AUTOMATED EXPLORATION DATA ANALYSIS TOOL

BY

KOH YU BIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JANUARY 2025

AUTOMATED DATA EXPLORATION ANALYSIS TOOL

BY

KOH YU BIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JANUARY 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr Tong Dong Ling who has given me this bright opportunity to engage in a data analysis project. It is my first step to establish a career in data analysis field. A million thanks to you.

To a very special person in my life, Tan Yung Hui, for her patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

This project focuses on developing a web-based automated EDA tool This tool designed to simplify and streamline the EDA process. The primary objective is to automate the labour-intensive manual EDA procedures, eliminating the need for deep statistical or programming knowledge. The project addresses the limitations of existing tools, particularly for analyzing textual data by enhancing text data exploration capabilities. The system involves building a user-friendly interface using HTML for the front-end and Flask for back-end processing. Users can upload datasets and receive immediate visual and statistical analysis, including correlation heatmaps, bar charts, word clouds, and other visualizations tailored to numerical, categorical, and text data types. Text preprocessing features, such as tokenization and stop word removal are incorporated to handle textual data more effectively. The system will provide automatic insights based on dataset characteristics to reduce human error in data exploration. In summary, the tool can democratise data analysis by lowering the time and effort required for data preprocessing and visualisation while also making it available to a wider audience.

Area of Study: Data Science, Web Application Development

Keyword: Exploratory Data Analysis, Data Visualization, Text Analytics, Flask, Automated EDA, Automated Insights

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 Introduction

The core goal of this project is to develop a web-based automated EDA tool. The tool will automate the entire EDA process. In this chapter, we present the problem statement and motivation of our project, project scope and direction, objectives, contribution and background.

1.1 Background Information

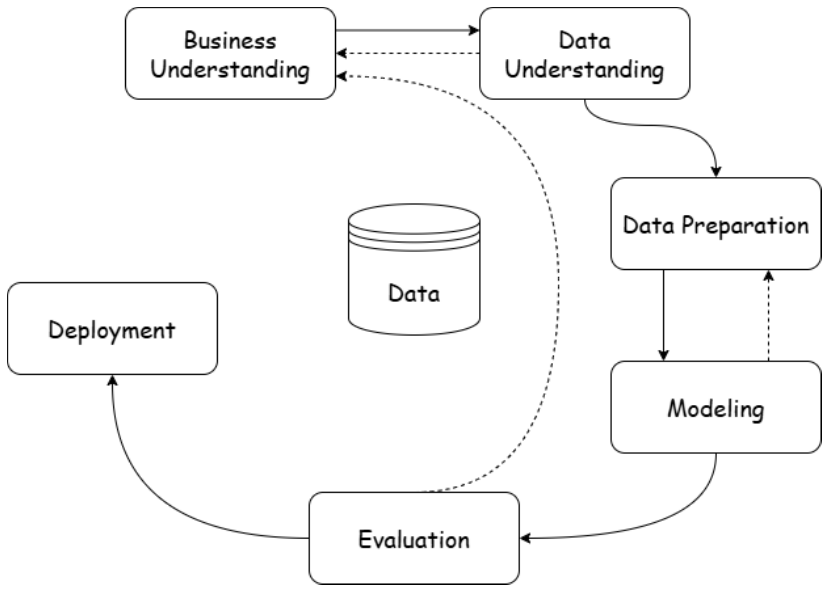


Figure 1-1 6 CRISP-DM Phase

CRISP-DM was published in 1999. It has since become the most common way for data mining, analytics and data science projects. Within the framework shown in Figure 1-1, EDA plays a crucial role in the data understanding phase. This phase starts with the data collection. When data material is successfully collected, it follows by data exploration. In this step, we need to understand the data and identify data quality problems. This step is to gain insights by querying and visualizing the data.

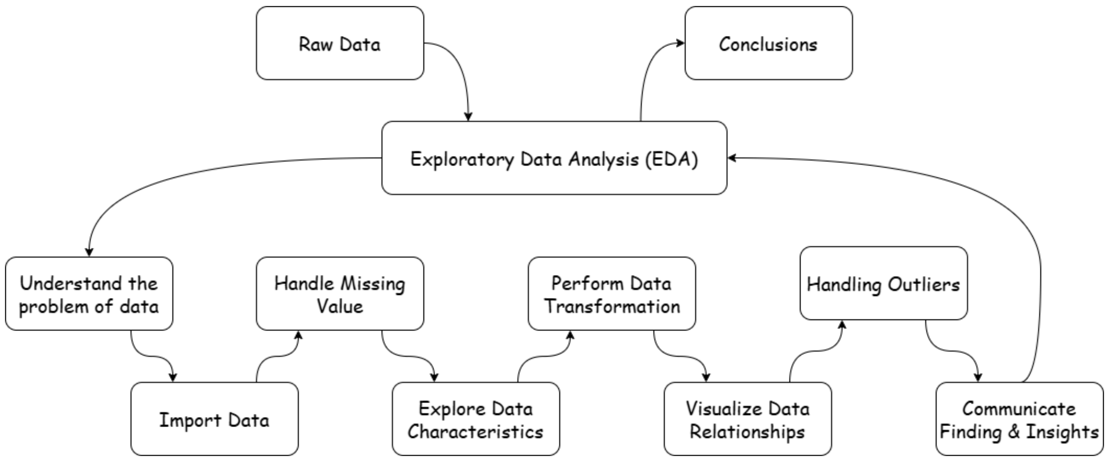

Raw Data
Conclusions
Exploratory Data Analysis (EDA)
Understand the problem of data
Handle Missing Value
Perform Data Transformation
Handling Outliers
Import Data
Explore Data Characteristics
Visualize Data Relationships
Communicate Finding & Insights

Figure 1-2 EDA Process Workflow

From the Figure 1-2, the first step in the EDA process is trying to understand the problem and the data. After clearly understanding the problem, the next step is to import data. The EDA process will then identify any missing values. If missing values are not handled correctly, it may cause misleading outcome. The next step is to explore data characteristics. This step will uncover the distribution of data points. For example, their range, central tendencies (mean, mode) and dispersion (variance, standard deviation). Summary statistics will be performed to calculate key statistics that provide insight into data trends. Correlation analysis was also performed to access the relationships between variables to understand how they affect each other. The next step is data transformation, which enables the preparation of statistics for similar evaluation and modelling. The following step is to visualize data relationships. The use of graphical representations like bar charts and box plots to visualize relationships in the data and distributions of variables. Afterwards, the EDA process will identify the outliers. Measurement or execution errors may be the cause. The last step is to communicate findings and insights. This includes summarising evaluation, highlighting discoveries and imparting outcomes cleanly.

EDA is an important step in the data analysis process because it helps in getting understanding data structures. We can understand the number of features, type of data in each feature and distribution of data points. It can identify patterns and relationships between variables by using visualizations and statistical summaries. EDA also can detect anomalies and outliers. The outliers may affect further analysis results. Furthermore, EDA can get insights to determine which features are most relevant to further analysis. EDA can improve data quality and integrity by detecting missing values and errors in the data.[2]

1.2 Problem Statement and Motivation

Although there are many EDA tools available, most of them focus on user interaction. Users require to **control the EDA process manually**. They often require users to manually adjust parameters, select variables, and choose the type of visualizations they wish to generate. Although the manual customization allows for greater flexibility, it also introduces the potential for errors and the risk of overlooking important information, such as outliers or trends. These seemingly small errors can lead to significant issues in further analysis. As a result, the manual nature makes the EDA process more labour-intensive and increases the likelihood of harming the quality of the analysis.[10]

Another problem for the existing EDA tools is their **limited ability to interpret text data**. Text data includes user comments, reviews, tweets, post and others. There is a wealth of information contained in these text data. However, most EDA tools only represent an output of a language model. They provide basic outputs such as word count, character length, word sequences and others. They do not represent the text directly. It is difficult for users to extract meaningful insights in text data. This makes a challenge to the further selection of models to analysis these text data.

In addition, existing EDA tools are **not generating summary at the end of EDA**. Users are usually required to understand and interpret the visualization and statistical output independently. It is difficult for users without professional knowledge to identify key relationships, trends, and potential data quality issues. Insights gained from EDA are important to determine which features are most relevant and should be included in further analysis. It also increases the likelihood of costly mistakes in further analysis. [8]

Existing EDA tools give users excessive customization. It will produce some errors that we cannot identify and ignore some important information. There is a lack of effective EDA tools to discover patterns and trends in text data. This project will be driven by reducing users' manual control of the entire EDA process to reduce users' unnecessary mistakes in manual EDA. This project will also aim to enhance EDA for text data.

1.3 Project Scope and Direction

The deliverable at the end of the project is a web-based automated EDA tool. This tool will enable users to perform EDA on their datasets with minimal interaction. The website extends the functionality of exploratory analysis of text data with frequency of word distribution visualization. The tool not only automates the EDA process but also generates insights at the end of EDA. The website will become a valuable nbresource for users in various fields, enabling them to discover important patterns, relationships, and potential data quality issues in their datasets without the need for technical expertise.

1.4 Research Objectives

The initiative that is being suggested aims to solve the previously mentioned issue. This will be achieved with the support of the project's three major objectives.

This first objective is **to automate the entire EDA process**. Users only need to upload a dataset and with just one click, the tool will present all relevant visualizations and all exploration results of the dataset. Users do not need to manually change parameters or variables to generate visualizations. This will simplify the EDA process and allows users to gain comprehensive insights in a short period of time.

Next, the second objective is **to enhance EDA to interpret text data.** The tool will expand the functionality of EDA for text data. It provides statistical analysis such as the length of text. In terms of visualization, it provides WordCloud and frequency distribution bar chart. These visualizations highlight frequently occurring words and vocabulary patterns in text data, making it easier for users to extract meaningful information from text data. The tool will incorporate sentiment analysis to detect and classify the emotional tone of text entries. This allows users to gain deeper insights into the sentiment trends present in textual data.

The third objective is **to generate summary at the end of EDA.** The tool will provide insights based on the results of the analysis. It will provide key findings from the dataset, alerting users to key aspects that should be focused on. For example, if the result shows a high correlation between variables, it will alert users to this relationship and suggest it as a point to focus on further. Similarly, if there is a large amount of missing data in variables, it will alert these issues, allowing users to address potential data quality issues.[8] This feature ensures that users are not only focus with raw data visualizations.

This project will not include complex machine learning model training or deployment. The focus will be on exploratory data analysis and visualization rather than predictive analysis or model development.

1.5 Contributions

EDA tools change the way individuals or organizations interact with data. In this era of increasing data-driven importance, our tool is more democratized and can be used by users of any technical background. By automating EDA, the website will present all relevant visualization and statistical analysis results at once, without the need for users to manually visualize and analyze data. Simpler and more accessible EDA will help users understand data to change decisions. Many existing EDA tools are either too complicated for non-experts or too simple to provide meaningful insights. Our tool will provide insights based on EDA results, allowing users to focus more on what is really important in the dataset. The ability to automatically detect and highlight key relationships, trends, and potential problems makes our tool unique.

Existing EDA tools focus mainly on numerical and categorical data, and text data analysis has not been fully developed and is limited to basic visualizations such as wordcloud. Our tool will perform statistical analysis on text data and provide bar chart of words and frequencies. These features will help users understand meaningful patterns and trends in text. With the proliferation of social media, text data example people's comments and some customer feedback on the media are growing exponentially. Our tool will help users discover potential trends from this huge amount of data.

CHAPTER 2 Literature Reviews

In this chapter, we will review four existing EDA tools. There are Datawrapper, Looker Studio, Tableau, and Orange. DataWrapper is a web-based tool designed for creating visualizations, including charts, maps and tables. Looker Studio known as Google Data Studio is a powerful data visualization tool developed by Google. Tableau is business intelligence and data visualization tool that helps users analyse and visualize data. Orange is an open-source data mining and machine learning tool that combines EDA.

2.1 Strength of Existing Tool (Tableau, Looker Studio, Datawrapper, Orange)

2.1.1 Chart Type used in Visualization

Looker Studio and Tableau both provide a wide range of visualization options. Looker Studio includes bar charts, time series, tree maps, geographic maps and tables. It not only has its own visualization options, it also embeds community visualization charts. This community visualization chart is a component built by third-party developers, which contains many visualization options of different types and functions.[4] Tableau supports traditional chart types like bar, line, and pie charts, as well as more advanced options such as Gantt charts, histograms, bullet charts and various types of maps. These features make Tableau strong for interactive visual exploration.[3]

In comparison, Datawrapper and Orange focus on more basic chart types. Datawrapper provides bar charts, line charts, pie charts, scatter plots and area charts, which are easy to create and ideal for quick visual representations. Orange also offers fewer visualization types including scatter plots, box plots, heatmaps and tree maps, with a strong focus on integrating these visualizations into data analysis workflows.[6]

We have to use different visualisations depending on the kind of variable we wish to see. Histograms and box plots are popular tools for graphically summarising quantitative data. Because they show information regarding lowest and maximum values, central location, and spread all at once, these plots are helpful. Furthermore, histograms can highlight trends that may influence an analysis. Each category variable's frequency of values can be immediately seen using a bar chart.[2]

2.1.2 Customization Flexibility on Chart

All four tools include extensive customization options, allowing users to change colors, labels, and formatting without requiring coding knowledge. Users can highlight key features of data by using custom colors, labels, and formatting, making it easier to read.[3],[4] For example, using different colors to represent distinct categories might assist consumers in easily identifying between different data groups, enhancing the visualization's readability. Besides that, customization allows users to highlight the most essential data points by changing font size, label position, and color intensity. This emphasis directs attention to critical insights. Customized visuals are more attractive and understandable, making it easier to explain findings to stakeholders who do not have technical expertise. Clear labels and well-chosen colors can help express ideas more effectively, preventing information from getting lost in complex data.

In comparison to Tableau, Orange offers less customization choices. However, it still includes the option to change colors, axis labels, and plot types.[6] In contrast, Datawrapper is designed to be clean and fast, with static visualizations, and all parameters must be configured before creating a chart. It lacks the interactivity and customization that Tableau and Looker Studio provide.

2.1.3 Data filtering

Looker Studio and Tableau are both highly interactive dashboards. Its features allow users to filter data, adjust date ranges, and drill down to specific details in visualizations, making it easier to discover insights.[3] Orange isolates filtered data before creating visualizations. The created visualization will change based on changes in the connected data.[6]

Filtering enables users to isolate specific subsets of data that are most relevant to their analysis. For example, if users are only interested in sales data for a specific region, they can filter out all other regions. After reducing noise, users are able to quickly identify trends, patterns, or outliers that may be obscured in a larger data set. Date range adjustment allows users to analyze data for a specific time period. This is essential for identifying trends, seasonality, or changes over time. For example, comparing data from different quarters or years can reveal growth trends, highlight periods of peaks or troughs in activity, and help predict future performance.

2.1.4 Feature Classification

Automatically classifying features into numerical, categorical, and other types allows users to easily understand a dataset's structure. This classification enables users to focus on the most appropriate analysis techniques for each feature type, such as summary statistics for numerical data or frequency distributions for categorical data.

Looker Studio automatically categorizes feature types using symbols such as "123" for numerical features and "ABC" for categorical features, allowing you to quickly distinguish between different types of data.[4] Similarly, Tableau and Orange divide features into dimensions (categorical data) and measures (numerical data), which simplifies data analysis. Datawrapper also automatically classifies feature types using a colored system: blue for numerical features, green for dates, and black for text data. However, in Datawrapper, this categorization is displayed within the dataset, requiring users to navigate through it to acquire a data summary.

2.2 Weakness of Existing Tool (Tableau, Looker Studio, Datawrapper, Orange)

2.2.1 Detecting Missing Data and Duplicate Data

Missing data can skew the results of your analysis. Missing values in key features might lead to inaccurate predictions and biased statistical results. Identifying missing values early allows you to determine the best strategy for addressing them, such as summarization, deletion, or other methods. Duplicate entries might influence your analysis by assigning more weight to specific observations, resulting in biased outcomes. For example, recording the same customer feedback many times may artificially inflate its perceived importance in the whole investigation.

Looker Studio and Tableau can visualize missing data using calculated fields and filters, but do not automatically summarize missing rates across the entire dataset. They do not have built-in functionality to automatically detect and report duplicate observations, but users can create custom queries to identify duplicate observations. Datawrapper does not have built-in functionality to directly assess missing data and does not support checking for duplicates or providing detailed data summaries. Orange has a widget for detecting and visualizing missing data, providing insight into overall missing rates. Orange does not automatically identify duplicates but allows users to use filters or create workflows to detect them.

2.2.2 Correlation Analysis Limitations

Correlation analysis is used to determine whether and to what extent variables are correlated. This is critical for understanding how one variable impact another, which can help guide future analysis or decisions. In machine learning, correlation analysis can be performed to determine which variables have a strong connection with the target variable. This is useful in feature selection, which may enhance model accuracy and prevent overfitting.

Looker Studio does not have built-in features for correlation analysis. In order to analyse correlations between variables, users must either generate custom formulas or employ calculated fields. This can be restrictive for unfamiliar users. Datawrapper is generally used to build simple and quick visualizations, however it lacks built-in features for correlation analysis. Orange includes a specific "Correlation" widget that enables users to easily conduct correlation analysis. It calculates the correlation between numerical variables and displays the findings in a heatmap, making them easy to understand.[6]

2.2.3 Challenges in EDA and Visualization for Text Data

Looker Studio, Tableau, and Datawrapper are mainly focused for numerical and categorical data visualisation and do not have built-in text analysis features. They provide very little support for text data exploration and are unable to analyse or visualise massive amounts of text. While it can show text data and perform simple operations such as word counts, it lacks features for generating word clouds, analysing text frequency, and creating other text-specific visualisations. Orange's Text Mining add-on includes some basic tools for exploring text data. The tool may do tasks such as word frequency analysis and preprocessing (tokenisation, stemming).[5]

2.2.4 Lack of Automated Summary Generation

Tableau and Looker Studio provide several chart types and interactive dashboards. Despite these advantages, they do not produce insights from data automatically. Users must manually analyse visualisations and statistical outputs to get insights, which is time-consuming and prone to human error, making it difficult for individuals without a thorough understanding of data analysis. Datawrapper cannot provide insights automatically. The program produces static visualisations with no automatic interpretation or data-driven suggestions. Users must conduct their own analysis to extract useful information from visualisations. Orange provides a variety of data mining and visualisation tools, including basic text mining capabilities. However, like other tools, it does not provide data insights automatically. Users must manually explore data using various widgets and workflows and evaluate the results themselves.

Users could overlook important patterns, trends, or correlations in data if automated generated insights are not available. This can lead to missed opportunities to make data-driven decisions that could improve corporate outcomes, optimise procedures, or identify new opportunities. Manual analysis requires users to understand complex visualisations and statistical findings on their own.[8] This raises the danger of human error, since users may misinterpret the data, overlook critical insights, or draw wrong inferences, resulting in poor decisions.

2.3 Table of Comparison (Tableau, Looker Studio, Datawrapper, Orange)

2.3.1 Comparison of Chart Type Used

Chart Type	**Datawrapper**	**Looker Studio**	**Tableau**	**Orange**
Bar Chart	√	√	√	√
Histogram			√	
Heatmap			√	√
Bubble		√	√	
Bullet Bars	√	√		

Tree Map		√	√	√
Waterfall		√		
Lines	√	√	√	√
Scorecard		√		
Area Chart	√	√	√	
Scatter Plot	√	√	√	√
Violin Plot				√
Geographical Chart		√	√	
Candlestick		√		
Election Donut	√	√		
Pie Chart	√	√	√	
Table	√	√	√	√
Donut Chart	√			
Time Series		√		
Partner Visualization		√		

Table 2-1 Comparison of Chart Type Used

2.3.2 Comparison of Features

Features	**Tableau**	**Looker Studio**	**Datawrapper**	**Orange**
Chart Type used in Visualization	Wide range, including bar, line, pie, Gantt, histograms, bullet charts, and maps	Wide range, including bar charts, time series, tree maps, geographic maps, tables, and community visualization charts.	Basic charts like bar, line, pie, scatter, and area charts.	Limited to scatter plots, box plots, heatmaps, and tree maps.
Customization Flexibility	Extensive; allows modifications to colors, fonts, data fields, and more.		Limited, static visualizations, parameters set before creation.	Limited, some customization in colors, axis labels, and plot types.
Filtering	Highly interactive, allows filtering, date range adjustments, and drill-down features.		No advanced filtering options,	Filtering is integrated into data workflows,

			focuses on static visualizations.	visualizations adapt to filtered data.
Features Classfication	Automatically categorizes features into dimensions (categorical) and measures (numerical).	Automatically categorizes features using symbols (e.g., "123" for numerical, "ABC" for categorical).	Uses colors to categorize blue for numerical, green for dates, black for text.	Categorizes features into dimensions and measures, uses widgets for easy analysis.
Missing Data & Duplicate Detection	No built-in automatic summarization of missing data or detection of duplicates, requires custom queries.		No built-in functionality for missing data or duplicates.	Widget for detecting and visualizing missing data, no automatic duplicate detection.
Correlation Analysis	No built-in features for correlation analysis, custom formulas or calculated fields needed.		No built-in features for correlation analysis.	Includes a "Correlation" widget for easy analysis, results displayed in a heatmap.
EDA & Visualization for Text Data	Primarily focused on numerical and categorical data, limited support for text data.			Basic text mining tools through add-ons, can perform word frequency analysis and preprocessing
Summary Generation	Does not automatically generate summary, users must manually interpret visualizations.			

Table 2-2 Comparison of Features

2.4 Discussion for proposed EDA tool

The proposed EDA tool will automatically summarise missing rates throughout the dataset. It will also discover and report duplicate observations, protecting data integrity by finding and dealing with duplicated entries. For numerical data analysis, the tool will create heatmaps with dendrograms to visualise correlations between variables, making it easier to recognise linkages and trends.

In terms of text data, the tool will generate word clouds and bar charts to show word frequencies, allowing users to easily understand prevalent themes and topics. It will also provide basic textual statistics, such as the minimum and maximum text length. Furthermore, the program will remove irrelevant variables before the EDA process begins, simplifying the analysis and focussing on data that is important.

Finally, the tool will offer summary based on visualisations and statistical analysis, allowing users to quickly understand critical findings.

CHAPTER 3 Proposed Method/Approach

In this chapter, we will discuss the methodology employed for the realization of the project. We will discuss the tools used, including the specific software and programming languages chosen, as well as the user requirements that guided the development process. Additionally, we will delve into the system design, addressing both the implementation strategies and the challenges encountered along the way. Finally, a detailed project timeline will be provided.

3.1 Design Specifications

3.1.1 System Methodologies

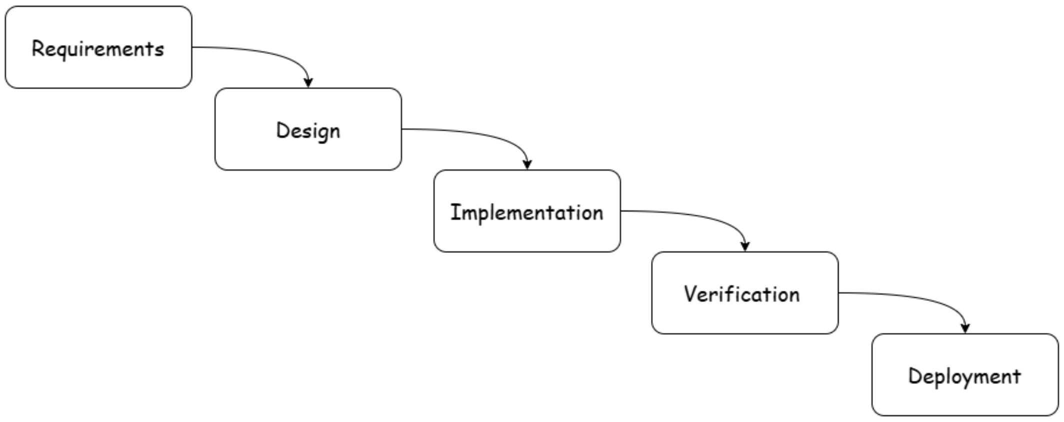


Figure 3-1 Waterfall Methodology

Requirements

During this phase, we outline the big picture of our project. Objectives of project and project scope will be outlined to state what the system should do. We discussed to the supervisor to verify project feasibility. If the project met all the necessary criteria, it would proceed to the next phase.

Design

In this phase, we have developed solutions that meet the requirements. We also determine the deliverables is web-based automated EDA tool. We design the system architecture and use cases in this phase.

Implementation

After design is finalized and approved, we proceeded to develop the system. We built the entire system and collect dataset needed for the development.

Verification

After completing the implementation, the system will test to ensure its quality and reliability. Various datasets that meet with the project requirements will be used to validate each function of the system. Comprehensive testing across different use cases is essential to confirm that all features work as intended and to identify and fix any bugs or issues that may arise during operation.

Deployment

In the deployment phase, the tool will be prepared for release and made accessible to users. The project will be published to GitHub. It allows others to access, review, and potentially contribute to the development.

3.1.2 Tools/Technologies to use

Platform

1. **Visual Studio Code**

   Visual Studio Code (VS Code) is a powerful open-source code editor created by Microsoft. It supports multiple programming languages and frameworks. VS Code is highly customisable with extensions. VS Code was chosen for this project because of its adaptability and compatibility for multiple languages and frameworks, including Python and HTML.

2. **Jupyter Notebook**

   Jupyter Notebook is an open-source web tool that enables to create and share documents that include live code, equations, visualisations, and narrative text. Jupyter Notebook was chosen for this project because it allows code to be executed in real time and visualised immediately. It is an excellent tool for prototyping and iterating on data analysis projects.

Language

1. **HTML**

HTML (Hypertext Markup Language) is the standard language used to create and structure content on the web. It defines the layout and elements of web pages, including text, images, links, and forms. HTML is chosen for this project to design the front-end user interface of the EDA tool.

2. **Python**

Python is a high-level, general-purpose programming language. It includes many libraries for data analysis, machine learning, and web development. Python is the major language for this project because it provides excellent support for data analysis and visualisation through libraries such as Pandas, Matplotlib, and Seaborn. It also integrates effectively with Flask for backend development, making it the backbone of both the EDA and web applications.

Framework

1. **Flask and Jinja2**

Flask is a lightweight and flexible web framework based on Python. It is designed to be simple and flexible, enabling developers to develop web apps fast and easily. Flask was chosen for this project to develop the backend of the EDA tool because of its ease of use, scalability and fine-grained control over the application's structure. Flask comes integrated with Jinja2. Jinja2 is a templating engine that allows dynamic generation of HTML pages by embedding Python-like expressions into HTML. [9]

3.1.3 Requirements

User Requirements

1. User can upload their dataset.
2. User can click button to proceed the dataset and system will start EDA process.
3. User can cancel the uploaded dataset.

System Requirements

1. System should receive dataset uploaded by users.
2. System should store the dataset into local file storage.
3. System should provide an overview of the dataset.
4. System should detect and report missing data.
5. System should filter out non-meaningful variables such as IDs.
6. System should classify dataset features into numerical, categorical, textural types.
7. System should compute statistics for all features.
8. System should generate visual representations.
9. System should summary the analysis results.

3.1.4 System Performance Definition

- The system shall complete the EDA process within 1 minutes for datasets up to 30,000 rows and 25 columns to ensure efficient and timely analysis.
- The system shall accurately identify and summarize key dataset features, including missing values, duplicates, correlations, and text statistics, with an accuracy rate of at least 95%.

3.1.5 Verification Plan

Testing will be carried out on a variety of datasets to provide full system verification. First, purely numerical datasets will be used to verify the accuracy of summary statistics, relationships, and visualisations. Next, datasets with categorical variables will be examined to ensure that frequency distributions, bar charts and data classification are generated correctly. For text-heavy datasets, the main focus will be on ensuring that the system accurately analyses word frequency and text length, as well as producing useful visualisations such word clouds. Mixed datasets containing numerical, category, and text variables will be used to ensure that the system successfully detects and processes each data type, treating them accordingly based on their properties. The system's scalability and performance will be evaluated using big datasets (up to 30,000 rows and 25 columns) to confirm that it can manage enormous amounts of data without affecting execution speed. Datasets with intentional missing values and duplicates will be used to test the system's capacity to correctly identify, summarise, and report these issues, assuring data quality throughout the analysis process.

3.2 System Design

3.2.1 System Architecture

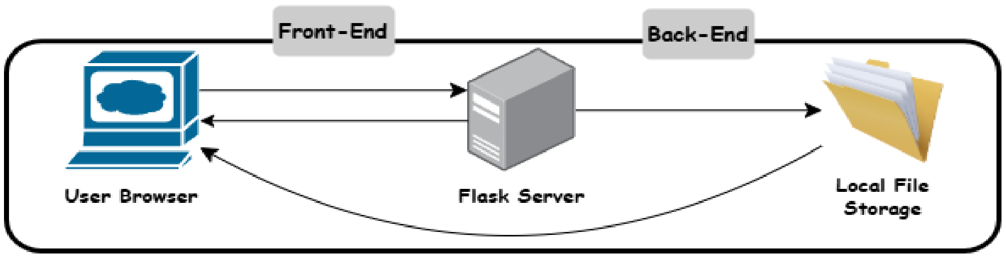


Figure 3-2 System Architecture

Figure 3-2 shows the architecture of the system. It uses a client-server model with Flask and Jinja2. The user interacts with the system through a web browser. When uploading a dataset, the browser sends the data with HTTP request to the Flask server. The Flask server processes the received dataset and saves it to the local file storage for further analysis.

Once the data is processed, the Flask backend prepares the results such as statistics, reports, and visualizations. These results are then passed to the front-end interface using Jinja2. By embedding the data directly into the templates, Jinja2 dynamically renders HTML pages. It allows users to view the processed information in a clear, interactive format.

3.2.2 Use Case Diagram

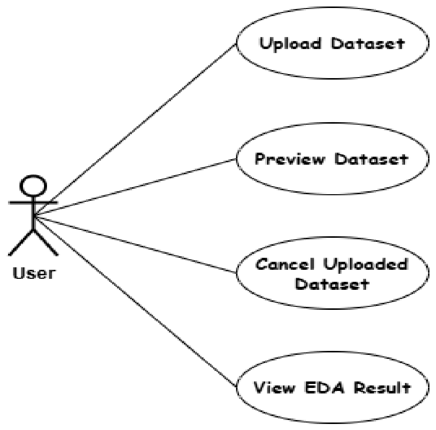


Figure 3-3 Use Case Diagram

The use case diagram for the EDA tool's user interface is displayed in Figure 3-3. The user will have the ability to provide the dataset they want to use for exploratory data analysis. The tool will automatically process the data after you submit the dataset. After that, the user will have access to all EDA results on the website.

3.2.3 Sequence Diagram

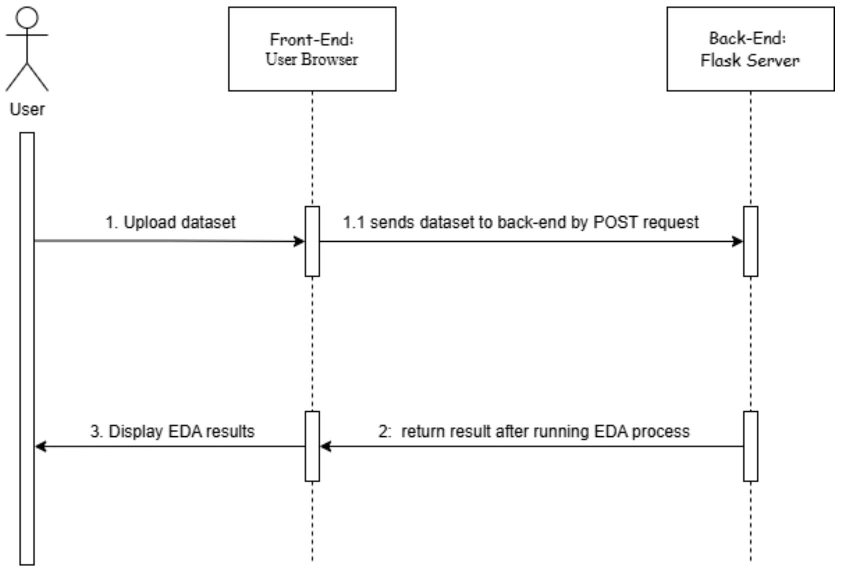


Figure 3-4 Sequence Diagram

The EDA tool's sequence diagram is displayed in Figure 3-4. The front end of the EDA tool, an HTML web page, will be used for user interaction. The dataset is uploaded by the user into an input box, and an HTTP POST request is used to send the dataset to the back end. Python code is used on the back end, which is driven by the Flask web application, to process the dataset. It processes the data through preset procedures in order to provide the EDA outputs. The user will then be moved to a new webpage where the results are shown after the result is submitted back to the front end of the website.

3.2.4 Activity Diagram

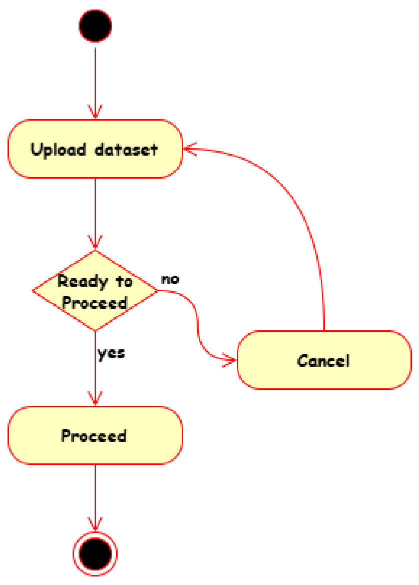


Figure 3-5 Upload Dataset Activity Diagram

The workflow begins when the **user uploads a dataset** through the interface. Once the dataset is successfully uploaded, the system waits for the user's next action.

At this point, the user is given two options:

- If the user is ready to begin EDA process, they can click the “Proceed” button, which initiates the Exploratory Data Analysis (EDA) process on the uploaded dataset.
- If the user decides not to proceed, they can click the “Cancel” button. This action will remove the uploaded dataset from the system and allowing them to start over.

After cancelling, the user can upload a new dataset if desired, repeating the process.

3.3 General Work Procedures

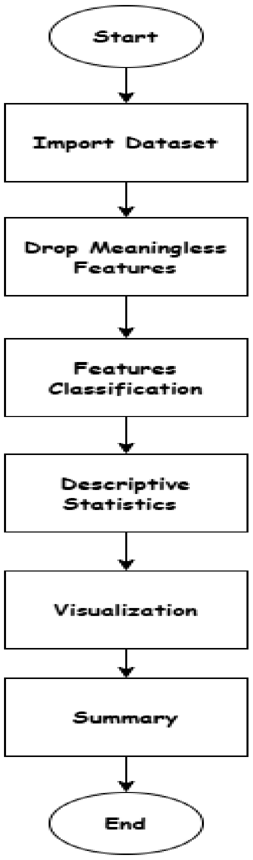


Figure 3-6 Automated EDA Web-based Tool Flowchart

Figure 3-6 shows general work procedures of the EDA tool. The process begins with importing the dataset into the system. Once the data is loaded, the tool will automatically remove meaningless features such as features with constant values or alphanumeric to ensure cleaner analysis.

Next, the tool proceeds to classify all features into appropriate types, such as numerical, categorical, or textual. After feature classification, the system performs descriptive statistical analysis tailored to each feature type. For numerical features, it computes metrics such as mean, median, standard deviation, and range. For categorical features, it calculates frequency distributions and proportions.

Then, the tool generates visualizations to help users better understand patterns and distributions in the data. These may include histograms, bar charts, boxplots, pie charts, and correlation heatmaps, depending on the nature of the features. Finally, the tool will generate summary at the end of EDA result.

This structured workflow ensures an automated exploratory analysis of any uploaded dataset.

3.4 Timeline

Timeline for FYP1

Task Name	Week											
	1	2	3	4	5	6	7	8	9	10	11	12
Project planning	■											
Define project background	■											
Understand encounter problems of the system	■											
Understand possible solutions of the system	■											
Finding existing systems and previous work	■											
Analysis of existing EDA tools		■										
Identify problem statements		■										
Identify motivation		■										
Identify project scope			■									
Identify project objectives			■									
Identify the impact, significance, and contribution of the project			■									
Define system specifications			■	■								
Determine the methodology used in this project			■	■								
Design system framework				■								
Determine technologies needed in this project				■								
Determine tools used in this project				■								
Design use case diagrams					■							
Design activity diagrams					■							
Preliminary work						■	■	■	■	■	■	
Evaluate preliminary work						■	■	■	■	■	■	
Final checking FYP Report 1												■

Timeline for FYP 2

Task Name	Week											
	1	2	3	4	5	6	7	8	9	10	11	12
Develop Front End	■	■	■	■	■	■	■	■	■	■	■	
Develop Back End	■	■	■	■	■	■	■	■	■	■	■	
Data Preparation						■	■	■	■			
Data Evaluation						■	■	■	■			
System Testing								■	■	■	■	
Final checking FYP 2 Report												■

CHAPTER 4 System Design

4.1 System Block Diagram

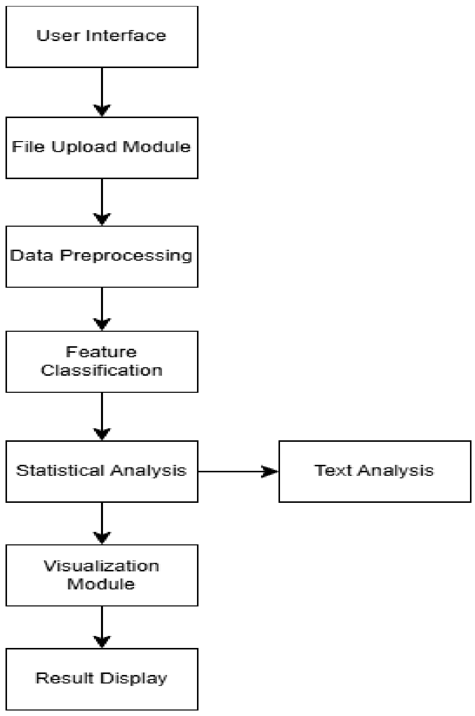


Figure 4-1 System Block Diagram

The system begins with the user interface. Users interact with the application to upload their dataset. Once a file is submitted, it enters the File Upload Module, which validates the file type. The uploaded file is then passed to the Data Preprocessing Module. In this module, initial cleaning operations are performed such as dropping meaningless features.

Following preprocessing, the dataset moves into the Feature Classification Module. The module automatically identifies and categorizes features as numerical, categorical or textural. This classification determines the type of analysis and visualization each feature will undergo. The processed and categorized data is then sent to the Statistical Analysis Module where descriptive statistics such as mean, median, standard deviation and frequency distributions are calculated. For text-based features, the data is forwarded to Text Analysis Module which handles tasks like tokenization, word frequency analysis and n-gram generation.

After analysis, the results are passed to the Visualization Module, which generates appropriate graphical representation including histograms, boxplots, bar charts, pie charts and heatmaps depending on the feature types and data patterns. Finally, all visual outputs and summary statistics are rendered through the Result Display Interface. This modular allows users to view and interpret the insights derived from their data.

4.2 System Component Workflow

4.2.1 User Interface

The UI is developed using HTML and CSS. It provides an intuitive front end for users to interact with the system including uploading datasets, initiating analysis and viewing results. The UI communicates with the backend through Flask routes and dynamically displays processed outputs and visualizations.

4.2.2 File Upload and Storage Workflow

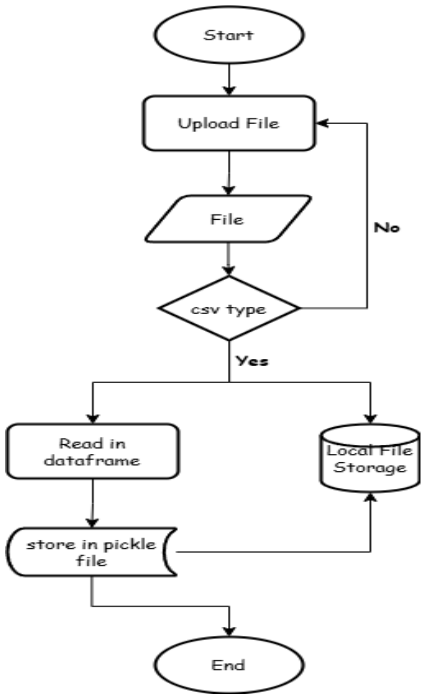


Figure 4-2 File Upload and Storage Workflow

The file upload and storage component manage user file submissions through the web interface. It validates the format of uploaded datasets and securely saves them to local storage. The system temporarily stores the processed dataset in local storage. This enables seamless reuse of processed datasets during a user session without repeated uploads.

4.2.3 Data Preprocessing Workflow

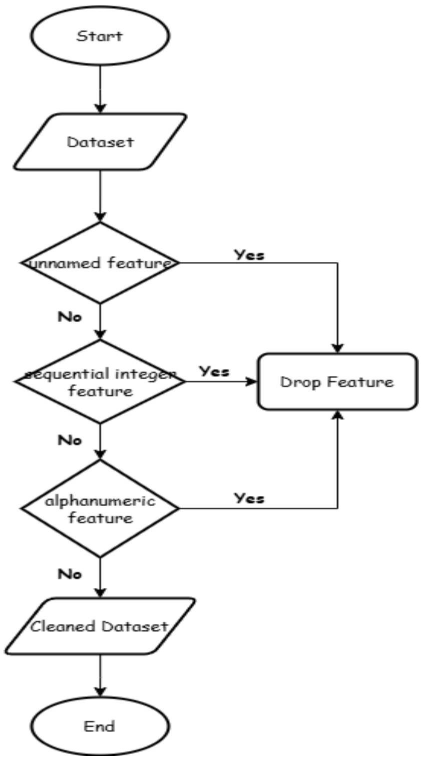


Figure 4-3 Data Preprocessing Workflow

The data preprocessing component ensures that the dataset is clean for analysis. It removes meaningless features, including unnamed columns, features containing sequential integers and alphanumeric strings that do not contribute useful information. This cleaning step improves the quality of analysis and ensures that only relevant data is passed to subsequent processing stages.

4.2.4 Feature Classification Workflow

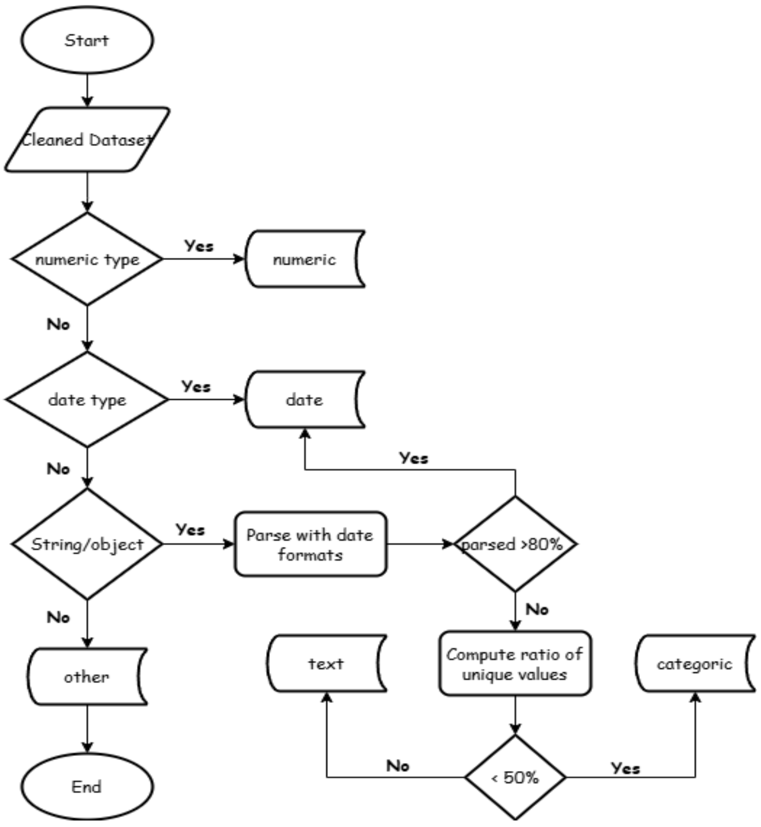


Figure 4-4 Feature Classification Workflow

The feature classification component is responsible for categorizing dataset features into appropriate types, numerical, categorical or textual. This is an important step, as the subsequent analysis and visualization depend on knowing the type of data. The classification process begins by checking if a feature is numeric or a recognized date format. Numeric features are stored in a numeric list, while valid date features are stored in a date list. For string-based features, the system attempts to parse them using predefined date formats. If more than 80% of the values can be successfully parsed, the feature is classified as a date. Otherwise, the system calculates the ratio of unique values in the feature. If the ratio exceeds 50%, it is treated as textual data. If not, it is considered categorical.

4.2.5 Statistical Analysis Workflow

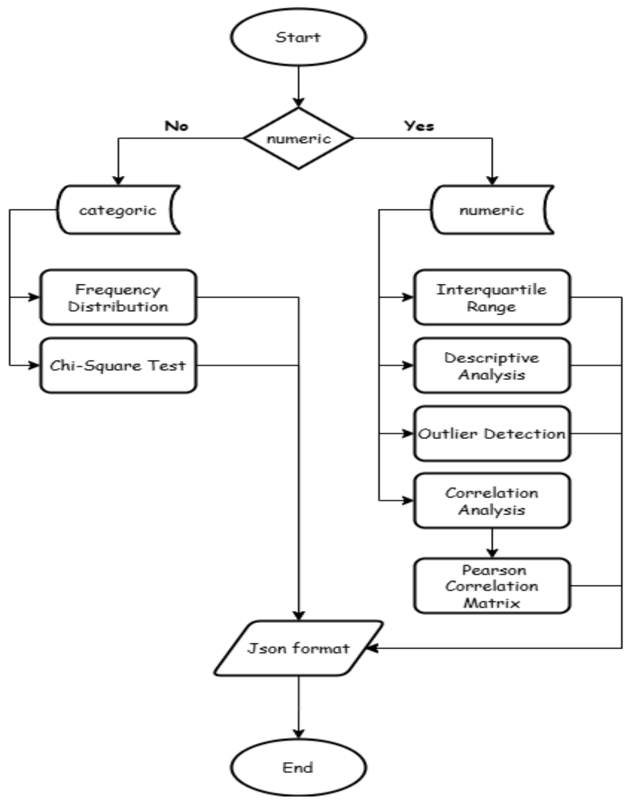


Figure 4-5 Statistical Analysis Workflow

This component performs descriptive analysis on the cleaned dataset. For numerical features, it calculates key statistical measures such as the mean, median, standard deviation, minimum, maximum, and interquartile range. It also includes outlier detection and generates a Pearson correlation matrix to identify relationships between numerical variables. For categorical features, it generates frequency tables and conducts chi-square tests to evaluate feature associations. All results are converted into JSON format for efficient use in other modules and frontend visualization.

4.2.6 Text Analysis Workflow

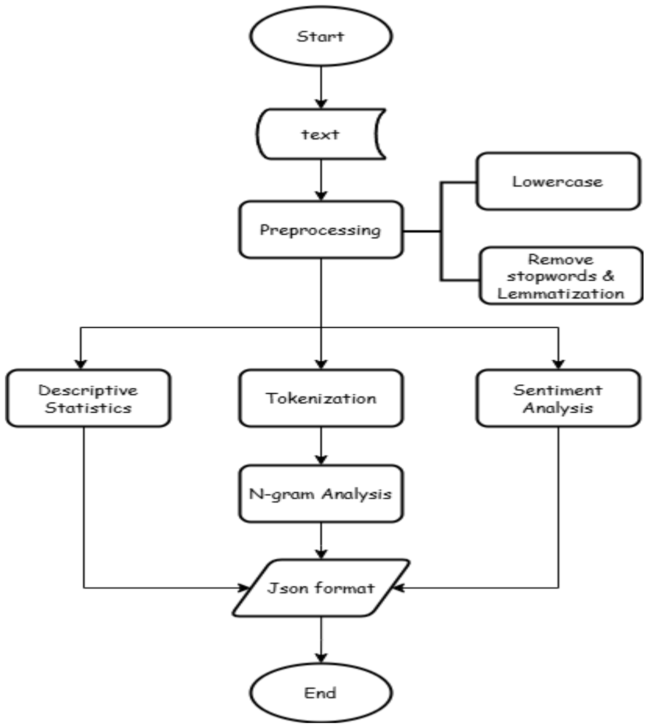


Figure 4-6 Text Analysis Workflow

For features containing textural data, the text analysis component processes natural language content. It begins with preprocessing steps such as converting text to lowercase, removing stop words and performing lemmatization to standardize the text. Once cleaned, the system computes descriptive statistics, including the minimum, maximum, and average sentence lengths. It also performs sentiment analysis to classify each observation as positive, negative, or neutral. Additionally, the pre-processed text is tokenized to support n-gram analysis (e.g., unigrams, bigrams, trigrams), which helps identify common word patterns. All results from the text analysis are converted into JSON format for efficient use in other modules and frontend visualization.

4.2.7 Visualization Generator Workflow

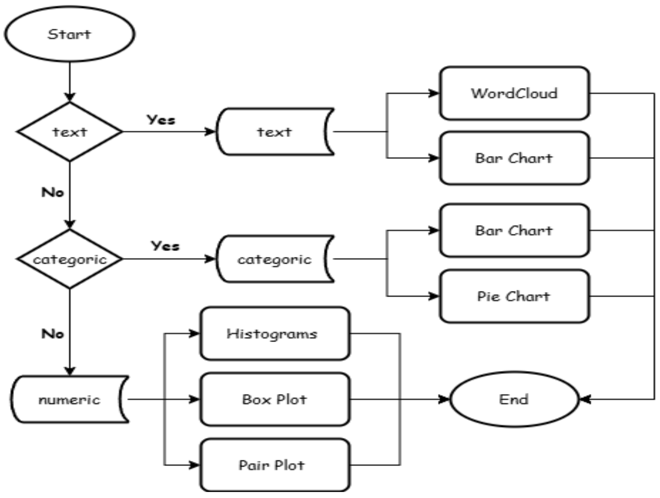
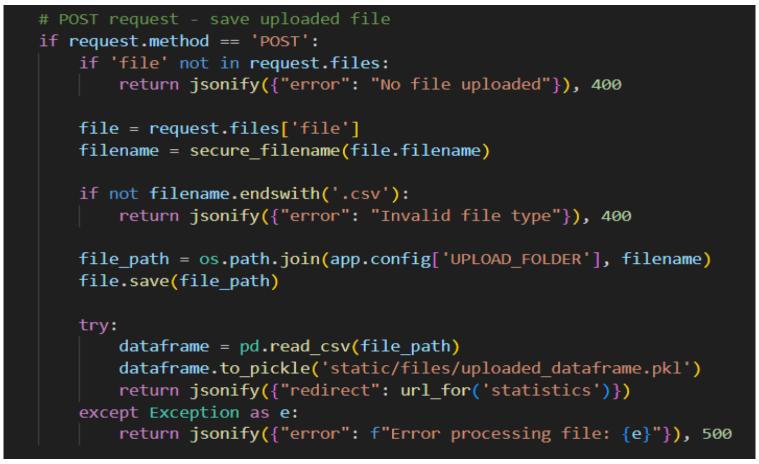


Figure 4-7 Visualization Workflow

The visualization module translates the statistical findings into graphical representations. Based on the type of data and the analysis performed, it dynamically generates appropriate visualizations. For numerical features, it creates histograms, box plots, and pair plots to show distributions, outliers and correlations. For categorical features, it generates bar charts and pie charts to display frequency distributions. For textual data, the module produces word clouds and bar charts to highlight common terms and patterns identified through n-gram analysis.

4.3 System Components Code Implementation

4.3.1 File Upload and Storage

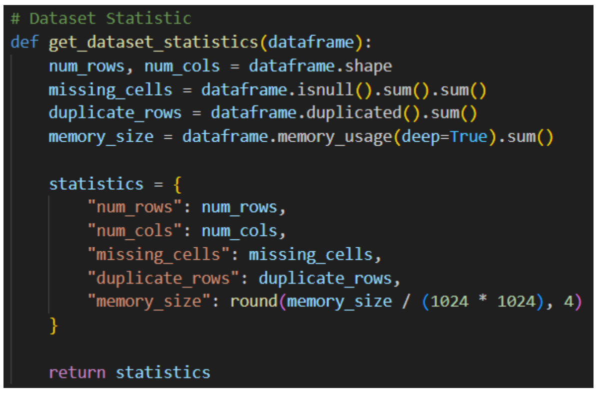


```
# POST request - save uploaded file
if request.method == 'POST':
    if 'file' not in request.files:
        return jsonify({"error": "No file uploaded"}), 400

    file = request.files['file']
    filename = secure_filename(file.filename)

    if not filename.endswith('.csv'):
        return jsonify({"error": "Invalid file type"}), 400

    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    file.save(file_path)

    try:
        dataframe = pd.read_csv(file_path)
        dataframe.to_pickle('static/files/uploaded_dataframe.pkl')
        return jsonify({"redirect": url_for('statistics')})
    except Exception as e:
        return jsonify({"error": f"Error processing file: {e}"}), 500
```

Figure 4-8 File Upload and Storage Code

The code implements the file upload and storage functionality within the system workflow using the Flask web framework. When a user submits a dataset through the frontend, the backend first verifies that the request is a POST request and that a file is included. The uploaded file's name is sanitized using Wekzeug's secure_filename function, and only files with a .csv extension are accepted. Once validated, the file is saved into a predefined local directory. The system then reads the uploaded CSV file using the Pandas library and serializes it into a Pickle file format.

4.3.2 Dataset Statistics Function

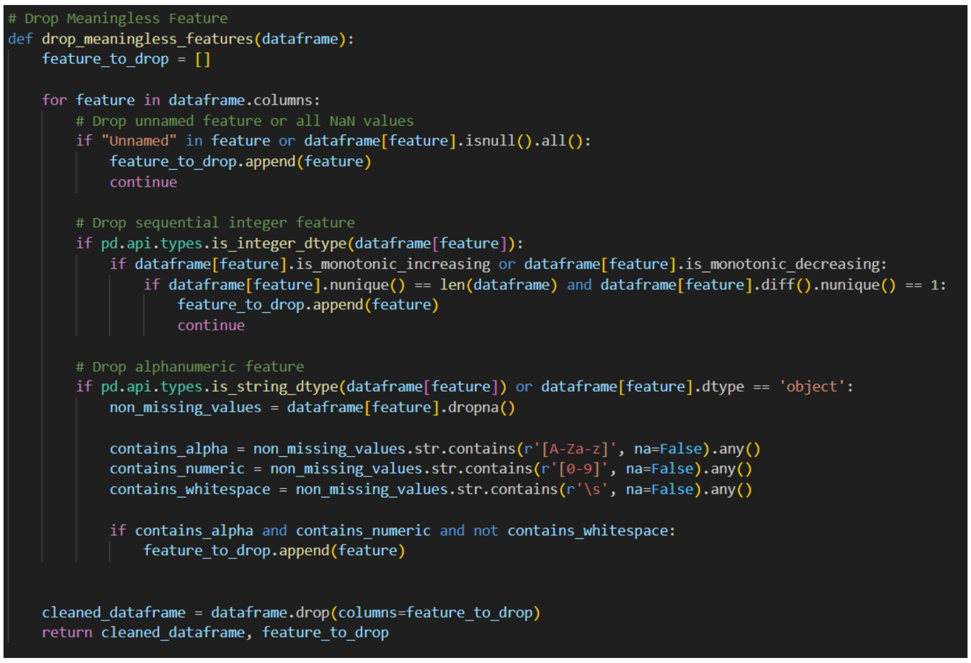


```python
# Dataset Statistic
def get_dataset_statistics(dataframe):
    num_rows, num_cols = dataframe.shape
    missing_cells = dataframe.isnull().sum().sum()
    duplicate_rows = dataframe.duplicated().sum()
    memory_size = dataframe.memory_usage(deep=True).sum()

    statistics = {
        "num_rows": num_rows,
        "num_cols": num_cols,
        "missing_cells": missing_cells,
        "duplicate_rows": duplicate_rows,
        "memory_size": round(memory_size / (1024 * 1024), 4)
    }

    return statistics
```

Figure 4-9 Dataset Statistics Function

The dataset statistics module provides a summary of the overall structure of the uploaded dataset. It calculates key metadata using the get_dataset_statistics() function. This includes the total number of rows and column of the dataset. It also counts the number of missing cells and the number of duplicate rows. Additionally, it computes the memory usage of the dataset in megabytes.

4.3.3 Data Preprocessing Function



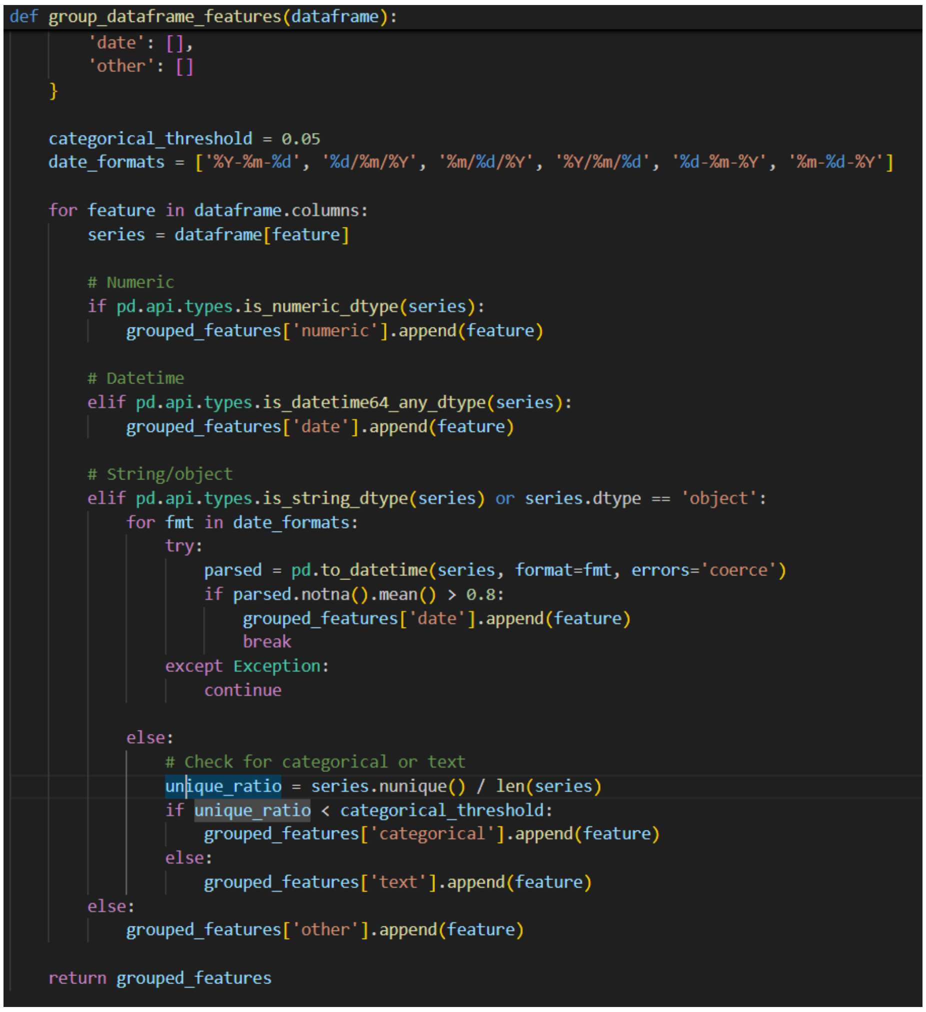
```python
# Drop Meaningless Feature
def drop_meaningless_features(dataframe):
    feature_to_drop = []

    for feature in dataframe.columns:
        # Drop unnamed feature or all NaN values
        if "Unnamed" in feature or dataframe[feature].isnull().all():
            feature_to_drop.append(feature)
            continue

        # Drop sequential integer feature
        if pd.api.types.is_integer_dtype(dataframe[feature]):
            if dataframe[feature].is_monotonic_increasing or dataframe[feature].is_monotonic_decreasing:
                if dataframe[feature].nunique() == len(dataframe) and dataframe[feature].diff().nunique() == 1:
                    feature_to_drop.append(feature)
                    continue

        # Drop alphanumeric feature
        if pd.api.types.is_string_dtype(dataframe[feature]) or dataframe[feature].dtype == 'object':
            non_missing_values = dataframe[feature].dropna()

            contains_alpha = non_missing_values.str.contains(r'[A-Za-z]', na=False).any()
            contains_numeric = non_missing_values.str.contains(r'[0-9]', na=False).any()
            contains_whitespace = non_missing_values.str.contains(r'\s', na=False).any()

            if contains_alpha and contains_numeric and not contains_whitespace:
                feature_to_drop.append(feature)


    cleaned_dataframe = dataframe.drop(columns=feature_to_drop)
    return cleaned_dataframe, feature_to_drop
```

Figure 4-10 Data Preprocessing Function

This function is designed to automatically clean a dataset by removing meaningless features. The function uses the Pandas library. The workflow starts by identifying features with names like “Unnamed” or those that contain only missing values. It then checks for sequential integer features such as row indices or counters by verifying if the feature values are strictly increasing or decreasing with uniform differences.

Next, it evaluates alphanumeric string features using regular expressions. Features containing both letters and numbers without whitespace are considered non-informative for data analysis. At the end of the process, the function returns the cleaned DataFrame and a list of the features that were removed.

4.3.4 Feature Classification Function



```python
def group_dataframe_features(dataframe):
        'date': [],
        'other': []
    }

    categorical_threshold = 0.05
    date_formats = ['%Y-%m-%d', '%d/%m/%Y', '%m/%d/%Y', '%Y/%m/%d', '%d-%m-%Y', '%m-%d-%Y']

    for feature in dataframe.columns:
        series = dataframe[feature]

        # Numeric
        if pd.api.types.is_numeric_dtype(series):
            grouped_features['numeric'].append(feature)

        # Datetime
        elif pd.api.types.is_datetime64_any_dtype(series):
            grouped_features['date'].append(feature)

        # String/object
        elif pd.api.types.is_string_dtype(series) or series.dtype == 'object':
            for fmt in date_formats:
                try:
                    parsed = pd.to_datetime(series, format=fmt, errors='coerce')
                    if parsed.notna().mean() > 0.8:
                        grouped_features['date'].append(feature)
                        break
                except Exception:
                    continue

            else:
                # Check for categorical or text
                unique_ratio = series.nunique() / len(series)
                if unique_ratio < categorical_threshold:
                    grouped_features['categorical'].append(feature)
                else:
                    grouped_features['text'].append(feature)
        else:
            grouped_features['other'].append(feature)

    return grouped_features
```

Figure 4-11 Feature Classification Function

This function is central of Feature Classification Workflow. It utilizes the Pandas library’s type checking utilities to sort features into predefined categories: numeric, date, categorical, text or other.

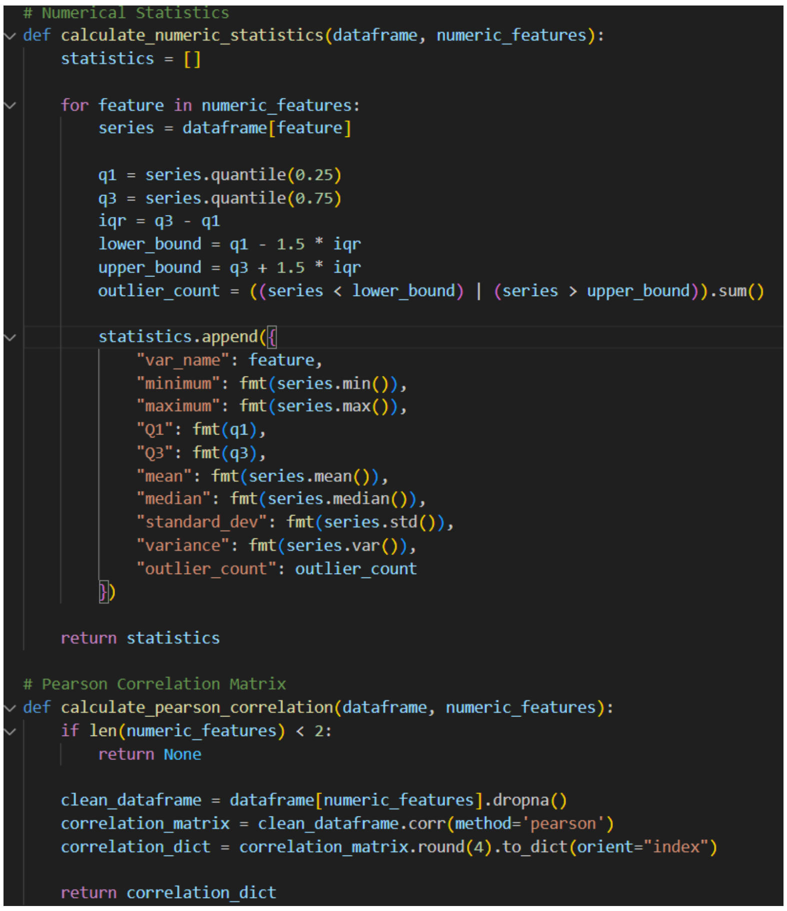
This function begins by initializing empty list for each type. It first checks whether each feature is numeric using is_numeric_dtype and adds it to the numeric list. For features recognized directly as datetime, it adds them to the date list.

String or object-type features go through a more detailed evaluation. The function attempts to parse these strings using multiple common date formats. If over 80% of the values can be successfully converted to dates, the feature is considered a date.

Otherwise, it calculates the unique value ratio. If the ratio is below a defined categorical threshold, the feature is treated as categorical. If higher, it is treated as textual data.

4.3.5 Statistical Analysis

4.3.5.1 Numeric Statistics Function

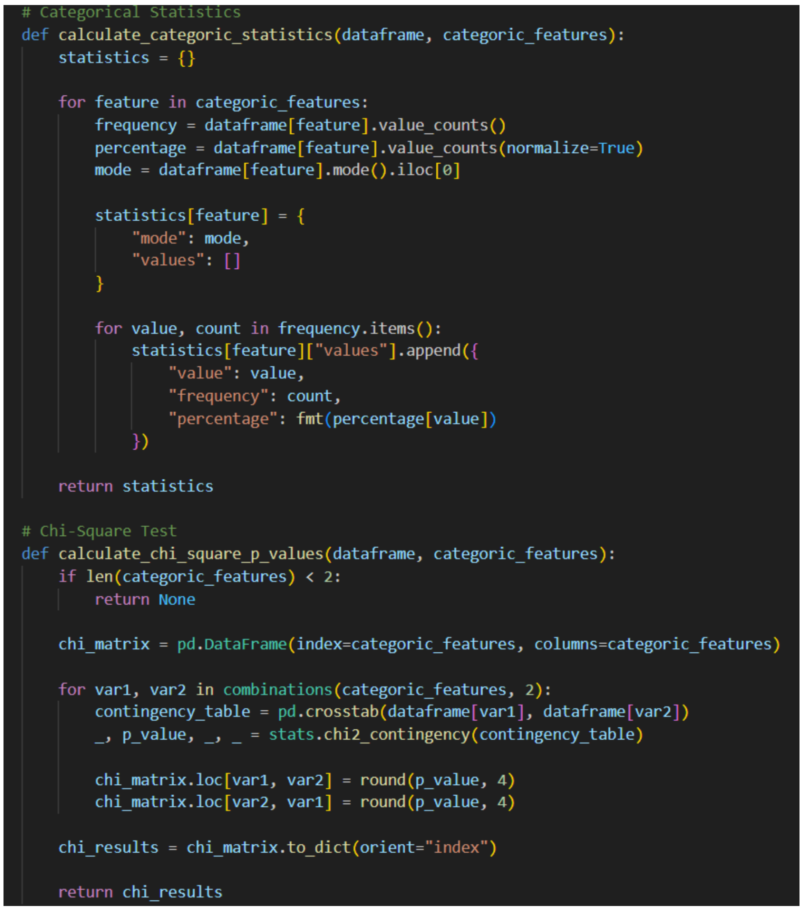


```python
# Numerical Statistics
def calculate_numeric_statistics(dataframe, numeric_features):
    statistics = []

    for feature in numeric_features:
        series = dataframe[feature]

        q1 = series.quantile(0.25)
        q3 = series.quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
        outlier_count = ((series < lower_bound) | (series > upper_bound)).sum()

        statistics.append({
            "var_name": feature,
            "minimum": fmt(series.min()),
            "maximum": fmt(series.max()),
            "Q1": fmt(q1),
            "Q3": fmt(q3),
            "mean": fmt(series.mean()),
            "median": fmt(series.median()),
            "standard_dev": fmt(series.std()),
            "variance": fmt(series.var()),
            "outlier_count": outlier_count
        })

    return statistics

# Pearson Correlation Matrix
def calculate_pearson_correlation(dataframe, numeric_features):
    if len(numeric_features) < 2:
        return None

    clean_dataframe = dataframe[numeric_features].dropna()
    correlation_matrix = clean_dataframe.corr(method='pearson')
    correlation_dict = correlation_matrix.round(4).to_dict(orient="index")

    return correlation_dict
```

Figure 4-12 Numeric Statistics Function

For numerical features, the system computes descriptive statistics such as the mean, median, standard deviation, variance, minimum, maximum and quartiles using the calculate_numeric_statistics function. It also detects outliers based on the interquartile range method. These metrics help summarize the central tendency, spread and variability of the data. Additionally, the calculate_pearson_correlation function generates a Pearson correlation matrix to identify linear relationships between numerical variables.

4.3.5.2 Categorical Statistics Function



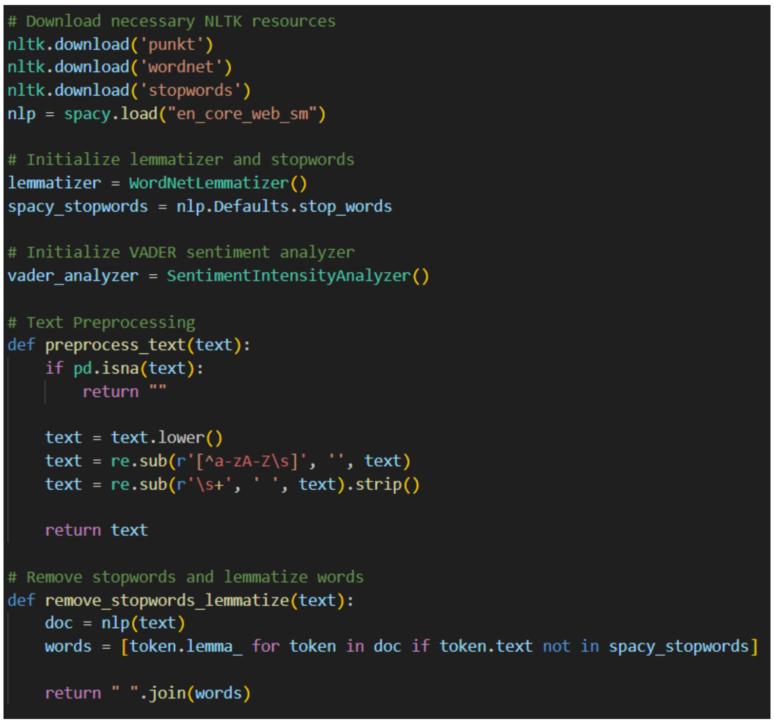
```python
# Categorical Statistics
def calculate_categoric_statistics(dataframe, categoric_features):
    statistics = {}

    for feature in categoric_features:
        frequency = dataframe[feature].value_counts()
        percentage = dataframe[feature].value_counts(normalize=True)
        mode = dataframe[feature].mode().iloc[0]

        statistics[feature] = {
            "mode": mode,
            "values": []
        }

        for value, count in frequency.items():
            statistics[feature]["values"].append({
                "value": value,
                "frequency": count,
                "percentage": fmt(percentage[value])
            })

    return statistics

# Chi-Square Test
def calculate_chi_square_p_values(dataframe, categoric_features):
    if len(categoric_features) < 2:
        return None

    chi_matrix = pd.DataFrame(index=categoric_features, columns=categoric_features)

    for var1, var2 in combinations(categoric_features, 2):
        contingency_table = pd.crosstab(dataframe[var1], dataframe[var2])
        _, p_value, _, _ = stats.chi2_contingency(contingency_table)

        chi_matrix.loc[var1, var2] = round(p_value, 4)
        chi_matrix.loc[var2, var1] = round(p_value, 4)

    chi_results = chi_matrix.to_dict(orient="index")

    return chi_results
```

Figure 4-13 Categorical Statistics Function

For categorical features, the calculate_categoric_statistics function computes the frequency and percentage distribution of each category along with identifying the mode. These statistics are essential for understanding the structure and imbalance in categorical data. Moreover, the calculate_chi_square_p_values function conducts a Chi-square test of independence to assess the statistical association between pairs of categorical variables. This helps identify whether two categorical variables are likely to be related.

4.3.6 Text Analysis

4.3.6.1 Text Preprocessing Function

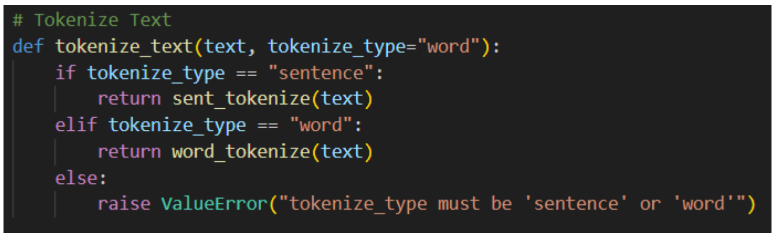


```python
# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
nlp = spacy.load("en_core_web_sm")

# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
spacy_stopwords = nlp.Defaults.stop_words

# Initialize VADER sentiment analyzer
vader_analyzer = SentimentIntensityAnalyzer()

# Text Preprocessing
def preprocess_text(text):
    if pd.isna(text):
        return ""

    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Remove stopwords and lemmatize words
def remove_stopwords_lemmatize(text):
    doc = nlp(text)
    words = [token.lemma_ for token in doc if token.text not in spacy_stopwords]

    return " ".join(words)
```

Figure 4-14 Text Preprocessing Function

The preprocessing phase involves cleaning raw text to prepare it for analysis. This includes converting all characters to lowercase, removing special characters and excessive whitespace using regular expressions, and eliminating stop words based on spaCy's default stop word list. Additionally, lemmatization is applied to reduce words to their base form using spaCy’s tokenizer and WordNetLemmatizer from NLTK. These steps help normalize the text, reduce noise, and improve the accuracy of subsequent analysis.

4.3.6.2 Tokenization Function

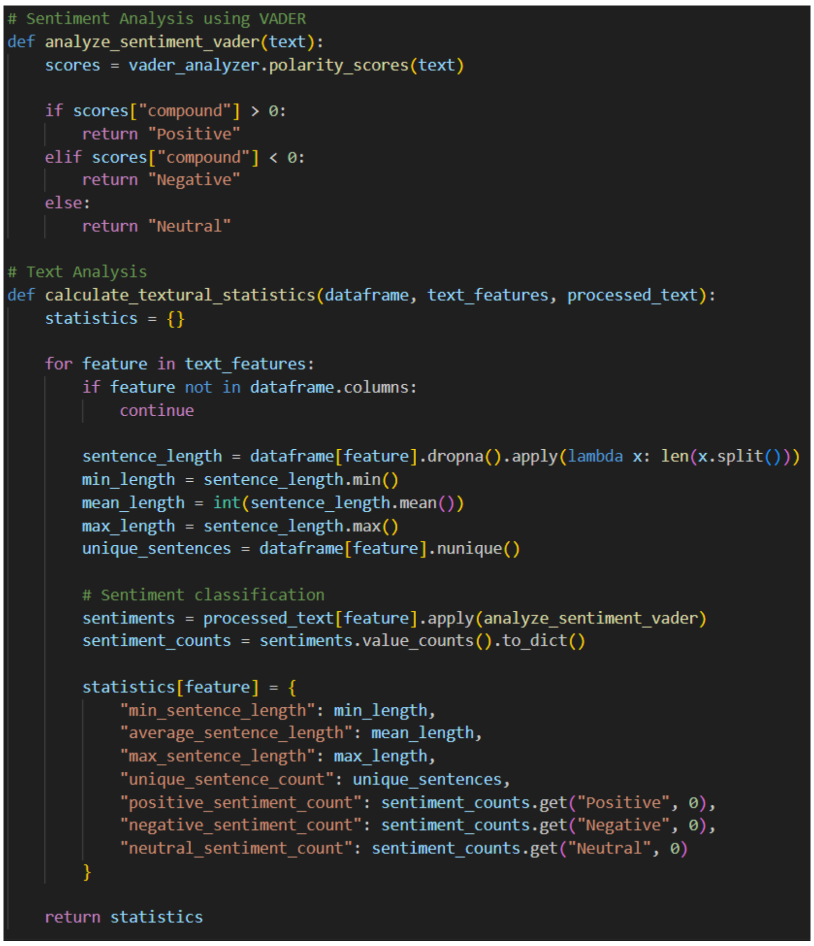


```python
# Tokenize Text
def tokenize_text(text, tokenize_type="word"):
    if tokenize_type == "sentence":
        return sent_tokenize(text)
    elif tokenize_type == "word":
        return word_tokenize(text)
    else:
        raise ValueError("tokenize_type must be 'sentence' or 'word'")
```

Figure 4-15 Tokenization Function

The system supports both word-level and sentence-level tokenization using NLTK's word_tokenize and sent_tokenize methods. Tokenization is essential for breaking down the text into manageable components for tasks like n-gram generation, frequency analysis and sentiment analysis.

4.3.6.3 Statistical Analysis and Sentiment Classification Function

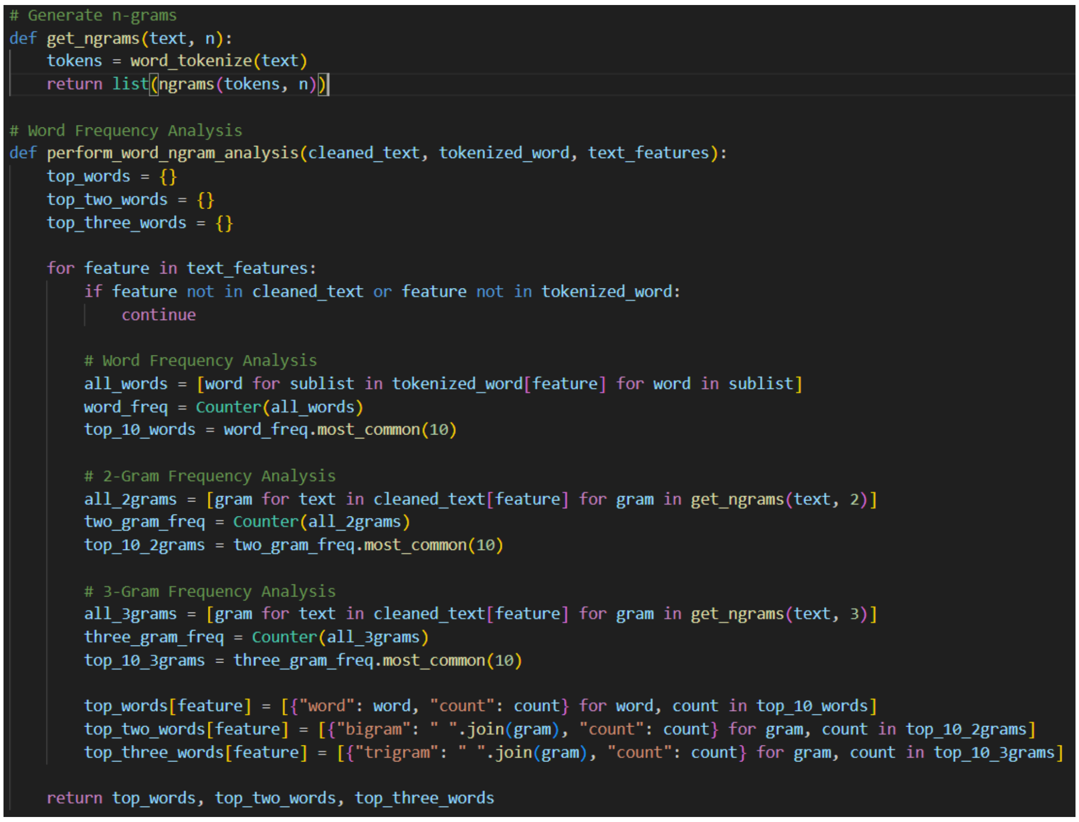


```python
# Sentiment Analysis using VADER
def analyze_sentiment_vader(text):
    scores = vader_analyzer.polarity_scores(text)

    if scores["compound"] > 0:
        return "Positive"
    elif scores["compound"] < 0:
        return "Negative"
    else:
        return "Neutral"

# Text Analysis
def calculate_textural_statistics(dataframe, text_features, processed_text):
    statistics = {}

    for feature in text_features:
        if feature not in dataframe.columns:
            continue

        sentence_length = dataframe[feature].dropna().apply(lambda x: len(x.split()))
        min_length = sentence_length.min()
        mean_length = int(sentence_length.mean())
        max_length = sentence_length.max()
        unique_sentences = dataframe[feature].nunique()

        # Sentiment classification
        sentiments = processed_text[feature].apply(analyze_sentiment_vader)
        sentiment_counts = sentiments.value_counts().to_dict()

        statistics[feature] = {
            "min_sentence_length": min_length,
            "average_sentence_length": mean_length,
            "max_sentence_length": max_length,
            "unique_sentence_count": unique_sentences,
            "positive_sentiment_count": sentiment_counts.get("Positive", 0),
            "negative_sentiment_count": sentiment_counts.get("Negative", 0),
            "neutral_sentiment_count": sentiment_counts.get("Neutral", 0)
        }

    return statistics
```

Figure 4-16 Statistical Analysis and Sentiment Function

The calculate_textural_statistics function calculates descriptive metrics for each text feature, including minimum, average and maximum sentence length, as well as the count of unique sentences. It also integrates the VADER sentiment analyzer. It classifies each sentence as Positive, Negative, or Neutral based on its compound sentiment score.

4.3.6.4 N-gram Analysis Function



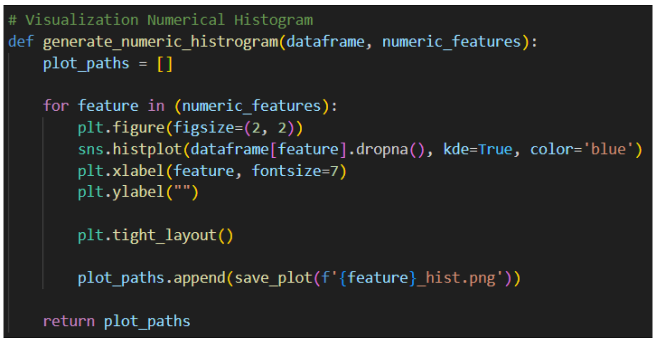
```python
# Generate n-grams
def get_ngrams(text, n):
    tokens = word_tokenize(text)
    return list(ngrams(tokens, n))

# Word Frequency Analysis
def perform_word_ngram_analysis(cleaned_text, tokenized_word, text_features):
    top_words = {}
    top_two_words = {}
    top_three_words = {}

    for feature in text_features:
        if feature not in cleaned_text or feature not in tokenized_word:
            continue

        # Word Frequency Analysis
        all_words = [word for sublist in tokenized_word[feature] for word in sublist]
        word_freq = Counter(all_words)
        top_10_words = word_freq.most_common(10)

        # 2-Gram Frequency Analysis
        all_2grams = [gram for text in cleaned_text[feature] for gram in get_ngrams(text, 2)]
        two_gram_freq = Counter(all_2grams)
        top_10_2grams = two_gram_freq.most_common(10)

        # 3-Gram Frequency Analysis
        all_3grams = [gram for text in cleaned_text[feature] for gram in get_ngrams(text, 3)]
        three_gram_freq = Counter(all_3grams)
        top_10_3grams = three_gram_freq.most_common(10)

        top_words[feature] = [{"word": word, "count": count} for word, count in top_10_words]
        top_two_words[feature] = [{"bigram": " ".join(gram), "count": count} for gram, count in top_10_2grams]
        top_three_words[feature] = [{"trigram": " ".join(gram), "count": count} for gram, count in top_10_3grams]

    return top_words, top_two_words, top_three_words
```

Figure 4-17 N-gram Analysis Function

The perform_word_ngram_analysis function conducts frequency analysis on single words (unigrams), two-word phrases (bigrams), and three-word phrases (trigrams). This is done using NLTK’s ngrams method and the top 10 most frequent n-grams are identified and returned for each feature. This analysis helps uncover recurring patterns and theme in the dataset.

4.3.7 Data Visualization

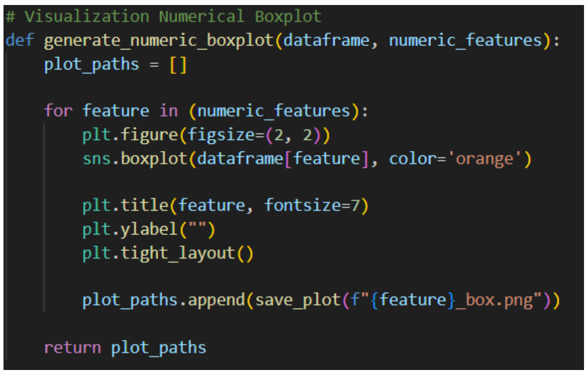
4.3.7.1 Numerical Feature Visualization Function



```python
# Visualization Numerical Histogram
def generate_numeric_histrogram(dataframe, numeric_features):
    plot_paths = []

    for feature in (numeric_features):
        plt.figure(figsize=(2, 2))
        sns.histplot(dataframe[feature].dropna(), kde=True, color='blue')
        plt.xlabel(feature, fontsize=7)
        plt.ylabel("")

        plt.tight_layout()

        plot_paths.append(save_plot(f'{feature}_hist.png'))

    return plot_paths
```

Figure 4-18 Histogram Function

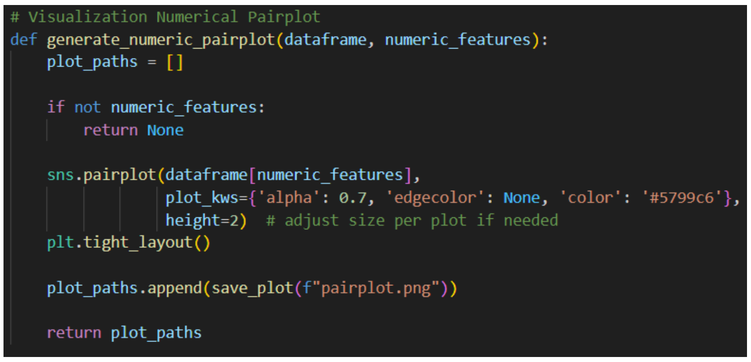
The generate_numeric_histrogram() function creates a histogram with a KDE overlay for each numeric feature.



```python
# Visualization Numerical Boxplot
def generate_numeric_boxplot(dataframe, numeric_features):
    plot_paths = []

    for feature in (numeric_features):
        plt.figure(figsize=(2, 2))
        sns.boxplot(dataframe[feature], color='orange')

        plt.title(feature, fontsize=7)
        plt.ylabel("")
        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_box.png"))

    return plot_paths
```

Figure 4-19 Boxplot Function

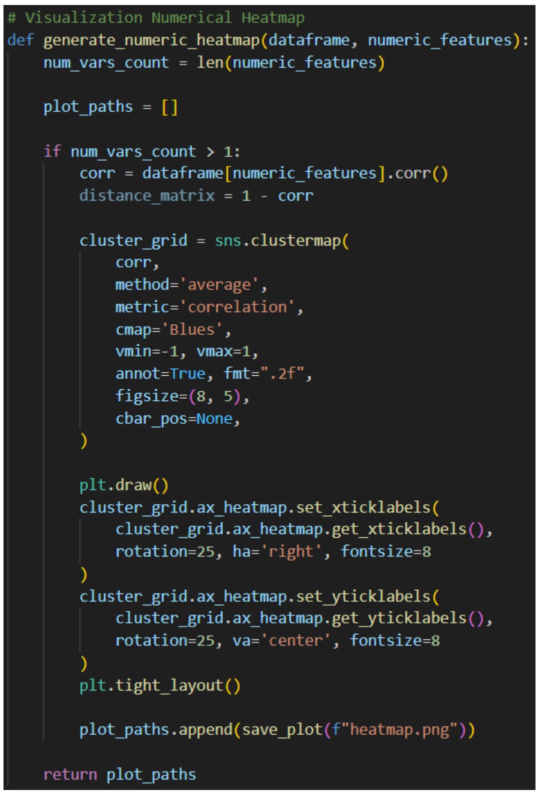
The generate_numeric_boxplot() function produces compact boxplots to identify outliers and understand the dispersion and central tendency of numerical features.



```python
# Visualization Numerical Pairplot
def generate_numeric_pairplot(dataframe, numeric_features):
    plot_paths = []

    if not numeric_features:
        return None

    sns.pairplot(dataframe[numeric_features],
                 plot_kws={'alpha': 0.7, 'edgecolor': None, 'color': '#5799c6'},
                 height=2)  # adjust size per plot if needed
    plt.tight_layout()

    plot_paths.append(save_plot(f"pairplot.png"))

    return plot_paths
```

Figure 4-20 Pair plot Function

The generate_numeric_pairplot() function displays pairwise scatter plots among all selected numeric features. This is especially useful for identifying correlations and clusters across multiple dimensions.

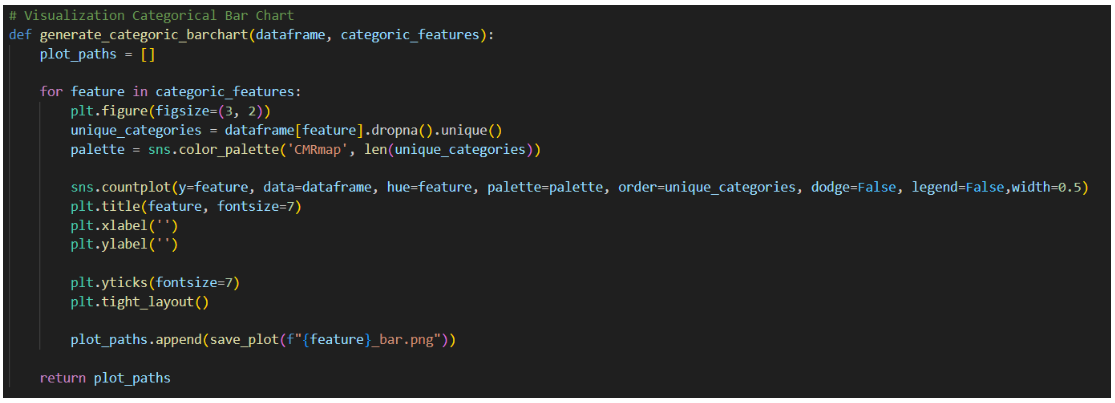


```python
# Visualization Numerical Heatmap
def generate_numeric_heatmap(dataframe, numeric_features):
    num_vars_count = len(numeric_features)

    plot_paths = []

    if num_vars_count > 1:
        corr = dataframe[numeric_features].corr()
        distance_matrix = 1 - corr

        cluster_grid = sns.clustermap(
            corr,
            method='average',
            metric='correlation',
            cmap='Blues',
            vmin=-1, vmax=1,
            annot=True, fmt=".2f",
            figsize=(8, 5),
            cbar_pos=None,
        )

        plt.draw()
        cluster_grid.ax_heatmap.set_xticklabels(
            cluster_grid.ax_heatmap.get_xticklabels(),
            rotation=25, ha='right', fontsize=8
        )
        cluster_grid.ax_heatmap.set_yticklabels(
            cluster_grid.ax_heatmap.get_yticklabels(),
            rotation=25, va='center', fontsize=8
        )
        plt.tight_layout()

        plot_paths.append(save_plot(f"heatmap.png"))

    return plot_paths
```

Figure 4-21 Heatmap Function

The generate_numeric_heatmap() function computes the correlation matrix and uses a hierarchical clustering heatmap to visualize inter-feature correlations.

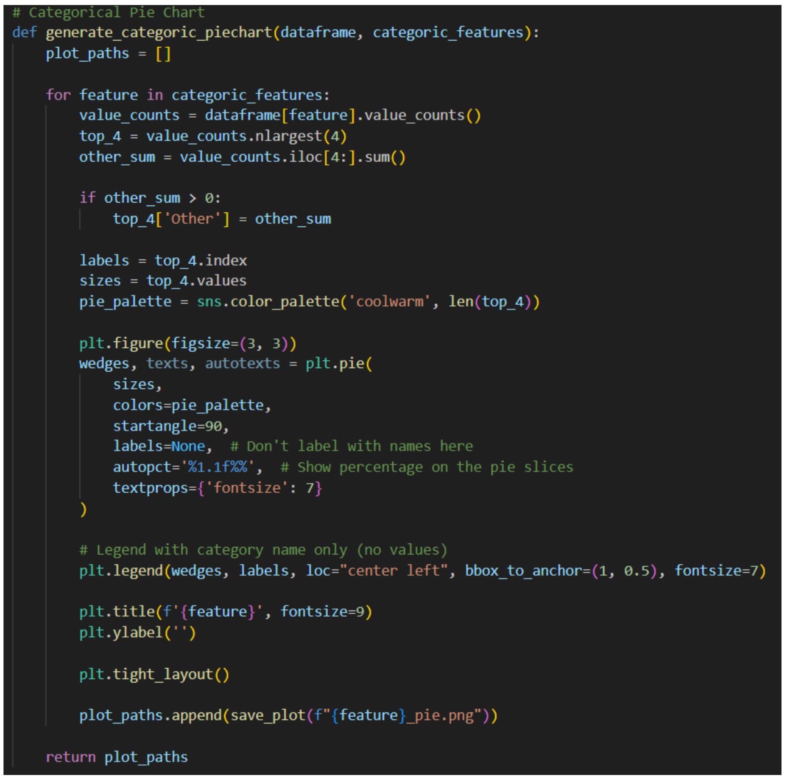
4.3.7.2 Categorical Feature Visualization Function



```python
# Visualization Categorical Bar Chart
def generate_categoric_barchart(dataframe, categoric_features):
    plot_paths = []

    for feature in categoric_features:
        plt.figure(figsize=(3, 2))
        unique_categories = dataframe[feature].dropna().unique()
        palette = sns.color_palette('CMRmap', len(unique_categories))

        sns.countplot(y=feature, data=dataframe, hue=feature, palette=palette, order=unique_categories, dodge=False, legend=False,width=0.5)
        plt.title(feature, fontsize=7)
        plt.xlabel('')
        plt.ylabel('')

        plt.yticks(fontsize=7)
        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_bar.png"))

    return plot_paths
```

Figure 4-22 Categoric Bar Chart Function

The generate_categoric_barchart() function creates horizontal bar charts showing the frequency of each category. It uses a colour palette to distinguish between categories.

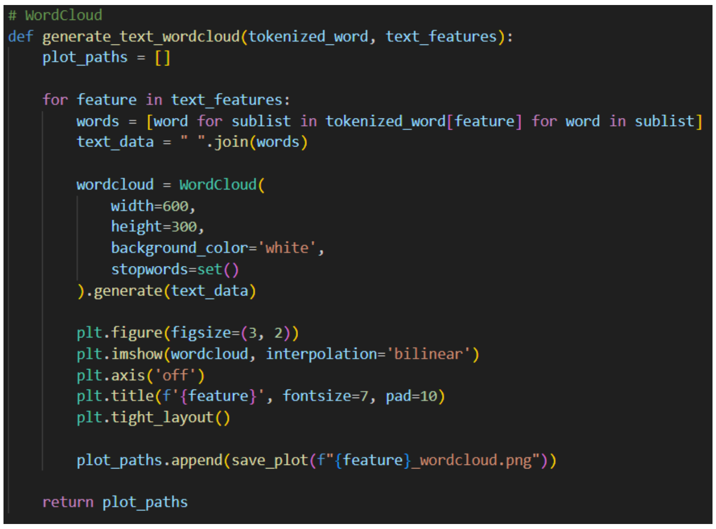


```python
# Categorical Pie Chart
def generate_categoric_piechart(dataframe, categoric_features):
    plot_paths = []

    for feature in categoric_features:
        value_counts = dataframe[feature].value_counts()
        top_4 = value_counts.nlargest(4)
        other_sum = value_counts.iloc[4:].sum()

        if other_sum > 0:
            top_4['Other'] = other_sum

        labels = top_4.index
        sizes = top_4.values
        pie_palette = sns.color_palette('coolwarm', len(top_4))

        plt.figure(figsize=(3, 3))
        wedges, texts, autotexts = plt.pie(
            sizes,
            colors=pie_palette,
            startangle=90,
            labels=None,  # Don't label with names here
            autopct='%1.1f%%',  # Show percentage on the pie slices
            textprops={'fontsize': 7}
        )

        # Legend with category name only (no values)
        plt.legend(wedges, labels, loc="center left", bbox_to_anchor=(1, 0.5), fontsize=7)

        plt.title(f'{feature}', fontsize=9)
        plt.ylabel('')

        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_pie.png"))

    return plot_paths
```

Figure 4-23 Pie Chart Function

The generate_categoric_piechart() function visualizes the top 4 categories in a pie chart, grouping the remaining into an "Other" category. Percentages are displayed on the slices, while category names are placed in a legend.

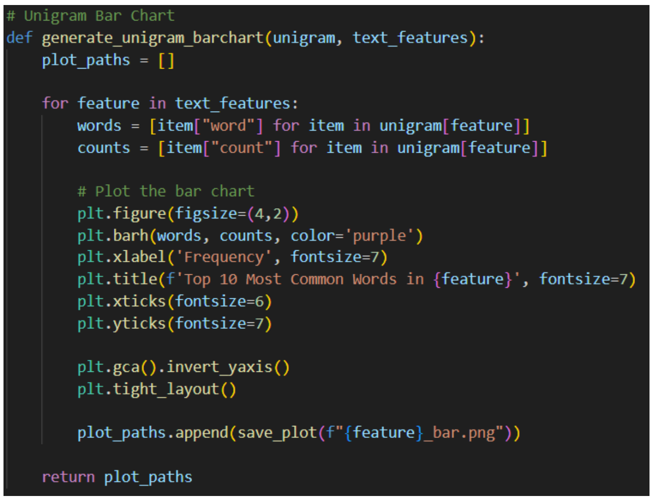
4.3.7.3 Textual Feature Visualization Function



```python
# WordCloud
def generate_text_wordcloud(tokenized_word, text_features):
    plot_paths = []

    for feature in text_features:
        words = [word for sublist in tokenized_word[feature] for word in sublist]
        text_data = " ".join(words)

        wordcloud = WordCloud(
            width=600,
            height=300,
            background_color='white',
            stopwords=set()
        ).generate(text_data)

        plt.figure(figsize=(3, 2))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.title(f'{feature}', fontsize=7, pad=10)
        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_wordcloud.png"))

    return plot_paths
```

Figure 4-24 Word cloud Function

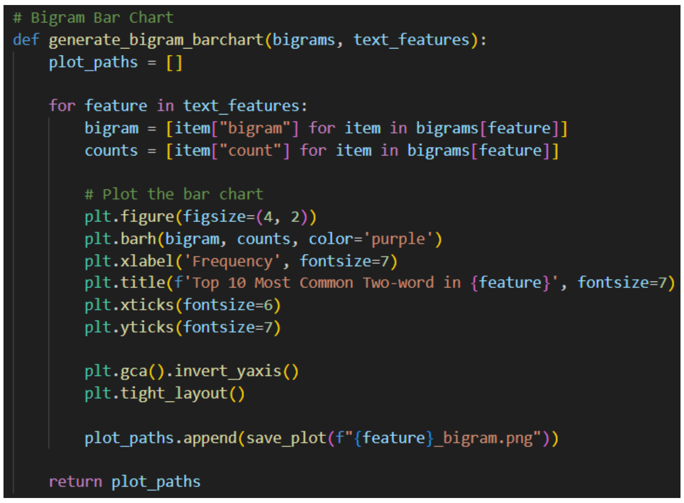
The generate_text_wordcloud() function generates a visual cloud where more frequent words appear larger.



```python
# Unigram Bar Chart
def generate_unigram_barchart(unigram, text_features):
    plot_paths = []

    for feature in text_features:
        words = [item["word"] for item in unigram[feature]]
        counts = [item["count"] for item in unigram[feature]]

        # Plot the bar chart
        plt.figure(figsize=(4,2))
        plt.barh(words, counts, color='purple')
        plt.xlabel('Frequency', fontsize=7)
        plt.title(f'Top 10 Most Common Words in {feature}', fontsize=7)
        plt.xticks(fontsize=6)
        plt.yticks(fontsize=7)

        plt.gca().invert_yaxis()
        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_bar.png"))

    return plot_paths
```

Figure 4-25 Unigram Bar Chart Function

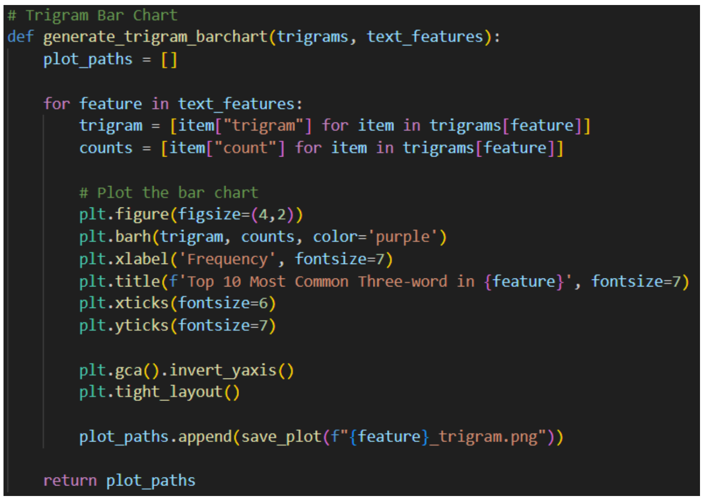
The generate_unigram_barchart() function plots the top 10 most frequent individual words for each text feature using a horizontal bar chart.



```python
# Bigram Bar Chart
def generate_bigram_barchart(bigrams, text_features):
    plot_paths = []

    for feature in text_features:
        bigram = [item["bigram"] for item in bigrams[feature]]
        counts = [item["count"] for item in bigrams[feature]]

        # Plot the bar chart
        plt.figure(figsize=(4, 2))
        plt.barh(bigram, counts, color='purple')
        plt.xlabel('Frequency', fontsize=7)
        plt.title(f'Top 10 Most Common Two-word in {feature}', fontsize=7)
        plt.xticks(fontsize=6)
        plt.yticks(fontsize=7)

        plt.gca().invert_yaxis()
        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_bigram.png"))

    return plot_paths
```

Figure 4-26 Bigram Bar Chart Function

The generate_bigram_barchart() function creates horizontal bar charts of the 10 most common two-word combinations.

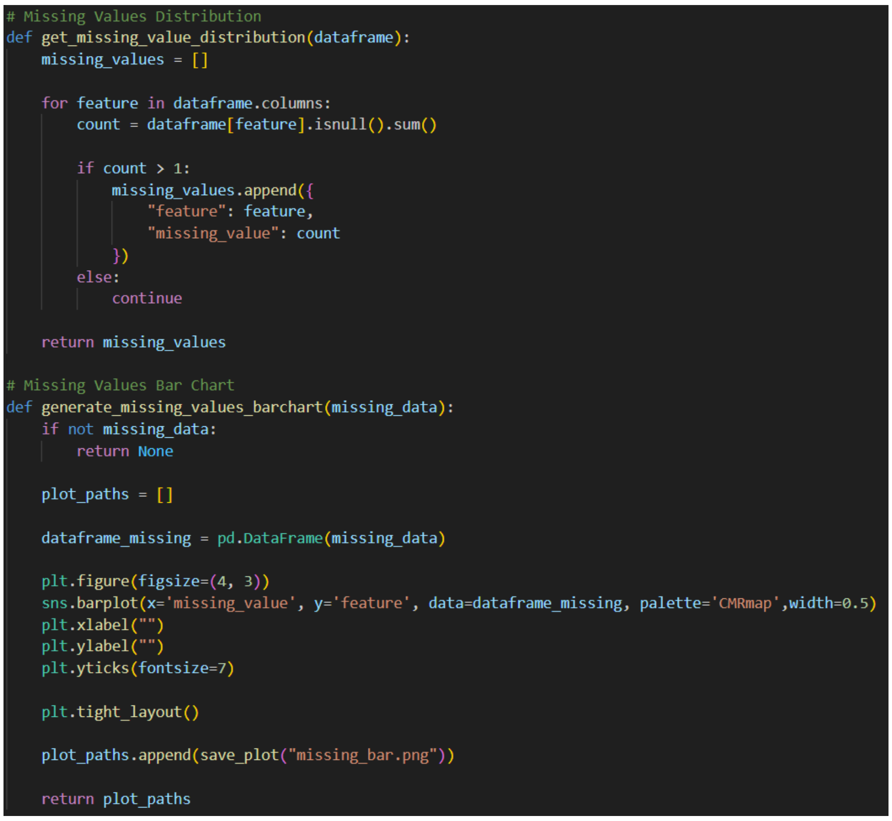


```python
# Trigram Bar Chart
def generate_trigram_barchart(trigrams, text_features):
    plot_paths = []

    for feature in text_features:
        trigram = [item["trigram"] for item in trigrams[feature]]
        counts = [item["count"] for item in trigrams[feature]]

        # Plot the bar chart
        plt.figure(figsize=(4,2))
        plt.barh(trigram, counts, color='purple')
        plt.xlabel('Frequency', fontsize=7)
        plt.title(f'Top 10 Most Common Three-word in {feature}', fontsize=7)
        plt.xticks(fontsize=6)
        plt.yticks(fontsize=7)

        plt.gca().invert_yaxis()
        plt.tight_layout()

        plot_paths.append(save_plot(f"{feature}_trigram.png"))

    return plot_paths
```

Figure 4-27 Trigram Bar Chart Function

The generate_trigram_barchart() function visualizes the most frequent three-word sequences in bar chart format.

4.3.8 Missing Values Analysis

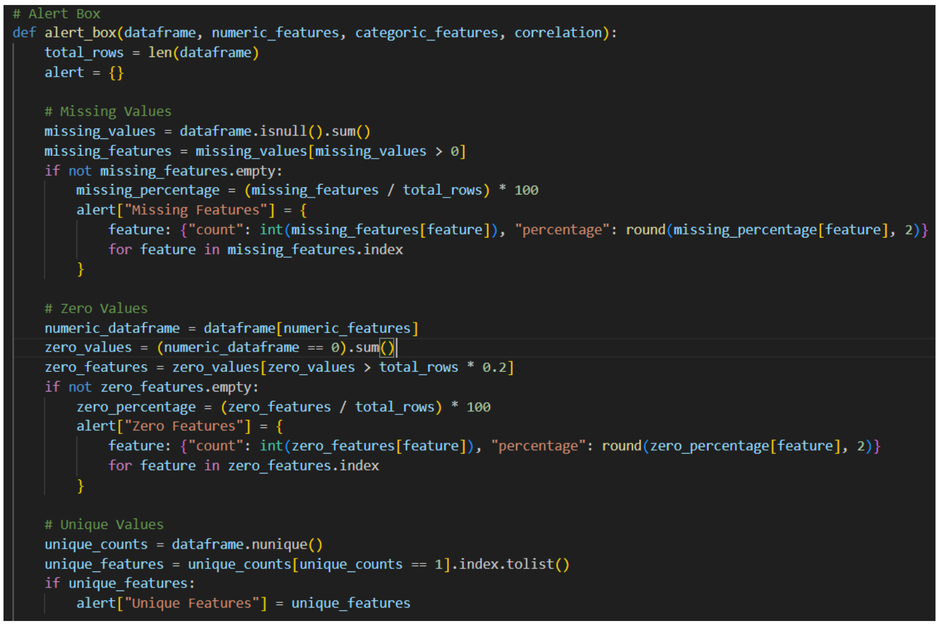


```python
# Missing Values Distribution
def get_missing_value_distribution(dataframe):
    missing_values = []

    for feature in dataframe.columns:
        count = dataframe[feature].isnull().sum()

        if count > 1:
            missing_values.append({
                "feature": feature,
                "missing_value": count
            })
        else:
            continue

    return missing_values

# Missing Values Bar Chart
def generate_missing_values_barchart(missing_data):
    if not missing_data:
        return None

    plot_paths = []

    dataframe_missing = pd.DataFrame(missing_data)

    plt.figure(figsize=(4, 3))
    sns.barplot(x='missing_value', y='feature', data=dataframe_missing, palette='CMRmap',width=0.5)
    plt.xlabel("")
    plt.ylabel("")
    plt.yticks(fontsize=7)

    plt.tight_layout()

    plot_paths.append(save_plot("missing_bar.png"))

    return plot_paths
```

Figure 4-28 Missing Analysis Function

The get_missing_value_distribution() function scans each feature to count how many values are missing. Only feature with more than one missing entry are considered to ensure relevance in reporting. The result is a summary that highlights which features may require imputation or special handling. The generate_missing_values_barchart() function creates a horizontal bar chart that displays the number of missing entries per feature to enhance interpretability.

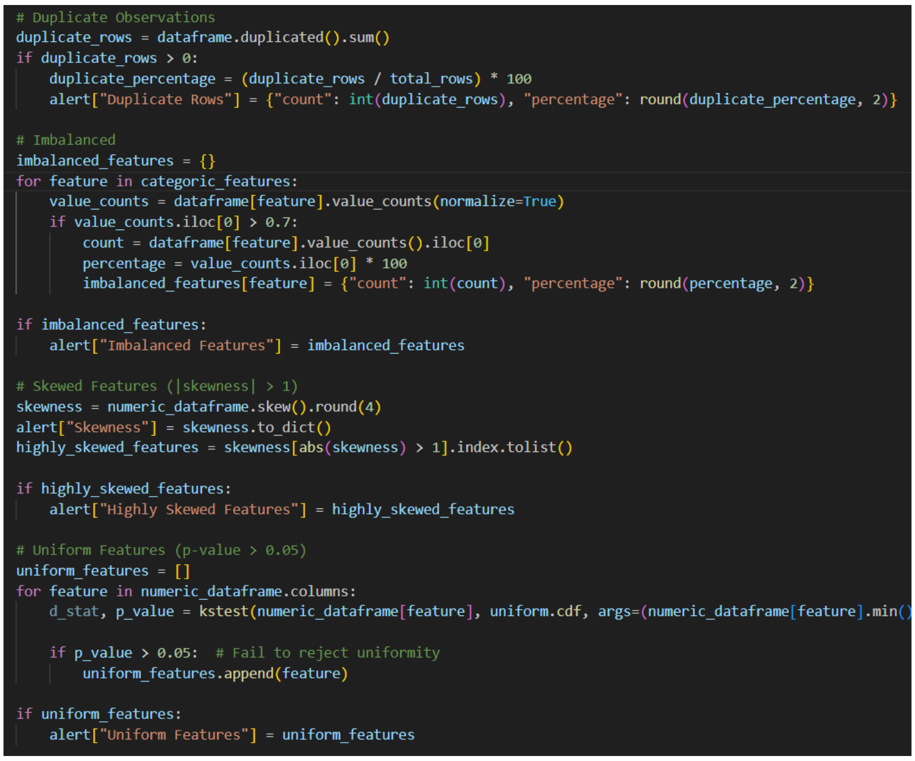
4.3.9 Data Quality Alert



```python
# Alert Box
def alert_box(dataframe, numeric_features, categoric_features, correlation):
    total_rows = len(dataframe)
    alert = {}

    # Missing Values
    missing_values = dataframe.isnull().sum()
    missing_features = missing_values[missing_values > 0]
    if not missing_features.empty:
        missing_percentage = (missing_features / total_rows) * 100
        alert["Missing Features"] = {
            feature: {"count": int(missing_features[feature]), "percentage": round(missing_percentage[feature], 2)}
            for feature in missing_features.index
        }

    # Zero Values
    numeric_dataframe = dataframe[numeric_features]
    zero_values = (numeric_dataframe == 0).sum()
    zero_features = zero_values[zero_values > total_rows * 0.2]
    if not zero_features.empty:
        zero_percentage = (zero_features / total_rows) * 100
        alert["Zero Features"] = {
            feature: {"count": int(zero_features[feature]), "percentage": round(zero_percentage[feature], 2)}
            for feature in zero_features.index
        }

    # Unique Values
    unique_counts = dataframe.nunique()
    unique_features = unique_counts[unique_counts == 1].index.tolist()
    if unique_features:
        alert["Unique Features"] = unique_features
```

Figure 4-29 Data Quality Alert Function (1)

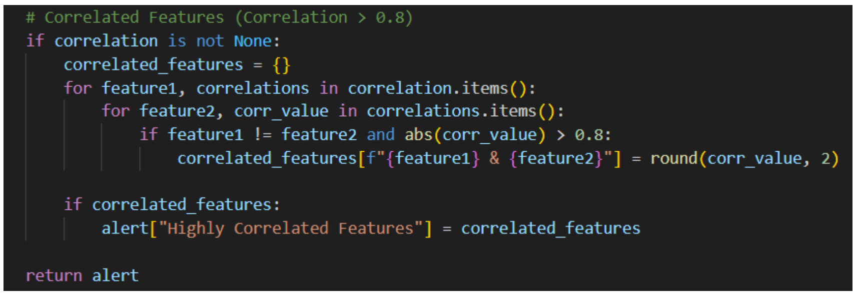
The module is designed to automatically flag potential data quality issues. It begins by identifying missing values, highlighting features with incomplete data and their corresponding percentages. Next, it checks for zero values in numeric columns. It flags features where over 20% of entries are zeros. The module also detects unique features.



```python
# Duplicate Observations
duplicate_rows = dataframe.duplicated().sum()
if duplicate_rows > 0:
    duplicate_percentage = (duplicate_rows / total_rows) * 100
    alert["Duplicate Rows"] = {"count": int(duplicate_rows), "percentage": round(duplicate_percentage, 2)}

# Imbalanced
imbalanced_features = {}
for feature in categoric_features:
    value_counts = dataframe[feature].value_counts(normalize=True)
    if value_counts.iloc[0] > 0.7:
        count = dataframe[feature].value_counts().iloc[0]
        percentage = value_counts.iloc[0] * 100
        imbalanced_features[feature] = {"count": int(count), "percentage": round(percentage, 2)}

if imbalanced_features:
    alert["Imbalanced Features"] = imbalanced_features

# Skewed Features (|skewness| > 1)
skewness = numeric_dataframe.skew().round(4)
alert["Skewness"] = skewness.to_dict()
highly_skewed_features = skewness[abs(skewness) > 1].index.tolist()

if highly_skewed_features:
    alert["Highly Skewed Features"] = highly_skewed_features

# Uniform Features (p-value > 0.05)
uniform_features = []
for feature in numeric_dataframe.columns:
    d_stat, p_value = kstest(numeric_dataframe[feature], uniform.cdf, args=(numeric_dataframe[feature].min()

    if p_value > 0.05:  # Fail to reject uniformity
        uniform_features.append(feature)

if uniform_features:
    alert["Uniform Features"] = uniform_features
```

Figure 4-30 Data Quality Alert Function (2)

Duplicate observations are noted along with their prevalence to help to detect data redundancy. The module also monitors imbalanced categorical features. It marks those where a single category dominates (over 70%). For numeric features, it calculates skewness, alerting users about features with high asymmetry (|skewness| > 1) and performs a Kolmogorov–Smirnov test to detect uniform distributions.

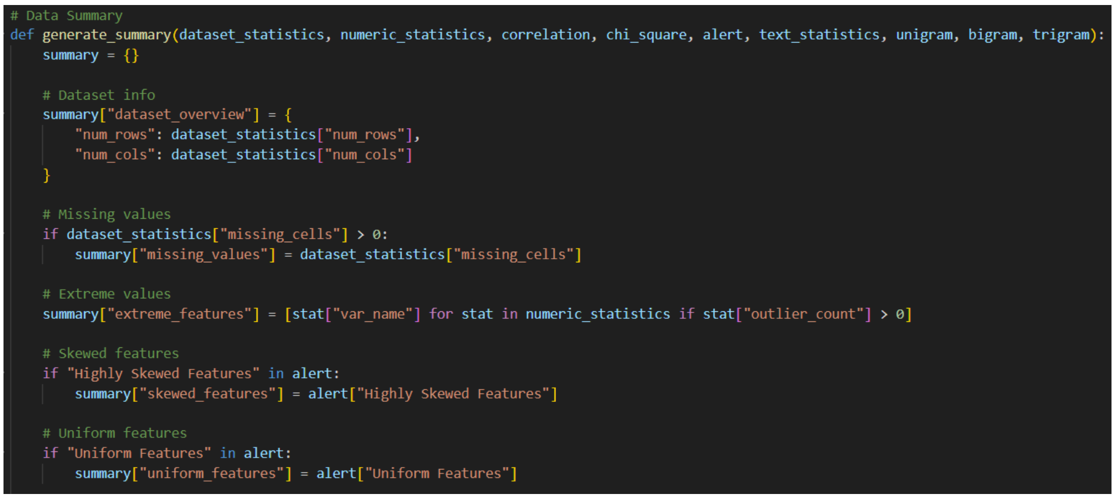


```python
# Correlated Features (Correlation > 0.8)
if correlation is not None:
    correlated_features = {}
    for feature1, correlations in correlation.items():
        for feature2, corr_value in correlations.items():
            if feature1 != feature2 and abs(corr_value) > 0.8:
                correlated_features[f"{feature1} & {feature2}"] = round(corr_value, 2)

    if correlated_features:
        alert["Highly Correlated Features"] = correlated_features

return alert
```

Figure 4-31 Data Quality Alert Function (3)

Additionally, it evaluates correlations, flagging pairs of features with high linear association (correlation > 0.8), which can lead to multicollinearity in modelling.

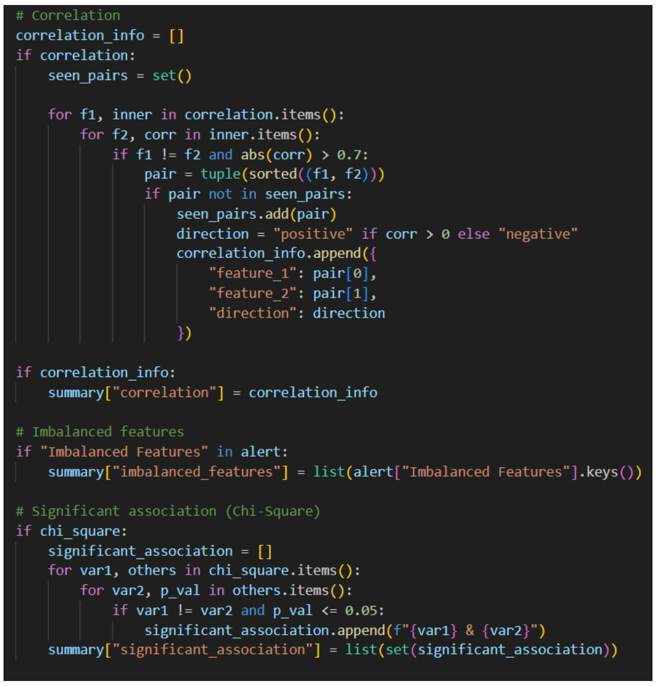
4.3.10 Data Summary



```python
# Data Summary
def generate_summary(dataset_statistics, numeric_statistics, correlation, chi_square, alert, text_statistics, unigram, bigram, trigram):
    summary = {}

    # Dataset info
    summary["dataset_overview"] = {
        "num_rows": dataset_statistics["num_rows"],
        "num_cols": dataset_statistics["num_cols"]
    }

    # Missing values
    if dataset_statistics["missing_cells"] > 0:
        summary["missing_values"] = dataset_statistics["missing_cells"]

    # Extreme values
    summary["extreme_features"] = [stat["var_name"] for stat in numeric_statistics if stat["outlier_count"] > 0]

    # Skewed features
    if "Highly Skewed Features" in alert:
        summary["skewed_features"] = alert["Highly Skewed Features"]

    # Uniform features
    if "Uniform Features" in alert:
        summary["uniform_features"] = alert["Uniform Features"]
```

Figure 4-32 Data Summary Function (1)

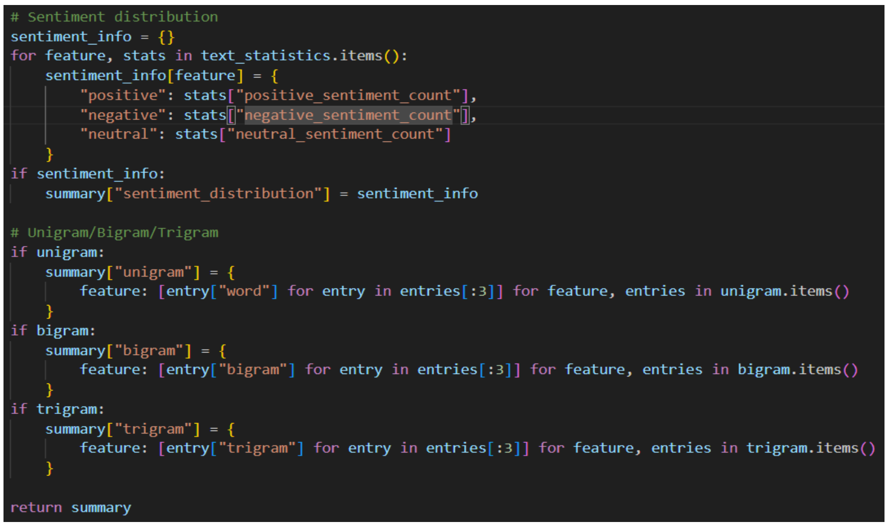
The generate_summary function compiles a comprehensive overview of a dataset by consolidating various statistical and analytical findings into a structured summary dictionary. It starts by recording the dataset's shape. It checks for missing values and includes the count if any are present. The function highlights feature with extreme values. It also identifies skewed and uniform features by referencing alerts generated during earlier analysis stages.



```python
# Correlation
correlation_info = []
if correlation:
    seen_pairs = set()

    for f1, inner in correlation.items():
        for f2, corr in inner.items():
            if f1 != f2 and abs(corr) > 0.7:
                pair = tuple(sorted((f1, f2)))
                if pair not in seen_pairs:
                    seen_pairs.add(pair)
                    direction = "positive" if corr > 0 else "negative"
                    correlation_info.append({
                        "feature_1": pair[0],
                        "feature_2": pair[1],
                        "direction": direction
                    })

if correlation_info:
    summary["correlation"] = correlation_info

# Imbalanced features
if "Imbalanced Features" in alert:
    summary["imbalanced_features"] = list(alert["Imbalanced Features"].keys())

# Significant association (Chi-Square)
if chi_square:
    significant_association = []
    for var1, others in chi_square.items():
        for var2, p_val in others.items():
            if var1 != var2 and p_val <= 0.05:
                significant_association.append(f"{var1} & {var2}")
    summary["significant_association"] = list(set(significant_association))
```

Figure 4-33 Data Summary Function (2)

In terms of relationships between features, the function detects highly correlated pairs (correlation > 0.7), recording both the direction and the involved features. It lists imbalanced categorical features where one class dominates and flags significant associations between categorical variables using chi-square tests ($p \leq 0.05$).



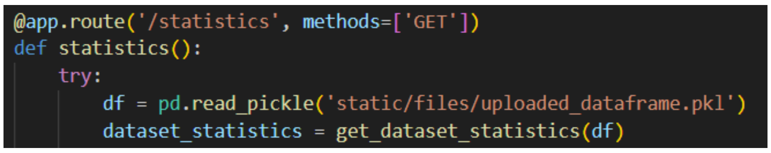
```python
# Sentiment distribution
sentiment_info = {}
for feature, stats in text_statistics.items():
    sentiment_info[feature] = {
        "positive": stats["positive_sentiment_count"],
        "negative": stats["negative_sentiment_count"],
        "neutral": stats["neutral_sentiment_count"]
    }
if sentiment_info:
    summary["sentiment_distribution"] = sentiment_info

# Unigram/Bigram/Trigram
if unigram:
    summary["unigram"] = {
        feature: [entry["word"] for entry in entries[:3]] for feature, entries in unigram.items()
    }
if bigram:
    summary["bigram"] = {
        feature: [entry["bigram"] for entry in entries[:3]] for feature, entries in bigram.items()
    }
if trigram:
    summary["trigram"] = {
        feature: [entry["trigram"] for entry in entries[:3]] for feature, entries in trigram.items()
    }

return summary
```

Figure 4-34 Data Summary Function (3)

For textual data, it summarizes sentiment distribution across features by detailing counts of positive, negative, and neutral sentiments. Furthermore, it extracts the top three most frequent unigrams, bigrams, and trigrams per text feature.

4.4 System Components Interaction Operations

4.4.1 Dataset Retrieval and Preparation

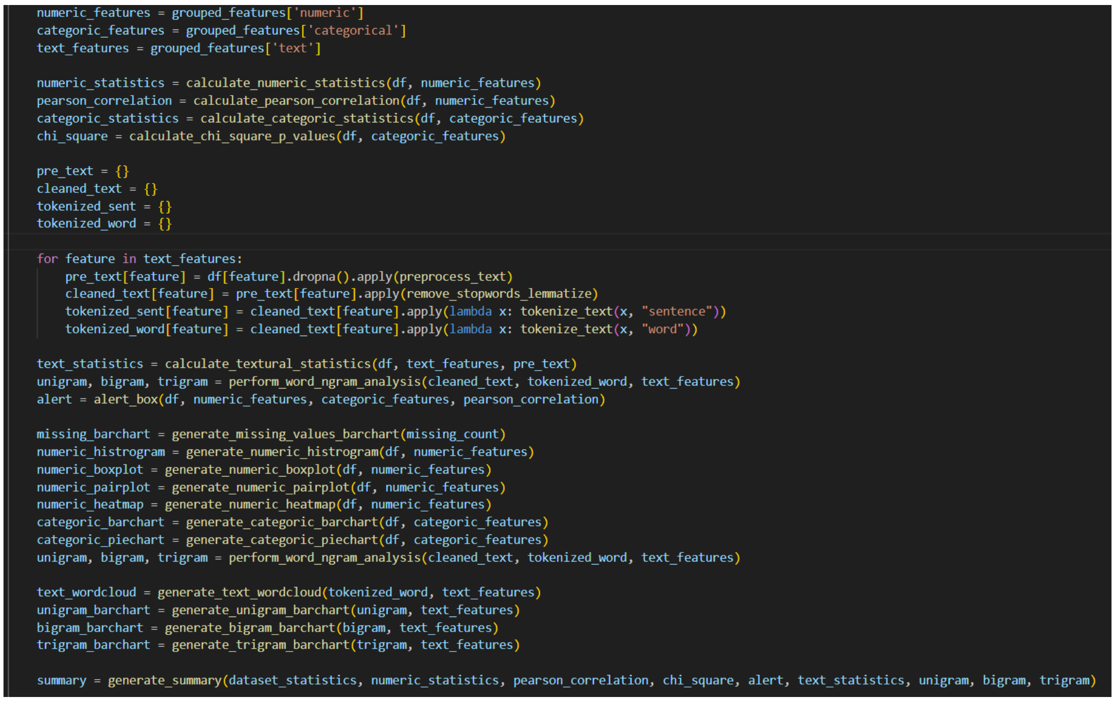


```python
@app.route('/statistics', methods=['GET'])
def statistics():
    try:
        df = pd.read_pickle('static/files/uploaded_dataframe.pkl')
        dataset_statistics = get_dataset_statistics(df)
```

Figure 4-35 Dataset Retrieval

When a user accesses the /statistics route via a GET request, the system loads the pre-processed dataset from a stored pickle file (uploaded_dataframe.pkl). The system then computes basic dataset statistics.

4.4.2 System Function Operations

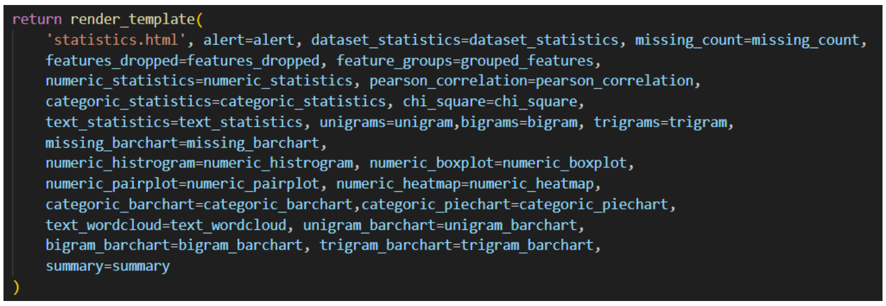


```python
numeric_features = grouped_features['numeric']
categoric_features = grouped_features['categorical']
text_features = grouped_features['text']

numeric_statistics = calculate_numeric_statistics(df, numeric_features)
pearson_correlation = calculate_pearson_correlation(df, numeric_features)
categoric_statistics = calculate_categoric_statistics(df, categoric_features)
chi_square = calculate_chi_square_p_values(df, categoric_features)

pre_text = {}
cleaned_text = {}
tokenized_sent = {}
tokenized_word = {}

for feature in text_features:
    pre_text[feature] = df[feature].dropna().apply(preprocess_text)
    cleaned_text[feature] = pre_text[feature].apply(remove_stopwords_lemmatize)
    tokenized_sent[feature] = cleaned_text[feature].apply(lambda x: tokenize_text(x, "sentence"))
    tokenized_word[feature] = cleaned_text[feature].apply(lambda x: tokenize_text(x, "word"))

text_statistics = calculate_textural_statistics(df, text_features, pre_text)
unigram, bigram, trigram = perform_word_ngram_analysis(cleaned_text, tokenized_word, text_features)
alert = alert_box(df, numeric_features, categoric_features, pearson_correlation)

missing_barchart = generate_missing_values_barchart(missing_count)
numeric_histrogram = generate_numeric_histrogram(df, numeric_features)
numeric_boxplot = generate_numeric_boxplot(df, numeric_features)
numeric_pairplot = generate_numeric_pairplot(df, numeric_features)
numeric_heatmap = generate_numeric_heatmap(df, numeric_features)
categoric_barchart = generate_categoric_barchart(df, categoric_features)
categoric_piechart = generate_categoric_piechart(df, categoric_features)
unigram, bigram, trigram = perform_word_ngram_analysis(cleaned_text, tokenized_word, text_features)

text_wordcloud = generate_text_wordcloud(tokenized_word, text_features)
unigram_barchart = generate_unigram_barchart(unigram, text_features)
bigram_barchart = generate_bigram_barchart(bigram, text_features)
trigram_barchart = generate_trigram_barchart(trigram, text_features)

summary = generate_summary(dataset_statistics, numeric_statistics, pearson_correlation, chi_square, alert, text_statistics, unigram, bigram, trigram)
```

Figure 4-36 System Function Operations

Once a dataset is uploaded, a series of processing steps are triggered to analyse and visualize the data. The system first generates an overview of the dataset, including basic statistics such as shape, memory usage, and missing values. It then cleans the dataset by removing features that are meaningless and classifies the remaining features into numeric, categorical, and textual groups. Each type undergoes tailored analysis. Numerical features are statistically profiled and visualized through plots like histograms, boxplots, and heatmaps. Categorical features are summarized and visualized via bar and pie chart. Textual features are pre-processed, tokenized and analysed using n-gram extraction, word clouds and summary statistics. Additionally, the system evaluates data quality by checking for missing values, duplicates, skewed or correlated features and imbalanced distributions. Finally, all insights including alerts and statistical summaries are compiled into a report to help users understand the structure and quality of their data.

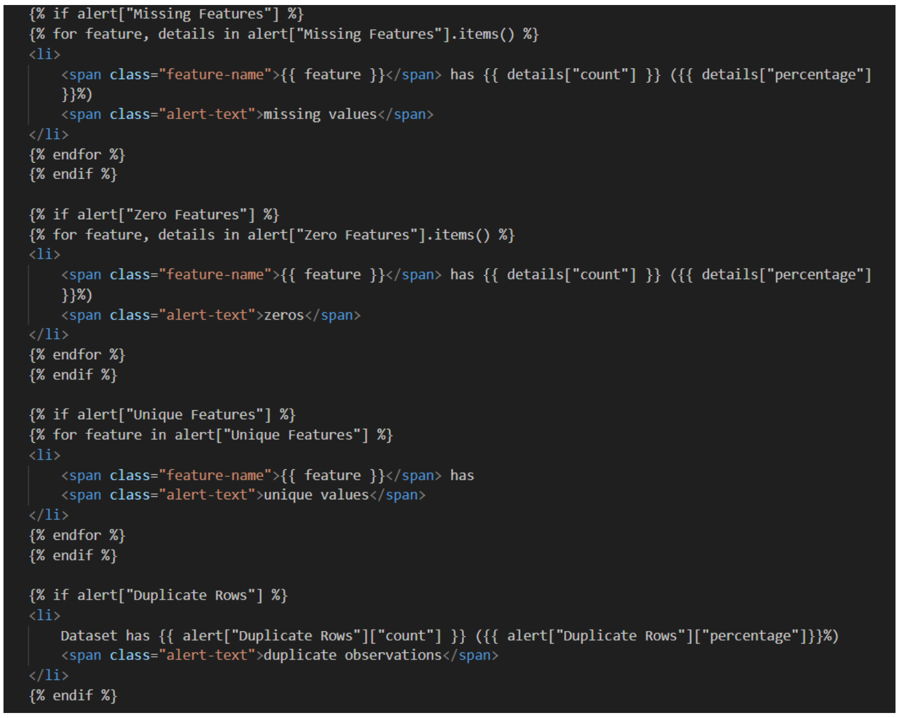
4.4.3 Frontend Integration



```
return render_template(
    'statistics.html', alert=alert, dataset_statistics=dataset_statistics, missing_count=missing_count,
    features_dropped=features_dropped, feature_groups=grouped_features,
    numeric_statistics=numeric_statistics, pearson_correlation=pearson_correlation,
    categoric_statistics=categoric_statistics, chi_square=chi_square,
    text_statistics=text_statistics, unigrams=unigram,bigrams=bigram, trigrams=trigram,
    missing_barchart=missing_barchart,
    numeric_histrogram=numeric_histrogram, numeric_boxplot=numeric_boxplot,
    numeric_pairplot=numeric_pairplot, numeric_heatmap=numeric_heatmap,
    categoric_barchart=categoric_barchart,categoric_piechart=categoric_piechart,
    text_wordcloud=text_wordcloud, unigram_barchart=unigram_barchart,
    bigram_barchart=bigram_barchart, trigram_barchart=trigram_barchart,
    summary=summary
)
```

Figure 4-37 Render template Function

The backend system integrates seamlessly with the frontend using Flask's render_template function. It allows dynamic data to be passed to HTML templates using the Jinja2 templating engine.



```
{% if alert["Missing Features"] %}
{% for feature, details in alert["Missing Features"].items() %}
<li>
    <span class="feature-name">{{ feature }}</span> has {{ details["count"] }} ({{ details["percentage"]
    }}%)
    <span class="alert-text">missing values</span>
</li>
{% endfor %}
{% endif %}

{% if alert["Zero Features"] %}
{% for feature, details in alert["Zero Features"].items() %}
<li>
    <span class="feature-name">{{ feature }}</span> has {{ details["count"] }} ({{ details["percentage"]
    }}%)
    <span class="alert-text">zeros</span>
</li>
{% endfor %}
{% endif %}

{% if alert["Unique Features"] %}
{% for feature in alert["Unique Features"] %}
<li>
    <span class="feature-name">{{ feature }}</span> has
    <span class="alert-text">unique values</span>
</li>
{% endfor %}
{% endif %}

{% if alert["Duplicate Rows"] %}
<li>
    Dataset has {{ alert["Duplicate Rows"]["count"] }} ({{ alert["Duplicate Rows"]["percentage"]}}%)
    <span class="alert-text">duplicate observations</span>
</li>
{% endif %}
```

Figure 4-38 Frontend with Jinja2

On the frontend, the statistics.html file utilizes Jinja syntax (e.g., {{ variable_name }} and {% for item in list %}) to display these variables dynamically. For example, the alert box section loops through various alert types using {% if alert["Missing Features"] %} to render relevant warnings. Similarly, graphs like histograms and heatmaps are embedded using <img src="{{ numeric_histrogram }}">, pulling in static images generated in the backend.

Chapter 5 System Implementation

This chapter briefly describes the hardware and software setup used for the implementation of the system. The setup and operation methods required to run the system will also be explained in this chapter.

5.1 Hardware Setup

The system was developed and tested on a personal computer equipped with an Intel Core i7 processor and 16GB of RAM. This hardware setup provided sufficient processing power for data analysis tasks and ensured smooth interaction with the developed application.

Description	**Specification**
Model	Acer Predator
Processor	13th Gen Intel(R) Core (TM) i7-13700HX 2.10 GHz
Operating System	Windows 11 64-bit
Installed RAM	16 GB
Graphic	NVIDIA GeForce RTX 4050

Table 5-1 Specification of Laptop

5.2 Software Setup

1. Visual Studio Code

Visual Studio Code was used as the primary integrated development environment due to its lightweight design and powerful extension support. The following extensions are recommended:

- **Python** – Provides IntelliSense, linting, debugging and code navigation for Python.
- **Pylance** – Offers performant, feature-rich language support for Python.
- **Live Server** – Allows quick frontend preview during HTML development.

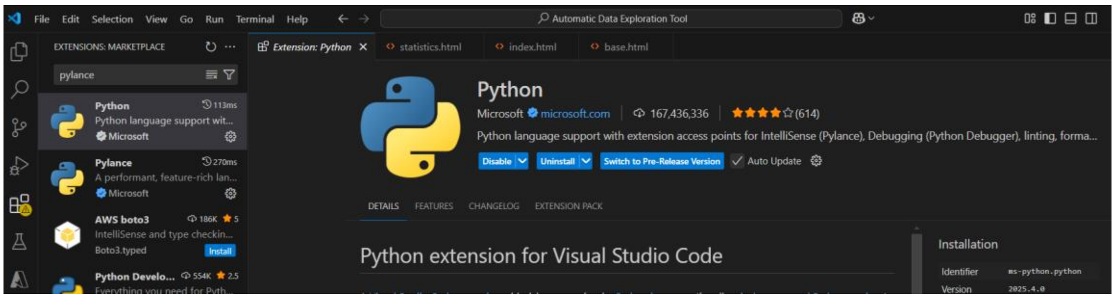
2. Python Environment

Python 3.12.7 was used as the core language with the following libraries required for system functionality. These can be installed via pip or a requirements.txt file:

- **Flask** – For handling web requests and rendering dynamic HTML pages.
- **pandas** – For data manipulation and analysis.

- **numpy** – For numerical computations.
- **matplotlib** and **seaborn** – For creating charts and statistical visualizations.
- **scikit-learn** – For statistical computations such as correlation and chi-square tests.
- **nltk** and **spacy** – For natural language processing and text preprocessing.
- **wordcloud** – To generate visual representations of word frequency.
- **pickle** – For saving and loading processed datasets.
- **Plotly** – If interactive plots are desired.

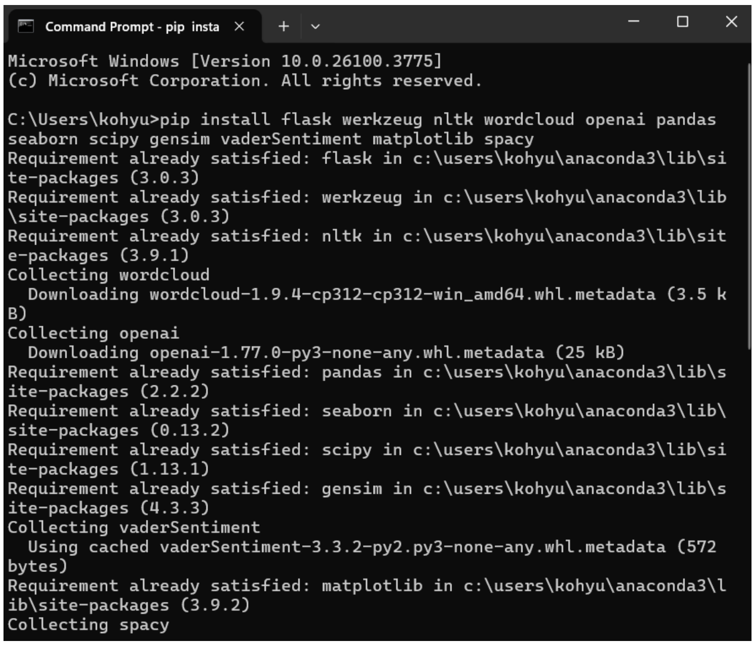
5.3 Setting and Configuration


File Edit Selection View Go Run Terminal Help
Automatic Data Exploration Tool
EXTENSIONS: MARKETPLACE
pylance
Extension: Python
statistics.html
index.html
base.html
Python
Microsoft microsoft.com
167,436,336
(614)
Disable
Uninstall
Switch to Pre-Release Version
Auto Update
DETAILS
FEATURES
CHANGELOG
EXTENSION PACK
Python extension for Visual Studio Code
Installation
Identifier ms-python.python
Version 2025.4.0
Python
Python language support wit...
Microsoft
Pylance
A performant, feature-rich lan...
Microsoft
AWS boto3
IntelliSense and type checkin...
Boto3.typed
Install
Python Develo...
Everything you need for Pyth...

Python language support with extension access points for IntelliSense (Pylance), Debugging (Python Debugger), linting, forma...

Figure 5-1 Visual Studio Code Extension

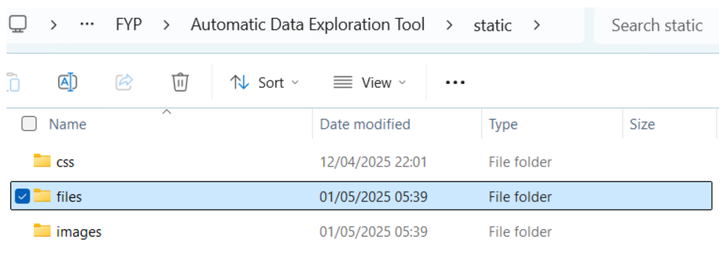
Visual Studio Code was used as the primary development environment. Several essential extensions were installed to facilitate Flask and Python development. These included Python and Pylance.



```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kohyu>pip install flask werkzeug nltk wordcloud openai pandas
seaborn scipy gensim vaderSentiment matplotlib spacy
Requirement already satisfied: flask in c:\users\kohyu\anaconda3\lib\si
te-packages (3.0.3)
Requirement already satisfied: werkzeug in c:\users\kohyu\anaconda3\lib
\site-packages (3.0.3)
Requirement already satisfied: nltk in c:\users\kohyu\anaconda3\lib\sit
e-packages (3.9.1)
Collecting wordcloud
  Downloading wordcloud-1.9.4-cp312-cp312-win_amd64.whl.metadata (3.5 k
B)
Collecting openai
  Downloading openai-1.77.0-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: pandas in c:\users\kohyu\anaconda3\lib\s
ite-packages (2.2.2)
Requirement already satisfied: seaborn in c:\users\kohyu\anaconda3\lib\
site-packages (0.13.2)
Requirement already satisfied: scipy in c:\users\kohyu\anaconda3\lib\si
te-packages (1.13.1)
Requirement already satisfied: gensim in c:\users\kohyu\anaconda3\lib\s
ite-packages (4.3.3)
Collecting vaderSentiment
  Using cached vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572
bytes)
Requirement already satisfied: matplotlib in c:\users\kohyu\anaconda3\l
ib\site-packages (3.9.2)
Collecting spacy
```

Figure 5-2 Installation libraries using Command Prompt

Python environment was configured by installing all necessary libraries. This was done by opening the Command Prompt and executing the pip install command , as illustrated in **Figure 5.2.**



Name	Date modified	Type	Size
css	12/04/2025 22:01	File folder	
files	01/05/2025 05:39	File folder	
images	01/05/2025 05:39	File folder	

Figure 5-3 File Directory

Configuration options such as file upload paths and allowed file types are handled within the Flask app settings. All output files, including pickled DataFrames and visualizations, are organized in designated static/files and static/images directories.

5.4 System Operation

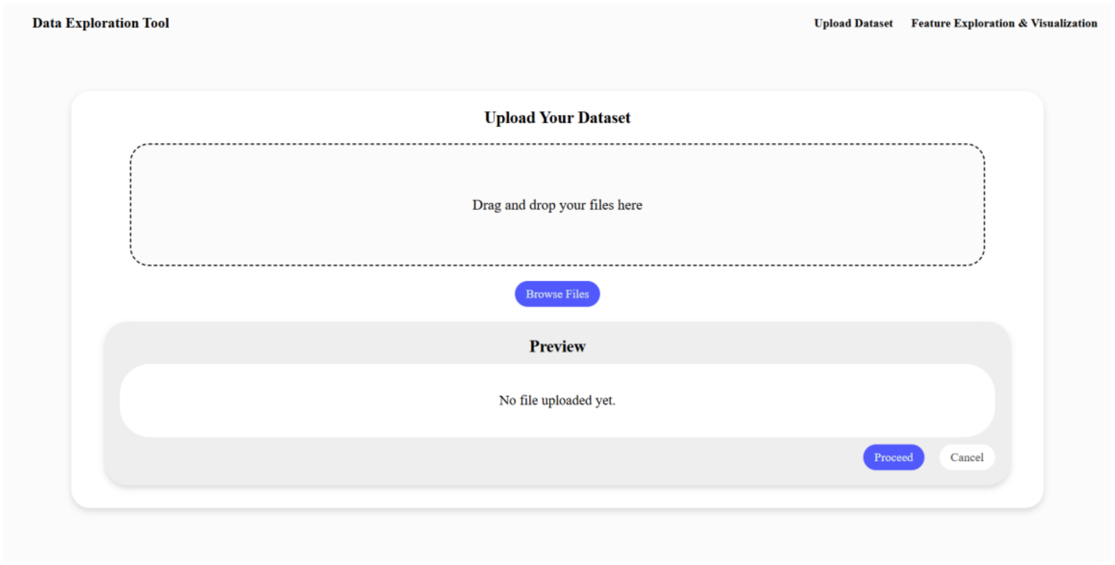
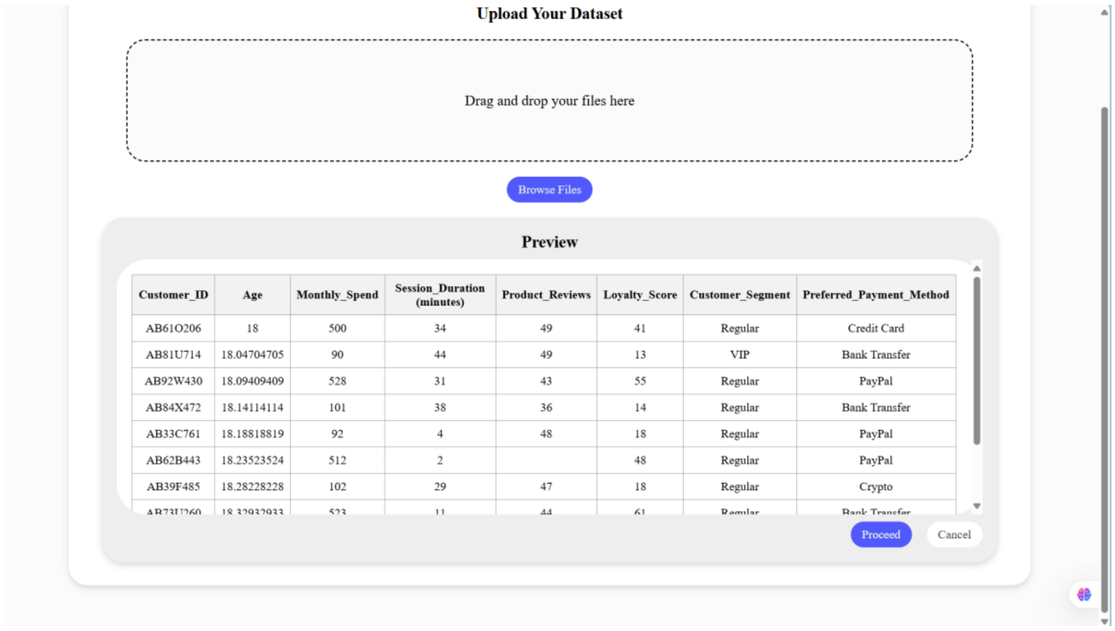


Figure 5-4 Home Page

On the upload page of the frontend, users are provided a interface for uploading their datasets. A drag-and-drop area allows users to easily drag files into the browser window, while a "Browse File" button is also available for users who prefer to select a file manually from their local storage.

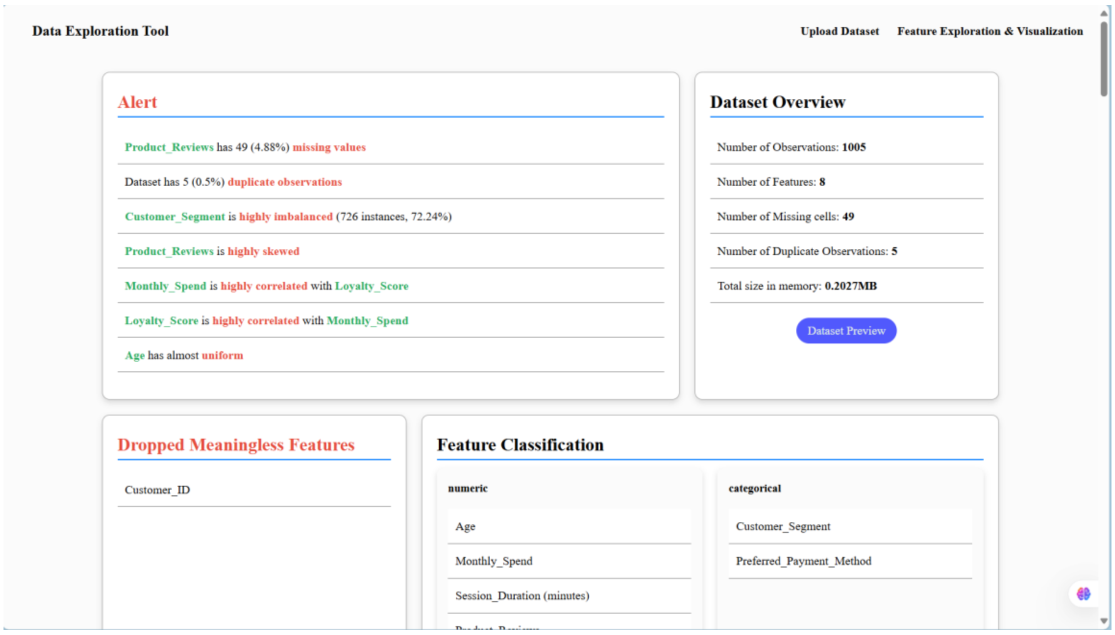


Customer_ID	Age	Monthly_Spend	Session_Duration (minutes)	Product_Reviews	Loyalty_Score	Customer_Segment	Preferred_Payment_Method
AB61O206	18	500	34	49	41	Regular	Credit Card
AB81U714	18.04704705	90	44	49	13	VIP	Bank Transfer
AB92W430	18.09409409	528	31	43	55	Regular	PayPal
AB84X472	18.14114114	101	38	36	14	Regular	Bank Transfer
AB33C761	18.18818819	92	4	48	18	Regular	PayPal
AB62B443	18.23523524	512	2		48	Regular	PayPal
AB39F485	18.28228228	102	29	47	18	Regular	Crypto
AB73U260	18.32932933	523	11	44	61	Regular	Bank Transfer

Figure 5-5 Preview Section

Once a dataset is uploaded, a preview box displays the contents of the dataset. It gives users an opportunity to review the data before proceeding. If the dataset is correct and the user is ready to perform Exploratory Data Analysis, they can click the "Proceed" button to continue.

Alternatively, if the user wants to cancel or upload a different dataset, they can simply click the "Cancel" button to reset the process.



Data Exploration Tool

Upload Dataset | Feature Exploration & Visualization

Alert

- Product_Reviews has 49 (4.88%) missing values
- Dataset has 5 (0.5%) duplicate observations
- Customer_Segment is highly imbalanced (726 instances, 72.24%)
- Product_Reviews is highly skewed
- Monthly_Spend is highly correlated with Loyalty_Score
- Loyalty_Score is highly correlated with Monthly_Spend
- Age has almost uniform

Dataset Overview

- Number of Observations: **1005**
- Number of Features: **8**
- Number of Missing cells: **49**
- Number of Duplicate Observations: **5**
- Total size in memory: **0.2027MB**

Dropped Meaningless Features

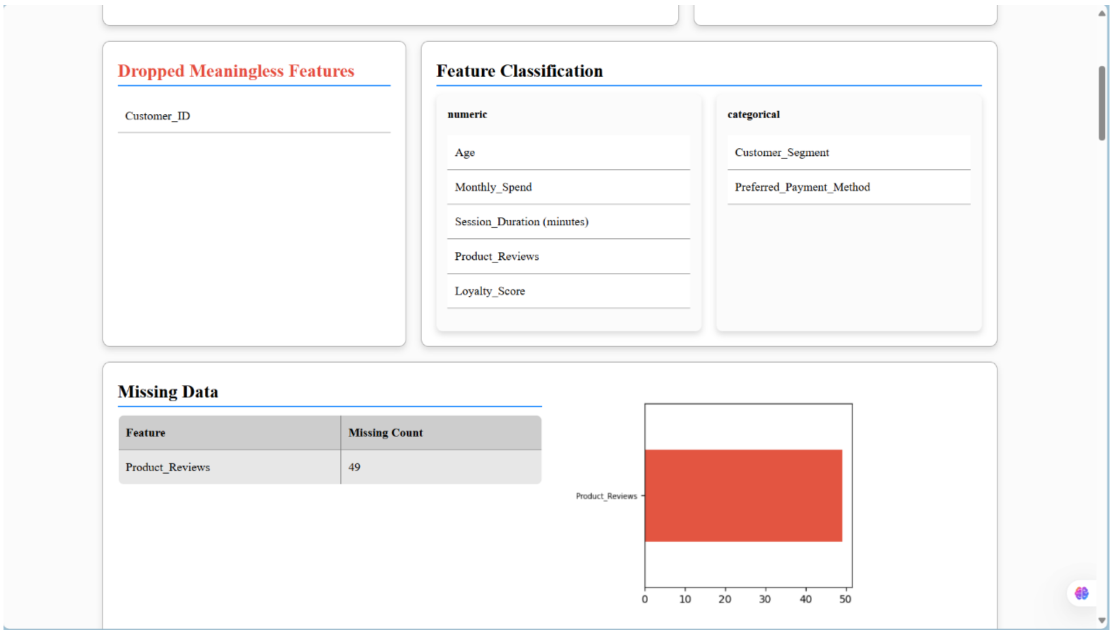
- Customer_ID

Feature Classification

numeric	categorical
Age	Customer_Segment
Monthly_Spend	Preferred_Payment_Method
Session_Duration (minutes)	

Figure 5-6 Statistics Page (1)

On the Statistics page, users are first presented with an alert section that highlights any important findings or potential issues detected in the dataset. This is followed by the Dataset Overview which provides a general summary of the dataset’s structure and characteristics. Within the overview section, a "Dataset Preview" button is available. When user clicks this button, it will redirect back to the upload page. Users can re-examine the original dataset preview if needed.



Dropped Meaningless Features

- Customer_ID

Feature Classification

numeric	categorical
Age	Customer_Segment
Monthly_Spend	Preferred_Payment_Method
Session_Duration (minutes)	
Product_Reviews	
Loyalty_Score	

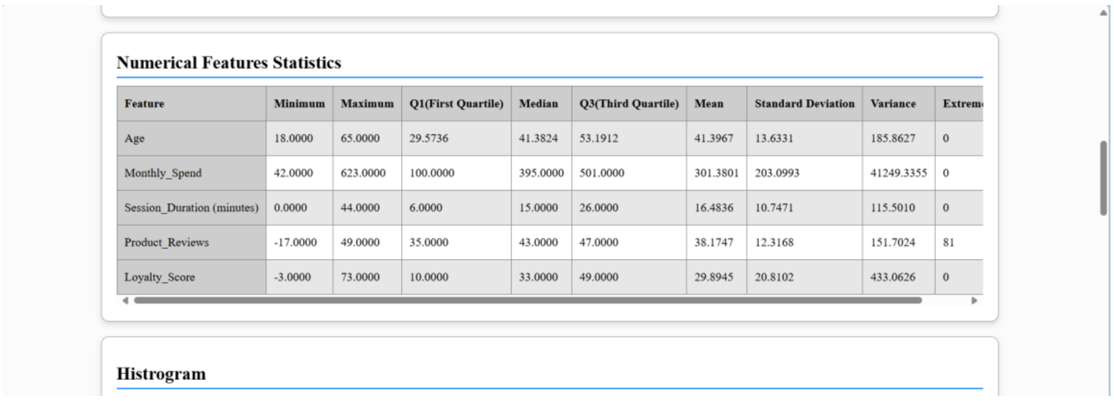
Missing Data

Feature	Missing Count
Product_Reviews	49

55

Figure 5-7 Statistics Page (2)

After the overview, the page displays information about any dropped meaningless features. This is followed by the feature classification section which groups the remaining features into numerical, categorical, and textual types for clarity. At the bottom, the Missing Data section provides a detailed summary and visualization of any null or missing values within the dataset.



Numerical Features Statistics

Feature	Minimum	Maximum	Q1(First Quartile)	Median	Q3(Third Quartile)	Mean	Standard Deviation	Variance	Extrem
Age	18.0000	65.0000	29.5736	41.3824	53.1912	41.3967	13.6331	185.8627	0
Monthly_Spend	42.0000	623.0000	100.0000	395.0000	501.0000	301.3801	203.0993	41249.3355	0
Session_Duration (minutes)	0.0000	44.0000	6.0000	15.0000	26.0000	16.4836	10.7471	115.5010	0
Product_Reviews	-17.0000	49.0000	35.0000	43.0000	47.0000	38.1747	12.3168	151.7024	81
Loyalty_Score	-3.0000	73.0000	10.0000	33.0000	49.0000	29.8945	20.8102	433.0626	0

Histrogram

Figure 5-8 Statistics Page (3)

After the Missing Data section, the page continues with the Feature Statistics analysis. It begins with the Numerical Feature Statistics

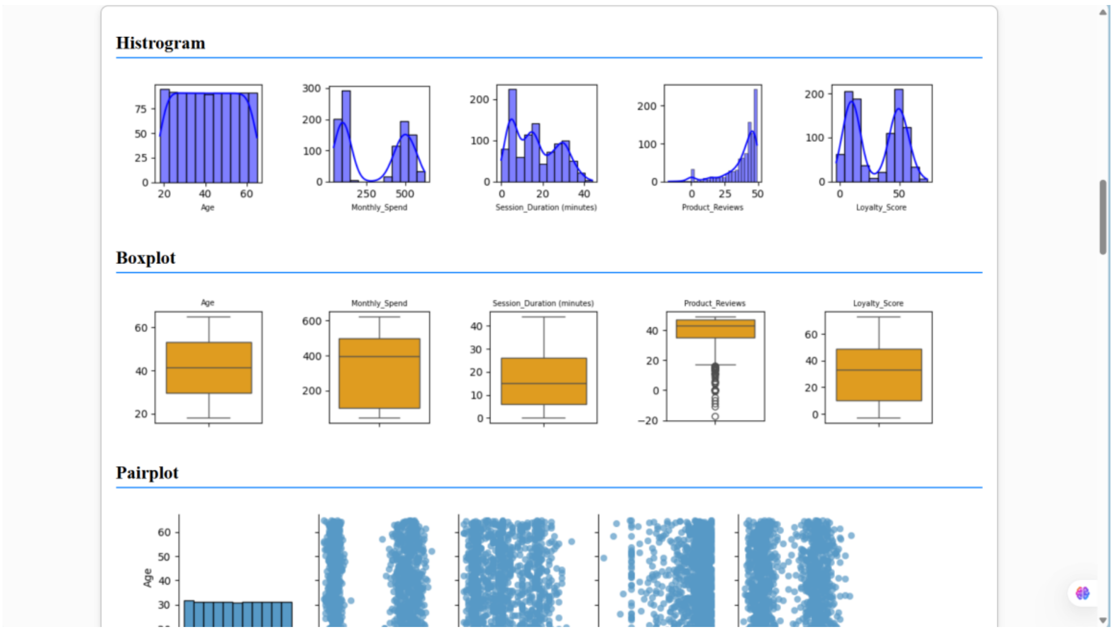
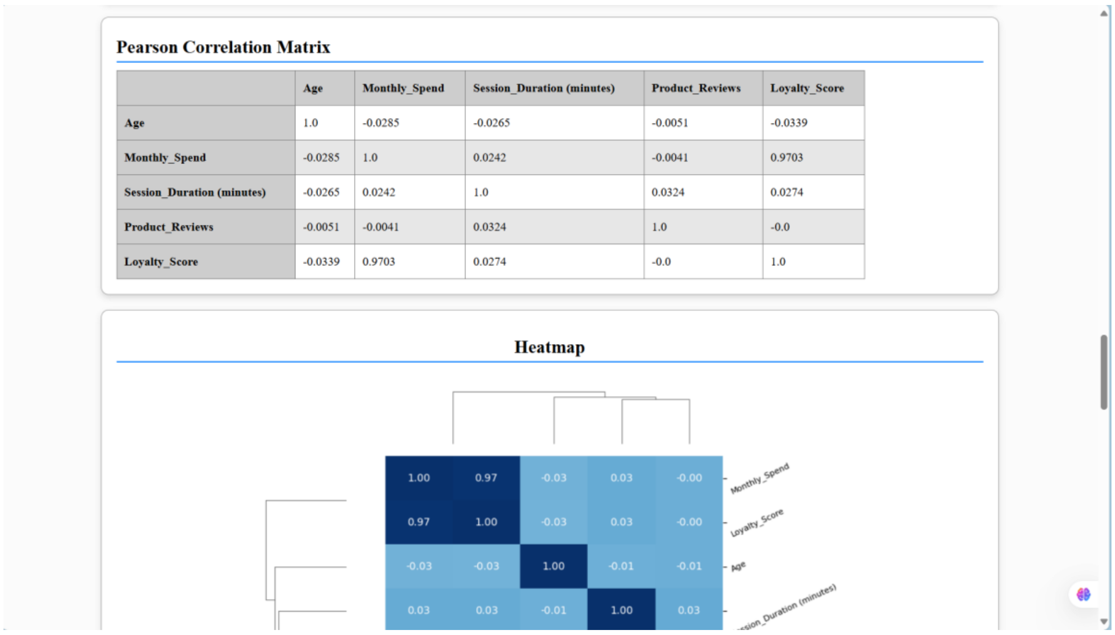


Figure 5-9 Statistics Page (4)

It follows a series of visualizations including histograms, boxplots and pair plots to help users visually explore the distribution and relationships between numerical features.

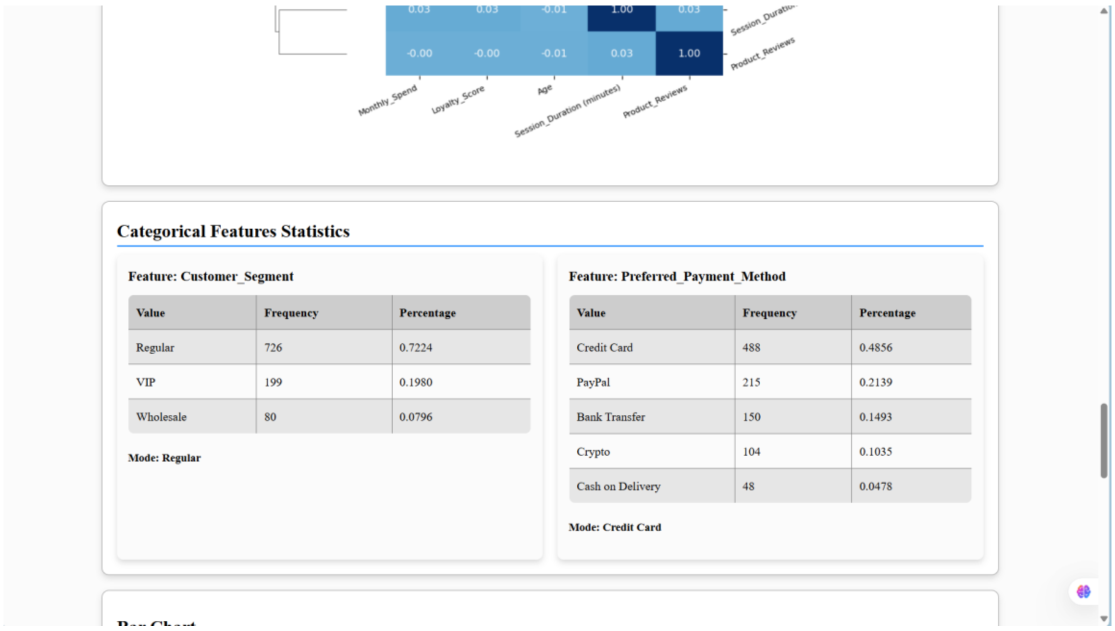

Pearson Correlation Matrix

	Age	Monthly_Spend	Session_Duration (minutes)	Product_Reviews	Loyalty_Score
Age	1.0	-0.0285	-0.0265	-0.0051	-0.0339
Monthly_Spend	-0.0285	1.0	0.0242	-0.0041	0.9703
Session_Duration (minutes)	-0.0265	0.0242	1.0	0.0324	0.0274
Product_Reviews	-0.0051	-0.0041	0.0324	1.0	-0.0
Loyalty_Score	-0.0339	0.9703	0.0274	-0.0	1.0

Heatmap

Figure 5-10 Statistics Page (5)

Next, the Pearson Correlation Matrix is displayed to show the linear relationships between numerical variables and followed heatmap.



Categorical Features Statistics

Feature: Customer_Segment

Value	Frequency	Percentage
Regular	726	0.7224
VIP	199	0.1980
Wholesale	80	0.0796

Mode: Regular

Feature: Preferred_Payment_Method

Value	Frequency	Percentage
Credit Card	488	0.4856
PayPal	215	0.2139
Bank Transfer	150	0.1493
Crypto	104	0.1035
Cash on Delivery	48	0.0478

Mode: Credit Card

Figure 5-11 Statistics Page (6)

Following that is the Categorical Feature Statistics section which summarizes the frequency and distribution of each categorical feature.

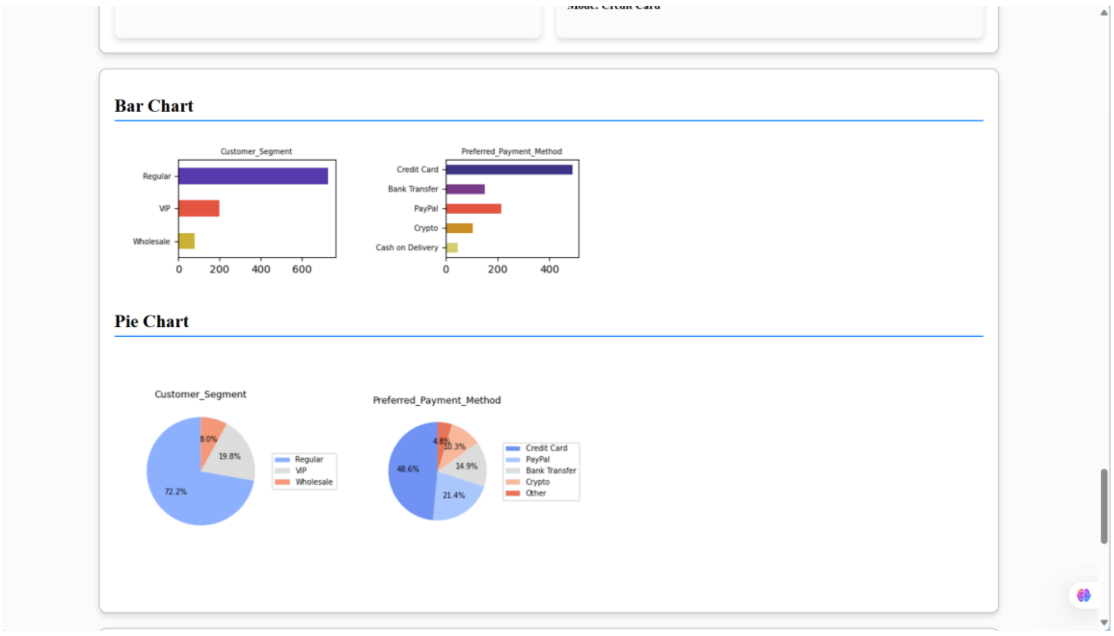
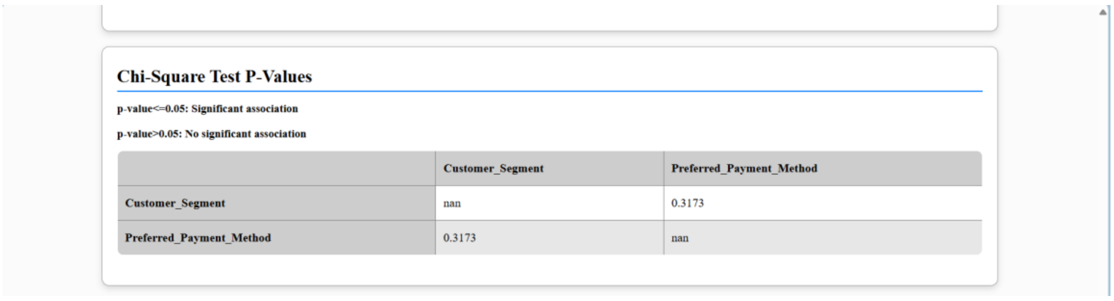


Figure 5-12 Statistics Page (7)

It is accompanied by visualizations such as bar charts and pie charts for better insight into the data.



Chi-Square Test P-Values

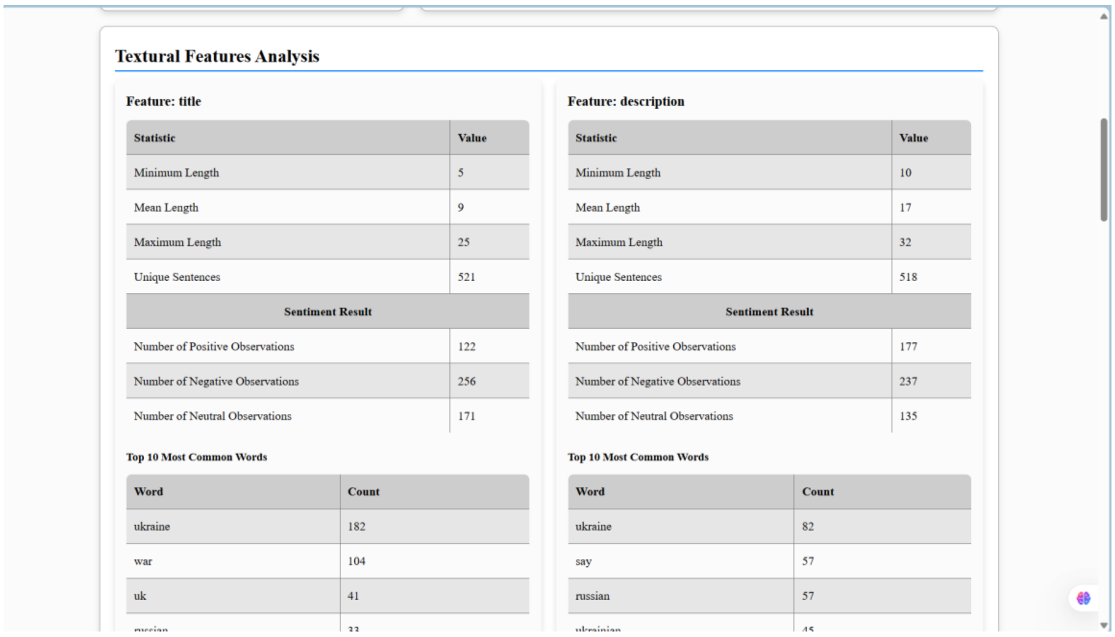
p-value<=0.05: Significant association

p-value>0.05: No significant association

	Customer_Segment	Preferred_Payment_Method
Customer_Segment	nan	0.3173
Preferred_Payment_Method	0.3173	nan

Figure 5-13 Statistics Page (8)

Then, the Chi-Square Test Table is shown.



Textural Features Analysis

Feature: title

Statistic	Value
Minimum Length	5
Mean Length	9
Maximum Length	25
Unique Sentences	521
Sentiment Result	
Number of Positive Observations	122
Number of Negative Observations	256
Number of Neutral Observations	171

Top 10 Most Common Words

Word	Count
ukraine	182
war	104
uk	41
russian	33

Feature: description

Statistic	Value
Minimum Length	10
Mean Length	17
Maximum Length	32
Unique Sentences	518
Sentiment Result	
Number of Positive Observations	177
Number of Negative Observations	237
Number of Neutral Observations	135

Top 10 Most Common Words

Word	Count
ukraine	82
say	57
russian	57
ukrainian	45

Figure 5-14 Statistics Page (9)

The page then presents the Textual Feature Analysis. Text data is explored through word frequency statistics and sentiment analysis.

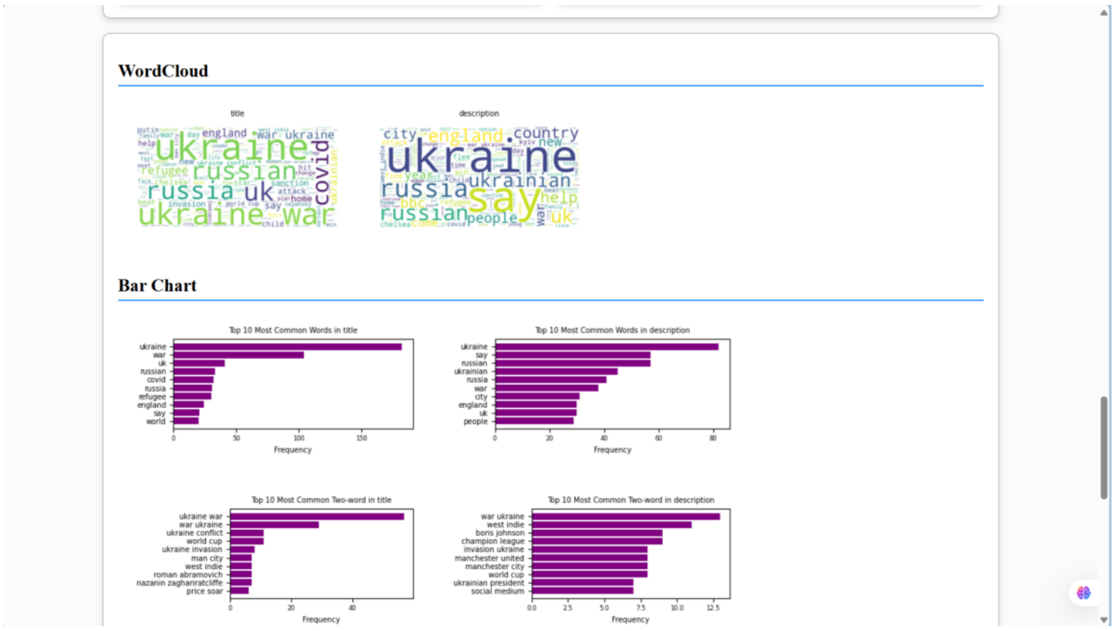
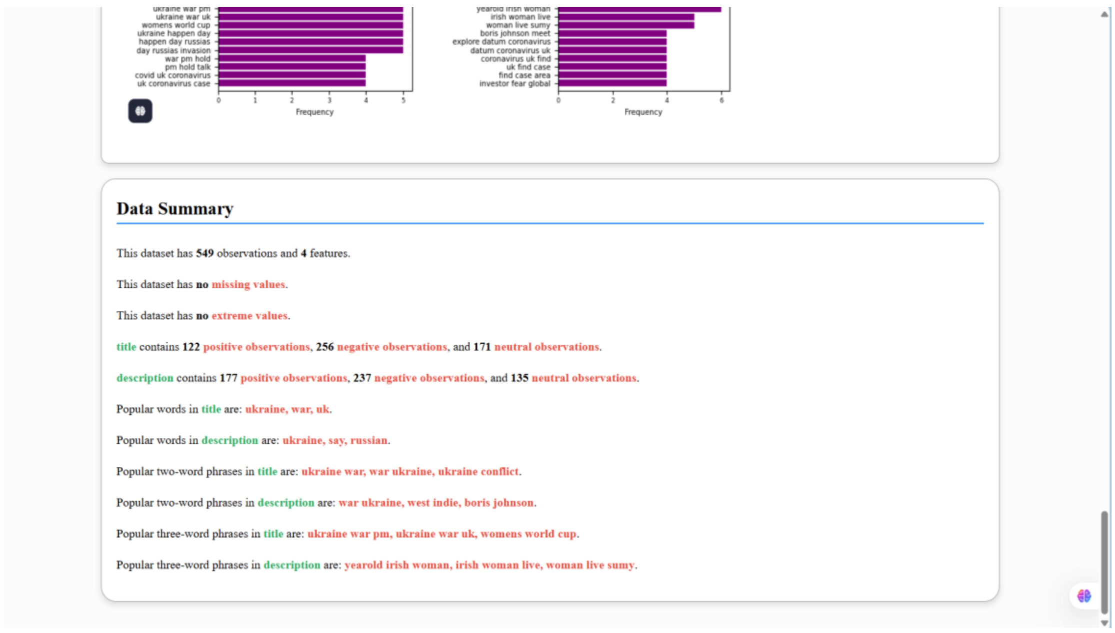


Figure 5-15 Statistics Page (10)

It follows a series of visualizations including word clouds, and n-gram visualizations (unigrams, bigrams, trigrams).



Data Summary

This dataset has **549** observations and **4** features.

This dataset has **no** **missing values**.

This dataset has **no** **extreme values**.

title contains **122** **positive observations**, **256** **negative observations**, and **171** **neutral observations**.

description contains **177** **positive observations**, **237** **negative observations**, and **135** **neutral observations**.

Popular words in **title** are: **ukraine, war, uk**.

Popular words in **description** are: **ukraine, say, russian**.

Popular two-word phrases in **title** are: **ukraine war, war ukraine, ukraine conflict**.

Popular two-word phrases in **description** are: **war ukraine, west indie, boris johnson**.

Popular three-word phrases in **title** are: **ukraine war pm, ukraine war uk, womens world cup**.

Popular three-word phrases in **description** are: **yearold irish woman, irish woman live, woman live sumy**.

Figure 5-16 Statistics Page (11)

Finally, the page concludes with a Data Summary. It provides a concise narrative overview of the dataset based on all the analyses performed above.

5.5 Implementation Issues and Challenges

One of the project's primary challenges is the system's inability to handle massive data sets efficiently. The system may have performance issues such as slow processing, server overload, or even crashes when uploading huge datasets with many rows and columns.

One key problem in visualization is dealing with datasets that contain a huge number of variables and values. When representing such data, graphs can become complex and difficult to read, reducing their effectiveness. It is critical to use methods such as feature selection or dimensionality reduction to simplify the data before visualizing it.

Another major challenge is text variable preprocessing, especially the removal of stop words. Stop words are popular words that contribute no sense to the analysis. However, finding the right lexicon for stop word elimination can be challenging. No single lexicon can completely remove all non-meaningful terms in every situation. Customizing stop word lists while keeping relevant material complicates text preprocessing.

5.6 Conclusion Remark

The implementation of the system successfully integrates both frontend and backend components to deliver an exploratory data analysis tool. The use of Visual Studio Code coupled with Python environment enabled smooth setup, configuration, and execution. The system interface was designed with user experience in mind. It allows users to upload, preview and analyse datasets with ease. Through clear visualizations and automated statistical reporting, users can gain insights into their data efficiently. Despite some challenges such as handling large datasets, complex visualizations and the intricacies of text preprocessing, the system meets its core objectives. Future improvements may focus on optimizing performance, enhancing visualization readability and refining text processing to improve adaptability for broader dataset types and sizes.

Chapter 6 System Evaluation and Discussion

6.1 System Testing and Performance Metrics

6.1.1 User Requirements

1. User should be able to upload dataset through the web interface.
2. After uploading the dataset, the user can proceed by clicking button which triggers the EDA process.
3. User should have the option to cancel the uploaded dataset.

6.1.2 System Requirements

1. The system should receive datasets uploaded by users.
2. The system should save the dataset into a local file storage for further processing.
3. The system should allow users to preview the dataset before initiating the EDA process.
4. The system should allow smooth navigation between upload, preview and analysis stages without losing uploaded data.
5. The system should generate an overview of the dataset.
6. The system should detect missing values in the dataset and report their distribution.
7. The system should filter out non-meaning features.
8. The system should classify dataset features into numerical, categorical, textural types.
9. The system should compute descriptive statistics for all features based on their types.
10. The system should perform unigram, bigram and trigram analysis on textual features.
11. The system should generate appropriate visual representations for each type of feature.
12. The system should compute Pearson correlation for numerical features and Chi-square test results for categorical features.
13. The system should provide a structured summary of the dataset’s key findings.

6.1.3 System Performance Definition

- The system shall complete the EDA process within 1 minutes for datasets up to 30,000 rows and 25 columns to ensure efficient and timely analysis.
- The system shall accurately identify and summarize key dataset characteristics, including:
    - Detection of missing values and duplicate entries
    - Correct classification of features into numerical, categorical and textual types
    - Computation of appropriate descriptive statistics for each feature type
    - Identification of significant correlations among numerical features
    - Extraction of textual insights such as word frequency, n-grams, and summary statistics
- The feature classification and statistical computations shall maintain an accuracy rate of at least 95%.
- The system shall generate clear and readable visualizations without rendering or display errors.

6.1.4 Verification Plan

A comprehensive verification plan has been developed to ensure the correctness, reliability and performance of the system. This plan includes testing the system with various datasets of different sizes and structures to evaluate its ability to handle diverse data scenarios. For initial testing, a small synthetic dataset with approximately 1005 rows containing only numerical and categorical features is used to evaluate the system's ability to perform statistical computations, feature classification, correlation analysis and generate appropriate visualizations such as histograms, boxplots, bar charts and heatmaps. The second dataset consists only textual data. It allows thorough testing of the text preprocessing pipeline including stop word removal, lemmatization, sentence and word tokenization, as well as unigram, bigram and trigram analysis.

Following this, real-world datasets like the Titanic or Iris dataset are employed to confirm the system's accuracy and effectiveness in handling common data structures and to ensure meaningful analytical output is generated.

6.2 Testing Setup and Result

Unit testing is a critical phase in our software development process that aims to verify the correctness and reliability of individual components and modules within our application. This summary provides an overview of the outcomes of our unit testing efforts. Unit testing was conducted on various modules of our application, including dataset overview Module, drop meaningless feature module, feature classification module, missing data module, numeric statistics module and visualization, pearson and heatmap, categoric statistics and visualization and textural statistics and visualization, and data summary.

6.2.1 Testing Setup

The system was tested using two datasets designed to evaluate specific functionalities:

- Dataset1: Customer Spending Behaviour

Feature: Customer_ID, Age, Monthly_Spend, Session_Duration, Product_Reviews, Loyalty_Score, Customer_Segment, Preferred_Payment_Method

Module	**Expected Result**	**Actual Result**	**Status**
Dataset Overview	Correct row/column count, detect missing/duplicate case and compute memory size	All statistics rendered correctly	Pass
Drop Meaningless Feature	Remove ID columns and unrelated features	ID columns dropped as expected	Pass
Feature Classification	Features correctly grouped into numeric/categoric/text	Grouping accurate	Pass
Missing Data	Show null count and visualize	Null statistics and chart match dataset	Pass
Numeric Statistics	Mean, standard deviation, min, max, outlier shown correctly	All statistics accurate	Pass
Pearson & Heatmap	Show correlations and heatmap	Heatmap and correlation matrix match expected	Pass
Categorical Statistics	Frequency counts and proportions	All results accurate	Pass

Chi-Square Test	Table with correct p-values and results	Chi-square values computed accurately	Pass

Table 6-1 Testing Case Result with Dataset1

- Dataset2: BBC News

Feature: title, guid, link, description

Module	**Expected Result**	**Actual Result**	**Status**
Dataset Overview	Correct row/column count, detect missing/duplicate case and compute memory size	All statistics rendered correctly	Pass
Drop Meaningless Feature	Remove guid and link columns and unrelated features	ID columns dropped as expected	Pass
Feature Classification	Features correctly grouped into numeric/categoric/text	Grouping accurate	Pass
Word Frequency Statistics	Accurate frequency list of words	Word frequency results accurate	Pass
Sentiment Analysis	Compute sentiment observations by sentiment score	Sentiment observations appear correctly	Pass
Wordcloud Visualization	Wordcloud reflects word frequencies	Wordcloud matches frequency statistics	Pass
N-Gram Visualizations	Generate unigram, bigram, trigram charts	All charts render correctly	Pass

Table 6-2 Testing Case Result with Dataset2

6.2.2 Testing Result

In conclusion, all the functions and features proposed in this project is built successfully. All positive test cases which validate the expected behaviour of our system have been executed successfully.

The result of these modules are tested and passed. Test metrics are shown as below:

- Total Test Cases Executed: 13
- Test Cases Passed: 13
- Test Cases Failed: 0
- Test Pass Rate: 100%

6.3 Project Challenges

High-Dimensional Data Visualization

When datasets contain numerous variables, visualizing the data becomes increasingly difficult. High-dimensional data can lead to cluttered and overly complex graphs. It makes it challenging for users to extract meaningful insights. Techniques such as dimensionality reduction are essential to simplify the data before visual representation. In the project, pie charts are designed to display only the top four value while the remaining values are grouped into an "Other" category to maintain clarity and readability.

Handling Large Datasets

Another challenge was optimizing the system to handle large datasets efficiently. Uploading and processing datasets with tens of thousands of rows and numerous columns can strain server resources. It results in slow performance or potential crashes.

Text Data Preprocessing and Analysis

Handling text data presented challenges, especially during preprocessing. Removing stop words is essential to reduce noise in the data. However, finding an appropriate lexicon that suits all textual contexts proved to be difficult. Additionally, accurately lemmatizing and tokenizing text in multiple languages or mixed-language datasets required customized pipelines to ensure consistent results. Moreover, generating meaningful n-grams while avoiding nonsensical word combinations posed another challenge.

6.4 Objectives Evaluation

The primary objective of this project was to develop a user-friendly and efficient system for performing Exploratory Data Analysis (EDA) on uploaded datasets. The system aimed to provide users with statistical insights and visual representations of their data. The system successfully achieved this goal by implementing the following key features:

Dataset Upload and Preview: The system allows users to upload datasets with a drag-and-drop interface or a file selection button. After uploading, a preview of the dataset is displayed. It enables users to verify the data before proceeding.

Data Cleaning and Feature Classification: The system automatically detects and removes meaningless features, such as IDs or redundant variables and classifies the remaining features into numerical, categorical, and textual types.

Data Analysis: The system efficiently computes various statistics, including numerical summaries, categorical distributions, correlations and text feature analysis. It accurately handles missing data and provides users with clear visualizations to better understand their datasets.

Visualization: The system generates multiple types of visual representations, including histograms, box plots, pair plots, correlation heatmaps, bar charts, pie charts, word clouds and n-gram graphs. It makes data interpretation more informative.

Data Summary and Insights: The final data summary consolidates key findings from the analysis. It provides users with a clear and concise interpretation of the dataset's characteristics and patterns.

6.5 Concluding Remark

In conclusion, this project successfully developed a comprehensive and user-friendly system for performing Exploratory Data Analysis (EDA) on uploaded datasets. The system meets the primary objectives by providing users to upload, analyse and visualize their data. It offers detailed insights into dataset characteristics.

Throughout the development process, several challenges were encountered, including handling high-dimensional data, processing large datasets and effectively preprocessing text data. These challenges were addressed through the implementation of dimensionality reduction techniques, optimization of data processing and customized text analysis pipelines.

The system was thoroughly tested with various datasets to confirm its robustness and accuracy. All implemented features and functions performed as expected. It achieves a 100% pass rate in unit testing. The final product provides clear data visualizations, accurate statistical computations and intuitive navigation between data upload, preview, and analysis stages.

By integrating data preprocessing statistical analysis, and visualization within a single platform, this system significantly simplifies the EDA process for users. It enhances their ability to explore and understand data efficiently.

CHAPTER 7 Conclusion

7.1 Project Review, Discussion and Conclusion

This project aimed to develop a comprehensive, web-based Exploratory Data Analysis (EDA) system to streamline data preprocessing, analysis and visualization. The system is designed to support diverse datasets including numerical, categorical and textual data. The system enables users to gain valuable insights through automated analysis and visual representation.

The project was driven by the need to simplify the EDA process which can be daunting when datasets are large or contain mixed data types. Traditional manual methods of data exploration can be time-consuming and prone to human error. The proposed system effectively addresses this problem by providing an intuitive interface where users can upload datasets, preview their content and proceed with analysis in a few clicks.

The project successfully implemented a user-friendly web interface that supports both drag-and-drop and manual file uploads. One of the key achievements was automating data preprocessing steps, including the detection and removal of non-meaningful features and the classification of data types into numerical, categorical and textual categories. The system also generated a comprehensive range of statistical outputs and visual representations tailored to each data type, such as histograms, correlation matrices, word clouds and n-grams. Additionally, the system offered detailed summaries to facilitate the interpretation of analytical results.

7.2 Novelties and Contributions

This project introduced several innovative approaches aimed at simplifying and enhancing the Exploratory Data Analysis process. One of the most significant contributions is the Integrated EDA Workflow which seamlessly combines data upload, preprocessing, statistical analysis and visualization within a single web application. Unlike traditional standalone analysis scripts, this integrated approach offers users a smooth experience from data input to insight generation. Another key innovation is the Data Classification feature. The system intelligently categorizes features as numerical, categorical or textual based on the content of the dataset. This dynamic approach not only saves time but also minimizes user errors.

The project also excels in Comprehensive Text Analysis by incorporating advanced natural language processing techniques. Beyond basic text statistics, the system performs unigram, bigram, and trigram analysis to offer valuable insights into text patterns. This capability is particularly beneficial when analysing datasets containing textual fields. It allows users to gain deeper understanding of the underlying text data.

Lastly, the project introduces Automated Data Summary Generation which consolidates the key insights from the analysis into a structured and easily understandable format. This feature greatly assists users in interpreting their data without the need for manual result compilation. It makes the analysis process more efficient and user-friendly.

By streamlining the EDA process and offering automated, intelligent and adaptive analysis features, this project significantly contributes to the field of data analysis. It reduces the complexity associated with exploratory tasks, facilitates faster data understanding and supports more informed decision-making.

7.3 Future Work

While the project successfully achieved its primary objectives, there are several areas where further improvements could significantly enhance the system's functionality and usability. One key area is performance optimization for very large datasets. Integrating parallel processing techniques and leveraging cloud-based computation would enable the system to handle significantly larger datasets with reduced latency.

Another area of potential enhancement is advanced statistical testing. Implementing more sophisticated tests, such as ANOVA or non-parametric methods would increase the system's applicability, particularly when dealing with data that does not conform to normality assumptions or requires comparative analysis across multiple groups.

The system could incorporate customizable preprocessing pipelines to increase flexibility. It allows users to specify their own data cleaning steps, such as selecting stop words or choosing data imputation methods.

The current system primarily supports English text processing, so adding multi-language support for text analysis would make it more universally applicable. Integrating language detection and context-specific stop word lists would ensure accurate text processing across various linguistic contexts.

Lastly, enhancing data export options by allowing users to download processed data, visualizations and generated summaries in multiple formats such as CSV, PDF, or JSON would facilitate easy integration with external reporting and data analysis tools.
By incorporating these enhancements, the system would evolve into a more comprehensive tool for data analysis, better equipped to meet diverse user needs and handle increasingly complex data challenges

REFERENCES

[1] X. Luo *et al.*, “Enhancing statistical charts: toward better data visualization and analysis,” *Journal of Visualization*, vol. 22, no. 4, pp. 819–832, Jun. 2019, doi: 10.1007/s12650-019-00569-2.

[2] M. Derthick, J. Kolojejchick, and S. Roth, “An Interactive Visualization Environment for Data Exploration.” Available: https://cdn.aaai.org/KDD/1997/KDD97-001.pdf

[3] S. Batt, T. Grealis, O. Harmon, and P. Tomolonis, “Learning Tableau: A data visualization tool,” *The Journal of Economic Education*, vol. 51, no. 3–4, pp. 317–328, Aug. 2020, doi: 10.1080/00220485.2020.1804503.

[4] “Data Storytelling with Google Looker Studio: A hands-on guide to using Looker Studio for building compelling and effective dashboards,” *Packt Publishing Books | IEEE Xplore*. https://ieeexplore.ieee.org/document/10162393

[5] “Effective text data preprocessing technique for sentiment analysis in social media data,” *IEEE Conference Publication | IEEE Xplore*, Oct. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8919368?casa_token=91alnDo4jYgAAAAA:jCcmjgBQG99UXODbm4gep6N9p8M1Q4a6EDJzJ_6x3_45fWEZKV-xEakDL0JbpPIaJGSOO_om6QuK

[6] “Analyzing COVID-19 Dataset through Data Mining Tool ‘Orange,’” *IEEE Conference Publication | IEEE Xplore*, Jan. 19, 2021. https://ieeexplore.ieee.org/abstract/document/9357754?casa_token=76VstjtGht4AAAAA:1dios1juE-JkQszc9qCQKBbL98Jx5sotJJQvSJ1RaGoKQv-8XLLxBXb3yWk3MJ0TJQdWsUM

[7] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, “Big data preprocessing: methods and prospects,” *Big Data Analytics*, vol. 1, no. 1, Nov. 2016, doi: 10.1186/s41044-016-0014-0.

[8] “Defining insight for visual analytics,” *IEEE Journals & Magazine | IEEE Xplore*, Apr. 01, 2009. https://ieeexplore.ieee.org/abstract/document/4797511?casa_token=iM3nmEo5PB0AAAAA:V9zl7BvjCubWMeG1BQi3irHIrDQTqGii5sBUuqxPEcX0voPiAgc5UhxJApwifZ24i5U6pxE

[9] D. Ghimire, “Comparative study on Python web frameworks: Flask and Django,” *Theseus*, 2020. https://www.theseus.fi/handle/10024/339796

[10] “Queriosity: Automated data exploration,” *IEEE Conference Publication | IEEE Xplore*, Jun. 01, 2015. https://ieeexplore.ieee.org/abstract/document/7207300?casa_token=PuhCY2y7KlwAAAAA:Yg1cqzOeup2bN5YLv2Scjwp1RNFcVcagk81XBfOzw_1_sCqz_YKUUSduf7lx74TCeBSsjdQ

POSTER

PROJECT DEVELOPER: KOH YU BIN
PROJECT SUPERVISOR: DR TONG DONG LING

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
Wholly owned by UTAR Education Foundation

Automated Exploration Data Analysis Tool

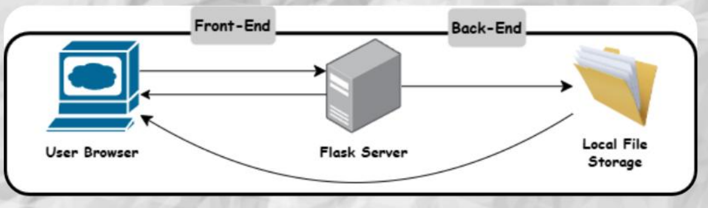
Project Overview

This project focuses on developing a web-based automated EDA tool aimed at simplifying the data analysis process. The primary objective is to automate EDA tasks. The system leverages Python libraries for data processing and visualization while using Flask for backend operations and HTML for the frontend.

Problem Statement

- User require control the EDA process manually
- Existing EDA tools has limited ability to interpret text data
- Existing EDA tools are not generating summary at the end of EDA

System Framework



- Browser sends the data with HTTP request to the Flask server.
- Flask server processes the data and saves it to the local file storage
- Flask backend passed the results to the browser using Jinja2.

Objectives

- to automate the entire EDA process
- to enhance EDA to interpret text data
- to generate summary at the end of EDA

Tools Used

python™ Flask HTML5 CSS { }

Result