DATA ANALYZER & VISUALIZER

By:
Rajkumar Gaurangbhai Shahu

(M.Sc. Data Science SEMESTER – III)

EXAM NO: - 30004

SUBMITTED AS

PARTIAL FULFILLMENT OF

M.Sc. DATA SCIENCE DEGREE

2025-26

C P PATEL AND F H SHAH COMMERCE (AUTONOMOUS) COLLEGE

SARDAR PATEL UNIVERSITY

ACKNOLEDGMENT

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose cease less cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

I am grateful to our project guide “Prof. Hirav Joshi” for the guidance, inspiration and constructive suggestion that helpful us in the preparation of this project.

I am also thankful to our Head of The Department “Dr. Nayan S Patel” for his continuous guidance.

I am also obliged to our Principal “Dr. Tejal Tanna” for creative support.

I also thank my colleagues who have helped in successful completion of the project.

Index

Project Introduction

The **Data Analysis & Visualization Dashboard** is a comprehensive web-based application built with Streamlit that enables users to upload, analyze, and visualize data without writing any code. The dashboard provides an intuitive interface for exploratory data analysis (EDA), interactive visualizations, and data export capabilities.

Key Objectives

- **Accessibility**: Enable non-technical users to perform data analysis
- **Interactivity**: Provide real-time, interactive visualizations
- **Flexibility**: Support multiple file formats and visualization types
- **Export**: Allow users to download processed data and insights

Target Users

- Data analysts and scientists
- Business analysts
- Students and researchers
- Non-technical stakeholders who need data insights

Features

□ Core Functionality

- **Multi-format file upload** (CSV, Excel, JSON, Parquet)
- **Interactive data exploration** with filtering and sorting
- **Comprehensive statistical analysis** and summaries
- **12+ visualization types** with customization options
- **Real-time data filtering** and transformation
- **Export capabilities** (CSV, Excel formats)
- **Responsive design** for different screen sizes

□ Analysis Capabilities

- Descriptive statistics
- Data type detection and conversion
- Missing value analysis
- Correlation analysis
- Distribution analysis
- Categorical data analysis

□ Visualization Features

- Interactive charts with Plotly
- Customizable color schemes
- Dynamic filtering and grouping
- Responsive chart sizing
- Professional styling

Installation & Setup

Prerequisites :

- Python 3.7 or higher
- pip package manager
- 2GB RAM minimum (8GB recommended)
- Modern web browser

Step 1: Environment Setup

```
# Create virtual environment (recommended)
python -m venv streamlit_dashboard
source streamlit_dashboard/bin/activate  # On Windows:
streamlit_dashboard\Scripts\activate

# Or using conda
conda create -n streamlit_dashboard python=3.9
conda activate streamlit_dashboard
```

Step 2: Install Dependencies

```
# Install required packages
pip install streamlit
pip install pandas
pip install plotly
pip install seaborn
pip install matplotlib
pip install scipy
pip install openpyxl
pip install pyarrow
```

Step 3: Download and Run

```
# Save the dashboard code as 'r1.py'

streamlit run r1.py
```

Quick Start Guide

- **. Launch the Application**
- **streamlit run r1.py**
- **The dashboard will open in your default browser at http://localhost:8501**
- **2. Upload Your Data**
- **Click "Choose a file" in the sidebar**
- **Select your data file (CSV, Excel, JSON, or Parquet)**
- **Wait for the upload confirmation**
- **3. Explore Your Data**
- **Review the Data Overview section for basic statistics**
- **Use the Data Preview to examine your dataset**
- **Check Data Types and Missing Values**
- **4. Create Visualizations**

4. Create Visualizations

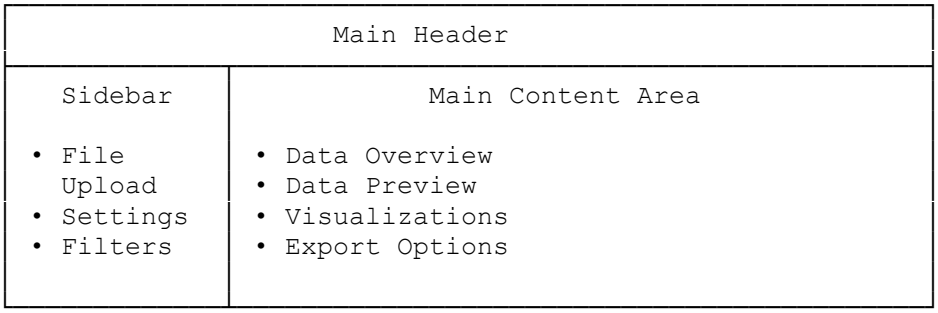
- Select a **Visualization Type** from the dropdown
- Choose appropriate **columns** for X and Y axes
- Customize colors, groupings, and aggregations
- View your interactive chart

5. Filter and Export

- Use the **Data Filtering** section to subset your data
- Export filtered results as **CSV** or **Excel**

User Interface Guide

Layout Structure



Sidebar Components

- □ **Data Upload**: File uploader with format validation
- □ **Sample Data**: Load demonstration dataset
- □ **File Info**: Display file status after upload

Main Content Sections

1. □ **Data Overview**: Dataset metrics and statistics
2. □ **Data Filtering**: Interactive filtering controls
3. □ **Data Visualizations**: Chart creation interface
4. □ **Export Data**: Download processed data

Feature Documentation

Data Upload System

Supported Operations

```
# File type detection and loading
file_types = {
    '.csv': pd.read_csv,
    '.xlsx': pd.read_excel,
    '.xls': pd.read_excel,
    '.json': pd.read_json,
    '.parquet': pd.read_parquet
}
```

Error Handling

- **File size limits**: 200MB maximum
- **Format validation**: Automatic file type detection
- **Encoding issues**: UTF-8 fallback handling
- **Memory management**: Chunked loading for large files

Data Overview Module

Metrics Displayed

- **Dataset Dimensions**: Rows × Columns
- **Column Types**: Numeric, Text, DateTime counts
- **Data Quality**: Missing values, duplicates
- **Memory Usage**: Dataset size in memory

Statistical Summary

```
# Numeric columns analysis
numeric_stats = df.describe()
# - Count, Mean, Std, Min, 25%, 50%, 75%, Max

# Categorical columns analysis
categorical_info = df.select_dtypes(include=['object']).describe()
# - Count, Unique, Top value, Frequency
```

Supported File Formats

CSV Files (.csv)

```
# Reading parameters
df = pd.read_csv(file,
    encoding='utf-8',
    parse_dates=True,
    infer_datetime_format=True
)
```

Best Practices:

- Use UTF-8 encoding
- Include headers in first row
- Consistent date formats (ISO 8601 recommended)

Excel Files (.xlsx, .xls)

```
# Multi-sheet support
df = pd.read_excel(file, sheet_name=0)  # First sheet
```

Limitations:

- Maximum 1,048,576 rows
- First sheet only (current version)
- No password-protected files

JSON Files (.json)

```
# Nested JSON flattening
df = pd.json_normalize(json_data)
```

Supported Structures:

- Array of objects: `[{key: value}, ...]`
- Nested objects: Automatically flattened
- Mixed data types: Automatic type inference

Parquet Files (.parquet)

```
# High-performance columnar format
df = pd.read_parquet(file)
```

Advantages:

- Faster loading for large datasets
- Built-in compression
- Preserves data types

Visualization Types

1. Scatter Plot

Purpose: Explore relationships between two numeric variables

Parameters:

- X-axis: Numeric column
- Y-axis: Numeric column
- Color: Categorical column (optional)
- Size: Numeric column (optional)

Use Cases:

- Correlation analysis
- Outlier detection
- Pattern identification

2. Line Chart

Purpose: Show trends over time or ordered categories

Parameters:

- X-axis: DateTime or numeric column
- Y-axis: Numeric column
- Group by: Categorical column (optional)

Use Cases:

- Time series analysis
- Trend visualization
- Performance tracking

3. Bar Chart

Purpose: Compare categories or show distributions

Parameters:

- Category: Categorical column
- Value: Numeric column
- Aggregation: sum, mean, count, median

Use Cases:

- Category comparison
- Distribution analysis
- Ranking visualization

4. Histogram

Purpose: Show distribution of a single numeric variable

Parameters:

- Column: Numeric column
- Bins: Number of bins (10-50)
- Overlay: Normal distribution curve

Use Cases:

- Distribution analysis
- Data quality assessment
- Statistical analysis

5. Box Plot

Purpose: Show distribution quartiles and outliers

Parameters:

- Y-axis: Numeric column
- Group by: Categorical column (optional)

Use Cases:

- Outlier detection
- Distribution comparison
- Statistical summary

6. Heatmap

Purpose: Show correlation between numeric variables

Parameters:

- Correlation method: Pearson, Spearman
- Color scale: Various options
- Annotations: Show correlation values

Use Cases:

- Feature selection
- Multicollinearity detection
- Pattern recognition

7. Pie Chart

Purpose: Show proportions of categorical data

Parameters:

- Category: Categorical column
- Limit: Top N categories

Use Cases:

- Market share analysis
- Category distribution
- Budget allocation

8. Distribution Plot

Purpose: Combine histogram with density curve

Parameters:

- Column: Numeric column
- KDE: Kernel density estimation overlay

Use Cases:

- Distribution shape analysis
- Statistical modeling
- Data exploration

9. Time Series

Purpose: Specialized line chart for temporal data

Parameters:

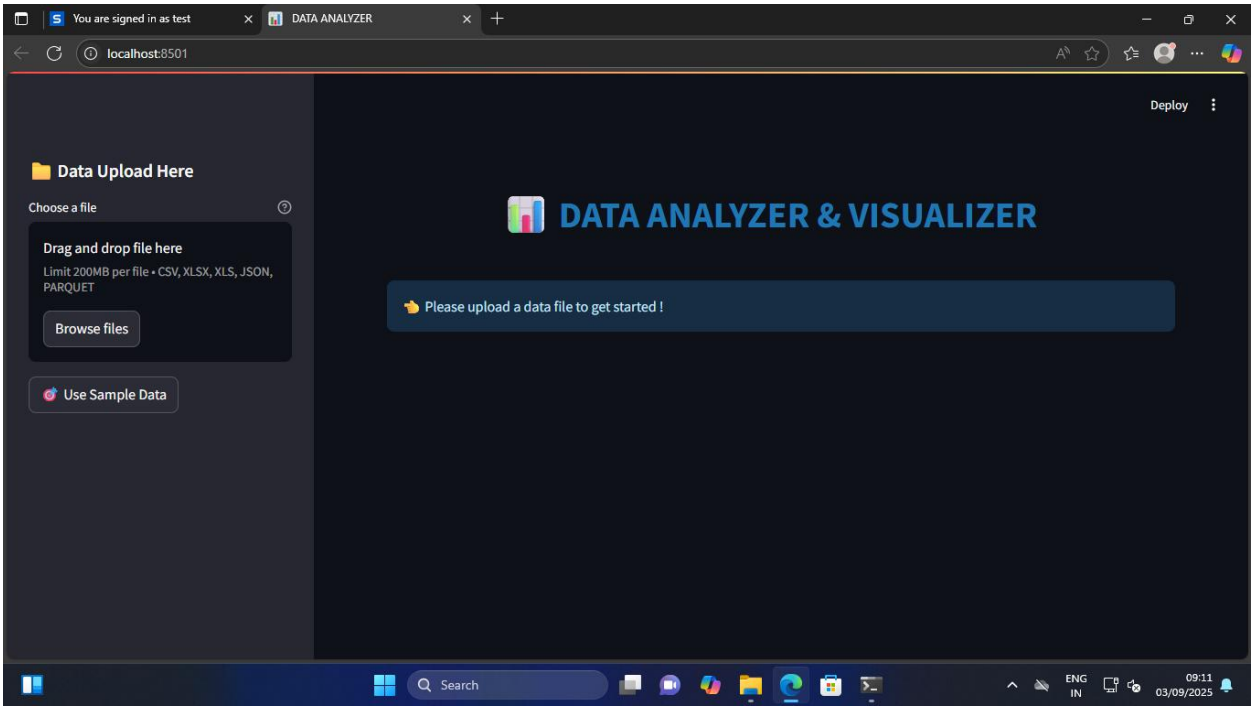
- Date column: DateTime column
- Value column: Numeric column
- Resampling: Daily, weekly, monthly

Use Cases:

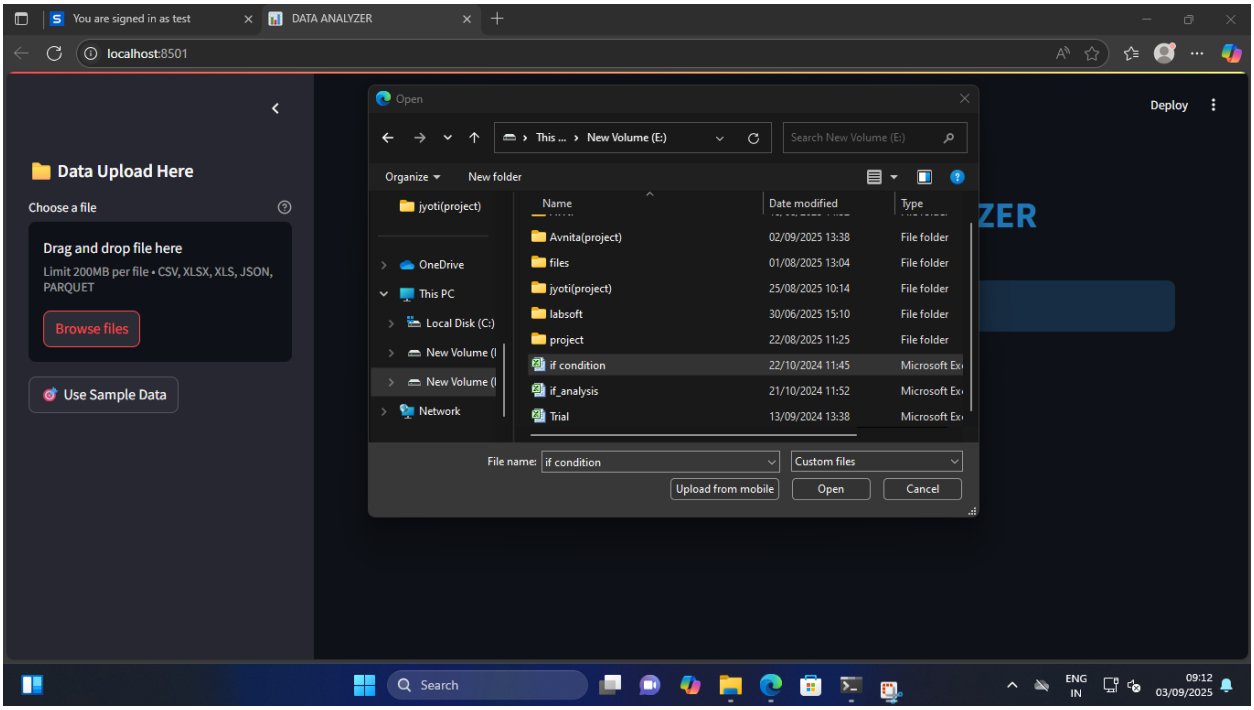
- Temporal trend analysis
- Seasonal pattern detection
- Forecasting preparation

User Interface

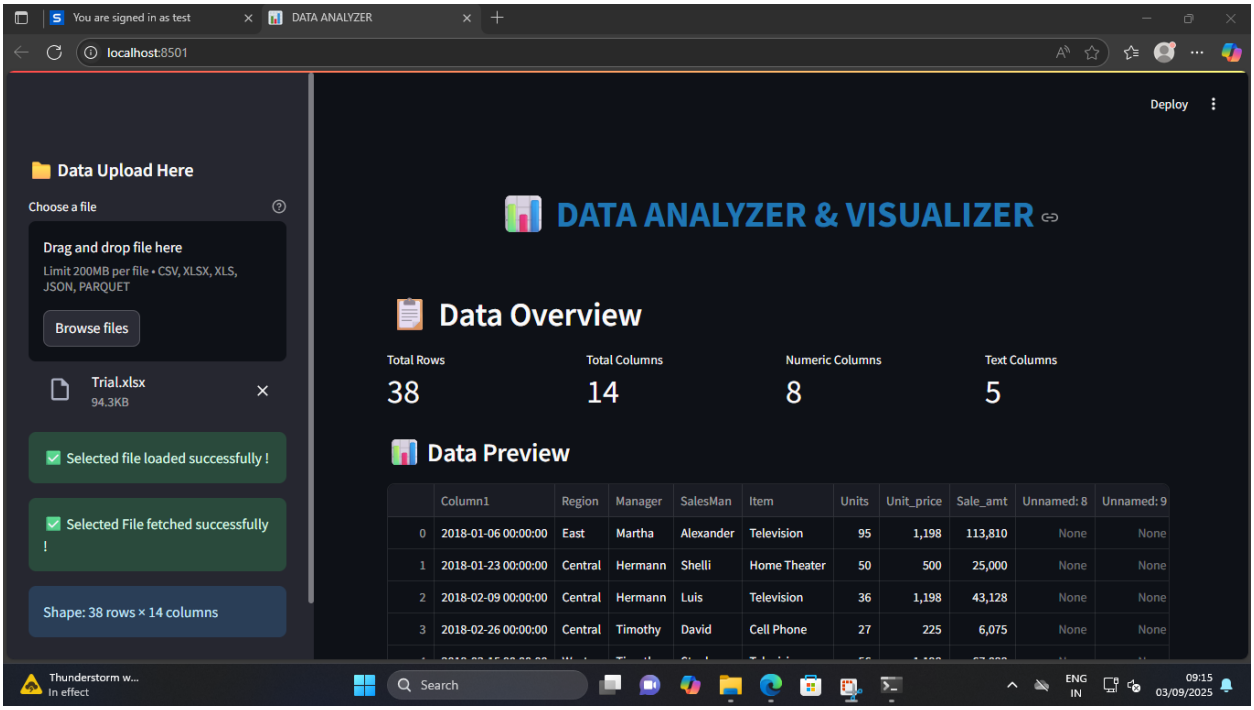
Home Page View :

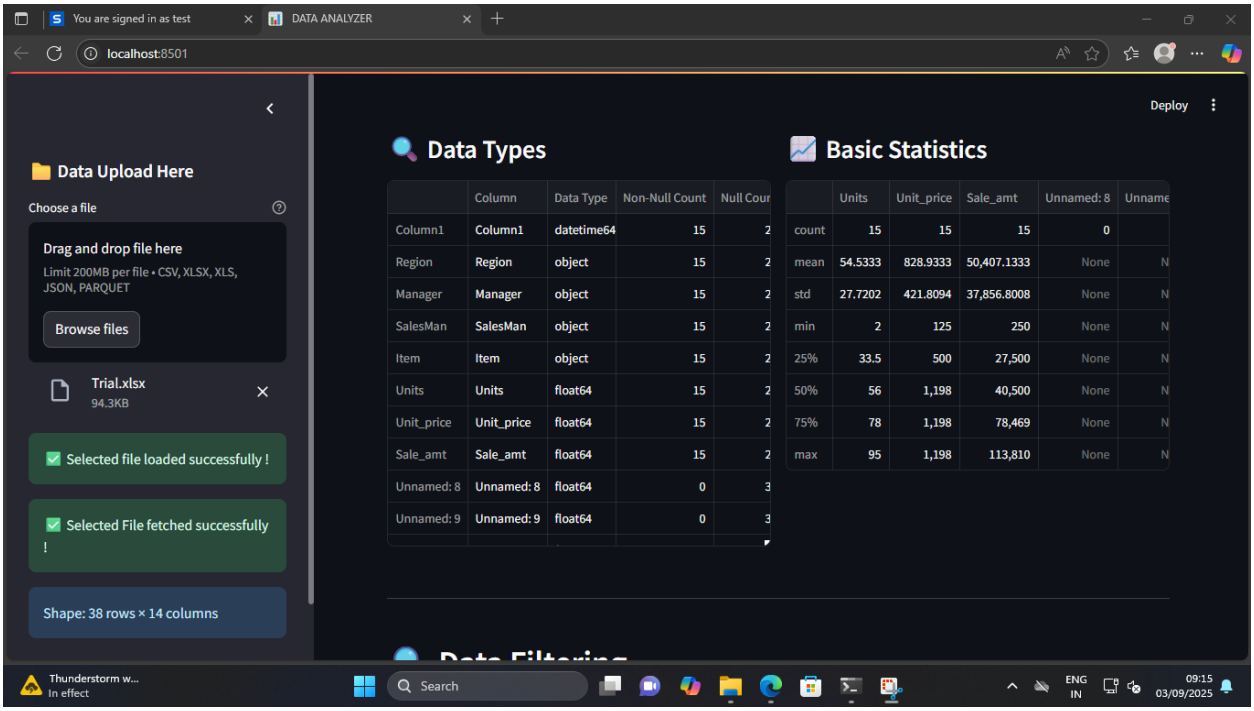


In this page user select a file for analyze and visualize.

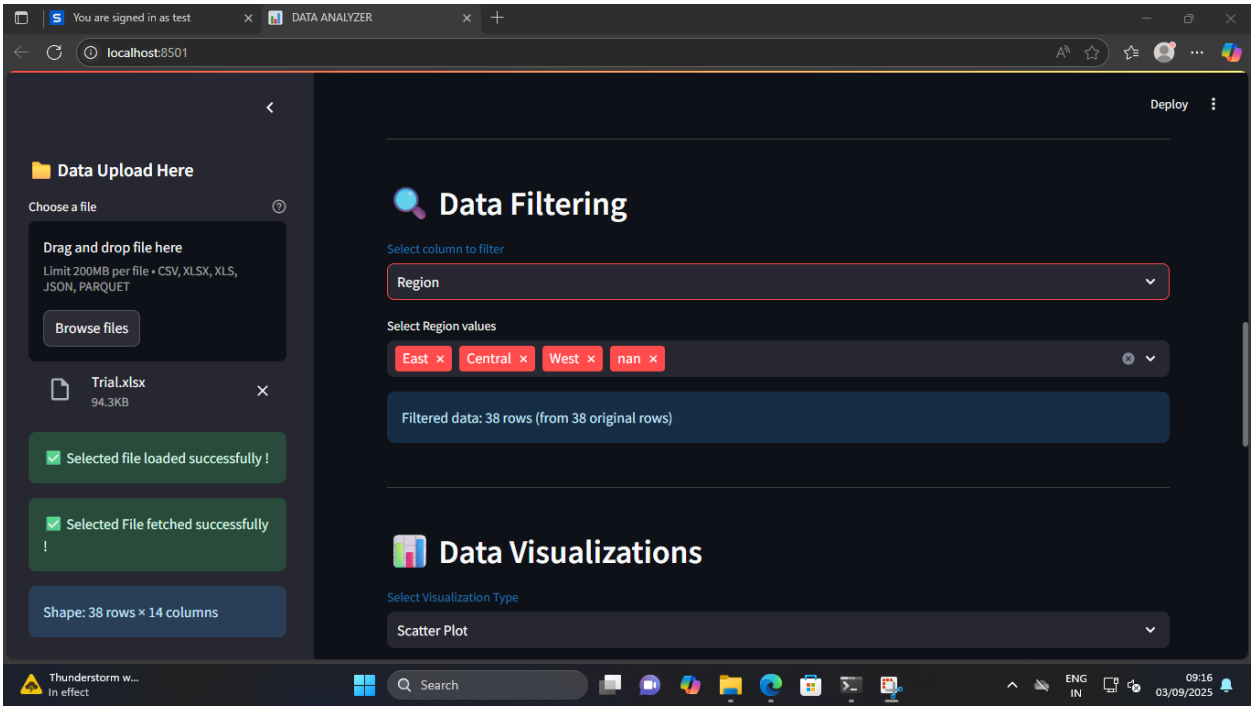


In this page user view all details of selected file.

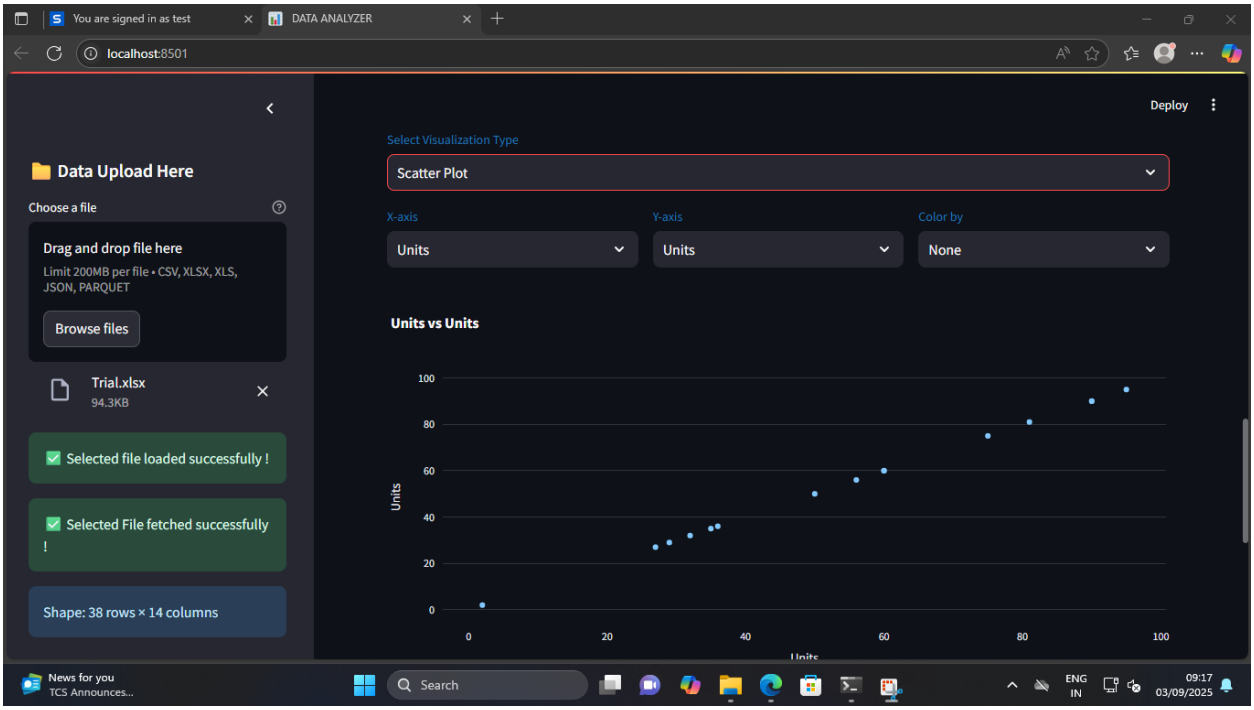




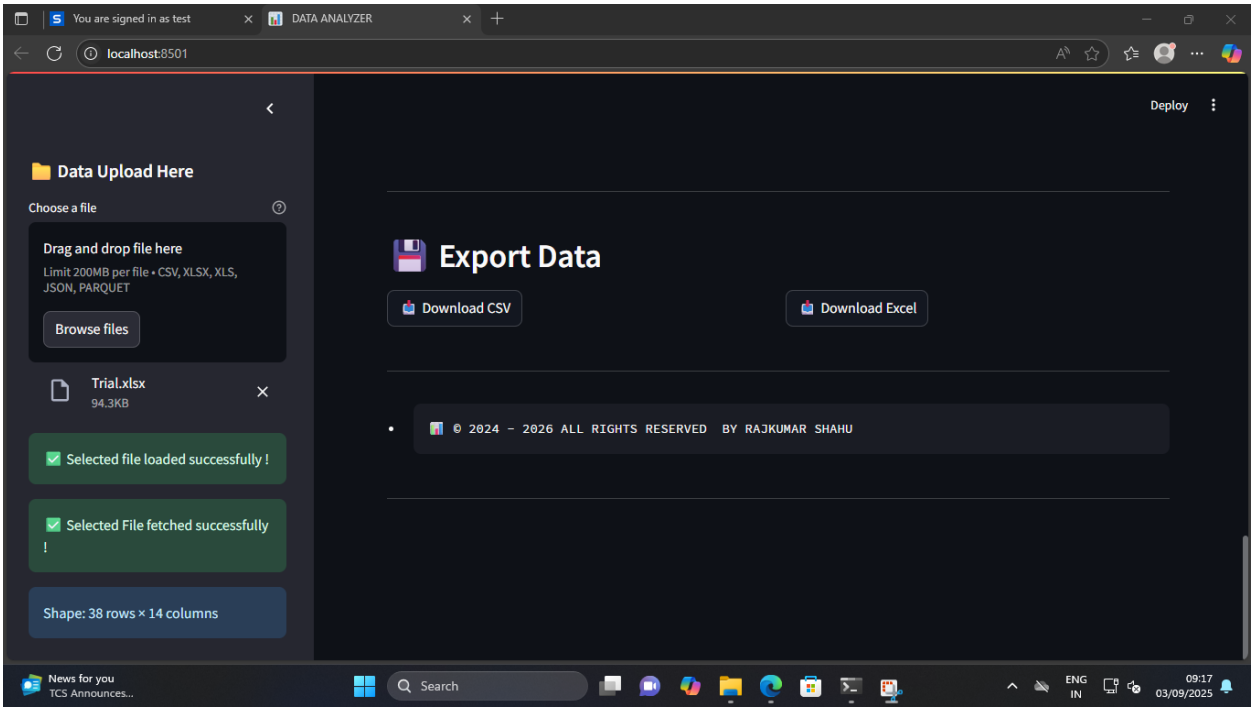
User can find a particular data in filtering section.



Visualization View.



Export data section.



Troubleshooting

Common Issues and Solutions

1. File Upload Errors

Problem: "Error loading file" message **Solutions:**

- Check file format (CSV, Excel, JSON, Parquet only)
- Verify file size (<200MB)
- Ensure file is not corrupted
- Try saving Excel files as CSV

2. Memory Issues

Problem: Application crashes with large datasets **Solutions:**

- Reduce dataset size
- Use Parquet format for better efficiency
- Close other applications
- Increase system RAM

3. Visualization Errors

Problem: Charts not displaying or errors in chart creation **Solutions:**

- Ensure appropriate column types selected
- Check for missing values in selected columns
- Verify sufficient data points
- Try different chart types

4. Browser Compatibility

Problem: Features not working in certain browsers **Solutions:**

- Use Chrome, Firefox, or Edge (latest versions)
- Check browser console for errors

Best Practices

Data Preparation

1. **Clean Data**: Remove unnecessary columns and rows
2. **Consistent Formatting**: Use standard date/time formats
3. **Proper Headers**: Include descriptive column names
4. **Data Types**: Ensure numeric columns are properly formatted
5. **File Size**: Keep files under 100MB for optimal performance

Visualization Guidelines

1. **Choose Appropriate Charts**: Match chart type to data type
2. **Limit Categories**: Use top 10-15 categories for clarity
3. **Color Coding**: Use consistent color schemes
4. **Clear Labels**: Ensure axes and titles are descriptive
5. **Interactive Elements**: Leverage hover effects and filtering

Performance Optimization

1. **Sample Large Datasets**: Use representative samples
2. **Cache Results**: Leverage Streamlit caching
3. **Efficient Filtering**: Apply filters before visualization
4. **Memory Management**: Clear unnecessary variables
5. **Browser Resources**: Close unused tabs

Conclusion

In this project, we developed an interactive **Data Analyzer and Visualizer** application using **Python** and **Streamlit**. The tool allows users to upload datasets, explore them through summary statistics, clean data, and generate a variety of insightful visualizations with ease.

By integrating libraries such as **Pandas**, **Matplotlib**, **Seaborn**, and **Plotly**, we provided dynamic data exploration capabilities, including:

- Viewing dataset structure and summary statistics
- Handling missing values and filtering data
- Creating visualizations like bar charts, histograms, scatter plots, heatmaps, and more
- Exporting cleaned or filtered data for further use

This project demonstrates how data science tools can be made accessible through user-friendly web interfaces. It serves as a foundation for more advanced analytics applications, where users can gain insights from their data without writing any code.

In future iterations, this tool can be enhanced with features like:

- Machine learning model integration
- Real-time data updates
- Dashboard export capabilities
- Collaboration tools or database integration

Overall, the project showcases the power of combining Python's data stack with Streamlit's simplicity to democratize data analysis for both technical and non-technical users.

Reference

Claude. AI