Software Testing Report

<Project Name>

Student Names Zhanrui Liao id:s5290972

Student Names Chien Hunag id:s5333270

Table of Contents

[1.0 Unit Tests 3](#_Toc49779837)

[2.0 Coverage Report 4](#_Toc49779838)

[3.0 Requirements Acceptance Testing 5](#_Toc49779839)

# Unit Tests

Delete the RED text and replace with your own

(In this table you fill out details about what unit tests you have done using the unittest module)

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| 1 | * This test function is designed to verify the functionality of the **get\_surburb\_list** method. * suburb1 = "Artarmon"   suburb2 = "Ashfield" | * suburb1（"Artarmon"）和suburb2 the suburb1 ("Artarmon") and suburb2 ("Ashfield") should both exist in the suburb\_list returned by the database. * The suburb\_list should be a dictionary object. | * If the database indeed contains the suburbs "Artarmon" and "Ashfield" * And the get\_surburb\_list function returns a dictionary object as expected, then the test should pass. |
| 2 | * This test function is designed to verify the functionality of the **get\_surburb\_list** method. with the invalid suburb list. * suburb1 = "Artarmon"   suburb2 = "Ashfield"  invalid\_suburb = "InvalidSuburb" | * suburb1 ("Artarmon") and suburb2 ("Ashfield") should both exist in the suburb\_list returned by the database. * invalid\_suburb ("InvalidSuburb") should not be present in the suburb\_list. | * Ensure that the invalid suburb ("InvalidSuburb") is indeed not in the database. * And the get\_surburb\_list function returns a dictionary object as expected. |
| 3 | * This test function is designed to verify the functionality of the **get\_year\_range** method. * min\_year = "2017"   max\_year = "2020" | * actual\_min\_year and actual\_max\_year should both be of string type. * actual\_min\_year should be less than or equal to actual\_max\_year. | * Ensure that the data in the database matches the test parameters ("2017" and "2020"). * And the get\_year\_range function returns two strings as expected. 。 |
| 4 | * This test function is intended to verify the functionality of the query\_location\_data method when provided with the parameter **total\_days** * total\_days = 7   expected\_result\_length = 216 | * Test **total\_days** * The result should not be None. * The result should be a list object. * The length of the result should be equal to expected\_result\_length (216). | * Ensure that the database contains data for the suburb "Alexandria." * And the query\_location\_data function, when called with the specified parameters, returns a list with a length matching expected\_result\_length |
| 5 | * This test function is intended to verify the functionality of the query\_location\_data method when provided with the parameter **suburb\_list.** * suburb\_list = ["Alexandria"]   expected\_result\_length = 216 | * Test **suburb\_list.** * The result should not be None. * The result should be a list object. * The length of the result should be equal to expected\_result\_length (216, 0, 1026). | * Ensure that the database contains data for these three suburbs. * And the query\_location\_data function, when called with the specified parameters, returns a list with a length matching expected\_result\_length. |
| 6 | * This test function is intended to verify the functionality of the query\_location\_data method when provided with the parameter **suburb\_list/date.** * suburb\_list = ["Alexandria"]   date = "2019-01-01"  expected\_result\_length = 216 | * Test **suburb\_list/date** * The result should not be None. * The result should be a list object. | * Ensure that the database contains data for these two suburbs. * And the query\_location\_data function, when called with the specified parameters, returns a list with a length matching expected\_result\_length. |
| 7 | * This test function is intended to verify the functionality of the query\_location\_data method when provided with the parameter **suburb\_list/date/total\_days** * suburb\_list = ["Alexandria"]   date = "2019-01-01"  expected\_result\_length = 216  total\_days = 7 | * test **suburb\_list /date/total\_days** * The result should not be None. * The result should be a list object. | * Ensure that the database contains data for these two suburbs. * And the query\_location\_data function, when called with the specified parameters, returns a list with a length matching expected\_result\_length. |
| 8 | * This test function is designed to verify the functionality of the query\_location\_data method when provided with specific parameters that represent an invalid or edge case scenario. * suburb\_list = [""]   date = "2019-08-01"  total\_days = 0  expected\_result = [] | * The result should be equal to the expected\_result, which is an empty list ([]). | * Verify that when the input suburb list is empty and total\_days is 0, the query\_location\_data function correctly returns an empty list as the result. |
| 9 | * This test function is created to verify the functionality of the query\_price\_distribution\_data method when provided with a **date** parameter. * date = "2019-01-01" | * Test **date.** * The result should not be None. * The result should be a Matplotlib figure object (plt.Figure). * The result should contain one or more axes. | * In this specific test case, the query\_price\_distribution\_data function should return a Matplotlib figure object. * which should contain at least one axis for displaying the data chart of price distribution. |
| 10 | * This test function is created to verify the functionality of the query\_price\_distribution\_data method when provided with two parameters. * date = "2019-01-01"   total\_days = 1 | * Test **date/total\_days** * The result should not be None. * The result should be a Matplotlib figure object (plt.Figure).The result should contain one or more axes. | * The test case is used to verify if the query\_price\_distribution\_data function can correctly generate a price distribution chart. * return a valid Matplotlib figure object when given input date and total days. |
| 11 | Test the database initialisation for success | Successful database initialisation is expected. | All data is read correctly, if the database does not exist it will read data from csv to the database |
| 12 | Test whether the input of different keywords under the search keyword function is successful in returning house information and related comments. | The keyword search function is successful for keywords. | This feature successfully feeds eligible listings and reviews matching the keywords based on the keywords entered by the user. |
| 13 | Verify that a single different date is entered into the query for the correct house information. | All of the housing information that met the requirements was found from the database and the results were returned. | Complete got the correct house information from the database by the date entered and returned the desired result. |
| 14 | Tests whether a change in the input value for dwell time returns the correct listing information. | By obtaining the user's dwell time, the time period of the user's demand is correctly calculated, and then the appropriate listings are identified and the correct results are returned. | Successfully got to calculate the time period requested by the customer and used this data to get the correct listing information and return the correct data. |
| 15 | The test targets user searches for cleanliness. | All listing information about cleanliness and all reviews about cleanliness can be found and successfully returned back data. | Get all the information about the cleanliness of the house and reviews that match the cleanliness. |
| 16 | Test the keyword search function to search for listing information and reviews by keyword and check-in date. | This feature finds information about homes from a database that matches the time description and matches homes and related reviews with keywords. | Successfully get the matching listings from the database, along with information about the reviews that were matched. And successfully returned the relevant data. |
| 17 | Test different inputs for setting the initial date and stay to get the listings. | Ability to correctly obtain the initial date, and identify the time period in which the customer chose the house, obtain the matching listings, and successfully return the data. | Successfully calculated the time period the customer chose to stay and found the correct listing in the database and returned the correct result. |
| 18 | Test a search function that searches for housing information by dwell time and keywords. | Calculate the time period and find out the listings and combine them with keywords to further find out the matching listings and reviews. | Identified the right listings with the right time period and keywords and returned the data accurately to the client. |
| 19 | Tested by start date, total time spent, keyword overlay to search for listings and review functionality. | Match the correct time period, and the data containing keywords for the listings and reviews can be returned. | The returned listing information matched the customer's requirements for the time period and also successfully found reviews from the database that matched the keyword requirements. |
| 20 | Tested the database linking function. | The database is successfully linked and the attributes and data in the database are complete and intact, and the data in the database can be successfully called. | The database connection is normal, and the data in the database is not empty, you can connect and then call the data. |

# Coverage Report

**A description of the coverage of your unit tests, including how you evaluated coverage (function, statement, branch, condition)**

**Description of the coverage of our unit tests:**

**Our unit tests primarily cover the following key functionalities and scenarios:**

1. Test function is the get\_surburb\_list method.
2. Test function is f the get\_surburb\_list method. with the invalid suburb list.
3. Test function is the get\_year\_range method.
4. Test function is query\_location\_datamethod when provided with the parameter **total\_days**
5. test function is query\_location\_data method when provided with the parameter **suburb\_list.**
6. Test function is query\_location\_data method when provided with the parameter **suburb\_list/date.**
7. Test function is the query\_location\_data method when provided with the parameter **suburb\_list/date/total\_days**
8. Test function is the query\_location\_data method when provided with specific parameters that represent an invalid or edge case scenario.
9. Test function is created the query\_price\_distribution\_data method when provided with the parameter **date**.
10. Test function is the query\_price\_distribution\_data method when provided with two parameters **date/total\_days.**
11. Successful initialization of the database.
12. Functionality to search for property information and related comments based on different keywords.
13. Querying property information for a specific date.
14. Returning the correct property information based on the input value for dwell time.
15. Searches targeting user preferences for cleanliness.
16. Functionality to search for property information and comments combining keywords and check-in dates.
17. Functionality to obtain property listings by setting an initial date and duration of stay.
18. Searching for property information combining dwell time and keywords.
19. Functionality to search for listings and reviews by overlaying start date, total dwell time, and keywords.
20. Testing the database linking function.

To evaluate the coverage of our tests, we adopted the following assessment methods:

* **Function Coverage**: Ensuring every function was tested.
* **Statement Coverage**: Ensuring every statement in the code was executed at least once.
* **Branch Coverage**: Ensuring every branch in the code (e.g., if-else statements) was executed.
* **Condition Coverage**: Ensuring each logical condition was evaluated at least once for both true and false.

Through the above methods, we ensured comprehensive coverage of the key functionalities and scenarios in our unit tests, thereby guaranteeing the quality and stability of the software.

# Requirements Acceptance Testing

(You will need to fill out the column on the left with the requirements listed in software design documents and the columns on the right with the results of your own testing)

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1 | Accept multiple file names as arguments from the command line |  |  |  |
| 2 | Display the details of all valid files |  |  |  |
| 3 | Display an appropriate message if a file does not exist or if a file name is invalid |  |  |  |
| 4 | Display a message if an argument is a directory instead of a file |  |  |  |
| 5 | File name can be a simple file name or include the full path of the file with one or more levels |  |  |  |
| 6 | file names must start with an alphabetical character |  |  |  |
| 7 | Valid file name extensions must be 3 or 4 alphabetical characters preceded by a dot) |  |  |  |
| 8 | Directory/level names must start with an alphabetical character to be considered valid |  |  |  |
| 9 | The program should be able to accept as many levels for each file name as the user wants to input. This is limited only by the number of levels allowed in Windows (approximately 120) |  |  |  |