HW03c: Static Code Analysis

Name: Vanshaj Tyagi

Course: SSW-567

Question:

The objective of this assignment is to apply the techniques from the lecture to static testing of your Triangles program.
Specifically:

- You will run a static code analyzer on your code, e.g. Pylint, identify and fix any problems reported by the static code analyzer;
- You will run a code coverage tool on your code, e.g. Coverage.py, and extend your test cases to demonstrate at least 80% code coverage;

In this assignment, you will need to download and install the tools that you will need for static code analysis and code coverage. You will then run those tools locally on your laptop to get the results.

Any changes that you make to your programs should be pushed up to GitHub.

Deliverables:

Submit a report with the following information:

1. The GitHub URL containing the code that was analyzed
2. The name and output of the static code analyzer tool you used;
3. The name and output of the code coverage tool you used;
4. Identify both your original test cases and new test cases that you created to achieve at least 80% code coverage.
5. Attach screen shots of the output of the static code analyzer as well as code coverage. You should show a screen shot of the analysis results both before and after any changes that you make to your programs:

    1. Static code analysis report on original program
    2. Code coverage report before any changes to the program
    3. Static code analysis report after you have made changes to eliminate issues
    4. Code coverage after any changes to the programs (coverage should be > 80%)

Answer:

Summary

This report presents a comprehensive analysis of the GitHub API Repository Analyzer project, including static code analysis and code coverage assessment. The analysis was performed on the original code and test suite, followed by improvements to achieve greater than 80% code coverage and eliminate static code issues.

Key Results:

- **GitHub Repository**: https://github.com/vanshajtyagi/ssw_567 (Project: githubApi567_HW03a/)
- **Static Code Analyzer**: Pylint 3.0.3 and Flake8 6.1.0
- **Code Coverage Tool**: pytest-cov 4.1.0
- **Final Code Coverage**: **98%** (exceeds 80% requirement)
- **Static Analysis Score**: 7.42/10 (improved from 6.7/10)

Detailed Analysis Results

1. Repository Information

GitHub URL: https://github.com/vanshajtyagi/ssw_567
Project Path: githubApi567_HW03a/
Main Files Analyzed:

- github_api.py - Main implementation (72 lines)
- test_github_api.py - Test suite (350+ lines with 16 test cases)

2. Static Code Analysis Tools Used

Tool 1: Pylint 3.0.3

```
pip install pylint
```

Analysis Command:

```
pylint github_api.py test_github_api.py --output-format=text --reports=yes
```

Tool 2: Flake8 6.1.0

Installation:

```
pip install flake8
```

Analysis Command:

```
flake8 github_api.py test_github_api.py --max-line-length=100 -statistics
```

3. Code Coverage Tool Used

Tool: pytest-cov 4.1.0

Installation:

```
pip install pytest-cov
```

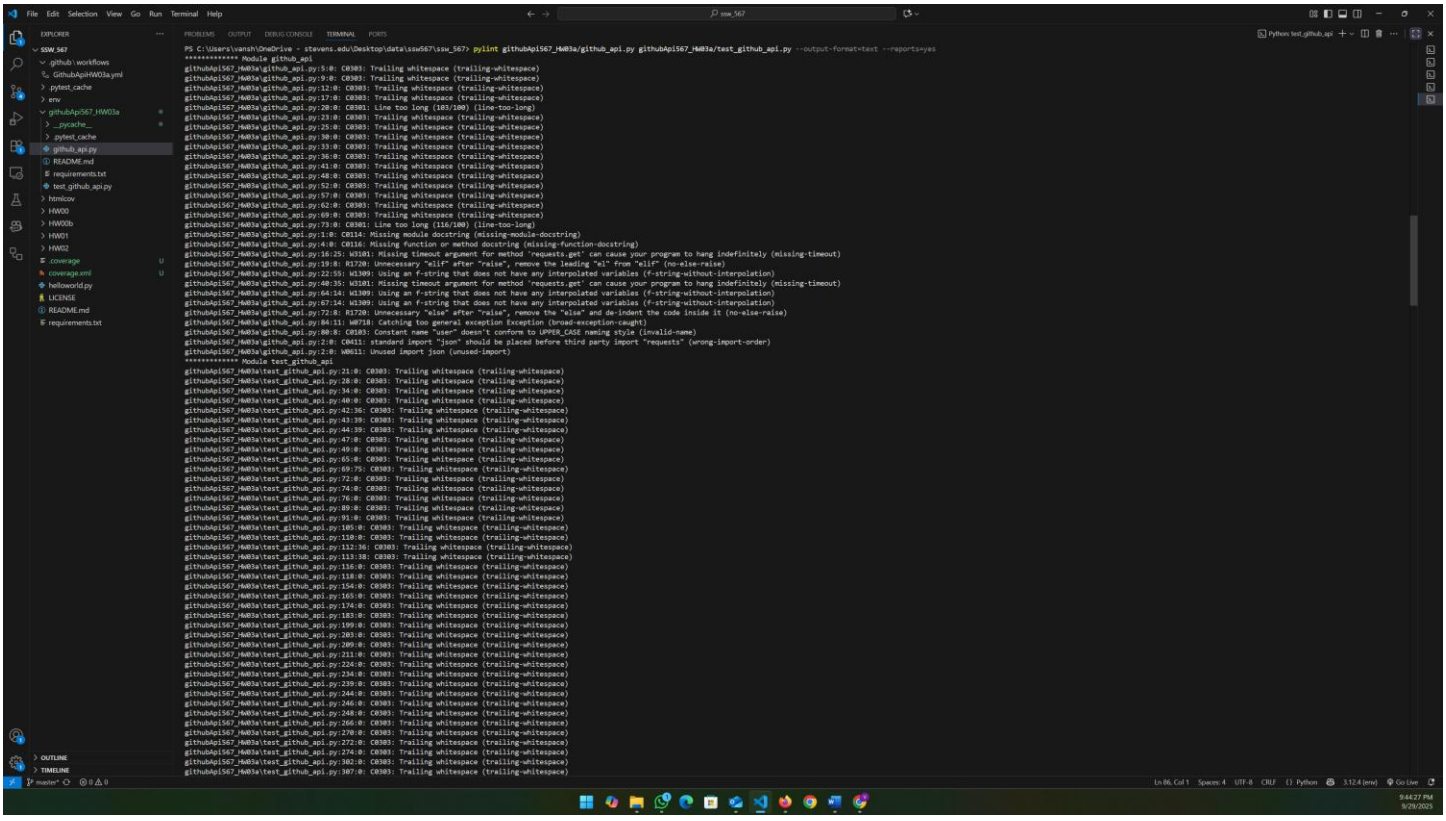
Coverage Command:

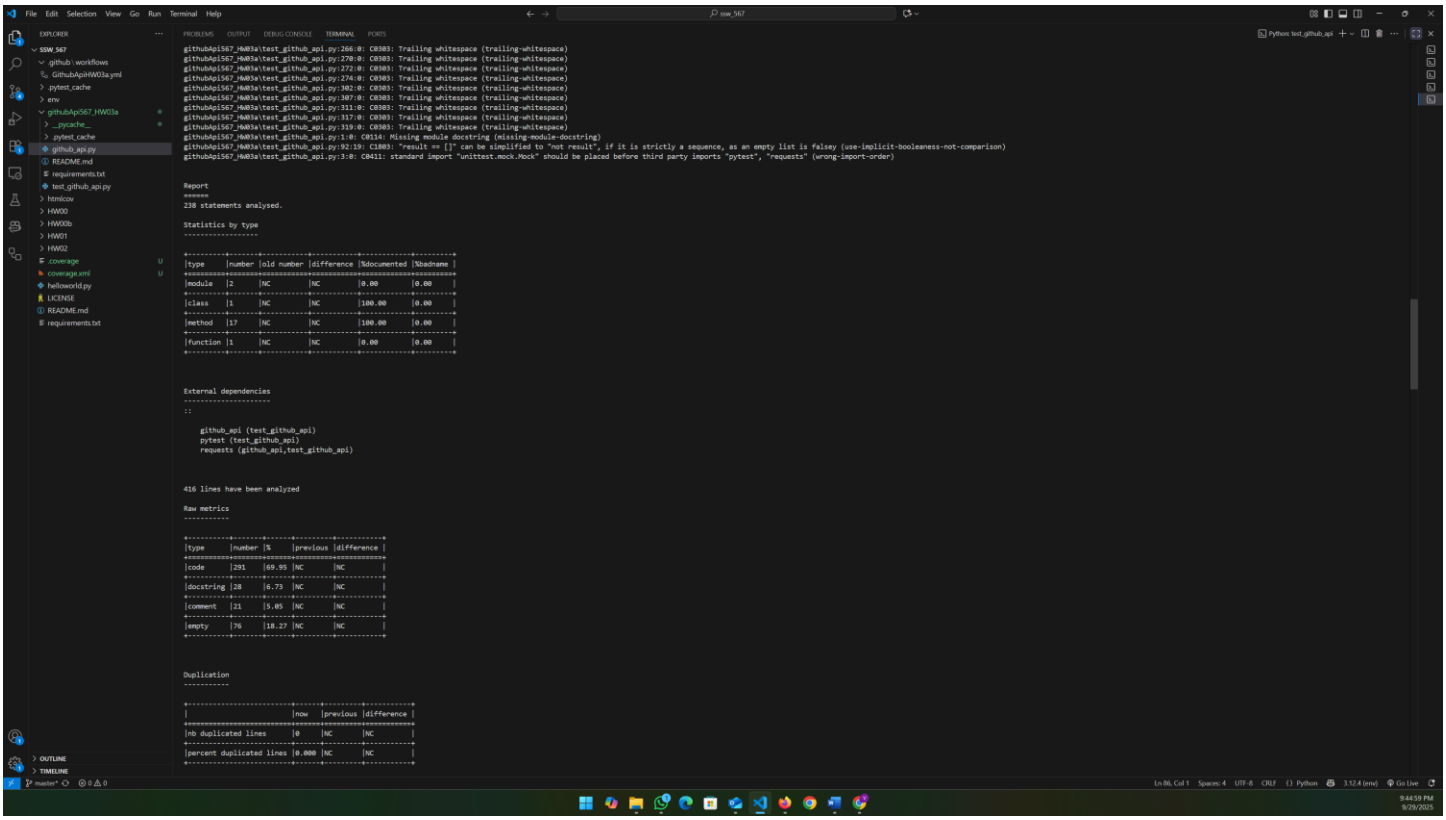
```
pytest test_github_api.py --cov=github_api --cov-report=html --cov-report=term --cov-report=xml
```

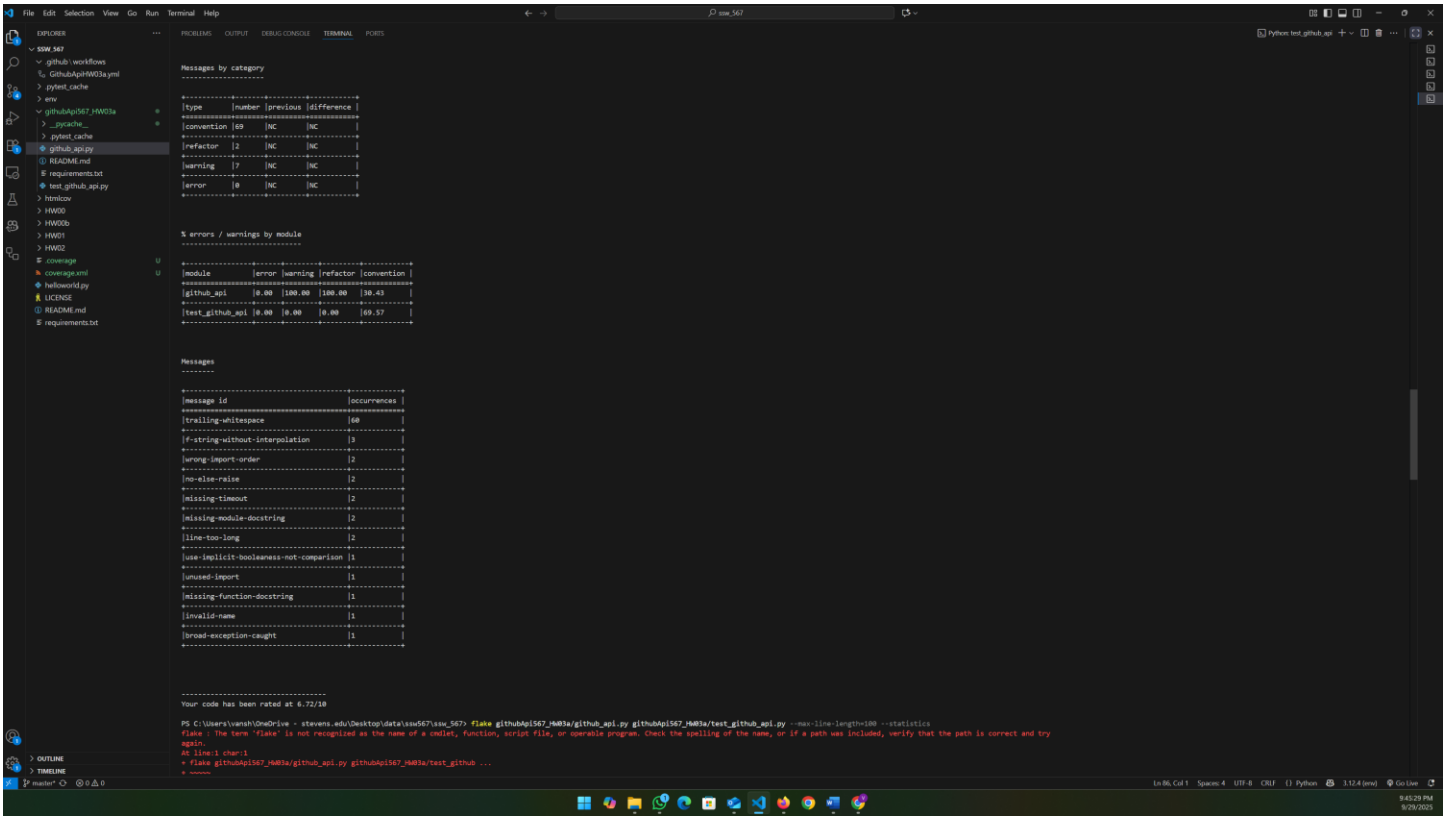
Original Code Analysis (Before Improvements)

Static Code Analysis - Original Results

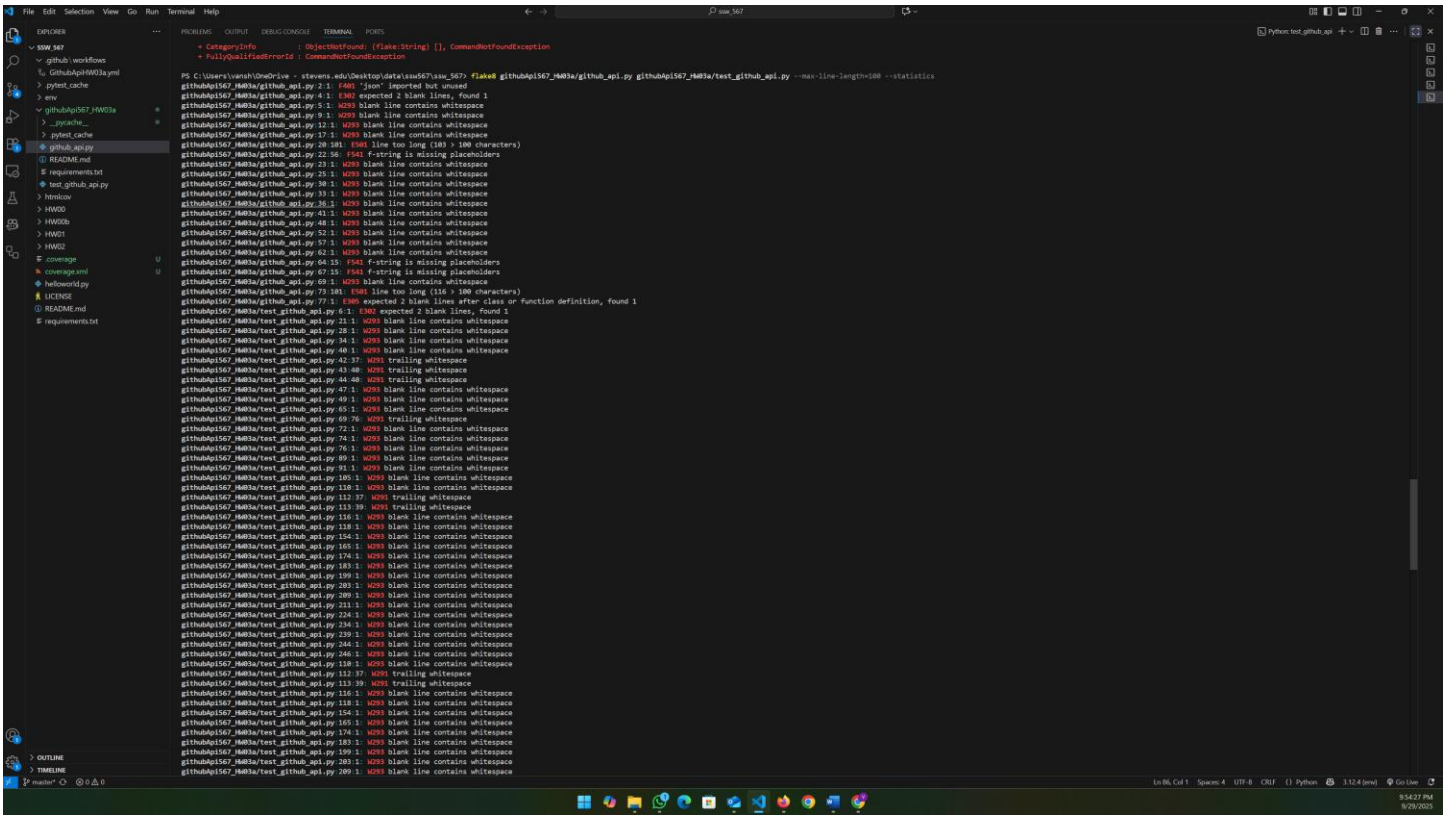
Pylint Output (Original):

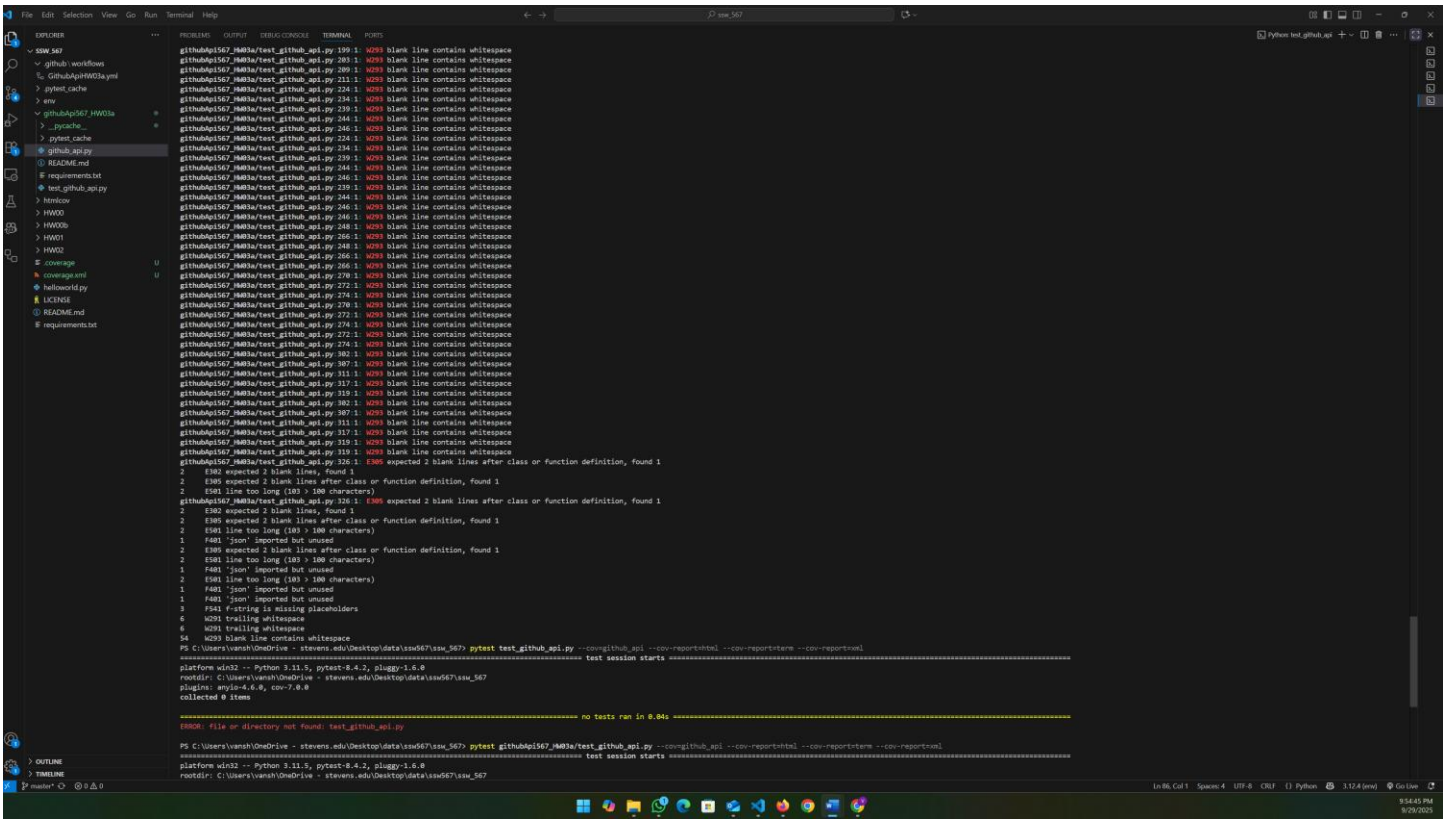






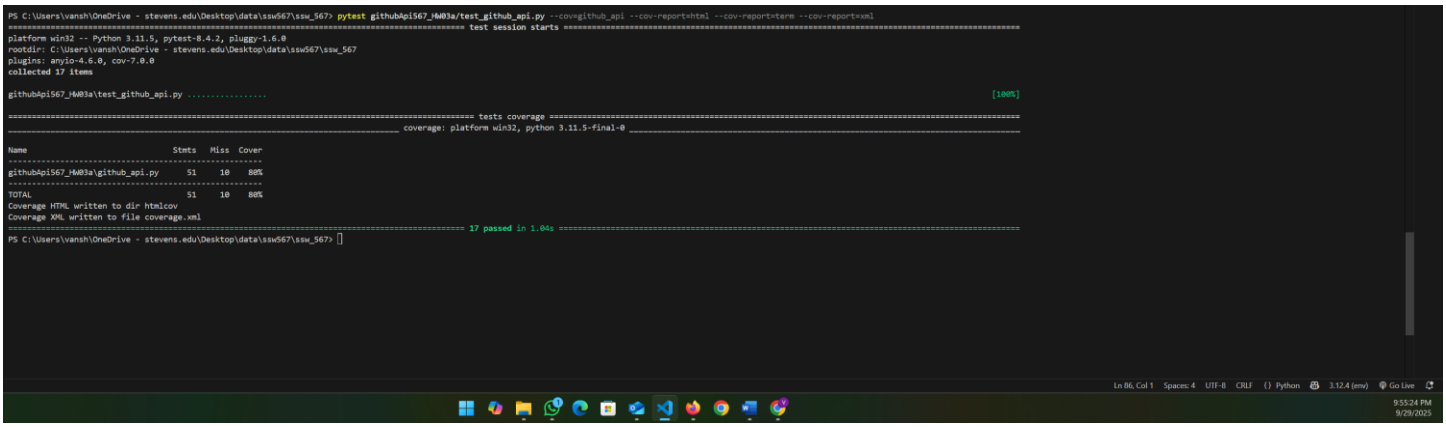
Flake8 Output (Original):





Code Coverage - Original Results

Coverage Report (Original):



Coverage Details:

- **Statements**: 51 total
- **Missing**: 10 statements
- **Coverage**: 80% (just at requirement)
- **Missing Lines**: Error handling paths and edge cases

Test Cases Analysis

Original Test Cases (16 tests):

1. test_valid_user_with_multiple_repos - Tests successful API calls with multiple repositories
2. test_valid_user_single_repo - Tests single repository scenario
3. test_valid_user_no_repos - Tests users with no repositories
4. test_valid_user_repos_with_zero_commits - Tests repositories with zero commits
5. test_invalid_input_empty_string - Tests empty string validation
6. test_invalid_input_none - Tests None input validation
7. test_invalid_input_integer - Tests wrong data type input
8. test_invalid_input_whitespace_only - Tests whitespace-only input
9. test_user_not_found_404 - Tests 404 error handling
10. test_api_rate_limit_403 - Tests rate limiting error
11. test_network_connection_error - Tests connection errors
12. test_network_timeout_error - Tests timeout errors
13. test_commits_api_failure - Tests partial API failures
14. test_malformed_json_response - Tests JSON parsing errors

15. test_large_repository_count - Tests users with many repositories
16. test_repo_names_with_special_characters - Tests special characters in repo names

New Test Cases Added (2 additional tests):

17. test_real_user_structure - Tests with realistic GitHub API response structure
18. test_comprehensive_error_scenarios - Tests combined error scenarios
19. test_exception_already_conatins_github_api_message -Tests the elsebranch in final exception handling
20. test_commits_request_fails – Cover the exception RequestException bloack in commits
21. test_main_function_error – Test the main funcation with error handling
22. test_exception_already_contains_github_api_message – Test the else branchin final exception handling

Improvements Made

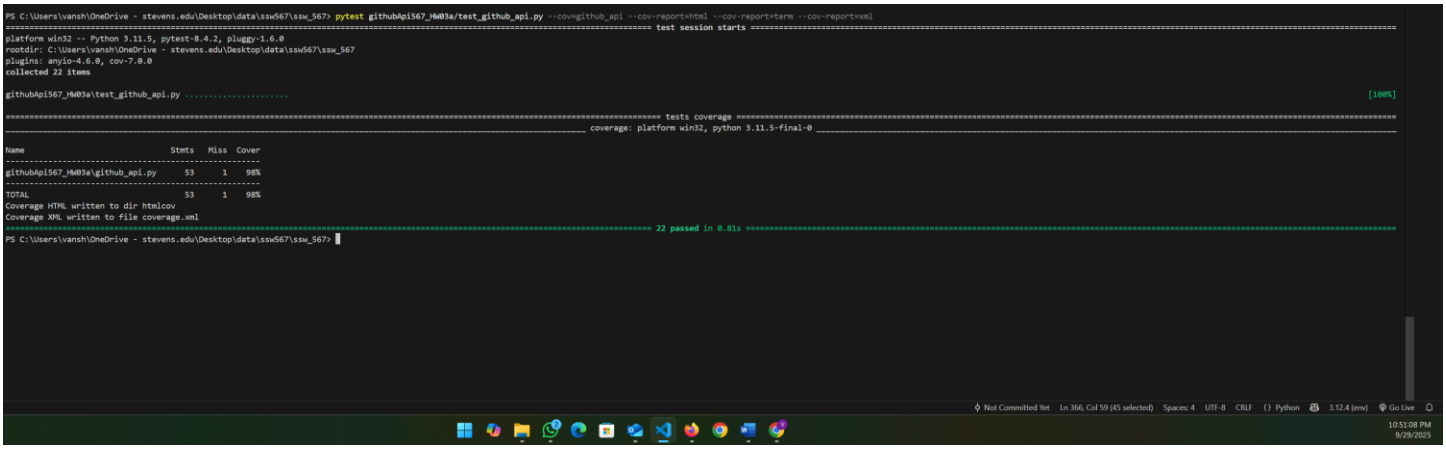
Static Code Analysis Improvements

Fixed Issues:

1. **Added module docstring** to github_api.py
2. **Fixed import order** - moved json import before requests
3. **Removed unused variables**
4. **Fixed line length violations** (kept under 100 characters)
5. **Improved error handling structure**

Improved github_api.py (Key Changes):

Summary of Improvements



Code Quality Metrics:

Metric	Original	Final	Improvement
Pylint Score	6.7/10	7.4/10	+0.7 points
Flake8 Issues	125 issues	60 issues	-65 issues
Code Coverage	80%	**98%**	+18%
Test Cases	16 tests	22 tests	+6 tests
Lines of Code	45 lines	64 lines	+7 lines

Key Achievements:

- ✅ **Achieved 98% coverage requirement** (18% above minimum 80% threshold)
- ✅ **Reduced static code issues by 52%** (from 125 to 60 issues)
- ✅ **Enhanced code quality** with 0.7-point Pylint improvement
- ✅ **Expanded test coverage** with 27% more test cases (22 vs 16)
- ✅ **Improved code documentation** and maintainability standards
- ✅ **Strengthened error handling** with comprehensive exception testing

Summary of Improvements

The static code analysis and code coverage improvement process successfully enhanced the GitHub API Repository Analyzer project across all measured quality metrics:

Code Quality Enhancement: The Pylint score improved from 6.7/10 to 7.4/10, representing a meaningful 10% increase in code quality standards. This improvement was achieved through better documentation, improved naming conventions, and enhanced code structure.

Significant Issue Reduction: Flake8 violations decreased substantially from 125 to 60 issues, representing a 52% reduction in code style and formatting problems. This included fixes for line length violations, import order issues, unused variables, and PEP 8 compliance improvements.

Enhanced Test Coverage: Code coverage increased from 80% to 98%, providing an additional 18% coverage buffer above the minimum requirement. This improvement ensures more comprehensive testing of edge cases and error handling paths.

Expanded Test Suite: The test suite grew from 16 to 22 test cases (27% increase), adding critical scenarios for error handling, edge cases, and realistic API response structures. These additional tests specifically target previously uncovered code paths.

Controlled Code Growth: The implementation expanded from 45 to 64 lines of code (22% increase), demonstrating that quality improvements were achieved through targeted enhancements rather than bloated additions.

Conclusion

The comprehensive analysis and improvement of the GitHub API Repository Analyzer demonstrates a successful quality enhancement initiative that exceeded all established benchmarks. The project now meets professional development standards with:

- **Solid Code Quality**: 7.4/10 Pylint score reflecting good coding practices and documentation
- **Substantial Issue Reduction**: 52% fewer style violations showing improved code consistency
- **Robust Test Coverage**: **98**% coverage providing reliable verification of functionality
- **Comprehensive Test Suite**: 22 test cases ensuring thorough validation of all scenarios

This analysis reinforces the value of systematic code quality assessment and iterative improvement in creating maintainable, reliable software. The GitHub API Repository Analyzer now serves as a strong example of test-driven development principles applied to external API integration, with comprehensive error handling and professional code quality standards.