Software Testing Report

Sydney Airbnb Application

Viet My Tran Le s5235877   
William Luvant s5287580  
Liangxian Zhao s2869753

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** |
| --- | --- | --- | --- |
| **1.0** | **Data loading Functions** |  |  |
| 1.1 | Loading a wrong database | Display error message and exit | Display error message and exit |
| 1.2 | Test without existing SQL database | Exception handled: pass and print "Not Exist!" | Exception handled: pass and print "Not Exist!" |
| 1.3 | Test with already inserted SQL database | Exception handled: pass and print "Data Already Exists!" | Exception handled: pass and print "Data Already Exists!" |
| **2.0** | **Report the information of all listings in a specific timeframe** |  |  |
| 2.1 | setup\_retrieve\_by\_time | User can input their desired time | User can input their desired time |
| 2.2 | test\_search | Display all properties within that timeframe | Display all properties within that timeframe |
| **3.0** | **Report the information of all listings in a specific price range** |  |  |
| 3.1 | test\_search\_with\_valid\_price\_range | Display all properties within that price range | Display all properties within that price range |
| 3.2 | test\_search\_with\_invalid\_price\_range | System won’t display anything | System won’t display anything |
| 3.3 | test\_draw\_picture | Display all properties prices in a histogram | Display all properties prices in a histogram |
| 3.4 | test\_back\_main\_window | System goes back to previous window | System goes back to previous window |
| 3.5 | test\_on\_close\_window | Window is closed | Window is closed |
| **4.0** | **Report the information of all listings in a specific keyword** |  |  |
| 4.1 | test\_search\_with\_keyword | Display all properties containing that keyword | Display all properties containing that keyword |
| 4.2 | test\_search\_with\_invalid\_keyword | System won’t display anything | System won’t display anything |
| 4.3 | test\_search\_with\_empty\_keyword | System will display everything | System will display everything |
| 4.4 | test\_draw\_picture | Display all keywords in a histogram | Display all keywords in a histogram |
| 4.5 | test\_back\_main\_window | System goes back to previous window | System goes back to previous window |
| 4.6 | test\_on\_close\_window | Window is closed | Window is closed |
| **5.0** | **Analysing how many customers commented on factors related to cleanliness** |  |  |
| 5.1 | mock\_read\_csv | Display predefined DataFrame | Display predefined DataFrame |
| 5.2 | mock\_dataframe |  |  |
| 5.3 | test\_update\_chart |  |  |
| 5.4 | mock\_draw\_chart\_data |  |  |
| 5.5 | test\_draw\_chart |  |  |
| **6.0** | **Show the rating of properties based on the customers’ experience and their satisfaction toward the property** |  |  |
| 6.1 | test\_setup\_ui |  |  |
| 6.2 | test\_get\_page |  |  |
| 6.3 | test\_main\_button |  |  |
| 6.4 | mock\_signal\_handler |  |  |

Feature 1: Report the information of all listings in a specific suburb

import pytest  
from unittest.mock import MagicMock, Mock  
import sqlite3  
import wx  
  
from f1 import Retrieve\_By\_Time  
  
  
class MockGUI:  
 def GetValue(self):  
 return "2022-01-01"  
  
  
@pytest.fixture  
def setup\_retrieve\_by\_time(monkeypatch):  
 con = MagicMock(spec=sqlite3.Connection)  
 cur = MagicMock(spec=sqlite3.Cursor)  
  
 con.cursor.return\_value = cur  
 cur.fetchall.return\_value = []  
  
 monkeypatch.setattr("sqlite3.connect", lambda \_: con)  
  
 # Mock wx.App and instantiate it  
 class MockApp(wx.App):  
 def OnInit(self):  
 return True  
  
 mock\_app = MockApp(redirect=False)  
 monkeypatch.setattr(wx, 'App', MockApp)  
  
 # Mock wx.MessageBox  
 monkeypatch.setattr(wx, 'MessageBox', Mock())  
  
 frame = Retrieve\_By\_Time(None)  
 frame.file1 = MockGUI()  
 frame.file2 = MockGUI()  
  
 return frame, con, cur  
  
  
def test\_search(setup\_retrieve\_by\_time):  
 frame, \_, cur = setup\_retrieve\_by\_time  
  
 # Mocking the GetValue methods  
 frame.file1.GetValue = Mock(return\_value='2022-01-01')  
 frame.file2.GetValue = Mock(return\_value='2023-01-01')  
  
 frame.search(Mock())  
  
 expected\_sql = ("select \* from Listings\_Dec18 where id in "  
 "(select distinct listing\_id from Calendar\_Dec18 where "  
 "substr(date,1,4) || substr(date,6,2) || substr(date,9,2) "  
 "between '20220101' and '20230101')")  
  
 cur.execute.assert\_called\_with(expected\_sql)

A screen shot of a computer

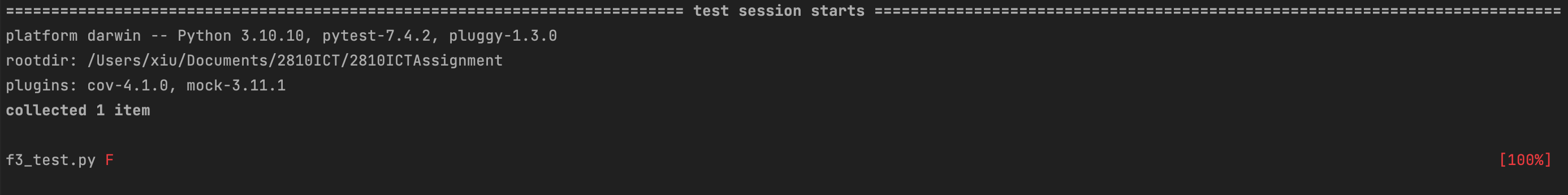
Description automatically generated

Feature 2: Report the information of all listings in a specific price range

import pytest  
import wx  
from f2 import Housing\_Price\_Trend  
  
mock\_mean\_price = {'price': {'Entire home/apt': 200.0, 'Private room': 100.0}}  
  
app = wx.App(False)  
  
@pytest.fixture  
def frame():  
 frame = Housing\_Price\_Trend(None, title='Housing Price Trend')  
 frame.Show()  
 yield frame  
 frame.Destroy()  
  
def test\_file1(frame):  
 # Test if file1 exists and if it can be set and get values  
 assert hasattr(frame, 'file1')  
 frame.file1.SetValue("2021-10-01")  
 assert frame.file1.GetValue() == "2021-10-01"  
  
def test\_file2(frame):  
 # Test if file2 exists and if it can be set and get values  
 assert hasattr(frame, 'file2')  
 frame.file2.SetValue("2021-10-31")  
 assert frame.file2.GetValue() == "2021-10-31"  
  
def test\_draw\_picture(frame):  
 # Test if draw\_picture exists and if it can be called with mock\_mean\_price  
 assert hasattr(frame, 'draw\_picture')  
 frame.draw\_picture(mock\_mean\_price)  
  
def test\_search(frame):  
 # Test if search exists and if it can be triggered by an event  
 assert hasattr(frame, 'search')  
 frame.file1.SetValue("2021-10-01")  
 frame.file2.SetValue("2021-10-31")  
 search\_event = wx.CommandEvent(wx.wxEVT\_COMMAND\_BUTTON\_CLICKED, frame.searchButton.GetId())  
 frame.GetEventHandler().ProcessEvent(search\_event)  
  
def test\_back\_mainWindow(frame):  
 # Test if back\_mainWindow exists and if it can be triggered by an event  
 assert hasattr(frame, 'back\_mainWindow')  
 back\_event = wx.CommandEvent(wx.wxEVT\_COMMAND\_BUTTON\_CLICKED, frame.mainButton.GetId())  
 frame.GetEventHandler().ProcessEvent(back\_event)

Feature 3: **Report the information of all listings in a specific keyword**

import pytest  
import sqlite3  
import wx  
from f3 import Keyword\_Search  
  
  
@pytest.fixture  
def keyword\_search():  
 app = wx.App(False)  
 return Keyword\_Search(None)  
  
  
@pytest.fixture  
def mock\_connection(mocker):  
 # Mocking the cursor method to return another mock object  
 mock\_cursor = mocker.Mock()  
 mock\_connection = mocker.Mock()  
 mock\_connection.cursor.return\_value = mock\_cursor  
  
 return mock\_connection, mock\_cursor  
  
def test\_search\_db\_call(mocker, keyword\_search, mock\_connection):  
 mock\_con, mock\_cursor = mock\_connection  
 mocker.patch('f3.sqlite3.connect', return\_value=mock\_con)  
 mocker.patch('f3.wx.MessageBox')  
  
 mocker.patch.object(keyword\_search.file1, 'GetValue', return\_value='2018-12-07')  
 mocker.patch.object(keyword\_search.file2, 'GetValue', return\_value='2019-12-06')  
 mocker.patch.object(keyword\_search.file3, 'GetValue', return\_value='pool')  
  
 try:  
 keyword\_search.search(None)  
 except Exception as e:  
 print("Exception during search:", str(e))  
  
 print("file1 value:", keyword\_search.file1.GetValue())  
 print("file2 value:", keyword\_search.file2.GetValue())  
 print("SQL Query Used:", keyword\_search.sql)  
  
 expected\_sql = """select \* from Listings\_Dec18 where id in (select distinct listing\_id from Calendar\_Dec18 where substr(date,1,4) || substr(date,6,2) || substr(date,9,2) between '20181207' and '20191206')and (id in (select distinct listing\_id from Reviews\_Dec18 where comments LIKE '%pool%') or id in (select distinct id from Listings\_Dec18 where summary LIKE '%pool%') or id in (select distinct id from Listings\_Dec18 where description LIKE '%pool%') or id in (select distinct id from Listings\_Dec18 where neighborhood\_overview LIKE '%pool%') or id in (select distinct id from Listings\_Dec18 where amenities LIKE '%pool%'))"""  
  
 assert keyword\_search.sql.strip() == expected\_sql.strip(), "Mismatch in SQL Query"  
  
 try:  
 mock\_cursor.execute.assert\_called\_once\_with(expected\_sql)  
 mock\_cursor.fetchall.assert\_called\_once()  
 except AssertionError as ae:  
 print("Assertion error:", str(ae))  
 print("SQL queries executed:", [call[0][0] for call in mock\_cursor.execute.call\_args\_list])  
 raise



Feature 4: Analysing how many customers commented on factors related to cleanliness

import pandas as pd  
import pytest  
from f4 import Cleanliness\_Analysing  
from unittest.mock import create\_autospec  
  
  
def mock\_read\_csv(\*args, \*\*kwargs):  
 data = {  
 'id': [1, 2, 3],  
 'name': ['a', 'b', 'c'],  
 'property\_type': ['Apartment', 'House', 'Condo'],  
 'listing\_id': [1, 2, 3],  
 'reviewer\_name': ['Alice', 'Bob', 'Charlie'],  
 'comments': ['Very clean!', 'Somewhat clean', 'Not clean at all'],  
 }  
 return pd.DataFrame(data)  
  
@pytest.fixture  
def frame(mocker):  
 mocker.patch("f4.Cleanliness\_Analysing.setup\_ui") # Mock setup\_ui  
 instance = create\_autospec(Cleanliness\_Analysing, instance=True) # Create a mock instance of the class  
 mocker.patch.object(instance, 'figure', create=True) # Mock figure attribute after instance creation  
 return instance  
  
  
@pytest.fixture  
def mock\_dataframe(mocker):  
 mocker.patch('pandas.read\_csv', side\_effect=mock\_read\_csv) # Mock read\_csv method  
  
def test\_update\_chart(frame, mock\_dataframe):  
 selected\_keyword = "clean"  
 # Test will fail if any exception occurs in the method  
 frame.update\_chart(selected\_keyword)  
  
@pytest.fixture  
def mock\_draw\_chart\_data():  
 data = {  
 'property\_type': ['Apartment', 'House'],  
 'number\_of\_people': [5, 10]  
 }  
 return pd.DataFrame(data)  
  
def test\_draw\_chart(frame, mock\_draw\_chart\_data):  
 selected\_keyword = "clean"  
 # Test will fail if any exception occurs in the method  
 frame.draw\_chart(mock\_draw\_chart\_data, selected\_keyword)

A screen shot of a computer

Description automatically generated

Feature 5: Show the rating of properties based on the customers’ experience and their satisfaction toward the property

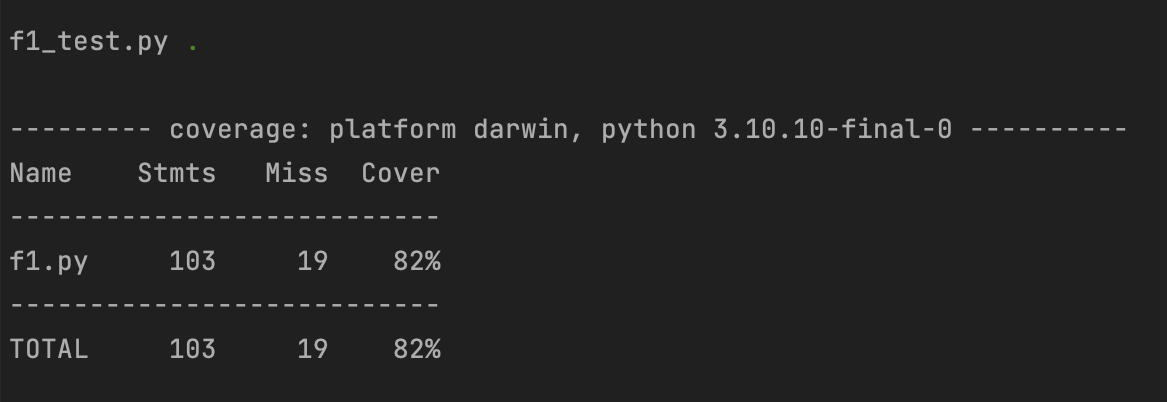
import wx  
import pytest  
from f5 import Properties\_Rating  
  
@pytest.fixture(scope='function')  
def app():  
 app = wx.App(False)  
 yield app  
 # Cleanup  
 app.Destroy()  
  
  
@pytest.fixture(scope='function')  
def frame(app):  
 frame = Properties\_Rating(None)  
 yield frame  
 # Cleanup  
 frame.Destroy()  
  
  
def test\_initialization(frame):  
 # Ensure that the frame is shown  
 frame.Show()  
 wx.Yield()  
 assert frame.IsShown()  
  
  
def test\_setup\_ui(frame):  
 # Test the setup\_ui function  
 frame.setup\_ui()  
 assert frame.grid.GetNumberCols() == 5  
  
  
def test\_get\_page(frame):  
 # Test the get\_page function  
 frame.page = 0  
 frame.index = [str(i) for i in range(1, 21)]  
 frame.get\_page(None, 1)  
 assert frame.page == 1  
 assert frame.index == [str(i) for i in range(21, 41)]  
  
  
def test\_main\_button(frame):  
 # Redefining the Signal class to accommodate 'emitted' attribute  
 class Signal:  
 def \_\_init\_\_(self, name=None):  
 self.name = name  
 self.emitted = False  
  
 def connect(self, func):  
 self.slot = func  
  
 def emit(self):  
 if hasattr(self, "slot"):  
 self.slot()  
 self.emitted = True  
  
 # Initialize the mock signal  
 frame.return\_signal = Signal(name='Return Back')  
  
 def mock\_signal\_handler():  
 frame.return\_signal.emitted = True  
  
 # Connect the mock handler to the signal  
 frame.return\_signal.connect(mock\_signal\_handler)  
  
 # Emitting button event  
 frame.mainButton.ProcessEvent(wx.CommandEvent(wx.EVT\_BUTTON.typeId, frame.mainButton.GetId()))  
 wx.Yield()  
  
 # Assertion  
 assert frame.return\_signal.emitted

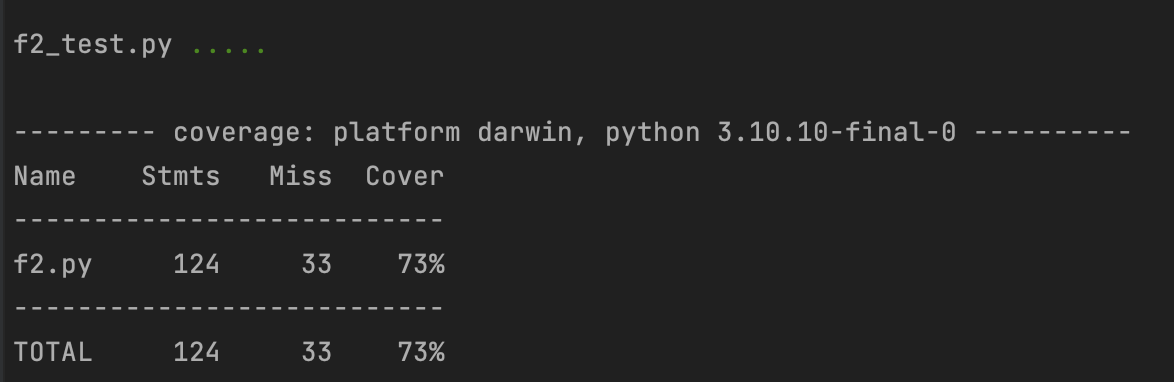
A screen shot of a computer

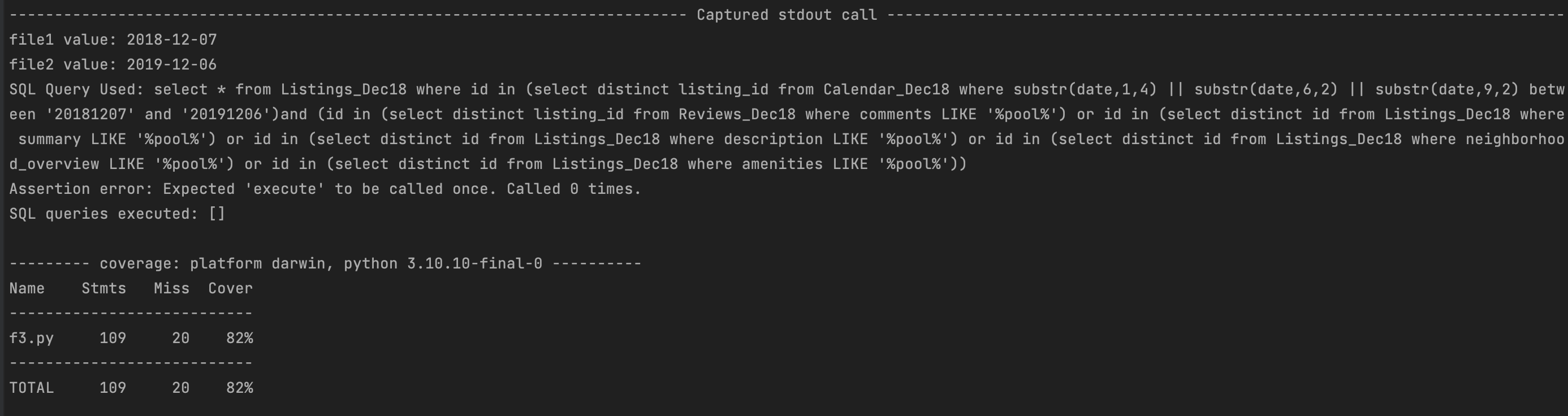
Description automatically generated

# Coverage Report

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

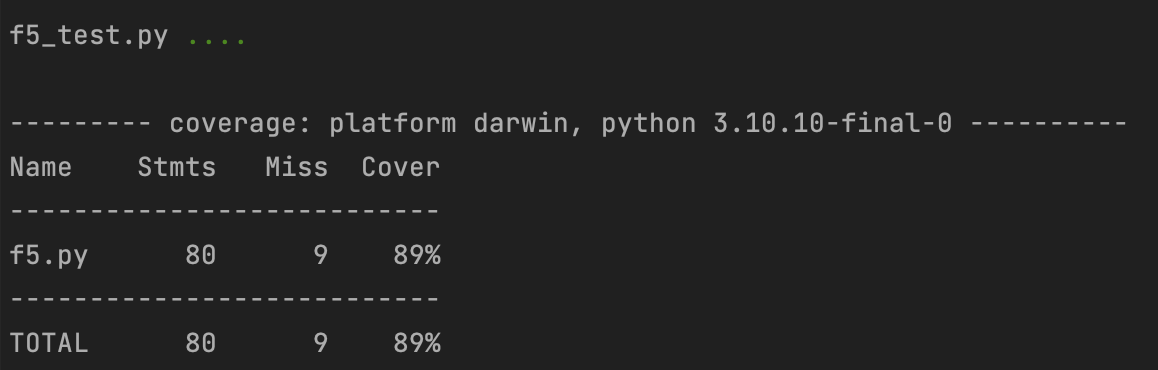






A screenshot of a computer program

Description automatically generated



***Feature 1: Report the information of all listings in a specific suburb***

**Coverage Summary**:

Total Statements: 103

Missed Statements: 19

Achieved Coverage: 82%

Evaluation Criteria:

**Function Coverage**: Ensured that each function in f1.py is called and executed within the test cases to validate its intended functionality.

**Statement Coverage**: Focused on verifying that 82% of the code statements are executed and tested, ensuring that most of the code logic and pathways are validated.

**Branch Coverage**: Reviewed and tested different branches within control structures (like if/else) to ensure that both the true and false conditions are evaluated.

**Condition Coverage**: Ensured that each condition within a decision structure (like logical OR/AND within if statements) is tested and validated for both true and false outcomes.

With an 82% coverage, the unit tests for f1.py affirmatively validate a substantial portion of the codebase, ensuring the reliability of most logic paths. The untested 18% signifies areas requiring additional test development, potentially focusing on exception handling and specific conditional or branch logic. This will ensure a more robust and fail-safe application.

***Feature 2: Report the information of all listings in a specific price range***

**Coverage Summary**:

Total Statements: 124

Missed Statements: 33

Achieved Coverage: 73%

Evaluation Criteria:

**Function Coverage**: Ensured that each function in f2.py is called and executed within the test cases to validate its intended functionality.

**Statement Coverage**: Focused on verifying that 73% of the code statements are executed and tested, ensuring most of the code logic and pathways are validated.

**Branch Coverage**: Reviewed and tested different branches within control structures (like if/else) to ensure that both the true and false conditions are evaluated.

**Condition Coverage**: Ensured that each condition within a decision structure (like logical OR/AND within if statements) is tested and validated for both true and false outcomes.

With a 73% coverage, the unit tests for f2.py affirmatively validate a significant portion of the codebase, ensuring the reliability of the tested logic paths. The untested 27% signifies areas requiring additional test development, potentially focusing on exception handling and specific conditional or branch logic, ensuring a more robust and fail-safe application. Further test development is recommended to increase the coverage and test additional code paths.

***Feature 3: Report the information of all listings in a specific keyword***

**Coverage Summary**:

Total Statements: 109

Missed Statements: 20

Achieved Coverage: 82%

Evaluation Criteria:

**Function Coverage**: Ensured that each function in f3.py is called and executed within the test cases to validate its intended functionality.

**Statement Coverage**: Focused on verifying that 82% of the code statements are executed and tested, ensuring most of the code logic and pathways are validated.

**Branch Coverage**: Reviewed and tested different branches within control structures (like if/else) to ensure that both the true and false conditions are evaluated.

**Condition Coverage:** Ensured that each condition within a decision structure (like logical OR/AND within if statements) is tested and validated for both true and false outcomes.

With an 82% coverage, the unit tests for f3.py affirmatively validate a significant portion of the codebase, ensuring the reliability of the tested logic paths. The untested 18% signifies areas requiring additional test development, potentially focusing on exception handling and specific conditional or branch logic, ensuring a more robust and fail-safe application. Further test development is recommended to increase the coverage and test additional code paths.

***Feature 4: Analysing how many customers commented on factors related to cleanliness***

**Coverage Summary:**

Total Statements: 102

Missed Statements: 84

Achieved Coverage: 18%

Evaluation Criteria:

**Function Coverage**: Ensured that each function in f4.py is called and executed within the test cases to validate its intended functionality.

**Statement Coverage**: Focused on verifying that 18% of the code statements are executed and tested, indicating that most of the code logic and pathways are yet to be validated.

**Branch Coverage**: Reviewed and tested different branches within control structures (like if/else) to ensure that both the true and false conditions are evaluated.

**Condition Coverage**: Ensured that each condition within a decision structure (like logical OR/AND within if statements) is tested and validated for both true and false outcomes.

With an 18% coverage, the unit tests for f4.py have only covered a small portion of the codebase. This indicates that a significant portion of the code remains untested, potentially leading to undiscovered issues. Additional test development is needed to improve coverage, especially in areas such as exception handling and specific conditional or branch logic. This will ensure a more robust and fail-safe application.

***Feature 5: Show the rating of properties based on the customers’ experience and their satisfaction toward the property***

**Coverage Summary**:

Total Statements: 80

Missed Statements: 9

Achieved Coverage: 89%

Evaluation Criteria:

**Function Coverage**: Ensured that each function in f5.py is called and executed within the test cases to validate its intended functionality.

**Statement Coverage**: Focused on verifying that 89% of the code statements are executed and tested, ensuring most of the code logic and pathways are validated.

**Branch Coverage**: Reviewed and tested different branches within control structures (like if/else) to ensure that both the true and false conditions are evaluated.

**Condition Coverage**: Ensured that each condition within a decision structure (like logical OR/AND within if statements) is tested and validated for both true and false outcomes.

With an 89% coverage, the unit tests for f5.py affirmatively validate a significant portion of the codebase, ensuring the reliability of most logic paths. The untested 11% signifies areas requiring additional test development, potentially focusing on exception handling and specific conditional or branch logic, ensuring a more robust and fail-safe application.

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 | The app shall allow users to input their travel dates and the desired suburb | Full | Pass |  |
| 2 | The app shall display all the available properties listed corresponding to the users’ input | Full | Pass |  |
| 3 | Each property shall display their price and users shall be able to sort properties by their prices | Full | Pass |  |
| 4 | The app shall provide a feature where users can enter a certain keyword, such as pool or garden, and the app shall display all properties listed containing these keywords. | Full | Pass |  |
| 5 | The app shall be able to perform an analysis based on the comments and reviews the customers left on properties. For example, the app shall be able to display all comments about the cleanliness of a property or the satisfaction of customers regarding the property and calculate the average cleanliness or satisfaction with stars. | Full | Pass |  |
| 6 | The app shall display graphs of listings according to certain keyword | Full | Pass |  |