Paper SM01

A Git-Driven Automated Workflow for Efficient ADaM and TFL Validation and Quality Control Documentation

Xavier Chénedé, Sanofi, Vitry-sur-Seine, France
Paulin Charliquart, Sanofi, Stockholm, Sweden

ABSTRACT

Traditional quality control (QC) processes for Analysis Data Model (ADaM) and Tables, Listings, & Figures (TLF) programming rely on manually filling Excel templates for planning and tracking program validation. We propose an evolution towards an automated end-to-end approach leveraging R, Git-based code development, workflow automation, AI-enhanced code review, and HTML dashboards.

Our framework integrates multiple complementary quality control components: programmer-developed unit tests for code validation, AI-powered reviews utilizing Large Language Models (LLM) fine-tuned by internal standards, and a comprehensive HTML tracker that consolidates all quality checks. This integrated approach significantly reduces manual interventions, minimizes human error, and accelerates turnaround time from data availability to delivery, while maintaining the rigorous quality standards essential to our work.

While demonstrating substantial benefits, this work represents a first step towards a more comprehensive QC framework. Future enhancements include integrating logs and SDTM data, implementing sophisticated dataset comparisons, and developing an editable standalone tracker. Challenges include technical constraints in secure environments, adapting to new methodologies, and maintaining a critical perspective on LLM implementation.

This innovative approach sets the foundation for optimizing quality control processes, contributing to operational excellence while navigating the complexities of integrating advanced technologies in a regulated environment.

Keywords: ADAM and TLF Quality Control Automation, Git, Unit Testing, LLM, Workflow Integration.

INTRODUCTION

In the field of statistical programming for ADaM and TLF analyses, traditional quality control processes often rely on time-consuming and error-prone manual methods. These approaches, typically based on Excel spreadsheets for planning and tracking program validation, no longer meet the efficiency and precision requirements of the modern pharmaceutical industry.

Facing these challenges, this document proposes an evolution in QC methodology for statistical programming. The innovative approach aims to fully automate the deployment of ADaM & TLF validation and quality control documentation. This next-generation solution leverages a suite of cutting-edge technologies:

- R programming for robust and reproducible statistical analysis
- Git-based code development for efficient version control
- Workflow automation via GitHub Actions for continuous integration and deployment
- AI-assisted code review utilizing advanced LLM
- Interactive HTML dashboards for real-time monitoring and clear visualization of quality metrics

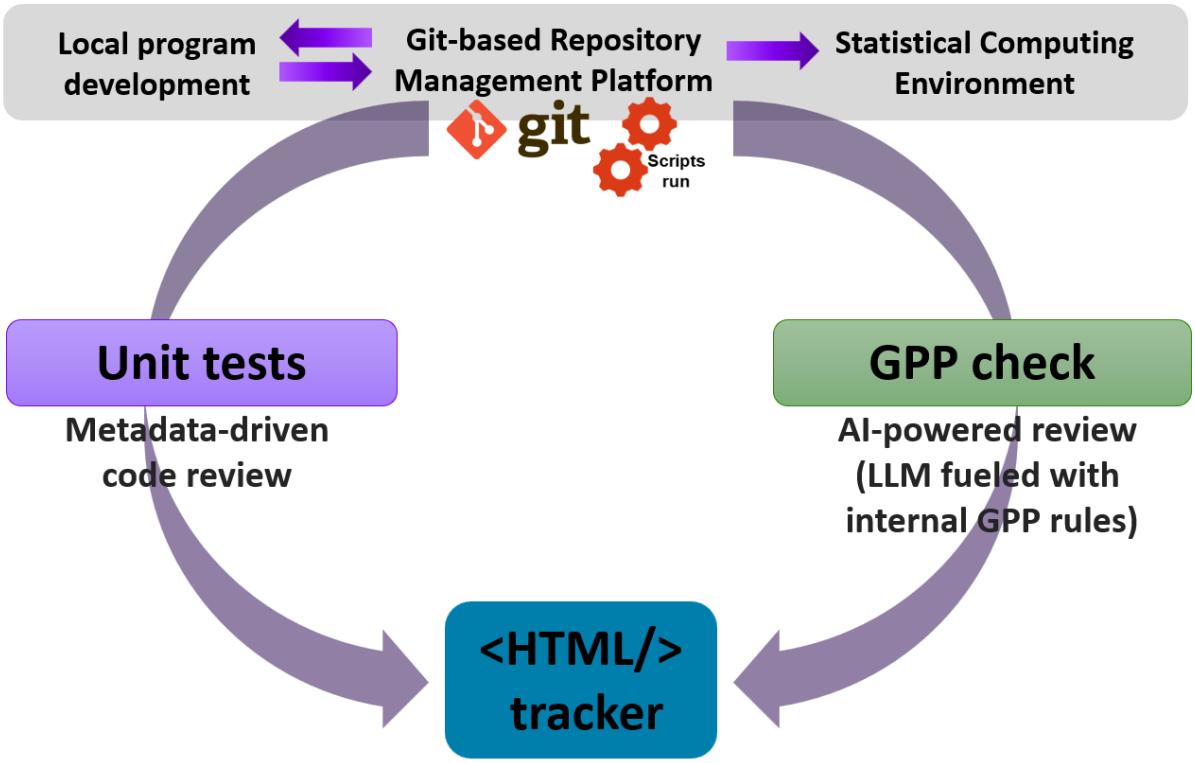
This integrated approach promises not only to significantly reduce the time and resources required for QC processes, but also to improve the accuracy, consistency, and traceability of validations. By automating repetitive tasks and leveraging artificial intelligence for code review, statistical programmers can be freed from tedious manual work, allowing them to focus on more critical and creative aspects of their responsibilities.

In this document, the overall concept and architecture of the solution are first presented. A detailed exploration of the process implementation follows, covering the overall workflow and development process, unit testing strategy and implementation, Good Programming Practice (GPP) automation with LLM, and the approach to final QC reporting and dashboard generation.

CONCEPT

The system is built around a Git-based repository, integrated with a management platform where local program development takes place within a statistical computing environment (SCE).

Metadata serves as foundation for code reviews associated with unit testing. Simultaneously, artificial intelligence plays a crucial role in this process, utilizing Large Language Model fine-tuned on internal Good Programming Practice rules. This model enables the automation of GPP-compliant code generation and review.



The entire process performance and compliance metrics are monitored and visualized through an interactive HTML-based dashboard, allowing teams to track progress and identify improvement opportunities.

PROCESS

OVERALL

The development process follows a structured approach to ensure quality and efficiency:

First, datasets and Tables, Listings, and Figures are built as the foundation of the analysis. This step is critical for establishing the data structure that will support all subsequent analyses.

Next, unit tests are developed locally to validate the code. These tests are designed according to the QC plan and focus on functions specifically developed for the study, ensuring that all custom components work as intended.

Once testing is complete, code is pushed and automated workflows are triggered. This process includes style checking to maintain code consistency and execution of all tests to verify functionality. After successful validation, the code is merged with the main branch, integrating the work with the project codebase.

Finally, the validated code is deployed on the Statistical Computing Environment (SCE). This deployment involves copying files to the target environment, running the necessary scripts, performing final checks, and generating a tracker to monitor the execution status and results.

UNIT TESTING

QC plan strategy followed a risk-based approach. Once QC plan is defined (see ADaM example in table below), unit tests can be implemented.

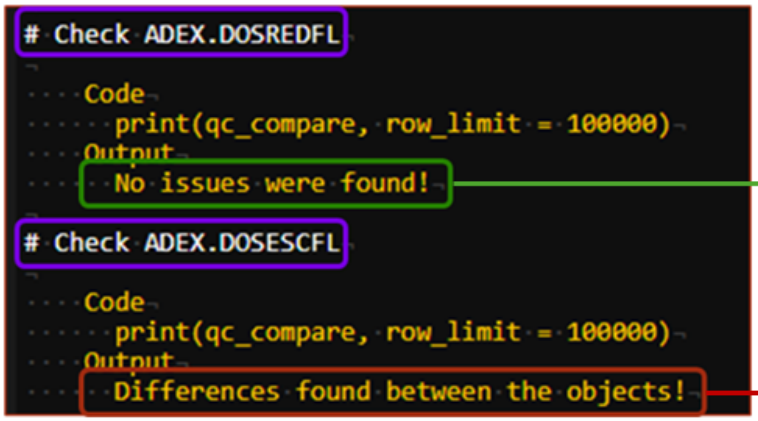
Dataset	Variables
ADEX	DOSESCFL
ADEX	DOSREDFL

QC programs use {testthat} package and follows specific conventions (see R script below):

1. Name the test as **"Check Dataset.Variable"**, for merge purpose
2. Complete the code with the variable derivation
3. Compare the derivation with the reference data
4. Capture of the comparison result in markdown file
5. Save file as test-dataset.R (ex: test-adex.R)

```
testthat::test_that("Check Dataset.Variable", { #e.g., ADEX.DOSREDFL
        ## derivations----
        qc <- #write derivation
        ## base dataset ----
        base <- #capture reference data
        ## Comparison with base ----
        qc_compare <-
                diffdf::diffdf(
                        base = base,
                        compare = qc) %>%
                print(row_limit = 100000)
        ## Overall results----
        tf_result <- expect_false(diffdf::diffdf_has_issues(
                qc_compare))
        ## Snapshot ----
        local_edition(3)
        expect_snapshot(
                print(qc_compare, row_limit = 100000),
                cran=TRUE,
                variant = "../../../QC/OUTPUT/ADS_VALIDATION"
        )
})
```

After execution of the tests, markdown files are produced with the results. The results are captured and merged with QC plan to generate the tracking (see below).



Markdown file

Dataset	Variables	Results
ADEX	DOSESCFL	Test not passed
ADEX	DOSREDFL	Test passed

Tracking dataset

The same process is used for functions and TLF programs.

GOOD PROGRAMMING PRACTICE (GPP)

Automatic review of the GPP is performed using LLM like GitHub Copilot. The adoption of GitHub Copilot in our development workflow is justified by several key advantages:

- Model Availability: GitHub Copilot is seamlessly integrated into the GitHub platform, providing immediate access to AI-assisted coding within our existing development environment.
- Customization: The tool offers customization options, allowing it to be tailored to our specific coding standards and project requirements.
- Setup Complexity: GitHub Copilot boasts an easy setup process, minimizing the time and resources needed for implementation and onboarding.
- Performance: GitHub Copilot shows a relatively fast execution speed, enhancing overall productivity.
- Code Review: GitHub Copilot is supported by GitHub's robust code review features, facilitating a smooth integration into our existing review processes.

GitHub Copilot code review can be configured in the repository settings to be automatically requested when a Pull Request (PR) is opened. The review process can be tailored to follow internal GPP rules by providing additional instructions to GitHub Copilot through a dedicated Markdown file, copilot-instructions.md, located in the repository at .github/copilot-instructions.md. This file is fully customizable to suit project-specific requirements.

Example of instructions in the copilot-instructions.md file:

Identify and clearly highlight bad practices, including but not limited to:

- *Absence of header.*
- *Using print instead of base::stop, base::warning, base::message, or function from the CLI package.*
- *Using paste or paste0 instead of file.path for constructing file paths.*
- *Inefficient data manipulation (prefer dplyr and vectorized functions over explicit loops where appropriate).*
- *Unsafe usage of global variables or non-standard evaluation.*
- *Poor handling of missing values.*
- *Inconsistent or unclear variable naming.*
- *Lack of error handling or input validation.*
- *Use of deprecated functions or packages.*
- *Inefficient reading/writing of data (prefer readr or data.table over base functions for large datasets).*
- *Not adhering to R style guides (such as the tidyverse style guide).*

Highlight strengths and positive aspects of the code, not just weaknesses.

Two situations are considered:

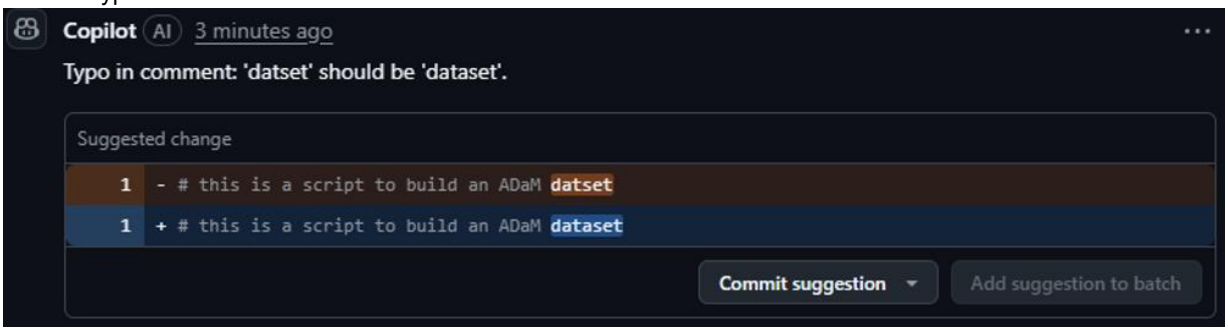
- Review following a Pull Request
- Full code review

PR Code review

When a Pull Request is opened, GitHub Copilot, acting as a reviewer, automatically examines the changes between the old and new versions. It summarizes its findings within the PR and suggests commits whenever possible. When Copilot suggests a commit, it prepares a set of changes that can be directly applied to the branch, allowing maintainers to quickly integrate improvements or fixes identified during the review.

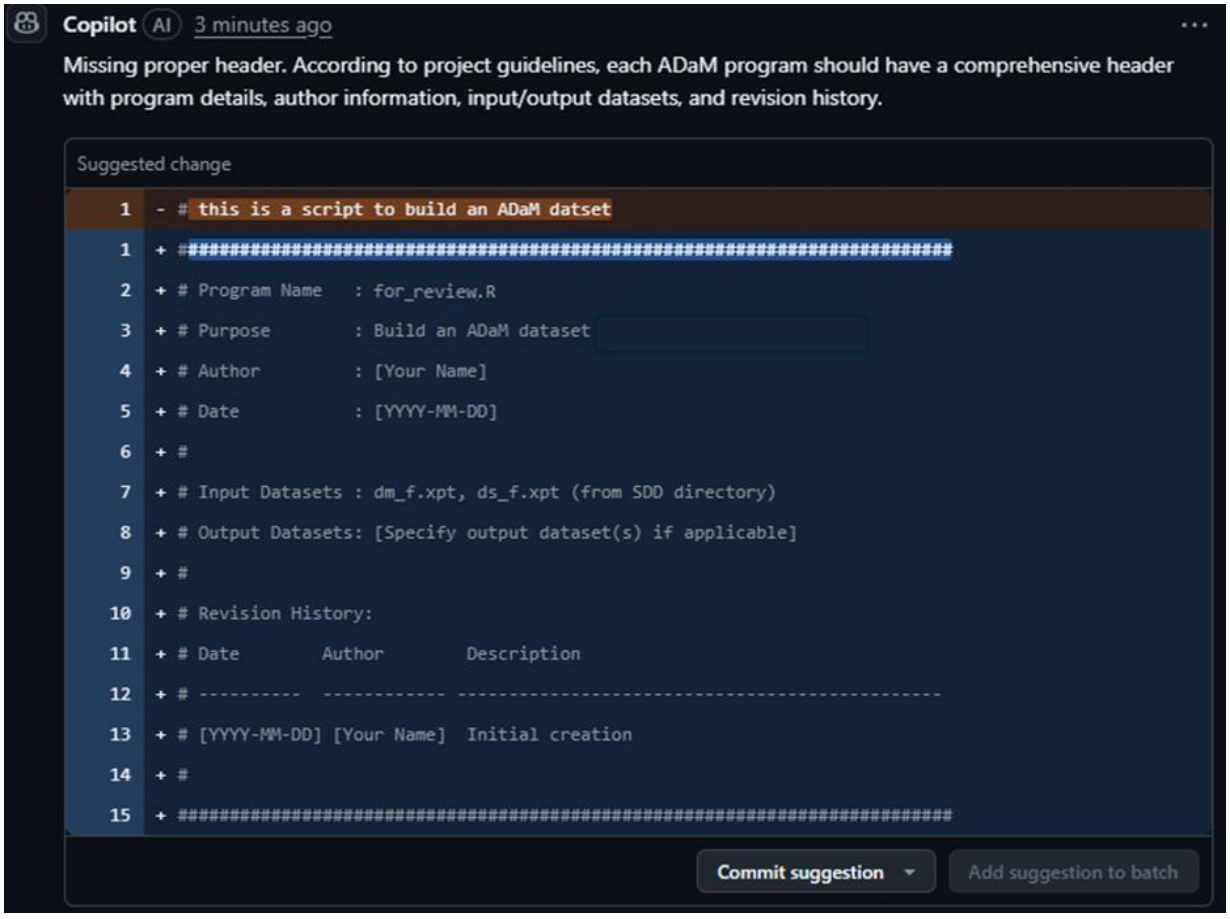
Some examples of GitHub Copilot abilities are presented below:

- Typo



- Specific rules

- Header

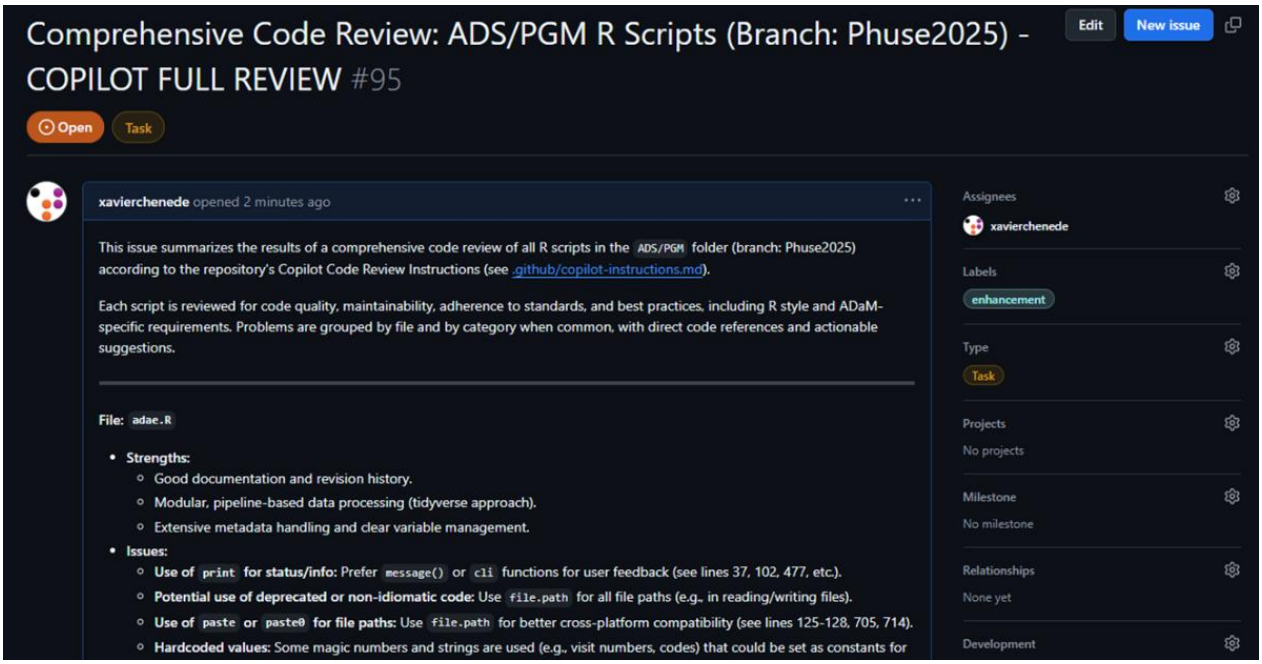


By automatically detecting and suggesting fixes for common coding issues, GitHub Copilot's PR review dramatically reduces the time investment required from human reviewers, allowing them to concentrate exclusively on critical aspects that demand human expertise.

Full Code review

A PR review does not allow the full review of the code. It is only focused on changes. For a full code review, the GitHub Copilot coding agent is a powerful tool to be used.
A coding agent is an AI-powered software assistant that actively helps programmers write, understand, and debug code by providing real-time suggestions, completing code blocks, and responding to natural language instructions during the development process.
The GitHub Copilot coding agent functions as an AI assistant with several advanced capabilities beyond basic code completion, and specifically with GitHub issue management features:

1. Automatically detects potential issues in code
2. Helps create detailed issue descriptions with context
3. Suggests issue titles and labels
4. Can include code snippets and error details in issues
5. Proposes potential solutions within issue descriptions
6. Customizable with specific/personal instructions (coming from the copilot-instructions.md for example)



FINAL QC REPORTING

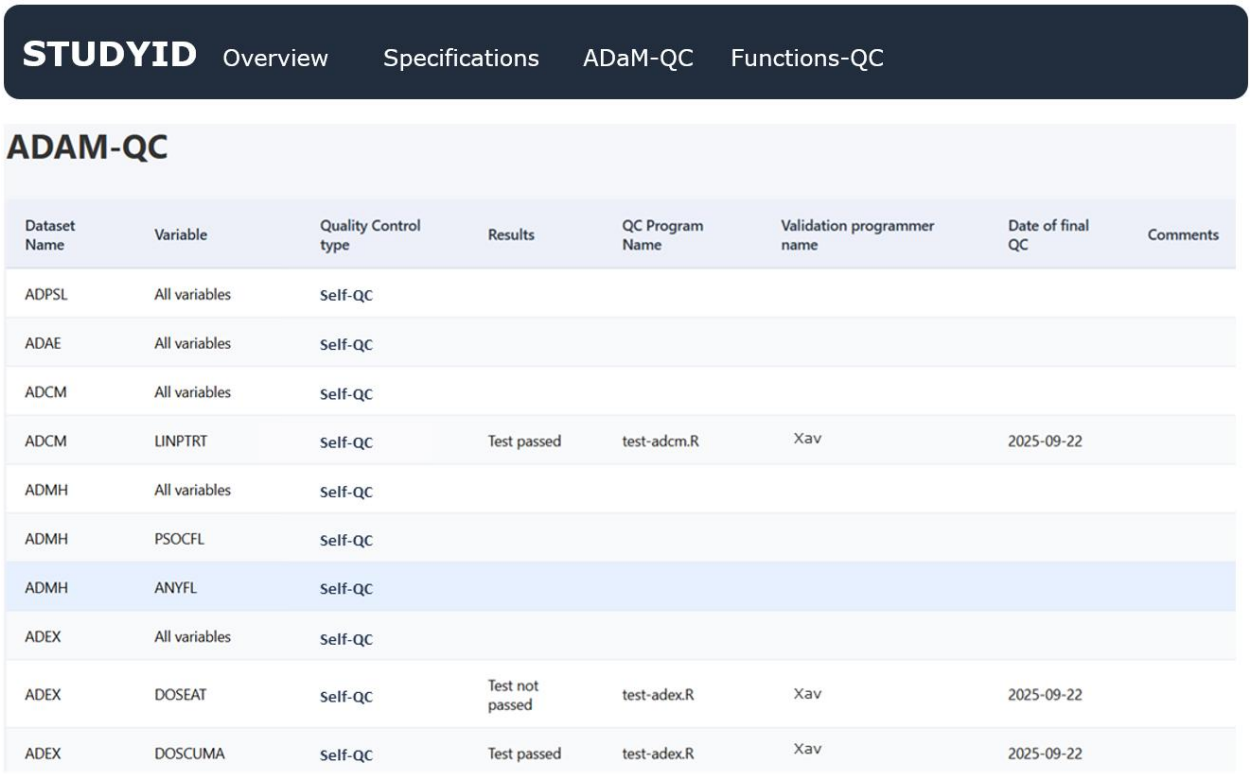
On pull requests, the following processes are initiated:

1. Unit tests are executed. This is a critical checkpoint - any failures in unit tests prevent the branch from being merged into the main branch.

2. Code review processes, assisted by GitHub Copilot, are engaged as previously described. Unlike unit tests, issues raised during code review do not automatically halt the workflow.

3. A QC tracker is generated as an HTML dashboard, consolidating all QC components.

At this stage, our QC tracker primarily focuses on the results of our unit tests (details provided below).



CONCLUSION

This work represents a significant first step toward a more comprehensive and integrated quality control framework. While the current implementation has demonstrated substantial benefits in reducing manual interventions, minimizing human error, and accelerating turnaround time from data availability to delivery, it maintains the rigorous quality standards essential in our industry.

The path forward includes several ambitious enhancements, particularly the integration of logs and SDTM data, implementation of sophisticated dataset comparison capabilities, and development of an editable standalone tracker. These improvements will further streamline the quality control process and enhance its robustness.

However, this journey is not without its challenges. The technical constraints regarding HTML and JavaScript usage in our secure environment necessitate careful architectural considerations. Additionally, the transition requires a shift in mindset among team members, encouraging adaptation to new methodologies and tools. Particularly noteworthy is the need for a judicious approach to Large Language Models, maintaining a critical perspective on their implementation and outputs.

Despite these challenges, the demonstrated workflow has already proven its value by significantly reducing repetitive manual tasks, minimizing the potential for human error, and substantially improving the efficiency of our delivery pipeline. These improvements have been achieved while maintaining, and in many cases enhancing, our quality standards. This foundation sets the stage for future developments that will further optimize our quality control processes and contribute to the overall excellence of our operations.

ACRONYMS

ADaM: Analysis Data Model
GPP: Good Programming Practice
LLM: Large Language Model
PR: Pull Request
QC: Quality Control
SCE: statistical computing environment
TLF: Tables, Listings, and Figures

ACKNOWLEDGMENTS

Sincere gratitude to the Paris Statistical Programming and Analysis Reporting Team for their valuable support and critical review of this work. Particular thanks are due to Alex Assuied, who not only helped initiate this topic but also demonstrated exceptional dedication throughout the project.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Names: Xavier Chénedé / Paulin Charliquart
Company: Sanofi R&D
Address: 1 impasse des Ateliers, 94 400 Vitry-sur-Seine, FRANCE
Emails: xavier.chenede@sanofi.com / paulin.charliquart@sanofi.com
Website: https://www.sanofi.com