

Coverage Testing Report

Please provide your GitHub repository link.

GitHub Repository URL: https://github.com/UniKatya/Milestone2_Group19.git

The testing report should focus solely on **testing all the self-defined functions related to the five required features**. There is no need to test the GUI components. Therefore, it is essential to decouple your code and separate the logic from the GUI-related code.

You should perform statement coverage testing and branch coverage testing. For each type, provide a description and an analysis explaining how you evaluated the coverage.

1. Test Summary

list all tested functions related to the five required features:

| Tested Functions |
|--|
| load_data(file_path) |
| search_food_by_name(food_name) |
| get_nutritional_info(food_name) |
| filter_nutritional_info(nutritional_info) |
| create_pie_chart(filtered_sizes, filtered_categories, explode, ax) |
| create_bar_graph(filtered_categories, filtered_sizes, ax) |
| filter_food_by_nutrient_range(nutrient, min_val, max_val) |
| filter_food_by_nutrient_level(nutrient, level) |
| get_food_details(food_name, meal_plan) |
| generate_meal_plan(meal_plan, food_name, quantity) |
| generate_total_calories(meal_plan) |
| remove_food_from_meal_plan(meal_plan, food_name, quantity) |
| DataTable.GetNumberRows() |
| DataTable.GetNumberCols() |

| Tested Functions |
|---|
| <code>DataTable.GetValue(row, col)</code> |
| <code>DataTable.SetValue(row, col, value)</code> |
| <code>DataTable.GetColLabelValue(col)</code> |
| <code>DataTable.GetAttr(col, row, col, prop)</code> |

2. Statement Coverage Test

2.1 Description

To achieve 100% statement coverage, test cases were meticulously designed to ensure that every line of code in the functions related to the five required features is executed at least once. This involves creating tests that cover all possible paths through the code, including valid and invalid inputs. For example, the function `load_data(file_path)` was tested with a valid file path to ensure data is loaded successfully and an invalid file path to ensure the function handles errors properly. Specifically, `test_load_data_valid()` checks if the function correctly loads a valid CSV file, while `test_load_data_invalid()` verifies that a `FileNotFoundError` is raised for a non-existent file. Similarly, the function `search_food_by_name(food_name)` was tested with a valid food name that exists in the database to ensure it returns `True` and an invalid food name that does not exist to ensure it returns `False`. Other functions, such as `get_nutritional_info(food_name)`, `filter_nutritional_info(nutritional_info)`, `create_pie_chart(filtered_sizes, filtered_categories, explode, ax)`, `create_bar_graph(filtered_categories, filtered_sizes, ax)`, `filter_food_by_nutrient_range(nutrient, min_val, max_val)`, `filter_food_by_nutrient_level(nutrient, level)`, `get_food_details(food_name, meal_plan)`, `generate_meal_plan(meal_plan, food_name, quantity)`, `generate_total_calories(meal_plan)`, `remove_food_from_meal_plan(meal_plan, food_name, quantity)`, `DataTable.GetNumberRows()`, `DataTable.GetNumberCols()`, `DataTable.GetValue(row, col)`, `DataTable.SetValue(row, col, value)`, `DataTable.GetColLabelValue(col)`, and `DataTable.GetAttr(col, row, col, prop)`, were similarly tested with both valid and invalid inputs to ensure all lines of code were executed. This comprehensive testing approach ensures that every statement in the code is covered, providing confidence that the code behaves as expected under various conditions.

2.2 Testing Results

You can use the following command to run the statement coverage test and generate the report in the terminal. Afterward, include a screenshot of the report.

You must provide the `test_all_functions.py` file, which contains all test functions, otherwise `pytest` will not be able to execute the tests.

```
pytest --cov=all_functions --cov-report=term
```

Note: In the command above, the file/module `all_functions` does not include the `.py` extension. `all_functions.py` should contain all the tested functions related to the five required features.

```
(EKATERINA_KOZUB) PS C:\Griffiths University\Trimester 2 2024\1. Software Technologies 2810ICT\GROUP ASSIGNMENT\Github\Milestone2_Group19\code> pytest --cov=all_functions --cov-report=term
===== test session starts =====
platform win32 -- Python 3.8.19, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Griffiths University\Trimester 2 2024\1. Software Technologies 2810ICT\GROUP ASSIGNMENT\Github\Milestone2_Group19\code
plugins: cov-4.1.0, html-3.1.1, metadata-3.0.0, mock-3.10.0
collected 34 items

test_all_functions.py ..... [100%]

----- coverage: platform win32, python 3.8.19-final-0 -----
Name                Stmts   Miss  Cover
-----
all_functions.py      121     0   100%
TOTAL                 121     0   100%

===== 34 passed in 2.19s =====
```

3. Branch Coverage Test

3.1 Description

To achieve 100% branch coverage, test cases were meticulously designed to cover all possible branches in the code, ensuring that every conditional statement and its outcomes are tested. For instance, consider the function `load_data(file_path)`, which reads a CSV file and handles errors. To cover all branches, tests were created for a valid CSV file path, an invalid file extension, and a non-existent file. Specifically, `test_load_data_valid()` checks if the function correctly loads a valid CSV file, `test_load_data_invalid()` verifies that a `FileNotFoundError` is raised for a non-existent file. Similarly, for the function `search_food_by_name(food_name)`, which checks if a food name exists in a dataset, tests were designed to handle valid food names, non-string inputs, and empty strings. `test_search_food_by_name_valid()` confirms that the function returns `True` for valid food names, while `test_search_food_by_name_invalid()` ensure that `False` inputs are returned. This comprehensive approach ensures that all branches, including edge cases, are thoroughly tested, providing confidence that the code handles all possible scenarios correctly.

3.2 Testing Results

You can use the following command to run the branch coverage test and generate the report in the terminal. Afterward, include a screenshot of the report.

You must provide the `test_all_functions.py` file, which contains all test functions, otherwise pytest will not be able to execute the tests.

```
pytest --cov=all_functions --cov-branch --cov-report=term
```

Note: In the command above, the file/module `all_functions` does not include the `.py` extension. `all_functions.py` should contain all the tested functions related to the five required features.

```
(EKATERINA_KOZUB) PS C:\Griffiths University\Trimester 2 2024\1. Software Technologies 2810ICT\GROUP ASSIGNMENT\Github\Milestone2_Group19\code> pytest --cov=all_functions --cov-bran
nch --cov-report=term
===== test session starts =====
platform win32 -- Python 3.8.19, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Griffiths University\Trimester 2 2024\1. Software Technologies 2810ICT\GROUP ASSIGNMENT\Github\Milestone2_Group19\code
plugins: cov-4.1.0, html-3.1.1, metadata-3.0.0, mock-3.10.0
collected 34 items

test_all_functions.py ..... [100%]

----- coverage: platform win32, python 3.8.19-final-0 -----
Name                               Stmts   Miss Branch BrPart  Cover
-----
all_functions.py                   121      0     44      0   100%
-----
TOTAL                               121      0     44      0   100%

===== 34 passed in 2.14s =====
```