Assignment 2

PYTHON

FLASK APP

Md Rakibul Hasan

JR9RXP

Task Overview

Section 1A - Enhancement of stats_graphs.py

This segment focuses on the development and integration of graphical and statistical elements within the stats_graphs.py script.

Graphical Representations:

My objective is to design a series of five distinct graphs, incorporating a histogram and a 3D scatter plot. Key aspects to consider include:

1. Concentrating on representing the 'age', 'salary', and 'joindate' fields, with an emphasis on the year 2022 for 'joindate', showcasing only the months within that year.
2. Appropriate data type conversions within the dataframe are essential for accurate graph rendering.
3. Each graph should be distinctly titled, with clearly marked axes.
4. Implementing visible grids on each graph to enhance readability.

Statistical Analysis:

Developing three specific statistical computations, with a minimum of two being adaptations from the 4.3.2.4.ipynb notebook.

Section 1B - Augmentation of flask_app.py

This part entails the expansion of the flask_app.py script, ensuring the preservation of existing code.

Integration and Functionality Tasks:

Begin by importing the stats_graphs.py script, using sg as its alias, at the start of the flask_app.py file.

Invoke the methods or functions from the stats_graphs.py script to populate the statistical data.

Generate the requisite graphs by employing the functions established for graph creation in stats_graphs.py.

Reference the examples provided within the file to guide the implementation of the initial graph and statistical element.

Description of the classes and methods

stats_graphs.py – For generating statistics and graphs from a given CSV file:

1. generate_first_three_rows(data): Generates an HTML table displaying the first three rows of the provided DataFrame.

2. generate_average_age(data): Calculates and returns a string stating the average age from the provided DataFrame.

3. generate_stat1(data): Determines and returns the most common join date from the DataFrame as a string.

4. generate_stat2(data): Computes and formats the average tenure by job from the DataFrame into a table string.

5 .generate_stat3(data): Calculates and presents the standard deviation of salary within defined age groups in a table string format.

6. generate_graph(data): Creates and saves a line graph showing the distribution of salary over age.

7. generate_graph1(data): Generates and saves a scatter plot illustrating the relationship between age and salary

8. generate_graph2(data): Constructs and saves a pie chart displaying the distribution of departments.

9. generate_graph3(data): Produces and saves a box plot showing salary distribution across different departments.

10. generate_graph4(data): Creates and saves a histogram visualizing the distribution of salaries.

11. scatter_view(x, y, z): Constructs a 3D scatter plot with the provided x, y, z axes data.

12. generate_graph5(data): Prepares data for a 3D scatter plot and saves the graph showing age, salary, and join month for the year 2022.

Flask_app.py – The web application for uploading CSV files and displaying data statistics and graphs:

1. **upload_file():** Handles file uploads and processing. It checks for file validity, reads the uploaded CSV file, and generates statistics and graphs using methods from stats_graphs.py.

2. **allowed_file(filename):** Checks if the uploaded file has a valid CSV extension.

3. uploaded_file(filename): Renders the uploaded file page, displaying data statistics, and graphs.

Message.html:

This HTML template is primarily used to display an error message with an automatic redirection feature.

1. **{{ message }}** is a placeholder for dynamically injected error messages.
2. The script initializes a countdown timer (counter) set to 3 seconds.
3. An interval function (setInterval) decreases the counter every second and updates the HTML content of the element with the ID counter.
4. Once the counter reaches zero, the interval is cleared, and the browser redirects to the URL provided in **{{ redirect_url }}**.

Upload.html:

This HTML template is designed for a file upload interface. It includes both styling and form elements for users to upload files.

1. A form (<form>) tag with method="post" and enctype="multipart/form-data" attributes, necessary for file uploading.
2. An input field of type file for users to choose the file they want to upload.
3. A submit button to upload the chosen file.

Uploaded.html:

This HTML template is designed to display the results of a file upload, including data summaries and generated graphs.

1. Uses Bootstrap 4.3.1 CSS for styling, enhancing the layout and responsiveness of the content.
2. A section confirming the successful upload of a file, with the filename dynamically displayed using {{ filename }}.
3. The content areas display data ({{ data }}, {{ data1 }}, etc.) dynamically inserted into the page, marked as safe to render as HTML.
4. Images for the graphs are sourced from static files using Flask's url_for function with filenames (graph_filename, graph_filename1, etc.) dynamically provided.
5. Includes jQuery, Popper.js, and Bootstrap JS for enhanced interactive features and responsive behavior.

How the Flask App Functions:

When users access the application, they are greeted with an "upload.html" template that prompts them to select and upload a CSV file. Once a file is uploaded, it is temporarily saved in the "uploads" directory.

The core of the application lies in its ability to process the uploaded data. This is handled by the "stats_graphs.py" module, which is adept at generating insightful statistics and a variety of graphs from the data. The types of graphs generated—such as scatter plots, histograms, and others—aid in elucidating different aspects and relationships within the dataset.

Post-processing, users are directed to the "uploaded.html" template. This page showcases the initial three rows of the data, giving a glimpse into the dataset. It also displays important statistical measures, like average age and tenure and most common join date, offering users a quantitative understanding of the data.

The six graphs presented are dynamically generated, based on the specific data uploaded by the user. The application assumes the use of a specific format, similar to that of "test.csv", ensuring consistency in data handling and visualization.

"stats_graphs.py" handles the data processing and graph generation, while the Flask application manages the web interface. This separation of concerns allows for more organized code and easier maintenance, enhancing the overall functionality and user experience of the application.