Refactored Code Summary

The refactoring is done on the HW1\_Data\_Analyzer part. The refactored code addresses several code smells, primarily focusing on long methods and **monolithic structures**. By breaking down the main logic into smaller, more focused methods (such as getUserInput, readDataFromCSV, and writeAnalysisToFile), the code becomes easier to understand and maintain. This **modularization** also helps in reducing duplication and making each method responsible for a single task, which aligns with the **Single Responsibility Principle** (SRP). Additionally, the refactoring improves the **readability** of the code, as each method now has a clear and descriptive purpose, making it more approachable for future developers.

Moreover, this approach adheres to the **Open/Closed Principle** (OCP), a key concept from SOLID design principles. The code is now more extendable without modifying existing functionality. For instance, if new features or analysis methods need to be added (such as calculating standard deviation or other statistical measures), they can be integrated by simply adding new methods or classes without altering the existing ones. This extensibility ensures that the code is open to future enhancements while being closed to modifications, which minimizes the risk of introducing bugs or regressions.

The refactoring not only addresses key code smells but also helps increase test coverage. By modularizing the code into smaller, more focused methods, it becomes easier to write targeted unit tests for each individual component. For instance, methods like calculateMean, calculateMedian, and readDataFromCSV can now be independently tested, ensuring that each piece of functionality works as expected in isolation. This reduces the likelihood of errors and increases the overall reliability of the application

Overall, the refactored code is now more **maintainable, flexible, and scalable**, making it easier to add new functionalities as the application evolves.