Unit Testing Report

GitHub Repository URL: https://github.com/kisangkay/Milestone2_Group62.git

1. Test Summary

Tested Functions	Test Functions
`on_search(event)`	`test_on_search_valid()` `test_on_search_invalid()`
`get_on_food_selected(event)`	`test_on_food_selected_valid()` `test_on_food_selected_invalid()`
`get_on_filter(food_data, nutrient, min_density, max_density)`	`test_on_filter_valid()` `test_on_filter_invalid()`
`get_filter_nutrition_level(food_data, selected_nutrient, level)`	`test_filter_nutrition_level_valid()` `test_filter_nutrition_level_invalid()`
`get_filter_nutrition_level(food_data, selected_nutrient, level)`	`test_get_comparison_click_valid()` `test_get_comparison_click_invalid()`

2. Test Case Details

Test Case 1:

- **Test Function/Module**
  - `test_on_search_valid()`
  - `test_on_search_invalid()`
- **Tested Function/Module**
  - `on_search(event)`
- **Description**
  - This function handles the logic of searching a term in a DataFrame column (food) when the user clicks a search button. It retrieves the search term from a text control, filters the data, and updates the GUI grid if matches are found. If no matches are found, it displays a message box.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`search_term='apple'`	`filtered_data contains rows with 'apple'`

Valid Input	Expected Output
search_term='banana'	filtered_data contains rows with 'banana'

- **1) Code for the Test Function**

```
def test_on_search_valid():
    df = pd.DataFrame({'food': ['apple', 'banana', 'cherry', 'apple pie', 'blueberry']})
    result = on_search(df, 'apple')
    assert not result.empty
    assert len(result) == 2  # 'apple' and 'apple pie'

    result = on_search(df, 'banana')
    assert not result.empty
    assert len(result) == 1  # 'banana'

    result = on_search(df, 'berry')
    assert not result.empty
    assert len(result) == 1  # 'blueberry'
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
search_term=''	Display message: "Search result not found"
search_term='group62'	Display message: "Search result not found"

- **2) Code for the Test Function**

```
def test_on_search_invalid():
    df = pd.DataFrame({'food': ['apple', 'banana', 'cherry', 'apple pie', 'blueberry']})
    result = on_search(df, '')
    assert result.empty  # Should be empty since no search term is provided

    result = on_search(df, 'group62')
    assert result.empty  # Should be empty as the search term does not exist in the data
```

Test Case 2:

- **Test Function/Module**
  - `test_on_food_selected_valid()`
  - `test_on_food_selected_invalid()`
- **Tested Function/Module**
  - `get_on_food_selected(event)`
- **Description**
  - This function processes the event of a food item being selected from a list, retrieves its nutrient information, and generates a pie chart to visualize the nutrient breakdown.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
event where the selected food is available in food_data	Nutrient data is extracted and matches the expected values for pie chart

- **1) Code for the Test Function**

```
def test_on_food_selected_valid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana'],
        'Caloric Value': [52, 89],
        'Fat': [0.2, 0.3],
        'Saturated Fats': [0.03, 0.11],
        'Protein': [0.3, 1.1]
    })

    selected_food = 'Apple'
    categories, sizes = get_on_food_selected(food_data, selected_food)

    # Assert that the extracted nutrient data matches the expected values for 'Apple'
    assert categories == ['Caloric Value', 'Fat', 'Saturated Fats', 'Protein']
    assert sizes == [52, 0.2, 0.03, 0.3]
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
event where the selected food is not in food_data	Handle Exception

- **2) Code for the Test Function**

```
def test_on_food_selected_invalid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana'],
        'Caloric Value': [52, 89],
        'Fat': [0.2, 0.3],
        'Saturated Fats': [0.03, 0.11],
        'Protein': [0.3, 1.1]
    })

    selected_food = 'Orange'

    # Call the function and check if the appropriate exception is handled
    with pytest.raises(IndexError) as exc_info:
        get_on_food_selected(food_data, selected_food)
    assert exc_info.type is IndexError
```

Test Case 3:

- **Test Function/Module**

- test_on_filter_valid()
- test_on_filter_invalid()

- **Tested Function/Module**
  - get_on_filter(food_data, nutrient, min_density, max_density)
- **Description**
  - This function filters a DataFrame of food items based on the specified nutrient density range.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
nutrient = 'Nutrition Density', min_density = 10, max_density = 50	Returns rows with Nutrition Density between 10 and 50
nutrient = 'Protein', min_density = 0.5, max_density = 2	-Returns rows with Protein value between 0.5 and 2

- **1) Code for the Test Function**

```
def test_on_filter_valid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana', 'Carrot'],
        'Nutrition Density': [30, 45, 10],
        'Protein': [0.3, 1.1, 0.9]
    })

    # Test with Nutrition Density filter
    nutrient = 'Nutrition Density'
    min_density = 10
    max_density = 50
    results = get_on_filter(food_data, nutrient, min_density, max_density)

    assert len(results) == 3  # All rows are within range
    assert 'Apple' in results['food'].values
    assert 'Banana' in results['food'].values
    assert 'Carrot' in results['food'].values

    # Test with Protein filter
    nutrient = 'Protein'
    min_density = 0.5
    max_density = 2
    results = get_on_filter(food_data, nutrient, min_density, max_density)

    assert len(results) == 2  # Only Banana and Carrot are within the range
    assert 'Banana' in results['food'].values
    assert 'Carrot' in results['food'].values
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
nutrient = 'group62', min_density = 0, max_density = 50)	Raises KeyError because the nutrient doesn't exist

- **2) Code for the Test Function**

```
def test_on_filter_invalid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana', 'Carrot'],
        'Nutrition Density': [30, 45, 10],
        'Protein': [0.3, 1.1, 0.9]
    })

    # Test with an invalid nutrient
    nutrient = 'group62'
    min_density = 0
    max_density = 50

    # Call the function and check if the appropriate exception is raised
    with pytest.raises(KeyError) as exc_info:
        get_on_filter(food_data, nutrient, min_density, max_density)
    assert exc_info.type is KeyError
```

Test Case 4:

- **Test Function/Module**
  - `test_filter_nutrition_level_valid()`
  - `test_filter_nutrition_level_invalid()`
- **Tested Function/Module**
  - `get_filter_nutrition_level(food_data, selected_nutrient, level)`
- **Description**
  - This function filters a DataFrame of food items based on a specified nutrient's value and its level (Low, Mid, High).
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`selected_nutrient = 'Protein', level = 'Low'`	`Returns rows where Protein is less than 0.33 * max`
`selected_nutrient = 'Protein', level = 'Mid'`	`Returns rows where 0.33 * max ≤ Protein ≤ 0.66 * max`
`selected_nutrient = 'Protein', level = 'High'`	`Returns rows where Protein is greater than 0.66 * max`

- **1) Code for the Test Function**

```
def test_filter_nutrition_level_valid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana', 'Carrot'],
        'Protein': [0.3, 1.1, 0.9]
    })
```

```
    max_protein = food_data['Protein'].max()  # max value is 1.1

    # Test with 'Low' level
    results = get_filter_nutrition_level(food_data, 'Protein', 'Low')
    assert len(results) == 1  # Only 'Apple' should be in the Low category
    assert 'Apple' in results['food'].values

    # Test with 'Mid' level
    results = get_filter_nutrition_level(food_data, 'Protein', 'Mid')
    assert len(results) == 0  # No food item falls in the Midcategory according to the range

    # Test with 'High' level
    results = get_filter_nutrition_level(food_data, 'Protein', 'High')
    assert len(results) == 2  # Both 'Banana' and 'Carrot' should be in the High category
    assert 'Banana' in results['food'].values
    assert 'Carrot' in results['food'].values
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
selected_nutrient = 'group62', level = 'Low'	Raises KeyError for an unknown nutrient
selected_nutrient = 'Protein', level = 'large'	Raises ValueError for an invalid level

- **2) Code for the Test Function**

```
def test_filter_nutrition_level_invalid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana', 'Carrot'],
        'Protein': [0.3, 1.1, 0.9]
    })

    # Test with an unknown nutrient
    with pytest.raises(KeyError) as exc_info:
        get_filter_nutrition_level(food_data, 'group62', 'Low')
    assert exc_info.type is KeyError

    # Test with an invalid level
    with pytest.raises(ValueError) as exc_info:
        get_filter_nutrition_level(food_data, 'Protein', 'large')
    assert exc_info.type is ValueError
```

Test Case 5:

- **Test Function/Module**
  - test_get_comparison_click_valid()
  - test_get_comparison_click_invalid()
- **Tested Function/Module**
  - get_comparison_click(food_data, selected_nutrient, selected_foods)
- **Description**

- This function retrieves nutrient data for three selected foods to compare their values in a bar chart.

- **1) Valid Input and Expected Output**

Valid Input	Expected Output
selected_nutrient = 'Protein', selected_foods = ['Apple', 'Banana', 'Carrot']	Labels: ['Apple', 'Banana', 'Carrot'], Values: [0.3, 1.1, 0.9]

- **1) Code for the Test Function**

```
def test_get_comparison_click_valid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana', 'Carrot'],
        'Protein': [0.3, 1.1, 0.9]
    })

    selected_nutrient = 'Protein'
    selected_foods = ['Apple', 'Banana', 'Carrot']

    labels, values = get_comparison_click(food_data, selected_nutrient, selected_foods)

    # Assert that the labels and values match the expected output
    assert labels == ['Apple', 'Banana', 'Carrot']
    assert values == [0.3, 1.1, 0.9]
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
selected_foods = ['Apple', 'Banana']	Raises ValueError for not providing exactly three foods
selected_foods = ['Apple', 'Banana', 'Orange']	Raises ValueError for food not present in the dataset

- **2) Code for the Test Function**

```
def test_get_comparison_click_invalid():
    # Mock food data
    food_data = pd.DataFrame({
        'food': ['Apple', 'Banana', 'Carrot'],
        'Protein': [0.3, 1.1, 0.9]
    })

    selected_nutrient = 'Protein'

    # Test with less than three foods
    selected_foods = ['Apple', 'Banana']
    with pytest.raises(ValueError) as exc_info:
        get_comparison_click(food_data, selected_nutrient, selected_foods)
    assert str(exc_info.value) == "Exactly three foods must be selected for comparison."
```

```
    # Test with a food not present in the dataset
    selected_foods = ['Apple', 'Banana', 'Orange']
    with pytest.raises(ValueError) as exc_info:
        get_comparison_click(food_data, selected_nutrient, selected_foods)
    assert str(exc_info.value) == "One or more selected foods are not present in the dataset."
```

3. Testing Report Summary