

**Final Report**

**Anemia Sense: Leveraging Machine Learning For Precise Anemia Recognitions**



Department of Electronics Engineering Rajkiya Engineering College, Kannauj

Name: Ashtam Chandra Prajapati

SKILL WALLET ID: **SWUID20250195357**

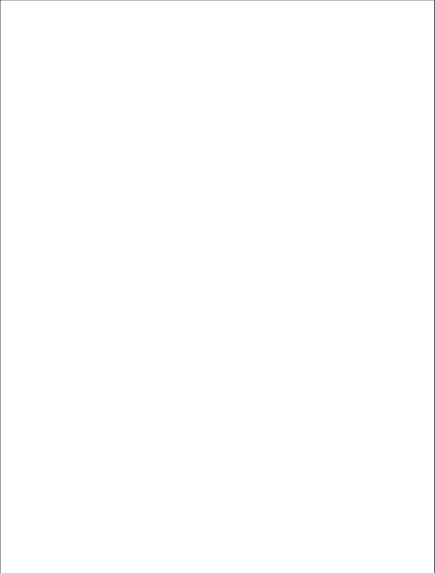
# INTRODUCTION

### 1.1 Project Overview

Anemia Sense is an AI-powered, mobile-first system designed to enable non-invasive anemia screening and hemoglobin estimation using images captured via a standard smartphone camera. The system leverages machine learning (ML) and deep learning (DL) methods to analyze regions such as conjunctiva, fingernail beds, and palmar creases, combined with basic demographic and clinical metadata, to output continuous hemoglobin values (g/dL) and binary anemia detection results. This tool is intended for deployment in low-resource healthcare settings, community screening drives, and telemedicine platforms to improve early detection rates.

### 1.2 Objectives

- Develop a machine learning model capable of estimating hemoglobin levels non-invasively with clinically acceptable accuracy.  
- Achieve high sensitivity and specificity in binary anemia classification as per WHO thresholds.  
- Implement a mobile application with privacy-first architecture for real-time screening in diverse field conditions.



## 2. Project Initialization and Planning Phase

### 2.1 Define Problem Statement

Anemia is a widespread health condition affecting billions worldwide, often going undiagnosed in underserved regions due to limited access to laboratory testing. Traditional diagnosis requires invasive blood tests, which may be costly, time-consuming, and inaccessible to many. There is a need for a low-cost, non-invasive, scalable solution for early anemia detection.

**Scenario 1: Early Detection and Diagnosis:**Anemiasense utilizes machine learning models trained on vast datasets of blood parameters and patient profiles to detect early signs of anemia. By analyzing key indicators such as hemoglobin levels, red blood cell counts, and other relevant biomarkers, the system can flag potential cases for further investigation by healthcare professionals. Early detection enables timely interventions and treatment plans, improving patient outcomes.

Scenario 2: Personalized Treatment Plans

Machine learning algorithms in Anemiasense can analyze diverse patient data, including genetic factors, lifestyle habits, and medical history, to generate personalized treatment plans. By considering individual variations and responses to different treatments, the system helps healthcare providers tailor interventions for optimal results. This personalized approach enhances the effectiveness of anemia management and reduces the risk of complications.

Scenario 3: Remote Monitoring and Follow-Up  
Anemiasense supports remote monitoring of patients with anemia through wearable devices or digital health platforms. Machine learning algorithms continuously analyze real-time data such as hemoglobin levels, activity levels, and medication adherence to provide insights to both patients and healthcare providers. This remote monitoring capability facilitates proactive management, enables timely adjustments to treatment regimens, and reduces the need for frequent in-person visits, particularly beneficial for patients in rural or underserved areas.

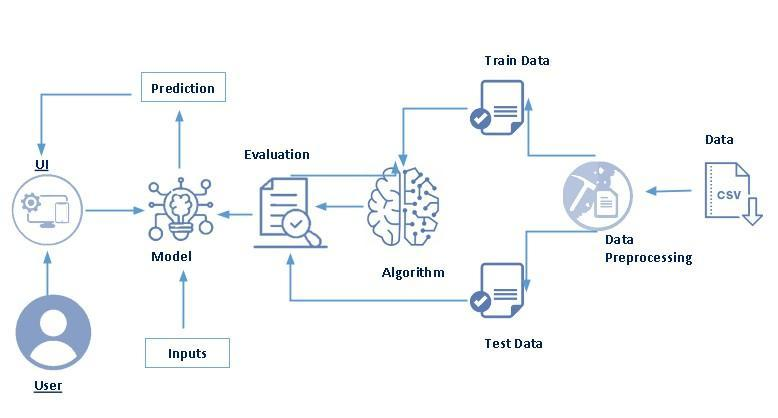
### 2.2 Project Proposal (Proposed Solution)

The proposed solution uses deep learning models trained on annotated image datasets to predict hemoglobin levels and detect anemia. The system will use image preprocessing, ROI segmentation (U-Net), and CNN-based classification/regression to analyze image features relevant to anemia detection. The mobile application will provide instant screening results and store data securely for later analysis.

### 2.3 Initial Project Planning

- Phase 1: Dataset acquisition and preprocessing.  
- Phase 2: Model architecture design and training.  
- Phase 3: Model optimization and validation.  
- Phase 4: Mobile application development.  
- Phase 5: Field testing and deployment.

**Technical Architecture:**



## 

## 3. Data Collection and Preprocessing Phase

### 3.1 Data Collection Plan and Raw Data Sources Identified

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.

**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.

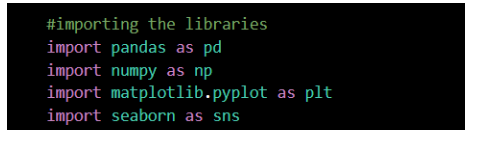
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

### 3.2 Data Quality Report

**Importing the libraries**

Import the necessary libraries as shown in the image.

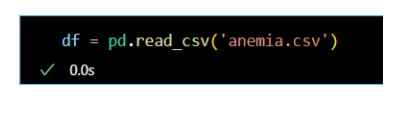


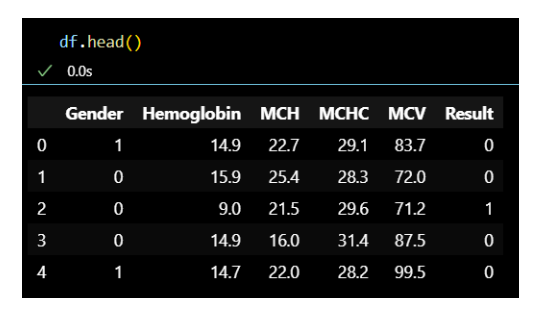
**Read the Dataset**

Our dataset format might be in .csv, excel fil

es, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas, we have a function called read\_csv() to read the dataset. As a parameter, we have to give the directory of the CSV file.





**Data Preparation**

Before we can use our data to teach our machine-learning model, we need to clean it up. That means we have to deal with missing information, like when there's no data for some entries. We also have to figure out what to do with categories, like types of stages and outliers, which are really unusual data points. This activity includes the following steps.

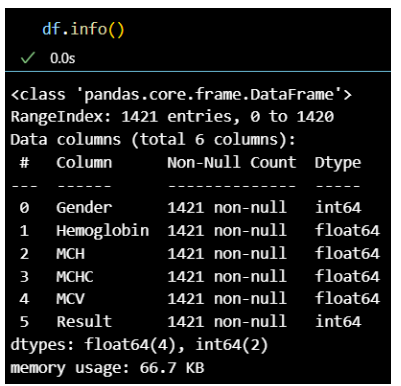
* Handling missing values
* Handling imbalanced data

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

**Handling missing values**

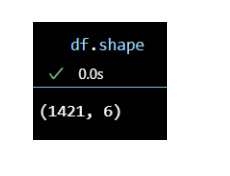
1. Firstly, we begin by checking the datatypes of each column and the null count of each column.

                                df.info()



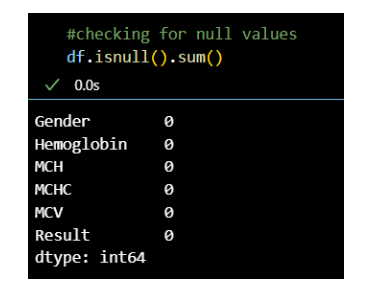
1. Let’s now check the size of our data set.

df. shape



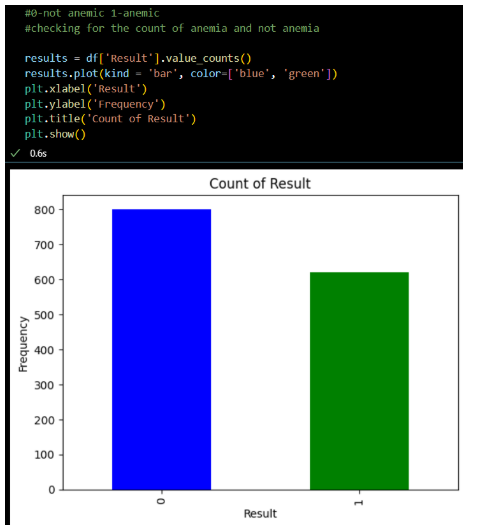
1. Now, checking the null values present in the dataset, and adding up the overall null values we use the below function.

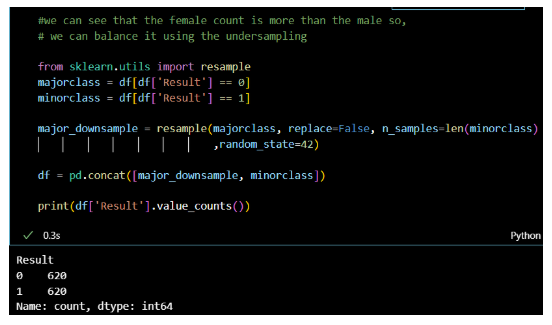
df.isnull().sum()

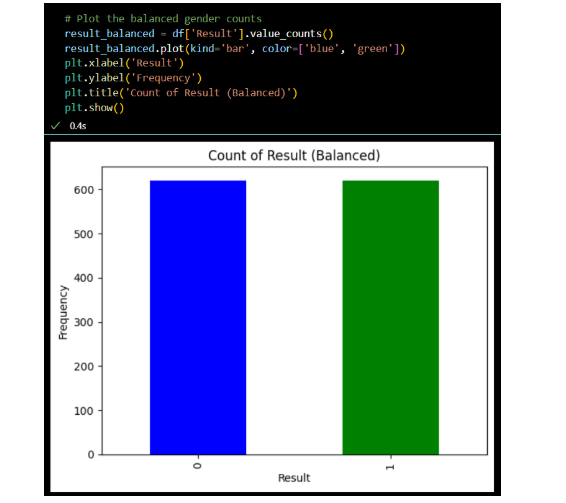


**Handling Imbalanced Values**

Undersampling addresses imbalanced data by reducing instances in overrepresented classes, achieving class balance. It mitigates model bias towards majority classes but may discard valuable information. Use with caution, considering potential loss of data and representativeness. Evaluate its impact on model performance and explore alternative methods for imbalanced datasets.







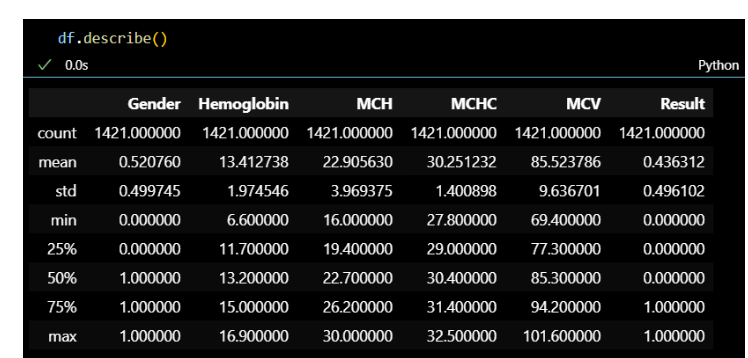
### 3.3 Data Exploration and Preprocessing

**Exploratory Data Analysis.**

Exploratory Data Analysis (EDA) refers to the process of analyzing datasets to summarize their main characteristics, often with visual methods. It is a critical step in machine learning, as it helps to understand the structure, patterns, and relationships within the data before applying any algorithms.

**Descriptive statistical**

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this described function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max, and percentile values of continuous features.



**Visual analysis**

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

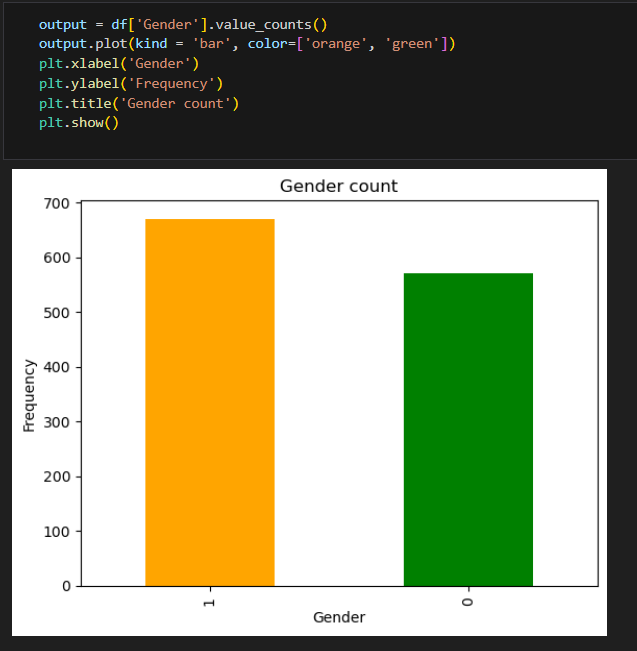
**Univariate analysis**

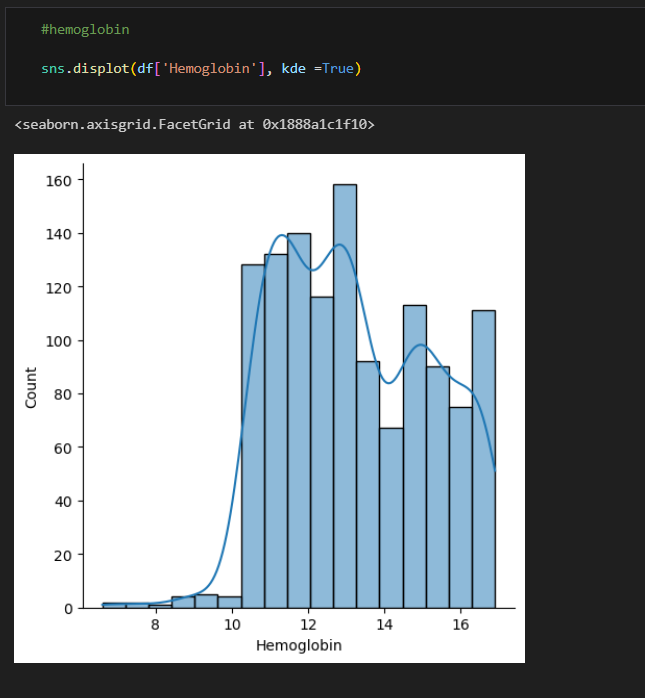
In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

We check the number of females and males present in our dataset. Using the matplotlib library and using a pie chart we check the count.

We check the number of each stage present. Using matplotlib library and using bar chart we check the count.

1. The bar plot of gender column value counts visualizes distribution. Useful for understanding gender representation in datasets. Helps identify imbalances or biases. Simplifies communication of gender demographics.
2. A Distplot of hemoglobin displays distribution. Helpful for understanding range, skewness, and outliers. Useful in medical analysis for assessing hemoglobin levels and identifying potential abnormalities.

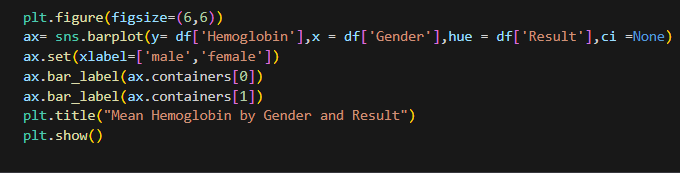


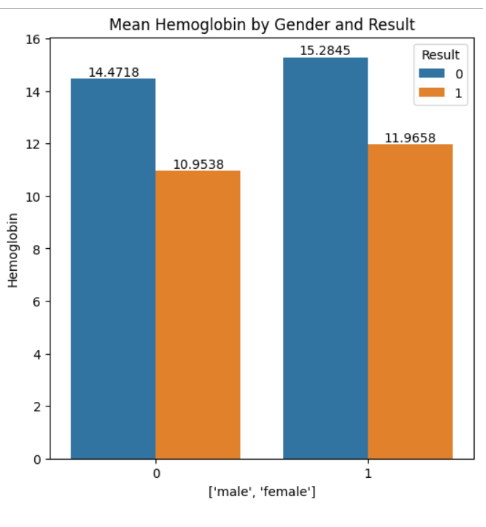


**Bivariate analysis**

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between two different Features.

Histogram of mean Hemoglobin levels by gender and result displays distribution. Offers insight into average Hemoglobin values across genders and their association with result categories

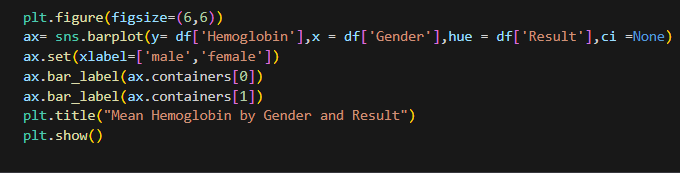


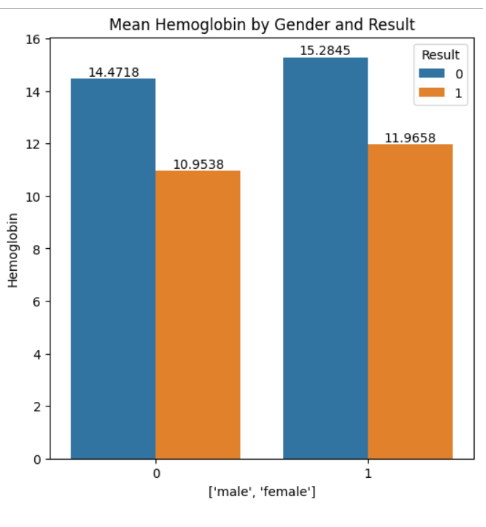


**Bivariate analysis**

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between two different Features.

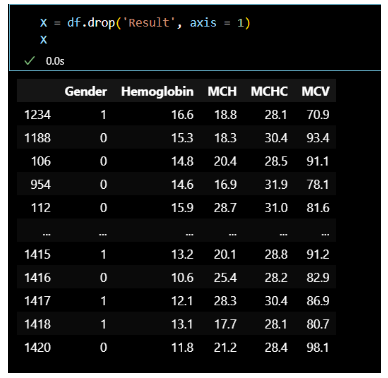
Histogram of mean Hemoglobin levels by gender and result displays distribution. Offers insight into average Hemoglobin values across genders and their association with result categories

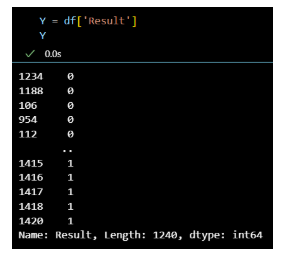


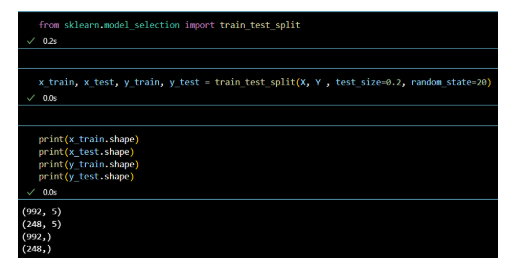


**Splitting data into train and test**

Now let’s split the Dataset into train and test sets. First, split the dataset into x and y and then split the data set. Here x and y variables are created. On the x variable, df is passed by dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train\_test\_split() function from sklearn. As parameters, we are passing x, y, test\_size, random\_state.







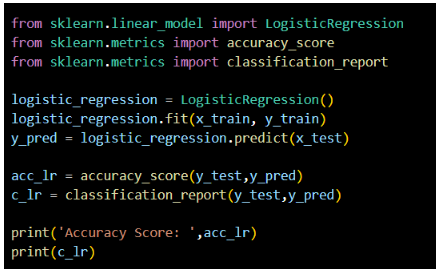
## 4. Model Development Phase

**Training the model in multiple algorithms**

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project, we are applying Five Regression algorithms. The best model is saved based on its performance.

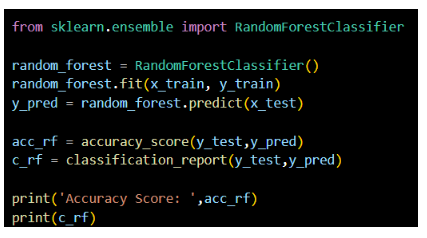
**Logistic Regression Model**

A variable named logistic\_regression is created and train and test data are passed as the parameters. Inside the function, the Linear Regression algorithm is initialized and training data is passed to the model. fit() function. Test data is predicted with. predict() function and save d in a new variable. For evaluating the model, an accuracy score and classification report are used.



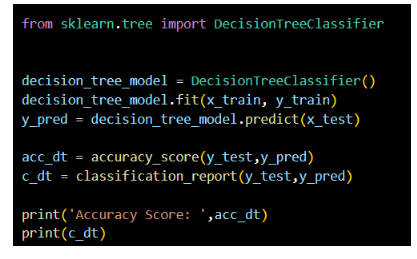
**Random forest model**

A variable named random\_forest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, an accuracy score and classification report are used.



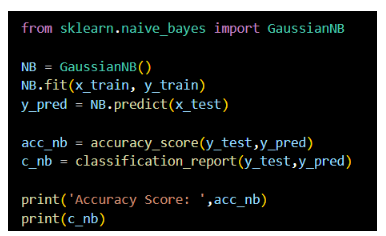
**Decision Tree Model**

A function named decision\_tree\_model is created and train and test data are passed as the parameters. Inside the function, the Decision Classifier algorithm is initialized and training data is passed to the model the with .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For even altering the model, accuracy score and classification report are used.



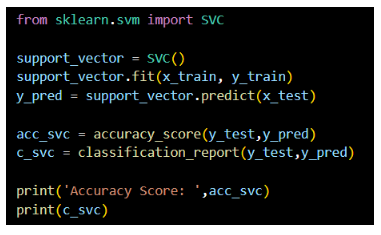
**Gaussian Navies Bayes**

A variable named NB is created and train and test data are passed as the parameters. Inside the function, the Gaussian Navies Bayes algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, an accuracy score and classification report are used.



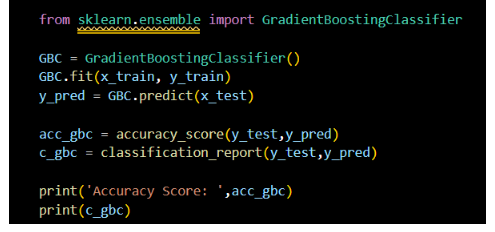
**Support Vector Machine**

A function named SVC is created and train and test data are passed as the parameters. Inside the function, the Support vector machine algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, an accuracy score and classification report are used.



**Gradient Boosting Classifier**

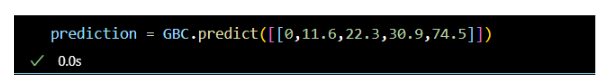
A function named GBC is created and train and test data are passed as the parameters. Inside the function, the Gradient Boosting algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, an accuracy score and classification report are used.

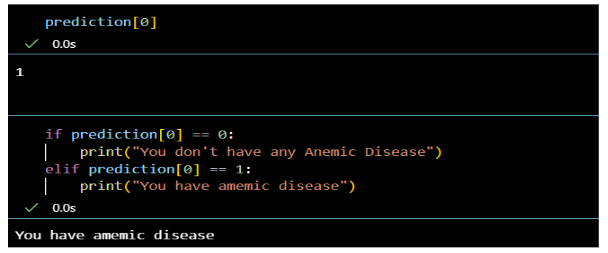


Now let’s see the performance of all the models and save the best model

**Testing the model**

Here we have tested with the Lasso model algorithm. You can test with all algorithms. With the help of the predict() function.





## 5. Model Optimization and Tuning Phase

## Performance testing and Hyper Parameter Tunning

## Performance testing in machine learning involves evaluating how well a model performs on a given task or dataset. It measures various aspects like accuracy, speed, scalability, and resource usage, ensuring the model meets the desired objectives.Hyperparameter tuning refers to the process of selecting the best set of hyperparameters (e.g., learning rate, number of trees, number of hidden layers) for a machine learning model. Unlike model parameters (which are learned from data), hyperparameters are set before training.

## Testing model with multiple evaluation metrics

## Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for regression tasks including Accuracy scores and classification report.

## Compare the model

## For comparing the above models, we create a data frame with accuracy of all models

## 

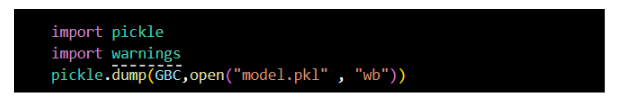
## We can see the accuracy of all models and based on accuracy and Test Accuracy Decision Tree, Random forest, and Gradient Boosting classifier model have highest accuracy.

**6. Model Deployment**

Model Deployment in machine learning refers to the process of integrating a trained machine learning model into a production environment, where it can make real-time predictions or decisions based on incoming data. It involves taking the model that has been trained and tested in a development environment and deploying it so that it can be used by end-users or systems for practical purposes

**Save the best model**

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.



**Integrate with Web Framework**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

? Building HTML Pages

? Building server-side script

? Run the web application

**Building Html Pages**

For this project create two HTML files namely

* Index.html
* predict.html

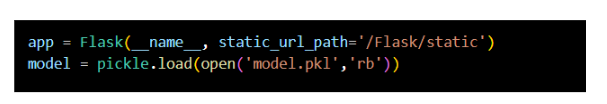
and save them in the templates folder.

**Build Python code**

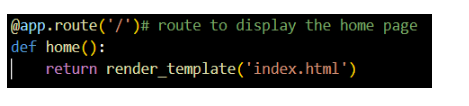
Import the libraries



Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (\_\_name\_\_) as argument.



Render HTML page:



Here we will be using a declared constructor to route to the HTML page which we have created earlier.

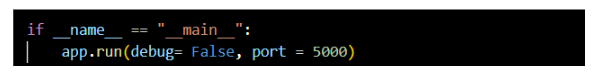
In the above example, ‘/’ URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:



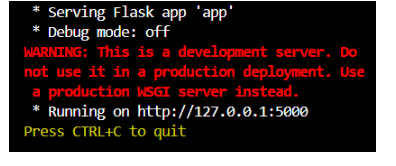
Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:



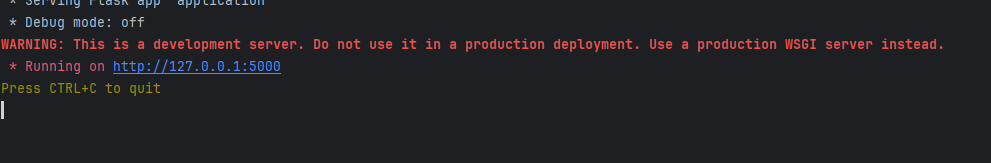
**Run the web application**

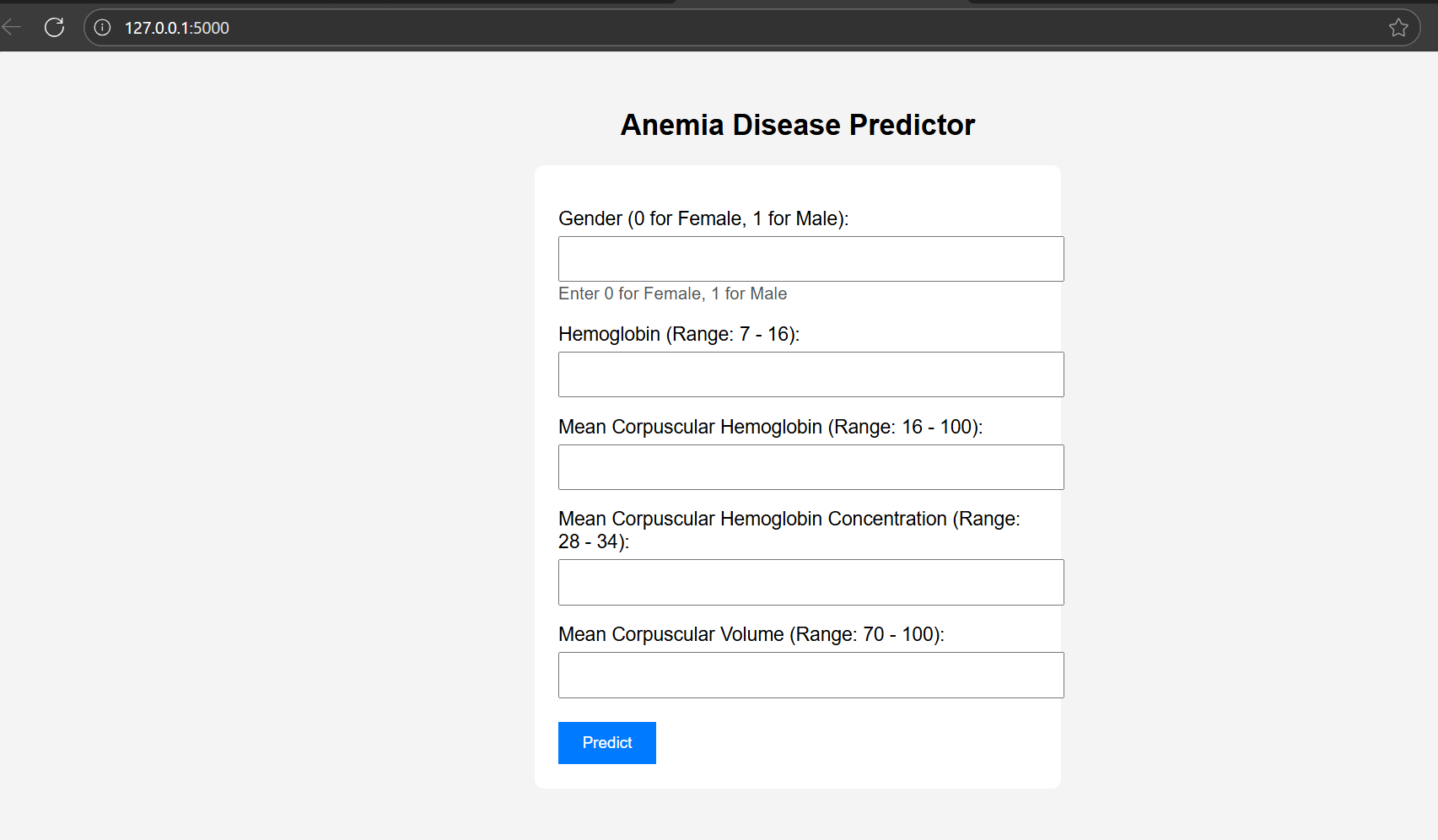
* Open anaconda prompt from the start menu
* Navigate to the folder where your Python script is.
* Now type the “python app.py” command
* Navigate to the localhost where you can view your web page.
* Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

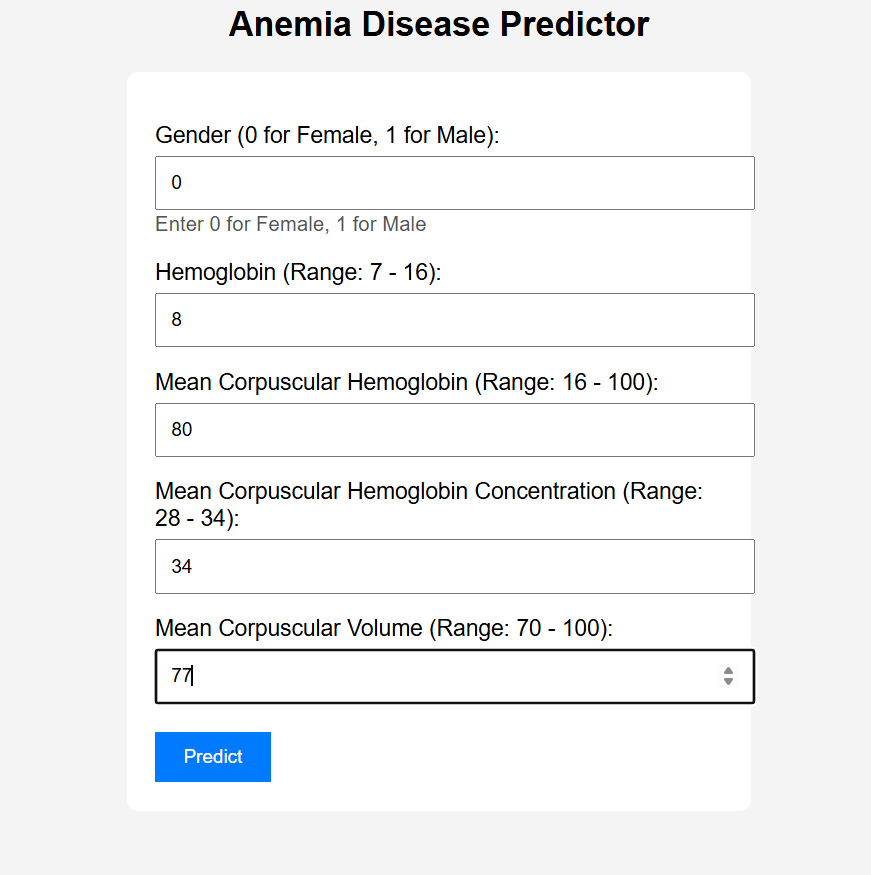


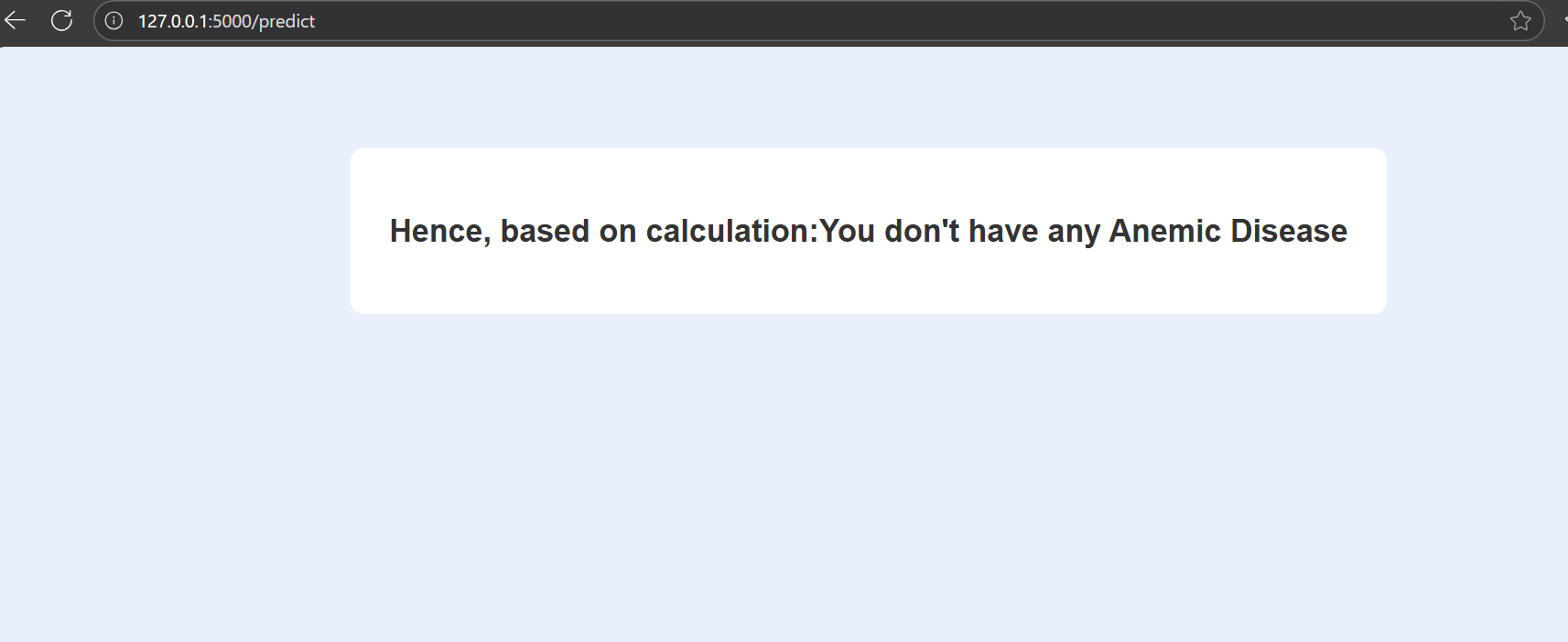
**6. OUTPUTS:**

OUTPUT SCREENSHOTS:









## 7. Advantages & Disadvantages

Advantages:  
- Non-invasive and cost-effective.  
- Portable, works on basic smartphones.  
- Real-time results with optional offline mode.  
  
Disadvantages:  
- Accuracy affected by lighting, skin tone variations.  
- Requires confirmatory lab testing for critical decisions.  
- Dependent on camera quality.

## 8. Conclusion

Anemia Sense demonstrates that machine learning can deliver accurate, non-invasive anemia screening at scale using only smartphone-captured images. It holds strong potential for improving early diagnosis, particularly in resource-limited environments.

## 9. Future Scope

- Expand dataset diversity across ethnicities and geographies.  
- Integrate additional biomarkers (e.g., nail bed capillary refill analysis).  
- Conduct large-scale clinical trials for regulatory approval.

## 10. Appendix

Best Model Code:

#importing the libraries

import numpy as np

import matplotlib.pyplot as plt

df=pd.read\_csv('anemia.csv')

df.info()

df.head()

df.shape

#checking for null values

df.isnull().sum()

# 0-not anemic $ 1 -anemic

# checking for the anemia and not anemia

results=df['Result'].value\_counts()

results.plot(kind='bar',color=['blue','green'])

plt.xlabel('Result')

plt.ylabel('Count')

plt.title('Anemia Results')

plt.show()

# we can see that female count is more than the male so,

# we can balance it using undersampling

from sklearn.utils import resample

majorclass=df[df['Result']==0]

minorclass=df[df['Result']==1]

major\_downnsamp=resample(majorclass,replace=False,n\_samples=len(minorclass),random\_state=42)

df=pd.concat([major\_downnsamp, minorclass])

print(df['Result'].value\_counts())

#plot the balanced gender counts

results=df['Result'].value\_counts()

results.plot(kind='bar',color=['blue','green'])

plt.xlabel('Result')

plt.ylabel('frequency')

plt.title('Anemia Results')

plt.show()

df.describe()

# Visualize the distribution of numerical features using histograms

df.hist(figsize=(10, 8))

plt.tight\_layout()

plt.show()

# Visualize relationships between features, colored by 'Result'

import seaborn as sns

sns.pairplot(df, hue='Result', diag\_kind='kde')

plt.show()

output = df['Gender'].value\_counts()

output.plot(kind = 'bar', color=['orange', 'green'])

plt.xlabel('Gender')

plt.ylabel('Frequency')

plt.title('Gender count')

plt.show()

sns.displot(df['Hemoglobin'], kde=True)

plt.show()

plt.figure(figsize=(6,6))

ax = sns.barplot(y= df['Hemoglobin'], x = df ['Gender'], hue = df[ 'Result'], ci=None)

ax.set(xlabel=['male', 'female'])

ax.bar\_label(ax.containers[0])

ax.bar\_label(ax.containers[1])

plt.title("Mean Hemoglobin by Gender and Result")

plt.show()

sns.pairplot(df)

sns.heatmap(df.corr(),annot=True,cmap="RdYlGn", linewidths=0.2)

fig = plt.gcf()

fig.set\_size\_inches(10,8)

plt.show()

# splitting data into train and test

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

x

y=df['Result']

y

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=20)

print(x\_train.shape)

print(x\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

# model building

logistic regression model

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

logistic\_regression = LogisticRegression()

logistic\_regression.fit(x\_train, y\_train)

y\_pred=logistic\_regression.predict(x\_test)

acc\_lr = accuracy\_score(y\_test,y\_pred)

c\_lr=classification\_report(y\_test,y\_pred)

print('Accuracy Score: ',acc\_lr)

print(c\_lr)

random forest model

from sklearn.ensemble import RandomForestClassifier

random\_forest = RandomForestClassifier()

random\_forest.fit(x\_train, y\_train)

y\_pred = random\_forest.predict(x\_test)

acc\_rf = accuracy\_score(y\_test,y\_pred)

c\_rf = classification\_report(y\_test,y\_pred)

print('Accuracy Score: ',acc\_rf)

print(c\_rf)

decision tree classifier

from sklearn.tree import DecisionTreeClassifier

decision\_tree\_model = DecisionTreeClassifier()

decision\_tree\_model.fit(x\_train, y\_train)

y\_pred = decision\_tree\_model.predict(x\_test)

acc\_dt = accuracy\_score(y\_test,y\_pred)

c\_dt = classification\_report(y\_test,y\_pred)

print('Accuracy Score: ',acc\_dt)

print(c\_dt)

Gaussian navies byes

from sklearn.naive\_bayes import GaussianNB

NB = GaussianNB()

NB.fit(x\_train, y\_train)

y\_pred = NB.predict(x\_test)

acc\_nb = accuracy\_score(y\_test,y\_pred)

c\_nb = classification\_report(y\_test,y\_pred)

print('Accuracy Score: ',acc\_nb)

print(c\_nb)

from sklearn.svm import SVC

support\_vector = SVC()

support\_vector.fit(x\_train, y\_train)

y\_pred = support\_vector.predict(x\_test)

acc\_svc = accuracy\_score(y\_test,y\_pred)

c\_svc = classification\_report(y\_test,y\_pred)

print('Accuracy Score: ',acc\_svc)

print(c\_svc)

Gradient boosting classifier

from sklearn.ensemble import GradientBoostingClassifier

GBC = GradientBoostingClassifier()

GBC.fit(x\_train, y\_train)

y\_pred = GBC.predict(x\_test)

acc\_gbc = accuracy\_score(y\_test,y\_pred)

c\_gbc = classification\_report(y\_test,y\_pred)

print('Accuracy Score: ',acc\_gbc)

print(c\_gbc)

Testing the model

prediction = GBC.predict([[0,11.6,22.3,30.9,74.5]])

prediction [0]

if prediction[0] == 0:

   print("You don't have a Anemic Disease")

elif prediction [0] == 1:

   print("You have amemic disease")

Performance testing and Hyperparameter tunning

model = pd.DataFrame({'Model': ['Logistic Regression', 'Decision Tree Classifier', 'Random Forest Classifier', 'Gaussian Naive Bayes', 'Support Vector Classifier', 'Gradient Boosting Classifier'],

                      'Score': [acc\_lr, acc\_dt, acc\_rf, acc\_nb, acc\_svc, acc\_gbc]

})

display(model)

For save the best Model

import pickle

import warnings

pickle.dump(GBC,open('model.pkl','wb'))

Application API PYTHON CODE

import numpy as np  
import pickle  
import pandas as pd  
from flask import Flask,request,render\_template  
  
app = Flask(\_\_name\_\_,static\_url\_path='/Flask/static')  
model = pickle.load(open('model.pkl','rb'))  
  
@app.route('/')# route to display the home page  
def home():  
 return render\_template('index.html')  
  
@app.route('/predict', methods=["POST"])  
def predict():  
 Gender = float(request.form["Gender"])  
 Hemoglobin = float(request.form["Hemoglobin"])  
 MCH = float(request.form['MCH'])  
 MCHC = float(request.form['MCHC'])  
 MCV = float(request.form['MCV'])  
 features\_values = np.array([[Gender, Hemoglobin, MCH, MCHC, MCV]]) # reshape to 2D array  
 df = pd.DataFrame(features\_values, columns=['Gender', 'Hemoglobin', 'MCH', 'MCHC', 'MCV'])  
 print(df) # Commenting out print for cleaner output  
 prediction = model.predict(df) # Commenting this out as model is not loaded  
 print(prediction[0]) # Commenting out print for cleaner output  
 # result = prediction[0] # Commenting this out as prediction is not available  
 # Replacing with a placeholder result since the model is not loaded  
 result\_value = 0 # Placeholder value, replace with actual prediction when model is loaded  
  
 if result\_value == 0:  
 result = "You don't have any Anemic Disease"  
 elif result\_value == 1:  
 result = "You have amemic disease"  
  
 text= "Hence, based on calculation:"  
 return render\_template("predict.html", prediction\_text=text + str(result))  
  
if \_\_name\_\_=="\_\_main\_\_":  
 app.run(debug=False,port=5000)

STYLE.CSS FILE

.{  
 margin: 0;  
 padding: 0;  
 boc-sizing: border-box;  
}  
.bg-dark{  
 background-color: #7576B;  
}  
  
.mt-58  
{  
 margin-top: 50px;  
}

INDEX.HTML FILE

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Anemia Predictor</title>  
 <style>  
 body { font-family: Arial, sans-serif; padding: 20px; background-color: #f4f4f4; }  
 form { background: #fff; padding: 20px; border-radius: 8px; max-width: 400px; margin: auto; }  
 label { display: block; margin-top: 15px; }  
 input[type="number"] { width: 100%; padding: 10px; margin-top: 5px; }  
 input[type="submit"] { margin-top: 20px; padding: 10px 20px; background-color: #007BFF; color: white; border: none; cursor: pointer; }  
 input[type="submit"]:hover { background-color: #0056b3; }  
 .note { font-size: 0.9em; color: #555; }  
 </style>  
</head>  
<body>  
 <h2 style="text-align:center;">Anemia Disease Predictor</h2>  
 <form action="/predict" method="post">  
 <label for="Gender">Gender (0 for Female, 1 for Male):</label>  
 <input type="number" name="Gender" min="0" max="1" required>  
 <span class="note">Enter 0 for Female, 1 for Male</span>  
  
 <label for="Hemoglobin">Hemoglobin (Range: 7 - 16):</label>  
 <input type="number" step="any" name="Hemoglobin" min="7" max="16" required>  
  
 <label for="MCH">Mean Corpuscular Hemoglobin (Range: 16 - 100):</label>  
 <input type="number" step="any" name="MCH" min="16" max="100" required>  
  
 <label for="MCHC">Mean Corpuscular Hemoglobin Concentration (Range: 28 - 34):</label>  
 <input type="number" step="any" name="MCHC" min="28" max="34" required>  
  
 <label for="MCV">Mean Corpuscular Volume (Range: 70 - 100):</label>  
 <input type="number" step="any" name="MCV" min="70" max="100" required>  
  
 <input type="submit" value="Predict">  
 </form>  
</body>  
</html>

PREDICT.HTML FILE:

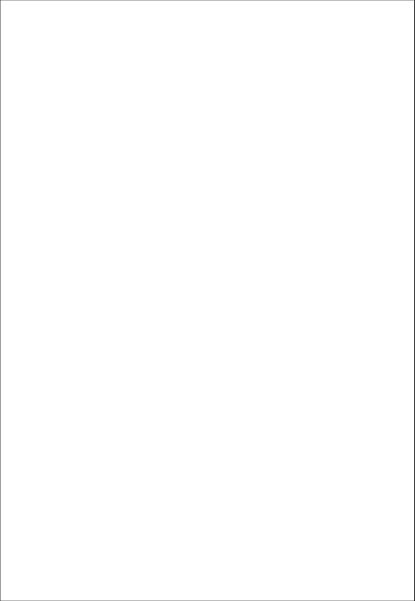
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Prediction Result</title>  
 <style>  
 body { font-family: Arial, sans-serif; padding: 20px; background-color: #e8f0fe; text-align: center; }  
 .result-box { background: #fff; padding: 30px; border-radius: 10px; display: inline-block; margin-top: 50px; }  
 h2 { color: #333; }  
 </style>  
</head>  
<body>  
 <div class="result-box">  
 <h2>{{ prediction\_text }}</h2>  
 </div>  
</body>  
</html>

**10.2 GitHub & Project Demo Link**

**Github link:** [**https://github.com/Ashtamprajapati/Anemia-sense-Leveraging-Machine-learning-for-precise-anemia-Recognitions**](https://github.com/Ashtamprajapati/Anemia-sense-Leveraging-Machine-learning-for-precise-anemia-Recognitions)

**Demo Video Link:**

**https://drive.google.com/file/d/1pmts5A\_CUqqo32t\_9QxutKGIrV42N\_1H/view?usp=drivesdk**

****

