Summative Assessment II - Development of a Web Application.
Chris Northall - Student #2402140

Task 3.

3.1 - Differences Between table.js and filtered_table.js

Both scripts load and display data from a CSV file in an HTML table using D3.js, but they differ in how they process and present the data.

1. Data Selection & Filtering

• **table.js**: Loads the entire CSV dataset without any filtering, displaying all available columns and rows.

• **filtered_table.js**: Selects only five specific columns (id, airline_sentiment, airline_sentiment_confidence, airline, and text) and limits the number of displayed rows to the first 40.

2. Sorting Mechanism

• **table.js**: Implements interactive column sorting. Users can click on column headers to toggle between ascending and descending order with an indicator in the header to make it clear the direction of sorting.

• **filtered_table.js**: Automatically sorts the data **only once** by the airline_sentiment_confidence column in ascending order. No interactive sorting is provided.

3. Handling of Missing Values

• **table.js**: Uses "N/A" as a placeholder for missing values.

• **filtered_table.js**: Also replaces missing values with "N/A", but additionally ensures the airline_sentiment_confidence column is converted to a number, replacing invalid values with 0.

4. Table Structure and Rendering

• **table.js**: Dynamically constructs a table with all dataset columns. It allows user interaction via sorting.

• **filtered_table.js**: Pre-selects specific columns, reformats headers (replacing underscores with spaces and capitalising words), and presents a predefined data subset.

Which Method is Better?

The “better” method depends on the use case:

• If the goal is **in-depth data exploration**, **table.js** is superior because it allows users to view all available data and interactively sort by any column.

• If the goal is **concise and focused presentation**, **filtered_table.js** is more efficient as it pre-filters relevant information and ensures a structured output.

For **general usability and interactivity**, table.js is the better choice as it provides dynamic sorting and supports viewing the full dataset. However, if performance and clarity are the priority, filtered_table.js is more efficient by reducing data load and restricting columns.

3.2 - Description of the Visualisation in creative.js

Why This Visualisation Technique?

The script implements a bubble chart with a force-directed layout to visualise sentiment counts (positive, neutral, and negative) for different airlines. This technique is chosen because-

- **Clear Representation of Volume** – The size of each bubble reflects the number of tweets with a specific sentiment, making it easy to compare the sentiment distribution for each airline.
- **Intuitive Colour Coding** – Different sentiments are colour-coded (green for positive, orange for neutral, and red for negative), providing an instant visual cue.
- **Dynamic Force Simulation** – The D3 force simulation ensures non-overlapping bubbles while maintaining a structured and aesthetically pleasing arrangement.
- **Interactivity** – A dropdown filter allows users to view sentiment distribution for a specific airline, enhancing usability.

Insights Gained from the Visualisation

• **Sentiment Distribution** – It is immediately clear which airlines receive more positive, neutral, or negative feedback based on bubble sizes and colours.

• **Comparative Analysis** – Some airlines may have a much higher proportion of negative sentiment compared to others, revealing possible customer dissatisfaction trends.

• **Tweet Volume Impact** – Airlines with larger bubbles (higher sentiment counts) may have greater engagement or publicity, indicating which brands are most talked about.

• **Data Filtering Effects** – The dropdown filter allows for focused analysis of individual airlines, helping identify specific trends for each brand.

This visualisation is an effective way to summarise and interpret large-scale sentiment data in a clear and engaging format.

3.3 - When designing the two test cases in test_app.py, the following key considerations were considered:

1. Route Status Code Test (test_routes_status_code)

• **Purpose**: Ensures that all critical routes (/, /about, /creative, /advanced, /basic) return a **200 OK** response.

• **Coverage**: Tests multiple predefined routes to confirm they are accessible.

• **Reliability**: Uses a loop to check each route systematically.

• **Error Handling**: If a route fails, an assertion error is raised, making it clear which route has an issue

2. Homepage Content Test (test_homepage_content)

• **Purpose**: Verifies that the homepage contains the expected HTML structure.

• **Considerations**:

• **Correct Rendering**: Checks for the <title> tag in the response to confirm that the page loads correctly.

• **Deliberate Error**: The test contains a deliberate mistake (X Airline Sentiment in the <title>) to verify that the test correctly fails when expectations are not met.

• **Content Validation**: Ensures the returned content is meaningful and includes essential elements.

• **Failure Handling**: Logs failures and raises exceptions to signal issues with the homepage rendering.

General Considerations

• **Comprehensive Coverage**: The tests validate both functional correctness (status codes) and content correctness (expected elements).

• **Structured Output**: Uses PrettyTable to neatly format test results.

• **Exception Handling**: Uses try-except blocks to handle assertion errors gracefully while ensuring test failures are logged.

• **Automated Execution**: The tests can be run with unittest.main(verbosity=2) for detailed output.