# Project Report

## Title: Data Visualization Dashboard

By:

Shreyas Kumar

# Abstract

The Data Visualization Dashboard is a comprehensive tool built using React, Node.js, and Electron, designed to facilitate the comparison and analysis of multiple datasets through interactive line charts. This project provides users with an intuitive interface for uploading datasets, selecting attributes for comparison, calculating attribute statistics, and visualizing data with advanced features such as zooming, panning, and custom axis scales. The application aims to enhance data analysis efficiency and accuracy by providing a robust platform for interactive data visualization.

# Table of content

# 1. Introduction

Data visualization is a critical aspect of data analysis, enabling users to interpret and understand complex datasets effectively. The Data Visualization Dashboard project aims to provide a user-friendly platform for visualizing and comparing multiple datasets through line charts. This project bridges the gap between raw data and actionable insights, facilitating better decision-making and analysis.

# 2. Objectives

The primary objectives of this project are:

- To create a user-friendly interface for uploading and managing datasets.
- To allow users to select specific attributes for visualization.
- To provide interactive charts with zooming and panning capabilities.
- To enable custom axis scaling for better data representation.
- To calculate and display statistics for selected attributes.
- To package the application as a desktop app using Electron.

# 3. Technology Stack

## Frontend

- **React**: For building the user interface, leveraging components and state management.
- **Chart.js**: For rendering interactive charts, providing a flexible and powerful charting library.
- **chartjs-plugin-zoom**: For adding zoom and pan functionalities to the charts.
- **Axios**: For making HTTP requests to the backend, handling API interactions.

### Backend

- **Node.js**: For server-side logic, providing a scalable and efficient runtime environment.
- **Express**: For creating API endpoints, enabling easy setup of server routes and middleware.
- **CSV-Parser**: For parsing CSV files, extracting and processing data for visualization.

### Desktop Application

- **Electron**: For packaging the web application as a desktop app, providing a cross-platform desktop application experience.

## 4. System Architecture

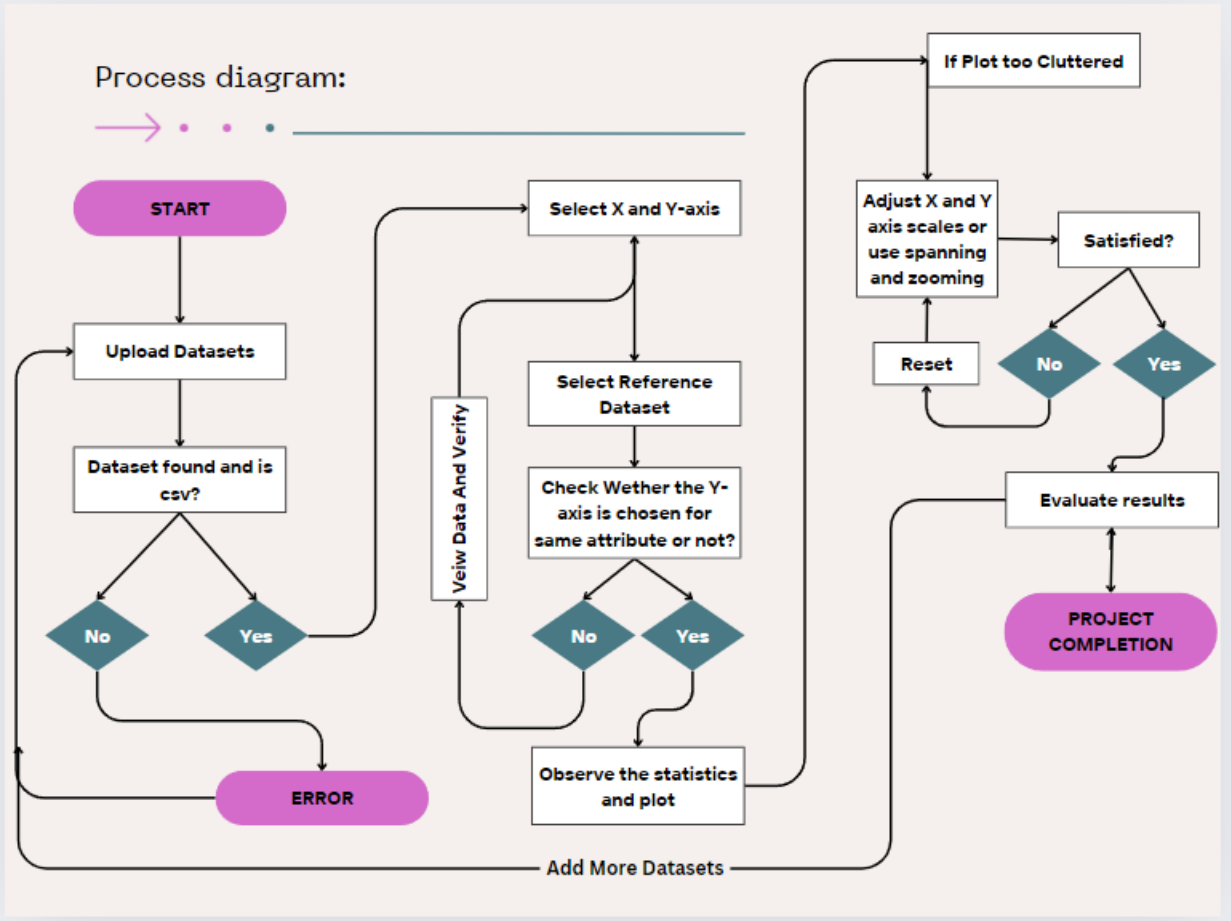
The system architecture consists of three main components:

1. **Frontend**: The React application that handles the user interface and interacts with the backend.
2. **Backend**: The Node.js server that processes API requests, handles file uploads, and manages dataset processing.
3. **Electron**: The framework that packages the frontend and backend into a cross-platform desktop application.

### Data Flow

1. **User Interaction**: Users interact with the frontend to upload datasets and select visualization options.
2. **API Requests**: The frontend sends API requests to the backend for uploading datasets and fetching processed data.
3. **Data Processing**: The backend processes the uploaded datasets and returns the necessary data for visualization.

4. **Chart Rendering**: The frontend receives the processed data and renders interactive charts using Chart.js.




# 5. Key Features

## Dataset Upload and Management

- **Upload Datasets**: Users can upload multiple datasets in CSV format.
- **Dataset List**: Uploaded datasets are displayed in a list with options to remove or customize them.
- **Custom Colors**: Users can assign custom colors to each dataset for better differentiation.

## Axis Selection

- **Select Axes**: Users can select attributes for the x and y axes from the uploaded datasets.
- **Dynamic Update**: The charts update dynamically based on the selected attributes.

## Custom Axis Scales

- **Custom Scales**: Users can set custom minimum and maximum values for the x and y axes.
- **Reset Scales**: A reset button allows users to revert to default axis scales if they input values outside dataset limits.

## Interactive Charts

- **Zooming**: Users can zoom in and out using mouse wheel or pinch gestures.
- **Panning**: Users can pan across the chart to explore different data sections.
- **Tooltip and Hover**: Interactive tooltips and hover effects provide additional data insights.

## Attribute Statistics Calculation

- **Statistics Calculation**: Users can calculate and view statistics (min, max, mean, variance, standard deviation) for selected attributes.
- **Statistics Display**: Statistics are displayed in scrollable cards for easy viewing.

# 6. Implementation Details

## Frontend

The frontend is built with React and consists of several key components:

- **Header and Footer**: Provide a consistent layout for the application.
- **DatasetUpload**: Component for uploading datasets. It uses an HTML file input to select files and sends them to the backend for processing.
- **DatasetList**: Displays a list of uploaded datasets with options to remove them or customize their colors.
- **AxisSelection**: Allows users to select attributes for the x and y axes from the datasets.
- **ChartComponent**: Renders the interactive charts using Chart.js and integrates zoom and pan functionalities.
- **StatisticsDisplay**: Displays calculated statistics for the selected attributes in scrollable cards.

## Backend

The backend is built with Node.js and Express. It handles file uploads, processes datasets, and serves API endpoints for data visualization.

- **API Endpoints**:
    - **/datasets/upload**: Endpoint for uploading datasets.
    - **/datasets/chart-data**: Endpoint for fetching processed chart data based on selected attributes.
    - **/datasets/statistics**: Endpoint for calculating and returning statistics for selected attributes.

## Electron

Electron is used to package the application as a cross-platform desktop app, providing a native application experience. The Electron setup includes the main process file and a preload script for security.

# 7. Testing and Validation

The application was tested extensively to ensure:

- Correct data upload and parsing.
- Accurate data visualization based on selected attributes.
- Proper functioning of zooming, panning, and custom axis scaling.
- Correct calculation and display of attribute statistics.
- Smooth packaging and execution of the Electron app on different platforms.

## Testing Scenarios

1. **Uploading Datasets**: Verified that multiple datasets can be uploaded and displayed correctly.
2. **Selecting Axes**: Ensured that selecting different attributes for x and y axes updates the chart correctly.
3. **Custom Axis Scales**: Tested custom scaling functionality to ensure the chart adjusts accordingly.
4. **Zoom and Pan**: Verified zooming and panning functionality across various datasets.
5. **Statistics Calculation**: Ensured accurate calculation and display of statistics for selected attributes.
6. **Packaging**: Ensured the Electron app packages correctly and runs on Windows, macOS, and Linux.

# 8. Challenges and Solutions

## Challenge 1: Handling Large Datasets

- **Solution**: Implemented data aggregation and decimation techniques to optimize chart rendering.

## Challenge 2: Maintaining UI Responsiveness

- **Solution**: Used React's state management efficiently and optimized component re-rendering.

## Challenge 3: Ensuring Cross-Platform Compatibility

- **Solution**: Thoroughly tested the Electron application on multiple platforms to ensure consistent behavior.

# 9. Future Enhancements

## Additional Chart Types

- Integrate more chart types (e.g., bar charts, scatter plots) for varied data visualization needs.

## Enhanced Data Processing

- Implement more advanced data processing techniques (e.g., data filtering, statistical analysis).

### Real-Time Data Updates

- Enable real-time data updates and live chart rendering for dynamic datasets.

### User Accounts and Data Saving

- Implement user authentication and data saving features to allow users to save their datasets and visualization settings.

## 10. Conclusion

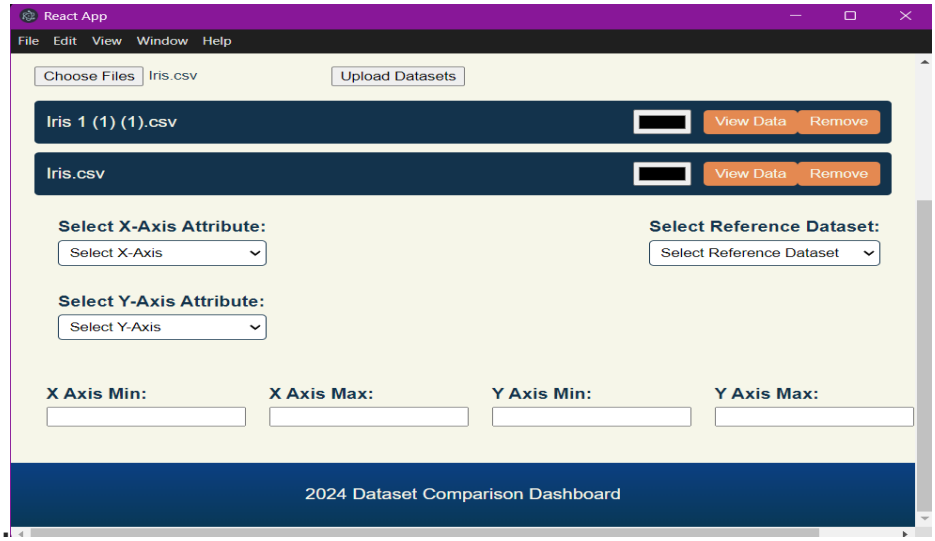
The Data Visualization Dashboard successfully achieves its goal of providing a robust and interactive platform for visualizing and comparing multiple datasets. By leveraging modern web technologies and Electron, the project offers a seamless user experience both as a web application and a desktop application. Future enhancements will focus on expanding the functionality and improving the user experience further.

This project demonstrates the power of combining React, Node.js, and Electron to create a versatile and user-friendly data visualization tool. The interactive features and customizability of the charts make it a valuable tool for data analysts and researchers.

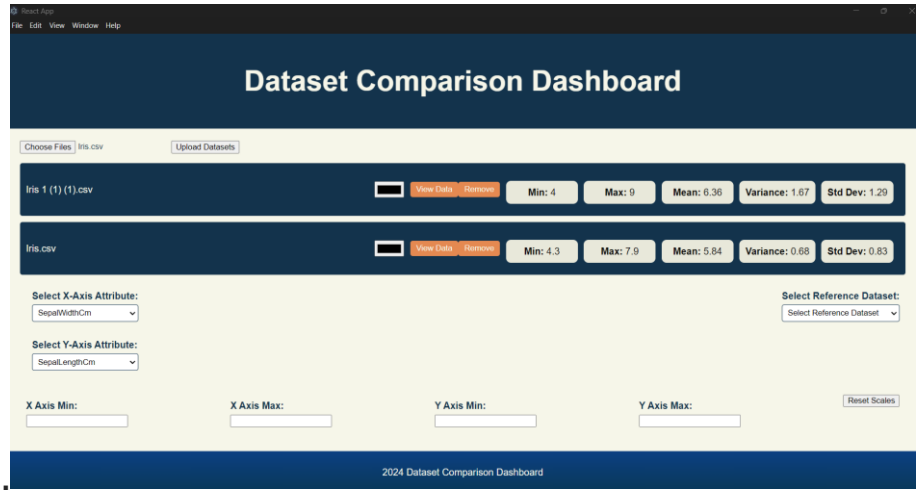
## 11.Appendix:

UI

1. 
React App
File Edit View Window Help
Dataset Comparison Dashboard
Choose Files No file chosen
Upload Datasets
2024 Dataset Comparison Dashboard

2. 
React App
File Edit View Window Help
Choose Files Iris.csv
Upload Datasets
Iris 1 (1) (1).csv
View Data
Remove
Iris.csv
View Data
Remove
Select X-Axis Attribute:
Select X-Axis
Select Reference Dataset:
Select Reference Dataset
Select Y-Axis Attribute:
Select Y-Axis
X Axis Min:
X Axis Max:
Y Axis Min:
Y Axis Max:
2024 Dataset Comparison Dashboard

3.


React App
File Edit View Window Help
Dataset Comparison Dashboard
Choose Files Iris.csv
Upload Datasets
Iris 1 (1) (1).csv
View Data
Remove
Min: 4
Max: 9
Mean: 6.36
Variance: 1.67
Std Dev: 1.29
Iris.csv
View Data
Remove
Min: 4.3
Max: 7.9
Mean: 5.84
Variance: 0.68
Std Dev: 0.83
Select X-Axis Attribute:
SepalWidthCm
Select Reference Dataset:
Select Reference Dataset
Select Y-Axis Attribute:
SepalLengthCm
X Axis Min:
X Axis Max:
Y Axis Min:
Y Axis Max:
Reset Scales
2024 Dataset Comparison Dashboard

4.


React App
File Edit View Window Help
X Axis Min:
1.7
X Axis Max:
4.5
Y Axis Min:
4
Y Axis Max:
9
Iris 1 (1) (1).csv
Iris.csv
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
1.7
2.0
2.5
3.0
3.5
4.0
4.5