PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Capstone Project Report Phase - I

on

Intellicleanse

Submitted by

Vaishnavi R Pachhapur - (PES1PG23CA159)

November 2024 - February 2025

Under the guidance of

Dr. Lekha A
Associate Professor
Department of Computer Applications,
PES University
Bengaluru – 560085

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER APPLICATIONS

PROGRAM - MASTER OF COMPUTER APPLICATIONS

Certificate

This is to certify that the project entitled

Intellicleanse

is a bonafide work carried out by

Vaishnavi R Pachhapur
(PES1PG23CA159)

in partial fulfilment for the completion of Capstone Project Phase - I work in the Program of Study MCA under the rules and regulations of PES University, Bengaluru during the period Nov. 2024 – Feb 2025. The project report has been approved as it satisfies the academic requirements 3th semester MCA.

Guide
Dr. Lekha A
Associate Professor
Dept. of Computer Applications
PES University
Bengaluru - 560085

Date :

Chairperson
Dr. Veena S
Professor
Dept. of Computer Applications
PES University
Bengaluru - 560085

Date :

Declaration

I, **Vaishnavi R Pachhapur**, bearing **PES1PG23CA159** hereby declare that the project entitled, **Intellicleanse**, is an original work done by me under the guidance of **Dr. Lekha A**, Associate Professor, PES University and is being submitted in partial fulfillment of the requirements for completion of 3rd Semester course work in the Program of Study MCA.

All corrections/suggestions indicated for internal assessment have been incorporated in the report.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other course.

Place: Bengaluru
Date : February 20, 2025

Vaishnavi R Pachhapur

PES1PG23CA159

Acknowledgment

I take great pleasure in expressing my sincere gratitude to all those who have guided me and supported me to successfully complete this project.

I express my sincere gratitude to the Vice Chancellor of PES University, **Dr. J Suryaprasad** and Chairperson **Dr. Veena S**, who gave me an opportunity to go ahead with this project.

I am grateful to my guide, **Dr. Lekha A**, Associate Professor, Department of Computer Applications, who has been my source of inspiration and provided me with guidance, encouragement and support, during the course of the project.

I sincerely thank the faculty of the Department of Computer Applications at PES University for their guidance and support. I am also grateful to my friends for their encouragement and insightful discussions. Lastly, I deeply appreciate my family for their constant support and motivation throughout this project.

Vaishnavi R Pachhapur

PES1PG23CA159

Abstract

Data accuracy plays a pivotal role in prudent analysis and decision-making. Raw datasets often contain inaccuracies, inconsistencies, and missing values, leading to misleading interpretations if not properly addressed. This project focuses on developing a web-based data cleaning tool to streamline and automate preprocessing tasks.

Ensuring the validity and consistency of raw data is essential for reliable analysis and decision-making. Unprocessed datasets frequently contain errors, inconsistencies, and missing values, which can lead to inaccurate conclusions. The primary objective of this project is to design an efficient and user-friendly tool that automates routine preprocessing tasks, thereby improving data quality for analysts, researchers, and businesses.

The program was developed using Python, utilizing libraries such as Pandas and Scikit-learn for data preprocessing. Key functionalities include duplicate removal, missing value imputation using statistical methods, outlier detection via interquartile range (IQR) and Z-score techniques, and comprehensive dataset evaluation through data profiling. The tool's accessibility and ease of use are ensured through a web-based interface, allowing users to optimize datasets efficiently before applying them to machine learning models or other analytical processes. By extracting meaningful insights from raw data through automated measures and visualizations, the solution streamlines data analysis and discovery.

The tool effectively automates data cleaning, reducing manual effort and improving accuracy. Performance metrics indicate high efficiency in handling large datasets, with precise detection and correction of inconsistencies. The automation process enhances data readiness for analysis by cutting preprocessing time by up to 50%.

In summary, this project contributes to data science by providing a rapid and efficient data cleaning solution. Future enhancements may include anomaly detection using machine learning and integration with cloud-based storage systems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Scenario

- Inconsistent, redundant, and incomplete datasets hinder effective data analysis, leading to inaccurate insights and delayed decision-making.
- Manual data cleaning processes are time-consuming, error-prone, and inefficient for handling large datasets.
- The lack of automated tools for data cleansing leads to suboptimal data quality, which negatively impacts decision-making in data-driven fields like business analytics, healthcare, and research.

1.2 Proposed Solution

- Intellicleanse is a web application that automates the data cleansing process using machine learning algorithms.
- The system will identify and resolve data issues such as duplicate entries, missing values, and outliers, ensuring high-quality and reliable datasets.
- The application will feature an intuitive user interface, allowing users to upload, profile, clean, and export datasets efficiently.
- Key technologies include Python for backend processing, TensorFlow/Scikit-learn for machine learning, and MySQL for database management.

1.3 Purpose

- The primary objective of Intellicleanse is to enhance data quality by addressing common data inconsistencies.

- The project aims to provide users with an efficient and user-friendly tool for preprocessing data before analysis.
- By improving data reliability, Intellicleanse will contribute to better decision-making in industries such as finance, healthcare, and scientific research.

1.4 Scope

- The application caters to data analysts, researchers, and enterprises requiring high-quality data for analytical and predictive tasks.
- Key functionalities include data upload and preview, duplicate detection and removal, missing value imputation, outlier detection with visualizations, data standardization, and collaborative cleaning.
- The system will support commonly used file formats such as CSV and XLSX and will offer export options with cleaning logs for transparency.
- Assumptions include users having basic knowledge of data preprocessing, and constraints involve potential computational limitations based on dataset size and complexity.

Chapter 2

Literature Survey

2.1 Domain Survey

2.1.1 Introduction to the Domain

- Data cleaning is a crucial preprocessing step in data science, ensuring data quality for accurate analysis and decision-making.
- The problem space includes handling missing values, duplicates, inconsistencies, and noise in datasets.
- Real-world applications include data analytics, machine learning, research databases, and business intelligence.

2.1.2 Core Terminologies and Definitions

- **Missing Data**: Data points that are absent from a dataset.
- **Data Duplication**: The presence of redundant entries in a dataset.
- **Outliers**: Data points significantly different from others in the dataset.
- **Data Imputation**: Techniques for filling in missing values.
- **Data Cleaning Tools**: Libraries such as Pandas (Python) with functions like *isnull()*, *fillna()*, and *dropna()*.

2.1.3 Associated Theoretical Concepts

- Machine learning-based approaches to detect and clean errors in datasets.
- Statistical models such as standard deviation, mean imputation, and clustering for data quality assessment.
- Use of fuzzy logic in handling uncertainty in data.

2.1.4 Challenges and Open Problems

- Scalability issues in cleaning large datasets.
- Handling unstructured data effectively.
- Ethical concerns regarding data modification.
- Lack of standardized techniques across different domains.

2.2 Literature Survey

2.2.1 Research Review

- **L. Ahuja, B. Singh, and R. Simonv (2024)**: *Data Cleaning: Paving a Way for Accurate and Clean Data*
  - Discusses techniques to handle missing values and duplicates.
  - Highlights automated tools for efficient cleaning.
- **Peng Li et al. (2021)**: *CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks*
  - Investigates the effect of data cleaning on ML performance.
  - Explores cleaning errors and their impact using different ML algorithms.
- **Nilu Singh (2023)**: *Data Cleaning Methods*
  - Covers techniques for handling missing values, duplicates, and outliers.
  - Discusses tools like *isnull()*, *fillna()*, and *dropna()*.
- **P. V. S. V. Narayana and V. M. G. S. Pandit (2018)**: *Data Cleaning Approaches in Data Mining*
  - Reviews techniques to handle noise, missing values, and inconsistencies.
  - Provides insights into algorithms ensuring data quality.
- **Huang Shan and E. Gubin (2019)**: *Data Cleaning for Data Analysis*
  - Highlights the importance of data cleaning in data analysis.
  - Proposes methods to handle missing values and inconsistencies.
- **Virender Kumar and Cherry Khosla (2018)**: *Data Cleaning: A Thorough Analysis and Survey on Unstructured Data*

  - Addresses challenges in cleaning unstructured data.
  - Surveys methodologies for handling text, multimedia, and semi-structured data.

- **Henning Koehler and Sebastian Link (2022)**: *Possibilistic Data Cleaning*
  - Explores a possibilistic approach using fuzzy logic for handling uncertain data.

- **Xi Wang and Chen Wang (2019)**: *Time Series Data Cleaning: A Survey*
  - Discusses techniques to clean time-series data.
  - Highlights challenges like temporal dependencies and seasonality.

- **Otmane Azeroual et al. (2018)**: *Data Quality Measures and Data Cleansing for Research Information Systems*
  - Explores data cleansing strategies for research databases.
  - Focuses on error detection and corrective actions.

- **Fakhitah Ridzuan and Wan Mohd Nazmee Wan Zainon (2019)**: *A Review on Data Cleansing Methods for Big Data*
  - Reviews data cleansing methods specific to big data environments.
  - Discusses challenges of traditional approaches and proposes advanced techniques.

2.2.2 Summary of Reviewed Research Papers

Paper	**Authors & Year**	**Problem Statement**	**Methodology**	**Key Findings**
Data Cleaning Techniques	L. Ahuja et al. (2024)	Handling missing values and duplicates	Automated tools for cleaning	Enhances decision-making and predictive accuracy
CleanML	Peng Li et al. (2021)	Impact of data cleaning on ML tasks	ML classification experiments	Cleaning improves ML performance

Data Cleaning Methods	Nilu Singh (2023)	Handling missing values and outliers	Data processing functions in Python	Improved data analysis accuracy
Possibilistic Cleaning	Henning Koehler et al. (2022)	Uncertainty in data	Fuzzy logic-based methods	Improved handling of ambiguous data
Big Data Cleansing	Fakhitah Ridzuan et al. (2019)	Challenges in cleaning big data	Advanced data cleansing methods	Enhanced decision-making in large datasets

2.2.3 Identified Research Gaps

- Need for scalable techniques for handling large datasets.
- Lack of standardization across different domains.
- Limited research on automated data cleaning for unstructured data.
- Ethical concerns regarding data modification.

2.2.4 Comparative Study of Existing Applications

Platform	**Features**	**AI Model Used**	**Strengths**	**Limitations**
OpenRefine	Data cleaning, transformation	Rule-based	Free and open-source	Requires technical knowledge
Trifacta Wrangler	Interactive data wrangling	Machine Learning	User-friendly interface	Expensive for large-scale use
DataBlist	Data cleaning, deduplication	Algorithmic Cleaning	Web-based, easy to use	Limited AI-based automation

CSVJSON Data Janitor	Convert, clean, and format data	None (Manual Operations)	Simple and fast processing	No AI automation, lacks advanced features

Chapter 3

Hardware and Software Requirements

3.1 Introduction

This chapter provides a detailed description of the **hardware and software** tools used in the development and deployment of this project. The selection of these technologies is based on **performance, scalability, and compatibility** with the project requirements.

3.2 Hardware Requirements

The project requires a system with sufficient computational power to support development, testing, and execution efficiently.

3.2.1 Minimum Hardware Specifications

The following table lists the **minimum hardware configuration** required:

Component	Specification
Processor	Intel Core i5 (9th Gen) / AMD Ryzen 5
RAM	8GB
Storage	256GB SSD
GPU (if required)	Integrated GPU
Network Interface	Wi-Fi 802.11ac / Ethernet
Peripherals	Keyboard, Mouse, Display

Table 3.1: Minimum Hardware Requirements

3.2.2 Recommended Hardware Specifications

For optimal performance, the **recommended configuration** is:

Component	Specification
Processor	Intel Core i7/i9 or AMD Ryzen 7/9
RAM	16GB or higher
Storage	512GB SSD or higher
GPU	NVIDIA RTX 3060 (for high-performance tasks)
Network Interface	Gigabit Ethernet / Wi-Fi 6

Table 3.2: Recommended Hardware Specifications

3.3 Software Requirements

The software stack used in this project includes **development tools, programming languages, and frameworks** that ensure smooth implementation and execution.

3.3.1 Operating System

The project was developed and tested on multiple platforms to ensure cross-platform compatibility.

Operating System	Version
Windows	Windows 10/11 (64-bit)
Linux (Preferred)	Ubuntu 20.04+ / Fedora 36+
macOS (Optional)	macOS 11+

Table 3.3: Operating System Requirements

3.3.2 Development Tools

To facilitate efficient coding and debugging, the following development tools were used:

Software	Purpose
Visual Studio Code	Code editor / IDE
Git & GitHub/GitLab	Version control & collaboration
XAMPP	Local server for MySQL database

Table 3.4: Development Tools

3.3.3 Programming Languages & Frameworks

The project was implemented using a combination of programming languages and frameworks:

Technology	**Usage**
HTML, CSS, JavaScript	Frontend development
Python (Flask)	Backend development
MySQL (XAMPP)	Database management

Table 3.5: Programming Languages and Frameworks

Chapter 4

Software Requirements Specification

4.1 Users

- **General User**

  Users can interact with the platform for basic data profiling and cleaning operations.

  – Upload datasets.

  – Preview and visualize data.

  – Apply basic cleaning operations.

4.2 Functional Requirements

- **Data Upload, Preview, and Profiling**

  Users can upload .csv or .xlsx files and view an instant preview with a summary report showing column names, data types, missing values, duplicates, and key statistics.

- **Redundancy and Consistency Cleaning**

  Detect and remove duplicate rows or columns, and handle missing values with configurable options like mean, median, custom values, or flagging for review.

- **Outlier Detection and Management with Visualizations**

  Identify statistical outliers and visualize them using histograms, scatter plots, and box plots to allow intuitive management (e.g., capping, flagging, or removal).

- **Data Standardization and Validation**

  Normalize data formats (e.g., dates, numerical values, text case) and allow users to define and apply validation rules to ensure schema consistency and compliance.

- **Automated Data Transformation Pipelines**

  Create reusable workflows for repetitive cleaning tasks like data formatting, redundancy removal, or missing value imputation.

- **Collaborative Cleaning with Version Control**

  Enable multiple users to work collaboratively on datasets with tracked changes, including the ability to save versions and revert to earlier stages.

- **Export Options with Cleaning Logs**

  Download cleaned data in formats like .csv or .xlsx along with a detailed log of the changes made during the cleaning process.

- **Enhanced Cleaning Insights**

  Extend profiling functionality with statistical insights, such as data distributions, correlations, and warnings for inconsistent patterns, to guide the cleaning process.

- **Dataset Balance Check**

  The system will analyze uploaded datasets to determine if they are balanced across categories. If an imbalance is detected, users will receive recommendations on handling unbalanced data through resampling techniques such as oversampling or undersampling.

4.3 Non-Functional Requirements

- **Performance**

  The system should handle 100 concurrent users with fast page load times under typical internet conditions.

- **Security**

  User data must be encrypted, and all communications should occur over HTTPS.

- **Scalability**

  The system must scale horizontally to handle increasing user demand.

- **Availability**

  The system should have minimal downtime with failover mechanisms for continuous service.

- **Usability**

  The application should be responsive and user-friendly across all devices.

Chapter 5

System Design

5.1 Architecture Diagram

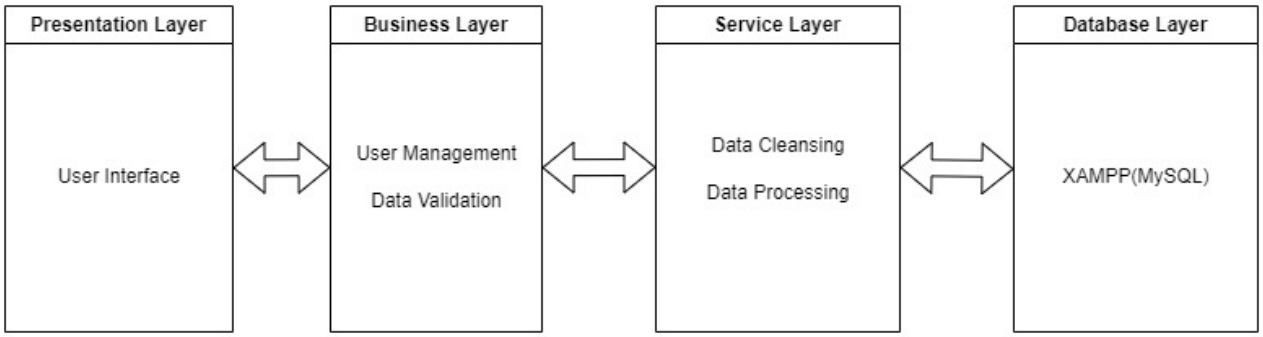

Figure 5.1: Architecture Diagram

The system architecture consists of four layers:

- **Presentation Layer:** This layer includes the user interface, where users interact with the system. It provides functionalities such as uploading datasets, viewing profiling reports, and performing cleaning operations.
- **Business Layer:** This layer contains core processing logic, such as user management, data validation, and handling user interactions with datasets.
- **Service Layer:** The service layer is responsible for executing data cleansing algorithms, processing data transformations, and ensuring consistency and accuracy in datasets.
- **Database Layer:** This layer stores and manages datasets using XAMPP with MySQL. It maintains user information, datasets, and cleaned data insights.

5.2 Data Flow Diagram

5.2.1 Context Diagram

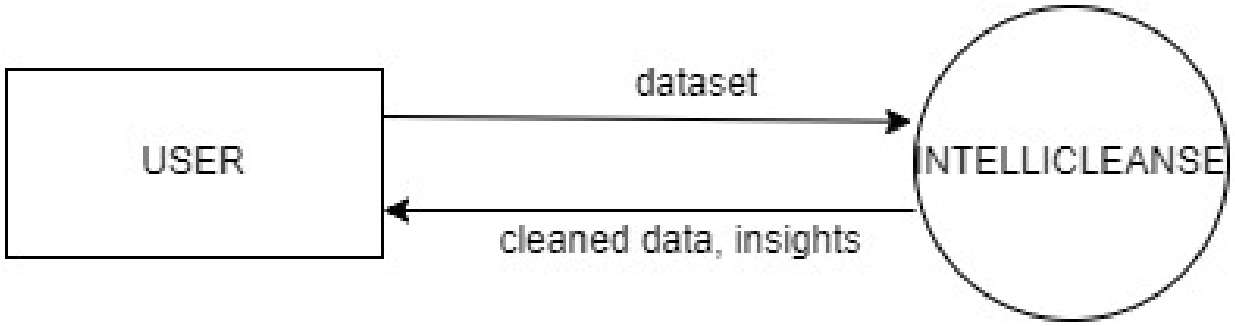

Figure 5.2: Context Diagram

The context diagram illustrates the interaction between the user and the system:

- **User:** Uploads a dataset to the system.
- **Intellicleanse:** Processes the dataset and returns cleaned data along with insights.

Chapter 6

Detailed Design

6.1 Use Case Diagram

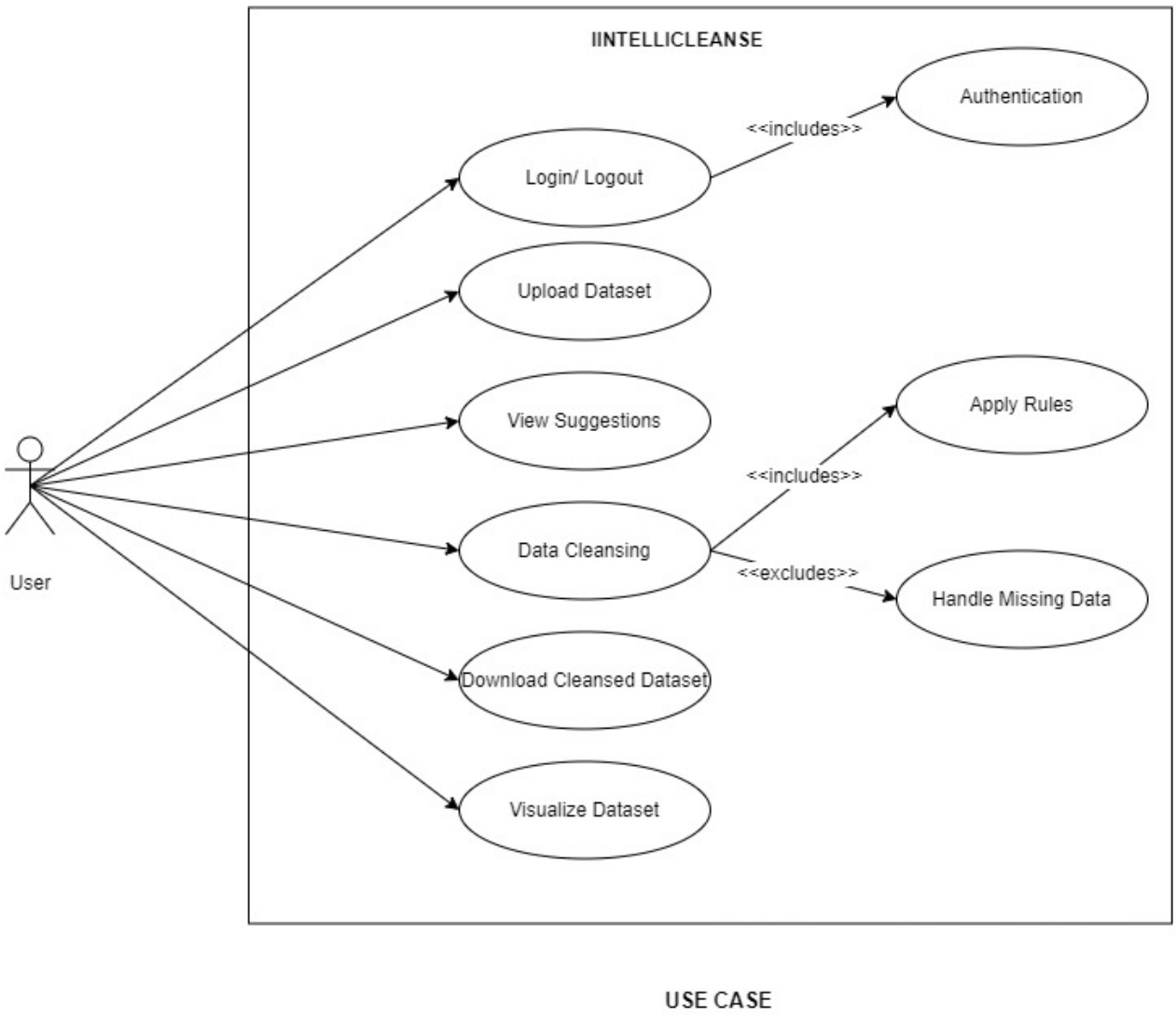

Figure 6.1: Use Case Diagram

The diagram depicts the user interacting with the system without showing

how it functions internally. The recipient will register or log in and request blood from a donor or blood bank. The donor will either accept or reject the recipient's request, and the blood bank will provide the blood and add it to its stock.

6.2 Process Flow Diagram

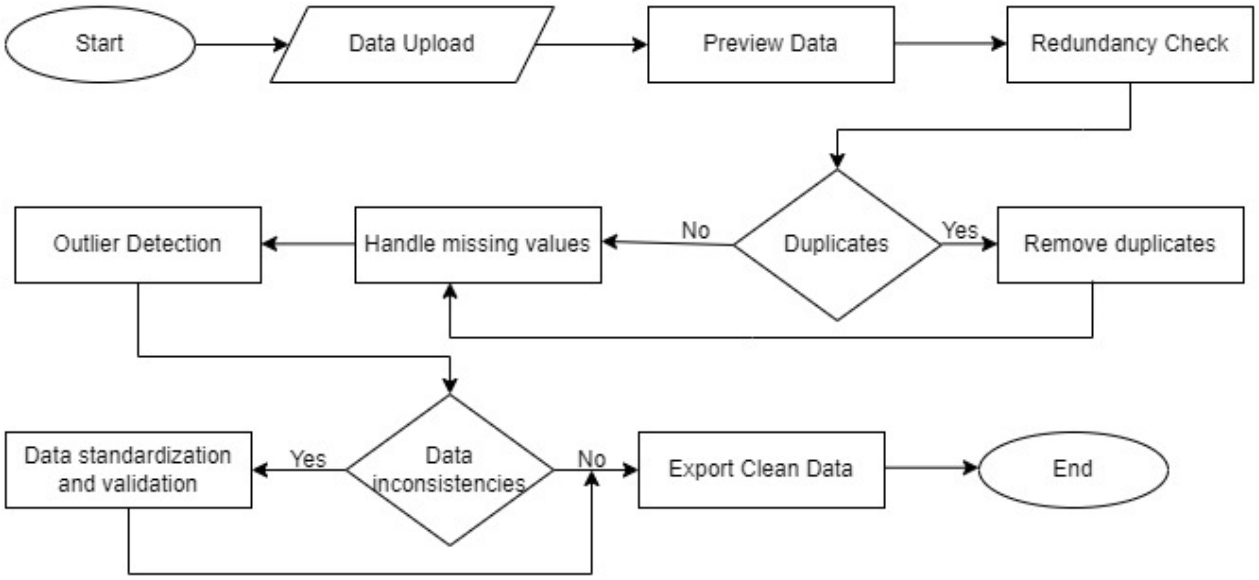

Figure 6.2: Process Flow Diagram

The process flow diagram represents the steps involved in the data cleansing workflow:

- **Start:** The process begins with data upload.
- **Preview Data:** Users can review the uploaded data.
- **Redundancy Check:** Identifies and removes redundant records.
- **Duplicate Handling:** Detects and removes duplicate values.
- **Missing Values Handling:** Resolves any missing data.
- **Outlier Detection:** Identifies and handles outliers.
- **Data Standardization and Validation:** Ensures data consistency.
- **Export Clean Data:** The final cleaned dataset is exported.
- **End:** The process concludes after data export.

Chapter 7

Methodology

7.1 Data Collection & Preprocessing

The project begins with gathering raw datasets from various sources, such as research datasets and business records. These datasets often contain inconsistencies, missing values, and duplicate records. To address this, data profiling is performed to analyze column names, data types, and missing values. Techniques like mean or median imputation and duplicate detection help in structuring the data before further processing. Ensuring uniform formats for numerical values, text, and date-time entries enhances data consistency.

7.2 Automated Data Cleaning & Transformation

Machine learning algorithms are utilized to automate the cleaning process. Outlier detection techniques such as Z-score and IQR help in identifying and handling anomalies. Standardization ensures uniform formats for numerical and categorical data. Users can define validation rules to maintain schema compliance. Additionally, data transformation pipelines allow the automation of repetitive cleaning tasks, making the process efficient and scalable.

7.3 Data Export & Visualization

Once the data is cleaned, it can be exported in formats such as CSV and XLSX, accompanied by a cleaning log that records all modifications. Visualization tech-

niques like histograms and scatter plots help users analyze data distributions and patterns. The platform also supports collaborative data cleaning, allowing multiple users to work together while tracking changes. This ensures the cleaned data is well-structured, documented, and ready for analysis.

Chapter 8

Implementation

8.1 Implementation Screenshots

8.1.1 Home Page

Below is a screenshot of the Home Page (index.html):

Figure 8.1: Home Page Screenshot

The screenshot showcases the main interface of the application, highlighting key functionalities and user navigation. It provides an overview of the layout and available features, ensuring an intuitive user experience.

8.1.2 Signup Page

Below is a screenshot of the Signup Page (signup.html):

This page allows new users to create an account by entering their details, ensuring secure authentication and access to the application features.

Figure 8.2: Signup Page Screenshot

8.1.3 Login Page

Below is a screenshot of the Login Page (login.html):

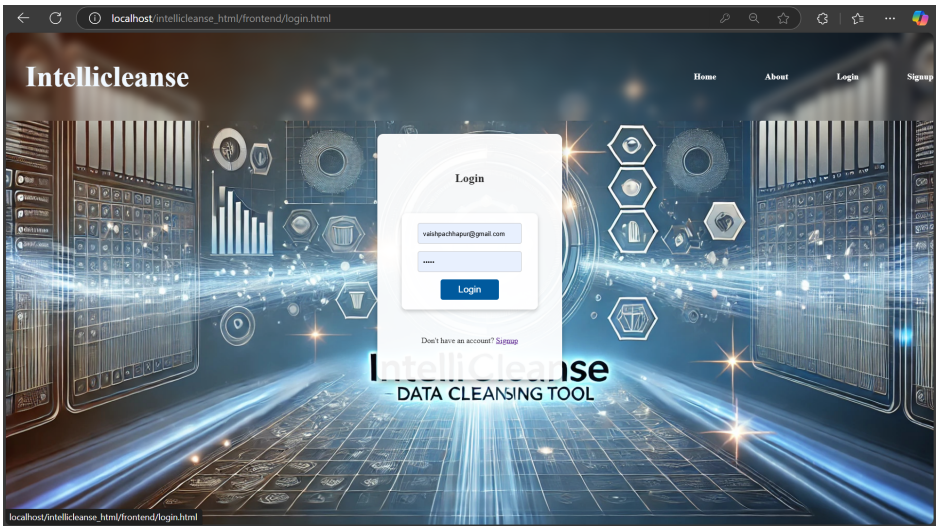

Figure 8.3: Login Page Screenshot

This page enables registered users to securely access their accounts using their credentials.

8.1.4 Data Upload, Preview, and Profile Page

Below is a screenshot of the Data Upload, Preview, and Profile Page:

This page allows users to upload datasets in .csv or .xlsx formats, preview data, and generate a profile report summarizing key attributes. It helps users quickly assess data quality, detect missing values, and understand distributions.

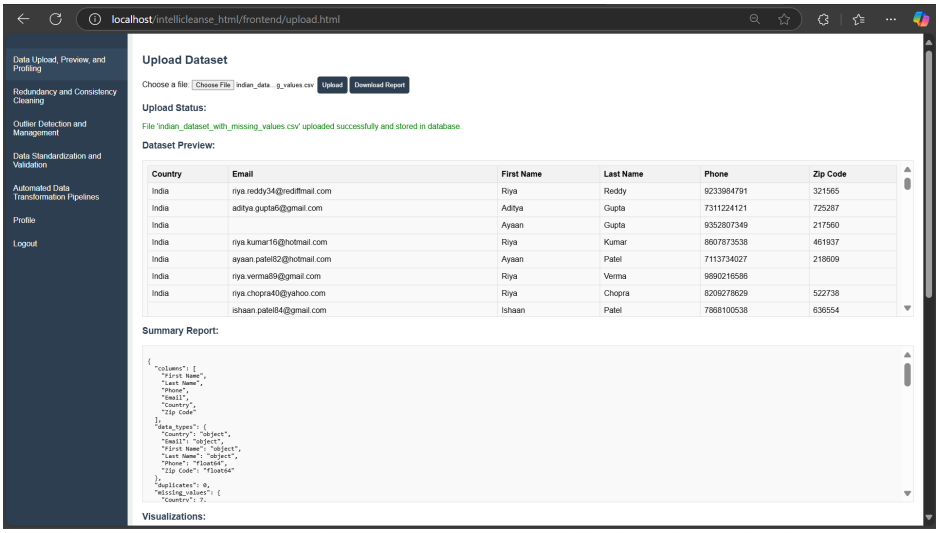

Figure 8.4: Data Upload, Preview, and Profile Page Screenshot

8.1.5 Redundancy and Consistency Cleaning Page

Below is a screenshot of the Redundancy and Consistency Cleaning Page:

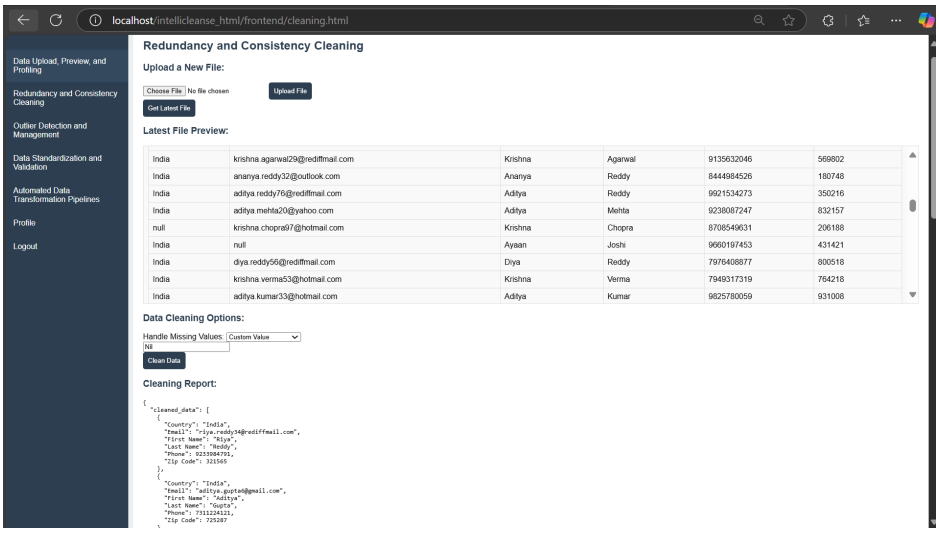

Figure 8.5: Redundancy and Consistency Cleaning Page Screenshot

This page provides tools to detect and remove duplicate records, ensuring data consistency and accuracy. It also helps streamline datasets by maintaining a single source of truth and preventing discrepancies.

8.1.6 Outlier Detection and Management Page

Below is a screenshot of the Outlier Detection and Management Page:

This page enables users to identify and manage statistical outliers using visual tools like histograms and scatter plots. It helps improve data quality by allowing users to analyze, filter, and handle anomalies effectively.

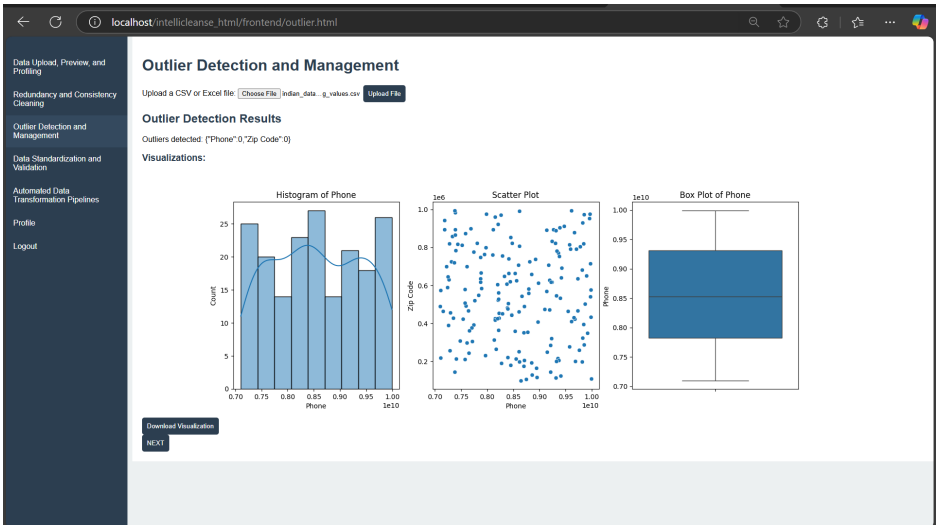

Figure 8.6: Outlier Detection and Management Page Screenshot

8.1.7 Data Standardization and Validation Page

Below is a screenshot of the Data Standardization and Validation Page:

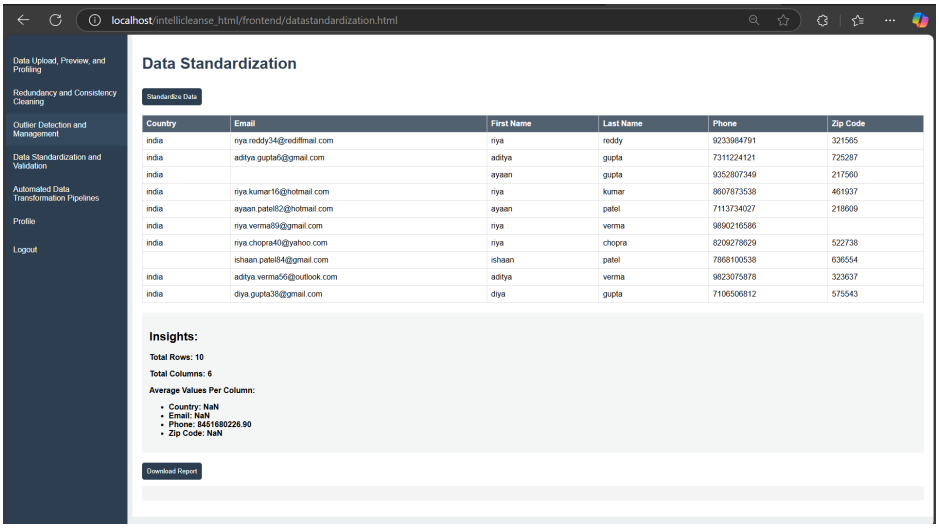

Figure 8.7: Data Standardization and Validation Page Screenshot

This page allows users to normalize data formats and apply validation rules, ensuring schema consistency and compliance. It helps maintain data accuracy and integrity by enforcing predefined standards.

8.2 Algorithm

The following algorithm implements various methods such as duplication removal, outlier detection, and normalization to ensure a clean and structured dataset.

Algorithm 1 Data Cleaning and Processing Algorithm

Require: Input dataset D
Ensure: Cleaned dataset D'

Initialize Flask application
Configure SQLAlchemy database connection
Define User and File models
Start Flask application
Receive file upload request
if file format is supported **then**
    Read file into DataFrame
    Identify and remove duplicate rows
    Handle missing values (mean, median, or custom values)
    Detect and handle outliers using Z-score method
    Normalize and validate data fields
    Store cleaned dataset in the database
    Return cleaned dataset as response
else
    Return error message
end if
Process user authentication requests
if signup request **then**
    Validate user data
    Store user credentials securely
    Return signup confirmation
end if
if login request **then**
    Authenticate user
    Create session
    Return login success message
end if
Process data retrieval requests
Return processed data to the user

8.3 Code Logic

The following code demonstrates the core logic of the implementation:

```
from flask import Flask, request, jsonify, session
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
from scipy.stats import zscore
from werkzeug.security import generate_password_hash, check_password_hash
import pandas as pd
import numpy as np
import pymysql
import io
import traceback
import matplotlib.pyplot as plt
import seaborn as sns
import base64
import re
from datetime import datetime
from io import BytesIO
from datetime import timedelta

pymysql.install_as_MySQLdb()

app = Flask(__name__)

CORS(app, resources={r"/*": {"origins": "*"}}, supports_credentials=True)

app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:@localhost/
    intellicleanse'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'your_secret_key'
app.permanent_session_lifetime = timedelta(days=1)

app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(days=1)

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    username = db.Column(db.String(150), nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(256), nullable=False)
```

```
    files = db.relationship('CollaborativeFile', backref='user', lazy=True)

class File(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(255), nullable=False)
    data = db.Column(db.LargeBinary, nullable=False)
    collaborations = db.relationship('CollaborativeFile', backref='file', lazy=
   True)

class CollaborativeFile(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    file_id = db.Column(db.Integer, db.ForeignKey('file.id'), nullable=False)
    role = db.Column(db.String(50), nullable=False)
    last_edit = db.Column(db.DateTime, default=datetime.utcnow)


@app.route('/upload', methods=['POST'])
def upload_file():
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No file uploaded"}), 400

        file = request.files['file']
        filename = file.filename
        file_data = file.read()

        if not filename:
            return jsonify({"error": "Invalid file"}), 400

        new_file = File(name=filename, data=file_data)
        db.session.add(new_file)
        db.session.commit()

        file.seek(0)
        df = None
        if filename.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(file_data))
        elif filename.endswith('.xlsx'):
            df = pd.read_excel(io.BytesIO(file_data))
        else:
            return jsonify({"error": "Unsupported file format"}), 400
```

```
80
81          profiling_info = {
82              "columns": list(df.columns),
83              "data_types": df.dtypes.astype(str).to_dict(),
84              "missing_values": df.isnull().sum().to_dict(),
85              "duplicates": int(df.duplicated().sum()),
86              "summary": df.describe(include='all').replace({pd.NA: None, float("
    nan"): None}).to_dict()
87          }
88
89          preview_data = df.replace({pd.NA: None, float("nan"): None}).to_dict(
    orient='records')
90
91
92          return jsonify({
93              "message": f"File '{filename}' uploaded successfully and stored in
    database.",
94              "preview": preview_data,
95              "profiling": profiling_info
96          }), 201
97
98      except Exception as e:
99          print("Error in /upload:", traceback.format_exc())
100         return jsonify({"error": str(e)}), 500
101
102 @app.route('/signup', methods=['POST'])
103 def signup():
104     try:
105         data = request.get_json()
106         print("Received signup data:", data)
107
108         if not data:
109             return jsonify({"error": "Invalid JSON data"}), 400
110
111         username = data.get('username')
112         email = data.get('email')
113         password = data.get('password')
114
115         if not username or not email or not password:
116             return jsonify({"error": "Missing fields"}), 400
117
118         user_exists = User.query.filter_by(email=email).first()
```

```
        if user_exists:
            return jsonify({"message": "Email already registered"}), 400

        hashed_password = generate_password_hash(password)
        new_user = User(username=username, email=email, password=
    hashed_password)
        db.session.add(new_user)
        db.session.commit()

        print("User created successfully:", email)
        return jsonify({"message": "Signup successful"}), 201
    except Exception as e:
        print("Error in /signup:", traceback.format_exc())
        return jsonify({"error": str(e)}), 500

@app.route('/login', methods=['POST'])
def login():
    try:
        data = request.get_json()
        if not data:
            return jsonify({"error": "Invalid JSON data"}), 400

        email = data.get('email')
        password = data.get('password')

        if not email or not password:
            return jsonify({"error": "Missing email or password"}), 400

        user = User.query.filter_by(email=email).first()
        if not user or not check_password_hash(user.password, password):
            return jsonify({"message": "Invalid credentials"}), 401

        session['user_id'] = user.id
        return jsonify({"message": "Login successful"}), 200
    except Exception as e:
        print("Error in /login:", traceback.format_exc())          return
    jsonify({"error": str(e)}), 500

@app.route('/get_latest_file', methods=['GET'])
def get_latest_file():
    try:
        latest_file = File.query.order_by(File.id.desc()).first()
```

```
        if not latest_file:
            return jsonify({"error": "No files found in the database"}), 404

        filename = latest_file.name
        file_data = latest_file.data

        df = None
        if filename.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(file_data))
        elif filename.endswith('.xlsx'):
            df = pd.read_excel(io.BytesIO(file_data))
        else:
            return jsonify({"error": "Unsupported file format"}), 400

        preview_data = df.replace({pd.NA: None, float("nan"): None}).to_dict(orient='records')

        return jsonify({
            "message": f"Retrieved latest file '{filename}' successfully.",
            "preview": preview_data
        }), 200
    except Exception as e:
        print("Error in /get_latest_file:", traceback.format_exc())
        return jsonify({"error": str(e)}), 500



@app.route('/clean_data', methods=['POST'])
def clean_data():
    try:
        latest_file = File.query.order_by(File.id.desc()).first()
        if not latest_file:
            return jsonify({"error": "No dataset found. Please upload a file first."}), 404

        filename = latest_file.name
        file_data = latest_file.data

        if filename.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(file_data))
        elif filename.endswith('.xlsx'):
            df = pd.read_excel(io.BytesIO(file_data))
```

```
        else:
            return jsonify({"error": "Unsupported file format"}), 400
\textit{\textit{}}
        duplicates_before = df.duplicated().sum()
        df = df.drop_duplicates()
        df = df.loc[:, ~df.T.duplicated()]
        duplicates_after = df.duplicated().sum()



        missing_values_before = df.isnull().sum().to_dict()
        handle_missing = request.form.get('missing_values', 'mean')

        numeric_cols = df.select_dtypes(include=['number']).columns

        if handle_missing == 'mean':
            df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
        elif handle_missing == 'median':
            df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median
    ())
        elif handle_missing == 'custom':
            custom_value = request.form.get('custom_value', '')
            df.fillna(custom_value, inplace=True)
        elif handle_missing == 'flag':
            df.fillna("MISSING", inplace=True)

        missing_values_after = df.isnull().sum().to_dict()

        cleaning_report = {
            "duplicates_removed": int(duplicates_before - duplicates_after),
            "missing_values_before": {k: int(v) for k, v in
    missing_values_before.items()},
            "missing_values_after": {k: int(v) for k, v in missing_values_after
    .items()}
        }

        cleaned_data = df.replace({np.nan: None}).astype(object).to_dict(orient
    ='records')

        return jsonify({
            "message": "Data cleaned successfully",
            "cleaning_report": cleaning_report,
            "cleaned_data": cleaned_data[:10]
```

```
        })
    except Exception as e:
        print("Error in /clean_data:", traceback.format_exc())
        return jsonify({"error": str(e)}), 500


@app.route('/outlier_detection', methods=['POST'])
def outlier_detection():
    try:
        latest_file = File.query.order_by(File.id.desc()).first()
        if not latest_file:
            return jsonify({"error": "No dataset found. Please upload a file first."}), 404

        filename = latest_file.name
        file_data = latest_file.data

        if filename.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(file_data))
        elif filename.endswith('.xlsx'):
            df = pd.read_excel(io.BytesIO(file_data))
        else:
            return jsonify({"error": "Unsupported file format"}), 400

        numeric_cols = df.select_dtypes(include=['number']).columns
        df_zscore = df[numeric_cols].apply(zscore)
        outliers = (df_zscore.abs() > 3).sum(axis=0)

        fig, axes = plt.subplots(1, 3, figsize=(15, 5))

        sns.histplot(df[numeric_cols[0]], kde=True, ax=axes[0])
        axes[0].set_title('Histogram of ' + numeric_cols[0])

        sns.scatterplot(data=df, x=numeric_cols[0], y=numeric_cols[1] if len(numeric_cols) > 1 else numeric_cols[0], ax=axes[1])
        axes[1].set_title('Scatter Plot')

        sns.boxplot(data=df[numeric_cols[0]], ax=axes[2])
        axes[2].set_title('Box Plot of ' + numeric_cols[0])

        img = BytesIO()
        plt.savefig(img, format='png')
```

```
        img.seek(0)
        img_b64 = base64.b64encode(img.getvalue()).decode('utf-8')
        plt.close(fig)

        return jsonify({
            "message": "Outlier detection and visualizations generated successfully.",
            "outliers": outliers.to_dict(),
            "plot": img_b64
        })

    except Exception as e:
        print("Error in /outlier_detection:", traceback.format_exc())
        return jsonify({"error": str(e)}), 500


@app.route('/data_standardization', methods=['POST'])
def standardize_data():
    try:
        latest_file = File.query.order_by(File.id.desc()).first()
        if not latest_file:
            return jsonify({"error": "No dataset found. Please upload a file first."}), 404

        filename = latest_file.name
        file_data = latest_file.data

        if filename.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(file_data))
        elif filename.endswith('.xlsx'):
            df = pd.read_excel(io.BytesIO(file_data))
        else:
            return jsonify({"error": "Unsupported file format"}), 400

        def standardize_text(col):
            return col.str.lower().str.strip()

        def standardize_date(col, date_format="%Y-%m-%d"):
            return pd.to_datetime(col, errors='coerce').dt.strftime(date_format)

        def standardize_numeric(col):
```

```
            return pd.to_numeric(col, errors='coerce')

        for col in df.columns:
            if df[col].dtype == 'object':  # Text columns
                df[col] = standardize_text(df[col])
            elif df[col].dtype == 'datetime64[ns]':
                df[col] = standardize_date(df[col])
            elif df[col].dtype in ['int64', 'float64']:
                df[col] = standardize_numeric(df[col])

        cleaned_data = df.replace({np.nan: None}).astype(object).to_dict(orient
    ='records')

        return jsonify({
            "message": "Data standardized successfully.",
            "standardized_data": cleaned_data[:10]         })
    except Exception as e:
        print("Error in /data_standardization:", traceback.format_exc())
        return jsonify({"error": str(e)}), 500


@app.route('/datastandardization', methods=['GET'])
def data_standardization_page():
    return render_template('datastandardization.html')


@app.route('/profile', methods=['GET'])
def profile():
    if 'user_id' not in session:
        return jsonify({"error": "User not logged in"}), 401

    user = User.query.get(session['user_id'])
    if not user:
        return jsonify({"error": "User not found"}), 404

    return jsonify({
        "username": user.username,
        "email": user.email
    })


@app.route('/edit_profile', methods=['GET', 'POST'])
```

```
def edit_profile():
    try:
        if 'user_id' not in session:
            return redirect(url_for('login'))

        user = User.query.get(session['user_id'])
        if not user:
            return jsonify({"error": "User not found"}), 404

        if request.method == 'POST':
            username = request.form.get('username')
            email = request.form.get('email')

            if username:
                user.username = username
            if email:
                user.email = email

            db.session.commit()

            return redirect(url_for('profile'))

        return render_template('edit_profile.html', user=user)
    except Exception as e:
        print("Error in /edit_profile:", traceback.format_exc())
        return jsonify({"error": str(e)}), 500

@app.route('/join_file', methods=['POST'])
def join_file():
    try:
        data = request.get_json()
        file_id = data.get('file_id')
        role = data.get('role')

        if not file_id or not role:
            return jsonify({"error": "File ID and role are required"}), 400

        if role not in ['editor', 'viewer']:
            return jsonify({"error": "Invalid role"}), 400

        if 'user_id' not in session:
            return jsonify({"error": "User not logged in"}), 401
```

```
399
400         user_id = session['user_id']
401
402         file = File.query.get(file_id)
403         if not file:
404             return jsonify({"error": "File not found"}), 404
405
406         existing_collab = CollaborativeFile.query.filter_by(user_id=user_id,
    file_id=file_id).first()
407         if existing_collab:
408             return jsonify({"error": "User already part of this file"}), 400
409
410         new_collab = CollaborativeFile(user_id=user_id, file_id=file_id, role=
    role)
411         db.session.add(new_collab)
412         db.session.commit()
413
414         return jsonify({"message": f"User joined the file with role '{role}'"})
    , 200
415
416     except Exception as e:
417         print("Error in /join_file:", traceback.format_exc())
418         return jsonify({"error": str(e)}), 500
419
420 @app.route('/get_collaborators/<int:file_id>', methods=['GET'])
421 def get_collaborators(file_id):
422     try:
423         collaborators = CollaborativeFile.query.filter_by(file_id=file_id).all
    ()
424         if not collaborators:
425             return jsonify({"error": "No collaborators found for this file"}),
    404
426
427         collaborator_data = []
428         for collab in collaborators:
429             user = User.query.get(collab.user_id)
430             collaborator_data.append({
431                 "username": user.username,
432                 "role": collab.role,
433                 "last_edit": collab.last_edit
434             })
435
```

```
436         return jsonify({"collaborators": collaborator_data}), 200
437
438     except Exception as e:
439         print("Error in /get_collaborators:", traceback.format_exc())
440         return jsonify({"error": str(e)}), 500
441
442
443 @app.route('/logout', methods=['GET'])
444 def logout():
445     try:
446         session.pop('user_id', None)
447         return redirect(url_for('login'))
448     except Exception as e:
449         print("Error in /logout:", traceback.format_exc())
450         return jsonify({"error": str(e)}), 500
451
452 if __name__ == '__main__':
453     with app.app_context():
454         db.create_all()
455     app.run(debug=True)
```

Listing 8.1: Flask Application Code

Chapter 9

Application in the Real World

The advancements in **Intellicleanse** have led to significant real-world applications across various domains. This section highlights key areas where the system can be effectively utilized.

9.1 Industry and Business Applications

Intellicleanse can be integrated into industries to enhance efficiency and automation:

- **Data-Driven Decision Making** – Ensures accurate data for business intelligence.
- **Automated Data Cleaning** – Reduces manual efforts and streamlines workflows.
- **Fraud Detection** – Identifies anomalies in financial transactions.

9.2 Healthcare and Medical Applications

In the healthcare sector, Intellicleanse plays a crucial role in:

- **Medical Data Standardization** – Improves patient record accuracy.
- **Predictive Analytics** – Enhances early disease detection and treatment planning.

- **Health Monitoring** – Assists in real-time patient data processing.

9.3 Education and Research

Academic institutions and research organizations benefit from:

- **Data Cleaning for Research** – Ensures integrity in scientific studies.
- **Automated Data Processing** – Reduces errors in educational assessments.
- **AI-Based Learning** – Enables personalized learning through clean data.

9.4 Security and Surveillance

The security sector utilizes Intellicleanse for:

- **Threat Detection** – Identifies suspicious patterns in security logs.
- **Cybersecurity** – Cleans and analyzes network traffic for anomalies.
- **Automated Monitoring** – Enhances video surveillance systems.

9.5 Future Scope and Expansion

The applications of Intellicleanse continue to evolve, with future developments including:

- **AI and Machine Learning Integration** – Enables smarter data processing.
- **IoT and Smart Systems** – Expands usage in automated environments.
- **Enhanced Automation** – Further reduces human intervention in data management.

9.6 Conclusion

This section demonstrates the practical applications of **Intellicleanse** across industries, healthcare, education, and security. With continuous advancements, it will play a crucial role in driving data-driven innovation and efficiency.

Appendix A: References

References

[1] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. Cambridge, MA, USA: MIT Press, 2001.

[2] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann, 2016.

[3] M. J. Pazzani, *Data Mining: Machine Learning Perspectives*. Palo Alto, CA, USA: AAAI Press, 2000.

[4] W. H. Inmon, *Building the Data Warehouse*, 4th ed. Hoboken, NJ, USA: Wiley, 2005.

[5] P. D. Turney, "Types of Data Preprocessing," 1999.

[6] D. J. Salgado, A. L. A. Coelho, and M. R. Silva, *Data Cleansing: Problems and Techniques in Data Science*. Cham, Switzerland: Springer, 2020.

[7] P. V. S. V. Narayana and V. M. G. S. Pandit, *Data Cleaning Approaches in Data Mining*. Cham, Switzerland: Springer, 2018.

[8] H. V. Jagadish, A. A. Imran, and M. A. U. Zaman, *Data Quality and Cleaning in Data Science*. Cham, Switzerland: Springer, 2020.

[9] A. M. MacQueen, "Efficient Methods for Data Preprocessing and Cleaning," 2015.

[10] R. Agerri and A. R. M. A. Yousaf, "Data Cleaning Techniques for Data Mining: A Review," *International Journal of Computer Applications*, 2015.

[11] M. M. M. A. Mahmud, "Data Quality and Cleaning Challenges in Machine Learning Models," 2017.

[12] S. C. G. J. H. P. J. Geurts, "Data Quality and the Impact of Data Cleansing on Predictive Performance," *Journal of Machine Learning Research*, 2019.

[13] B. L. Masnadi-Shirazi, *Outlier Detection and Data Preprocessing in Data Science*. Cham, Switzerland: Springer, 2014.

[14] J. L. Ambroise and R. N. Dubes, "Data Cleaning and Normalization: A Comprehensive Survey," *Journal of Data Science*, 2002.