Paper SM12

Automating Analysis Report Programming in SAS with VGSART: A Metadata-Driven Approach

Piyush Makkapati, Vita Global Sciences, Waltham, MA
Swathi Nattam, Vita Global Sciences, Waltham, MA
Bhargav Makkapati, Vita Global Sciences, Waltham, MA

ABSTRACT

The clinical programming industry is evolving, driven by the imperative to reduce costs and accelerate delivery timelines. To remain competitive, programming teams must go beyond and develop tools that drive efficiency ensuring consistency through innovation. One area consuming significant time is analysis output programming.

VGSART is a metadata-driven SAS framework designed to streamline analysis outputs. It uses structured CSV metadata for table layouts, analysis variables, formats, titles and footnotes, and a single macro generates outputs ranging from simple summaries to complex shift tables. A detailed user guide lets contributors without prior TFL experience create accurate tables.

In testing against QC-reviewed deliverables, VGSART cut programming time by 25% with fewer errors. The paper will cover framework design, adoption, macro flow, lessons learned, and outline plans to expand to additional output types. By using a metadata-driven approach, teams deliver results faster without sacrificing quality, strengthening stakeholders' trust and value.

INTRODUCTION

Analysis output programming is a time-consuming part of clinical programming that requires careful attention, despite its repetitive nature across studies. A typical package includes the same core table families each time, but programmers still need to apply the appropriate analysis flags and where conditions, enforce ordering and display rules, manage formats for consistent display, and update titles and footnotes as mock shells change. If any of these steps are applied incorrectly, the final table will not match the reviewer's expectations.

Teams often rely on copying programs forward to keep up with accelerated timelines. While this can save time initially, it is also a common way for preventable errors to slip into final deliverables. A missed where condition, an outdated format, or an incorrect header label carried over from a prior study can create inconsistencies that are not caught until late QC, when fixes are most disruptive.

VGSART was developed to shift effort away from rewriting the same output logic and toward reviewing outputs with confidence by generating tables from metadata rather than study-specific code edits. This paper describes how the framework is used in practice, including the macro flow from metadata to final output, the functionality implemented to date, and the testing approach used to compare results against QC-reviewed deliverables. It also summarizes key challenges and lessons learned, and the roadmap for expanding to additional output types while continuing to improve efficiency and usability as the framework moves toward production use.

OVERALL WORKFLOW OVERVIEW

VGSART is a metadata-driven SAS® framework composed of interconnected macros that work together to standardize and automate the generation of clinical analysis outputs. Instead of building and updating table programs study by study, users define the table requirements in structured CSV metadata, including the analysis variables, where conditions, formats, and titles and footnotes. The framework then uses that metadata to run the same standardized table logic across studies, producing outputs that follow the

annotated mock shells, from basic descriptive summaries to more complex analyses such as shift tables though a unified process. The intent of this tool is that any new user can operate the full workflow of table generation with minimal assistance/ programming knowledge to create consistent, high-quality outputs regardless of any given study.

Using VGSART, analysis tables can be generated through a simple four-step process. As shown in Figure 1, the workflow mentioned in the user-guide provides any user step-by-step instructions from defining the table structure to producing final outputs in a clear and controlled manner.

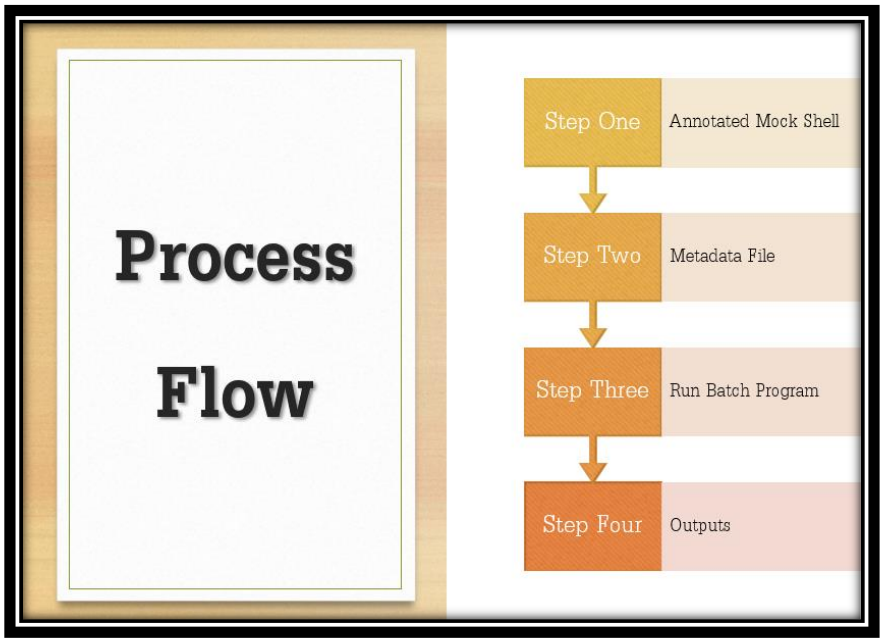


Figure 1: Process Flow

Step 1: Annotated Mock Shell

The annotated mock shell acts as a blueprint for the final output and clearly defines what each table should look like before any programming begins. As shown below in Figure 2, it specifies the table layout, titles, column structure for treatment groups, row labels, and formatting rules. Each row is annotated with the corresponding data source fields (for example, ADSL.AGE, ADSL.SEX, ADSL.RACE) and the required statistics such as means or frequencies.

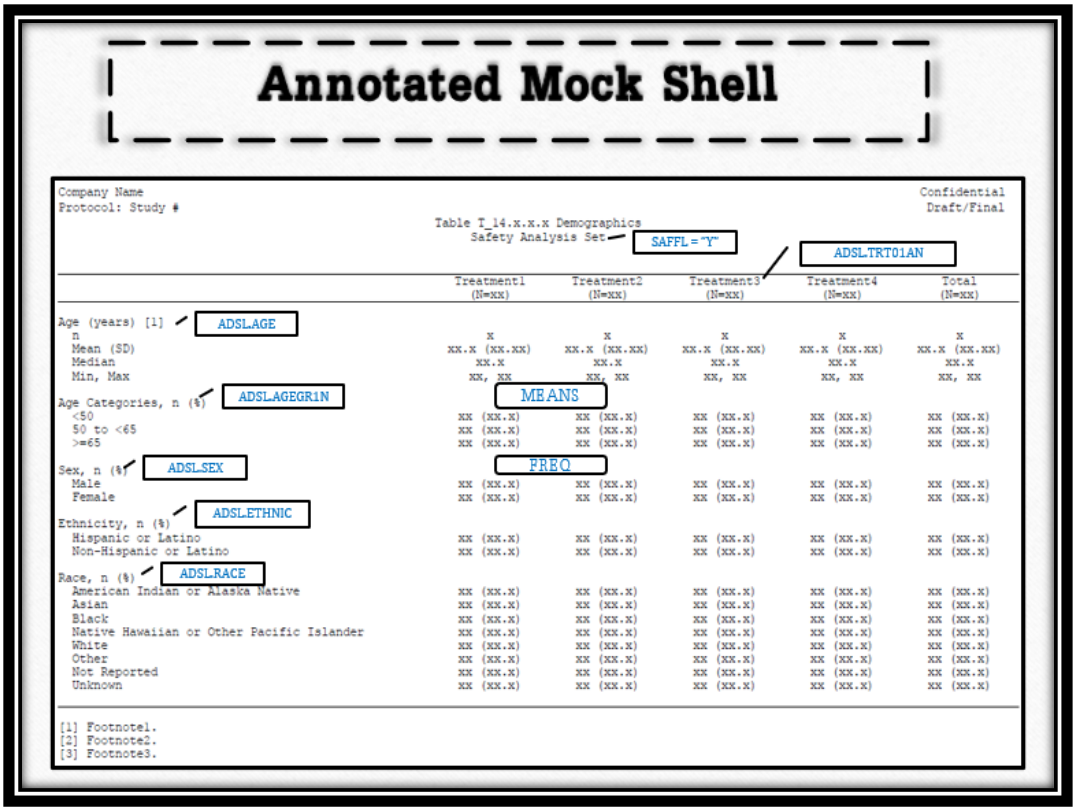
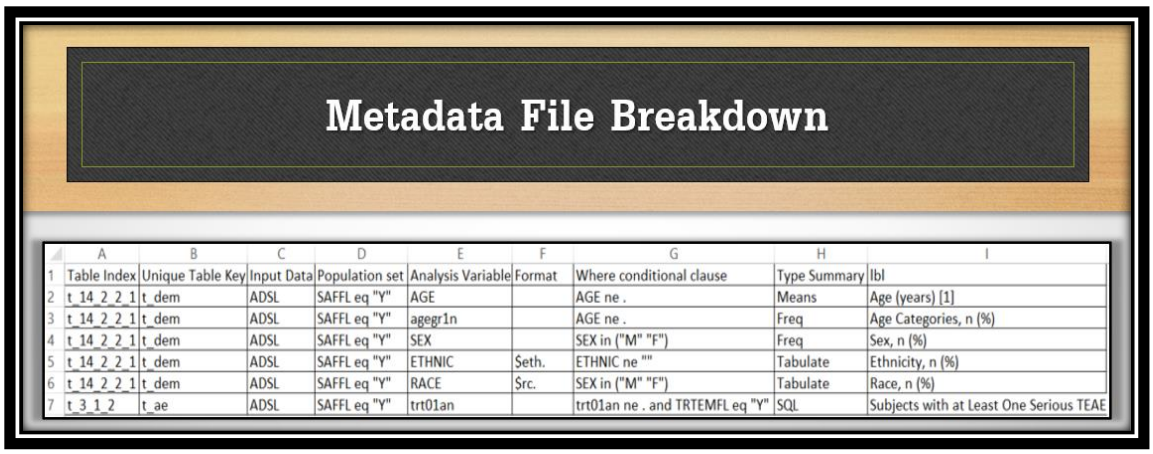


Figure 2: Annotated Mock Shell

By documenting these requirements upfront, the mock shell removes any ambiguity and ensures that all study team members share a common understanding of the expected output. It also reduces avoidable rework, minimizes programming errors, and supports consistent interpretation of specifications. The annotated mock shell gives programming teams a structured reference to follow, supporting a standardized, repeatable approach that speeds up development and ensures outputs are accurate and consistent across studies and programmers.

Step 2: Metadata Files

The metadata serves as the central place where all analysis information is defined and maintained. The files include essential table-specific requirements such as input dataset names, analysis required variables, conditional clauses, and more as shown in Figure 3. By bringing everything together in one location, it makes it easier for any end user or reviewer to understand what is being produced and why, without having to search through multiple programs or documents.



	A	B	C	D	E	F	G	H	I
1	Table Index	Unique Table Key	Input Data	Population set	Analysis Variable	Format	Where conditional clause	Type Summary	lbl
2	t_14_2_2_1	t_dem	ADSL	SAFFL eq "Y"	AGE		AGE ne .	Means	Age (years) [1]
3	t_14_2_2_1	t_dem	ADSL	SAFFL eq "Y"	agegr1n		AGE ne .	Freq	Age Categories, n (%)
4	t_14_2_2_1	t_dem	ADSL	SAFFL eq "Y"	SEX		SEX in ("M" "F")	Freq	Sex, n (%)
5	t_14_2_2_1	t_dem	ADSL	SAFFL eq "Y"	ETHNIC	$eth.	ETHNIC ne ""	Tabulate	Ethnicity, n (%)
6	t_14_2_2_1	t_dem	ADSL	SAFFL eq "Y"	RACE	$rc.	SEX in ("M" "F")	Tabulate	Race, n (%)
7	t_3_1_2	t_ae	ADSL	SAFFL eq "Y"	trt01an		trt01an ne . and TRTEMFL eq "Y"	SQL	Subjects with at Least One Serious TEAE

Figure 3: Metadata File Breakdown

The user can fill the metadata information either from following the annotated mock shells or from a prefilled starting set based on FDA-approved mock shells. The completed metadata helps the tool determine how the analyses should run and how the results should be displayed. If any study requirements change, updates can be made directly in the metadata file.

Step 3: Run Batch Program

After the metadata files are finalized, the outputs can be generated hitting "Batch Submit" as shown below in Figure 4, VGSART reads the metadata and drives table generation through the macro framework in a controlled sequence. The user can choose to either run a specific table or multiple tables at a time depending on the need.

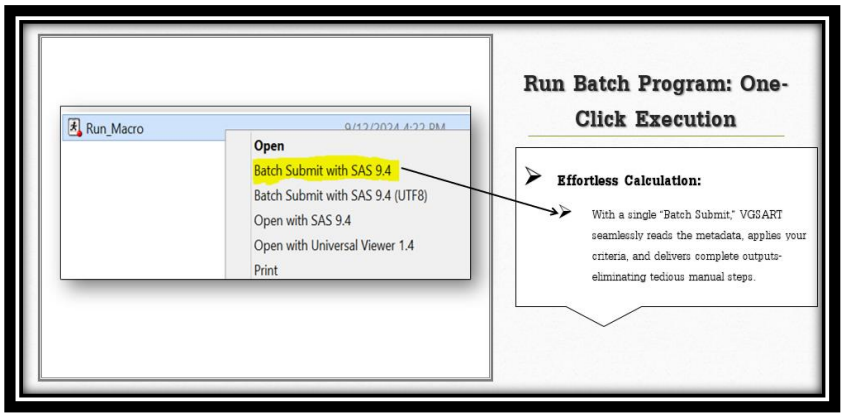
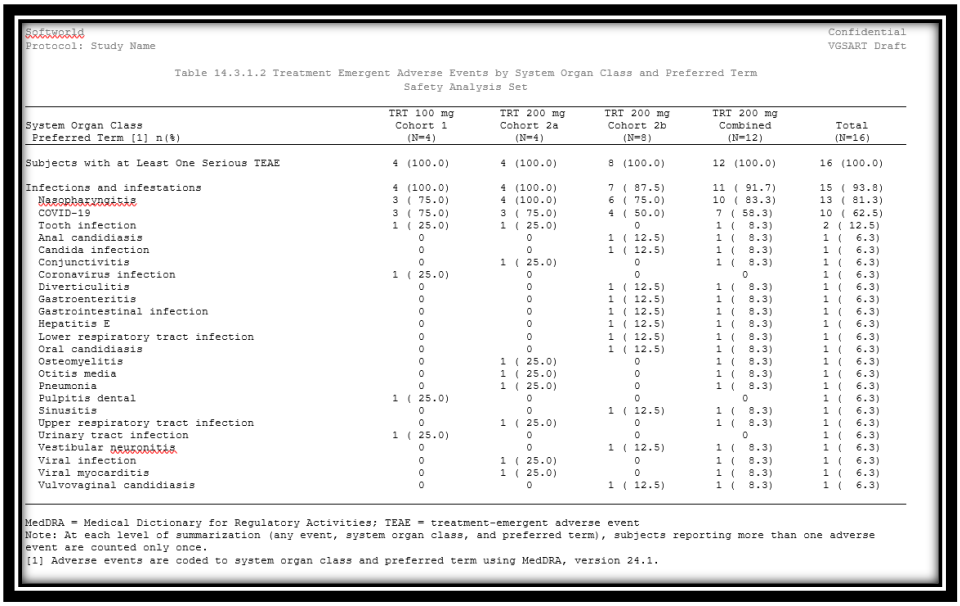


Figure 4: Batch Execution

Step 4: Outputs

As shown in Figures 5 and 6, the final outputs can be generated in Rich Text Format (RTF) or Portable Document Format (PDF) which are ready to be validated before submission. Using the VGSART tool, a comprehensive set of safety and clinical summary tables has been successfully generated such as Treatment-Emergent Adverse Events (TEAEs) and AE summaries by System Organ Class (SOC) and Preferred Term (PT) with standard breakdowns by seriousness, relatedness, severity, and dose.



Softworld
Protocol: Study Name

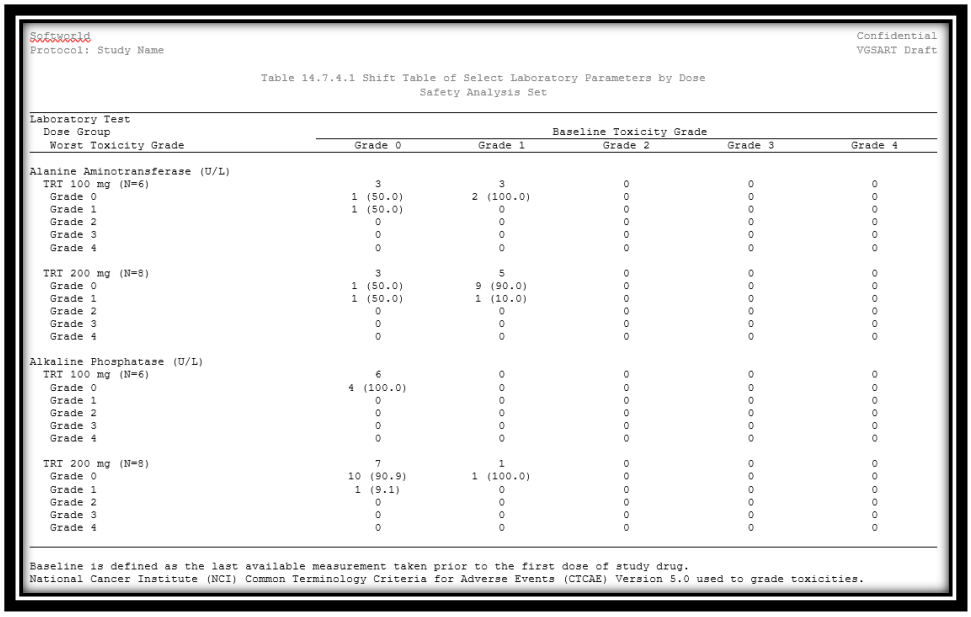
Confidential
VGSART Draft

Table 14.3.1.2 Treatment Emergent Adverse Events by System Organ Class and Preferred Term
Safety Analysis Set

System Organ Class Preferred Term [1] n(%)	TRT 100 mg Cohort 1 (N=4)	TRT 200 mg Cohort 2a (N=4)	TRT 200 mg Cohort 2b (N=8)	TRT 200 mg Combined (N=12)	Total (N=16)
Subjects with at Least One Serious TEAE	4 (100.0)	4 (100.0)	8 (100.0)	12 (100.0)	16 (100.0)
Infections and infestations	4 (100.0)	4 (100.0)	7 (87.5)	11 (91.7)	15 (93.8)
Nasopharyngitis	3 (75.0)	4 (100.0)	6 (75.0)	10 (83.3)	13 (81.3)
COVID-19	3 (75.0)	3 (75.0)	4 (50.0)	7 (58.3)	10 (62.5)
Tooth infection	1 (25.0)	1 (25.0)	0	1 (8.3)	2 (12.5)
Anal candidiasis	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Candida infection	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Conjunctivitis	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Coronavirus infection	1 (25.0)	0	0	0	1 (6.3)
Diverticulitis	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Gastroenteritis	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Gastrointestinal infection	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Hepatitis E	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Lower respiratory tract infection	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Oral candidiasis	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Osteomyelitis	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Otitis media	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Pneumonia	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Pulpitis dental	1 (25.0)	0	0	0	1 (6.3)
Sinusitis	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Upper respiratory tract infection	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Urinary tract infection	1 (25.0)	0	0	0	1 (6.3)
Vestibular neuronitis	0	0	1 (12.5)	1 (8.3)	1 (6.3)
Viral infection	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Viral myocarditis	0	1 (25.0)	0	1 (8.3)	1 (6.3)
Vulvovaginal candidiasis	0	0	1 (12.5)	1 (8.3)	1 (6.3)

MedDRA = Medical Dictionary for Regulatory Activities; TEAE = treatment-emergent adverse event
Note: At each level of summarization (any event, system organ class, and preferred term), subjects reporting more than one adverse event are counted only once.
[1] Adverse events are coded to system organ class and preferred term using MedDRA, version 24.1.

Figure 5: TEAEs by SOC & PT Safety Table



Softworld
Protocol: Study Name

Confidential
VGSART Draft

Table 14.7.4.1 Shift Table of Select Laboratory Parameters by Dose
Safety Analysis Set

Laboratory Test Dose Group Worst Toxicity Grade	Baseline Toxicity Grade Grade 0	Grade 1	Grade 2	Grade 3	Grade 4
Alanine Aminotransferase (U/L)					
TRT 100 mg (N=6)	3	3	0	0	0
Grade 0	1 (50.0)	2 (100.0)	0	0	0
Grade 1	1 (50.0)	0	0	0	0
Grade 2	0	0	0	0	0
Grade 3	0	0	0	0	0
Grade 4	0	0	0	0	0
TRT 200 mg (N=8)	3	5	0	0	0
Grade 0	1 (50.0)	9 (90.0)	0	0	0
Grade 1	1 (50.0)	1 (10.0)	0	0	0
Grade 2	0	0	0	0	0
Grade 3	0	0	0	0	0
Grade 4	0	0	0	0	0
Alkaline Phosphatase (U/L)					
TRT 100 mg (N=6)	6	0	0	0	0
Grade 0	4 (100.0)	0	0	0	0
Grade 1	0	0	0	0	0
Grade 2	0	0	0	0	0
Grade 3	0	0	0	0	0
Grade 4	0	0	0	0	0
TRT 200 mg (N=8)	7	1	0	0	0
Grade 0	10 (90.9)	1 (100.0)	0	0	0
Grade 1	1 (9.1)	0	0	0	0
Grade 2	0	0	0	0	0
Grade 3	0	0	0	0	0
Grade 4	0	0	0	0	0

Baseline is defined as the last available measurement taken prior to the first dose of study drug.
National Cancer Institute (NCI) Common Terminology Criteria for Adverse Events (CTCAE) Version 5.0 used to grade toxicities.

Figure 6: Shift Table of Select Lab Parameters by Dose

In addition to the basic safety summaries such as demographics, subject disposition, medical history, and protocol deviations, it also supports more complex safety outputs like change from baseline summaries for vital signs, physical examinations, and ECG by visit and dose, and shift tables for selected laboratory parameters (Figure 6).

VGSART System Design

This section describes the system architecture and execution logic of the VGSART framework, including project setup, metadata preparation, and runtime behavior.

Environment Setup
The execution pipeline begins with creation of a standardized project directory structure. A main project folder is established, containing predefined subfolders named **Data**, **Docs**, **Macros**, and **Outputs**. This structure gives the framework a consistent way to locate input datasets, metadata files, macro programs, and generated outputs.

Blank metadata template files, including **Import_file**, **Formats**, and **Titles_footnotes**, are placed in the Docs folder. All reusable SAS macro programs required by the framework are stored in the Macros folder.

The environment is initialized through execution of the **Init.sas** program which defines library references and file paths for data, metadata documents, macro locations, and output destinations. It runs automatically as part of the table generation process and ensures the framework consistently points to the correct project directories at runtime.

In parallel, approved table mock shells provided by the statistics team are obtained for each planned output. These mock shells are reviewed to identify analysis variables, population definitions, grouping and ordering rules, expected layouts, titles, footnotes, and display formats. The mock shells serve as the source for metadata population.

Metadata Population
Following environment setup, metadata files are populated to fully define table content and presentation. Metadata entry is guided directly by the approved mock shells.

1. **Table Metadata**
Table-level metadata is captured in the **Import_file** metadata document. For each table to be produced, the required fields are populated to uniquely identify the output and define how it should be generated. These fields include, but are not limited to, the unique table key, input dataset, population set, analysis variables, variable ordering, conditional logic, and other structural attributes. This metadata also specifies the analysis category, which later determines the statistical processing path used during execution.

2. **Titles and Footnotes Metadata**
Titles and footnotes are maintained separately in the **Titles_footnotes** metadata document. Required columns are completed for each table, using the same unique table key defined in the table metadata to ensure correct alignment. Maintaining textual elements independently allows updates to titles and footnotes without modifying analytical specifications.

3. **Formats Metadata**
Any custom display formats required for table presentation are defined in the **Formats** metadata document. References to these formats are specified in the table metadata to ensure consistent application during output generation.

Formats Metadata: Formats

Column/Variable Name	Purpose	Usage Instructions + Examples
FMTNAME	This variable is to specify the name of the format the user is trying to create.	• Ex. Value: Agegrp
START	This variable is the specific value that gets mapped to a label. Represents the exact input value that should be formatted in a certain way. 1 to 1 mapping.	• Ex. Value: 1
LABEL	Is the user-defined formatted text/value that is displayed in place of the raw data value mentioned in START.	• Ex. Value: 50 to <65
TYPE	Indicates whether the format specified by the user is for numeric or character data.	• Ex. Value: C

Figure 7: Formats Metadata Descriptions

This separation of metadata components supports clarity, reuse, and controlled updates while minimizing the need for program-level changes.

Execution Flow

Once metadata population is complete, table generation is initiated through the framework's master execution program.

The user runs **Run_macro.sas**, which serves as the central control program for the framework. This program reads the environment settings established in **Init.sas** and coordinates execution of all required components. The user may specify whether to generate a single table or all tables defined in the table metadata document.

The main driver macro reads the populated metadata files and determines which tables are eligible for output. Based on the analysis specifications provided in the metadata, the framework dynamically routes processing to the appropriate statistical macro. Input datasets are prepared as required, and table-specific titles and footnotes are applied programmatically during output generation.

These internally developed macros are designed to compute the required statistical outputs (as shown in Figure 8), including frequency counts and percentages for categorical variables, descriptive statistics such as N, mean, median, minimum, and maximum for continuous variables, and adverse event counts using dedicated SQL-based macros.

```
%include "&macroPath\freq_Macro.sas";
%include "&macroPath\means_Macro.sas";
%include "&macroPath\sql_Macro.sas";
%include "&macroPath\R_macros.sas";
%include "&macroPath\processing.sas";
```

Figure 8: Set of Integrated Macros

The metadata framework adds flexibility by letting users specify statistical methods at both the table and analysis variable level, controlling which statistics are applied for each output. In addition to the stat macros, the reporting macros work together with pre-processing and post-processing macros work seamlessly to ensure consistent data handling, formatting, and output. The pre-processing macro prepares treatment groups by combining them as specified in the macro parameters, applies where conditions and filter data-based population flags, calculates total counts (BigN) for analysis.

Preprocessing and Post processing

Program	Macro	Associated Tables	Purpose
	%PP(trt var=, cmb trts=, display trts=)	ALL	Preprocesses treatment groups by combining as specified within parameters, applies where conditions & page flags, calculates total counts (BigN) for each group, and creates preprocessed aDaM datasets ready for analysis.
Processing	%POSTP	ALL	Applies sequence orders, sub labels, page count calculations for the final datasets used by reporting macros.

Figure 9: Processing Macro Descriptions

The post-processing macro then applies sequence ordering, adds sub-labels, and calculates page counts for the final datasets used by the reporting macros. Their modular design allows smooth integration throughout the table generation. The final tables are generated and saved in the predefined Outputs folder. This metadata-driven approach enables controlled output generation while minimizing the need for direct interaction with the underlying SAS code.

FUNCTIONALITY ACHIEVED TO DATE

VGSART has progressed to the point where it can automate majority of safety analysis tables in a consistent, repeatable way. The current focus is safety tables, concentrating on core templates that can be reused across any study to reduce manual programming mistakes and shorten table production time.

The framework currently supports various unique templates across multiple table families with different structures. Within the implemented templates, the framework does not limit the number of tables a user can generate. Users can define multiple outputs by varying treatment groups, character variables, column definitions, population subsets, and where conditions. This matters because the same table family often needs multiple variations and this approach avoids treating each variation as a separate programming effort requiring multiple programs. A group of macro programs support the current build and provide the processing, analysis, and reporting needed to generate the implemented templates with customized metadata.

Figure 10. Snapshot Creation & Selection

To prioritize practicality and user friendliness, we developed a functional GUI utilizing R Shiny that serves as the front end for routine work with VGSART. Users can import existing metadata files to continue work that is already in progress or create a new snapshot **(Figure 10)**. Each snapshot acts as a self-contained working library within the study for traceability, including its own metadata files, ADaM datasets, and output folders. Snapshots can be handed off so another team member can continue in the same working directory without rebuilding the setup.

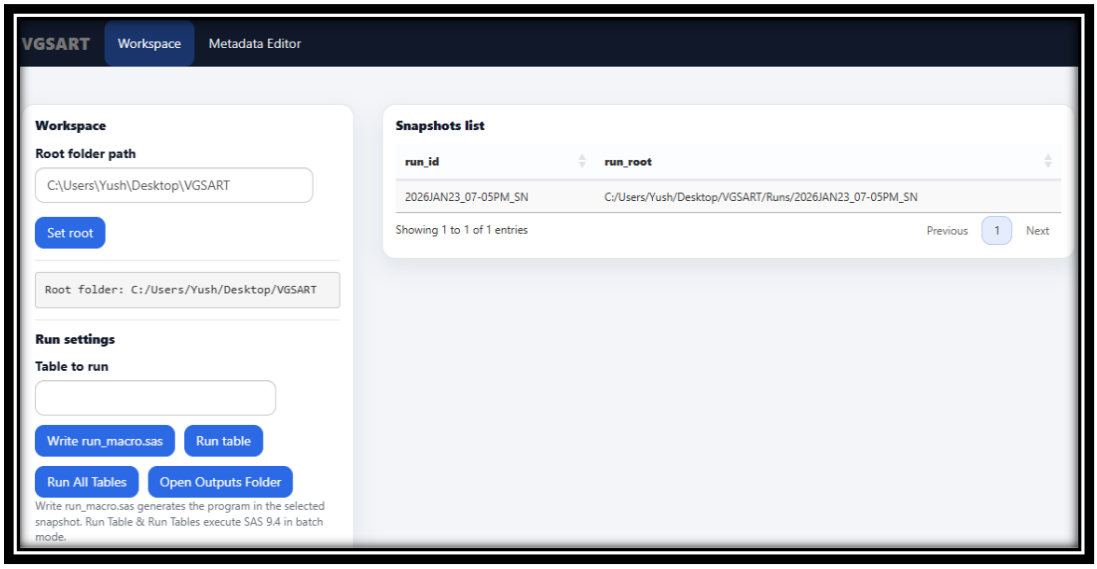


Figure 11. VGSART Workspace Tab

Additionally, the GUI supports SAS execution directly within the interface so that the user can run a single table by specifying it in a text box or run all tables populated in the metadata **(Figure 11)**.

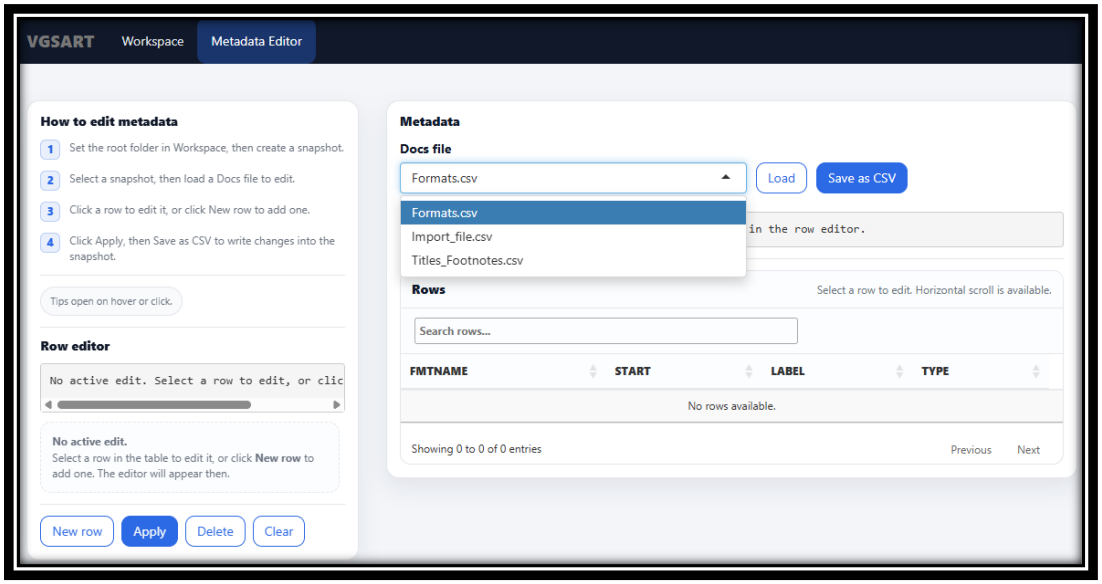


Figure 12. Metadata Files Importing. Editing, & Saving

All metadata fields are editable in the GUI, and changes are saved directly back to the snapshot files. Tooltips are embedded in the metadata fields so users can view field descriptions while working rather than

stopping to reference the user guide **(Figure 12)**. Although the GUI is still being tested and refined by the development team, once it is finalized it will strengthen the overall workflow by reducing user setup effort and making routine table generation easier and more consistent.

In its current state, VGSART provides a consistent metadata-driven foundation for safety table production, combining reusable templates with a workflow that reduces manual steps and supports traceable execution.

TESTING AND VALIDATION

Our testing and validation approach was designed to demonstrate that our tool can reduce programming time while maintaining the same quality outputs produced through standard programming. The goal was to match QC-reviewed deliverables without mismatches in counts, presentation, or table structure. Programming time refers to the number of hours and effort spent generating the tables, excluding the QC comparison process.

To keep the comparison meaningful, we developed and tested our tool in a dedicated VGSART folder structure using example FDA-submitted studies, serving as the reference point for comparison. This kept testing grounded in validated deliverables and allowed the framework to behave dynamically across any study.

For each study, we first confirmed the number of working hours spent on producing the tables using standard programming with the project manager, serving as our baseline. We then used the exact ADaM datasets that supported the final deliverables and populated VGSART metadata to match the mock shells. The tool generated the same set of safety tables, which we compared against the delivered, verified outputs, while also tracking the time of completion to recreate the package and reallocating the time saved from programming to QC comparison against the deliverables.

After establishing this approach, we expanded testing to confirm that the usage of the tool was not dependent on the developers. Two users were given access to the tool and used the user guide to follow the same workflow, recording the time spent producing the same tables. One user had TFL programming experience. The other primarily worked on SDTMs and had not previously produced TFLs.

On average, the programmer with TFL experience reproduced the delivered safety tables in 25% less programming time compared with standard programming. The user without TFL experience reproduced the same tables in ~18% to ~20% less programming time. In both cases, time was better utilized, from writing and reworking code to a deeper QC review process. This is the outcome we expect when applying VGSART to new and ongoing studies once the tool is fully ready for production use.

CHALLENGES AND LESSONS LEARNED

One of our main goals was to eliminate the need to access SAS programs to obtain final outputs. Working toward that goal highlighted the difference between a framework that runs dynamically from metadata and one that still requires users to make code edits.

The first challenge was connecting metadata to macros in a way that stayed flexible while remaining controlled. Reading user input from CSV files and translating it into usable SAS macro logic took more effort than expected until we fully utilized the metadata structure. We treated column variables as macro variables, built a loop to gather columns from a given CSV, and structured the rows of values in SAS so the macros could work dynamically to generate analysis reports.

Early versions supported only one table type, the demographics table. While this was a useful start, it did not prove that the framework could generate a full package. Expanding coverage required building

templates as reusable structures, not one-off programs, and it also exposed a limitation. Initially, the tool struggled to run more than one table at a time because it lacked a looping SQL step that returned to the start and processed metadata row by row. After implementing that loop, users could then populate multiple tables in one metadata file and run them together, which aligns with how safety packages are produced.

Presentation rules also created early complications. Titles, footnotes, and header labels, such as treatment names, were initially hard-coded, making updates inefficient and undermining the purpose of automation. We moved titles, footnotes, and main header labels into dedicated metadata files and added logic to handle any number of titles and footnotes while respecting table type. This improved maintainability and was a clear step toward truly dynamic table production, where presentation updates can be controlled through metadata instead of hard-coded edits.

Preprocessing was one of the biggest barriers to making the framework behave consistently across studies. Early on, we still had to go into the study macros and apply study level setup by hand so the automated templates would run correctly. In this context, preprocessing is the repeatable preparation step that turns the source ADaM inputs into table-ready datasets by applying population flags and where conditions, establishing Big N denominators, and standardizing key display variables such as treatment and visit so the templates align with the mock shells. We addressed this by creating a separate processing program that includes both preprocessing and post-processing macros. The preprocessing macro applies the population and where condition criteria, calculates Big N counts, and produces an analysis-ready input dataset that the analysis macros can use directly.

Adoption introduced a different challenge. A walkthrough was enough to get a tester started; however, it prompted follow-up questions and left progress dependent on the development team. As a result, we created a detailed user guide documenting the end-to-end process, describing macros and metadata fields, providing examples and parameters, and walking through the entire process of generating a test table with screenshots. With that in place, testers were able to build full TFL packages that mimicked FDA-submitted studies with less back-and-forth guidance from the dev team.

SCALABILITY AND ROADMAP

The next phase of VGSART focuses on expanding what the framework can produce, while keeping the same metadata-driven approach that has made the current build usable across studies. The immediate priority is coverage. We are targeting more complicated tables with more specific structures, then expanding into efficacy tables once the safety foundation is fully stable. Figures remain a longer-term target, and we plan to approach them only after the table framework is mature enough to support consistent, repeatable outputs.

Scaling also means further expanding the template library without adding a more time-consuming workflow for users. Our aim is to give users more control through metadata while keeping outputs aligned to the TFL conventions expected in regulated deliverables and reducing the study-specific rework that tends to creep in as packages evolve.

Alongside this effort, we are working toward improved efficiency beyond what we have observed so far. Our target is a ~30-40% reduction in programming time and errors, and achieving that depends on two concrete changes: improving macro efficiency and reducing the amount of manual input required to define an output. The goal is to simplify the metadata fields users must fill out without reducing the templates' flexibility.

The GUI already supports metadata importing, editing, executing options, and snapshot-based traceability, so the next step would be to continue adding features that improve day-to-day usability. The end goal is to

have a user who has never worked on TFL programming to generate tables independently by following the same workflow each time.

Future-proofing is part of the roadmap as well. The current implementation uses SAS for the macro framework and R Shiny for the GUI, and one longer-term goal is to expand the project to adopt R more directly while keeping the same metadata-driven structure that supports reproducibility and scalability.

CONCLUSION

In this paper, we described how VGSART is used in practice, what has been implemented so far, our testing technique, and the key obstacles we had to overcome while developing this metadata-driven approach work in an existing environment. The results to date show that this approach can reduce time spent rebuilding repetitive table logic and shift effort toward what matters most, careful review and QC of the final outputs.

VGSART was built to fit the way our teams already work, using SAS and structured metadata to drive consistent, traceable table generation while keeping the calculation logic transparent and familiar. As we expand coverage and continue making the tool more efficient, the core takeaway remains the same: when table requirements are captured in metadata and execution is standardized, teams can spend less time rewriting code and more time delivering outputs they trust.

ACKNOWLEDGEMENTS

The authors would like to thank Sivakumar Ramakrishnan for initiating the project and providing valuable guidance and review, and Maggie Jiang for mentoring and testing support during the development of this work.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author Name: Piyush Makkapati

Organization: Vita Global Sciences, a Kelly Company

Address: 281 Winter St, Waltham, MA 02451

Email: Pmakkapati@softworldinc.com

Author Name: Swathi Nattam

Organization: Vita Global Sciences, a Kelly Company

Address: 281 Winter St, Waltham, MA 02451

Email: Snattam@softworldinc.com

Author Name: Bhargav Makkapati

Organization: Vita Global Sciences, a Kelly Company

Address: 281 Winter St, Waltham, MA 02451

Email: Bmakkapati@softworldinc.com