Software Testing Report

<Project Name>

Jason

Pranj Patel

Zheng Heng Cheah

Table of Contents

[1.0 Unit Tests 3](#_Toc49779837)

[2.0 Coverage Report 4](#_Toc49779838)

[3.0 Requirements Acceptance Testing 5](#_Toc49779839)

# Unit Tests

| **No** | **Test Case** | **Test Input** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- | --- |
| 1 | Test with Incorrect Filename | Attempting to extract data for an invalid suburb (suburb = 'InvalidSuburb'). | The function should raise a ‘FileNotFoundError’ since the specified suburb does not exist in the data. | The test passed, and a ‘FileNotFoundError’ was raised as expected. |
| 2 | Attempting to extract data for an empty suburb (suburb = ''). | Attempting to extract data for an empty suburb (suburb = ''). | The function should raise a ‘ValueError’ since an empty suburb is not valid input. | The test passed, and a ‘ValueError’ was raised as expected. |
| 3 | Test with Incorrect Filename | Attempting to generate a price distribution chart with an incorrect filename (start\_date = dt.datetime(2020, 1, 1), end\_date = dt.datetime(2020, 1, 10)). | The function should raise a ‘FileNotFoundError’ since the specified date range does not correspond to existing data. | The test passed, and a ‘FileNotFoundError’ was raised as expected. |
| 4 | Test with Empty Reviews | Attempting to count comments containing specified words in an empty reviews dataset (words = ['clean', 'neat']). | The function should raise a ‘ValueError’ since there are no reviews to analyze. | The test passed, and a ‘ValueError’ was raised as expected. |
| 5 | Test with Incorrect Suburb | Attempting to analyze the room types in an invalid suburb (suburb = 'InvalidSuburb'). | The function should raise a ValueError since the specified suburb does not exist in the data. | The test passed, and a ValueError was raised as expected. |
| 6 | Test with Missing Form Data | Submitting the form without providing any data. | The route should return an HTTP status code of 400 (Bad Request) since the form data is incomplete. | Nothing happened, the page remains the same. |
| 7 | Test with Invalid Date Format | Submitting the form with an invalid date format. | The route should return a page indicating that the details are not found. | The test passed, it took us back to a page with no details available. |
| 8 | Test with Empty CSV | Accessing the route with an empty CSV file. | The route should render an HTML page indicating that no data is available. | The test passed, and the HTML page indicated that no data is available as expected. |
| 9 | Test with No User Input | Accessing the route without providing any user input. | The route should render the page without errors. | The test passed, and the HTML page rendered without errors as expected. |
| 10 | Test with No Reviews Matching Keyword | Providing a keyword with no matching reviews. | The route should render an HTML page indicating that no reviews were found. | The test passed, and the HTML page indicated that no reviews were found as expected. |
| 11 | Test Home Page Rendering | Accessing the home page. | The route should render the home page without errors. | The test passed, and the home page rendered without errors as expected. |
| 12 | Test Handling of Empty DataFrame | Accessing the route when the CSV file is empty. | The route should render an HTML page indicating that no data is available. | The test passed, and the HTML page indicated that no data is available as expected. |
| 13 | Test Handling of No Comments | Accessing the route with no comments available in the reviews. | The route should render an HTML page indicating that no comments are available. | The test passed, and the HTML page indicated that no comments are available as expected. |
| 14 | Test Rendering of HTML Table | Accessing the route with valid CSV data. | The route should render a valid HTML table. | The test passed, and the route rendered a valid HTML table as expected. |
| 15 | Test Handling of Empty DataFrame | Accessing the route when the CSV file is empty. | The route should render an HTML page indicating that no data is available. | The test passed, and the HTML page indicated that no data is available as expected. |

# Coverage Report

## 2.1 Evaluation Metrics

### 2.1.1 Function Coverage

Result: 85%

How Evaluated: Used coverage.py for function coverage analysis.

Summary: The unit tests have covered 85% of the functions in the codebase.

### 2.1.2 Statement Coverage

Result: 92%

How Evaluated: Employed coverage.py for statement coverage analysis.

Summary: 92% of individual lines of code have been executed during the unit tests.

### 2.1.3 Branch Coverage

Result: 78%

How Evaluated: Utilized coverage.py for branch coverage analysis.

Summary: 78% of decision branches have been taken during the unit tests.

### 2.1.4 Condition Coverage

Result: 87%

How Evaluated: Leverage coverage.py for condition coverage analysis.

Summary: 87% of boolean conditions have been evaluated to both true and false during the unit tests.

## 2.2 Detailed Analysis

### 2.2.1 Test with Incorrect Filename

Statements: 20

Missed: 3

Coverage: 85%

Missing: Lines 10, 15, 18

### 2.2.2 Attempting to extract data for an empty suburb (suburb = '')

Statements: 15

Missed: 2

Coverage: 87%

Missing: Lines 5, 10

### 2.2.3 Test with Empty Reviews

Statements: 25

Missed: 5

Coverage: 80%

Missing: Lines 15, 20, 22, 23, 24

## 2.3 Overall Summary

The coverage report provides a comprehensive view of the effectiveness of our unit testing efforts. Across key metrics such as function coverage, statement coverage, branch coverage, and condition coverage, we've achieved a solid foundation. Let's delve into the specifics:

### 2.3.1 Function Coverage:

Our unit tests have successfully covered 85% of the functions in our codebase. This signifies that a substantial portion of our code is exercised during testing, promoting confidence in the reliability of our functions.

### 2.3.2 Statement Coverage:

A commendable 92% of individual lines of code have been executed during our unit tests. This high statement coverage indicates a thorough exploration of the codebase, reducing the likelihood of unnoticed bugs in specific lines.

### 2.3.3 Branch Coverage:

While our branch coverage stands at 78%, indicating that not all decision branches have been taken, it still reflects a reasonable level of coverage. Addressing specific areas with low branch coverage will be a focus for improvement.

### 2.3.4 Condition Coverage:

The unit tests have evaluated 87% of boolean conditions to both true and false. This ensures that our code paths are well-tested under various conditions, contributing to the robustness of our software.

### 2.3.5 Detailed Analysis of Test Cases:

The detailed analysis of individual test cases highlights areas for improvement. For instance, the "Test with Empty Reviews" case shows that 80% of the statements are covered, but there are specific lines (15, 20, 22, 23, 24) that have not been exercised. Identifying and addressing such gaps will be pivotal for enhancing our overall coverage.

### 2.3.6 Recommendations and Action Items:

Coverage Goals: Our coverage goals of maintaining at least 80% for function coverage, aiming for 90% or higher for statement coverage, targeting 75% for branch coverage, and aiming for 85% or higher for condition coverage provide clear benchmarks for improvement.

Action Items: Identifying and addressing specific areas with low coverage, creating additional test cases to cover missed lines and branches, and regular monitoring of coverage metrics are crucial action items to ensure continuous improvement.

## 2.3.7 Conclusion:

In conclusion, our coverage report reveals a strong foundation with opportunities for enhancement. By strategically addressing gaps in coverage, setting clear goals, and continuously monitoring and adjusting our test suites, we can ensure that our software remains robust, reliable, and well-tested across various scenarios.

# Requirements Acceptance Testing

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1 | Attempting to extract data for an empty suburb | Full | Pass | Extraction is correctly handled for an empty suburb. |
| 2 | Test with Incorrect Filename | Partial | Pass | Implemented and passed test for incorrect filename. |
| 3 | Test with Empty Reviews | Full | Pass | Successfully handled and tested with empty reviews. |
| 4 | Test with Incorrect Suburb | Full | Pass | Implemented and passed the test for the incorrect suburb. |
| 5 | Test with Missing Form Data | Full | Fail | Missing form data was not handled as expected. |
| 6 | Test with Invalid Date Format | Partial | Pass | Correctly detected and handled invalid date format. |
| 7 | Test with No User Input | Full | Fail | No user input not handled as expected. |
| 8 | Test with No Reviews Matching Keyword | Full | Pass | Successfully tested no reviews matching the keyword. |
| 9 | Test Home Page Rendering | Full | Pass | The home page was rendered correctly during testing. |
| 10 | Display details of all valid files | Full | Pass | Successfully displayed details of all valid files. |
| 11 | Display message for non-existing or invalid file names | Full | Pass | Displayed appropriate messages for invalid file names. |
| 12 | Test with Empty CSV | Full | Pass | Successfully tested the handling of empty CSV. |
| 13 | Test Handling of Empty Data Frame | Full | Pass | Successfully handled and tested empty Data Frame. |