Software Testing Report

AJJ BNB

Student Names:

Ashley Pergoliti (s5311775)

Jonas Sajonas (s5284977)

Table of Contents

[1.0 Unit Tests 2](#_Toc147410585)

[2.0 Coverage Report 5](#_Toc147410586)

[3.0 Requirements Acceptance Testing 6](#_Toc147410587)

# Unit Tests

Our software testing method used Python's Unittest framework, focusing on smaller functions inside the code due to their testability. However, unit testing proved difficult because many functions were complex and not class-based, restricting the testing scope. Time constraints made full GUI and database function testing even more difficult. Despite these constraints, we performed critical tests, such as input validation for multiple test cases and date selection algorithms. Minimal GUI testing verified essential operation and call checks, while asset loading methods were tested to identify any missing external components. Additionally, SQLparse was used to test database functions for valid SQL syntax, and file reading functions were evaluated under various scenarios to ensure accurate database formation from CSV files and dependable data fetching.

The report describes our approach to software testing despite constraints such as function size, a lack of class structures, and time constraints. Despite these constraints, we conducted 29 test cases in total, with 22 passing successfully and 7 resulting in expected failures. Refactoring the code into class-based, modular functions would greatly benefit future testing efforts, allowing for a more comprehensive and efficient testing process.

The table below contains a summary of the report, followed by a screenshot of the actual report, which is depicted in the figure below.

**Table 1: Summary of Test Cases**

| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| **1.0** | **Input Functions** |  |  |
| 1.1 | test\_clean\_user\_input\_turn\_string\_into\_list | PASSED | PASSED |
| 1.2 | test\_clean\_user\_input\_test\_strip | PASSED | PASSED |
| 1.3 | test\_clean\_user\_input\_integer\_input | ValueError | Display ValueError PASSED = Error: Input not string |
| 1.4 | test\_clean\_user\_input\_none\_input | ValueError | Display ValueError PASSED = Error: Input not string |
| 1.5 | test\_clean\_user\_input\_empty\_string | ValueError | Display ValueError PASSED = Error: Input not string |
| **2.0** | **Date Functions** |  |  |
| 2.1 | test\_select\_date\_return\_something | PASSED | PASSED |
| 2.2 | test\_select\_date\_return\_tuple | PASSED | PASSED |
| 2.3 | test\_select\_date\_return\_two\_results | PASSED | PASSED |
| 2.4 | test\_select\_date\_start\_get\_date\_called | PASSED | PASSED |
| 2.5 | test\_select\_date\_end\_get\_date\_called | PASSED | PASSED |
| 2.6 | test\_select\_date\_future\_dates | ValueError | Display ValueError  PASSED |
| 2.7 | test\_select\_date\_dates\_not\_in\_database | ValueError | Display ValueError  PASSED |
| **3.0** | **Tkinter Functions** |  |  |
| 3.1 | test\_make\_window | PASSED | PASSED |
| 3.2 | test\_close | PASSED | PASSED |
| 3.3 | test\_close\_not\_called | PASSED | PASSED |
| 3.4 | test\_close\_called\_multiple\_times | PASSED | PASSED |
| 3.5 | test\_throttle\_click | PASSED | PASSED |
| **4.0** | **Asset Loading Functions** |  |  |
| 4.1 | test\_load\_images | PASSED | PASSED |
| 4.2 | test\_relative\_to\_assets | PASSED | PASSED |
| 4.3 | test\_relative\_to\_assets\_expected\_failure | ExpectedFailure  WindowsPath('/wrong/expected/path/some/file/path.txt') | ExpectedFailure  WindowsPath('/wrong/expected/path/some/file/path.txt')  PASSED |
| **5.0** | **Database Functions** |  |  |
| 5.1 | test\_sql\_syntax | PASSED | PASSED |
| 5.2 | test\_sql\_syntax\_failure | AssertionError | AssertionError  PASSED |
| 5.3 | test\_successful\_file\_read | PASSED | PASSED |
| 5.4 | test\_file\_not\_found | PASSED | PASSED |
| 5.6 | test\_empty\_or\_corrupted\_file | ExpectedFailure  File is empty or corrupted | ExpectedFailure  File is empty or corrupted |
| 5.7 | test\_connect\_to\_db | ExpectedFailure  Failed to connect to the database: \_\_enter\_\_ | ExpectedFailure  Failed to connect to the database: \_\_enter\_\_ |
| 5.8 | test\_run\_create\_db | PASSED | PASSED |
| 5.9 | test\_run\_create\_db\_failure | ExpectedFailure  Assertion Error | ExpectedFailure  Assertion Error |
| 5.a | test\_get\_price\_chart\_data | PASSED | PASSED |

A screenshot of a computer program

Description automatically generated

Figure 1: Test Case Report

# Coverage Report

During the unit testing phase, functions were separated into 'mod\_utils' and tested using 'test\_modutils.py', with constants kept in a separate 'mod\_constants' file. This method ensured thorough and dependable testing of the dependability of each function.

**Function Coverage**

We tested about 12 commonly used functions in our software and ran 29 unit tests for each of the 12 functions. This is done to ensure that a single function can handle multiple scenarios, including failure tests.

**Statement Coverage**

The coverage report found 88 statements in the mod\_utils.py file and 197 in the test\_modutils.py file. The coverage report also included the dependency file mod\_constants.py, which contains 16 statements for a total of 301 statements executed.

**Branch coverage**

The report showed that we have 22 branches for the file being tested and 4 for the testing file, for a total of 26 if-else statements covered.

There were 19 missing statements from both files, and partial testing yielded only four from the file being tested. Because we only had one boolean, we didn't have condition coverage. Regarding coverage percentages, we tested the mod\_utils file for 88% coverage and the testing file for 95% coverage, for a total average of 93% coverage.

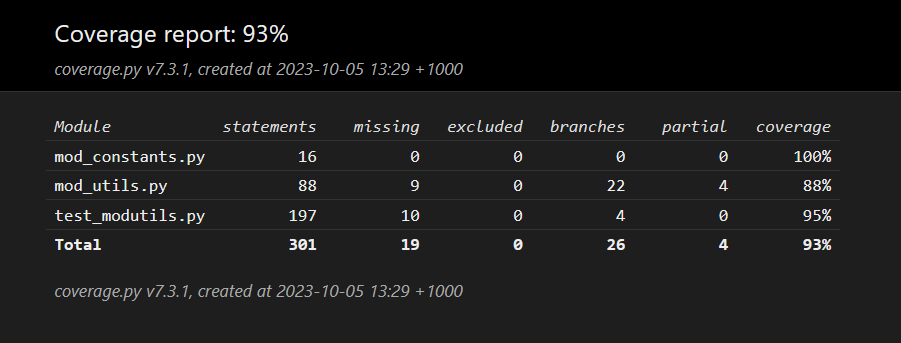


Figure 2: Coverage Report

# Requirements Acceptance Testing

Requirements Acceptance Testing involves comparing defined software requirements to actual capabilities, categorising conditions as fully met, partially met, or unmet, and indicating whether they passed or failed.

Table 2: Acceptance Testing

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1 | The system shall be able to interpret Graphical User Interface scripts. | Full | Pass | The application can make a GUI Interface. |
| 2 | The system shall be able to display the Graphical User Interface effectively. | Full | Pass | The application can be interacted with by the User without errors. |
| 3 | The system shall be able to read and process multiple Excel files. | Full | Pass | The application allows for simultaneously reading the CSV files. |
| 4 | The system shall be able to display different pages within the system. | Full | Pass | The application has multiple Interfaces. |
| 5 | The system shall be able to generate records, lists, and similar tabular data. | Full | Pass | The application can generate data records and display them to the User. |
| 6 | The system shall be able to produce charts and graphs. | Full | Pass | The application can produce Charts and Graphs. |
| 7 | The system shall be able to support keyword and category searches within the data files. | Full | Pass | The application can process keyword searches and categorical searches. |
| 8 | The system shall be able to get consistent results based on user actions. | Full | Pass | The application produces the same results every time it is interacted with. |
| 9 | The system shall be able to close gracefully without errors. | Full | Pass | The application can be closed without crashing. |