1. **Data Accuracy and Integrity Tests**:

Verify that the data displayed on the dashboard matches the source data.

Check for correct data aggregation, summation, averaging, or other calculations.

Ensure that data updates are reflecting accurately in real-time or as per the specified update frequency.

* + SQL:

**SELECT SUM(sales) FROM sales\_data**

**WHERE date BETWEEN '2023-01-01' AND '2023-01-31';**

* + Python: Compare the SQL query result to the total sales displayed on the dashboard for January 2023.

1. **Functionality Tests**:

Test all clickable elements like buttons, dropdown menus, and links to ensure they work as expected.

Validate the functionality of filters and search options – check if they correctly modify the displayed data.

Test any export features to ensure the dashboard data can be downloaded in various formats (e.g., CSV, PDF) without issues.

* + SQL: **SELECT \* FROM customer\_data WHERE status = 'active';**
  + Python: Use a web automation tool like Selenium to activate a filter on the dashboard and verify that it correctly displays data for active customers.

1. **Performance and Load Tests**:

Evaluate the dashboard’s loading time and performance under different conditions, including high data volume and multiple simultaneous users.

Test the response time of various features like data filters and search functions.

* + SQL: **SELECT \* FROM large\_dataset;** (Assuming **large\_dataset** is a very large table)
  + Python: Measure the time taken to load the large dataset on the dashboard and assess if it meets performance criteria.

1. **User Interface and Usability Tests**:

Ensure that the dashboard is user-friendly and intuitive. Test the ease of navigation between different sections.

Check the readability of text, clarity of graphs/charts, and overall visual appeal.

Confirm that the dashboard layout is responsive and displays correctly on various devices and screen sizes.

* + No SQL needed.
  + Python: Use tools like PyAutoGUI to simulate user interactions with the dashboard and check for ease of navigation and responsiveness.

1. **Security and Access Control Tests**:

Verify that access controls are working correctly, and users can only view data they are authorized to see.

Test for any security vulnerabilities, especially if the dashboard is accessible via the internet.

* + SQL: **SELECT \* FROM user\_access WHERE user\_id = 'test\_user';**
  + Python: Verify that the test user can only access the data they are permitted to see on the dashboard, as defined in the **user\_access** table.

1. **Compatibility and Cross-Browser Tests**:

Ensure the dashboard works seamlessly across different browsers (e.g., Chrome, Firefox, Safari).

Test on different operating systems like Windows, macOS, and various mobile platforms.

* + No SQL needed.
  + Python: Use Selenium to open the dashboard in different browsers and verify its functionality and layout.

1. **Error Handling Tests**:

Check how the dashboard handles data errors or missing data.

Validate the system’s behavior under failure scenarios, like server downtime or network issues.

* + SQL: Intentionally query a non-existent table: **SELECT \* FROM invalid\_table;**
  + Python: Check if the dashboard displays a user-friendly error message when the SQL query fails.

1. **Compliance and Regulatory Tests**:

If applicable, ensure that the dashboard complies with relevant industry standards and regulations.

* + SQL: **SELECT COUNT(\*) FROM transactions WHERE transaction\_date < '2023-01-01' AND is\_compliant = 'no';**
  + Python: Verify that the number of non-compliant transactions before 2023 on the dashboard matches the SQL query result.

1. **Feedback and Improvement Tests**:

Gather feedback from actual users on the dashboard’s usability and functionality.

Test any modifications or enhancements based on user feedback.

* + No specific SQL query.
  + Python: Implement a feedback form on the dashboard and use Python to analyze the feedback data for improvements.

1. **Disaster Recovery and Backup Tests**:

Test the dashboard’s backup mechanisms and recovery procedures in case of data loss or system failure.

* + SQL: **BACKUP DATABASE your\_dashboard TO DISK = 'backup\_location';**
  + Python: Simulate a recovery process using Python scripts to restore the database from the backup and verify data integrity.