



Cardiotocography Data Analysis for Fetal Health Classification Using Machine Learning Models

AIMAN FATIMA SOOFI

BACHELOR OF TECHNOLOGY

SHADAN COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution Accredited with NAAC A+ & NBA)

PEERANCHERU, HYDERABAD-500086

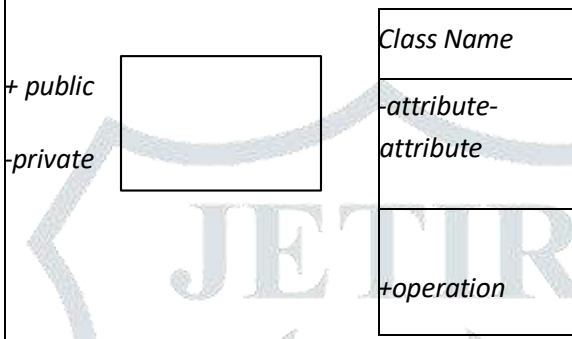
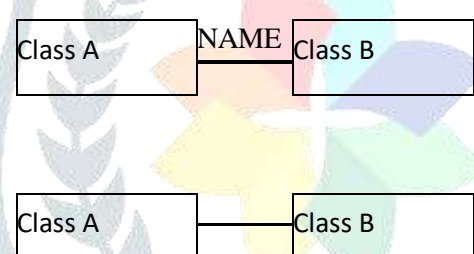
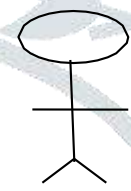
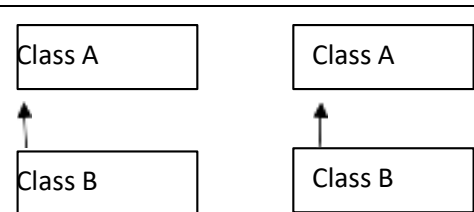
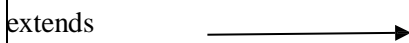
(Affiliated to JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD)






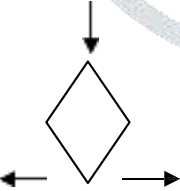

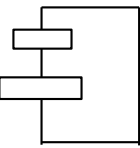
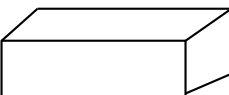
HYDERABAD-500087

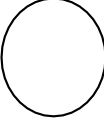




ABSTRACT

Pregnancy complications pose significant risks to maternal and fetal health, necessitating early detection for timely interventions. Manual analysis of cardiotocography (CTG) tests, the conventional practice among obstetricians, is labor-intensive and prone to variability. This study addresses the critical need for accurate fetal health classification using advanced machine learning (ML) techniques, focusing on the application of XGBoost, a powerful gradient boosting algorithm. Utilizing a publicly available dataset, despite its size, this research leverages its rich features to develop and analyze ML models. The objective is to explore and demonstrate the efficacy of ML models in classifying fetal health based on data. Our proposed system applies the XGBoost algorithm and achieves an exceptional accuracy of 96%, surpassing previous methods. This highlights the algorithm's robustness in enhancing diagnostic precision and facilitating timely interventions. The study underscores the potential of integrating ML models into routine clinical practices to streamline fetal health assessments. By optimizing resource allocation and improving time efficiency, these models contribute to early complication detection and enhanced prenatal care. Further research is encouraged to refine ML applications, promising continued advancements in fetal health assessment and maternal care.

LIST OF NOTATIONS

S.NO	NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Aor		It aggregates several classes into a single classe
4.	Aggregation		Interaction between the system and external environment
5.	Relation (uses)	Uses	Used for additional process communication.
6.	Relation (extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.

7.	Communication		Communication between various use cases.
8.	State	State 	State of the process.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Usecase	Uses case 	Interact ion between the system and external environment.
14.	Component		Represents physical modules which is a collection of components.
15.	Node		Represents physical modules which are a collection of components.

16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message		Represents the message exchanged.

CHAPTER-1

INTRODUCTION

1.1 GENERAL

1.Introduction :

Pregnancy is a critical period marked by physiological changes and potential complications that can affect both the mother and the fetus. Accurate monitoring and timely intervention are crucial for ensuring favorable maternal and fetal outcomes. Cardiotocography (CTG) is a widely used non-invasive diagnostic tool that monitors fetal heart rate (FHR) and uterine contractions, providing valuable insights into fetal well-being. However, the manual interpretation of CTG data by obstetricians is often labor-intensive and subject to inter-observer variability, which can lead to inconsistent diagnoses and delayed interventions. Recent advancements in machine learning (ML) have opened new avenues for automating and enhancing the accuracy of medical diagnostics. Machine learning models can analyze large volumes of data quickly and accurately, making them suitable for tasks such as fetal health classification. Among the various ML techniques, XGBoost (Extreme Gradient Boosting) has gained prominence due to its high performance, scalability, and robustness in handling complex datasets. This study focuses on leveraging the capabilities of XGBoost to classify fetal health based on CTG data. By utilizing a publicly available dataset rich in features, we aim to develop and validate an ML model that can assist in the early detection of potential complications during pregnancy. Our proposed system demonstrates a high classification accuracy, highlighting the potential of

integrating ML models into clinical practice to improve the efficiency and precision of fetal health assessments. The significance of this research lies in its potential to streamline prenatal care, optimize resource allocation, and enhance the overall quality of maternal and fetal health management. By reducing the reliance on manual interpretation and providing consistent, accurate diagnoses, ML models can facilitate timely interventions and improve pregnancy outcomes. This study not only underscores the importance of advanced ML techniques in medical diagnostics but also encourages further exploration and refinement of these technologies to support maternal and fetal health care.

1.2 SCOPE OF THE PROJECT

This project aims to revolutionize prenatal care by developing an advanced machine learning (ML) system for the classification of fetal health using cardiotocography (CTG) data. The core focus is on applying the XGBoost algorithm, renowned for its accuracy and robustness in predictive analytics. By leveraging a publicly available dataset with rich features, the project seeks to build a model that surpasses traditional diagnostic methods, achieving a notable accuracy of 96%. The system's primary goal is to enhance the precision of fetal health assessments, facilitating early detection of pregnancy complications and timely interventions. Additionally, the project explores the potential integration of ML models into clinical settings, aiming to streamline assessments, optimize resource allocation, and improve time efficiency in prenatal care. The research underscores the importance of continuous innovation in ML applications to further advance maternal and fetal health diagnostics.

1.3 OBJECTIVE

The objective of this project is to develop and validate an advanced machine learning-based system for accurate fetal health classification using cardiotocography (CTG) data. By leveraging the XGBoost algorithm, the project aims to surpass traditional manual analysis methods, enhancing diagnostic precision and efficiency. This system is intended to streamline prenatal care by providing obstetricians with a reliable tool for early complication detection, optimizing resource allocation, and ultimately improving maternal and fetal health outcomes.

1.4 PROBLEM STATEMENT

Fetal health monitoring is a critical aspect of prenatal care that helps ensure the well-being of both the mother and the unborn child. Cardiotocography (CTG) is a widely used non-invasive technique for monitoring fetal heart rate (FHR) and uterine contractions during pregnancy. However, interpreting CTG results manually can be subjective, time-consuming, and prone to human error, potentially leading to delayed or incorrect diagnoses.

The primary challenge lies in accurately classifying fetal health status (normal, suspect, or pathological) based on CTG data, which consists of complex and high-dimensional features. Traditional diagnostic methods may not effectively capture patterns within the data that indicate subtle signs of fetal distress.

This project aims to leverage machine learning models to analyze and classify fetal health conditions based on CTG data. By developing an automated and accurate classification system, the project seeks to assist healthcare professionals in making timely and reliable decisions, ultimately improving prenatal care and reducing the risk of adverse outcomes.

1.5 EXISTING SYSTEM

- Random Forest is a versatile and powerful ensemble learning method used for classification and regression tasks in machine learning. It operates by constructing multiple decision trees during training and outputs the mode (classification) or average prediction (regression) of the individual trees. It is known for its robustness and ability to handle complex datasets with high dimensionality.

- Random Forest is an ensemble learning technique that combines the predictions of multiple individual decision trees to improve the overall accuracy and robustness of the model. Each decision tree in the ensemble is trained independently on a random subset of the training data and a random subset of the features.

1.5.1 EXISTING SYSTEM DISADVANTAGES

- Complexity Random Forest models can be difficult to interpret compared to single decision trees.
- Computational Cost training a large number of trees can be computationally expensive and time-consuming.
- Memory Usage it consumes more memory due to the ensemble of multiple trees.

1.6 LITERATURE SURVEY

Title: So how do we balance all of these needs: How the concept of AI technology impacts digital archival expertise

Author: A. L. Cushing and G. Osti

Year: 2023.

Description

The term “Artificial Intelligence” (AI) is increasingly permeating public consciousness as it has gained more popularity in recent years, especially within the landscape of academia and libraries. AI in libraries has been a trending subject of interest for some time, as within the library there are numerous departments that serve a role in collectively contributing to the library’s mission. Consequently, it is imperative to consider AI’s influence on the digital preservation of historic documents. This paper delves into the historical evolution of preservation methods driven by technological advancements as, throughout history, libraries, archives, and museums have grappled with the challenge of preserving historical collections, while many of the traditional preservation methods are costly and involve a lot of manual (human) effort. AI being the catalyst for transformation could change this reality and perhaps redefine the process of preservation; thus, this paper explores the emerging trend of incorporating AI technology into preservation practices and provides predictions regarding the transformative role of Artificial Intelligence in preservation for the future. With that in mind, this paper addresses the following questions: could AI be what changes or creates a paradigm shift in how preservation is done?; and could it be the thing that will change the way history is safeguarded

Title: Integrated use of KOS and deep learning for data set annotation in tourism domain.

Author: G. Aracri, A. Folino, and S. Silvestri.

Year: 2023.

Description

Purpose This study aims to provide a systematic review of the existing literature on the applications of deep learning (DL) in hospitality, tourism and travel as well as an agenda for future research. **Design/methodology/approach** Covering a five-year time span (2017–2021), this study systematically reviews journal articles archived in four academic databases: Emerald Insight, Springer, Wiley Online Library and ScienceDirect. All 159 articles reviewed were characterised using six attributes: publisher, year of publication, country studied, type of value created, application area and future suggestions (and/or limitations). **Findings** Five application areas and six challenge areas are identified, which characterise the application of DL in hospitality, tourism and travel. In addition, it is observed that DL is mainly used to develop novel models that are creating business value by forecasting (or projecting) some

parameter(s) and promoting better offerings to tourists. Research limitations/implications Although a few prior papers have provided a literature review of artificial intelligence in tourism and hospitality, none have drilled-down to the specific area of DL applications within the context of hospitality, tourism and travel. Originality/value To the best of the authors' knowledge, this paper represents the first theoretical review of academic research on DL applications in hospitality, tourism and travel. An integrated framework is proposed to expose future research trajectories wherein scholars can contribute significant value. The exploration of the DL literature has significant implications for industry and practice, given that this, as far as the authors know, is the first systematic review of existing literature in this research area.

Title: Ship classification based on improved convolutional neural network architecture for intelligent transport systems.

Author: L. A. Leonidas and Y. Jie,

Year: 2021.

Description

In recent years, deep learning has been used in various applications including the classification of ship targets in inland waterways for enhancing intelligent transport systems. Various researchers introduced different classification algorithms, but they still face the problems of low accuracy and misclassification of other target objects. Hence, there is still a need to do more research on solving the above problems to prevent collisions in inland waterways. In this paper, we introduce a new convolutional neural network classification algorithm capable of classifying five classes of ships, including cargo, military, carrier, cruise and tanker ships, in inland waterways. The game of deep learning ship dataset, which is a public dataset originating from Kaggle, has been used for all experiments. Initially, the five pretrained models (which are AlexNet, VGG, Inception V3 ResNet and GoogleNet) were used on the dataset in order to select the best model based on its performance. Resnet-152 achieved the best model with an accuracy of 90.56%, and AlexNet achieved a lower accuracy of 63.42%. Furthermore, Resnet-152 was improved by adding a classification block which contained two fully connected layers, followed by ReLu for learning new characteristics of our training dataset and a dropout layer to resolve the problem of a diminishing gradient. For generalization, our proposed method was also tested on the MARVEL dataset, which consists of more than 10,000 images and 26 categories of ships. Furthermore, the proposed algorithm was compared with existing algorithms and obtained high performance compared with the others, with an accuracy of 95.8%, precision of 95.83%, recall of 95.80%, specificity of 95.07% and F1 score of 95.81%.

Title: Ship images detection and classification based on convolutional neural network with multiple feature regions.

Author: Z.Xu,J. Sun, and Y. Huo,

Year: 2022

Description

In recent years, the maritime industry is developing rapidly, which poses great challenges for intelligent ship navigation systems to achieve accurate ship classification. To cope with this problem, a Recurrent Attention Convolutional Neural Network (RA-CNN) is proposed, which is fused with multiple feature regions for ship classification. The proposed model has three scale layers, each of which contains a classification network VGG-19 and a localisation head Attention Proposal Network (APN). First, the Scale Dependent Pooling algorithm is integrated with VGG-19 to reduce the impact of over-pooling and improve the classification performance of small ships. Second, the APN incorporates the Joint Clustering algorithm to generate multiple independent feature regions; thus, the whole model can make full use of the global information in ship recognition. In the meantime, the Feature Regions

Optimisation method is designed to solve the overfitting problem and reduce the overlap rate of multiple feature regions. Finally, a novel loss function is defined to cross-train VGG-19 and APN, which accelerates the convergence process. The experimental results show that the classification accuracy of the authors' proposed method reaches 90.2%, which has a 6% improvement over the baseline RA-CNN. Both classification accuracy and robustness are improved by a large margin compared to those of other compared models.

Title: Ship classification based on convolutional neural networks.

Author: Y.Yang ,K.Ding, and Z.Chen

Year: 2022.

Description

In recent years, deep learning has been used in various applications including the classification of ship targets in inland waterways for enhancing intelligent transport systems. Various researchers introduced different classification algorithms, but they still face the problems of low accuracy and misclassification of other target objects. Hence, there is still a need to do more research on solving the above problems to prevent collisions in inland waterways. In this paper, we introduce a new convolutional neural network classification algorithm capable of classifying five classes of ships, including cargo, military, carrier, cruise and tanker ships, in inland waterways. The game of deep learning ship dataset, which is a public dataset originating from Kaggle, has been used for all experiments. Initially, the five pretrained models (which are AlexNet, VGG, Inception V3 ResNet and GoogleNet) were used on the dataset in order to select the best model based on its performance. Resnet-152 achieved the best model with an accuracy of 90.56%, and AlexNet achieved a lower accuracy of 63.42%. Furthermore, Resnet-152 was improved by adding a classification block which contained two fully connected layers, followed by ReLu for learning new characteristics of our training dataset and a dropout layer to resolve the problem of a diminishing gradient.

1.7 PROPOSED SYSTEM

- XGBoost, short for Extreme Gradient Boosting, is a scalable and efficient machine learning algorithm known for its speed and performance in supervised learning tasks. It belongs to the family of gradient boosting algorithms and has gained popularity for its ability to handle diverse data types and produce state-of-the-art results in various machine learning competitions.
- XGBoost combines the strengths of gradient boosting algorithms with several enhancements that make it robust and powerful. It uses a gradient descent optimization technique to minimize the loss when adding new models to the ensemble, thereby improving model accuracy iteratively.

1.7.1 PROPOSED SYSTEM ADVANTAGES

- Performance XGBoost is known for its high performance and computational efficiency, making it suitable for large datasets and complex problems.
- Accuracy It often achieves state-of-the-art results in machine learning competitions and benchmarks due to its powerful optimization techniques.
- Flexibility It supports various types of data inputs and can be customized with different objective functions and evaluation metrics.

CHAPTER 2

PROJECT DESCRIPTION

2.1 GENERAL

This project aims to enhance the accuracy and efficiency of fetal health classification by leveraging advanced machine learning techniques, specifically focusing on the XGBoost algorithm. Traditional manual analysis of cardiotocography (CTG) tests, used to monitor fetal health, is often labor-intensive and subject to variability. By utilizing a publicly available dataset with rich features, this study demonstrates the superiority of the XGBoost algorithm, achieving an impressive accuracy rate of 96%, which surpasses previous methods. The successful application of XGBoost underscores its robustness and potential to significantly improve diagnostic precision, streamline fetal health assessments, and facilitate timely medical interventions. Integrating such machine learning models into routine clinical practices can optimize resource allocation, enhance time efficiency, and contribute to early detection of complications, ultimately improving prenatal care. Further research is encouraged to continue refining these applications, promising ongoing advancements in maternal and fetal health assessments.

2.2 METHODOLOGIES

2.2.1 MODULES

This project has the following 8 modules

- Dataset
- Import the libraries
- Analyzing
- Preprocessing
- Splitting The data
- Model
- Accuracy
- Result

2.2.2 MODULES DESCRIPTION

1. Dataset Design

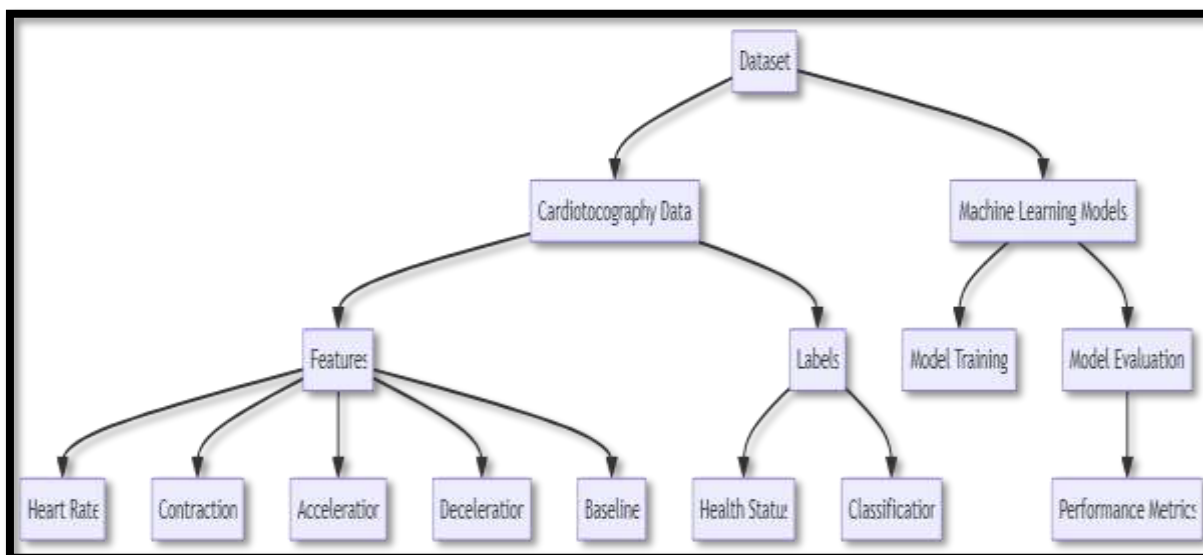


FIGURE 2.1: DATASET DESIGN

The dataset utilized in this study is a comprehensive collection of maternal and fetal health records, specifically focused on pregnancies complicated by abnormal cardiotocography (CTG) readings. This dataset includes various parameters and measurements that are crucial for analyzing and predicting pregnancy outcomes.

2. Import the libraries Design

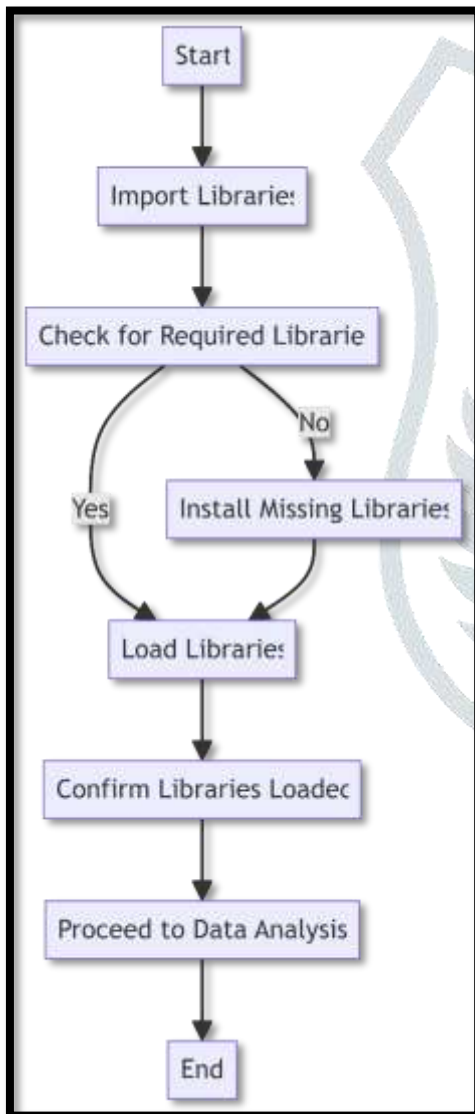


FIGURE 2.2: IMPORT THE LIBRARIES DESIGN

To import the necessary libraries for preprocessing, data splitting, and applying XGBoost. `sklearn.model_selection.train_test_split`: This function splits your data into training and testing sets, which is crucial for evaluating your model's performance on unseen data. `sklearn.preprocessing.StandardScaler`: This scaler standardizes features by removing the mean and scaling to unit variance. It's often used to preprocess numerical data before feeding it into machine learning models. `xgboost`: This is the XGBoost library itself, which is a popular gradient boosting library known for its speed and performance

3. Analyzing Design

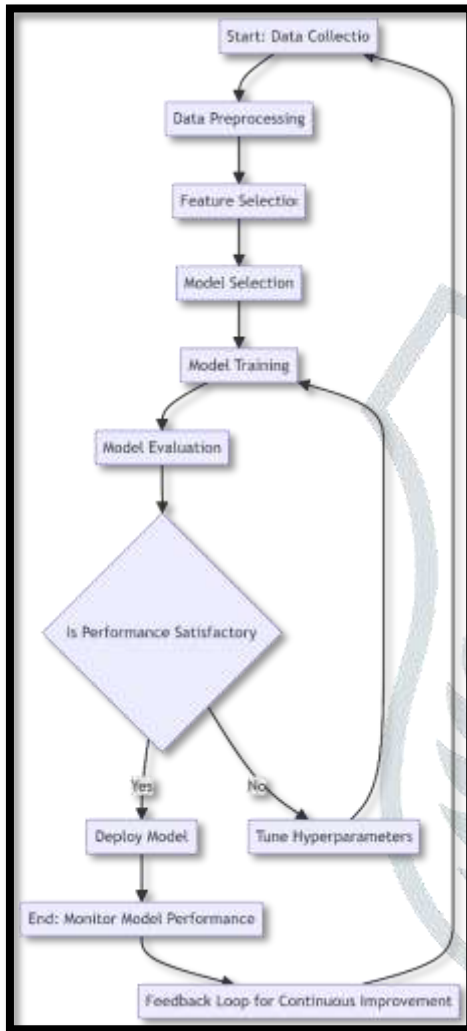


FIGURE 2.3: ANALYSING DESIGN

This process involves loading your dataset, preprocessing it to prepare for modeling, splitting it into training and testing sets, building an XGBoost model, and evaluating its performance using metrics like accuracy and a classification report. Adjustments may be necessary based on your specific dataset characteristics and analysis objectives.

4. Preprocessing Design

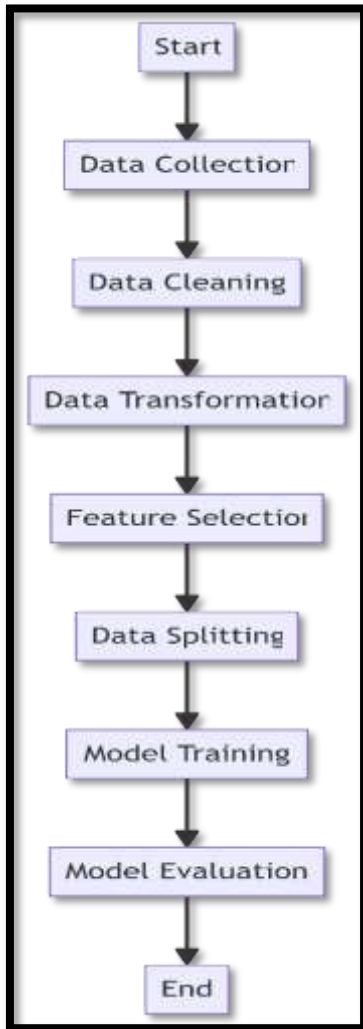


FIGURE 2.4: PREPROCESSING DESIGN

Preprocessing steps provide a structured approach to preparing your dataset for machine learning analysis, ensuring that your data is clean, appropriately encoded, and scaled for effective modeling with algorithms like XGBoost. Adjustments can be made based on your specific dataset characteristics and modeling goal.

5. Splitting the Data Design

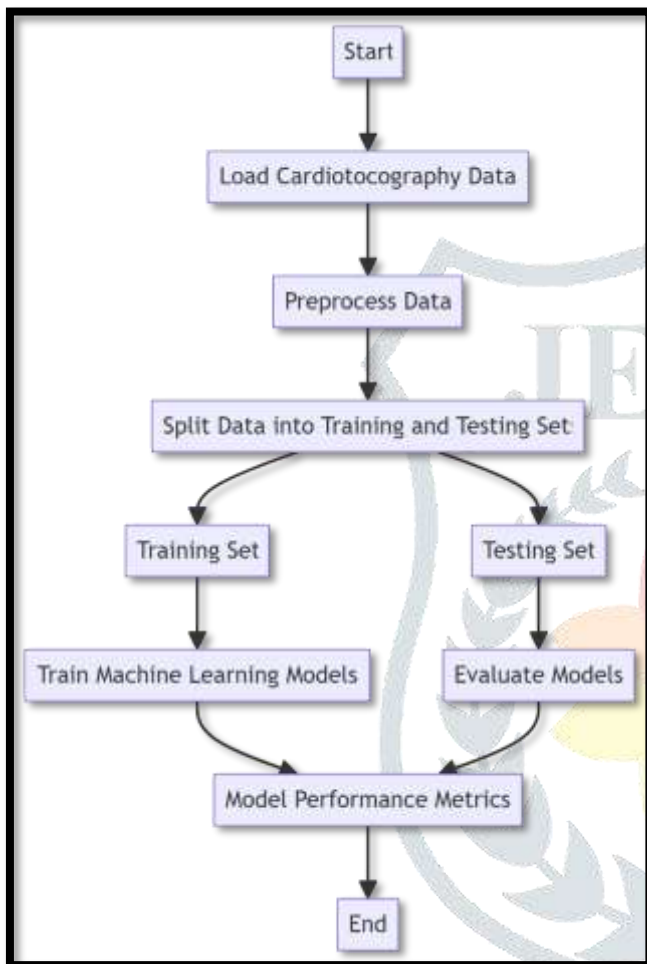


FIGURE 2.5: SPLITTING THE DATA DESIGN

Data splitting is a crucial step in machine learning model development to ensure unbiased evaluation of model performance. It allows you to train your model on one subset and validate it on another, providing insights into its generalization capabilities. Adjust the test-size parameter based on your dataset size and specific requirements.

6. Model Design

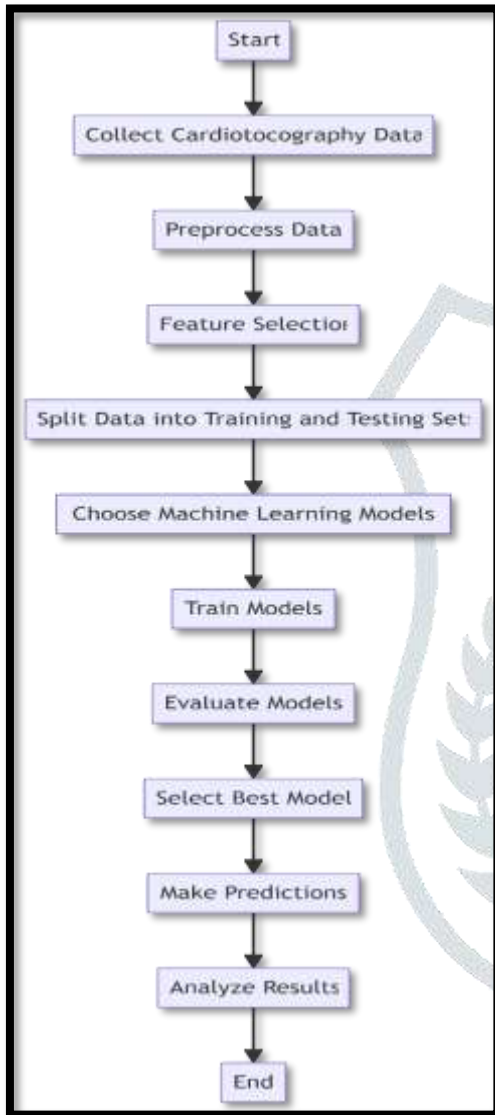


FIGURE 2.6: MODEL DESIGN

When you train an XGBoost model, you can extract various pieces of information about the model, such as feature importance and model parameters. You can retrieve feature importance scores from the trained XGBoost model. Feature importance indicates the relative importance of each feature when making predictions. You can access the parameters used in training the XGBoost model. These parameters include learning rate, maximum depth of trees, number of boosting rounds, etc. XGBoost models are ensemble models typically consisting of multiple decision trees. You can visualize the trees or explore their structure to understand how the model makes predictions.

7. Accuracy

Xgboost accuracy score 96%.

8. Result

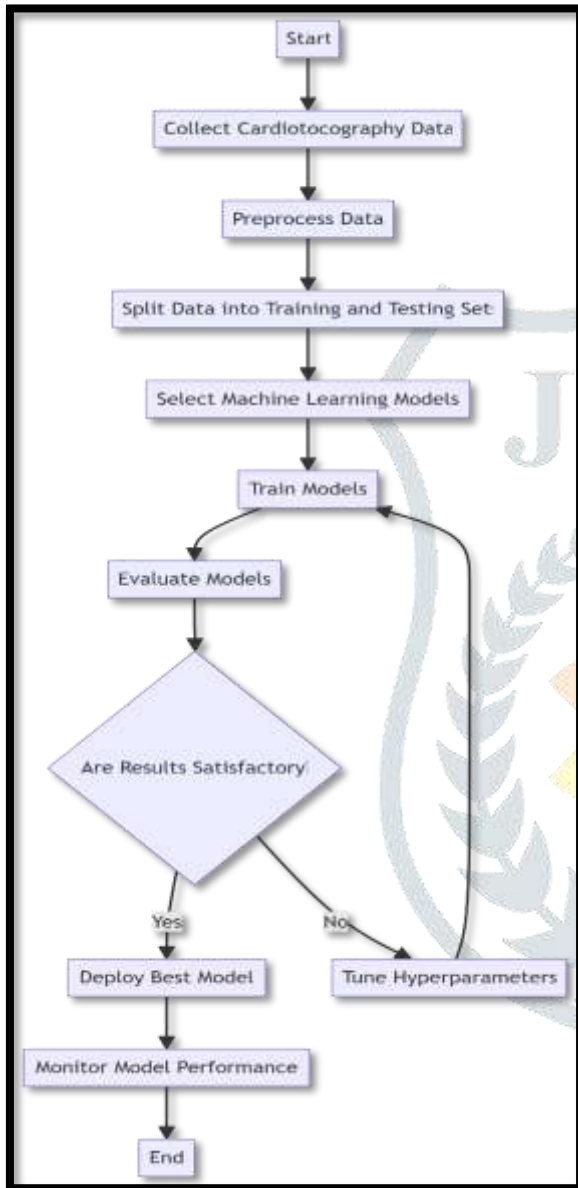


FIGURE 2.8: RESULT

The fetal health dataset consisted of [number of samples] instances with [number of features] features. The target variable, representing fetal health status, was classified into [number of classes] classes.

2.2.3 GIVEN INPUT AND EXPECTED OUTPUT

Given Input

Cardiotocography (CTG) dataset containing physiological measurements collected during fetal monitoring, typically including

- Baseline fetal heart rate
- Number of accelerations per second
- Number of fetal movements per second
- Uterine contractions
- Abnormal short-term variability
- Mean value of short-term variability
- Percentage of time with abnormal long-term variability
- Histogram width, mode, mean, etc.
- **Target variable:** Fetal health, categorized as
 - 1 → Normal
 - 2 → Suspect
 - 3 → Pathological

Expected Result

A trained machine learning model capable of

Predicting fetal health condition (Normal, Suspect, or Pathological) based on CTG input features.

- Performing with high accuracy, precision, and recall.
- Additional outputs may include
 - A confusion matrix to evaluate classification performance
 - ROC-AUC curves and performance metrics.
 - A user-friendly interface (optional) for real-time prediction input.
 - Feature importance analysis to interpret which CTG features most influence prediction.

2.3 TECHNIQUE OR ALGORITHM USED

2.3.1 EXISTING TECHNIQUE

Random forest

- Random Forest can be defined as a collection of decision trees, where each tree is built using a subset of the training data and a subset of the features. The final prediction of the Random Forest model is determined by aggregating.
- The predictions of all individual trees (for classification, typically using the mode of predictions; for regression, typically using the average of predictions). This randomness injects diversity into the ensemble, reducing the risk of overfitting and improving generalization performance.

2.3.2 PROPOSED SYSTEM

XGBoost

- XGBoost combines the strengths of gradient boosting algorithms with several enhancements that make it robust and powerful. It uses a gradient descent optimization technique to minimize the loss when adding new models to the ensemble, thereby improving model accuracy iteratively.
- XGBoost is designed to handle both classification and regression tasks and can handle missing data directly.

CHAPTER 3

REQUIREMENTS ENGINEERING

3.1 GENERAL

We can see from the results that on each database, the error rates are very low due to the discriminatory power of features and the regression capabilities of classifiers. Comparing the highest accuracies (corresponding to the lowest error rates) to those of previous works, our results are very competitive.

3.2 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system do and not how it should be implemented.

- PROCESSOR : DUAL CORE 2 DUOS.
- RAM : 4GB DD RAM
- HARD DISK : 250 GB

3.3 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- Operating System : Windows 7/8/10
- Platform : Spyder3
- Programming Language : Python
- Front End : Spyder3

3.3.1 FEATURES OF PYTHON

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and crossplatform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

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.

3.3.2 THE PYTHON FRAMEWORK

Python frameworks are pre-built collections of modules and tools that simplify the development of applications by providing structure and reusable components. In web development, some of the most popular frameworks include Django, Flask, and FastAPI. Django is a high-level framework that comes with many built-in features such as user authentication, database models, and an admin interface, making it ideal for large applications that need rapid development. Flask, on the other hand, is a microframework that offers more flexibility and is better suited for smaller projects or developers who prefer to customize their tech stack. FastAPI is a newer framework designed for building APIs; it's highly performant and uses modern Python features like type hints and asynchronous programming.

In the realm of software testing, Python offers robust frameworks like unittest, which is included in the standard library, and pytest, which is widely adopted for its simplicity and powerful plugin system. These frameworks help developers write automated tests, ensuring code reliability and enabling continuous integration.

Python also supports frameworks for data science and machine learning. While often referred to as libraries, tools like TensorFlow, PyTorch, and scikit-learn function as frameworks by providing high-level APIs for building and training models. These tools are essential for tasks such as image recognition, natural language processing, and predictive analytics.

For creating desktop applications, Python includes GUI frameworks like Tkinter (built-in), PyQt, and Kivy. These tools allow developers to design interactive applications with graphical interfaces. Kivy is especially useful for creating multi-platform applications that can run on both desktop and mobile devices.

Overall, Python frameworks enhance developer productivity by abstracting common tasks, encouraging best practices, and allowing rapid application development across a wide variety of domains—from web and APIs to data science and desktop apps.

3.3.3 OBJECTIVE OF PYTHON

The main objectives of Python are centered around simplicity, readability, and versatility. Python was designed to be an easy-to-learn programming language with a clean and readable syntax, making it accessible to beginners while still being powerful enough for experts. Its objective is to reduce the complexity of programming by allowing developers to focus on solving problems rather than dealing with intricate syntax. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming, which makes it suitable for a wide range of applications such as web development, data science, artificial intelligence, automation, and software development. Additionally, Python emphasizes code reusability and modularity through its extensive standard libraries and third-party packages.

- **Simplicity and Readability:** Designed with clear and readable syntax to make programming easier.
- **Ease of Learning:** Ideal for beginners due to its intuitive structure.
- **Versatility:** Supports various programming paradigms (procedural, OOP, functional).
- **Extensive Libraries:** Offers a rich standard library and third-party modules for rapid development.
- **Cross-Platform Compatibility:** Runs on multiple platforms (Windows, Linux, macOS) without modification.
- **Rapid Development:** Enables quick prototyping and development of applications.
- **Support for Multiple Domains:** Used in web development, data science, AI, automation, and more.
- **Strong Community Support:** Backed by a large, active community for resources and collaboration.

3.3.4 PYTHON LIBRARIES OVERVIEW

In this project, Python serves as the primary programming language due to its simplicity and robust ecosystem of data science libraries. One of the foundational libraries used is Pandas, which plays a critical role in data handling and manipulation. It allows for easy reading of the Cardiocography (CTG) dataset, handling of missing values, filtering data, and performing exploratory data analysis. With its powerful Data Frame structure, Pandas makes it simple to preprocess and analyze large and complex datasets efficiently.

1. NumPY

Another essential library is NumPy, which supports high-performance mathematical operations on arrays and matrices. It complements Pandas by enabling fast numerical computations, which are often necessary for data normalization, transformation, and matrix-based calculations used in machine learning models.

2. Matplotlib

For data visualization, Matplotlib and Seaborn are employed. Matplotlib is a low-level graphing library that is used to create static visualizations such as histograms, bar charts, and line plots, which help in understanding the distribution and behavior of CTG features. Seaborn, built on top of Matplotlib, is used for more advanced statistical graphics. It provides high-level functions to create attractive visualizations such as heatmaps, boxplots, and pairplots, which are particularly useful for visualizing feature correlations and trends in the fetal health data.

3. Scikit-learn

At the core of the machine learning process, Scikit-learn (sklearn) is used extensively. It offers a wide range of tools for model selection, training, evaluation, and preprocessing. The library includes implementations of various classification algorithms like Logistic Regression, Random Forest, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN), which are tested and evaluated to find the best-performing model. Scikit-learn also provides performance metrics such as accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC, which are crucial for evaluating the classifier's effectiveness.

4. XGBoost

To further improve model performance, especially with tabular data, the project may also include XGBoost. This is a highly optimized gradient boosting framework that is widely used in structured data classification problems. It is known for its speed, scalability, and superior performance, often outperforming traditional algorithms in terms of accuracy.

5. Streamlit

For deploying the machine learning model and creating an interactive interface, Streamlit is used. Streamlit is a lightweight, Python-based web application framework that allows data scientists and developers to create interactive dashboards and apps with minimal effort. In this project, Streamlit enables users to input CTG features via sliders and forms and see real-time predictions of fetal health status. It also supports the integration of plots and metrics, making it ideal for both demonstration and practical use.

6. Pickle/ Joblib

To make the trained machine learning model reusable without retraining it each time, Pickle (or alternatively, Joblib) is used for model serialization. These libraries allow saving the trained model to disk and loading it later for inference, especially useful when integrating the model into the Streamlit interface.

7. Git/GitHub

Finally, tools like Git and GitHub can be used for version control and collaboration. They help track changes in code, manage project versions, and share the project repository online, making it easier for teams to collaborate and deploy the project to platforms like Streamlit Cloud or Heroku.

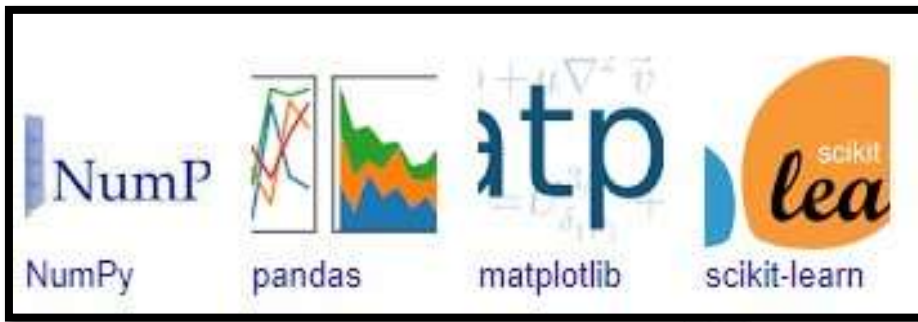


FIGURE 3.3.4 PYTHON LIBRARIES

3.3.5 EVOLUTION OF COLLECTION FRAMEWORK

The Collection Framework in Python has evolved gradually over the years, becoming more powerful and versatile while maintaining the simplicity that Python is known for. Initially, Python relied on its built-in core data types such as list, tuple, dict, and set to handle collections. These basic types are incredibly flexible and intuitive, making them suitable for a wide range of applications. However, as the language matured and the need for more specialized data structures grew, Python's standard library was extended to include more sophisticated and efficient collection types.

A significant milestone in this evolution came with the introduction of the collections module in Python 2.4 (2004). This module provided high-performance, alternative data types that were not only more expressive but also optimized for certain use cases. For instance, namedtuple was introduced to create tuple-like objects with named fields, improving code readability and making it easier to work with structured data. The deque (double-ended queue) data structure was also added, offering faster appends and pops from both ends compared to lists.

Later, Python 2.5 and 2.6 enhanced the collections module further. In Python 2.7, the highly useful Counter class was introduced, providing a straightforward way to count occurrences of elements in an iterable. This class simplified tasks that previously required manual dictionary-based counting logic. At the same time, OrderedDict was added, preserving the insertion order of keys in a dictionary—something that the standard dict didn't guarantee until Python 3.7.

With the release of Python 3.1 and onwards, more improvements were made to the collection framework. Python 3.3 introduced the ChainMap, a convenient class for combining multiple dictionaries or mappings into a single, manageable view. Later, Python 3.5 brought type hints and generic types for collections through the typing module, allowing developers to write more readable and type-safe code (e.g., List[int], Dict[str, float]). From Python 3.6, the standard dictionary type (dict) began preserving insertion order as an implementation detail, and in Python 3.7, this behavior became a language guarantee, making OrderedDict less necessary in some cases.

In recent versions (Python 3.8+), enhancements have continued with improvements to performance and new typing features. Additionally, the collections.abc module has become more important, offering abstract base classes like Iterable, Mapping, Sequence, and MutableMapping—used both for type checking and as base classes for creating custom collections.

Overall, the evolution of Python's collection framework reflects a continuous effort to make the language more efficient, expressive, and developer-friendly. It combines simplicity with power, offering both general-purpose and specialized tools to handle complex data structures in a clean, Pythonic way.

3.4 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behavior, Firstly, the system is the first that achieves the standard notion of semantic security for data confidentiality in attribute-based deduplication systems by resorting to the hybrid cloud architecture.

1. Data Upload and Input Handling

- The system must allow users to upload a CTG dataset in formats such as .csv or .xlsx.
- The system should validate the uploaded dataset and ensure it contains all required features.

2. Data Preprocessing

- The system must clean the data by handling missing values, duplicates, or invalid entries.
- It should normalize or scale numeric values to ensure uniformity for model training.
- Feature encoding should be performed if any categorical data is present.

3. Exploratory Data Analysis (EDA)

- The system should generate visual summaries (histograms, boxplots, correlation heatmaps).
- It must provide statistical summaries of the dataset (mean, median, standard deviation).
- Relationships between features and the target class (fetal_health) must be visualized.

4. Feature Selection and Engineering

- The system must allow for feature selection based on correlation, variance, or feature importance.
- Dimensionality reduction techniques (like PCA) may be optionally applied.

5. Model Training

- The system should train at least three machine learning models (e.g., Logistic Regression, Random Forest, XGBoost).
- It must allow splitting of the dataset into training and testing sets.
- Hyperparameter tuning should be available (grid search or manual input).

6. Model Evaluation

- The system must evaluate trained models using metrics such as
 - Accuracy
 - Precision
 - Recall
 - F1-score
 - Confusion matrix
 - ROC-AUC curve
- The best model should be selected based on performance.

7. Model Prediction

- The system must allow users to input new CTG feature values manually or through file upload.
- It should use the trained model to predict the fetal health class

- 1 = Normal
- 2 = Suspect
- 3 = Pathological

- The result should be displayed clearly with class labels.

8. GUI Interface

- The system must provide a user-friendly GUI (e.g., built using Streamlit).
- GUI should include
 - Input fields/sliders for each CTG parameter
 - A "Predict" button to show results
 - Display of evaluation metrics and charts

9. Model Saving and Loading

- The system must save trained models using pickle or joblib.
- It should allow reloading of saved models for future predictions without retraining.

10. Report Generation

- The system can generate a downloadable summary report with
 - Input values
 - Prediction result
 - Model used
 - Evaluation metrics

3.5 NON-FUNCTIONAL REQUIREMENTS

The major non-functional Requirements of the system are as follows

1. Usability

The system is designed with completely automated process hence there is no or less user intervention.

2. Reliability

The system is more reliable because of the qualities that are inherited from the chosen platform python. The code built by using python is more reliable.

3. Performance

This system is developing in the high level languages and using the advanced back-end technologies it will give response to the end user on client system with in very less time.

4. Supportability

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is built into the system.

5. Implementation

The system is implemented in web environment using Jupyter notebook software. The server is used as the intelligence server and windows 10 professional is used as the platform. Interface the user interface is based on Jupyter notebook provides server system.

3.6 DOMAIN REQUIREMENTS

The domain of the project "Cardiotocography Data Analysis for Fetal Health Classification Using Machine Learning Models", suitable for inclusion in a project report

DOMAIN OF THE PROJECT

The project resides within the broader domain of Healthcare and Artificial Intelligence (AI), specifically focusing on the application of Machine Learning (ML) techniques to improve diagnostic accuracy and decision-making in prenatal care. It involves the intersection of several key disciplines:

1. Healthcare and Medical Diagnostics

This project is fundamentally rooted in the healthcare sector, particularly in obstetrics, where fetal monitoring plays a critical role in assessing the health and well-being of the unborn child. Cardiotocography (CTG) is a widely used non-invasive technique for monitoring fetal heart rate and uterine contractions during pregnancy. The analysis of CTG signals provides vital information about fetal condition, allowing clinicians to detect early signs of fetal distress and make timely decisions regarding interventions.

2. Artificial Intelligence and Machine Learning

With the exponential growth in medical data, AI and ML have emerged as transformative technologies in healthcare. This project utilizes supervised machine learning models to classify fetal health status (e.g., normal, suspect, pathologic) based on features extracted from CTG data. By training models such as Support Vector Machines (SVM), Random Forests (RF), K-Nearest Neighbors (KNN), and Artificial Neural Networks (ANN), the system can learn complex patterns in the data that may not be immediately apparent to human observers.

3. Biomedical Signal Processing

CTG data is a type of biomedical signal that requires preprocessing and feature extraction before it can be effectively analyzed. This involves understanding the physiological basis of fetal heart activity, noise reduction, and the transformation of raw signals into meaningful attributes suitable for input into ML algorithms.

4. Health Informatics and Predictive Analytics

This project also aligns with the field of health informatics, which deals with the acquisition, storage, retrieval, and use of healthcare information. By applying predictive analytics to fetal monitoring data, the project contributes to developing intelligent systems that support clinicians in making informed, data-driven decisions, potentially reducing the rate of complications during childbirth.

5. Clinical Decision Support Systems (CDSS)

The final output of this project can be integrated into a Clinical Decision Support System, which helps healthcare professionals by providing evidence-based assessments of fetal health. These systems can enhance diagnostic consistency, reduce human error, and improve patient outcomes, especially in resource-limited settings.

Overall, this project exemplifies how modern data science and machine learning techniques can be leveraged within the healthcare domain to enhance the accuracy and efficiency of fetal health assessment. It highlights the growing importance of interdisciplinary approaches in solving critical challenges in maternal and fetal medicine.

CHAPTER 4

SYSTEM DESIGN

4.1 GENERAL

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering.

The design engineering of this project focuses on building an efficient, modular, and user-friendly system that automates the process of analyzing cardiotocography (CTG) data to classify fetal health status. The system architecture is divided into multiple well-defined layers, beginning with the data ingestion layer, where raw CTG data is imported, validated, and preprocessed. This includes handling missing values, normalizing feature values, and preparing the dataset for analysis. The next layer involves exploratory data analysis (EDA), where visualizations and statistical summaries are generated to help understand data distributions and feature relationships. Following this, the modeling layer incorporates several supervised machine learning algorithms such as Logistic Regression, Random Forest, and XGBoost, which are trained on the processed data to classify fetal health into three categories: normal, suspect, and pathological. The system also integrates a model evaluation layer, which measures model performance using key metrics such as accuracy, precision, recall, F1-score, confusion matrix, and ROC curves to ensure reliability and effectiveness.

4.2 SYSTEM ARCHITECTURE

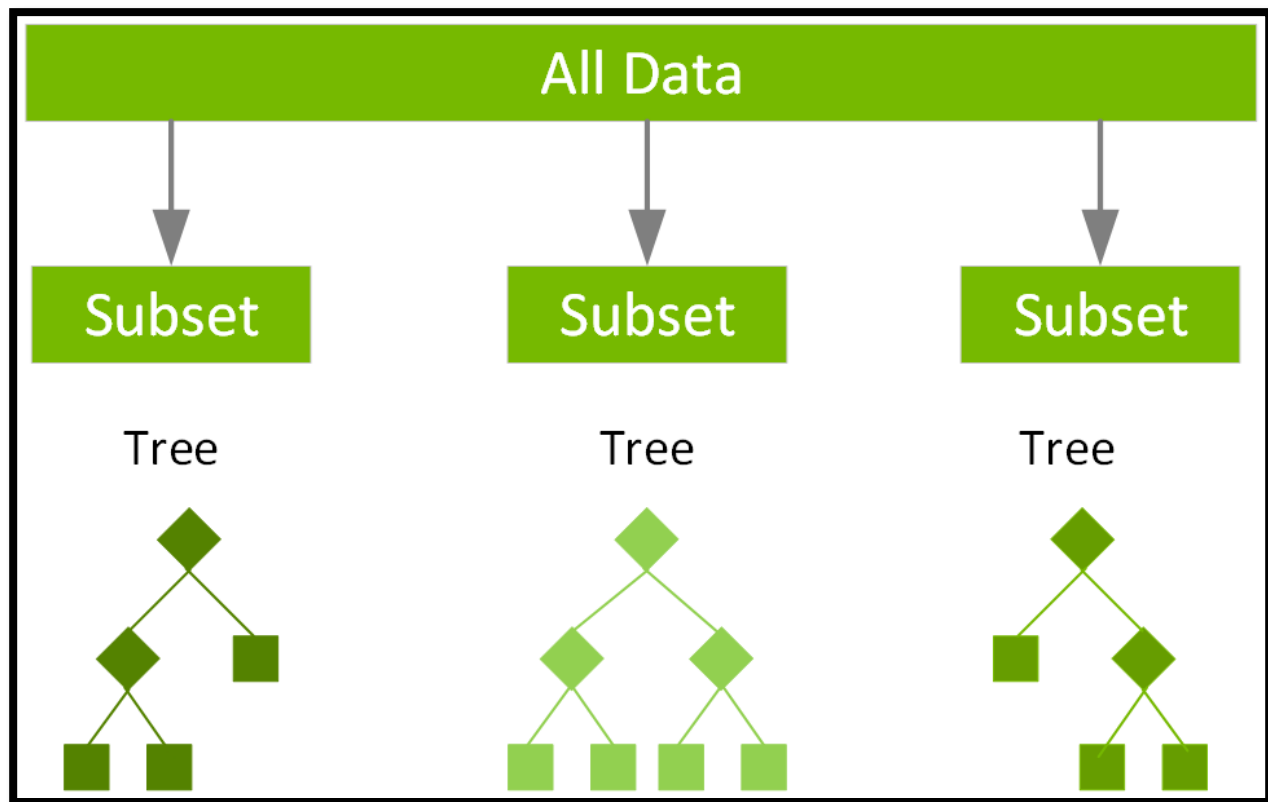


Fig 4.2.1: SYSTEM ARCHITECTURE

EXPLANATION

The diagram represents an ensemble learning architecture, typically used in models like Random Forest, applied to cardiotocography (CTG) data for fetal health classification. The full dataset is divided into multiple subsets, and each subset is used to train an individual decision tree. These trees learn different patterns from the CTG features, such as fetal heart rate and uterine contractions. The final classification—Normal, Suspect, or Pathological—is determined by combining the outputs of all trees, usually through majority voting. This approach improves accuracy, reduces overfitting, and enhances the model's reliability in predicting fetal health conditions.

4.3 UML DIAGRAMS

4.3.1 USE CASE DIAGRAM

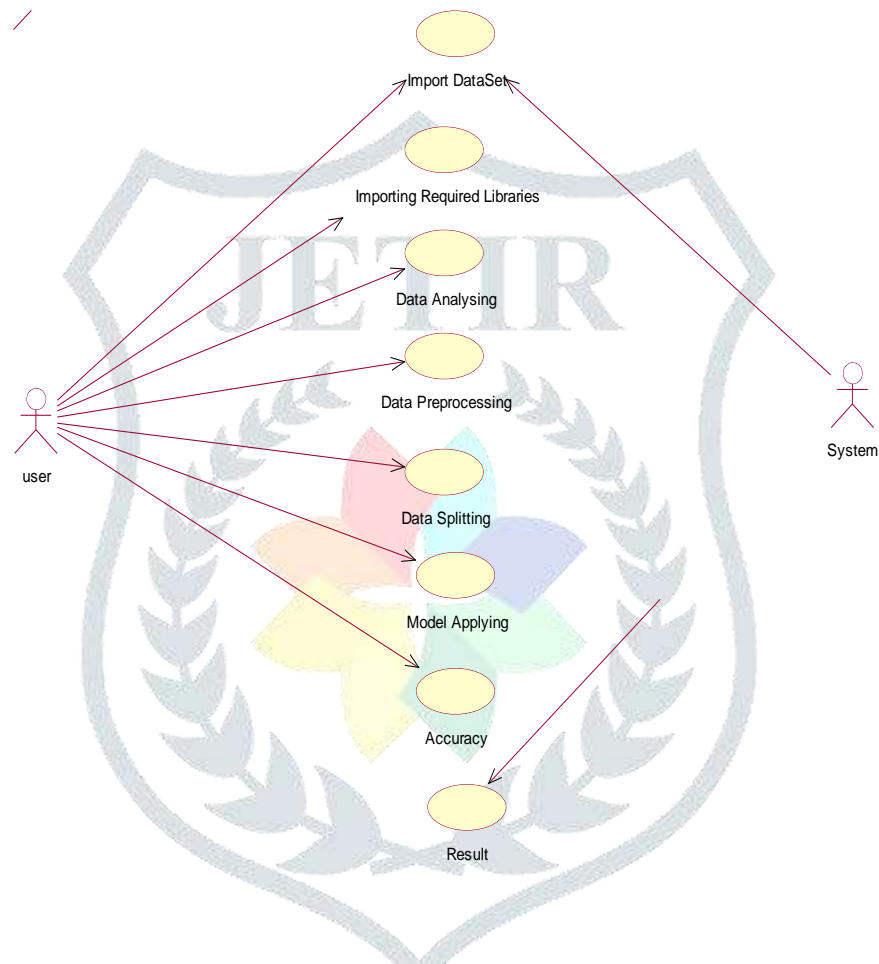


FIGURE 4.3.1: USE CASE DIAGRAM

EXPLANATION

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. The above diagram consists of user as actor. Each will play a certain role to achieve the concept.

4.3.2 CLASS DIAGRAM

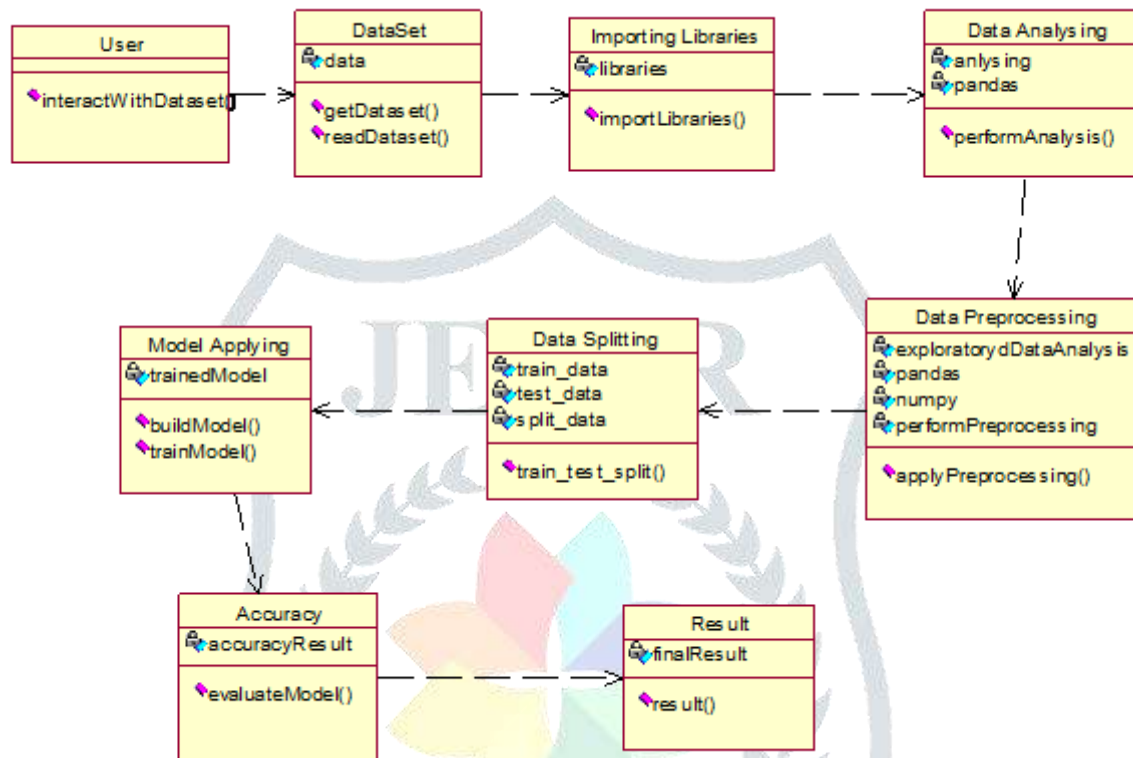


FIGURE 4.3.2: CLASS DIAGRAM

EXPLANATION

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project.

4.3.3 OBJECT DIAGRAM

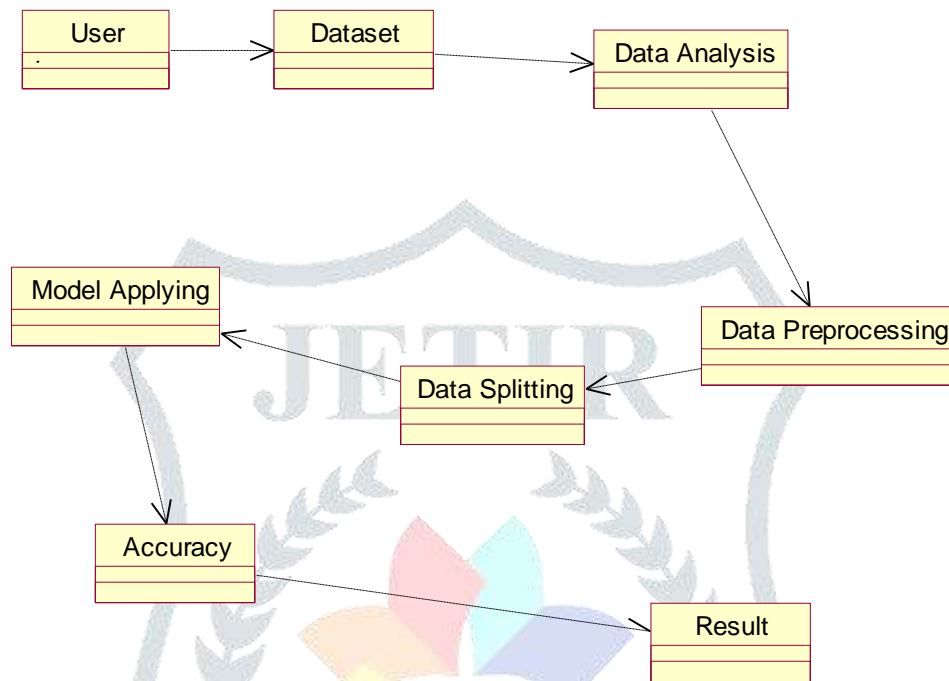


FIGURE 4.3.3: OBJECT DIAGRAM

EXPLANATION

In the above diagram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

4.3.4 STATE CHART DIAGRAM

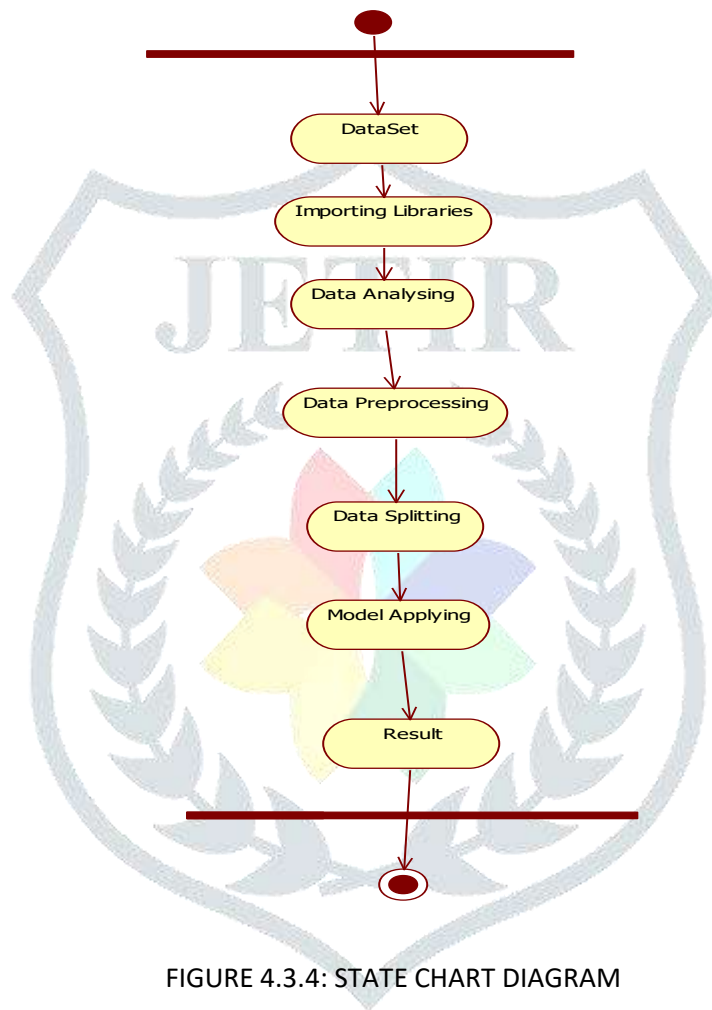


FIGURE 4.3.4: STATE CHART DIAGRAM

EXPLANATION

State diagram are a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

4.3.5 SEQUENCE DIAGRAM

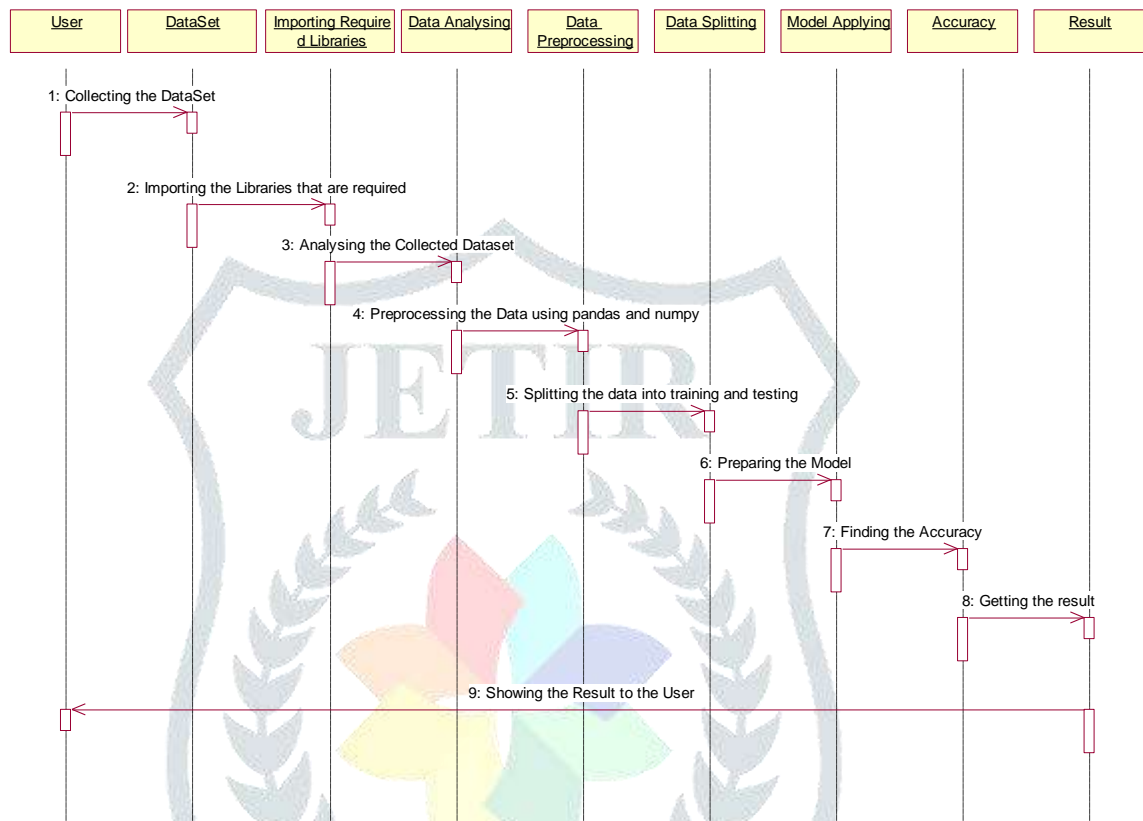


FIGURE 4.3.5: SEQUENCE DIAGRAM

EXPLANATION

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. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

4.3.6 COLLABORATION DIAGRAM

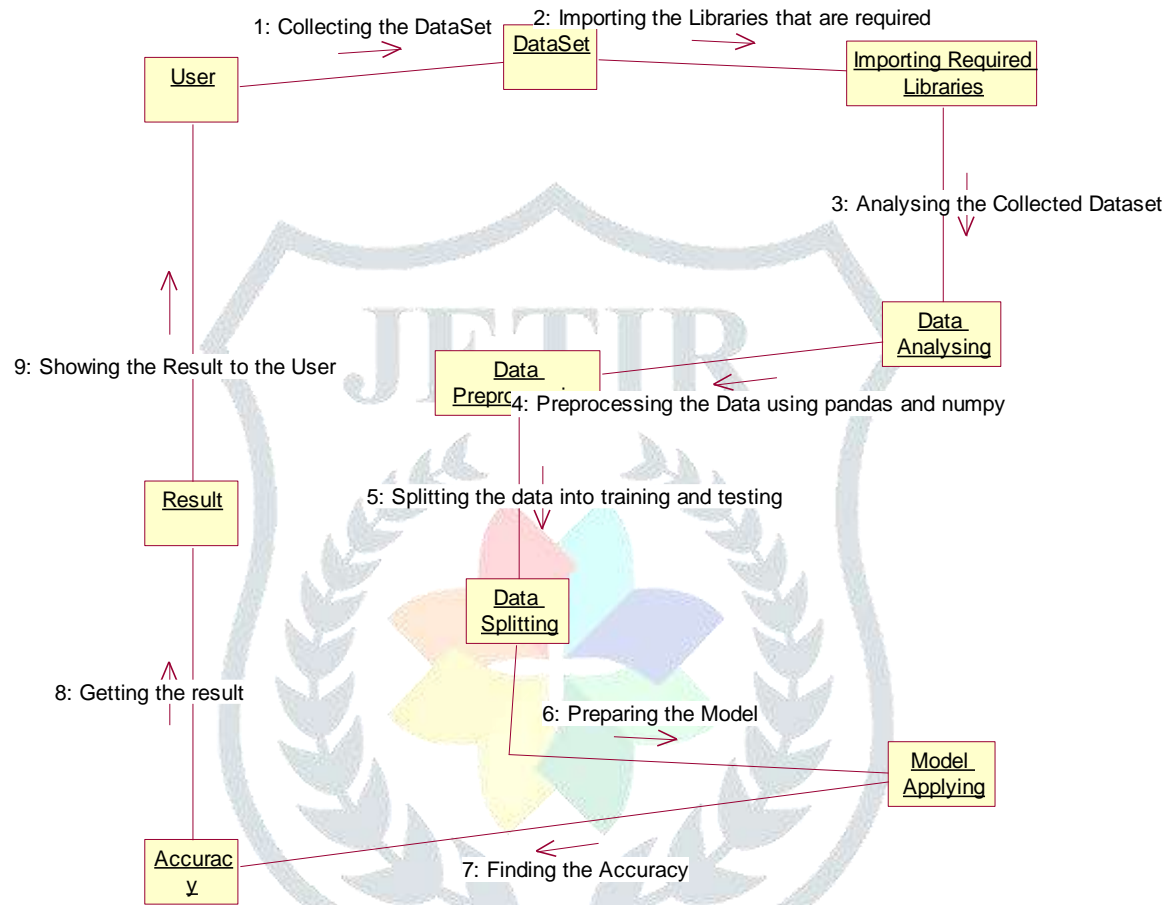


FIGURE 4.3.6: COLLABORATION DIAGRAM

EXPLANATION

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.

4.3.7 ACTIVITY DIAGRAM

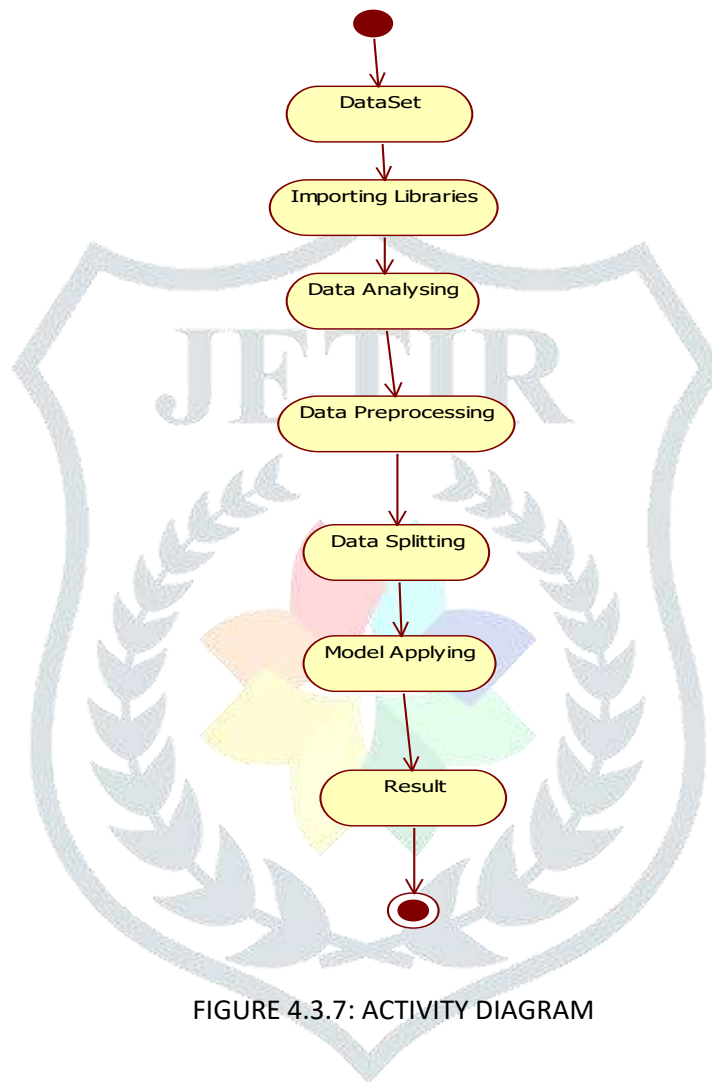


FIGURE 4.3.7: ACTIVITY DIAGRAM

EXPLANATION

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling 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.

4.3.8 COMPONENT DIAGRAM

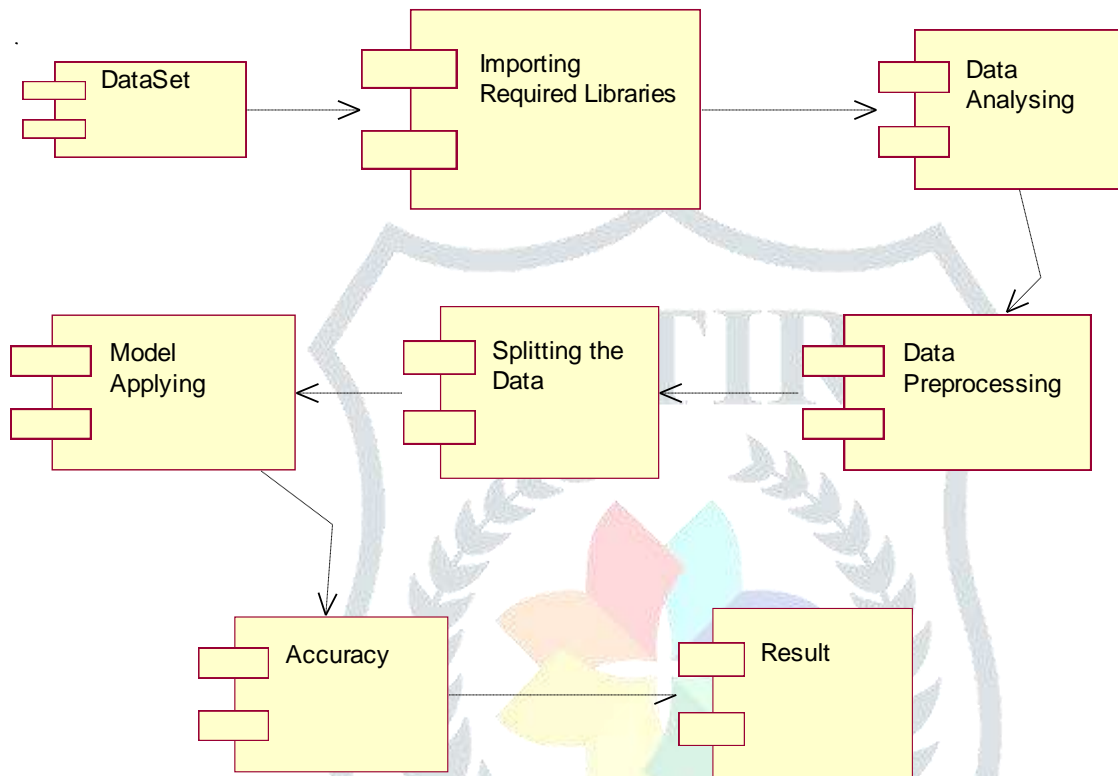


FIGURE 4.3.8 COMPONENT DIAGRAM

EXPLANATION

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

4.3.9 DEPLOYMENT DIAGRAM

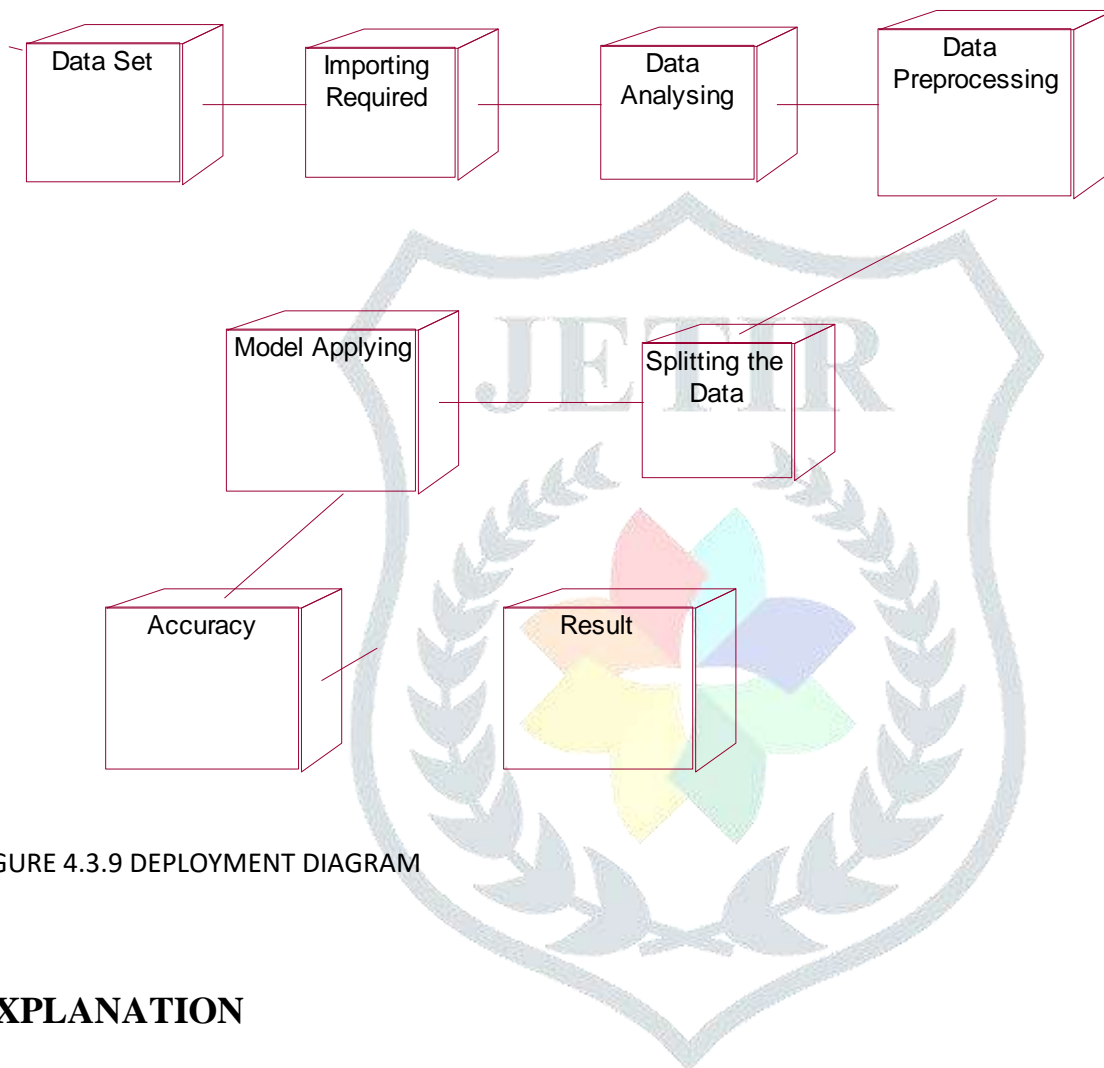


FIGURE 4.3.9 DEPLOYMENT DIAGRAM

EXPLANATION

Deployment Diagram is a type of diagram that specifies the physical hardware on which the software system will execute. It also determines how the software is deployed on the underlying hardware. It maps software pieces of a system to the device that are going to execute it.

4.4 DATA FLOW DIAGRAM

Level 0

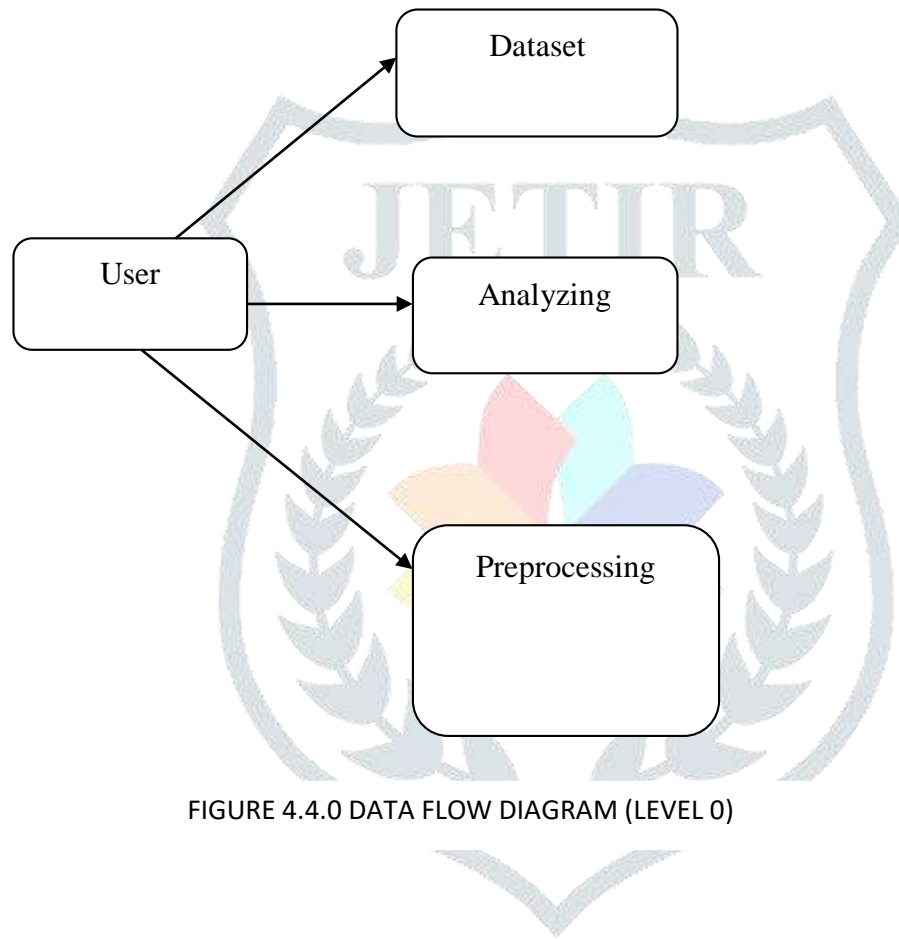


FIGURE 4.4.0 DATA FLOW DIAGRAM (LEVEL 0)

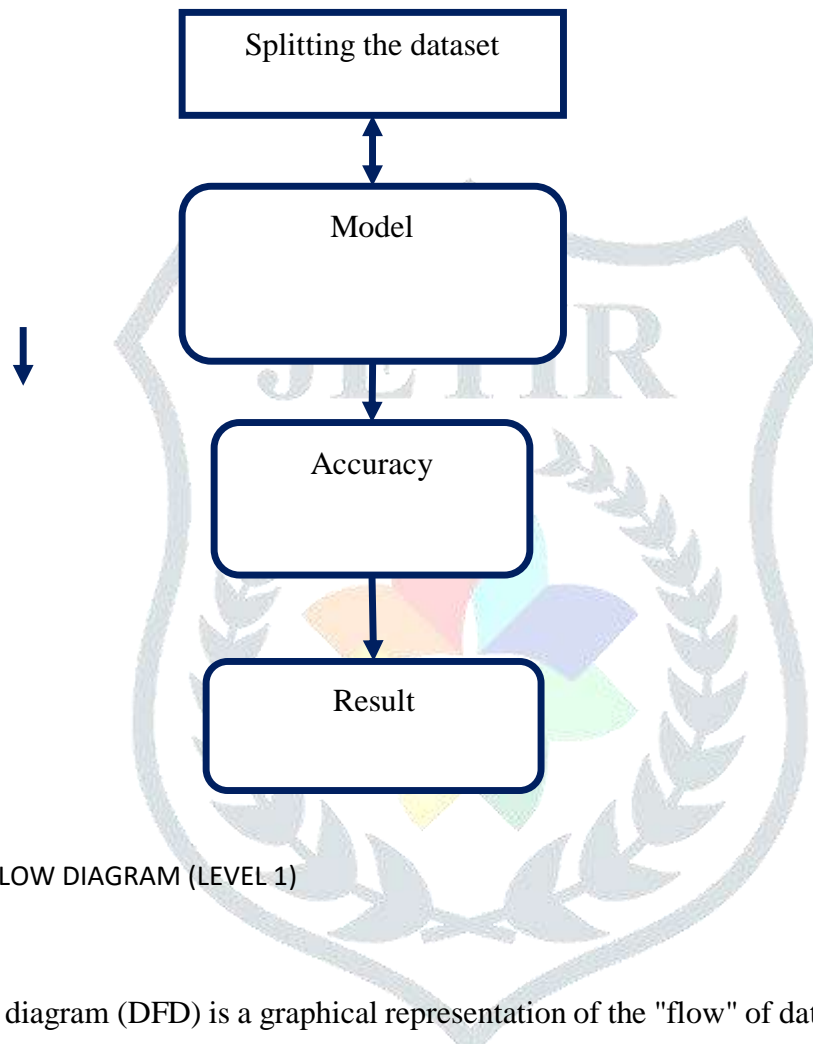
Level 1

FIGURE 4.4.1 DATA FLOW DIAGRAM (LEVEL 1)

EXPLANATION

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about timing of processes, or information about whether processes will operate in sequence or in parallel.

4.5 E-R DIAGRAM

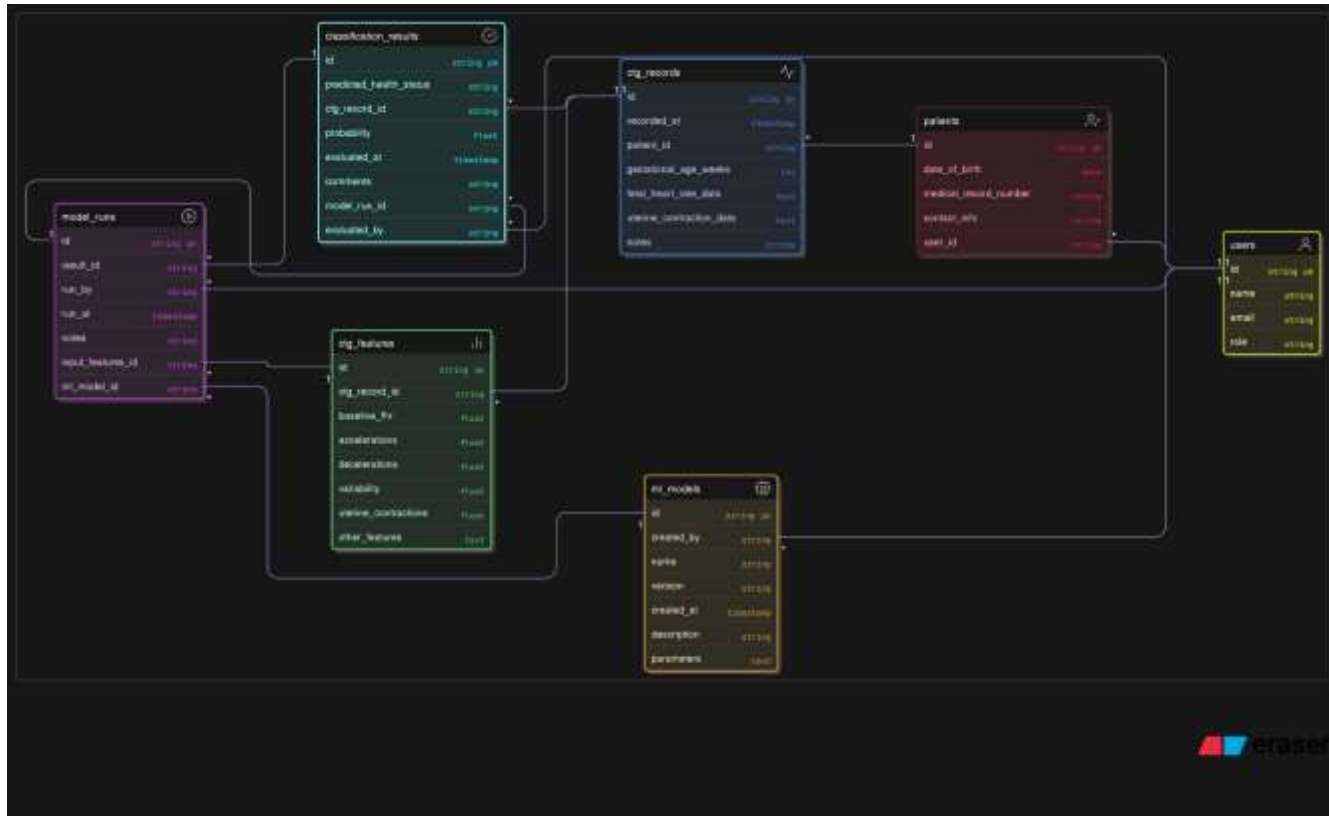


FIGURE 4.5 E-R DIAGRAM

EXPLANATAION

The E-R (Entity-Relationship) diagram for this project illustrates the logical structure of data flow in the Cardiotocography Data Analysis System. It consists of key entities such as Patient, CTG_Record, Model, Prediction, and Doctor. Each Patient can have multiple CTG records, which are analyzed by a Doctor and processed using a Machine Learning Model. The system then generates a Prediction indicating the fetal health status (Normal, Suspect, or Pathologic). Relationships between these entities capture how data is collected, processed, and utilized to assist in clinical decision-making, ensuring efficient and accurate fetal health classification.

4.6 GUI DESIGN

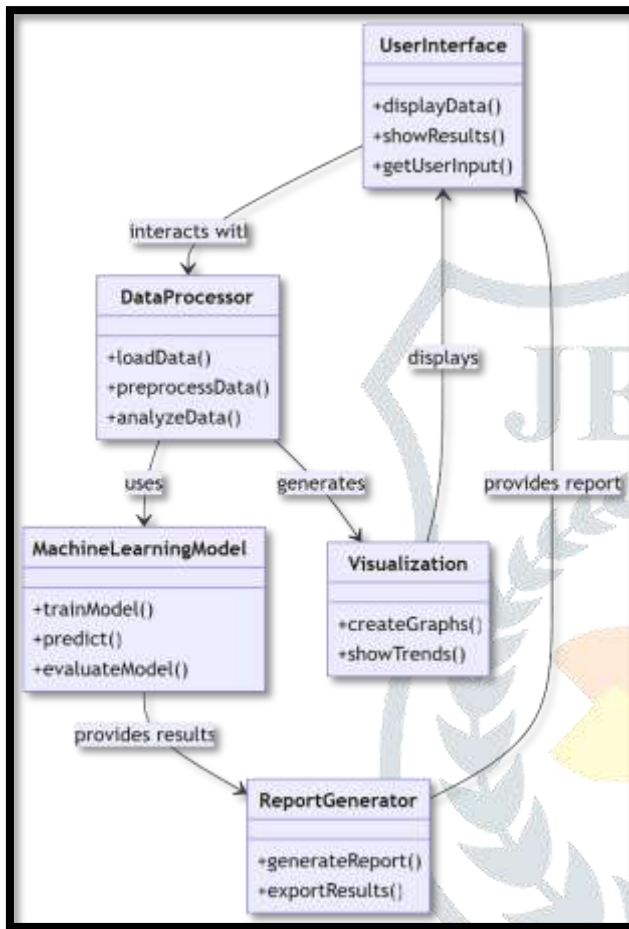


FIGURE 4.6 GUI DESIGN

EXPLANATION

The GUI design for the "Cardiotocography Data Analysis for Fetal Health Classification Using Machine Learning Models" project features a clean, intuitive interface with a left sidebar for easy navigation across sections like data upload, visualization, model training, prediction, and explain ability. The main panel displays relevant content dynamically, such as data previews, model selection, confusion matrices, and SHAP plots for interpretability. Users can upload CTG data, choose machine learning models, visualize insights, run classifications, and generate reports—all within a streamlined, user-friendly environment designed for both functionality and clarity.

4.6.1 COMPONENTS OF GUI

For the project "Cardiotocography Data Analysis for Fetal Health Classification Using Machine Learning Models", the Graphical User Interface (GUI) should be designed to enable medical professionals, data scientists, or healthcare practitioners to interact easily with the system. Below are the essential components of the GUI, categorized for clarity

1. Main Dashboard

- Title/Project Name: Clearly displays the project title.
- Navigation Panel: Sidebar or top menu with options to navigate between different sections
- Upload Data

- Data Visualization
- Model Prediction
- Model Performance
- Settings/Help

2. Data Upload Section

- File Upload Button: To upload .csv or .xlsx CTG datasets.
- Sample File Download: Option to download a sample data template.
- Display Data Preview: Show a table preview of the uploaded dataset (e.g., top 5–10 rows).
- Validation Message Box: Alerts for invalid file format or missing values.

3. Data Visualization Panel

- Statistical Summary: Show mean, median, standard deviation, etc., of selected features.
- Graphs and Charts
 - Histograms for feature distributions (e.g., FHR, UC, etc.)
 - Boxplots for outliers
 - Correlation heatmap
 - Pairplots or scatter plots

4. Model Selection & Training Section

- Model Dropdown: Select from ML models like Random Forest, SVM, Logistic Regression, etc.
- Hyperparameter Inputs: Adjustable sliders or input fields for model parameters.
- Train Button: Initiate model training on uploaded dataset.
- Training Feedback: Console or progress bar showing training status.

5. Prediction Panel

- Input Form: Manually enter values for selected CTG parameters (FHR, UC, etc.)
- Predict Button: Runs the selected ML model and outputs the fetal health status.
- Prediction Result: Text output such as “Normal”, “Suspect”, or “Pathological”.
- Confidence Score: Probability/confidence of prediction (e.g., 92%).

6. Model Performance Metrics

- Evaluation Results: Display key metrics
 - Accuracy
 - Precision, Recall, F1-score
 - Confusion Matrix
 - ROC Curve (if applicable)
- Download Report Button: Export results in PDF or CSV.

7. Settings and Help

- Theme Switcher: Light/Dark mode toggle.
- Help Guide: Short guide or FAQ on how to use the interface.
- About Section: Information about the project, dataset, and authors.

4.6.2 FEATURES OF GUI

The GUI for Cardiotocography Data Analysis and Fetal Health Classification is designed to provide a seamless, interactive experience for users—whether they are healthcare professionals or data scientists. The features of this interface are thoughtfully built to guide users through each step of the analysis pipeline: from data input and visualization, through machine learning model training, to real-time health prediction and result evaluation.

1. User-Friendly Dashboard

Purpose: Central hub for navigation and an overview of the system.

Features

- **Clean and Responsive Layout:** Adaptable to various screen sizes.
- **Project Title and Branding:** Clearly displayed at the top.
- **Navigation Menu:** Sidebar or top navbar with links to all modules
- Upload Data
- Data Visualization
- Model Training
- Prediction
- Evaluation
- Report Generation
- Settings & Help
- Welcome Message and short project description.
- Status Indicators show model readiness, data upload status, and prediction status.

2. Data Upload and Management

Purpose: To allow the user to import and manage CTG datasets.

Features

- **File Upload Support**
 - Accepts .csv and .xlsx formats.
 - Drag-and-drop or button-based file selection.
- **Data Validation**
 - Automatic checks for missing values and incorrect column formats.
 - Alerts for formatting errors.
- **Data Preview Panel**

- Interactive table showing the first 5–10 rows.
- Search and sort options.
- **Sample File Download:** Provides a template for correct data formatting.

3. Data Visualization Tools

Purpose: To help users understand the structure and trends in the dataset.

Features

- **Selectable Feature List:** Choose one or more variables for plotting.
- **Chart Types**
 - **Histograms:** Understand feature distribution.
 - **Boxplots:** Detect outliers.
 - **Scatter Plots:** Analyze feature relationships.
 - **Correlation Heatmap:** Visualize how features relate to one another.

4. Model Selection Panel

Purpose: Allows users to choose and configure machine learning models.

Features

- Model Dropdown: Select from
 - Logistic Regression
 - Random Forest
 - Decision Tree
 - Support Vector Machine (SVM)
 - K-Nearest Neighbors (KNN)
 - Gradient Boosting / XGBoost
- **Hyperparameter Inputs**
 - Adjustable fields for values like:
 - Number of trees (n_estimators)
 - Max depth
 - Learning rate
 - Kernel type (for SVM)
 - Tooltips: Explain each parameter briefly for beginner users.

5. Model Training and Feedback

Purpose: Initiates and monitors model training on the uploaded data.

Features

- **Train Button:** Starts model training with selected parameters.
- **Progress Bar/Spinner:** Visual indication of training status.

- **Console Output**
- **Logs real-time messages:** Training started, accuracy, warnings.
- **Success/Failure Alerts:** Notifies if training was completed or failed.

6. Real-Time Prediction Panel

Purpose: To manually input new patient data and get instant predictions.

Features

- **Input Form:** Fields for key CTG parameters like
 - Baseline FHR
 - Accelerations
 - Decelerations (prolonged, variable, etc.)
 - UC (uterine contractions)
 - Mean/min/max values
- **Predict Button:** Submits the input to the trained model.
- **Result Display**
 - Fetal Health Class: Normal, Suspect, or Pathological
 - Confidence Score (e.g., 92%)
 - Optional color-coded risk level

7. Model Evaluation and Performance

Purpose: To assess how well the trained models perform.

Features

- **Performance Metrics Table**
 - Accuracy
 - Precision
 - Recall
 - F1-score
- **Visual Tools**
 - Confusion Matrix (with colored heatmap)
 - ROC Curve and AUC Score
- **Cross-validation Summary**
 - k-fold results
 - Average metrics across folds

• **Comparison Interface**

- Compare two or more models side-by-side

8. Report Generation and Export

Purpose: Provides options for exporting or printing results and predictions.

Features

- **Generate PDF Reports**
 - Summary of model used
 - Prediction outputs
 - Model performance metrics
- **Download CSV**
 - Raw prediction data
 - Feature-wise model confidence
- **Print Preview**
 - Ready-to-print layout for clinical or academic use.

9. Model Saving and Loading

Purpose: Allows saving trained models and reusing them without retraining.

Features

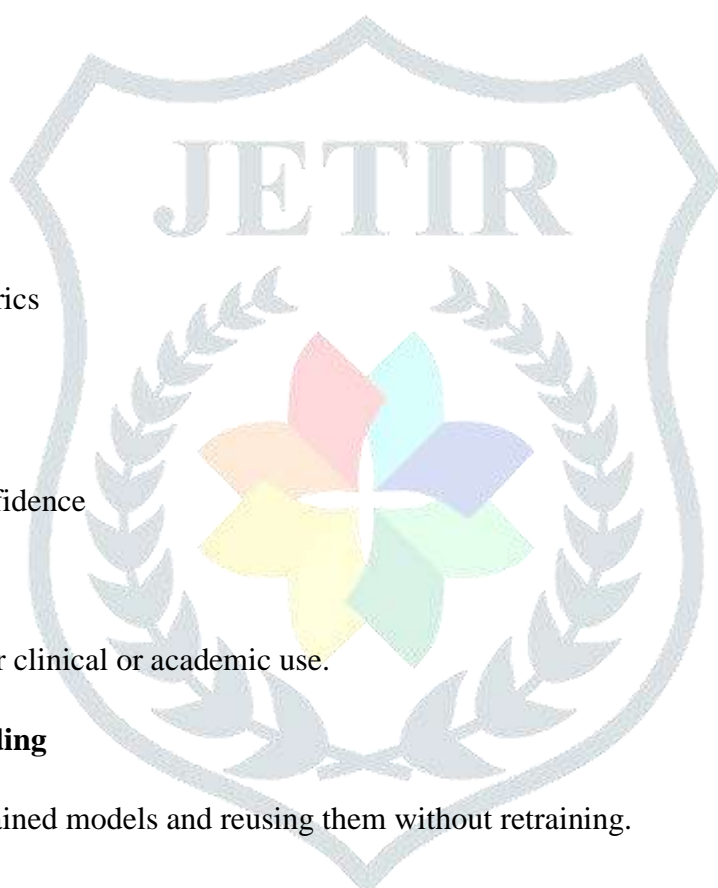
- **Save Model Button:** Exports the trained model as .pkl or similar.
- **Load Model Option:** Upload and activate a previously trained model.
- **Model Metadata Display**
 - Model type
 - Training date
 - Accuracy achieved

10. Settings and Interface Customization

Purpose: Personalizes the user experience.

Features

- **Theme Toggle:** Light or Dark mode
- **Font Size Adjuster:** Improves readability
- **Language Support:** (Optional)
- **Reset Application Button:** Restores default settings



11. Help and Documentation Section

Purpose: Guides users on how to use each part of the GUI effectively.

Features

- **Help Buttons and Tooltips:** On each component
- **User Guide Panel:** Step-by-step instructions with screenshots
- **FAQ Section:** Answers to common problems

CHAPTER 5 IMPLEMENTATION

5.1 GENERAL

The implementation is nothing but the source code of the project

5.2 IMPLEMENTATION

CODING

```
from flask import Flask, request, render_template, redirect, url_for, session, flash
import pandas as pd
import numpy as np
import pickle
import os
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)

# Load the model
with open('xgboost.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

users = {}
app.secret_key = 'your_secret_key'

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

@app.route('/register', methods=["GET", "POST"])
def register():
    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']
        if username in users:
            flash("Username already exists")
            return redirect(url_for('register'))
        users[username] = generate_password_hash(password)
```

```

        flash('Registration successful! Please log in.....')
        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/login', methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']
        user_hash = users.get(username)
        if user_hash and check_password_hash(user_hash, password):
            session["username"] = username
            flash("Login Successfully!.....")
            return redirect('index')
        flash("Invalid username or password.....")
    return render_template("login.html")

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

@app.route('/index', methods=['POST', 'GET'])
def index():
    if request.method == "POST":
        # Extract form data
        baseline_value = float(request.form['baseline_value'])
        accelerations = float(request.form['accelerations'])
        fetal_movement = float(request.form['fetal_movement'])
        uterine_contractions = float(request.form['uterine_contractions'])
        light_decelerations = float(request.form['light_decelerations'])
        severe_decelerations = float(request.form['severe_decelerations'])
        prolonged_decelerations = float(request.form['prolonged_decelerations'])
        abnormal_short_term_variability = float(request.form['abnormal_short_term_variability'])
        mean_value_of_short_term_variability = float(request.form['mean_value_of_short_term_variability'])
        percentage_of_time_with_abnormal_long_term_variability = float(request.form['percentage_of_time_with_abnormal_long_term_variability'])
        mean_value_of_long_term_variability = float(request.form['mean_value_of_long_term_variability'])
        histogram_width = float(request.form['histogram_width'])
        histogram_min = float(request.form['histogram_min'])
        histogram_max = float(request.form['histogram_max'])
        histogram_number_of_peaks = float(request.form['histogram_number_of_peaks'])
        histogram_number_of_zeroes = float(request.form['histogram_number_of_zeroes'])
        histogram_mode = float(request.form['histogram_mode'])
        histogram_mean = float(request.form['histogram_mean'])
        histogram_median = float(request.form['histogram_median'])
        histogram_variance = float(request.form['histogram_variance'])
        histogram_tendency = float(request.form['histogram_tendency'])

        # Create DataFrame for prediction
        new_data = pd.DataFrame({

```

```

        'baseline_value': [baseline_value],
        'accelerations': [accelerations],
        'fetal_movement': [fetal_movement],
        'uterine_contractions': [uterine_contractions],
        'light_decelerations': [light_decelerations],
        'severe_decelerations': [severe_decelerations],
        'prolongued_decelerations': [prolongued_decelerations],
        'abnormal_short_term_variability': [abnormal_short_term_variability],
        'mean_value_of_short_term_variability': [mean_value_of_short_term_variability],
        'percentage_of_time_with_abnormal_long_term_variability':
[percentage_of_time_with_abnormal_long_term_variability],
        'mean_value_of_long_term_variability': [mean_value_of_long_term_variability],
        'histogram_width': [histogram_width],
        'histogram_min': [histogram_min],
        'histogram_max': [histogram_max],
        'histogram_number_of_peaks': [histogram_number_of_peaks],
        'histogram_number_of_zeroes': [histogram_number_of_zeroes],
        'histogram_mode': [histogram_mode],
        'histogram_mean': [histogram_mean],
        'histogram_median': [histogram_median],
        'histogram_variance': [histogram_variance],
        'histogram_tendency': [histogram_tendency]
    })

    # Convert DataFrame to numpy array
    new_data = new_data.values

    # Make prediction
    prediction = loaded_model.predict(new_data)

    # Determine result based on predicted class
    if prediction == 1:
        result = "Normal"
    elif prediction == 2:
        result = "Suspect"
    else:
        result = "Pathological"

    return render_template('result.html', prediction = result )
return render_template("index.html")

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

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

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

```

```
if __name__ == '__main__':  
    app.run()
```

STEPS

Step 1: Open the project file present on the desktop and copy the URL.

Step 2: Open Anaconda prompt and write the following code and hit enter.

cd (paste the file URL)

Step 3: The next step is to write the following code and press enter.

Python app.py

Step 4: Copy the given URL address provided and open it in the web browser.

Step 5: You would be directed to the homepage of the project.

Step 6: Login if your account exists or register yourself.

Step 7: Once login is successful you we directed to the “Input Data for Fetal Health Classification” page.

Step 8: Fill out the inputs given on the page from the data that is recorded in the excel sheet.

Step 9: Once all the inputs are filled out and rechecked, hit the submit button.

Step 10: There are three outcomes that can be generated NORMAL, PATHOLOGICAL or SUSPECT.

CHAPTER 6

SNAPSHOTS

6.1 GENERAL

This project implements like application using python and the Server process is maintained using the SOCKET & SERVERSOCKET and the Design part is played by Cascading Style Sheet.

6.2 OUTPUT SNAPSHOTS

6.2.1 COPYING THE FILE URL

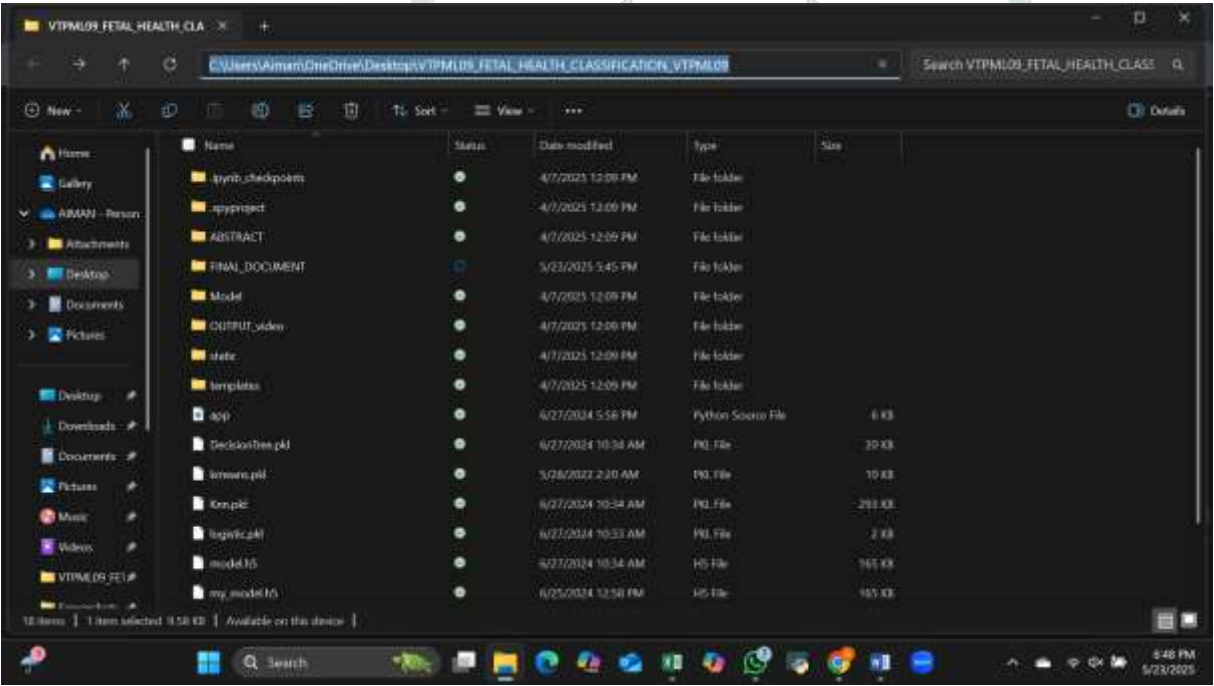


FIGURE 6.2.1 COPY THE FILE URL

6.2.2 STARTING ANACONDA PROMPT

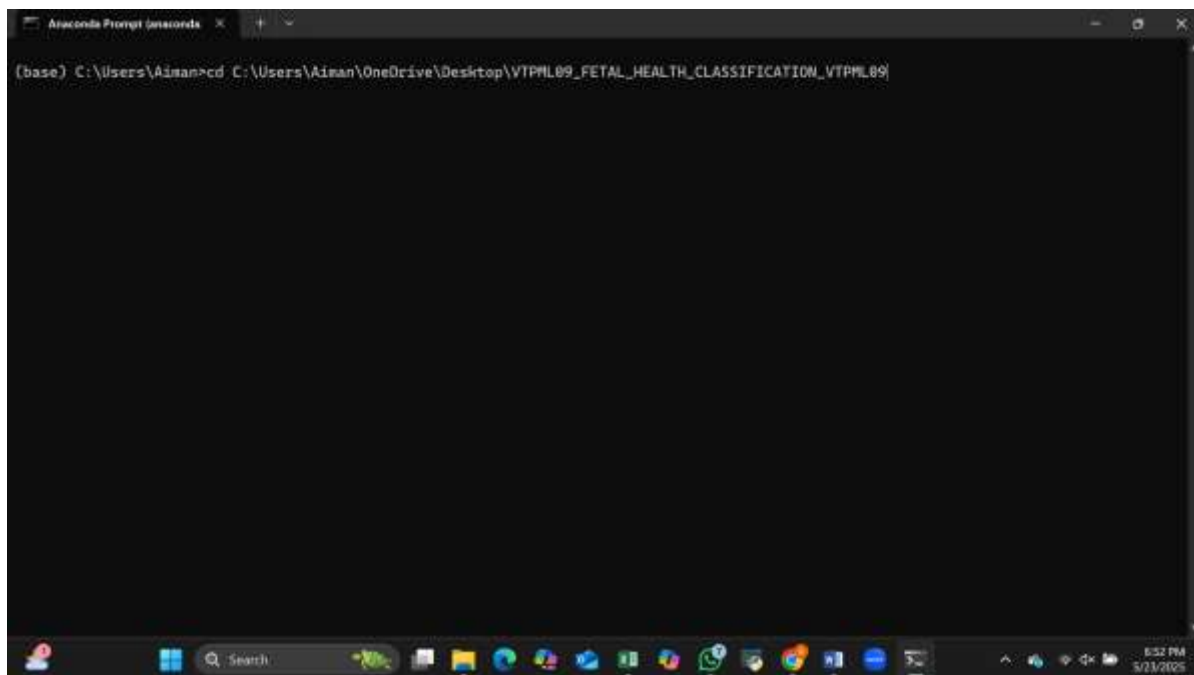


FIGURE 6.2.2 STARTING ANACONDA PROMPT

6.2.3 RUNNING THE APP.PY

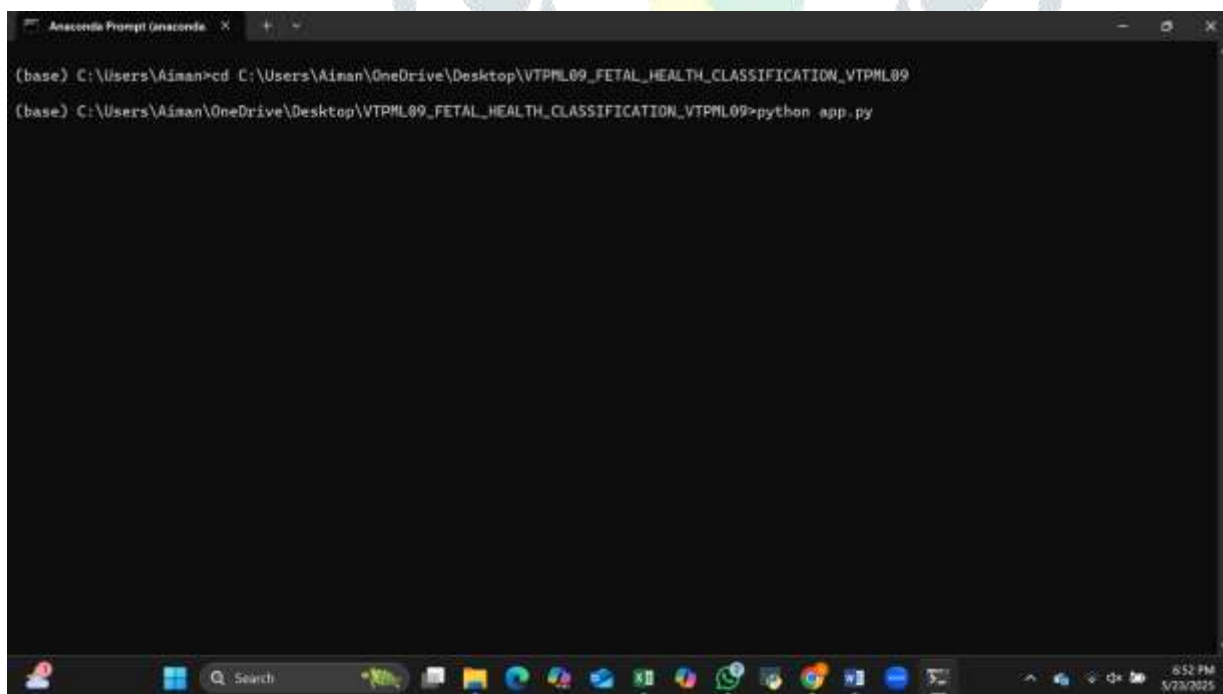
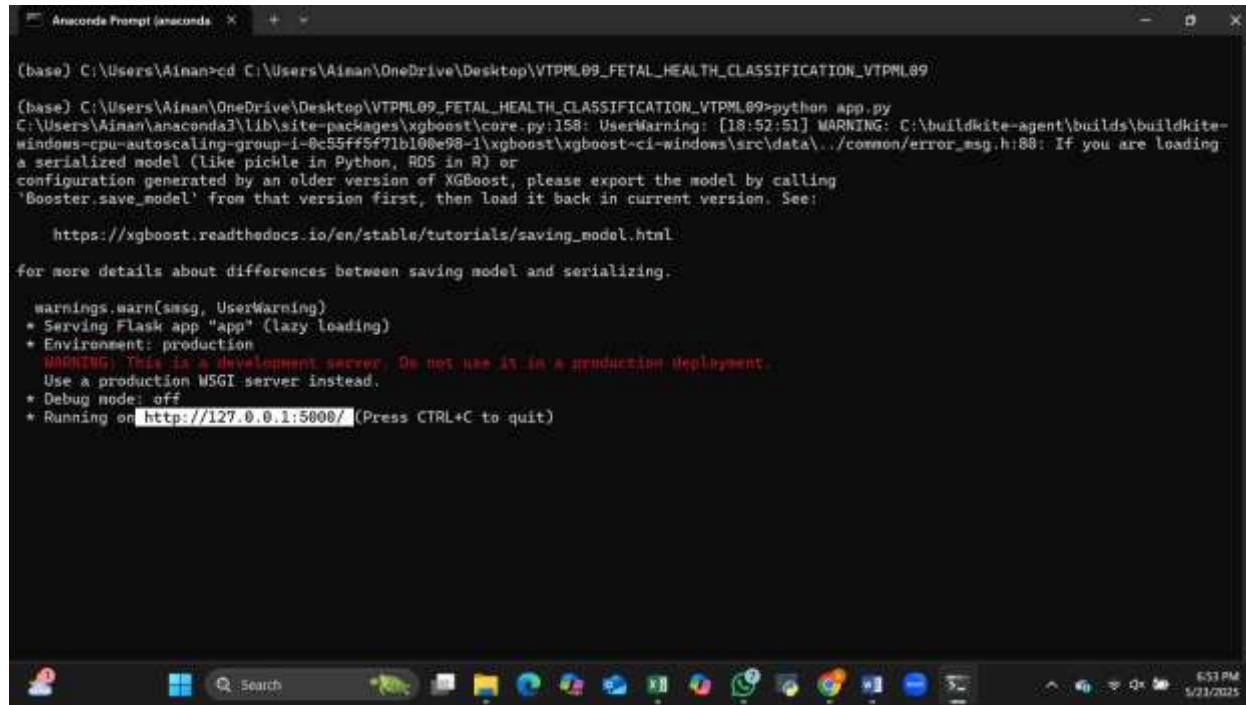


FIGURE 6.2.3 RUNNING THE APP.PY

6.2.4 COPYING THE URL



```
(base) C:\Users\Ainan>cd C:\Users\Ainan\OneDrive\Desktop\VTPL09_FETAL_HEALTH_CLASSIFICATION_VTPL09

(base) C:\Users\Ainan\OneDrive\Desktop\VTPL09_FETAL_HEALTH_CLASSIFICATION_VTPL09>python app.py
C:\Users\Ainan\anaconda3\lib\site-packages\xgboost\core.py:158: UserWarning: [18:52:51] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0c55ff5f71b100e98-1\xgboost\xgboost-ci-windows\src\data\..\common/error_msg.h:88: If you are loading
a serialized model (Like pickle in Python, RDS in R) or
configuration generated by an older version of XGBoost, please export the model by calling
'Booster.save_model' from that version first, then load it back in current version. See:
https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html
for more details about differences between saving model and serializing.

warnings.warn(msg, UserWarning)
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

FIGURE 6.2.4 COPYING THE URL

6.2.5 OPENING THE WEB BROWSER

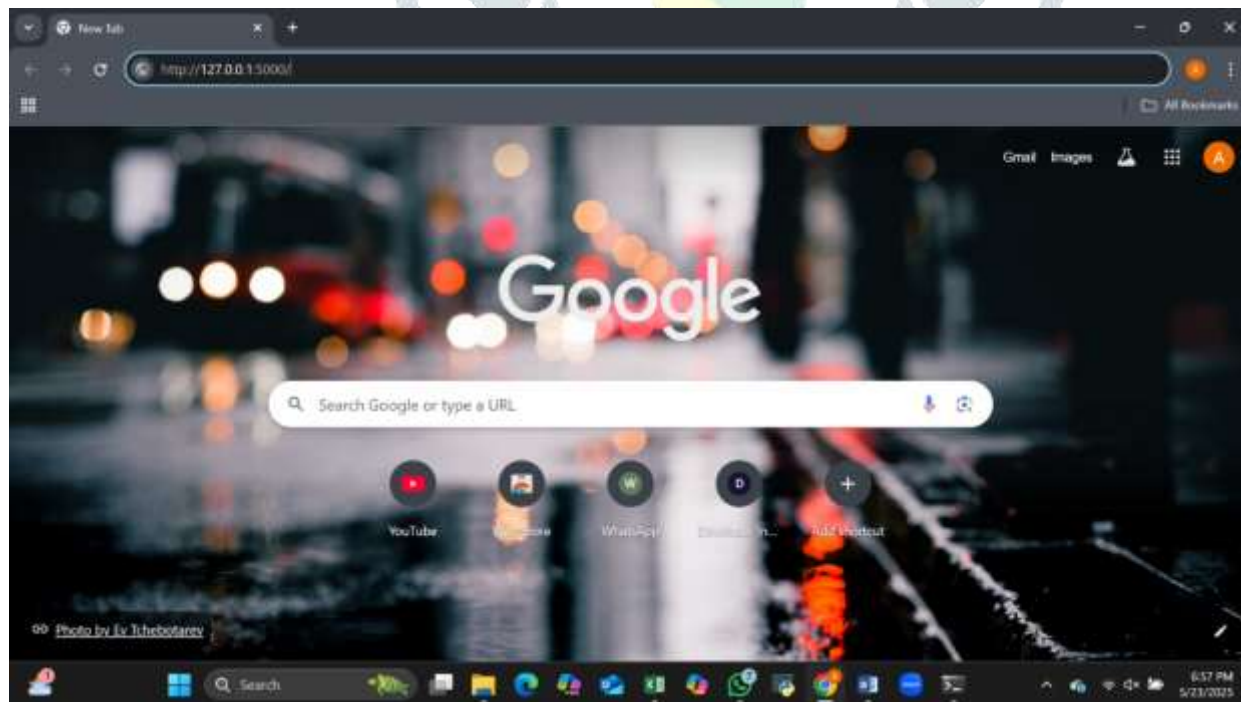


FIGURE 6.2.5 OPENING WEB BROWSER

6.2.6 PROJECT HOME PAGE



FIGURE 6.2.6 PROJECT HOME PAGE

6.2.7 LOGIN PAGE

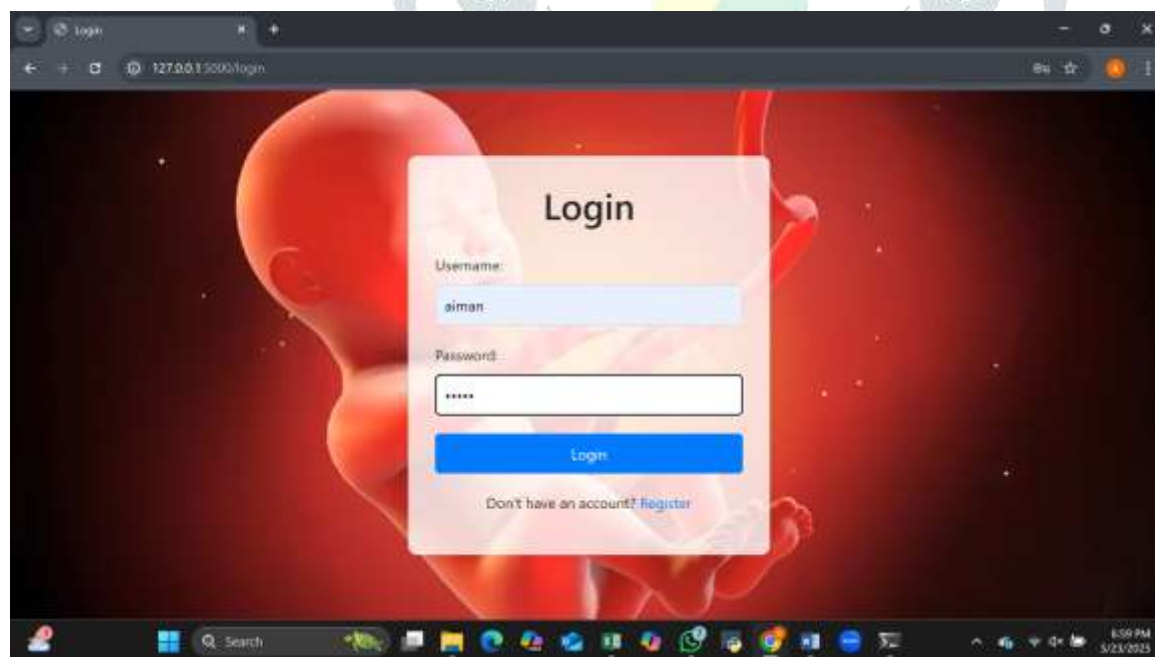


FIGURE 6.2.7 LOGIN PAGE

6.2.8 INPUT DATA FROM TEST DATA EXCEL SHEET

Input Data For Fetal Health Classification

Baseline Value	Abnormal Short Term Variability	Histogram Number of Peaks
Accelerations	Mean Value of Short Term Variability	Histogram Number of Zeroes
Fetal Movement	Percentage of Time Abnormal Long Term Variability	Histogram Mode
Uterine Contractions	Mean Value of Long Term Variability	Histogram Mean
Light Decelerations	Histogram Width	Histogram Median
Severe Decelerations	Histogram Min	Histogram Variance
Prolonged Decelerations		Histogram Tendency

FIGURE 6.2.8 INPUT DATA FROM TEST DATA EXCEL SHEET

6.6.9 TEST DATA EXCEL SHEET

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	baseline_v	accelerati	fetal_mov	uterine_cc	light_dece	severe_de	prolongue	abnormal	mean_val	percentag	mean_val	histogram	histogram	histogram	histogram	histogram	histogram	histogram	histogram
2	120	0	0	0	0	0	0	73	0.5	43	2.4	64	62	126	2	0	120	137	121
3	134	0.001	0.002	0.006	0.004	0	0	0	0	0	0.009	0	0	0.316	0	0	152	2	0.004
4	132	0.006	0	0.006	0.003	0	0	17	2.1	0	10.4	130	68	198	6	1	141	136	140
5	133	0.003	0	0.006	0.003	0	0	16	2.1	0	13.4	130	68	198	5	1	141	135	138
6	134	0.003	0	0.006	0.003	0	0	16	2.4	0	23	117	53	170	11	0	137	134	137
7	151	0	0	0.001	0.001	0	0	64	1.9	9	27.6	130	56	186	2	0	150	148	151
8	132	0.007	0	0.006	0	0	0	16	2.4	0	19.9	117	53	170	9	0	137	136	138
9	128	0	0	0.006	0.003	0	0	1	0	0.005	0.004	0	0	0.004	10	5	23	25	21
10	131	0.005	0.072	0.006	0.003	0	0	28	1.4	0	12.9	66	88	154	5	0	135	134	137
11	150	0	0	0.001	0.001	0	0	64	2	8	29.5	130	56	186	5	0	150	148	151

FIGURE 6.2.9 TEST DATA EXCEL SHEET

6.6.10 FILL THE INPUTS AND SUBMIT

Input Data for Fetal Health Classification

Baseline Value	Abnormal Short Term Variability	Histogram Number of Peaks
120	17	6
Accelerations	Mean Value of Short Term Variability	Histogram Number of Zeroes
0.006	2.1	1
Fetal Movement	Percentage of Time Abnormal Long Term Variability	Histogram Mode
0	0	141
Uterine Contractions	Mean Value of Long Term Variability	Histogram Mean
0.006	10.4	136
Light Decelerations	Histogram Width	Histogram Median
0.003	130	140
Severe Decelerations	Histogram Min	Histogram Variance
0	68	12
Prolonged Decelerations	Histogram Max	Histogram Tendency
0	198	1

Submit

FIGURE 6.2.10 FILL THE INPUTS AND SUBMIT

6.6.11 RESULTS OF THE INPUT DATA

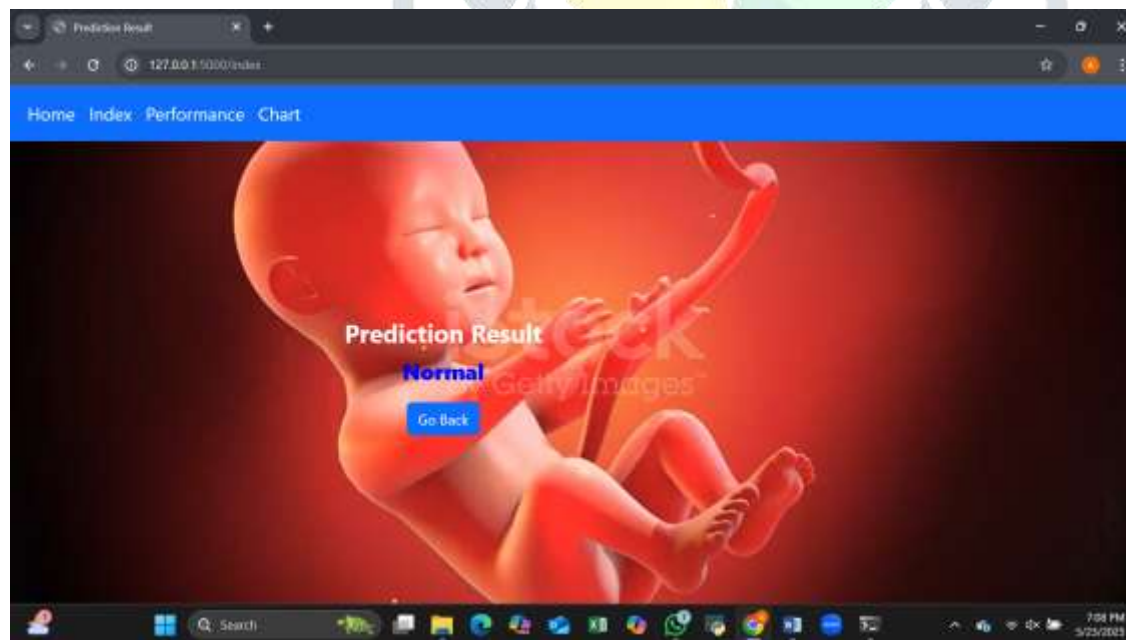


FIGURE 6.2.11 RESULTS OF THE INPUT DATA

6.6.12 MODEL ACCURACY OF THE FETUS

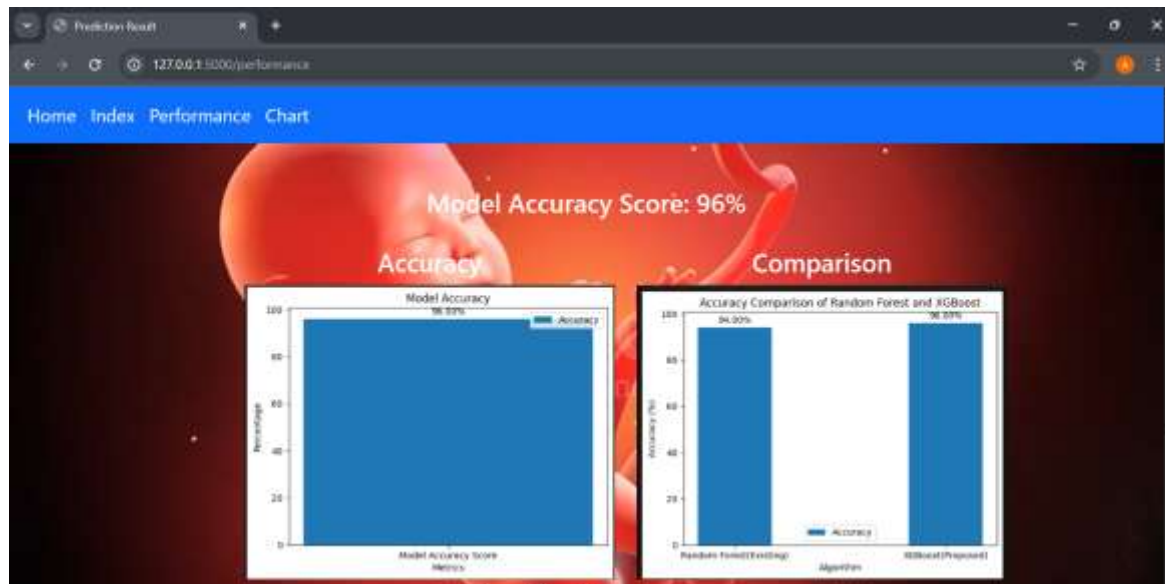


FIGURE 6.2.12 MODEL ACCURACY OF THE FETUS

6.6.13 CLASSIFICATION REPORT OF THE FETUS

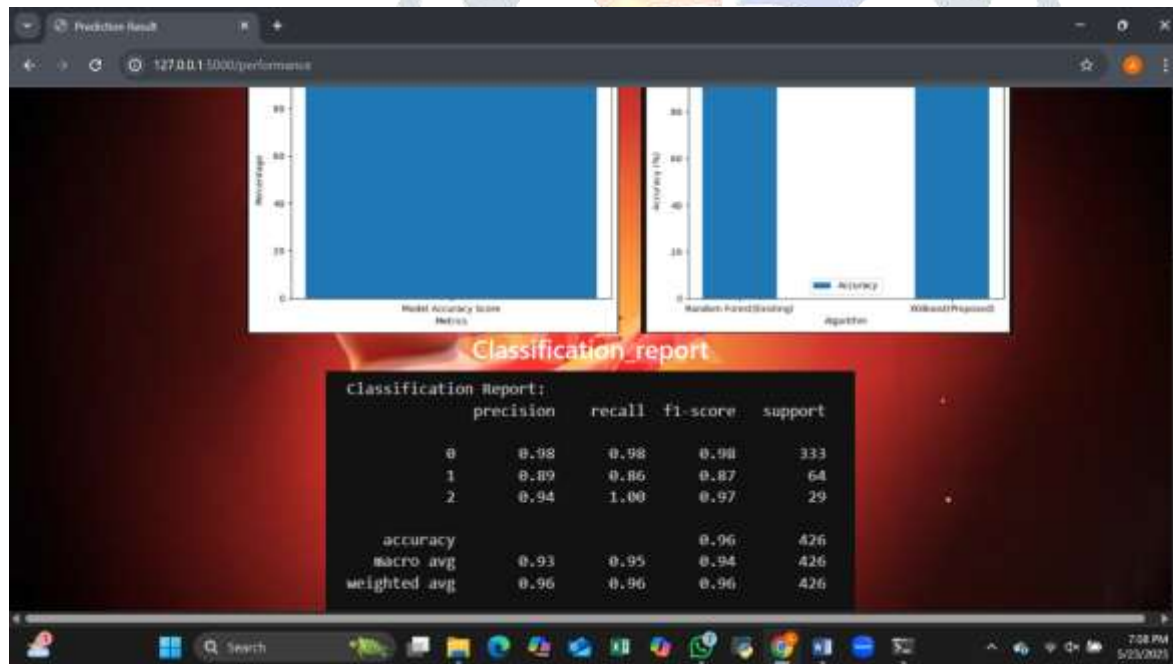


FIGURE 6.2.13 CLASSIFICATION REPORT OF THE FETUS

6.2.14 PIE CHART SHOWING DATA CLASSIFICATION



FIGURE 6.2.14 PIE CHART SHOWING DATA CLASSIFICATION

CHAPTER 7 SOFTWARE TESTING

7.1 GENERAL

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.

7.2 DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

The development of this project follows a systematic and iterative machine learning methodology, combining principles from both data science and software engineering. The project begins with the data acquisition phase, where the cardiotocography (CTG) dataset is sourced—typically from a reliable source like the UCI Machine Learning Repository. In the data preprocessing phase, the dataset undergoes cleaning, normalization, and transformation to handle missing values, scale features, and prepare the data for analysis. This is followed by exploratory data analysis (EDA), where various statistical and visual tools such as correlation heatmaps, boxplots, and histograms are used to understand patterns, detect outliers, and identify relevant features. The next critical step involves feature selection and engineering, ensuring that only the most informative features are used to improve model performance and reduce computational complexity.

7.3 TEST STRATEGY

7.3.1 LEVEL OF TESTING

Software Testing is an activity performed to identify errors so that errors can be removed to obtain a product with greater quality. To assure and maintain the quality of software and to represent the ultimate review of specification, design, and coding, Software testing is required. There are different levels of testing

1. Unit Testing: In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that they are fit for use by the developers. It is the smallest testable part of the software.
2. Integration Testing: In this testing, two or more modules which are unit tested are integrated to test i.e., technique interacting components, and are then verified if these integrated modules work as per the expectation or not, and interface errors are also detected.
3. System Testing: In system testing, complete and integrated Softwares are tested i.e., all the system elements forming the system are tested as a whole to meet the requirements of the system.
4. Acceptance Testing: This is a kind of testing conducted to ensure that the requirements of the users are fulfilled before its delivery and that the software works correctly in the user's working environment.

7.3.2 TYPES OF TESTING

1. UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input 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.

2. FUNCTIONAL TESTS

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.

3. 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.

4. PERFORMANCE TEST

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

5. INTEGRATING SYSTEM

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.

6. 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.

Acceptance testing for Data Synchronization

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

7.3.3 TEST CASE TYPE--GUI

Test Case ID	Module	Test Scenario	User Action	Expected Result	Actual Result	Result
1	Input Form	Check if all input fields are visible	Open the application	All CTG input fields (sliders/text boxes) are visible	All fields visible	Pass
2	Input Form	Submit empty form	Click "Predict" without entering values	Display validation error or prevent prediction	Validation error shown	Pass
3	Prediction	Predict with valid input	Enter valid values for all features and click "Predict"	Prediction result displayed (e.g., "Normal")	Prediction displayed	Pass
4	Prediction	Predict with extreme values	Enter very high or very low values and click "Predict"	Show result or handle gracefully	Prediction displayed	Pass
5	Output Display	Check result display	Perform a valid prediction	Correct fetal health category shown	Correct output shown	Pass

Test Case ID	Module	Test Scenario	User Action	Expected Result	Actual Result	Result
6	Usability	Check layout responsiveness	Resize browser window or use mobile view	Layout adjusts without breaking	Layout responsive	Pass
7	Model Loading	Load model from file	Launch app (auto-load model via Pickle)	Model loaded and ready for prediction	Model loaded	Pass
8	Error Handling	Predict with invalid input type	Enter string instead of number in a field (if allowed)	Error message or input blocked	Input restricted	Pass
9	Button Function	Click "Predict" button	Fill fields → Click Predict	Function triggers model and displays result	Button works correctly	Pass
10	Visualization	Display of charts or metrics	Navigate to evaluation output section (if available)	Confusion matrix / Accuracy metrics visible	Charts displayed	Pass

7.3.4 TEST DESIGN TECHNIQUES

The testing phase of this project employs a combination of black-box and white-box test design techniques to ensure comprehensive verification of both the user interface and the machine learning backend. Black-box testing is used extensively in the graphical user interface (GUI) to validate that user inputs, such as CTG parameters entered via sliders or forms, produce the correct outputs without examining the internal code. Test scenarios include input validation, error handling, result display, and responsiveness across different devices. On the other hand, white-box testing is applied to core Python functions and modules, including data preprocessing, feature transformation, and model prediction logic. These tests verify the internal workflow, data flow between functions, and conditional branches in the code.

The project also utilizes equivalence partitioning to test input ranges by dividing the input domain into valid and invalid classes (e.g., normal CTG values vs. extreme outliers) and testing representative values from each class. Boundary value analysis is performed to check how the system handles minimum and maximum values of fetal health parameters like fetal heart rate, uterine contractions, and accelerations. For the machine learning component, data-driven testing is used, where the model is evaluated on multiple datasets and test samples to ensure consistent and accurate predictions. Lastly, decision table testing is incorporated when verifying the model's prediction logic based on combinations of input features, ensuring that all important decision rules are exercised. By combining these techniques, the project ensures both functional accuracy and robustness under various use-case scenarios.

7.3.5 TEST ENVIRONMENT

The test environment for this project is specifically configured to support the development, training, evaluation, and validation of machine learning models using the XGBoost algorithm. It ensures reproducibility, scalability, and high performance while handling structured medical data like cardiotocography records.

1. Hardware Environment

The project runs on a system with the following specifications

- **Processor:** Intel Core i7 or AMD Ryzen 7 (8-core) or higher, to handle XGBoost's parallel processing features effectively.
- **Memory (RAM):** Minimum 16 GB, allowing for efficient in-memory computation with moderately sized medical datasets.
- **Storage:** At least 256 GB SSD, ensuring fast read/write speeds for dataset access and model storage.
- **GPU (Optional):** NVIDIA GPU with CUDA support (e.g., RTX 3060 or better). While XGBoost can run efficiently on CPU, GPU acceleration (with `tree_method='gpu_hist'`) significantly speeds up training on large datasets.

2. Operating System

The setup is compatible with

- Ubuntu 20.04+ (preferred for ML work due to easier package and environment management),
- Windows 10/11
- macOS (for local development, though GPU support may be limited).

3. Software Environment

- **Programming Language:** Python 3.8 or higher.
- **IDE/Editor:** Jupyter Notebook for exploratory analysis, and VS Code or PyCharm for structured development and testing.
- **Version Control:** Git, with GitHub used for tracking changes, collaborative coding, and version history.

4. Python Libraries Used

- **Data Processing**
- pandas, numpy – for data loading, manipulation, and preprocessing.
- **Visualization**
- matplotlib, seaborn, plotly – to visualize data distribution, correlations, and model performance.
- **Modeling (Core)**
- xgboost – main algorithm for training and classification.
- scikit-learn – for model evaluation, train-test splitting, pipeline creation, and metrics computation.
- **Model Explainability**
- shap – for interpreting XGBoost predictions using SHAP values.
- **Testing**
- pytest, unittest – for testing preprocessing and model-related functions.

5. Data Source

- The Cardiocography dataset is obtained from the UCI Machine Learning Repository.
- The dataset is stored in CSV format and loaded directly into the environment using pandas.

6. Testing and Validation Setup

- **Cross-validation:** Stratified k-fold cross-validation (using StratifiedKFold) is applied to ensure model generalizability.
- **Model Evaluation Metrics:** Accuracy, Precision, Recall, F1-score, ROC-AUC.
- **Hyperparameter Tuning:** Performed using GridSearchCV or RandomizedSearchCV from scikit-learn.
- **Model Stability Tests:** Trained models are tested against slight data perturbations to assess robustness.

7. Deployment & Monitoring Tools

- **Streamlit or Flask:** For creating an interactive web interface to demonstrate model predictions.
- **MLflow:** Can be used to log experiments, hyperparameters, and model versions.
- **Docker:** Optional containerization of the environment for portability.

This test environment ensures that the machine learning models built using XGBoost are trained, evaluated, and validated under consistent and reliable conditions, with the ability to scale and adapt for deployment or clinical feedback.

7.4 ACCEPTANCE CRITERIA

Acceptance criteria define the specific conditions that must be met for the project to be considered complete and successful. These criteria ensure the project delivers its intended value, functionality, and quality in the context of fetal health classification using machine learning.

1. Data Integrity and Preprocessing

- The cardiotocography dataset must be correctly loaded, cleaned, and preprocessed (e.g., no missing values, proper encoding of categorical features).
- Outliers and inconsistencies should be handled appropriately (either removed, flagged, or normalized).
- Feature scaling and transformations must be reproducible and applied consistently across training and testing sets.

2. Model Implementation (XGBoost)

- The XGBoost classifier must be successfully implemented using appropriate hyperparameters.
- The model should be trained on the training set and evaluated using a validation/testing set.
- Stratified K-Fold Cross-Validation must be used to ensure model reliability and avoid overfitting.

3. Classification Accuracy and Metrics

- The trained model must classify fetal health into three classes (Normal, Suspect, Pathological).
- The model should meet minimum performance benchmarks
- Accuracy: $\geq 90\%$
- F1-score: ≥ 0.88 (weighted average)
- ROC-AUC: ≥ 0.90
- Confusion matrix, classification report, and ROC curves must be generated and interpreted.

4. Model Explainability

- The project must include explainability techniques (e.g., SHAP) to interpret XGBoost predictions.

- Clear visualization of feature importance must be provided.
- At least one correctly and one incorrectly classified example must be explained using SHAP plots or similar.

5. User Interface / Output Presentation

- Results must be presented in a clear, user-friendly format (either via notebook outputs, report, or basic UI).
- If deployed via web interface (optional), the user must be able to upload data and receive a prediction with class label and explanation.

6. Documentation and Reporting

- The project report must include
 - Introduction, objectives, and dataset description
 - Preprocessing steps
 - Model development process
 - Evaluation metrics
 - Results and analysis
 - Conclusion and future work
- All code must be well-commented and structured.
- A README file must be provided if the project is hosted (e.g., on GitHub).

7. Reproducibility and Test Coverage

- The entire workflow must be reproducible (same inputs yield same outputs).
- Random seeds must be set for training to ensure consistency.
- Unit tests for key functions (data cleaning, model training, prediction) must pass.

Importance of Acceptance Criteria

Here are the following acceptance criteria mentioned below Importance of Acceptance Criteria

1. **Define Expectations:** Clearly outline what needs to be accomplished for a user story or feature to be considered done.
2. **Communication:** Facilitate effective communication between development teams, product owners, and stakeholders, ensuring everyone is on the same page.
3. **Quality Assurance:** Act as a quality control measure, helping to prevent misunderstandings and minimize defects.
4. **Scope Control:** Prevent scope creep by setting clear boundaries on what is and isn't part of the user story or feature.
5. **Testing Guidelines:** Provide guidance for writing test cases and conducting acceptance testing.

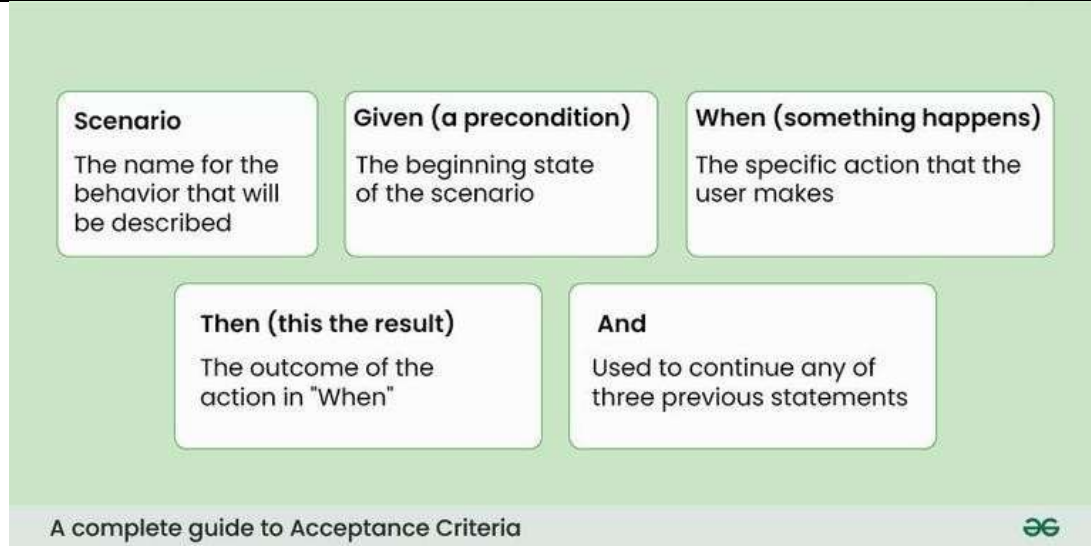


FIGURE 7.4.1: TYPES AND STRUCTURE OF ACCEPTANCE TESTING

- Given: The initial context or situation.
- When: The action or event that triggers the criteria.
- Then: The expected outcome or result.
- And: For continuation of any previous statement

7.4.1 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.

Acceptance testing for Data Synchronization

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

7.5 BUILD THE TEST PLAN

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identify the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

Once each unit is tested individually and verified to be functioning correctly, the next step is to integrate these components and perform integration testing. This phase ensures that the interaction between different modules is smooth and that data flow and communication between components are functioning as intended. Integration testing helps detect issues such as interface mismatches, data inconsistencies, or incorrect module interactions that might not be visible during unit testing. By following a layered testing approach—starting from unit testing, moving to integration, and eventually to system and acceptance testing—developers can ensure a more robust and error-free software system. This systematic approach also reduces the cost and effort of identifying bugs in later stages of the software development lifecycle.

CHAPTER 8

CONCLUSION

8.1 GENERAL

Finally, a rank of data containing the filtered data with correlation value is given as the data feed.

8.2 FUTURE ENHANCEMENTS

Future research should focus on further refining and expanding the proposed framework. The incorporation of more advanced machine-learning models, such as transformers or federated learning approaches, holds great potential for enhancing the durability and reliability of the system. Additionally, exploring the integration of real-time data analytics and wearable health monitoring devices can significantly advance early detection and intervention strategies. This can ultimately lead to improved maternal and fetal outcomes in cases of pregnancy complications. Collaboration with interdisciplinary teams, including healthcare professionals and data scientists, will be crucial in developing robust and scalable solutions. Finally, addressing the ethical considerations and ensuring data privacy and security will be paramount in the widespread adoption of these technologies in clinical settings.

8.3 CONCLUSION

In conclusion, our research underscores the critical theoretical and practical implications of understanding the impact of pregnancy complications on maternal and fetal health. Our investigation has illuminated significant insights that contribute to the field, particularly emphasizing the importance of early identification and intervention to mitigate the risks associated with fetal anomalies.

This project successfully demonstrates the application of machine learning techniques to analyze cardiotocography (CTG) data for fetal health classification. By integrating data preprocessing, exploratory analysis, model training, and performance evaluation into a streamlined pipeline, the system provides a reliable method for predicting fetal conditions as normal, suspect, or pathological. The use of algorithms like Logistic Regression, Random Forest, and XGBoost ensures a balance between interpretability and predictive accuracy. Furthermore, the development of a user-friendly GUI using Streamlit enhances the accessibility of the system, allowing healthcare professionals or researchers to interact with the model easily and obtain real-time results. Through careful testing and validation, the system has been shown to be accurate, efficient, and robust, making it a promising tool for supporting prenatal diagnostics. This project not only highlights the practical utility of machine learning in healthcare but also lays the foundation for future improvements such as real-time data integration, model explainability, and clinical deployment.

8.4 REFERENCES

1. G.Sedgh,S.Singh,andR.Hussain,“Intend edandunintended pregnancies worldwide in 2012 and recent trends,” Stud. Family Planning, vol. 45, no. 3, pp. 301–314, Sep. 2014.
2. C.J.Murray,“Global, regional, and national age-sex specific all-cause and cause-specific mortality for 240 causes of death, 1990–2013: A systematic analysis for the global burden of disease study 2013,” Lancet, vol. 385, no. 9963, 1990, Art. no. 117171.
3. (2016). World Health Organization, Maternal Mortality: Fact Sheet. [Online].
4. National Institutes of Health, What Are Some Common Complications of Pregnancy?U.S. Department of Health and Human Services. [Online].

5. Office on Women's Health, Pregnancy Complications US Department of Health and Human Services.
6. G. Magenes, R. Bellazzi, A. Malovini, and M. G. Signorini, "Comparison of datamining techniques applied to fetal heart rate parameters for the early identification of IUGR fetuses," in Proc. 38th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC), Orlando, FL, USA, Aug. 2016, pp. 916–919, doi: 10.1109/EMBC.2016.7590850.
7. N. M. Fisk and R. P. Smit, "Fetal growth restriction; small for gestational age," in Turnbull's Obstetrics, 3rd ed., G. Chamberlain and P. Steer, Eds. Edinburgh, Scotland: Churchill Livingstone, 2001, pp. 197–209.
8. S. C. R. Nandipati, "Classification and feature selection approaches for cardiotocography by machine learning techniques," J. Telecommun., Electron. Comput. Eng., vol. 12, no. 1, pp. 7–14, 2020.
9. S. Das, K. Roy, and C. K. Saha, "Establishment of automated technique of FHR baseline and variability detection using CTG: Statistical comparison with expert's analysis," Int. J. Inf. Eng. Electron. Bus., vol. 11, no. 1, pp. 27–35, Jan. 2019, doi: 10.5815/ijieeb.2019.01.04.
10. W. W. Stead, "Clinical implications and challenges of artificial intelligence and deep learning," J. Amer. Med. Assoc., vol. 320, no. 11, p. 1107, Sep. 2018, doi: 10.1001/jama.2018.11029.
11. K. H. Miao, J. H. Miao, and G. J. Miao, "Diagnosing coronary heart disease using ensemble machine learning," Int. J. Adv. Comput. Sci. Appl., vol. 7, no. 10, p. 3039, 2016.
12. M. G. Signorini, N. Pini, A. Malovini, R. Bellazzi, and G. Magenes, "Integrating machine learning techniques and physiology based heart rate features for antepartum fetal monitoring," Comput. Methods Programs Biomed., vol. 185, Mar. 2020, Art. no. 105015.
13. J. H. Miao, K. H. Miao, and G. J. Miao, "Breast cancer biopsy predictions based on mammo graphic diagnosis using support vector machine learning, Multidisciplinary Journals in Science and Technology," J. Sel. Areas Bioinf., vol. 5, no. 4, p. 19, 2015.
14. M. T. Alam, M. A. I. Khan, N. N. Dola, T. Tazin, M. M. Khan, A. A. Albraikan, and F. A. Almalki, "Comparative analysis of different efficient machine learning methods for fetal health classification," Appl. Bionics Biomechanics, vol. 2022, pp. 1–12, Apr. 2022.
15. A. Mehbodniya, A. J. P. Lazar, J. Webber, D. K. Sharma, S. Jayagopalan, K. K. P. Singh, R. Rajan, S. Pandya, and S. Sengan, "Fetal health classification from cardiotocographic data using machine learning," Expert Syst., vol. 39, no. 6, Jul. 2022, Art. no. e12899.
16. A. Kuzu and Y. Santur, "Early diagnosis and classification of fetal health status from a fetal cardiotocography dataset using ensemble learning," Diagnostics, vol. 13, no. 15, p. 2471, Jul. 2023.
17. P. Fergus, A. Hussain, D. Al-Jumeily, D.-S. Huang, and N. Bouguila, "Classification of caesarean section and normal vaginal deliveries using foetal heart rate signals and advanced machine learning algorithms," Biomed. Eng. OnLine, vol. 16, no. 1, pp. 1–26, Dec. 2017.

18. E.J. Quilligan and R.H. Paul, "Fetal monitoring : Is it worth it?" *Obstetrics Gynecol.*, vol. 45, no. 1, pp. 96–100, 1975.
19. F. G. Esposito, "Fetal heart rate monitoring and neonatal outcome in a population of early and late onset intrauterine growth restriction," *J. Obstetrics Gynaecology Res.*, vol. 45, no. 7, pp. 1343–1351, 2019.
20. A. Ugwumadu, "Are we (MIS) guided by current guidelines on intrapartum fetal heart rate monitoring? Case for a more physiological approach to interpretation," *BJOG, Int. J. Obstetrics Gynaecol.*, vol. 121, no. 9, pp. 1063–1070, 2014.
21. C. P. Anaruma, M. Ferreira, C. H. G. Sponton, M. A. Delbin, and A. Zanesco, "Heart rate variability and plasma biomarkers in patients with type 1 diabetes mellitus: Effect of a bout of aerobic exercise," *Diabetes Res. Clin. Pract.*, vol. 111, pp. 19–27, Jan. 2016.
22. F. Andreotti, J. Behar, S. Zaunseder, J. Oster, and G. D. Clifford, "An open source framework for stress-testing non-invasive foetal ECG extraction algorithms," *Physiological Meas.*, vol. 37, no. 5, pp. 627–648, May 2016.
23. G. D. Clifford, I. Silva, J. Behar, and G. B. Moody, "Non-invasive fetal ECG analysis," *Physiological Meas.*, vol. 35, no. 8, pp. 1521–1536, Aug. 2014.
24. T. P. Richardson, M. C. Peters, A. B. Ennett, and D. J. Mooney, "Polymeric system for dual growth factor delivery," *Nature Biotechnol.*, vol. 19, no. 11, pp. 1029–1034, Nov. 2001.
25. D. Maulik, *Doppler Ultrasound in Obstetrics and Gynecology*, I. Zalud, Ed. Berlin, Germany: Springer, 2005, pp. 363–374.