ABSTRACT

Computer-aided illness diagnosis is less expensive, saves time, is more accurate, and removes the need for additional personnel in medical decision making. Many nutrition surveys indicate that about a quarter of the world's population is anaemic. As a result, there is a pressing need to create an effective machine learning regressor capable of properly detecting anaemia.

The goal is to find out which individual classifier or group of classifier combinations obtain the highest accuracy in Red blood cell categorization for anaemia detection. We used Lasso and Ridge regressions to detect and estimate the anaemia. However, the classifier Ridge performs better achieves an accuracy higher than the Lasso regression.

Hence to achieve maximum accuracy in medical decision making, a better and powerful algorithm should be used. The outcomes of this algorithms decides whether the patient is infected with anaemia or not. The proposed version generates a better response to the inputs to confirm the disease.

KEYWORDS: Machine learning, Lasso, Ridge, detection.

INDEX

1. INTRODUCTION

1.1 FEASABILITY STUDY

1.2 EXISTING SYSTEM

1.3 PROPOSED SYSTEM

2. SOFTWARE REQUIREMENT SPECIFICATION

2.1 INTRODUCTION

2.2 EXTERNAL INTERFACE REQUIREMNET

2.3 SYSTEM FEATURES

2.4 OTHER NON-FUNCTIONAL REQUIREMENTS

3. ANALYSIS

3.1 USE CASE DIAGRAM

3.2 SEQUENCE DIAGRAM

3.3 COLLABORATION DIAGRAM

3.4 STATE CHART DIAGRAM

3.5 ACTIVITY DIAGRAM

4. DESIGN

4.1 ARCHITECTURE

4.2 PROJECT FLOW

4.3 DATABASE DESGIN/MODEL

4.4 CLASS DIAGRAM

4.5 COMPONENT DIAGRAM

4.6 DEPLOYEMENT DIAGRAM

5. IMPLEMENTATION

5.1 INTRODUCTION TO TECHNOLOGY

5.2 SAMPLE CODE

5.3 SCREENSHOTS

6. TESTING

6.1 INTRODUCTION TO TESTING

6.2 SAMPLE TEST CASES

7. CONCLUSION

8. FUTURE ENHANCEMENT

9. REFERENCES

1. INTRODUCTION

The computer has automated nearly every area of life, including medical, and disease diagnosis is no exception. Computer-aided illness diagnostic technologies make disease detection easier, quicker, and less expensive. They save time and effort in the decision-making process. To add to that, the automated diagnosis method is less susceptible to variability in physician choices. Another area where computers may assist is in the diagnosis and prediction of anaemia. Anaemia is a significant illness that mostly affects women and children, and it is estimated that anaemia affects nearly every second kid under the age of five. A better model is utilised by the majority of clinical decision-making tools. But one should check whether the mower is giving or resulting the better accuracy in decision making or not.

Anemia is defined as the decrease in amount of red blood cells (RBCs) or haemoglobin in the blood that has significant adverse health consequences, as well as adverse impacts on economic and social development. Although the most reliable indicator of anemia is blood hemoglobin concentration, there are a number of factors that can cause anemia such as iron deficiency, chronic infections such as HIV, malaria, and tuberculosis, vitamin deficiencies, e.g., vitamins B12 and A, cancer, and acquired disorders that affect red blood cell production and hemoglobin synthesis.

Anemia causes fatigue and low productivity and, when it occurs in pregnancy, may be associated with increased risk of maternal and perinatal mortality. According to World Health Organization (WHO), maternal and neonatal mortality were responsible for 3.0 million deaths in 2013 in developing countries. Anemia disease prediction plays a most important role in order to detect other associated diseases. Anemia disease is classified on the basis of morphology or on the basis of its underlying cause. Based on the morphology, anemia is divided into three types, which are normocytic, microcytic and macrocytic. Based on cause, anemia is classified into three types, namely blood loss, inadequate production of normal blood and excessive destruction of blood cells.

Here are four main tests that are ordered to diagnose anemia disorder which are complete blood count (CBC), ferritin, PCR (polymerase chain reaction) and hemoglobin electrophoresis. CBC test is the most frequently blood test to measure overall health and determine a wide range of diseases including anemia, infection and leukemia. A complete blood count test measures almost 15 parameters including: hemoglobin (Hb), red blood cells (RBC), hematocrit (HCT), mean corpuscular hemoglobin (MCH), mean corpuscular volume (MCV), and so on . 466 M.

Jaiswal et al. A ferritin test measures the amount of iron store in the body. High levels of ferritin indicate an iron storage disorder, such as hemochromatosis. Low levels of ferritin indicate iron deficiency, which causes anemia. PCR test is a molecular test, which is used to diagnose genetic disorder. A hemoglobin electrophoresis test is a blood test used to measure and identify the different types of hemoglobin in the bloodstream.

We have used two algorithms, namely Lasso Regression and, Ridge regression algorithm. We depict the flowchart of the proposed method.as defined.

1.1 FEASIBILITY STUDY

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

1. ECONOMICAL FEASIBILITY
2. TECHNICAL FEASIBILITY
3. OPERATIONAL FEASIBILITY

1.1.1 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

1.1.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead

to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

1.1.3 OPERATIONAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

1.2 EXISTING SYSTEM

The increasing growth of machine learning, computer techniques divided into traditional methods and machine learning methods. This section describes the related works of anaemia detection and how machine learning methods are better than traditional methods. The existing method in this project have a certain flow and also KNN is used for model development. But it requires large memory and result is not accurate.

Disadvantages:

- Low Accuracy
- High complexity.
- Highly inefficient.
- Requires skilled persons

1.3 PROPOSED SYSTEM

We propose this application that can be considered a useful system since it helps to reduce the limitations obtained from traditional and other existing methods. The objective of this study to develop fast and reliable method which detects and estimates anaemia accurately. To design this system is we used a powerful algorithm in a based Python environment with flask frame work

Advantages:

- Accuracy is good.
- Low complexity.
- Highly efficient.
- No need of skilled persons

2. SOFTWARE REQUIREMENT SPECIFICATION

2.1 INTRODUCTION

Software Requirement Specification (SRS) Format as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-functional depending upon type of requirement.

Software requirement specification plays an important role in creating quality software solutions. Specification is basically a representation process. Requirements are represented in a manner that ultimately leads to successful software implementation.

There are a set of guidelines to be followed while preparing the software requirement specification document. This includes the purpose, scope, functional and non-functional requirements, software and hardware requirements of the project. In addition to this, it also contains the information about environmental conditions required, safety and security requirements, software quality attributes of the project etc.

2.1.1 PURPOSE OF THE DOCUMENT

The purpose of the document is to collect and analysis all assorted ideas that have come up to define the system, its requirements with respect to consumers. In short, the purpose of this SRS document is to provide a detailed overview of our software product, its parameters and goals. This document describes the project's target audience and its user interface, hardware and software requirements. It defines how our client, team and audience see the product and its functionality. This document is meant to delineate the features of the system, so as to serve as a guide to the developers on one hand and a software validation document for the prospective client on the other

2.1.2 SCOPE OF THE DOCUMENT

The constraints of this software development effort should include:

- Scheduling enough time to complete the requirements laid-out by the client.
- Completing the necessary research to obtain the required skillsets to finish the project.

2.1.3 OVERVIEW OF THE DOCUMENT

The main objective of this project is to create an Effective Detection system for Anemia among individuals and taking necessary treatment in order to get rid fr anemia.

Using this the website the user can provide the specified details and know whether he/she has anemia or not.

2.2 EXTERNAL INTERFACE REQUIREMENTS

2.2.1 USER INTERFACE

- Frontend - HTML,CSS
- Backend - Python

2.2.2 HARDWARE INTERFACES

- Processor - I3/Intel Processor
- RAM - 8GB (min)
- Hard Disk - 128GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - Any

2.2.3 SOFTWARE INTERFACES

- Operating System - Windows 10
- Server-side Script - Python 3.6
- IDE - PyCharm
- Libraries Used - Pandas, NumPy, Scikit-Learn.
- Frame Work - Flask
- Data Base - MySql

2.3 SYSTEM FEATURES

Requirement's analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and non-functional requirements.

2.3.1 Functional Requirements:

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

1.User

Registration:

- Every new user needs to register themselves in the system with a unique name and email.
- The user will enters the details in the registration form according to the required fields. The fields include
- Name
- Password
- Confirm password
- Email
- Mobile number

Login:

Registered users can log in to the system. If it is a successful login the user will be directed to the main home page. Else if the user enters invalid information he will be asked to check the entered information.

Upload and View Data:

After logging into the website successfully, the user can upload the dataset and view their uploaded dataset.

Splits and Select Model:

The user can select the desired split ratio and can choose the required model to view the result from the dataset.

Prediction:

The user can predict outcomes i.e the person is affected with anemia or not from the system by providing the required input.

Logout:

User logout from the system. If the user wants to end his session and sign out of the website then he can use the logout option.

2.System

Take Dataset:

The System receives data given by the user i.e the dataset uploaded by the user.

Split:

Splits the dataset into the ratio input which is given by the user**.**

Model Training:

The user can select model based on the variables to get response from the data and can predict the outputs from the model. The user can either select Lasso Model or Ridge Model.

Generate Results:

The system can deliver the predicted results and can be displayed to the user.

2.4 OTHER NON-FUNCTIONAL REQUIREMENTS

2.4.1 Non-functional requirements:

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Performance:

The proposed system works harmoniously for different patients and for the accuracy of result for anemia detection.

Security:

The system's security shall be ensured by admin. The system's security shall be ensured by admin, the administrator shall be able to login and modify the content on the various web pages.

Usability:

How easily the system supports the interactions of user with input and output of the applications**.** The website is user-friendly which makes the user easier to use by its features.

3. ANALYSIS

UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

The Primary goals in the design of the UML are as follows:

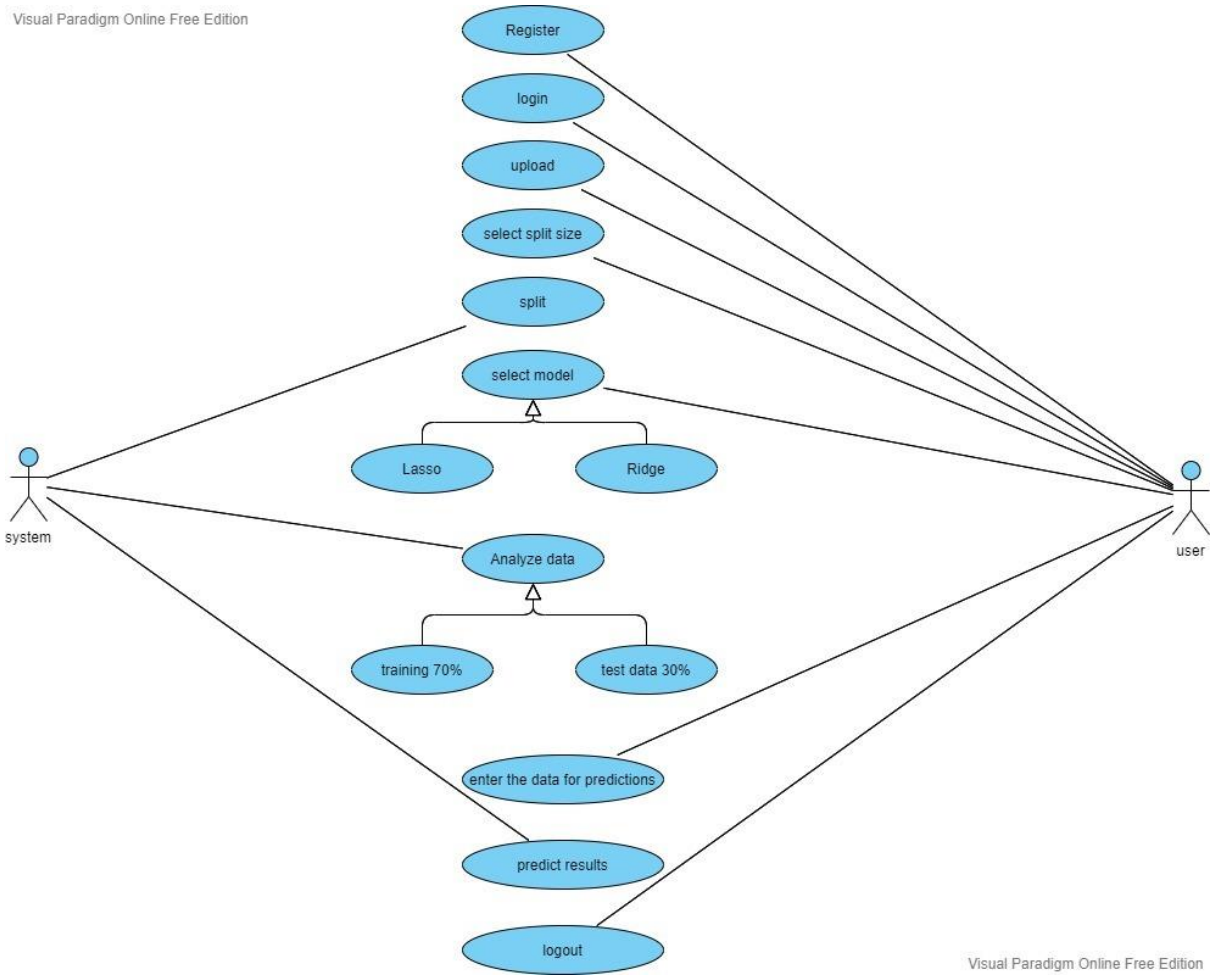
Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.

1. Provide extensibility and specialization mechanisms to extend the core concepts.
2. Be independent of particular programming languages and development process.
3. Provide a formal basis for understanding the modelling language.
4. Encourage the growth of OO tools market.
5. Support higher level development concepts such as collaborations, frameworks, patterns and components.
6. Integrate best practices.

3.1 USECASE DIAGRAM

To model a system the most important aspect is to capture the dynamic behaviour. Dynamic behaviour means the behaviour of the system when it is running. Only static behaviour is not sufficient to model a system rather dynamic behaviour is more important than static behaviour.

In UML there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we have to discuss the use case diagram is dynamic in nature, there should some internal or external factors for making the interaction. The internal and external agents are known as actors. use case diagram consists of actors, use case and their relationships. The diagram is used to model the system of an application. A single use case diagram captures a particular functionality of a system .

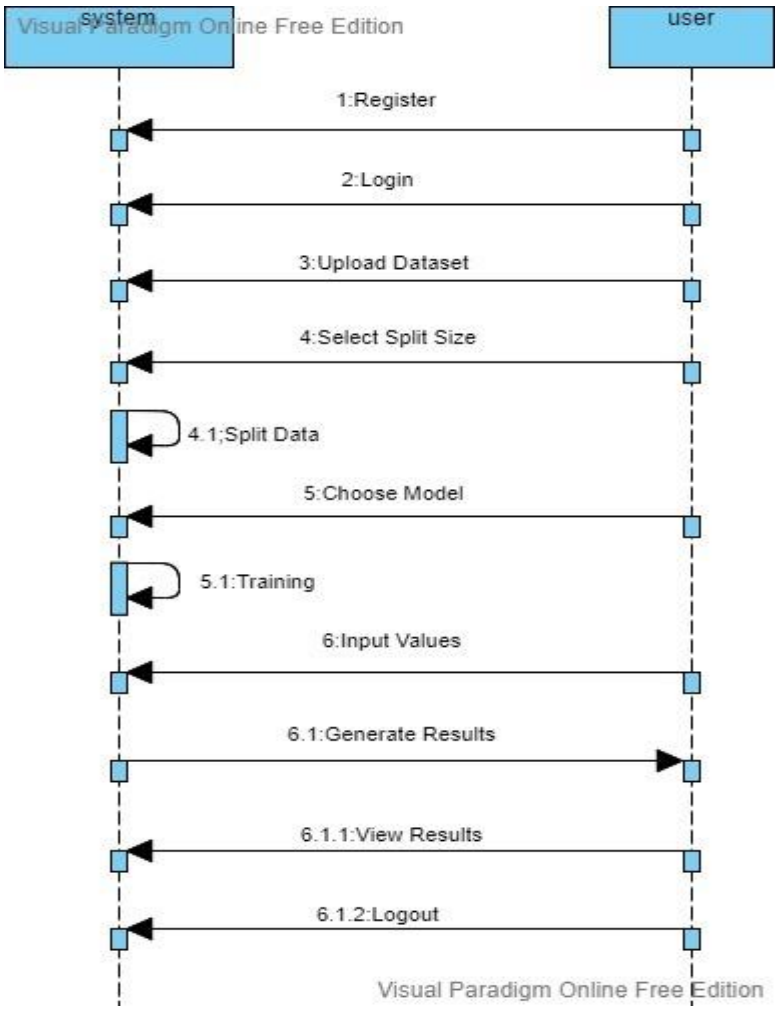


In the above Usecase Diagram system and user are the Actors and Register, Login, Upload, Selecting split size, selecting the model, Analyze data, enter the data for predictions, predicting results, logout are the usecases.

3.2 SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order.

It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

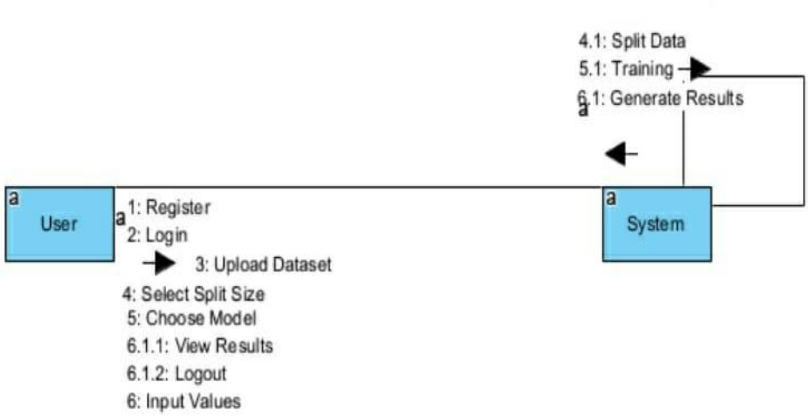


In Sequence Diagram it has objects like System and user. The interactions between these classes are register, login, upload dataset etc. That is Interaction between the system and the user.

If the user is logged in successfully it goes to the next step upload dataset else it will not go to the next step.

3.3 COLLABORATION DIAGRAM

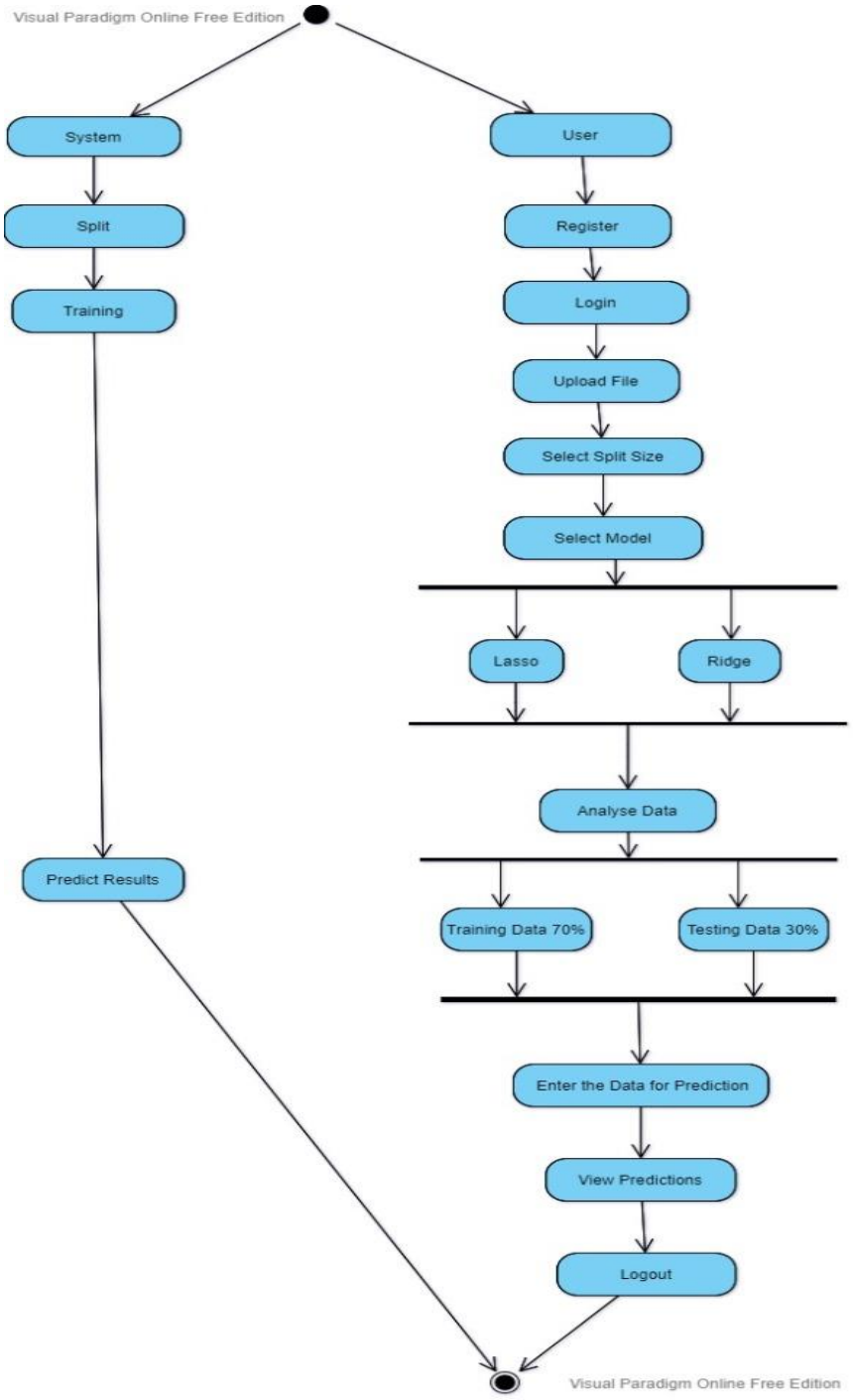
In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.



After uploading the dataset into system, it requires some input parameters which are split size, machine learning model for analyzing the uploaded data. This communication should be transferred between the user and system and represent it has a organization system.

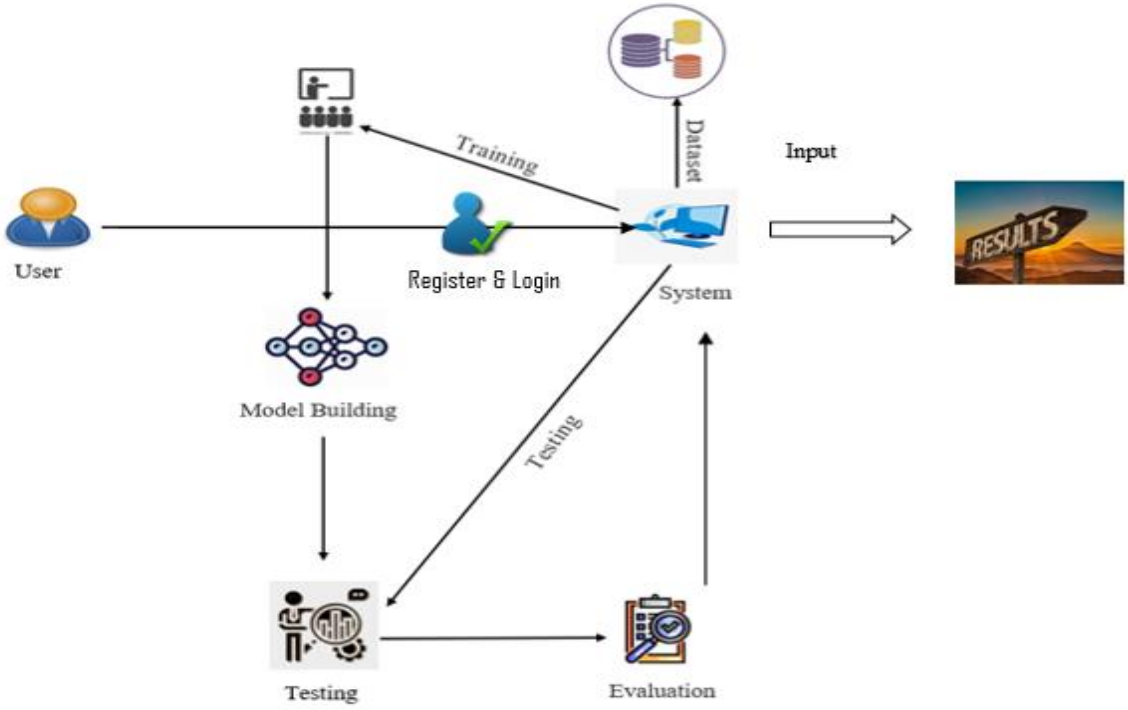
3.4 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



4. DESIGN

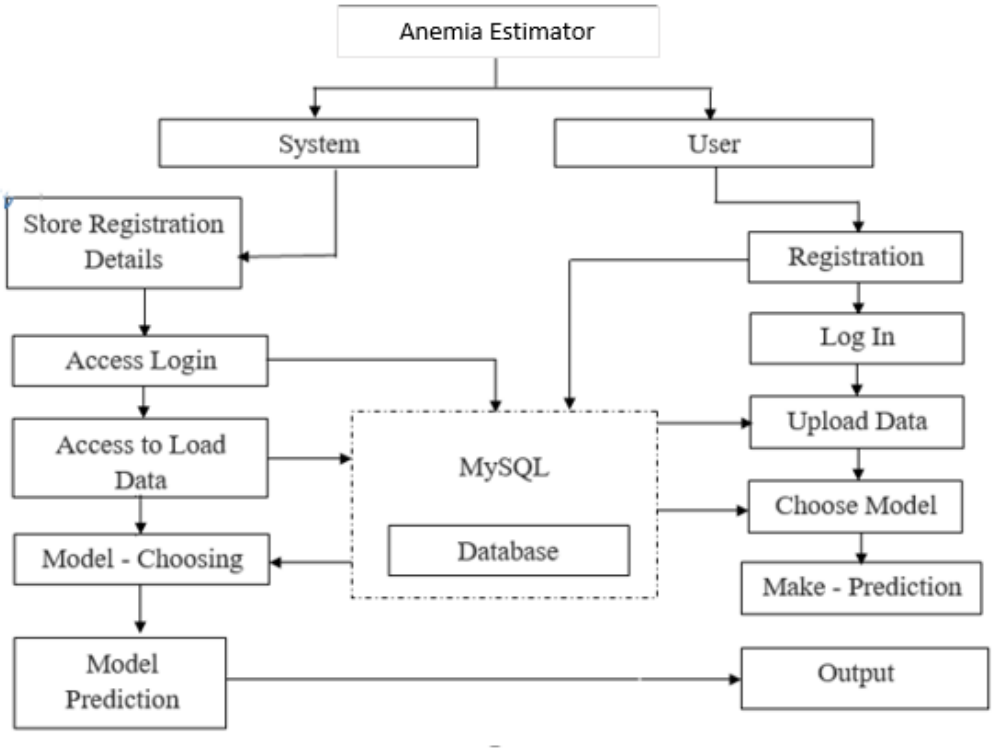
4.1 ARCHITECTURE



In User registration, every new user needs to register themselves in the system with a unique name and e-mail. After registration the registered users can log into the system. After logging in successfully, the user can upload and view their dataset. For splitting and select model, the user can select the desired split ratio and can choose the required model to view the result from the dataset. After splits and select model, the user can predict outcomes from the system which is the prediction step. Final step of the user will be logout from the system. This is the process of the user.

Here the process of the system is it will take dataset, splits the dataset and model training will be done to generate results. The system receives data given by the user. Then it splits the dataset into the ratio input given by the user. For model training the user can select model based on the variables to get response from the data and can predict the outputs from the model. To generate results, the system can deliver the predicted results and can be displayed to the user.

4.2 PROJECT FLOW



4.4 CLASS DIAGRAM

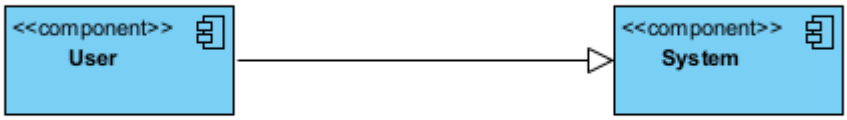
Class diagram is a static diagram. It represents the static view of an application .Class diagram is not only used for visualizing ,describing and documenting different aspects of a system but also for constructing executable code of the software application. Class diagram describe the attributes and operations of a class and also the constraints imposed other system. The class diagram is widely used in the modeling of object 18 oriented systems because they are the UML diagrams, which can be mapped directly with object oriented languages. Class diagram shows a collection of classes, interfaces, association, collaboration and constraints.it is also known as a structural diagram.

In the above class diagram system and user are the classes. And in each one has a collection of Objects. Each object has Some of Attributes, Methods and set of Behaviors.
In User Class Registration and Login are the attributes and upload Dataset, view data, select split size etc are the operations.

4.5 COMPONENT DIAGRAM

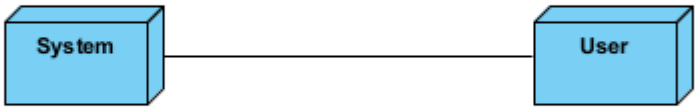
A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical **c**omponents in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required function is covered by planned development.

A component diagram is used to illustrate how components of a system interact with each other.



4.6 DEPLOYMENT DIAGRAM

The term deployment itself describes the purpose of diagrams used for describing hardware components, where software components are deployed. Component diagram and deployment diagram are closely related. Components diagrams are used to describe the components and deployment diagrams show how they are deployed in hardware. UML is mainly designed to focus on the software artifacts of an system. However these two diagrams are special diagrams used to focus on software and hardware components



5. IMPLEMENTATION

5.1 INTRODUCTION TO TECHNOLOGY

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

It is a platform independent programming language used to secure and robust applications that may run on any computer.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a general-purpose programming language, so it can be used for many things. Python is used for web development, AI, machine learning, operating systems, mobile application development, and video games. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. ... Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

History of Python

Python was developed by Guido van Possum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small talk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Possum still holds a vital role in directing its progress.

Features of Python:

1. Python is Interpreted:

Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

2. Python is Interactive:

You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

3. Python is Object Oriented:

Python supports Object-Oriented style or technique of programming that encapsulates code within objects

4. Python is a beginner's Language:

Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games

5. Expressive Language:

Python can perform complex tasks using a few lines of code. A simple example, the hello world program where you simply type print("Hello World"). It will take only one line to execute, while Java or C takes multiple lines.

6. Cross Platform Language:

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

7. GUI Programming support:

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

8. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to x, then we don't need to write int x = 15. Just write x = 15

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java

DYNAMIC VS. STATIC

- Types Python is a dynamic-typed language. Many other languages are static typed, such as C/C++ and Java. A static typed language requires the programmer to explicitly tell the computer what type of “thing” each data value is.
- For example, in C if you had a variable that was to contain the price of something, you would have to declare the variable as a “float” type.
- This tells the compiler that the only data that can be used for that variable must be a floating point number, i.e. a number with a decimal point.
- If any other data value was assigned to that variable, the compiler would give an error when trying to compile the program.

- Python, however, doesn’t require this. You simply give your variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects your program is using. This also means that you can change the size of the values as you develop the program. Say you have another decimal number (a.k.a. a floating point number) you need in your program.
- With a static typed language, you have to decide the memory size the variable can take when you first initialize that variable. A double is a floating point value that can handle a much larger number than a normal float (the actual memory sizes depend on the operating environment).
- If you declare a variable to be a float but later on assign a value that is too big to it, your program will fail; you will have to go back and change that variable to be a double.
- With Python, it doesn’t matter. You simply give it whatever number you want and Python will take care of manipulating it as needed. It even works for derived values.
- For example, say you are dividing two numbers. One is a floating point number and one is an integer. Python realizes that it’s more accurate to keep track of decimals so it automatically calculates the result as a floating point number.

VARIABLES

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

STANDARD DATA TYPES

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

- Numbers

- String
- List
- Tuple
- Dictionary

PYTHON NUMBERS

Number data types store numeric values. Number objects are created when you assign a value to them

PYTHON STRINGS

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

PYTHON LISTS

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

PYTHON TUPLES

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

PYTHON DICTIONARY

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

DIFFERENT MODES IN PYTHON

Python has two basic modes: normal and interactive.

The normal mode is the mode where the scripted and finished .pie files are run in the Python interpreter.

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole

20 PYTHON LIBRARIES

1. Requests: The most famous http library written by Kenneth remits. It's a must have for every python developer.

2. Scrappy: If you are involved in web scraping then this is a must have library for you. After using this library, you won't use any other.

3. Python: A guy toolkit for python. I have primarily used it in place of tinder. You will really love it.

4. Pillow: A friendly fork of PIL (Python Imaging Library). It is more user friendly than PIL and is a must have for anyone who works with images.

5. SQL Alchemy: A database library. Many love it and many hate it. The choice is yours.

6. Beautiful Soup: I know it's slow but this xml and html parsing library is very useful for beginners.

7. Twisted: The most important tool for any network application developer. It has a very beautiful ape and is used by a lot of famous python developers.

8. Numbly: How can we leave this very important library? It provides some advance math functionalities to python.

9. Skippy: When we talk about numbly then we have to talk about spicy. It is a library of algorithms and mathematical tools for python and has caused many scientists to switch from ruby to python.

10. Matplotlib: A numerical plotting library. It is very useful for any data scientist or any data analyser.

11. Pygmy: Which developer does not like to play games and develop them? This library will help you achieve your goal of 2d game development.

12. Piglet: A 3d animation and game creation engine. This is the engine in which the famous python port of mine craft was made

13. Pit: A GUI toolkit for python. It is my second choice after python for developing GUI's for my python scripts.

14. Pit: Another python GUI library. It is the same library in which the famous Bit torrent client is created.

15. Scaly: A packet sniffer and analyser for python made in python.

16. Pywin32: A python library which provides some useful methods and classes for interacting with windows.

17. Notch: Natural Language Toolkit – I realize most people won't be using this one, but it's generic enough. It is a very useful library if you want to manipulate strings. But its capacity is beyond that. Do check it out.

18. Nose: A testing framework for python. It is used by millions of python developers. It is a must have if you do test driven development.

19. Simply: Simply can do algebraic evaluation, differentiation, expansion, complex numbers, etc. It is contained in a pure Python distribution.

20. I Python: I just can't stress enough how useful this tool is. It is a python prompt on steroids. It has completion, history, shell capabilities, and a lot more. Make sure that you take a look at it.

PYTHON CLASS AND OBJECTS

These are the building blocks of OOP. Class creates a new object. This object can be anything, whether an abstract data concept or a model of a physical object, e.g. a chair. Each class has individual characteristics unique to that class, including variables and methods. Classes are

very powerful and currently “the big thing” in most programming languages. Hence, there are several chapters dedicated to OOP later in the book.

The class is the most basic component of object-oriented programming. Previously, you learned how to use functions to make your program do something.

Now will move into the big, scary world of Object-Oriented Programming (OOP). To be honest, it took me several months to get a handle on objects.

When I first learned C and C++, I did great; functions just made sense for me.

Having messed around with BASIC in the early ’90s, I realized functions were just like subroutines so there wasn’t much new to learn.

However, when my C++ course started talking about objects, classes, and all the new features of OOP, my grades definitely suffered.

Once you learn OOP, you’ll realize that it’s actually a pretty powerful tool. Plus many Python libraries and APIs use classes, so you should at least be able to understand what the code is doing.

One thing to note about Python and OOP: it’s not mandatory to use objects in your code in a way that works best; maybe you don’t need to have a full-blown class with initialization code and methods to just return a calculation. With Python, you can get as technical as you want.

As you’ve already seen, Python can do just fine with functions. Unlike languages such as Java, you aren’t tied down to a single way of doing things; you can mix functions and classes as necessary in the same program. This lets you build the code

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Here’s a brief list of Python OOP ideas:

The class statement creates a class object and gives it a name. This creates a new namespace.

Assignments within the class create class attributes. These attributes are accessed by qualifying the name using dot syntax: ClassName.Attribute.

Class attributes export the state of an object and its associated behavior. These attributes are shared by all instances of a class.

INHERITANCE

First off, classes allow you to modify a program without really making changes to it.

To elaborate, by sub classing a class, you can change the behaviour of the program by simply adding new components to it rather than rewriting the existing components.

As we've seen, an instance of a class inherits the attributes of that class.

However, classes can also inherit attributes from other classes. Hence, a subclass inherits from a super class allowing you to make a generic super class that is specialized via sub classes.

The sub classes can override the logic in a super class, allowing you to change the behavior of your classes without changing the super class at all.

EXCEPTIONS

They are special events that can occur due to an error, e.g. trying to open a file that doesn't exist, or when the program reaches a marker, such as the completion of a loop.

Exceptions, by definition, don't occur very often; hence, they are the "exception to the rule" and a special class has been created for them. Exceptions are everywhere in Python.

Virtually every module in the standard Python library uses them, and Python itself will raise them in a lot of different circumstances.

Here are just a few examples:

- Accessing a non−existent dictionary key will raise a Key Error exception.
- Searching a list for a non−existent value will raise a Value Error exception
- Calling a non−existent method will raise an Attribute Error exception.
- Referencing a non−existent variable will raise a Name Error exception.

PYTHON MODULES

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a *module*; definitions from a module can be *imported* into other modules or into the *main* module.

FUNCTIONS IN PYTHON

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

To carry out that specific task, the function might or might not need multiple inputs. When the task is carved out, the function can or cannot return one or more values.

5.2 SAMPLE CODE

```
import mysql
from flask import Flask,render_template,url_for,request
from mysql.connector import cursor
```

```
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score
import pandas as pd

mydb =
mysql.connector.connect(host='localhost',user='root',password="",port='3306',database='ane
mia')
app=Flask(__name__)

def preprocessing(file):
    file.drop(index=df.index[0], axis=0, inplace=True)
    file.rename(columns={'Sex  ':'Sex','Age      ': 'Age', '  RBC    ': 'RBC', 'MCV  ': 'MCV', '
MCHC  ': 'MCHC', ' RDW    ': 'RDW',
                        ' PLT /mm3': 'PLT', ' HGB ': 'HGB'}, inplace=True)
    return file



@app.route('/')
def index():
    return render_template('index.html')

@app.route('/registration',methods=['POST','GET'])
def registration():
    if request.method=="POST":
        print('a')
        un=request.form['name']
        print(un)
        em=request.form['email']
        pw=request.form['password']
        print(pw)
```

```
        cpw=request.form['cpassword']
        if pw==cpw:
            sql = "SELECT * FROM hmg"
            cur = mydb.cursor()
            cur.execute(sql)
            all_emails=cur.fetchall()
            mydb.commit()
            all_emails=[i[2] for i in all_emails]
            if em in all_emails:
                return render_template('registration.html',msg='a')
            else:
                sql="INSERT INTO hmg(name,email,password) values(%s,%s,%s)"
                values=(un,em,pw)
                cursor=mydb.cursor()
                cur.execute(sql,values)
                mydb.commit()
                cur.close()
                return render_template('registration.html',msg='success')
        else:
            return render_template('registration.html',msg='repeat')
    return render_template('registration.html')

@app.route('/login',methods=["POST","GET"])
def login():
    if request.method=="POST":
        em=request.form['email']
        print(em)
        pw=request.form['password']
        print(pw)
        cursor=mydb.cursor()
        sql = "SELECT * FROM hmg WHERE email=%s and password=%s"
        val=(em,pw)
        cursor.execute(sql,val)
        results=cursor.fetchall()
```

```
        mydb.commit()
        print(results)
        print(len(results))
        if len(results) >= 1:
            return render_template('home.html',msg='login succesful')
        else:
            return render_template('login.html',msg='Invalid Credentias')



    return render_template('login.html')

@app.route('/home')
def home():
    return render_template('home.html')

@app.route('/upload',methods=['POST','GET'])
def upload():
    global df
    if request.method=="POST":
        file=request.files['file']
        print(type(file.filename))
        print('hi')
        df=pd.read_csv(file)
        print(df.heapd(2))
        return render_template('upload.html', msg='Dataset Uploaded Successfully')
    return render_template('upload.html')

@app.route('/view_data')
def view_data():
    print(df)
    print(df.head(2))
    print(df.columns)
    return
render_template('viewdata.html',columns=df.columns.values,rows=df.values.tolist())
```

```
@app.route('/split',methods=["POST","GET"])
def split():
    global X,y,X_train,X_test,y_train,y_test
    if request.method=="POST":
        size=int(request.form['split'])
        size=size/100
        print(size)
        dataset=preprocessing(df)
        print(df)
        print(df.columns)
        dataset = dataset.loc[0:364]
        X=dataset.drop(['HGB','S. No.'],axis=1)
        y=dataset['HGB']
        X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=size,random_state=52)
        print(y_test)

        return render_template('split.html',msg='Data Preprocessed and It Splits Succesfully')
    return render_template('split.html')

@app.route('/model',methods=['POST','GET'])
def model():
    if request.method=="POST":
        model=int(request.form['algo'])
        if model==0:
            return render_template('model.html',msg='Please Choose any Algorithm')
        elif model==1:
            model=Lasso()
            model.fit(X_train,y_train)
            y_pred=model.predict(X_test)
            score=r2_score(y_test,y_pred).round(4)
            score=score*100
            msg='The R2 Score for Lasso is ', score
            return render_template('model.html',msg=msg)
```

```
        else:
            model = Ridge()
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            score = r2_score(y_test, y_pred).round(4)
            score = score * 100
            msg = 'The R2 Score for Ridge is ', score
        return render_template('model.html',msg=msg)
    return render_template('model.html')

@app.route('/prediction',methods=["POST","GET"])
def prediction():
    if request.method=="POST":
        val=request.form['d']
        # if f1=="male":
        #     val=0
        # else:
        #     val=1
        print(val)
        f2=request.form['age']
        f3=request.form['rbc']
        f4=request.form['pcv']
        f5=request.form['mcv']
        f6=request.form['mch']
        f7=request.form['mchc']
        f8=request.form['rdw']
        f9=request.form['tlc']
        f10=request.form['plt']
        print(f10)
        print(type(f10))
        l=[val,f2,f3,f4,f5,f6,f7,f8,f9,f10]
        model=Ridge()
        model.fit(X_train,y_train)
        ot=model.predict([l])
```

```
        print(ot)
        if ot<12:
            a='The Person is Diagnosed with Anemia'
        else:
            a="The Person is safe and not effected with Anemia"
        return render_template('prediction.html',msg=a)
    return render_template('prediction.html')

@app.route('/about')
def about():
    return render_template('about.html')



if __name__=="__main__":
    app.run(debug=True)
```

5.3 SCREENSHOTS

OUTPUT SCREEN SHOTS

Home:

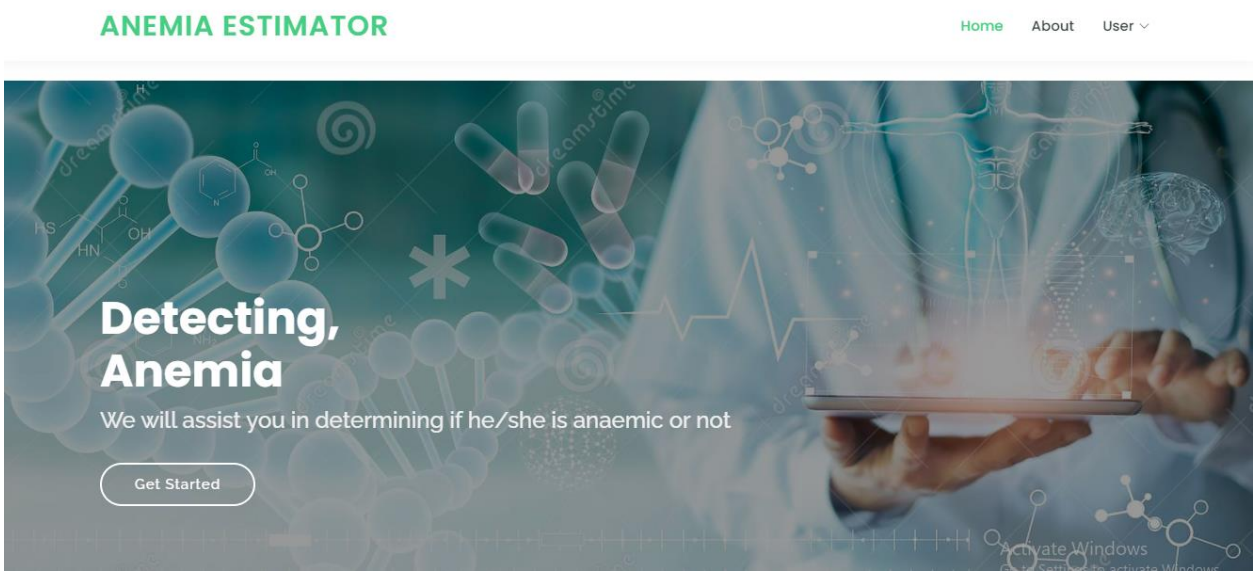


Fig1: Home Page

Registration:

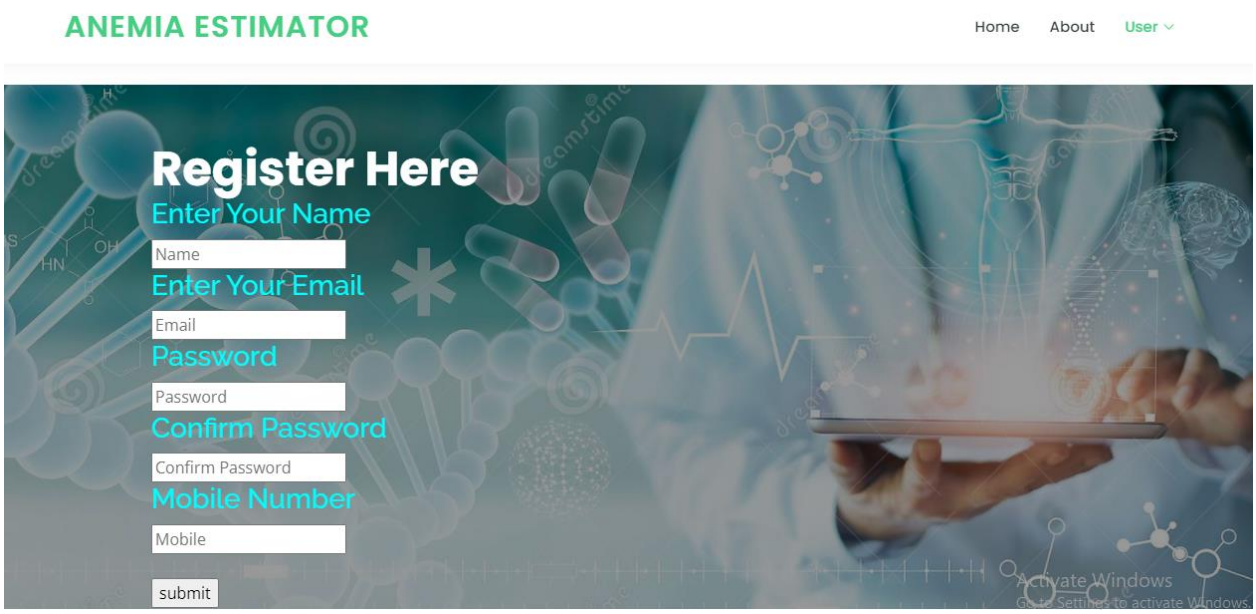


Fig2: Registration Page

LOGIN

Fig3: Login Page

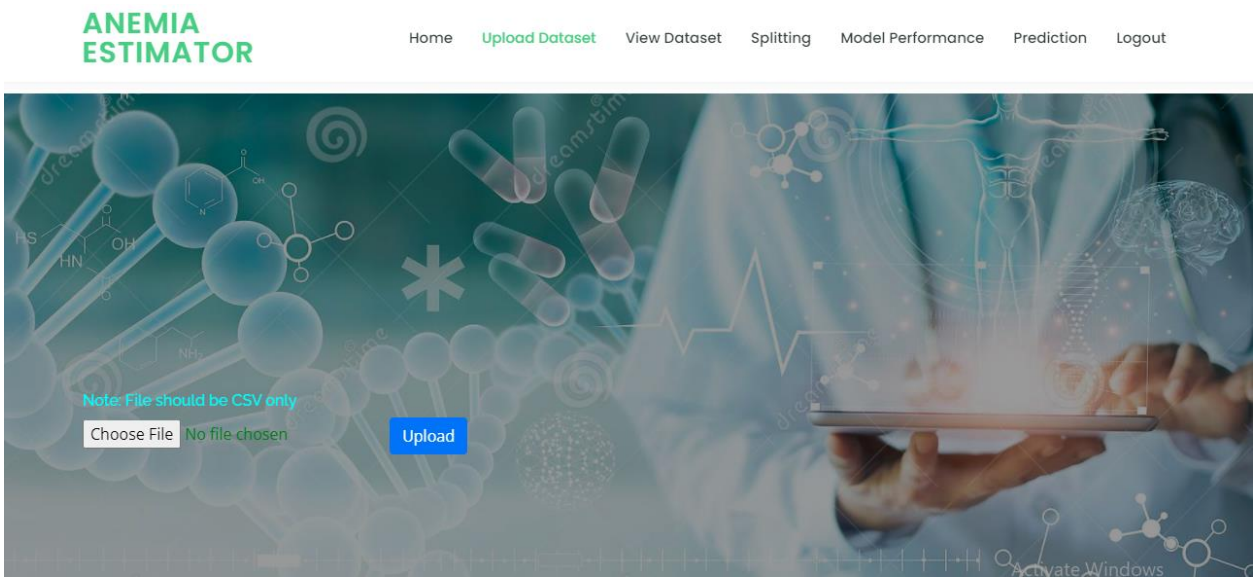
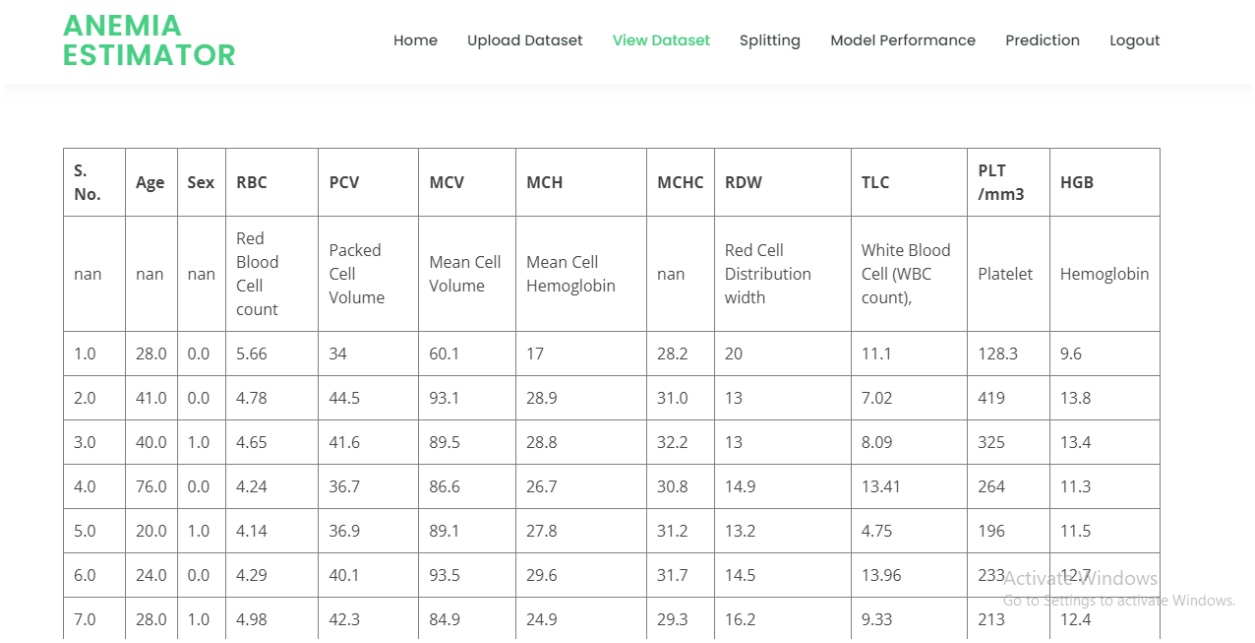


Fig4: Data Upload Page

VIEW DATA



S. No.	Age	Sex	RBC	PCV	MCV	MCH	MCHC	RDW	TLC	PLT /mm3	HGB
nan	nan	nan	Red Blood Cell count	Packed Cell Volume	Mean Cell Volume	Mean Cell Hemoglobin	nan	Red Cell Distribution width	White Blood Cell (WBC count),	Platelet	Hemoglobin
1.0	28.0	0.0	5.66	34	60.1	17	28.2	20	11.1	128.3	9.6
2.0	41.0	0.0	4.78	44.5	93.1	28.9	31.0	13	7.02	419	13.8
3.0	40.0	1.0	4.65	41.6	89.5	28.8	32.2	13	8.09	325	13.4
4.0	76.0	0.0	4.24	36.7	86.6	26.7	30.8	14.9	13.41	264	11.3
5.0	20.0	1.0	4.14	36.9	89.1	27.8	31.2	13.2	4.75	196	11.5
6.0	24.0	0.0	4.29	40.1	93.5	29.6	31.7	14.5	13.96	233	12.7
7.0	28.0	1.0	4.98	42.3	84.9	24.9	29.3	16.2	9.33	213	12.4

Fig5: View Dataset Page

SPLITS

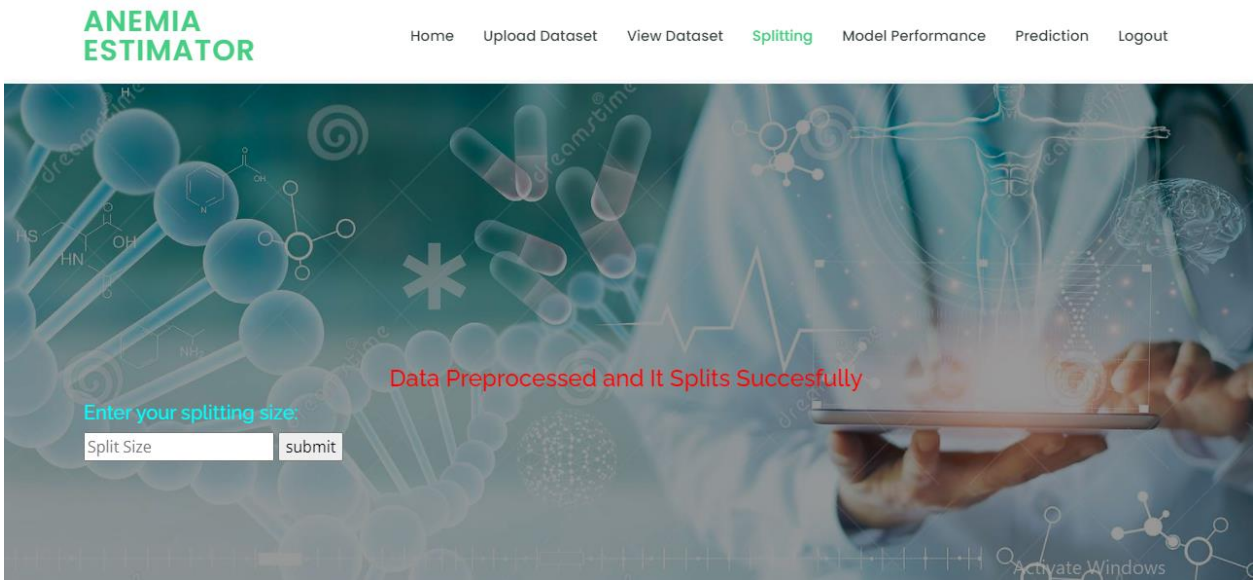


Fig6: Split Page

MODEL PERFORMANCE

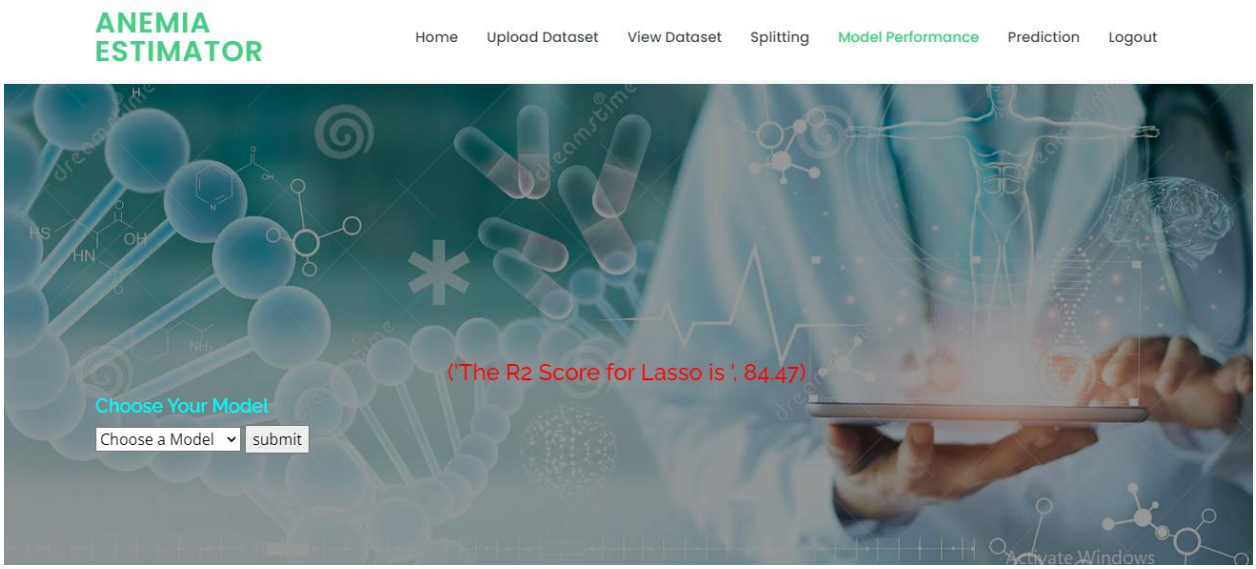


Fig7: Model Training

PREDICTION

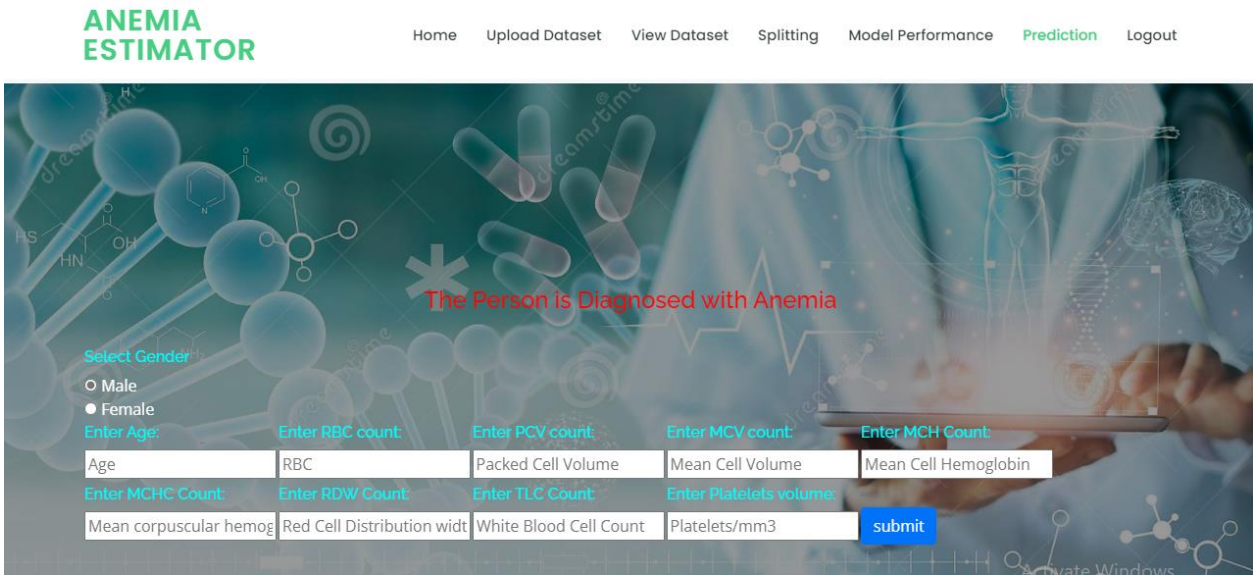


Fig8: Prediction Page

6.TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements

6.2 SAMPLE TEST CASES

S.NO	Test cases	Input Action Performance	Expected Output	Actual Output	P/F
1	Read the dataset.	Dataset.	Dataset need to read successfully.	Dataset fetched successfully.	Pass
2	Performing pre-	Pre-processing	Pre-processing should be	Pre-processing	Pass

	processing on the dataset	part takes place	performed on dataset	successfully completed.	
3	Model Building	Model Building for the clean data	Need to create model using required algorithms	Model Created Successfully.	Pass
4	Anaemia Estimation	Input fields provided.	Output should be whether anaemic or not	Model predicted as patient is infected with anaemia	Pass
5	Anaemia Estimation	Input fields provided.	Output should be whether anaemic or not	Model predicted as patient is not infected with anaemia	Pass

8.CONCLUSION

We have successfully developed a system to detect anaemia in this application. This is created in a user-friendly environment with Python programming and Flask. The system is likely to

gather data from the user in order to determine whether or not patient is infected with anaemia or not

.