A

Minor Project Report

On

DYNAMIC DATA VISUALIZATION DASHBOARD

Submitted in partial fulfillment of the requirements for the award of Degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

by

B. POORVAJA DHANUSHREE (228R1A67D5)

ALAVALA SRAVANI (228R1A67D2)

M.VINAY KUMAR (228R1A67G3)

N. NAVYA SRI (228R1A67H6)

Under the Guidance of

MRS. M. CHANDANA

Assistant Professor, CSE-DATA SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401)

2024-2025

CMR ENGINEERING COLLEGE

(UGC Autonomous)

(Accredited by NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad)
Kandlakoya, Medchal Road, Hyderabad-501 401)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

CERTIFICATE

This is to certify that the project entitled **"DYNAMIC DATA VISUALIZATION DASHOARD"** is a bonafide work carried out by

B. POORVAJA DHANUSHREE	**(228R1A67D5)**
ALAVALA SRAVANI	**(228R1A67D2)**
M.VINAY KUMAR	**(228R1A67G3)**
N. NAVYA SRI	**(228R1A67H6)**

In the degree of B. Tech in Computer Science and Engineering (Data Science) to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by them under our guidance and supervision during the year 2024-25.

Internal Guide	**Mini Project Coordinator**	**Head of the Department**
Mrs. M. Chandana	**Mr. E. Laxman**	**Dr. M. Laxmaiah**
Assistant Professor	Assistant Professor	Professor & H.O. D
CSE(DS), CMREC	CSE(DS), CMREC	CSE(DS), CMREC

DECLARATION

This is to certify that the work reported in the present project entitled **“Dynamic Data Visualization Dashboard**” is a record of bona fide work done by us in the Department of Computer Science and Engineering, CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

.

B. POORVAJA DHANUSHREE	**(228R1A67D5)**
ALAVALA SRAVANI	**(228R1A67D2)**
M.VINAY KUMAR	**(228R1A67G3)**
N. NA VYA SRI	**(228R1A67H6)**

ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr.M. Laxmaiah**, **HOD**, **Department of CSE-DATA SCIENCE, CMR Engineering College** for their constant support**.**

We are extremely thankful to **Dr. M. LAXMAIAH,** Professor & H.O.D, Internal Guide, Department of **CSE-DATA SCIENCE**, for his constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if I do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **Mr.E.Laxman** Assistant Professor, Mini Project Coordinator for his constant support in carrying out the project activities and reviews.

We express my thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time**.**

B. POORVAJA DHANUSHREE	**(228R1A67D5)**
ALAVALA SRAVANI	**(228R1A67D2)**
M.VINAY KUMAR	**(228R1A67G3)**
N. NAVYA SRI	**(228R1A67H6)**

CONTENTS

ABSTRACT

Understanding complex datasets is made simpler when data is presented in an interactive, visually appealing manner. This project is dedicated to creating an interactive dashboard using Python's Matplotlib and Seaborn libraries that brings data to life. The aim is to facilitate dynamic data exploration and provide clear insights that support informed decision-making.
To achieve this, the dashboard combines static and dynamic visual elements, enabling users to filter and drill down into specific datasets. The design integrates interactivity with responsive charts and graphs that update in real time, ensuring that the presentation remains intuitive and informative. The technical approach leverages Python's robust visualization libraries to achieve seamless interactivity.

Users will benefit from a tool that enhances data interpretation and engagement by transforming raw data into clear visual insights. In summary, the dashboard simplifies the process of understanding complex data while offering room for future expansion with additional interactive features.

LIST OF FIGURES

1. INTRODUCTION

1.1 Introduction &Objectives

In today's data-driven world, the ability to effectively interpret and communicate data insights is essential across nearly every field, from business and healthcare to science and education. As data becomes increasingly complex and voluminous, traditional static visualizations often fall short in providing the depth of analysis needed to uncover patterns, trends, and anomalies. This has led to a growing demand for **interactive and dynamic data visualization tools** that offer more flexibility and user engagement.

This project explores the development of a dynamic data visualization dashboa**rd** using two widely-used Python libraries—Matplotlib and Seaborn. Matplotlib serves as the foundational plotting library in Python, offering extensive capabilities for creating static, animated, and interactive plots. Seaborn, built on top of Matplotlib, simplifies the creation of attractive and informative statistical graphics, making it ideal for exploratory data analysis.

By combining the strengths of Matplotlib's interactivity and Seaborn's statistical plotting features, the dashboard serves as a lightweight yet powerful solution for dynamic data analysis. It caters to data scientists, analysts, students, and educators who seek a customizable and responsive environment to explore datasets and make informed decisions based on visual evidence.

1.2 Project Objectives

The primary aim of this project is to develop an interactive and user-friendly data visualization dashboard using Python's Matplotlib and Seaborn libraries. The specific objectives include:

1. To design an intuitive interface that allows users to upload and interact with their own datasets without requiring programming expertise.
2. To implement dynamic visualization tools using Matplotlib and Seaborn, including but not limited to line plots, bar charts, scatter plots, histograms, box plots, and heatmaps.
3. To enable real-time interactivity through features such as filter controls, dropdown menus, sliders, and selection tools for dynamic updates of plots based on user input.
4. To support exploratory data analysis (EDA) by visually summarizing the distribution, relationships, and trends present in the dataset.
5. To ensure modularity and scalability so that the dashboard can be extended in the future with additional features like time-series animation, machine learning insights, or integration with web frameworks.
6. To promote open-source accessibility, providing clear documentation and clean code so that students, analysts, and researchers can adapt and reuse the dashboard in various data analysis contexts.

1.3 Purpose of the Project

The purpose of this project is to create a dynamic, interactive dashboard that simplifies the process of visual data analysis. By utilizing the capabilities of Matplotlib and Seaborn, the project aims to empower users—especially those without advanced programming skills—to explore datasets, uncover trends, and gain insights through visual storytelling.

This tool is intended to bridge the gap between raw data and informed decision-making by making data exploration more accessible, engaging, and insightful. Whether used in education, research, or industry, the dashboard serves as a practical solution for visualizing complex data in a way that is both intuitive and customizable.

Ultimately, the project promotes a deeper understanding of data through visual interaction, enabling users to make better, evidence-based conclusions.

1.4 Existing System with Disadvantages

The current data visualization tools like Power BI, Tableau, and Google Data Studio, though powerful, have several limitations. Most of these platforms require paid licenses to unlock essential features such as real-time collaboration, data export, or advanced filtering. They offer limited flexibility for customization and often depend on cloud services, leading to vendor lock-in. These tools can be resource-heavy, making them unsuitable for low-end systems or quick prototyping. Additionally, their proprietary scripting and configuration systems pose a steep learning curve for beginners or non-technical users, restricting accessibility for small teams and students

Disadvantages

- **High cost**

Popular tools like Power BI and Tableau require paid licenses for full features, making them expensive for small businesses or students.

- **Limited Free Features**

Free versions often have restrictions on data size, refresh rates, sharing, and collaboration options.

- **Lack of Real-Time Updates**

Some systems do not support live data integration or have limited refresh intervals.

- **Low Customizability**

Dashboards may have limited flexibility for user-specific customization without advanced knowledge

1.5 Proposed System with Features

The proposed system is an **interactive, dynamic, and customizable data visualization dashboard** built using **Python, Streamlit, and Plotly**. It is designed to empower users—especially data analysts, students, and business users to **upload datasets, filter data,** and **generate various types of insightful visualizations** effortlessly. The system is open-source, lightweight, and completely free of cost, eliminating the licensing and subscription barriers posed by commercial tools like Power BI.

Key Features:

1. **CSV Upload Interface**
    - Users can upload their own CSV datasets through a user-friendly sidebar interface.
    - File validation and success notifications ensure smooth data onboarding.
2. **Theme Toggle (Light/Dark Mode)**
    - Offers dynamic theming to switch between light and dark mode, enhancing the visual experience for different environments or user preferences.
3. **Automatic Data Profiling**
    - Displays quick metrics such as the number of numeric and categorical columns, and total rows, providing instant insights into the dataset.
4. **Advanced Filtering Options**
    - Categorical filters using multiselect options.
    - Numeric range filters using interactive sliders.
    - Real-time filtering applied to all visualizations and data views.
5. **Live Filtered Data Preview**
    - A real-time preview of the filtered dataset is shown in a stylized and interactive data table, aiding exploratory analysis

1.6 INPUT AND OUTPUT DESIGN

INPUT DESIGN

The input design defines how users interact with the dashboard and provide data to the system. It aims to ensure **user-friendliness, accuracy**, and **flexibility**, while supporting various datasets and customization options. It uses multiple interactive components from Streamlit such as file uploaders, dropdowns, sliders, and text inputs.

Input Components and Description:

1. **File Uploader (st.file_uploader)**
   - Users upload their own dataset in CSV format.
   - Ensures only .csv files are accepted.
   - Upload status is shown with success messages.
2. **Theme Selection (st.radio)**
   - Users can switch between Light and Dark themes.
   - Affects overall dashboard and chart appearance.
3. **Filter Controls**
   - **Categorical Filters (st.selectbox, st.multiselect)**
     - Users can choose a category column and filter specific values (e.g., Region, Product).
     - Dynamically populated based on uploaded data.
   - **Numeric Range Filters (st.slider)**
     - Allows users to define a range (min–max) for selected numeric columns.
     - Useful for focusing on specific value ranges (e.g., Price between 100–500
4. **Chart Configuration Inputs**
   - Users can choose:
     - **Chart Type**: Scatter, Line, Bar, Pie, Heatmap, etc.
     - **Axes Selection**: Select which columns represent X and Y axes.
     - **Color Grouping**: Select categorical data for coloring points.
     - **Chart Title**: Enter custom title for the visualization.
     - **Chart Size**: Choose between Small, Medium, or Large.
5. **Reset Filters Button**
   - A button that clears all filters and reruns the dashboard from the beginning.
   - Helps users start a new analysis session easily.

OUTPUT DESIGN

The output design is focused on **delivering meaningful, interactive, and real-time results** to the user based on their inputs. It uses Streamlit and Plotly components for dynamic data display and advanced visualizations.

Output Components and Description:

1. **Upload Confirmation**
   - Upon uploading the CSV, the user receives a green success message indicating readiness.
2. **Data Summary (st.metric)**
   - Displays real-time metrics including:
     - Number of numeric columns
     - Number of categorical columns
     - Total number of rows
3. **Filtered Data Table (st.dataframe)**
   - Displays the dataset after applying selected filters.
   - Styled with highlighted min/max values to assist visual comparison.
4. **Dynamic Charts (st.plotly_chart)**
   - Based on user configuration, the dashboard generates interactive charts.
   - Plotly features like zooming, tooltips, hover labels, and legends enhance the analytical experience.
5. **Download Filtered Data (st.download_button)**
   - Exports the currently filtered data as a downloadable .csv file.
   - Useful for offline analysis or sharing filtered datasets.
6. **Theme-Sensitive Layout**
   - Output elements such as charts and tables adapt based on selected theme (Light/Dark).
   - Ensures clarity and professional design under both modes.

2. LITERATURE SURVEY

“A Survey on ML4VIS: Applying Machine Learning Advances to Data Visualization” by Wang, Chen, Wang, and Qu (December 2020, revised December 2021) - offers a comprehensive review of 88 research papers to examine how machine learning (ML) can be integrated into the data visualization pipeline. The authors categorize the role of ML across seven core areas, including data–visual mapping, interaction modeling, and user profiling. These areas enable automatic selection of visual encodings based on data type, learn from user interactions to improve recommendations, and adapt visualizations to individual user preferences. The study also emphasizes ML's ability to support insight recommendation, data reduction, and system optimization. Together, these contributions pave the way for intelligent dashboard systems that can automatically generate charts, suggest filters, and personalize views, significantly enhancing user experience and decision-making.

In **“AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization”, Wu, Wang, Shu, Moritz, Cui, Zhang, Zhang, and Qu (February 2021)** - expand the scope from ML to broader AI techniques in the context of visualization. This survey treats visualizations themselves as data, opening up the possibility for AI to analyze, manipulate, and generate visual content just as it does with text or images. The authors introduce a taxonomy covering representation learning, insight extraction, and visualization-style adaptation. These approaches enable systems to extract meaningful patterns automatically, adjust visualization styles dynamically, and tailor insights to user needs. AI4VIS highlights the potential for dashboards to evolve in real-time, adapting to both changing data and user preferences, and marks a shift toward more intelligent, responsive, and autonomous visualization systems.

“Dashboard Design Patterns” (2022) by Benjamin Bach, Nathalie Henry Riche, Christophe Hurter, Bongshin Lee, and Sheelagh Carpendale. takes a practical approach by analyzing 144 existing dashboard implementations to uncover recurring patterns and best practices. The research identifies common design elements such as filter panels, drill-down features, synchronized multi-chart views, and guided narratives. These elements contribute to improved usability, better data exploration, and enhanced storytelling within dashboards. The paper’s insights serve as valuable guidelines for designing user-friendly dashboards that support interactive data exploration. The recommended patterns directly inform the development of features like dynamic filters, coordinated chart updates, and clear, informative layouts—core elements in any effective dashboard solution.

“Creating Dynamic Dashboards with Streamlit” by Mohammad Khorasani (2020) serves as a practical guide for building interactive dashboards using the Streamlit framework. The tutorial focuses on Streamlit’s ability to handle real-time updates using placeholder and rerun functions. This enables the creation of dashboards that respond instantly to user input and changing data. Later, in **“Building Interactive Dashboards with Dynamic Data Using Streamlit”** (Nagarajan, October 2024), the concepts are extended further by incorporating simulations of live data streams and real-time chart updates. The updated work demonstrates how developers can build filter-driven dashboards that dynamically respond to user interactions and changing datasets. These practical contributions make it easier to deploy intelligent dashboards with minimal effort, especially in production environments.

3. SOFTWARE REQUIREMENTS ANALYSIS

3.1 Problem Specification

In today's data-driven environment, organizations generate large volumes of data from various sources such as sales records, web analytics, IoT sensors, and social media platforms. However, traditional data visualization methods—often relying on static tools like Excel, Tableau, or Power BI—are not equipped to handle real-time data updates, interactive filtering, or complex customizations without significant cost or effort.

These existing systems suffer from limitations including high licensing fees, limited scalability, manual data updates, and restricted interactivity. They often lack the flexibility to integrate with live data sources or support dynamic user interfaces tailored to specific business needs. As a result, decision-makers may not receive timely insights, which can hinder quick response to operational or strategic changes.

The problem, therefore, is the absence of a lightweight, cost-effective, and fully customizable system that can provide real-time, interactive visualizations tailored to specific datasets and user preferences.

3.2 Modules and Their Functionalities

1. Data Input Module

Functionality:

- Loads data from various sources such as CSV files, Excel files, SQL databases, or APIs.
- Validates the structure and format of incoming data.
- Preprocesses data (e.g., handling missing values, date formatting, type conversion).

2. Data Processing Module

Functionality:

- Uses **Pandas** to clean, transform, and organize raw data for analysis.
- Aggregates data based on selected filters (e.g., by region, date, product).
- Supports custom groupings, sorting, and summary statistics.

3. User Interface (UI) Module

Functionality:

- ☐ Provides an intuitive layout using **Streamlit** or **Dash**.
- ☐ Displays dropdowns, sliders, date pickers, and checkboxes for dynamic filtering.
- ☐ Reactively updates visualizations based on user input.

4. Visualization Module

Functionality:

- Creates interactive visualizations using **Plotly** (or other libraries like Altair/Matplotlib).
- Supports line charts, bar charts, pie charts, area charts, and possibly map-based visuals.
- Dynamically updates graphs in real time based on filtered data.

5. Deployment Module

Functionality:

- ☐ Prepares the app for deployment on platforms like Streamlit Cloud, Heroku, or Render.
- ☐ Ensures responsive UI and secure access from any device.
- ☐ Handles environment variables, requirements, and scalability considerations.

3.3 Functional Requirements

1. **Data Upload and Integration**
   - The system shall allow users to upload data in formats such as CSV or Excel.
   - The system shall support connecting to external data sources like SQL databases or REST APIs.
   - The system shall automatically load and refresh data at regular intervals (if configured).
2. **Data Preprocessing**
   - The system shall clean and preprocess uploaded data (handle missing values, convert data types, etc.).
   - The system shall allow grouping, filtering, and sorting of data based on user-selected parameters.
3. **Interactive Dashboard Interface**
   - The system shall display an interactive web interface with user-friendly controls (e.g., dropdowns, date pickers).
   - The system shall allow users to apply filters to visualize specific subsets of data.
   - The system shall dynamically update visualizations based on user input.

4. **Data Visualization**
    - The system shall generate charts such as bar graphs, line charts, pie charts, and tables using libraries like Plotly.
    - The system shall update visualizations in real time or on user-triggered refresh.
    - The system shall display key performance indicators (KPIs) and summary statistics.
5. **User Actions and Output**
    - The system shall allow users to download filtered data in CSV format.
    - The system shall support exporting reports or visualizations as images or PDFs.
    - The system may allow users to reset filters and reload original data.
6. **Deployment and Access**
    - The system shall be accessible through a w.eb browser without installation.
    - The system shall ensure responsive design for desktop and mobile devices

3.4 Non Functional Requirements

1. Performance Requirements

- The system shall load and display visualizations within **3 seconds** for standard datasets.
- The dashboard shall support datasets with up to **100,000 records** without significant performance degradation.
- Data filtering and visualization updates shall occur **in near real-time** upon user interaction.

2. Scalability

- The system shall be scalable to support increasing volumes of data and additional data sources.
- It shall be able to integrate with more charts, modules, or external services in the future without requiring a full redesign.

3. Usability

- The user interface shall be clean, intuitive, and easy to navigate for both technical and non-technical users.
- Tooltips, labels, and interactive guides shall be available to help users understand the dashboard's functionality.

4. Reliability & Availability

- The system shall be available at least **99% of the time** during operational hours.
- It shall handle unexpected input gracefully and provide error messages when data format issues or connection failures occur.

5.Maintainability

- The system codebase shall follow modular design principles to allow for easy updates and bug fixes.
- Documentation for both users and developers shall be provided to support maintenance and future development.

6.Security (Optional, for deployed dashboards)

- The system shall restrict access to data and dashboard features based on user roles (e.g., admin, viewer).
- Sensitive information, if any, shall be encrypted and securely transmitted.

3.5 FEASIBILITY STUDY

A feasibility study assesses various aspects of the proposed system to determine its viability. The main types of feasibility considered in this project are **technical, economic, operational,** and **schedule feasibility.**

1. **Technical Feasibility** : The project is technically feasible as it leverages widely-used, well-supported, and open- source Python libraries such as **Pandas**, **Plotly**, **Dash**, and **Streamlit**. These tools allow for fast development, real-time interaction, and integration with various data sources such as CSV files, databases, and APIs. Additionally, no advanced hardware is required—the dashboard can run on standard systems and be accessed via any modern web browser.

2. **Economic Feasibility** : The dashboard is cost-effective since it relies on open-source software, eliminating the need for expensive commercial licenses (like Tableau or Power BI). Hosting can be done on low- cost or free platforms such as Streamlit Cloud, Heroku, or Render, making it financially feasible for individuals, startups, or educational use.

3. **Operational Feasibility** : The system is user-friendly and intuitive, even for non-technical users. With its interactive filters, visual feedback, and web-based accessibility, it is designed for ease of use. Organizations and users can adapt to the system with minimal training, and it can easily be integrated into existing workflows.

4. SOFTWARE AND HARDWARE REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS:

Operating System	:	Windows 10 / macOS / Linux (64-bit)
Coding Language	:	Python 3.8+
Development Tool	:	Visual Studio Code / Jupyter Notebook / PyCharm
Database	:	SQLite3 / APIs / CSV / Excel
Libraries Used	:	Pandas – for data manipulation and analysis
Framework	:	Streamlit (easy & fast) / Dash (customizable layout)

4.2 HARDWARE REQUIREMENTS

System	:	Intel Core i5 or higher
RAM	:	16 GB
Hard Disk	:	1 TB (SSD preferred)
Monitor	:	15" LED (Full HD recommended)
Input Devices	:	Keyboard, Mouse
Screen Resolution	:	1366 x 768 (1920 x 1080 recommended)

5. SOFTWARE DESIGN

5.1 SYSTEM ARCHITECTURE:

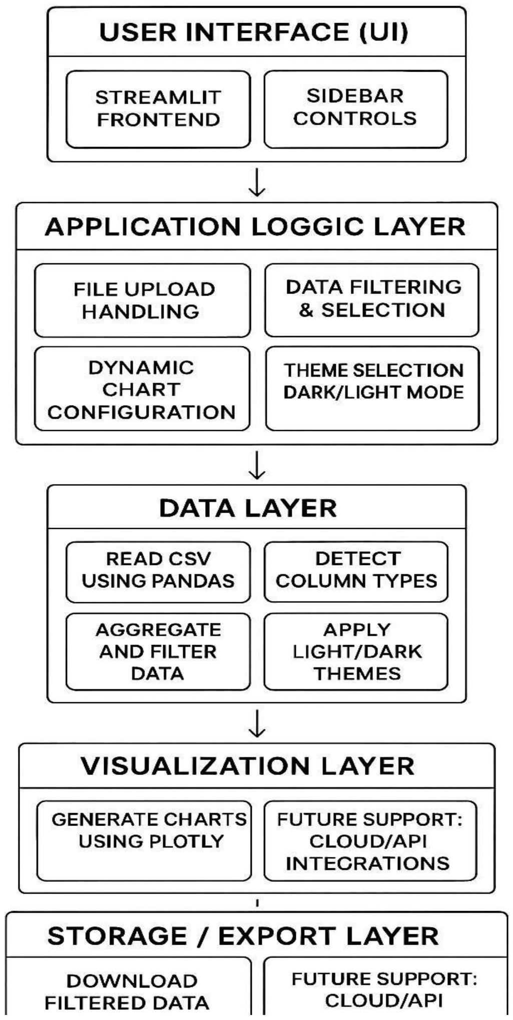


Figure5.1System Architecture

1. User Interface (UI) Layer:

At the top of the architecture is the **User Interface (UI) Layer.** This is where users interact with the application. The UI is built using Streamlit, a popular Python framework for creating interactive web apps. The interface features sidebar controls for user input (such as file uploads, filter selections, or chart preferences), and displays the resulting charts using Plotly, a powerful graphing library. The UI is designed to be intuitive, responsive, and visually appealing, providing users with easy access to all core functionalities.

2. Application Layer

Beneath the UI lies the **Application Logic Layer**. This layer acts as the brain of the application, handling all user interactions and business logic. It manages file uploads, processes user selections for filtering or configuring data, and dynamically updates chart settings based on user input. Additionally, it allows users to switch between light and dark themes, ensuring a personalized experience. This layer ensures that the application responds intelligently to user actions and maintains a seamless workflow.

3. Data Layer

The **Data Layer** is responsible for all data-related operations. Here, uploaded CSV files are read using Pandas, a robust data analysis library in Python. The application automatically detects column types (e.g., numerical, categorical), aggregates and filters data as needed, and calculates summary metrics for display. This layer ensures that data is properly prepared and structured before visualization, enabling accurate and meaningful insights.

4. Visualization Layer

Next is the **Visualization Layer**, which brings data to life. Using Plotly, the application generates interactive charts and graphs based on the processed data. This layer also manages chart customization, such as adjusting size, colors, and interactivity options. It applies the selected theme (light or dark) to the visualizations, ensuring consistency with the user's preferences and enhancing the overall user experience.

5. Storage /Export Layer

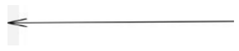
At the base of the architecture is the **Storage / Exp ort Layer**. This layer provides options for users to download their filtered or processed data as CSV files for offline analysis or sharing. It also hints at future enhancements, such as integrating cloud storage or APIs for more advanced data export and sharing capabilities

5.2 Data Flow Diagrams

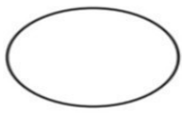
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The Basic Notation used to create a DFD's are as follows:

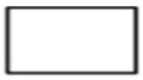
1. Dataflow: Data move in a specific direction from an origin to a destination.



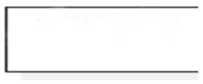
2. Process: People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.



3. Source: External sources or destination of data, which may be People, programs.



4. Data Store: Here data are stored or reference by a process in the System.



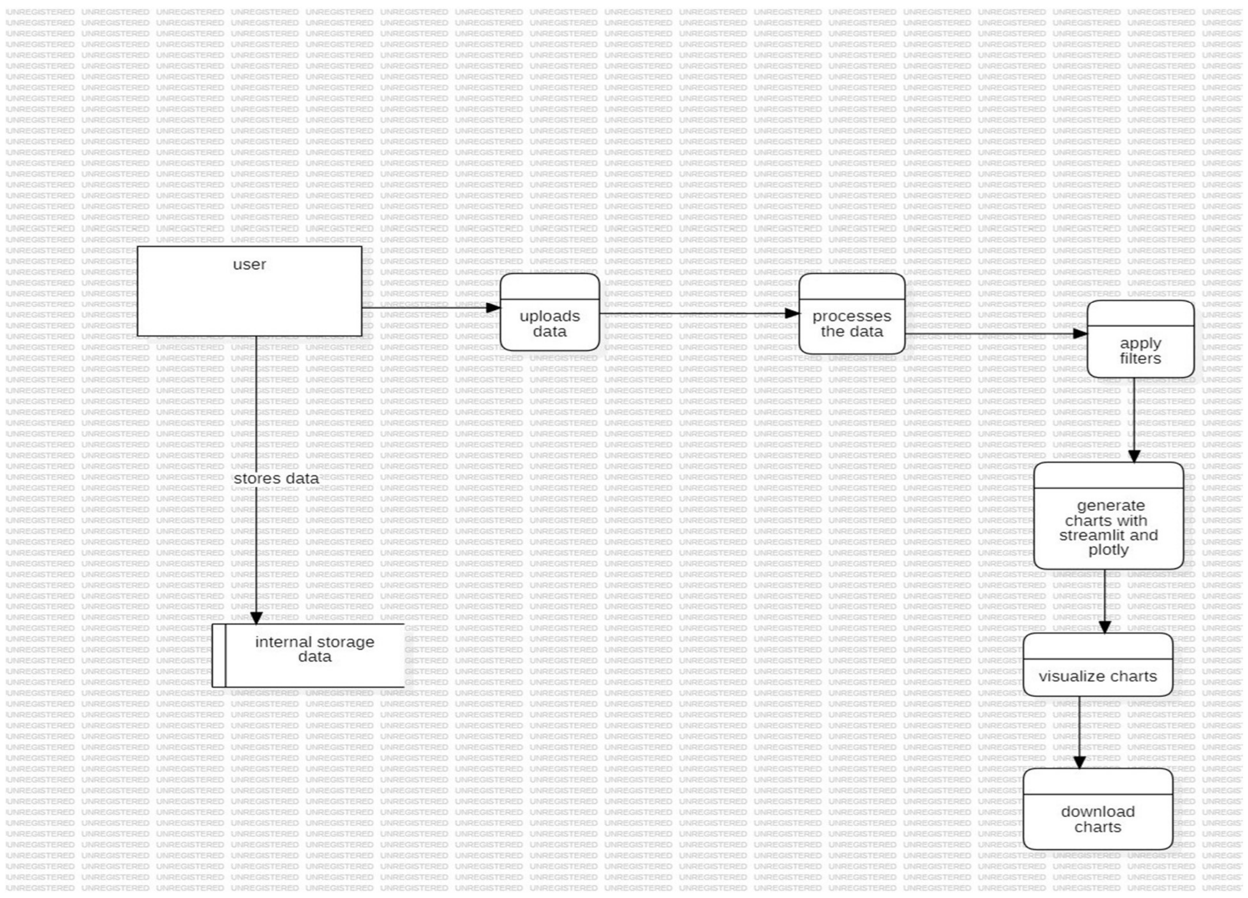


Figure 5.2 Data flow diagram

What is UML diagram?

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation).

The two broadest categories that encompass all other types are:

1. Behavioral UML diagram.
2. Structural UML diagram.

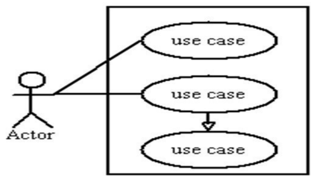
What is a UML Use Case Diagram?

Use case diagrams model the functionality of a system using actors and use cases. Use cases are services or functions provide by the system to its users.

Basic Use Case Diagram Symbols and Notations

System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

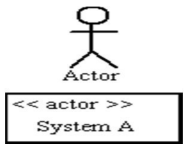


UseCase

Draw use cases using ovals. Label with ovals with verbs that represent the system's function.

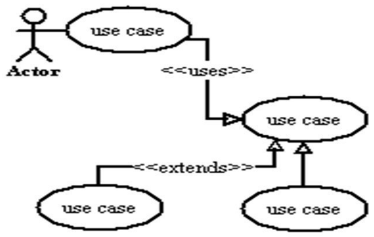
Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.



Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labelled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task.



What is a UML activity diagram?

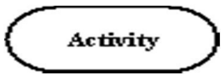
Activity Diagram

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special king of state chart diagram, it uses some of the same modeling conventions.

Basic Activity Diagram Symbols and Notations

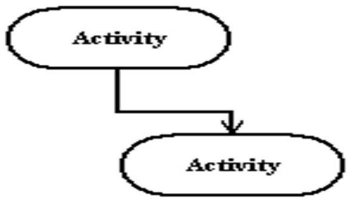
Action states

Action states represent the non-interruptible actions of objects. You can grow an action state in Smart Draw using a rectangle with rounded corners.



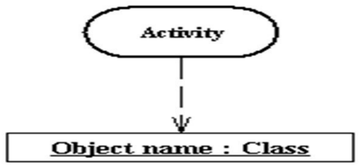
Action Flow

Action flow arrows illustrate the relationships among action states.



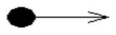
Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



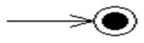
Initial State

A filled circle followed by an arrow represents the initial action state.



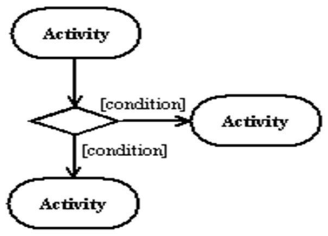
Final State

An arrow pointing to a filled circle nested inside another circle represents the final action state.



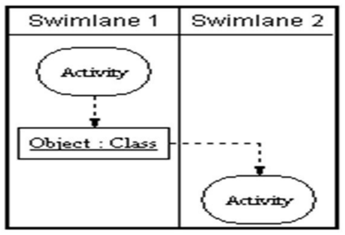
Branching

A gaining represents a decision with alternate paths. The outgoing alternates should be labelled with a cognition or guard expression.

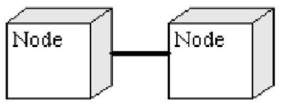


Swim lanes

Swim lanes group related activities into one column.



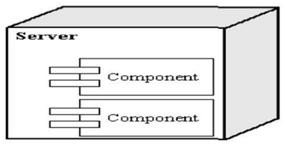
Synchronization



how to connect two nodes.

Components and Nodes

Place components inside the node that deploys them.



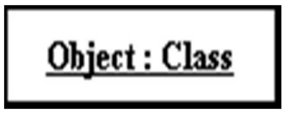
What is a UML sequence diagram?

Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages overtime

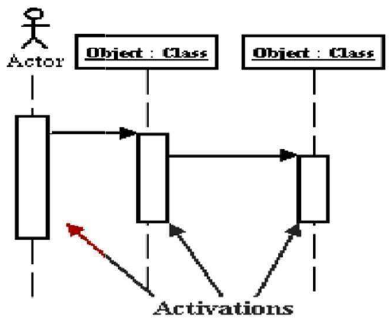
Basic Sequence Diagram Symbols and Notations Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



Activation

Activation boxes represent the time an object needs to complete a task.



5.3. E-R Diagram

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

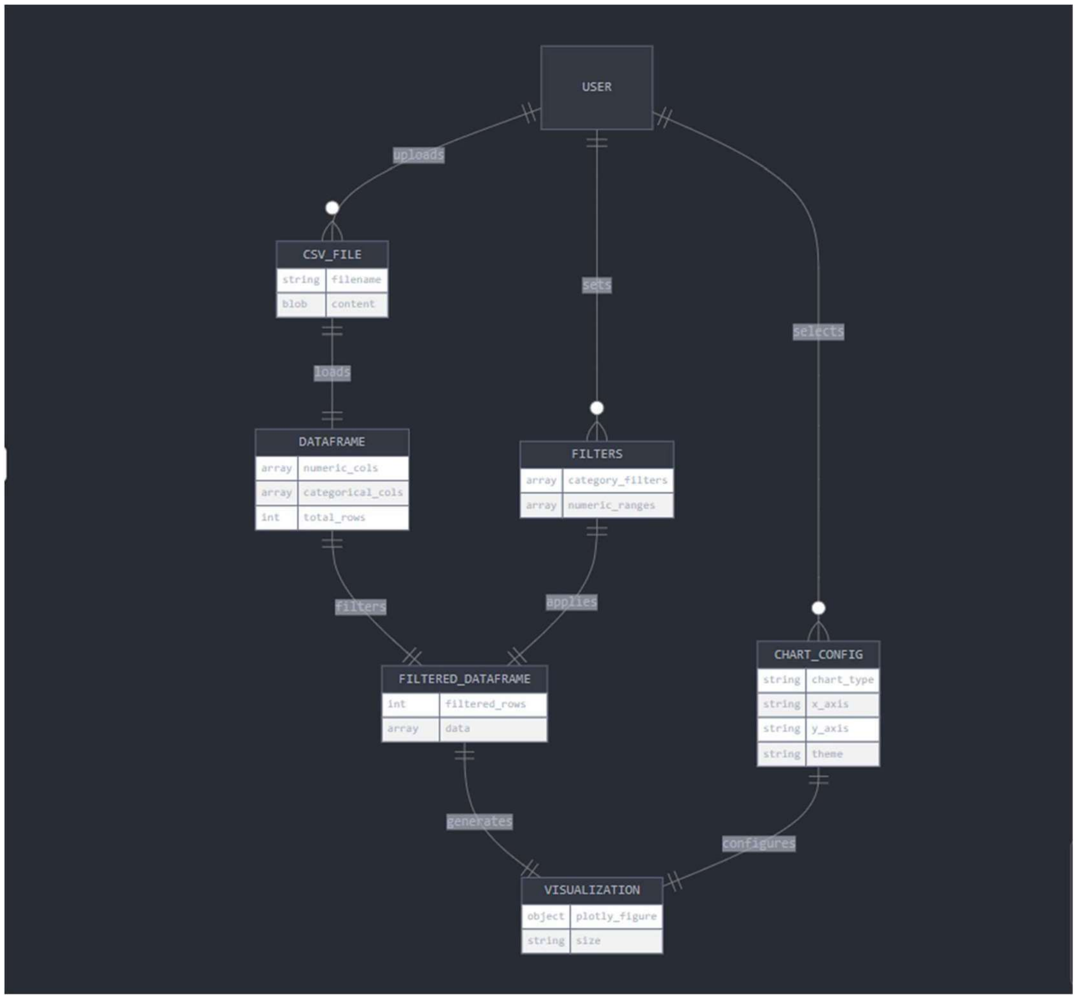


Figure 5.3 E-R diagram

5.4 UML Diagrams

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation). UML has a direct relation with object oriented analysis and design. After some standardization, UML has become an OMG standard.

The two broadest categories that encompass all other types are:

1. Behavioral UML diagram
2. Structural UML diagram.

As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components.

The different types are broken down as follows:

1. Sequence diagram
2. Use case Diagram
3. Activity diagram

1. Sequence diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems

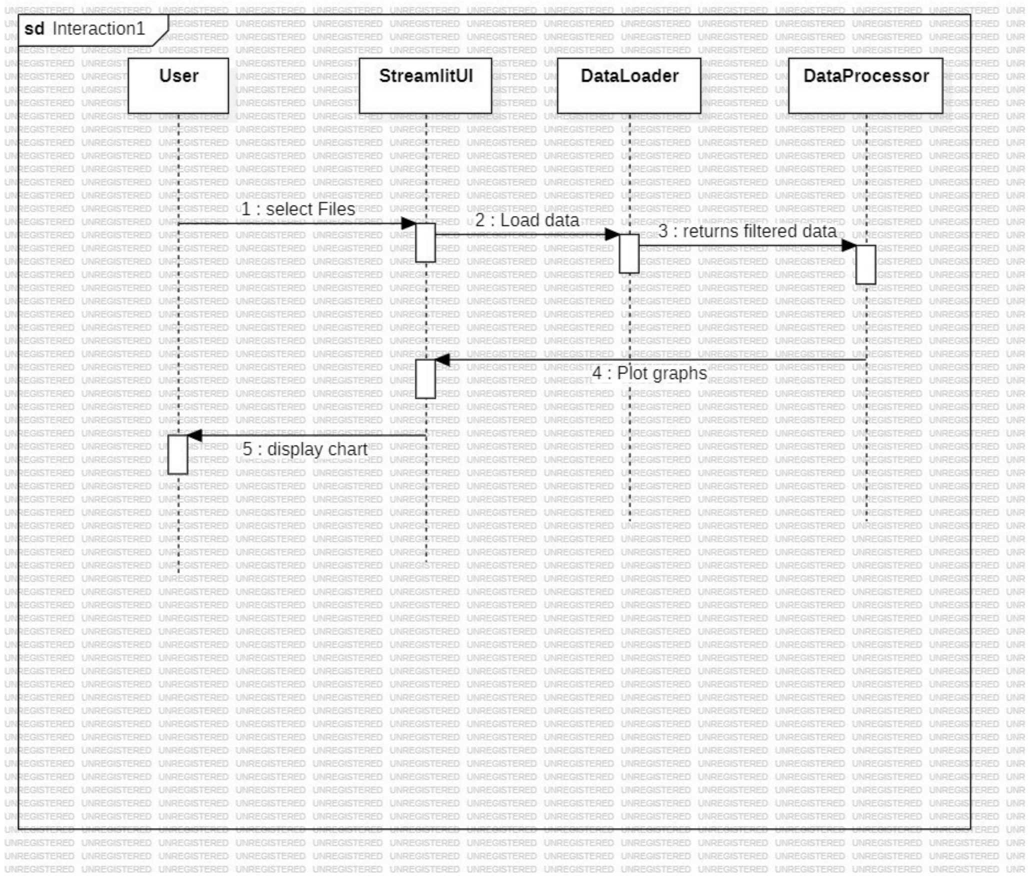


Figure 5.4.1 Sequence diagram

2. Use case diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use case in which the user is involved. A use case diagram is used to structure of the behavior thing in a model. The use cases are represented by either circles or ellipses.

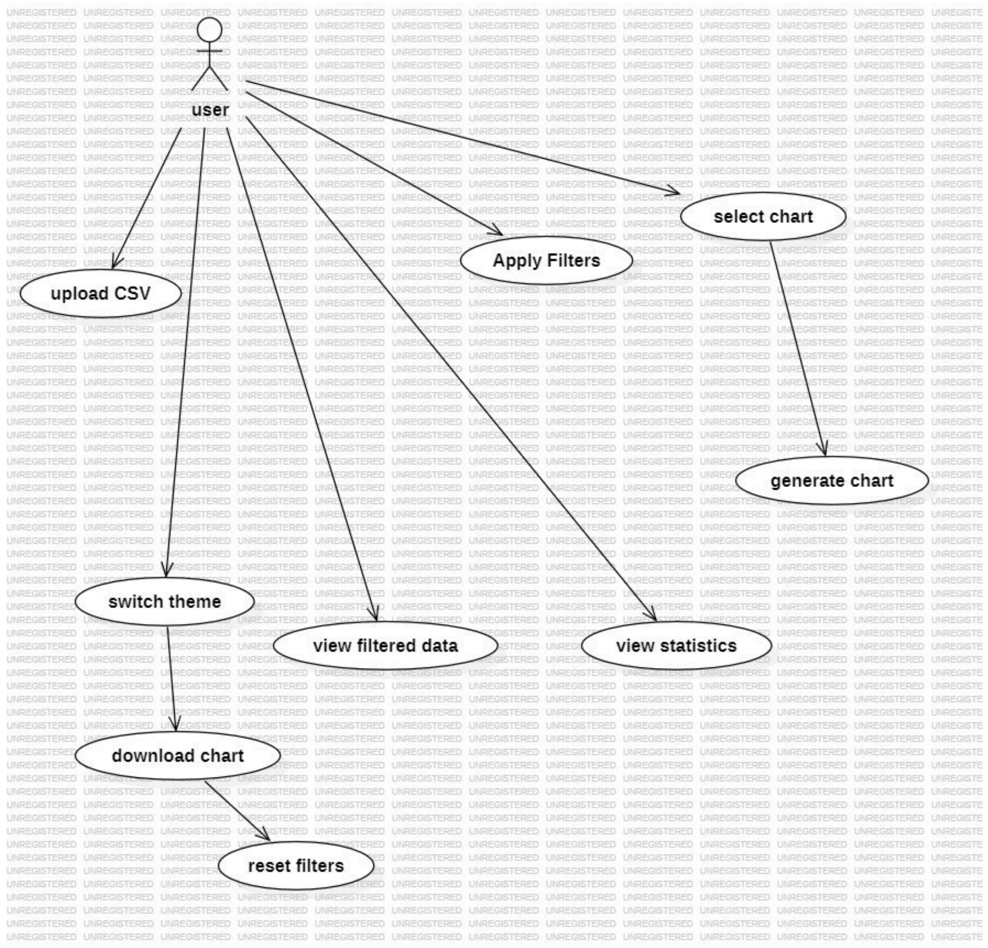


Figure 5.4.2 use case diagram

3. Activity diagram

Activity diagram is another important diagram in UML to describe the dynamic aspectsof the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

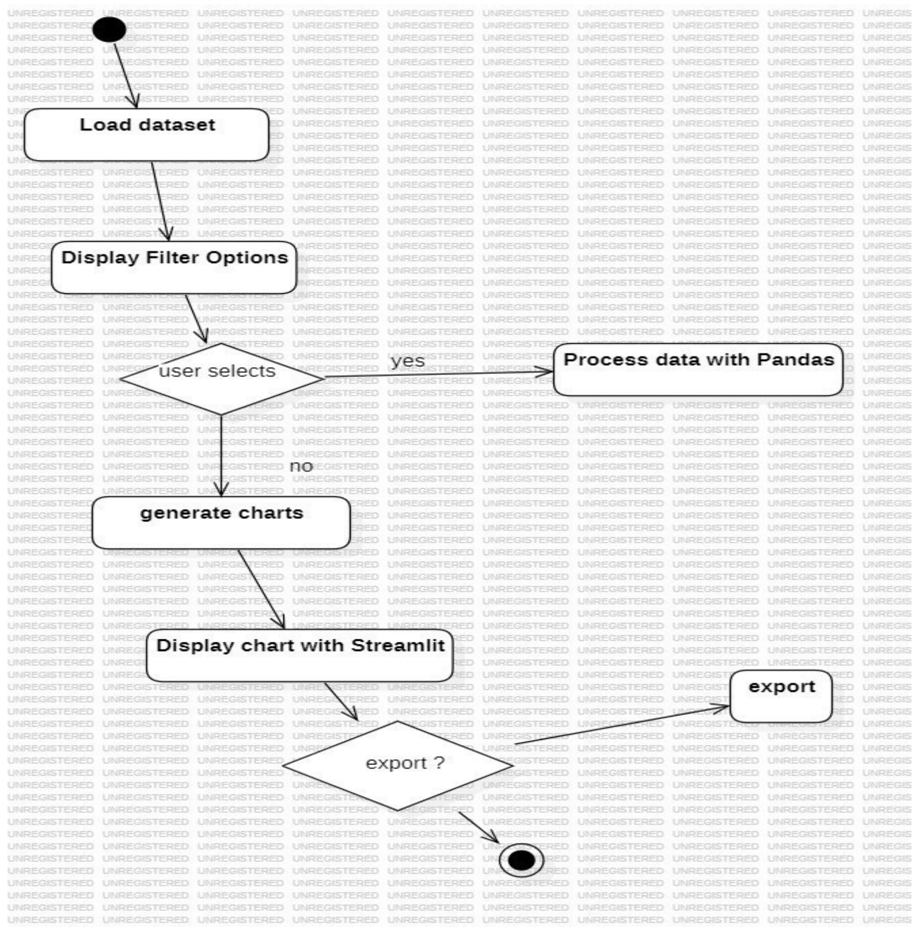


Figure 5.4.3 activity diagram

6. CODING AND ITS IMPLEMENTATION

6.1 Source code

```
import streamlit as st
import pandas as pd
import   plotly.express   as   px
import plotly.graph_objects as go

# Page configuration
st.set_page_config(page_title="🌈  Interactive  and  Colorful  Data  Dashboard",
layout="wide")
st.markdown("<h1   style='text-align:center;   color:#FF6347;'>🚀   Supercharged
Interactive Data Dashboard</h1>", unsafe_allow_html=True)

#     Sidebar:     Upload     and     Theme
st.sidebar.markdown("## 💼 Upload & Options")
uploaded_file = st.sidebar.file_uploader("Upload your CSV file", type="csv")

# Dark/Light Theme Toggle
theme = st.sidebar.radio("◐ Select Theme", ["Light", "Dark"])

# Track theme across reruns
if 'theme' not in st.session_state:
    st.session_state.theme = theme
else:
    st.session_state.theme = theme

# Load Data
if    uploaded_file:
    try:
        df        =        pd.read_csv(uploaded_file)
        st.success("✅ File uploaded successfully!")
        df.dropna(axis=1, how='all', inplace=True)  # Drop empty columns

        # Column classification
        numeric_cols             =             df.select_dtypes(include=['int64',
'float64']).columns.tolist()
        cat_cols               =              df.select_dtypes(include=['object',
'category']).columns.tolist()

        # Summary Metrics
        col1, col2, col3 = st.columns(3)
        col1.metric("📊 Numeric Columns", len(numeric_cols))
        col2.metric("🔠 Categorical Columns", len(cat_cols))
        col3.metric("🎁 Total Rows", df.shape[0])

        # Sidebar Filters
        st.sidebar.markdown("###   Filter Settings")
        filters_df = df.copy()
        if cat_cols:
            selected_category = st.sidebar.selectbox("Select Category to Filter",
cat_cols)
            selected_values = st.sidebar.multiselect("Filter Values",
df[selected_category].dropna().unique())
            if selected_values:
                filters_df =
filters_df[filters_df[selected_category].isin(selected_values)]

        for col in numeric_cols[:3]: # Limit filters to 3 for performance
            min_val, max_val = float(df[col].min()), float(df[col].max())
            sel_min, sel_max = st.sidebar.slider(f"Filter {col}", min_val,
max_val, (min_val, max_val))
            filters_df = filters_df[(filters_df[col] >= sel_min) &
(filters_df[col] <= sel_max)]

        # Reset filters button
        if st.sidebar.button("🔄 Reset Filters"):
            st.experimental_rerun()

        # Data View Options
        view_choice = st.radio("📊 Choose Data to Display", ["Filtered Data",
"Original Data"])
        display_df = filters_df if view_choice == "Filtered Data" else df

        st.markdown(f"### 📋 {view_choice} Preview")

st.dataframe(display_df.style.highlight_max(axis=0).highlight_min(axis=0),
use_container_width=True)

        # Summary Statistics
        if st.checkbox("📈 Show Summary Statistics"):
            st.write(display_df.describe(include='all'))

        # Sidebar: Chart Configuration
        st.sidebar.markdown("### 📊Chart Options")
        chart_type = st.sidebar.selectbox("Chart Type", ["Scatter", "Line",
"Histogram", "Box", "Heatmap", "Pie", "Bar", "Area", "Violin", "Treemap"])
        chart_title = st.sidebar.text_input("Chart Title", value="📊 Interactive
Data Chart")
        chart_size = st.sidebar.selectbox("Chart Size", ["Small", "Medium",
"Large"])
        chart_width = {"Small": 400, "Medium": 700, "Large": 1000}[chart_size]
        chart_height = {"Small": 300, "Medium": 500, "Large": 700}[chart_size]

        fig = None # Initialize chart

        # Chart Generation Logic
        if chart_type == "Scatter":
            x = st.sidebar.selectbox("X-axis", numeric_cols)
            y = st.sidebar.selectbox("Y-axis", numeric_cols)
            color = st.sidebar.selectbox("Color by", [None] + cat_cols)
            fig     =     px.scatter(filters_df,     x=x,     y=y,     color=color,
title=chart_title)

        elif chart_type == "Line":
            x = st.sidebar.selectbox("X-axis", numeric_cols + cat_cols)
            y = st.sidebar.selectbox("Y-axis", numeric_cols)
            color = st.sidebar.selectbox("Color by", [None] + cat_cols)
            fig = px.line(filters_df, x=x, y=y, color=color, title=chart_title)

        elif chart_type == "Histogram":
            col  =  st.sidebar.selectbox("Column",  numeric_cols)
            bins = st.sidebar.slider("Bins", 5, 100, 30)
            fig = px.histogram(filters_df, x=col, nbins=bins, title=chart_title)

        elif chart_type == "Box":
            y = st.sidebar.selectbox("Y-axis", numeric_cols)
            x = st.sidebar.selectbox("Group by", cat_cols)
            color = st.sidebar.selectbox("Color by", [None] + cat_cols)
            fig = px.box(filters_df, x=x, y=y, color=color, title=chart_title)

        elif  chart_type  ==  "Heatmap":
            if len(numeric_cols) >= 2:
                corr = filters_df[numeric_cols].corr()
                fig          =          px.imshow(corr,          text_auto=True,
color_continuous_scale="RdBu_r",     title=chart_title)
            else:
                st.warning("At least 2 numeric columns required for Heatmap.")

        elif chart_type == "Pie":
            labels   =   st.sidebar.selectbox("Labels",   cat_cols)
            values = st.sidebar.selectbox("Values", numeric_cols)
            fig     =     px.pie(filters_df,     names=labels,     values=values,
title=chart_title)

        elif chart_type == "Bar":
            x = st.sidebar.selectbox("X-axis", cat_cols)
            y = st.sidebar.selectbox("Y-axis", numeric_cols)
            color = st.sidebar.selectbox("Color by", [None] + cat_cols)
            fig = px.bar(filters_df, x=x, y=y, color=color, title=chart_title)

        elif chart_type == "Area":
            x = st.sidebar.selectbox("X-axis", numeric_cols + cat_cols)
            y = st.sidebar.selectbox("Y-axis", numeric_cols)
            color = st.sidebar.selectbox("Color by", [None] + cat_cols)
            fig = px.area(filters_df, x=x, y=y, color=color, title=chart_title)

        elif chart_type == "Violin":
            y = st.sidebar.selectbox("Y-axis", numeric_cols)
            x = st.sidebar.selectbox("Category", cat_cols)
            color = st.sidebar.selectbox("Color by", [None] + cat_cols)
            fig   =   px.violin(filters_df,   y=y,   x=x,   color=color,   box=True,
points="all", title=chart_title)

        elif chart_type == "Treemap":
            path_col  =  st.sidebar.multiselect("Hierarchy  (Top  to   Bottom)",
cat_cols)
            value_col = st.sidebar.selectbox("Size by", numeric_cols)
            if path_col:
                fig  =  px.treemap(filters_df,  path=path_col,  values=value_col,
title=chart_title)
            else:
                st.warning("Select at least one category for Treemap hierarchy.")

        #  Apply  Theme
        if fig:
            fig.update_layout(
                width=chart_width,
                height=chart_height,
                hovermode="closest",
                template="plotly_dark" if theme == "Dark" else "plotly_white",
                plot_bgcolor="rgba(0,0,0,0)"    if    theme    ==    "Dark"    else
"rgba(255,255,255,1)",
                paper_bgcolor="rgba(0,0,0,0)"    if    theme    ==    "Dark"    else
"rgba(255,255,255,1)"
            )
            st.plotly_chart(fig,  use_container_width=True)

        # Download Buttons
        st.download_button("⬇          Download          Filtered          Data",
filters_df.to_csv(index=False), file_name="filtered_data.csv", mime="text/csv")

    except Exception as e:
        st.error(f"❌ Error loading file: {e}")

else:
    st.info("📂 Please upload a CSV file to begin.
```

6.2 Implementation

PYTHON

Python serves as the core programming language for this data visualization dashboard. It provides powerful libraries such as **Pandas** for data manipulation, **Plotly** for rich interactive charts, and **NumPy** for numerical processing (if needed). The uploaded CSV data is read using Pandas, which allows for easy extraction of numeric and categorical columns, applying filters, and computing statistical summaries. Python's syntax and dynamic typing make it ideal for rapid development and integration of various components such as file handling, filtering logic, data aggregation, and exporting functionality.

Moreover, Python enables seamless customization of features like filter sliders, chart options, and user interactivity without the complexity of traditional full-stack web development.

SREAMLIT:

Streamlit acts as the front-end framework to build and display the dashboard directly in the browser, using only Python. It allows users to interact with the application through a clean and responsive UI, including components such as sidebar widgets, file uploaders, radio buttons, dropdowns, sliders, and charts. In this project, Streamlit handles the entire user interface: it enables users to upload CSV files, select chart types, apply filters, switch between light/dark themes, and visualize data dynamically. Streamlit also supports interactive Plotly charts, enabling zoom, hover, and click actions. Furthermore, Streamlit's `st.dataframe`, `st.plotly_chart`, and `st.download_button` functions are crucial in presenting filtered data and enabling data export—all without requiring any front-end code like HTML, CSS, or JavaScript.

7. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.1 TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
Systems/Procedures	: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

7.2 TESTING STRATEGIES

1. Unit Testing

Unit testing focuses on verifying the correctness of individual components and functions within the dashboard. Each function, such as reading CSV files, identifying numeric/categorical columns, filtering data, or generating a chart, is tested independently to ensure it produces the expected output. For example, a unit test might check whether a filtering function returns the correct subset of rows based on user inputs.

Tools: `unittest`, `pytest`
Example: Testing if the slider filter correctly restricts data to a selected range.

2. Integration Testing

Integration testing ensures that various components of the dashboard work together cohesively. This includes testing the workflow starting from file upload → data preprocessing → filter application → chart generation → data export. Integration tests catch bugs that arise when individual components interact, such as incorrect chart rendering due to bad data input or improper filter logic.

Tools: `pytest`, mock data, Streamlit component simulation
Example: Ensuring that applying a category filter updates both the chart and the preview table correctly.

3. Functional Testing

Functional testing verifies that the dashboard's features operate according to user requirements. This involves checking whether users can successfully upload files, apply filters, switch chart types, toggle between light and dark themes, and download the filtered dataset. It also includes testing the "Reset Filters" button and ensuring real-time updates are functional.

Example Test Cases:

- Uploading a valid CSV triggers a success message.
- Selecting a chart type dynamically renders the correct chart.
- Downloading the filtered data produces an accurate CSV file.

4. Performance Testing

Performance testing is critical to ensure the dashboard remains responsive, even when handling large datasets. This involves uploading CSV files with tens of thousands of rows and checking how long it takes to apply filters and render charts. It ensures the application is optimized and does not hang or crash during heavy data operations.

Tools: Python's `time` module, `cProfile`, Streamlit logs

Test Scenarios:

- Load test with 10k, 50k, 100k rows
- Time taken for filtering and rendering each chart type

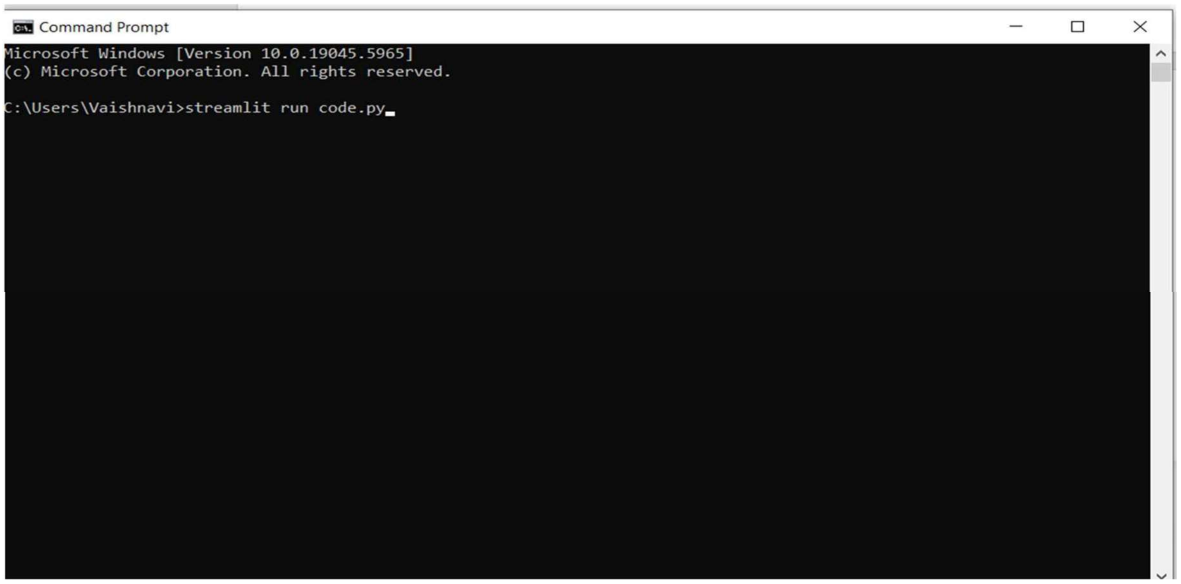
Sample Test Cases

Test Case ID	Test Scenario	Test Steps	Expected Result
TC01	Upload Valid CSV File	Upload a `.csv` file with valid structure and data	File is successfully read, and success message is shown
TC02	Upload Invalid File Format	Upload a `.txt` or `.xlsx` file	Error or no file accepted; no success message shown
TC03	Apply Category Filter	Select a categorical column and choose specific values	Data table and chart update to reflect selected values
TC04	Apply Numeric Range Filter	Use slider to set a numeric range	Data and visualizations are filtered based on the selected range
TC05	Change Chart Type	Switch between chart types (Bar, Pie, Line, etc.)	Corresponding chart is rendered correctly with selected parameters
TC06	Switch Theme (Light/Dark)	Select either Light or Dark mode from sidebar	Theme changes for entire dashboard including charts
TC07	Input Custom Chart Title	Enter a custom title in the chart title textbox	Title appears on the chart
TC08	Download Filtered Data	Click on "Download Filtered Data" after applying filters	CSV file is downloaded with only the filtered rows
TC09	Reset Filters	Click on "Reset Filters" button in sidebar	All filters are cleared, and dashboard reloads with full dataset
TC10	Render Heatmap with Correlation Matrix	Select "Heatmap" chart type with numeric data present	A heatmap is displayed showing correlation values

Table 7.1

8. OUTPUTSCREEN

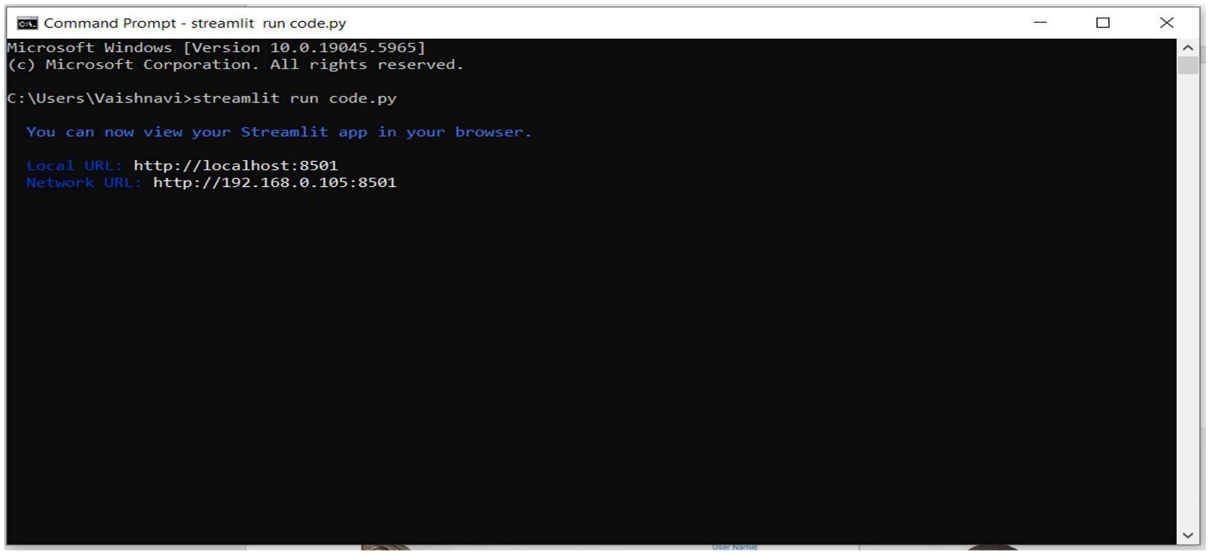
Task 1: Open cmd prompt to execute code



```
Command Prompt
Microsoft Windows [Version 10.0.19045.5965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Vaishnavi>streamlit run code.py
```

Figure 8.1: Command prompt Instructions



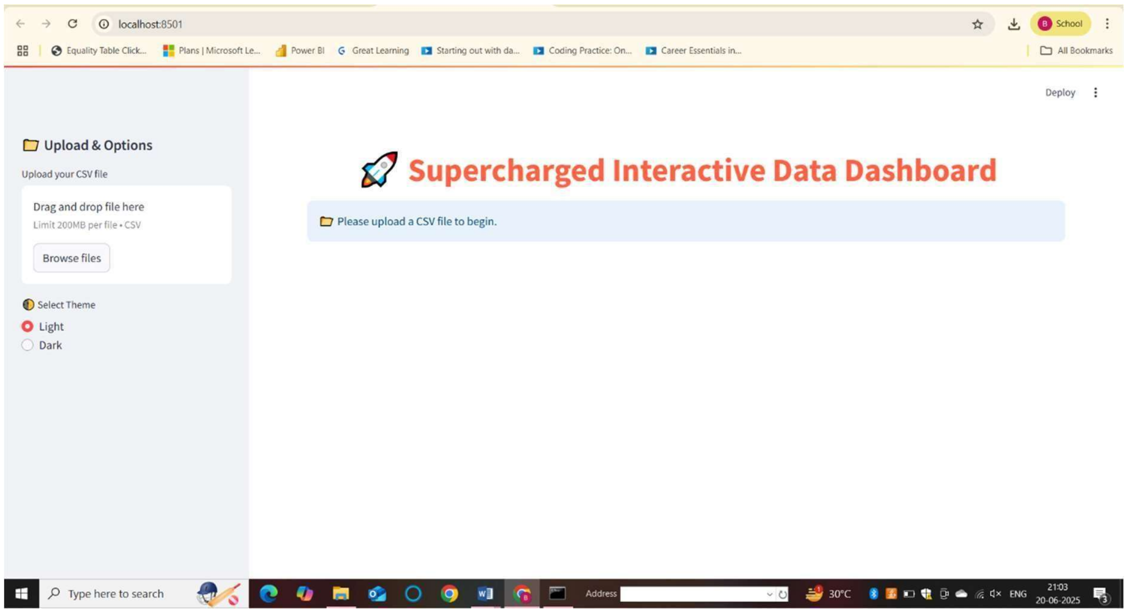
```
Command Prompt - streamlit  run code.py
Microsoft Windows [Version 10.0.19045.5965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Vaishnavi>streamlit run code.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.0.105:8501
```

Figure 8.2: Implementation



Upload & Options

Upload your CSV file

Drag and drop file here

Limit 200MB per file • CSV

Browse files

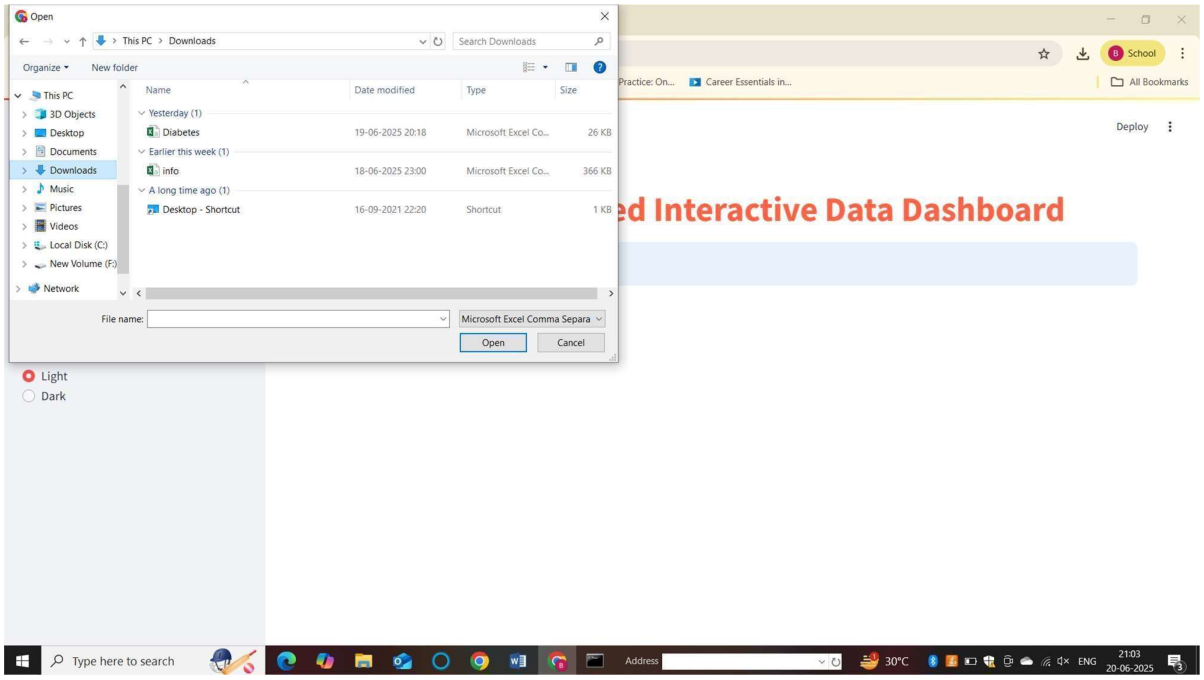
Select Theme

[✓ Light] Dark

🚀 Supercharged Interactive Data Dashboard

Please upload a CSV file to begin.

Figure 8.3 :User Interface



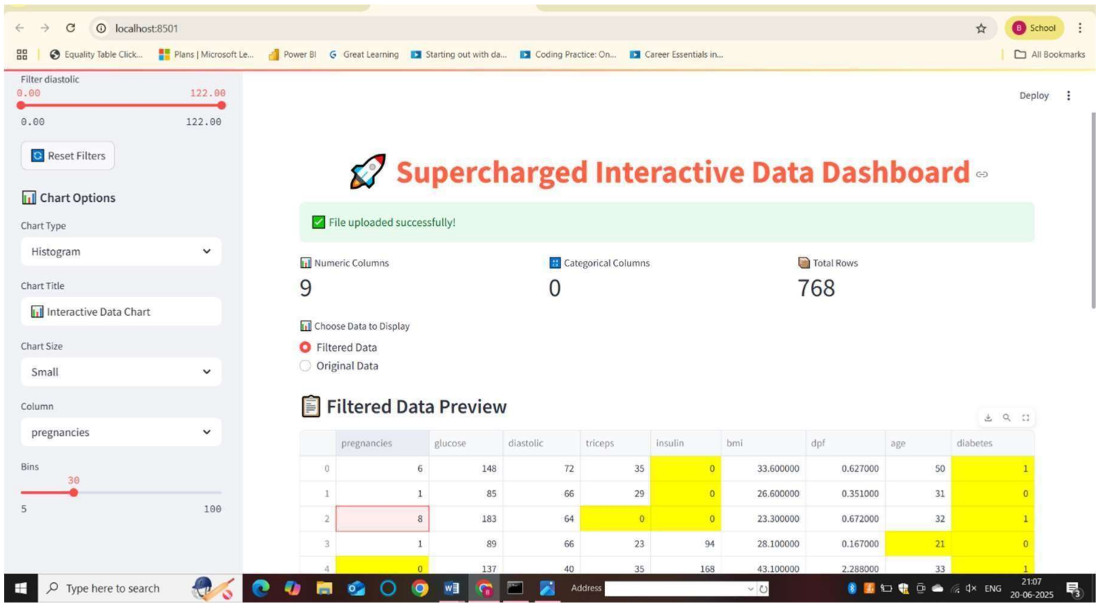
Name	Date modified	Type	Size
Yesterday (1)			
Diabetes	19-06-2025 20:18	Microsoft Excel Co...	26 KB
Earlier this week (1)			
info	18-06-2025 23:00	Microsoft Excel Co...	366 KB
A long time ago (1)			
Desktop - Shortcut	16-09-2021 22:20	Shortcut	1 KB

File name: Microsoft Excel Comma Separa

...ed Interactive Data Dashboard

[✓ Light] Dark

Figure 8.4 :Uploading Data



Filter diastolic

0.00 – 122.00

Reset Filters

Chart Options

Chart Type: Histogram

Chart Title: Interactive Data Chart

Chart Size: Small

Column: pregnancies

Bins: 30 (5 – 100)

Supercharged Interactive Data Dashboard

File uploaded successfully!

Numeric Columns	Categorical Columns	Total Rows
9	0	768

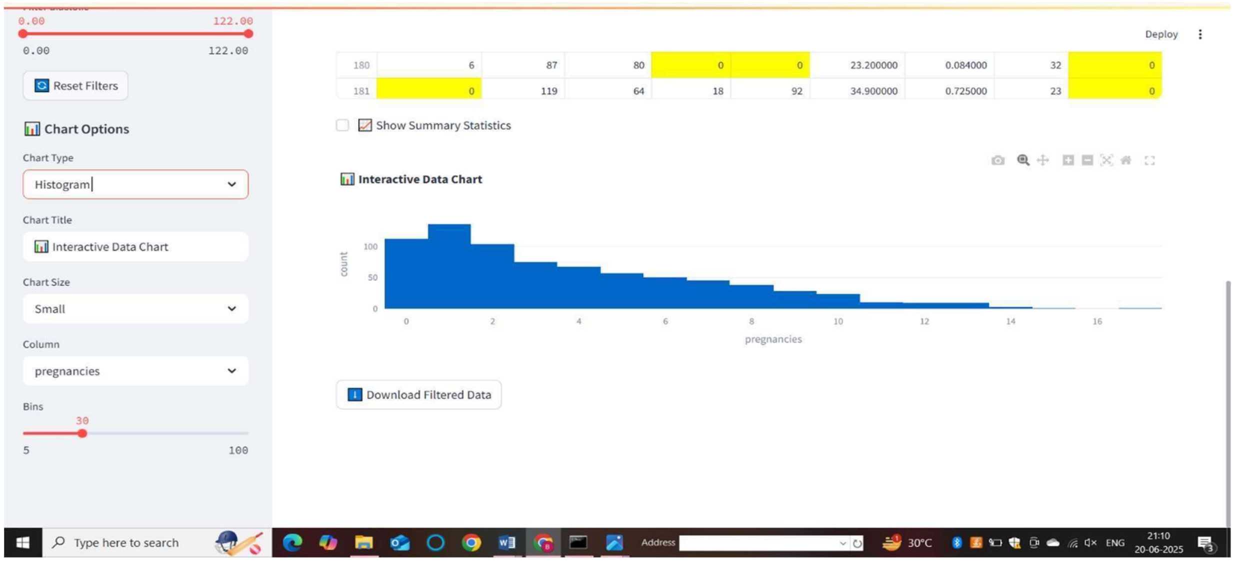
Choose Data to Display

[✓ Filtered Data]
Original Data

Filtered Data Preview

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.600000	0.627000	50	1
1	1	85	66	29	0	26.600000	0.351000	31	0
2	8	183	64	0	0	23.300000	0.672000	32	1
3	1	89	66	23	94	28.100000	0.167000	21	0
4	0	137	40	35	168	43.100000	2.288000	33	1

Figure 8.5 :Uploaded data



0.00 – 122.00

Reset Filters

Chart Options

Chart Type: Histogram

Chart Title: Interactive Data Chart

Chart Size: Small

Column: pregnancies

Bins: 30 (5 – 100)

180	6	87	80	0	0	23.200000	0.084000	32	0
181	0	119	64	18	92	34.900000	0.725000	23	0

Show Summary Statistics

Interactive Data Chart

Figure 8.6:Output

9. CONCLUSION

The **Dynamic Data Visualization Dashboard** project successfully demonstrates the integration of modern data visualization techniques using Python-based tools such as **Streamlit**, **Pandas**, and **Plotly**. It allows users to effortlessly upload datasets, apply dynamic filters, select visualization types, and explore data through an intuitive and interactive web interface. With features like real-time chart rendering, theme customization (Light/Dark mode), downloadable filtered data, and support for multiple chart types, the dashboard enhances both the analytical experience and usability for a wide range of users—from analysts to non-technical stakeholders.

The system addresses key challenges in traditional static reporting by offering **live filtering, responsive visuals, and easy deployment** without the need for backend frameworks or complex frontends. Furthermore, its modular design makes it extensible for future improvements like AI-driven insights or database connectivity. Overall, the project not only meets its objectives but also showcases a scalable and user-friendly approach to **modern data storytelling and exploration**.

10. FUTURE ENHANCEMENTS

The current version of the Interactive Data Visualization Dashboard provides a solid foundation for exploring and analyzing data interactively. However, there are several opportunities to extend its capabilities in future iterations to enhance user experience, intelligence, and scalability.

One major enhancement is the integration of **AI-powered chart recommendations**. By leveraging machine learning, the system could automatically suggest the most suitable visualization type based on the uploaded dataset's structure and the user's past interactions. This would simplify the process for users unfamiliar with data visualization best practices. Additionally, introducing **Natural Language Processing (NLP)** could allow users to input plain English queries like "show average sales by product category," enabling more intuitive interaction with the dashboard.

Another significant improvement would be the addition of **real-time data integration**. Connecting the dashboard to live APIs or databases would enable it to visualize dynamic, continuously updating data—ideal for applications such as stock monitoring or IoT analytics. To complement this, adding **user authentication and profile management** would allow personalization features such as saving filter settings, theme preferences, and recently accessed files.

Further, adding **collaboration features** such as shareable views, comment threads, or embedding options would promote teamwork and data sharing. Improving **mobile responsiveness** will ensure a seamless experience across all devices, while a **custom theme editor** could provide options for personalized aesthetics and branding.

Incorporating these enhancements would significantly broaden the dashboard's usability and functionality, making it more intelligent, accessible, and enterprise-ready.

10. REFERENCES

1. W. Wang, L. Chen, J. Wang, and H. Qu, "A Survey on ML4VIS: Applying Machine Learning Advances to Data Visualization," *arXiv preprint arXiv:2007.13493*, 2020, revised 2021.
2. Y. Wu, H. Wang, M. Shu, D. Moritz, W. Cui, K. Zhang, H. Zhang, and H. Qu, "AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization," *arXiv preprint arXiv:2102.01350*, 2021.
3. B. Bach, N. H. Riche, and T. Dwyer, "Design Patterns for Data-Driven Storytelling," in *Proc. IEEE VIS*, vol. 29, no. 2, pp. 70–80, 2022.
4. M. Khorasani, "Creating Dynamic Dashboards with Streamlit," *Towards Data Science*, 2020.
5. P. Nagarajan, "Building Interactive Dashboards with Dynamic Data Using Streamlit," *Medium*, Oct. 2024.
6. J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
7. P. J. van der Walt et al., "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.
8. W. McKinney, "Data Structures for Statistical Computing in Python," in *Proc. of the 9th Python in Science Conference (SciPy)*, 2010.
9. Plotly Technologies Inc., "Plotly Express: Documentation and Examples," [Online]. Available: https://plotly.com/python/plotly-express
10. Streamlit Inc., "Streamlit Documentation," [Online]. Available: https://docs.streamlit.io/
11. Jupyter Team, "Project Jupyter," [Online]. Available: https://jupyter.org/
12. T. Chen, C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. ACM SIGKDD*, pp. 785–794, 2016 (for future ML enhancement reference).