INTERACTIVE HTML DASHBOARD FOR DATA VISUALIZATION

Web Dashboard

Abstract

This is a web based interactive dashboard created using python

Sowmya Vejandla

Document id	Language	Version	Author	Page
DOC/PRJ-WD	En	1.0	Sowmya	1

Table of Contents

Step-by-Step Explanation of the Python Code for Web-Based Dashboard

This document provides a detailed, step-by-step explanation of a Python program that creates a web-based dashboard using Flask.
This program allows users to upload CSV or Excel files and perform basic analysis on the uploaded datasets. Let's go through each part of the code and explain its purpose and functionality.

1. Importing Libraries

- `Flask` is imported from `flask` to build the web application and handle HTTP requests and responses.
- `redirect`, `render_template`, `request`, and `jsonify` are used for various Flask operations, including rendering HTML pages and handling redirects.
- `os` is used to interact with the file system for handling uploaded files.
- `pandas` (imported as `pd`) is used to load and analyze data from CSV and Excel files.
- `secure_filename` is imported from `werkzeug.utils` to secure file names when saving uploads.

2. Setting Up Flask App and Configuring Upload Folder

- `app = Flask(__name__)`: Creates a Flask application instance.
- `app.config['UPLOAD_FOLDER'] = 'uploads'`: Specifies the folder where uploaded files will be saved.
- `os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)`: Ensures the 'uploads' folder exists; if not, it creates it.

3. Home Route (`/`)

- `@app.route('/')`: Sets the route for the homepage.
- `def home()`: Defines a function called `home` that will render `home.html` as the homepage.
- `render_template('home.html')`: Renders an HTML template named `home.html` to be displayed at the root URL (`/`).

4. Upload Route (`/upload`)

- `@app.route('/upload', methods=['POST'])`: Defines a route that handles file uploads, only accepting POST requests.
- `upload_file()`: A function that handles the file upload logic.

- `request.files.getlist('file')`: Retrieves the list of files uploaded through a form.
- For each uploaded file, the code checks if the file has a `.csv` or `.xlsx` extension.
- `secure_filename(file.filename)`: Secures the file name to prevent directory traversal attacks.
- `file.save(file_path)`: Saves the uploaded file to the specified path.
- `pd.read_csv(file_path)`: Reads the file if it's a CSV; otherwise, `pd.read_excel(file_path)` reads Excel files.
- `os.listdir(app.config['UPLOAD_FOLDER'])`: Lists all files in the upload folder, which is passed to `index.html`.

5. Analyze Route (`/analyze`)

- `@app.route('/analyze', methods=['POST'])`: Defines a route that allows analysis on uploaded files.
- `analyze()`: A function to perform basic analysis on a chosen dataset.
- `file_name = request.form['file_name']`: Gets the name of the selected file from the form.
- `pd.read_csv(file_path)`: Reads the file into a DataFrame.
- `data.columns.tolist()`: Extracts column names.
- `data.dtypes.astype(str).tolist()`: Retrieves data types for each column as strings.
- `data.head(10).to_dict(orient='records')`: Converts the first 10 rows into a dictionary for sample data display.
- `jsonify(data_summary)`: Returns a JSON object with column names, data types, and sample data.

6. Running the Flask App

- `if __name__ == '__main__':`: Checks if this script is being run directly.
- `app.run(debug=True)`: Runs the Flask app in debug mode, which provides more detailed error messages and enables auto-reload for development.

Step-by-Step Explanation of the Customizable Graph Generator

This document provides a detailed step-by-step explanation of an HTML and JavaScript code snippet that generates a customizable graph interface using D3.js. The application allows users to choose a dataset, select columns for the X and Y axes, and choose a chart type (scatter plot, bar chart, pie chart, or line chart) for visualization.

JavaScript Dataset Tabs Generation

Using JavaScript, the script dynamically generates tabs based on the datasets provided by the server. The `datasets1` variable is populated by converting server data to JSON. For each dataset, a tab is created and appended to the `tabsContainer`, and corresponding tab content with dropdowns for selecting columns and chart type is added to the `tabContentsContainer`.

Tab Content Generation and Event Handling

For each dataset, tab content includes dropdowns for selecting X-axis and Y-axis columns, a dropdown for chart type selection, a color picker, and a 'Generate Chart' button. Upon clicking, the dropdowns are populated with columns retrieved from the server.

showTab() Function

This function controls tab visibility by adding the 'active' class to the selected tab content, making it visible while hiding others. This function also adds a 'selected-tab' style to the active tab to visually highlight it.

generateChart() Function

This function, triggered by the 'Generate Chart' button, captures the selected options (X-axis, Y-axis, chart type, color) and fetches data from the server. It then clears any previous chart in the visualization area and sets up the chart dimensions.

Chart Creation Functions

Depending on the selected chart type, different functions are called to create a scatter plot, bar chart, pie chart, or line chart. These functions use D3.js to append SVG elements and scale the data accordingly for visual representation.

- createScatterPlot(): Generates a scatter plot by plotting circles based on X and Y values.

- createBarChart(): Creates a bar chart by appending rectangles for each data point.

- createPieChart(): Builds a pie chart by setting up arcs for each data segment.

- createLineChart(): Plots a line chart by connecting points in sequence.

SVG Element and D3.js

Each chart function utilizes D3.js for binding data to SVG elements. Scale functions are defined based on the selected data range, allowing accurate axis rendering. SVG attributes are set for colors, positions, and dimensions, providing an interactive and dynamic chart.