Paper OS03

gridify: Enrich Figures and Tables with Custom Headers and Footers

Alexandra Wall, UCB, Bristol, United Kingdom
Jennifer Winick-Ng, UCB, Toronto, Canada
Maciej Nasinski, UCB, Warsaw, Poland

INTRODUCTION

The pharmaceutical industry follows strict rules for presenting clinical trial data. Tables, listings, and figures (TLFs) must be accompanied by detailed information that ensures every output is traceable, reproducible, and compliant with standards. This information often includes company names, study details, dates, and references. The R packages used to create tables, listings, and figures are powerful, but they lack a common way to add these required details, which creates major operational hurdles.

This lack of a unified method causes several problems. Outputs using `ggplot2` for figures add annotations differently than those using `gt` or `flextable` for tables. This leads to inconsistent-looking reports, which can appear unprofessional. Furthermore, without a standard process, it is easy to forget required information, which can delay regulatory reviews or lead to costly resubmissions. The manual work needed to keep hundreds of outputs consistent in a clinical study is a significant drain on resources, taking programmers away from their main job of data analysis.

The `gridify` package solves these problems by providing a systematic way to enhance outputs using R's `grid` graphics system. By offering a single method for adding uniform text elements, such as headers, titles, captions, footnotes, and watermarks to different types of objects, `gridify` turns the creation of consistent reports from a manual craft into a streamlined process. This change brings clear benefits in consistency, efficiency, and quality, all while keeping the process fully transparent for audits.

THE POWER OF GRIDIFY

The value of `gridify` is clear in many areas of pharmaceutical software development. As part of the pharmaverse ecosystem, `gridify` fills a key gap in the reporting pipeline by offering a standard way to add required metadata. The package's Apache 2.0 license makes it widely available, and its strong testing and continuous integration provides the reliability needed for validated systems. Its design allows it to work smoothly with other pharmaverse packages.

From a user's point of view, `gridify` finds the right balance between being easy to use and being flexible. It offers simple functions that hide the complexity of `grid` programming but still provides enough options to meet different company needs. Pre-built pharmaceutical layouts help new users get started quickly, while the custom layout system lets organizations create reusable templates that match their specific branding and rules.

The "tinyverse" approach `gridify` takes is a commitment to keeping things simple and stable. By relying only on the base R `grid` and `methods` packages, `gridify` avoids the complex dependency issues that can cause problems in regulated software environments. This design choice makes installation easier, reduces version conflicts, and ensures long-term stability, all of which are vital for maintaining validated systems over the many years of a clinical program.

The package uses S4 object-oriented programming to create a well-structured and expandable design. The package's technical documentation provides clear details on its architecture for users and regulators who need to understand how the software works.

By using metaprogramming, `gridify` is more than just a simple utility; it is a framework that lets users see its internal workings. Users can get the exact `grid` code for any layout, which allows for perfect reproduction of an output without needing the `gridify` package at all. This transparency is very useful in a regulatory setting, where it might be necessary to show exactly how an output was made, even years later.

IMPLEMENTATION ARCHITECTURE

The `gridify` architecture has three main parts that work together to enhance outputs. The **normalization engine** takes various objects, like tables or plots, and converts them into a standard format called a grid graphical object, or "grob." This conversion keeps the original look of the output while making it possible to modify it with the `grid` system. The **layout management system** provides templates that define the structure for headers, footers, and

other elements. These layouts use S4 classes to ensure consistency while still allowing for customization. Finally, the **content injection interface** offers functions to add text or graphics to the layout, keeping a record of every change.

The workflow is designed to be predictable and traceable. First, `gridify` converts the input object into a grob and places it inside the chosen layout. The system then shows a diagram of the layout with empty cells, giving users a clear picture of what information is needed. Users then fill these cells using a series of `set_cell()` commands, and each command is recorded. The final object is a standard grob, which can be further modified with `grid` functions or saved to different image formats with full control over its size and resolution.

The package supports the most common table and figure creation tools used in the R community. It works directly with `ggplot2` graphics, `gt` tables, `flextable` objects, `rtables` objects (by first converting to `flextable`), `gtable` structures, and base R plots. It can also support other frameworks, like `rtables`, through conversion, with testing to ensure the output remains accurate. The export system gives users fine-grained control over output details like positioning, dimensions, and resolution, meeting the different needs of regulatory submissions, publications, and presentations. Outputs can be exported in PDF, PNG, TIFF, and JPEG formats. Markdown tools such as R Markdown and Quarto can be used to render DOC and HTML files.

Performance optimization ensures that `gridify` adds very little overhead and manages memory efficiently, which is important when processing the large, multi-page displays often used in clinical trial reports. Repetitive tasks can be automated by looping through lists of text elements for entry in `gridify` functions.

PRACTICAL APPLICATION

The following examples show how to use `gridify` in common pharmaceutical reporting tasks. They illustrate the complete workflow from data input to a final compliant output. The data are synthetic and come from random.cdisc.data. The labels for company and drug are artificial.

The process in each example follows four clear steps:

1. **Create the Core Object:** An analysis output, such as a table with `gt` or a plot with `ggplot2`, is generated first. This is the central piece of content.
2. **Initialize `gridify`:** This core object is then passed to the `gridify()` function, along with a layout (predefined or custom). This step converts the object into a `grid` grob and places it within the layout structure.
3. **Populate Metadata:** Using the `set_cell()` function, headers, footers, titles, and other required metadata are systematically added to the named cells of the layout.
4. **Render the Final Output:** The final object, now containing both the analysis and the contextual metadata, is printed to produce the complete, submission-ready output.

EXAMPLE 1: ENHANCING A GT TABLE

This example creates a clinical summary table with `gt` using `the random.cdisc.data::cadsl` dataset and then adds a standard pharmaceutical header and footer. See Appendix for the example code.

PharmaCorp International CONFIDENTIAL
ONCOLOGY / Advanced NSCLC Final
Study PC-2025-001 Data Cut-off: 2025-09-30

Table 14.1.1
Summary of Demographics and Baseline Characteristics
Baseline Biomarker 1 (BMRKR1) by Geographic Region
Safety Analysis Set

	Region	n	BMRKR1
A: Drug X	Africa	8	6.47 (4.70)
	Asia	91	6.00 (3.32)
	Eurasia	5	4.88 (1.61)
	Europe	4	5.44 (4.48)
	North America	13	6.84 (4.92)
	South America	13	5.18 (3.51)
B: Placebo	Africa	7	4.65 (1.92)
	Asia	94	5.97 (3.53)
	Eurasia	8	6.35 (3.89)
	Europe	3	4.17 (0.47)
	North America	15	4.21 (1.62)
	South America	7	6.22 (3.44)
C: Combination	Africa	11	6.57 (3.24)
	Asia	83	5.44 (3.53)
	Eurasia	6	3.58 (0.87)
	Europe	2	2.35 (0.42)
	North America	20	6.26 (4.13)
	South America	10	6.62 (2.71)

Note: BMRKR1 values shown as Mean (SD).

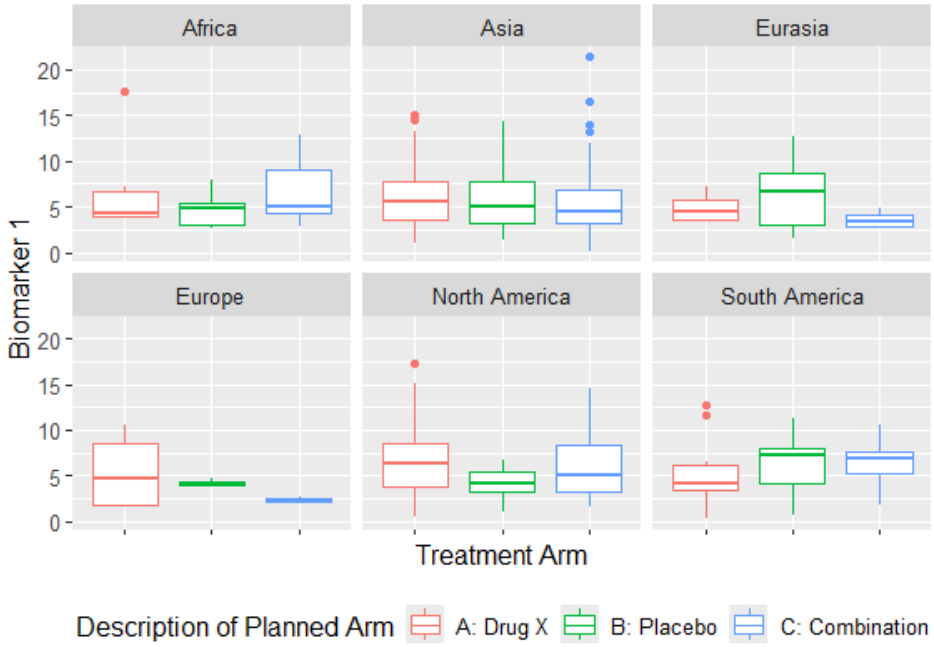
Source: ADSL dataset random.cdisc.data
Program: table_posit.R, 2025-10-08 15:43 CEST Page 1 of 1

EXAMPLE 2: ENHANCING A GGPLOT2 PLOT

This example displays how `gridify` can be applied to a `ggplot2` graphic using the `random.cdisc.data::cadsl` dataset, demonstrating that plots and tables share a consistent look. See Appendix for the example code.

PharmaCorp International CONFIDENTIAL
ONCOLOGY / Advanced NSCLC Final
Study PC-2025-001 Data Cut-off: 2025-09-30

Plot 12.1.1
Summary of Demographics and Baseline Characteristics
Baseline Biomarker 1 (BMRKR1) by Geographic Region
Safety Analysis Set



Source: ADSL dataset random.cdisc.data
Program: table_posit.R, 2025-10-08 15:43 CEST Page 1 of 1

These examples show how the `set_cell()` function provides a clear and auditable trail for adding each piece of information. The semantic cell names remove any confusion about where content should go, and `gridify` adds the new elements without altering the original table or plot.

ADVANCED IMPLEMENTATION

For ease, many predefined layouts are included in `gridify`, including the following:

- `simple_layout()`, which surrounds the output with a title and footer;
- `complex_layout()`, which includes headers and footers with left/middle/right positioning and titles and footnotes;
- `pharma_layout_base()`, `pharma_layout_letter()`, `pharma_layout_A4()`, which include common elements required in pharmaceutical reporting, with differences in margins.

Users may also define their own custom layouts, with the underlying mechanism behind `gridify` being the `grid` package. Layouts are simply grids.

Steps for creating a custom layout are as follows:

1. **Specify the grid**: This includes arguments for the number of rows and columns, their dimensions, and the margin surrounding them.
2. **Place the main object within the grid**: Using the `gridifyObject()` function, specify the rows and columns the object should span.

3. **Place the other cells in the grid**: Add remaining cells as named arguments around the main object with `gridifyCells()`. Each cell is defined by a call to the `gridifyCell()` function and allows specification of positioning and graphical parameters.
4. **Create the `gridify` layout object**: Use `gridifyLayout()` to combine the grid, object, and cell placements.

The custom layout is used in the same way as any of the predefined layouts. See Appendix for an example and the vignette on the website (vignette("create_custom_layout", package = "gridify")) for further details on creating custom layouts.

EDUCATIONAL RESOURCES AND DOCUMENTATION

The `gridify` package is supported by comprehensive documentation, available at **https://pharmaverse.github.io/gridify/**. This site serves as the central hub for all learning materials and is designed to help users of all experience levels master the framework.

The website guides users through a series of articles, starting with a foundational **"Get Started"** guide (vignette("gridify", package = "gridify")). This tutorial provides a practical introduction to the core concepts by walking through a complete example of enhancing clinical output. It explains key design principles and demonstrates best practices for achieving regulatory compliance.

Other key articles available on the site include a gallery of **Simple Examples** for various input types (vignette("simple_examples", package = "gridify")), instructions for developing **Custom Layouts** (vignette("create_custom_layout", package = "gridify")), guidance on creating **Multi-Page Outputs** (vignette("multi_page_examples", package = "gridify")), and a detailed explanation of the package's architecture in the **Transparency of gridify** article (vignette("transparency", package = "gridify")). This final article is crucial for regulated work, as it explains how to extract the underlying `grid` code to ensure long-term reproducibility.

The documentation is an excellent resource for structured training programs. Its modular articles allow trainers to develop curricula for different audiences, from programmers needing technical details to quality assurance personnel focused on validation principles. The hands-on code examples and clear explanations are ideal for both workshops and self-directed learning.

DISCUSSION

Using `gridify` in pharmaceutical development leads to clear improvements. It greatly reduces visual inconsistencies across reports, as standard layouts replace varied, manual formatting.

Separating the analysis from the presentation changes how updates are managed. If regulatory rules or company standards change, the layout can be updated without needing to re-validate the analysis code. This reduces the work involved in making formatting changes and helps organizations respond faster to feedback during regulatory reviews.

The transparent design of `gridify` meets regulatory demands for reproducibility and traceability. The ability to save the exact `grid` code used to create an output provides a permanent record for audits, even if `gridify` is not available in the future. This forward-thinking approach is a best practice in regulated software development, where product lifecycles can span decades.

The named layout cells also improve communication with regulators. Instead of referring to positions, clear terms like "header_company" or "footer_note" can be used. This reduces confusion and helps resolve issues more quickly during a submission review.

LIMITATIONS

While `gridify` is a powerful tool, there are a few limitations to consider. Currently, users need to write their own code for creating multi-page outputs, which can lead to different solutions across teams. A common workaround is to create shared functions for this, but built-in support would be better.

For some object types, `gridify` relies on conversion functions provided by the original packages to create a standard `grid` object. Specifically, it uses `gt::as_gtable`, `flextable::gen_grob`, `ggplot2::ggplotGrob`, and `grid::grid.grabExpr(gridGraphics::grid.echo)` for base R plots. This approach means `gridify` is not responsible for the conversion itself; it simply orchestrates these well-established functions. While this method is robust, teams should still verify that complex or highly customized outputs are converted with full fidelity, as the responsibility for the conversion accuracy lies with the underlying packages.

FUTURE DIRECTIONS

Future work on `gridify` will focus on built-in support for multi-page outputs and new community-proposed layouts. The built-in support for multi-page will improve the user experience even further. Community contributions to layouts would allow organizations to share specific regulatory requirements, promoting standardization across the industry. This community-driven model would help speed up adoption and reduce duplication of effort.

CONCLUSION

The `gridify` package offers a new and better way to produce regulatory-compliant TLF outputs in the pharmaceutical industry. By providing a single framework for adding headers and footers to different kinds of outputs, `gridify` turns a difficult, manual task into a streamlined and reproducible workflow. Its foundation on R's `grid` system ensures stability, and its transparent design meets the strict audit requirements of regulated work.

The benefits of using `gridify`, such as better consistency, less manual effort, and improved quality, lead directly to faster submission times and lower regulatory risk. As the package grows with new features, it will become an even more essential tool for pharmaceutical reporting. By adopting `gridify`, organizations can meet today's regulatory demands and stay ready for future changes in reporting standards.

The success of `gridify` shows the power of focused solutions for specific industry problems. By tackling the important challenge of output enhancement, `gridify` provides a great deal of value from a small package. This, along with its excellent documentation and community support, makes `gridify` a model for effective tool development in the pharmaceutical industry.

APPENDIX

CODE FOR EXAMPLE 1

```
# Setup
library(dplyr)
library(gt)
library(gridify)
library(random.cdisc.data)
library(scales)

primary_color <- "#08306b"
secondary_color <- "#6baed6"

# Data Preparation
adsl <- random.cdisc.data::cadsl |>
  mutate(ARM = factor(ARM), REGION1 = factor(REGION1))
summary_tbl <- adsl |>
  group_by(ARM, REGION1) |>
  summarise(
    N = n(),
    MEAN_BMRKR1 = mean(BMRKR1, na.rm = TRUE),
    SD_BMRKR1 = sd(BMRKR1, na.rm = TRUE),
    .groups = "drop"
  ) |>
  mutate(MEAN_BMRKR1 = round(MEAN_BMRKR1, 2), SD_BMRKR1 = round(SD_BMRKR1, 2)) |>
  mutate(BMRKR1 = factor(sprintf("%0.2f (%0.2f)", MEAN_BMRKR1, SD_BMRKR1))) |>
  mutate(BMRKR1 = factor(BMRKR1, levels = BMRKR1[order(MEAN_BMRKR1)])) |>
  select(ARM, REGION1, N, BMRKR1)

# gt Table
gt_tbl <- summary_tbl |>
  gt(rowname_col = NULL, groupname_col = "ARM") |>
  cols_label(REGION1 = "Region", N = "n", BMRKR1 = "BMRKR1") |>
  cols_width(ARM ~ pct(30), REGION1 ~ pct(30), N ~ pct(10), BMRKR1 ~ pct(30)) |>
  cols_align(align = "center", columns = N) |>
  tab_options(
    table.width = pct(70),
    table.font.size = 12,
    data_row.padding = px(6),
    table.font.color = "black",
    table.font.weight = "400",
    heading.align = "left",
```

```
    table.background.color = "gray98",
    heading.background.color = "gray98",
    column_labels.background.color = "gray95",
    row_group.font.weight = "bold",
    row_group.as_column = TRUE
  ) |>
  opt_row_striping() |>
  data_color(
    columns = BMRKR1,
    colors = col_factor(
      palette = c("white", secondary_color, primary_color),
      domain = NULL
    )
  ) |>
  data_color(
    columns = N,
    colors = col_factor(
      palette = c("white", secondary_color, primary_color),
      domain = NULL
    )
  ) |>
  tab_style(
    style = cell_text(weight = "bold"),
    locations = list(cells_column_labels(), cells_title())
  ) |>
  tab_style(
    style = cell_text(color = primary_color),
    locations = cells_column_labels()
  ) |>
  tab_style(
    style = cell_text(color = primary_color),
    locations = cells_title(groups = "title")
  )

# Create gridify object with predefined gridify pharma layout
pharma_gt <- gridify(
  gt_tbl,
  pharma_layout_base(
    margin = grid::unit(c(t = 0.5, r = 0.5, b = 0.5, l = 0.5), "inch"),
    global_gpar = grid::gpar(
      fontfamily = "mono",
      fontsize = 10,
      col = "black"
    )
  )
) |>
  set_cell("header_left_1", "PharmaCorp International") |>
  set_cell("header_left_2", "ONCOLOGY / Advanced NSCLC") |>
  set_cell("header_left_3", "Study PC-2025-001") |>
  set_cell("header_right_1", "CONFIDENTIAL") |>
  set_cell("header_right_2", "Final") |>
  set_cell("header_right_3", "Data Cut-off: 2025-09-30") |>
  set_cell("output_num", "Table 14.1.1") |>
  set_cell("title_1", "Summary of Demographics and Baseline Characteristics") |>
  set_cell("title_2", "Baseline Biomarker 1 (BMRKR1) by Geographic Region") |>
  set_cell("title_3", "Safety Analysis Set") |>
  set_cell("note", "Note: BMRKR1 values shown as Mean (SD).") |>
  set_cell("references", "Source: ADSL dataset random.cdisc.data") |>
  set_cell(
    "footer_left",
    sprintf("Program: table_posit.R, %s", format(Sys.time(), "%Y-%m-%d %H:%M %Z"))
  ) |>
  set_cell("footer_right", "Page 1 of 1") |>
  print()
```

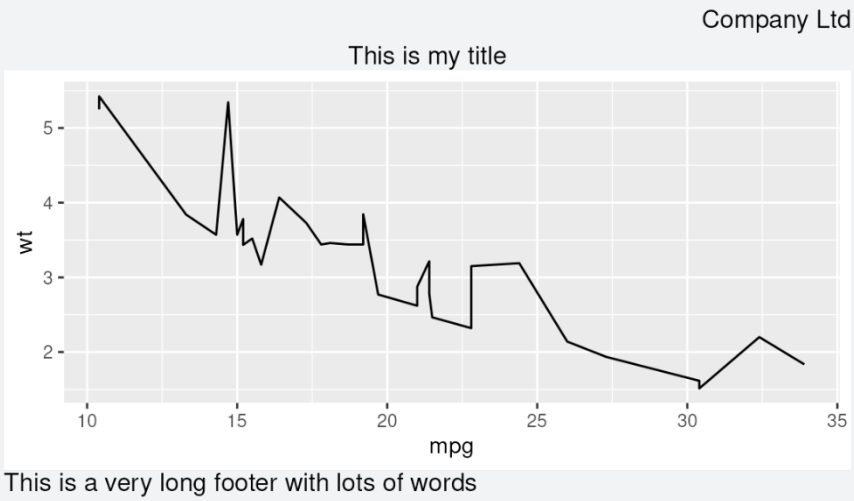
CODE FOR EXAMPLE 2

```r
# Setup
library(ggplot2)
library(gridify)
library(random.cdisc.data)

current_time <- format(Sys.time(), "%Y-%m-%d %H:%M %Z")

# Create a boxplot of lab values by treatment arm
gg_obj <- ggplot(cadsl, aes(x = ARM, y = BMRKR1, col = ARM)) +
  geom_boxplot(na.rm = TRUE) +
  facet_wrap(~REGION1) +
  labs(
    x = "Treatment Arm",
    y = "Biomarker 1"
  ) +
  theme(
    legend.position = "bottom",
    text = element_text(size = 12),
    axis.text.x = element_blank()
  )

# Create gridify object with predefined gridify pharma layout
pharma_gg <- gridify(
  gg_obj,
  pharma_layout_base(
    margin = grid::unit(c(t = 0.5, r = 0.5, b = 0.5, l = 0.5), "inch"),
    global_gpar = grid::gpar(
      fontfamily = "mono",
      fontsize = 10,
      col = "black"
    )
  )
) |>
  set_cell("header_left_1", "PharmaCorp International") |>
  set_cell("header_left_2", "ONCOLOGY / Advanced NSCLC") |>
  set_cell("header_left_3", "Study PC-2025-001") |>
  set_cell("header_right_1", "CONFIDENTIAL") |>
  set_cell("header_right_2", "Final") |>
  set_cell("header_right_3", "Data Cut-off: 2025-09-30") |>
  set_cell("output_num", "Plot 12.1.1") |>
  set_cell("title_1", "Summary of Demographics and Baseline Characteristics") |>
  set_cell("title_2", "Baseline Biomarker 1 (BMRKR1) by Geographic Region") |>
  set_cell("title_3", "Safety Analysis Set") |>
  set_cell("references", "Source: ADSL dataset random.cdisc.data") |>
  set_cell("footer_left", sprintf("Program: table_posit.R, %s", current_time)) |>
  set_cell("footer_right", "Page 1 of 1") |>
  print()
```

CODE FOR USER-DEFINED LAYOUT

```r
# Define custom layout
new_layout <- gridifyLayout(
  # specify the grid
  nrow = 4L,
  ncol = 2L,
  heights = grid::unit(c(1, 1, 1, 0.05), c("lines", "lines", "null", "npc")),
  widths = grid::unit(c(0.5, 0.5), "npc"),
  margin = grid::unit(c(t = 0.1, r = 0.1, b = 0.1, l = 0.1), units = "npc"),
  global_gpar = grid::gpar(),
  adjust_height = FALSE,
  # Place the main object
  object = gridifyObject(row = 3, col = 1:2),
  # Place the other cells
  cells = gridifyCells(
    company = gridifyCell(row = 1, col = 2, x = 1, hjust = 1, y = 1, vjust = 1),
    title = gridifyCell(row = 2, col = 1, x = 1, hjust = 0.5),
```

```
    footer = gridifyCell(row = 4, col = 1, x = 0, hjust = 0)
  )
)

# Use custom layout
gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = new_layout
) |>
  set_cell("company", "Company Ltd") |>
  set_cell("title", "This is my title") |>
  set_cell("footer", "This is a very long footer with lot of words")
```



REFERENCES

- Iannone R, Cheng J, Schloerke B, Hughes E, Lauer A, Seo J, Brevoort K, Roy O (2025). *gt: Easily Create Presentation-Ready Display Tables*. R package version 1.0.0, https://github.com/rstudio/gt, https://gt.rstudio.com.
- Murrell, P. (2018). R Graphics, Third Edition (3rd ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9780429422768
- Nasinski M, Wall A, Robson S, Dash P, Winick-Ng J (2025). *gridify: Enrich Figures and Tables with Custom Headers and Footers and More*. R package version 0.7.4, https://pharmaverse.github.io/gridify/.
- R Core Team (2024). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/
- H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.
- Wickham H, François R, Henry L, Müller K, Vaughan D (2023). *dplyr: A Grammar of Data Manipulation*. R package version 1.1.4, https://github.com/tidyverse/dplyr, https://dplyr.tidyverse.org.
- Rucki P, Paszty N, Stoilova J, Zhu J, Garolini D, de la Rua E, DiPietrantonio C, Waddell A (2024). *random.cdisc.data: Create Random ADaM Datasets*. R package version 0.3.16, https://github.com/insightsengineering/random.cdisc.data/, https://insightsengineering.github.io/random.cdisc.data/.

ACKNOWLEDGEMENTS

The authors thank the `gridify` development team, the pharmaverse community, and the wider R pharmaceutical user community for their contributions, feedback, and support. Special thanks go to the early adopters whose feedback helped improve the design and to the regulatory professionals whose insights guided the compliance-focused features. This project is a great example of how open-source collaboration can solve industry-specific challenges.

CONTACT INFORMATION

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

Author: Alexandra Wall
Company: UCB
Email: alexandra.wall@ucb.com
Author: Jennifer Winick-Ng
Company: UCB
Email: jennifer.winick-ng@ucb.com
Author: Maciej Nasinski
Company: UCB
Email: maciej.nasinski@ucb.com
Website: https://www.ucb.com/