Final Project Report - Group 5 ENPM611

- By Palak Gupta

Project Title: GitHub Issue Analysis Tool for Python Poetry

Team Members :

- Palak Gupta (Project Manager)
- Bhavna Kumari (Developer & Tester)
- Bolla Sai Saketh (Developer & Tester)
- Souhardya Pal (Developer & Tester)

1. Introduction

1.1 Project Context

This project focuses on analyzing GitHub issues from the open-source Python Poetry project (https://github.com/python-poetry/poetry). GitHub issue data provides insights into development processes, contributor activity, and community interactions distinct from code commit history. Poetry is an actively maintained package management platform for Python.

1.2 Project Description & Objectives

Developed for the ENPM611 Software Engineering class, this project aimed to create a tool to analyze the Python Poetry repository's GitHub issues. The objective was to extract meaningful insights about contributor activity, issue trends, resolution patterns, and community interactions, visualizing these patterns using matplotlib. The analysis involved leveraging the GitHub API for data extraction, designing the application structure, implementing analysis features, and presenting findings.

1.3 Success Criteria

The project's success criteria included delivering all milestones on time, successfully extracting issue data, presenting actionable insights, and achieving at least 90% unit test coverage for the implemented code.

2. Design (Milestone 1)

2.1 Domain Model (ERD)

An Entity Relationship Diagram (ERD) was created using Mermaid.js to model the application's domain. Key entities identified include ISSUE, EVENT, USER, LABEL, and REACTION. Relationships modeled include an ISSUE having many EVENTs, being created and closed by a USER, tagged with LABELs, and having REACTIONs. The diagram specification is available in EntityRelationshipDiagrams/team_5_erd.txt.

2.2 Application Design (Class Diagram)

A Class Diagram, also created using Mermaid.js, outlines the application's structure, modeling Python modules and classes. Key classes include Issue, Event, User, Label, Comment, Reaction, Config, DataLoader, WorkspaceIssues, and Analysis. The diagram shows dependencies, such as the Analysis class utilizing Issue data loaded by DataLoader. The diagram specification can be found in ClassDiagram/team_5_class_diagram.txt.

3. Implementation (Milestone 2)

3.1 Approach & Tools

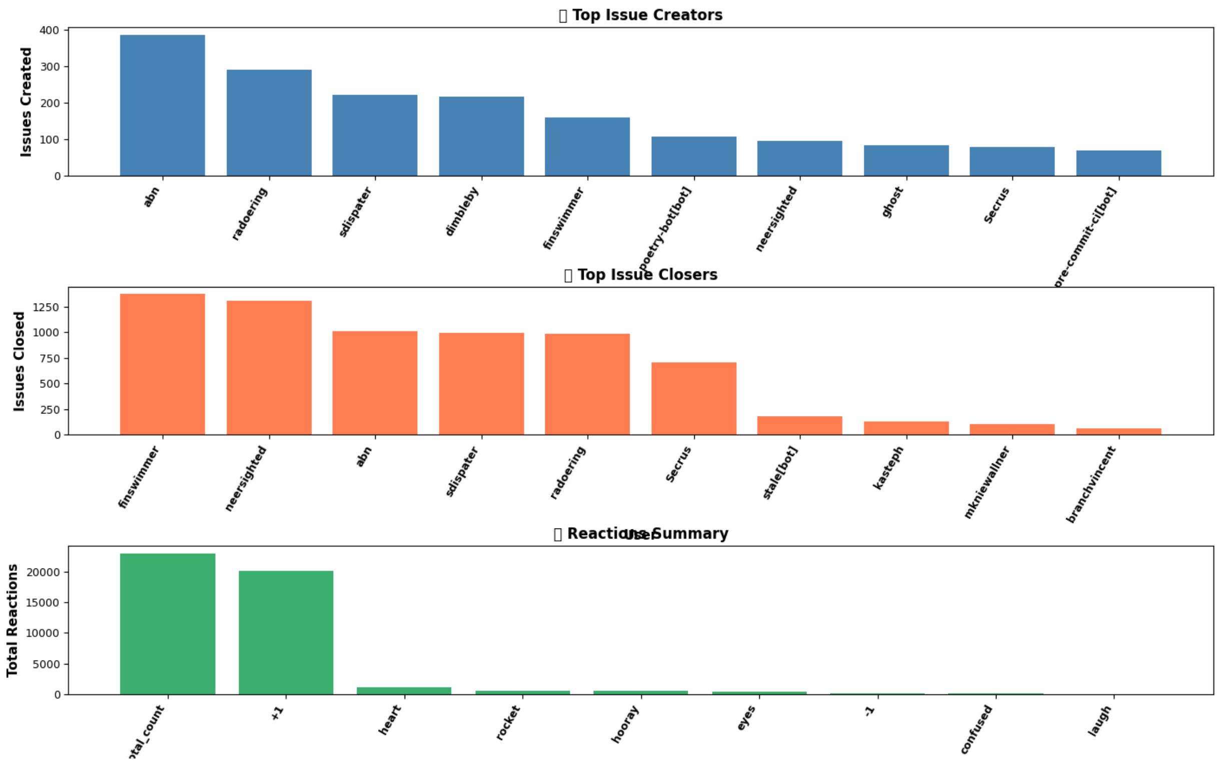
The team used the GitHub API, accessed via Python's requests library and requiring a GITHUB_TOKEN, to fetch issue and event data from the Poetry repository (scripts/fetch_issues.py). This data was stored in data/poetry_data.json. The analysis features were implemented in Python, utilizing libraries such as pandas for data manipulation, matplotlib for generating visualizations, and python-dateutil for date parsing. Configuration was managed via config.py and config.json. The application structure includes modules for each analysis (analysis/contributorAndReactionAnalysis.py,analysis/frequentLabelAndResolutionTimeAnalysis.py, analysis/labelBasedDeepDiveAnalysis.py) invoked by run.py.

3.2 Implemented Features

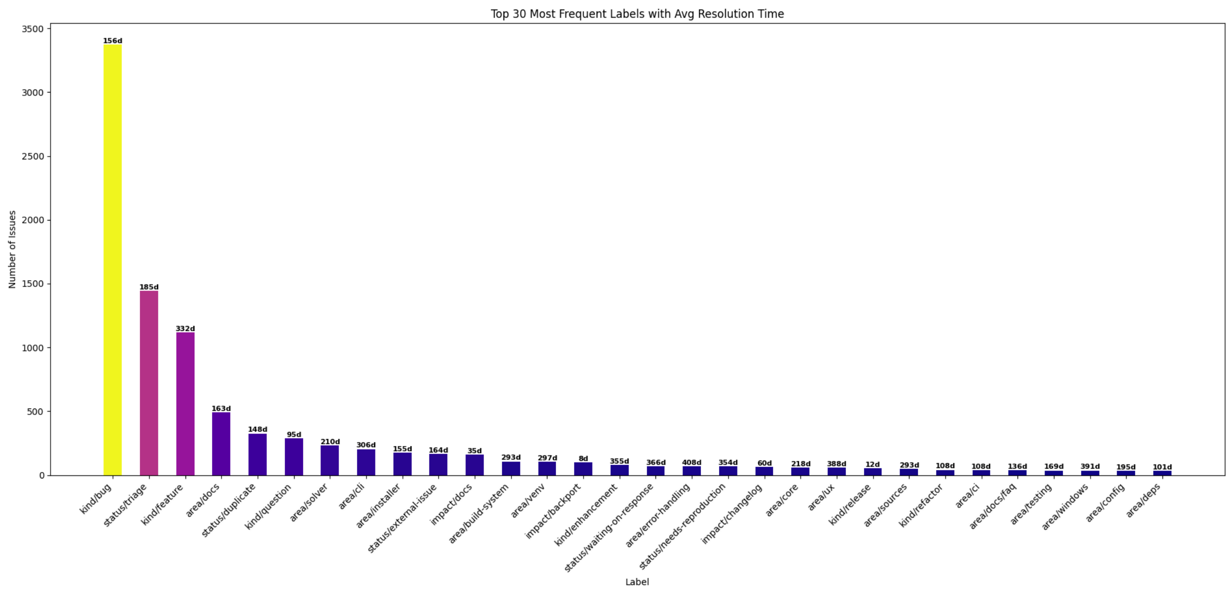
Three distinct analysis features were developed:

- **Feature 1: Contributor and Reaction Analysis (Analysis/analysisOne.py)**
    - Identifies and visualizes the top issue creators and top issue closers using bar charts.
    - Summarizes and plots the total count for each type of emoji reaction across all issues.

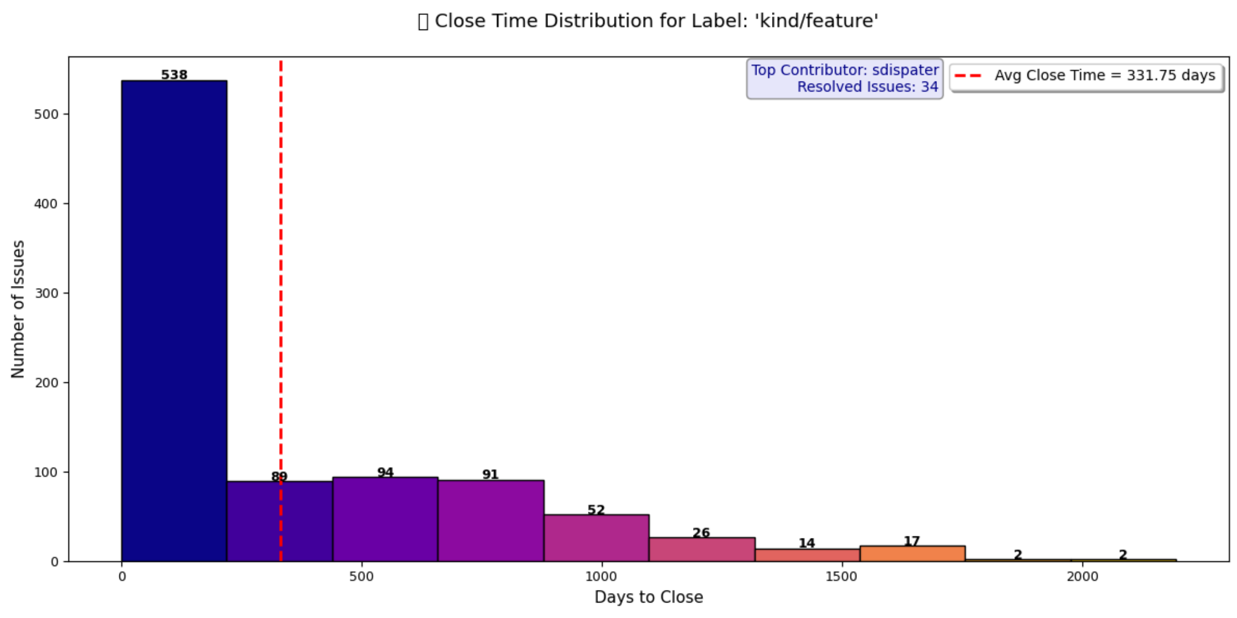
- *Key Findings:* Highlighted key contributors like 'abn' and 'raddening' for creation, 'finswimmer' and 'neersighted' for closing, and showed strong community engagement through '+1' and 'heart' reactions.



- **Feature 2: Label Frequency and Resolution Time (Analysis/ analysisTwo.py)**
  - Analyzes the frequency of the top 30 most used labels across issues.
  - Calculates and visualizes the average resolution time (in days) for issues associated with each label, annotating these times on a bar chart showing issue counts per label.
  - *Key Findings:* Identified high-volume labels ('bug', 'dependencies', 'status/triage') and labels potentially indicating complexity or needing more attention due to longer resolution times. This aids in understanding workflow patterns.



- **Feature 3: Label-Based Deep Dive (`Analysis/labelBasedDeepDiveAnalysis.py`)**
  - Accepts a specific label as user input via the command line (`--label`).
  - Calculates and prints metrics for the given label: total issues, average comments, average time to close, and the most active contributor (by issues created/resolved).
  - Generates a histogram showing the distribution of days taken to close issues tagged with the specified label, including an average close time indicator and an annotation for the top contributor.
  - *Key Findings:* Helps analyze workflow consistency and identify potential bottlenecks or leadership patterns for specific issue types (e.g., 'kind/feature').



3.3 Running the Application

1. Setup: Clone the repository, create a virtual environment, and install dependencies: `pip install -r requirements.txt`. Ensure `config.json` contains a valid `GITHUB_TOKEN`.
2. Data Fetching (Optional): Run `python scripts/fetch_issues.py` to update `data/poetry_data.json`.
3. Run Analyses: Use the `run.py` script with the `--feature` flag:
   - `python run.py --feature 1` (Contributor/Reaction Analysis)
   - `python run.py --feature 2` (Label Frequency/Resolution Time)
   - `python run.py --feature 3 --label <label_name>` (Label Deep Dive, e.g., `--label kind/feature`)

4. Testing (Milestone 3)

4.1 Strategy & Goal

The project plan included writing unit tests to ensure code robustness and achieve at least 90% statement coverage. The assignment involved potentially testing another team's code and submitting test results, coverage reports, and a link to a Pull Request containing the tests.

4.2 Results

(Note: The uploaded files did not include specific test reports, coverage results, or details on bugs found. Therefore, the final coverage percentage achieved and specific findings from the testing phase cannot be reported here. This information would typically be found in files like team_<team_number>_test_failures.txt and team_<team_number>_test_coverage.txt as per the assignment PDF guidelines.)

5. Conclusion

This project successfully developed and implemented a tool for analyzing GitHub issues from the Python Poetry repository. Three distinct analysis features were created, providing valuable insights into contributor engagement, issue labeling trends, resolution times, and community reactions, presented through command-line outputs and `matplotlib` visualizations. The project adhered to software engineering practices, including design documentation (ERD, Class Diagram) and modular implementation. While final testing metrics are not available from the provided files, the implemented features fulfill the core objectives of the assignment.