**1. INTRODUCTION**

**1.1 About the Project**

The **Data Visualization of Academic Audit Data** application is designed to assist users in analyzing and interpreting academic audit data through interactive and customizable visual representations. Academic audits play a crucial role in assessing institutional performance, ensuring compliance with academic standards, and identifying areas for improvement. However, raw data can often be complex and difficult to interpret without effective visualization tools.

This application simplifies the process by enabling users to upload datasets, dynamically select data for visualization, and generate insightful charts. Users can customize chart elements such as colors, grid settings, and axis selection to enhance readability and relevance. The interactive web-based dashboard allows seamless data exploration, and users can export charts for reports, presentations, or further analysis. The application is built using Python (Flask) for backend processing, SQLite for data management, and HTML/CSS/JavaScript for frontend interactions.

**1.2 Objective(s) of this Project**

The primary goal of this project is to provide a user-friendly data visualization platform that simplifies data processing and representation**.**

**1.3 Scope of this Project**

* Data Upload and Management: Users can upload datasets, which are then stored for further processing and visualization.
* Customization Options: The application offers customization features such as color, grid settings, and axis selection to enhance visualization clarity.
* Chart Generation: Users can select specific data columns to generate customized charts, making analysis more flexible and user-friendly.

**2. SYSTEM STUDY**

**2.1 Existing System**

In the current system, users manually generate charts using Excel, which can be a tedious and time-consuming process, especially when dealing with large datasets. There is no centralized database for data storage, meaning users must manage and organize their data files manually, leading to inconsistencies, redundancy, and difficulties in data retrieval. Additionally, the lack of automation requires users to repeatedly select and process data for visualization, making the analysis process inefficient. As datasets grow in size and complexity, generating meaningful insights becomes increasingly challenging, often resulting in errors and delays in decision making. The absence of interactive features further limits the ability to explore data dynamically, reducing the overall effectiveness of academic audit analysis.

**2.2 Disadvantages of Existing System**

* No automation in chart creation.
* Lack of interactive web interface for dynamic visualization.

**2.3 Proposed System**

The system features automated chart generation directly from an SQLite database, ensuring that visualizations are dynamically updated as data changes. With a web-based UI, users can interact with engaging dashboards that facilitate in-depth exploration and analysis of academic audit data. Together, these features create a cohesive environment where raw data is effortlessly transformed into meaningful insights, enhancing decision making and academic performance evaluation.

**2.4 Advantages of Proposed System**

* User-friendly dashboard with dynamic inputs.
* Flexible export options for charts.

**2.5 Problem Definition and Description**

**Problem Definition**

Data management and visualization are critical components of effective decision-making across various industries, including education. Academic institutions generate vast amounts of data related to audits, assessments, and performance evaluations. However, managing this data efficiently, maintaining its integrity, and extracting meaningful insights can be complex and time-consuming. Traditional methods of handling academic audit data often involve manual processes, such as spreadsheet-based tracking and static reporting, which are prone to errors, inconsistencies, and inefficiencies. Without proper visualization tools, stakeholders struggle to interpret large datasets, identify trends, and make data-driven decisions.

To address these challenges, there is a need for an automated, web-based system that simplifies data handling, ensures structured storage, and provides interactive visual analytics. Such a system would enhance the accuracy and efficiency of academic audit evaluations, facilitating better decision-making and institutional improvements.Organizations struggle with managing and visualizing structured data from Excel files. Manual processing is time-consuming and prone to errors.

**Problem Description**

The project includes the following modules:

1. **Import Data**: Allows users to upload data in .xlsx format and specify the table name for import. After successful data upload, users can proceed to the Category module.
2. **Category**: Users can select a table, specify a year, choose chart types, and set X and Y axis values to prepare the data for visualization.
3. **Charts**: Displays the generated chart based on the selected parameters. It allows users to view and export the chart to different formats.
4. **Export** : Provides options for exporting the generated chart in formats like PDF or PNG. Users can specify the file name and folder location before exporting.

**3. SYSTEM ANALYSIS**

* 1. **Packages Selected**
* **Front-end:** HTML5 , CSS3 , JavaScript
* **Back-end:** Python 3.8
  1. **Resources Required**

**3.2.1 Hardware Resources**

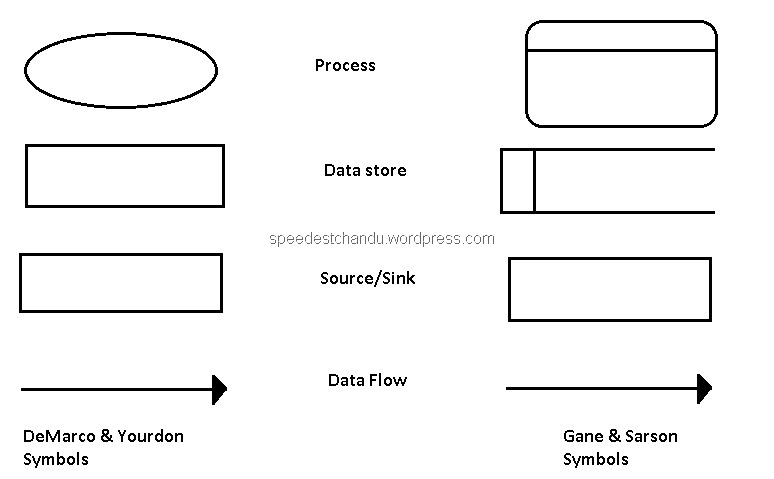
* **Processor**: Intel Core i3 1.8 GHz or higher
* **RAM**: 4GB minimum
* **Storage**: 500MB free space

**3.2.2 Software Resources**

* **Framework :** Python 3.8 with Flask 2.3
* **Database:** SQLite 3.47
* **Libraries:** Pandas , Matplotlib , Openpyxl , Werkzeug

**3.3 Data Flow Diagram**

**** A Data Flow Diagram (DFD) is a graphical representation that illustrates how data flows within a system, highlighting the processes, data stores, and external entities that interact with one another. DFDs are instrumental in understanding, designing, and documenting the way data moves through systems.

****

Process

****

Data Store

Source / Sink

Data Flow

**Data Flow Diagram (Level – 0)**

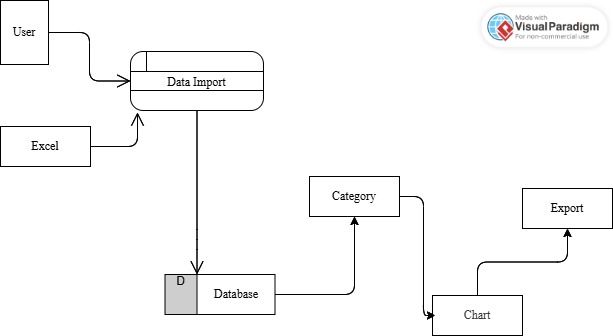
Data Store

Export

Data Import

User

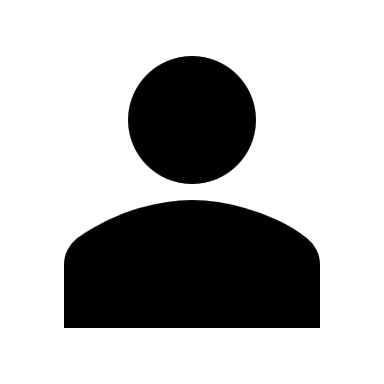
**Data Flow Diagram (Level – 1)**

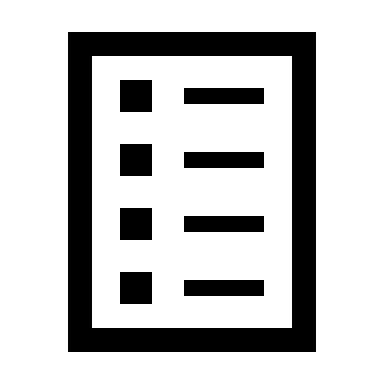
****

**4. SYSTEM DESIGN**

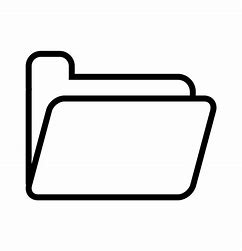
**4.1 Architectural Design**

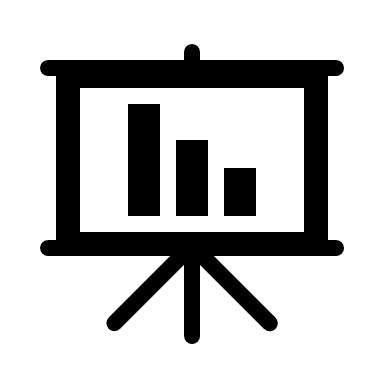
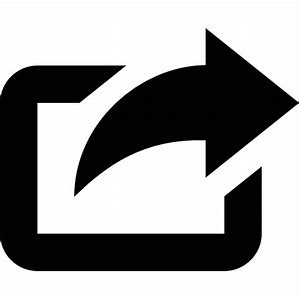
****

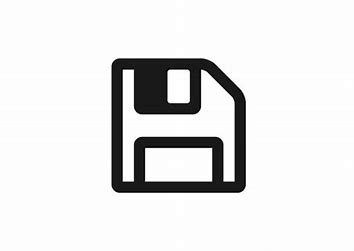
****

****

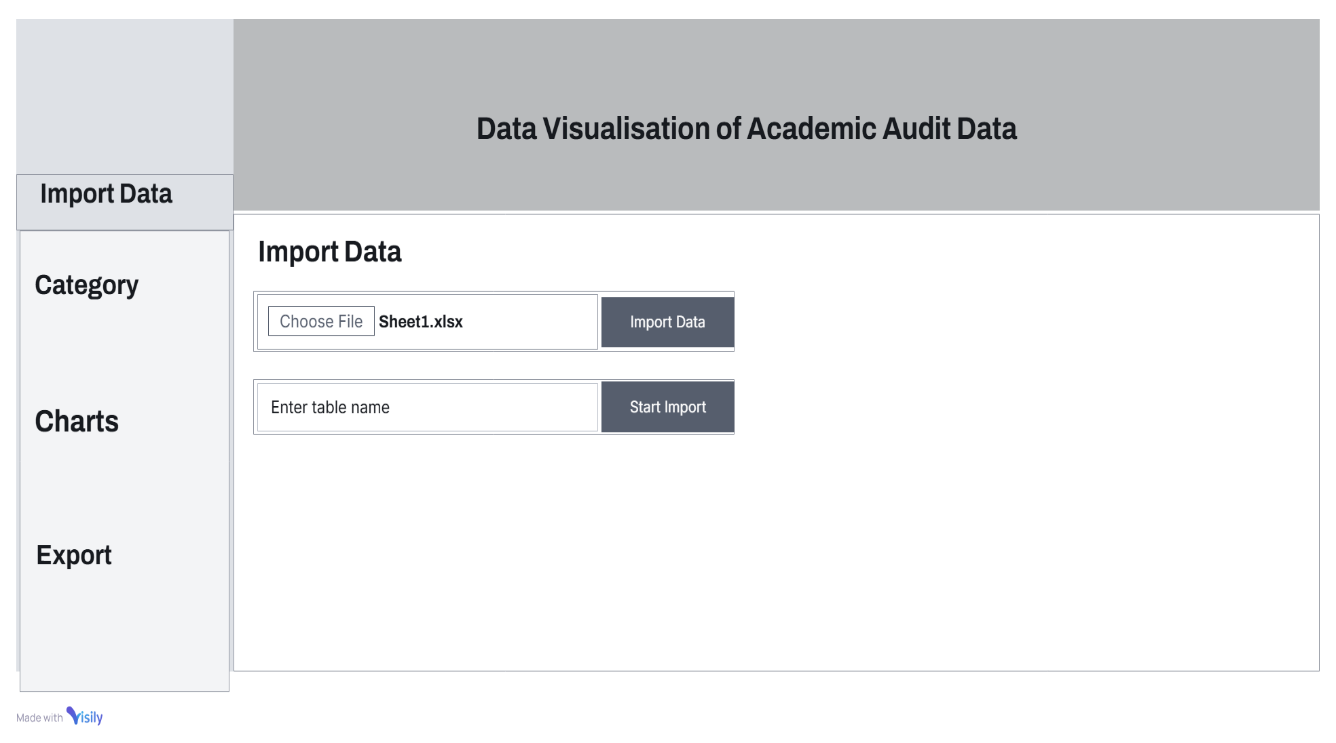
****

****

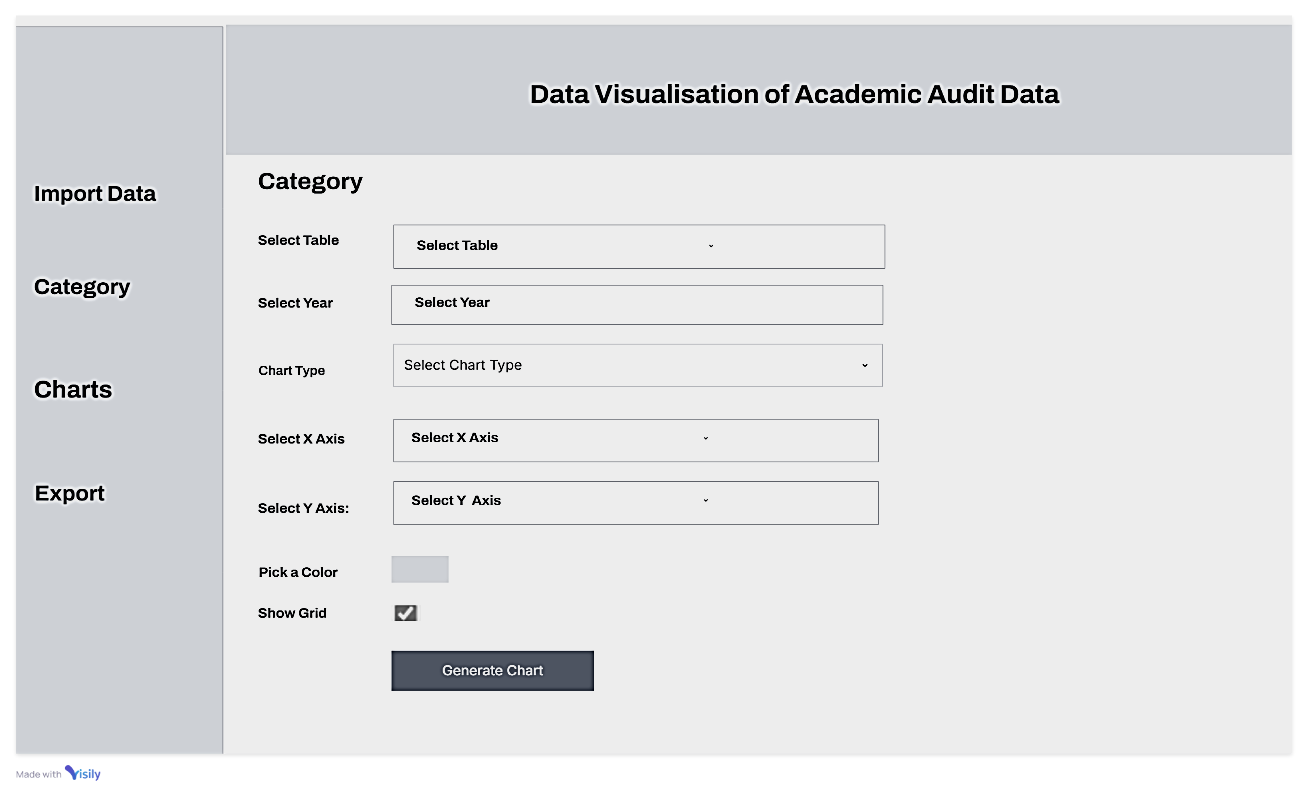
****

****

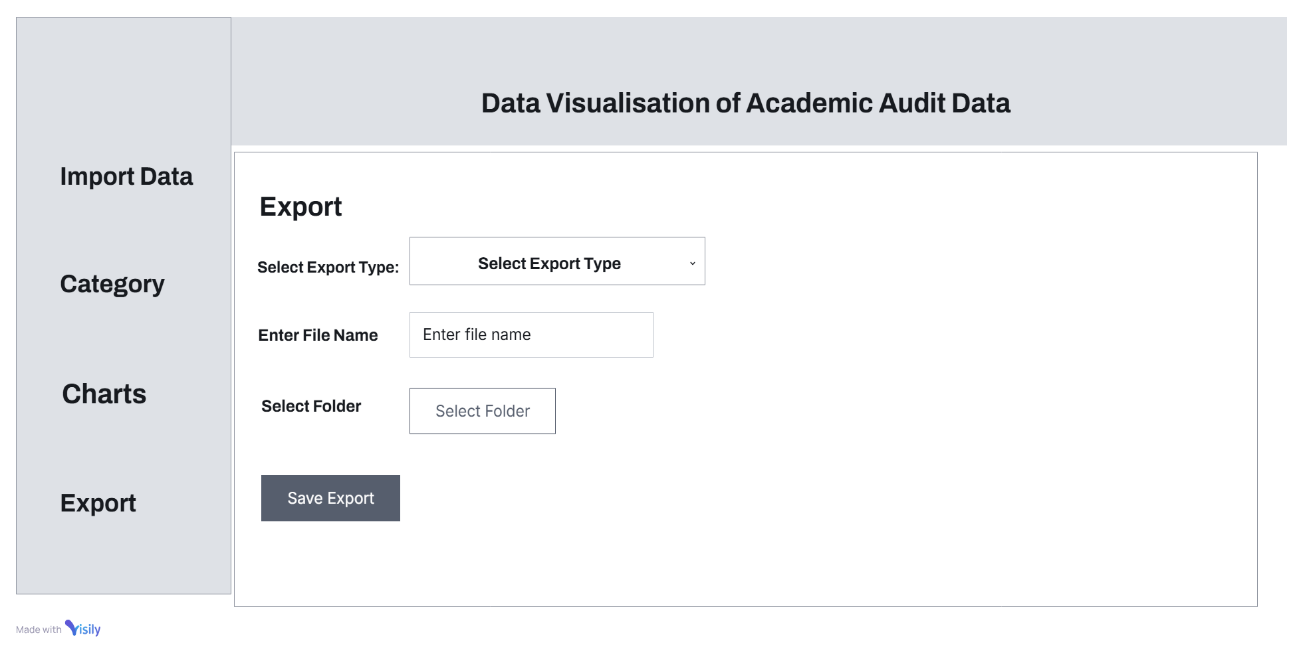
**4.2 I/O Form Design**

** Form :** Import Data

The above form is Import Data page allows users to upload only Excel files for data processing.

 **Form:** Category

The above form is Category Selection screen lets users choose a table, year, chart type, X & Y axes, and colors. It includes a grid option and a 'Generate Chart' button for visualization.

** Form:** Export

The Export page allows users to choose an export type (PDF/PNG), enter a file name, select a folder, and save the exported file.

**4.3 Tables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Width** | **Constraints** |
| Year | INT | - | NOT NULL |
| Title\_of\_the\_Course | VARCHAR | 55 | NOT NULL |
| Course\_Code | VARCHAR | 15 | PRIMARY KEY , UNIQUE |
| Title\_of\_the\_Content | VARCHAR | 60 | NOT NULL |
| Nature\_of\_e\_content | VARCHAR | - | NULL |
| Department | VARCHAR | - | NULL |

**Table:** Course Data

**Table:** Faculty Record

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Width** | **Constraints** |
| Year | INT | - | NOT NULL |
| Name\_of\_the\_Faculty | VARCHAR | 30 | PRIMARY KEY |
| Title\_of\_the\_Talk | VARCHAR | 100 | NOT NULL |
| Nature | VARCHAR | - | NULL |
| Type\_of\_Programme | VARCHAR | - | NULL |
| Title\_of\_the\_Seminar | VARCHAR | 80 | NULL |
| Institution\_and\_Place | VARCHAR | 90 | NULL |
| Date | DATE | - | NOT NULL |

**Table:** Participation Tracking

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Width** | **Constraints** |
| Year | INT | - | NOT NULL |
| Name\_of\_the\_Faculty | VARCHAR | 25 | PRIMARY KEY |
| Nature | TEXT | - | NULL |
| Type\_of\_Programme | TEXT | - | NULL |
| Name\_of\_the\_Program | VARCHAR | 10 | NOT NULL |
| Name\_and\_Place\_of\_the\_Institution | VARCHAR | 90 | NULL |
| Date | DATE | - | NOT NULL |
| Duration\_in\_Days | INT | 4 | NOT NULL |

**4.4 Data Dictionary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Table Name** | **Datatype** | **Description** | **Sample Data** |
| Course\_Code | Course Data | VARCHAR | Unique identify of a course. | 23PCA1EL02B |
| Date | Faculty Record, Participation Tracking | DATE | Date when program took place. | 31.01.2024 |
| Department | Course Data | TEXT | The department offering the course. | DR,J |
| Duration\_in\_Days | Participation Tracking | INT | The Duration of the program in days. | 5 |
| Institution\_and\_Place | Faculty Record | VARCHAR | Name and location of the institution conducting the seminar or event. | St. Josephs College, Tiruchirappalli |
| Name\_and\_Place\_of\_the\_Institution | Participation Tracking | VARCHAR | Name and place of the institution where the program was held. | Loyola College(Autonomous), Chennai |
| Name\_of\_the\_Faculty | Faculty Record, Participation Tracking | VARCHAR | Name of the faculty member involved in the event. | Dr. A. Aloysius |
| Name\_of\_the\_Programme | Participation Tracking | VARCHAR | Name of the academic or training program. | AI Powered Presentation Tools |
| Nature | Faculty Record, Participation Tracking | TEXT | The type or nature of the program or event. | N,IN |
| Nature\_of\_e\_content | Course Data | TEXT | Specifies the type of electronic content | PPT,PDF |
| Title\_of\_the\_Content | Course Data | VARCHAR | Name or topic of the content. | Event Handling |
| Title\_of\_the\_Course | Course Data | VARCHAR | Title of the academic course. | Computer Networks |
| Title\_of\_the\_Seminar | Faculty Record | VARCHAR | Title of the seminar in which faculty participated. | A Special Invited Talk |
| Title\_of\_the\_Talk | Faculty Record | VARCHAR | Title of the talk delivered by the faculty member. | ICT Tools |
| Type\_of\_Programme | Faculty Record, Participation Tracking | TEXT | Category of the program | WS,O,S |
| Year | Course Data ,Faculty Record, Participation Tracking | INT | The academic or event year. | 2023-2024 |

**5. SYSTEM DEVELOPMENT**

**5.1 Functional Documentation**

The Project **Data Visualization of Academic Audit Data** contains following module

**Import Data**

1. **Functionality**

* Users upload .xlsx files via the frontend
* The system validates the file and maps it to an existing SQLite table
* Stores only new records, avoiding duplicates

1. **Operations**

* User selects .xlsx file and clicks Upload
* Flask backend verifies the file and extracts data
* Checks if table exists, then inserts only new data
* Sends a success or failure response to the frontend

**Category**

1. **Functionality**

* Users select a table from the category page.
* Fetches column names and identifies year-based filtering options.

1. **Operations**:

* User selects a table from dropdown.
* System retrieves table structure (columns & year values).
* Displays year-based filtering options.

**Chart Generation**

1. **Functionality:**

* Generates Bar, Line, or Pie charts based on user selections.
* Supports X & Y axis selection, color customization, and grid options.

1. **Operations:**

* User selects chart type, X & Y axis, and color options.
* System fetches filtered data based on user choices.
* Generates chart dynamically and displays it.

**Export Chart**

1. **Functionality:**

* Exports generated charts as PNG or PDF.
* Users choose file name and folder location.

1. **Operations:**

* User selects export format (PNG/PDF) and file details.
* System verifies chart availability.
* Saves chart to the chosen location.

**5.2 Special Features of Language / Utility**

**5.2.1 HTML (Hyper Text Markup Language)**

* Semantic Structure: Uses structured elements like <header>, <div>, <select>, and <button> to create a well-organized UI.
* Form Elements: Includes <input> for file uploads and text inputs, <select> for dropdowns, and <button> for interactions.
* Modal Popups: Uses <div> elements for popups and notifications.
* Responsive Design: <meta viewport> ensures adaptability to different screen sizes.

**5.2.2 CSS (Cascading Style Sheets)**

* Dropdown Styling: Custom dropdown with hover effect to display selectable years dynamically.
* Popup Styling: Uses position: fixed, box-shadow, and border-radius for a modern, user-friendly popup design.
* Flexbox Layout: display: flex; ensures structured sidebars and content areas.
* Color Customization: Provides a color picker (<input type="color">) for chart customization.

**5.2.3 JavaScript**

**Dynamic UI Updates:**

* Uses document.querySelector() to manipulate the DOM dynamically.
* showContent(page) dynamically loads sections (Import, Category, Charts, Export).

**Popup Notification System:**

* + showPopup(message) displays alerts with auto-close functionality.

**File Handling:**

* + document.getElementById('file-uploader').files[0] allows .xlsx file uploads.

**Fetch API for Data Handling:**

* Uses fetch('/upload', { method: 'POST' }) for sending files to the server.
* fetch('/get\_tables') retrieves database tables dynamically.

**Data Visualization Support:**

* + Generates charts based on selected parameters (chart-type, x-axis, y-axis, etc.).

**Checkbox-based Multi-Selection:**

* + Dynamically adds checkboxes for selecting years in year-dropdown.

**Validation & Error Handling:**

* + Ensures required inputs are selected before proceeding to the next step.

**5.2.4 Backend Interactions**

* AJAX Calls with Fetch API: Enables seamless interaction between frontend and backend.
* Database Connectivity: Retrieves tables and columns dynamically for data visualization.
* Export Functionality: Supports PDF and PNG file exports (/export API).

**5.2.5 Utility Features**

* Interactivity: Sidebar navigation with event-based content display.
* Cross-Browser Compatibility: Ensures UI and functions work across multiple browsers.
* Custom Folder Selection: Allows users to choose export file locations via selectFolder().

**5.3 Pseudo Code**

**Import Data**

Step 1: Start

Step 2: Select .xlsx file.

Step 3: Enter table name for import.

Step 4: Click "Import Data" → File is uploaded.

Step 5: Data is parsed and stored in SQLite.

Step 6: Redirect to Category Page.

**Category**

Step 1: Fetch available table names.

Step 2: Populate dropdown with imported tables.

Step 3: Select a table → Fetch columns and unique years.

Step 4: Choose year(s) → Store selection.

Step 5: Configure chart settings (X-Axis, Y-Axis, Color, Grid).

Step 6: Click "Generate Chart" → Display chart.

**Export**

Step 1: Click "Export Chart".

Step 2: Select export format (PNG/PDF).

Step 3: Choose folder location.

Step 4: Click "Save" → Chart is saved in selected folder.

**6. TESTING**

* 1. **Types of Testing Done**

In this project validation testing is performed

**Form:** Import Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Test Data** | **Expected Result** |
| T01 | Clicking Import data without choosing a file, triggering an error message | Excel = Null | Please Upload .xlsx file |
| T02 | User uploads a CSV file, and the system validates the file format before proceeding. | Uploaded "data.csv" | Invalid file name. Use PS1.xlsx, PS2.xlsx, or PS3.xlsx. |
| T03 | A valid file is uploaded, but the corresponding database table is missing, resulting in an error. | Uploaded "PS4.xlsx" | Table PS4 does not exist. |
| T04 | A valid file is uploaded, and some records already exist in the database, possibly prompting an update or duplicate warning. | Uploaded "PS1.xlsx" | No new data to insert. |
| T05 | A valid file is uploaded containing entirely new records, which are successfully inserted into the database. | Uploaded "PS2.xlsx" | Data uploaded to sheet2 successfully |

**Form:** Export

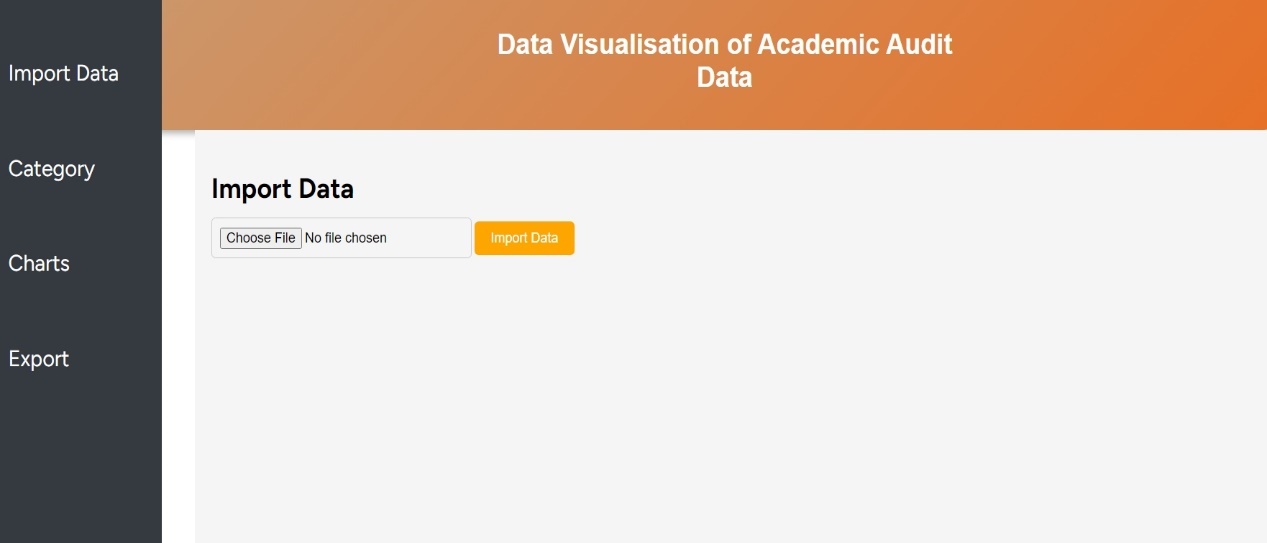
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Description | Test Data | Expected Result |
| T06 | User attempts to export data without choosing a destination folder, triggering an error message. | File format: PDF, file name provided, folder not selected | Missing required fields. |
| T07 | User initiates the export process but does not provide a file name, leading to a prompt or error. | File format: PNG, file name missing | Missing required fields. |
| T08 | User selects a destination folder and successfully exports the file with the specified name. | File format: PDF, file name provided, folder not selected | Export Successful!! Saved at “C://” |

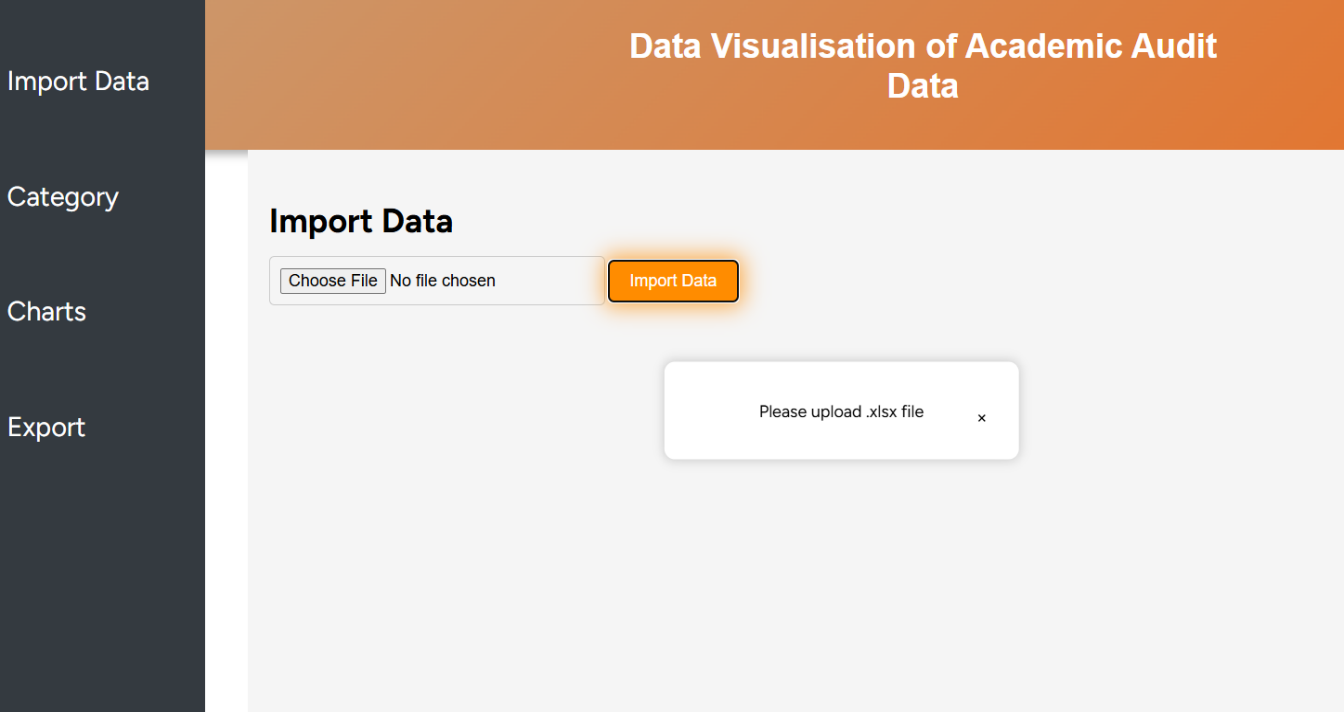
* 1. **Test Data & Output**

**Form:** Import Data

**Test Case ID:** T01

**Description:** Clicking Import data without choosing a file  
**Test Data:** Error =Null  
**Expected Result:** Please Upload .xlsx file

 Initial Screen

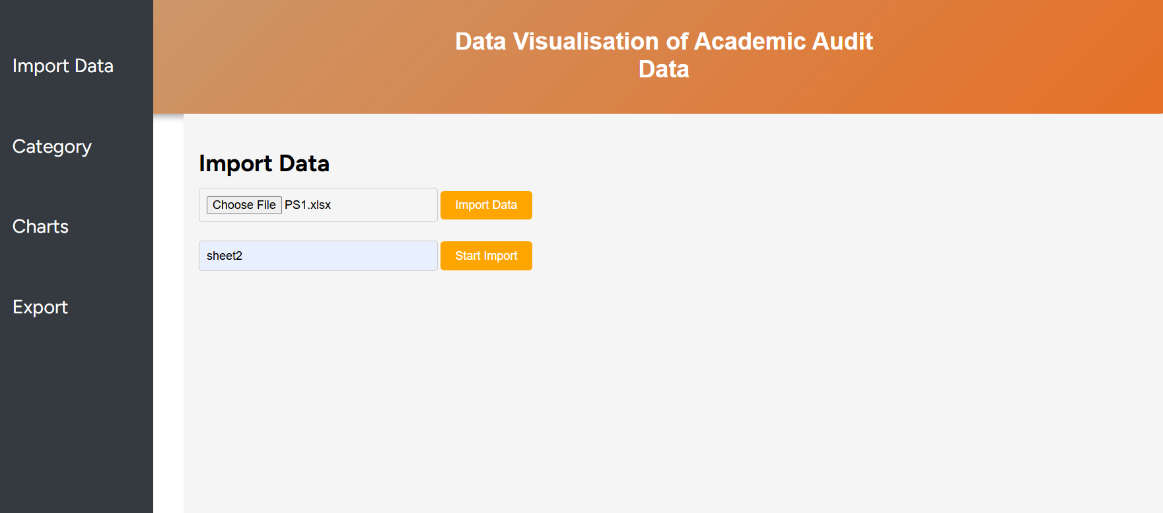
 Output Screen

**Form:** Import Data

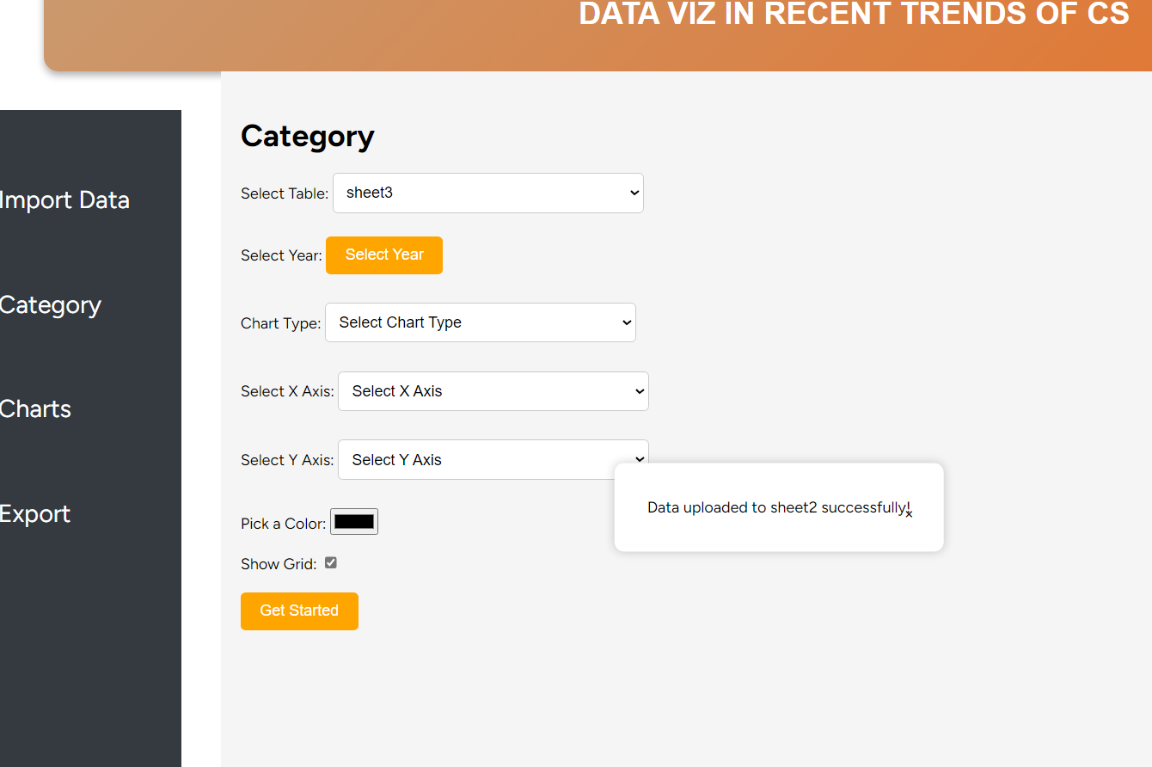
**Test Case ID:** T05

**Description:** A valid file is uploaded containing entirely new records, which are successfully inserted into the database   
**Test Data:** "PS2.xlsx"   
**Expected Result:** Data uploaded to sheet2 successfully!

Intial Screen



Output Screen



**7. USER MANUAL**

**7.1 Hardware Requirements**

* Processor: Any modern dual-core processor (Intel/AMD)
* RAM: 2 GB or more (Recommended: 4GB for smoother performance)
* Storage: Minimal space required (~100MB for installation)
* Display: Any standard screen resolution (1024×768 or higher)
* Internet Connection: Not required (only for updates)

**7.2 Software Requirements**

* Operating System: Windows 7/10/11 (64-bit), macOS 10.12+, or Linux (Ubuntu 16.04+)
* Pre-installed Dependencies: Not required – Everything is bundled with the app
* Web Browser: Any modern browser (Chrome, Firefox, Edge, etc.)

**7.3 Installation Procedure**

Download the **DataViz** Application

**Windows Users**

1. Download the latest **DataViz.exe**
2. Save the file to a folder of your choice (e.g., C:\MyApp\).

**macOS Users**

1. Download the latest **DataViz-mac.zip**
2. Extract the ZIP file to a location of your choice.

**Linux Users**

1. Download the latest **DataViz-linux.tar.gz**
2. Extract the tarball using:

**tar -xvzf DataViz-linux.tar.gz**

**Running the Application**

**Windows**

1. Open the folder where you saved **DataViz.exe.**
2. Double-click **DataViz.exe** to launch the application.
3. If a security warning appears, click "**Run Anyway**".
4. The app will open in your default web browser at:

**http://127.0.0.1:5000/**

**macOS**

1. Open the extracted folder.
2. Right-click on app and select "Open" (the first time, you may need to confirm security permissions).
3. If prompted, click "Open Anyway" in System Preferences > Security & Privacy.
4. The app will launch in your browser at:

**http://127.0.0.1:5000/**

**Linux**

1. Open the terminal and navigate to the extracted folder:

**cd path/to/extracted-folder**

1. Make the app executable:

**chmod +x DataViz**

1. Run the app:

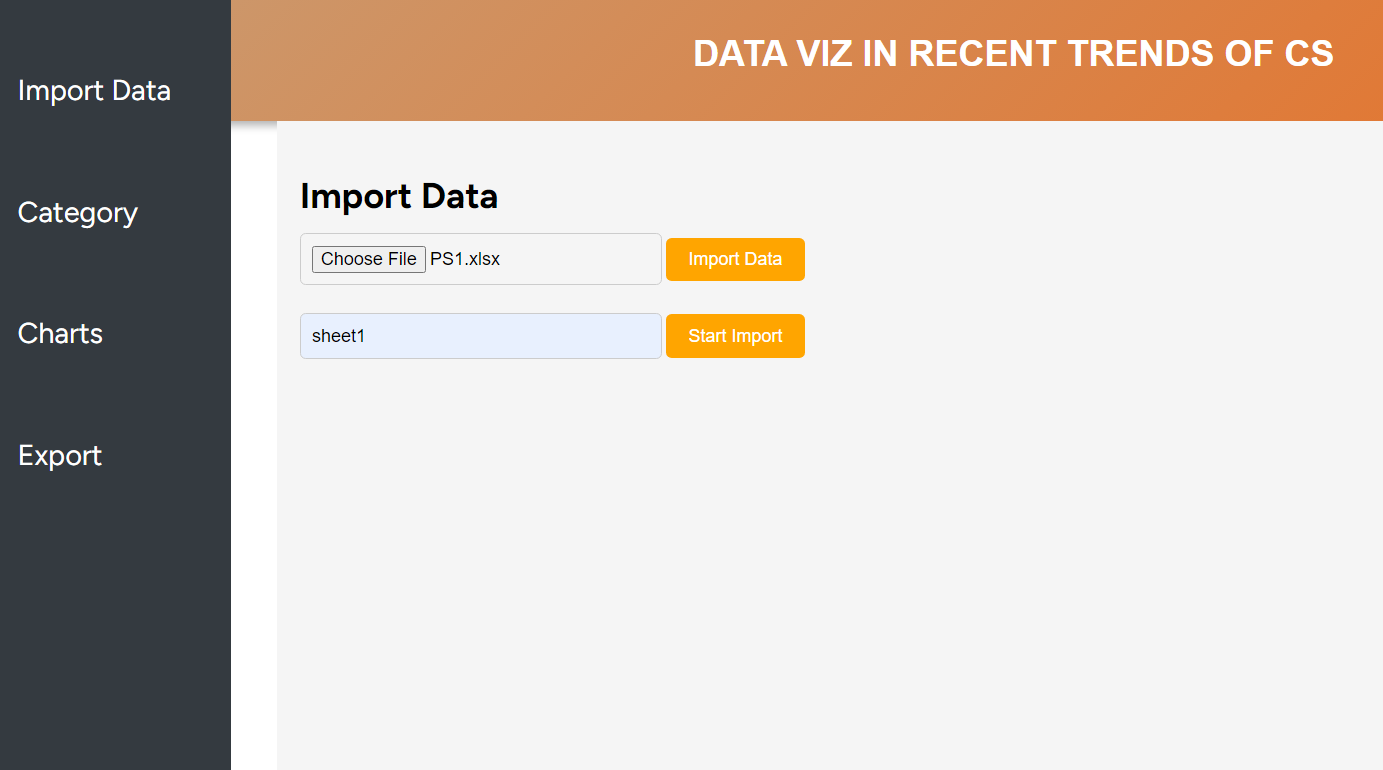
**./DataViz**

1. Open your browser and go to:

**http://127.0.0.1:5000/**

**7.4 Sample I/O**

**Form:** Import page



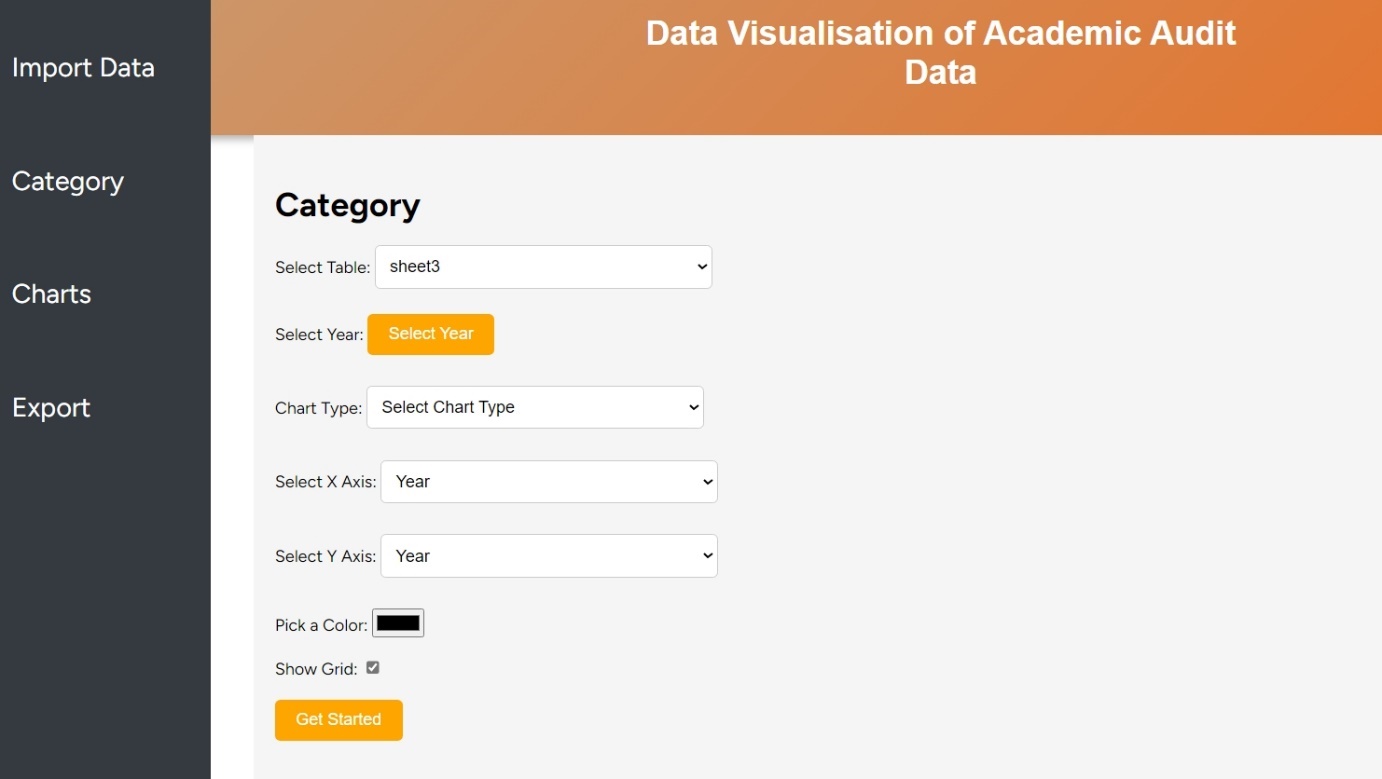
Choose File

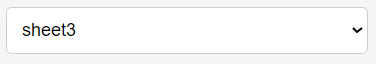
Uploader Button we can choose .xlsx File for upload

Start Import Stores Data in appropriate table in SQL Database

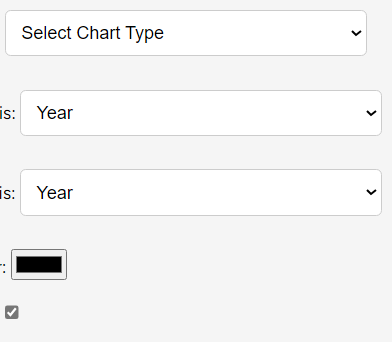
Start Import

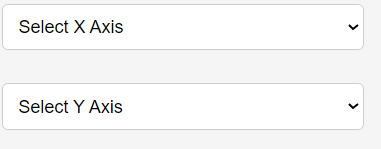
**Form:** Category



Table Dropdown which contains stored tables

Select year Button contains dropdowned-Check which contains years



Chart Dropdown contains Chart Type

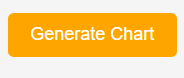
X axis Dropdown contains Columns of Table

Y axis Dropdown Contains Columns of Table



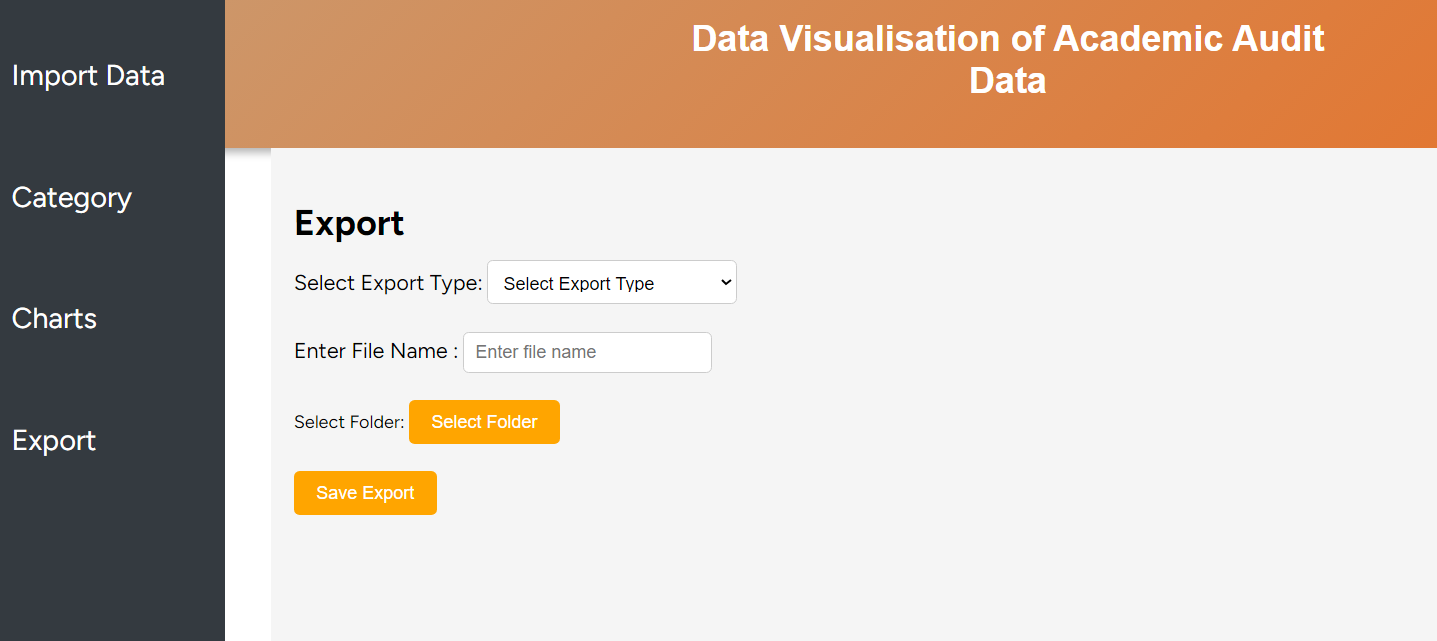
Color Picker for give color to chart

Grid Checkbox for Grid Chart



Generate chart button Generates Chart

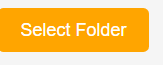
**Form:** Export



Export Type Dropdown contains format for the file



Enter File Name placeholder is customizable Field for giving File name

Select Folder Button asks folder for saving generated chart

Save Export Button saves the file in the selected folders

**7.5 Error Messages**

**a) Error Message:** "No file uploaded."

**Form:** Import Data

**Description:** The error occurs when clicks the "Import" button without selecting a file.

**Solution:** Select a valid Excel file (PS1.xlsx, PS2.xlsx, or PS3.xlsx) before clicking "Start Import."

**b) Error Message:** "No new data to insert."

**Form:** Import Data

**Description:** The uploaded file contains data in the database. No new entries are found.

**Solution**: Modify the file to include new or unique data and try again.

**c) Error Message:** "No folder selected."

**Form:** Export

**Description:** The user did not select a destination folder when trying to export a chart.

**Solution:** Choose a valid folder before proceeding with the export process.

**d) Error Message:** "Unexpected error occurred."

**Form:** Any Form

**Description:** A generic error due to system malfunction or incorrect input.

**Solution:** Refresh the page and try again.

**8.CONCLUSION**

**8.1 Summary of the Project**

The **Data Visualization of Academic Audit Data** application is a user-friendly application designed to transform raw data from Excel files into meaningful visual representations. The project enables users to seamlessly import data, store it securely in an SQLite database, and generate dynamic charts based on selected parameters. With support for various chart types like bar, line, and pie, users can customize their visualizations by selecting X and Y axes, filtering by year, and choosing colors and grid options. Additionally, the system allows for exporting charts as PNG or PDF, making it easy to share and utilize the visualized data. The project ensures efficient data handling, smooth user experience, and error-free operations, providing a simple yet powerful tool for non-technical users to analyze and interpret data effectively.

**8.2 Future Possibilities**

**1. New Categories of Data**

* Support for CSV and JSON file imports in addition to Excel

**2️. More Export & Sharing Options**

* Direct Email Export (Send charts via email)
* Generate Shareable Links for team collaboration
* Print Chart Feature for physical reports

**3️. Additional Pages & Features**

* Dashboard Page summarizing uploaded data & latest charts
* Data Insights Page showing basic analytics (min/max, trends)
* History/Logs Page for tracking previous imports & exports

**BIBLIOGRAPHY**

**Book Bibliography**

**Books for Reference**

1. Nathan Yau, Data Points: Visualization That Means Something, Wiley, 2013.
2. Ben Fry, Visualizing Data: Exploring and Explaining Data with the Processing Environment, O'Reilly Media, 2008.
3. Edward Tufte, The Visual Display of Quantitative Information, Graphics Press, 2001.

**Websites for Reference**

1. Matplotlib Documentation for Chart Customization

https://matplotlib.org/stable/contents.html

1. Pandas Data Handling Guide - https://pandas.pydata.org/docs/
2. Flask Framework Documentation - https://flask.palletsprojects.com/en/2.3.x/
3. SQLite for Python Applications - https://www.sqlite.org/index.html

**APPENDIX**

* 1. **Sample Code**

**Python**

from flask import Flask, render\_template, request, jsonify

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

from werkzeug.utils import secure\_filename

import os

import matplotlib

import shutil

import zipfile

import subprocess

from matplotlib.backends.backend\_pdf import PdfPages # Add this import

# Use 'Agg' backend to prevent GUI-related errors

matplotlib.use('Agg')

app = Flask(\_\_name\_\_)

app.secret\_key = 'your\_secret\_key'

# SQLite Database Configuration

DATABASE\_PATH = "lazarus.db"

CHARTS\_FOLDER = "static/charts"

os.makedirs(CHARTS\_FOLDER, exist\_ok=True)

@app.route('/')

def dashboard():

return render\_template('dashboard.html')

@app.route('/upload', methods=['POST'])

def upload\_file():

try:

file = request.files['file']

filename = secure\_filename(file.filename)

if not file:

return jsonify({"success": False, "message": "No file uploaded."}), 400

# 🔹 Mapping file names to pre-existing SQLite tables

table\_mapping = {

"PS1.xlsx": "sheet1",

"PS2.xlsx": "sheet2",

"PS3.xlsx": "sheet3"

}

table\_name = table\_mapping.get(filename)

if not table\_name:

return jsonify({"success": False, "message": "Invalid file name. Use PS1.xlsx, PS2.xlsx, or PS3.xlsx."}), 400

df = pd.read\_excel(file)

# 🔹 Connect to SQLite database

conn = sqlite3.connect(DATABASE\_PATH)

cursor = conn.cursor()

# 🔹 Ensure table exists

cursor.execute(f"SELECT name FROM sqlite\_master WHERE type='table' AND name=?", (table\_name,))

if not cursor.fetchone():

return jsonify({"success": False, "message": f"Table {table\_name} does not exist."}), 400

existing\_data = pd.read\_sql\_query(f"SELECT \* FROM {table\_name}", conn)

new\_data = df[~df.apply(tuple, axis=1).isin(existing\_data.apply(tuple, axis=1))]

if new\_data.empty:

return jsonify({"success": False, "message": "No new data to insert."}), 200

# Append new data

new\_data.to\_sql(table\_name, conn, if\_exists='append', index=False)

conn.commit()

conn.close()

return jsonify({"success": True, "message": f"Data uploaded to {table\_name} successfully!"})

except Exception as e:

return jsonify({"success": False, "message": str(e)}), 500

@app.route('/get\_tables', methods=['GET'])

def get\_tables():

try:

conn = sqlite3.connect(DATABASE\_PATH)

cursor = conn.cursor()

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table'AND name NOT LIKE 'sqlite\_sequence';")

tables = [row[0] for row in cursor.fetchall()]

conn.close()

return jsonify({"success": True, "tables": tables})

except Exception as e:

return jsonify({"success": False, "message": str(e)}), 500

@app.route('/get\_tables', methods=['GET'])

def fetch\_tables():

tables = get\_tables()

return jsonify({"success": True, "tables": tables})

@app.route('/get\_columns', methods=['POST'])

def get\_columns():

try:

table\_name = request.json.get('table\_name')

if not table\_name:

return jsonify({"success": False, "message": "Table name is required."}), 400

conn = sqlite3.connect(DATABASE\_PATH)

cursor = conn.cursor()

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name=?", (table\_name,))

table\_check = cursor.fetchone()

if not table\_check:

return jsonify({"success": False, "message": "Table does not exist."}), 404

cursor.execute(f"PRAGMA table\_info({table\_name})")

columns = [row[1] for row in cursor.fetchall()]

year\_column = next((col for col in columns if "year" in col.lower()), None)

years = []

if year\_column:

cursor.execute(f"SELECT DISTINCT {year\_column} FROM {table\_name} ORDER BY {year\_column} ASC")

years = [row[0] for row in cursor.fetchall() if row[0] is not None]

conn.close()

return jsonify({"success": True, "columns": columns, "year\_column": year\_column, "years": years})

except Exception as e:

return jsonify({"success": False, "message": f"Error fetching columns: {str(e)}"}), 500

@app.route('/generate\_chart', methods=['POST'])

def generate\_chart():

try:

data = request.json

table\_name = data['table\_name']

year = data.get('year', None)

chart\_type = data['chart\_type']

x\_axis = data['x\_axis']

y\_axis = data['y\_axis']

color = data['color']

show\_grid = data.get('show\_grid', True)

conn = sqlite3.connect(DATABASE\_PATH)

query = f"SELECT {x\_axis}, {y\_axis} FROM {table\_name}"

if year:

query += f" WHERE year = '{year}'"

df = pd.read\_sql\_query(query, conn)

conn.close()

if df.empty:

return jsonify({"success": False, "message": "No data available for the selected criteria."})

df.dropna(subset=[x\_axis, y\_axis], inplace=True)

df[x\_axis] = df[x\_axis].astype(str)

df[y\_axis] = df[y\_axis].astype(str)

plt.figure(figsize=(10, 6))

if chart\_type == "bar":

df\_counts = df.groupby([x\_axis, y\_axis]).size().reset\_index(name='count')

df\_pivot = df\_counts.pivot(index=x\_axis, columns=y\_axis, values='count').fillna(0)

df\_pivot.plot(kind='bar', color=color, ax=plt.gca())

elif chart\_type == "line":

df\_counts = df.groupby([x\_axis, y\_axis]).size().reset\_index(name='count')

df\_pivot = df\_counts.pivot(index=x\_axis, columns=y\_axis, values='count').fillna(0)

df\_pivot.plot(kind='line', marker='o', color=color, ax=plt.gca())

elif chart\_type == "pie":

df\_pie = df[y\_axis].value\_counts()

df\_pie.plot(kind='pie', autopct='%1.1f%%', colors=[color], ax=plt.gca())

if show\_grid:

plt.grid(True)

plt.xlabel(x\_axis)

plt.ylabel(y\_axis)

#plt.title(f"{chart\_type.capitalize()} Chart of {y\_axis} vs {x\_axis}")

chart\_path = os.path.join(CHARTS\_FOLDER, "chart.png")

plt.savefig(chart\_path)

plt.close()

return jsonify({"success": True, "chart\_url": f"/{chart\_path}"})

except Exception as e:

return jsonify({"success": False, "message": str(e)}), 500

@app.route('/export', methods=['POST'])

def export\_chart():

try:

export\_type = request.form.get('export\_type').lower() # 'png' or 'pdf'

file\_name = request.form.get('file\_name') # Example: 'my\_chart'

file\_location = request.form.get('file\_location') # Example: '/path/to/save'

if not export\_type or not file\_name or not file\_location:

return jsonify({'success': False, 'message': 'Missing required fields.'})

source\_path = os.path.join(CHARTS\_FOLDER, "chart.png") # The generated chart

export\_path = os.path.join(file\_location, f"{file\_name}.{export\_type}")

if not os.path.exists(source\_path):

return jsonify({'success': False, 'message': 'No chart found. Generate a chart first.'})

if export\_type == "png":

shutil.copy(source\_path, export\_path) # Copy PNG directly

elif export\_type == "pdf":

with PdfPages(export\_path) as pdf:

img = plt.imread(source\_path) # Read the PNG image

plt.figure(figsize=(10, 6))

plt.imshow(img)

plt.axis('off') # Hide axis

pdf.savefig() # Save as PDF

plt.close()

else:

return jsonify({'success': False, 'message': 'Unsupported export format. Use PNG or PDF.'})

return jsonify({'success': True, 'message': f'Exported successfully as {export\_type.upper()}', 'location': export\_path})

except Exception as e:

return jsonify({'success': False, 'message': f'Error: {str(e)}'})

@app.route('/select\_folder', methods=['POST'])

def select\_folder():

try:

result = subprocess.run(['python', 'select\_folder\_helper.py'], capture\_output=True, text=True)

folder\_path = result.stdout.strip()

if folder\_path and folder\_path != "No folder selected.":

return jsonify({'folder\_path': folder\_path})

else:

error\_message = result.stderr.strip() or "No folder selected."

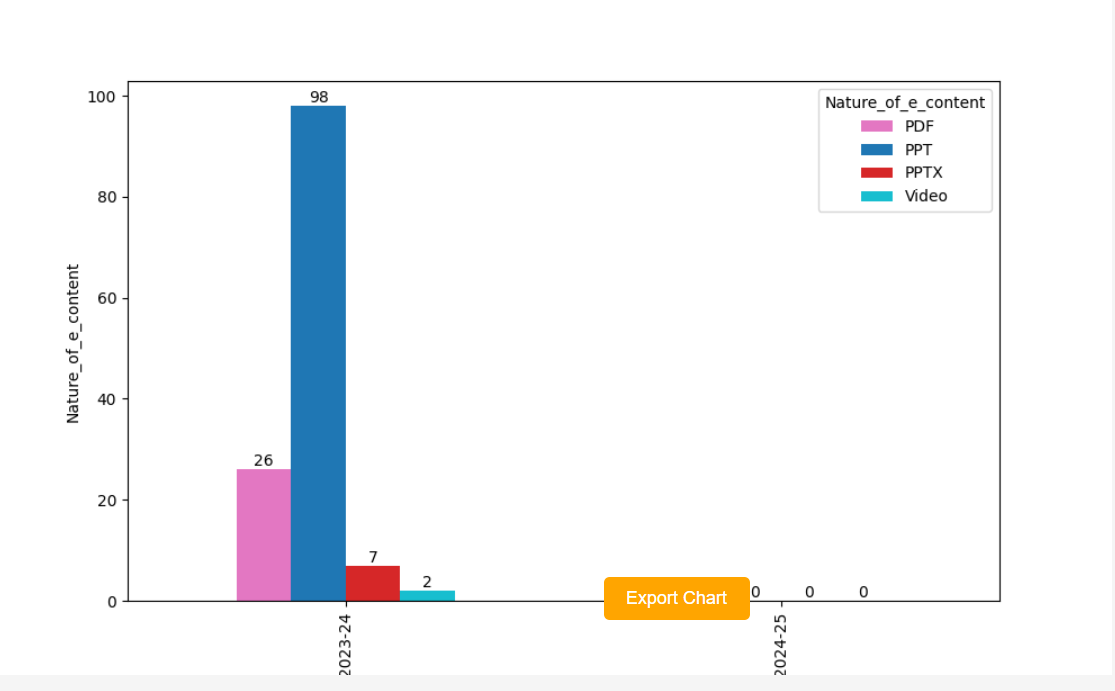
return jsonify({'folder\_path': None, 'error': error\_message})

except Exception as e:

return jsonify({'folder\_path': None, 'error': str(e)})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, threaded=False) # Disable threading to prevent socket errors

* 1. **Sample Output**

2023- 24