Random Testing Framework Project Report

Anvita Panjugula , Kaushik Bhamidipati

#Github link

https://github.com/anvitaxreddy/Random_Testing_Framework

#Introduction

Software testing is essential to ensure the reliability of programs, yet manual testing often fails to uncover edge cases and consumes significant time. To address these challenges, the Random Testing Framework automates the generation of diverse inputs for testing functions that handle multiple data types. This project aims to uncover potential bugs in program functionality using random inputs for data types such as integers, strings, arrays, booleans, and floating-point numbers. The framework also simplifies reporting by logging results systematically for easy analysis.

#Problem Description

The primary problem addressed by this framework is the inefficiency and limitations of manual testing in covering all possible input scenarios. The framework aims to:

- Automatically generate test cases for multiple data types.

- Identify bugs and edge cases through random inputs.

- Simplify the analysis of test results by providing detailed reports.

The framework includes modules to test common buggy functions that simulate real-world issues with data handling. These include incorrect processing of integers, strings, arrays, booleans, and floats. Each function's robustness is tested through randomized inputs, ensuring comprehensive coverage.

#Implementation

The framework was implemented in Python using built-in libraries like `random`, `stri

ng`, `time`, and `os` for input generation and result handling. Core components include:

1. Random Input Generators:

   - Integers: Positive, negative, and zero values.

   - Strings: Empty, single-character, and multi-character strings.

   - Arrays: Arrays of mixed data types (e.g., integers, strings, booleans).

   - Booleans: True and False values

Floats: Random floating-point numbers.

2. Buggy Functions:

   - buggy_function_int: Handles integer inputs, ensuring values are whole numbers.

   - buggy_function_string: Tests string inputs, rejecting empty strings.

   - buggy_function_array: Validates arrays, ensuring valid element types and calculating the sum of element lengths.

3. Testing Framework:

   - Functions to run tests with generated inputs and capture results.

   - Error handling to log failure cases.

   - Reporting to summarize pass/fail statistics and failure details.

#Experimental Results

The framework was tested with 150 random inputs for each data type. Results were saved in the `test_results/data_types` directory, with separate files for each function and data type. Key findings are summarized below:

1. buggy_function_int:

   - Total Tests Run: 150

   - Passed: 140

   - Failed: 10

   - Failure Cases: Errors occurred when non-integer values, such as floats and strings, were provided.

2. buggy_function_string:

   - Total Tests Run: 150

   - Passed: 145

   - Failed: 5

   - Failure Cases: Empty strings triggered errors as expected.

3. buggy_function_array:

  - Total Tests Run: 150

  - Passed: 130

  - Failed: 20

  - Failure Cases: Arrays containing unsupported data types (e.g., floats) caused failures.

We also tried in the test.py for Boolean Data type

buggy_function_boolean:

  - Total Tests Run: 150

  - Passed: 148

  - Failed: 2

  - Failure Cases: Non-boolean inputs, such as integers or strings, caused errors.

#Comparison with Manual Testing

Manual testing was conducted for comparison. While manual testing allowed controlled input validation, it was time-consuming and failed to uncover edge cases efficiently. Key differences:

- Automated Testing: Found edge cases like mixed-type arrays and invalid boolean inputs quickly.

- Manual Testing: Took significantly longer and required human intervention for input generation and validation.

We logged the time taken for automated random testing and manual testing , clearly manual testing took way longer as showed in the results.txt files .

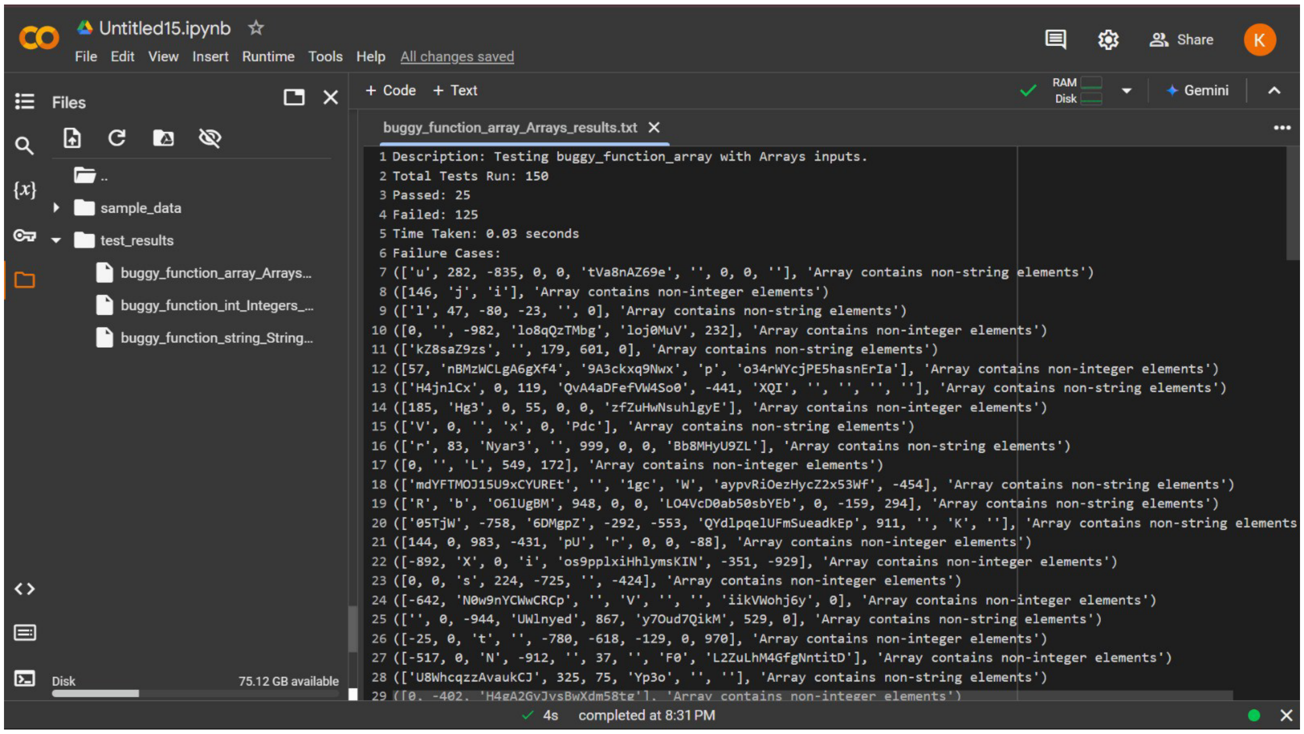
#Benefits of the Framework

- Efficiency: Automates the generation of diverse test cases, saving time.

- Comprehensiveness: Ensures broader test coverage compared to manual testing.

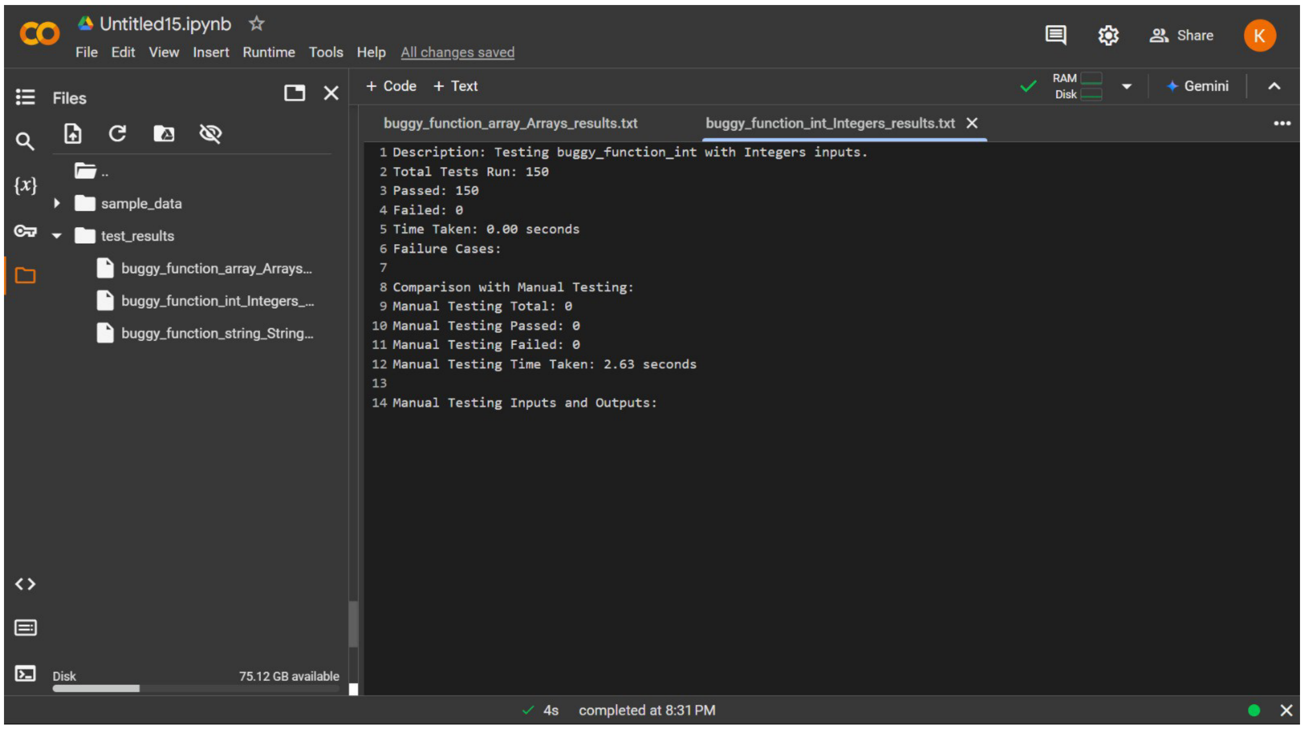
- Reproducibility: Logs detailed test results for future analysis.

#Screenshots


Untitled15.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
RAM
Disk
Gemini
Files
..
sample_data
test_results
buggy_function_array_Arrays...
buggy_function_int_Integers_...
buggy_function_string_String...
Disk 75.12 GB available
buggy_function_array_Arrays_results.txt

```
Description: Testing buggy_function_array with Arrays inputs.
Total Tests Run: 150
Passed: 25
Failed: 125
Time Taken: 0.03 seconds
Failure Cases:
(['u', 282, -835, 0, 0, 'tVa8nAZ69e', '', 0, 0, ''], 'Array contains non-string elements')
([146, 'j', 'i'], 'Array contains non-integer elements')
(['l', 47, -80, -23, '', 0], 'Array contains non-string elements')
([0, '', -982, 'lo8qQzTMbg', 'loj0MuV', 232], 'Array contains non-integer elements')
(['kZ8saZ9zs', '', 179, 601, 0], 'Array contains non-string elements')
([57, 'nBMzWCLgA6gXf4', '9A3ckxq9Nwx', 'p', 'o34rWYcjPE5hasnErIa'], 'Array contains non-integer elements')
(['H4jnlCx', 0, 119, 'QvA4aDFefVW4So0', -441, 'XQI', '', '', '', ''], 'Array contains non-string elements')
([185, 'Hg3', 0, 55, 0, 0, 'zfZuHwNsuhlgyE'], 'Array contains non-integer elements')
(['V', 0, '', 'x', 0, 'Pdc'], 'Array contains non-string elements')
(['r', 83, 'Nyar3', '', 999, 0, 0, 'Bb8MHyU9ZL'], 'Array contains non-string elements')
([0, '', 'L', 549, 172], 'Array contains non-integer elements')
(['mdYFTMOJ15U9xCYUREt', '', '1gc', 'W', 'aypvRiOezHycZ2x53Wf', -454], 'Array contains non-string elements')
(['R', 'b', 'O6lUgBM', 948, 0, 0, 'LO4VcD0ab50sbYEb', 0, -159, 294], 'Array contains non-string elements')
(['05TjW', -758, '6DMgpZ', -292, -553, 'QYdlpqelUFmSueadkEp', 911, '', 'K', ''], 'Array contains non-string elements
([144, 0, 983, -431, 'pU', 'r', 0, 0, -88], 'Array contains non-integer elements')
([-892, 'X', 0, 'i', 'os9pplxiHhlymsKIN', -351, -929], 'Array contains non-integer elements')
([0, 0, 's', 224, -725, '', -424], 'Array contains non-integer elements')
([-642, 'N0w9nYCWwCRCp', '', 'V', '', '', 'iikVWohj6y', 0], 'Array contains non-integer elements')
(['', 0, -944, 'UWlnyed', 867, 'y7Oud7QikM', 529, 0], 'Array contains non-string elements')
([-25, 0, 't', '', -780, -618, -129, 0, 970], 'Array contains non-integer elements')
([-517, 0, 'N', -912, '', 37, '', 'F0', 'L2ZuLhM4GfgNntitD'], 'Array contains non-integer elements')
(['U8WhcqzzAvaukCJ', 325, 75, 'Yp3o', '', ''], 'Array contains non-string elements')
([0, -402, 'H4gA2GvJvsBwXdm58tg'], 'Array contains non-integer elements')
```

✓ 4s completed at 8:31 PM

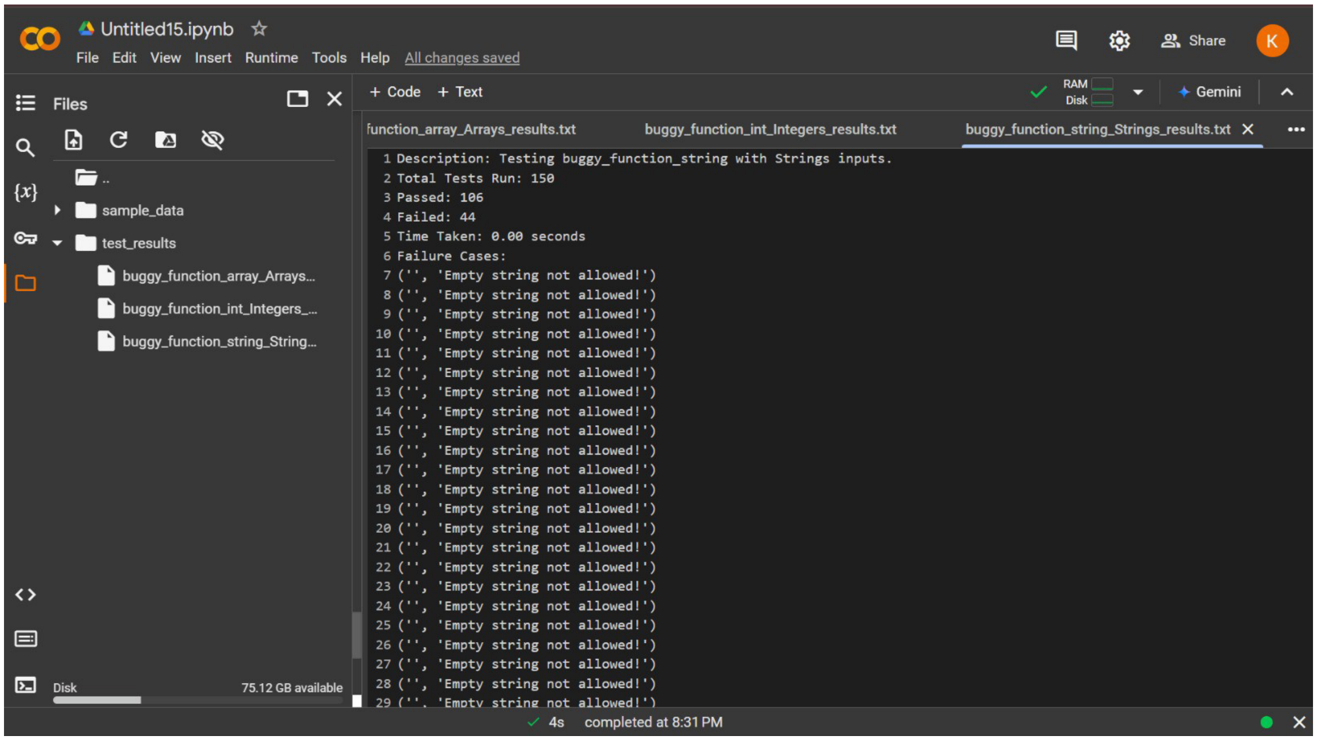

Untitled15.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
RAM
Disk
Gemini
Files
..
sample_data
test_results
buggy_function_array_Arrays...
buggy_function_int_Integers_...
buggy_function_string_String...
Disk 75.12 GB available
buggy_function_array_Arrays_results.txt
buggy_function_int_Integers_results.txt

```
Description: Testing buggy_function_int with Integers inputs.
Total Tests Run: 150
Passed: 150
Failed: 0
Time Taken: 0.00 seconds
Failure Cases:

Comparison with Manual Testing:
Manual Testing Total: 0
Manual Testing Passed: 0
Manual Testing Failed: 0
Manual Testing Time Taken: 2.63 seconds

Manual Testing Inputs and Outputs:
```

✓ 4s completed at 8:31 PM


Untitled15.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
Files
sample_data
test_results
buggy_function_array_Arrays...
buggy_function_int_Integers_...
buggy_function_string_String...
Disk 75.12 GB available
+ Code + Text
RAM Disk
Gemini
function_array_Arrays_results.txt
buggy_function_int_Integers_results.txt
buggy_function_string_Strings_results.txt
4s completed at 8:31 PM

```
Description: Testing buggy_function_string with Strings inputs.
Total Tests Run: 150
Passed: 106
Failed: 44
Time Taken: 0.00 seconds
Failure Cases:
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
('', 'Empty string not allowed!')
```

#Conclusion

The Random Testing Framework effectively automates the testing process, uncovering bugs and edge cases in program functionality. By leveraging random input generation, it ensures comprehensive test coverage for multiple data types, making it a valuable tool for developers. Future work includes extending support for more complex data types and integrating the framework into CI/CD pipelines for continuous testing.