# Developer Performance Dashboard: Analysis Report

## 1. Introduction

### Objective

The objective of this project was to develop a Streamlit-based dashboard that provides insights into developer performance using data fetched from an open-source GitHub repository. The dashboard is designed to offer a comprehensive view of various developer metrics including commit frequency, pull requests (PRs), issue resolution, and code review participation. This proof of concept (POC) focuses on analyzing data from a single repository with an extension planned for multiple repositories in the future.

### Scope

- **Single Repository Analysis**: By default, data collection pulls the last 5000 commits for the demonstration, but this limit can be dynamically adjusted to collect more commits based on user preferences.
- **Extendable**: The system is scalable and can be extended to include multiple repositories for a broader analysis.

## 2. Data Collection & Metrics

### 2.1 Data Points Collected

We used the GitHub API to pull the following data points from the repository:

- **Commits**: Frequency and lines of code added/deleted.
- **Pull Requests (PRs)**: Number of PRs, PR review time, PR size, and merge rate.
- **Issues**: Number of issues resolved and average time taken to resolve issues.
- **Code Reviews**: Number of reviews conducted and the depth of the reviews (comments per review).

### 2.2 Commit Data

For this demo, we focused on the last 5,000 commits from the repository. These include:

- **Author Information**: The name of the contributor who made the commit.

- **Date of Commit**: The timestamp of the commit.
- **Code Churn**: The number of lines added and deleted in each commit.

### 2.3 Metrics Calculated

The following key metrics were extracted and analyzed:

- **Commit Metrics**: Frequency of commits per developer and the extent of code churn (lines added/deleted).
- **PR Metrics**: PR size distribution, PR merge rates, and review times.
- **Issue Metrics**: The number of issues resolved and average issue resolution time.
- **Cycle Time Metrics**: The time taken from issue creation to PR creation and from PR creation to merge.
- **Code Review Metrics**: Number of reviews and the average review depth.

## 3. Dashboard Components

### 3.1 Overview Page

The overview page offers a high-level summary of the repository's key performance metrics. It allows the user to quickly view:

- Total commits
- Number of PRs opened
- Resolved issues
- Trends in developer activity over time

### 3.2 Individual Developer Page

This page focuses on providing detailed insights for individual developers. Metrics include:

- Commit frequency
- Code churn
- PRs opened, merged, and their respective sizes
- Issues resolved and their average resolution times

### 3.3 PR Analysis Page

The PR analysis page provides insights into:

- The size distribution of PRs
- Merge rates per developer
- PR review times

### 3.4 Issue Tracking Page

This page highlights trends in issue resolution across developers. It visualizes:

- Number of issues resolved over time
- Average time taken by developers to resolve issues

### 3.5 Code Review Insights

Detailed analysis of code reviews conducted by each developer, including the average review depth (number of comments per review).

## 4. Technical Architecture

### 4.1 Technology Stack

- **Frontend**: Streamlit for creating an interactive, user-friendly dashboard.
- **Backend**: FastAPI to manage data extraction and API calls.
- **Data Management**: Pandas for data manipulation.
- **Visualizations**: Plotly for interactive data visualizations.
- **GitHub Integration**: PyGithub to interface with the GitHub API and pull relevant repository data.

### 4.2 Data Storage

- For the demo phase, CSV files were used to store the extracted data for easier handling and processing.

## 5. Results

### 5.1 Key Observations

- **Commit Frequency**: Developers showed varying commit frequencies, with some contributing significantly more than others.
- **PR Analysis**: The majority of pull requests had smaller sizes, but a few had disproportionately large numbers of code changes, leading to extended review times.
- **Issue Resolution**: The time taken to resolve issues varied greatly, with a few developers showing significantly higher efficiency in issue resolution.

### 5.2 Performance

- The dashboard performs efficiently, loading data from up to 10,000 commits within 2 seconds on a standard development machine.

## 6. Recommendations

- **Extend to Multiple Repositories**: Future iterations of the dashboard could integrate data from multiple repositories to provide cross-repository comparisons of developer performance.
- **Enhance Natural Language Querying**: The current system supports natural language queries for fetching data, but further refinement in query processing would enhance user experience.
- **Automated Data Updates**: Implementing automated daily data updates would ensure that the dashboard reflects the latest performance metrics.

## 7. Conclusion

This project successfully demonstrated the ability to gather and analyze developer performance metrics from an open-source GitHub repository. The dashboard offers valuable insights into individual developer contributions, PR lifecycles, and issue resolution trends, which can aid project managers and team leads in assessing team efficiency and productivity.

## Appendices

- **Sample Queries for Natural Language Interface**:
  - "Who made the most commits last month?"
  - "What’s the average PR size in the repository?"
  - "Show me the most active developer in terms of PR reviews."