1. METHODOLOGY

1.1 Introduction

In this section, we present the methodological approach employed to address the functions and components of the proposed system. Our research follows a structured methodology rooted in the software lifecycle model, ensuring a systematic and well-organized approach to system implementation. Extensive prior research in the same domain has provided valuable insights and knowledge, which we harness to fulfil the primary and secondary objectives of this study. The wealth of information and findings from previous studies serves as a foundational resource, guiding us in the development and execution of the proposed system.

1.2 System Overview

This section provides an overview of the proposed integrated web application, which has been developed based on the findings from the literature review and extensive research in the field. The objective was to select appropriate technologies, software solutions, and tools for the implementation phase. The resulting system comprises following four components, each designed to address specific aspects of data analysis, prediction, and classification.

- Accident Case Analysis
- Crime Case Analysis
- Case Document Classification
- Crimes against Women Analysis.

Together, these components form a cohesive and comprehensive platform intended for use by police departments and investigators.

The proposed integrated web application combines the power of data analysis, prediction, and classification to provide a comprehensive solution for law enforcement agencies and investigators. By harnessing diverse algorithms and analytical techniques, this system empowers users to gain deeper insights into accident patterns, crime trends, legal documents, and crimes against women. These insights enable more informed decision-making and proactive measures, contributing to improved public safety and law enforcement effectiveness.

In the following sections, we will delve into each component in greater detail, elucidating their functionalities and benefits.

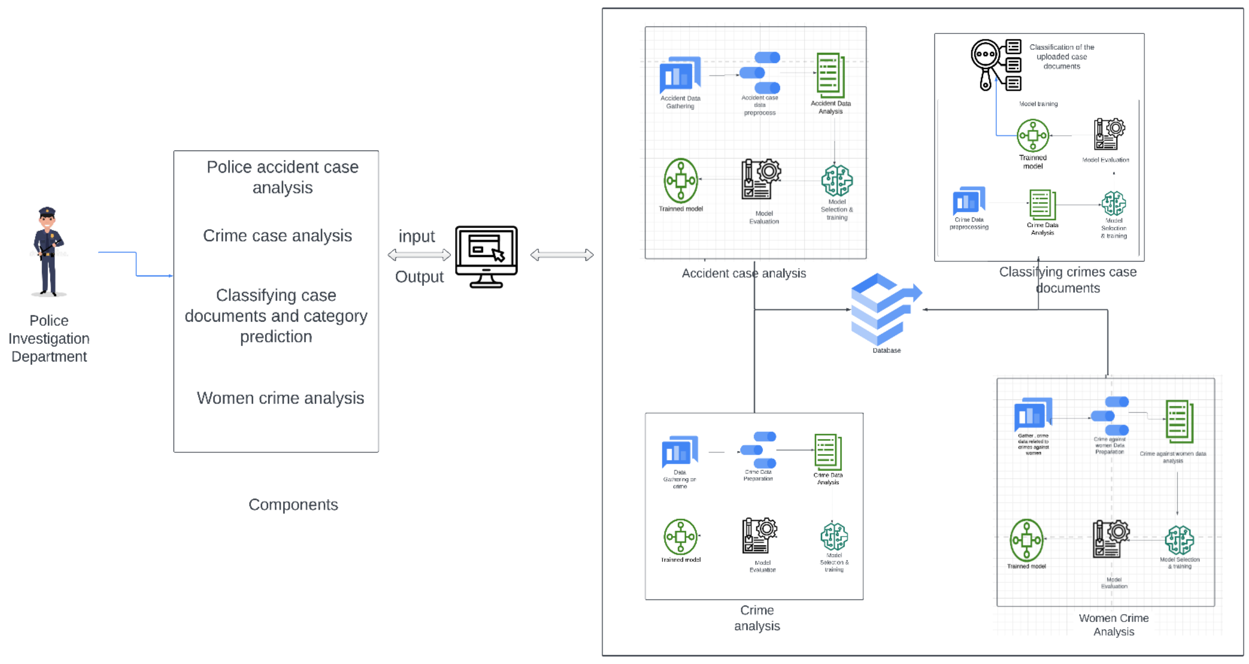

Police Investigation Department
Police accident case analysis
Crime case analysis
Classifying case documents and category prediction
Women crime analysis
Components
input
Output
Accident Data Gathering
Accident case data preprocess
Accident Data Analysis
Trainned model
Model Evaluation
Model Selection & training
Accident case analysis
Classification of the uploaded case documents
Model training
Trainned model
Model Evaluation
Crime Data preprocessing
Crime Data Analysis
Model Selection & training
Classifying crimes case documents
Database
Data Gathering of crime
Crime Data Preparation
Crime Data Analysis
Trainned model
Model Evaluation
Model Selection & training
Crime analysis
Gather crime data related to crimes against women
Crime against women Data Preparation
Crime against women data analysis
Trainned model
Model Selection & training
Model Evaluation
Women Crime Analysis

***Figure 2.2.1**:* System high-level architecture diagram

1.3 Component Overview

The integrated system comprises four main components, each serving a purpose, police accident case analysis, crime analysis, case document classification using text analysis, and crimes against women analysis. Each of these components is geared towards predictive and analytical functions, contributing to various aspects of the system's capabilities.

The police accident case analysis component within the integrated system serves as a crucial tool for enhancing road safety planning and optimizing resource allocation. This component comprises multiple stages, each contributing to a comprehensive understanding of accident predictions, patterns and their underlying causes. Initially, the autoregressive integrated moving average (ARIMA) model is employed to predict future accident percentages by analyzing historical data, considering variables such as division name and year. This predictive analysis enables proactive measures for improving road safety. Subsequently, the K-means clustering technique is applied to categorize accidents based on factors such as light conditions, weather, drunken driving, and traffic. This clustering process identifies patterns and similarities within the dataset, allowing the system to categorize accidents according to their root causes. The ARIMA model is then utilized to forecast future accident trends within each cluster, utilizing statistical methods to enhance the accuracy of predictions. Furthermore, the component employs various statistical analysis techniques to uncover meaningful patterns, trends, and relationships within accident cases. In this analysis, we aim to understand the distribution and trends in accident data based on various temporal factors such as day, month, day of the week, and hour of the day. Leveraging this information, the system can derive appropriate prevention strategies tailored to address specific accident causes. This comprehensive approach empowers authorities to implement proactive measures and improve road safety. Users of this component can input historical and current accident data, and the system dynamically predicts accident percentages and fatal percentages for future years, incorporating updated datasets. The outcomes are presented through percentages and graphical representations, enabling law enforcement agencies to make informed decisions, formulate prevention strategies, and take actions aimed at preventing accidents, ultimately contributing to enhanced public safety.

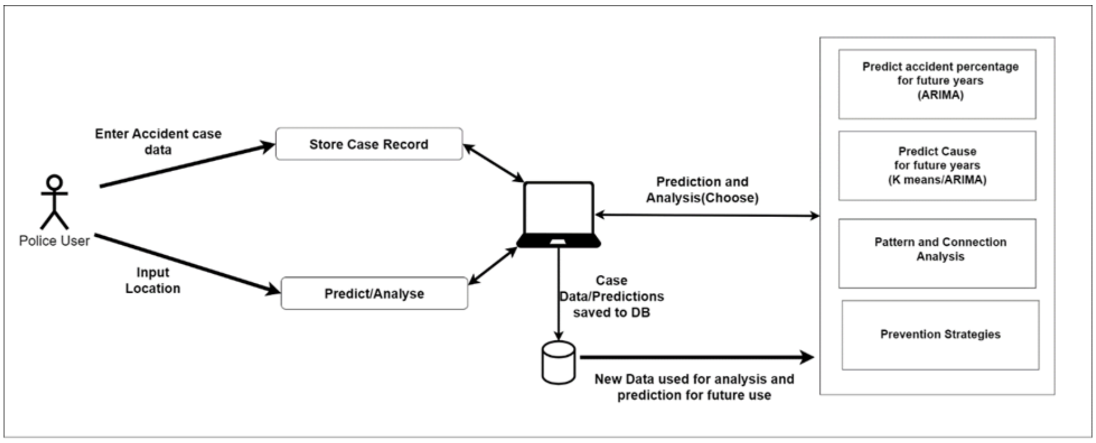

Enter Accident case data
Store Case Record
Police User
Input Location
Predict/Analyse
Prediction and Analysis(Choose)
Case Data/Predictions saved to DB
New Data used for analysis and prediction for future use
Predict accident percentage for future years (ARIMA)
Predict Cause for future years (K means/ARIMA)
Pattern and Connection Analysis
Prevention Strategies

***Figure 2.3.1:* -** High Level Architectural Diagram of police case analysis

The cornerstone of the crime analysis system lies in the gathering and pre-processing of data sourced from various law enforcement agencies. This critical phase ensures that the data is transformed into a format conducive to comprehensive analysis. During pre-processing, a significant portion of the data is encoded, optimizing its suitability for model training, thereby enhancing efficiency in subsequent phases. Decision Tree algorithm strategically selected for its distinctive attributes and applicability in predicting crime patterns. Decision Trees are renowned for their exceptional interpretability, offering a clear and intelligible decision-making path. This feature is instrumental in providing transparency and aiding in understanding why a specific prediction was made, which is crucial for both law enforcement and stakeholders. Furthermore, Decision Trees demonstrate versatility in handling a combination of categorical and numerical features, a common scenario in real-world datasets like crime data. This flexibility is paramount as it allows the model to effectively learn from a diverse range of input variables and find out the patterns and connections in between the crimes. Following the analysis, the outputs are not only insightful but also presented in a visually intuitive format. The crime analysis system employs visualization tools to convert the findings into charts, graphs, and other visual representations. This approach ensures that the results are not only accurate but also accessible, enabling law enforcement and other users to quickly grasp and act upon the insights gleaned from the data. In essence, our component harmoniously integrates data collection, preprocessing, Decision Tree modelling, and visual representation of the results. The transparency and adaptability of Decision Trees, combined with intuitive visualization, offer a holistic system for predicting and comprehending criminal activities. This integrated approach is poised to revolutionize crime analysis, providing actionable insights in an easily digestible format.

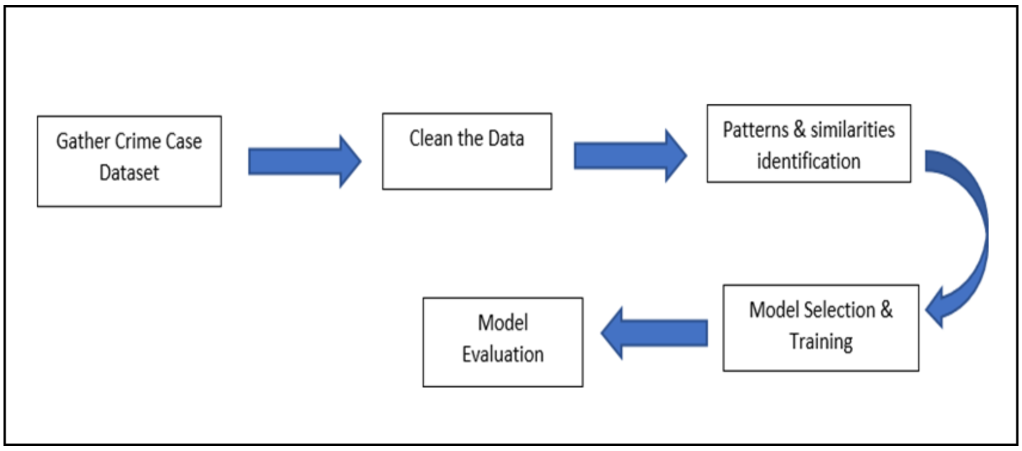

Gather Crime Case Dataset
Clean the Data
Patterns & similarities identification
Model Selection & Training
Model Evaluation

Figure 2.3.2: - High level architecture diagram of Analysing and grouping commonalities among criminal cases and predicting the future crimes in terms of the pattern of the crime.

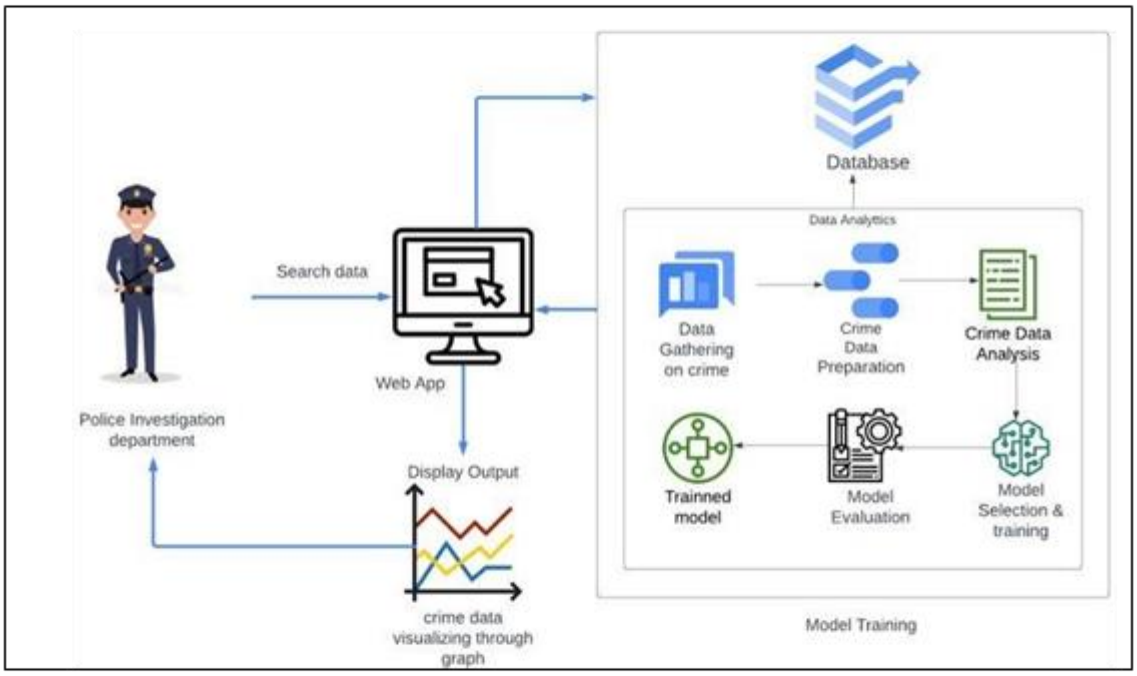


Figure 2.3.2: - - Component overview for Analysing and grouping commonalities among criminal cases and predicting the future crimes in terms of the pattern of the crime.

The case document classification component to develop an automated text analysis system for Sri Lankan law enforcement agencies. It will enhance the quality and accuracy of criminal investigations by replacing labour-intensive manual processes. The process begins with meticulous text extraction from various sources, ensuring precise information capture. Extracted data undergoes manual review, with each document carefully assigned relevant labels based on its content. This labelling process is vital, requiring a deep understanding of document context. It creates a structured training dataset with rows representing individual documents and columns indicating labels, each with a binary value (True or False) indicating label presence. To address multi-label text classification, the research evaluates three algorithms: OneVsRestClassifier with Naive Bayes, LinearSVC, and Logistic Regression. These are crucial for complex legal documents, enabling simultaneous classification of multiple labels. Text data pre-processing techniques like removing punctuation, stop words, and stemming/lemmatization standardize the data. The dataset is split into training and testing sets for unbiased algorithm evaluations. Metrics tailored for multi-label classification, including accuracy, precision, recall, and F1-score, assess algorithm effectiveness in predicting label presence.

The research aims to identify the most suitable algorithm for multi-label text classification in PDF case documents, considering factors like prediction accuracy, handling multiple labels, and computational efficiency. Implementation promises streamlined investigations, rapid identification of similar past cases, improved reliability, and efficiency in document categorization. Efficient label extraction provides investigators with swift access to information, enhancing decision-making and expediting investigations. By eliminating manual reading and labelling, it conserves time and resources, allowing more focus on analysis and interpretation. The system's accuracy and consistency contribute to a robust knowledge base, empowering the investigative team with valuable insights, supporting legal research advancements, and ultimately improving public safety.

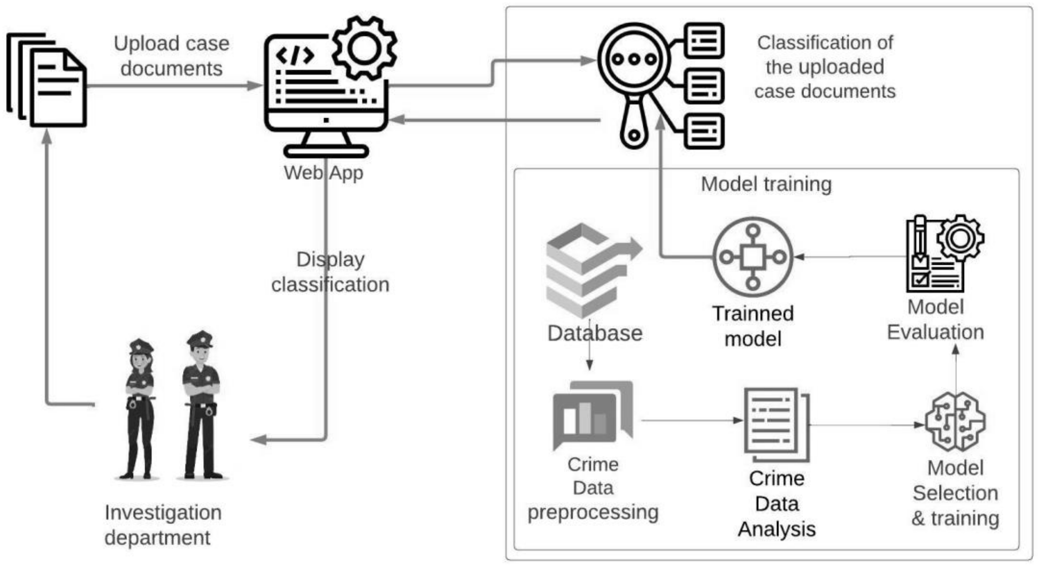


Figure 2.3.3: - Component overview for Analyze and Classify Similar Case Documents and Predict Category.

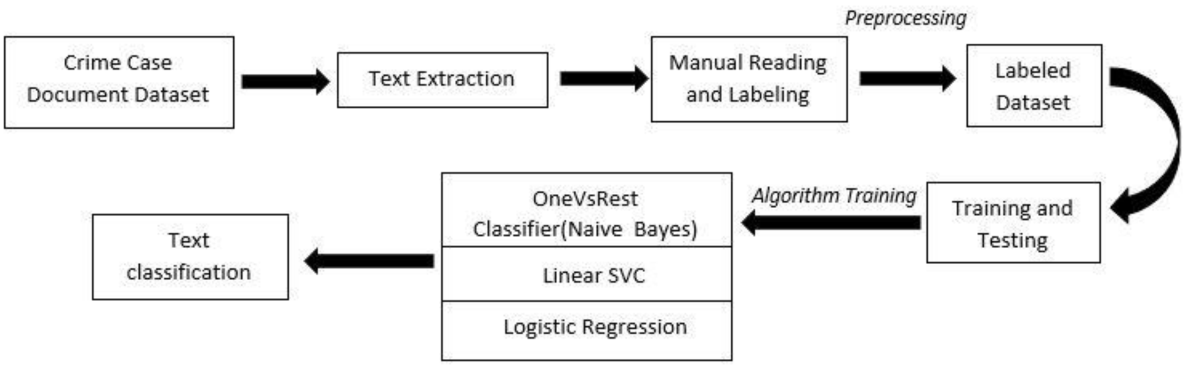


Figure 2.3.4: - High level architecture diagram of Analyze and Classify Similar Case Documents and Predict Category.

Meanwhile, the crimes against women component adopts a comprehensive methodology for the analysis and visualization of women's crime data. The objective of this study is to develop a predictive model to analyse and visualize crimes against women with a focus on predicting future crime rates in various states and federations of Sri Lanka. The methodology included several main steps. It begins with the collection of crime statistics from authoritative sources, followed by thorough data pre-processing to ensure data quality. A Random Forest Regressor model is employed for predicting future crime counts based on user-defined parameters like state, year, and crime type. Predicted results are then interpreted into categorical crime rate levels, aiding in understanding anticipated crime trends. Random Forest Regressor model was chosen because of its ability to handle complex, non-linear relationships between predictors and crime rates. This model has been carefully trained using historical data to capture patterns and trends. To evaluate its predictive performance, user-friendly interfaces were created that accept user inputs, predict future crime rates based on a selected crime category, and interpret the results into different categories such as "low crime area" or "very high crime area". Data visualization techniques, including bar and pie charts, are utilized to represent crime patterns visually. In addition, they were used to graphically depict crime trends. Finally, the study was extended to provide crime prevention strategies tailored to specific criminal groups. The Random Forest Regression was chosen for its robustness in using multi-attribute datasets, its ability to perform non-linear modelling and its ability to provide reliable predictions of future crime rates, making it a suitable choice for our analytical purposes.

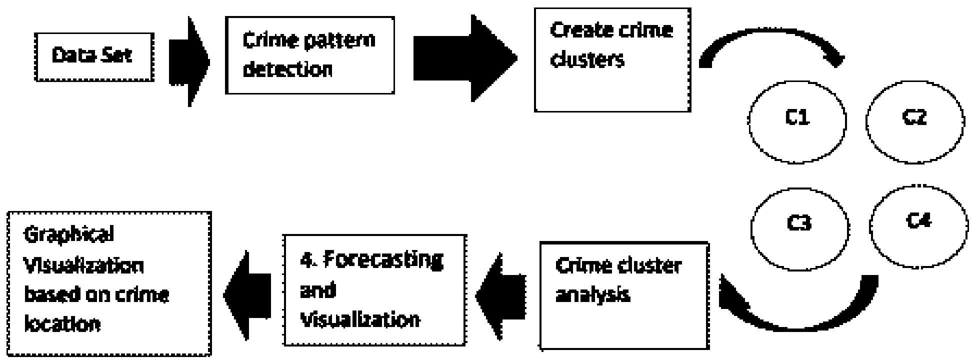


Figure 2.3.5: - High level architecture diagram of clustering crimes against women and crime forecasting

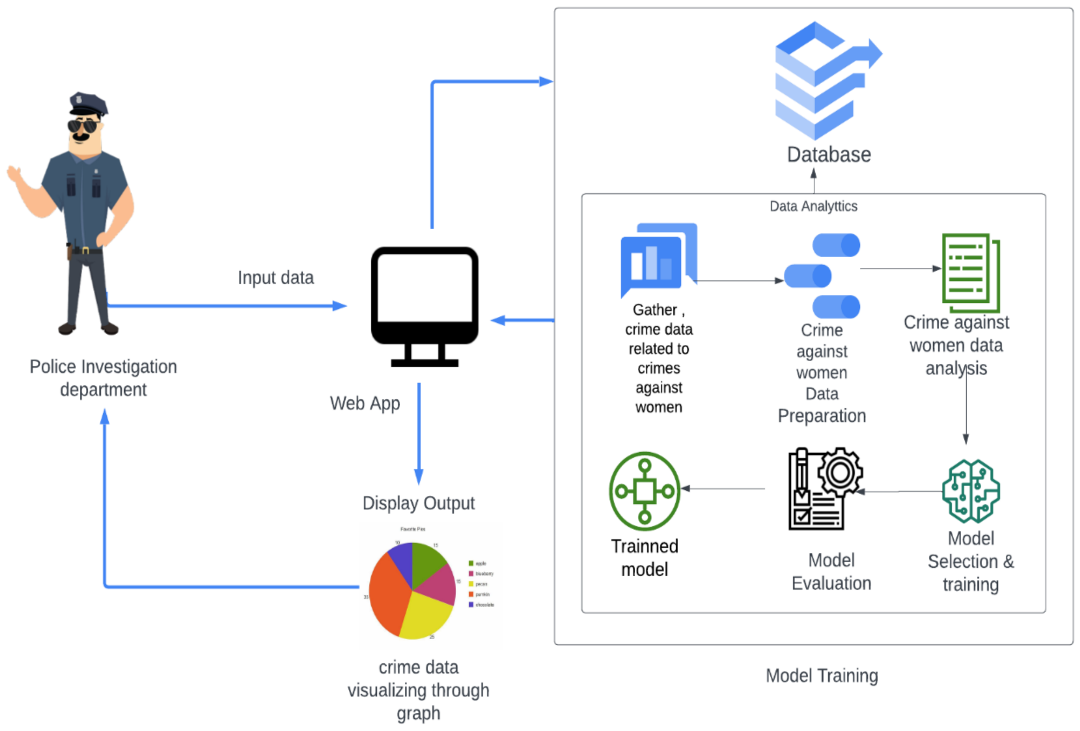

Database
Data Analyttics
Police Investigation department
Input data
Web App
Gather , crime data related to crimes against women
Crime against women Data Preparation
Crime against women data analysis
Display Output
Trainned model
Model Evaluation
Model Selection & training
crime data visualizing through graph
Model Training

Figure 2.3.6: - Component overview for clustering crimes against women and future crime forecasting prediction.

The proposed integrated web application combines the power of data analysis, prediction, and classification to provide a comprehensive solution for police departments and investigators. By harnessing diverse algorithms and analytical techniques, this system empowers users to gain deeper insights into accident predictions, crime analysis, documents classification, and crimes against women.

1.3.1 Work Breakdown Structure (WBS)

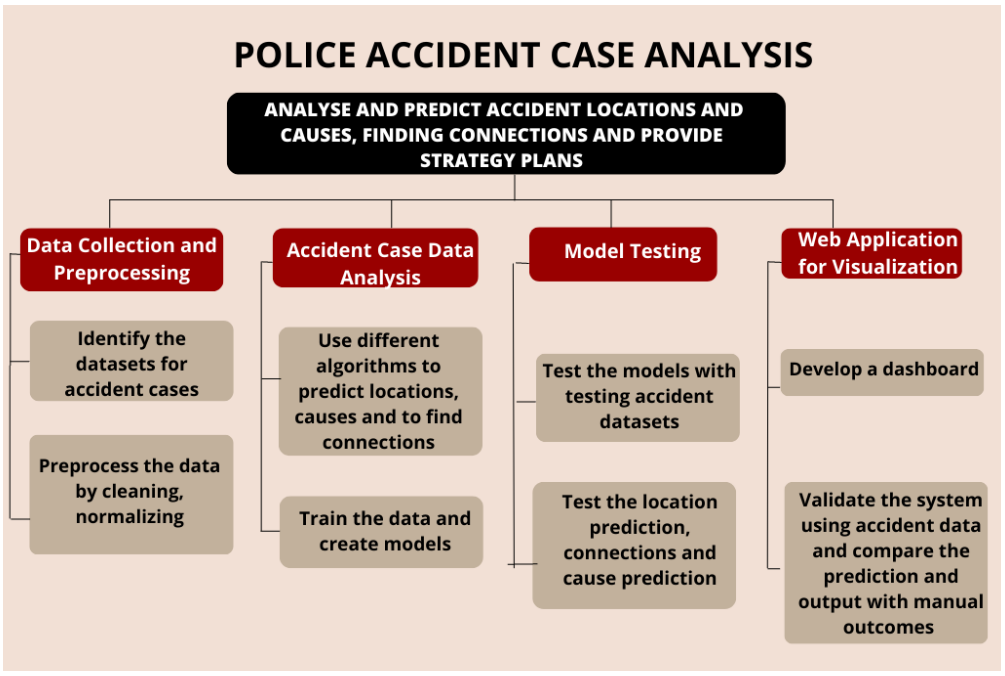

POLICE ACCIDENT CASE ANALYSIS
ANALYSE AND PREDICT ACCIDENT LOCATIONS AND CAUSES, FINDING CONNECTIONS AND PROVIDE STRATEGY PLANS
Data Collection and Preprocessing
Identify the datasets for accident cases
Preprocess the data by cleaning, normalizing
Accident Case Data Analysis
Use different algorithms to predict locations, causes and to find connections
Train the data and create models
Model Testing
Test the models with testing accident datasets
Test the location prediction, connections and cause prediction
Web Application for Visualization
Develop a dashboard
Validate the system using accident data and compare the prediction and output with manual outcomes

Figure 2.3.1.1: - WBS of accident case analysis

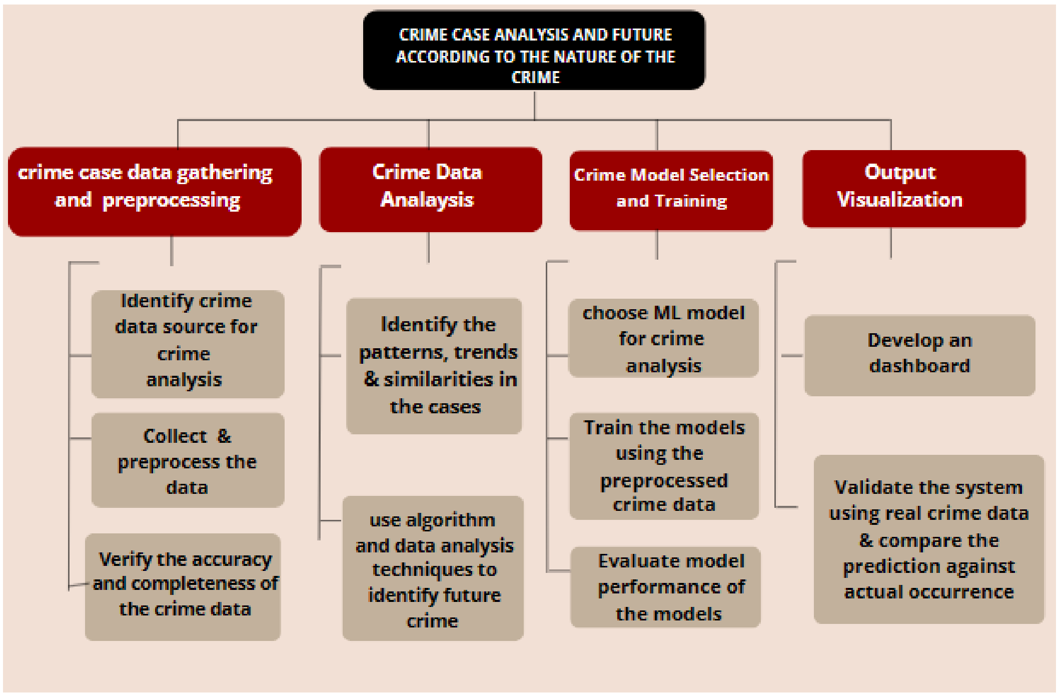

CRIME CASE ANALYSIS AND FUTURE ACCORDING TO THE NATURE OF THE CRIME
crime case data gathering and preprocessing
Identify crime data source for crime analysis
Collect & preprocess the data
Verify the accuracy and completeness of the crime data
Crime Data Analysis
Identify the patterns, trends & similarities in the cases
use algorithm and data analysis techniques to identify future crime
Crime Model Selection and Training
choose ML model for crime analysis
Train the models using the preprocessed crime data
Evaluate model performance of the models
Output Visualization
Develop an dashboard
Validate the system using real crime data & compare the prediction against actual occurrence

Figure 2.3.1.2: - WBS of crime case analysis

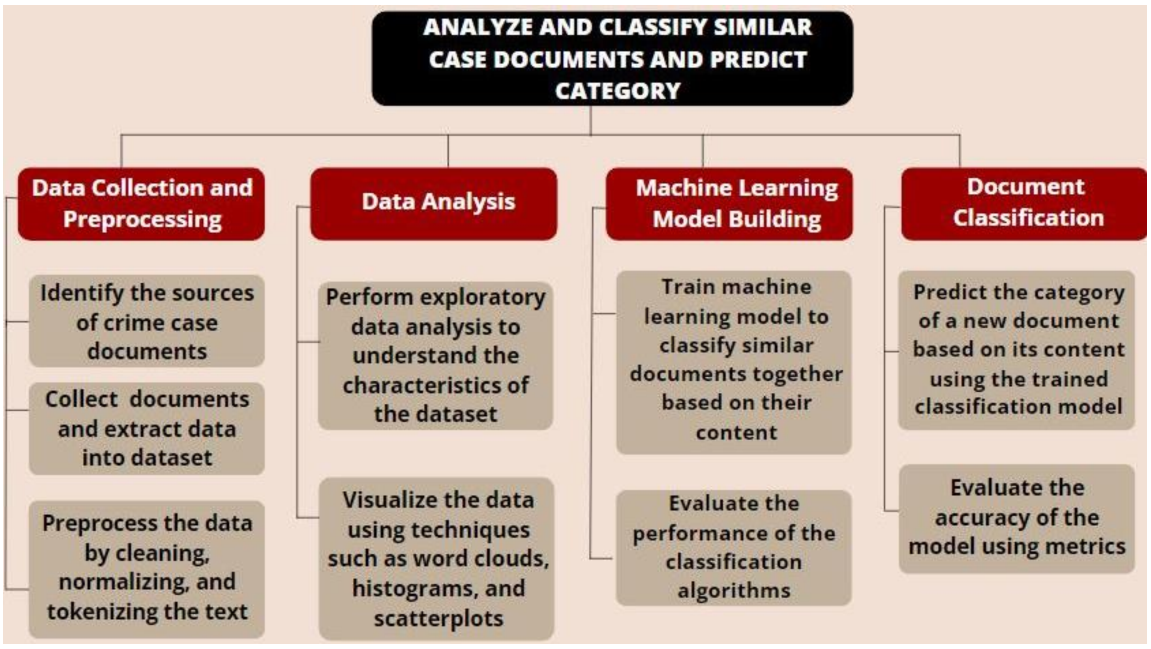

ANALYZE AND CLASSIFY SIMILAR CASE DOCUMENTS AND PREDICT CATEGORY
Data Collection and Preprocessing
Identify the sources of crime case documents
Collect documents and extract data into dataset
Preprocess the data by cleaning, normalizing, and tokenizing the text
Data Analysis
Perform exploratory data analysis to understand the characteristics of the dataset
Visualize the data using techniques such as word clouds, histograms, and scatterplots
Machine Learning Model Building
Train machine learning model to classify similar documents together based on their content
Evaluate the performance of the classification algorithms
Document Classification
Predict the category of a new document based on its content using the trained classification model
Evaluate the accuracy of the model using metrics

Figure 2.3.1.3: - WBS of document classification analysis

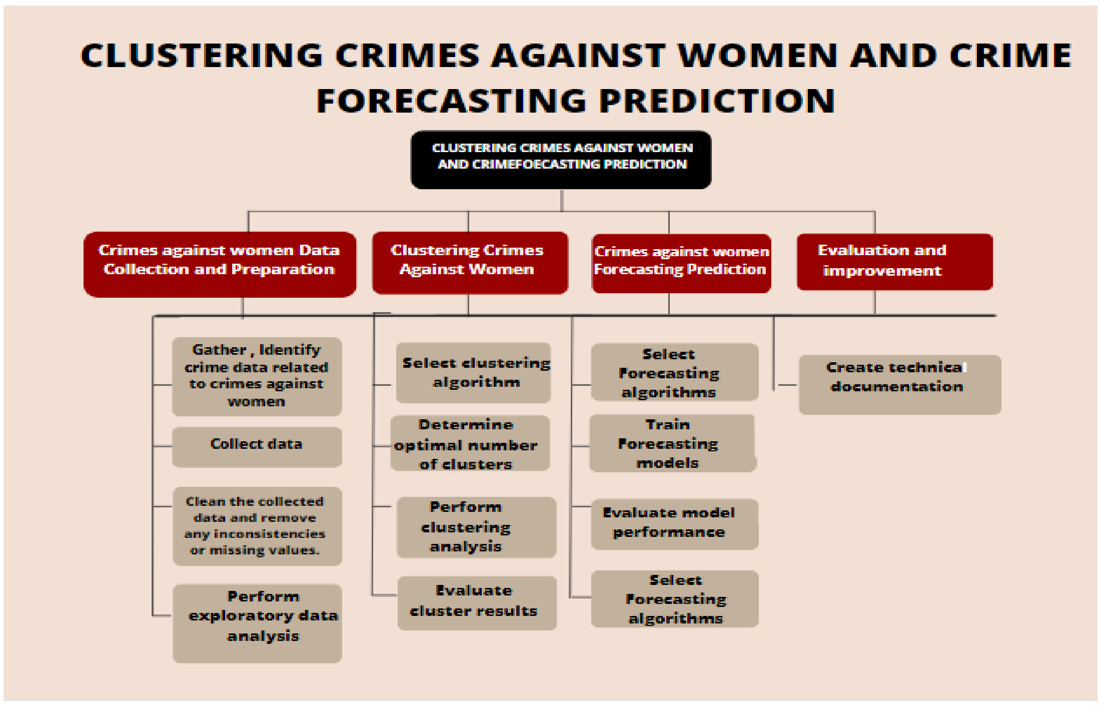

CLUSTERING CRIMES AGAINST WOMEN AND CRIME FORECASTING PREDICTION
CLUSTERING CRIMES AGAINST WOMEN AND CRIMEFOECASTING PREDICTION
Crimes against women Data Collection and Preparation
Gather , Identify crime data related to crimes against women
Collect data
Clean the collected data and remove any inconsistencies or missing values.
Perform exploratory data analysis
Clustering Crimes Against Women
Select clustering algorithm
Determine optimal number of clusters
Perform clustering analysis
Evaluate cluster results
Crimes against women Forecasting Prediction
Select Forecasting algorithms
Train Forecasting models
Evaluate model performance
Select Forecasting algorithms
Evaluation and improvement
Create technical documentation

Figure 2.3.1.4: - WBS of crimes against women analysis

1.4 Development Process

The development process chosen for our system follows a Waterfall model. This model is selected because it offers a linear and systematic approach, making it well-suited for our project's needs and clearly defined specifications. The Waterfall model involves distinct stages, each with tasks that can be scheduled and completed within specific timeframes. These stages and the associated strategy proceed sequentially without overlapping. Following the guidelines in this section, we will address and investigate the issues raised, provide a figurative characterization of the system's functions aimed at resolving these issues, examine the expected outcomes, and emphasize the importance of effective time management, which has been crucial throughout our year-long research endeavour. The system's requirements have been systematically divided into functional phases, as illustrated in Figure 2.5.1, which include Requirement Analysis, Design, Development, Testing, and Maintenance.

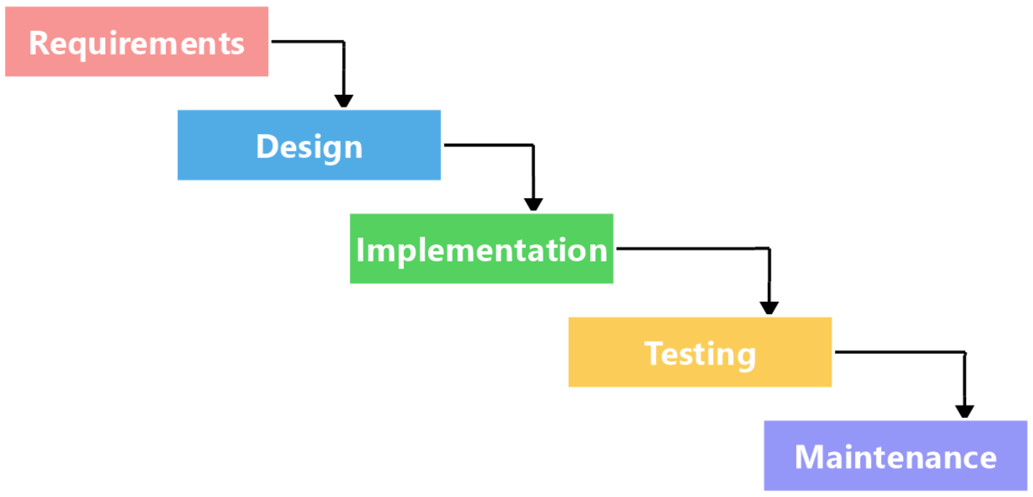
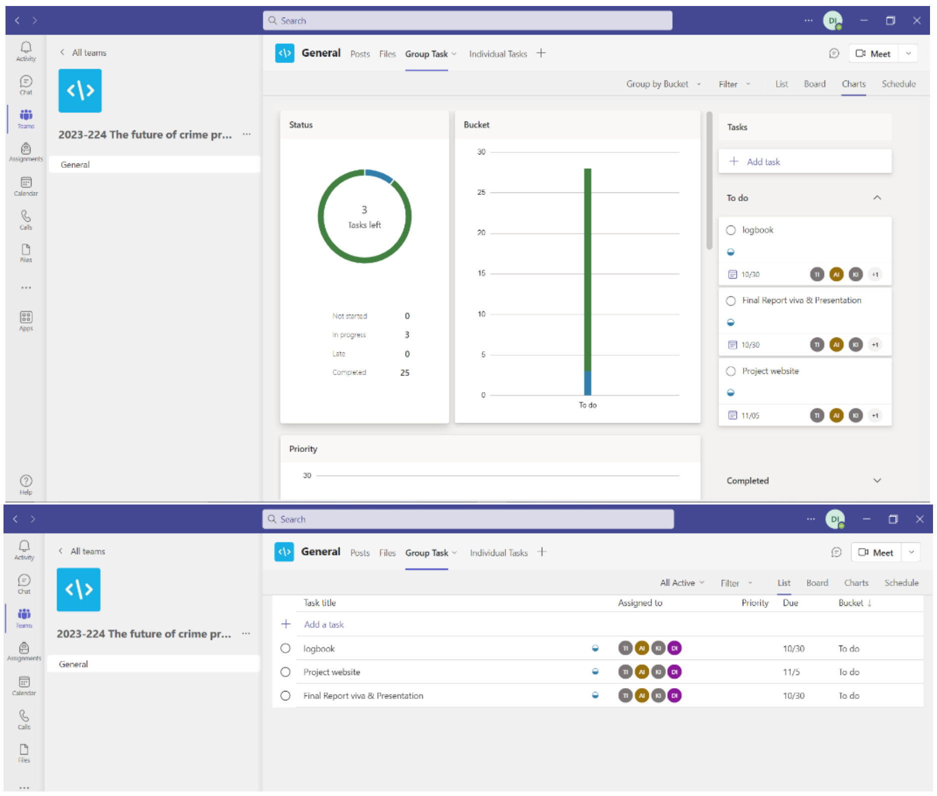


***Figure 2.5.1**:* Development process of the system

1.4.1 Project Management

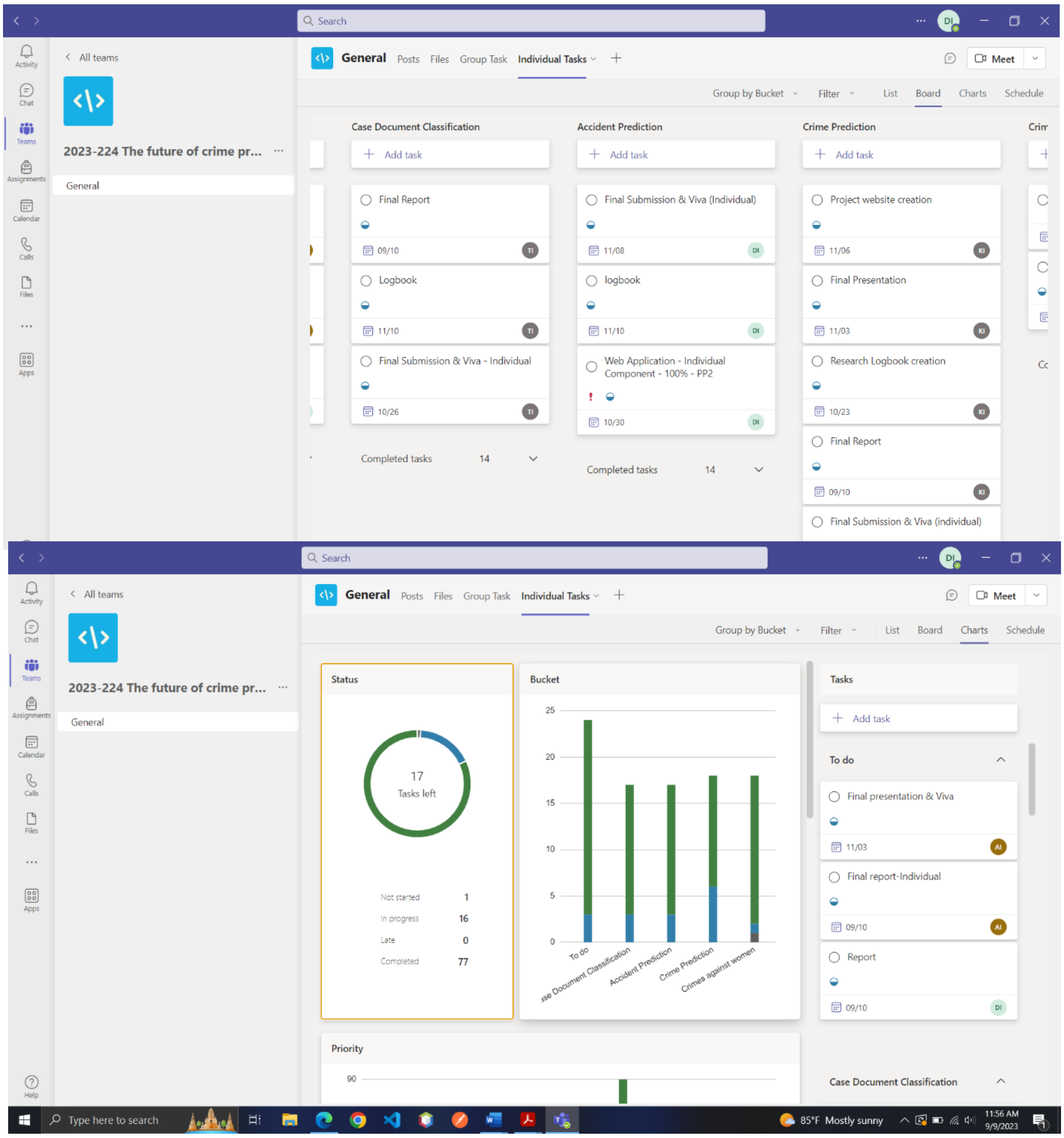
Project Management in the context of our Police Case Analysis project involves a distinctive approach due to the unique software development life cycle. Unlike traditional project management, software development entails multiple iterations, encompassing testing, updates, and continuous user feedback. To align with the dynamic nature of our project and adapt to evolving requirements from law enforcement agencies and stakeholders, we have embraced an agile methodology. Within our project teams, we have used a collaborative tool such as Microsoft teams for scheduling meetings and formulating project plans. These teams are responsible for both group and individual tasks, ensuring that every aspect of the project is systematically addressed and executed. The visual representations below provide insights into our structured approach to project management and task allocation



Status	
Not started	0
In progress	3
Late	0
Completed	25

Task title	Assigned to	Priority	Due	Bucket
logbook	TI AI KI DI		10/30	To do
Project website	TI AI KI DI		11/5	To do
Final Report viva & Presentation	TI AI KI DI		10/30	To do

Figure 2.4.1.1: Project management of the system through MS teams



Case Document Classification
- Final Report — 09/10 — TI
- Logbook — 11/10 — TI
- Final Submission & Viva - Individual — 10/26 — TI
- Completed tasks 14

Accident Prediction
- Final Submission & Viva (Individual) — 11/08 — DI
- logbook — 11/10 — DI
- Web Application - Individual Component - 100% - PP2 — 10/30 — DI
- Completed tasks 14

Crime Prediction
- Project website creation — 11/06 — KI
- Final Presentation — 11/03 — KI
- Research Logbook creation — 10/23 — KI
- Final Report — 09/10 — KI
- Final Submission & Viva (individual)

Status

17 Tasks left

Not started	1
In progress	16
Late	0
Completed	77

Bucket

Tasks

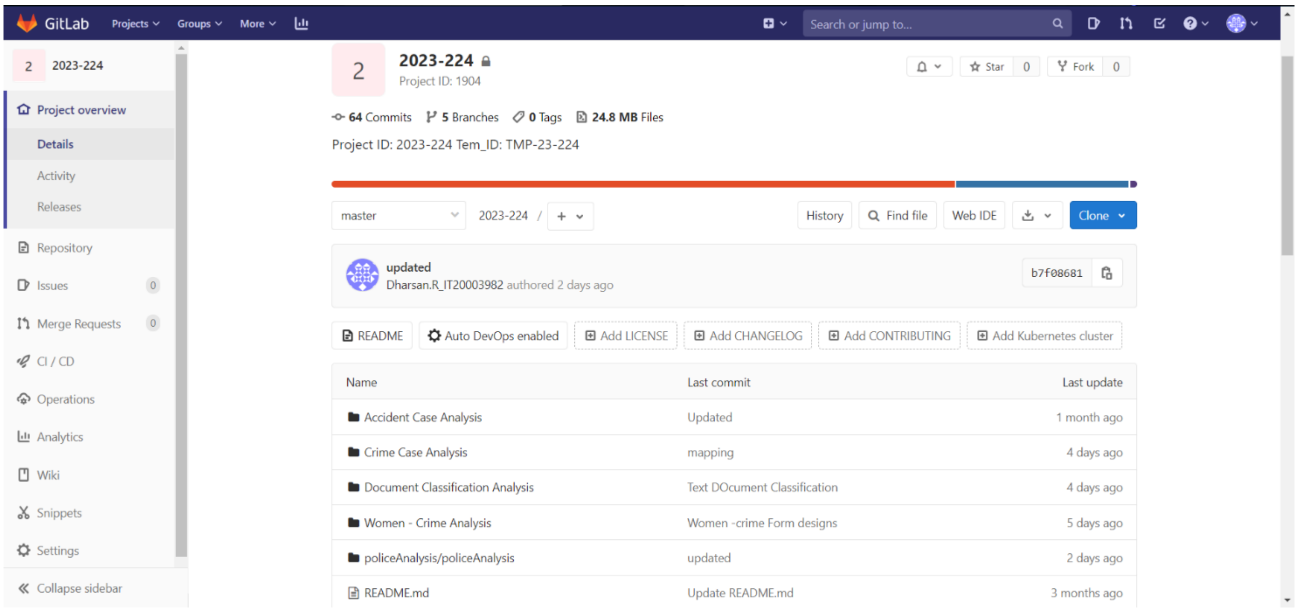
To do
- Final presentation & Viva — 11/03 — AI
- Final report-Individual — 09/10 — AI
- Report — 09/10 — DI

Case Document Classification

Priority

Figure 2.4.1.2: Project management of the system through MS teams (Individual)

1.4.1.1 Project Code Management



2023-224

Project ID: 1904

64 Commits 5 Branches 0 Tags 24.8 MB Files

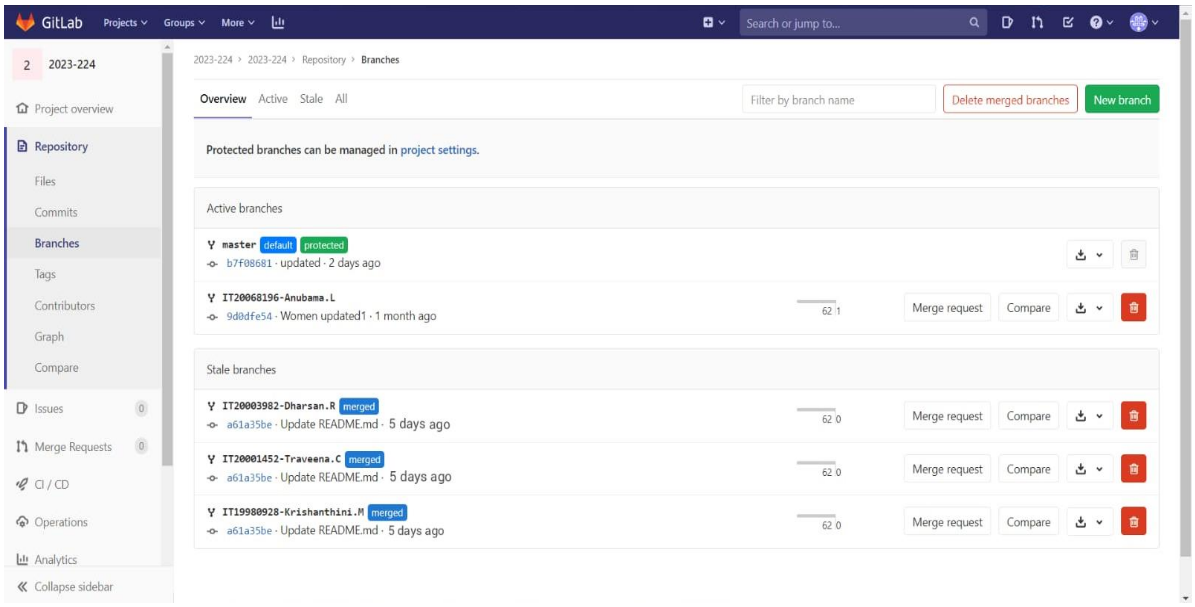
Project ID: 2023-224 Tem_ID: TMP-23-224

master 2023-224 /

updated
Dharsan.R_IT20003982 authored 2 days ago b7f08681

Name	Last commit	Last update
Accident Case Analysis	Updated	1 month ago
Crime Case Analysis	mapping	4 days ago
Document Classification Analysis	Text DOcument Classification	4 days ago
Women - Crime Analysis	Women -crime Form designs	5 days ago
policeAnalysis/policeAnalysis	updated	2 days ago
README.md	Update README.md	3 months ago

Figure 2.4.1.1.1: Code management in gitlab



2023-224 › 2023-224 › Repository › Branches

Protected branches can be managed in project settings.

Active branches

master default protected
b7f08681 · updated · 2 days ago

IT20068196-Anubama.L 62 1
9d0dfe54 · Women updated1 · 1 month ago

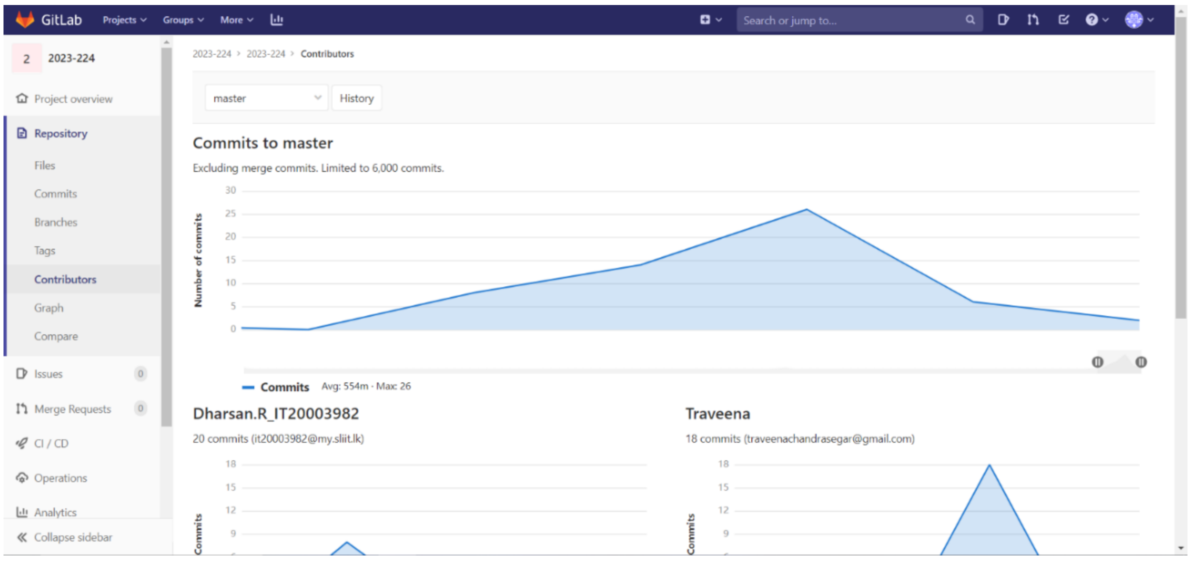
Stale branches

IT20003982-Dharsan.R merged 62 0
a61a35be · Update README.md · 5 days ago

IT20001452-Traveena.C merged 62 0
a61a35be · Update README.md · 5 days ago

IT19980928-Krishanthini.M merged 62 0
a61a35be · Update README.md · 5 days ago

Figure 2.4.1.1.2: Merge branches



Commits to master

Excluding merge commits. Limited to 6,000 commits.

Commits Avg: 554m · Max: 26

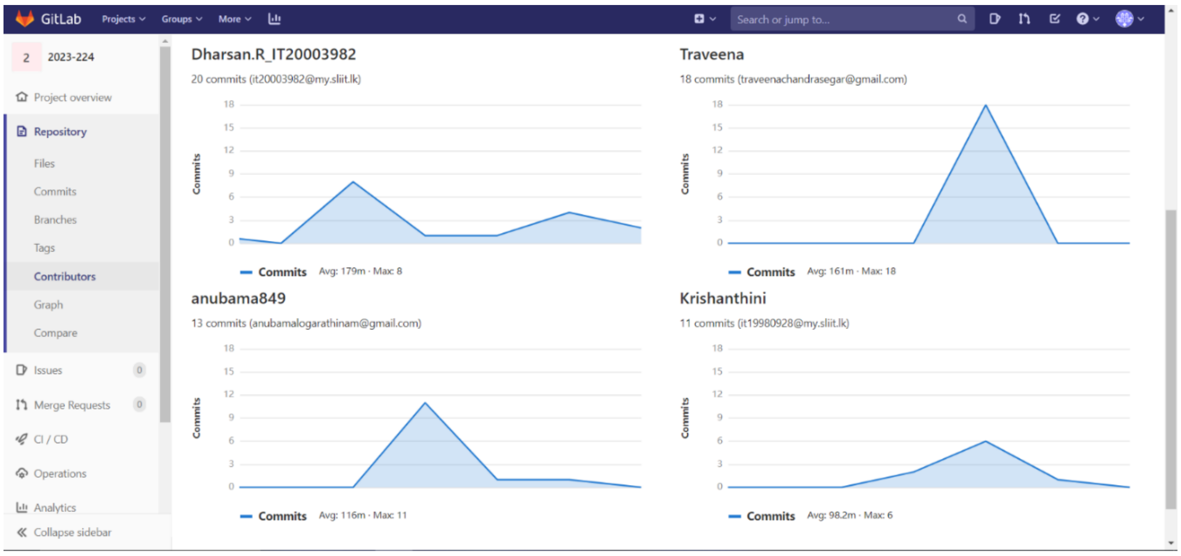
Dharsan.R_IT20003982

20 commits (it20003982@my.sliit.lk)

Traveena

18 commits (traveenachandrasegar@gmail.com)

Figure 2.4.1.1.3: Overall commits



Dharsan.R_IT20003982

20 commits (it20003982@my.sliit.lk)

Commits Avg: 179m · Max: 8

Traveena

18 commits (traveenachandrasegar@gmail.com)

Commits Avg: 161m · Max: 18

anubama849

13 commits (anubamalogarathinam@gmail.com)

Commits Avg: 116m · Max: 11

Krishanthini

11 commits (it19980928@my.sliit.lk)

Commits Avg: 98.2m · Max: 6

Figure 2.4.1.1.4: Individual commits

1.5 Requirement Gathering

The primary aim of this system is to serve the requirements of the police force, without any commercial objectives. In our pursuit of gathering the necessary data for each system component, we engaged in collaborative efforts with key stakeholders. We initiated discussions with the Deputy Inspector General (DIG) in Kurunegala, resulting in the acquisition of essential datasets. Additionally, we sought the expertise of legal professionals and senior lecturers from the Faculty of Law in Colombo, who graciously contributed valuable data. The datasets we meticulously collected encompassed accident and crime-related information, as well as case documents. This comprehensive dataset compilation exceeded five thousand records for each component. Importantly, the data we gathered strictly pertains to case details and does not involve any personally identifiable information, ensuring the utmost privacy and security.

This extensive dataset forms the cornerstone of our machine learning algorithms, enabling us to derive accurate predictions and conduct in-depth analyses. Users of our system will have the opportunity to visualize these outcomes through graphical representations. It's worth noting that the process of dataset acquisition presented significant challenges, necessitating a year-long effort and close collaboration with law enforcement authorities, which ultimately culminated in the successful completion of our project.

Our requirements encompass both functional and non-functional aspects, which we will elaborate on in the sections below.

1.5.1 Functional Requirements:

- Functional requirements for accident case analysis include predicting accident percentages for future years and divisions, forecasting accident percentages and causes for future years, conducting accident pattern analysis to identify high-risk days, hours, and months, and offering prevention strategies based on analysis results.
- Functional requirements for crime analysis entail an intuitive user interface for querying crime data based on user-provided parameters. The system must implement a Decision Tree machine learning model to predict prevalent crime patterns, affected demographics, vehicles involved, and stolen objects.

Additionally, predictive models for estimating crime rates over the next five years and visualization tools for generating graphical representations of location-specific crimes are essential features.

- The system should collect crime case documents, pre-process text data, and employ NLP and machine learning techniques to identify patterns and relationships in the documents. It must use topic modelling to uncover trends and employ supervised machine learning and text classification for categorization and prediction of new cases, ensuring adaptability for improved accuracy.
- The system must detect patterns and trends in crimes against women, employing clustering based on factors such as location, crime type, and time. It should provide insights and recommendations for prevention and forecast future crimes using historical data.

1.5.2 Non-functional Requirements:

- Performance
- Reliability
- Security
- Usability
- Scalability

1.6 Resources Used

1.6.1.1 Software Boundaries

Backend - Python Language

Backend - Python Language: Python serves as the backbone of the application. It's used to write the server-side logic for handling HTTP requests and responses. In this case, Python is used for processing user inputs, making predictions based on machine learning models, and handling data manipulation tasks. Libraries like Pandas, NumPy, Scikit-Learn, and Django are employed to manage data, perform calculations, and implement machine learning models.

Visual Studio Code Editor

Visual Studio Code (VS Code) is the integrated development environment (IDE) where Python scripts are written and developed. It provides features like code autocompletion, debugging tools, and extensions for Python, making it a powerful tool for writing and managing Python code efficiently.

Frontend – HTML, CSS, JavaScript

HTML, CSS, JavaScript: The frontend of the web application is built using HTML, CSS, and JavaScript. HTML is used for creating the structure and content of web pages. CSS is used for styling the user interface, making it visually appealing. JavaScript is employed for adding interactivity to the web application, such as handling user inputs and triggering requests to the backend.

Framework – Django

Within the realm of software boundaries, we have harnessed Django, a high-level Python web framework renowned for its ability to expedite the development of secure and robust websites. Crafted by seasoned developers, Django alleviates many of the complexities associated with web development, enabling us to focus on building our application without the need to reinvent the wheel. It's important to note that all four components seamlessly operate within the Django framework, facilitating cohesion and consistency across the system.

Web Application

The web application acts as the user interface through which users interact with the backend component. Users can input data, select options, and trigger predictions through web forms. The web application communicates with the Python backend by sending HTTP requests with user inputs and receives responses containing prediction results.

In summary, Python and VS Code handle the backend logic and machine learning, while HTML, CSS, and JavaScript are used for creating an interactive and visually pleasing frontend. Together, these technologies enable the development of a user-friendly web application for crime prediction and prevention.

Libraries Used

1. **Pandas** - Pandas is a versatile library that provides data structures like DataFrames and Series. It's essential for data manipulation tasks such as data cleaning, transformation, and analysis. It simplifies working with structured data and supports various data sources.

2. **Matplotlib and Seaborn -** Matplotlib is a widely-used library for creating static, animated, or interactive visualizations in Python. Seaborn is built on top of Matplotlib and specializes in creating informative and attractive statistical graphics. Together, they facilitate data exploration and presentation.

3. **Scikit-Learn (sklearn) -** Scikit-Learn is a comprehensive machine learning library offering tools for classification, regression, clustering, and more. The imported classifiers, such as LinearSVC and MultinomialNB, are used for building machine learning models. It also includes modules for feature extraction, selection, and model evaluation.

4. **Django** - Django is a robust web framework that simplifies web application development. It follows the Model-View-Controller (MVC) architectural pattern and provides built-in tools for handling HTTP requests, managing databases, and rendering views. Django is particularly well-suited for building data-driven web applications.

5. **Numpy -** Numpy is the fundamental library for numerical computing in Python. It offers efficient array operations and mathematical functions, making it indispensable for scientific and mathematical applications.

6. **PyPDF2** - PyPDF2 is used to extract text and perform operations on PDF documents. It's useful for parsing and extracting data from PDF files, which can be valuable in various applications, including data analysis and document processing.

7. **NLTK (Natural Language Toolkit) -** NLTK is a comprehensive library for natural language processing (NLP) tasks. It includes tokenizers, stemmers, and access to linguistic resources, making it a valuable resource for text analysis and processing**.**

8. **Neattext -** Neattext simplifies text pre-processing and cleaning tasks. It provides functions to remove unwanted characters, normalize text, and perform various cleaning operations, enhancing the quality of text data for analysis.

9. **String -** The `string` library provides constants and utilities for working with strings, including character sets like ASCII letters and punctuation. It's handy for text manipulation tasks.

10. **Tempfile and Os -** Tempfile and Os are essential for managing temporary files and handling file paths. They ensure efficient and safe file operations within the application.

11. **Urllib and Base64 -** Urllib is used for making HTTP requests and handling URLs. Base64 encodes and decodes binary data, which can be valuable for encoding and decoding data for transmission or storage.
12. **IO -** The `io` module provides classes and functions for handling input and output operations. It's often used for working with streams and data buffers.

13. **Pickel -** The `pickle` library is used for serializing (pickling) and deserializing (unpickling) Python objects. It's valuable for storing and retrieving complex data structures.

14. **Base64 (import base64) -** Base64 is used to encode binary image data into a format that can be embedded directly into HTML and displayed as images in the web application.

15. **CSV (import csv) -** The CSV module is used for reading and writing CSV (Comma-Separated Values) files. It's used to add new data entries to the dataset.

16. **Random (import random) -** The Random module is used for generating random values. In this project, it's used to generate random colours for data visualization.

17. **Plotly Express (import plotly.express as px) -** Plotly Express is a library for interactive data visualization. It's used for creating interactive charts and graphs, although it's not used extensively in this code.

18. **statsmodels.tsa. arima.model.ARIMA:** This is a part of the stats models library and is specifically used for fitting ARIMA (AutoRegressive Integrated Moving Average) models. ARIMA is employed for time series forecasting and is crucial for predicting future accident percentages.

19. **sklearn.cluster.KMeans:** This module is part of scikit-learn, a machine learning library in Python. It's used for K-means clustering, which groups data points into clusters based on similarity.

1.7 Commercialization aspects of the product

The proposed system is intended for a service purpose, with the goal of leveraging machine learning techniques to address important societal issues related to public safety and crime prevention. By providing accurate and reliable analysis, prediction, and classification of accident and crime-related data, the system has the potential to help law enforcement agencies, policy makers, and other stakeholders make informed decisions, allocate resources effectively, and take proactive measures to prevent and reduce crime. Furthermore, the system's ability to identify patterns and commonalities across cases can facilitate the development of targeted intervention strategies and support efforts to improve public safety in a more holistic and comprehensive manner.

1.8 Testing and Implementation

1.8.1 Testing

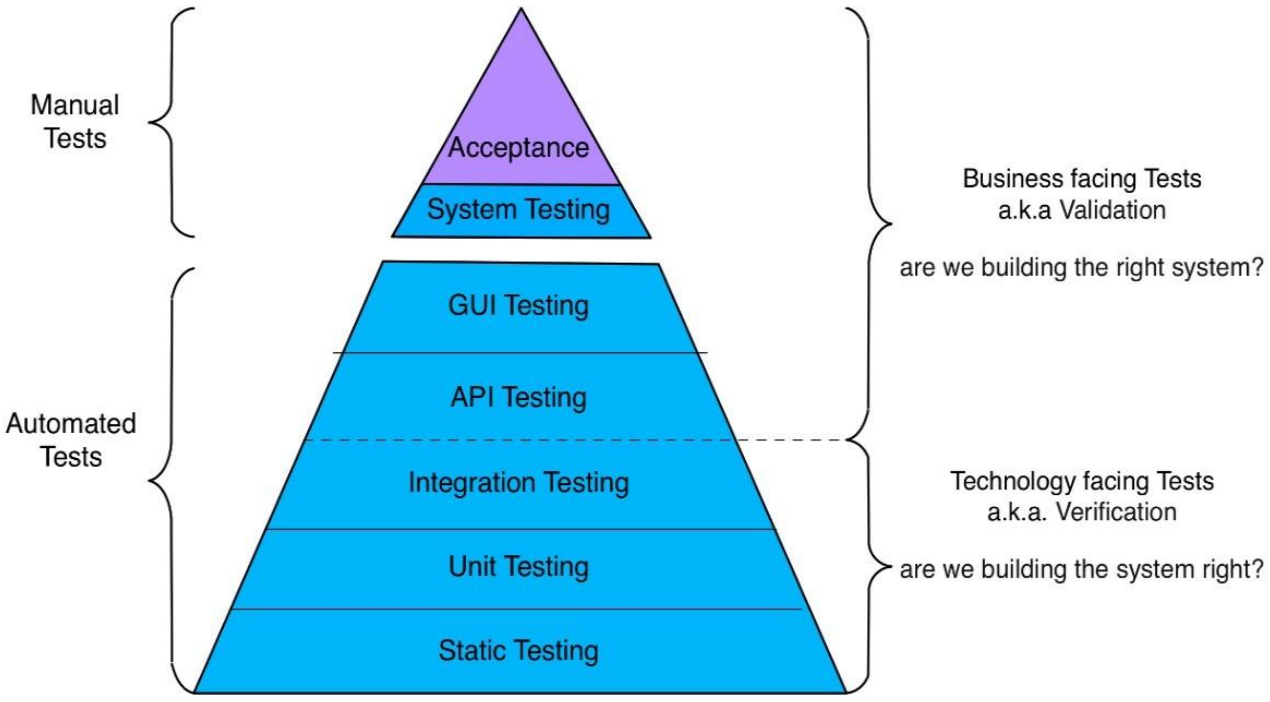


Figure 2.8.1.1: Testing Phase.

The diagram above illustrates a systematic approach to software testing at each developmental stage, encompassing applications and defect resolution. This comprehensive testing methodology consistently enhances the quality of applications, ensuring they are top-notch deliverables. Testing is a fundamental component of the software development life cycle, serving to unearth and rectify issues that may have silently accumulated during various development phases. Furthermore, it is imperative to validate the quality of the product or software application. Regardless of the specific component in our SDLC, which includes Accident Case Analysis, Crime Analysis, Case Document Classification, and Crimes Against Women Analysis, each stage must undergo thorough testing to guarantee its functionality and reliability.

1. **Unit Testing**

Unit testing is a crucial phase in our project's development, ensuring the functionality and accuracy of each component in isolation. Starting with the Accident Case Analysis component, unit tests meticulously verify data pre-processing, ARIMA model predictions, and the effectiveness of visualization tools, guaranteeing precise accident percentage and pattern forecasts. Moving to the Crime Analysis component, unit tests assess the accuracy of data processing, Decision Tree model predictions, and visualization tools, validating precise crime pattern and prediction outcomes. In the Case Document Classification component, the testing process rigorously evaluates the system's ability to accurately categorize case documents based on various factors like content, metadata, and context. Lastly, in the Crimes Against Women Analysis component, unit tests ensure that clustering algorithms accurately group similar cases and predictive models provide precise forecasts using historical data. These tests also validate the correctness of data processing and transformation functions, critical for accurate analysis in all components.

2. **Module Testing**

In module testing, each of the four key components of our system, namely Accident Case Analysis, Crime Analysis, Case Document Classification, and Crimes Against Women Analysis, undergoes individual evaluation. This phase verifies that each component functions correctly in isolation, addressing specific functionalities and predictions. By examining and validating the accuracy of these individual modules, we ensure their reliability and effectiveness within the larger system Integration Testing.

3. **Integration Testing**

Integration testing is the subsequent phase, where the interactions between these components are rigorously tested. It assesses how well these modules work together, ensuring that data flows seamlessly between them and that they collectively produce accurate and coherent outcomes. This step helps identify and resolve any integration-related issues, guaranteeing the smooth operation of the integrated web application.

4. **System Testing**

System testing encompasses a comprehensive evaluation of the entire integrated web application. It verifies that the system functions as a cohesive whole, accurately predicting accident percentages, crime patterns, and document classifications. This phase validates the system's ability to provide valuable insights and user-friendly visualizations. System testing is essential to ensure the reliability and robustness of the entire platform.

5. **User Acceptance Testing**

User Acceptance Testing (UAT) is a critical phase where the system is put to the test by actual users, such as police departments and investigators. During UAT, users interact with the system, input parameters, and assess its performance, usability, and the accuracy of predictions. Feedback and observations from users play a pivotal role in refining and fine-tuning the system to meet their needs effectively.

6. **Maintenance**

The maintenance phase represents an ongoing commitment to keeping the system up to date and ensuring its long-term reliability. This involves addressing any issues that arise during real-world usage, updating datasets, and incorporating user feedback for continuous improvement. Maintenance ensures that the system remains a valuable and dependable tool for law enforcement agencies and investigators over time.

Our integrated system, consisting of the Accident Case Analysis, Crime Analysis, Case Document Classification, and Crimes Against Women Analysis components, has successfully passed through all testing phases without encountering errors. During the testing process, we divided the entire system into its constituent components, allowing for a realistic and thorough examination of each part. This approach ensured that our system was rigorously tested, guaranteeing its reliability and accuracy in delivering valuable insights and predictions to law enforcement agencies and investigators.

Test cases that are done for each testing method is shown below.

Test Case No	Test Case 01
Description	Predict Accident Percentage for specified division name and future year
Test Steps	1. Login to the system. 2. Navigate to accident 3. Navigate to accident prediction section 4. Type or select the division name 5. Select the future year 6. Click predict button
Test Data	String
Expected Result	Successfully predicting the accident percentage and fatal percentage for future years
Actual Result	Pass
User Role	Police

Table 2.8.1.1 Predict Accident Percentage Test Case 01

Test Case No	Test Case 02
Description	Predict Accident and fatal percentage for future years
Test Steps	1. Login to the system. 2. Navigate to accident 3. Navigate to accident prediction section 4. Enter or select division name 5. Click analysis button
Test Data	String
Expected Result	Predicted percentages for accident and fatal for future years displayed and graph also displayed.
Actual Result	Pass
User Role	Police

Table 2.8.1.2 Predict accident and fatal percentage for future years - Test Case 02

Test Case No	Test Case 03
Description	Predict cause of the accident and the percentage for each cause
Test Steps	1. Login to the system. 2. Navigate to accident 3. Navigate to cause prediction section 4. Enter or select future year 5. Click submit
Test Data	String

Expected Result	Predicted percentage for each cause should be displayed and the graph for those values should be displayed
Actual Result	Pass
User Role	Police

Table 2.8.1.3 Cause Prediction - Test Case 03

Test Case No	Test Case 04
Description	Provide prevention strategies
Test Steps	1. Login to the system. 2. Navigate to accident 3. Navigate to cause prediction section 4. Click prevention strategy button 5. Select cause 6. Click submit
Test Data	String
Expected Result	All prevention strategies displayed
Actual Result	Pass
User Role	Police

Table 2.8.1.4 Prevention Strategies Test Case 04

Test Case No	Test Case 05
Description	Select the type of analysis in the pattern analysis section

Test Steps	1. Login to the system. 2. Navigate to accident. 3. Navigate to pattern analysis section. 4. Select one option from (Day, Hour, Month).
Test Data	String
Expected Result	Predicted and statistical graph for the selected option and the most accident occurring day/hour/month should be displayed
Actual Result	Pass
User Role	Police

Table 2.8.1.5 Pattern Analysis Test Case 05

Test Case No	Test Case 06
Description	Verify whether empty validation message is fired for each empty mandatory field.
Test Steps	1. Login to the system. 2. Navigate to the ‘crime’ option. 3. Click on the ‘Predict Crime Pattern’ option. 4. Submit the form by clicking on the ‘Predict’ button without providing values to the fields.
Test Data	String, Integer

Expected Result	The form should not be submitted with missing data. Validation messages should get fired for each empty mandatory field
Actual Result	Pass
User Role	Police team.

Table 2.8.1.6 Ensure the form is not submitted with missing data - Test Case 06

Test Case No	Test Case 07
Description	Verify whether error message is fired when user enters invalid year is given as input (e.g., year that is not in the range).
Test Steps	1. Login to the system. 2. Navigate to the ‘Crime’ option. 3. Click on the ‘Predict Crime Pattern’ option. 4. Enter ‘2050’ in the year field. 5. Submit the form by clicking on the ‘Predict’ button.
Test Data	Integer
Expected Result	Verify that appropriate error messages are displayed.

	Ensure the form is not submitted with invalid data.
Actual Result	Pass
User Role	Police team.

Table 2.8.1.7 Form Submission with Invalid Data - Test Case 07

Test Case No	Test Case 08
Description	Verify whether correct form of data is submitted
Test Steps	1. Login to the system. 2. Navigate to the ‘Crime’ option. 3. Click on the ‘Predict Crime Rate” 4. Input text for date 5. Click on the ‘Predict’ button.
Test Data	String, Integer
Expected Result	The system should pop up error message. The system should not predict any outcomes.
Actual Result	Pass
User Role	Police team.

Table 2.8.1.8 Display of Prediction Results - Test Case 08

Test Case No	Test Case 09
Description	Verify that the pattern of the crime is displayed to specific input given
Test Steps	1. Login to the system. 2. Navigate to the 'Crime' option. 3. Click on the 'Predict Crime Pattern. 4. Select 'Area', 'Month' options from the dropdown. 5. Given the desired Year. 6. Click on the 'Predict' button. 7. Predicted pattern for each crime is displayed.
Test Data	String
Expected Result	The system should Display the area, month and year selected with the other information.
Actual Result	Pass
User Role	Police team.

Table 2.8.1.9 Functionality of Prevention Strategies Test Case 09

Test Case No	Test Case 10
Description	Verify whether error message displayed when user does not select any option in a checkbox field of a form.
Test Steps	5. Login to the system.

	6. Navigate to the "Crime Classifier" option. 7. Click on the 'ADD DATA' option. 8. Not select any option in the 'Categories' feild 1. Submit the form by clicking on the 'Upload' button.
Test Data	String
Expected Result	The form should not be submitted with missing data. Validation messages should get fired for each empty mandatory field
Actual Result	Pass
User Role	Investigation Team.

Table 2.8.1.10 Ensure the form is not submitted with missing data Manual Test Case 10

Test Case No	Test Case 11
Description	Verify whether message displayed after re-train model is successful
Test Steps	1. Login to the system. 2. Navigate to the "Crime Classifier" option. 3. Click on the 'ADD DATA' option.

	4. Submit the form by clicking on the ‘Upload’ button. 5. Click on the ‘re-train Model’ button.
Test Data	String
Expected Result	The form should be submitted. Alert message should be displayed and accepted after the re-training of model
Actual Result	Pass
User Role	Investigation Team.

Table 2.8.1.11 Ensure success message is displayed after re-training Manual Test Case 11

Test Case No	Test Case 12
Description	Verify whether after a successful form submission, the prediction results are displayed correctly.
Test Steps	1. Login to the system. 2. Navigate to the ‘Crime Classifier’ option. 3. Select Pdf document that need to predict category. 4. Click ‘Upload’ button. 5. Click on the ‘Predict Category’ button.
Test Data	String, Integer

Expected Result	The predicted category for the uploaded document should be displayed.
Actual Result	Pass
User Role	Investigation Team.

Table 2.8.1.12 Display Category Prediction - Test Case 12

Test Case No	Test Case 13
Description	Verify that you are redirected to the correct page, with data that is newly added to the through 'ADD DATA' option.
Test Steps	1. Login to the system. 2. Navigate to the "Crime Classifier" option. 3. Click on the 'ADD DATA' option. 4. Submit the form by clicking on the 'Upload' button.
Test Data	String
Expected Result	Should redirect to the 'Preview Data' page. Newly added data should be there as the last record of the data table
Actual Result	Pass
User Role	Investigation Team.

Table 2.8.1.13 Functionality Add data Page - Test Case 13

Test Case No	Test Case 14
Description	Verify whether empty validation message is fired for each empty mandatory field.
Test Steps	1. Login to the system. 2. Navigate to the 'Women-crime' option. 3. Click on the 'Add data' option. 4. Submit the form by clicking on the 'Upload' button without providing values to the fields.
Test Data	String, Integer
Expected Result	The form should not be submitted with missing data. Validation messages should get fired for each empty mandatory field
Actual Result	Pass
User Role	Police team.

Table 2.8.1.14 Ensure the form is not submitted with missing data Manual Test Case 14

Test Case No	Test Case 15
Description	Verify whether error message is fired when user enters invalid data in one or more fields (e.g., text in the "Year" field).
Test Steps	1. Login to the system. 2. Navigate to the 'Women-crime' option.

	3. Click on the ‘Add data’ option. 4. Enter text in ‘Year’ field or crime fields(e.g, Rape, Kidnapping and Abduction etc) 5. Submit the form by clicking on the ‘Upload’ button.
Test Data	Integer
Expected Result	Verify that appropriate error messages are displayed. Ensure the form is not submitted with invalid data.
Actual Result	Pass
User Role	Police team.

Table 2.8.1.15 Form Submission with Invalid Data Manual Test Case 15

Test Case No	Test Case 16
Description	Verify whether after a successful form submission, the prediction results are displayed correctly.
Test Steps	1. Login to the system. 2. Navigate to the ‘Women-crime’ option. 3. Click on the ‘Highest Crime Prediction’ option. 4. Select ‘Division’, ‘Year’ options from the dropdown. 5. Click on the ‘Predict’ button.

Test Data	String, Integer
Expected Result	The predicted counts for each crime type should be shown. Highest predicted crime and its count should be displayed.
Actual Result	Pass
User Role	Police team.

Table 2.8.1.16 Display of Prediction Results – Test Case 16

Test Case No	Test Case 17
Description	Verify that you are redirected to the correct page, possibly with information related to preventing the highest predicted crime.
Test Steps	1. Login to the system. 2. Navigate to the ‘Women-crime’ option. 3. Click on the ‘Highest Crime Prediction’ option. 4. Select ‘Division’, ‘Year’ options from the dropdown. 5. Click on the ‘Predict’ button. 6. Predicted counts for each crime type are shown
Test Data	String
Expected Result	‘Prevention Strategies’ option should be displayed.

	User should be redirected to the correct page, possibly with information related to preventing the highest predicted crime.
Actual Result	Pass
User Role	Police team.

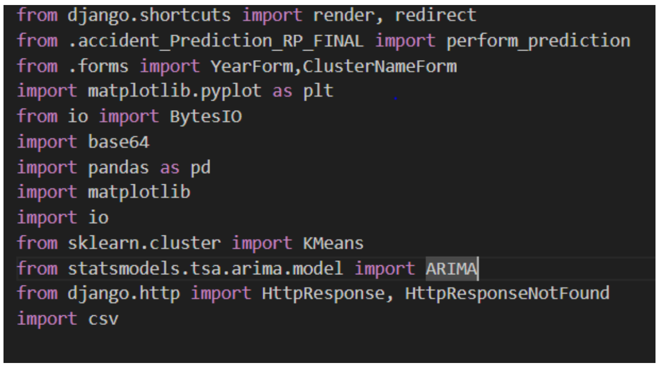
Table 2.8.1.17 Functionality of Prevention Strategies Link Manual Test Case 17

1.8.2 Implementation

In the implementation phase, we leveraged the wealth of data collected during the project's research and development stages. The extensive datasets gathered for each component, including accident and crime-related data, case documents, and demographic information, served as the foundation for building our integrated web application. Utilizing the Django framework with Python, we seamlessly translated our research findings into a functional and user-friendly system. This implementation not only enables us to predict accident percentages for future years and analyze crime patterns but also offers robust data classification and analysis capabilities. By harnessing machine learning algorithms, statistical models, and data visualization tools, we have created a powerful platform that empowers law enforcement agencies and investigators with the insights needed to enhance safety planning, crime prevention, and decision-making. Our implementation process ensures that our system is well-equipped to address the complex challenges in the fields of accident case analysis, crime analysis, case document classification, and crimes against women analysis, ultimately contributing to improved public safety and law enforcement effectiveness.

Code

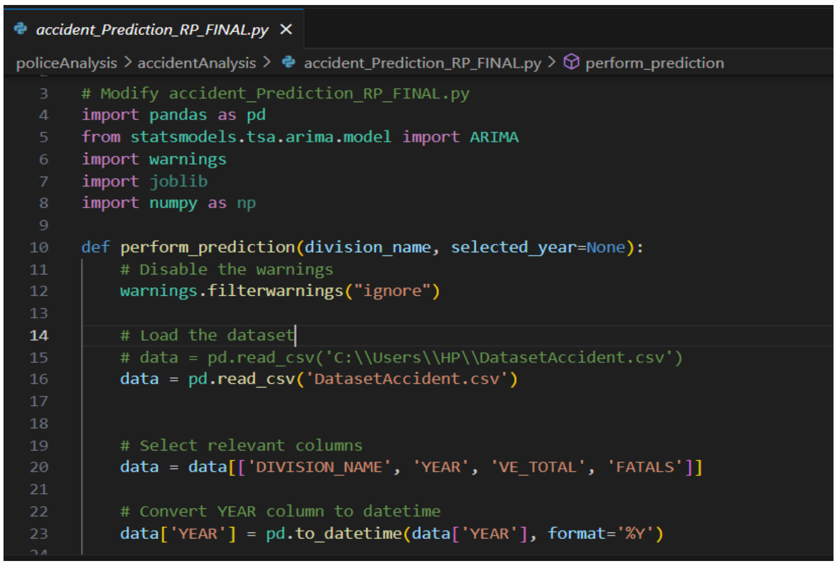
Accident case analysis implementation



```python
from django.shortcuts import render, redirect
from .accident_Prediction_RP_FINAL import perform_prediction
from .forms import YearForm,ClusterNameForm
import matplotlib.pyplot as plt
from io import BytesIO
import base64
import pandas as pd
import matplotlib
import io
from sklearn.cluster import KMeans
from statsmodels.tsa.arima.model import ARIMA
from django.http import HttpResponse, HttpResponseNotFound
import csv
```

Figure 2.8.2.1: Import libraries for accident prediction

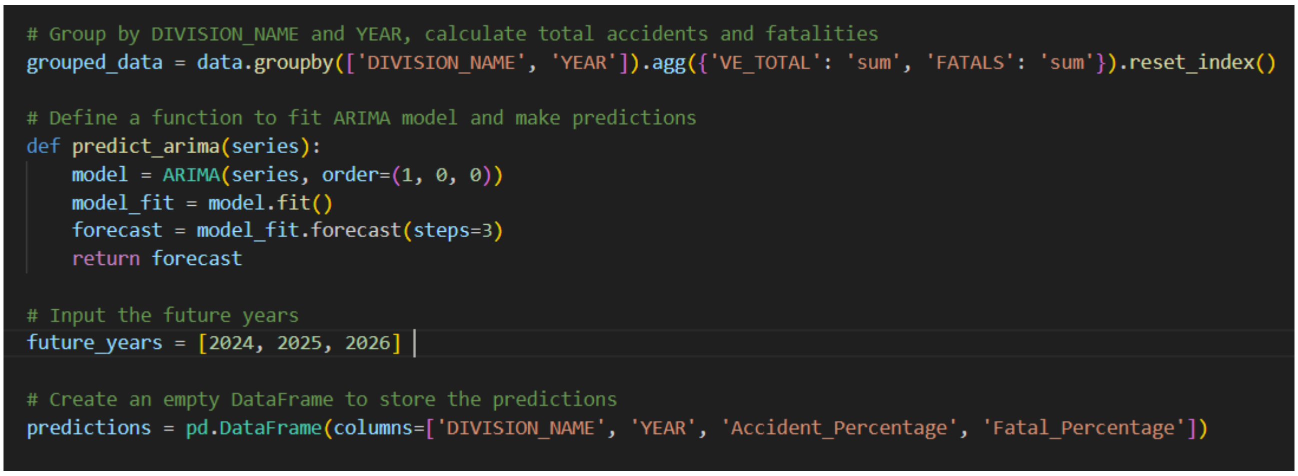
The code snippets imported in our component play a pivotal role in enabling the functionality of our integrated web application. We utilize Django for web framework support, allowing seamless user interactions. The 'accident_Prediction_RP_FINAL' module integrates predictive analysis using the ARIMA model to forecast accident percentages and fatal percentages for future years and divisions. We employ forms, such as 'YearForm' and 'ClusterNameForm,' to gather user inputs for customized queries. Data visualization is facilitated by 'matplotlib' to generate graphical representations for user insights. Data manipulation and analysis are performed with 'pandas,' while 'io' aids in data handling. The 'KMeans' module from 'scikit-learn' is used for K-means clustering, and 'statsmodels' is employed for time series analysis using ARIMA. The code ensures a seamless user experience and accurate predictions, enhancing the overall functionality of our system.


accident_Prediction_RP_FINAL.py
policeAnalysis > accidentAnalysis > accident_Prediction_RP_FINAL.py > perform_prediction

```python
# Modify accident_Prediction_RP_FINAL.py
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import warnings
import joblib
import numpy as np

def perform_prediction(division_name, selected_year=None):
    # Disable the warnings
    warnings.filterwarnings("ignore")

    # Load the dataset
    # data = pd.read_csv('C:\\Users\\HP\\DatasetAccident.csv')
    data = pd.read_csv('DatasetAccident.csv')


    # Select relevant columns
    data = data[['DIVISION_NAME', 'YEAR', 'VE_TOTAL', 'FATALS']]

    # Convert YEAR column to datetime
    data['YEAR'] = pd.to_datetime(data['YEAR'], format='%Y')
```

Figure 2.8.2.2: Read dataset and use necessary columns

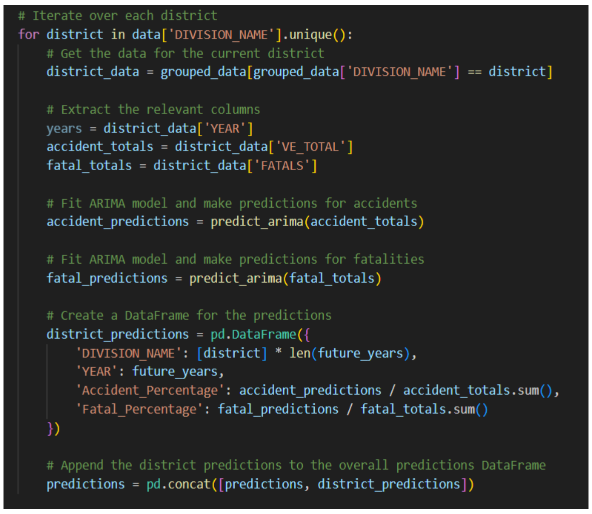
The provided code excerpt is a Python script named 'accident_Prediction_RP_FINAL.py,' which is essential for the accident case analysis component of our system. This script performs predictive analysis using the ARIMA (AutoRegressive Integrated Moving Average) model to forecast accident percentages and fatal percentages for specific divisions and years. It first disables warnings for smoother execution. The script loads the accident dataset, selects relevant columns containing information about division names, years, total accidents (VE_TOTAL), and fatalities (FATALS). It then converts the 'YEAR' column to a datetime format to facilitate time-based analysis.



```
# Group by DIVISION_NAME and YEAR, calculate total accidents and fatalities
grouped_data = data.groupby(['DIVISION_NAME', 'YEAR']).agg({'VE_TOTAL': 'sum', 'FATALS': 'sum'}).reset_index()

# Define a function to fit ARIMA model and make predictions
def predict_arima(series):
    model = ARIMA(series, order=(1, 0, 0))
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=3)
    return forecast

# Input the future years
future_years = [2024, 2025, 2026]

# Create an empty DataFrame to store the predictions
predictions = pd.DataFrame(columns=['DIVISION_NAME', 'YEAR', 'Accident_Percentage', 'Fatal_Percentage'])
```

Figure 2.8.2.3: fit ARIMA model

In this portion of the code, data from the accident dataset is grouped by 'DIVISION_NAME' and 'YEAR' to facilitate subsequent analysis. The grouped data is then aggregated to calculate the total number of accidents (VE_TOTAL) and fatalities (FATALS) for each division and year, creating a structured dataset. Additionally, a function named 'predict_arima' is defined to implement the ARIMA (AutoRegressive Integrated Moving Average) model for predictive analysis. This function takes a time series data series as input, fits the ARIMA model with specified parameters (order= (1, 0, 0)), and forecasts accident trends for future years (in this case, three years ahead). The code specifies the future years of interest (2024, 2025, 2026) and initializes an empty DataFrame named 'predictions' to store the forecasted accident percentages and fatal percentages for these years, categorized by division and year. This code segment lays the foundation for predicting accident-related metrics and generating valuable insights within our integrated web application.



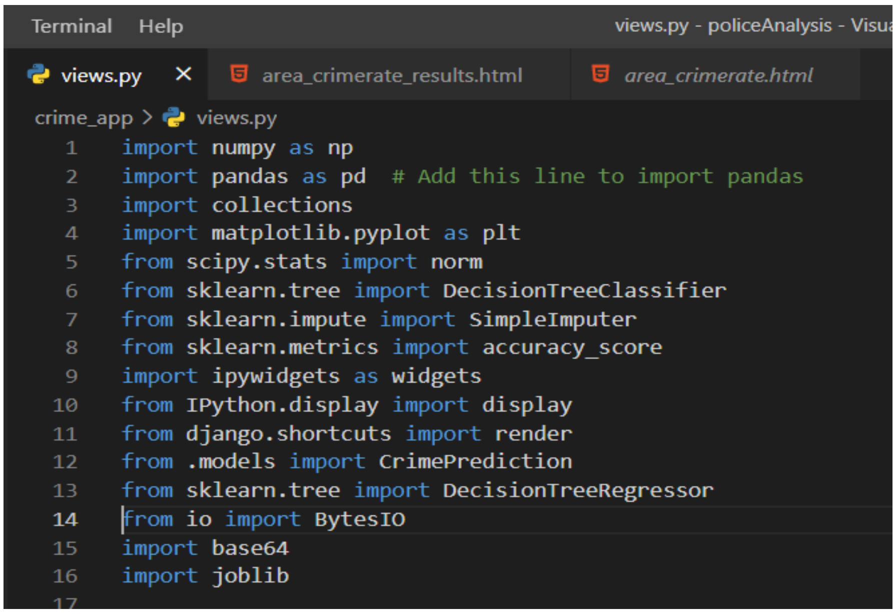
```python
# Iterate over each district
for district in data['DIVISION_NAME'].unique():
    # Get the data for the current district
    district_data = grouped_data[grouped_data['DIVISION_NAME'] == district]

    # Extract the relevant columns
    years = district_data['YEAR']
    accident_totals = district_data['VE_TOTAL']
    fatal_totals = district_data['FATALS']

    # Fit ARIMA model and make predictions for accidents
    accident_predictions = predict_arima(accident_totals)

    # Fit ARIMA model and make predictions for fatalities
    fatal_predictions = predict_arima(fatal_totals)

    # Create a DataFrame for the predictions
    district_predictions = pd.DataFrame({
        'DIVISION_NAME': [district] * len(future_years),
        'YEAR': future_years,
        'Accident_Percentage': accident_predictions / accident_totals.sum(),
        'Fatal_Percentage': fatal_predictions / fatal_totals.sum()
    })

    # Append the district predictions to the overall predictions DataFrame
    predictions = pd.concat([predictions, district_predictions])
```

Figure 2.8.2.4: Predict accident and fatal percentage

In this section of the code, a loop iterates through unique division names within the accident dataset. For each division, the corresponding data is extracted from the previously grouped and aggregated dataset. Relevant columns, including 'YEAR,' 'VE_TOTAL' (total accidents), and 'FATALS' (total fatalities), are selected for further analysis. Two separate ARIMA models are then applied to predict accident and fatality trends within the current division. These predictions are calculated based on historical data for accidents and fatalities. Subsequently, a new DataFrame named 'district_predictions' is created to organize the forecasted accident and fatality percentages for the specified future years (2024, 2025, 2026), attributed to the current division. The remaining codes are added in the appendix.

crime case analysis implementation

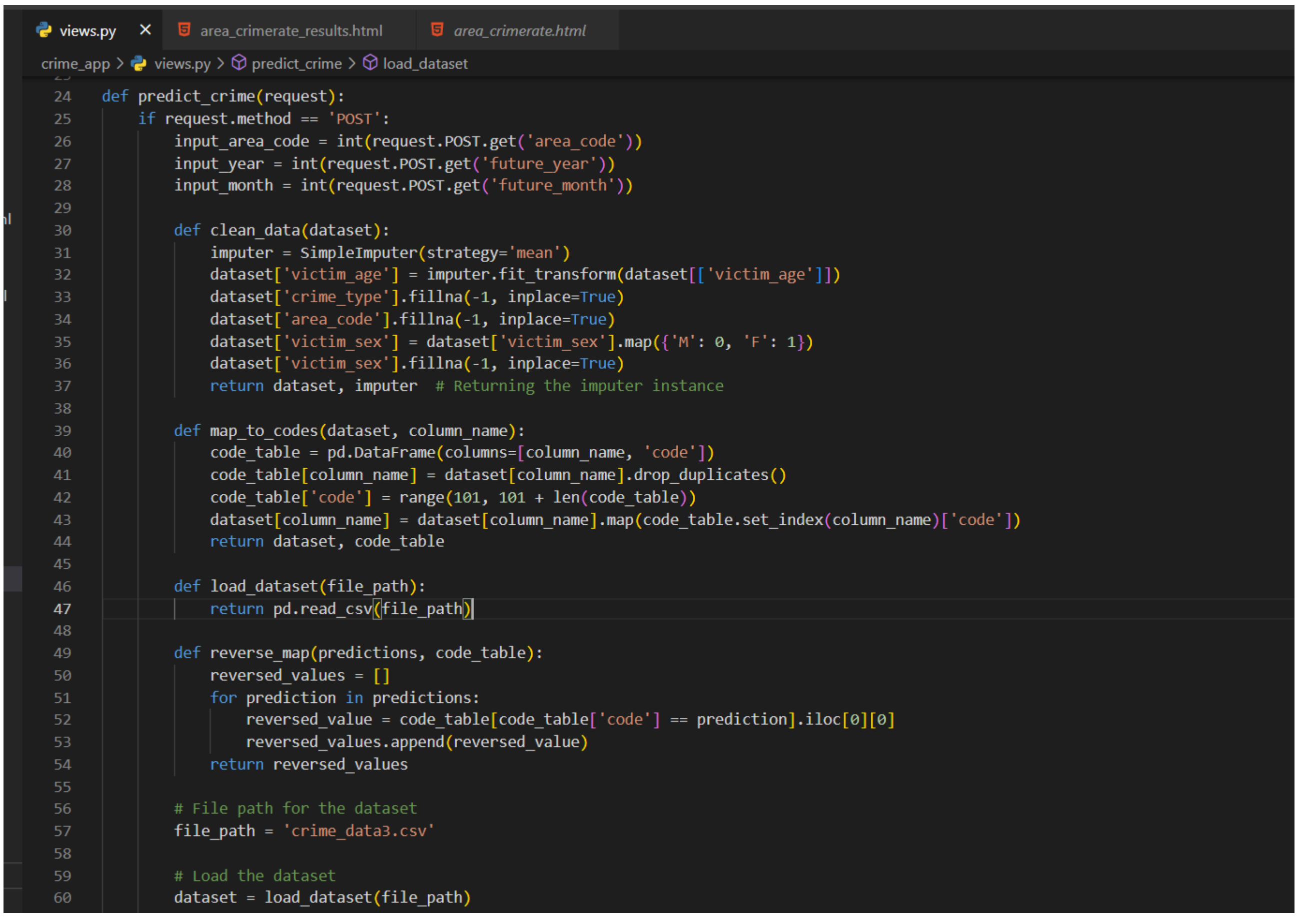
Install the necessary libraries:



```python
import numpy as np
import pandas as pd  # Add this line to import pandas
import collections
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.tree import DecisionTreeClassifier
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
import ipywidgets as widgets
from IPython.display import display
from django.shortcuts import render
from .models import CrimePrediction
from sklearn.tree import DecisionTreeRegressor
from io import BytesIO
import base64
import joblib
```

Figure 2.8.2.5: Import libraries for Clustering crimes against women and crime forecasting prediction

In the development of crime analysis component, these import statements play pivotal roles in crafting the functionality and capabilities of the system. By importing the Pandas library as 'pd,' I harness the power of Pandas for efficient data manipulation and pre-processing, ensuring that crime data is ready for analysis. The integration of the DecisionTreeClassifier and DecisionTreeRegressor classes from Scikit-Learn enables the utilization of decision tree-based machine learning algorithms for both classification and regression tasks, aiding in crime pattern prediction and rate forecasting. Furthermore, the inclusion of SimpleImputer from Scikit-Learn ensures the handling of missing data, enhancing the quality and completeness of the crime dataset. Django's 'render' function, imported from 'django. shortcuts,' serves as a bridge to create the web-based user interface, allowing users to interact with and visualize the results of crime analysis. Additionally, the 'Crime Prediction' model import from Django's database models indicates the integration of a database to store and retrieve crime-related data. Altogether, these imports form the foundation of my crime analysis component, enabling data handling, predictive modelling, web rendering, and database integration.


views.py
area_crimerate_results.html
area_crimerate.html
crime_app > views.py > predict_crime > load_dataset

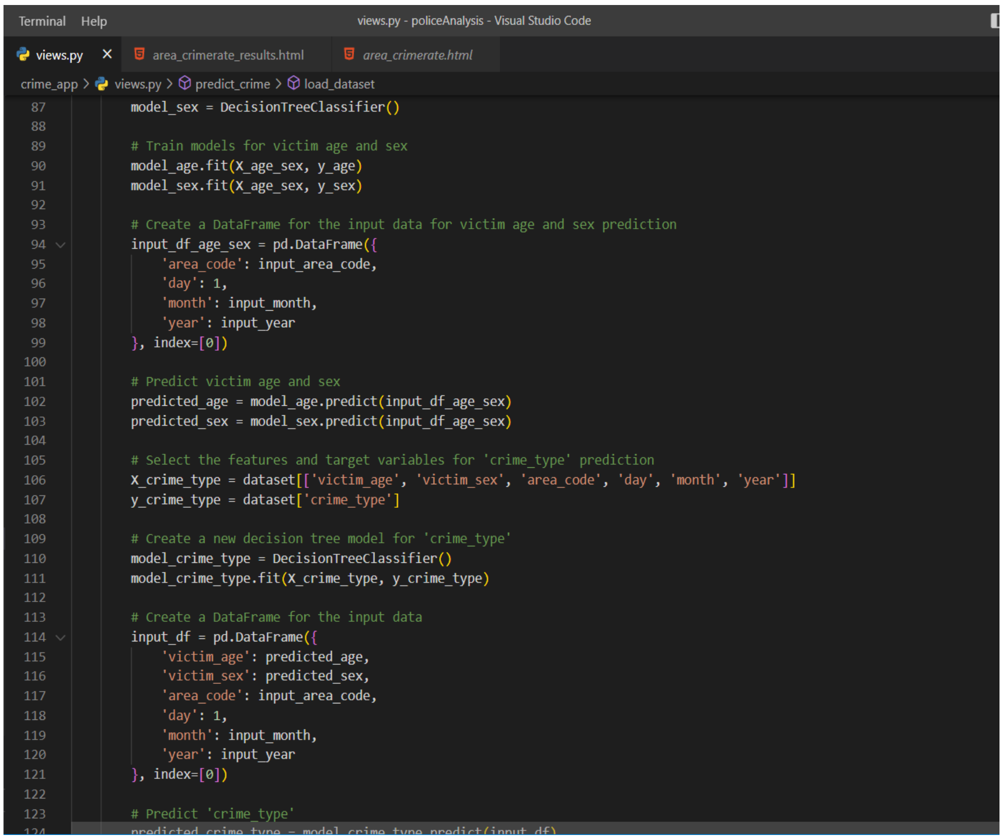
```python
def predict_crime(request):
    if request.method == 'POST':
        input_area_code = int(request.POST.get('area_code'))
        input_year = int(request.POST.get('future_year'))
        input_month = int(request.POST.get('future_month'))

        def clean_data(dataset):
            imputer = SimpleImputer(strategy='mean')
            dataset['victim_age'] = imputer.fit_transform(dataset[['victim_age']])
            dataset['crime_type'].fillna(-1, inplace=True)
            dataset['area_code'].fillna(-1, inplace=True)
            dataset['victim_sex'] = dataset['victim_sex'].map({'M': 0, 'F': 1})
            dataset['victim_sex'].fillna(-1, inplace=True)
            return dataset, imputer  # Returning the imputer instance

        def map_to_codes(dataset, column_name):
            code_table = pd.DataFrame(columns=[column_name, 'code'])
            code_table[column_name] = dataset[column_name].drop_duplicates()
            code_table['code'] = range(101, 101 + len(code_table))
            dataset[column_name] = dataset[column_name].map(code_table.set_index(column_name)['code'])
            return dataset, code_table

        def load_dataset(file_path):
            return pd.read_csv(file_path)

        def reverse_map(predictions, code_table):
            reversed_values = []
            for prediction in predictions:
                reversed_value = code_table[code_table['code'] == prediction].iloc[0][0]
                reversed_values.append(reversed_value)
            return reversed_values

        # File path for the dataset
        file_path = 'crime_data3.csv'

        # Load the dataset
        dataset = load_dataset(file_path)
```

Figure 2.8.2.6: Data Pre-processing for Clustering and Prediction

The predict_crime(request) function is a pivotal component of the crime analysis system, designed to handle HTTP POST requests from a web interface. It extracts and processes user-provided inputs like 'area_code,' 'future_year,' and 'future_month,' which likely serve as parameters for predicting future crime patterns. Within the function, auxiliary functions like 'clean_data' are defined to preprocess the crime dataset by filling missing values, mapping categorical data to numerical codes, and ensuring data quality. Additionally, the 'load_dataset' function is used to import the crime dataset from a specified file path. Collectively, these functionalities enable the system to receive user inputs, prepare the crime data, and potentially employ machine learning models to forecast future crime patterns, forming an integral part of the crime analysis component.

It begins by extracting and transforming data columns, such as 'area_code,' 'crime_type,' 'vehicle_involved,' and 'object_stolen,' into numerical codes for machine learning compatibility. Subsequently, relevant columns for predicting victim age and sex are selected, and Decision Tree models are created and trained for these predictions using the crime dataset.

User-provided inputs, including 'area_code,' 'input_month,' and 'input_year,' are utilized to predict victim age and sex.


Terminal
Help
views.py - policeAnalysis - Visual Studio Code
views.py
area_crimerate_results.html
area_crimerate.html
crime_app > views.py > predict_crime > load_dataset

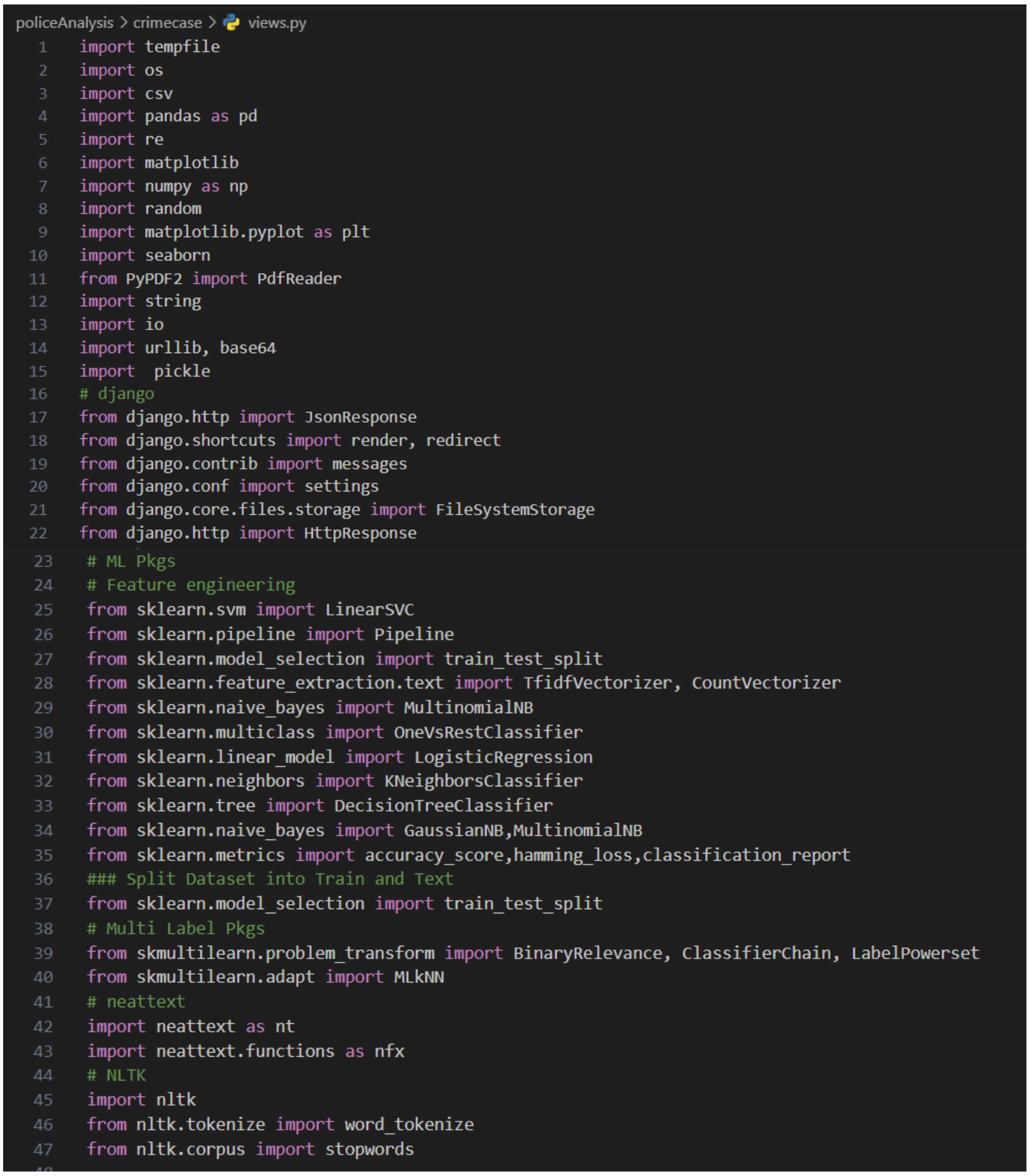
```python
    model_sex = DecisionTreeClassifier()

    # Train models for victim age and sex
    model_age.fit(X_age_sex, y_age)
    model_sex.fit(X_age_sex, y_sex)

    # Create a DataFrame for the input data for victim age and sex prediction
    input_df_age_sex = pd.DataFrame({
        'area_code': input_area_code,
        'day': 1,
        'month': input_month,
        'year': input_year
    }, index=[0])

    # Predict victim age and sex
    predicted_age = model_age.predict(input_df_age_sex)
    predicted_sex = model_sex.predict(input_df_age_sex)

    # Select the features and target variables for 'crime_type' prediction
    X_crime_type = dataset[['victim_age', 'victim_sex', 'area_code', 'day', 'month', 'year']]
    y_crime_type = dataset['crime_type']

    # Create a new decision tree model for 'crime_type'
    model_crime_type = DecisionTreeClassifier()
    model_crime_type.fit(X_crime_type, y_crime_type)

    # Create a DataFrame for the input data
    input_df = pd.DataFrame({
        'victim_age': predicted_age,
        'victim_sex': predicted_sex,
        'area_code': input_area_code,
        'day': 1,
        'month': input_month,
        'year': input_year
    }, index=[0])

    # Predict 'crime_type'
    predicted_crime_type = model_crime_type.predict(input_df)
```

Figure 2.8.2.7: Building a predictive model

The code proceeds to train a Decision Tree model for predicting 'crime_type' based on a combination of features, and another model for 'vehicle_involved' and 'object_stolen.' Predictions are made for each of these crime attributes based on the user's inputs and the models created earlier. The code then reverses the mapping of these predictions to their original categorical values using reference tables. Finally, the predictions and additional details, such as the highest predicted crime, affected gender, age group, vehicles involved, and objects stolen, are stored as a 'CrimePrediction' object in a Django database and rendered to a web page, enabling users to view the results of the crime analysis component.

Case document classification implementation

Install the necessary libraries:



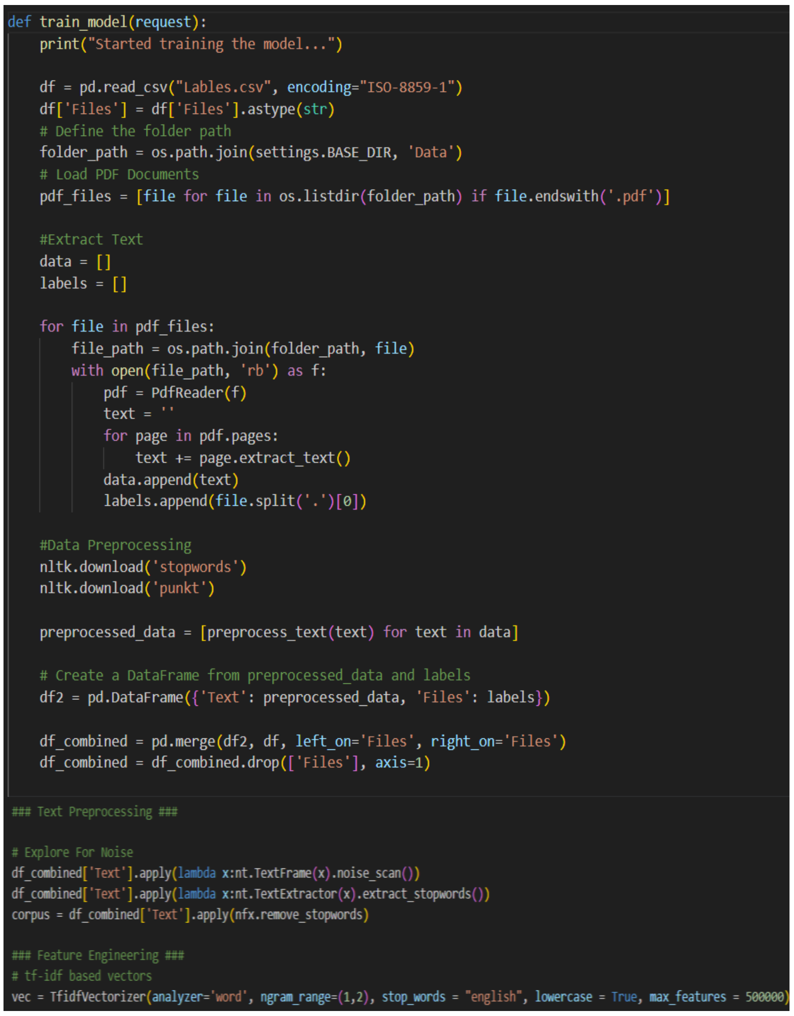
policeAnalysis > crimecase > views.py

```python
import tempfile
import os
import csv
import pandas as pd
import re
import matplotlib
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn
from PyPDF2 import PdfReader
import string
import io
import urllib, base64
import  pickle
# django
from django.http import JsonResponse
from django.shortcuts import render, redirect
from django.contrib import messages
from django.conf import settings
from django.core.files.storage import FileSystemStorage
from django.http import HttpResponse
# ML Pkgs
# Feature engineering
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.metrics import accuracy_score,hamming_loss,classification_report
### Split Dataset into Train and Text
from sklearn.model_selection import train_test_split
# Multi Label Pkgs
from skmultilearn.problem_transform import BinaryRelevance, ClassifierChain, LabelPowerset
from skmultilearn.adapt import MLkNN
# neattext
import neattext as nt
import neattext.functions as nfx
# NLTK
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

Figure 2.8.2.8: Import libraries for Analyse and Classify Similar Case Documents and Predict Category

These imports cover a wide range of functionalities for a Django-based web application with machine learning capabilities. They include libraries for file management (`tempfile`, `os`, `csv`), data manipulation and analysis (`pandas`), data visualization (`matplotlib`, `seaborn`), text processing and natural language processing (`re`, `nltk`), machine learning (`sklearn` for various classifiers and metrics), multi-label classification (`skmultilearn`), text preprocessing (`neattext`), and Django-specific modules for web development.

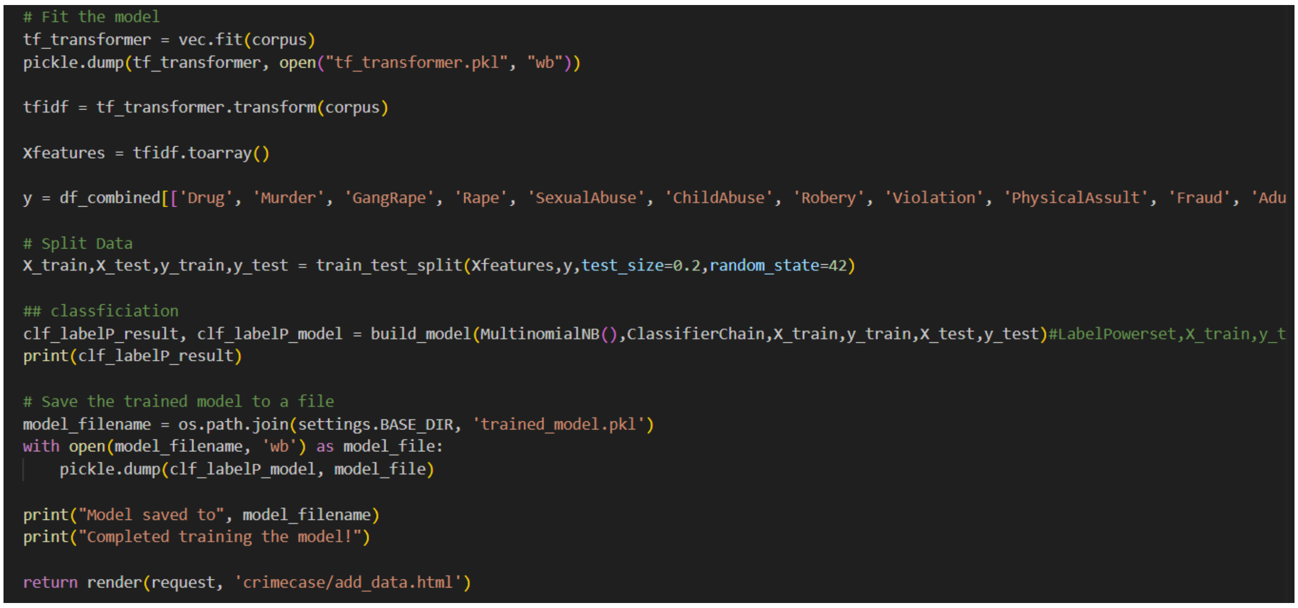
Together, these imports provide a comprehensive toolkit for building a web-based machine learning application that can handle data, text, and model-related tasks.In summary, these imports collectively provide a robust toolkit for building a Django-based web application with machine learning capabilities. They facilitate data handling, analysis, visualization, text processing, model training, and web development tasks, allowing for the development of a feature-rich application that can process, analyze, and visualize data, perform text-related tasks, and make predictions or classifications using machine learning models.



```python
def train_model(request):
    print("Started training the model...")

    df = pd.read_csv("Lables.csv", encoding="ISO-8859-1")
    df['Files'] = df['Files'].astype(str)
    # Define the folder path
    folder_path = os.path.join(settings.BASE_DIR, 'Data')
    # Load PDF Documents
    pdf_files = [file for file in os.listdir(folder_path) if file.endswith('.pdf')]

    #Extract Text
    data = []
    labels = []

    for file in pdf_files:
        file_path = os.path.join(folder_path, file)
        with open(file_path, 'rb') as f:
            pdf = PdfReader(f)
            text = ''
            for page in pdf.pages:
                text += page.extract_text()
            data.append(text)
            labels.append(file.split('.')[0])

    #Data Preprocessing
    nltk.download('stopwords')
    nltk.download('punkt')

    preprocessed_data = [preprocess_text(text) for text in data]

    # Create a DataFrame from preprocessed_data and labels
    df2 = pd.DataFrame({'Text': preprocessed_data, 'Files': labels})

    df_combined = pd.merge(df2, df, left_on='Files', right_on='Files')
    df_combined = df_combined.drop(['Files'], axis=1)

### Text Preprocessing ###

# Explore For Noise
df_combined['Text'].apply(lambda x:nt.TextFrame(x).noise_scan())
df_combined['Text'].apply(lambda x:nt.TextExtractor(x).extract_stopwords())
corpus = df_combined['Text'].apply(nfx.remove_stopwords)

### Feature Engineering ###
# tf-idf based vectors
vec = TfidfVectorizer(analyzer='word', ngram_range=(1,2), stop_words = "english", lowercase = True, max_features = 500000)
```

Figure 2.8.2.9: PDF Text Data Extraction and Pre-processing

This code performs several essential tasks to prepare text data from PDF files for machine learning. It starts by reading information from a CSV file and PDF documents in a folder. For each PDF, it extracts the text content and the associated labels. To make this text data usable, the code performs data pre-processing, which involves removing unnecessary words and noise. It combines this cleaned text data with information from the CSV file. Further text pre-processing steps include identifying and removing noise and stop words from the text. Finally, the code transforms the text into numerical vectors using the TF-IDF technique, making it ready for machine learning. In simpler terms, this code sets the stage for building a machine learning model that can analyze and make predictions based on the text content of these PDF documents, which could be valuable for various applications like text classification or information extraction.



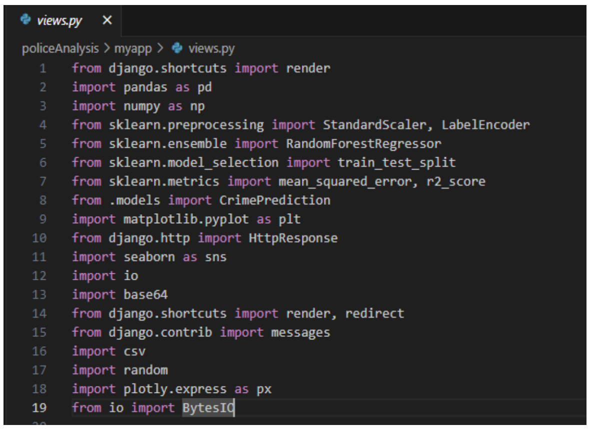
```
# Fit the model
tf_transformer = vec.fit(corpus)
pickle.dump(tf_transformer, open("tf_transformer.pkl", "wb"))

tfidf = tf_transformer.transform(corpus)

Xfeatures = tfidf.toarray()

y = df_combined[['Drug', 'Murder', 'GangRape', 'Rape', 'SexualAbuse', 'ChildAbuse', 'Robery', 'Violation', 'PhysicalAssult', 'Fraud', 'Adu

# Split Data
X_train,X_test,y_train,y_test = train_test_split(Xfeatures,y,test_size=0.2,random_state=42)

## classficiation
clf_labelP_result, clf_labelP_model = build_model(MultinomialNB(),ClassifierChain,X_train,y_train,X_test,y_test)#LabelPowerset,X_train,y_t
print(clf_labelP_result)

# Save the trained model to a file
model_filename = os.path.join(settings.BASE_DIR, 'trained_model.pkl')
with open(model_filename, 'wb') as model_file:
    pickle.dump(clf_labelP_model, model_file)

print("Model saved to", model_filename)
print("Completed training the model!")

return render(request, 'crimecase/add_data.html')
```

Figure 2.8.2.10: Text Data Transformation, Model Training, and Saving for Crime Case Categorization

Here taking several essential steps to prepare and train a machine learning model for a specific application. First, we transform the text data from PDF documents into a numerical format known as TF-IDF, making it suitable for machine learning. Additionally, we organize and prepare labels that represent different categories or types of cases. Next, we split our data into two parts: a training set for teaching the model and a testing set for evaluating its performance. We then utilize the Multinomial Naive Bayes algorithm and Classifier Chain technique to build a predictive model capable of assigning labels to new, unseen cases. Finally, we save this trained model to a file, ensuring it can be easily accessed and utilized in the future. In summary, this code plays a crucial role in our research, enabling us to analyze and categorize crime cases based on their textual descriptions effectively.

Crimes against women implementation

Install the necessary libraries:

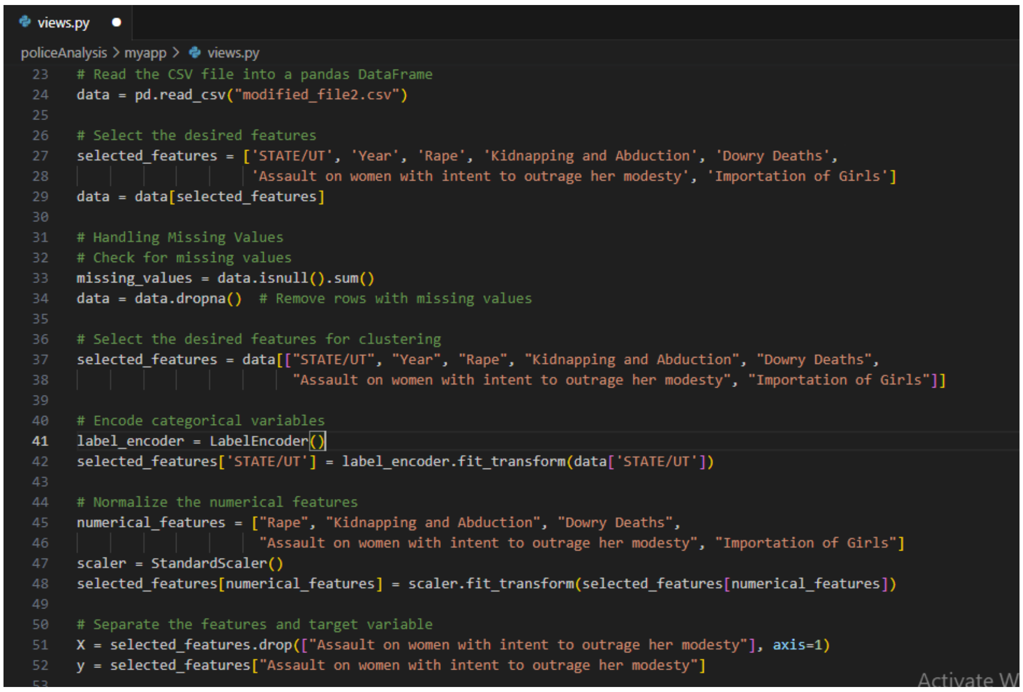

views.py ×

policeAnalysis > myapp > views.py

```python
from django.shortcuts import render
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from .models import CrimePrediction
import matplotlib.pyplot as plt
from django.http import HttpResponse
import seaborn as sns
import io
import base64
from django.shortcuts import render, redirect
from django.contrib import messages
import csv
import random
import plotly.express as px
from io import BytesIO
```

Figure 2.8.2.11: Import libraries for Clustering crimes against women and crime forecasting prediction

Several important external libraries have been added to this Django-based website to improve various aspects of Clustering crimes against women and crime forecasting prediction project. These libraries include pandas and numpy for efficient data processing and manipulation, scikit-learn for machine learning tasks like feature scaling and regression, and matplotlib with seaborn for creating informative data visualizations. In addition, the io and base64 libraries handle file I/O operations, while Django's built-in modules such as django. shortcuts and. models handle web rendering and database interactions. Interactive data visualization is realized with plotly.express, and binary data processing is smoother with BytesIO. Finally, the csv and random libraries play a role in file management and random value generation. Together, these libraries enable an application to process data, build predictive models, visualize insights, and deliver a robust website.

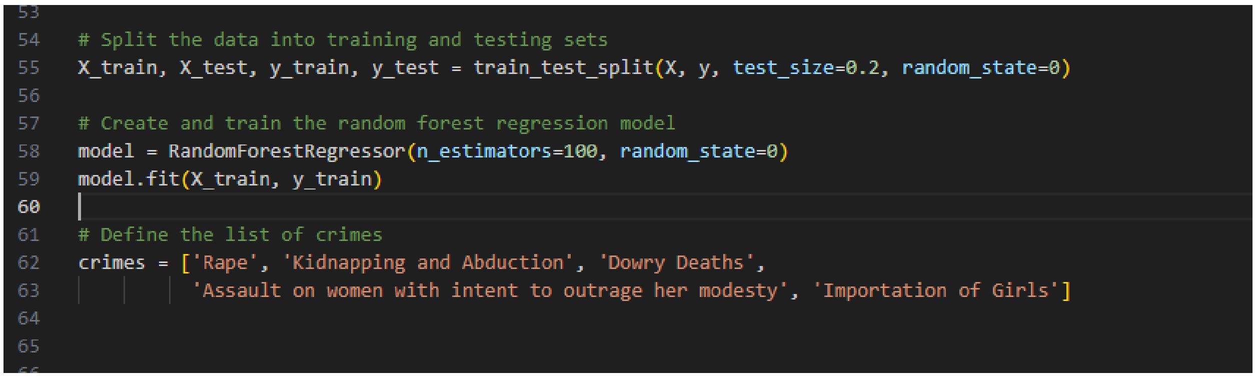


```python
# Read the CSV file into a pandas DataFrame
data = pd.read_csv("modified_file2.csv")

# Select the desired features
selected_features = ['STATE/UT', 'Year', 'Rape', 'Kidnapping and Abduction', 'Dowry Deaths',
                     'Assault on women with intent to outrage her modesty', 'Importation of Girls']
data = data[selected_features]

# Handling Missing Values
# Check for missing values
missing_values = data.isnull().sum()
data = data.dropna()  # Remove rows with missing values

# Select the desired features for clustering
selected_features = data[["STATE/UT", "Year", "Rape", "Kidnapping and Abduction", "Dowry Deaths",
                          "Assault on women with intent to outrage her modesty", "Importation of Girls"]]

# Encode categorical variables
label_encoder = LabelEncoder()
selected_features['STATE/UT'] = label_encoder.fit_transform(data['STATE/UT'])

# Normalize the numerical features
numerical_features = ["Rape", "Kidnapping and Abduction", "Dowry Deaths",
                      "Assault on women with intent to outrage her modesty", "Importation of Girls"]
scaler = StandardScaler()
selected_features[numerical_features] = scaler.fit_transform(selected_features[numerical_features])

# Separate the features and target variable
X = selected_features.drop(["Assault on women with intent to outrage her modesty"], axis=1)
y = selected_features["Assault on women with intent to outrage her modesty"]
```

Figure 2.8.2.12: Data Pre-processing for Clustering and Prediction

It begins by loading the CSV dataset into a panda DataFrame, focusing on specific columns relevant to the project's goals, such as crime statistics for different states and years. The next step involves handling missing values, which is essential to maintain data quality. It detects and counts the missing values in the dataset and then deletes the rows with missing values. After cleaning the data, the code encodes categorical variables using label encoding, a necessary transformation to convert non-numerical data into a format suitable for machine learning models.

To ensure consistency and comparability of numeric properties, the code standardizes them using a StandardScaler, which scales the values to have a mean of 0 and a standard deviation of 1. Finally, the code separates the dataset into characteristic variables (X) and target variables (y), where X contains all the independent variables and y represents the dependent variable, which is the number of such attacks. This pre-processing prepares the data for subsequent machine learning tasks such as model training and evaluation.

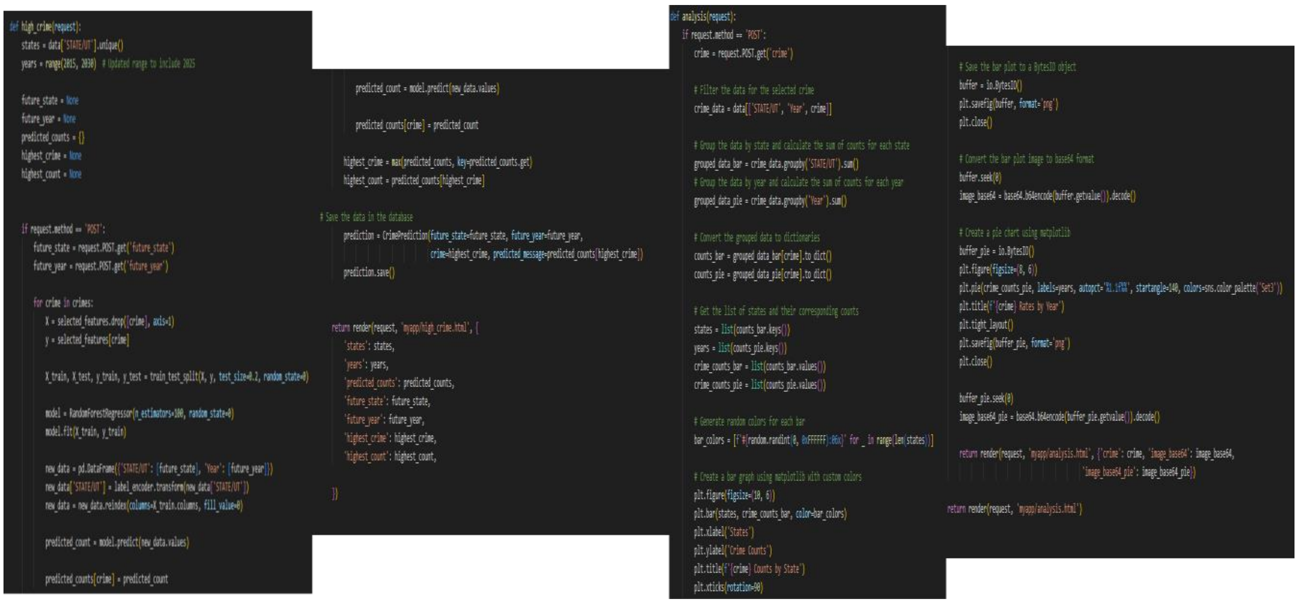


```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Create and train the random forest regression model
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(X_train, y_train)

# Define the list of crimes
crimes = ['Rape', 'Kidnapping and Abduction', 'Dowry Deaths',
          'Assault on women with intent to outrage her modesty', 'Importation of Girls']
```

Figure 2.8.2.13: Building a predictive model

This code segment is an important step in building a predictive model. It first divides the dataset into training and testing sets and reserves 80% of the data for training and 20% for testing. This separation is necessary to accurately assess model performance.

Next, it creates a random forest regression model, a powerful machine learning technique used for both classification and regression tasks. In this case, it is used for regression, specifically to predict "attack on women to outrage her modesty" based on other selected characteristics. The model configuration includes specifying 100 decision trees (n_estimators) in a Random Forest, which ensures robustness and reduces overfitting. The "random_state" parameter ensures repeatability.



```python
def high_crime(request):
    states = data['STATE/UT'].unique()
    years = range(2015, 2030)  # Updated range to include 2025

    future_state = None
    future_year = None
    predicted_counts = {}
    highest_crime = None
    highest_count = None


    if request.method == 'POST':
        future_state = request.POST.get('future_state')
        future_year = request.POST.get('future_year')

        for crime in crimes:
            X = selected_features.drop([crime], axis=1)
            y = selected_features[crime]

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

            model = RandomForestRegressor(n_estimators=100, random_state=0)
            model.fit(X_train, y_train)

            new_data = pd.DataFrame({'STATE/UT': [future_state], 'Year': [future_year]})
            new_data['STATE/UT'] = label_encoder.transform(new_data['STATE/UT'])
            new_data = new_data.reindex(columns=X_train.columns, fill_value=0)

            predicted_count = model.predict(new_data.values)

            predicted_counts[crime] = predicted_count
```

```python
            predicted_count = model.predict(new_data.values)

            predicted_counts[crime] = predicted_count

        highest_crime = max(predicted_counts, key=predicted_counts.get)
        highest_count = predicted_counts[highest_crime]

# Save the data in the database
        prediction = CrimePrediction(future_state=future_state, future_year=future_year,
                                     crime=highest_crime, predicted_message=predicted_counts[highest_crime])
        prediction.save()


    return render(request, 'myapp/high_crime.html', {
        'states': states,
        'years': years,
        'predicted_counts': predicted_counts,
        'future_state': future_state,
        'future_year': future_year,
        'highest_crime': highest_crime,
        'highest_count': highest_count,

    })
```

```python
def analysis(request):
    if request.method == 'POST':
        crime = request.POST.get('crime')

        # Filter the data for the selected crime
        crime_data = data[['STATE/UT', 'Year', crime]]

        # Group the data by state and calculate the sum of counts for each state
        grouped_data_bar = crime_data.groupby('STATE/UT').sum()
        # Group the data by year and calculate the sum of counts for each year
        grouped_data_pie = crime_data.groupby('Year').sum()

        # Convert the grouped data to dictionaries
        counts_bar = grouped_data_bar[crime].to_dict()
        counts_pie = grouped_data_pie[crime].to_dict()

        # Get the list of states and their corresponding counts
        states = list(counts_bar.keys())
        years = list(counts_pie.keys())
        crime_counts_bar = list(counts_bar.values())
        crime_counts_pie = list(counts_pie.values())

        # Generate random colors for each bar
        bar_colors = [f'#{random.randint(0, 0xFFFFFF):06x}' for _ in range(len(states))]

        # Create a bar graph using matplotlib with custom colors
        plt.figure(figsize=(10, 6))
        plt.bar(states, crime_counts_bar, color=bar_colors)
        plt.xlabel('States')
        plt.ylabel('Crime Counts')
        plt.title(f'{crime} Counts by State')
        plt.xticks(rotation=90)
```

```python
        # Save the bar plot to a BytesIO object
        buffer = io.BytesIO()
        plt.savefig(buffer, format='png')
        plt.close()

        # Convert the bar plot image to base64 format
        buffer.seek(0)
        image_base64 = base64.b64encode(buffer.getvalue()).decode()

        # Create a pie chart using matplotlib
        buffer_pie = io.BytesIO()
        plt.figure(figsize=(8, 6))
        plt.pie(crime_counts_pie, labels=years, autopct='%1.1f%%', startangle=140, colors=sns.color_palette('Set3'))
        plt.title(f'{crime} Rates by Year')
        plt.tight_layout()
        plt.savefig(buffer_pie, format='png')
        plt.close()

        buffer_pie.seek(0)
        image_base64_pie = base64.b64encode(buffer_pie.getvalue()).decode()

        return render(request, 'myapp/analysis.html', {'crime': crime, 'image_base64': image_base64,
                                                       'image_base64_pie': image_base64_pie})

    return render(request, 'myapp/analysis.html')
```

Figure 2.8.2.14: Crime Prediction and Analysis Views