Indian Institute of Technology Gandhinagar

Project Report

CS 434: Software Engineering and Testing

Submitted By: Aditi Dey (20110007)

Faculty Guide: Prof. Shouvick Mondal

INDEX

INTRODUCTION

- In the age of big data, organisations rely significantly on a variety of tools and frameworks to process, analyse, and extract insights from huge amounts of data. Apache Kafka, Presto, Apache Flink, Apache NiFi, and Apache Commons IO are some of the most popular big data solutions, each with its own set of capabilities and strengths. In this project, we will do a comparative analysis of these tools to determine their features, performance characteristics, and applicability for various use cases.

- **Objectives**
    1. We will analyze the complexity of the codebases of Apache Kafka, Presto, Apache Flink, Apache NiFi, and Apache Commons IO using Lizard.
    2. Metrics such as cyclomatic complexity, lines of code, and code duplication will be assessed to understand the inherent complexity of each codebase.
    3. We will calculate the maintainability index for each codebase using Lizard, providing a quantitative measure of how maintainable and understandable the code is.

GITHUB REPOS STUDIED

The following big data processing Github repos were cosen for the project.

1. Apache Kafka:
    - Repository: https://github.com/apache/kafka
    - Number of Stars: 27.4k
    - Language: Java
    - Description: Apache Kafka is a distributed event streaming platform used for building real-time data pipelines and streaming applications. It provides a scalable and fault-tolerant architecture for publishing and processing data streams. Kafka is widely used for building real-time analytics, log aggregation, and messaging systems.
2. Presto:
    - Repository: https://github.com/prestodb/presto
    - Number of Stars: 15.6k
    - Language: Java
    - Description: Presto is a distributed SQL query engine optimized for interactive analytics. It allows users to query large-scale datasets in real-time using SQL syntax, supporting various data sources such as Hadoop, Amazon S3, MySQL, and more. Presto is commonly used for ad-hoc querying, interactive analysis, and business intelligence applications.
3. Apache Flink:
    - Repository: https://github.com/apache/flink
    - Number of Stars: 23.2k
    - Language: Java
    - Description: Apache Flink is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications. It supports event-driven applications, batch processing, and iterative algorithms. Flink provides powerful APIs and libraries for building real-time analytics, stream processing, and batch processing applications.

4. Apache NiFi:
    - Repository: https://github.com/apache/nifi
    - Number of Stars: 4.4k
    - Language: Java
    - Description: Apache NiFi is an easy-to-use, powerful, and reliable system to process and distribute data. It provides a graphical user interface for designing data flows and a scalable architecture for managing data ingestion, transformation, and routing. NiFi is commonly used for data integration, data routing, and real-time data processing.
5. Apache Commons IO:
    - Repository: https://github.com/apache/commons-io
    - Number of Stars: 984
    - Language: Java
    - Description: Apache Commons IO is a library for working with input/output (I/O) operations in Java. It provides utility classes and methods for reading, writing, and manipulating files, streams, and other I/O resources. Commons IO is widely used in Java applications for handling file operations, stream processing, and data manipulation tasks.

STATIC ANALYSIS TOOL USED

1. Lizard is a versatile software tool designed for analyzing the complexity of source code in various programming languages. By providing insights into code structure and complexity metrics, Lizard aids developers in understanding the maintainability and readability of their codebases.

2. Key Features:
    - Language Support: Lizard supports multiple programming languages, including C, C++, Java, Python, and JavaScript, making it applicable to a wide range of projects. This project will be using it for Java.
    - Metrics Calculated: Lizard calculates essential code complexity metrics, such as cyclomatic complexity, lines of code (LOC), code duplication, and nesting depth.
        i. **Total nloc**: This represents the total number of lines of code (LOC) in the analyzed source code files. It includes both comment lines and code lines.
        ii. **Avg.NLOC** (Average NLOC): This metric calculates the average number of lines of code per function or method in the source code. It provides an indication of the size or complexity of individual functions or methods.
        iii. **AvgCCN** (Average Cyclomatic Complexity Number): Cyclomatic complexity is a software metric used to measure the complexity of a program's control flow. It represents the number of independent paths through a program's source code. The average cyclomatic complexity calculates the average complexity across all functions or methods in the source code.
        iv. **Avg.token** (Average Tokens): This metric calculates the average number of tokens per line of code in the source files. Tokens include keywords, identifiers, operators, and punctuation symbols.
        v. **Fun Cnt** (Function Count): This metric counts the total number of functions or methods in the source code files. It provides an indication of the modularity and structure of the codebase.
        vi. **Warning cnt** (Warning Count): This represents the total number of warnings generated by the static code analysis tool. Warnings may indicate potential issues or areas of improvement in the source code.

vii. **Fun Rt** (Function Ratio): This metric calculates the ratio of the number of functions or methods to the total number of lines of code (Total nloc). It provides insight into the distribution of functions or methods relative to the overall size of the codebase.

viii. **nloc Rt** (NLOC Ratio): Similar to the Function Ratio, this metric calculates the ratio of the total number of lines of code to the number of functions or methods. It offers another perspective on the relationship between code size and modularity.

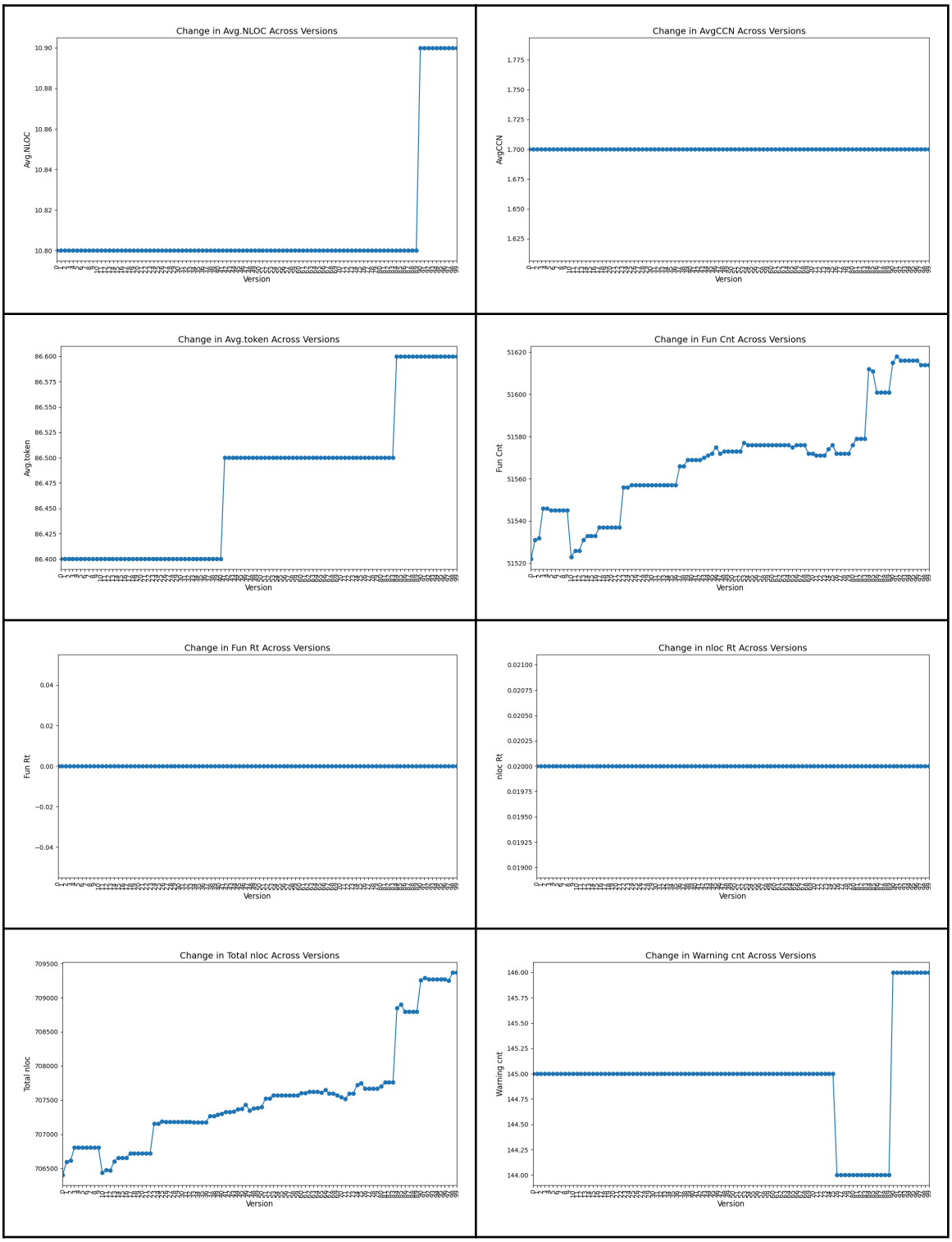
3. Output Formats: Lizard generates reports in plain text, XML, and JSON formats, facilitating integration with existing development workflows and tools. In this project, the reports have been generated in a text format.

IMPORTANT NOTES

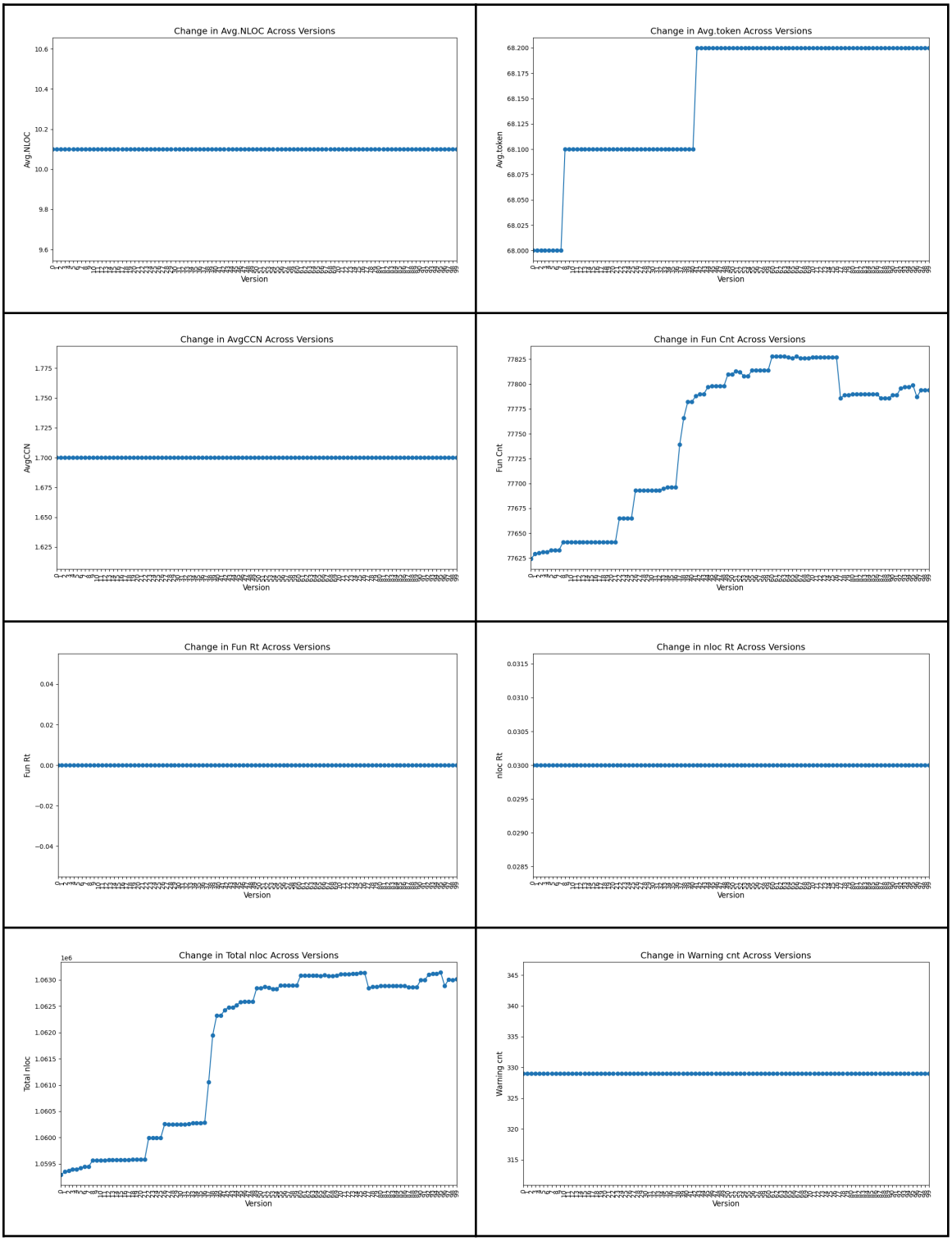
1. SET-IITGN-VM has been used for the project.
2. There were multiple Python versions available in the VM, and pydriller wasn't working in some of them, so I found it better to create my own virtual environment to do the setup.
3. I created a new file called my_deleter.sh to delete all the files except the lizard.txt file from all the different versions. Github
4. The repos chosen were all large-scale, real-world open-source software projects from GitHub. The problem was that some of them were too large to fit into my system. So Apache Flink was analysed only till version 57, and Apache Nifi was only analysed till version 93.
5. The plots for various matrices have been plotted and analyzed in the next sections. The code for analysis has been uploaded on Github.

RESULTS

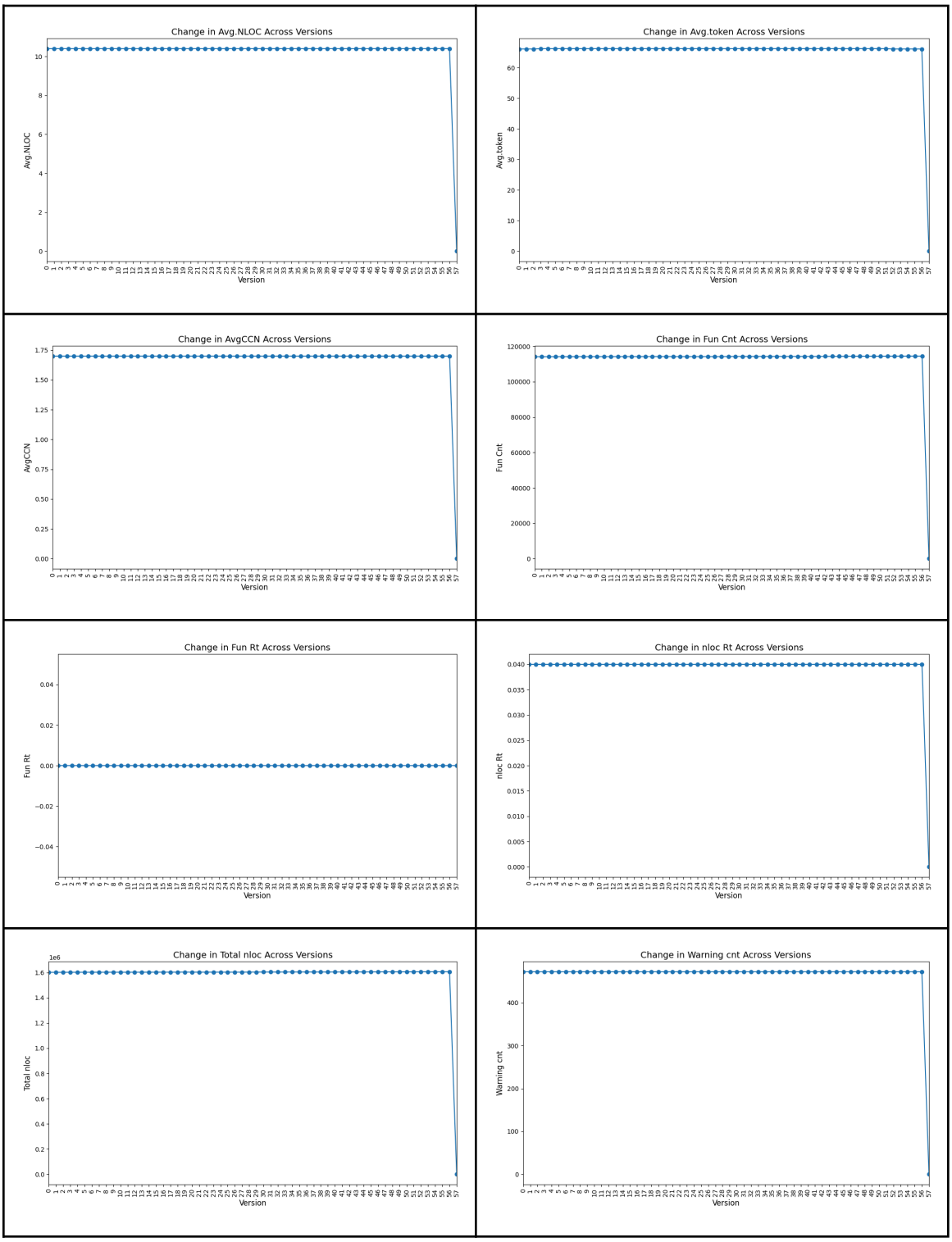
1. Apache Kafka:



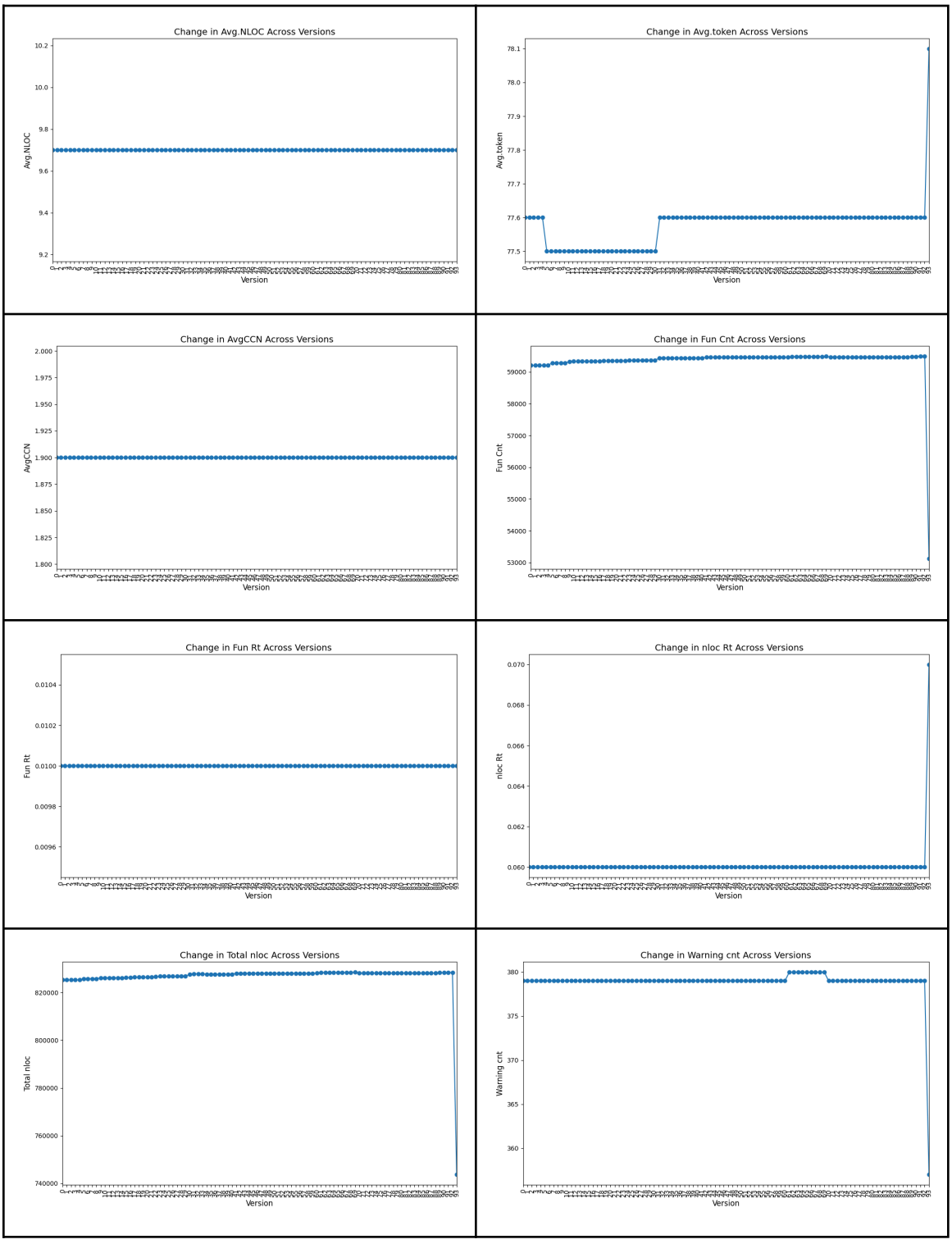
2. Presto:


Change in Avg.NLOC Across Versions
Avg.NLOC
Version
Change in Avg.token Across Versions
Avg.token
Version
Change in AvgCCN Across Versions
AvgCCN
Version
Change in Fun Cnt Across Versions
Fun Cnt
Version
Change in Fun Rt Across Versions
Fun Rt
Version
Change in nloc Rt Across Versions
nloc Rt
Version
Change in Total nloc Across Versions
1e6
Total nloc
Version
Change in Warning cnt Across Versions
Warning cnt
Version

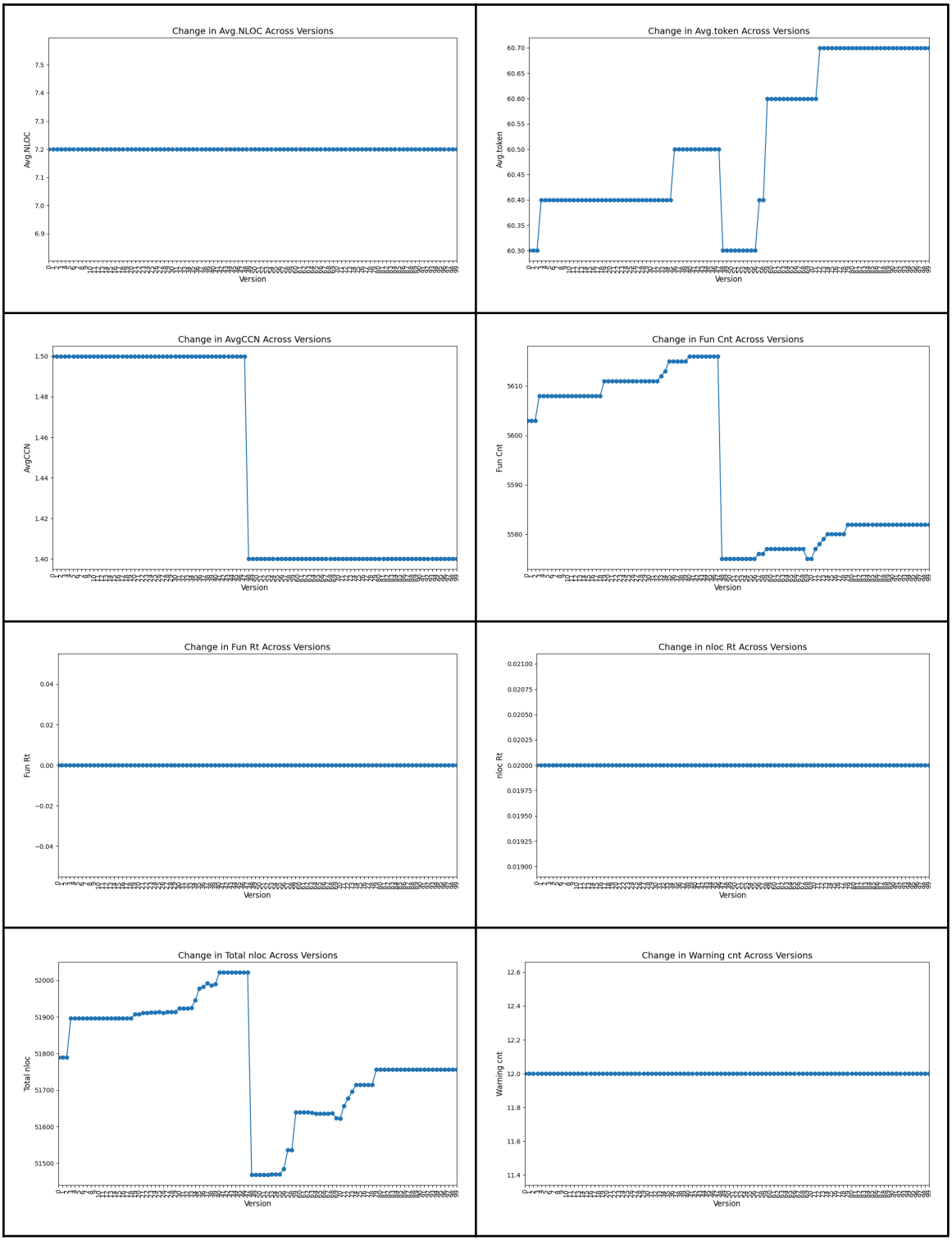
3. Apache Flink:



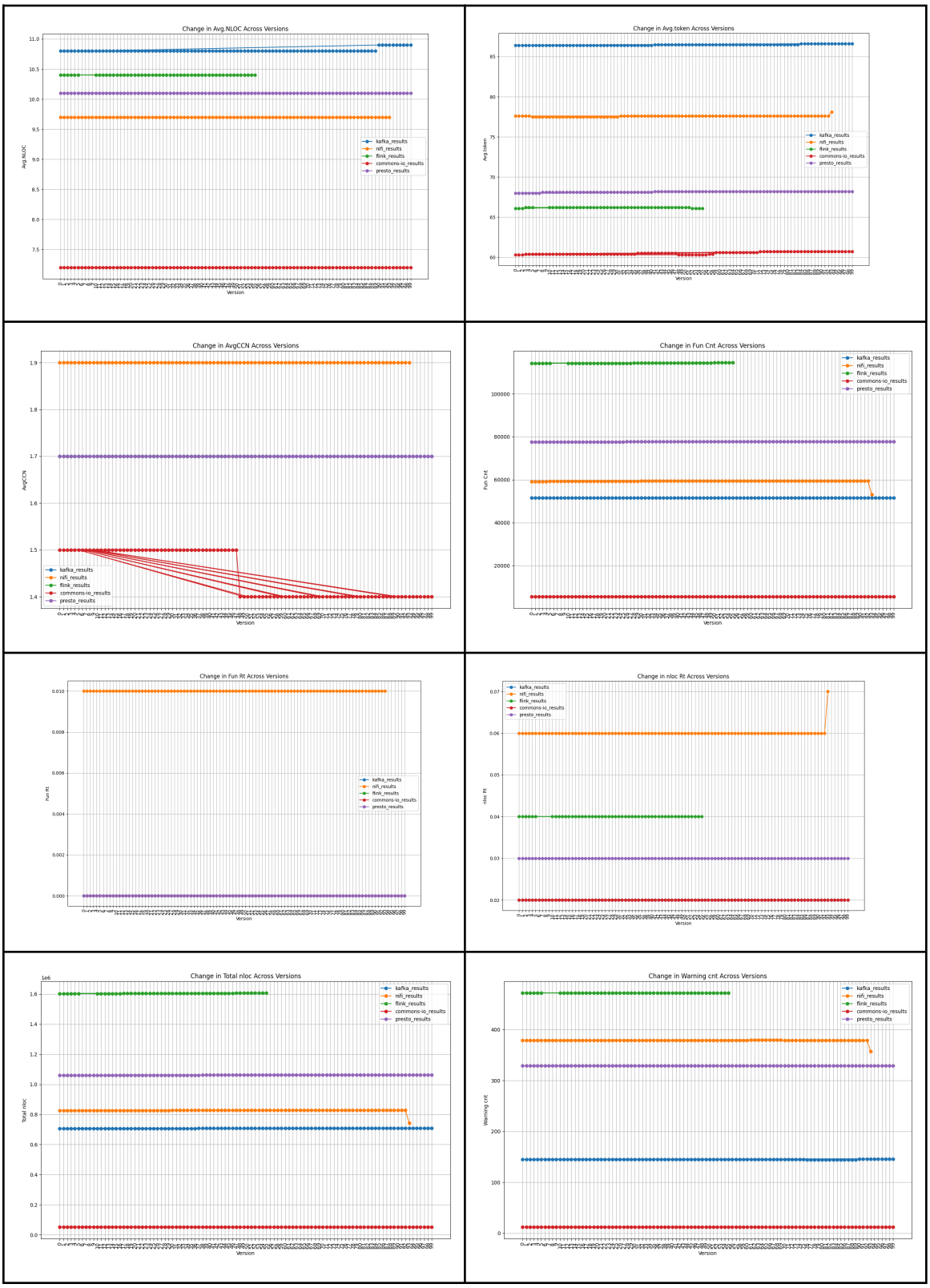
4. Apache NiFi:



5. Apache Commons IO



Comparison Across Different Repos



DATA ANALYSIS

1. We can see that Kafka has the highest number of lines of code, followed by Flink, Nifi, Preso, and Commons-io.
2. We can see that Kafka has the highest average number of tokens, followed by Nifi, Preso, Flink, and Commons-io.
3. AvgCCN (Cyclomatic Complexity) of Presto, Kafka and Flink are very close to each other, hence these lines overlap and we only see 3 separate line when we plot all 5 of them together.
4. The number of functions is maximum in Flink, followed by Presto, Nifi, Kafka, and Commons-io.
5. nloc RT of Kafka and Nifi are almost same, therefore we see only 4 lines in the nloc RT plot.
6. Function Ratio of Kafka, Presto and Flink are almost zero! Common-io and Nifi both have a Function Ratio of 0.01.