Paper SM09

Unlocking the Future of Tables with the {tfrmt} R Package

Becca Krouse, GSK

ABSTRACT

Clinical trial displays come in many forms; even highly standard displays require modifications. Table customization and subsequent QC drain resources, and creating mock tables separately during analysis planning causes duplicated efforts. With a new industry standard on the horizon, Analysis Results Data (ARD), we worked to rethink the overall reporting process. The {tfrmt} package transforms table creation by leveraging the consistent structure of ARDs. With {tfrmt}, a user defines a table's formatting independently of the data, allowing reuse of code for both mock and final tables. Formatting templates enable automation of standard tables, as well as the ability to quickly refine tables by layering study-specific customizations. These benefits extend beyond R users, as {tfrmt} can generate and consume language-agnostic metadata. Furthermore, the companion Shiny app, {tfrmtbuilder}, simplifies the learning process for newcomers and facilitates customizations for non-programmers. Attendees will discover how to incorporate {tfrmt} and {tfrmtbuilder} into their unique workflows.

INTRODUCTION

Clinical trials reporting is traditionally very resource intensive, with its multiple stages and repeated quality control. With the emerging data standard of Analysis Results Data (ARD), there is an opportunity to reevaluate existing workflows and develop new tools based on a consistent input data structure. This paper introduces two R packages, {tfrmt} and {tfrmtbuilder}, and discusses their potential to transform the reporting process by reducing rework and improving collaboration. The paper describes new possibilities for {tfrmt}-based workflows and how they can be adapted to an organization's unique needs.

The {tfrmt} and {tfrmtbuilder} R packages

{tfrmt}

The {tfrmt} R package, first released to CRAN in 2022, leverages the consistent structure of ARDs to streamline table-making, specifically for pharma users. It allows precise control over display formatting, which is defined independently of the data and can be stored as reusable metadata. Therefore, the structure of the table can be specified once and reused for both mock and final tables. Further, organizations can create formatting templates to ensure consistency and increase the speed of reporting. Study or report specific customizations can be layered onto existing tables, facilitating smaller adjustments without redefining the whole table. Details about specific options and syntax for making {tfrmt} tables are out of scope for this paper but can be found on the website.

{tfrmtbuilder}

The {tfrmtbuilder} R package, first released to CRAN in 2023, provides a point-and-click interface to the {tfrmt} package to reduce the learning curve and broaden its range of users to include non-programmers. More information about the {tfrmtbuilder} package can be found on its website.

New and improved workflow for programmers

In R, the programmer defines a tfrmt (using `tfrmt::tfrmt()`) including information about:

- Names of key columns in the ARD corresponding to components of the table (e.g., `column`, `value`, etc.)
- Formatting instructions specified via a series of "plans", which are functions for specific aspects of the table

This tfrmt can be assigned to an object in the environment as such:

```
tfrmt_ae <- tfrmt(
  # names of key columns expected in the ARD
  # informs the layout of the table (rows, columns, etc.)
  group = "AEBODSYS",
  label = "AETERM",
  column = "TRT",
```

```
  ...
  # formatting-related specifications
  body_plan = body_plan(...),
  col_plan = col_plan(...),
  ...
)
```

This object (with class tfrmt) is purely metadata; it dictates the structure/orientation of the table as well as details such as the alignment and rounding of data values. The following examples show how this object can be reused to create tables during various stages of a study.

Create mock displays

Mock displays are typically created manually before data is available. {tfrmt} offers the novel ability to create mock tables programmatically. The user can apply one additional line of code to the initial tfrmt object to generate a generic mock table:

```
tfrmt_ae |>
  print_mock_gt()
```

	TRT1 N = xx	TRT2 N = xx	TRT3 N = xx
AEBODSYS_1			
AETERM_1	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AETERM_2	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AETERM_3	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AEBODSYS_2			
AETERM_1	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AETERM_2	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AETERM_3	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AEBODSYS_3			
AETERM_1	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AETERM_2	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
AETERM_3	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)

Without any study data, {tfrmt} gleans information from the metadata contained in the tfrmt object to generate a mock table on the fly. For example, the names of key columns expected in the ARD are used to populate row and column levels. Metadata from the `body_plan` and `big_n` functionality are used to generate the location and appearance of the x's, which are placeholders for future data values.

The user can optionally provide sample data (created programmatically or pulled from a previous study) to add more information to the mock, such as the exact row/column levels. If the user opts to supply data, any numeric results values are ignored for the table.

```
tfrmt_ae |>
  print_mock_gt(.data = mock_dat_ae)
```

	Placebo N = xx	Xanomeline Low Dose N = xx	Xanomeline High Dose N = xx
ANY BODY SYSTEM	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
CARDIAC DISORDERS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
GASTROINTESTINAL DISORDERS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
APPLICATION SITE PRURITUS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
APPLICATION SITE ERYTHEMA	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
APPLICATION SITE IRRITATION	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
INFECTIONS AND INFESTATIONS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
NERVOUS SYSTEM DISORDERS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
DIZZINESS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
RESPIRATORY, THORACIC AND MEDIASTINAL DISORDERS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
SKIN AND SUBCUTANEOUS TISSUE DISORDERS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
PRURITUS	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
ERYTHEMA	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)
RASH	XX (xx.x %)	XX (xx.x %)	XX (xx.x %)

Create final displays

Once the study is finalized and data becomes available, the user can transform the mock table into a table populated with values by modifying the print function call.

```
tfrmt_ae |>
  print_to_gt(.data = dat_ae)
```

	Placebo N = 86	Xanomeline Low Dose N = 84	Xanomeline High Dose N = 84
ANY BODY SYSTEM	65 (75.6 %)	77 (91.7 %)	76 (90.5 %)
CARDIAC DISORDERS	12 (14.0 %)	13 (15.5 %)	15 (17.9 %)
GASTROINTESTINAL DISORDERS	17 (19.8 %)	14 (16.7 %)	20 (23.8 %)
GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	21 (24.4 %)	47 (56.0 %)	40 (47.6 %)
APPLICATION SITE PRURITUS	6 (7.0 %)	22 (26.2 %)	22 (26.2 %)
APPLICATION SITE ERYTHEMA	3 (3.5 %)	12 (14.3 %)	15 (17.9 %)
APPLICATION SITE IRRITATION	3 (3.5 %)	9 (10.7 %)	9 (10.7 %)
INFECTIONS AND INFESTATIONS	16 (18.6 %)	9 (10.7 %)	13 (15.5 %)
NERVOUS SYSTEM DISORDERS	8 (9.3 %)	20 (23.8 %)	25 (29.8 %)
DIZZINESS	2 (2.3 %)	8 (9.5 %)	11 (13.1 %)
RESPIRATORY, THORACIC AND MEDIASTINAL DISORDERS	8 (9.3 %)	9 (10.7 %)	10 (11.9 %)
SKIN AND SUBCUTANEOUS TISSUE DISORDERS	20 (23.3 %)	39 (46.4 %)	40 (47.6 %)
PRURITUS	8 (9.3 %)	21 (25.0 %)	26 (31.0 %)
ERYTHEMA	8 (9.3 %)	14 (16.7 %)	14 (16.7 %)
RASH	5 (5.8 %)	13 (15.5 %)	9 (10.7 %)

Customize final displays

Suppose a team member needs to modify this table slightly for a new report. Instead of duplicating old code or beginning from scratch, this colleague layers minor changes to the text and column order onto to the original table:

```
custom_tfrmt_ae <- tfrmt(
    title = "Adverse Events by Treatment Group",
    col_plan = col_plan(
```

```
      AEBODSYS, AETERM,
      contains("High"),
      contains("Low"),
      Placebo
    ),
    footnote_plan = footnote_plan(
      footnote_structure("Note: Percentages are based on ...")
    )
  )

tfrmt_ae |>
  layer_tfrmt(custom_tfrmt_ae) |>
  print_to_gt(dat_ae)
```

Adverse Events by Treatment Group

	Xanomeline High Dose N = 84	Xanomeline Low Dose N = 84	Placebo N = 86
ANY BODY SYSTEM	76 (90.5 %)	77 (91.7 %)	65 (75.6 %)
CARDIAC DISORDERS	15 (17.9 %)	13 (15.5 %)	12 (14.0 %)
GASTROINTESTINAL DISORDERS	20 (23.8 %)	14 (16.7 %)	17 (19.8 %)
GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	40 (47.6 %)	47 (56.0 %)	21 (24.4 %)
APPLICATION SITE PRURITUS	22 (26.2 %)	22 (26.2 %)	6 (7.0 %)
APPLICATION SITE ERYTHEMA	15 (17.9 %)	12 (14.3 %)	3 (3.5 %)
APPLICATION SITE IRRITATION	9 (10.7 %)	9 (10.7 %)	3 (3.5 %)
INFECTIONS AND INFESTATIONS	13 (15.5 %)	9 (10.7 %)	16 (18.6 %)
NERVOUS SYSTEM DISORDERS	25 (29.8 %)	20 (23.8 %)	8 (9.3 %)
DIZZINESS	11 (13.1 %)	8 (9.5 %)	2 (2.3 %)
RESPIRATORY, THORACIC AND MEDIASTINAL DISORDERS	10 (11.9 %)	9 (10.7 %)	8 (9.3 %)
SKIN AND SUBCUTANEOUS TISSUE DISORDERS	40 (47.6 %)	39 (46.4 %)	20 (23.3 %)
PRURITUS	26 (31.0 %)	21 (25.0 %)	8 (9.3 %)
ERYTHEMA	14 (16.7 %)	14 (16.7 %)	8 (9.3 %)
RASH	9 (10.7 %)	13 (15.5 %)	5 (5.8 %)

Note: Percentages are based on the number of subjects in the safety population within each group.

More information about layering can be found in the Layering tfrmts section of the website.

Save for reuse

In the above example, the user defined the tfrmt during the study planning phase. Teams or organizations can save standard tfrmts for use in future studies, eliminating the need to define all formatting up front.

- R-based templates

  It is possible to build a function in R that represents a standard tfrmt, such as a standard demographics or AE table, to be used across studies. Teams can utilize this template as-is or layer it with additional customizations. For example, a user can apply an organization-level template (i.e., `org_tfrmt_ae()`) as well as a therapeutic area template (i.e., `ta_tfrmt_ae()`, then layer study-specific customizations:

  ```
  org_tfrmt_ae() |>
    ta_tfrmt_ae() |>
    layer_tfrmt(custom_tfrmt_ae)
  ```

  More information about templates can be found in the Creating Template tfrmts section of the website.

- Language-agnostic metadata (JSON)

  Alternatively, an existing tfrmt object can be converted into a JSON format for future use. As JSON, this metadata can be conveniently stored in databases or file shares. Because JSON is language-agnostic, the metadata can be defined or edited outside of R, thereby facilitating integration in multilingual pipelines.

  The tfrmt object can easily be converted to JSON (and optionally saved to a file) with a single line of code:

  ```
  tfrmt_ae |>
    tfrmt_to_json()
  ```

with the resulting JSON appearing as such:

```
{
  "group": ["AEBODSYS"],
  "label": ["AETERM"],
  "param": ["param"],
  "value": ["value"],
  "column": ["TRT"],
  "row_grp_plan": {
    "struct_list": [],
    "label_loc": {
      "location": ["indented"],
      "indent": ["  "]
    }
  },
  "body_plan": [
  ...
}
```

Similarly, the JSON can be quickly loaded back into R, where it becomes a tfrmt object:

```
tfrmt_ae <- json_to_tfrmt("tfrmt_ae.json")
```

Information about the JSON functionality is available in the Export to JSON vignette.

New and improved workflow for everyone

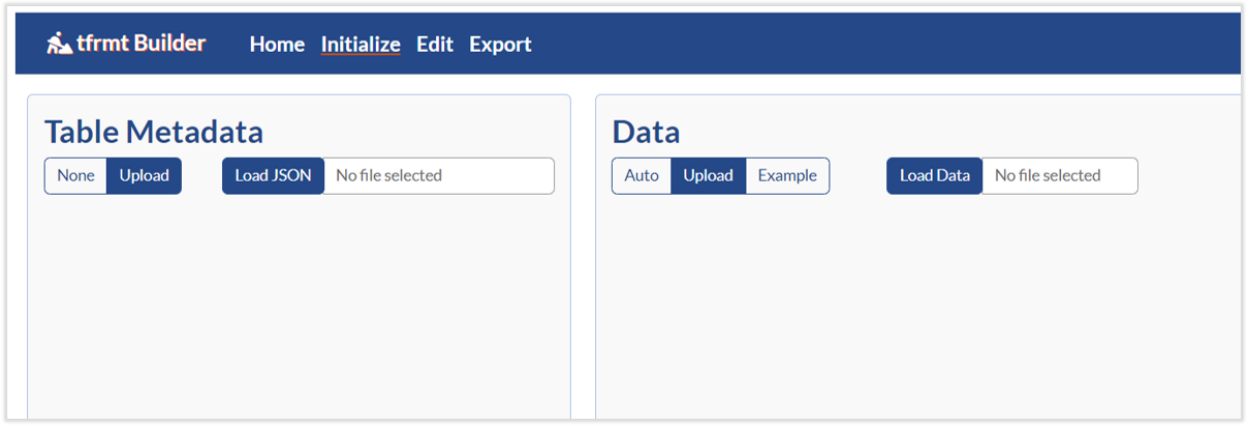
The core functionality of `tfrmt::tfrmt()` is ideal for an R-centric programmer's workflow. However, many teams operate in a heterogeneous environment with different types of software depending on the task or personal preference. Additionally, both technical and non-technical team members alike are consumers of tables and wish to modify their appearances. Fortunately, the companion {tfrmtbuilder} package contains a point-and-click interface for {tfrmt}, enabling a wide range of users to quickly customize tables regardless of role or experience.

The example below focuses on one of the primary use cases of the app: modifying an existing table, possibly for use in a new report or presentation. With {tfrmtbuilder}, non-programming staff are equipped to make these edits and preserve the edited metadata for reproducibility. The following example assumes a previous programmer created the original table and stored its formatting in JSON format.

Defining the starting point

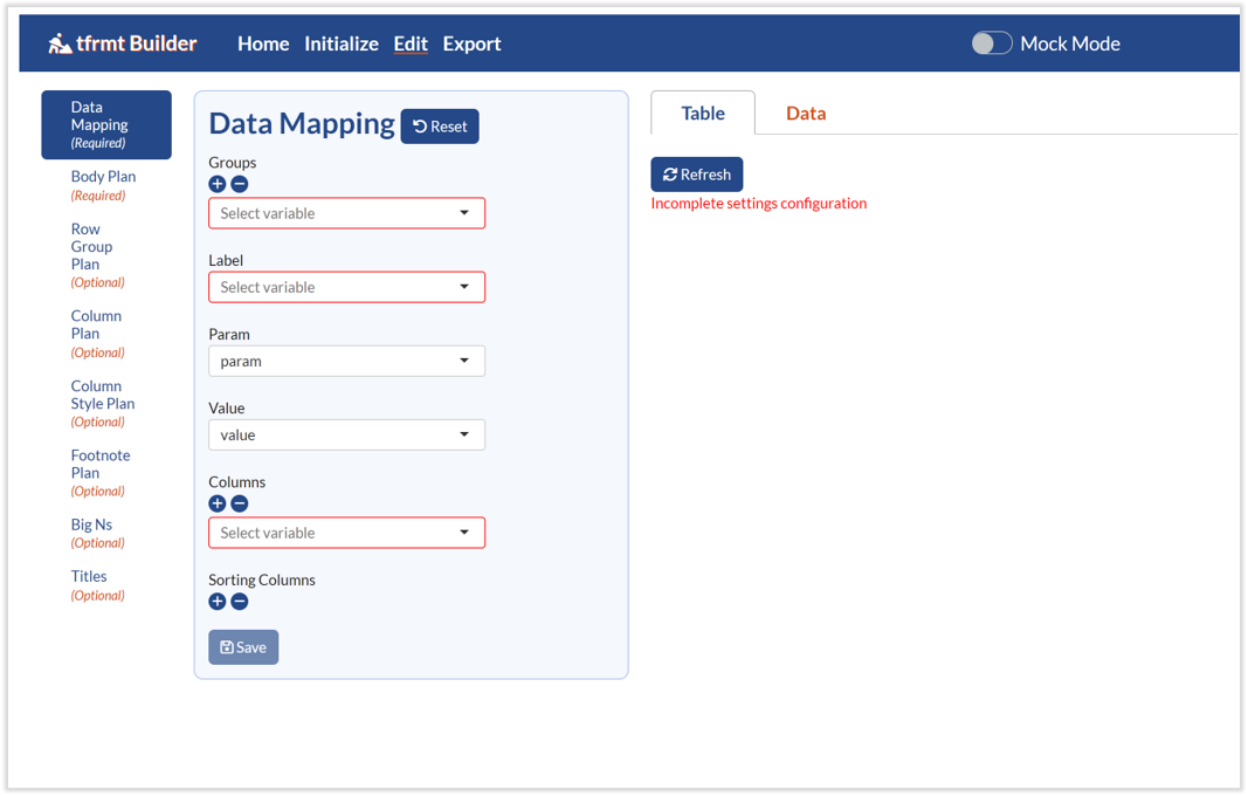
In the top toolbar of the app, there is a toggle for "Mock mode". By default, option is enabled, and mock displays will be created. Because the user is creating a table with values, they will turn off mock mode as a first step.

The "Initialize" tab is the first available tab following the landing page. Here, the user loads in the JSON file to reproduce the existing tfrmt, along with the ARD file.

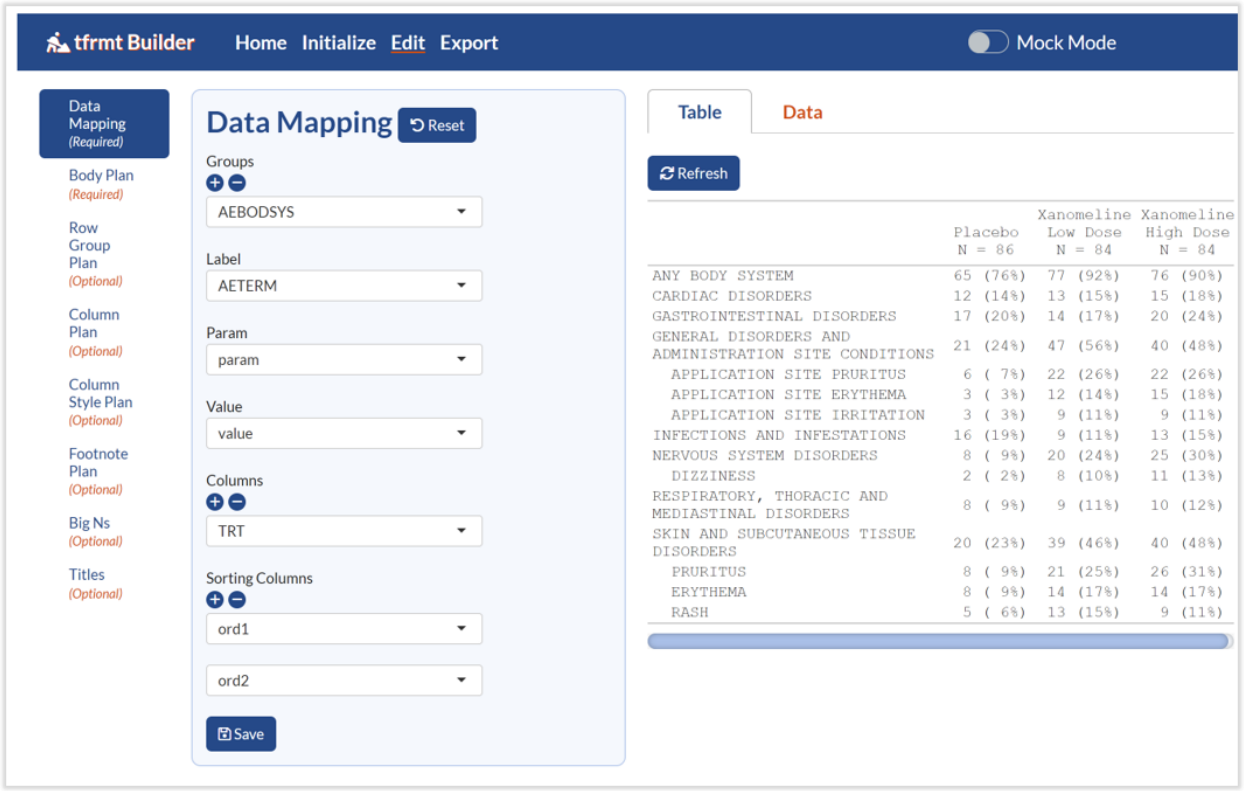


Build or modify table

Table formatting adjustments occur within the “Edit” tab. If the user had not provided any existing tfrmt metadata (JSON), they would define the formatting from scratch. Note the initial view would indicate areas of incompleteness:

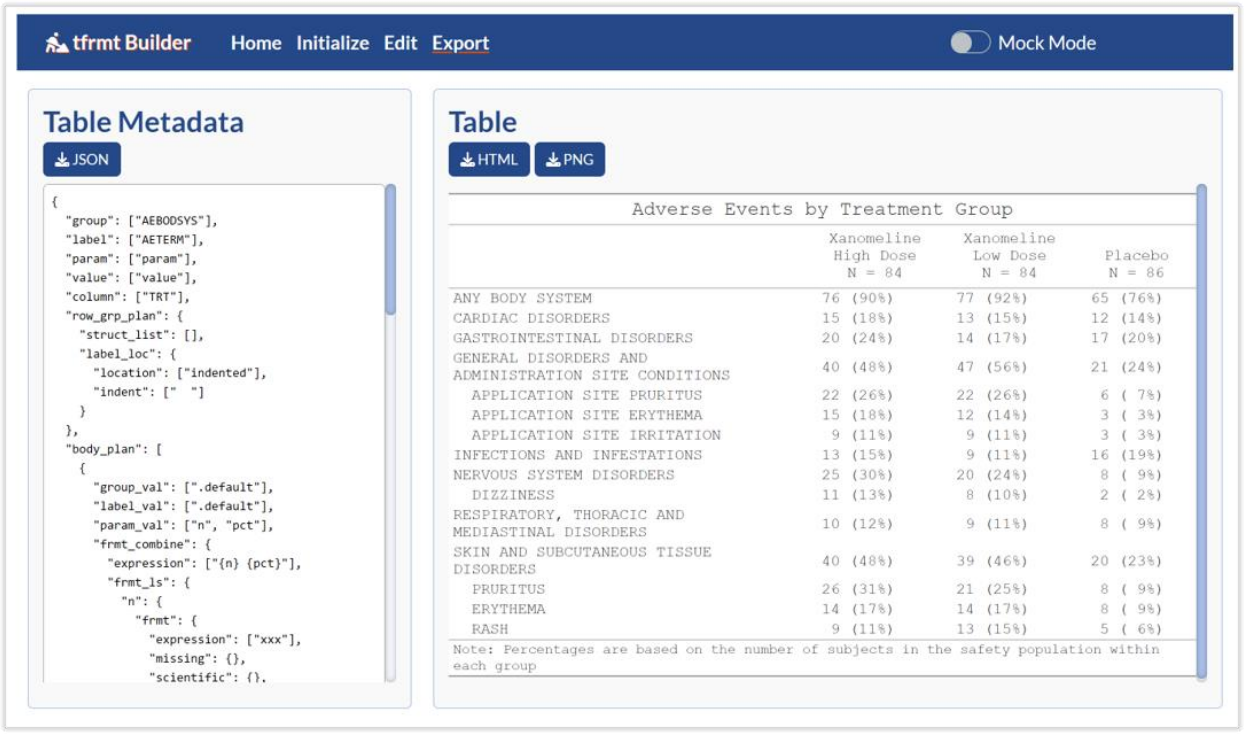


However, after loading in the existing tfrmt, the app populates settings accordingly and recreates the original table.



From here, the user may choose to adjust options for formatting the table. For example, they can navigate to the “Body Plan” tab to adjust the rounding of the values, or the “Column Plan” tab to adjust column order. With the side-by-side view of the formatting options and the table view, the user can see the immediate impact of each change on the final table. Additionally, each formatting section has a reset button for restoring the original state of the table.

Save the result



When the user is satisfied with the results, they may save the final display and export an updated JSON file to ensure reproducibility.

Continue the cycle

Other team members can load the updated tfrmt (as JSON) into future sessions of R or {tfrmtbuilder} for further modification. Programmers and non-programmers alike can easily reproduce existing tables, make modifications, and save the result for further use.

CONCLUSION

The {tfrmt} and {tfrmtbuilder} packages transform clinical trial reporting by streamlining table creation and customization. The separation of analysis and reporting saves resources by concentrating QC on the results, independently of their final display. These packages empower non-programmers and foster collaboration across different roles and experiences through their language-agnostic capabilities. With {tfrmt} and {tfrmtbuilder}, organizations are better positioned for future success and facilitate the adoption of emerging data standards.

REFERENCES

{tfrmt}: https://gsk-biostatistics.github.io/tfrmt/
{tfrmtbuilder}: https://gsk-biostatistics.github.io/tfrmtbuilder/

DISCLAIMER

The content of this paper is my personal opinion and not that of GSK.

CONTACT INFORMATION

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

Author Name: Becca Krouse
Company: GSK
Email: becca.z.krouse@gsk.com