Modern University for Technology and Information

Faculty of Computers and Information

Department of Information Systems

Design and Implementation of Data Cleansing System

Founders:

Mohammed Ahmed Rashad

Mohammed Osama Abdelghany

Mohamed Adly Mohammed

Toka Saied Nafie

Mohammed Abdelkarim Mohammed

Eyad Ahmed Ali

Mohammed Abdelmotaleb Mohammed

Abdelrahman Ghoniem Ghoniem

Supervised By:

Dr. Assem El-Ashaar

Dr. Omar Khalid

Acknowledgement

In the Name of **Allah**, the Most Merciful, the Most Compassionate,

Alhamdulillah all praises belong to Almighty Allah, peace be upon **Muhammad** His servant and messenger.

Our greatest gratitude to Modern University for Technology and Information **(M.T.I.)** for the role it played in forming our personalities, providing us with the means necessary for our educational development.

We would like to seize this opportunity to express **thanks to Dr. Assem Elashaar**, **Dr. Omar Khalid** for their efforts and support during the course of this project.

Furthermore, we would like to show our gratitude to all our Professors and University Staff whose efforts guided us through our academic endeavors. Special thanks to our Families for their continuous support without which we would have never been able to make it.

This would not have been possible unless **Prof.Dr. Mohamed Elgazzar**, President of the MTI University has established such a learning environment, providing the most effective facilities for our job to be done.

It is a pleasure to thank those who maneuvered this Possible, **Prof. Dr. Mohamed Taher El-Mayah**, and Dean of the faculty of Computers and Information for his unlimited help, support, encouragement, and guidance. Special thanks to **Prof.Dr. Hanafy Ismail**, head of Computer Science Department , We are Honored and got the pleasure of working with **Prof. Dr. Hesham El Deeb,** Head of AI department, As well as the Department staff, **Prof.Dr Alaa Abd El-Raheem** for their effort during our university studies. Also, Special thanks to **Prof. Dr. Hafez Abdel-Wahab**, head of Information System Department, as well as, for their effort along our university studies and **Prof. Dr. Mahamoud El-Shishtawy** for their effort along our university studies.

Special thanks to **Prof. Dr. Eman Taha**, head of Basic Science Department. As well as, the department staff, and for their effort teaching us the basic for our computer science degree, **Prof. Dr. Sayed Bakar**, **Prof.Dr. Rasha Saeed** and **Dr. Rania Ahmed** Special thanks to our families who always supported us during our academic life and we couldn’t do it without their help.

Abstract

In the current landscape, one of the paramount challenges across various sectors is maintaining data integrity and quality. Data cleansing systems emerge as a vital solution, especially in domains like finance, retail, and healthcare, where accurate data is crucial for decision-making and compliance. Our project aims to design a robust data cleansing system. This system operates through a user-friendly graphical user interface, allowing seamless interaction with **datasets & databases** of varying complexities. The development process began with extensive research into existing data cleansing methodologies and tools, identifying key requirements and approaches. Subsequently, the design phase delineated the system's functionalities via comprehensive reports, outlining the expected outcomes. Implementation involved the utilization of cutting-edge frameworks and programming languages tailored to the specific needs of data cleansing. Rigorous testing ensued, ensuring the system's efficacy in identifying and rectifying anomalies within datasets & databases. Ultimately, the decision to develop the data cleansing system as a versatile application ensures accessibility and usability across diverse organizational contexts, empowering users to maintain data integrity effortlessly.

Table of Contents

Table of figures

Chapter 1

Introduction

Chapter-1-Introduction

1.1Introduction

A data cleansing website system serves as a fundamental tool in ensuring the integrity and reliability of organizational data. It embodies a sophisticated application designed to interact with users in a seamless and intuitive manner, akin to human conversation. At its core, this system leverages advanced technologies to identify and rectify inconsistencies, errors, and redundancies within datasets, thereby enhancing data quality and accuracy. By harnessing the power of machine learning algorithms and data processing techniques, it offers comprehensive solutions for data validation, standardization, and enrichment. From financial institutions to e-commerce platforms, data cleansing systems play a pivotal role in optimizing decision-making processes and ensuring regulatory compliance. The development journey commences with thorough research into data cleansing methodologies and user requirements, followed by meticulous design and implementation phases tailored to the unique needs of the target audience. Rigorous testing ensures the robustness and efficacy of the system, ultimately culminating in a reliable and efficient data cleansing solution. With the potential to revolutionize data management practices across industries, a data cleansing website system represents a significant advancement towards achieving data-driven excellence and operational efficiency.

1.2 Problem Definition

The primary challenge at hand revolves around the need for efficient data management solutions that mitigate the risks associated with erroneous or inconsistent datasets & databases. Organizations grapple with the arduous task of sifting through vast amounts of data plagued by inaccuracies, duplicates, and inconsistencies, leading to inefficiencies in decision-making processes and regulatory non-compliance. Moreover, the conventional methods of data cleansing often entail significant time and resource investments, hindering operational agility and scalability. Additionally, the proliferation of data across disparate systems exacerbates the challenge of maintaining data integrity and consistency. Consequently, there arises a pressing need for a comprehensive data cleansing website system that offers streamlined processes for identifying, correcting, and standardizing data anomalies. Such a solution must cater to diverse organizational needs while ensuring user-friendly interfaces and robust security measures to safeguard sensitive information. By addressing these challenges, organizations can enhance data quality, optimize operational efficiency, and foster informed decision-making, thereby unlocking new opportunities for growth and innovation.

Given that

Current data is plagued by inconsistencies, inaccuracies, and duplicates. This leads to unreliable analysis, poor decision-making, and inefficient operations.

It is required to have a system to

Data analysis: Identify inconsistencies, inaccuracies, and duplicates.

Correction: Achieve high accuracy in fixing errors automatically.

User-friendliness: Monitor, manage, and generate reports with ease.

Security & Compliance: Ensure data privacy and regulatory adherence.

Such that

Data-driven decisions are based on reliable and accurate information.

Analysis becomes more efficient and insightful due to cleaner data.

Operational costs are reduced by eliminating errors and redundancies.

Compliance and security risks are minimized through data governance.

1.3 Objective

The main objectives are:-

• In General:-

-the main goal is to make users able to clean their datasets & databases for mainly 4 problems

-decide whether they want to clean their datasets & database by their own selves or hand us theirs to clean it for them

-save time and money by cleaning their own datasets & databases instead of paying highly amounts of money to offices specialized in data cleansing services

• In the Application:-

-develop a robust query subsystem to handle our 4 database problems

1-invalid formats

2-dublicates

3-outliers

4- Missing values

-develop dashboard that tracks changes that happen after and before performing a query subsystem action

1.4 Project Outline

Chapter 2: System Planning involves outlining project team tasks and presenting planning diagrams like Gantt charts and network diagrams.

Chapter 3: Literature Review provides an overview of related systems, including examples with screenshots to illustrate their advantages, disadvantages, and a comparison of pros and cons.

Chapter 4: System Analysis details user interactions, requirements, and uses diagrams like use case diagrams, application interaction diagrams (class, sequence, and activity diagrams) to depict functionalities.

Chapter 5: System Design presents the intended system design, starting with interface prototypes and followed by system flowcharts illustrating functionalities.

Chapter 6: System Implementation discusses Android development tools with detailed screenshots of usage, showcasing application screenshots and interface descriptions.

Chapter 7: System Testing covers the testing process during and after system implementation, including user feedback evaluation through messages.

Chapter 8: Conclusion and Future Work summarizes the implemented application and outlines potential future enhancements.

Chapter 2
System Planning

Chapter-2-system planning

2.1 introduction

The purpose of the Planning stage is to analyze the project in terms of work breakdown, cost, resources, and timing. At the end of this stage all team members should be clear on the sub tasks and deliverables with the project, the time constraints they are working on too and the roles and responsibilities that are expected from them.
Software project plan defines what the work is, and how this work can be completed.

In This Chapter, the project planning is presented using the project 356 software to manage the project activities and usually with defined stages, and with designated resources the project planning contains three parts: Project task, Gantt chart and Network diagram

- Creating a comprehensive Project Plan is a critical step in the Project Lifecycle, as it is used to :
  - Monitor and control the overall progress of the project.
  - Determine whether the project activities are complete.
  - Determine if the project is ready for closure and assess the level of success of the project after it has been closed.

2.2 Stages Involved in Creating a Project Plan

First we need to define the work breakdown structure (WBS). This means listing the processes, activities and tasks required to undertake the project, as well as the key project milestones. Then you'll need to quantify the human resources required to carry out each activity listed.

The initial step involves building a project schedule which describes the low of project activities and the timeframes involved, as well as any plan in assumptions and constraints.

Figure 1 stages involved in creating a project plan

2.3 Define the Work Breakdown Structure

The First step to creating a detailed project plan for your project is to develop a comprehensive WBS, listing all of the activities and tasks required to undertake the project. The life cycle defines a methodology for improving the quality of software and the overall development process

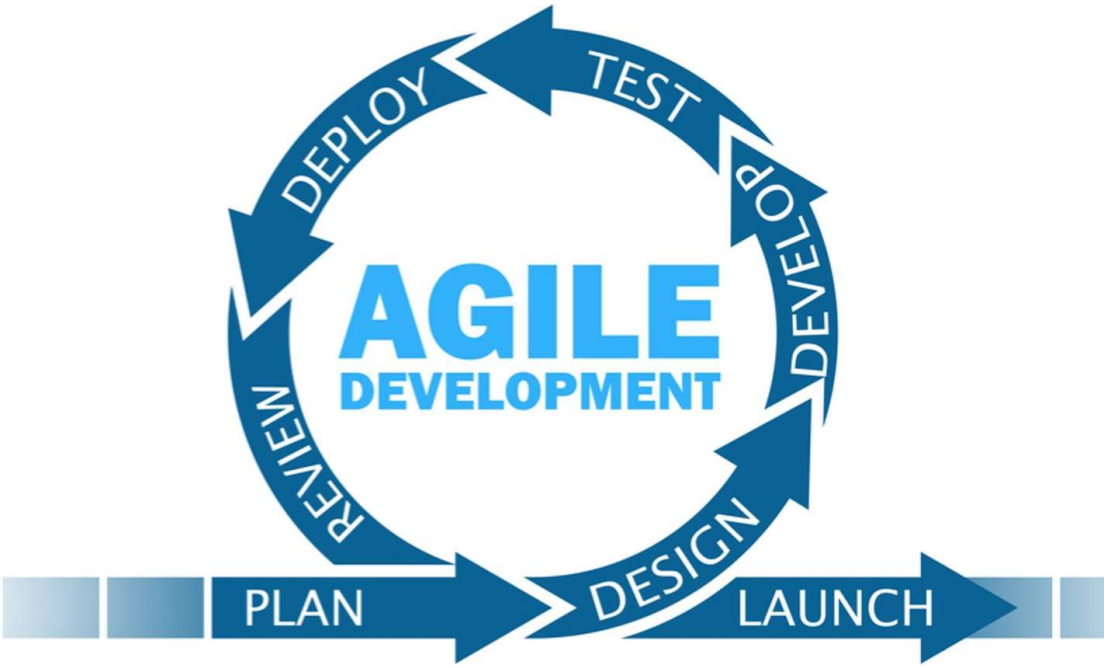


Figure 2 agile lifecycle

The project includes the following tasks:

1) Analysis and Data Collection
2) System Design
3) System Implementation
4) Testing
5) Documentation
6) Demo
7) Final Presentation

2.4 Identify the Required Resources

Having listed all the tasks required to undertake the project, you now need to identify the generic resources required to complete each task, as described in the table below. You will create a detailed list of the

Resources required for the project when you create a Resource Plan in the next stage

Tasks Description:-

A task is an item of work to be completed within a project. The table below will help us list each task, describing the tasks and identify the sequencing order that is appropriate. The completed activity and task lists will comprise the WBS for the project but we also need to specify any critical project milestones

• Data Collection Task

Involves actions and methods performed on data that help

Describe facts, detect patterns, develop explanations, and test

Hypotheses. This includes data quality assurance, statistical data

Analysis, modeling, and interpretation of analysis results.

• Involved to:-

1) Create new data
2) using your own previously collected data
3) collecting other data
4) purchasing data

- **System Design**

  Is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements?

  Systems design could be seen as the application of systems theory to product development

- **System Implementation**

  Describe the database environment where the software system and the database(s), if any is present, will be installed. Include a description of the diverse types of database and library environments (such as production, test, and training databases).

- **Documentation**

  Documentation is a set of documents provided on the paper used as a quick reference or a guidebook to help the reader understand what is going on.

- **System Testing**

  Is the type of testing to check the behavior of a complete and fully integrated software product based on the software requirements?

- **Demo**

  A system demo is a freely distributed piece of an upcoming or recently released app. Demos are typically released by the system publisher to help consumers get a feel of the system before deciding whether to buy the full version.

- **Presentation**

  Representing the final work and everything that was done project by explaining what was learned and what was done by whom and how long it took to be done.

2.5 Construct a Project Schedule

Project Schedule Create a detailed project schedule by listing the processes, activities and tasks required to complete the project, as well as the dependencies, and sequence.

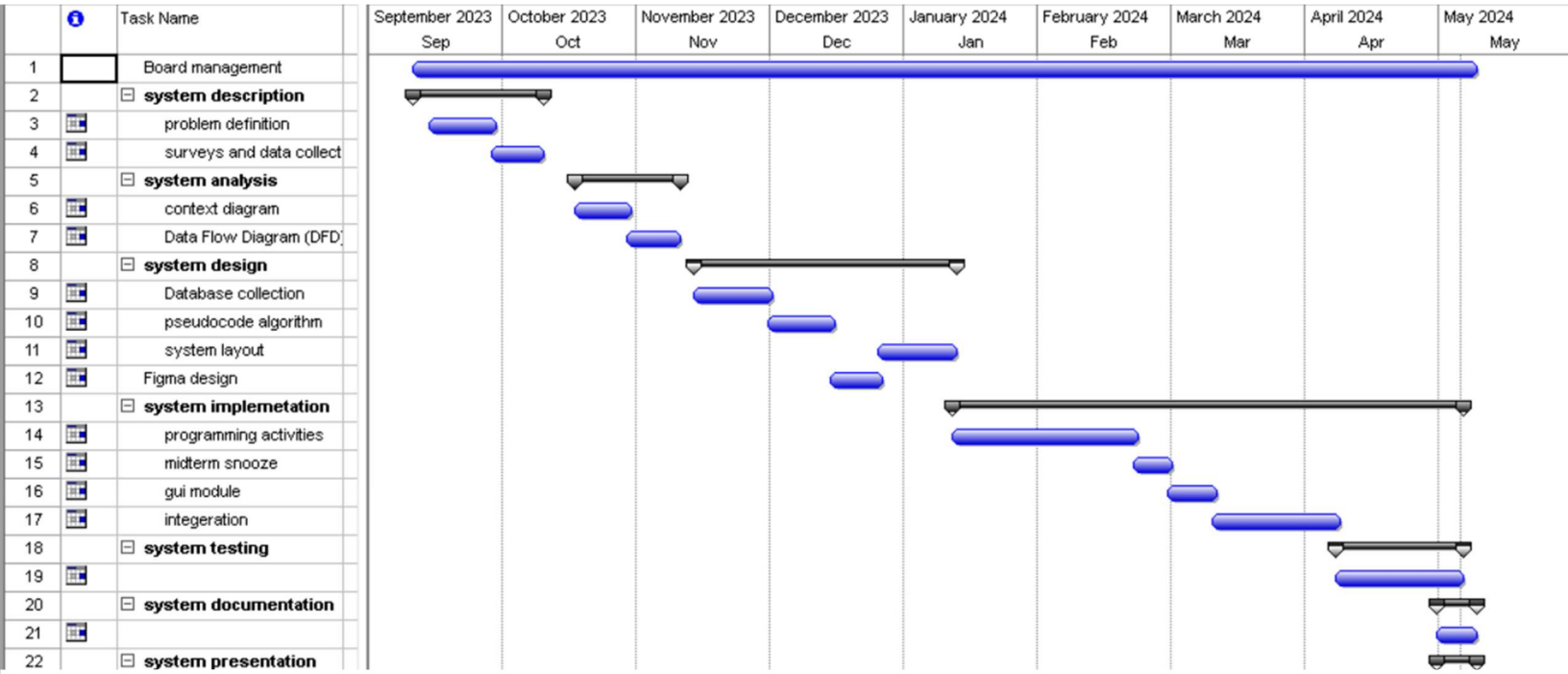


Figure 3 task assignment and gantt chart

Network diagram:-

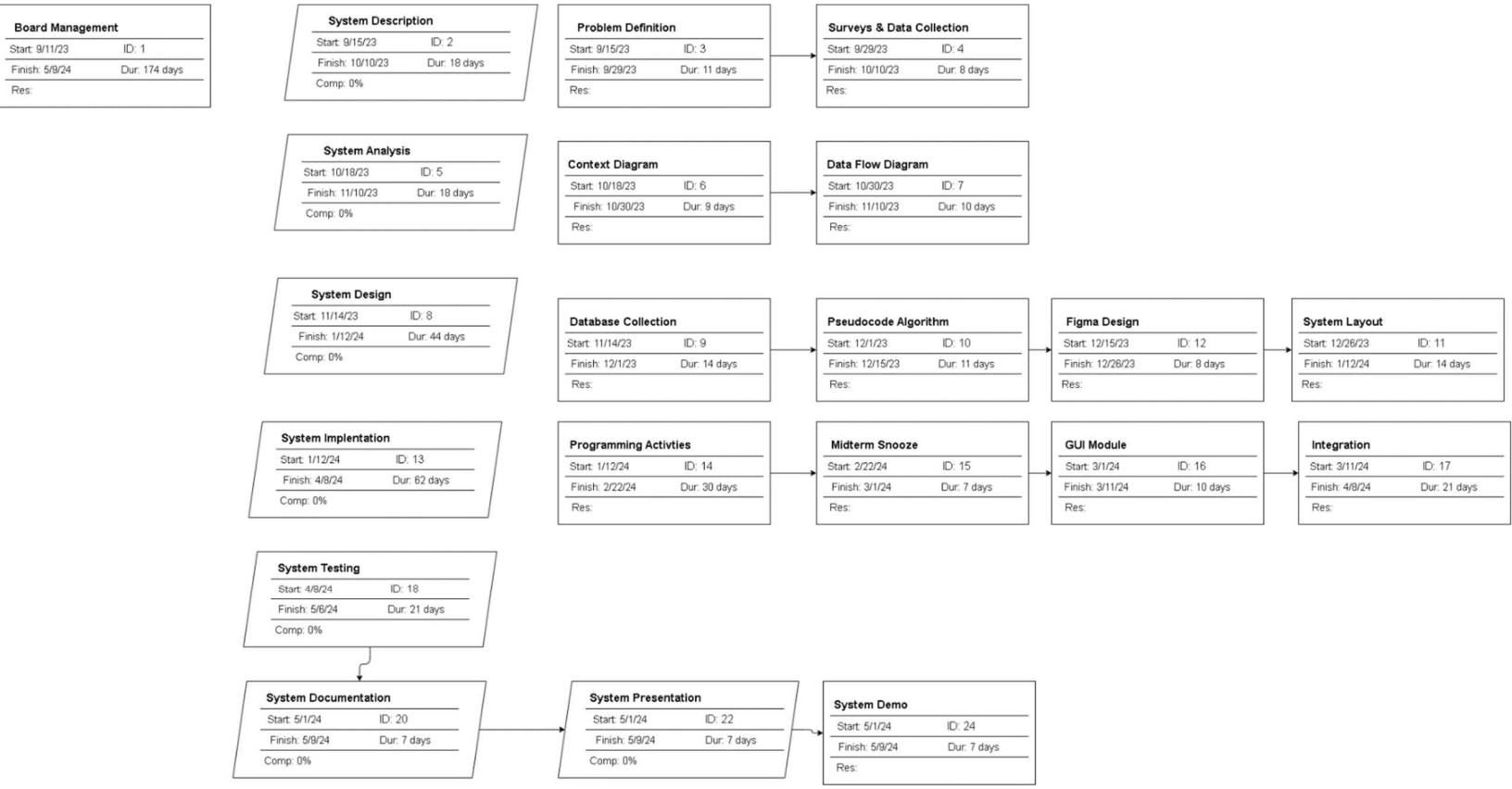


Figure 4 network diagram

2.6 Assumptions

List any assumptions made during this planning process. For example, it may be assumed that:

- The project will not change in scope.
- The resources identified will be available upon request.
- Approved funding will be available upon request.

2.7 Constraints

List any constraints identified during this planning process. For example:

- The project team must create all the physical deliverables within the allocated budget.
- Work must be undertaken within normal working hours only

2.8 functional view:-

1st Maintain Basic Data:

User:

First Name, Last Name, Email, Password, Phone, Address

Admin:

Username, Email, First Name, Last Name, Password

Log:

Username, Database Name, Log Entries

Reports:

Timestamp, Cleansing_Techniques_Used, Summary_Statistics, Missing_Values, Duplicates, Invalid Formats, Statistical_Measures, user_inputs, Data_Cleansing_Status

2nd generate reports:

Summary Report:

1. Total number of records processed.
2. Overall cleansing status (success/failure).
3. Timestamp of when the cleansing process occurred.
4. User responsible for the cleansing process.

Summary Statistics:

1. Negative Numbers Found: Number of negative values encountered.

Outliers:

1. Outliers Found: Number of outliers detected.
2. Outliers Removed: Number of outliers removed.
3. Percentage of outliers before and after removal.

Missing Values:

1. String: Number of missing string values.
2. Integer: Number of missing integer values.
3. Missing Values Found: Total number of missing values encountered.
4. Missing Values Removed: Number of missing values removed.
5. Percentage of missing values before and after removal.

Duplicates:

1. Duplicates Found: Number of duplicate records found.
2. Duplicates Removed: Number of duplicate records removed.
3. Percentage of duplicates before and after removal.

Invalid Formats:

1. Various types of invalid formats encountered (e.g., email, phone number, cases).
2. Invalid Formats Found: Total number of invalid format instances.
3. Invalid Formats Removed: Number of invalid format instances removed.
4. Percentage of invalid formats before and after removal.

Statistical Measures:

1. Mean: Average value of the dataset.
2. Median: Middle value of the dataset.
3. Mode: Most frequent value(s) in the dataset.
4. Standard Deviation: Measure of the dispersion of values from the mean.
5. Variance: Measure of how spread out the values are.
6. Minimum Value: Lowest value in the dataset.
7. Maximum Value: Highest value in the dataset.

Data Source Information:

1. File Name: Name of the file from which the data was sourced (if applicable).
2. Database Name: Name of the database from which the data was sourced (if applicable).
3. Table Name: Name of the table within the database from which the data was sourced (if applicable).

User Inputs:

1. Any additional information provided by the user during the cleansing process.

Data Cleansing Status:

1. Final status of the data cleansing process (e.g., completed, in progress, failed).

2.8 proposed approach solution

We chose to work as an agile self-developing team as we focused through the process in presenting deliverable items at the end of each sprint

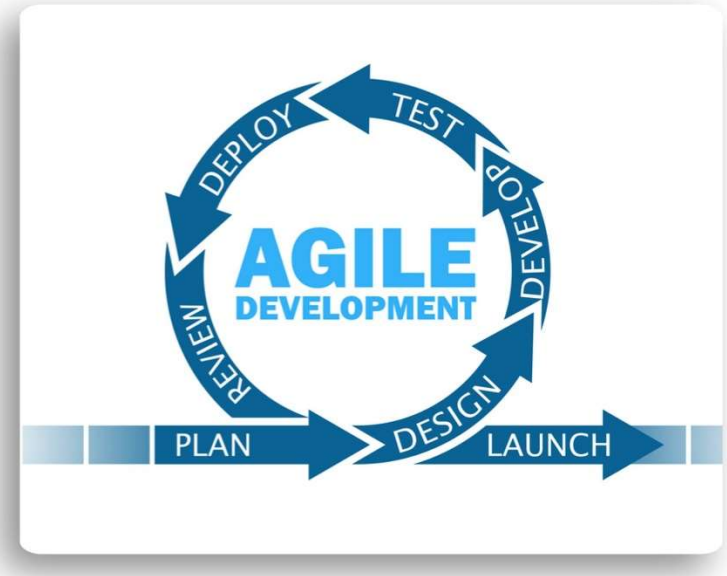


Figure 5 agile life cycle

2.9 Agile framework selection

We chose to work with a **Scrumban** framework that depended on many thing we did in the last sprint like

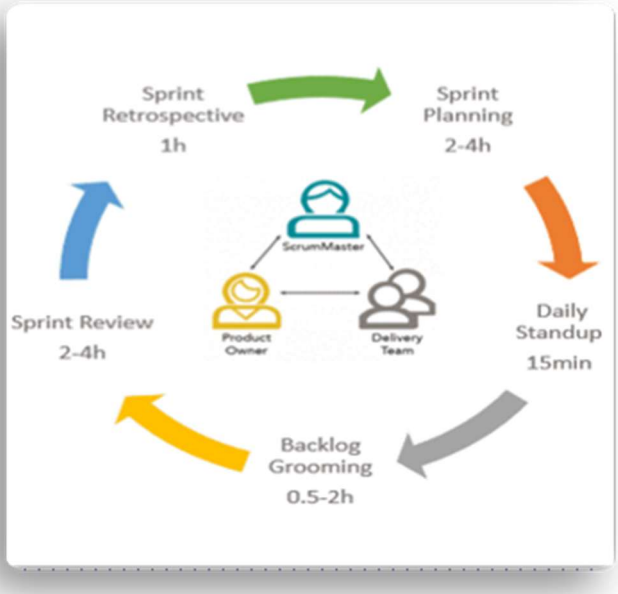


Figure 6 Scrum meetings lifecycle

2.10 Board management

We chose **Trello** software to manage our tasks where trello is a visual tool that empowers your team to manage any type of project, workflow, or task tracking. Add files, checklists, or even automation: Customize it all for how your team works best.

What is a board?

A board (A) represents a place to keep track of information — often for large projects, teams, or workflows. Whether you are launching a new website, tracking sales, or planning your next office party, a Trello board is the place to organize tasks, all the little details, and most importantly—collaborate with your colleagues.

What is a list?

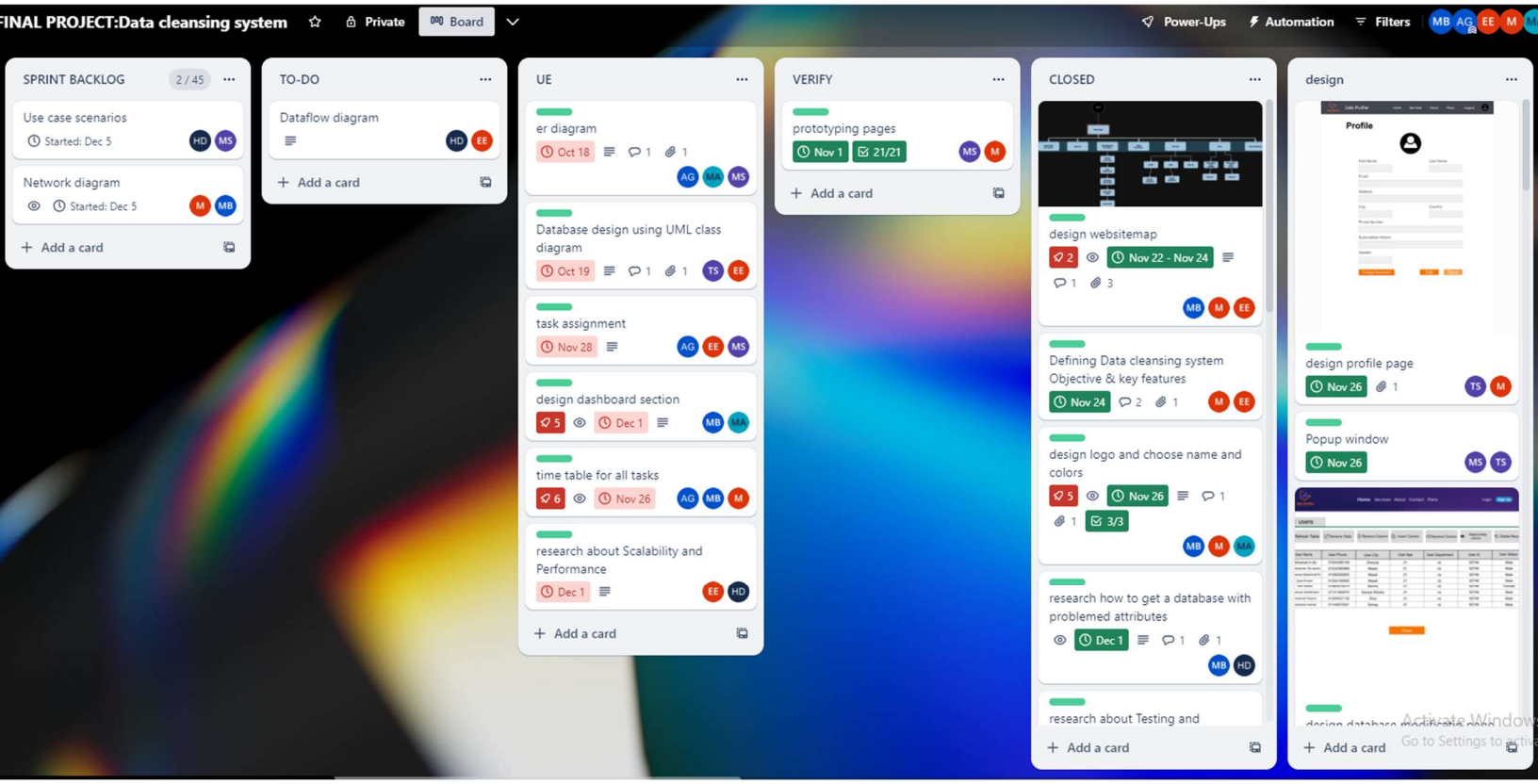
Lists (B) keep cards, or specific tasks or pieces of information, organized in their various stages of progress. Lists can be used to create a workflow where cards are moved across each step in the process from start to finish, or simply act as a place to keep track of ideas and information. There's no limit to the number of lists you can add to a board, and they can be arranged and titled however you'd like.

What is a card?

The smallest, but most detailed unit of a board is a card (C). Cards are used to represent tasks and ideas. A card can be something that needs to get done, like a blog post to be written, or something that needs to be remembered, like company vacation policies. Just click "Add a card…" at the bottom of any list to create a new card, and give it a name like "Hire a new marketing manager" or "Write a blog post."

What is the board menu?

On the right side of your Trello board is the menu (D)—the mission control center for your board. The menu is where you manage members' board permissions, control settings, search cards, enable Power-Ups, and create automations. You can also see all of the activity that has taken place on a board in the menu's activity feed. Take some time to check out everything the menu has to offer.



https://trello.com/b/5xonGb74/final-projectdata-cleansing-system

Figure 7 trello board management tool

How we setup deadlines in our tasks?

Figure 8 scrum poker estimation diagram

we used a technique called **scrum poker estimation**

What is scrum poker?

Scrum poker, also known as “planning poker” and “pointing poker”, is a gamified technique that development teams use to guess the effort of project management tasks. These estimations are based on the entire group’s input and consensus, making them more engaging and accurate than other methods. To help gauge the number of story points for the relevant tasks, teams use planning poker cards, which are similar to poker cards.

How does planning poker work?

At the beginning of a poker planning session, the product owner or customer reviews an agile user story and reads it aloud. A user story is a general and informal explanation of a software feature that describes how it will offer value to the end-user (i.e. the customer).

Step 1: Hand out the cards to participants

Distribute an identical deck of cards to everybody. Each one has a number that the team has agreed to use as their estimate. Each player should have a deck consisting of different numbers. Cohn recommended a sequence of 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100.

Other common sequences include doubling the next number (e.g. 1, 2, 4, 8, and so on). These values can represent a number of things: the number of story points, ideal days, or other units that the team uses for estimation.

The decks are intentionally kept minimal with considerable number-jumps. Doing this ensures that for each story, everyone can reach a consensus number. Otherwise, if they have a card for each number from one to 50, the process would be painfully slow.

Step 2: Read the story out loud

The moderator (either the product owner or product manager) narrates the story to the group. If participants have any questions, the moderator answers them.

Step 3: Discuss the story

Once the group finishes listening to the story, everyone shares their views on it. Some of these discussion points will likely include:

How should we handle the work?

How many people are expected to get involved?

What skills will be needed to work on the story?

How should we tackle any roadblocks that delay progress?

The group will also try to learn more about the story and ask questions to understand it better.

Step 4: Select and share

After the discussion, each person will privately select a card from the deck. Usually, it's used to show an estimate of story points (but can also be used to represent the number of ideal days). Once everyone selects a card, they show them at the same time.

If a player shows a higher card, it conveys that the story will be completed with greater difficulty and take a longer period to complete. Keep in mind that it's common for estimates to vary a lot.

Step 5: Reach a consensus

When team members show the same card, that number turns into a consensus. Now, the group can move forward and work on the next story.

However, if the cards continue to vary, then further discussions on the story will follow. Participants with higher or lower estimates than others will communicate their points of view. Then, they'll attempt to convince their teammates to understand their differing numbers.

Once this new deliberation ends, everyone will go through their deck and show them again. If a participant continues to agree with their last choice, then they will repeat the card or eventually choose a new one.

Usually, the estimates start to converge after the second round. If not, then the process repeats itself until the team agrees on a single number.

The benefits of scrum poker

According to one study, estimates from planning poker are statistically higher than individual ones. It was also noted that for the same tasks, planning poker estimates were more accurate than individual ones.

Other benefits include:

Estimating tasks relative to each other. Often, it can be tricky to gauge the time required to complete a project, especially when it's your first time. Planning poker familiarizes teams with assessing them. After playing the game for a while, you end up building a series of tasks that act as a future reference for comparison.

Lending an equal voice to everyone on the team. It can encourage newer employees to speak up by playing a card and explaining their logic. For example, imagine making a food reservation app. You and your colleague might give a smaller estimate, like 10 or 15. However, a new employee might give one of 100. Maybe they had experience with creating a similar app at their last job and knows that such an app takes a lot of time, especially compared to other ones.

Identifying gaps in requirement and implementation. When participants disclose their estimates, they will have to back them up with reasoning on why they are high or low. This can open up questions on the requirement and implementation – a feedback loop that can detect the gaps.

Who to include in scrum poker meetings

The right people should join the meeting or it becomes difficult to reap the benefits outlined above. These crucial roles include:

Scrum team members: scrum members deliver the items from the product backlog – a list of deliverables (e.g. new features). They will also provide their input for the discussions of story points.

Scrum master: a scrum leader is the facilitator in agile meetings. They should take part in all standard meetings.

Product owner: the owner or manager will describe all user stories to the team and answer their questions.

When to hold a planning poker session

Usually, teams arrange a session after creating the initial backlog. Although sessions can sometimes take more than one day, it leads to the development of initial estimates that are helpful in sizing or scoping the project.

Items are added to the product backlog incrementally throughout the project's lifespan. That's why it's usually more convenient for teams to conduct sessions once per iteration. In most cases, this happens some days after the iteration ends. Similarly, it also occurs right after a daily standup (a type of agile meeting) because the entire team is present.

2.11 workflow

We chose to work with **pair programming technique** where a task can never be hold by only one member

-pair programming is an integrated practice From **XP (extreme programming)** framework

-this can help the team to improve their teamwork skills and team management skills

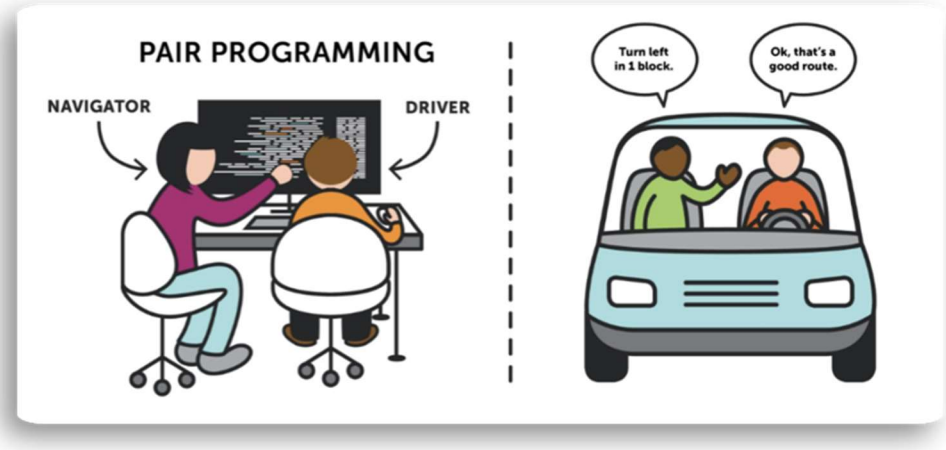


Figure 9 pair programmimg workflow

Chapter 3
Survey & Literature Review

Chapter 3 – Survey & Literature Review

3.1 Introduction

We will review some of the data cleansing systems listed in the top review before and build a comparison to show developments about the use of data cleansing systems reviewing previous works is about technology used at present technical possibilities. By addressing some of the techniques that modern data cleansing systems can understand the language user through it

We will explain:-

An introduction to previous works that revolve around our topic make a comparison between some works to clarify the best aspect of either of them in terms of technology, purpose, advantages, and disadvantages clarification of some topics on which our topic depends

3.2 Related Work

We found similar apps and similar research papers giving an idea about our desired system. In this section we made a comparison between a number of applications, each with its year, objective, technology, results, advantages and disadvantages, let's show the most interactive applications using a specific technology.

#	Name	Link	Advantages	Disadvantages
1	**Winpure**	https://winpure.com/	• Industry-leading Data Matching Tool, quick, intelligent and easy-to-use	• **Complexity:** Some users have found the software to be complex, Requiring training to fully utilize all its features. • **Exporting Results:** Exporting the Results of matching can be cumbersome and not user-friendly.
2	**OpenRefine**	https://openrefine.org/	• It has an easy to use interface • OpenRefine supports a wide variety of data	• Some users complain that the user interface looks a bit dated

3	Knowledge Net	https://kn-it.com/	• Connected view of relationships beyond 1st & 2nd degree connections. • ease of use and the insights that KN gives on people	• Not having a tool prior to Knowledge Net and not fully leveraging the power of the tool on clients
4	Demand tools	https://www.validity.com/demandtools/	• Straightforward UI that does what it needs to do. • Makes after-the-fact de-duplication a breeze. • Easy mass updates to data sets - making cleaning data a breeze.	• UI is outdated and could use a bit of a facelift. • Some competitors in this space are more all-in-one tools where DemandTools. • Lack of automation for regular data cleansing - everything we use DemandTools for is manual.
5	Sage master	https://sagemaster.io/	• Streamlined Financial Management • Enhanced accuracy • Time and cost savings • Scalability • Reporting and analytics	• Initial setup complexity • High Learning curve • Cost of ownership • Limited mobility • Compatibility issues

3.3 Related Papers

A survey was done on 12 research papers which includes the required technologies needed to create the desired project. Those papers will be briefly discussed and the tables following will include the final results that are derived from the paper.

#	title	objective	result	Advantage	Disadvantage	limitation
1	"Exploring Data Profiling Techniques for Enhanced Data Cleansing Systems"	To investigate various data profiling methods for improving the effectiveness of data cleansing systems.	Identification of key data profiling techniques and their applicability in optimizing data cleansing processes.	Enables data cleansing systems to gain deeper insights into data characteristics and anomalies.	May require substantial computational resources and expertise to implement sophisticated profiling techniques.	Profiling techniques may not be universally applicable and may need customization based on data types and sources.
2	"Automation Strategies for Streamlining Data Cleansing Processes"	To develop automation strategies that reduce manual intervention in data cleansing tasks.	Implementation of automated workflows and algorithms for efficient and scalable data cleansing.	Increases operational efficiency and reduces the time required for data cleansing.	Over-reliance on automation may overlook nuanced data issues that require human intervention.	Automation may be challenging to implement in complex data environments with diverse data formats.
3	"Implementing Effective Data Governance Practices in Data Cleansing Systems"	To integrate robust data governance principles into data cleansing workflows.	Establishment of policies, procedures, and controls to ensure data quality and compliance during cleansing processes.	Enhances data integrity, security, and regulatory compliance.	Implementation of stringent governance practices may introduce additional overhead and complexity.	Data governance frameworks may vary in effectiveness across different organizational contexts.
4	"Anomaly Detection Methods for Proactive Data Cleansing and Quality Assurance"	To develop anomaly detection techniques for identifying irregularities in datasets and initiating proactive data cleansing.	Deployment of anomaly detection algorithms to detect and address data inconsistencies and outliers.	Enables early detection and mitigation of data anomalies, improving overall data quality.	False positives may lead to unnecessary data cleansing efforts and potential loss of valid information.	Anomaly detection algorithms may struggle with detecting subtle anomalies or patterns in noisy data.
5	"Developing Robust Data Quality Metrics for Assessing Data Cleansing Effectiveness"	To design comprehensive data quality metrics for evaluating the effectiveness of data cleansing efforts.	Creation of quantitative measures to assess data accuracy, completeness, consistency, and timeliness post-cleansing.	Facilitates objective evaluation of data cleansing outcomes and performance improvement over time.	Defining universal data quality metrics may be challenging due to the subjective nature of data quality.	Metrics may not capture qualitative aspects of data quality, such as relevance and interpretability.
6	"Strategies for Identifying and Handling	To develop strategies for identifying and	Implementation of techniques for attribute profiling,	Improves the accuracy and reliability of data	Manual identification and handling of	Strategies may vary in effectiveness

	Problematic Attributes in Databases for Data Cleansing"	addressing problematic attributes that hinder data quality.	anomaly detection, and targeted cleansing.	by addressing specific issues at the attribute level.	problematic attributes may be time-consuming and error-prone.	depending on the complexity and diversity of data attributes.
7	"A Comparative Analysis of Data Cleaning Techniques for Improved Data Quality"	To conduct a comparative analysis of various data cleaning techniques to determine their effectiveness in improving data quality.	Evaluation of different data cleaning methods, such as deduplication, standardization, and error correction, in terms of their impact on data quality.	Provides insights into the strengths and weaknesses of different data cleaning approaches, aiding in informed decision-making.	Comparative analysis may overlook contextual factors that influence the suitability of cleaning techniques for specific datasets.	Results may not generalize to all datasets and may require validation in diverse data environments.
8	"Scalability and Performance Optimization in Large-Scale Data Cleansing Systems"	To address scalability and performance challenges in data cleansing systems when dealing with large-scale datasets.	Development of optimization strategies, parallel processing techniques, and distributed computing architectures to enhance scalability and performance.	Enables data cleansing systems to handle massive volumes of data efficiently, reducing processing time and resource consumption.	Implementation of scalable solutions may require significant upfront investment in infrastructure and technology.	Scalability enhancements may introduce complexity and overhead, impacting system maintainability and agility.
9	"Efficient Data Transformation Approaches for Seamless Integration in Data Cleansing Pipelines"	To explore efficient data transformation techniques that facilitate seamless integration into data cleansing pipelines.	Identification and implementation of data transformation methods, such as normalization, aggregation, and enrichment, to prepare data for cleansing.	Streamlines data preprocessing tasks and enhances compatibility with downstream cleansing processes.	Complex transformation requirements may necessitate custom development and configuration, increasing implementation effort.	Data transformation approaches may not adequately address all data compatibility and integration challenges, requiring ongoing refinement.
10	"Testing and Validation Frameworks for Ensuring the Accuracy and Reliability of Data Cleansing Processes"	To establish testing and validation frameworks for assessing the accuracy and reliability of data cleansing processes.	Development of systematic approaches, test suites, and validation metrics to verify the effectiveness of cleansing techniques.	Enhances confidence in data cleansing outcomes and validates adherence to predefined quality standards.	Testing and validation processes may introduce additional overhead and delay project timelines.	Validation frameworks may not capture all aspects of data quality and may require continuous refinement to align with evolving requirements.
11	"Training and Documentation Strategies for Empowering Users	To design training and documentation strategies that	Creation of user-friendly training materials, tutorials, and documentation	Promotes user engagement, knowledge transfer, and self-	Developing comprehensive training and documentation	User training and documentation may not fully address

	in Data Cleansing Systems"	empower users to effectively utilize data cleansing systems.	resources to facilitate user adoption and proficiency.	sufficiency in data cleansing operations.	resources may require significant time and effort.	individual learning preferences and skill levels, necessitating ongoing support and education.
12	"Data Integration Techniques for Seamlessly Incorporating Cleansed Data into Organizational Workflows"	To explore data integration techniques that enable seamless incorporation of cleansed data into organizational workflows.	Implementation of data integration mechanisms, such as APIs, ETL processes, and data warehouses, to facilitate data flow and interoperability.	Facilitates the integration of cleansed data with existing systems, applications, and business processes, enhancing data accessibility and utilization.	Data integration efforts may encounter compatibility issues, data format mismatches, and synchronization challenges.	Integration techniques may require ongoing maintenance and updates to adapt to evolving data requirements and technological landscapes.

1. **Exploring Data Profiling Techniques for Enhanced Data Cleansing Systems**
   - This research investigates various data profiling methods to improve data cleansing systems by gaining deeper insights into data characteristics and anomalies.
2. **Automation Strategies for Streamlining Data Cleansing Processes**
   - This study focuses on developing automation strategies to reduce manual intervention in data cleansing tasks, thereby enhancing operational efficiency and scalability.
3. **Implementing Effective Data Governance Practices in Data Cleansing Systems**
   - The objective is to integrate robust data governance principles into data cleansing workflows, ensuring data quality, security, and regulatory compliance.
4. **Anomaly Detection Methods for Proactive Data Cleansing and Quality Assurance**

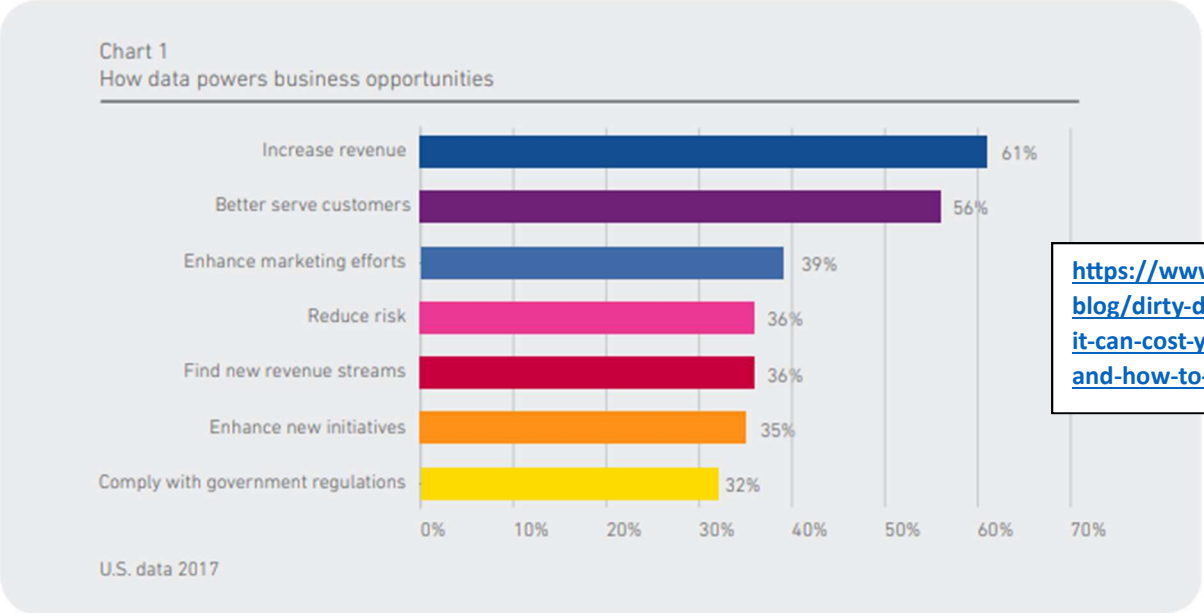
- This research aims to develop anomaly detection techniques to identify irregularities in datasets and initiate proactive data cleansing, improving overall data quality.

5. **Developing Robust Data Quality Metrics for Assessing Data Cleansing Effectiveness**
   - The goal is to design comprehensive data quality metrics to evaluate the effectiveness of data cleansing efforts objectively and improve data quality over time.
6. **Strategies for Identifying and Handling Problematic Attributes in Databases for Data Cleansing**
   - This study focuses on developing strategies to identify and address problematic attributes hindering data quality, thereby improving overall data reliability.
7. **A Comparative Analysis of Data Cleaning Techniques for Improved Data Quality**
   - This research conducts a comparative analysis of different data cleaning techniques to determine their effectiveness in improving data quality, aiding in informed decision-making.
8. **Scalability and Performance Optimization in Large-Scale Data Cleansing Systems**
   - The objective is to address scalability and performance challenges in data cleansing systems when dealing with large-scale datasets, reducing processing time and resource consumption.
9. **Efficient Data Transformation Approaches for Seamless Integration in Data Cleansing Pipelines**
   - This study explores efficient data transformation techniques to prepare data for cleansing and ensure compatibility with downstream processes, streamlining data preprocessing tasks.
10. **Testing and Validation Frameworks for Ensuring the Accuracy and Reliability of Data Cleansing Processes**
    - This research establishes testing and validation frameworks to verify the accuracy and reliability of data cleansing processes, enhancing confidence in data quality outcomes.

11. **Training and Documentation Strategies for Empowering Users in Data Cleansing Systems**

- The goal is to design training and documentation strategies to empower users in effectively utilizing data cleansing systems, promoting user engagement and proficiency.

12. **Data Integration Techniques for Seamlessly Incorporating Cleansed Data into Organizational Workflows**
    - This study explores data integration techniques to seamlessly incorporate cleansed data into organizational workflows, enhancing data accessibility and utilization.



https://www.cazoomi.com/blog/dirty-data-how-much-it-can-cost-your-business-and-how-to-get-rid-of-it/

Figure 10 how data powers business opportunities

What Do You Use Data for?

Typically, data is analyzed to help businesses predict future or emerging consumer behavior. This process involves understanding the data collected and knowing how to organize it, analyze it, and apply it to improve marketing efforts.

Most marketers use the same data to improve and personalize their customer experience. Seventy-eight percent of organizations say that the data they collect helps them increase customer acquisitions and lead conversions.

Data-driven marketing has, therefore, become a crucial strategy for marketers who want to succeed in the current hypercompetitive economy. Ninety-five percent of organizations are using data to power their business opportunities.

However, this can only occur if your data is clean. Dirty data can negatively affect your bottom line. Eighty-nine percent of business executives say that inaccurate data affects their ability to provide great customer experiences.

When your company is swimming in dirty data, the sales and marketing teams cannot make well-informed and accurate, data-driven decisions. On average, organizations believe that 25 percent of their data is inaccurate, a factor that impacts on the bottom line.

Dirty data can be costly in the long run. It can eventually lead to lower productivity, unnecessary spending, and unreliable decision-making.

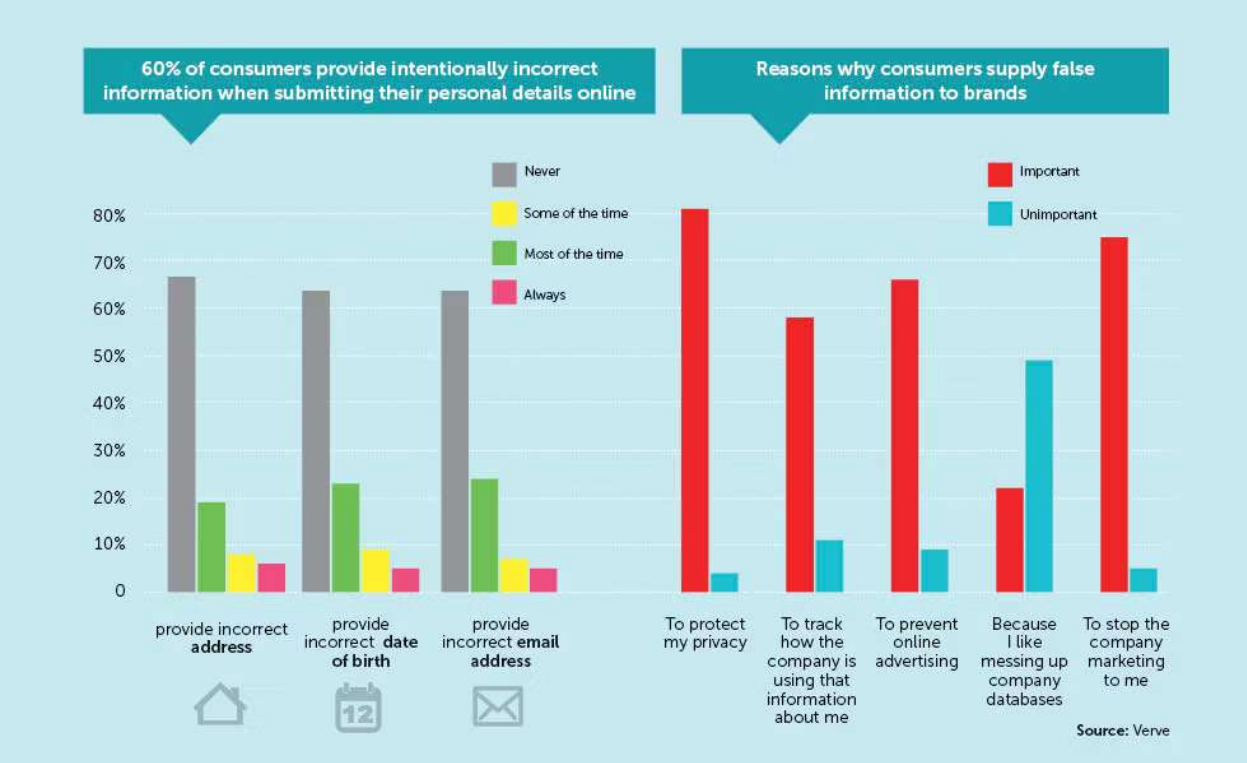


Figure 11 wh users give false data to brands databases

Why do we give false data to brands databases?

People are deliberately giving brands false data about themselves to protect their privacy, and are ignoring brands' efforts to empower them to take control of their data, according to a study of more than 2,400 UK consumers by research company Verve.

The research shows that 60% of consumers intentionally provide incorrect information when submitting their personal details online. Broken down by the types of data provided, birth dates are the most commonly falsified, as almost a quarter of consumers (23%) give the wrong date of birth to companies 'some of the time', 9% do this 'most of the time' and 5% 'always' give the wrong date.

The research also shows that nearly one-third of people give a fake email address and a made-up name at least some of the time. It is a similar story for incorrect information given about home addresses, phone numbers, job titles and company names.

“The upside of providing information has not been articulated,” says managing director at Verve Colin Strong. “The case is not always made by companies about what consumers

are going to get in return for providing information, but people see the immediate effects of being put on more marketing lists and being pursued by online advertising and email spam.”

The study therefore looks at why people give false data. Privacy concerns top the list with 81% of people saying this is important, while 77% believe that the information being requested is intrusive and 76% say it is unnecessary.

Also high on the list of reasons for lying on forms is to stop companies marketing to them – 75% of consumers say this – and two thirds say their motivation for giving false information is to prevent online advertising. Malice towards companies and embarrassment about having internet browsing behavior tracked are less common reasons.

Vanda Candeias

Digital & E-Commerce Transformation Director

https://www.linkedin.com/pulse/why-do-we-give-false-data-brands-bases-vanda-candeias/

Chapter 4
Analysis Phase

Chapter-4-Analysis phase

4.1 Introduction

The output gained from chapter three is an overview of the system's functionality and operations. Based on this output, the system analysis phase will take place, which involves going through all expected scenarios in order to create our system in terms of what the system will do, how each part of the system will be linked with other parts, and how it will be implemented from scratch.

4.2 System Scenario

1- User Login: Mohammed, a Data Analyst, starts his day by accessing the data cleansing system. He navigates to the login page and enters his username and password.

2- Multi-Factor Authentication: The system prompts Mohammed for multi-factor authentication. He receives a code on his registered mobile device, enters it, and gains access to the system.

3- Dashboard Overview: Once logged in, Mohammed is presented with a dashboard displaying recent activities, status of ongoing data cleansing processes, and notifications.

4- Upload Raw Data: Mohammed initiates a new data cleansing task by selecting the "Upload Data" option. He uploads raw data files from his local machine, including CSV and Excel files.

5- Select Cleansing Techniques: Mohammed reviews the uploaded data and selects specific cleansing rules based on the data quality requirements. She chooses rules for handling missing values, removing duplicates, and standardizing formats.

6- Initiate Cleansing Process: With the selected rules in place, Mohammed initiates the data cleansing process. The system begins processing the data according to the defined rules.

7- Validation and Sanitization: The system validates and sanitizes the data, identifying and removing any anomalies, errors, or potential threats. It logs the cleansing activities for auditing purposes.

8- Data Review: Once the cleansing process is complete, Mohammed reviews the cleansed data using interactive visualization tools and data preview features provided by the system.

9- Approval of Cleansed Data: Satisfied with the quality of the cleansed data, Mohammed approves it for further use. He acknowledges the system-generated summary of changes made during the cleansing process.

10- Export Cleansed Data: Mohammed exports the cleansed data in CSV format to share with other team members for analysis. The system generates a download link, and Mohammed saves the file to his local machine.

11- Generate Data Cleansing Report: Before logging out, Mohammed generates a comprehensive report summarizing the data cleansing activities, including the number of records processed, errors identified, and the effectiveness of applied rules.

12- Logout: Mohammed logs out of the data cleansing system, ensuring the termination of her session to maintain security.

13-Missing Values Detection: The system identifies columns with missing values and provides options for handling them, such as imputation or removal.

14-Imputation Strategy Selection: Mohammed selects the imputation strategy for handling missing values, opting for mean imputation for numerical columns and mode imputation for categorical columns.

15-Invalid Format Detection: The system detects columns with invalid formats, such as dates or phone numbers, and suggests standardization methods.

16-Standardization Method Selection: Mohammed chooses to standardize date formats to YYYY-MM-DD and phone numbers to a specific pattern.

17-Duplicate Records Identification: The system identifies duplicate records based on specified criteria, such as unique identifiers or combinations of attributes.

18-Duplicate Removal Strategy Selection: Mohammed selects a strategy for removing duplicate records, opting for keeping the first occurrence and discarding subsequent duplicates.

19-Outlier Detection: The system detects outliers in numerical columns using statistical methods like Z-score or IQR.

20-Outlier Removal Strategy Selection: Mohammed selects a strategy for handling outliers, choosing to remove them based on a predefined threshold.

21-Progress Monitoring: Mohammed monitors the progress of the cleansing process through the dashboard, observing the completion status and any encountered errors.

22-Error Handling: If errors occur during cleansing, Mohammed receives notifications and accesses detailed error logs to troubleshoot and resolve issues.

23-Data Quality Assessment: After cleansing, Mohammed assesses the data quality metrics provided by the system, including completeness, accuracy, and consistency.

24-Preview Cleansed Data: Mohammed previews the cleansed SQL data to ensure it meets the desired quality standards before proceeding.

25-User Access Management: Mohammed manages user access to the cleansing system, granting permissions based on roles and responsibilities.

26- Audit Trail Review: Mohammed reviews the audit trail of cleansing activities to track changes made to the data and maintain data governance compliance.

27- Documentation Generation: Mohammed generates documentation summarizing the cleansing process, techniques applied, and results achieved for record-keeping and compliance purposes.

28- Performance Optimization: Mohammed optimizes the cleansing process by fine-tuning parameters, such as batch size or parallel processing, to enhance efficiency and reduce processing time.

29 - Feedback Submission: Mohammed provides feedback on the cleansing system's performance and suggests improvements for future enhancements.

30 - Training and Support: Mohammed accesses training materials and support resources provided by the system vendor to enhance his skills and troubleshoot issues effectively.

31- System Updates: Mohammed receives notifications about system updates and new features, ensuring he stays informed about the latest improvements and enhancements.

4.3 DFD diagram

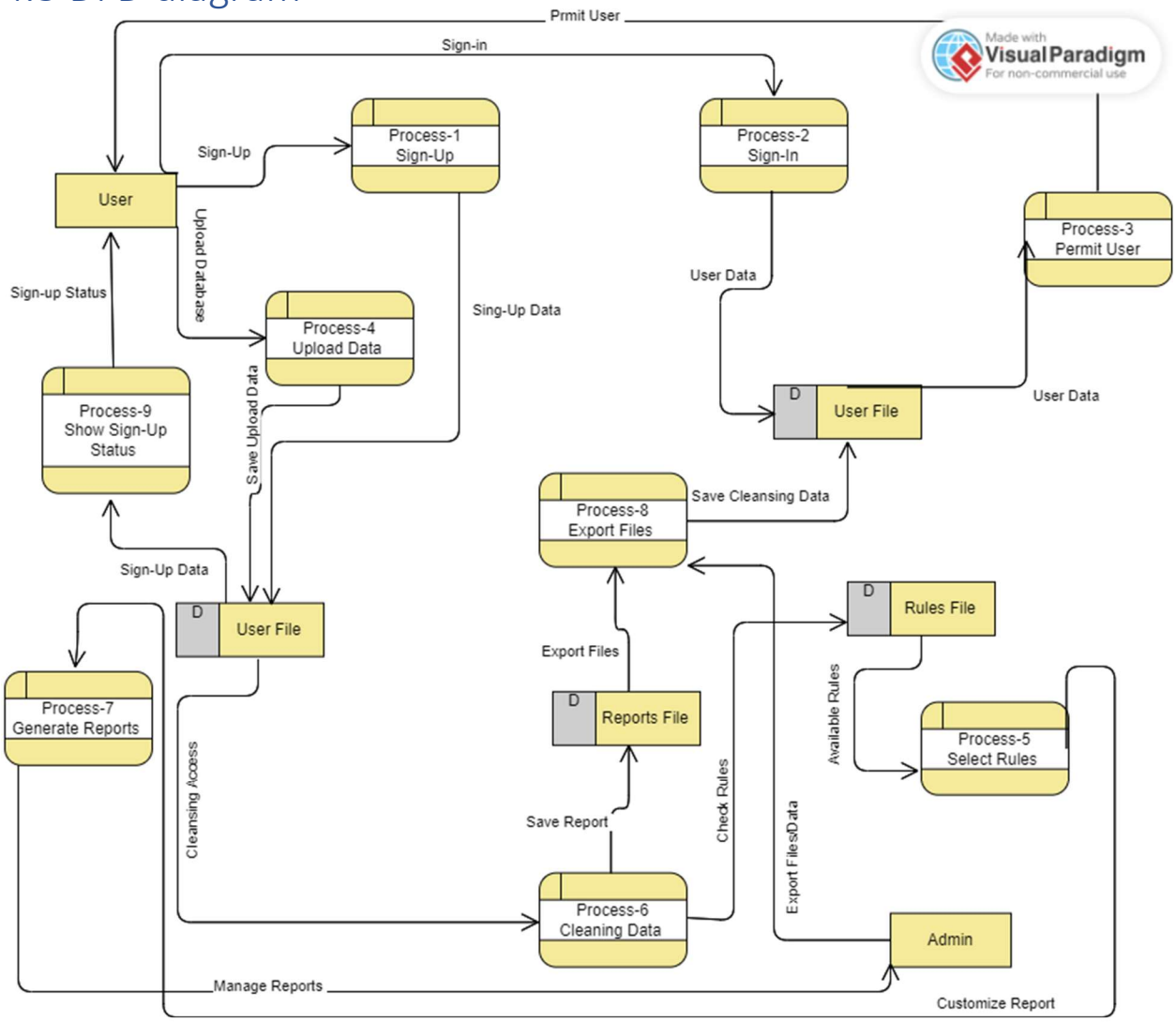


Figure 12 DFD diagram

DFD Level (0) Process:-

1.0: Sign-Up ->

The user initiates the sign-up process by providing their user data.

2.0: Sign-In ->

The user signs in to the system using their credentials.

3.0: Permit User ->

The admin permits the user to access the system.

4.0: Upload Data ->

The user uploads their data to the system.

5.0: Select Rules -> The user selects the rules for data cleansing.

6.0: Cleaning Data -> The system cleanses the uploaded data based on the selected rules.

7.0: Generate Reports -> The system generates reports based on the cleansed data.

8.0: Export Files -> The user exports the generated reports in a file format.

9.0: Show Sign-Up Status -> The system displays the sign-up status to the user.

DFD Level (0) Data stores:-

1. User File
2. Rules File
3. Reports File

DFD Level (0) Data entities:-

1. User
2. Admin

4.4 Context diagram:-

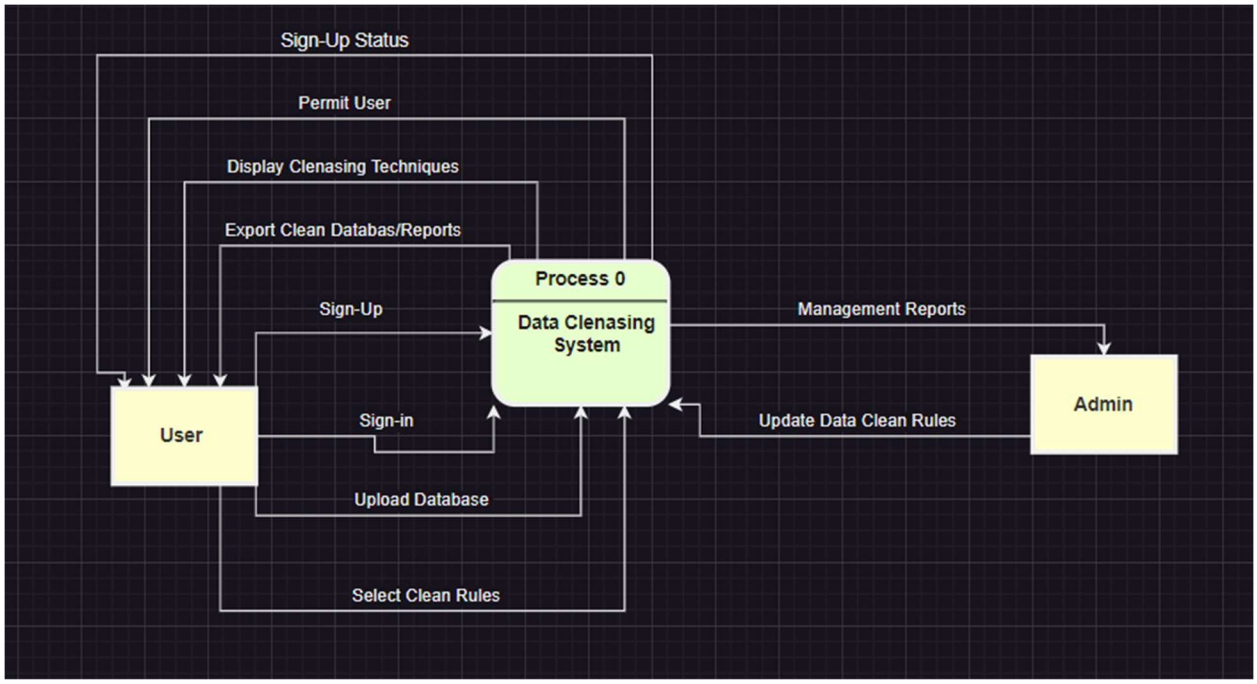


Figure 13 context diagram

Context diagram entities:-

1. User File
2. Admin

4.5 Required Database Type

By having large amounts of data, it's required to have a suitable database to be able to store this amount of data. In our case, using NoSQL database type enables the storage of the system to be simpler having data in a JSON-like tree structure and storing data in documents rather than tables. It handles unstructured data better than relational database types. One of these databases is called Firebase, which is a cloud-hosted NoSQL database, meaning that data is always available for retrieval.

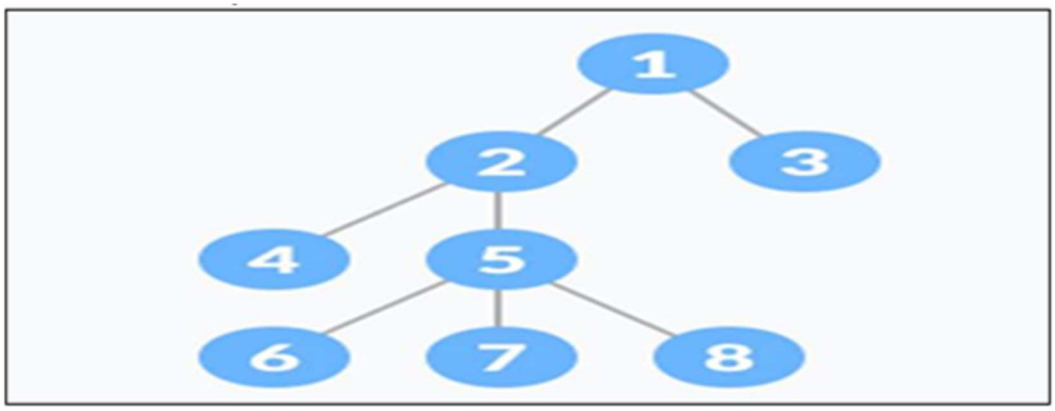


Figure 14 NOSQL Structure

4.6 Site map

A hierarchical diagram that represents the different pages of the system by visualizing a map that provides the flow of the system, the relationship between each page, and information about each.

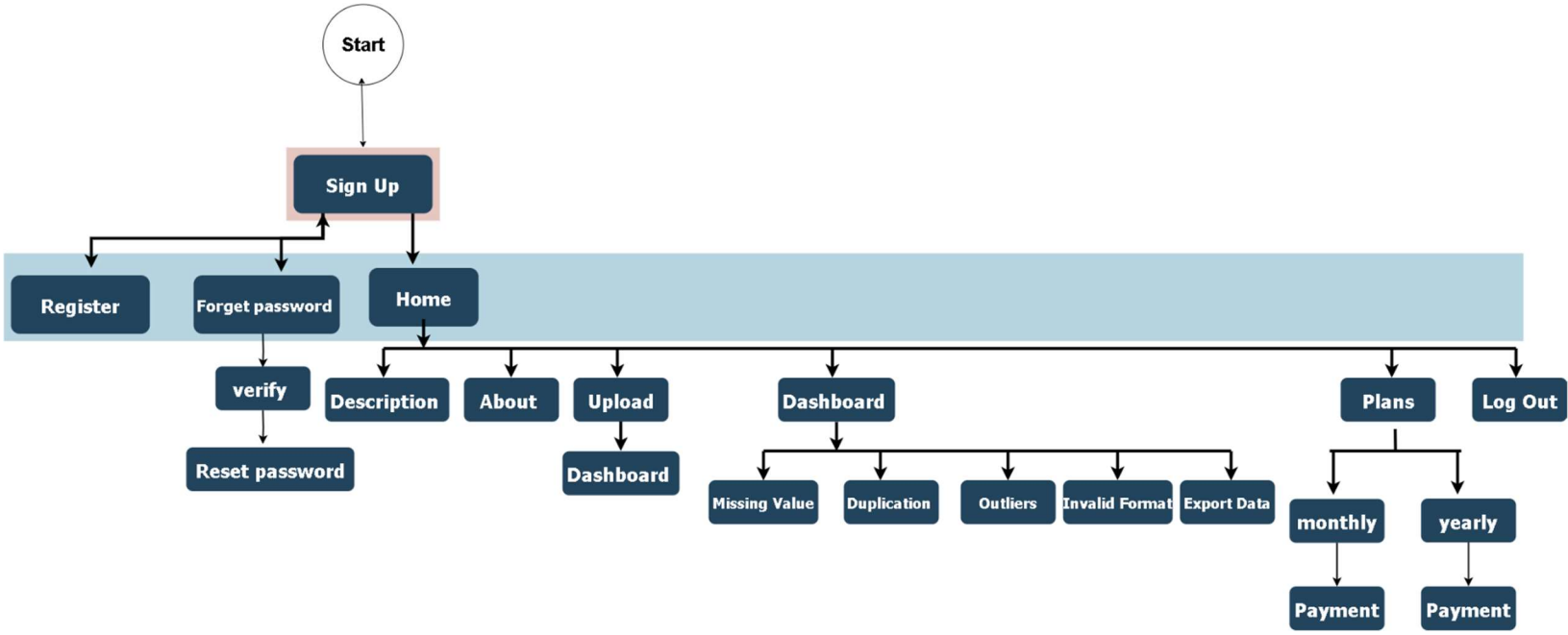


Figure 15 Sitemap

Chapter FIVE

Design Phase

Chapter 5 – design phase

5.1 Introduction

The design phase aims to construct the system by planning the development of the system, which is achieved by having all the basic concepts of the system understood, gathering all the data needed, and clarifying how each part will be connected with other parts in order to achieve the full functionality of the system. This chapter aims to provide flowcharts and diagrams that describe the functionality of the different parts of the system.

5.2 Flowcharts and pseudocodes

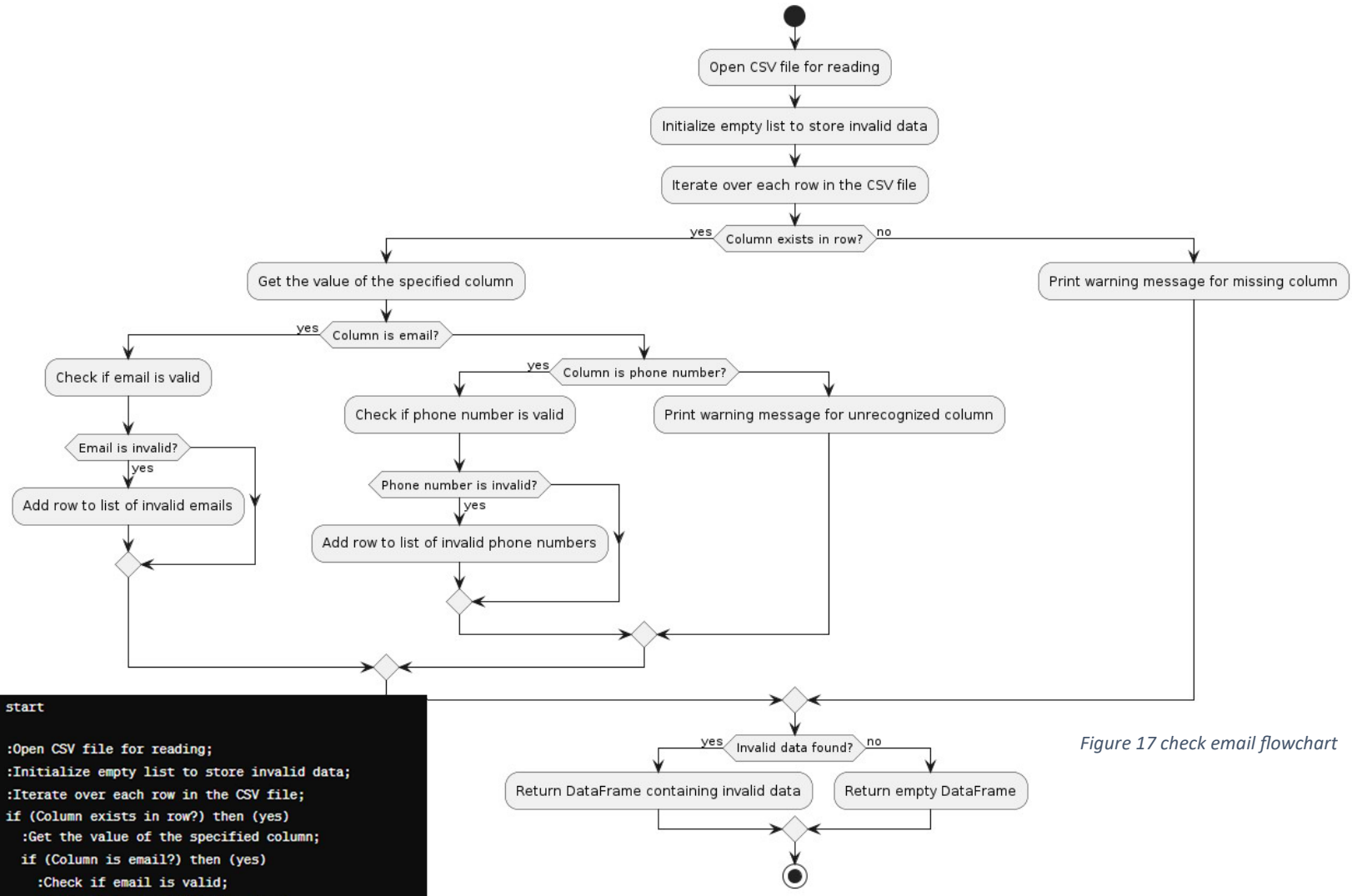


Figure 17 check email flowchart

```
start

:Open CSV file for reading;
:Initialize empty list to store invalid data;
:Iterate over each row in the CSV file;
if (Column exists in row?) then (yes)
  :Get the value of the specified column;
  if (Column is email?) then (yes)
    :Check if email is valid;
    if (Email is invalid?) then (yes)
      :Add row to list of invalid emails;
    endif
  else
    if (Column is phone number?) then (yes)
      :Check if phone number is valid;
      if (Phone number is invalid?) then (yes)
        :Add row to list of invalid phone numbers;
      endif
    else
      :Print warning message for unrecognized column;
    endif
  endif
else (no)
  :Print warning message for missing column;
endif

if (Invalid data found?) then (yes)
  :Return DataFrame containing invalid data;
else (no)
  :Return empty DataFrame;
endif

stop
```

Figure 16 check email pseudocode

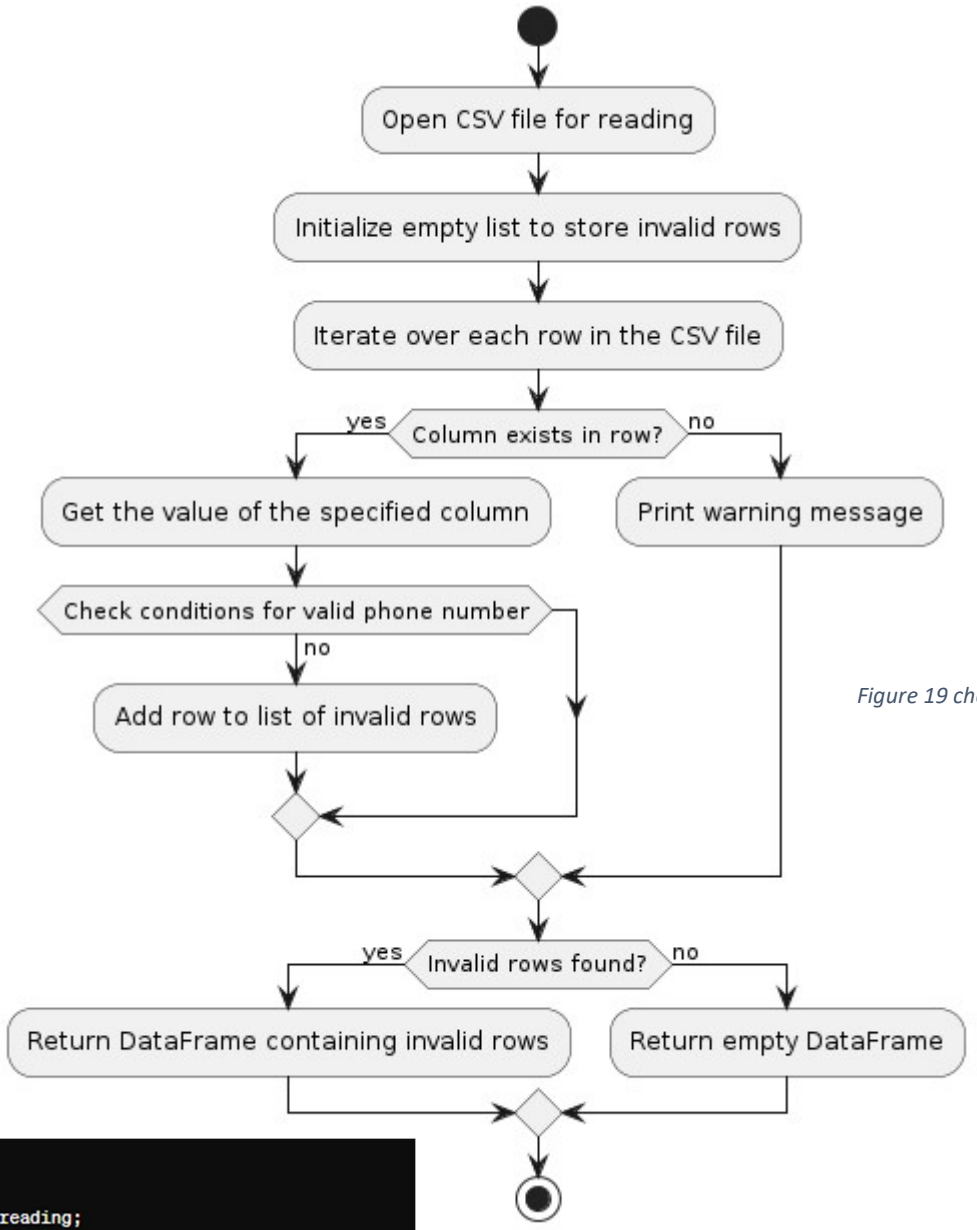


Figure 19 check phone number flowchart

```
start

:Open CSV file for reading;
:Initialize empty list to store invalid rows;
:Iterate over each row in the CSV file;
if (Column exists in row?) then (yes)
  :Get the value of the specified column;
  if (Check conditions for valid phone number) then (no)
    :Add row to list of invalid rows;
  endif
else (no)
  :Print warning message;
endif

if (Invalid rows found?) then (yes)
  :Return DataFrame containing invalid rows;
else (no)
  :Return empty DataFrame;
endif

stop
```

Figure 18 check phone number pseudocode

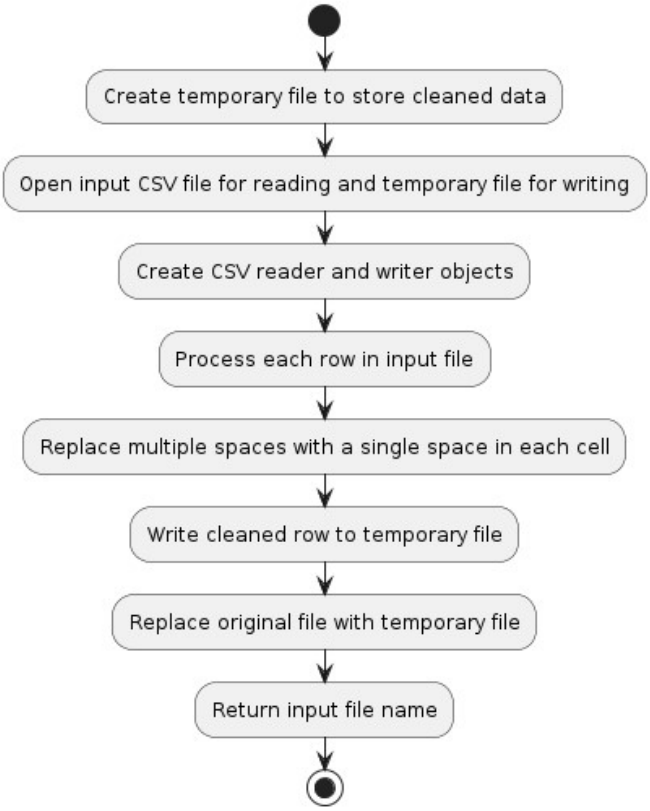


Figure 25 remove double spaces flowchart

```
start

:Create temporary file to store cleaned data;
:Open input file for reading and temporary file for writing;
:Create CSV reader and writer objects;
:Process each row in input file;
:Remove leading and trailing spaces from each cell;
:Write cleaned row to temporary file;
:Replace original file with temporary file;
:Return input file name;

stop
```

Figure 24 remove double spaces pseudocode

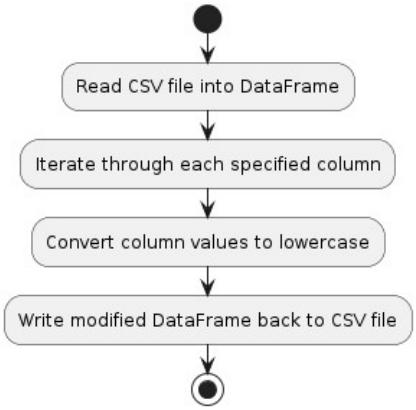


Figure 23 lowercase flowchart

```
start

:Read CSV file into DataFrame;
:Iterate through each specified column;
:Convert column values to lowercase;
:Write modified DataFrame back to CSV file;

stop
```

Figure 22 lowercase pseudocode

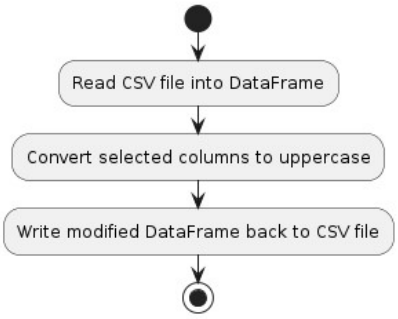


Figure 21 uppercase flowchart

```
start

:Read CSV file into DataFrame;
:Convert selected columns to uppercase;
:Write modified DataFrame back to CSV file;

stop
```

Figure 20 uppercase pseudocode

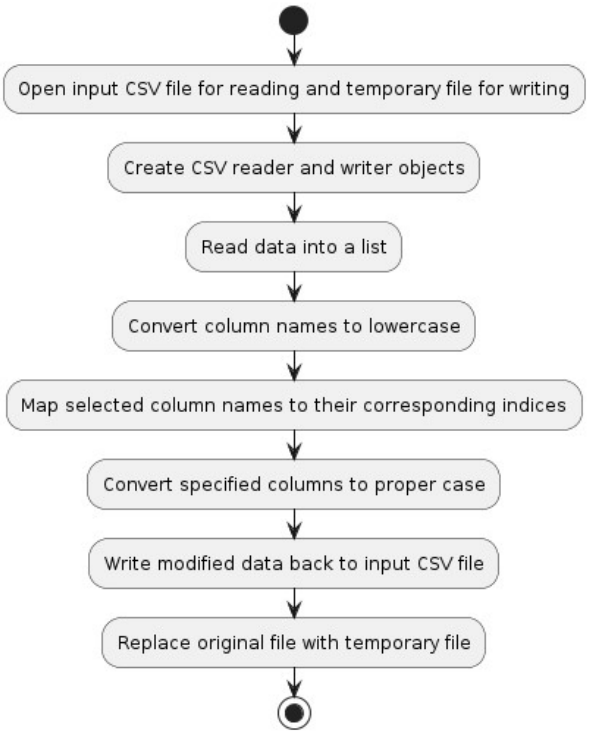


Figure 27 propercase flowchart

```
start

:Open input CSV file for reading and temporary file for writing;
:Create CSV reader and writer objects;
:Read data into a list;
:Convert column names to lowercase;
:Map selected column names to their corresponding indices;
:Convert specified columns to proper case;
:Write modified data back to input CSV file;
:Replace original file with temporary file;

stop
```

Figure 26 propercase pseudocode

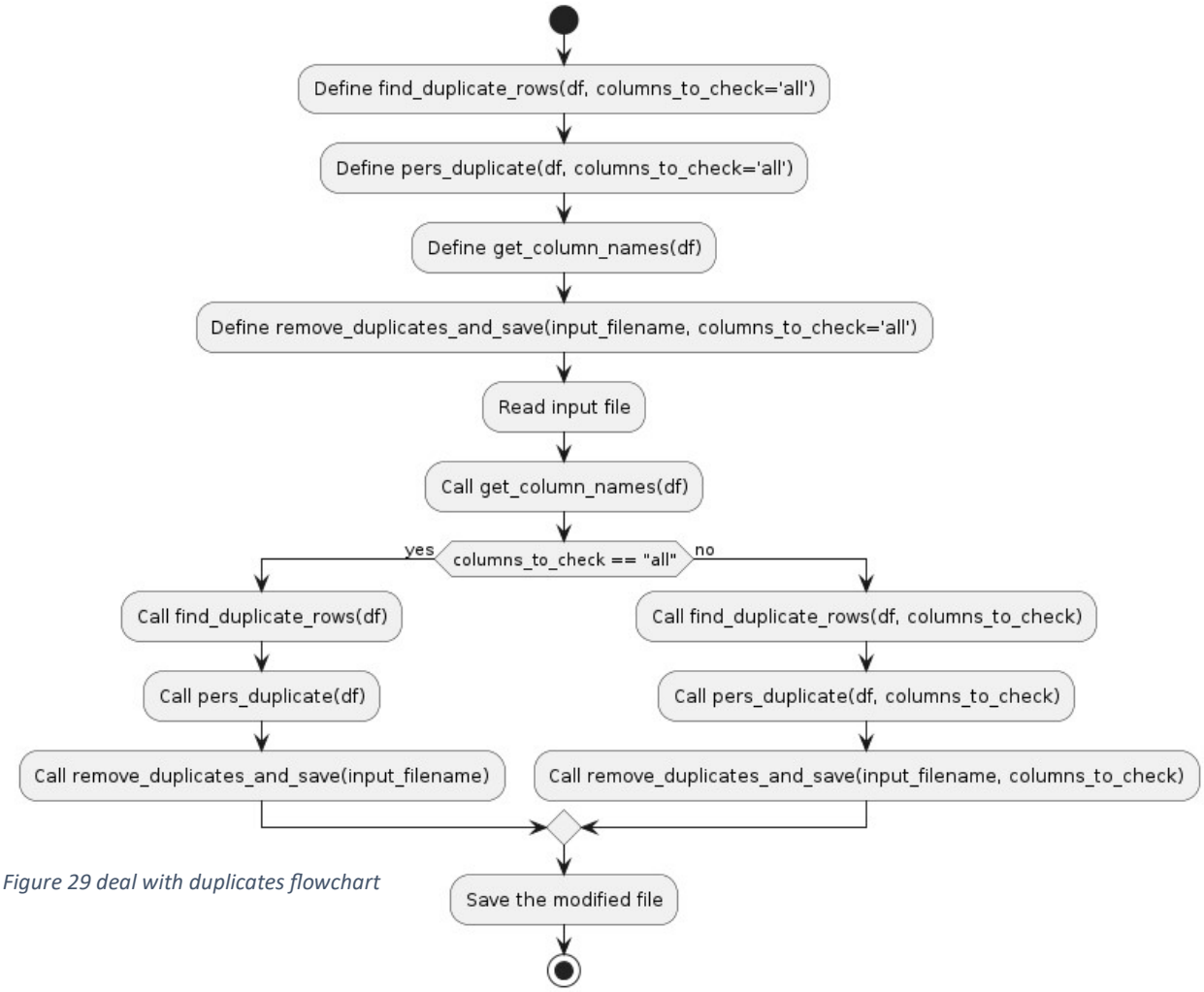


Figure 29 deal with duplicates flowchart

```
start

:Define find_duplicate_rows(df, columns_to_check='all');
:Define pers_duplicate(df, columns_to_check='all');
:Define get_column_names(df);
:Define remove_duplicates_and_save(input_filename, columns_to_check='all');

:Read input file;

:Call get_column_names(df);
if (columns_to_check == "all") then (yes)
  :Call find_duplicate_rows(df);
  :Call pers_duplicate(df);
  :Call remove_duplicates_and_save(input_filename);
else (no)
  :Call find_duplicate_rows(df, columns_to_check);
  :Call pers_duplicate(df, columns_to_check);
  :Call remove_duplicates_and_save(input_filename, columns_to_check);
endif

:Save the modified file;

stop
```

Figure 28 deal with duplicates pseudocode

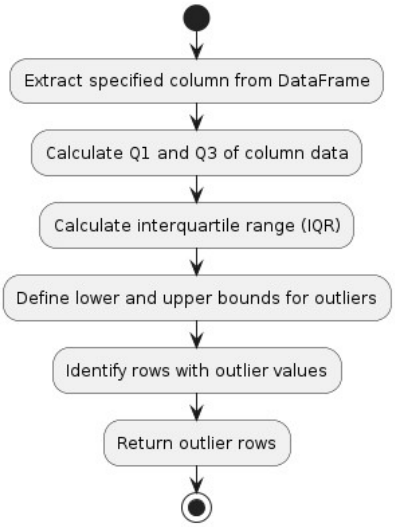


Figure 32 find outliers flowchart

```
start

:Extract specified column from DataFrame;
:Calculate Q1 and Q3 of column data;
:Calculate interquartile range (IQR);
:Define lower and upper bounds for outliers;
:Identify rows with outlier values;
:Return outlier rows;

stop
```

Figure 33 find outliers pseudocode

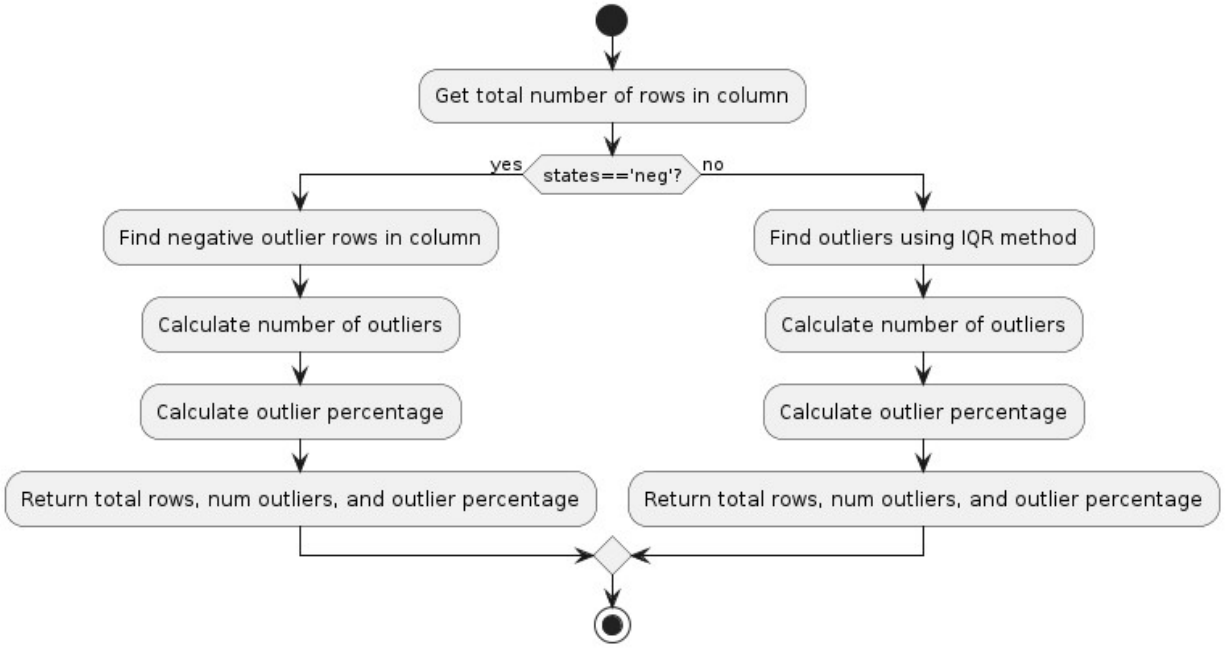


Figure 31 calculate outliers flowchart

```
start

:Get total number of rows in column;
if (states=='neg'?) then (yes)
  :Find negative outlier rows in column;
  :Calculate number of outliers;
  :Calculate outlier percentage;
  :Return total rows, num outliers, and outlier percentage;
else (no)
  :Find outliers using IQR method;
  :Calculate number of outliers;
  :Calculate outlier percentage;
  :Return total rows, num outliers, and outlier percentage;
endif

stop
```

Figure 30 calculate outlier pseudocode

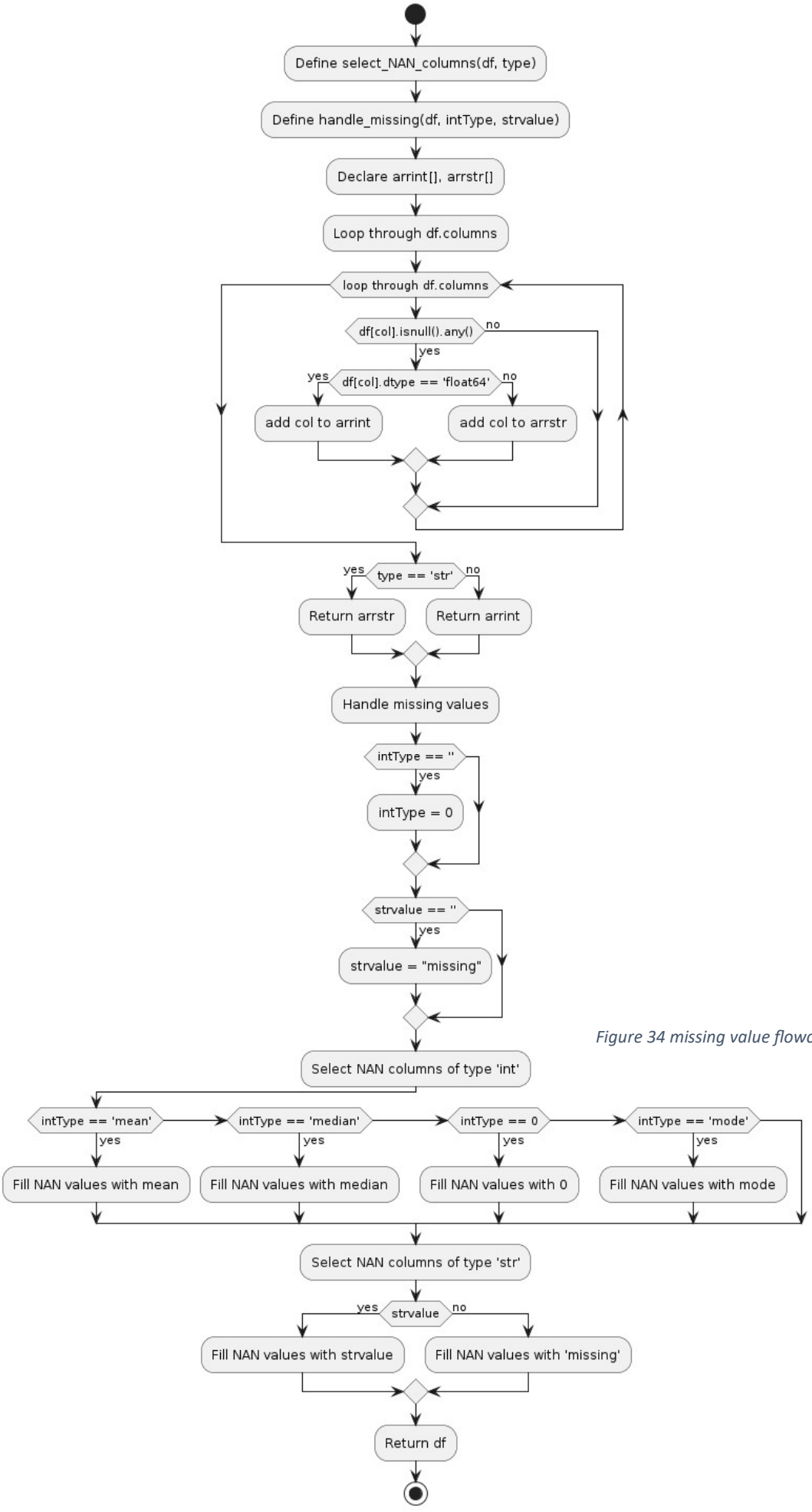


Figure 34 missing value flowchart

```
start
:Define select_NAN_columns(df, type);
:Define handle_missing(df, intType, strvalue);
:Declare arrint[], arrstr[];
:Loop through df.columns;
while (loop through df.columns)
if (df[col].isnull().any()) then (yes)
if (df[col].dtype == 'float64') then (yes)
:add col to arrint;
else (no)
:add col to arrstr;
endif
else (no)
endif
endwhile
if (type == 'str') then (yes)
:Return arrstr;
else (no)
:Return arrint;
endif
:Handle missing values;
if (intType == '') then (yes)
:intType = 0;
endif
if (strvalue == '') then (yes)
:strvalue = "missing";
endif
:Select NAN columns of type 'int';
if (intType == 'mean') then (yes)
:Fill NAN values with mean;
elseif (intType == 'median') then (yes)
:Fill NAN values with median;
elseif (intType == 0) then (yes)
:Fill NAN values with 0;
elseif (intType == 'mode') then (yes)
:Fill NAN values with mode;
endif
:Select NAN columns of type 'str';
if (strvalue) then (yes)
:Fill NAN values with strvalue;
else (no)
:Fill NAN values with 'missing';
endif
:Return df;
stop
```

Figure 35 missing value pseudocode

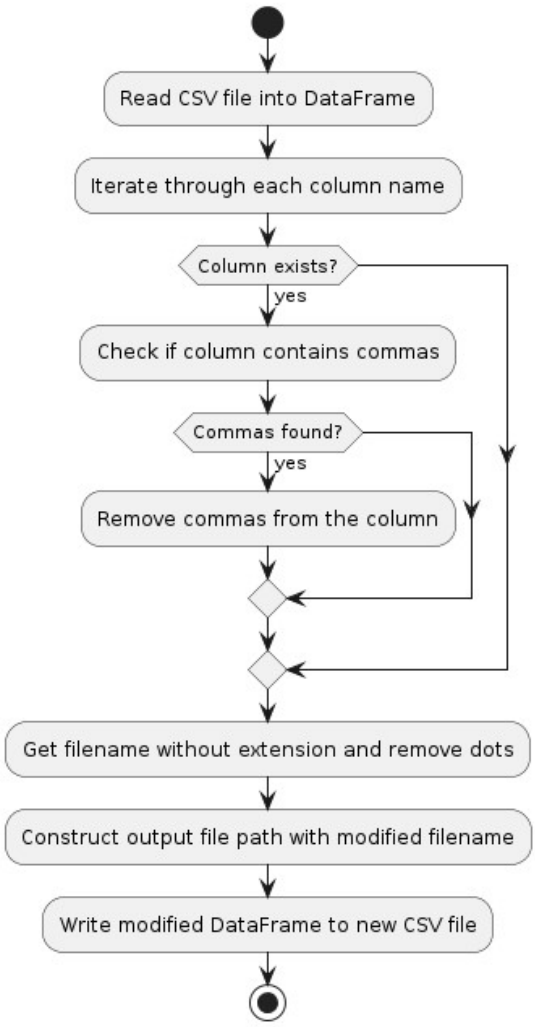


Figure 37 remove commas flowchart

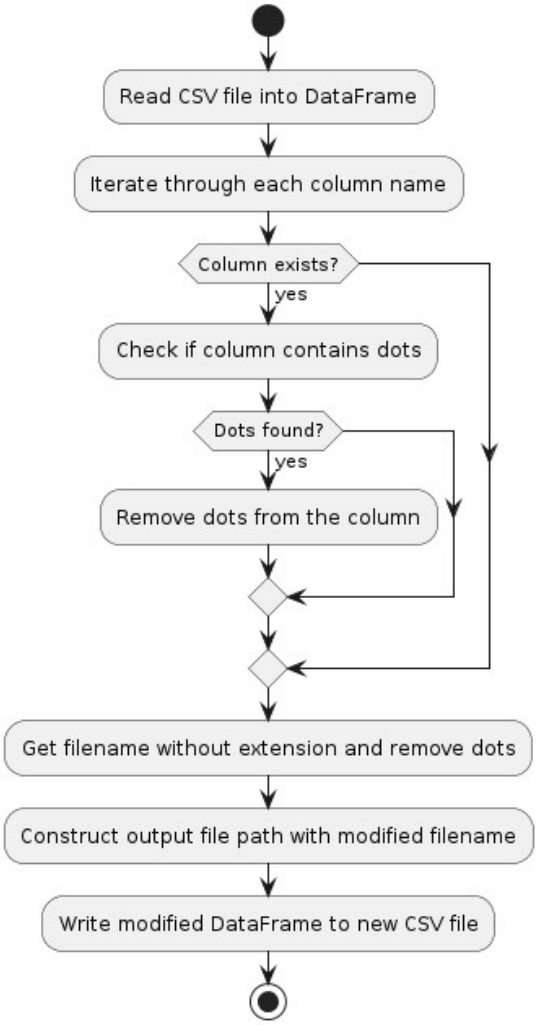


Figure 36 remove dots flowchart

```
start

:Read CSV file into DataFrame;
:Iterate through each column name;
if (Column exists?) then (yes)
  :Check if column contains commas;
  if (Commas found?) then (yes)
    :Remove commas from the column;
  endif
endif
:Get filename without extension and remove dots;
:Construct output file path with modified filename;
:Write modified DataFrame to new CSV file;

stop
```

Figure 39 remove commas pseudocode

```
start

:Read CSV file into DataFrame;
:Iterate through each column name;
if (Column exists?) then (yes)
  :Check if column contains dots;
  if (Dots found?) then (yes)
    :Remove dots from the column;
  endif
endif
:Get filename without extension and remove dots;
:Construct output file path with modified filename;
:Write modified DataFrame to new CSV file;

stop
```

Figure 38 remove dots pseudocode

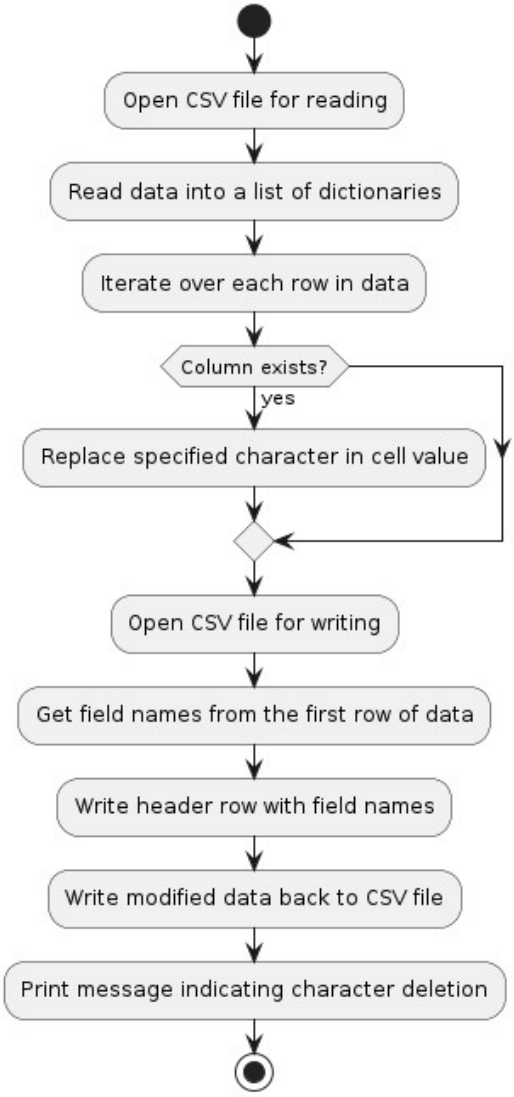


Figure 40 remove letters flowchart

```
start

:Open CSV file for reading;
:Read data into a list of dictionaries;
:Iterate over each row in data;
if (Column exists?) then (yes)
  :Replace specified character in cell value;
endif
:Open CSV file for writing;
:Get field names from the first row of data;
:Write header row with field names;
:Write modified data back to CSV file;
:Print message indicating character deletion;

stop
```

Figure 43 remove letters pseudocode

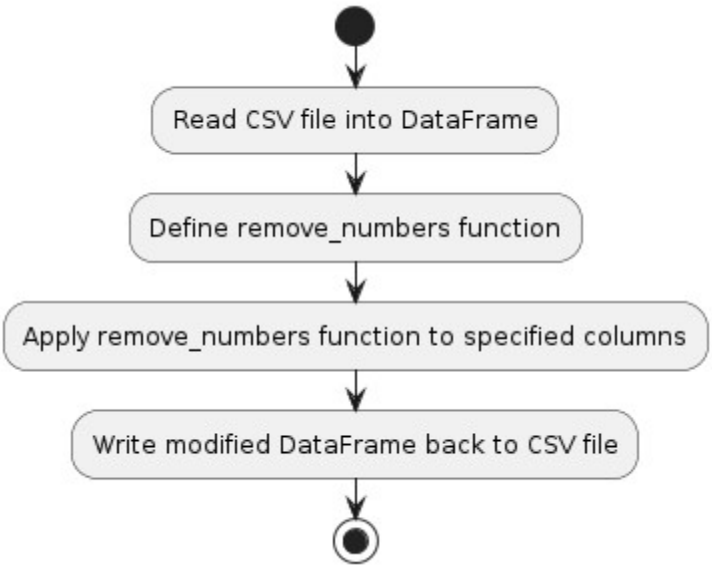


Figure 41 remove numbers flowchart

```
start

:Read CSV file into DataFrame;
:Define remove_numbers function;
:Apply remove_numbers function to specified columns;
:Write modified DataFrame back to CSV file;

stop
```

Figure 42 remove numbers pseudocode

5.3 Block diagram

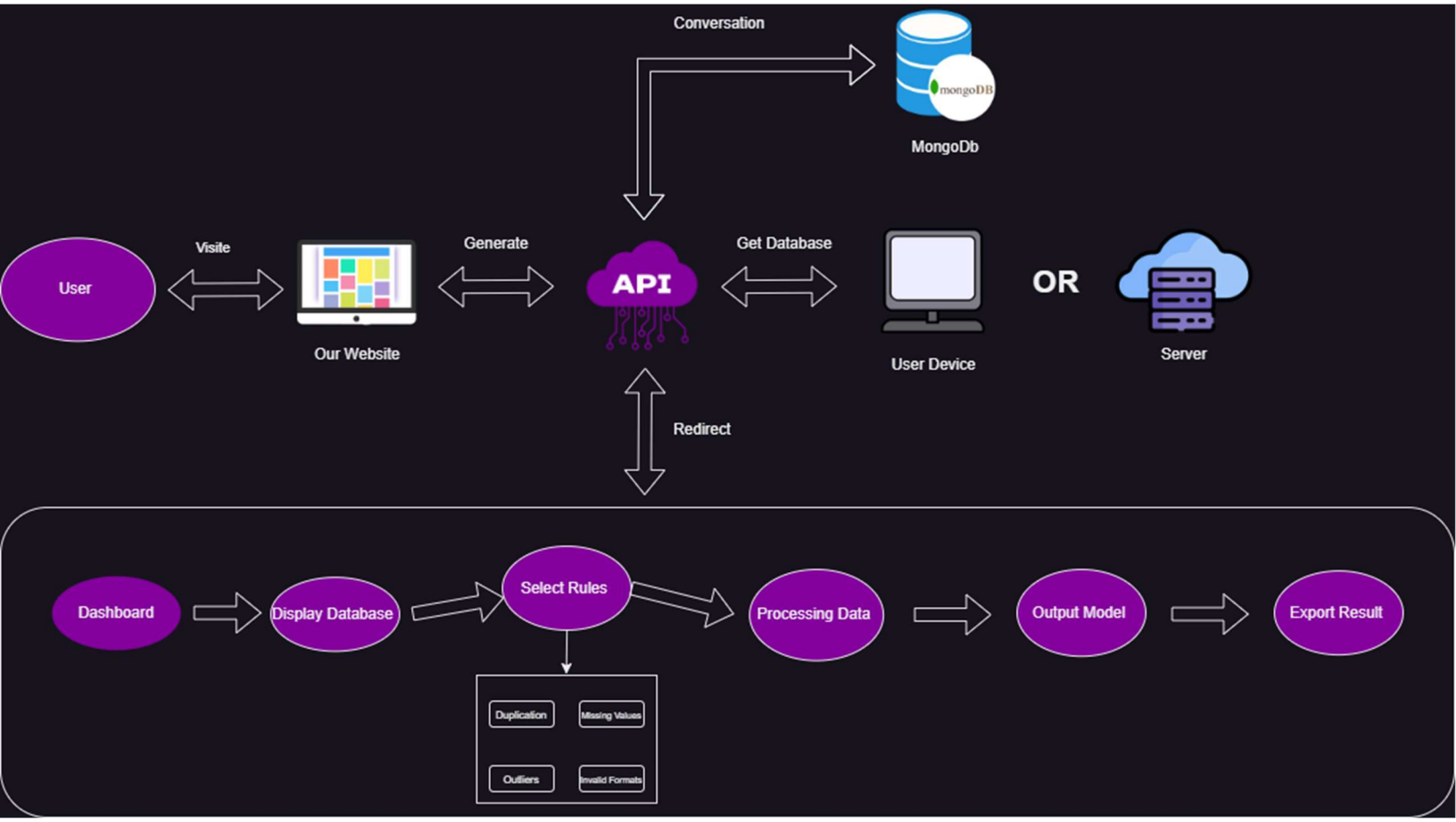


Figure 44 Block Diagram

5.4 Figma designs

At this part, prototypes of the system interface are created to simulate the initial design of the system. :-

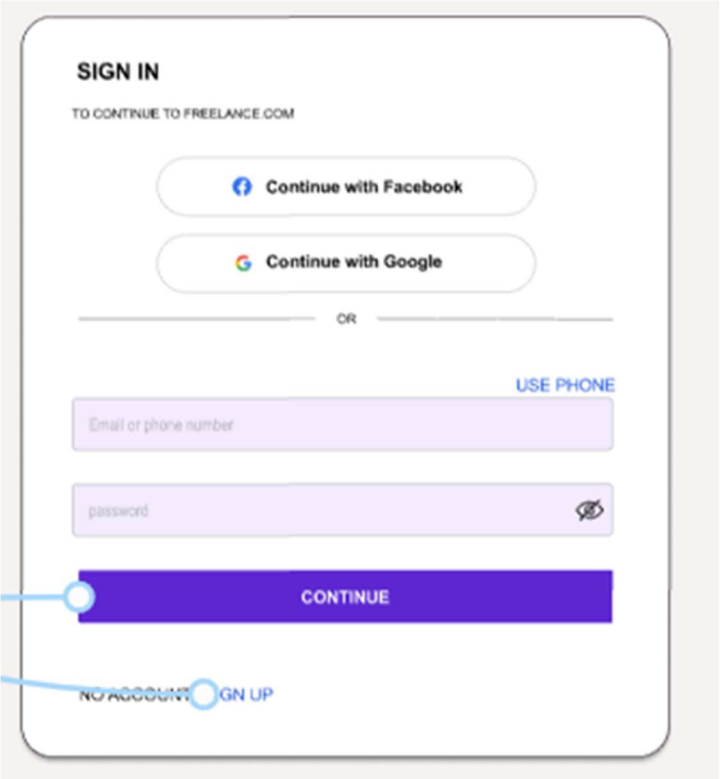


Figure 46 Sign in page

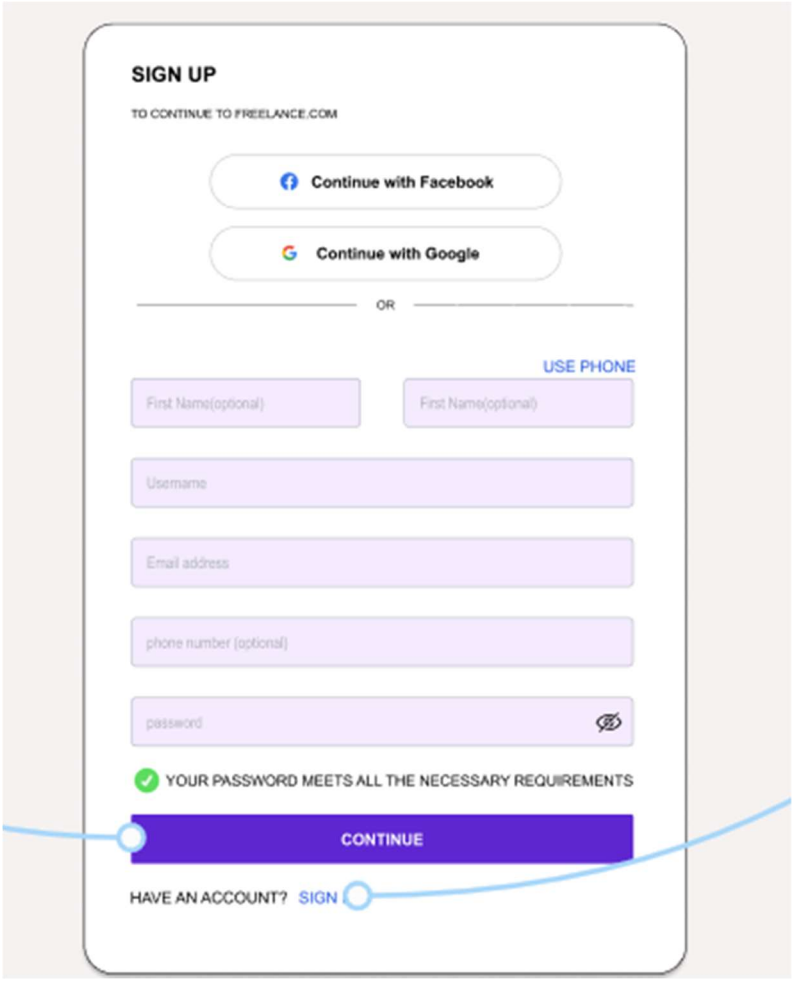


Figure 45 Sign up page

Figure 48 verify email

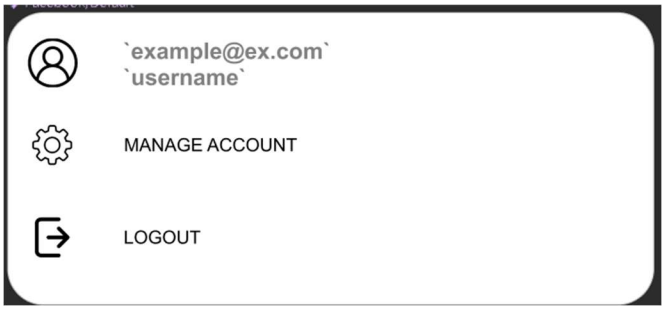


Figure 47 profile popup

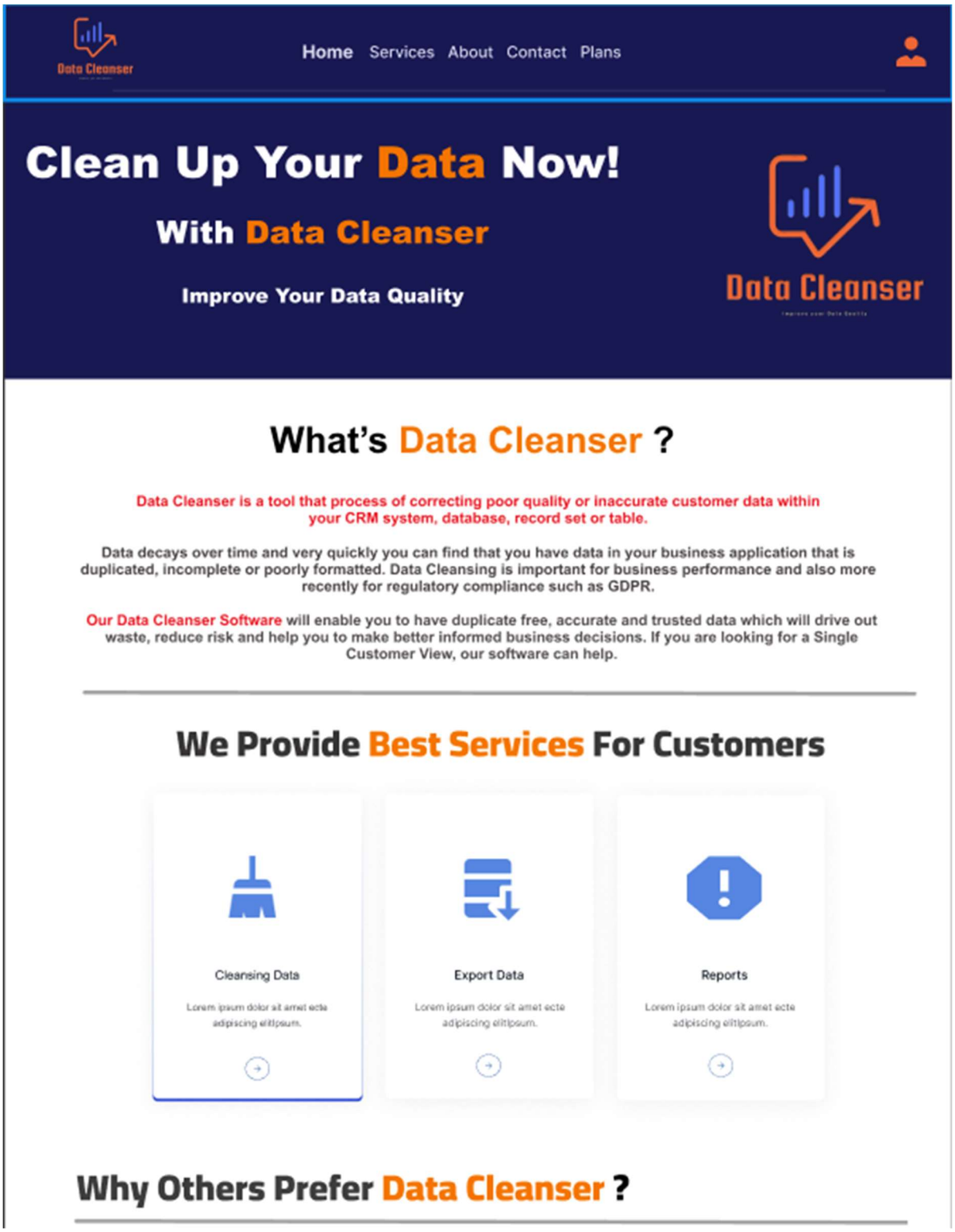


Figure 49 Homepage

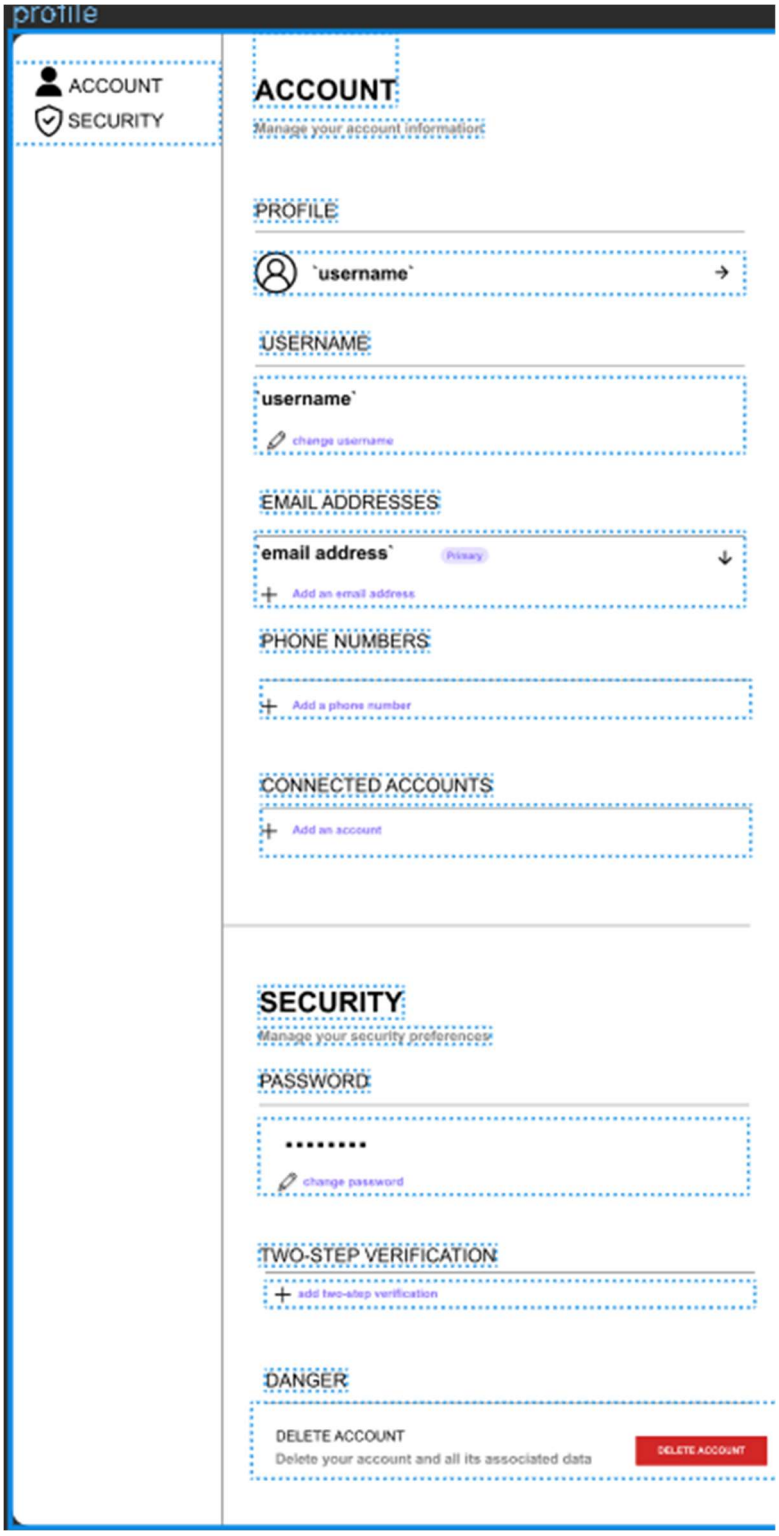


Figure 50 user account slider

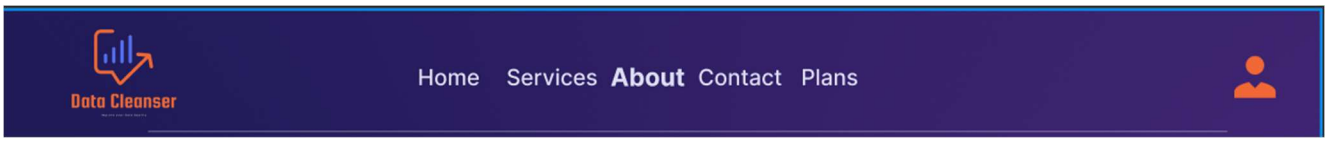


Figure 52 header

Figure 51 Footer

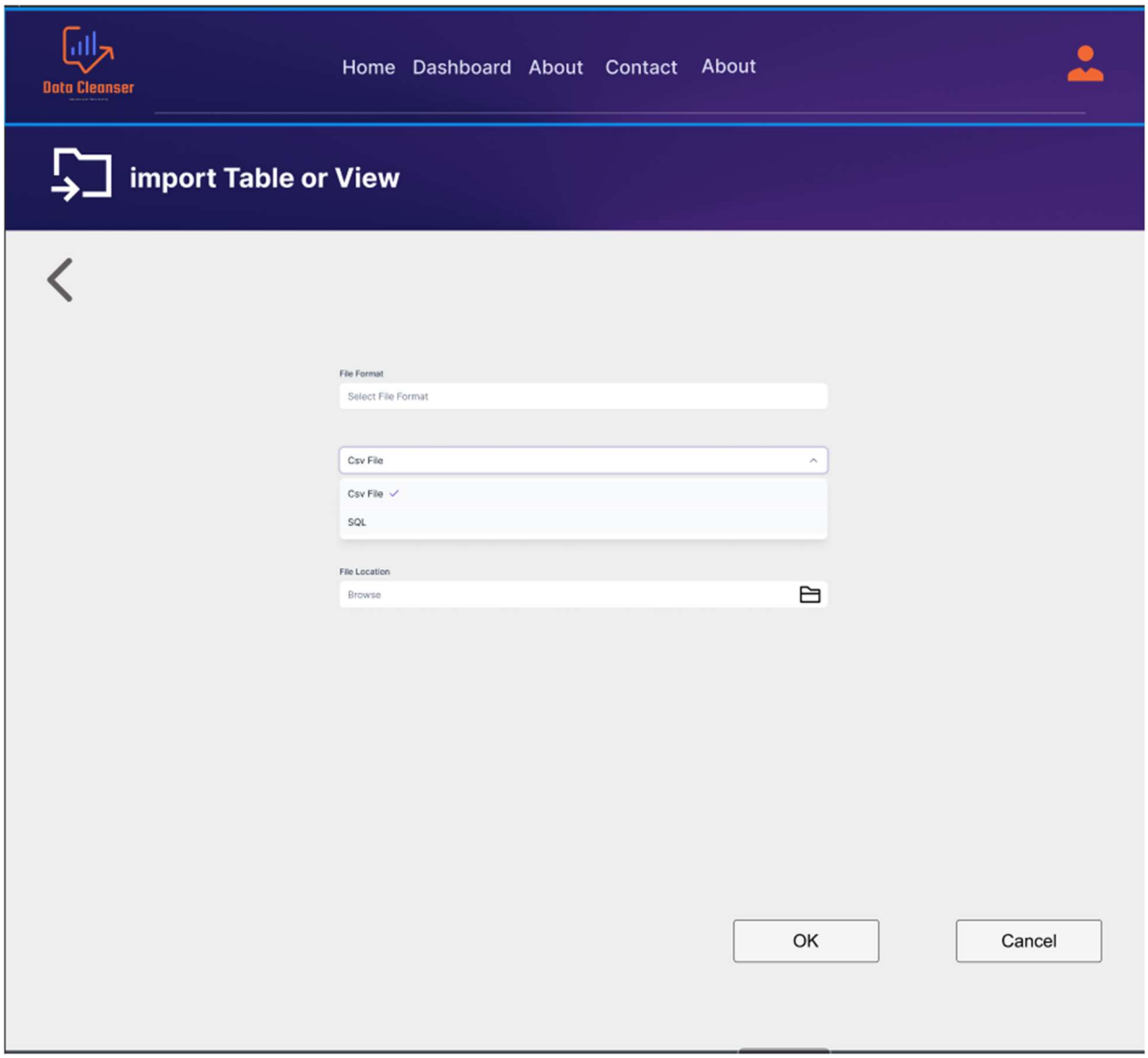


Figure 53 Import page

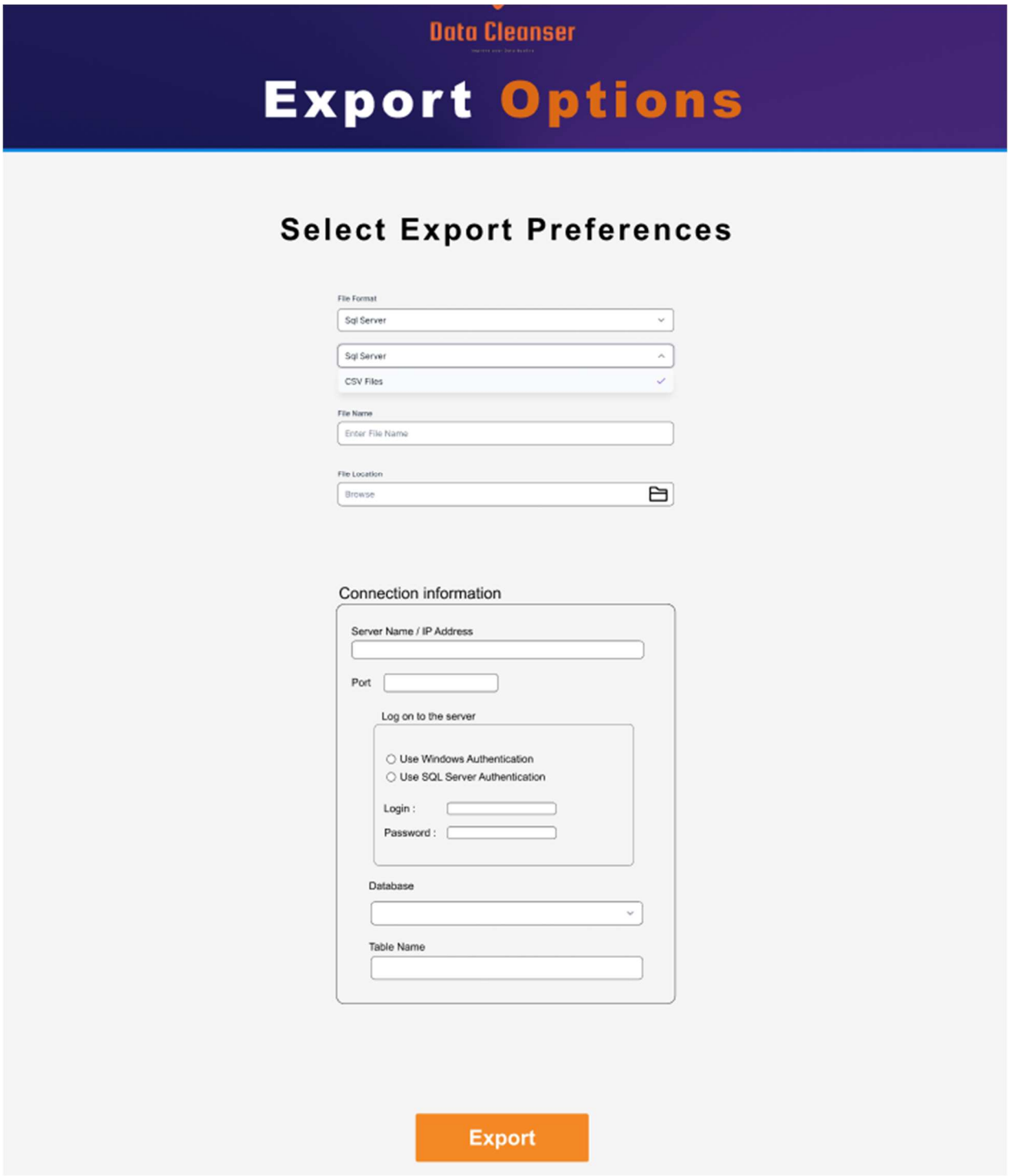


Figure 54 export page

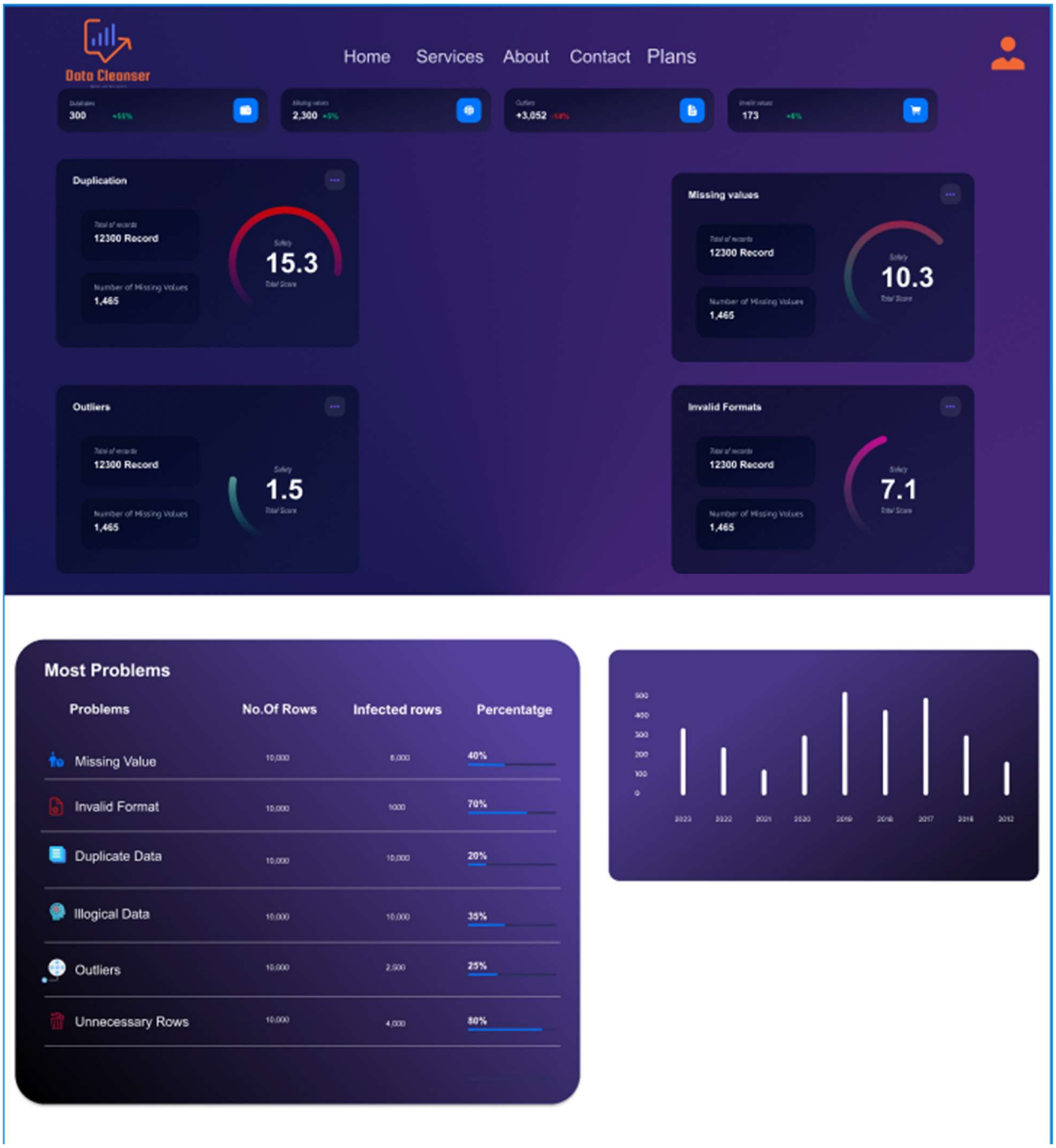


Figure 55 Dashboard

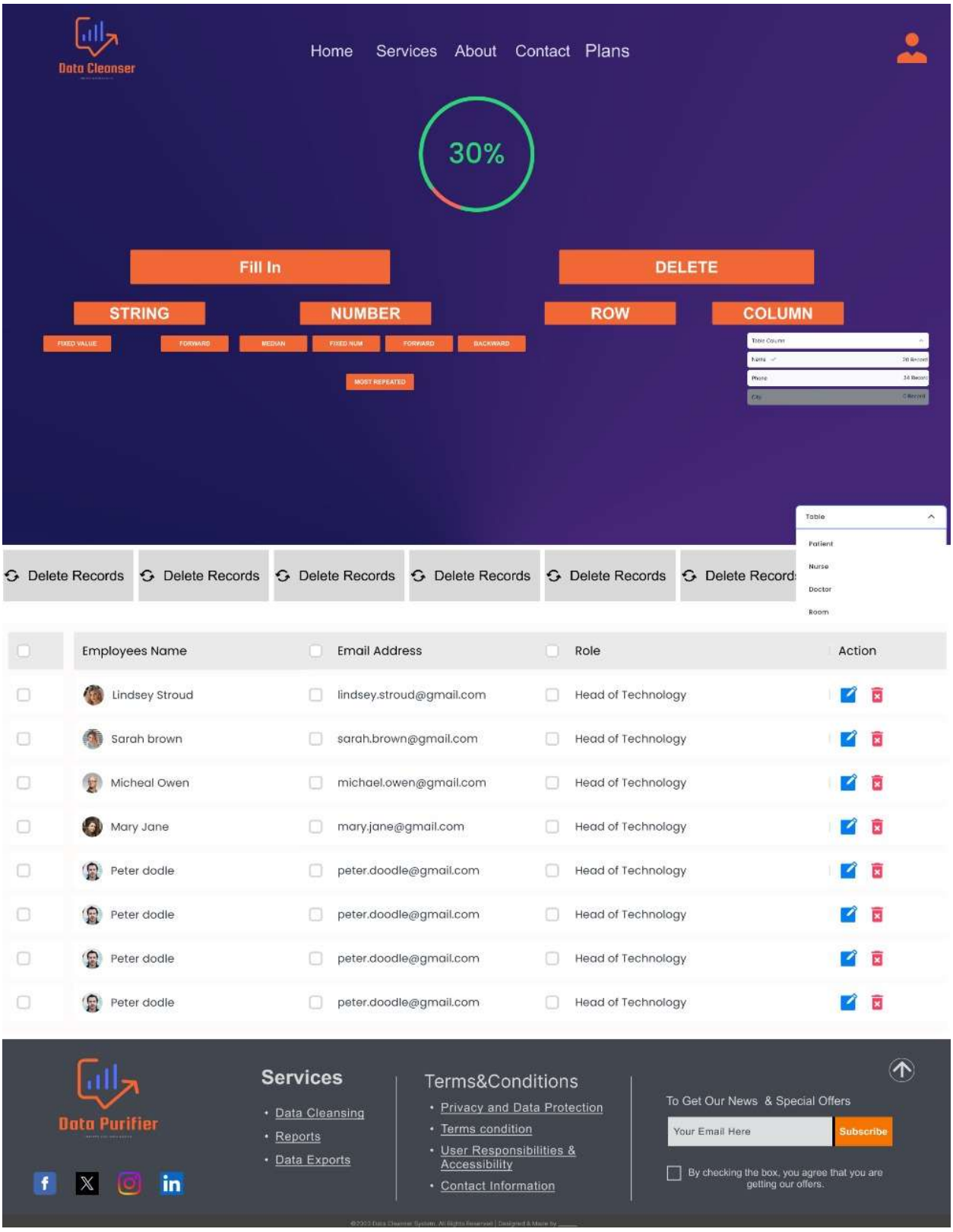


Figure 56 Missing values page

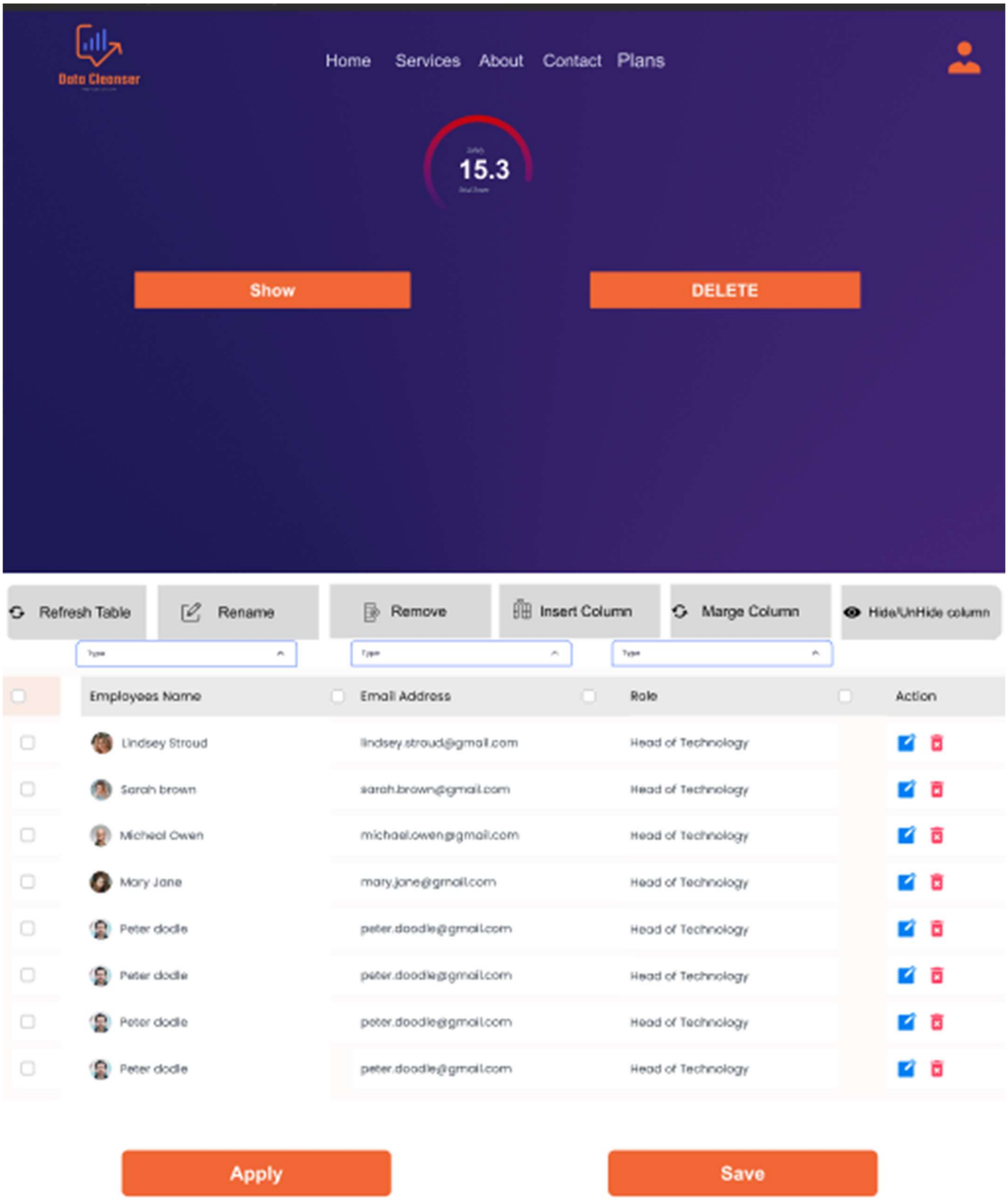


Figure 57 Handling duplicates page

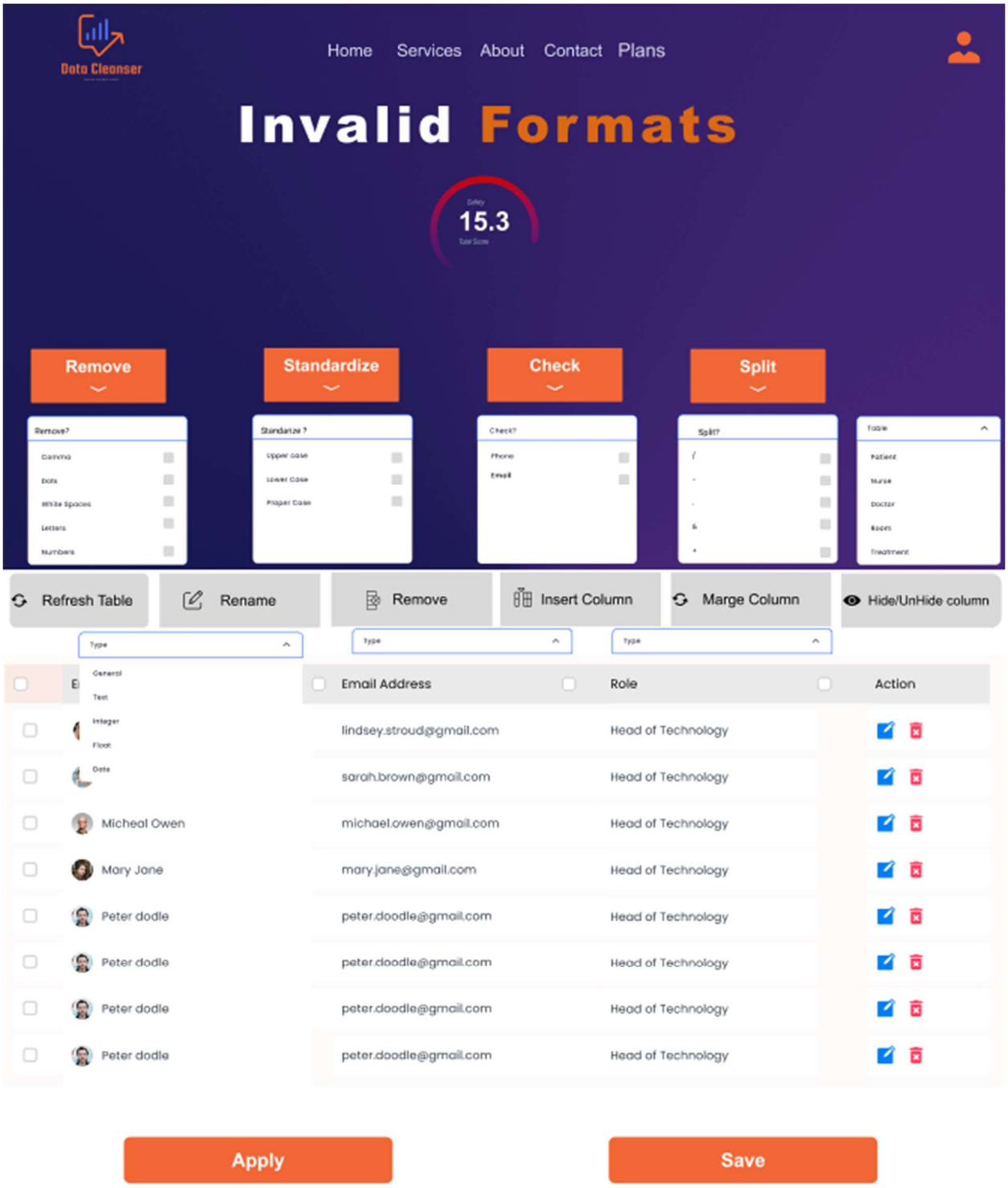
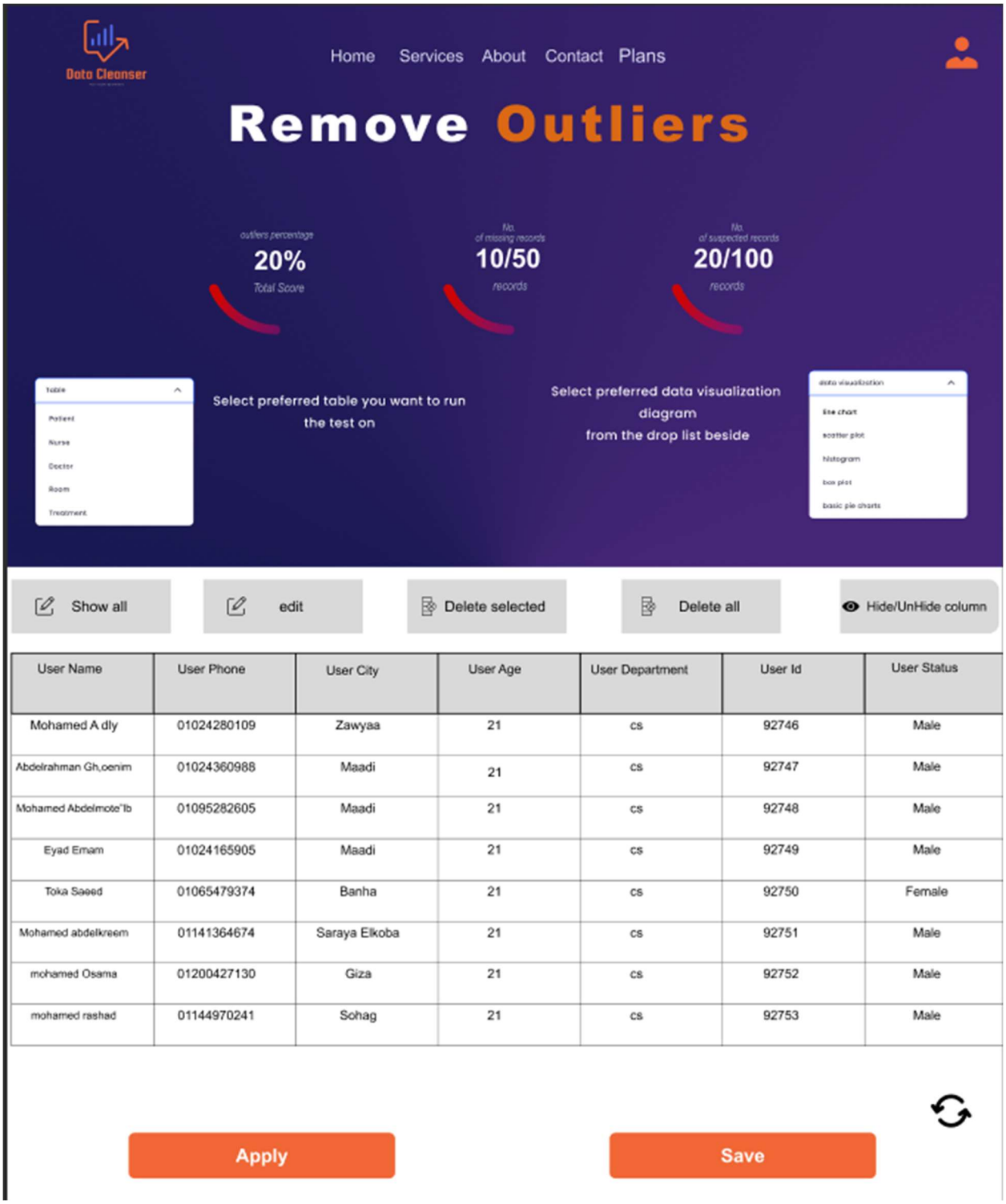


Figure 58 invalid formats page



User Name	User Phone	User City	User Age	User Department	User Id	User Status
Mohamed A dly	01024280109	Zawyaa	21	cs	92746	Male
Abdelrahman Gh,oenim	01024360988	Maadi	21	cs	92747	Male
Mohamed Abdelmote'lb	01095282605	Maadi	21	cs	92748	Male
Eyad Emam	01024165905	Maadi	21	cs	92749	Male
Toka Saeed	01065479374	Banha	21	cs	92750	Female
Mohamed abdelkreem	01141364674	Saraya Elkoba	21	cs	92751	Male
mohamed Osama	01200427130	Giza	21	cs	92752	Male
mohamed rashad	01144970241	Sohag	21	cs	92753	Male

Figure 59 Remove outliers page

Chapter SIX
System Implementation

Chapter 6– system implementation

6.1 Database implementation

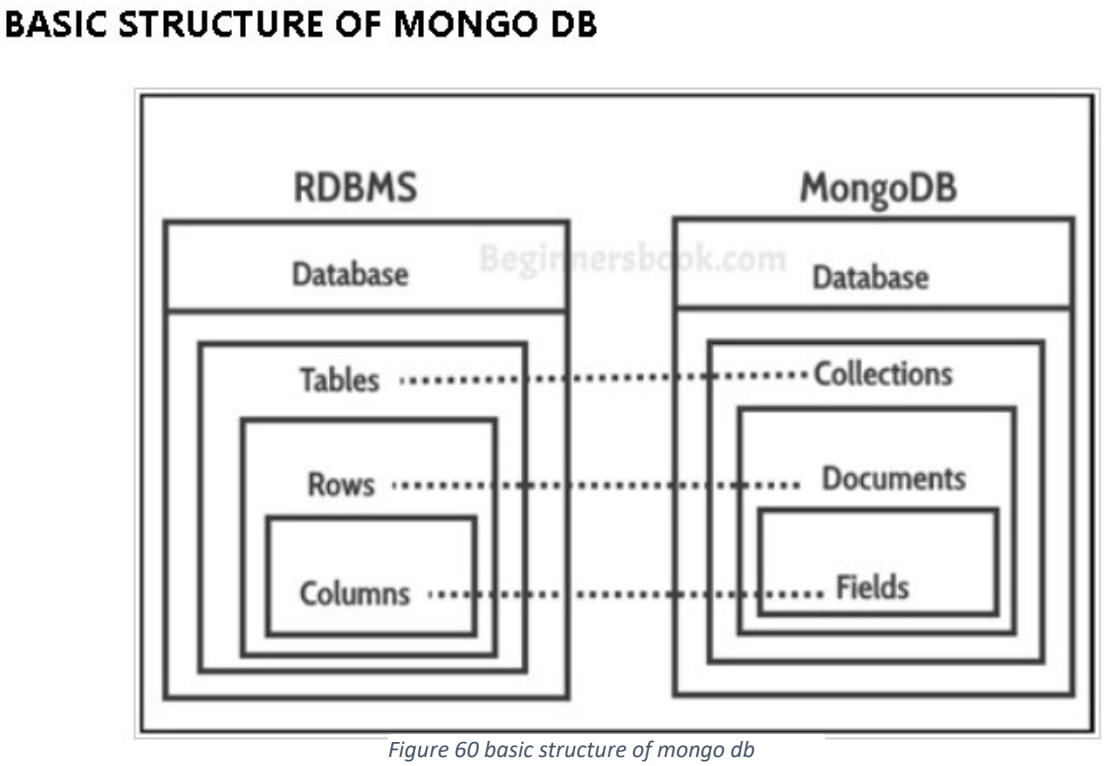


Figure 60 basic structure of mongo db

We chose mongo compass to implement our database as a non relational one in order to achieve the highest retrieval speed and quality

We wanted to create a containerized database in order to achieve the concept of **lightweight database on a virtual machine**

This means that you can run and retrieve fast your database on all types of machines without making assumptions for the operating system working right now or the database version your working with

So how do these containers actually look like?

Database server container is defined by elements called Dockerfiles, which basically define a sequence of steps required to build it. Dockerfiles begin with a "base image," followed by databases and SQL scripts. It's also possible to use snapshots and database clones when required.

Databases can be copied and run within the container file system, or mounted to the container directly by using the MOUNTDB command.

Pros:

1 – CI/CD Friendly – Database Containers are extremely consistent, which is very important while creating and maintaining an agile environment. Furthermore, these containers are proving to be very effective with approaches like Continuous Integration and Continuous Delivery (CI/CD).

2 – Multi-Cloud Compatibility – More and more companies today are leaving the on-premise format or creating hybrid pipelines to enable the seamless scaling up of their operations during peak-times. Not only can containers operate smoothly on the cloud, but they can do so on multiple cloud platforms.

The widespread popularity of the Docker image format further helps with portability. You can now use containers wherever you wish to operate.

3 – Cost Efficiency – Containers are very cost-efficient. In spite of the initial investment required for memory, CPU, and storage, it is possible to run many containers on the same infrastructure. They also integrate better with third party solutions (i.e – replication, mirroring, etc.), hence saving time and money.

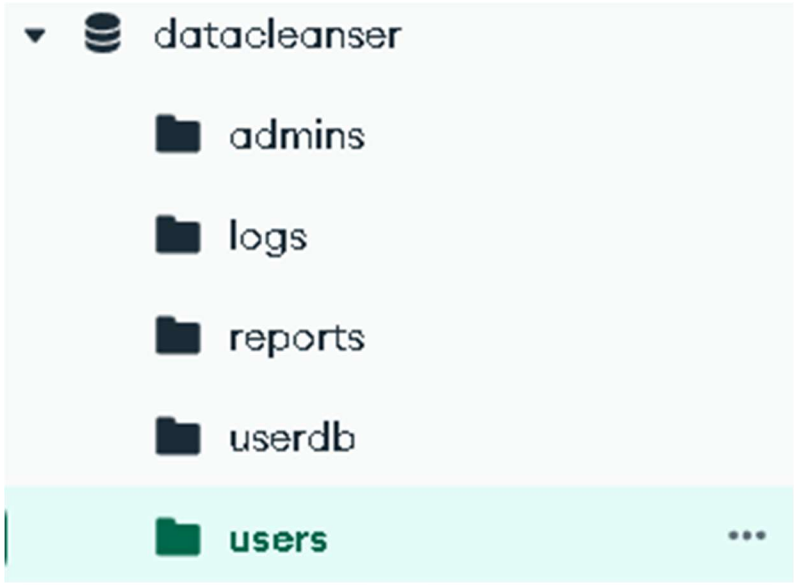
Cons:

1 – Security – Since all of the database containers usually share a common host, it becomes potentially easy to penetrate to the system due to the lack of isolation. Traditional security methods can be helpful in detecting such breaches, but this is by no means an air-tight methodology, something that needs to be clear.

2 – Lack of Isolation – Another issue with a container is the lack of OS flexibility which means containers must be of the same OS as of the Base OS. It can also become challenging to monitor database activity when hundreds (and potentially thousands) of containers are running on the same server.

3 – New Technology – As mentioned earlier, the use of Database Containers is a new practice and the methodology has not yet matured. There are still many aspects that need to be developed and polished before database containerization becomes a must-have component in every ecosystem.

Database Collection Implementation:



admins

Storage size: 20.48 kB
Documents: 1
Avg. document size: 97.00 B
Indexes: 1
Total index size: 20.48 kB

logs

Storage size: 20.48 kB
Documents: 1
Avg. document size: 94.00 B
Indexes: 1
Total index size: 20.48 kB

reports

Storage size: 20.48 kB
Documents: 1
Avg. document size: 1.15 kB
Indexes: 1
Total index size: 20.48 kB

userdb

Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 1
Total index size: 4.10 kB

users

Storage size: 20.48 kB
Documents: 1
Avg. document size: 97.00 B
Indexes: 1
Total index size: 20.48 kB

1-User db

- **_id**: This field represents the unique identifier for the document in Integer data type.
- **Username**: This field stores the username of the user in Integer data type.
- **Email**: This field stores the email address of the user in varchar data type.
- **Name**: This field stores the name of the user in varchar.
- **Password**: This field stores the hashed password of the user for authentication purposes in varchar data type.

2- Log db

- **_id**: This field represents the unique identifier for the document in Integer data type.
- **Username**: The username of the user associated with the logged activity.
- **Database_Name**: The name of the database on which the operation was performed.
- **Log_Entries**: An array of objects representing individual log entries, capturing details such as the operation performed and the timestamp.

3- Reports db


```
_id: ObjectId('6677713b4e24d291c9cdfac6')
Timestamp : Timestamp({ t: 0, i: 0 })
User : "JohnDoe"
Cleansing_Techniques_Used : Array (empty)
Summary_Statistics : Object
  Negative_Numbers_Found : 0
  Outliers : Object
    Outliers_Found : 0
    Outliers_Removed : 0
    percentage_before_out : 0
    percentage_after_out : 0
  Missing_Values : Object
    String : 0
    Integer : 0
    Missing_Values_Found : 0
    Missing_Values_Removed : 0
    percentage_before_ms : 0
    percentage_after_ms : 0
  Duplicates : Object
    Duplicates_Found : 0
    Duplicates_Removed : 0
    percentage_before_dup : 0
    percentage_after_dup : 0
  Invalid Formats : Object
    Email : 0
    Phone : 0
    Lower_Cases : 0
    Upper_Cases : 0
    Proper_Cases : 0
    Merge : 0
    Split : 0
    Proper_Case : 0
    Remove_Dots : 0
    Remove_Letters : 0
    Remove_Numbers : 0
    Remove_Selected_Value_From_Column : 0
    Remove_Double_Spaces : 0
    Strip : 0
    Invalid_formats_found : 0
    Invalid_formats_removed : 0
    percentage_before_Inv : 0
    percentage_after_Inv : 0
  Statistical_Measures : Object
    Mean : 0
    Median : 0
    Mode : 0
    Standard_Deviation : 0
    Variance : 0
    Minimum_Value : 0
    Maximum_Value : 0
user_inputs : "    "
Data_Source : Object
  File_Name : "    "
  Database_Name : "    "
  Table_name : "    "
```

_id: This field represents the unique identifier for the document in Object ID data type.

User: The username of the user associated with the report.

Timestamp: The date and time when the report was generated.

Summary_Statistics: Contains various summary statistics calculated during data cleansing, including:

Negative_Numbers_Found: Count of negative numbers found in the data.

Outliers: Count of outliers detected in the data.

Missing_Values: An object containing counts of missing values identified as string and integer types.

Duplicates: Count of duplicate records found in the data.

Invalid Formats: An object containing counts for various invalid format validations and manipulations, including email and phone number formats, conversions to lower case, upper case, proper case, merge and split operations, removal of dots, letters, numbers, selected values from columns, double spaces, and leading and trailing whitespaces.

Statistical_Measures: An object containing statistical measures calculated from the data, including mean, median, mode, standard deviation, variance, minimum value, and maximum value.

user_inputs: Additional inputs provided by the user.

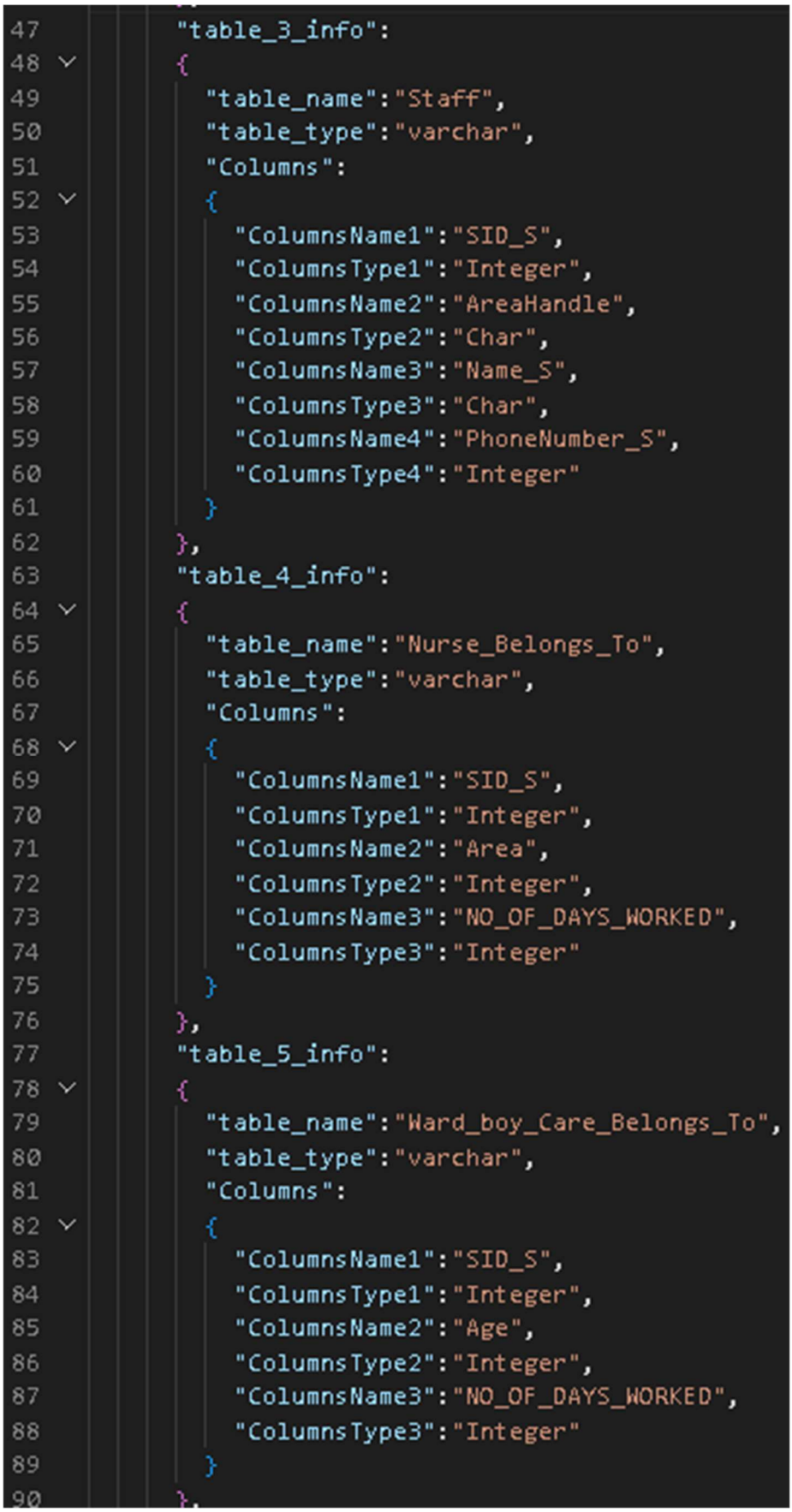
Data_Source: Information about the source of the data being analyzed, including the name of the file, database, and table containing the data.

Data_Cleansing_Status: Indicates the status of the data cleansing process.

Cleansing_Techniques_Used: An array containing the cleansing techniques applied during data processing.

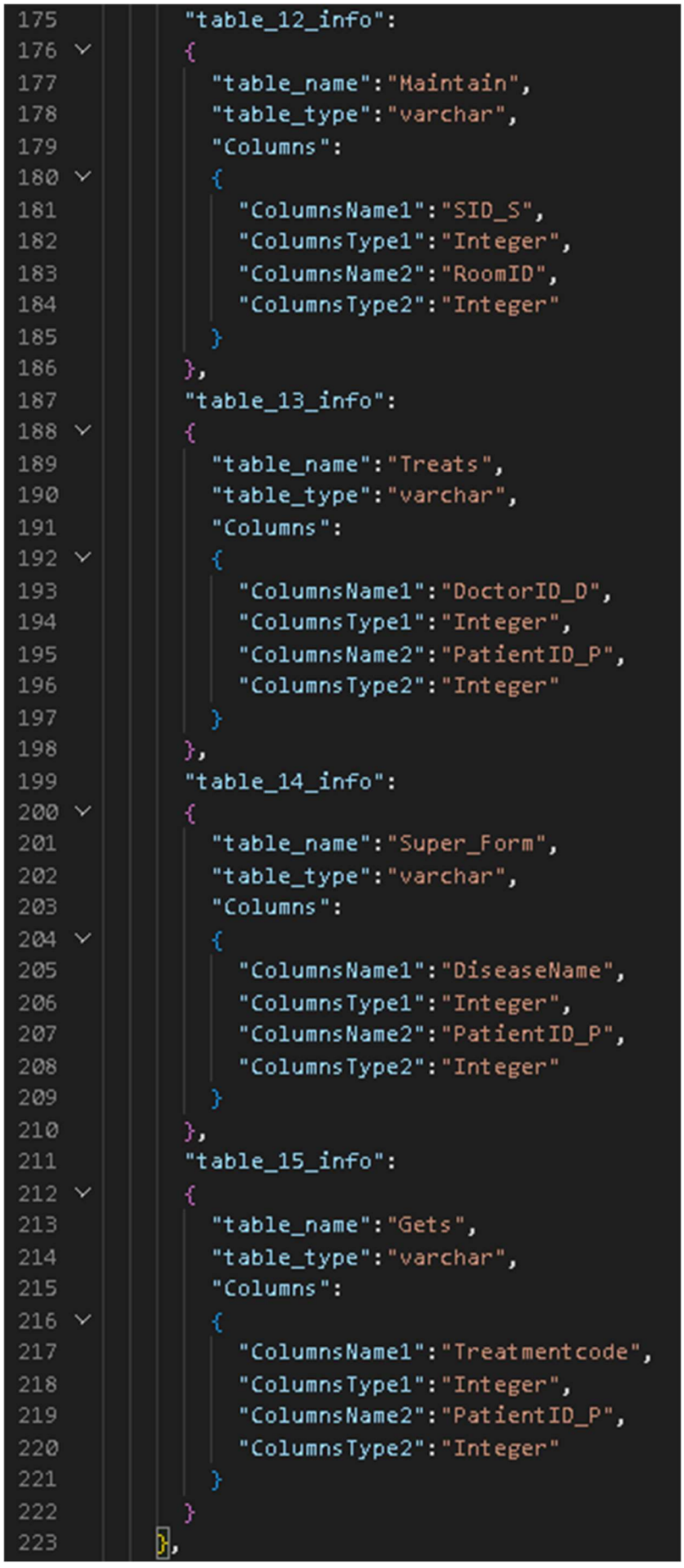
4-user db

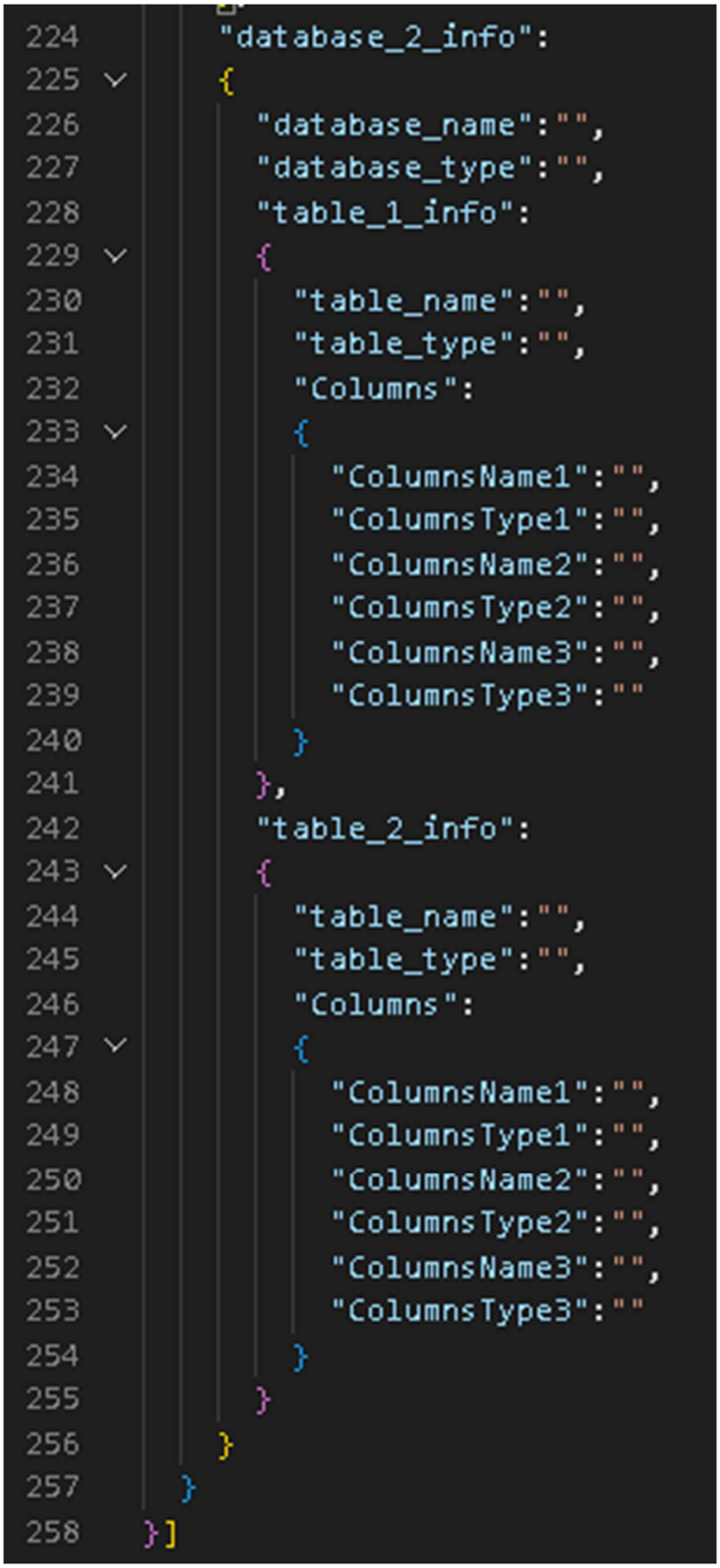

WebDb.User_db.json
Click here to ask Blackbox to help you code faster
[{
"_id":
{
"$oid": "6635198d4760878753526692"
},
"Username": "",
"Databases":
{
"database_1_info":
{
"database_name":"Hospital_Management_System",
"database_type":"SQLiteDatabase",
"table_1_info":
{
"table_name":"Patient",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"SID_P",
"ColumnsType1":"Integer",
"ColumnsName2":"PatientID_P",
"ColumnsType2":"Integer",
"ColumnsName3":"PatientName",
"ColumnsType3":"Char",
"ColumnsName4":"Gender",
"ColumnsType4":"Char",
"ColumnsName5":"Address",
"ColumnsType5":"Char",
"ColumnsName6":"PhoneNumber_P",
"ColumnsType6":"Integer"
}
},
"table_2_info":
{
"table_name":"Room",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"RoomID",
"ColumnsType1":"Integer",
"ColumnsName2":"Type_of_room",
"ColumnsType2":"Char",
"ColumnsName3":"NoOfDays",
"ColumnsType3":"Char"
}
},


"table_3_info":
{
"table_name":"Staff",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"SID_S",
"ColumnsType1":"Integer",
"ColumnsName2":"AreaHandle",
"ColumnsType2":"Char",
"ColumnsName3":"Name_S",
"ColumnsType3":"Char",
"ColumnsName4":"PhoneNumber_S",
"ColumnsType4":"Integer"
}
},
"table_4_info":
{
"table_name":"Nurse_Belongs_To",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"SID_S",
"ColumnsType1":"Integer",
"ColumnsName2":"Area",
"ColumnsType2":"Integer",
"ColumnsName3":"NO_OF_DAYS_WORKED",
"ColumnsType3":"Integer"
}
},
"table_5_info":
{
"table_name":"Ward_boy_Care_Belongs_To",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"SID_S",
"ColumnsType1":"Integer",
"ColumnsName2":"Age",
"ColumnsType2":"Integer",
"ColumnsName3":"NO_OF_DAYS_WORKED",
"ColumnsType3":"Integer"
}
},

```
    "table_6_info":
    {
      "table_name":"Treatment",
      "table_type":"varchar",
      "Columns":
      {
        "ColumnsName1":"Treatmentcode",
        "ColumnsType1":"Integer",
        "ColumnsName2":"DiseaseName_T",
        "ColumnsType2":"Char",
        "ColumnsName3":"Description_T",
        "ColumnsType3":"Char"
      }
    },
    "table_7_info":
    {
      "table_name":"Bill_Payed_By",
      "table_type":"varchar",
      "Columns":
      {
        "ColumnsName1":"PatientID_P",
        "ColumnsType1":"Integer",
        "ColumnsName2":"Bill_ID",
        "ColumnsType2":"Integer",
        "ColumnsName3":"NoOfDays",
        "ColumnsType3":"Char",
        "ColumnsName4":"fees",
        "ColumnsType4":"REAL"
      }
    },
    "table_8_info":
    {
      "table_name":"Disease",
      "table_type":"varchar",
      "Columns":
      {
        "ColumnsName1":"DiseaseName",
        "ColumnsType1":"Integer",
        "ColumnsName2":"Symptoms",
        "ColumnsType2":"Char"
      }
    },
```

```
    "table_9_info":
    {
      "table_name":"Doctor",
      "table_type":"varchar",
      "Columns":
      {
        "ColumnsName1":"DoctorID_D",
        "ColumnsType1":"Integer",
        "ColumnsName2":"Speciallization",
        "ColumnsType2":"Char",
        "ColumnsName3":"DoctorName",
        "ColumnsType3":"Char",
        "ColumnsName4":"Qualification",
        "ColumnsType4":"Char",
        "ColumnsName5":"Phone_Number",
        "ColumnsType5":"Integer"
      }
    },
    "table_10_info":
    {
      "table_name":"Assigned",
      "table_type":"varchar",
      "Columns":
      {
        "ColumnsName1":"RoomID",
        "ColumnsType1":"Integer",
        "ColumnsName2":"PatientID_P",
        "ColumnsType2":"Integer"
      }
    },
    "table_11_info":
    {
      "table_name":"Care",
      "table_type":"varchar",
      "Columns":
      {
        "ColumnsName1":"SID_S",
        "ColumnsType1":"Integer",
        "ColumnsName2":"PatientID_P",
        "ColumnsType2":"Integer"
      }
    },
```


"table_12_info":
{
"table_name":"Maintain",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"SID_S",
"ColumnsType1":"Integer",
"ColumnsName2":"RoomID",
"ColumnsType2":"Integer"
}
},
"table_13_info":
{
"table_name":"Treats",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"DoctorID_D",
"ColumnsType1":"Integer",
"ColumnsName2":"PatientID_P",
"ColumnsType2":"Integer"
}
},
"table_14_info":
{
"table_name":"Super_Form",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"DiseaseName",
"ColumnsType1":"Integer",
"ColumnsName2":"PatientID_P",
"ColumnsType2":"Integer"
}
},
"table_15_info":
{
"table_name":"Gets",
"table_type":"varchar",
"Columns":
{
"ColumnsName1":"Treatmentcode",
"ColumnsType1":"Integer",
"ColumnsName2":"PatientID_P",
"ColumnsType2":"Integer"
}
}
},


"database_2_info":
{
"database_name":"",
"database_type":"",
"table_1_info":
{
"table_name":"",
"table_type":"",
"Columns":
{
"ColumnsName1":"",
"ColumnsType1":"",
"ColumnsName2":"",
"ColumnsType2":"",
"ColumnsName3":"",
"ColumnsType3":""
}
},
"table_2_info":
{
"table_name":"",
"table_type":"",
"Columns":
{
"ColumnsName1":"",
"ColumnsType1":"",
"ColumnsName2":"",
"ColumnsType2":"",
"ColumnsName3":"",
"ColumnsType3":""
}
}
}
}]

- **_id**: This field represents the unique identifier for the document in Integer data type.
- **Username**: This field stores the username of the user to whom the database configuration belongs and it’s a foreign key to the “Users” collection in varchar data type.
- **Databases**: This field contains information about various databases associated with the user. The "User_db" collection appears to provide a mechanism for users to define and manage their own databases and associated column configurations. This kind of flexibility can be useful in applications where users have varying data storage needs or where custom database structures are required for different purposes. By storing database configurations in the "User_db" collection, the application can dynamically create, modify, and access databases and their associated columns based on user-defined settings.

6.2.1 Handling invalid formats

"Data consistency is paramount in any system or database. However, ensuring that all entries adhere to a specific format or standard can be challenging. From email addresses and phone numbers to whitespace discrepancies and case sensitivity, inconsistencies can arise in various forms. In this endeavor, we aim to categorize and tackle these invalid formats systematically, employing techniques such as merging, splitting, converting to proper case, and removing unwanted characters or values. By addressing these issues, we can enhance data integrity and optimize the functionality of our systems."

1. **Format Validation:**
    - Email
    - Phone
2. **Case Handling:**
    - Lower cases
    - Upper cases
    - Proper cases
3. **Data Manipulation:**
    - Merge
    - Split
    - Proper case
    - Remove dots
    - Remove letters
    - Remove numbers
    - Remove Selected Value From Column
    - Remove double spaces
    - Strip
4. **Data Analysis:**
    - Show Repeated in all tables
    - Show Repeated in one column

1st check emails

```python
import csv
import re
import dns.resolver
def is_valid_email(email):
    # Adjusted regular expression pattern for email validation
    email_pattern = re.compile(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$')

    # Match the email pattern
    if not email_pattern.match(email):
        return False  # Email pattern does not match

    # Extract domain from the email address
    domain = email.split('@')[1]
    try:
            # Resolve MX records for the domain
            mx_records = dns.resolver.resolve(domain, 'MX')
            return True  # Domain has valid MX records
    except dns.resolver.NoAnswer:
            print(f"No MX records found for the domain: {domain}")
            return False  # No MX records found for the domain
    except dns.resolver.NXDOMAIN:
        print(f"Domain does not exist: {domain}")
        return False  # Domain does not exist
    except dns.resolver.Timeout:
        print(f"Timeout occurred while querying DNS servers for domain: {domain}")
        return False  # Timeout occurred while querying DNS servers
    except Exception as e:
        print(f"An error occurred while validating email {email}: {e}")
        return False  # Other errors

def show_valid_emails_only(input_file, email_column_index):
    with open(input_file, 'r', newline='') as csv_in:
        reader = csv.reader(csv_in)
        header = next(reader)  # Skip the header
        print("Email,Is_Valid_Email")  # Print header for email and validation status

        for row in reader:
            email = row[email_column_index]  # Choose email address from the specified column index
            is_valid = is_valid_email(email)
            print(f"{email},{is_valid}")  # Print the email address and its validation status
# Example usage:
input_file = 'Example.csv'
chosen_column_index = 2  # Specify the column index for email addresses
show_valid_emails_only(input_file, chosen_column_index)
```

2nd check phone

```python
import csv

def check_phone_numbers_in_column(csv_file, column_index):
    invalid_numbers = []
    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the first row
        for row_number, row in enumerate(reader, start=2):
            if len(row) >= column_index:
                item = row[column_index - 1]  # Adjust index to 0-based
                if len(item) != 11 or not item.startswith(('012', '010', '011', '015')) or not item.isdigit():
                    invalid_numbers.append(f"Invalid phone number at row {row_number}, column {column_index}: {item}")
            else:
                print(f"Warning: Row {row_number} does not have enough columns.")

    return invalid_numbers

# Example usage:
column_index = 4  # Adjust this to the index of the column you want to check (1-based index)
invalid_numbers = check_phone_numbers_in_column('comma.csv', column_index)
if invalid_numbers:
    print("Invalid phone numbers:")
    for number in invalid_numbers:
        print(number)
else:
    print("All phone numbers in the specified column are valid.")

```

Rest in **Appendex chapter**…

6.2.2 Handling missing values

```
import pandas as pd
def select_NAN_columns(df,type):
            arrint=[]
            arrstr=[]
            for col in df.columns:
              if df[col].isnull().any():
                if df[col].dtype == 'float64':
                    arrint.append(col)
                else:
                    arrstr.append(col)
            if(type=='str') :
              return arrstr
            else:
              return arrint

def handle_missing(df,intType,strvalue):
            # handling missing int values
            if (intType==''):
                intType=0
            if (strvalue==''):
                strvalue="missing"

            arrint=select_NAN_columns(df,'int')
            if intType=='mean':
                for i in arrint:
                 df[i]=df[i].fillna(df[i].mean())
            elif intType=='median':
                for i in arrint:
                  df[i] = df[i].fillna(df[i].median())
            elif intType==0:
                for i in arrint:
                  df[i] = df[i].fillna(0)
            elif intType=='mode':
                for i in arrint:
                  mode=df[i].value_counts().idxmax()
                  df[i] = df[i].fillna(mode)
            # handling missing str values
            arrstr=select_NAN_columns(df,'str')
            if (strvalue):
                for i in arrstr:
                  df[i] = df[i].fillna(strvalue)
            else:
                for i in arrstr:
                 df[i] = df[i].fillna('missing')
```

6.2.3 Removing outliers

Outliers, although sometimes disruptive, are pivotal in data cleaning. They can distort results and mislead interpretations but may also reveal important insights or errors. Researchers employ various methods to manage outliers, including removal, replacement, or transformation. The choice depends on the data's nature, analysis goals, and domain knowledge. Transparent reporting of outlier handling methods is crucial for research integrity. Ultimately, understanding and appropriately managing outliers enhance the reliability and validity of data-driven insights.

Detecting and handling outliers in a dataset is crucial for several reasons:

1. **Data Quality**: Outliers can skew statistical analyses and machine learning models, leading to incorrect conclusions or predictions. Removing or adjusting outliers helps maintain the quality and integrity of the dataset, ensuring more accurate results.
2. **Model Performance**: Outliers can influence the parameters of predictive models, reducing their effectiveness. By removing or replacing outliers, models can better capture the underlying patterns in the data and make more reliable predictions.
3. **Assumption Violation**: Many statistical methods and machine learning algorithms assume that the data is normally distributed or follows a specific distribution. Outliers can violate these assumptions, leading to biased estimates or incorrect inferences. Handling outliers helps ensure that the assumptions of the chosen analysis method are met.
4. **Data Interpretation**: Outliers can sometimes represent genuine but rare events or errors in data collection. Understanding the nature of outliers is essential for accurately interpreting the data and drawing meaningful conclusions. Handling outliers appropriately allows analysts to differentiate between anomalous observations and true insights.
5. **Data Visualization**: Outliers can distort visualizations such as scatter plots, histograms, and box plots, making it challenging to visualize the underlying patterns in the data. Removing or adjusting outliers improves the interpretability of visualizations and facilitates better insights into the data.

Methods for handling outliers include:

- **Removing**: Simply removing outliers from the dataset can be appropriate if they are likely to be errors or do not represent meaningful information. However, this approach should be used cautiously, as it can lead to loss of valuable data and potentially bias the analysis.
- **Replacing**: Outliers can be replaced with more typical values, such as the mean, median, or mode of the dataset. This approach can help mitigate the impact of outliers without removing them entirely. However, it may also introduce some distortion to the data if the outliers are indicative of genuine variability.

Ultimately, the choice of how to handle outliers depends on the specific characteristics of the dataset, the objectives of the analysis, and the domain knowledge of the analyst. It's essential to carefully consider the implications of different outlier handling strategies and choose the approach that best suits the particular context and goals of the analysis.

1. Detecting outliers in a numerical dataset using the Interquartile Range (IQR) method and Plot them:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def find_outliers_iqr(column_data):
    """
    Detect outliers in a numerical dataset using the Interquartile Range (IQR) method.

    Args:
    - column_data: A pandas Series representing a column of numerical values.

    Returns:
    - outliers: A list of tuples containing outlier values found in the column and their
indices.
    """
    # Calculate the first quartile (Q1) and third quartile (Q3) of the column data
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)

    # Calculate the interquartile range (IQR)
    iqr = q3 - q1

    # Define the lower and upper bounds for outlier detection
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Identify outliers and their indices
    outliers = [(value, idx) for idx, value in enumerate(column_data) if value < lower_bound
or value > upper_bound]

    return outliers

# Load the CSV file into a pandas DataFrame
df = pd.read_csv("example11.csv")

# Choose the column index you want to perform outlier detection on
column_index = 2  # Change this to the desired column index

# Extract the specified column from the DataFrame as a pandas Series
column_data = df.iloc[:, column_index]

# Get the column name from the DataFrame based on the provided column index
column_name = df.columns[column_index]

# Find outliers in the specified column using the modified function
outliers = find_outliers_iqr(column_data)

# Calculate the total number of data points
total_data_points = len(column_data)

# Calculate the number of outliers
num_outliers = len(outliers)

# Calculate the number of non-outliers
num_non_outliers = total_data_points - num_outliers

# Calculate the percentage of outliers
percentage_outliers = (num_outliers / total_data_points) * 100
```

```
# Print the outliers and their locations
print("Outliers found in column '{}'".format(column_name))
for outlier in outliers:
    row_index = outlier[1]  # Assuming the index corresponds to row number
    print("Value:", outlier[0], "at Row:", row_index)

# Print the number of outliers in the dataset
print("Number of outliers in the dataset: {} out of {}".format(num_outliers,
total_data_points))

# Print the percentage of outliers in the dataset
print("Percentage of outliers in the dataset: {:.2f}%".format(percentage_outliers))

# Scatter Plot
plt.figure(figsize=(8, 6))
plt.scatter(range(len(column_data)), column_data, color='blue', label='Data Points')
plt.scatter(*zip(*outliers), color='red', label='Outliers')
plt.title('Scatter Plot of ' + column_name)
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.savefig('scatter_plot.png')  # Save scatter plot as image
plt.show()

# Box Plot
plt.figure(figsize=(8, 6))
plt.boxplot(column_data, vert=False)
plt.title('Box Plot of ' + column_name)
plt.xlabel('Values')
plt.grid(True)
plt.savefig('box_plot.png')  # Save box plot as image
plt.show()

# Histogram
plt.figure(figsize=(8, 6))
plt.hist(column_data, bins=20, color='skyblue', edgecolor='black')
plt.title('Histogram of ' + column_name)
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.grid(True)
plt.savefig('histogram.png')  # Save histogram as image
plt.show()

# Line Chart
plt.figure(figsize=(8, 6))
plt.plot(range(len(column_data)), column_data, color='green')
plt.title('Line Chart of ' + column_name)
plt.xlabel('Index')
plt.ylabel('Values')
plt.grid(True)
plt.savefig('line_chart.png')  # Save line chart as image
plt.show()
```

Here's how the function works:

1. **Input**: It takes a pandas Series (**column_data**), representing a column of numerical values, as input.
2. **Calculating Quartiles and IQR**: The function calculates the first quartile (Q1) and third quartile (Q3) of the column data, and then computes the Interquartile Range (IQR), which is the difference between Q3 and Q1.

3. **Defining Bounds**: Using the IQR, the function defines lower and upper bounds for outlier detection. Values outside these bounds are considered outliers. The bounds are typically set at 1.5 times the IQR away from the first and third quartiles.
4. **Identifying Outliers**: It then iterates through the column data, identifying values that fall below the lower bound or above the upper bound. For each outlier found, it stores the value and its index in a list of tuples.
5. **Output**: The function returns a list of tuples containing the outlier values and their indices.

In addition to outlier detection, the script also includes data visualization techniques such as scatter plots, box plots, histograms, and line charts to visually represent the data and highlight the detected outliers.

2. Remove outlier values from a specified column:

```
import numpy as np
import pandas as pd
import os

def delete_outliers(df, column_index):
    """
    Delete outlier values from a DataFrame.

    Args:
    - df: A pandas DataFrame.
    - column_index: The index of the column containing outliers.

    Returns:
    - df_cleaned: The DataFrame with outlier values removed.
    """
    # Find outliers in the specified column
    column_data = df.iloc[:, column_index]
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Delete outlier values
    df_cleaned = df.copy()  # Make a copy of the original DataFrame
    df_cleaned.loc[(df_cleaned.iloc[:, column_index] < lower_bound) |
(df_cleaned.iloc[:, column_index] > upper_bound), df.columns[column_index]] = np.nan

    return df_cleaned

# Load the CSV file into a pandas DataFrame
input_filename = "example11.csv"  # Change this to the desired input filename
df = pd.read_csv(input_filename)

# Choose the column index you want to perform outlier detection on
column_index = 1  # Change this to the desired column index

# Delete outlier values from the specified column
df_cleaned = delete_outliers(df, column_index)

# Extract the basename of the original filename without the extension
filename, extension = os.path.splitext(input_filename)
basename = os.path.basename(filename)
```

```
# Construct the output filename with the suffix "deleted_outliers"
output_file = basename + "_deleted_outliers" + extension

# Save the cleaned DataFrame to a new CSV file with the modified filename
df_cleaned.to_csv(output_file, index=False)

print("Cleaned DataFrame with outlier values removed has been saved to
'{}'.".format(output_file))
```

Here's how the function works:

1. **Input**: It takes two arguments: the Data Frame (**df**) and the index of the column containing outliers (**column_index**).
2. **Outlier Detection**: Using the Interquartile Range (IQR) method, the function identifies outliers in the specified column. It calculates the first quartile (Q1), third quartile (Q3), and the interquartile range (IQR) of the column data.
3. **Defining Bounds**: Lower and upper bounds for outlier detection are then determined based on the IQR. Values falling below the lower bound or above the upper bound are considered outliers.
4. **Deleting Outliers**: The function creates a copy of the original Data Frame (**df_cleaned**) and replaces outlier values in the specified column with NaN (missing values). This effectively removes the outliers from the Data Frame.
5. **Output**: The cleaned Data Frame (**df_cleaned**) with outlier values removed is returned.

After cleaning the Data Frame, the script saves the cleaned Data Frame to a new CSV file with the suffix "_deleted_outliers" appended to the original filename.

3. Replace outlier values in a specified column with the average of non-outlier values within that column:

```
import numpy as np
import pandas as pd
import os

def replace_outliers_with_average(df, column_index):
    """
    Replace outliers in a DataFrame with the average of values in the column.

    Args:
    - df: A pandas DataFrame.
    - column_index: The index of the column containing outliers.

    Returns:
    - df_corrected: The DataFrame with outliers replaced by the average of values in the
column.
    """
    # Find outliers in the specified column
    column_data = df.iloc[:, column_index]
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Calculate the average of values within the interquartile range
    avg = np.mean(df[(df.iloc[:, column_index] >= lower_bound) & (df.iloc[:, column_index] <=
upper_bound)].iloc[:, column_index])

    # Ensure that the average value has the same data type as the column
    avg = df.iloc[:, column_index].dtype.type(avg)

    # Replace outliers with the average of values
    df_corrected = df.copy()
    df_corrected.loc[(df_corrected.iloc[:, column_index] < lower_bound) |
(df_corrected.iloc[:, column_index] > upper_bound), df.columns[column_index]] = avg

    return df_corrected

# Load the CSV file into a pandas DataFrame
input_filename = "example11.csv"  # Change this to the desired input filename
df = pd.read_csv(input_filename)

# Choose the column index you want to perform outlier detection on
column_index = 1  # Change this to the desired column index

# Replace outliers with the average of values in the specified column
df_corrected = replace_outliers_with_average(df, column_index)

# Extract the basename of the original filename without the extension
filename, extension = os.path.splitext(input_filename)
basename = os.path.basename(filename)

# Construct the output filename with the suffix "_average_to_outliers"
output_file = basename + "_average_to_outliers" + extension

# Save the corrected DataFrame to a new CSV file with the modified filename
df_corrected.to_csv(output_file, index=False)

print("Corrected DataFrame with outliers replaced by the average of values has been saved to
'{}'.".format(output_file))
```

Here's how the function works:

1. **Input**: The function takes two arguments: the Data Frame (**DF**) and the index of the column containing outliers (**column_index**).
2. **Outlier Detection**: Using the Interquartile Range (IQR) method, the function identifies outliers in the specified column. It calculates the first quartile (Q1), third quartile (Q3), and the interquartile range (IQR) of the column data.
3. **Calculating Average**: It then calculates the average of values within the interquartile range, excluding outliers. This average serves as the replacement value for outliers.
4. **Replacing Outliers**: Outliers are replaced with the calculated average of non-outlier values in the column.
5. **Output**: The corrected Data Frame (**df_corrected**) with outliers replaced by the average of values is returned.

After replacing outliers, the script saves the corrected Data Frame to a new CSV file with the suffix "_average_to_outliers" appended to the original filename.

4. Replace outlier values in a specified column of with a chosen value:

```
import numpy as np
import pandas as pd
import os

def replace_outliers_with_value(df, column_index, new_value):
    """
    Replace outliers in a DataFrame with a chosen value.

    Args:
    - df: A pandas DataFrame.
    - column_index: The index of the column containing outliers.
    - new_value: The new value to replace outliers with.

    Returns:
    - df_corrected: The DataFrame with outliers replaced by the chosen value.
    """
    # Find outliers in the specified column
    column_data = df.iloc[:, column_index]
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Replace outliers with the chosen value
    df_corrected = df.copy()
    df_corrected.loc[(df_corrected.iloc[:, column_index] < lower_bound) |
(df_corrected.iloc[:, column_index] > upper_bound), df.columns[column_index]] = new_value

    return df_corrected

# Load the CSV file into a pandas DataFrame
input_filename = "example11.csv"  # Change this to the desired input filename
df = pd.read_csv(input_filename)

# Choose the column index you want to perform outlier detection on
column_index = 1  # Change this to the desired column index

# Choose the value to replace outliers with
```

```
chosen_value = 999  # Change this to the desired value

# Replace outliers with the chosen value in the specified column
df_corrected = replace_outliers_with_value(df, column_index, chosen_value)

# Extract the basename of the original filename without the extension
filename, extension = os.path.splitext(input_filename)
basename = os.path.basename(filename)

# Construct the output filename with the suffix "_outliers_replaced_with_{chosen_value}"
output_file = basename + "_outliers_replaced_with_" + str(chosen_value) + extension

# Save the corrected DataFrame to a new CSV file with the modified filename
df_corrected.to_csv(output_file, index=False)

print("Corrected DataFrame with outliers replaced by the chosen value has been saved to
'{}'.".format(output_file))
```

Here's how the function works:

1. **Input**: The function takes three arguments: the Data Frame (**DF**), the index of the column containing outliers (**column_index**), and the value to replace outliers with (**new_value**).
2. **Outlier Detection**: Using the Interquartile Range (IQR) method, the function identifies outliers in the specified column. It calculates the first quartile (Q1), third quartile (Q3), and the interquartile range (IQR) of the column data.
3. **Replacing Outliers**: Outliers are replaced with the chosen value (**new_value**). Any value falling below the lower bound or above the upper bound is replaced with the specified value.
4. **Output**: The corrected Data Frame (**df_corrected**) with outliers replaced by the chosen value is returned.

After replacing outliers, the script saves the corrected Data Frame to a new CSV file. The filename is modified to include information about the chosen value used for outlier replacement.

6.2.4 Handling duplicates

Detect Duplicates:

```
import csv

def find_duplicates(input_file, column_index):
    duplicates = {}
    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        next(reader)  # Skip header row if present
        for row_number, row in enumerate(reader, start=1):
            value = row[column_index]
            if value in duplicates:
                duplicates[value].append(row_number)
            else:
                duplicates[value] = [row_number]

    # Filter duplicates
    duplicate_entries = {value: rows for value, rows in duplicates.items() if
len(rows) > 1}
    if not duplicate_entries:
        print("No duplicates found.")
        return

    print("Duplicates found:")
    for value, rows in duplicate_entries.items():
        print(f"Value '{value}' duplicated at rows: {', '.join(map(str, rows))}")


input_file = 'output Dots.csv'
column_index = 2  # Specify the index of the column to check for duplicates
find_duplicates(input_file, column_index)
```

The `find_duplicates` function is designed to identify and report duplicate entries within a specified column of a CSV file. Let's delve into how this function operates:

1. Imports: The function imports the `csv` module for handling CSV files.

2. Function Definition: `find_duplicates` is defined with two parameters:

- `input_file`: The path to the CSV file to be analyzed.

- `column_index`: The index of the column within the CSV file where duplicates are to be checked.

3. Dictionary Initialization: A dictionary named `duplicates` is initialized. This dictionary will hold the values from the specified column as keys, and the corresponding row numbers where these values occur as values.

4. File Reading and Enumeration: The function opens the input CSV file using a `with` statement and iterates over its rows using a `csv.reader` object. It skips the header row (if present) using `next (reader) `.

5. Duplicate Identification: For each row in the CSV file, the function retrieves the value from the specified column and checks if it's already present in the `duplicates` dictionary. If it is, the row number is appended to the list associated with that value. Otherwise, a new list is created with the row number.

6. Filtering Duplicate Entries: After processing all rows, the function filters out duplicate entries by retaining only those values in the `duplicates` dictionary which have more than one associated row number. This filtered dictionary is named `duplicate_entries`.

7. Output: If no duplicates are found, the function prints a message indicating so. Otherwise, it prints each duplicate value along with the rows where it occurs.

8. Function Invocation: Finally, the function is invoked with the specified input file and column index.

This function offers a straightforward approach to identify and report duplicate entries within a CSV file, making it useful for data validation and cleaning tasks.

Remove Duplicates:

```
import csv
import os

def remove_duplicates(input_file, column_index):
    unique_values = set()
    output_file = os.path.splitext(input_file)[0] + '_unique.csv'

    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        with open(output_file, 'w', newline='') as outfile:
            writer = csv.writer(outfile)
            for row in reader:
                value = row[column_index]
                if value not in unique_values:
                    writer.writerow(row)
                    unique_values.add(value)

input_file = 'output Dots.csv'
column_index = 0  # Specify the index of the column to check for duplicates
remove_duplicates(input_file, column_index)
```

This function, `remove_duplicates`, serves the purpose of removing duplicate rows from a CSV file based on a specified column index. Here's a breakdown of how it works:

1. Imports: The function imports the necessary modules `csv` and `os` for handling CSV files and file paths, respectively.

2. Function Definition: The function `remove_duplicates` takes two arguments:

- `input_file`: The path to the CSV file from which duplicates are to be removed.
- `column_index`: The index of the column in the CSV file that is used to identify duplicates.

3. Set Initialization: A set named `unique_values` is initialized to store unique values encountered in the specified column.

4. Output File Path Determination: The output file path is determined by appending '_unique.csv' to the input file's base name. This ensures that the original file is not overwritten.

5. File handling with `with` Statement: The function uses `with` statements to open the input and output files. This ensures that the files are properly closed after use, even if an exception occurs.

6. CSV Reading and Writing: Inside the first `with` block, the function reads the input CSV file using `csv.reader`. Inside the second `with` block, it writes to the output CSV file using `csv.writer`.

7. Duplicate Removal Logic: For each row in the input CSV file, the function checks the value in the specified column (determined by `column_index`). If the value is not already in `unique_values`, the row is written to the output CSV file, and the value is added to `unique_values`. This ensures that only unique rows are written to the output file.

8. Function Invocation: Finally, the function is invoked with the specified input file and column index.

This function provides a convenient way to remove duplicate rows from a CSV file while retaining the order of appearance and without requiring the entire file to fit into memory at once.

6.3 System Modules Implementation

6.3.1 Missing values

```
@app.route('/Missing', methods=["GET", "POST"])
def miss():
  filenames=get_uploaded_files()
  if request.method == 'POST':
    operation = request.form['operation']
    filename = request.form['table']
    if filename=="Not select":
      return render_template('Missing_value.html', filenames=filenames,table_original="You haven't selected any Table!",missingPers=0,missingCount=0,totalRows=0)
    else:
      df = pd.read_csv('uploads/'+filename)
      def show():
          missing_Table =find_missing_values_rows(df)
          if missing_Table.count().max()==0:
            missing_Html_Table="No Missing Value Found.!"
          else:
            missing_Html_Table = missing_Table.to_html(classes='table table-bordered table-striped sizee', index=False)
          table_html_original = df.to_html(classes='table table-bordered table-striped sizee', index=False)
          missingPers,missingCount,totalRows=pers_missing(df)
          return missingPers,missingCount,totalRows,missing_Html_Table,table_html_original
      if (operation=='show'):

        missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
        return render_template('Missing_value.html', table_original=table_html_original, table_cleaned=missing_Html_Table,filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
      elif (operation=='fill'):
        intType = request.form['fill-int-type']
        if intType=='Not select int':
          return render_template('Missing_value.html',intmsg='Select Option To Fill Integer Missing Value',missingPers=0,missingCount=0,totalRows=0)
        else:
          strValue = request.form['fill-str-value']
          if strValue=='':
            return render_template('Missing_value.html',strmsg='Select Option To Fill String Missing Value',missingPers=0,missingCount=0,totalRows=0)
          else:
            newdata=handle_missing(df,intType,strValue)
            newdata.to_csv("uploads/"+filename,index=False)
            table_html_cleaned = newdata.to_html(classes='table table-bordered table-striped ', index=False)
            missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
            return render_template('Missing_value.html', table_original=table_html_original, table_cleaned=table_html_cleaned,filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
      elif operation=='delete' :
        df=df.dropna()
        df.to_csv('uploads/'+filename)
        missingPers,missingCount,totalRows=pers_missing(df)
        table_html_original = df.to_html(classes='table table-bordered table-striped ', index=False)
        missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
        return render_template('Missing_value.html', table_original=table_html_original,table_cleaned="No Missing Value Found.!", filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
  else:
    return render_template('Missing_value.html', filenames=filenames,missingPers=0,missingCount=0,totalRows=0)
```

1. Routing and Request Handling:

    - The Flask application defines a route `/Missing` to handle requests related to missing value operations using both `GET` and `POST` methods.
2. View Function miss():
    - This function serves as the controller for the `/Missing` route.
    - For `POST` requests:
        - It retrieves the list of uploaded filenames using `get_uploaded_files()`.
        - Depending on the operation specified by the user (`show`, `fill`, `delete`), it performs corresponding actions on the uploaded CSV files:
            - If the operation is `show`, it displays missing values in the CSV file along with basic statistics.
            - If the operation is `fill`, it handles missing values based on user-defined criteria and updates the CSV file accordingly.

        - If the operation is `delete`, it removes rows with missing values from the CSV file.
    - For `GET` requests, it renders the initial page for uploading files.

3. Data Processing Functions:
    - The function renders the HTML template `Missing_value.html`, passing relevant data such as filenames, original and modified table HTML, and missing value statistics.
    - The template dynamically adjusts its content based on user interactions and the operation performed.
4. Data Processing Functions:
    - The code utilizes several helper functions such as `get_uploaded_files()`, `find_missing_values_rows()`, `pers_missing()`, and `handle_missing()` to process data and perform operations related to missing values in CSV files.
5. Input Validation and Error Handling:
    - Basic input validation is implemented to ensure that users select a file and provide necessary inputs for filling missing values.
    - Error messages are displayed when required inputs are missing or invalid.
6. File Operations:
    - CSV files uploaded by users are read into pandas Data Frames for processing.
    - Modified Data Frames are saved back to CSV files in the `uploads` directory.
7. User Interface:
    - The system provides a user-friendly interface for uploading files, selecting operations, and viewing results.
    - Feedback messages and error alerts are displayed to guide users through the process.

Overall, this implementation facilitates the exploration and manipulation of missing values in CSV files through a web-based interface, enhancing usability and accessibility for users.

6.3.2 Duplicates

```python
@app.route('/duplicate',methods=['POST', 'GET'])
def duplicate():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        operation = request.form['operation']
        filename = request.form['table']
        if filename=="Not select":
          return render_template('duplicate.html', filenames=filenames,table_original="You haven't selected any Table!",pers=0,effectedRows=0,totalRows=0)
        else:
          def show_dupl():
                selected_columns = request.form.getlist('columns')
                filename = request.form['table']
                df = pd.read_csv('uploads/'+filename)
                pers,effectedRows,totalRows=pers_duplicate(df)
                columns=get_column_names(df)
                if selected_columns:
                  show_dup=find_duplicate_rows(df,selected_columns)
                  pers,effectedRows,totalRows=pers_duplicate(df,selected_columns)
                else:
                  show_dup=find_duplicate_rows(df,'all')


                if show_dup.count().max() ==0:
                    table_html_duplicate='No Duplicates Found.'
                else:
                    table_html_duplicate = show_dup.to_html(classes='table table-striped table-bordered sizee ', index=False)
                table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                return pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns
          if operation == 'show':
            pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
            return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows,columns=columns,selected_columns=selecte
          else:
            try:
              selected_columns = request.form.getlist('columns')
              if selected_columns:
                remove_duplicates_and_save(filename,selected_columns)
                pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
                return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)
              else:
                remove_duplicates_and_save(filename,'all')
                pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
                return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)
            except:
              pers,effectedRows,totalRows,table_html_original,table_html_duplicate=show_dupl()
              return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)


    return render_template('duplicate.html',filenames=filenames,pers=0,effectedRows=0,totalRows=0)
```

1. Route Definition:

    - The Flask application defines a route `/duplicate` to handle requests related to duplicate value operations using both `GET` and `POST` methods
2. View Function duplicate ():
    - This function acts as the controller for the `/duplicate` route.
    - For `POST` requests:
        - It retrieves the list of uploaded filenames using `get_uploaded_files()`.
        - Depending on the operation specified by the user (`show`, `remove`), it carries out corresponding actions on the selected CSV file.
    - For `GET` requests, it renders the initial page for selecting files and operations.
3. Data Processing Function show_dupl():
    - This function, nested within `duplicate()`, processes data related to duplicate values.
    - It reads the selected CSV file into a Data Frame (`DF`) and calculates statistics such as the percentage of duplicate rows and the number of affected rows.

- Depending on the selected columns, it identifies duplicate rows using `find_duplicate_rows()` function.
- It generates HTML tables for displaying the original and duplicate rows.

4. Template Rendering:
    - The function renders the HTML template `duplicate.html`, passing relevant data such as filenames, original and duplicate table HTML, statistics, columns, and selected columns.
    - The template dynamically adjusts its content based on user interactions and the operation performed.
5. Input Validation and Error Handling:
    - Basic input validation ensures that users select a file and provide necessary inputs for identifying and removing duplicate values.
    - Error handling manages exceptions that may occur during file processing.
6. File Operations:
    - CSV files uploaded by users are read into pandas Data Frames for processing.
    - Modified Data Frames are saved back to CSV files in the `uploads` directory after removing duplicate rows.
7. User Interface:
    - The system offers a user-friendly interface for selecting files, columns, and operations related to duplicate values.
    - It provides feedback messages and error alerts to assist users throughout the process.

This implementation provides a convenient web-based interface for users to explore and manage duplicate values in CSV files effectively, enhancing usability and accessibility.

6.3.3 Outliers

```
@app.route('/outlier',methods=['GET','POST'])
def outlier():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        filename = request.form['table']
        operation = request.form['operation']
        def show_outlier(filename):
            df = pd.read_csv('uploads/'+filename)
            columns=get_column_names(df)
            table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
            table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
            outliers,total_rows,num_outliers,outlier_percentage=getOutliersDetails(selected_columns[0],df)
            if outliers.count().max() ==0:
                return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="No Outlier Found!",total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
            else:
                table_html_outliers = outliers.to_html(classes='table table-striped table-bordered ', index=False)
                return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers=table_html_outliers,total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
        if  filename=='Select':
            return render_template('outlier.html',filenames=filenames,table_original="You Should Select Table !",total_rows=0,num_outliers=0,outlier_percentage=0)
        else:
            if operation == 'show':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
                    return render_template('outlier.html',filenames=filenames,table_original=table_html_original,total_rows=0,num_outliers=0,outlier_percentage=0,columns=columns)

            elif operation == 'find_outlier':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    chart = request.form['chart']
                    selected_columns = request.form.getlist('columns')
                    save_and_display_plot(df,selected_columns[0],chart)

                    table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
                    if selected_columns:
                      outliers,total_rows,num_outliers,outlier_percentage=getOutliersDetails(selected_columns[0],df)

                      if outliers.count().max() ==0:
                          return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="No Outlier Found!",total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
                      else:
                        table_html_outliers = outliers.to_html(classes='table table-striped table-bordered ', index=False)
                        return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers=table_html_outliers,total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
                    else:
                      return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="You Don't Select Any Row!",total_rows=0,num_outliers=0,outlier_percentage=0,columns=columns)
            elif operation == 'negative':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    selected_columns = request.form.getlist('columns')
                    table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
                    if selected_columns:
                      outliers,total_rows,num_outliers,outlier_percentage=getNegative(selected_columns[0],df)
                      if outliers.count().max() ==0:
                          return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="No Negative Value Found!",total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
                      else:
                        table_html_outliers = outliers.to_html(classes='table table-striped table-bordered ', index=False)
                        return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers=table_html_outliers,total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
                    else:
                      return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="You Don't Select Any Row!",total_rows=0,num_outliers=0,outlier_percentage=0,columns=columns)
            elif operation == 'delete_outlier':
                    selected_columns = request.form.getlist('columns')
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    if selected_columns:
                      remove_outliers_and_save(selected_columns[0],'uploads/'+filename)
                      return  show_outlier(filename)
            elif operation == 'delete_negative':
                    selected_columns = request.form.getlist('columns')
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    if selected_columns:
                      remove_negative_rows(selected_columns[0],'uploads/'+filename)
                      return  show_outlier(filename)
    else:
        return render_template('outlier.html',filenames=filenames,total_rows=0,num_outliers=0,outlier_percentage=0)
```

1. **Route Definition:**
    - The route `/outlier` is defined using the `@app.route()` decorator in Flask.
    - The route handles both `GET` and `POST` requests, allowing users to interact with the system through form submissions.
2. **View Function outlier():**
    - This function is responsible for processing user requests related to outlier detection.
    - Upon receiving a `POST` request, it retrieves the list of uploaded filenames using the `get_uploaded_files()` function.
    - It extracts the selected filename and operation from the form data.
    - Depending on the operation specified (`show`, `find_outlier`, `negative`, `delete_outlier`, `delete_negative`), it performs corresponding actions.
    - For `GET` requests, it renders the initial page for selecting files and operations.

3. **Data Processing Function show_outlier:()**

    - This nested function handles the core data processing tasks related to outlier detection.
    - It reads the selected CSV file into a pandas Data Frame (`DF`).
    - It generates HTML tables for displaying the original data and outlier details.
    - Depending on the operation, it calculates outlier details such as the number of outliers, total rows, and outlier percentage using appropriate functions (**getOutliersDetails()**, **getNegative()**).
    -

4. **Template Rendering:**
    - The function renders the HTML template `outlier.html` to display the user interface.
    - It passes relevant data such as filenames, original and outlier table HTML, statistics, and column names to the template for rendering.
    - The template is designed to dynamically update its content based on user interactions and the operation performed.
5. **Input Validation and Error Handling:**
    - Basic input validation is implemented to ensure that users select a file and provide necessary inputs for outlier detection.
    - Error handling is incorporated to manage exceptions that may occur during file processing or data manipulation.
6. **File Operations:**
    - Uploaded CSV files are read into pandas Data Frames for processing.
    - Modified Data Frames, after outlier removal, are saved back to CSV files in the `uploads` directory.

- Functions like `remove_outliers_and_save()` and `remove_negative_rows()` handle the removal of outliers or negative values from the Data Frame and save the modified data back to files.

7. **User Interface:**
    - The system provides a user-friendly interface for users to select files, columns, and operations related to outlier detection.
    - It includes interactive elements such as dropdown menus, checkboxes, and buttons to facilitate user interactions.
    - Feedback messages and error alerts are displayed to guide users through the process and provide status updates.

By combining these components, the system offers a comprehensive and intuitive web-based solution for users to detect, analyze, and manage outliers in their CSV datasets.

6.3.4 Invalid formats

```python
@app.route('/invalid' ,methods=['POST', 'GET'])
def invalid():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        operation = request.form['operation']
        filename = request.form['table']
        def show_dupl() :
                filename = request.form['table']
                df = pd.read_csv('uploads/'+filename)
                columns=get_column_names(df)
                table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,columns=columns)
        if filename=="Not select":
          return render_template('invalid_format.html', filenames=filenames,table_original="You haven't selected any Table!",pers=0,effectedRows=0,totalRows=0)
        else:
          if operation == 'show':
              return show_dupl()
          if operation == 'apply':
              Remove_operation = request.form['Remove']
              Standarize = request.form['Standarize']
              Split = request.form['Split']
              check = request.form['check']
              selected_columns = request.form.getlist('columns')
              if not (Remove_operation==''):
                  if Remove_operation=='Comma':
                    Remove.remove_commas_from_csv(filename,selected_columns)
                    return show_dupl()
                  elif Remove_operation=='Data':
                    Remove.remove_data_from_csv(filename,selected_columns)
                    return show_dupl()
                  elif Remove_operation=='Strip':
                      Remove.remove_strip(filename)
                      return show_dupl()
                  elif Remove_operation=='WhiteSpace':
                        Remove.remove_white_spaces_in_csv(filename)
                        return show_dupl()
                  elif Remove_operation=='char':
                      selected_columns = request.form.getlist('columns')
                      char = request.form['char']
                      Remove.delete_character_from_csv(filename,selected_columns[0],char)
                      return show_dupl()
                  elif Remove_operation=='num':
                      selected_columns = request.form.getlist('columns')
                      Remove.remove_numbers_and_write_by_column_name(filename,selected_columns)
                      return show_dupl()
              if not (Standarize==''):
                      if Standarize=='lower':
                          st.convert_csv_columns_to_lowercase(filename,selected_columns)
                          return show_dupl()
                      elif Standarize=='upper':
                          st.convert_csv_to_uppercase(filename,selected_columns)
                          return show_dupl()
                      elif Standarize=='Proper':
                          st.convert_csv_columns_to_propercase(filename,selected_columns)
                          return show_dupl()
              if not (Split==''):
                      first_col = request.form['first_col']
                      second_col = request.form['second_col']
                      selected_columns = request.form.getlist('columns')

                      if Split=='slash':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,'/')
                          return show_dupl()
                      elif Split=='and':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,'&')
                          return show_dupl()
                      elif Split=='dot':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,'.')
                          return show_dupl()
                      elif Split=='comma':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,',')
                          return show_dupl()
                      elif Split=='specific':
                          seperator = request.form['seperator']
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,sign=seperator)
                          return show_dupl()

              if not (check==''):
                    if check=='Email':
                          filename = request.form['table']
                          df = pd.read_csv('uploads/'+filename)
                          columns=get_column_names(df)
                          invalid_rows=ch.get_invalid_emails(filename,selected_columns[0])
                          table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                          if invalid_rows.count().max() !=0:
                              table_cleaned = invalid_rows.to_html(classes='table table-striped table-bordered sizee', index=False)
                              return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned=table_cleaned,columns=columns)
                          else:
                            return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned="No Invalid Rows Found!",columns=columns)
                    elif check=='Phone':
                          filename = request.form['table']
                          df = pd.read_csv('uploads/'+filename)
                          columns=get_column_names(df)
                          invalid_rows=ch.check_phone_numbers_in_column(filename,selected_columns[0])
                          table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                          if invalid_rows.count().max() !=0:
                              table_cleaned = invalid_rows.to_html(classes='table table-striped table-bordered sizee', index=False)
                              return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned=table_cleaned,columns=columns)
                          else:
                            return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned="No Invalid Rows Found!",columns=columns)
    return render_template('invalid_format.html',filenames=filenames)
```

1. **Route Definition:**
    - The Flask application defines a route `/invalid` to handle requests related to data validation and formatting.
2. **View Function invalid():**
    - This function serves as the controller for the `/invalid` route.
    - It retrieves the list of uploaded filenames using the get_uploaded_files() function.
    - For POST requests:
        - It extracts the operation and filename from the form data submitted by the user.
        - Depending on the selected operation (`show`, `apply`), it performs corresponding actions.
    - For GET requests, it renders the initial page for selecting files and operations.
3. **Data Processing Function show_dupl():**
    - This nested function handles the core data processing tasks related to data validation and formatting.
    - It reads the selected CSV file into a DataFrame (`DF`).
    - It generates HTML tables for displaying the original data.
    - It is used to render the `invalid_format.html` template.
4. **Template Rendering:**
    - The function renders the HTML template `invalid_format.html`, passing relevant data such as filenames, original table HTML, columns, and cleaned data (if applicable).
    - The template dynamically adjusts its content based on user interactions and the operation performed.
5. **Data Formatting Operations:**
    - Various data formatting operations are supported, including removing characters (e.g., commas, dots, whitespace), converting text case (lowercase, uppercase, proper case), and splitting columns based on separators (e.g., slash, dot, comma).
    - The user can choose specific columns and apply formatting operations accordingly.
        1. Data Validation Operations:
    - Data validation operations such as checking for invalid email addresses or phone numbers in specific columns are supported.
    - If invalid data is found, the corresponding rows are displayed in a separate table to inform the user.
6. **Input Validation and Error Handling:**
    - Basic input validation is implemented to ensure that users select a file and provide necessary inputs for data validation and formatting.
    - Error handling manages exceptions that may occur during file processing or data manipulation.

7. User Interface:

- The system provides a user-friendly interface for users to select files, columns, and operations related to data validation and formatting.
- Interactive elements such as dropdown menus, checkboxes, and buttons facilitate user interactions.
- Feedback messages and error alerts guide users through the process and provide status updates.

This implementation offers a comprehensive web-based solution for users to validate and format data in CSV files, enhancing data quality and consistency.

6.4 system integration

Introduction to System Integration with Flask and Jinja in an MVC Architecture:

In the dynamic landscape of web development, effective system integration is fundamental to building robust and scalable applications. Harnessing the power of frameworks like Flask and templating engines such as Jinja, developers can seamlessly integrate various components within the Model-View-Controller (MVC) architecture. This introduction sets the stage for exploring how Flask, Jinja, and MVC intertwine to create cohesive web applications.

Flask Framework:

Flask provides a simple yet powerful framework for developing web applications in Python. Its minimalistic design allows developers to quickly set up and extend functionality as needed. Flask offers features such as URL routing, request handling, and template rendering, making it an ideal choice for building web applications of varying complexities.

```
app=Flask(__name__)

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

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

Jinja:

Jinja is a modern and designer-friendly templating engine for Python, inspired by Django's template system. It allows developers to create dynamic web pages by embedding Python code within HTML templates. Jinja provides features like template inheritance, macros, filters, and control structures, enabling developers to create reusable and maintainable templates.

```
<select name="columns" class="form-control mt-3 fv py-5 Table-name " id="column-name" multiple>
  <option disabled value="">Select Which Column To Clean</option>
  <option value="all">all</option>

  {% for column in columns %}
  <option value={{column}}>{{ column }}</option>
  {% endfor %}

</select>
```

```
{% if table_original=="You haven't selected any Table!" %}
<p style="margin-top: 3%;" class="alert alert-danger mx-auto w-100" role="alert">{{table_original}}</p>

{% elif table_original %}
<div class="card">
  <div class="card-body" style="max-height: 400px; overflow-y: scroll;">
    {{ table_original|safe }}
  </div>
</div>

{% endif %}
```

Template Rendering with Jinja:

Jinja templates enable developers to generate dynamic HTML content by embedding Python code within HTML markup. This allows for the seamless integration of data processed by Flask into the presentation layer of the application. Developers can use Jinja's features like template inheritance and macros to create modular and maintainable templates.

```
{% extends "layout.html" %}
```

Flask Structure Files:

1-_pycache_: This is a hidden folder that Python creates to store compiled bytecode files to speed up program startup.

2- Img: This folder likely h images used in the project.

3- Static: This folder typically stores static files such as CSS, JavaScript, or images that are used by the web application.

4- Templates: This folder likely stores HTML templates used to generate the web application's user interface.

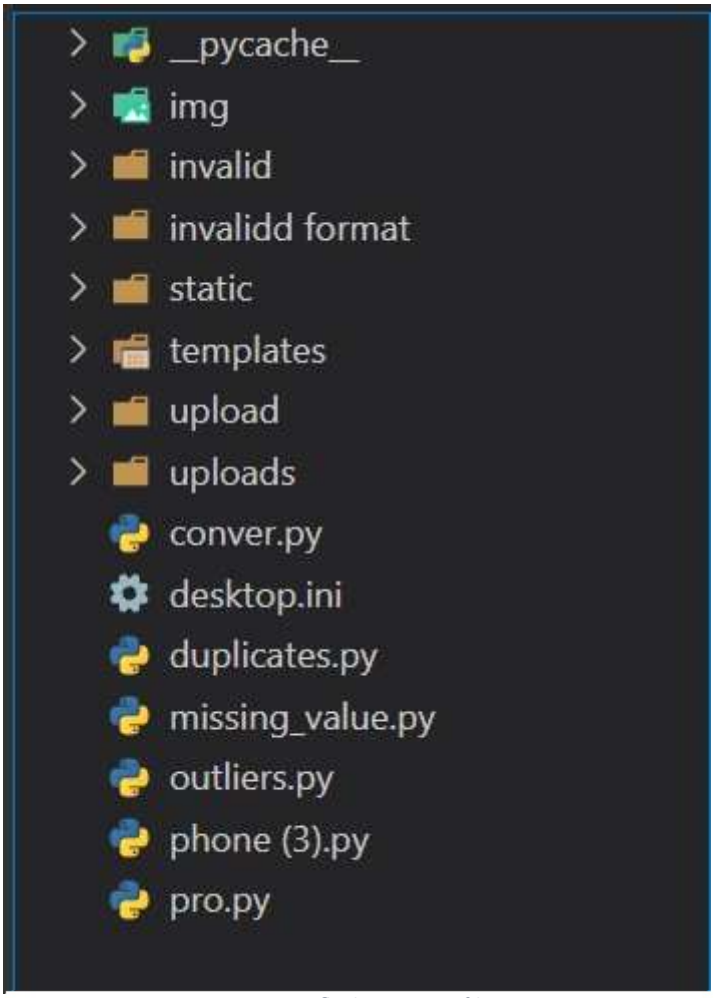
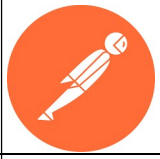
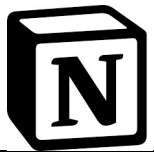
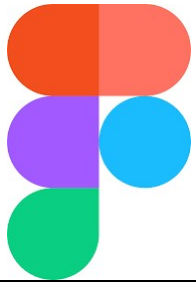


Figure 61 flask structure files

6.5 used softwares

They are the tools required for the system to be built. The usage of those tools is determined by previous studies performed during research on the appropriate tools to be used to achieve the desired results.

#	Name	Description
1	Trello 	Trello is the visual work management tool that empowers teams to ideate, plan, manage, and celebrate their work together in a collaborative, productive, and organized way.
2	Draw.io 	Draw.io is an online diagramming tool with integrations with Jira, google and confluence available free online or at accost depending on integration chosen
3	Postman 	Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.
4	Visual studio code 	Visual Studio is an integrated development environment (IDE) and Visual Studio Code is a rich text editor like Sublime Text and Atom. But the difference between the tools is more than just IDE and text editor. An IDE is a robust tool for writing, editing, debugging, and running your code.
5	Notion 	Notion is a single space where you can think, write, and plan. Capture thoughts, manage projects, or even run an entire company — and do it exactly the way you want.
6	Discord 	Discord is a free communications app that lets you share voice, video, and text chat with friends, game communities, and developers. It has hundreds of millions of users, making it one of the most popular ways to connect with people online.
7	PowerPoint ai 	PowerPoint (PPT) is a powerful, easy-to-use presentation graphics software program that allows you to create professional-looking electronic slide shows.

8	Figma 	Figma design is for people to create, share, and test designs for websites, mobile apps, and other digital products and experiences. It is a popular tool for designers, product managers, writers and developers and helps anyone involved in the design process contribute, give feedback, and make better decisions, faster.
9	Mongo db 	MongoDB Compass is a user-friendly graphical interface for interacting with MongoDB databases. It simplifies tasks like data exploration, querying, and administration, offering features for performance monitoring, schema visualization, and index management. It's designed to streamline database interaction for developers, administrators, and analysts.

4.2.4 Used technologies

#	Name	description
1	**Flask API** 	Flask is a lightweight web framework for Python. Flask API allows developers to build web APIs using Flask, providing tools and extensions to streamline the process.
2	**Fast API** 	D is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It is designed for high performance and productivity, with automatic interactive API documentation.
3	**REST API** 	REST (Representational State Transfer) is an architectural style for designing networked applications. REST APIs allow systems to communicate over HTTP by defining a set of operations that correspond to the HTTP methods GET, POST, PUT, DELETE, etc.
4	**jinga 2 API** 	Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is a text-based template language that renders templates to produce documents.

5	**Python 3** 	Python is a high-level programming language known for its simplicity and readability. Python 3 is the latest major release of the Python programming language, featuring improvements over Python 2.x.
6	**Bootstrap** 	Bootstrap is a popular open-source CSS framework for building responsive and mobile-first websites and web applications. It provides pre-designed templates and components for easier development.
7	**CSS** 	CSS is the acronym of “Cascading Style Sheets”. CSS is a computer language for laying out and structuring web pages (HTML or XML). This language contains coding elements and is composed of these “cascading style sheets” which are equally called CSS files.
8	**Csv editor** 	A CSV (Comma-Separated Values) editor is a tool used to view, edit, and manipulate CSV files. It typically provides features for importing, exporting, sorting, filtering, and formatting CSV data, making it easier to work with tabular data in spreadsheet-like formats.

System shot screens:-

1-missing values page

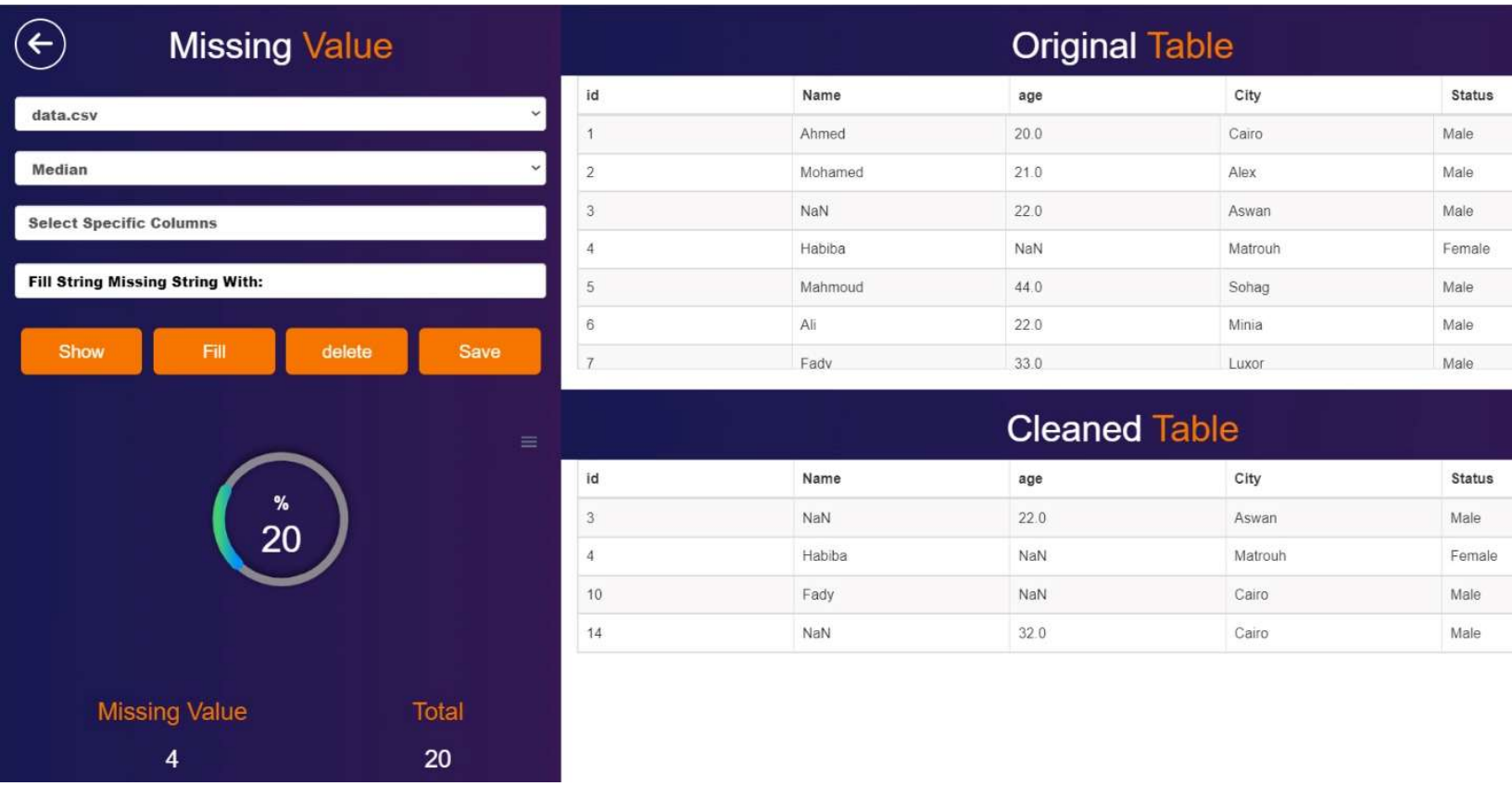


Figure 62 missing values system shotscreen

2-missing values results

Cleaned Table

id	Name	age	City	Status
1	Ahmed	20.0	Cairo	Male
2	Mohamed	21.0	Alex	Male
3	Hamza	22.0	Aswan	Male
4	Habiba	32.0	Matrouh	Female
5	Mahmoud	44.0	Sohag	Male
6	Ali	22.0	Minia	Male
7	Fady	33.0	Luxor	Male
8	Bassant	32.0	Aswan	Female
9	Nader	40.0	Cairo	Male
10	Fady	32.0	Cairo	Male

Figure 63 missing values results system shotscreen

3-duplicates page

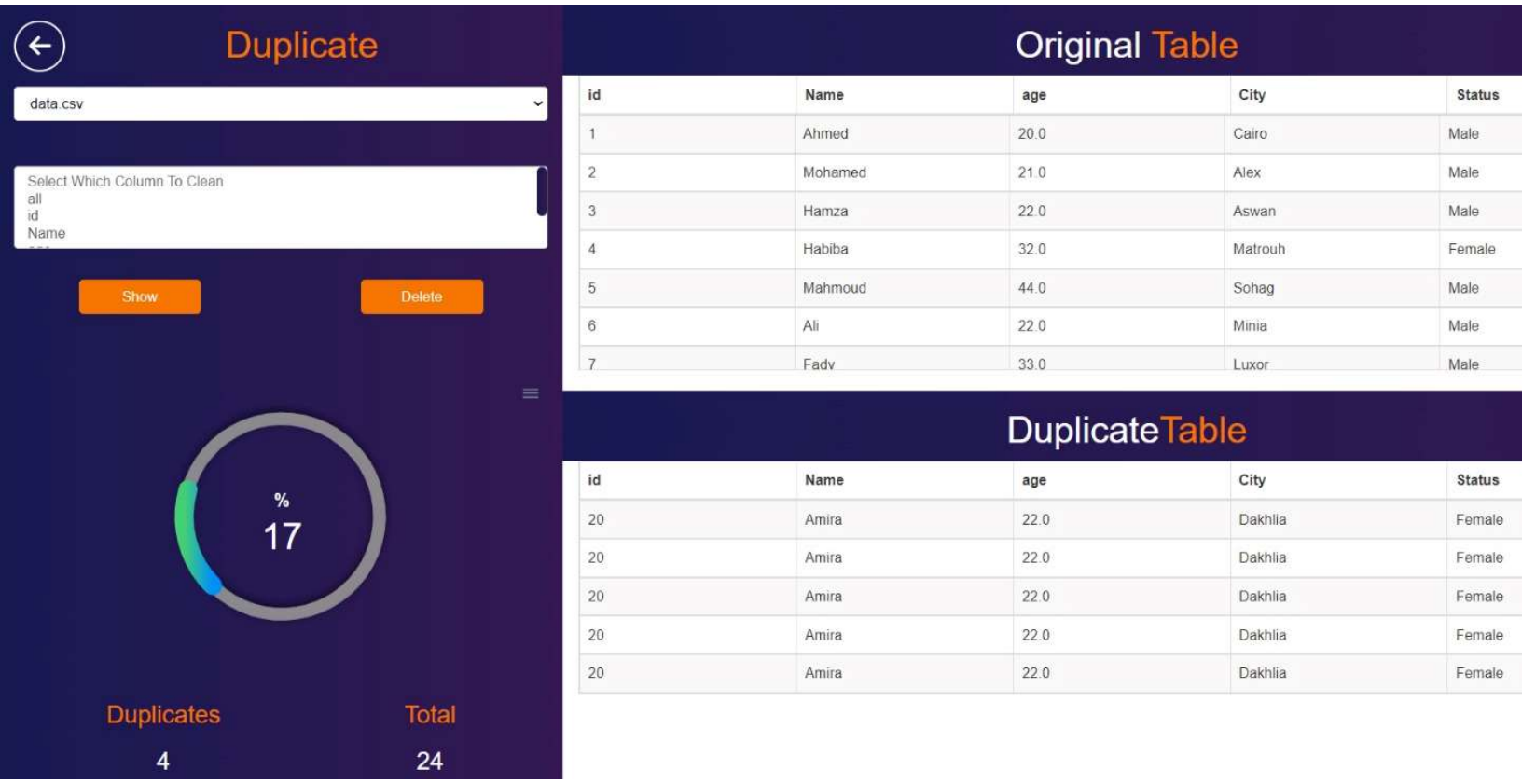


Figure 64 duplicates page system shotscreen

4-duplicates results

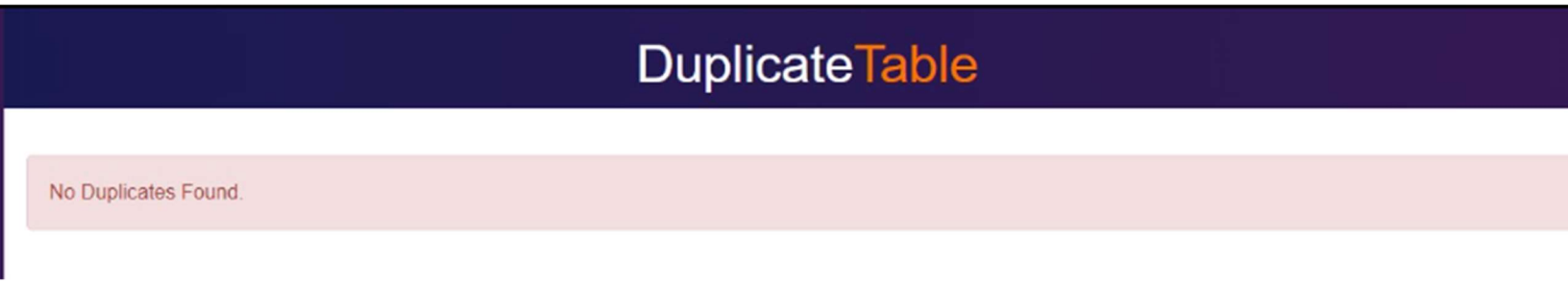


Figure 65 duplicates results system shotscreen

5-outliers page

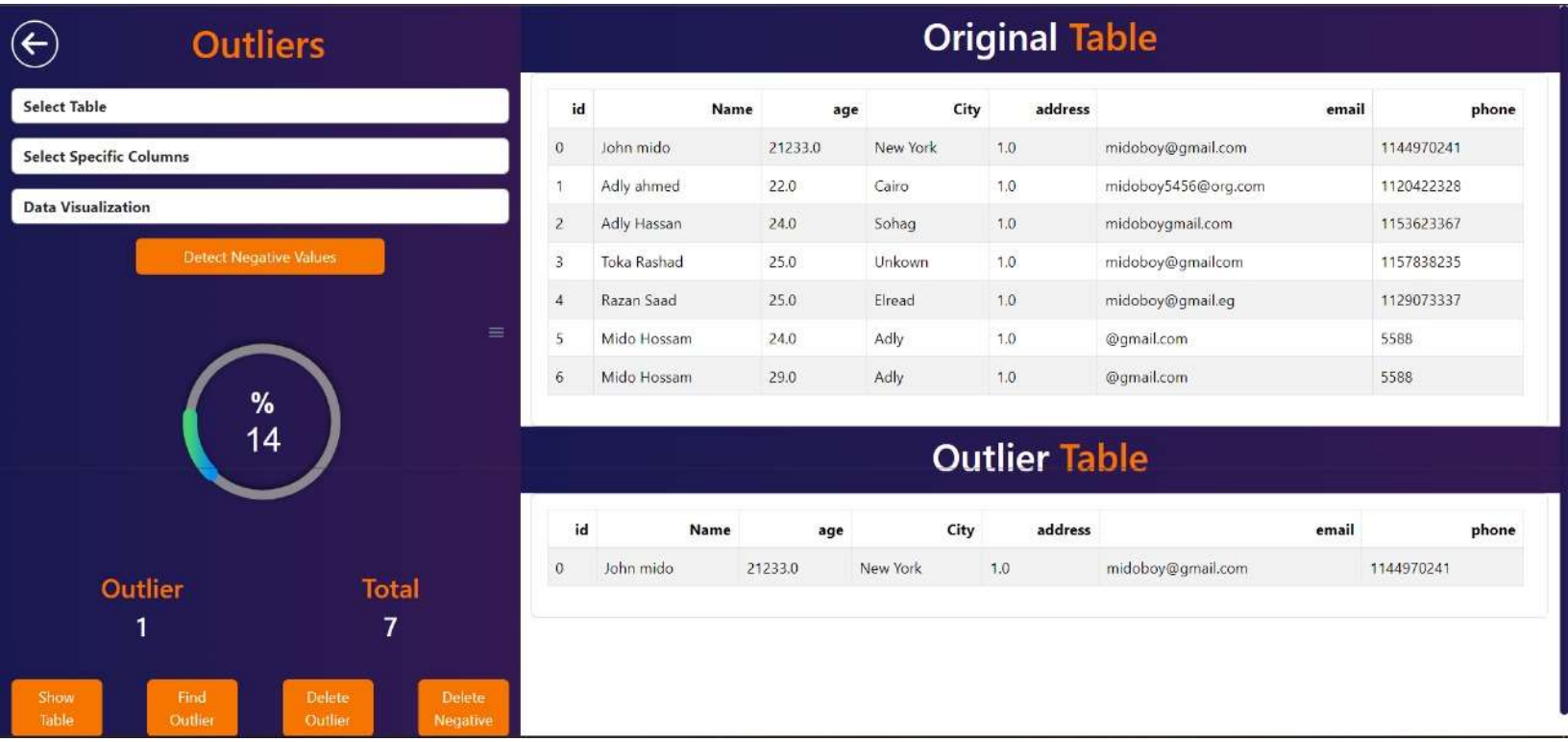


Figure 66 outliers page system shot screen

6-outliers results

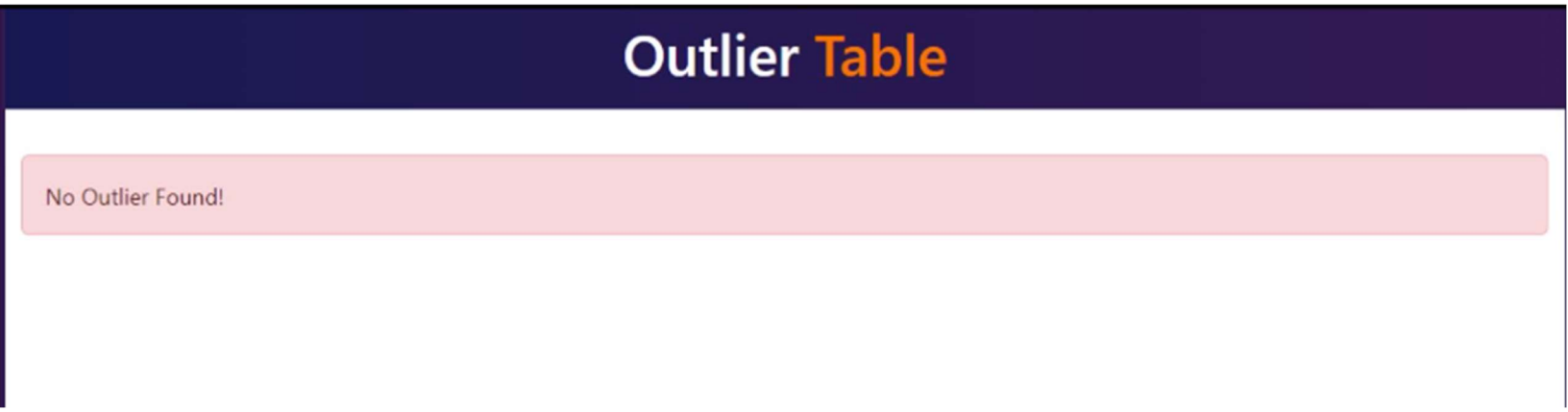
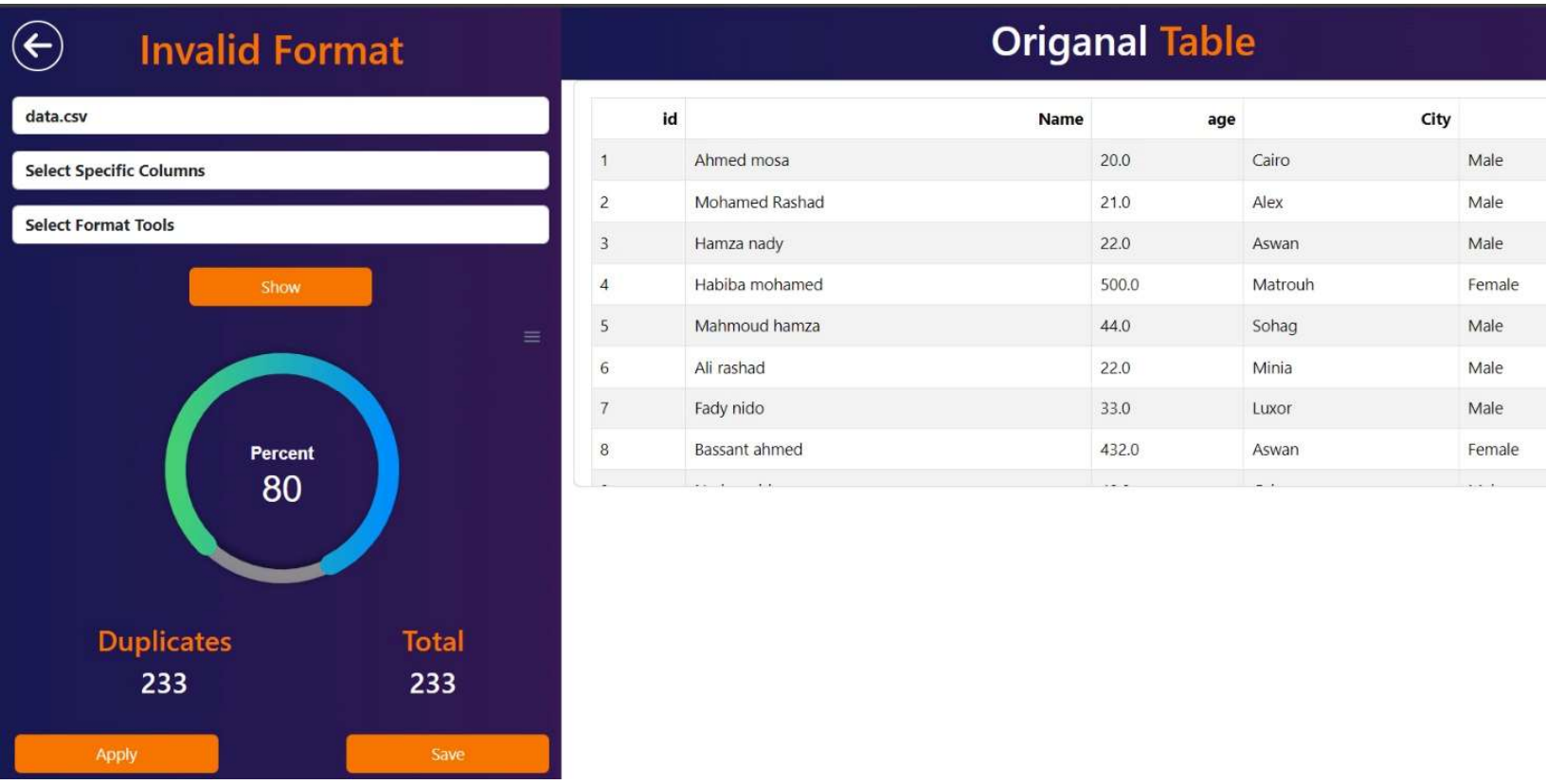


Figure 67 outliers results system shotscreen

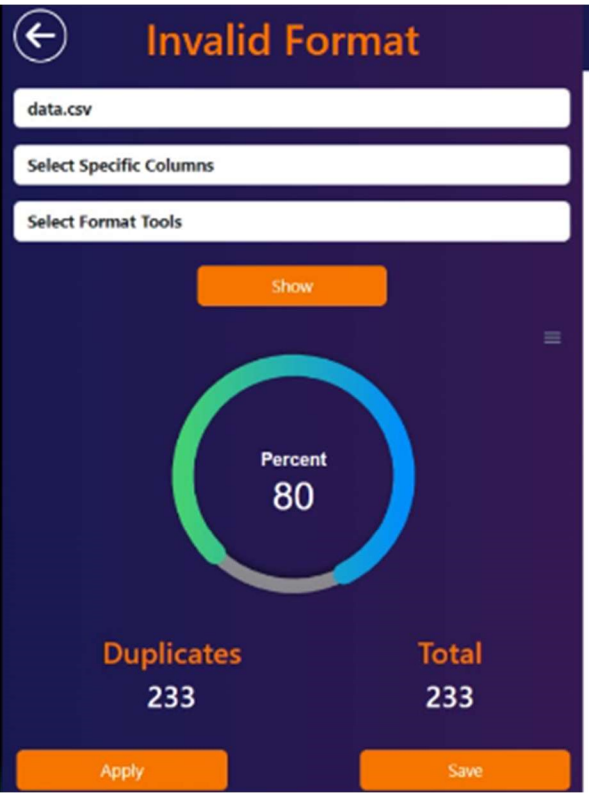
7-Invalid formats page



id	Name	age	City	
1	Ahmed mosa	20.0	Cairo	Male
2	Mohamed Rashad	21.0	Alex	Male
3	Hamza nady	22.0	Aswan	Male
4	Habiba mohamed	500.0	Matrouh	Female
5	Mahmoud hamza	44.0	Sohag	Male
6	Ali rashad	22.0	Minia	Male
7	Fady nido	33.0	Luxor	Male
8	Bassant ahmed	432.0	Aswan	Female

Figure 68 invalid formats page system shotscreen

8-invalid formats results



Id	First Name	Last Name	age	City	
1	Ahmed	mosa	20.0	Cairo	Male
2	Mohamed	Rashad	21.0	Alex	Male
3	Hamza	nady	22.0	Aswan	Male
4	Habiba	mohamed	500.0	Matrouh	Female
5	Mahmoud	hamza	44.0	Sohag	Male
6	Ali	rashad	22.0	Minia	Male
7	Fady	nido	33.0	Luxor	Male
8	Bassant	ahmed	432.0	Aswan	Female

Figure 69 invalid formats results system shotscreen

Chapter SEVEN
System Testing

Chapter-7-System Testing

7.1 Introduction

The main purpose of system testing is to evaluate and assess the quality of work performed at each step of the system development process. It is a crucial step in ensuring that the system functions as intended and meets the desired requirements. The primary goal of testing is to verify that the system performs its intended functions accurately and reliably. By conducting thorough testing, any defects, errors, or issues in the system can be identified and addressed, thereby improving the overall quality, reliability, and maintainability of the system. Testing helps in validating the system against the specified requirements, ensuring that it meets the expected standards and performs optimally. Additionally, through testing, areas of improvement can be identified, allowing for enhancements and

Refinements to be made to the system, further enhancing its quality and reliability.

7.2 Types of Testing

There are numbers of types of software testing, categorized by what is being tested and the purpose, or objective, of the test. The objectives range from usability to disaster recovery. Generally, the most common testing types are unit testing, interface testing, functionality was testing, compatibility testing, performance testing, scalability testing, and security testing. A simple definition of the previous testing types is as follows:

• Unit testing: Functional and reliability testing in an Engineering environment. Producing tests for the behavior of components of a product to ensure their correct behavior before system integration.

• Interface testing: done by the user.

• Functionality testing: Validating an application or website conforms to its specifications and correctly accomplishes all its required functions. It may include testing the mathematical and algorithm correctness of scientific and financial software, as well as testing GUI functionality.

• Compatibility testing: To ensure compatibility of an application or website with different browsers, operating systems, and hardware platforms. Compatibility testing can be performed manually or can be driven by an automated functional or regression test suite.

• Performance testing: Performance testing determines how well the software performs in terms of the speed of computations or responsiveness to the user.

• Scalability testing: Scalability testing is performed to ensure that the software will function well as the number of users, size of data sets or other factors change from small to large values.

• Security testing: Application security testing determines how well the software can defend against attacks, such as firewall software securing a computer against 99 Internet viruses and worms. In the case of the web application, it refers to the

Testing of the site and web server configuration to eliminate any security or access loopholes.

7.3 Test cases

Test case 1: upload database:-

A	B	C	D
Step	**Action**	**Expected Result**	**Status**
1	**Upload WIth Local Server**		
1.1	Click on Local Server Button	Display Select option	Pass
1.2	Select Data Base Name Which want to upload	Display upload button	Pass
1.3	Click on Upload button	Message "Data Successfully Uploaded To MongoDb"	Pass
1.4	Click On Dashboard button in nav	Go to Dashboard Page To start cleaning	Pass
2	**Upload WIth Another Server**		
2.1	Click on Another Server Button	Display input field to enter server name	Pass
2.2	Enter Server Name	Display submit button	Pass
2,3	Click on Submit Button	Message "Connection server Success"	Pass
2.4	Click On Dashboard button in nav	Go to Dashboard Page To start cleaning	Pass

Figure 70 testcase 1 Database upload

Test case 2: login:-

A	B	C	D
Step	**Action**	**Expected Result**	**Status**
Prereq.	**1. User already registered with valid data 2. Internet connection is present**		
1	**Login with a valid username & a valid password**		
1.1	Enter a valid username	Username is entered successfully and the placeholder is removed	Pass
1.2	Enter a valid password	Password is entered successfully and encrypted	Ready to Test
1.3	Click on Log in	User is logged in successfully and redirected to home page	Ready to Test
2	**Login with a valid username & a invalid password**		
2.1	Enter a valid username	Username is entered successfully and the placeholder is removed	Ready to Test
2.2	Enter an invalid password	Password is entered successfully and encrypted	Ready to Test
2.3	Click on Log in	An error message appears telling the user that his username or password is not correct	Ready to Test
3	**Login with an invalid username**		
3.1	Enter an invalid username	Username is entered successfully and the placeholder is removed	Ready to Test
3.2	Enter any password	Password is entered successfully and encrypted	Ready to Test
3.3	Click on Log in	An error message appears telling the user that his username or password is not correct	Ready to Test
4	**Login with leaving both username and password fields empty**		
4.1	Leave username and password fields empty and click on login	An error message appears telling the user that he can't leave the fields empty	Ready to Test

Figure 71 testcase 2 Login

Test case 3: register:-

A	B	C	D
Step	**Action**	**Expected Result**	**Status**
Prereq.			
1	**User Entered Valid Email, username and Password**		
1.1	input username field	Username is entered successfully and the placeholder is removed	Pass
1.2	input Email field	Email is entered successfully and the placeholder is removed	Pass
1.3	input Password field	Password is entered successfully and encrypted	Pass
2	**User Entered invalid Password**		
2.1	Enter a password	Message"Please Enter Valid Password"	Pass
3	**User Entered invalid Username**		
3.1	Enter a password	Message"Please Enter Valid Username	Pass

Figure 72 testcase 3 register

Test case 4: remove duplicates:-

Step	Action	Expected Result	Status
Prereq.	User Already Choosed a Table		
1	User Want to Delete Duplicates		
1.1	Click in multiple select Specific column and choose option	option is choosed	Pass
1.2	Click on Show Button	Display origanal and Duplicates table with Duplicated Row rows	Pass
1.3	Click on Delete Button	Display origanal and Duplicates table without Duplicated Row rows	Pass

Figure 73 testcase 4 remove duplicates

Test case 5: missing values:-

Step	Action	Expected Result	Status
Prereq.	User Already Choosed a Table		
1	User Want to Fill integer Missing Value		
1.1	Click in fill integer select option	Option is displayed (Mean, Median Mode)	Pass
1.2	Choose Fill integer Option	option is choosed	Pass
1.3	Click in multiple select Specific column and choose option	option is choosed	Pass
1.4	Click on Show Button	Display origanal and Missing value table with effected rows	Pass
1.5	Click on Fill Button	Display origanal and Cleaned Table after fill the integers	Pass
1.6	Click on Save Button	Save Data after Cleaned	Pass
2	User Want to Fill String Missing Value		
2.1	Click in multiple select Specific column and choose option	option is choosed	Pass
2.2	Enter String to fill string missing value	String is entered successfully	Pass
2.3	Click on Show Button	Display origanal and Missing value table with effected rows	Pass
2.4	Click on Fill Button	Display origanal and Cleaned Table after fill the String	Pass
2.5	Click on Save Button	Save Data after Cleaned	Pass
3	User Want to Delete Missing Value		
3.1	Click in multiple select Specific column and choose option	option is choosed	Pass
3.2	Click on Show Button	Display origanal and Missing value table with effected rows	Pass
3.3	Click on Delete Button	Display Message "No Missing Value Found.!"	Pass

Figure 74 test case 5 missing values

Test case 6: delete outlier:-

A	B	C	D
Step	**Action**	**Expected Result**	**Status**
Prereq.	**User Already Choosed a Table**		
1	**User Want to Delete Outlier**		
1.1	Click in multiple select Specific column and choose option	option is choosed	Pass
1.2	Select Kind of Data Visualzaton	option is choosed	Pass
1.3	Click on Show Button	Display origanal table	Pass
1.4	Click on Find Outlier Button	Display origanal and Outlier table	Pass
1.5	Click on Delete Outlier Button	Message"Outlier is Deleted"	Pass
2	**User Want to Delete Negative Values**		
2.1	Click in multiple select Specific column and choose option	option is choosed	Pass
2.2	Select Kind of Data Visualzaton	option is choosed	Pass
2.3	Click on Show Button	Display origanal table	Pass
2.4	Click on Detect Negative Values Button	Display origanal and Negative Values Table	Pass
2.5	Click on Delete Negative Values Button	Message"Negative Values is Deleted"	Pass

Figure 75 test case 6 delete outlier

Test case 7: Invalid Formats:--

A	B	C	D
Step	**Action**	**Expected Result**	**Status**
Prereq.	**User Already Choosed a Table**		
1	**User Want to Remove (Comma,Dots,White Space)**		
1.1	Click in multiple select Specific column and choose option	option is choosed	Pass
1.2	Select option From Remove Options	option is choosed	Pass
1.3	Click On Apply Button	Message "Updated Format Done"	Pass
2	**User Want to Remove (Character,Number)**		
2.1	Click in multiple select Specific column and choose option	option is choosed	Pass
2.2	Select option From Remove Options (Character,Number)	option is choosed	Pass
2.3	input Character Field	Input Entered Successfully	Pass
2.3	input Number Field	Input Entered Successfully	Fail
2.4	Click On Apply Button	Message "Updated Format Done"	Pass
3	**User Want to Standarize Column**		
3.1	Click in multiple select Specific column and choose option	option is choosed	Pass
3.2	Select option From Standarize Options	option is choosed	Pass
3.3	Click On Apply Button	Display Message "Updated Format Done"	Pass
4	**User Want to Check (Phone,Email)**		
4.1	Click in multiple select Specific column and choose option	option is choosed	Pass
4.2	Select option From Check Options (Phone,Email)	option is choosed	Pass
4.3	Click On Apply Button	Display Valid Format Table	Pass
4.4	Click On Save Button	Display Message "Updated Format Done"	Pass
5	**User Want to Split Column**		
5.1	Click in multiple select Specific column and choose option	option is choosed	Pass
5.2	Select Specific Seperator from Split options	option is choosed	Pass
5.3	input First Column Name Field	Input Entered Successfully	Pass
5.4	input Second Column Name Field	Input Entered Successfully	Pass
5.5	Click On Apply Button	Display Valid Format Table	Pass
5.6	Click On Save Button	Display Message "Updated Format Done"	Pass
6	**User Want to Merge Column**		
6.1	Click in multiple select Specific column and choose 2 columns To merge	option is choosed	Pass
6.2	Check Merge Column Input	Input is Checked	Pass
6.3	input New Column Name	Input Entered Successfully	Pass
6.4	Click On Apply Button	Display Valid Format Table	Pass
6.5	Click On Save Button	Display Message "Updated Format Done"	Pass
7	**User Want to Converte Date**		
7.1	Click in multiple select Specific column and choose option	option is choosed	Pass
7.2	Check Convert Date Column Input	Input is Checked	Pass
7.3	Click On Apply Button	Convert Select Column From String Format To Date Format	Pass
7.4	Click On Save Button	Display Message "Updated Format Done"	Pass

Figure 76 testcase 7 invalid formats

Chapter EIGHT
Conclusion & Future Work

Chapter-8- Conclusion

8.1 introduction

In this chapter the project has been created and tested in terms of its functionalities and the overall use of the application. The system achieved results will be displayed as what is concluded from the creation process. Also, a comparison of our project and other related work will be shown. In addition, some extra features will be given as future updates to the final project.

8.2 Conclusion

1. Our data cleansing website, integrated with Flask and developed using HTML, CSS, and Python, marks a significant leap forward in addressing data quality challenges comprehensively.
2. By prioritizing critical issues like missing values, invalid formats, outliers, and duplicates, our approach ensures a targeted cleansing process that effectively resolves key data quality challenges.
3. The platform boasts an intuitive user interface and detailed graphical representations, empowering users to effortlessly clean and visualize their databases while making informed decisions.
4. Offering flexibility in exporting data across various formats, the platform also provides comprehensive training modules to equip users with effective database management skills.
5. Our cleansing process preserves the original structure of the database, maintaining the format of the data throughout.
6. Guided by Agile methodologies, our development process is iterative and responsive, allowing us to continually refine and enhance the platform to meet evolving user needs effectively.
7. Looking ahead, the platform remains poised for future enhancements, promising even greater efficiency and responsiveness to evolving user needs.
8. User feedback plays a crucial role in shaping the evolution of the platform, ensuring that it continues to meet the diverse needs of its user base.

8.3 Future work

1-Advanced Machine Learning Techniques:

- **Automated Error Detection:** Implement machine learning algorithms to automatically detect and suggest corrections for common data errors, such as typos, duplicates, and inconsistencies.
- **Natural Language Processing (NLP):** Use NLP to clean and standardize text data, especially useful for processing unstructured datalike reviews or comments

2. Enhanced User Interface and Experience:

- **Drag-and-Drop Functionality:** Simplify the user experience by enabling drag-and-drop capabilities for data import and export
- **Customization Options:** Allow users to customize cleaning rules and workflows to better fit their specific needs.
-

3. Integration and Interoperability:

- **Cloud Storage Support:** Enable integration with popular cloud storage solutions like AWS S3, Google Drive, and Dropbox for easy data import and export.

4. Security:

- **Data Encryption:** Ensure that all data, both in transit and at rest, is encrypted to protect sensitive information.

5. Enhanced Reporting and Monitoring:

- **Audit Logs:** Maintain detailed logs of all data cleaning activities for accountability and tracking purposes.
- **Custom Reports:** Provide tools for generating custom reports that summarize the results of data cleaning operations.

5. Training and Support:

- **In-App Tutorials:** Develop comprehensive in-app tutorials and guides to help users understand how to use the application effectively
- **Community and Support Forum:** Provide tools for generating custom reports that summarize the results of data cleaning operations.

8.4 Appendix:-

1st raw backend implementation

1st check emails

```python
import csv
import re
import dns.resolver
def is_valid_email(email):
    # Adjusted regular expression pattern for email validation
    email_pattern = re.compile(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$')

    # Match the email pattern
    if not email_pattern.match(email):
        return False  # Email pattern does not match

    # Extract domain from the email address
    domain = email.split('@')[1]
    try:
            # Resolve MX records for the domain
            mx_records = dns.resolver.resolve(domain, 'MX')
            return True  # Domain has valid MX records
    except dns.resolver.NoAnswer:
            print(f"No MX records found for the domain: {domain}")
            return False  # No MX records found for the domain
    except dns.resolver.NXDOMAIN:
        print(f"Domain does not exist: {domain}")
        return False  # Domain does not exist
    except dns.resolver.Timeout:
        print(f"Timeout occurred while querying DNS servers for domain: {domain}")
        return False  # Timeout occurred while querying DNS servers
    except Exception as e:
        print(f"An error occurred while validating email {email}: {e}")
        return False  # Other errors

def show_valid_emails_only(input_file, email_column_index):
    with open(input_file, 'r', newline='') as csv_in:
        reader = csv.reader(csv_in)
        header = next(reader)  # Skip the header
        print("Email,Is_Valid_Email")  # Print header for email and validation status

        for row in reader:
            email = row[email_column_index]  # Choose email address from the specified column index
            is_valid = is_valid_email(email)
            print(f"{email},{is_valid}")  # Print the email address and its validation status
# Example usage:
input_file = 'Example.csv'
chosen_column_index = 2  # Specify the column index for email addresses
show_valid_emails_only(input_file, chosen_column_index)
```

2nd check phone

```python
import csv

def check_phone_numbers_in_column(csv_file, column_index):
    invalid_numbers = []
    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the first row
        for row_number, row in enumerate(reader, start=2):
            if len(row) >= column_index:
                item = row[column_index - 1]  # Adjust index to 0-based
                if len(item) != 11 or not item.startswith(('012', '010', '011', '015')) or not item.isdigit():
                    invalid_numbers.append(f"Invalid phone number at row {row_number}, column {column_index}: {item}")
            else:
                print(f"Warning: Row {row_number} does not have enough columns.")

    return invalid_numbers

# Example usage:
column_index = 4  # Adjust this to the index of the column you want to check (1-based index)
invalid_numbers = check_phone_numbers_in_column('comma.csv', column_index)
if invalid_numbers:
    print("Invalid phone numbers:")
    for number in invalid_numbers:
        print(number)
else:
    print("All phone numbers in the specified column are valid.")

```

3rd lower cases

```python
import csv
import os

def convert_csv_columns_to_lowercase(input_file, columns):
    output_file = os.path.splitext(input_file)[0] + '_lowercase.csv'
    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        with open(output_file, 'w', newline='') as outfile:
            writer = csv.writer(outfile)
            for row in reader:
                lower_row = [cell.lower() if index in columns else cell for index, cell in enumerate(row)]
                writer.writerow(lower_row)

# Example usage:
input_file = 'input.csv'
selected_columns = [0, 2]  # Selecting columns 1 and 3 (0-based index)
convert_csv_columns_to_lowercase(input_file, selected_columns)

```

4th upper cases

```python
import csv
import os

def convert_csv_to_uppercase(input_file, columns):
    output_file = os.path.splitext(input_file)[0] + '_uppercase.csv'
    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        with open(output_file, 'w', newline='') as outfile:
            writer = csv.writer(outfile)
            for row in reader:
                upper_row = [cell.upper() if index in columns else cell for index, cell in enumerate(row)]
                writer.writerow(upper_row)

# Example usage:
input_file = 'input.csv'
selected_columns = [0, 2]  # Selecting columns 1 and 3 (0-based index)
convert_csv_to_uppercase(input_file, selected_columns)

```

5th Proper case

```python
import csv
import os

def convert_csv_columns_to_propercase(input_file, columns):
    output_file = os.path.splitext(input_file)[0] + '_propercase.csv'
    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        with open(output_file, 'w', newline='') as outfile:
            writer = csv.writer(outfile)
            for row in reader:
                proper_row = [cell.title() if index in columns else cell for index, cell in enumerate(row)]
                writer.writerow(proper_row)

# Example usage:
input_file = 'input.csv'
selected_columns = [0, 2]  # Selecting columns 1 and 3 (0-based index)
convert_csv_columns_to_propercase(input_file, selected_columns)
```

6th Merge

```python
import csv
import os

def merge_selected_columns(input_file, column1, column2, new_header):
    # Extract the file name and extension
    file_name, file_extension = os.path.splitext(input_file)

    output_file = f"{file_name}_merged{file_extension}"

    with open(input_file, 'r', newline='') as infile, open(output_file, 'w', newline='') as outfile:
        reader = csv.reader(infile)
        writer = csv.writer(outfile)

        # Read the header row
        header = next(reader)

        # Modify the header to replace the merged column header
        header[column1] = new_header

        # Remove the second column header
        del header[column2]

        # Write modified header to the output file
        writer.writerow(header)

        # Write modified rows
        for row in reader:
            row[column1] = f"{row[column1]} {row[column2]}"
            del row[column2]  # Delete the second column after merging
            writer.writerow(row)

# Example usage:
input_file = "splitt_first name_splitted.csv"
column1 = 1  # First column to merge (ID)
column2 = 2  # Second column to merge (Name)
new_header = "Name"  # New header name for the merged column
merge_selected_columns(input_file, column1, column2, new_header)
```

7th split

```python
import csv
import os

def split_name_column(input_file, first_column_index, second_column_index, first_column_name, last_column_name, sign):
    output_file_name, extension = os.path.splitext(input_file)
    column_name = first_column_name.lower()
    output_file_name = f"splitt_{column_name}_splitted{extension}"

    with open(input_file, 'r') as csv_input, open(output_file_name, 'w', newline='') as csv_output:
        reader = csv.reader(csv_input)
        writer = csv.writer(csv_output)

        header = next(reader)  # Read the header row
        header.insert(second_column_index, last_column_name)  # Insert new column for last name
        header[first_column_index] = first_column_name  # Update existing column name for first name
        writer.writerow(header)  # Write the modified header row

        for row in reader:
            if sign in row[first_column_index]:
                first_name, last_name = row[first_column_index].split(sign, 1)
                row[first_column_index] = first_name.strip()  # Update first name
                row.insert(second_column_index, last_name.strip())  # Insert last name
            writer.writerow(row)

# Example usage:
input_filename = 'splitt.csv'
name_of_first_column = "First Name"
name_of_second_column = "Last Name"
index_for_first_column = 1
index_for_second_column = 2
sign = "/"

split_name_column(input_filename, index_for_first_column, index_for_second_column, name_of_first_column, name_of_second_column, sign)
```

8th remove dots

```python
import pandas as pd
import os

def remove_dots_from_csv(input_file, selected_columns=None):
    # Read CSV file into a DataFrame
    df = pd.read_csv(input_file)

    # Iterate through each column index
    for col_index in selected_columns:
        col = df.columns[col_index]
        # Check if column contains dots
        if df[col].dtype == 'object' and df[col].str.contains('\.').any():
            # Remove dots from the column
            df[col] = df[col].str.replace('.', '')

    # Get the filename without extension and remove dots
    filename_without_dots = os.path.splitext(os.path.basename(input_file))[0].replace('.', '')

    # Construct the output file path with the modified filename
    output_file = filename_without_dots + '_without_dots.csv'

    # Write the modified DataFrame to a new CSV file
    df.to_csv(output_file, index=False)

# Example usage:
input_file = 'Bad Dots.csv'
selected_columns = [1, 2]  # Add the column indices you want to select
remove_dots_from_csv(input_file, selected_columns)
```

9th remove letters

```python
import csv
import re

def remove_letters_from_column(input_csv, column_index):
    with open(input_csv, 'r', newline='') as infile:
        reader = csv.reader(infile)
        header = next(reader)  # Read the header row
        column_name = header[column_index]  # Get the name of the specified column
        output_csv = f"{column_name}_without_letters.csv"  # Create new file name

        with open(output_csv, 'w', newline='') as outfile:
            writer = csv.writer(outfile)
            writer.writerow(header)  # Write the header row to the output file

            for row in reader:
                if column_index < len(row):
                    cell = row[column_index]
                    processed_cell = re.sub(r'[a-zA-Z]', '', cell)
                    row[column_index] = processed_cell
                writer.writerow(row)

input_csv = 'upper.csv'
column_index = 2  # Assuming the column index to process is 2 (0-indexed)
remove_letters_from_column(input_csv, column_index)
```

10th remove numbers

```python
import csv
import re

def remove_numbers_and_write_by_index(input_file, column_index):
    output_file = ""
    # Extract the column name
    with open(input_file, 'r') as f:
        reader = csv.reader(f)
        headers = next(reader)
        column_name = headers[column_index]
        output_file = f"{column_name}_without_numbers.csv"

    with open(input_file, 'r') as f, open(output_file, 'w', newline='') as f_out:
        reader = csv.reader(f)
        writer = csv.writer(f_out)

        # Define a function to remove numbers from a string
        def remove_numbers(text):
            return re.sub(r'\d+', '', text)

        # Process each row
        for row in reader:
            row[column_index] = remove_numbers(row[column_index])
            writer.writerow(row)

# Example usage:
input_file = 'upper.csv'
column_index = 3  # Assuming you want to operate on the third column (index 2)
remove_numbers_and_write_by_index(input_file, column_index)
```

11th remove selected from column

```python
import csv
import re

def remove_numbers_and_write_by_index(input_file, column_index):
    output_file = ""
    # Extract the column name
    with open(input_file, 'r') as f:
        reader = csv.reader(f)
        headers = next(reader)
        column_name = headers[column_index]
        output_file = f"{column_name}_without_numbers.csv"

    with open(input_file, 'r') as f, open(output_file, 'w', newline='') as f_out:
        reader = csv.reader(f)
        writer = csv.writer(f_out)

        # Define a function to remove numbers from a string
        def remove_numbers(text):
            return re.sub(r'\d+', '', text)

        # Process each row
        for row in reader:
            row[column_index] = remove_numbers(row[column_index])
            writer.writerow(row)

# Example usage:
input_file = 'upper.csv'
column_index = 3  # Assuming you want to operate on the third column (index 2)
remove_numbers_and_write_by_index(input_file, column_index)
```

12th remove double spaces

```python
import csv
import re

def remove_numbers_and_write_by_index(input_file, column_index):
    output_file = ""
    # Extract the column name
    with open(input_file, 'r') as f:
        reader = csv.reader(f)
        headers = next(reader)
        column_name = headers[column_index]
        output_file = f"{column_name}_without_numbers.csv"

    with open(input_file, 'r') as f, open(output_file, 'w', newline='') as f_out:
        reader = csv.reader(f)
        writer = csv.writer(f_out)

        # Define a function to remove numbers from a string
        def remove_numbers(text):
            return re.sub(r'\d+', '', text)

        # Process each row
        for row in reader:
            row[column_index] = remove_numbers(row[column_index])
            writer.writerow(row)

# Example usage:
input_file = 'upper.csv'
column_index = 3  # Assuming you want to operate on the third column (index 2)
remove_numbers_and_write_by_index(input_file, column_index)
```

13th strip

```python
import csv
import re
import os

def remove_spaces_in_csv(input_file):
    # Extract file name and extension
    file_name, file_extension = os.path.splitext(input_file)
    output_file = f"{file_name}_Striped{file_extension}"

    with open(input_file, 'r', newline='') as infile, \
            open(output_file, 'w', newline='') as outfile:
        reader = csv.reader(infile)
        writer = csv.writer(outfile)

        for row in reader:
            cleaned_row = [re.sub(r'^\s+|\s+$', '', cell) for cell in row]  # Remove leading and trailing spaces
            writer.writerow(cleaned_row)

    return output_file

# Example usage:
input_filename = 'spaces.csv'
output_filename = remove_spaces_in_csv(input_filename)
print(f"Spaces removed successfully! Output saved to {output_filename}")
```

14th show repeated in all tables

```python
import csv
from collections import Counter, defaultdict

def count_repeated_values(csv_file):
    repeated_values = defaultdict(list)

    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        headers = next(reader)  # Read the header row
        for column_index, column_name in enumerate(headers):
            column_values = Counter()
            file.seek(0)  # Reset file pointer to read again
            next(reader)  # Skip the header row
            for line_num, row in enumerate(reader, start=2):  # start=2 because we skipped header
                try:
                    value = row[column_index]
                    column_values[value] += 1
                    if column_values[value] > 1:
                        repeated_values[(column_name, value)].append(line_num)
                except IndexError:
                    print(f"Error: Row {line_num} in column '{column_name}' has fewer elements.")
                    continue

    return repeated_values

# Example usage:
csv_file = 'upper_unrepeated.csv'
repeated_values = count_repeated_values(csv_file)

for (column_name, value), rows in repeated_values.items():
    print(f"Value '{value}' in column '{column_name}' is repeated {len(rows)} times in rows:", rows)
```

15th show repeated on one column

```python
import csv
from collections import Counter

def find_repeated_values(csv_file, column_index):
    repeated_values = Counter()
    column_name = None
    repeated_rows = {}

    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        header = next(reader)
        if column_index < len(header):
            column_name = header[column_index]
            for row_num, row in enumerate(reader, start=2):  # Start counting rows from 2 (1-indexed)
                if len(row) > column_index:
                    value = row[column_index]
                    if value in repeated_values:
                        repeated_values[value] += 1
                        repeated_rows[value].append(row_num)
                    else:
                        repeated_values[value] = 1
                        repeated_rows[value] = [row_num]

    return column_name, repeated_values, repeated_rows

# Example usage:
csv_file = 'upper_unrepeated.csv'
column_index = 3  # Replace 0 with the index of the column you want to analyze
column_name, repeated_values, repeated_rows = find_repeated_values(csv_file, column_index)

if repeated_values:
    print(f"Repeated values in column '{column_name}':")
    for value, count in repeated_values.items():
        if count > 1:
            print(f"  Value '{value}' repeated {count} times in rows: {', '.join(map(str, repeated_rows[value]))}.")
else:
    print(f"No repeated values found in column '{column_name}'.")
```

16th handling missing values

```python
import pandas as pd
def select_NAN_columns(df,type):
                arrint=[]
                arrstr=[]
                for col in df.columns:
                  if df[col].isnull().any():
                    if df[col].dtype == 'float64':
                        arrint.append(col)
                    else:
                        arrstr.append(col)
                if(type=='str') :
                   return arrstr
                else:
                   return arrint

def handle_missing(df,intType,strvalue):
              # handling missing int values
              if (intType==''):
                   intType=0
              if (strvalue==''):
                   strvalue="missing"

              arrint=select_NAN_columns(df,'int')
              if intType=='mean':
                    for i in arrint:
                     df[i]=df[i].fillna(df[i].mean())
              elif intType=='median':
                    for i in arrint:
                      df[i] = df[i].fillna(df[i].median())
              elif intType==0:
                    for i in arrint:
                      df[i] = df[i].fillna(0)
              elif intType=='mode':
                    for i in arrint:
                      mode=df[i].value_counts().idxmax()
                      df[i] = df[i].fillna(mode)
              # handling missing str values
              arrstr=select_NAN_columns(df,'str')
              if (strvalue):
                  for i in arrstr:
                    df[i] = df[i].fillna(strvalue)
              else:
                    for i in arrstr:
                     df[i] = df[i].fillna('missing')
              return  df
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def find_outliers_iqr(column_data):
    """
    Detect outliers in a numerical dataset using the Interquartile Range (IQR) method.

    Args:
    - column_data: A pandas Series representing a column of numerical values.

    Returns:
    - outliers: A list of tuples containing outlier values found in the column and their
indices.
    """
    # Calculate the first quartile (Q1) and third quartile (Q3) of the column data
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)

    # Calculate the interquartile range (IQR)
    iqr = q3 - q1

    # Define the lower and upper bounds for outlier detection
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Identify outliers and their indices
    outliers = [(value, idx) for idx, value in enumerate(column_data) if value < lower_bound
or value > upper_bound]

    return outliers

# Load the CSV file into a pandas DataFrame
df = pd.read_csv("example11.csv")

# Choose the column index you want to perform outlier detection on
column_index = 2  # Change this to the desired column index

# Extract the specified column from the DataFrame as a pandas Series
column_data = df.iloc[:, column_index]

# Get the column name from the DataFrame based on the provided column index
column_name = df.columns[column_index]

# Find outliers in the specified column using the modified function
outliers = find_outliers_iqr(column_data)

# Calculate the total number of data points
total_data_points = len(column_data)

# Calculate the number of outliers
num_outliers = len(outliers)

# Calculate the number of non-outliers
num_non_outliers = total_data_points - num_outliers

# Calculate the percentage of outliers
percentage_outliers = (num_outliers / total_data_points) * 100

# Print the outliers and their locations
print("Outliers found in column '{}'".format(column_name))
for outlier in outliers:
```

```
    row_index = outlier[1]  # Assuming the index corresponds to row number
    print("Value:", outlier[0], "at Row:", row_index)

# Print the number of outliers in the dataset
print("Number of outliers in the dataset: {} out of {}".format(num_outliers,
total_data_points))

# Print the percentage of outliers in the dataset
print("Percentage of outliers in the dataset: {:.2f}%".format(percentage_outliers))

# Scatter Plot
plt.figure(figsize=(8, 6))
plt.scatter(range(len(column_data)), column_data, color='blue', label='Data Points')
plt.scatter(*zip(*outliers), color='red', label='Outliers')
plt.title('Scatter Plot of ' + column_name)
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.savefig('scatter_plot.png')  # Save scatter plot as image
plt.show()

# Box Plot
plt.figure(figsize=(8, 6))
plt.boxplot(column_data, vert=False)
plt.title('Box Plot of ' + column_name)
plt.xlabel('Values')
plt.grid(True)
plt.savefig('box_plot.png')  # Save box plot as image
plt.show()

# Histogram
plt.figure(figsize=(8, 6))
plt.hist(column_data, bins=20, color='skyblue', edgecolor='black')
plt.title('Histogram of ' + column_name)
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.grid(True)
plt.savefig('histogram.png')  # Save histogram as image
plt.show()

# Line Chart
plt.figure(figsize=(8, 6))
plt.plot(range(len(column_data)), column_data, color='green')
plt.title('Line Chart of ' + column_name)
plt.xlabel('Index')
plt.ylabel('Values')
plt.grid(True)
plt.savefig('line_chart.png')  # Save line chart as image
plt.show()
```

18th Remove outlier values from a specified column:

```
import numpy as np
import pandas as pd
import os

def delete_outliers(df, column_index):
    """
    Delete outlier values from a DataFrame.

    Args:
    - df: A pandas DataFrame.
    - column_index: The index of the column containing outliers.

    Returns:
    - df_cleaned: The DataFrame with outlier values removed.
    """
    # Find outliers in the specified column
    column_data = df.iloc[:, column_index]
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Delete outlier values
    df_cleaned = df.copy()  # Make a copy of the original DataFrame
    df_cleaned.loc[(df_cleaned.iloc[:, column_index] < lower_bound) |
(df_cleaned.iloc[:, column_index] > upper_bound), df.columns[column_index]] = np.nan

    return df_cleaned

# Load the CSV file into a pandas DataFrame
input_filename = "example11.csv"  # Change this to the desired input filename
df = pd.read_csv(input_filename)

# Choose the column index you want to perform outlier detection on
column_index = 1  # Change this to the desired column index

# Delete outlier values from the specified column
df_cleaned = delete_outliers(df, column_index)

# Extract the basename of the original filename without the extension
filename, extension = os.path.splitext(input_filename)
basename = os.path.basename(filename)

# Construct the output filename with the suffix "deleted_outliers"
output_file = basename + "_deleted_outliers" + extension

# Save the cleaned DataFrame to a new CSV file with the modified filename
df_cleaned.to_csv(output_file, index=False)

print("Cleaned DataFrame with outlier values removed has been saved to
'{}'.".format(output_file))
```

19th Replace outlier values in a specified column with the average of non-outlier values within that column:

```
import numpy as np
import pandas as pd
import os

def replace_outliers_with_average(df, column_index):
    """
    Replace outliers in a DataFrame with the average of values in the column.

    Args:
    - df: A pandas DataFrame.
    - column_index: The index of the column containing outliers.

    Returns:
    - df_corrected: The DataFrame with outliers replaced by the average of values in the
column.
    """
    # Find outliers in the specified column
    column_data = df.iloc[:, column_index]
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Calculate the average of values within the interquartile range
    avg = np.mean(df[(df.iloc[:, column_index] >= lower_bound) & (df.iloc[:, column_index] <=
upper_bound)].iloc[:, column_index])

    # Ensure that the average value has the same data type as the column
    avg = df.iloc[:, column_index].dtype.type(avg)

    # Replace outliers with the average of values
    df_corrected = df.copy()
    df_corrected.loc[(df_corrected.iloc[:, column_index] < lower_bound) |
(df_corrected.iloc[:, column_index] > upper_bound), df.columns[column_index]] = avg

    return df_corrected

# Load the CSV file into a pandas DataFrame
input_filename = "example11.csv"  # Change this to the desired input filename
df = pd.read_csv(input_filename)

# Choose the column index you want to perform outlier detection on
column_index = 1  # Change this to the desired column index

# Replace outliers with the average of values in the specified column
df_corrected = replace_outliers_with_average(df, column_index)

# Extract the basename of the original filename without the extension
filename, extension = os.path.splitext(input_filename)
basename = os.path.basename(filename)

# Construct the output filename with the suffix "_average_to_outliers"
output_file = basename + "_average_to_outliers" + extension

# Save the corrected DataFrame to a new CSV file with the modified filename
df_corrected.to_csv(output_file, index=False)

print("Corrected DataFrame with outliers replaced by the average of values has been saved to
'{}'.".format(output_file))
```

20th Replace outlier values in a specified column of with a chosen value:

```
import numpy as np
import pandas as pd
import os

def replace_outliers_with_value(df, column_index, new_value):
    """
    Replace outliers in a DataFrame with a chosen value.

    Args:
    - df: A pandas DataFrame.
    - column_index: The index of the column containing outliers.
    - new_value: The new value to replace outliers with.

    Returns:
    - df_corrected: The DataFrame with outliers replaced by the chosen value.
    """
    # Find outliers in the specified column
    column_data = df.iloc[:, column_index]
    q1 = np.percentile(column_data, 25)
    q3 = np.percentile(column_data, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Replace outliers with the chosen value
    df_corrected = df.copy()
    df_corrected.loc[(df_corrected.iloc[:, column_index] < lower_bound) |
(df_corrected.iloc[:, column_index] > upper_bound), df.columns[column_index]] = new_value

    return df_corrected

# Load the CSV file into a pandas DataFrame
input_filename = "example11.csv"  # Change this to the desired input filename
df = pd.read_csv(input_filename)

# Choose the column index you want to perform outlier detection on
column_index = 1  # Change this to the desired column index

# Choose the value to replace outliers with
chosen_value = 999  # Change this to the desired value

# Replace outliers with the chosen value in the specified column
df_corrected = replace_outliers_with_value(df, column_index, chosen_value)

# Extract the basename of the original filename without the extension
filename, extension = os.path.splitext(input_filename)
basename = os.path.basename(filename)

# Construct the output filename with the suffix "_outliers_replaced_with_{chosen_value}"
output_file = basename + "_outliers_replaced_with_" + str(chosen_value) + extension

# Save the corrected DataFrame to a new CSV file with the modified filename
df_corrected.to_csv(output_file, index=False)

print("Corrected DataFrame with outliers replaced by the chosen value has been saved to
'{}'.".format(output_file))
```

21st Detect Duplicates:

```
import csv

def find_duplicates(input_file, column_index):
    duplicates = {}
    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        next(reader)  # Skip header row if present
        for row_number, row in enumerate(reader, start=1):
            value = row[column_index]
            if value in duplicates:
                duplicates[value].append(row_number)
            else:
                duplicates[value] = [row_number]

    # Filter duplicates
    duplicate_entries = {value: rows for value, rows in duplicates.items() if
len(rows) > 1}
    if not duplicate_entries:
        print("No duplicates found.")
        return

    print("Duplicates found:")
    for value, rows in duplicate_entries.items():
        print(f"Value '{value}' duplicated at rows: {', '.join(map(str, rows))}")


input_file = 'output Dots.csv'
column_index = 2  # Specify the index of the column to check for duplicates
find_duplicates(input_file, column_index)
```

22nd Remove Duplicates:

```
import csv
import os

def remove_duplicates(input_file, column_index):
    unique_values = set()
    output_file = os.path.splitext(input_file)[0] + '_unique.csv'

    with open(input_file, 'r', newline='') as infile:
        reader = csv.reader(infile)
        with open(output_file, 'w', newline='') as outfile:
            writer = csv.writer(outfile)
            for row in reader:
                value = row[column_index]
                if value not in unique_values:
                    writer.writerow(row)
                    unique_values.add(value)

input_file = 'output Dots.csv'
column_index = 0  # Specify the index of the column to check for duplicates
remove_duplicates(input_file, column_index)
```

2nd System Modules Implementation

1st Missing values

```
@app.route('/Missing', methods=["GET", "POST"])
def miss():
  filenames=get_uploaded_files()
  if request.method == 'POST':
    operation = request.form['operation']
    filename = request.form['table']
    if filename=="Not select":
      return render_template('Missing_value.html', filenames=filenames,table_original="You haven't selected any Table!",missingPers=0,missingCount=0,totalRows=0)
    else:
      df = pd.read_csv('uploads/'+filename)
      def show():
          missing_Table =find_missing_values_rows(df)
          if missing_Table.count().max()==0:
             missing_Html_Table="No Missing Value Found.!"
          else:
            missing_Html_Table = missing_Table.to_html(classes='table table-bordered table-striped sizee', index=False)
          table_html_original = df.to_html(classes='table table-bordered table-striped sizee', index=False)
          missingPers,missingCount,totalRows=pers_missing(df)
          return missingPers,missingCount,totalRows,missing_Html_Table,table_html_original
      if (operation=='show'):

        missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
        return render_template('Missing_value.html', table_original=table_html_original, table_cleaned=missing_Html_Table,filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
      elif (operation=='fill'):
        intType = request.form['fill-int-type']
        if intType=='Not select int':
          return render_template('Missing_value.html',intmsg='Select Option To Fill Integer Missing Value',missingPers=0,missingCount=0,totalRows=0)
        else:
          strValue = request.form['fill-str-value']
          if strValue=='':
            return render_template('Missing_value.html',strmsg='Select Option To Fill String Missing Value',missingPers=0,missingCount=0,totalRows=0)
          else:
            newdata=handle_missing(df,intType,strValue)
            newdata.to_csv("uploads/"+filename,index=False)
            table_html_cleaned = newdata.to_html(classes='table table-bordered table-striped ', index=False)
            missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
            return render_template('Missing_value.html', table_original=table_html_original, table_cleaned=table_html_cleaned,filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
      elif operation=='delete' :
        df=df.dropna()
        df.to_csv('uploads/'+filename)
        missingPers,missingCount,totalRows=pers_missing(df)
        table_html_original = df.to_html(classes='table table-bordered table-striped ', index=False)
        missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
        return render_template('Missing_value.html', table_original=table_html_original,table_cleaned="No Missing Value Found.!", filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
  else:
      return render_template('Missing_value.html', filenames=filenames,missingPers=0,missingCount=0,totalRows=0)
```

2nd Duplicates

```
@app.route('/duplicate',methods=['POST', 'GET'])
def duplicate():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        operation = request.form['operation']
        filename = request.form['table']
        if filename=="Not select":
          return render_template('duplicate.html', filenames=filenames,table_original="You haven't selected any Table!",pers=0,effectedRows=0,totalRows=0)
        else:
          def show_dupl():
                selected_columns = request.form.getlist('columns')
                filename = request.form['table']
                df = pd.read_csv('uploads/'+filename)
                pers,effectedRows,totalRows=pers_duplicate(df)
                columns=get_column_names(df)
                if selected_columns:
                  show_dup=find_duplicate_rows(df,selected_columns)
                  pers,effectedRows,totalRows=pers_duplicate(df,selected_columns)
                else:
                  show_dup=find_duplicate_rows(df,'all')


                if show_dup.count().max() ==0:
                    table_html_duplicate='No Duplicates Found.'
                else:
                    table_html_duplicate = show_dup.to_html(classes='table table-striped table-bordered sizee ', index=False)
                table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                return pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns
          if operation == 'show':
            pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
            return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows,columns=columns,selected_columns=selecte
          else:
            try:
              selected_columns = request.form.getlist('columns')
              if selected_columns:
                remove_duplicates_and_save(filename,selected_columns)
                pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
                return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)
              else:
                 remove_duplicates_and_save(filename,'all')
                 pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
                 return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)
            except:
              pers,effectedRows,totalRows,table_html_original,table_html_duplicate=show_dupl()
              return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)


    return render_template('duplicate.html',filenames=filenames,pers=0,effectedRows=0,totalRows=0)
```

3rd Invalid formats

```
@app.route('/invalid' ,methods=['POST', 'GET'])
def invalid():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        operation = request.form['operation']
        filename = request.form['table']
        def show_dupl() :
                filename = request.form['table']
                df = pd.read_csv('uploads/'+filename)
                columns=get_column_names(df)
                table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,columns=columns)
        if filename=="Not select":
          return render_template('invalid_format.html', filenames=filenames,table_original="You haven't selected any Table!",pers=0,effectedRows=0,totalRows=0)
        else:
          if operation == 'show':
              return show_dupl()
          if operation == 'apply':
              Remove_operation = request.form['Remove']
              Standarize = request.form['Standarize']
              Split = request.form['Split']
              check = request.form['check']
              selected_columns = request.form.getlist('columns')
              if not (Remove_operation==''):
                  if Remove_operation=='Comma':
                    Remove.remove_commas_from_csv(filename,selected_columns)
                    return show_dupl()
                  elif Remove_operation=='Data':
                    Remove.remove_data_from_csv(filename,selected_columns)
                    return show_dupl()
                  elif Remove_operation=='Strip':
                      Remove.remove_strip(filename)
                      return show_dupl()
                  elif Remove_operation=='WhiteSpace':
                       Remove.remove_white_spaces_in_csv(filename)
                       return show_dupl()
                  elif Remove_operation=='char':
                      selected_columns = request.form.getlist('columns')
                      char = request.form['char']
                      Remove.delete_character_from_csv(filename,selected_columns[0],char)
                      return show_dupl()
                  elif Remove_operation=='num':
                      selected_columns = request.form.getlist('columns')
                      Remove.remove_numbers_and_write_by_column_name(filename,selected_columns)
                      return show_dupl()
              if not (Standarize==''):
                      if Standarize=='lower':
                          st.convert_csv_columns_to_lowercase(filename,selected_columns)
                          return show_dupl()
                      elif Standarize=='upper':
                          st.convert_csv_to_uppercase(filename,selected_columns)
                          return show_dupl()
                      elif Standarize=='Proper':
                          st.convert_csv_columns_to_propercase(filename,selected_columns)
                          return show_dupl()
              if not (Split==''):
                      first_col = request.form['first_col']
                      second_col = request.form['second_col']
                      selected_columns = request.form.getlist('columns')

                      if Split=='slash':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,'/')
                          return show_dupl()
                      elif Split=='and':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,'&')
                          return show_dupl()
                      elif Split=='dot':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,'.')
                          return show_dupl()
                      elif Split=='comma':
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,',')
                          return show_dupl()
                      elif Split=='specific':
                          seperator = request.form['seperator']
                          sp.split_name_column(filename,selected_columns[0],first_col,second_col,sign=seperator)
                          return show_dupl()

              if not (check==''):
                    if check=='Email':
                          filename = request.form['table']
                          df = pd.read_csv('uploads/'+filename)
                          columns=get_column_names(df)
                          invalid_rows=ch.get_invalid_emails(filename,selected_columns[0])
                          table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                          if invalid_rows.count().max() !=0:
                              table_cleaned = invalid_rows.to_html(classes='table table-striped table-bordered sizee', index=False)
                              return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned=table_cleaned,columns=columns)
                          else:
                            return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned="No Invalid Rows Found!",columns=columns)
                    elif check=='Phone':
                          filename = request.form['table']
                          df = pd.read_csv('uploads/'+filename)
                          columns=get_column_names(df)
                          invalid_rows=ch.check_phone_numbers_in_column(filename,selected_columns[0])
                          table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                          if invalid_rows.count().max() !=0:
                              table_cleaned = invalid_rows.to_html(classes='table table-striped table-bordered sizee', index=False)
                              return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned=table_cleaned,columns=columns)
                          else:
                            return render_template('invalid_format.html',filenames=filenames,table_original=table_html_original,table_cleaned="No Invalid Rows Found!",columns=columns)
    return render_template('invalid_format.html',filenames=filenames)
```

3rd Python:-

1st _Init_

```python
from flask import Flask ,request,render_template,url_for,redirect
import pandas as pd
import os
from missing_value import handle_missing,get_uploaded_files,pers_missing,find_missing_values_rows
from duplicates import find_duplicate_rows,get_column_names,pers_duplicate,remove_duplicates_and_save
# from outliers import getOutliersDetails,getNegative,remove_negative_rows
#remove_outliers_and_save,save_and_display_plot

# from conver import get_sql_server_databases,db_convert,get_server_name
# from dashboard import calculate_null_percentage_in_database
# from outliers import getOutliersDetails

app=Flask(__name__)

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER



UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


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



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


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

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

```python
@app.route('/dashboard')
def dashboard():
    return render_template('dashboard.html')


##################################################Missing##############################

@app.route('/Missing', )
def miss():
  filenames=get_uploaded_files()
  if request.method == 'POST':
    operation = request.form['operation']
    filename = request.form['table']
    if filename=="Not select":
      return render_template('Missing_value.html', filenames=filenames,table_original="You haven't selected any Table!",missingPers=0,missingCount=0,totalRows=0)
    else:
      df = pd.read_csv('uploads/'+filename)

      if (operation=='show'):


        return render_template('Missing_value.html')
      elif (operation=='fill'):
        intType = request.form['fill-int-type']
        if intType=='Not select int':
          return render_template('Missing_value.html',intmsg='Select Option To Fill Integer Missing Value',missingPers=0,missingCount=0,totalRows=0)
        else:
          strValue = request.form['fill-str-value']
          if strValue=='':
            return render_template('Missing_value.html',strmsg='Select Option To Fill String Missing Value',missingPers=0,missingCount=0,totalRows=0)
          else:
            newdata=handle_missing(df,intType,strValue)
            newdata.to_csv("uploads/"+filename,index=False)
            table_html_cleaned = newdata.to_html(classes='table table-bordered table-striped ', index=False)

            return render_template('Missing_value.html', table_original=table_html_original, table_cleaned=table_html_cleaned,
                                   filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
```

```
    elif operation=='delete' :
     df=df.dropna()
     df.to_csv('uploads/'+filename)
     missingPers,missingCount,totalRows=pers_missing(df)
     table_html_original = df.to_html(classes='table table-bordered table-striped ', index=False)

     return render_template('Missing_value.html', table_original=table_html_original,table_cleaned="No Missing Value Found.!",
                            filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows)
    # if (operation=='Not select int'):

    #   missingPers,missingCount,totalRows,missing_Html_Table,table_html_original=show()
    #   return render_template('Missing_value.html', table_original=table_html_original, table_cleaned=missing_Html_Table,
    #filenames=filenames,missingPers=missingPers,missingCount=missingCount,totalRows=totalRows,intmsg='Select Option To Fill Integer Missing Value')
  else:
     return render_template('Missing_value.html', filenames=filenames,missingPers=0,missingCount=0,totalRows=0)


#################################endMissing############################
```

```
@app.route('/duplicate',methods=['POST', 'GET'])
def duplicate():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        operation = request.form['operation']
        filename = request.form['table']
        if filename=="Not select":
          return render_template('duplicate.html', filenames=filenames,table_original="You haven't selected any Table!",pers=0,effectedRows=0,totalRows=0)
        else:
          def show_dupl():
                selected_columns = request.form.getlist('columns')
                filename = request.form['table']
                df = pd.read_csv('uploads/'+filename)
                pers,effectedRows,totalRows=pers_duplicate(df)
                columns=get_column_names(df)
                if selected_columns:
                  show_dup=find_duplicate_rows(df,selected_columns)
                  pers,effectedRows,totalRows=pers_duplicate(df,selected_columns)
                else:
                  show_dup=find_duplicate_rows(df,'all')


                if show_dup.count().max() ==0:
                    table_html_duplicate='No Duplicates Found.'
                else:
                    table_html_duplicate = show_dup.to_html(classes='table table-striped table-bordered sizee ', index=False)
                table_html_original = df.to_html(classes='table table-striped table-bordered sizee', index=False)
                return pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns
          if operation == 'show':
            pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
            return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,table_duplicate=table_html_duplicate,pers=pers,
                                   effectedRows=effectedRows,totalRows=totalRows,columns=columns,selected_columns=selected_columns)
          else:
            try:
              selected_columns = request.form.getlist('columns')
              if selected_columns:
                remove_duplicates_and_save(filename,selected_columns)
                pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
                return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,
                                       table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)
              else:
                remove_duplicates_and_save(filename,'all')
                pers,effectedRows,totalRows,table_html_original,table_html_duplicate,columns,selected_columns=show_dupl()
                return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,
                                       table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)
            except:
              pers,effectedRows,totalRows,table_html_original,table_html_duplicate=show_dupl()
              return render_template('duplicate.html',filenames=filenames,table_original=table_html_original,
                                     table_duplicate=table_html_duplicate,pers=pers,effectedRows=effectedRows,totalRows=totalRows)


    return render_template('duplicate.html',filenames=filenames,pers=0,effectedRows=0,totalRows=0)

#####################################################endDuplicate#############################
```

```python
#####################################outlier#################################

@app.route('/outlier')
def outlier():
    filenames=get_uploaded_files()
    if request.method == 'POST':
        filename = request.form['table']
        operation = request.form['operation']
        def show_outlier(filename):
           df = pd.read_csv('uploads/'+filename)
           columns=get_column_names(df)
           table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
           table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
           if outliers.count().max() ==0:
               return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="No Outlier Found!",
                                      total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
           else:
             table_html_outliers = outliers.to_html(classes='table table-striped table-bordered ', index=False)
             return render_template('outlier.html',filenames=filenames,table_original=table_html_original,
                                    table_outliers=table_html_outliers,total_rows=total_rows,num_outliers=num_outliers,
                                    outlier_percentage=outlier_percentage,columns=columns)

        if  filename=='Select':
           return render_template('outlier.html',filenames=filenames,table_original="You Should Select Table !",total_rows=0,num_outliers=0,outlier_percentage=0)
        else:

            if operation == 'show':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
                    return render_template('outlier.html',filenames=filenames,table_original=table_html_original,total_rows=0,num_outliers=0,
                                           outlier_percentage=0,columns=columns)

            elif operation == 'find_outlier':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    chart = request.form['chart']
                    selected_columns = request.form.getlist('columns')
            elif operation == 'find_outlier':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)
                    chart = request.form['chart']
                    selected_columns = request.form.getlist('columns')


                    table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
                    if selected_columns:

                      if outliers.count().max() ==0:
                          return render_template('outlier.html',filenames=filenames,table_original=table_html_original,
                                                 table_outliers="No Outlier Found!",total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,
                                                 columns=columns)
                      else:

                        table_html_outliers = outliers.to_html(classes='table table-striped table-bordered ', index=False)
                        return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers=table_html_outliers,total_rows=total_rows
                                               num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)

                    else:
                      return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="You Don`t Select Any Raw!",
                                             total_rows=0,num_outliers=0,outlier_percentage=0,columns=columns)
            elif operation == 'negative':
                    df = pd.read_csv('uploads/'+filename)
                    columns=get_column_names(df)

                    selected_columns = request.form.getlist('columns')
                    table_html_original = df.to_html(classes='table table-striped table-bordered ', index=False)
                    if selected_columns:
                      outliers,total_rows,num_outliers,outlier_percentage=[0]
                      if outliers.count().max() ==0:
                          return render_template('outlier.html',filenames=filenames,table_original=table_html_original,table_outliers="No Negative Value Found!",
                                                 total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,columns=columns)
                      else:

                        table_html_outliers = outliers.to_html(classes='table table-striped table-bordered ', index=False)
                        return render_template('outlier.html',filenames=filenames,table_original=table_html_original,
                                               table_outliers=table_html_outliers,total_rows=total_rows,num_outliers=num_outliers,outlier_percentage=outlier_percentage,
                                               columns=columns)
                    else:
                      return render_template('outlier.html',filenames=filenames,table_original=table_html_original,
                                             table_outliers="You Don`t Select Any Raw!",total_rows=0,num_outliers=0,outlier_percentage=0,columns=columns)
```

```
            elif operation == 'delete_outlier':
                selected_columns = request.form.getlist('columns')
                df = pd.read_csv('uploads/'+filename)
                columns=get_column_names(df)
                if selected_columns:
                    return  show_outlier(filename)
            elif operation == 'delete_negative':
                selected_columns = request.form.getlist('columns')
                df = pd.read_csv('uploads/'+filename)
                columns=get_column_names(df)
                if selected_columns:
                    return  show_outlier(filename)

    else:
        return render_template('outlier.html',filenames=filenames,total_rows=0,num_outliers=0,outlier_percentage=0)

################################endoutlier#########################
```

```
@app.route('/invalid')
def invalid():
    return render_template('invalid_format.html')



if __name__=='__main__':
    app.run(debug=False,port=2020)
```

2nd Configure

```python
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'a_hard_to_guess_string'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'sqlite:///mydatabase.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    LOGGING_LEVEL = 'INFO'

```

3rd Routs

```python
from flask import request, jsonify
from app import db
from app.models import User
from flask import current_app as app

@app.route('/users', methods=['POST'])
def create_user():
    data = request.get_json()
    new_user = User(name=data['name'], email=data['email'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'message': 'User created successfully!'}), 2

@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    user = User.query.get_or_404(id)
    return jsonify({'name': user.name, 'email': user.email})

```

4th Models

```python
from app import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return f'<User {self.name}>'

```

5th Convert

```python
import pyodbc
import pandas as pd
import socket
import os


def get_sql_server_databases(server):
        try:
            connection_string = f"DRIVER=SQL Server;SERVER={server};UID={''};PWD={''}"
            connection = pyodbc.connect(connection_string)
            cursor = connection.cursor()
            databases = []
            cursor.execute("SELECT name FROM sys.databases")
            rows = cursor.fetchall()
            for row in rows:
                databases.append(row[0])
            return databases
        except Exception as e:
            print("Error:", e)
            return None
def get_server_name():
        try:
            server_name = socket.gethostname()
            return server_name
        except Exception as e:
            print("Error:", e)
            return None


def db_convert(db):
    try:
        def convert_all_tables_to_csv(server, db):
            sql_server_connection_string = 'DRIVER={SQL Server};SERVER=' + server + ';DATABASE=' + db + ';Trusted_Connection=yes;'
            conn = pyodbc.connect(sql_server_connection_string)
            cursor = conn.cursor()
            tables = cursor.tables(tableType='TABLE')
            table_names = [table.table_name for table in tables]
            upload_dir = 'uploads'
            os.makedirs(upload_dir, exist_ok=True)  # Create upload directory if it doesn't exist
            for table_name in table_names:
                query = f'SELECT * FROM {table_name}'
                df = pd.read_sql_query(query, conn)
                csv_file_path = os.path.join(upload_dir, f'{table_name}.csv')  # Path to save CSV in upload directory
                df.to_csv(csv_file_path, index=False)
            conn.close()
        server = get_server_name()  # Assuming you have a function to get server name
        convert_all_tables_to_csv(server, db)
    except Exception as e:
        return None


# server=get_server_name()
get_sql_server_databases("DESKTOP-OA4UFIV")


```

6th Duplicates

```python
import pandas as pd

def binary_search(arr, x):
    """
    Perform binary search on a sorted list.

    Args:
    - arr: The sorted list to search in.
    - x: The value to search for.

    Returns:
    - index: The index of the value in the list if found, otherwise -1.
    """
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid
    return -1

def fx1(df, columns_to_check='all'):
    if columns_to_check == "all":
        duplicate_rows = df[df.duplicated(keep=False)]
    else:
        duplicate_rows = df[df.duplicated(subset=columns_to_check, keep=False)]

    return duplicate_rows

def fx2(df, columns_to_check='all'):
    if columns_to_check == "all":
        duplicate_rows = df[df.duplicated()].count().max()
        total_rows = df.count().max()
        pers = round((duplicate_rows / total_rows) * 100)
        return pers, duplicate_rows, total_rows
    else:
        duplicate_rows = df[df.duplicated(subset=columns_to_check)].count().max(
        total_rows = df.count().max()
        pers = round((duplicate_rows / total_rows) * 100)
        return pers, duplicate_rows, total_rows

def fx3(df):
    column_names = df.columns.tolist()
    return column_names

def fx4(input_filename, columns_to_check='all'):
    df = pd.read_csv('uploads/' + input_filename)
    if columns_to_check == "all":
        df_no_duplicates = df.drop_duplicates()
    else:
        df_no_duplicates = df.drop_duplicates(subset=columns_to_check)

    df_no_duplicates.to_csv('uploads/' + input_filename, index=False)

def remove_duplicates_and_save_binary(input_filename, columns_to_check='all'):
    df = pd.read_csv('uploads/' + input_filename)
    sorted_df = df.sort_values(by=columns_to_check)
    duplicate_indices = []
    for col in columns_to_check:
        sorted_col_values = sorted_df[col].tolist()
        for i in range(1, len(sorted_col_values)):
            if sorted_col_values[i] == sorted_col_values[i - 1]:
                idx = binary_search(sorted_col_values, sorted_col_values[i])
                duplicate_indices.append(idx)
    df_no_duplicates = df.drop(duplicate_indices)
    df_no_duplicates.to_csv('uploads/' + input_filename, index=False)

# filename = 'data.csv'
# df = pd.read_csv(filename)
# pers, effectedRows, totalRows = fx2(df, ['id', 'Name'])

# print(f"pers: {pers}")
# print(f"effectedRows: {effectedRows}")
# print(f"totalRows: {totalRows}")
# fx4(filename, ['id', 'Name'])
# print('#'*50)
# pers, effectedRows, totalRows = fx2(df, ['id', 'Name'])
```

```python
    # print(f"pers: {pers}")
    # print(f"effectedRows: {effectedRows}")
    # print(f"totalRows: {totalRows}")
    # fx4(filename, ['id', 'Name'])
    # print('#'*50)
    # pers, effectedRows, totalRows = fx2(df, ['id', 'Name'])

    # print(f"pers: {pers}")
    # print(f"effectedRows: {effectedRows}")
    # print(f"totalRows: {totalRows}")

    # remove_duplicates_and_save_binary(filename, ['id', 'Name'])

```

7th Missing values

```python
import pandas as pd
class Node:
    def __init__(self, data, dtype):
        self.data = data
        self.dtype = dtype
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data, dtype):
        new_node = Node(data, dtype)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

    def get_columns(self):
        columns = []
        current = self.head
        while current:
            columns.append(current.data)
            current = current.next
        return columns

    def get_dtype(self, data):
        current = self.head
        while current:
            if current.data == data:
                return current.dtype
            current = current.next
        return None
def extract_nan_columns_ll(linked_list, dtype):
    nan_columns_int = []
    nan_columns_str = []
    for column in linked_list.get_columns():
        if linked_list.get_dtype(column) == dtype and any(pd.isnull(column)):
            if dtype == 'float64':
                nan_columns_int.append(column)
            else:
                nan_columns_str.append(column)
    return nan_columns_str if dtype == 'str' else nan_columns_int
```

```python
# Create a linked list
ll = LinkedList()

# Append columns (data should be in form of pandas Series)
ll.append(pd.Series([1, 2, None, 4]), 'float64')
ll.append(pd.Series([None, 'b', 'c', 'd']), 'str')
ll.append(pd.Series([5.5, None, 6.7, None]), 'float64')

# Manage missing data
ll = (ll, 'mean', 'unknown')

# Print the processed linked list
current = ll.head
while current:
    print(current.data)
    current = current.next

```

8th Outliers

```python
import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.io as pio
import os
import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.io as pio
import os
import pandas as pd
import plotly.graph_objs as go
import plotly.io as pio
import cv2
import os

import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.io as pio
import os
import cv2
from queue import Queue

class DataQueue:
    def __init__(self):
        self.queue = Queue()

    def enqueue(self, item):
        self.queue.put(item)

    def dequeue(self):
        return self.queue.get()

    def is_empty(self):
        return self.queue.empty()

def fn1(col_name, df, operation):
    q = DataQueue()
    col_data = df[col_name]

def fn1(col_name, df, operation):
    q = DataQueue()
    col_data = df[col_name]

    if operation == 'op1':
        cc1, cc3 = np.percentile(col_data.dropna(), [25, 75])
        iqccr = cc3 - cc1
        lower_bound = cc1 - 1.5 * iqccr
        upper_bound = cc3 + 1.5 * iqccr
        for idx, val in col_data.iteritems():
            if val < lower_bound or val > upper_bound:
                q.enqueue(idx)
    elif operation == 'op2':
        for idx, val in col_data.iteritems():
            if val < 0:
                q.enqueue(idx)

    return q

def fn2(col_name, df, operation, replacement=None):
    q = fn1(col_name, df, operation)
    while not q.is_empty():
        idx = q.dequeue()
        if operation == 'rmv':
            df = df.drop(index=idx)
        elif operation == 'rplc':
            df.at[idx, col_name] = replacement
    return df

def fn3(column_name, df):
    return fn1(column_name, df, 'op1')

def fn4(column_name, df):
    return fn1(column_name, df, 'op2')
```

```python
def fn5(column_name, df, states='all'):
    total_rows = len(df)
    if states == 'neg':
        outlier_queue = fn4(column_name, df)
    else:
        outlier_queue = fn3(column_name, df)

    num_outliers = 0
    while not outlier_queue.is_empty():
        outlier_queue.dequeue()
        num_outliers += 1

    outlier_percentage = (num_outliers / total_rows) * 100
    return total_rows, num_outliers, round(outlier_percentage)

def fn6(column_name, df):
    outliers = fn3(column_name, df)
    total_rows, num_outliers, outlier_percentage = fn5(column_name, df)
    return outliers, total_rows, num_outliers, outlier_percentage

def fn7(column_name, df):
    negtiveOutliers = fn4(column_name, df)
    total_rows, num_outliers, outlier_percentage = fn5(column_name, df, 'neg')
    return negtiveOutliers, total_rows, num_outliers, outlier_percentage

def fn8(column_name, df, output_file):
    cleaned_df = fn2(column_name, df, 'rmv')
    cleaned_df.to_csv(output_file, index=False)
    return cleaned_df

def fn9(column_name, fileName):
    df = pd.read_csv(fileName)
    cleaned_df = fn2(column_name, df, 'rmv')
    if cleaned_df is not None:
        cleaned_df.to_csv(fileName, index=False)
    return cleaned_df
```

```python
def remove_negative_rows(column_name,fileName):
    df = pd.read_csv(fileName)
    column_numeric = pd.to_numeric(df[column_name], errors='coerce')

    cleaned_df = df[column_numeric >= 0]
    if cleaned_df is not None:
       cleaned_df.to_csv(fileName, index=False)
    return cleaned_df


def save_and_display_plot(df, column_name, plot_type):
    column_data = df[column_name]
    output_dir = "./img"

    if plot_type == 'Scatter_Plot':
        plot = go.Scatter(x=df.index, y=column_data, mode='markers', name='Data Points')
        layout = go.Layout(title=f"Scatter Plot of {column_name}",
                           xaxis=dict(title='Index'),
                           yaxis=dict(title=column_name))
    elif plot_type == 'Box_Plot':
        plot = go.Box(y=column_data, name='Data Distribution')
        layout = go.Layout(title=f"Box Plot of {column_name}", yaxis=dict(title=column_name))
    elif plot_type == 'Histogram':
        plot = go.Histogram(x=column_data, name='Data Distribution', nbinsx=20)
        layout = go.Layout(title=f"Histogram of {column_name}",
                           xaxis=dict(title=column_name),
                           yaxis=dict(title='Frequency'))
    elif plot_type == 'Line_Chart':
        plot = go.Scatter(x=df.index, y=column_data, mode='lines', name='Data Points')
        layout = go.Layout(title=f"Line Chart of {column_name}",
                           xaxis=dict(title='Index'),
                           yaxis=dict(title=column_name))
    else:
        print("Unknown plot type")
        return

    fig = go.Figure(data=[plot], layout=layout)

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
```

```python
def fn10(df, column_name, plot_type):
    column_data = df[column_name]
    output_dir = "./img"

    if plot_type == 'Scatter_Plot':
        plot = go.Scatter(x=df.index, y=column_data, mode='markers', name='Data Points'
        layout = go.Layout(title=f"Scatter Plot of {column_name}",
                           xaxis=dict(title='Index'),
                           yaxis=dict(title=column_name))
    elif plot_type == 'Box_Plot':
        plot = go.Box(y=column_data, name='Data Distribution')
        layout = go.Layout(title=f"Box Plot of {column_name}", yaxis=dict(title=column_
    elif plot_type == 'Histogram':
        plot = go.Histogram(x=column_data, name='Data Distribution', nbinsx=20)
        layout = go.Layout(title=f"Histogram of {column_name}",
                           xaxis=dict(title=column_name),
                           yaxis=dict(title='Frequency'))
    elif plot_type == 'Line_Chart':
        plot = go.Scatter(x=df.index, y=column_data, mode='lines', name='Data Points')
        layout = go.Layout(title=f"Line Chart of {column_name}",
                           xaxis=dict(title='Index'),
                           yaxis=dict(title=column_name))
    else:
        print("Unknown plot type")
        return

    fig = go.Figure(data=[plot], layout=layout)

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    output_file_name = os.path.join(output_dir, f"{column_name}_plot.png")
    pio.write_image(fig, output_file_name)

    saved_image = cv2.imread(output_file_name)
    cv2.imshow("Saved Image", saved_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```python
        output_file_name = os.path.join(output_dir, f"{column_name}_plot.png")
        pio.write_image(fig, output_file_name)


        saved_image = cv2.imread(output_file_name)
        cv2.imshow("Saved Image", saved_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

9th Run

```
1   from app import create_app
2
3   app = create_app()
4
5   if __name__ == '__main__':
6       app.run(debug=False, host='0.0.0.0', port=5000)
7
```

4th system integration

Flask Framework:

```
app=Flask(__name__)

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

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

Jinja

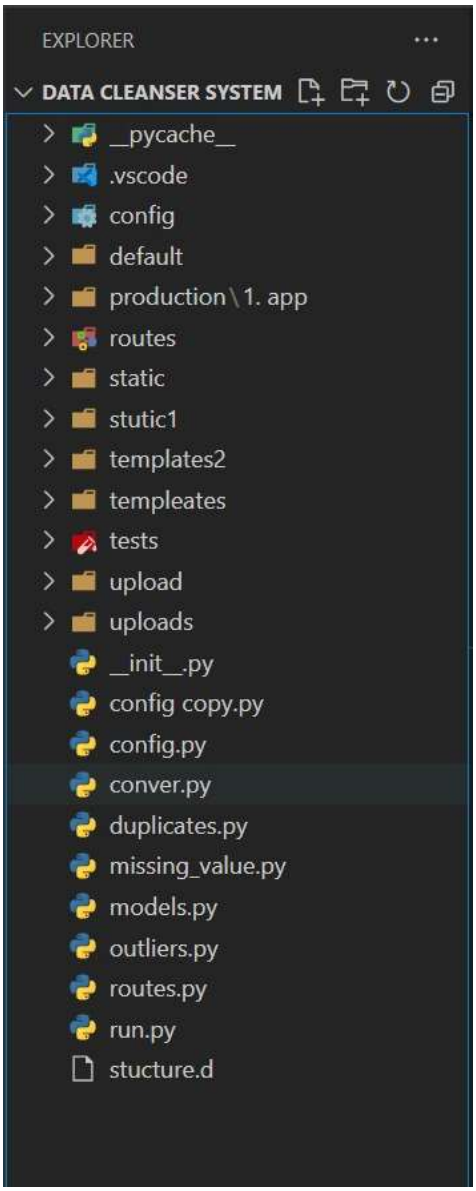
```
<select name="columns" class="form-control mt-3 fv py-5 Table-name " id="column-name" multiple>
  <option disabled value="">Select Which Column To Clean</option>
  <option value="all">all</option>

  {% for column in columns %}
  <option value={{column}}>{{ column }}</option>
  {% endfor %}

</select>
```

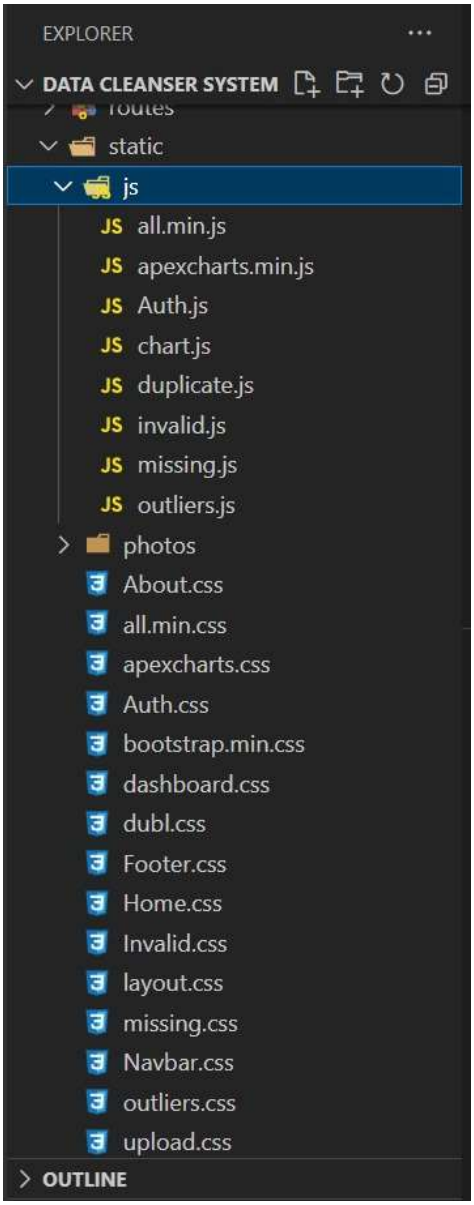
```
{% if table_original=="You haven't selected any Table!" %}
<p style="margin-top: 3%;" class="alert alert-danger mx-auto w-100" role="alert">{{table_original}}</p>

{% elif table_original %}
<div class="card">
  <div class="card-body" style="max-height: 400px; overflow-y: scroll;">
    {{ table_original|safe }}
  </div>
</div>

{% endif %}
```

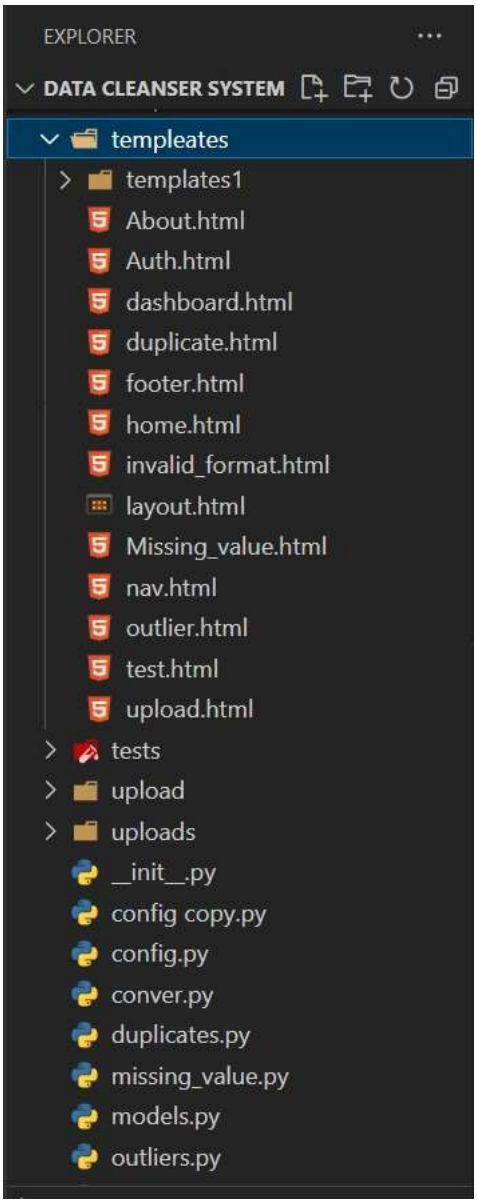
Template Rendering with Jinja:

```
{% extends "layout.html" %}
```

Flask Structure Files:







8.5 References:-

https://pmsoftlifecycle.blogspot.com/

https://jasonlemauk.com/the-agile-testers-mindset/

https://openrefine.org/

https://winpure.com/

https://www.validity.com/demandtools/

https://kn-it.com/

https://sagemaster.io/

https://www.blue-pencil.ca/data-cleansing-what-is-it-and-why-is-it-important/

https://blog.hubspot.com/marketing/data-cleansing

https://www.techtarget.com/searchdatamanagement/definition/data-scrubbing

https://careerfoundry.com/en/blog/data-analytics/best-data-cleaning-tools/

https://www.mrpeasy.com/blog/erp-system-implementation/

https://www.oracle.com/eg-ar/applications/what-is-saas/

https://jinja.palletsprojects.com/en/3.1.x/

https://plotly.com/

https://flask.palletsprojects.com/en/3.0.x/

https://getbootstrap.com/

https://www.tibco.com/glossary/what-is-data-cleansing

https://aws.amazon.com/what-is/data-cleansing/

https://www.kaggle.com/code/rtatman/data-cleaning-challenge-handling-missing-values

https://www.kaggle.com/code/rtatman/data-cleaning-challenge-deduplication

https://www.kaggle.com/code/rtatman/data-cleaning-challenge-imputing-missing-values

https://www.kaggle.com/code/rtatman/data-cleaning-challenge-cleaning-numeric-columns

https://www.kaggle.com/code/rtatman/data-cleaning-challenge-json-txt-and-xls

https://easygui.sourceforge.net/

https://community.fabric.microsoft.com/t5/Desktop/Simple-data-cleaning-split-columns-that-contain-multiple-values/td-p/3434064

https://stats.stackexchange.com/questions/546421/do-i-split-before-or-after-cleaning-code

https://towardsdatascience.com/how-to-clean-your-data-in-python-8f178638b98d?gi=d523f5fe8029

https://realpython.com/python-data-cleaning-numpy-pandas/

https://www.techtarget.com/searchdatamanagement/definition/data-scrubbing

https://blog.hubspot.com/marketing/data-cleansing

https://www.cazoomi.com/blog/dirty-data-how-much-it-can-cost-your-business-and-how-to-get-rid-of-it/

https://zenodo.org/records/1171077

https://softwarerecs.stackexchange.com/questions/11853/outlier-detection-in-latitude-longitude-csv-file

https://www.inciter.io/blog-posts/data-cleaning-in-excel-101-part-2-how-to-split-columns-in-excel

https://medium.com/@gregory2/4-easy-ways-to-handle-outliers-in-your-data-47f125a3f779

https://www2.microstrategy.com/producthelp/Current/MSTRWeb/WebHelp/Lang_1033/Content/Functions_for_Wrangling_Your_Data.htm

https://towardsdatascience.com/how-to-clean-your-data-in-python-8f178638b98d

https://www.dbmaestro.com/blog/database-automation/database-containerization