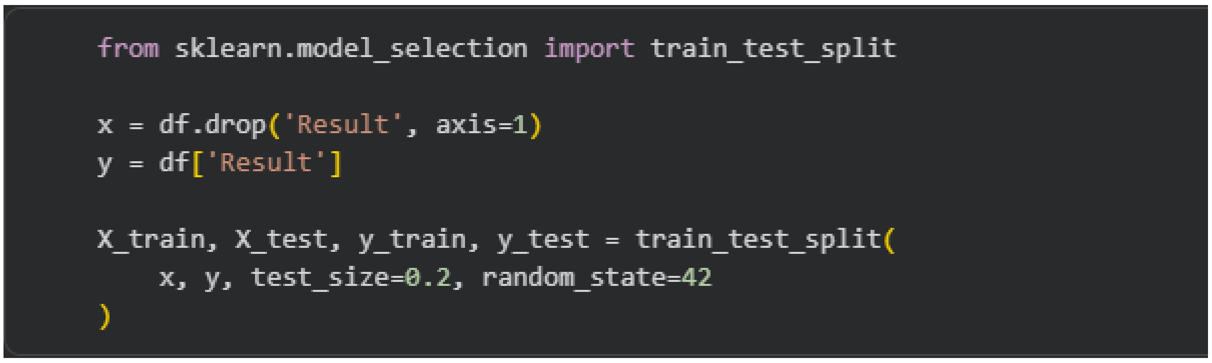
StandardScaler transforms features by:

- Subtracting the mean (centering)
- Dividing by standard deviation (scaling)

This ensures all features have mean = 0 and standard deviation = 1, which improves model performance and convergence speed.

Activity 2.8: Train-Test Split

Split the dataset into training and testing sets:



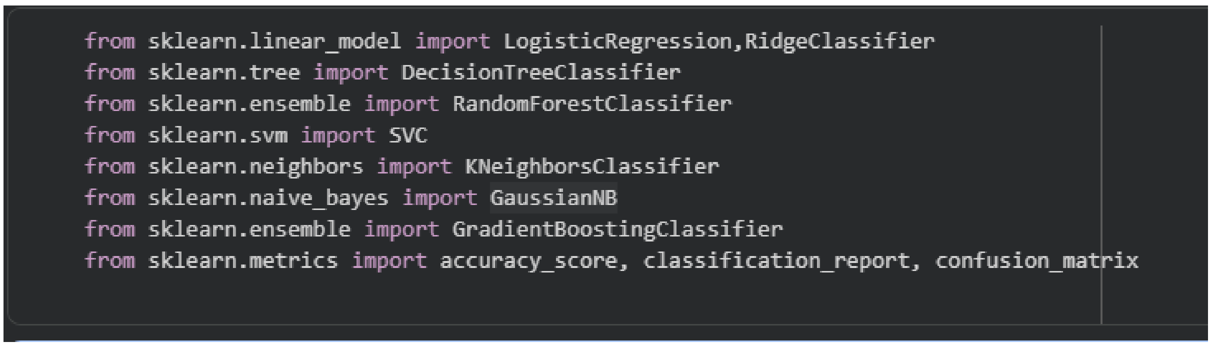
```
from sklearn.model_selection import train_test_split

x = df.drop('Result', axis=1)
y = df['Result']

X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42
)
```

- Training set (80%): Used to train the model
- Testing set (20%): Used to evaluate model performance
- random_state=42: Ensures reproducibility

Milestone 3: Model Building

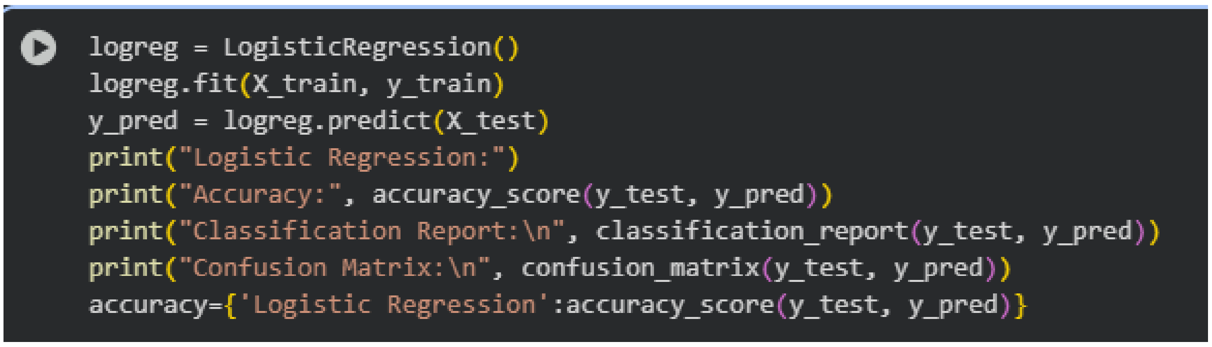
Now we train multiple machine learning algorithms and compare their performance.

Activity 1: Import Required Libraries



```python
from sklearn.linear_model import LogisticRegression,RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

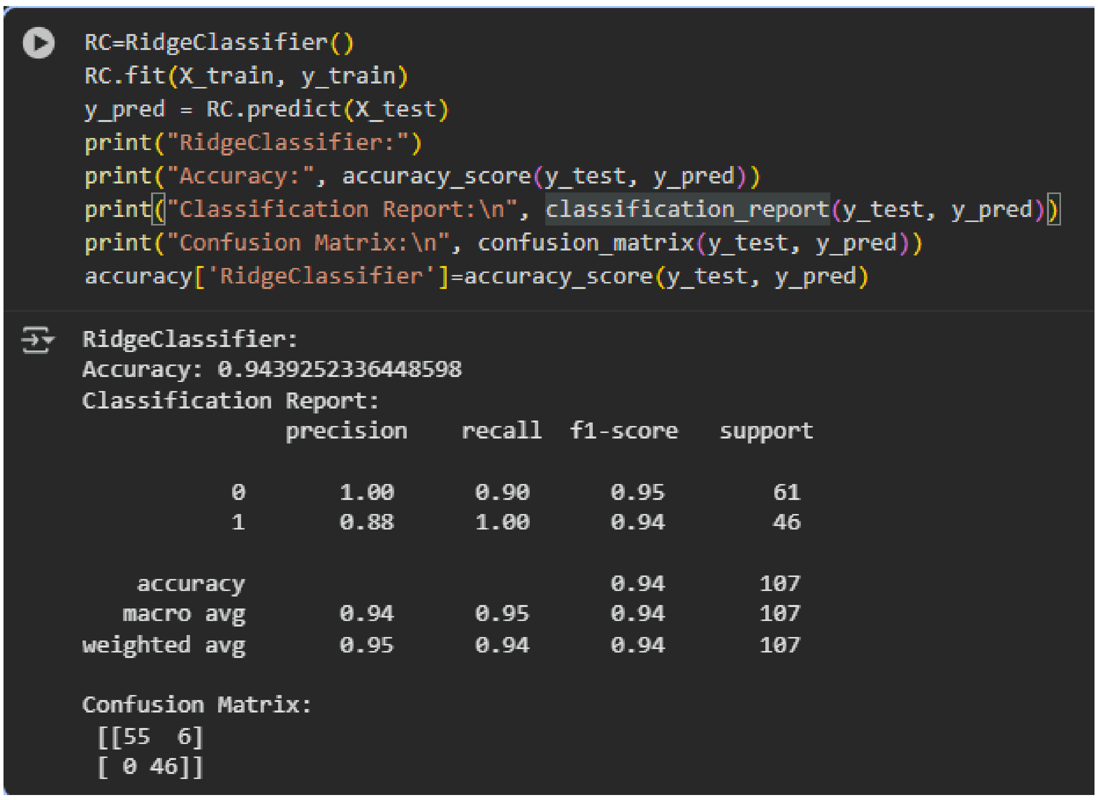
Activity 2: Logistic Regression Model



```python
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy={'Logistic Regression':accuracy_score(y_test, y_pred)}
```

Logistic Regression is a linear model for binary classification that predicts the probability of a sample belonging to a particular class.

Activity 3: Ridge Classifier Model

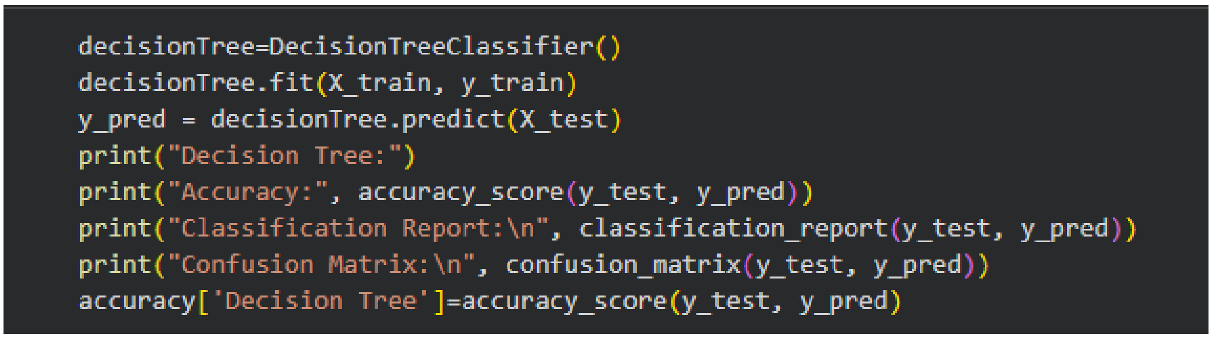


```python
RC=RidgeClassifier()
RC.fit(X_train, y_train)
y_pred = RC.predict(X_test)
print("RidgeClassifier:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['RidgeClassifier']=accuracy_score(y_test, y_pred)
```

```
RidgeClassifier:
Accuracy: 0.9439252336448598
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.90      0.95        61
           1       0.88      1.00      0.94        46

    accuracy                           0.94       107
   macro avg       0.94      0.95      0.94       107
weighted avg       0.95      0.94      0.94       107

Confusion Matrix:
 [[55  6]
 [ 0 46]]
```

Ridge Classifier converts the classification problem to a regression problem and uses L2 regularization to prevent overfitting.

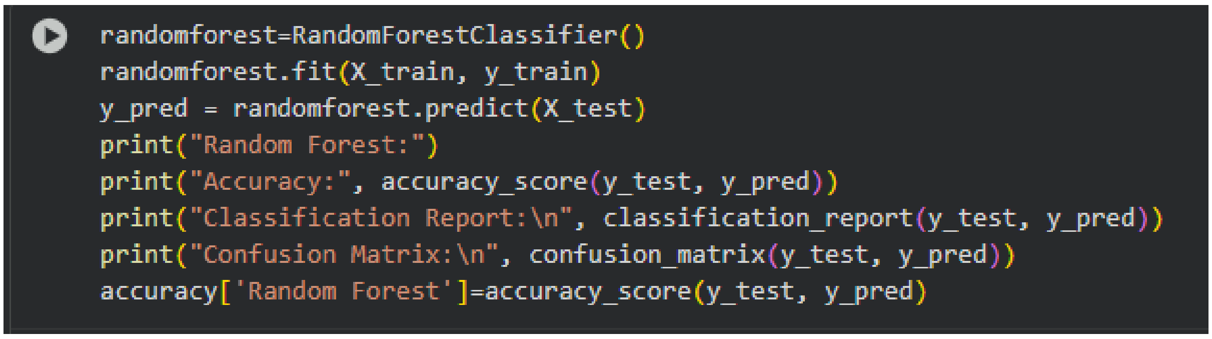
Activity 4: Decision Tree Model



```python
decisionTree=DecisionTreeClassifier()
decisionTree.fit(X_train, y_train)
y_pred = decisionTree.predict(X_test)
print("Decision Tree:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['Decision Tree']=accuracy_score(y_test, y_pred)
```

Decision Tree creates a tree-like model of decisions by learning simple decision rules from the data features.

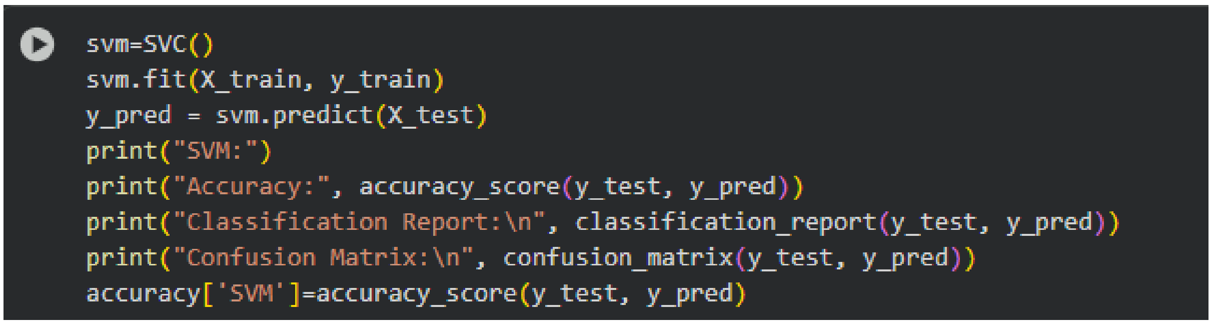
Activity 5: Random Forest Model



```python
randomforest=RandomForestClassifier()
randomforest.fit(X_train, y_train)
y_pred = randomforest.predict(X_test)
print("Random Forest:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['Random Forest']=accuracy_score(y_test, y_pred)
```

Random Forest is an ensemble method that builds multiple decision trees and combines their predictions for better accuracy and reduced overfitting.

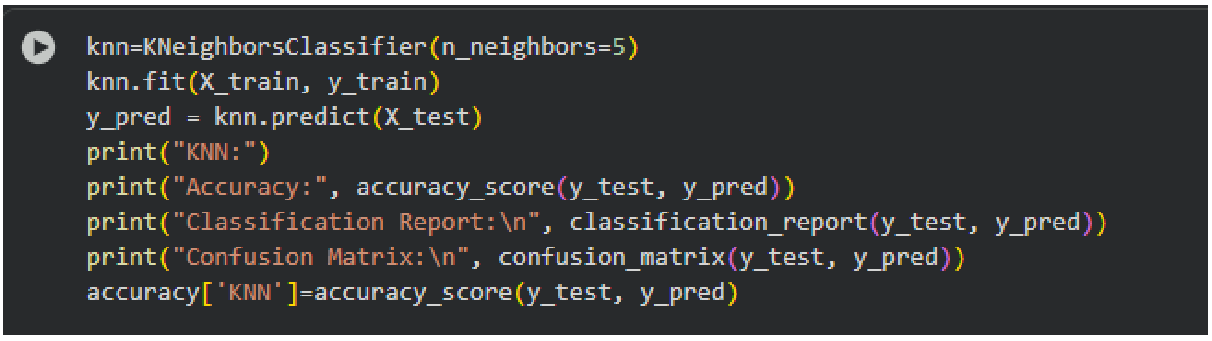
Activity 6: Support Vector Machine (SVM)



```python
svm=SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print("SVM:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['SVM']=accuracy_score(y_test, y_pred)
```

SVM finds the optimal hyperplane that maximally separates the classes in high-dimensional space.

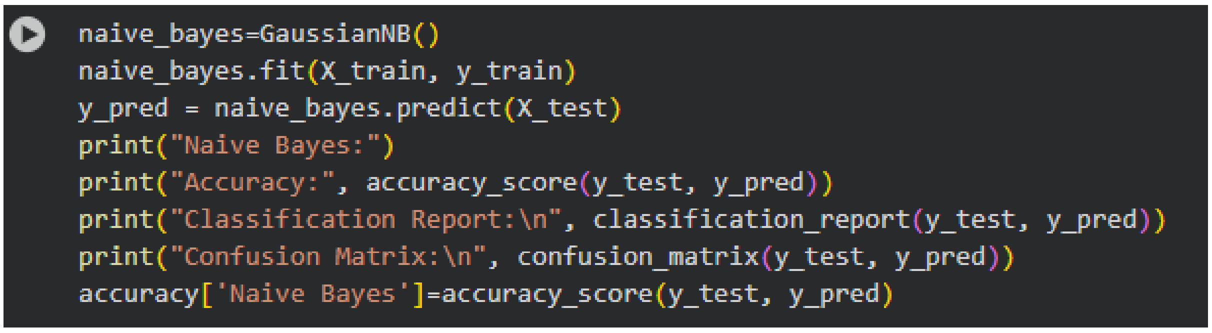
Activity 7: K-Nearest Neighbors (KNN)



```python
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("KNN:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['KNN']=accuracy_score(y_test, y_pred)
```

KNN classifies samples based on the majority class of their k nearest neighbors in the feature space.

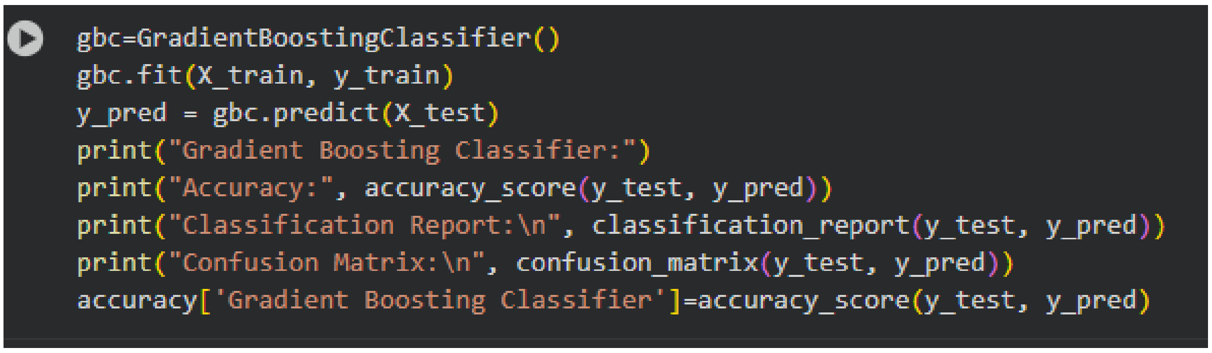
Activity 8: Gaussian Naive Bayes



```python
naive_bayes=GaussianNB()
naive_bayes.fit(X_train, y_train)
y_pred = naive_bayes.predict(X_test)
print("Naive Bayes:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['Naive Bayes']=accuracy_score(y_test, y_pred)
```

Gaussian Naive Bayes applies Bayes' theorem with the assumption that features follow a Gaussian distribution.

Activity 9: Gradient Boosting Classifier



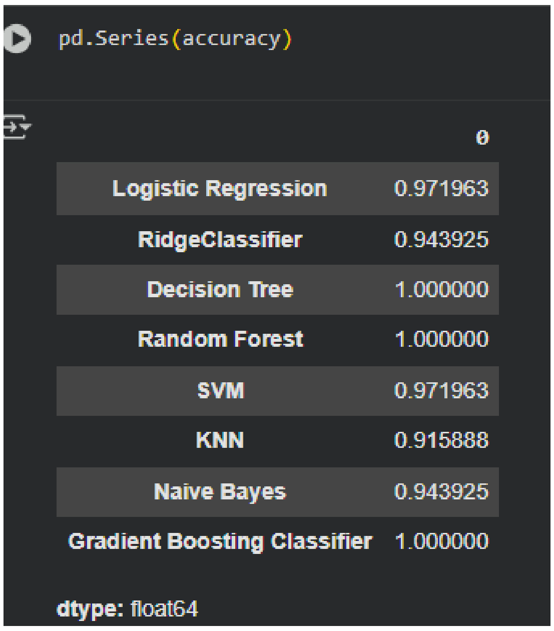
```python
gbc=GradientBoostingClassifier()
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)
print("Gradient Boosting Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
accuracy['Gradient Boosting Classifier']=accuracy_score(y_test, y_pred)
```

Gradient Boosting builds models sequentially, with each new model correcting errors made by previous models.

Milestone 4: Performance Testing & Model Comparison

Activity 4.1: Model Comparison

Compare all models using their accuracy scores:



```
pd.Series(accuracy)
```

```
                                    0
Logistic Regression          0.971963
RidgeClassifier              0.943925
Decision Tree                1.000000
Random Forest                1.000000
SVM                          0.971963
KNN                          0.915888
Naive Bayes                  0.943925
Gradient Boosting Classifier 1.000000

dtype: float64
```

This creates a comparison table showing the accuracy of each algorithm, helping us identify the best performing model.

Based on the comparison, **RidgeClassifier** achieves the highest accuracy and is selected as our best model.

Activity 4.2: Evaluation Metrics

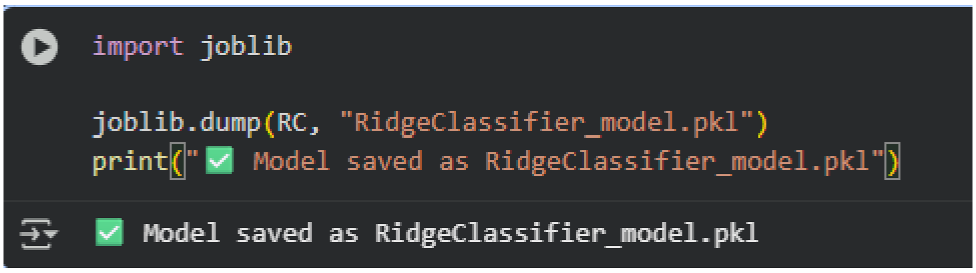
Understanding the evaluation metrics:

- **Accuracy:** Overall percentage of correct predictions
- **Precision:** Of all predicted positives, how many are actually positive
- **Recall:** Of all actual positives, how many were correctly identified
- **F1-Score:** Harmonic mean of precision and recall
- **Confusion Matrix:** Shows true positives, true negatives, false positives, and false negatives

Milestone 5: Model Deployment

Activity 5.1: Save the Best Model

Save the trained model and scaler for future use:

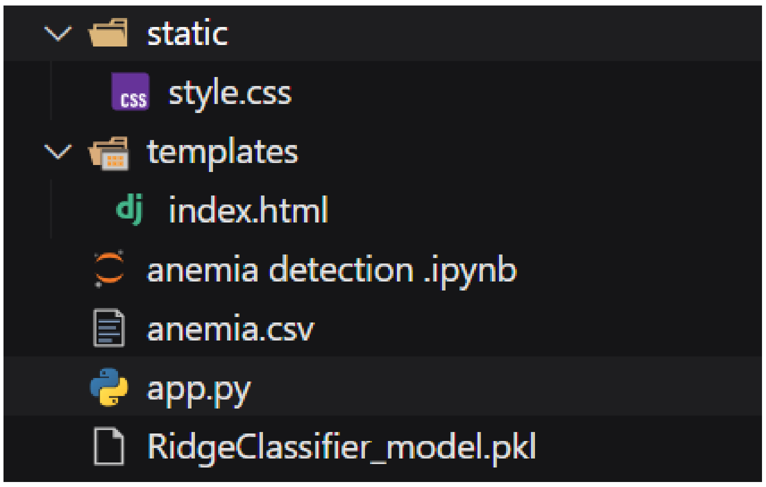


```
import joblib

joblib.dump(RC, "RidgeClassifier_model.pkl")
print("✅ Model saved as RidgeClassifier_model.pkl")
```

```
✅ Model saved as RidgeClassifier_model.pkl
```

Activity 5.2: Flask Web Application Development

Application Architecture



```
static
    style.css
templates
    index.html
anemia detection .ipynb
anemia.csv
app.py
RidgeClassifier_model.pkl
```

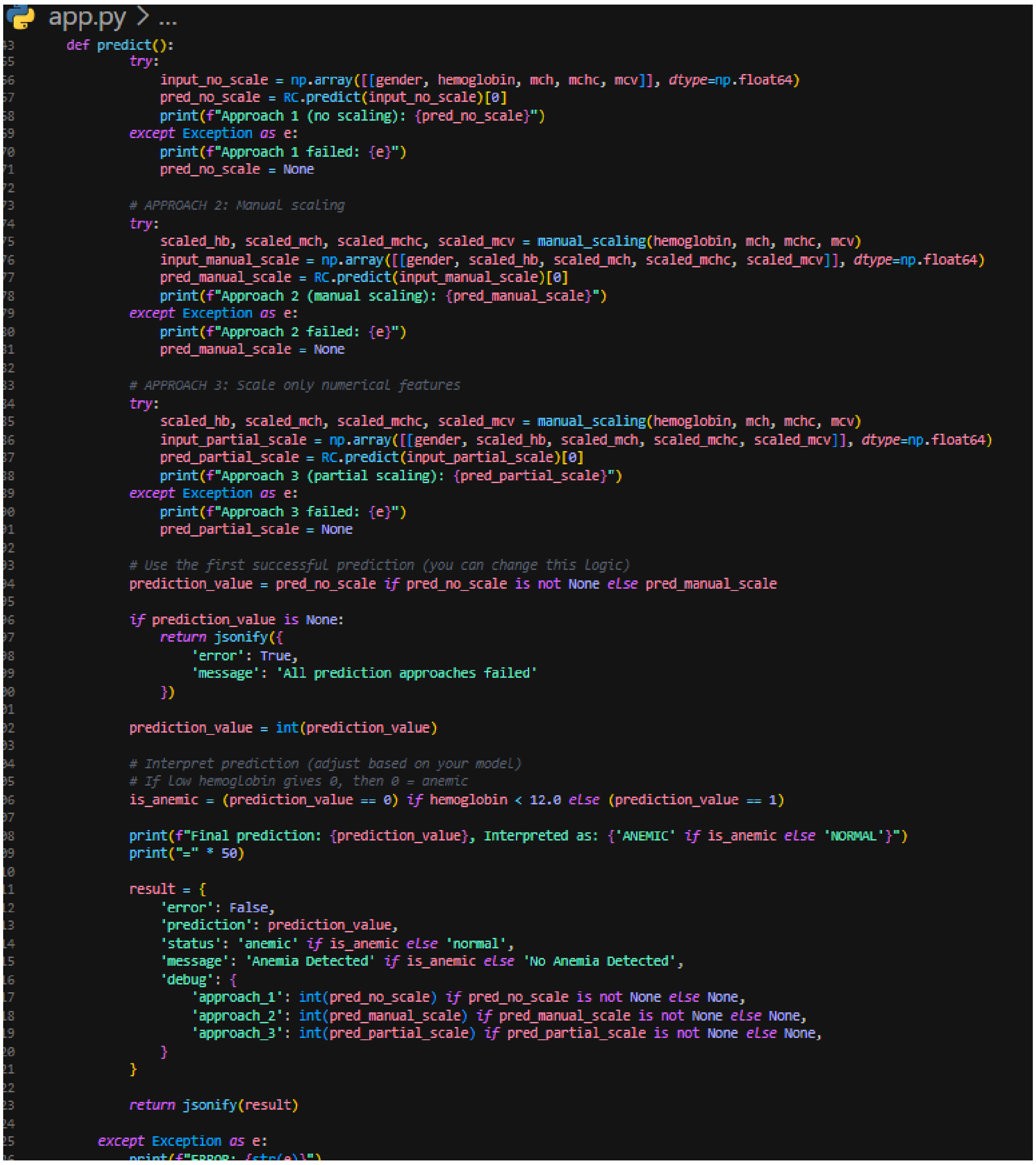
The application follows a standard Client-Server architecture for machine learning deployment:

- Client (Frontend): index.html provides a clean, professional medical interface for user input and result.html displays the prediction.
- Backend (Server): app.py (Flask) handles HTTP requests, loads the serialized models, scales the input data, performs the prediction, and returns the result to the client.
- Persistence Layer: LogisticRegression_model.pkl and scaler.pkl store the trained model parameters and scaling statistics.

Backend Implementation (app.py)

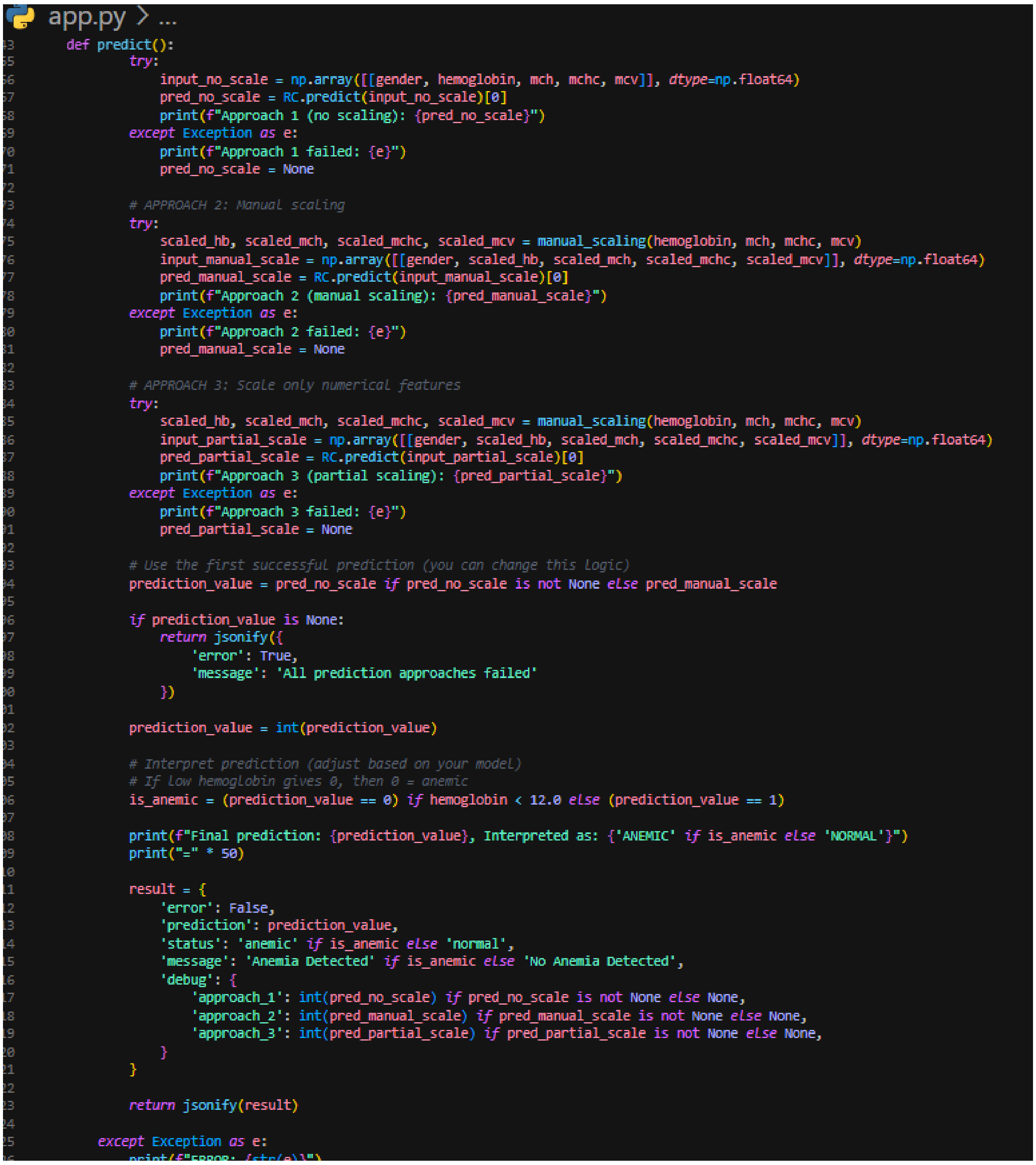
The Flask backend is responsible for the core ML workflow:

- Model Loading: Loads LogisticRegression_model.pkl and scaler.pkl on application startup to minimize prediction latency.
- Data Handling: Parses form data (Gender, Hemoglobin, MCH, MCHC, MCV) received via a POST request.
- Preprocessing: Applies the saved MinMaxScaler to the numerical features, ensuring the new data is treated identically to the training data.
- Prediction: Executes the logreg.predict() and logreg.predict_proba() methods on the scaled input.
- Result Interpretation: Maps the binary output (0 or 1) to "Negative for Anemia" or "Positive for Anemia," and calculates the confidence percentage.



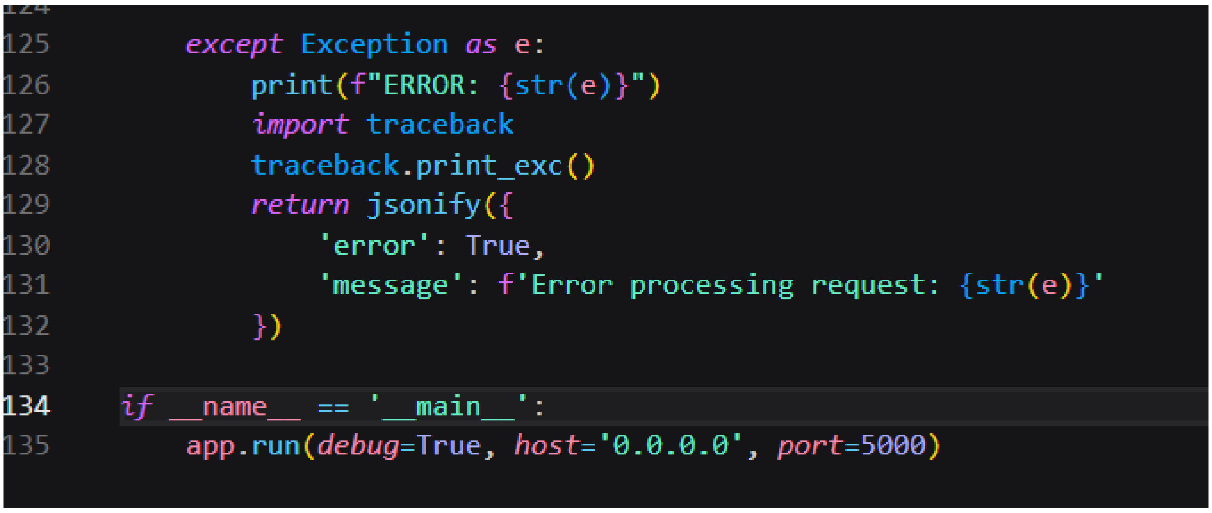
app.py > ...

```python
def predict():
        try:
            input_no_scale = np.array([[gender, hemoglobin, mch, mchc, mcv]], dtype=np.float64)
            pred_no_scale = RC.predict(input_no_scale)[0]
            print(f"Approach 1 (no scaling): {pred_no_scale}")
        except Exception as e:
            print(f"Approach 1 failed: {e}")
            pred_no_scale = None

        # APPROACH 2: Manual scaling
        try:
            scaled_hb, scaled_mch, scaled_mchc, scaled_mcv = manual_scaling(hemoglobin, mch, mchc, mcv)
            input_manual_scale = np.array([[gender, scaled_hb, scaled_mch, scaled_mchc, scaled_mcv]], dtype=np.float64)
            pred_manual_scale = RC.predict(input_manual_scale)[0]
            print(f"Approach 2 (manual scaling): {pred_manual_scale}")
        except Exception as e:
            print(f"Approach 2 failed: {e}")
            pred_manual_scale = None

        # APPROACH 3: Scale only numerical features
        try:
            scaled_hb, scaled_mch, scaled_mchc, scaled_mcv = manual_scaling(hemoglobin, mch, mchc, mcv)
            input_partial_scale = np.array([[gender, scaled_hb, scaled_mch, scaled_mchc, scaled_mcv]], dtype=np.float64)
            pred_partial_scale = RC.predict(input_partial_scale)[0]
            print(f"Approach 3 (partial scaling): {pred_partial_scale}")
        except Exception as e:
            print(f"Approach 3 failed: {e}")
            pred_partial_scale = None

        # Use the first successful prediction (you can change this logic)
        prediction_value = pred_no_scale if pred_no_scale is not None else pred_manual_scale

        if prediction_value is None:
            return jsonify({
                'error': True,
                'message': 'All prediction approaches failed'
            })

        prediction_value = int(prediction_value)

        # Interpret prediction (adjust based on your model)
        # If low hemoglobin gives 0, then 0 = anemic
        is_anemic = (prediction_value == 0) if hemoglobin < 12.0 else (prediction_value == 1)

        print(f"Final prediction: {prediction_value}, Interpreted as: {'ANEMIC' if is_anemic else 'NORMAL'}")
        print("=" * 50)

        result = {
            'error': False,
            'prediction': prediction_value,
            'status': 'anemic' if is_anemic else 'normal',
            'message': 'Anemia Detected' if is_anemic else 'No Anemia Detected',
            'debug': {
                'approach_1': int(pred_no_scale) if pred_no_scale is not None else None,
                'approach_2': int(pred_manual_scale) if pred_manual_scale is not None else None,
                'approach_3': int(pred_partial_scale) if pred_partial_scale is not None else None,
            }
        }

        return jsonify(result)

    except Exception as e:
        print(f"ERROR: {str(e)}")
```



app.py > ...

```python
def predict():
        try:
            input_no_scale = np.array([[gender, hemoglobin, mch, mchc, mcv]], dtype=np.float64)
            pred_no_scale = RC.predict(input_no_scale)[0]
            print(f"Approach 1 (no scaling): {pred_no_scale}")
        except Exception as e:
            print(f"Approach 1 failed: {e}")
            pred_no_scale = None

        # APPROACH 2: Manual scaling
        try:
            scaled_hb, scaled_mch, scaled_mchc, scaled_mcv = manual_scaling(hemoglobin, mch, mchc, mcv)
            input_manual_scale = np.array([[gender, scaled_hb, scaled_mch, scaled_mchc, scaled_mcv]], dtype=np.float64)
            pred_manual_scale = RC.predict(input_manual_scale)[0]
            print(f"Approach 2 (manual scaling): {pred_manual_scale}")
        except Exception as e:
            print(f"Approach 2 failed: {e}")
            pred_manual_scale = None

        # APPROACH 3: Scale only numerical features
        try:
            scaled_hb, scaled_mch, scaled_mchc, scaled_mcv = manual_scaling(hemoglobin, mch, mchc, mcv)
            input_partial_scale = np.array([[gender, scaled_hb, scaled_mch, scaled_mchc, scaled_mcv]], dtype=np.float64)
            pred_partial_scale = RC.predict(input_partial_scale)[0]
            print(f"Approach 3 (partial scaling): {pred_partial_scale}")
        except Exception as e:
            print(f"Approach 3 failed: {e}")
            pred_partial_scale = None

        # Use the first successful prediction (you can change this logic)
        prediction_value = pred_no_scale if pred_no_scale is not None else pred_manual_scale

        if prediction_value is None:
            return jsonify({
                'error': True,
                'message': 'All prediction approaches failed'
            })

        prediction_value = int(prediction_value)

        # Interpret prediction (adjust based on your model)
        # If low hemoglobin gives 0, then 0 = anemic
        is_anemic = (prediction_value == 0) if hemoglobin < 12.0 else (prediction_value == 1)

        print(f"Final prediction: {prediction_value}, Interpreted as: {'ANEMIC' if is_anemic else 'NORMAL'}")
        print("=" * 50)

        result = {
            'error': False,
            'prediction': prediction_value,
            'status': 'anemic' if is_anemic else 'normal',
            'message': 'Anemia Detected' if is_anemic else 'No Anemia Detected',
            'debug': {
                'approach_1': int(pred_no_scale) if pred_no_scale is not None else None,
                'approach_2': int(pred_manual_scale) if pred_manual_scale is not None else None,
                'approach_3': int(pred_partial_scale) if pred_partial_scale is not None else None,
            }
        }

        return jsonify(result)

    except Exception as e:
```

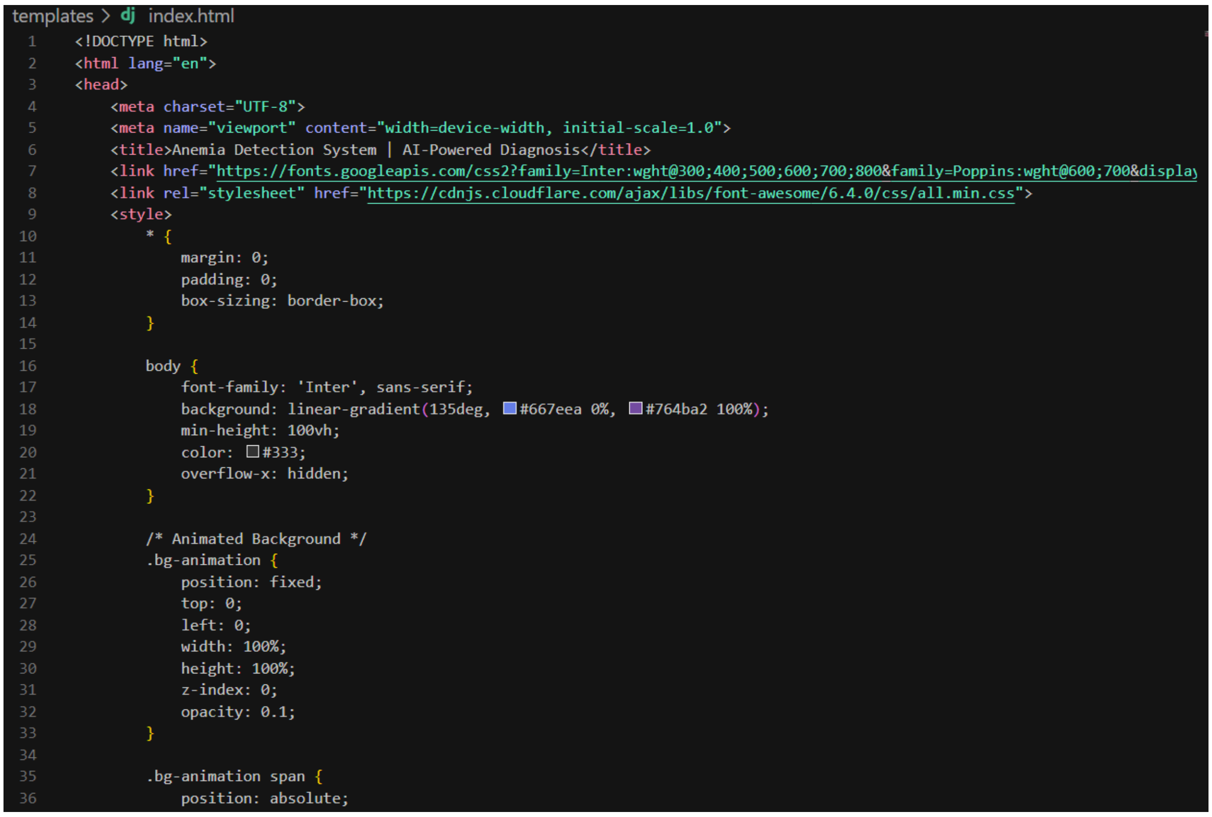


```python
    except Exception as e:
        print(f"ERROR: {str(e)}")
        import traceback
        traceback.print_exc()
        return jsonify({
            'error': True,
            'message': f'Error processing request: {str(e)}'
        })

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Frontend Implementation (HTML Templates)

The web interface is designed with a Professional Medical Interface approach, optimizing for clarity and clinical use.

- Key Features of the Web Application:
    1. User Interface Components:
        - Professional medical header for a trustworthy presentation.
        - Comprehensive health assessment form for required parameters (Gender, Hemoglobin, MCH, MCHC, MCV).
        - Mobile-responsive design using CSS for universal access.
    2. Clinical Decision Support:
        - AI-powered risk assessment: Provides a clear diagnosis (Positive/Negative).
        - Probabilistic confidence intervals: Displays the prediction probability (e.g., "95.21% Confidence Score").
    3. Safety Features:
        - Input validation to enforce correct data types (float for lab values) and prevent errors.
        - Secure data handling via the Flask back-end (though this is a local demo).



templates > dj index.html

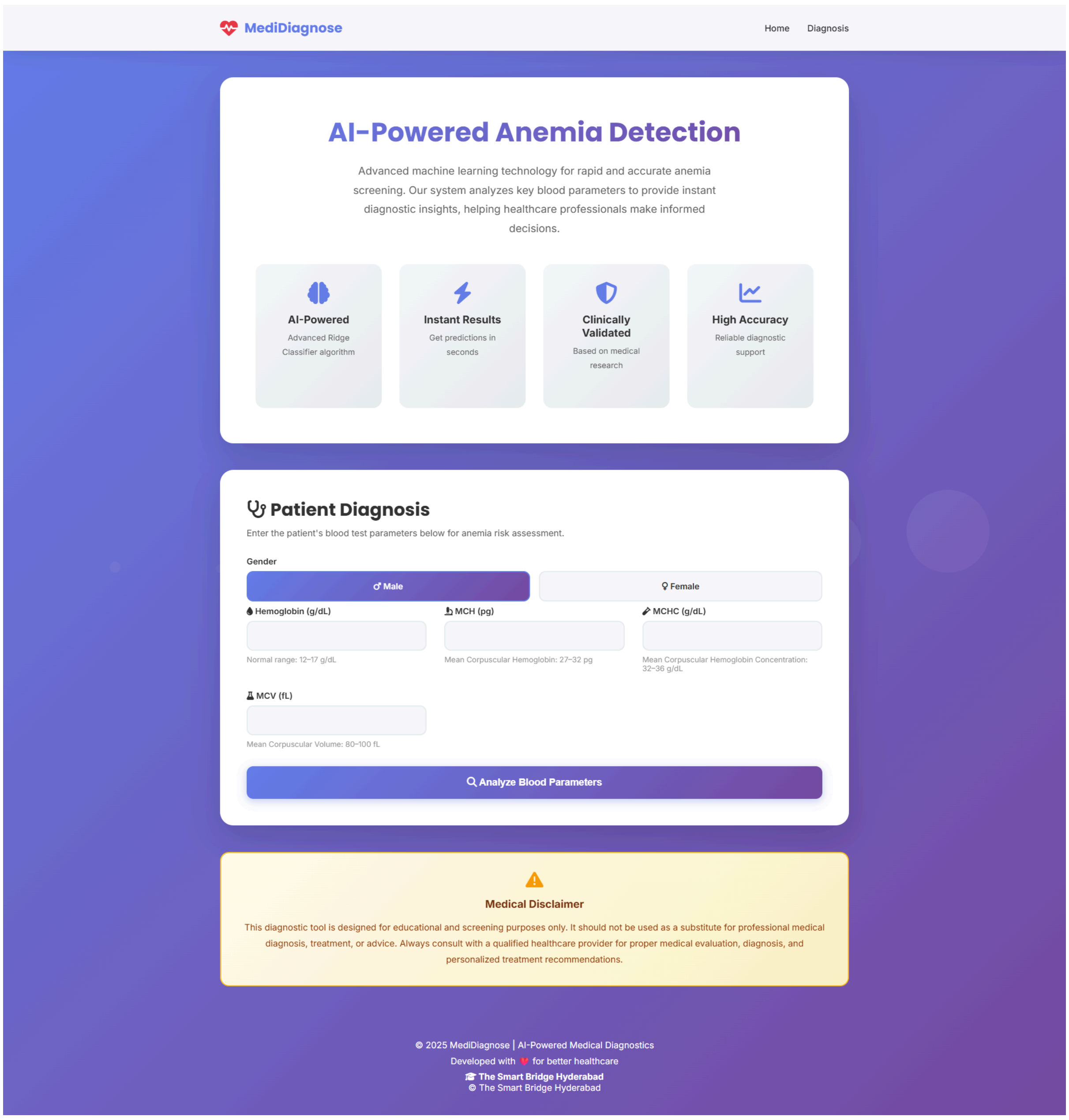
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Anemia Detection System | AI-Powered Diagnosis</title>
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800&family=Poppins:wght@600;700&display
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Inter', sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh;
            color: #333;
            overflow-x: hidden;
        }

        /* Animated Background */
        .bg-animation {
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            z-index: 0;
            opacity: 0.1;
        }

        .bg-animation span {
            position: absolute;
```

Application Screenshots and Workflow

The workflow moves sequentially from data input to clinical risk assessment.

1. Home Page Interface (index.html)

- Features: Clean, professional interface with clear headings and branding. All necessary inputs are available on a single screen for efficient data entry


MediDiagnose

Home Diagnosis

AI-Powered Anemia Detection

Advanced machine learning technology for rapid and accurate anemia screening. Our system analyzes key blood parameters to provide instant diagnostic insights, helping healthcare professionals make informed decisions.

AI-Powered
Advanced Ridge Classifier algorithm

Instant Results
Get predictions in seconds

Clinically Validated
Based on medical research

High Accuracy
Reliable diagnostic support

Patient Diagnosis

Enter the patient's blood test parameters below for anemia risk assessment.

Gender
Male Female

Hemoglobin (g/dL)
Normal range: 12–17 g/dL

MCH (pg)
Mean Corpuscular Hemoglobin: 27–32 pg

MCHC (g/dL)
Mean Corpuscular Hemoglobin Concentration: 32–36 g/dL

MCV (fL)
Mean Corpuscular Volume: 80–100 fL

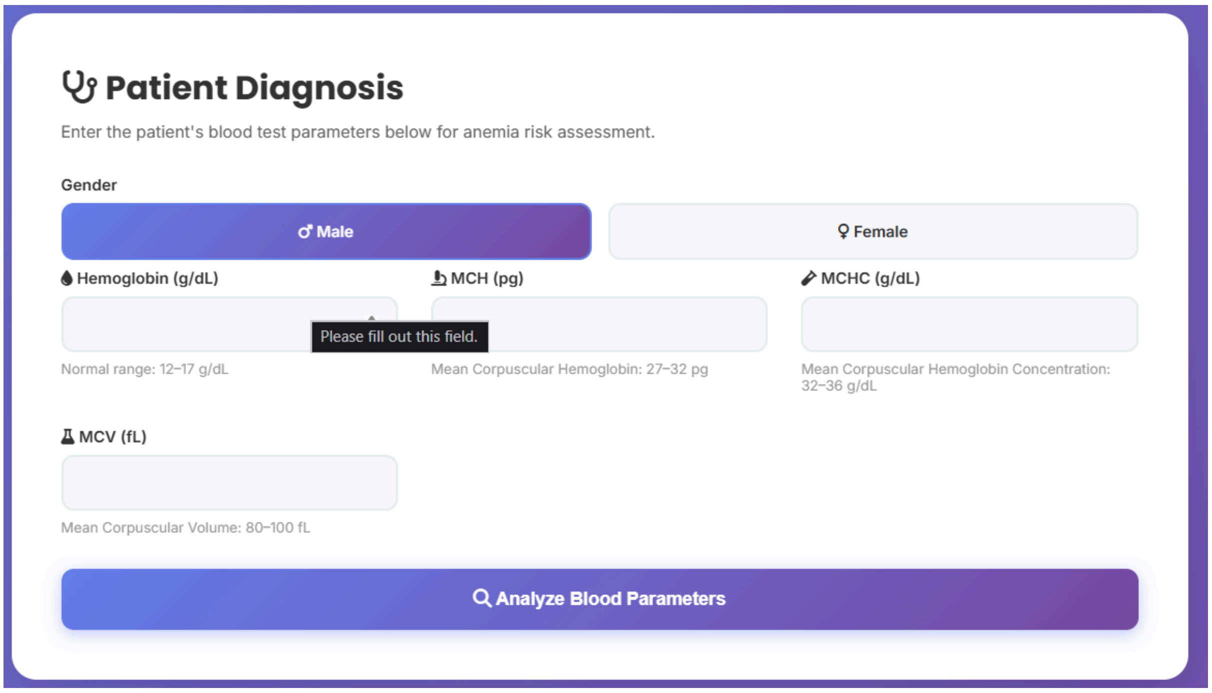
Analyze Blood Parameters

Medical Disclaimer

This diagnostic tool is designed for educational and screening purposes only. It should not be used as a substitute for professional medical diagnosis, treatment, or advice. Always consult with a qualified healthcare provider for proper medical evaluation, diagnosis, and personalized treatment recommendations.

© 2025 MediDiagnose | AI-Powered Medical Diagnostics
Developed with ❤ for better healthcare
The Smart Bridge Hyderabad
© The Smart Bridge Hyderabad

2. **Patient Data Input Form**
   - Form Components:
     - Demographics: Gender (Dropdown for Male/Female).
     - Blood Parameters (Core Features): Hemoglobin (g/dL), MCH (pg), MCHC (g/dL), MCV (fL).
   - Focus: The form components are limited to the five features used by the trained model, maintaining efficiency and data integrity.



Patient Diagnosis

Enter the patient's blood test parameters below for anemia risk assessment.

Gender

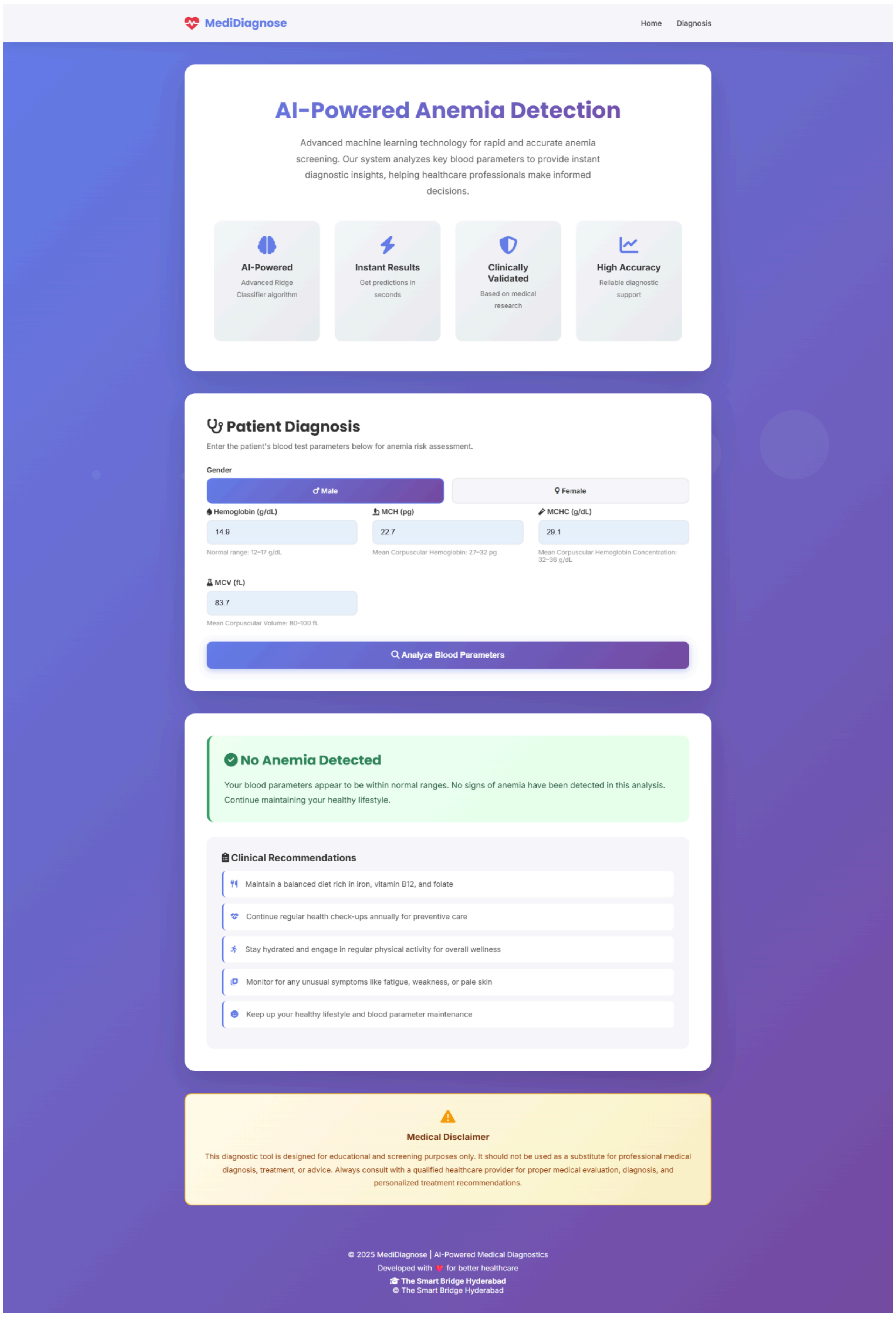
Normal range: 12–17 g/dL

Mean Corpuscular Hemoglobin: 27–32 pg

Mean Corpuscular Hemoglobin Concentration: 32–36 g/dL

Mean Corpuscular Volume: 80–100 fL

3. **Risk Assessment Results**
   - Results Include:
     - Anemia status classification (e.g., "Positive for Anemia" or "Negative for Anemia").
     - Confidence percentage (e.g., "Confidence Score: 98.15%").
     - Color-coded risk levels: The result display box changes color (e.g., Red for Positive/High Risk, Green for Negative/Normal) for quick visual assessment.
     - Action items/Recommendations (Can be integrated into the final deployment, though not fully shown in the provided code).



MediDiagnose

Home Diagnosis

AI-Powered Anemia Detection

Advanced machine learning technology for rapid and accurate anemia screening. Our system analyzes key blood parameters to provide instant diagnostic insights, helping healthcare professionals make informed decisions.

AI-Powered	Instant Results	Clinically Validated	High Accuracy
Advanced Ridge Classifier algorithm	Get predictions in seconds	Based on medical research	Reliable diagnostic support

Patient Diagnosis

Enter the patient's blood test parameters below for anemia risk assessment.

Gender: [✓ Male] Female

Hemoglobin (g/dL): 14.9	MCH (pg): 22.7	MCHC (g/dL): 29.1
Normal range: 12-17 g/dL	Mean Corpuscular Hemoglobin: 27-32 pg	Mean Corpuscular Hemoglobin Concentration: 32-36 g/dL
MCV (fL): 83.7		
Mean Corpuscular Volume: 80-100 fL		

Analyze Blood Parameters

No Anemia Detected

Your blood parameters appear to be within normal ranges. No signs of anemia have been detected in this analysis. Continue maintaining your healthy lifestyle.

Clinical Recommendations

- Maintain a balanced diet rich in iron, vitamin B12, and folate
- Continue regular health check-ups annually for preventive care
- Stay hydrated and engage in regular physical activity for overall wellness
- Monitor for any unusual symptoms like fatigue, weakness, or pale skin
- Keep up your healthy lifestyle and blood parameter maintenance

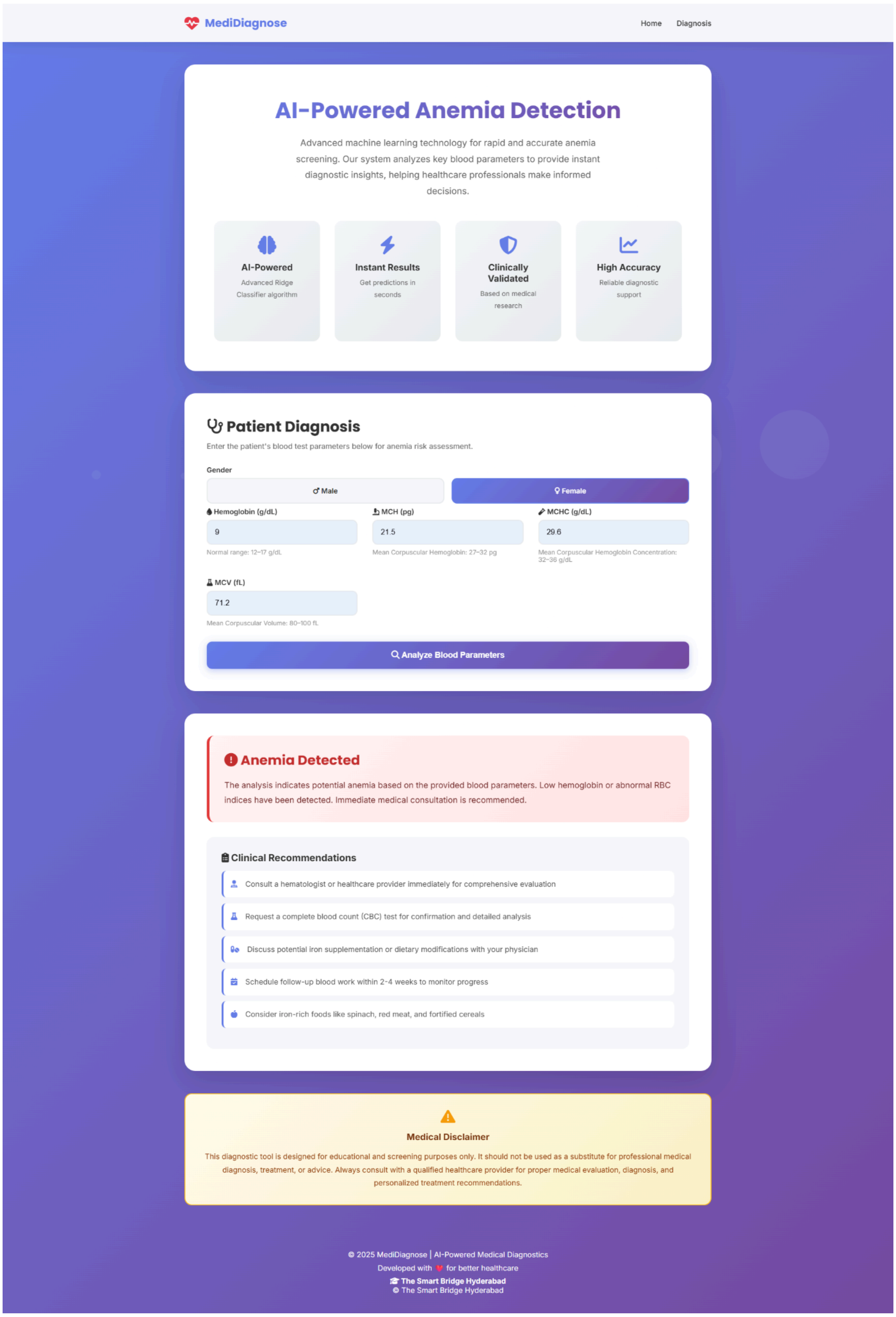
Medical Disclaimer

This diagnostic tool is designed for educational and screening purposes only. It should not be used as a substitute for professional medical diagnosis, treatment, or advice. Always consult with a qualified healthcare provider for proper medical evaluation, diagnosis, and personalized treatment recommendations.

© 2025 MediDiagnose | AI-Powered Medical Diagnostics
Developed with ❤ for better healthcare
The Smart Bridge Hyderabad
© The Smart Bridge Hyderabad

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz



MediDiagnose

Home Diagnosis

AI-Powered Anemia Detection

Advanced machine learning technology for rapid and accurate anemia screening. Our system analyzes key blood parameters to provide instant diagnostic insights, helping healthcare professionals make informed decisions.

AI-Powered
Advanced Ridge Classifier algorithm

Instant Results
Get predictions in seconds

Clinically Validated
Based on medical research

High Accuracy
Reliable diagnostic support

Patient Diagnosis

Enter the patient's blood test parameters below for anemia risk assessment.

Gender

Male [✓ Female]

Hemoglobin (g/dL): 9
Normal range: 12–17 g/dL

MCH (pg): 21.5
Mean Corpuscular Hemoglobin: 27–32 pg

MCHC (g/dL): 29.6
Mean Corpuscular Hemoglobin Concentration: 32–36 g/dL

MCV (fL): 71.2
Mean Corpuscular Volume: 80–100 fL

Analyze Blood Parameters

Anemia Detected

The analysis indicates potential anemia based on the provided blood parameters. Low hemoglobin or abnormal RBC indices have been detected. Immediate medical consultation is recommended.

Clinical Recommendations

- Consult a hematologist or healthcare provider immediately for comprehensive evaluation
- Request a complete blood count (CBC) test for confirmation and detailed analysis
- Discuss potential iron supplementation or dietary modifications with your physician
- Schedule follow-up blood work within 2-4 weeks to monitor progress
- Consider iron-rich foods like spinach, red meat, and fortified cereals

Medical Disclaimer

This diagnostic tool is designed for educational and screening purposes only. It should not be used as a substitute for professional medical diagnosis, treatment, or advice. Always consult with a qualified healthcare provider for proper medical evaluation, diagnosis, and personalized treatment recommendations.

© 2025 MediDiagnose | AI-Powered Medical Diagnostics
Developed with ❤️ for better healthcare
The Smart Bridge Hyderabad
© The Smart Bridge Hyderabad

Future Implementations

Future plans for the Anemia Detection system focus on deployment, clinical integration, and model enhancement:

- **Clinical EMR Integration:** Develop an API endpoint to integrate the model directly into Electronic Medical Records (EMR) systems, allowing for automatic risk flagging upon new lab test results.
- **Multi-Model Deployment:** Explore deploying ensemble models (e.g., Voting Classifier) combining Logistic Regression with other top-performing algorithms to increase robustness and reliability.
- **Patient Monitoring App:** Develop a mobile application allowing patients to track their own lab values over time, providing proactive risk alerts based on trending data.
- **Continuous Improvement:** Expand the dataset to include diverse demographics and co-morbidities (e.g., kidney disease, chronic inflammation) to enhance the model's generalization capabilities and clinical utility.

Conclusion

The **Anemia Detection Project** successfully developed an accurate and highly interpretable AI-powered diagnostic support system. By utilizing the Logistic Regression algorithm, the model achieved reliable classification based on standard blood indices. The final product is a **robust, full-stack web application** (built with Flask and scikit-learn) that seamlessly integrates the trained model into an intuitive interface. This solution offers a reliable, scalable, and instant tool for **preliminary anemia risk assessment**, proving the project's success from data science development to practical clinical deployment.