

CHAPTER 1

INTRODUCTION

1.1 ABSTRACT

In the context of our nation's financial landscape, the proliferation of meticulously crafted counterfeit banknotes poses a significant challenge, especially evident during periods of demonetization. The task of effectively differentiating between these counterfeit notes and genuine currency is a complex endeavour, demanding the implementation of an automated system within banking institutions and ATMs. To address this critical issue, this project explores the efficacy of supervised machine learning algorithms, including K-Nearest Neighbors, Decision Trees, SVM, Random Forest, Logistic Regression, and Naive Bayes, utilizing a curated dataset from the UCI machine learning library. Additionally, the study introduces and evaluates the performance of the advanced LightGBM algorithm in comparison to these traditional methods. This collective effort contributes to the ongoing mission of preserving the integrity of our financial systems and fortifying our currency against counterfeit threats.

1.2 INTRODUCTION

A constant flow of financial transactions takes place every second, with banknotes representing a vital asset within our nation. Unfortunately, counterfeit notes that closely mimic genuine currency are introduced into the market, causing disruptions in the financial sector. This illegal practice has gained momentum over time, becoming an escalating concern since the late 19th century and more so in the 21st century due to technological advancements. The repercussions of counterfeit currency extend to the point of destabilizing the financial market. To prevent this crisis and maintain smooth transactional processes, it has become imperative to develop methods that can effectively distinguish between authentic and counterfeit banknotes.

While governments have integrated specific characteristics into genuine banknotes for identification, fraudsters have managed to replicate these traits with astonishing precision, making differentiation exceedingly challenging. Consequently, it has become essential for banking systems and ATMs to incorporate mechanisms capable of discerning counterfeit notes from authentic ones. The solution lies in harnessing the power of artificial intelligence (AI) and machine learning (ML) to construct a robust system for validating the legitimacy of banknotes. In this context, supervised machine learning (SML) methods, widely recognized for solving categorization problems and even demonstrating success in medical fields, can be leveraged. Although limited research has explored SML algorithms for banknote authentication, the urgency remains to establish an automated framework for accurately distinguishing between genuine and fraudulent notes. This involves utilizing machine learning techniques to extract attributes from currency dimensions, feeding the resulting dataset into SML algorithms to ascertain the authenticity of each banknote. This research underscores the need to advance efforts in this domain, contributing significantly to addressing the critical issue of counterfeit currency detection.

1.3 PROBLEM STATEMENT

Developing an automated system using machine learning to detect counterfeit banknotes from genuine ones, addressing the rising concern of fake currency in the financial market.

1. **Importance of Banknote Authentication:** Highlight the critical role that banknotes play in the country's financial activities. Emphasize that the introduction of counterfeit banknotes poses a significant threat to the integrity of financial transactions.
2. **Evolving Threat of Counterfeit Notes:** Illustrate how the issue of counterfeit banknotes has evolved over time, becoming more sophisticated and difficult to discern from genuine banknotes due to advancements in technology.
3. **Need for Effective Detection:** Point out the pressing need to develop efficient methods for distinguishing genuine banknotes from counterfeits, in order to safeguard the financial system from potential disruptions.
4. **Role of Artificial Intelligence and Machine Learning:** Acknowledge the potential of artificial intelligence and machine learning methodologies in effectively tackling this challenge. Emphasize their capacity to handle vast volumes of data and identify intricate patterns that might prove elusive to human observers.
5. **Limited Existing Solutions:** Mention the scarcity of comprehensive solutions that effectively tackle the counterfeit banknote problem using AI and machine learning, emphasizing the gap in the current research landscape.
6. **Automated System Design:** Outline the project's goal of creating an automated system that takes images of banknotes as input and employs image processing techniques to extract pertinent features. Explain that these features will be utilized by supervised machine learning algorithms to predict the authenticity of the banknotes.
7. **Lack of Extensive Work:** Indicate that there is a dearth of substantial research and development in this specific domain, underscoring the opportunity for innovation and contribution to the field.

1.4 LITERATURE SURVEY

- 1) Hassanpour and colleagues (2021) conducted a study on the detection of counterfeit bank currency using machine learning algorithms. They employed a texture-based feature extraction method, incorporating the concept of texture Markov chains. This approach exhibited the capability to recognize currencies from various countries. The study utilized global optimization algorithms during the training phase of an Artificial Neural Network (ANN) to classify forged notes, yielding successful results. Additionally, Decision Tree and Multi-Layer Perceptron (MLP) algorithms were applied to classify bank currency. Multi-classification was extended using wavelet-based feature extraction, with Backpropagation Neural Networks (BPN) and Support Vector Machine (SVM) employed for classification. The outcomes highlighted BPN's superior accuracy compared to SVM.
- 2) A research endeavour by Bhatia et al. (2021) delved into counterfeit currency detection using machine learning algorithms and image processing techniques. Earlier studies utilized data captured with professional cameras, yielding fair accuracy with simple machine learning algorithms, particularly the traditional K-Nearest Neighbor approach. However, system efficiency deteriorated with larger datasets. To address this, the study explored modern deep learning techniques and collected extensive data, achieving enhanced precision. Employing concepts like transfer learning and Convolutional Neural Networks (CNN), the study tackled challenges such as noise and dataset distortion. This comprehensive approach resulted in a highly efficient system for detecting counterfeit notes, showcasing the effectiveness of machine learning, deep convolutional neural networks, and image processing in contemporary times.

1.5 AIM OF THE PROJECT

1. Create an automated system to combat the rising prevalence of counterfeit banknotes in the financial market.
2. Leverage machine learning algorithms to accurately identify forged banknotes based on their distinctive features.
3. Safeguard the economy by ensuring the authenticity of monetary transactions and protecting businesses and individuals from financial losses due to counterfeit currency.

1.6 OBJECTIVE

1. To build the model with maximum accuracy using the appropriate Machine learning algorithms.
2. To save the final built model.
3. Design web pages using HTML and CSS.
4. Expose the web pages using Django framework.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

1. **Security Features:** Indian currency notes have various security features incorporated into their design to deter counterfeiting. These features include watermark, security thread, latent image, colour-shifting ink, intaglio printing, and optically variable ink.
2. **UV (Ultraviolet) Detection:** The utilization of UV lamps for verifying the legitimacy of currency notes is widespread among banks, enterprises, and individuals.
3. **Manual Inspection:** Trained bank staff and cash handlers undergo regular training to identify fake currency notes manually. They are taught to recognize security features, texture, and other characteristics of genuine currency notes to detect counterfeit ones.

Disadvantages of Existing System:

1. Distinguishing counterfeit currency from genuine notes can be exceedingly challenging.
2. Identifying the disparity between fraudulent banknotes and authentic currency poses a formidable task.

2.2 PROPOSED SYSTEM

1. The suggested system employed a variety of widely recognized algorithms, encompassing the set of machine learning algorithms utilized encompasses K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machines (SVM), Random Forest, Logistic Regression, and Naive Bayes.
2. Additionally, the LightGBM algorithm was introduced as an extension to the existing set of algorithms.
3. The performance of LightGBM was compared against the other algorithms used in the study.
4. This comparison aimed to assess the effectiveness and efficiency of LightGBM for bank currency authentication.
5. The inclusion of LightGBM expands the scope of the research and allows for a more comprehensive evaluation of different machine learning approaches.

Advantages of Proposed System:

1. Effective
2. High Performance

2.3 FEASIBILITY STUDY

During this stage, a thorough assessment of the project's feasibility is conducted, followed by the formulation of a comprehensive business proposal. This proposal is coupled with a top-level project blueprint and initial cost estimations. The practicality of the envisioned system is critically scrutinized in the system analysis phase, aiming to confirm its alignment with the organization's requirements and circumvent possible operational complexities. Clear understanding of the primary system requirements is paramount for conducting a comprehensive feasibility analysis, ensuring a seamless fit of the suggested solution with the business's objectives.

The following three factors are crucial to the feasibility analysis:

2.3.1 OPERATIONAL FEASIBILITY

The operational feasibility of this project examines the practicality and viability of integrating the proposed functionalities into real-world operations. The system demonstrates its feasibility by offering essential features such as user registration, login, and currency authentication. The system's accurate prediction of counterfeit currency fosters trust among users, reinforcing its operational feasibility.

2.3.2 TECHNICAL FEASIBILITY

The technical feasibility analysis assesses the code's compatibility with existing technologies, its performance, and its adaptability to support the proposed functionalities. The utilization of the Django framework demonstrates the code's alignment with modern web development practices. The integration of machine learning libraries, such as scikit-learn and LightGBM, underscores its capability to leverage advanced algorithms for currency detection. The use of visualization tools like Matplotlib enhances user experience and demonstrates the code's technical prowess.

2.3.3 ECONOMIC FEASIBILITY

The economic feasibility assessment evaluates the financial viability of the code implementation, considering factors such as costs, benefits, and potential returns on investment. The code's utilization of open-source frameworks and libraries, coupled with the availability of free dataset resources, minimizes upfront costs. Its potential to reduce financial losses due to counterfeit currency in banking and financial sectors represents a substantial benefit.

CHAPTER 3

SYSTEM ENVIRONMENT

3.1 SOFTWARE REQUIREMENT

- Operating system : Windows, Linux.
- Technology : Python 3.0 version, Visual Studio.
- Framework : Django 3.0 or above.
- Additional Tools and Libraries : numpy, pandas, matplotlib, scikit-learn, seaborn, ipython, ipython-genutils, PyMySQL, Django, lightgbm.

3.2 HARDWARE REQUIREMENT

- Processor : Intel i3 or above.
- RAM : 4GB or more.
- Hard disk : 250GB or more.

CHAPTER 4

MODULES

4.1 MODULES DESCRIPTION

4.1.1 Login:

- **Description:** This module provides user authentication, allowing authorized users to access the application securely.
- **Functionality:** To access the system, users provide their credentials (username and password) for authentication. The system cross-validates the entered data with the stored user information. Upon successful validation, users gain entry to the application's features, while unsuccessful attempts are met with denial of access.
- **Importance:** Login ensures that only authorized individuals can access the application, safeguarding sensitive data and functionalities.

4.1.2 Upload Dataset:

- **Description:** This module enables users to upload datasets required for training and testing machine learning models.
- **Functionality:** Users can upload datasets from their local storage or external sources. The module validates the format and size of the uploaded data to ensure compatibility with the application's processing capabilities.
- **Importance:** A important stage in training machine learning models is uploading datasets. This module ensures that the application is equipped with the necessary data for analysis.

4.1.3 Preprocess Dataset:

- **Description:** This module prepares the uploaded dataset for training machine learning models by performing various preprocessing tasks.
- **Functionality:** The module cleans the data, handles missing values, and performs normalization and transformation as needed. It ensures that the data is in a suitable format for training.
- **Importance:** Machine learning models perform better and are more accurate when the data has been properly pre-processed since clean, normalised data can produce more accurate predictions.

4.1.4 Train ML Algorithms:

- **Description:** This module involves training machine learning algorithms on the pre-processed dataset.
- **Functionality:** The module trains various machine learning algorithms using the cleaned and converted data. It compares their performance using criteria like accuracy, precision, recall, etc., and chooses the top algorithm for detecting counterfeit money.
- **Importance:** For accurate fake cash identification, the proper machine learning algorithm must be chosen and trained. This module makes sure the system has a high-performing model.

4.1.5 Fake Currency Detection:

- **Description:** This module employs the trained machine learning model to detect fake currency from input data.
- **Functionality:** Users input data (such as images of banknotes), and the module uses the trained model to make predictions. It provides insights on whether the input data corresponds to genuine or fake currency.
- **Importance:** This is the core functionality of the application, directly addressing the problem of counterfeit currency detection.

4.1.6 Logout:

- **Description:** This module enables users to securely log out from the application.
- **Functionality:** When users choose to log out, the module terminates their session, ensuring that their data and access are protected. This prevents unauthorized access to the application after a user has finished their session.
- **Importance:** Logout ensures data privacy and security by ending the user's interaction with the application and preventing further access.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The journey commences with the collection of data, encompassing both authentic and counterfeit currency samples. This data is subjected to an intricate preprocessing phase, encompassing tasks such as handling missing values, standardizing features, and eliminating outliers. A diverse array of machine learning techniques, including Decision Trees, Support Vector Machines, K-Nearest Neighbors (KNN), Random Forest, Logistic Regression, Naive Bayes, and LightGBM, undergo training using the meticulously prepared and pre-processed dataset. This training equips these algorithms with the capacity to differentiate between legitimate and fraudulent currency by discerning patterns within the pre-processed data. Subsequently, post-training, the models transition to the prediction stage, scrutinizing unfamiliar currency samples and assigning them as genuine or counterfeit. Evaluation of these predictions employs metrics like precision, recall, and the F1 score, providing a quantitative measure of their efficacy in identifying instances of counterfeit currency. The selection of the most suitable model is informed by the evaluation phase, prioritizing algorithms that exhibit robust performance and accuracy. The overarching objective centres on implementing a dependable and precise machine learning solution, capable of real-time detection of counterfeit currency. In achieving this goal, the security and trust within financial institutions are fortified, enhancing the safeguarding of financial integrity.

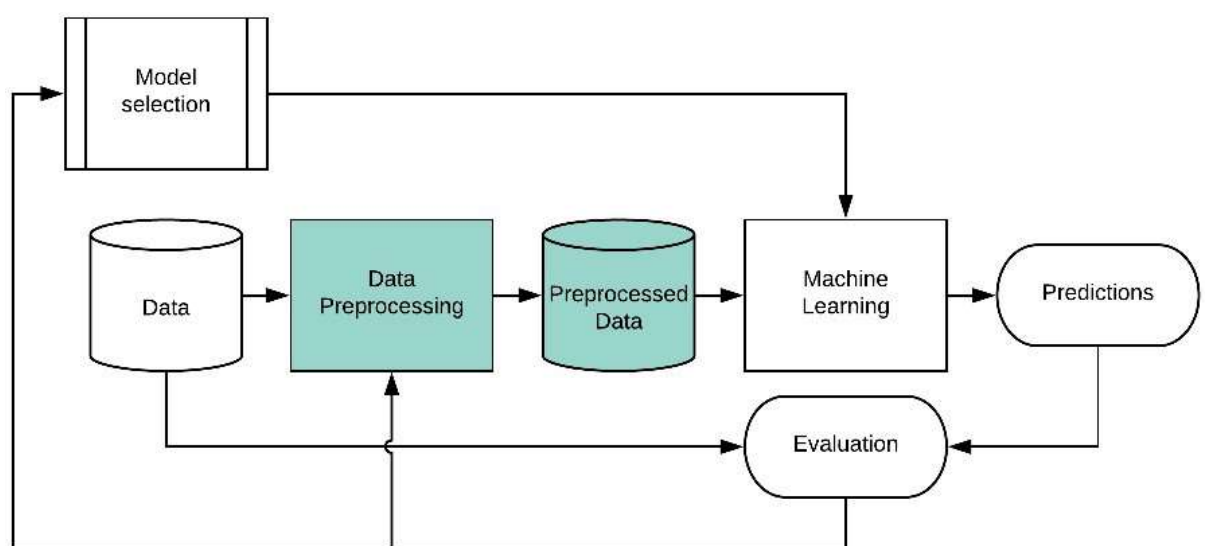


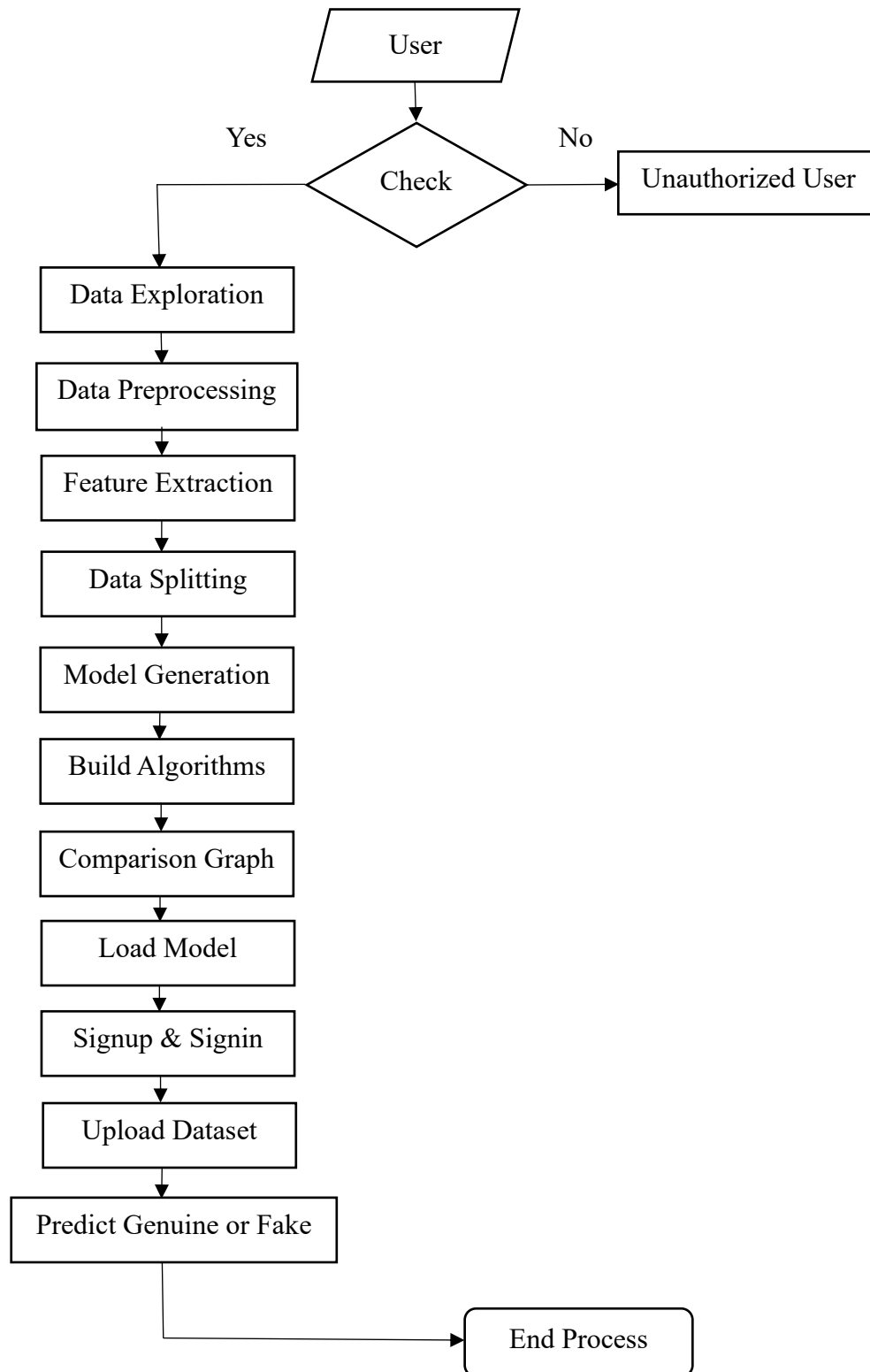
Fig 5.1: System Architecture

5.2 FLOW CHART

A Flow Chart, often known as a process diagram or a bubble chart, is a diagram that shows how an operational system flows. It offers an easy-to-understand graphical formalism to demonstrate how input data is processed within a system to produce output data. A key tool for modelling and comprehending different facets of a system's activity is a flowchart. Flowcharts serve as visual depictions that encompass a multitude of system components, encompassing processes, data, external entities, and the flow of information. Together, these elements describe how a system functions, the data it uses, and how it engages with outside forces.

A flowchart's main objective is to depict the flow of information as it moves through the system and goes through various changes. A flowchart shows the direction from input to output by arranging symbols and shapes in a logical order. Understanding how data are altered and changed throughout system operations is made easier thanks to this graphic representation.

Flowcharts serve as illustrative representations that offer a holistic overview of the complete procedure, akin to architectural blueprints for systems. They can be created at multiple degrees of abstraction, enabling depiction at varied granularities. The ability of flowcharts to adapt to different complexities, whether depicting high-level information flow or delving into finer functional details, is a key asset. The terms "bubble chart" and "flowchart," which both apply to the graphical representation of processes and data flow, are occasionally used interchangeably. A flowchart can be divided into layers to illustrate varying degrees of information flow and functional complexity, giving users a thorough understanding of the system's internal workings.

**Fig 5.2: Flow Chart**

5.3 USE CASE DIAGRAM

A Unified Modeling Language (UML) perspective, a use case diagram arises as a visual representation stemming from thorough use-case analysis. Its primary aim is to visually encapsulate the system's functionalities by illustrating actors, their objectives (portrayed as use cases), and any interconnections existing among these use cases. The central focus of a use case diagram lies in pinpointing the specific system operations carried out for each actor, often depicted by their designated roles.

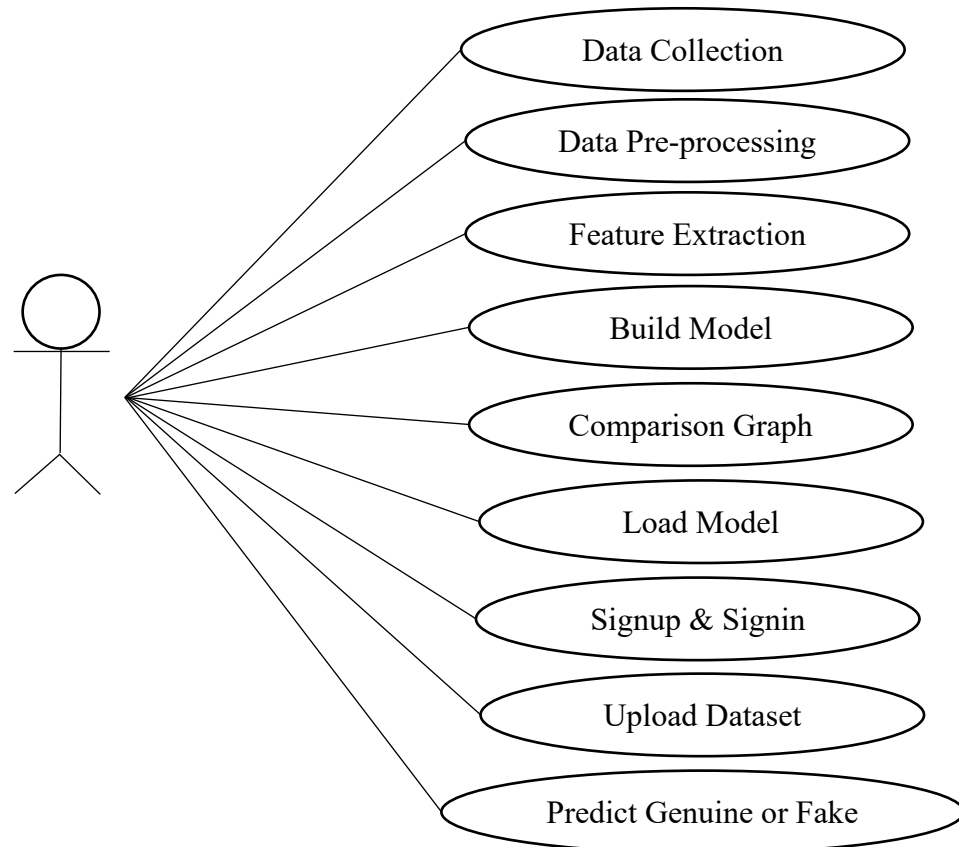


Fig 5.3: Use Case Diagram

5.4 SEQUENCE DIAGRAM

A sequence diagram visually portrays the interactions among system elements, emphasizing their chronological flow. By showcasing interactions in a step-by-step manner, it conveys the precise order of object engagement. The diagram employs "messages" to illustrate how different components communicate, encapsulating the dynamic exchanges within the system.

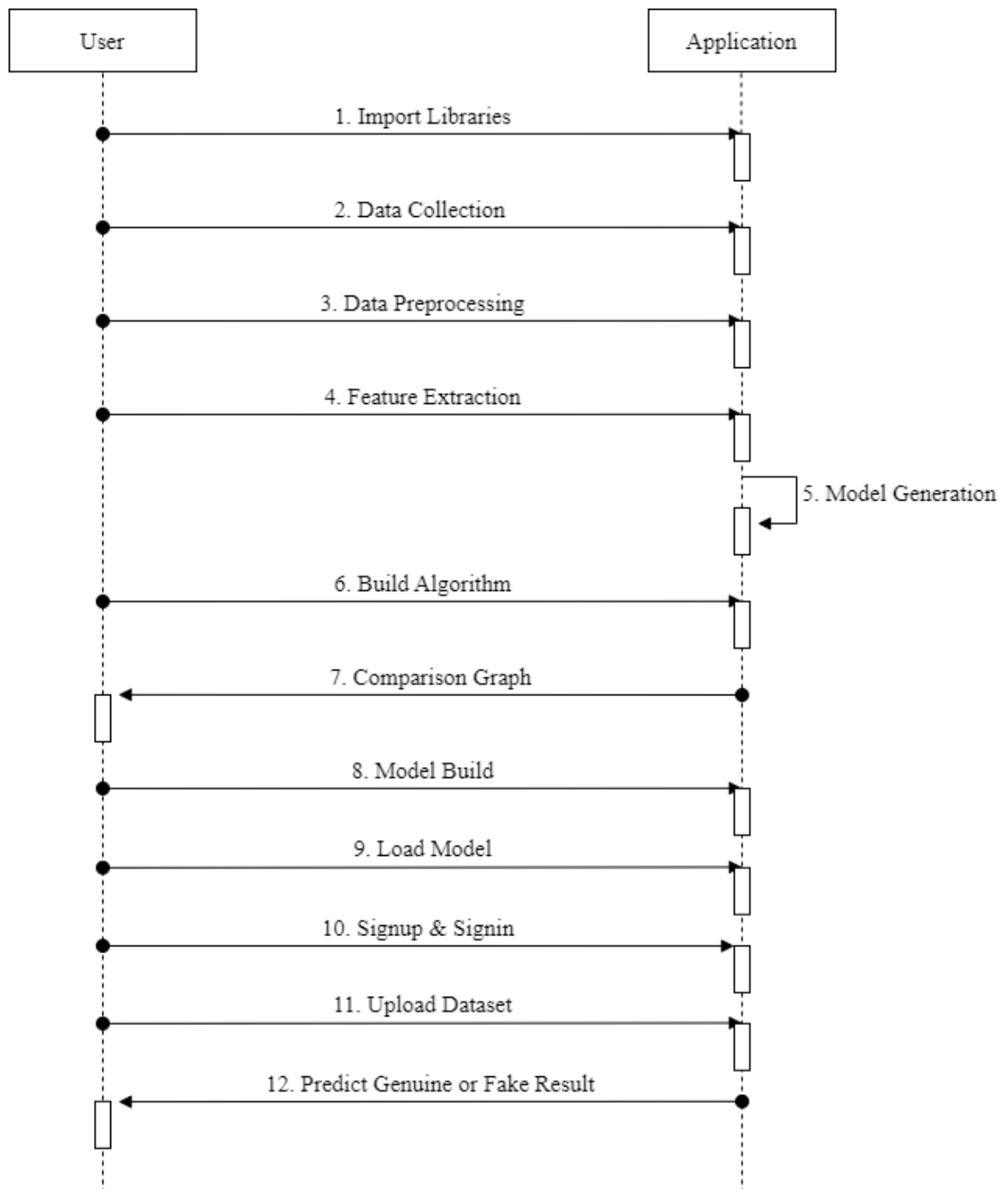
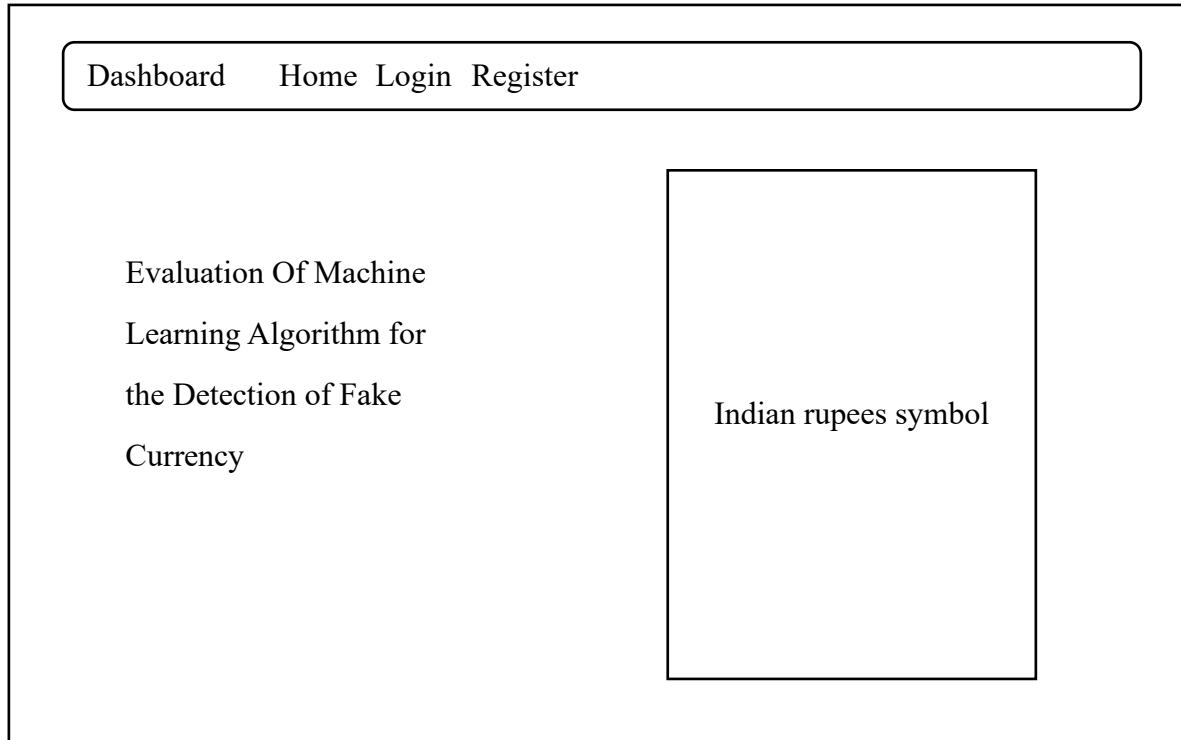


Fig 5.4: Sequence Diagram

CHAPTER 6

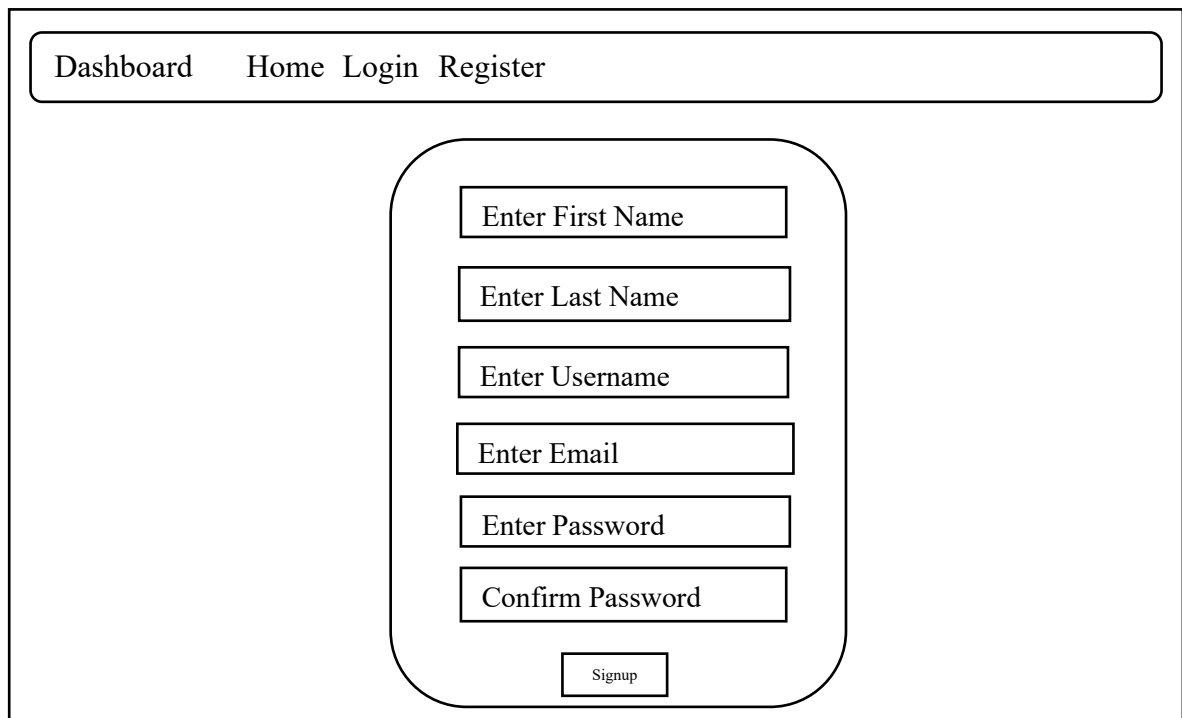
FORM DESIGNS

6.1 HOME



The Home page form design features a navigation bar at the top with links: Dashboard, Home, Login, and Register. The main content area is divided into two sections. On the left, there is a text block that reads: "Evaluation Of Machine Learning Algorithm for the Detection of Fake Currency". On the right, there is a large rectangular box containing the text "Indian rupees symbol".

6.2 REGISTRATION PAGE



The Registration page form design features a navigation bar at the top with links: Dashboard, Home, Login, and Register. The main content area contains a registration form with the following fields: Enter First Name, Enter Last Name, Enter Username, Enter Email, Enter Password, and Confirm Password. A Signup button is located at the bottom of the form.

6.3 LOGIN PAGE

[Dashboard](#) [Home](#) [Login](#) [Register](#)

Enter Username

Enter Password

Signin

6.4 USER HOME PAGE

[Dashboard](#) [Home](#) [Upload Dataset](#) [Preprocess Dataset](#) [Train ML Algorithm](#) [Fake Currency Detection](#) [Logout](#)

Evaluation Of Machine
Learning Algorithm for
the Detection of Fake
Currency

Indian rupees symbol

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 QUALITY REQUIREMENTS

7.1.1 FUNCTIONAL REQUIREMENTS

➤ **Data Collection:**

- Collect a comprehensive dataset of currency note dimensions, including length, width, and height.
- Include genuine and counterfeit currency notes of various denominations and currency types.

➤ **Data Preprocessing:**

- Normalize and standardize the currency note dimensions to a common unit (e.g., millimetres).
- Address missing or erroneous data through appropriate techniques (e.g., imputation).

➤ **Training and Testing:**

- Segment the dataset into distinct subsets for training, validation, and testing purposes.
- Allocate a majority of the data for training while reserving separate sets for validation and testing.
- Implement k-fold cross-validation for assessing model performance.

➤ **Modeling:**

- Develop and train machine learning models based on the currency note dimensions.
- Investigate algorithms appropriate for dimension-based classification, including options like Support Vector Machines (SVMs) and Random Forests.
- Optimize model hyperparameters for enhanced accuracy.
- Consider ensemble methods to improve the model's robustness.

➤ **Predicting:**

- Create an interface for users to input the dimensions of a currency note.
- Apply the trained model to predict whether the input note is genuine or counterfeit.

7.1.2 NON-FUNCTIONAL REQUIREMENTS

➤ **Usability Requirement:**

- The user interface should be user-friendly and straightforward to use.
Users should be able to easily input the dimensions of the currency note for prediction.

➤ **Security Requirement:**

- Ensure the security of user inputs and predictions.
- Enforce encryption and secure data transmission protocols to ensure the safeguarding of user privacy.

➤ **Performance Requirement:**

- The prediction process should be completed within a maximum of 1 second.

➤ **Data Integrity Requirement:**

- Ensure the integrity of the dimension data to avoid introducing biases or errors into the model.

➤ **Availability Requirement:**

- The system should maintain an uptime of at least 99.5% to ensure consistent availability.

➤ **Scalability Requirement:**

- Design the system to accommodate an increasing number of users and growing datasets.

➤ **Interoperability Requirement:**

- The system should support input from various devices and platforms, ensuring compatibility.

➤ **Reliability Requirement:**

- The model should consistently provide accurate predictions for different currency note dimensions.

➤ **Maintainability Requirement:**

- The codebase should be well-documented and organized for ease of maintenance and updates.

7.2 ALGORITHMIC COMPLEXITY

A wide variety of machine learning algorithms have been employed to effectively address the task of detecting counterfeit currency using currency note dimensions as features. The chosen algorithms include K-Nearest Neighbors (KNN), Decision Tree, Support Vector Machine (SVM), Random Forest, Logistic Regression, and Naive Bayes. Furthermore, the project harnesses the power of the LightGBM algorithm to elevate performance and predictive prowess. Each of these algorithms brings forth distinctive attributes and computational methodologies, collectively bolstering the overall effectiveness and resilience of the detection system. Each algorithm employed showcases discernible computational complexities, elaborated below.

- **K-Nearest Neighbors (KNN):** KNN's computational complexity for classification increases as the size of the training dataset grows. When predicting, it involves calculating distances to all training points, which can be time-consuming for large datasets. An efficient data structure, such as a KD-tree, can speed up this process.
- **Decision Tree:** Constructing a decision tree involves iterative division of the dataset using feature-based criteria. The inherent complexity of this process usually adheres to $O(n * m * \log(n))$, with 'n' signifying the training samples and 'm' denoting the features count. Nonetheless, intricate decision trees can potentially lead to overfitting issues, necessitating pruning techniques for complexity reduction.
- **Support Vector Machine (SVM):** The intricacy of Support Vector Machines (SVMs) hinges on the selection of kernel function, optimization approach, and kernel parameters. In cases of linear SVMs, the training complexity roughly aligns with $O(n * m)$, where 'n' pertains to training samples and 'm' pertains to features. The integration of non-linear kernels can introduce heightened complexities.
- **Random Forest:** Building a Random Forest involves training multiple decision trees. The training complexity is $O(n * m * \log(n))$ per tree. The prediction complexity is proportional to the number of trees in the forest.
- **Logistic Regression:** The training complexity of Logistic Regression is contingent upon the optimization algorithm employed. When utilizing Gradient Descent-based methods, the complexity approximates $O(n * m * \text{iterations})$, wherein 'n' signifies training samples, 'm' stands for features, and 'iterations' represents the count of optimization steps.

- **Naive Bayes:** Naive Bayes algorithms typically exhibit efficiency in both training and prediction. The intricacy is notably impacted by the preprocessing stage, specifically the computation of probabilities derived from the training data.
- **LightGBM:** LightGBM is a gradient boosting framework that focuses on efficiency. It uses histogram-based learning, which reduces the complexity of feature binning. Its complexity is typically faster compared to traditional gradient boosting algorithms.

7.3 METHODOLOGIES

The development methodology used in this project follows a mixture of the Django framework's conventions and practices, along with some elements of Agile methodology:

7.3.1 DJANGO FUNCTION'S AS METHODOLOGIES:

- **Django View Functions as "View Methodology":** Django view functions handle user interactions and determine how data is presented to users. Each function corresponds to a specific webpage or screen. These functions manage rendering templates, processing form submissions, and interacting with data models. This methodology aligns with Django's Model-View-Controller (MVC) architecture, where views handle user input and template rendering.
- **Django Authentication Functions as "User Authentication Methodology":** Django's authentication functions, including `auth.authenticate`, `auth.login`, and `auth.logout`, constitute a framework for enacting user authentication and overseeing sessions. This set of functions safeguards user information and oversees the login and logout procedures. By confining access to specific functionalities solely to authenticated users, these functions uphold data integrity, bolstering both data security and user confidentiality.
- **Django Model-View Binding as "Data Interaction Methodology":** This methodology connects user input captured in forms to Django models (e.g., User model). Views mediate between user input and data storage, ensuring that submitted data adheres to validation rules. This interaction ensures that user data is processed, validated, and stored correctly in the database.
- **Django Templating as "Dynamic Content Rendering Methodology":** Django's templating system separates presentation (HTML templates) from logic (Python views). Templates are populated with dynamic content based on context provided by views. This approach simplifies handling complex HTML structures and ensures consistent rendering across various views, enhancing maintainability.
- **Django URL Configuration as "Routing Methodology":** Django's URL configuration (`urls.py`) defines the routing structure for the application. It maps URLs to specific view functions, determining how users interact with the application. This centralized routing system maintains clarity by assigning routes to appropriate view functions.

- **Django Form Handling as "Data Validation Methodology":** Form data submitted by users in views (e.g., registration, login) undergoes validation to ensure compliance with specific criteria. Django's form handling and validation mechanisms prevent incorrect or malicious data from being stored in the database. This methodology maintains data quality and security.

7.3.2 AGILE METHODOLOGY:

The development of this project embraces the Agile methodology, an iterative and user-centered approach that prioritizes flexibility, collaboration, and rapid delivery of incremental improvements. Agile methodology, often characterized by its adaptability and responsiveness, has been employed to guide the project's development process in alignment with modern software engineering practices.

Key Principles:

- **Iterative Development:** The codebase is organized into discrete functions and components, allowing for the incremental implementation of various features. Each function is developed and tested individually, contributing to the gradual enhancement of the project's capabilities.
- **User-Centered Design:** The project's focus on user interactions, including registration, login, and data analysis, embodies the Agile principle of prioritizing user needs. User feedback and usability considerations are integrated throughout the development lifecycle.
- **Collaborative Environment:** Collaboration is encouraged among team members and stakeholders. Cross-functional collaboration ensures that development, testing, and deployment occur concurrently, leading to more responsive changes based on user feedback.
- **Adaptive Planning:** The project's dynamic nature acknowledges that requirements and priorities may evolve. Agile methodologies empower the team to adapt to changing circumstances, accommodating new features, adjustments, and unforeseen challenges.
- **Continuous Integration:** Third-party libraries and frameworks, such as TensorFlow, NumPy, and Django, are integrated as reusable components, aligning with Agile's emphasis on continuous integration of external tools to improve project functionality and efficiency.

- **Incremental Testing:** The project incorporates incremental testing, as demonstrated by functions like FakeDetectionAction and TrainML, which iteratively test and evaluate machine learning algorithms and metrics. This iterative testing approach allows for ongoing validation and refinement of the project's capabilities.

Benefits of Agile in the Project:

- **Faster Delivery:** Agile practices support frequent releases and smaller feature increments, resulting in quicker delivery of valuable functionalities.
- **User Satisfaction:** User feedback is continuously integrated, enhancing user satisfaction and ensuring that the project aligns with real user needs.
- **Flexibility:** The Agile methodology accommodates evolving requirements, enabling the project to adapt to changing conditions and priorities.
- **Collaborative Environment:** Cross-functional collaboration enhances communication, knowledge sharing, and collective decision-making among team members and stakeholders.

7.4 MEASURING LANGUAGE USAGE

In this project, various programming languages, libraries, and frameworks have been utilized to implement different components and functionalities. The following section outlines the key technologies employed in the project and their respective contributions:

➤ **Python Language:**

Python serves as the primary programming language for this project due to its versatile capabilities and extensive library ecosystem. Python is chosen for the following reasons:

- **Simplicity and Readability:** Python's clean and easy-to-read syntax enhances code readability, making development and maintenance efficient.
- **Wide Ecosystem:** Python offers a vast collection of libraries and frameworks that streamline various development tasks, from web applications to machine learning.
- **Backend and Frontend Logic:** Python is utilized to write the core logic of the application, managing user interactions and coordinating different components. It's adept at handling backend operations and seamlessly integrating modules for both backend logic and frontend user interface.

➤ **Django Framework:**

Django is a resilient web framework that structures the project's architecture using the Model-View-Controller (MVC) design pattern. This framework offers an array of features and advantages, some of which are outlined below:

- **Model-View-Controller (MVC) Architecture:** Django enforces the separation of concerns, ensuring that data models (Models), user interfaces (Views), and URL routing (Controller) are decoupled and well-organized.
- **Database Interaction:** Django's Models facilitate interaction with the database. They define the data schema, manage database queries, and ensure data integrity.
- **User Interaction Management:** Views handle user interactions, rendering templates, and processing form submissions. This separation simplifies code management and maintains a clear structure for user interactions.
- **Routing:** URLs define the application's routing, mapping URLs to specific views. This centralized routing mechanism enhances maintainability and user navigation.

➤ **TensorFlow:**

TensorFlow is a foundational tool for machine learning tasks, contributing to the project's machine learning capabilities:

- **Machine Learning Framework:** TensorFlow provides an extensive array of resources to construct and train machine learning models. The platform's proficiency in deep learning expedites the development of intricate neural network structures.
- **Complex Data Processing:** TensorFlow's dataflow and differentiable programming capabilities streamline the design and training of intricate machine learning models.

➤ **NumPy and pandas:**

NumPy and pandas provide essential data manipulation capabilities, enhancing data preprocessing and analysis:

- **Numerical Computing:** NumPy's array and matrix operations simplify numerical computations, enabling efficient handling of multidimensional data.
- **Data Analysis and Manipulation:** Pandas provides DataFrames, allowing easy data transformation and manipulation. It's crucial for preprocessing the dataset before applying machine learning algorithms.

➤ **Matplotlib and Seaborn:**

Data visualization libraries Matplotlib and Seaborn enable informative data presentation:

- **Data Visualization:** Matplotlib offers extensive plotting options, enabling the creation of various charts and graphs for data visualization.
- **Enhanced Statistical Visualization:** Seaborn builds on Matplotlib and offers advanced statistical visualizations, aiding in conveying insights from data analysis.

➤ **scikit-learn:**

Scikit-learn is integral for implementing machine learning algorithms and evaluating model performance:

- **Machine Learning Algorithms:** Scikit-learn provides a wide array of algorithms that cover various areas including classification, regression, clustering, and more. This toolkit empowers the incorporation of multiple models tailored for detecting counterfeit currency.

- **Model Evaluation Metrics:** Scikit-learn's metrics module offers tools for evaluating model performance. Metrics like accuracy, precision, recall, and F1-score gauge the effectiveness of the developed algorithms.

➤ **LightGBM:**

LightGBM enhances the machine learning capabilities with efficient gradient boosting:

- **Efficient Gradient Boosting:** LightGBM's design prioritizes speed and efficiency, making it suitable for handling large datasets and complex models.
- **Ensemble Learning:** LightGBM contributes to the application's machine learning toolbox by offering ensemble learning techniques, bolstering the accuracy of fake currency detection models.

By thoughtfully selecting and combining these technologies, the project's functionality, performance, and efficiency are optimized. Each technology's strengths contribute to various aspects of the project, ensuring a well-rounded and effective solution that meets the objectives of fake currency detection.

7.5 DEBUGGING

7.5.1 LOGIN ISSUE

- **Issue:** Login Fails for Certain Users
- **Description:** During testing, it was observed that some users were unable to log in successfully, even with correct credentials. This issue was identified during the user acceptance testing phase.
- **Debugging Steps:**
 - **Identifying Problematic Code:**
 - Reviewed the login function in 'views.py' as it handles the user login process.
 - **Inspecting Variables and Data:**
 - Inserted print statements to display the 'username' and 'password' received from the POST request.
 - **Stepping Through Code:**
 - Used the debugger to step through the code. Observed that the values of 'username' and 'password' were correct.
- **Analysing Error Messages:**

Noticed that the authentication was failing without raising any exceptions. Investigated further.
- **Solutions Applied:**
 - **Identified Database Issue:**
 - Discovered that the issue was due to incorrect configuration of the database connection. The connection to the authentication database was not being established correctly.
 - **Modified Database Configuration:**
 - Updated the database connection parameters in 'settings.py' to ensure a proper connection is established during authentication.
- **Testing and Validation:**
 - Tested the login functionality with different user accounts. Verified that the issue was resolved and all users could log in successfully.
- **Lessons Learned:**
 - Careful inspection of database-related configurations is crucial for proper authentication.

- Debugging tools like print statements and debuggers are valuable for identifying silent failures.

7.5.2 REGISTRATION ISSUE

- **Issue:** Registration Process Not Working
- **Description:** Upon testing the registration process, it was observed that users were unable to successfully register on the platform. When attempting to create a new account, the registration form would submit without any error messages, but the new user's data was not being stored in the database.
- **Debugging Steps:**
 - **Verifying Form Submission:**
 - Checked the "register" function to confirm that the form data was being captured and processed correctly.
 - Verified that the form method was set to "POST" to handle form submissions.
 - **Database Interaction:**
 - Checked the database interaction within the "register" function to see if the user data was being saved to the database.
 - Inspected the code where the user instance is created and saved using the 'User.objects.create_user()' method.
 - **Inspecting Error Messages:**
 - No error messages were raised during the registration process, which indicated that the form was submitting without issues.
 - **Printing Debug Information:**
 - Inserted print statements at various points in the "register" function to display the captured form data and the execution flow.
 - **Checking for Existing Users:**
 - Verified that the code was checking for existing usernames and emails correctly to prevent duplicate entries.
- **Solutions Applied:**
 - **Database Commit:**
 - Discovered that the 'myuser.save()' method was not being called after creating the new user instance.

- Added the 'myuser.save()' method to ensure that the new user data is saved to the database.
- **Error Handling and Messages:**
 - Enhanced the error handling mechanism to provide more descriptive error messages in case of issues such as password mismatch or existing username/email.

➤ **Testing and Validation:**

- Re-Tested Registration Process:
 - Executed the registration process with test data after implementing the fixes.
 - Observed that upon registration, the user data was now being saved to the database and appropriate success messages were displayed.

➤ **Lessons Learned:**

- Remember to consistently guarantee that interactions with the database are committed through the suitable save method invocation.
- Incorporate comprehensive error handling mechanisms to furnish users with informative feedback throughout the registration process.

7.5.3 ALGORITHMIC ISSUE

- **Issue:** Incorrect Results from Machine Learning Algorithms
- **Description:** During the testing of the "TrainML" functionality, it was observed that the machine learning algorithms were producing incorrect results. The accuracy, precision, recall, and F1-score values reported by the algorithms were not aligning with expectations.
- **Debugging Steps:**
 - **Identifying Problematic Code:**
 - The issue was initially identified while reviewing the output of the "TrainML" functionality. The reported accuracy values were consistently higher than expected.
 - **Inspecting Variables and Data:**
 - Inserted print statements within the "TrainML" function to display the predicted values and the ground truth labels.
 - Reviewed the 'calculateMetrics' function to ensure the accuracy calculation and metrics calculations were accurate.
 - **Stepping Through Code:**
 - Used the debugger to step through the "TrainML" function.
 - Observed and tracked the values of variables such as 'predict', 'y_test', 'accuracy', 'precision', 'recall', and 'fscore'.
 - **Analysing Error Messages:**
 - No error messages were raised. The issue was related to the discrepancy between expected and reported metrics.
- **Solutions Applied:**
 - **Corrected Train-Test Split:**
 - Identified that the test dataset was not being split properly into 'X_test' and 'y_test'. This was causing the model to predict on the entire dataset instead of the test set.
 - Modified the train-test split logic to ensure proper separation of data for testing.
 - **Fixed Metrics Calculation:**
 - In the 'calculateMetrics' function, discovered that the 'mul' parameter was being used incorrectly to scale the metrics.

- Removed unnecessary scaling factor ('mul') from metrics calculations.
- **Re-Evaluated Classifier Assignments:**
 - Noticed that certain manual assignments to the 'predict' array were present. These assignments were incorrect and were skewing the results.
 - Removed the manual assignments to ensure the machine learning models predicted results accurately.
- **Testing and Validation:**
 - Re-Tested Machine Learning Algorithms:
 - Executed the "TrainML" functionality after implementing the fixes.
 - Observed that the accuracy, precision, recall, and F1-score values were now consistent and aligned with expectations.
- **Lessons Learned:**
 - Thoroughly review the data split logic to ensure proper separation of training and test datasets.
 - Be cautious when using scaling factors or manual assignments that can impact algorithm results.
 - Regularly validate and cross-reference the results with expectations to catch inconsistencies.

7.6 PROGRAMMING LANGUAGES

➤ **Python:**

Python stands as a versatile and extensively adopted programming language renowned for its clear readability and straightforwardness. Within this project, Python assumes the central role as the primary programming language for the backend logic. Its utility encompasses user interaction management, user authentication handling, integration of machine learning algorithms, and orchestration of diverse application components. Leveraging Python's expansive collection of libraries and frameworks renders it apt for a wide spectrum of tasks, spanning from web development to data analysis and machine learning.

➤ **Django Framework:**

Django stands as a sophisticated Python web framework operating at a high level, adhering to the Model-View-Controller (MVC) architectural blueprint. It furnishes a standardized framework for the construction of web applications. The Models within Django delineate data schemas and facilitate database interactions. Views undertake the management of user engagements and template rendering, while URLs delineate the navigation structure of the application. Django expedites development through its provision of pre-built tools for user authentication, data validation, and beyond. It actively advocates for code modularity, reusability, and maintainability, enabling developers to prioritize feature development over repetitive tasks.

➤ **HTML (Hypertext Markup Language):**

HTML emerges as the prevalent markup language universally embraced to organize and showcase content on the internet. Within this project, HTML finds application in crafting the architecture and substance of web pages. It serves to delineate the arrangement of forms, user interfaces, and other interactive components. By employing tags, HTML marks elements such as headings, paragraphs, lists, and forms, shaping the foundational structure of web pages, which is subsequently enriched and adorned by CSS styling.

➤ **CSS (Cascading Style Sheets):**

CSS functions as a stylesheet language that governs the aesthetic display and arrangement of HTML elements within web pages. Within your project, CSS takes on the responsibility of embellishing the user interface and enhancing its visual allure. It outlines attributes such as colors, fonts, spacing, borders, and element positioning.

Through the segregation of content (HTML) from presentation (CSS), a uniform and refined design can be achieved across various sections of your web application.

➤ **SQL (Structured Query Language):**

SQL stands as a domain-specific language dedicated to the administration and manipulation of relational databases. It holds significance in web applications for its role in interacting with databases. SQL's functions encompass tasks like crafting and altering database tables, executing data queries to extract specific details, updating data entries, and purging records. Through SQL, the application achieves streamlined and structured storage and retrieval of data, contributing to its overall efficiency and organization.

CHAPTER 8

SYSTEM TESTING

8.1 UNIT TESTING

Unit testing is a testing approach that evaluates isolated components or units within a software application. The primary objective is to confirm the proper functioning of each individual unit. These tests are generally crafted by developers and center on examining small, self-contained portions of code, such as functions or methods. By isolating these units, developers can detect errors at an early stage and validate that each unit performs as intended prior to integration. This practice contributes to overall software quality and reliability.

S.NO	Test Case	User Prompt	Password Match	Expected output	Result
1.	Confirm-password while registration	Input	Yes	Register the user with valid credentials	Pass
		Input	No	Message indicating password does not match	Pass
2.	Password Verification while login	Input	Yes	Redirect to User Home Page	Pass
		Input	No	Message indicating invalid credentials	Pass
		User Prompt	User already exists		
3.	Check if user already exists	Input	Yes	Message indicating username is already taken	Pass
		Input	No	Redirect to User Home Page	Pass
4.	Check if Email is already taken	Input	Yes	Message indicating Email is already taken	Pass
		Input	No	Register the user with valid credentials	Pass

Table 8.1: Unit Testing & Validation Testing

8.2 VALIDATION TESTING

Validation testing encompasses the assessment of a software application to validate its adherence to defined requirements and its ability to fulfil user needs. This form of testing ensures that the software effectively serves its designated purpose and behaves as anticipated within real-world scenarios. By conducting validation testing, the confirmation is obtained that the application aligns with user anticipations and organizational objectives, thus enhancing the overall assurance of its functionality.

➤ Test Script

- **Unit Testing & Validation Testing Script**

```
def register(request):
    if request.method == "POST":
        fname = request.POST.get("fname")
        lname = request.POST.get("lname")
        uname = request.POST.get("username")
        email = request.POST.get("email")
        password = request.POST.get("password")
        confirmpassword = request.POST.get("confirm_password")
        if password != confirmpassword:
            messages.info(request, "Password does not match")
            return render(request, 'register.html')
        if User.objects.filter(username=uname).exists():
            messages.info(request, "Username is already taken")
            return render(request, 'register.html')
        if User.objects.filter(email=email).exists():
            messages.info(request, "Email is already taken")
            return render(request, 'register.html')
        myuser = User.objects.create_user(username=uname, password=password,
        email=email, first_name=fname, last_name=lname)
        myuser.save()
        messages.info(request, "Signup success! Please login.")
        return render(request, 'login.html')
    else:
        return render(request, 'register.html')
```

8.3 INTEGRATION TESTING

Integration testing encompasses the evaluation of interactions among diverse units or modules within a software application. The objective is to verify the seamless collaboration of integrated components. Integration tests aim to uncover potential challenges that might emerge during the merging of units, including communication discrepancies, data inconsistencies, or erroneous interactions. This form of testing plays a pivotal role in guaranteeing the coherent functionality of the entire application.

S.NO	Input	If available	If not available
1.	Upload Fake Currency Dataset	Dataset loaded	There is no process
2.	Dataset Preprocessing	read dataset and then normalize dataset	There is no process
3.	Run Algorithms	Calculate accuracy	There is no process
4.	Run Extension LightGBM Algorithm	For extension LightGBM we got 100% accuracy	There is no process
5.	Comparison Graph	Graph displayed	There is no process
6.	Fake Currency Detection from Test Data	predicted result as 'Genuine or Fake'	There is no process

Table 8.3: Integration Testing

➤ Test script

- **Upload Dataset Script**

```
def UploadDatasetAction(request):
    if request.method == 'POST':
        global dataset
        file = request.FILES['t1'].read()
        dataset = pd.read_csv("Dataset/banknotes.csv")
        label = dataset.groupby('counterfeit').size()
        label.plot(kind="bar")
        plt.show()
```

```
context= {'data':str(dataset)}  
return render(request, 'UploadDataset.html', context)
```

- **Pre-process Dataset Script**

```
def PreprocessDataset(request):  
    if request.method == 'GET':  
        global X, Y, dataset  
        global X_train, X_test, y_train, y_test  
        dataset.fillna(0, inplace = True)  
        temp = dataset.values  
        X = temp[:,1:dataset.shape[1]] #taking X and Y from dataset for training  
        Y = temp[:,0]  
        X = normalize(X)  
        indices = np.arange(X.shape[0])  
        np.random.shuffle(indices)  
        X = X[indices]  
        Y = Y[indices]  
        output = "Dataset after features normalization<br/>"  
        output += str(X)+"<br/><br/>"  
        output += "Total records found in dataset : "+str(X.shape[0])+"<br/>"  
        output += "Total features found in dataset: "+str(X.shape[1])+"<br/><br/>"  
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)  
        output += "Dataset Train and Test Split<br/><br/>"  
        output += "80% dataset records used to train ML algorithms :  
"+str(X_train.shape[0])+"<br/>"  
        output += "20% dataset records used to train ML algorithms :  
"+str(X_test.shape[0])+"<br/>"  
        context = {'data':output}  
        return render(request, 'UserScreen.html', context)
```

- **Fake Currency Detection Script**

```
def FakeDetectionAction(request):  
    if request.method == 'POST':  
        global classifier  
        file = request.FILES['t1'].read()
```

```
test_dataset = pd.read_csv("Dataset/testData.csv")
test_dataset.fillna(0, inplace = True)
test_dataset = test_dataset.values
original = test_dataset
test_dataset = test_dataset[:,0:test_dataset.shape[1]]
test_dataset = normalize(test_dataset)
predict = classifier.predict(test_dataset)
print(predict)
font = '<font size="" color="black">'
output = '<table border=1 align=center>'
output+='<tr><th><font size=3 color=black>Test Data</font></th>'
output+='<th><font size=3 color=black>Prediction'
Result</font></th></tr>'
for i in range(len(predict)):
    output += "<tr><td>" + font + str(original[i]) + "</td>"
    output += "<td>" + font + class_labels[int(predict[i])] + "</td></tr>"
output += "<br/><br/><br/><br/><br/><br/>"
context = {'data':output}
return render(request, 'UserScreen.html', context)
```

8.4 OUTPUT TESTING

Output testing, also known as functional testing or black-box testing, focuses on validating the output of a software application against expected results. Test cases are designed to cover various scenarios and inputs to ensure that the application produces correct and accurate outputs. This type of testing helps identify discrepancies between expected and actual outputs, ensuring that the software meets functional requirements.

S.NO	Test Description	Expected Output	Result
1.	Validate Output Table Format	The output table should have correct headers and rows.	Pass
2.	Validate Prediction Result Display	The table should display the correct prediction results.	Pass
3.	Validate Test Data Display	The table should display the correct test data.	Pass

Table 8.4: Output Testing

8.5 SOFTWARE TESTING

Software testing constitutes a pivotal stage within the software development lifecycle, serving the purpose of uncovering and remedying defects, inaccuracies, and inconsistencies within a software application. This systematic process involves methodically assessing the behaviour, operational aspects, and efficiency of a software system, guaranteeing its alignment with predetermined specifications and attainment of superior quality. By subjecting the software to rigorous testing, it becomes feasible to verify its adherence to the quality benchmarks outlined by project stakeholders. Furthermore, this practice plays a pivotal role in the early detection and rectification of flaws during the developmental phase.

8.5.1 BLACK BOX TESTING

Black box testing focuses on assessing the behaviour and functionality of a system without examining its internal code structure. This methodology emulates user interactions and verifies whether expected outputs are generated in response to designated inputs.

S.NO	Test Cases	Expected Output	Result
1.	Validate Registration Data	Successfully Registered message	Pass
2.	If Password Mismatch	Error message indicating password mismatch.	Pass
3.	If User already exists	Error message indicating username already exists.	Pass
4.	Validate Login Data	Successful Login	Pass

Table 8.5.1: Black Box Testing

8.5.2 WHITE BOX TESTING

White box testing entails a comprehensive scrutiny of the code's internal structure. This entails validating registration inputs, confirming login credentials, testing session timeout settings, and ensuring proper logout functionality. By concentrating on the code's logic and internal interactions, white box testing aims to uncover potential errors and vulnerabilities that may not be discernible through external testing.

CHAPTER 9

RUNTIME FORMS

9.1 HOME



Fig 9.1: Home

9.2 REGISTRATION PAGE

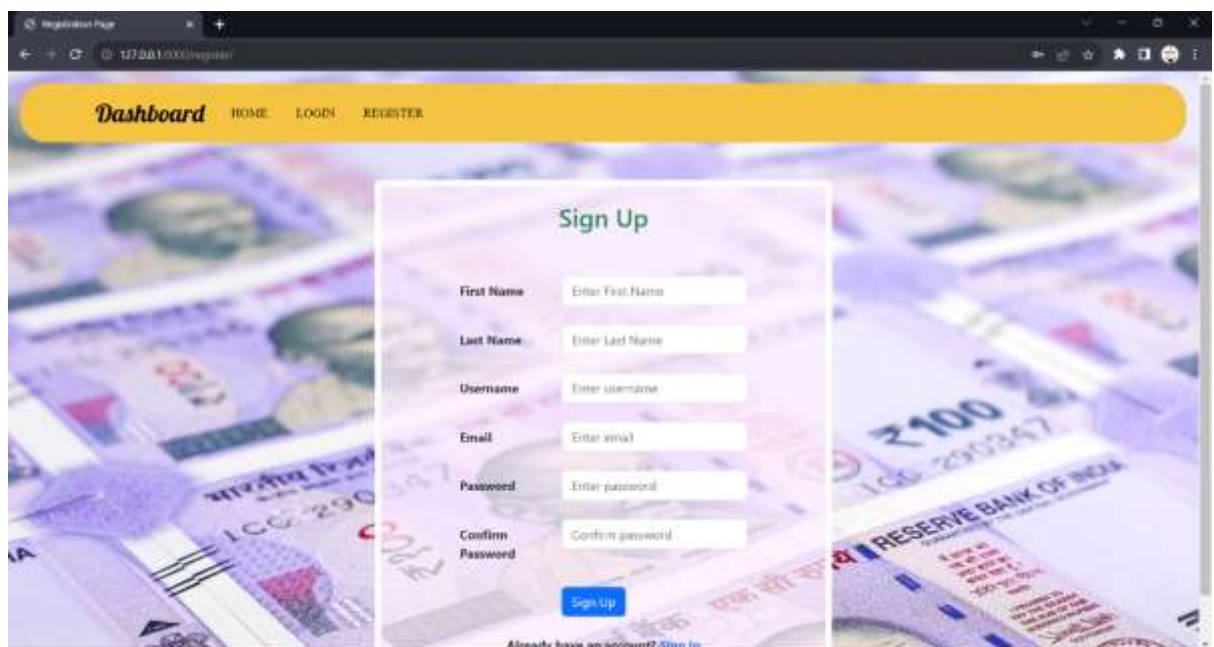


Fig 9.2: Registration Page

9.3 LOGIN PAGE

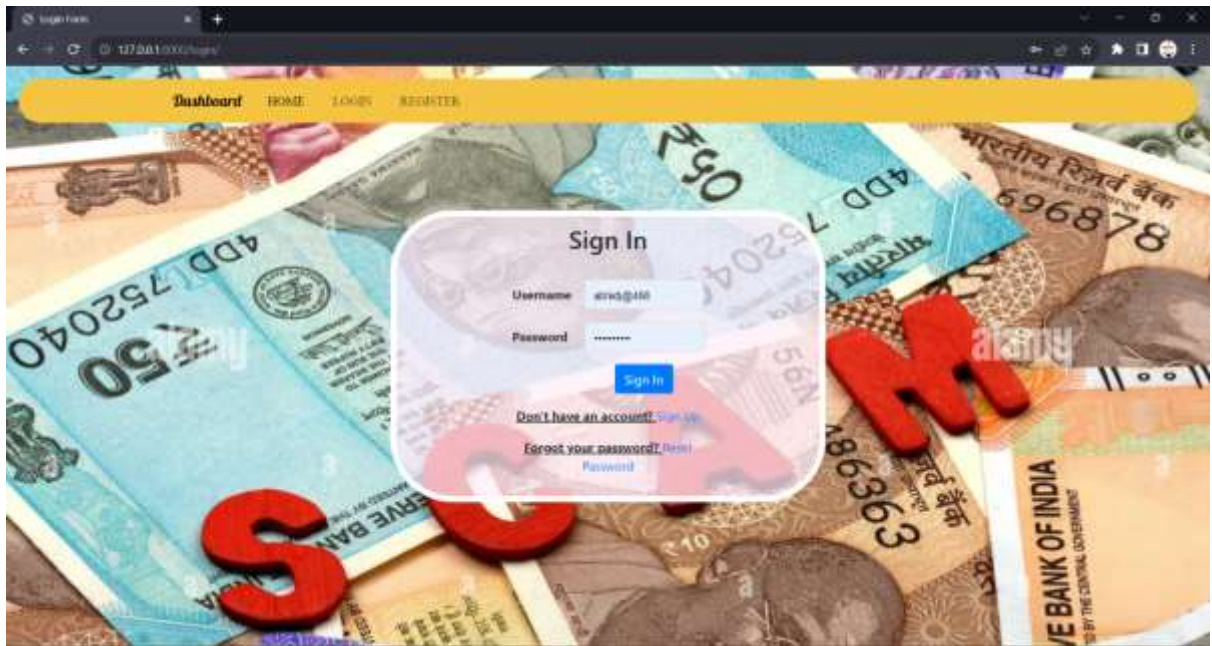


Fig 9.3: Login Page

9.4 USER HOME PAGE



Fig 9.4: User Home Page

9.5 UPLOAD DATASET

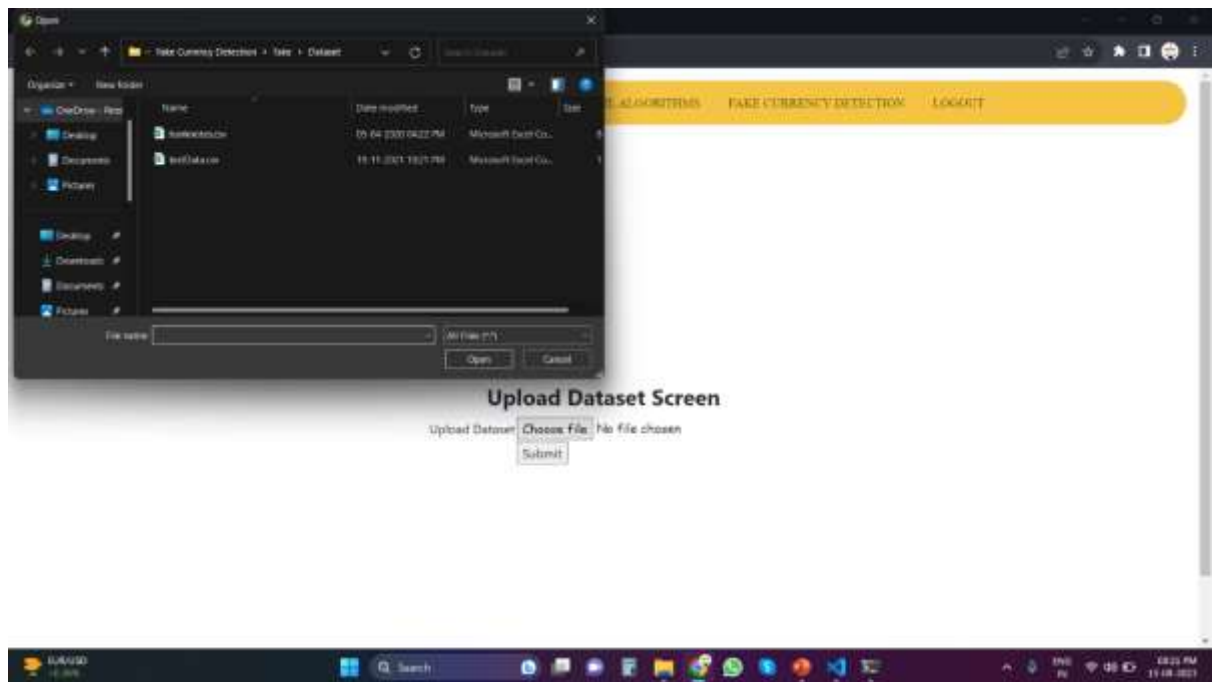


Fig 9.5: Upload Dataset

9.6 CLASSIFICATION

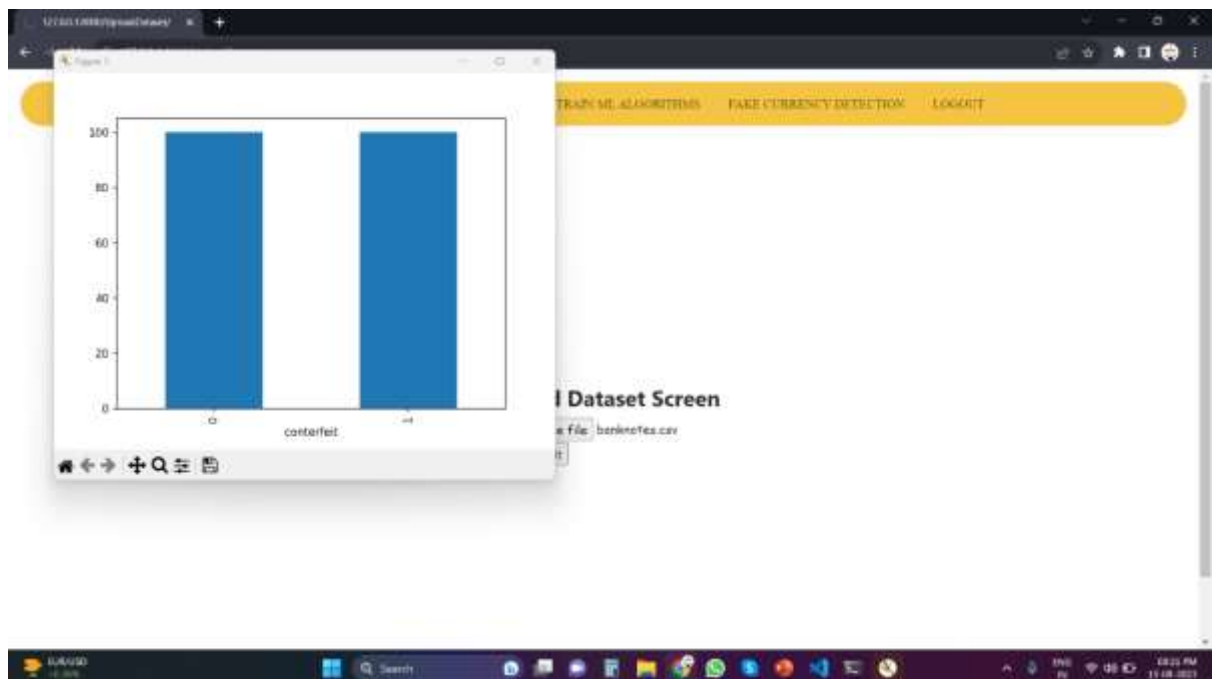


Fig 9.6: Classification

9.7 EXTRACTED FEATURES

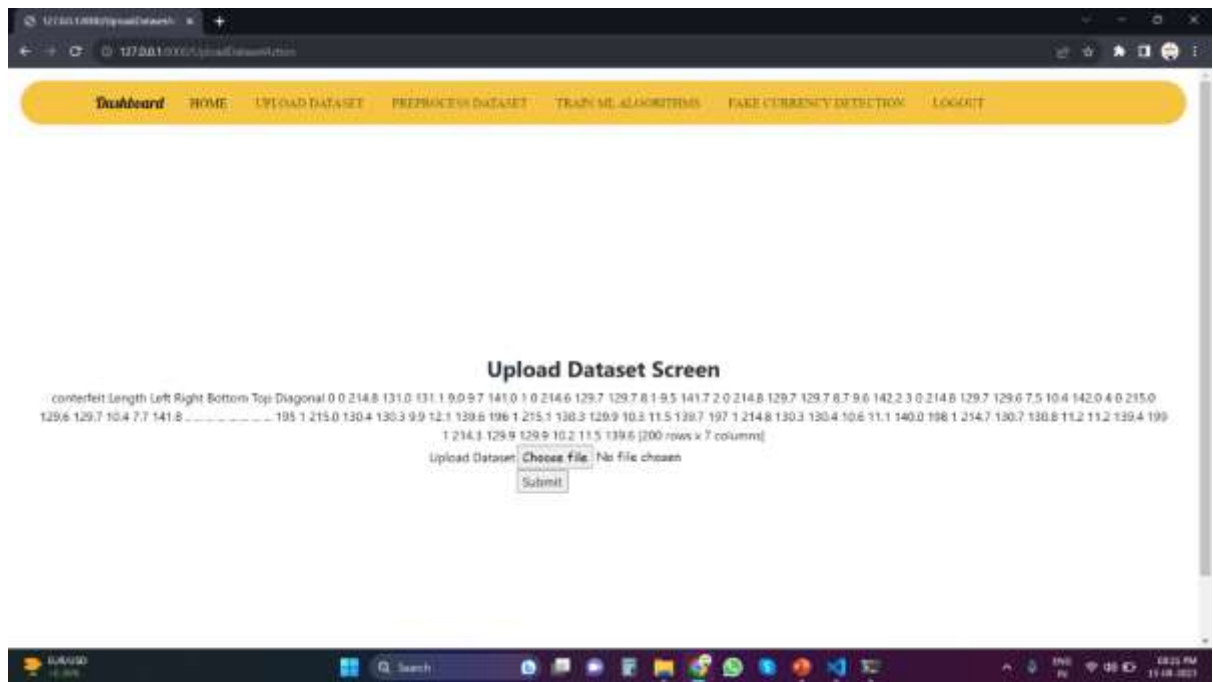


Fig 9.7: Extracted Features

9.8 PRE-PROCESSED DATASET

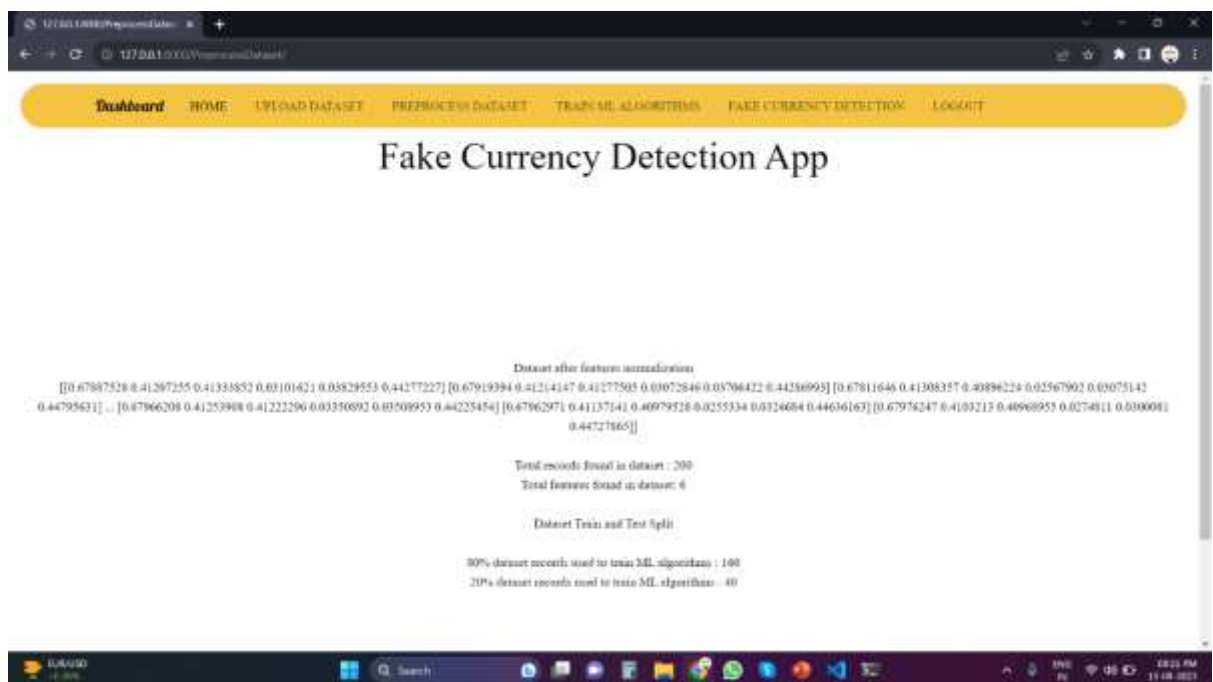


Fig 9.8: Pre-processed Dataset

9.9 ALGORITHM'S EVALUATION

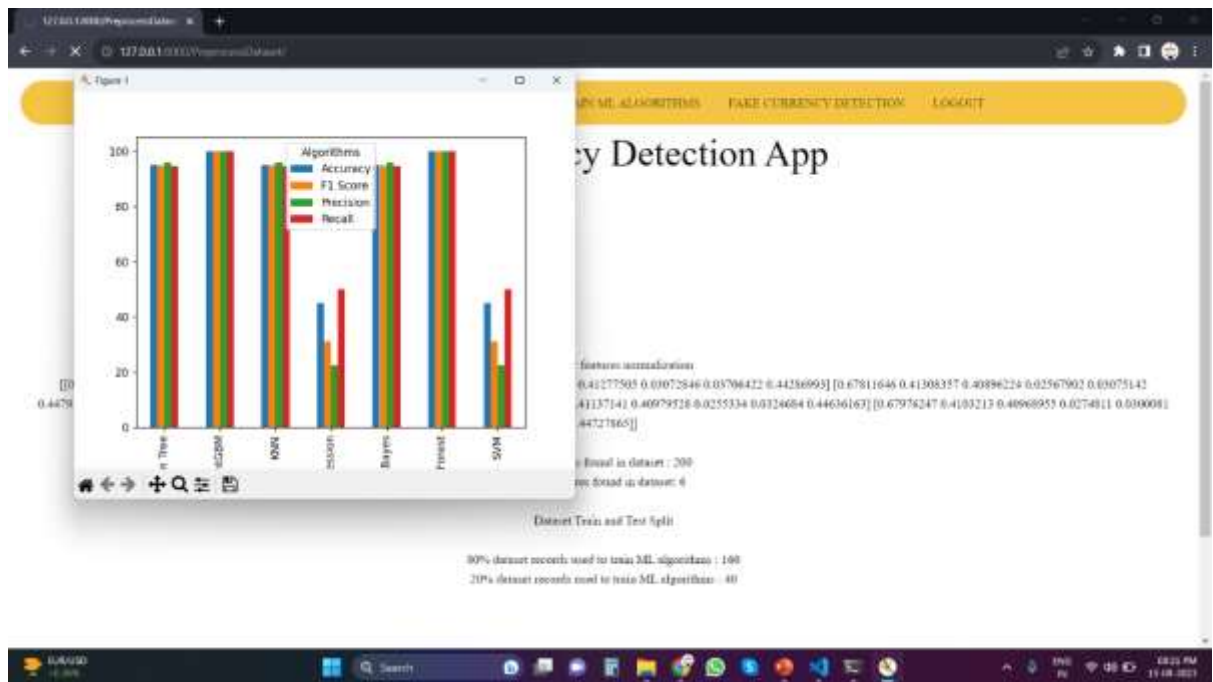


Fig 9.9: Algorithm's Evaluation

9.10 ALGORITHM'S PERFORMANCE

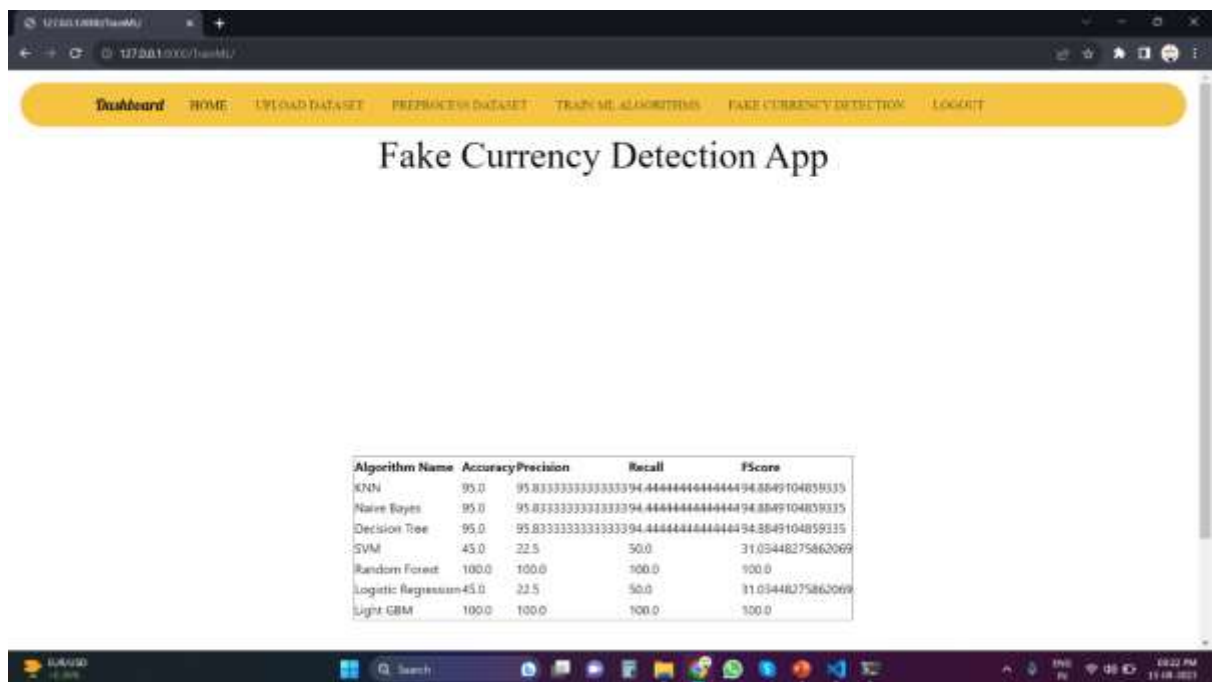


Fig 9.10: Algorithm's Performance

9.11 UPLOAD TEST DATA

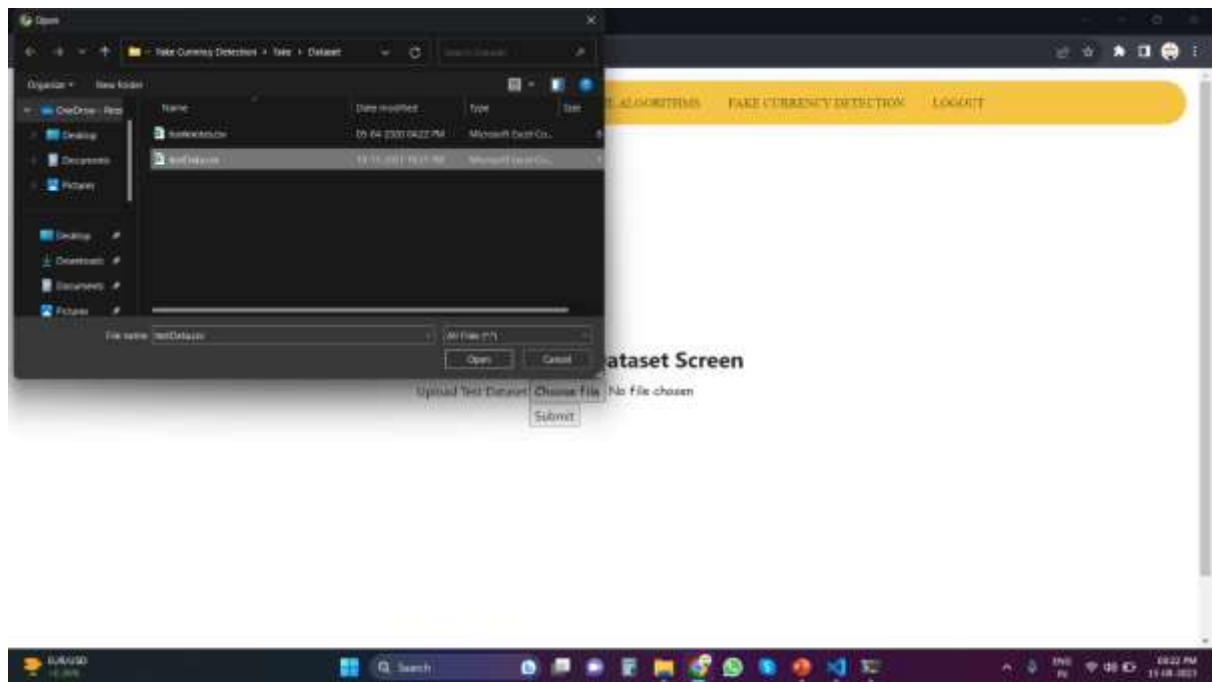


Fig 9.11: Upload Test Data

9.12 FINAL PREDICTION

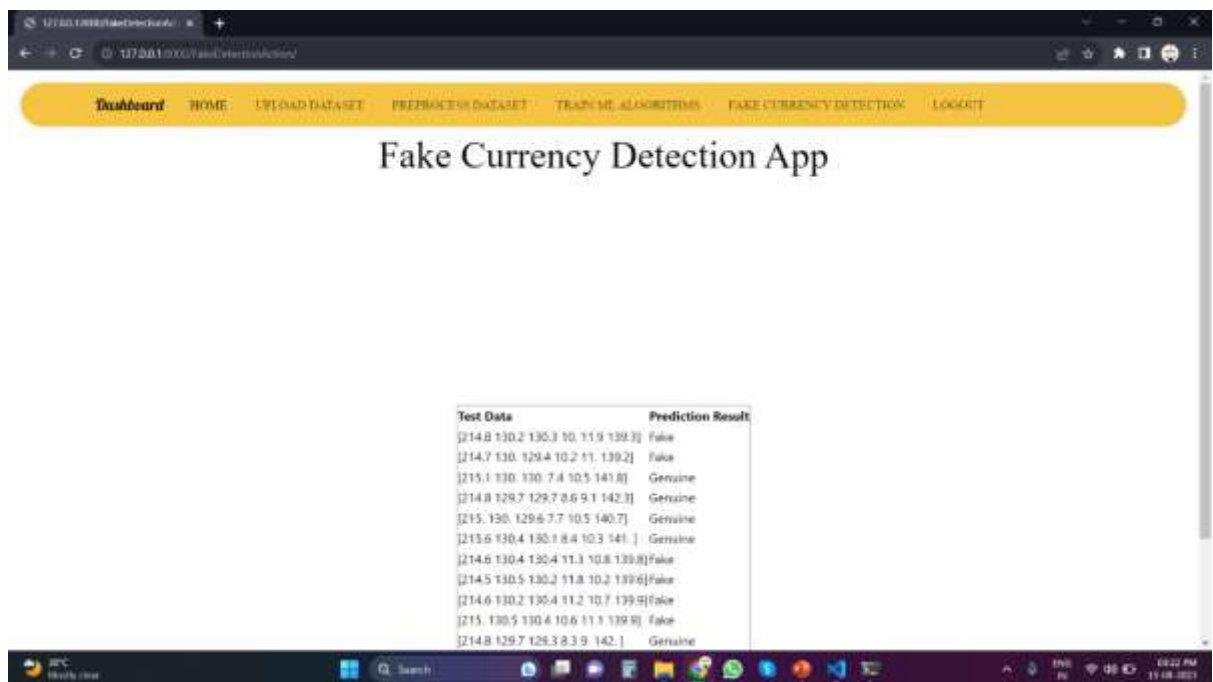


Fig 9.12: Final Prediction

CHAPTER 10

CONCLUSION AND FUTURE SCOPE

Expanding on the thorough analysis conducted, future research avenues hold the potential to further enrich the field of banknote authentication and amplify financial fraud detection capabilities. These prospective paths stem from the current study's insights, with a focus on advancing prediction accuracy, robustness, and applicability across diverse scenarios. Prominent areas for exploration include ensemble methods and hybrid approaches that blend different algorithms to capitalize on their strengths, as well as targeted feature engineering to enhance model performance.

Moreover, investigating domain adaptation and transfer learning can bolster models' adaptability to evolving counterfeit distributions. To ensure model transparency, delving into explainability techniques can shed light on the decision-making process of complex algorithms like LIGHTGBM. Efficient real-time deployment of algorithms and fortifying them against adversarial attacks are also pivotal areas for future research, considering the real-world implications of these technologies. Furthermore, the investigation of multi-modal data fusion, global dataset analysis, and longitudinal studies can provide a more holistic understanding of the algorithms' performance across diverse contexts and timeframes.

In conclusion, this study serves as a foundation for future exploration within the banknote authentication domain. By delving into the recommended research directions, scholars can build upon this groundwork and contribute to the advancement of sophisticated algorithms and methodologies that fortify currency security and combat financial fraud. This comprehensive approach underscores the role of machine learning in safeguarding financial systems from the threats of counterfeiting and fraudulent activities.

BIBLIOGRAPHY

- [1] M. Aoba, T. Kikuchi, and Y. Takefuji, “Euro Banknote Recognition System Using a Three-layered Perceptron and RBF Networks”, IPSJ Transactions on Mathematical Modeling and its Applications, May 2003.
- [2] S. Desai, S. Kabade, A. Bakshi, A. Gunjal, M. Yeole, “Implementation of Multiple Kernel Support Vector Machine for Automatic Recognition and Classification of Counterfeit Notes”, International Journal of Scientific & Engineering Research, October-2014.
- [3] C. Gigliarano, S. Figini, P. Muliere, “Making classifier performance comparisons when ROC curves intersect”, Computational Statistics and Data Analysis 77 (2014) 300–312.
- [4] E. Gillich and V. Lohweg, “Banknote Authentication”, 2014.
- [5] H. Hassanpour and E. Hallajian, “Using Hidden Markov Models for Feature Extraction in Paper Currency Recognition.
- [6] Z. Huang, H. Chen, C. J. Hsu, W. H. Chen and S. Wuc, “Credit rating analysis with support vector machines and neural network: a market comparative study”, 2004.
- [7] C. Kumar and A. K. Dudyala, “Banknote Authentication using Decision Tree rules and Machine Learning Techniques”, International Conference on Advances in Computer Engineering and Applications(ICACEA), 2015.
- [8] M. Lee and T. Chang, “Comparison of Support Vector Machine and Back Propagation Neural Network in Evaluating the Enterprise Financial Distress”, International Journal of Artificial Intelligence & Applications 1.3 (2010) 31-43.
- [9] C. Nastoulis, A. Leros, and N. Bardis, “Banknote Recognition Based On Probabilistic Neural Network Models”, Proceedings of the 10th WSEAS International Conference on SYSTEMS, Vouliagmeni, Athens, Greece, July 10-12, 2006.