

## Building an Internal R Package for Statistical Analysis and Reporting in Clinical Trials: A SAS User's Perspective

Huei-Ling Chen, Heng Zhou, Nan Xiao  
Merck & Co., Inc., Rahway, NJ, USA

### ABSTRACT

The programming language R has seen an increase in usage in the analysis and reporting sector of the pharmaceutical industry. Similar to how SAS programmers regularly write SAS macros, it is common for R users to write R functions to complete repetitive tasks, thus facilitating programming work. An R package is similar to a well-built SAS macro library; this includes a collection of functions, instruction documentation, sample data, and testing code with validation evidence. An R package formalizes access to the R functions. Yet new users may find a steep learning curve associated with creating an R package from scratch. This paper outlines the essential components of an R package and the valuable tools to help create these components. Relevant online reference materials are provided as well.

### KEYWORDS

R Packages, devtools, roxygen2, RStudio IDE, R Markdown, pkgdown

### INTRODUCTION

R provides many open-source packages for data processing, analysis, and visualization. In addition, we also need to develop internal R packages for the company or therapeutic area standard. Similar to a well-maintained SAS macro library, a purpose-built R package should equip with all the essential components to serve clinical statistical analyses.

Zhu et al. (2020) offered process guidelines for developing and deploying internal R packages. Their guidance covers the development, validation, documentation, and compliance checks with essential regulatory requirements. Our paper will demonstrate an example of building an internal R package achieving the industry standard based on this reference.

Early-phase oncology studies often use dose-limiting toxicity (DLT) as the primary endpoint to determine the maximum tolerated dose (MTD), where a specific design is needed to guide the dose-finding process. This paper presents an internal R package, *mkdosefinding*, which delivers the modified Toxicity Probability Interval (mTPI) design decision table and the DLT analysis summary table. Step-by-step instructions on creating R functions, testing code, documentation, package metadata, example data, R Markdown vignettes, and package website for the *mkdosefinding* package are also provided.

The R package *mkdosefinding* has two functions for generating the mTPI table and five functions for carrying out the DLT analysis summary table.

#### mTPI table

- `get_decision_mtpi()`
- `mtpi_tbl()`

#### DLT analysis summary table

- `get_pava_table()`
- `pava_tbl()`
- `pava()`
- `pava_2d()`
- `bci()`

## ESSENTIAL COMPONENTS OF AN R PACKAGE

	Folder Location	File Type	Examples
R functions	R/	.R	get_decision_mtpi.R
Testing code	tests/testthat/	.R	test-independent-testing-get_decision_mtpi.R
Documentation	man/	.Rd	get_decision_mtpi.Rd
Package metadata	root directory	Various	DESCRIPTION, README.md
Example data	data/	.rda	mkdosefinding_adae.rda
R Markdown vignettes	vignettes/	.Rmd	decidision0mtpi0table.Rmd
pkgdown Website	docs/	Various	index.html, references.html, articles.html

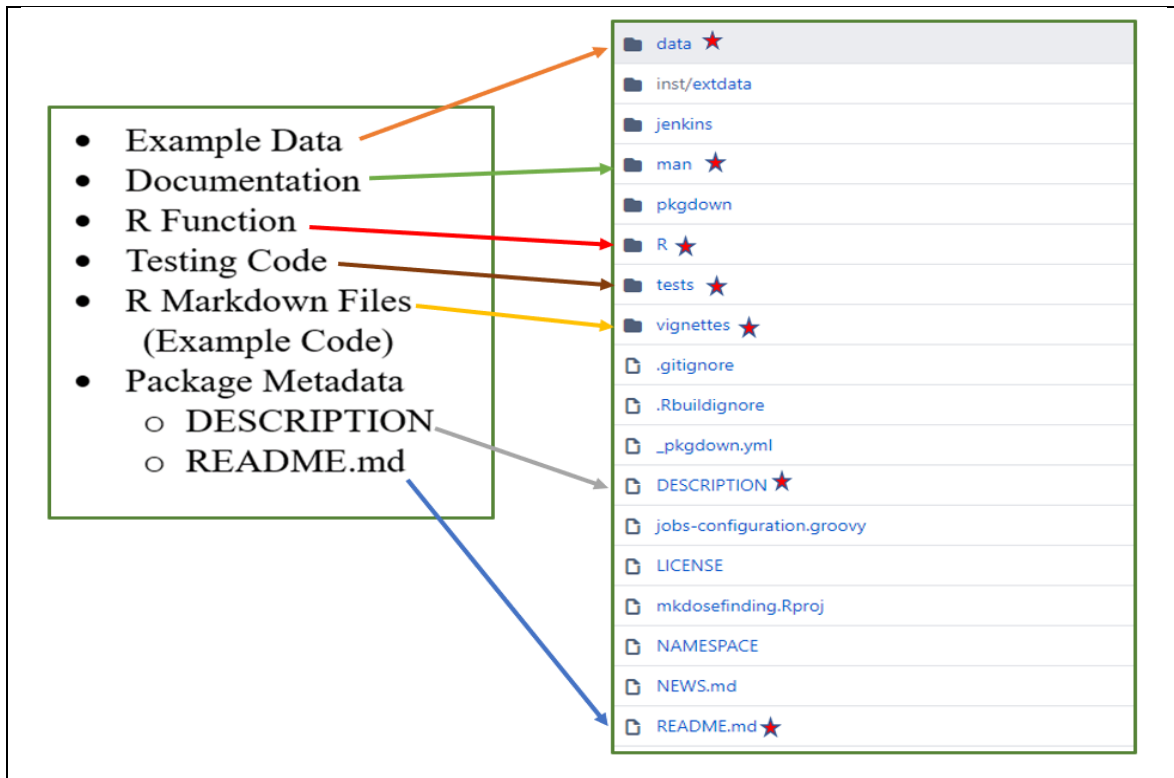


Figure 1. Components of an R package and their locations.

## KEY TOOLS TO CREATE AN R PACKAGE

To create an R package, the first thing is to ensure that the essential tools are installed in the development environment. In particular, a few R packages can facilitate and accelerate the package development process.

- [devtools](#) provides R functions that simplify and automate common tasks in package development using best practices, for example, `document()`, `load_all()`, `test()`, `test_coverage()`, and `check()`.
- [roxygen2](#) helps preparing and generating the documentation of the package. It processes the R code and formatted comments to produce R's documentation files (.Rd) in the `man/` directory.
- [testthat](#) offers a framework to write and run tests.
- [usethis](#) is a workflow package which makes repetitive development tasks such as file structure scaffolding easier. The function `usethis::create_package()` creates

template files and folders for project setup and development. The function `usethis::use_test()` creates individual test files during project development.

- [rmarkdown](#) creates high-quality HTML or PDF vignettes. Vignettes can combine code, rendered output (such as figures), and long-form guide to provide extra insight of the work.

For people unfamiliar with R package development, it is worth starting the process with the package `devtools` which also installs most of the other key tool packages automatically. Then, install the missing packages as needed.

In cases where a needed package is not in your R library, for example, `rmarkdown`, you can use `install.packages("rmarkdown")` to install it from the CRAN or an internal CRAN-like repository.

## R FUNCTIONS

A SAS programmer will find familiarity when writing an R function. An R function includes four key elements: the function name, the function arguments, the R statements that the function runs, and the object returned by the R function.

The input arguments of an R function are similar to the parameters of a SAS %macro statement. An R function can have multiple arguments defined, separated by a comma, similarly to SAS macro parameters. An R argument can have a default value.

Note that there are still some differences between an R function and a SAS macro. For example, an R function returns one object while no such requirement exists for a SAS %macro statement.

Develop the functions following the structure below.

```
myfunction <- function(arg1, arg2, ... ){  
  statements  
  return(object)  
}
```

After the R function development is completed and the function works fine, we can use the `roxygen2` package to create specially formatted comments, which will be utilized to generate R's documentation files (`.Rd`). To add the `roxygen` comments, insert the `roxygen skeleton` before the function definition. The following steps, demonstrated in Figure 2, create a header template for the function.

1. Select to highlight the function name.
2. Click 'Code'.
3. Click 'Insert Roxygen Skeleton'.

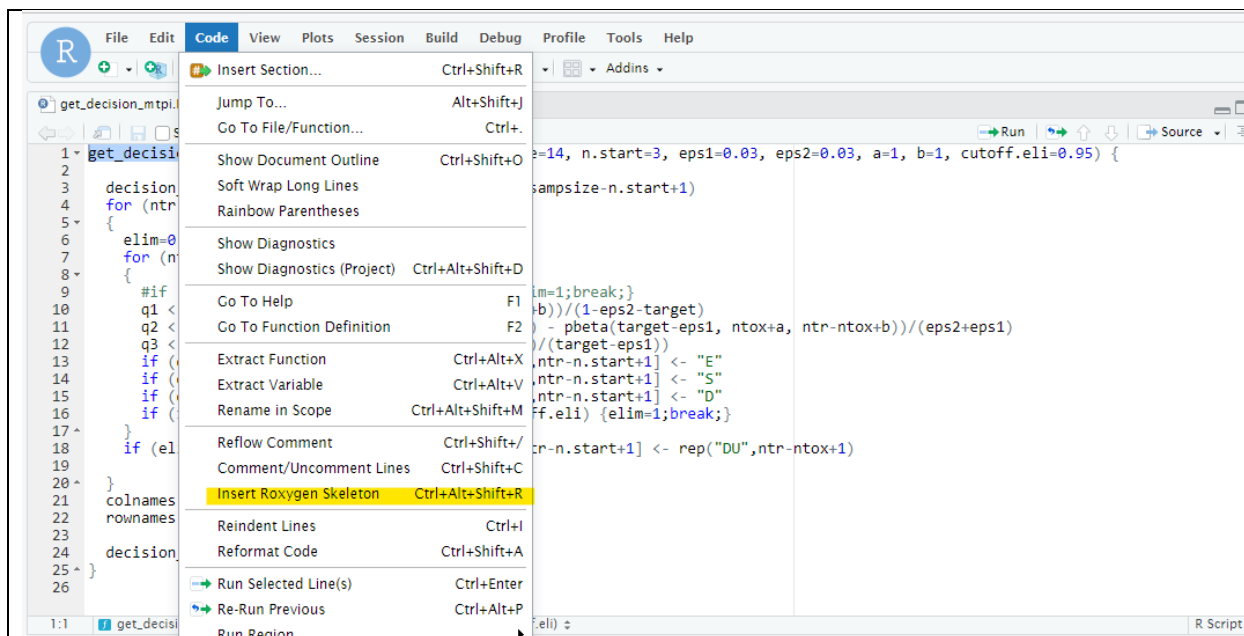


Figure 2. Insert roxygen skeleton for a defined R function.

After completing the step in Figure 2, a template with `roxygen` tags will be automatically created above the R function.

The next step is to fill out the relevant information in the `roxygen` skeleton. This often includes **title**, **description**, **author**, **parameter description**, **returns**, and **code examples**.

Figure 3 is the final look of an R function with the `roxygen` comments built in the file.

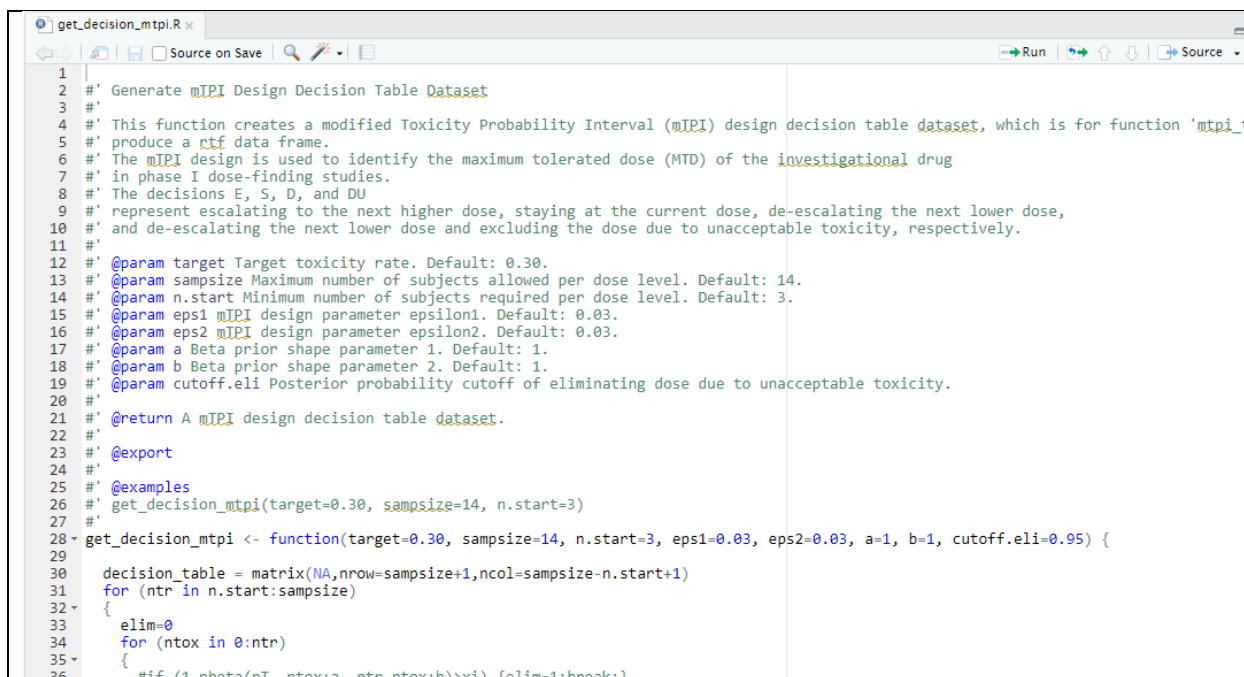


Figure 3. R function with the roxygen comments built in the file.

Yang et al. (2021) introduced a strategy using `roxygen2` to develop specifications for R functions in regulated clinical trial environments. It introduced concepts around R functions and `roxygen`

comments. It also demonstrated how the `roxygen` documentation approach can be extended to easily generate the R document file and assist creating the documentation for the package.

## TESTING AND VALIDATION

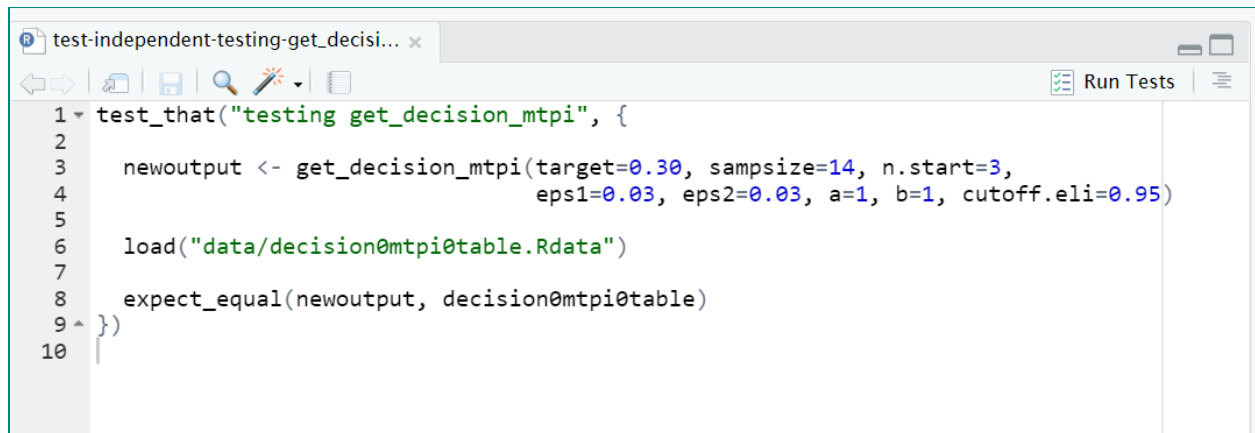
Validation is a critical part for an internally developed R package. Like every statistical software used in clinical trials, all programming works need to be robust and reusable. In R, a user-friendly testing framework is provided by the R package `testthat`, which has many testing functions to ensure the R programs work as expected. Among those `expect_*()` functions, the most frequently used functions are `expect_equal()`, `expect_error()`, and `expect_snapshot_file()`.

Ginnaram et al. (2021) proposed a process to validate internal R packages under a regulatory-compliant computing environment, utilizing both `testthat` and another useful package `usethis` to facilitate the validation work. In the proposal, each R function should have its correspondent testing file. All testing files are saved in the `tests/testthat/` folder. For example, “`get_decision_mtpi.R`” has a testing file “`test-independent-testing-get_decision_mtpi.R`” in the `tests/testthat/` folder.

The command below shows how to use the `use_test()` function to set up the testing file “`test-independent-testing-get_decision_mtpi.R`” in the `tests/testthat/` folder.

```
usethis::use_test("test-independent-testing-get_decision_mtpi")
```

Select the appropriate `testthat::expect_*()` function to complete the testing work.



```
1 test_that("testing get_decision_mtpi", {
2
3   newoutput <- get_decision_mtpi(target=0.30, sampsize=14, n.start=3,
4                                 eps1=0.03, eps2=0.03, a=1, b=1, cutoff.eli=0.95)
5
6   load("data/decision0mtpi0table.Rdata")
7
8   expect_equal(newoutput, decision0mtpi0table)
9 ^ })
10 |
```

Figure 4. Testing code in “`test-independent-testing-get_decision_mtpi.R`”

Palukuru et al. (2022) discussed more advanced `testthat` features in detail, and provided clear demonstrations regarding how to utilize R as a validation tool to its greatest capability.

## DOCUMENTATION

The main benefits of using `roxygen2` for R function documentation are that the R document (`.Rd`) files can be automatically generated from comments written in a simple syntax by running one R command, `devtools::document()`. This function converts the `roxygen` comments for each function, and creates individual `Rd` files under the `man/` folder. Figure 5 is a screenshot of the subfolder `man/` of the package.

More details and discussions regarding the documentation of an R package can be found in Yang et al. (2021).

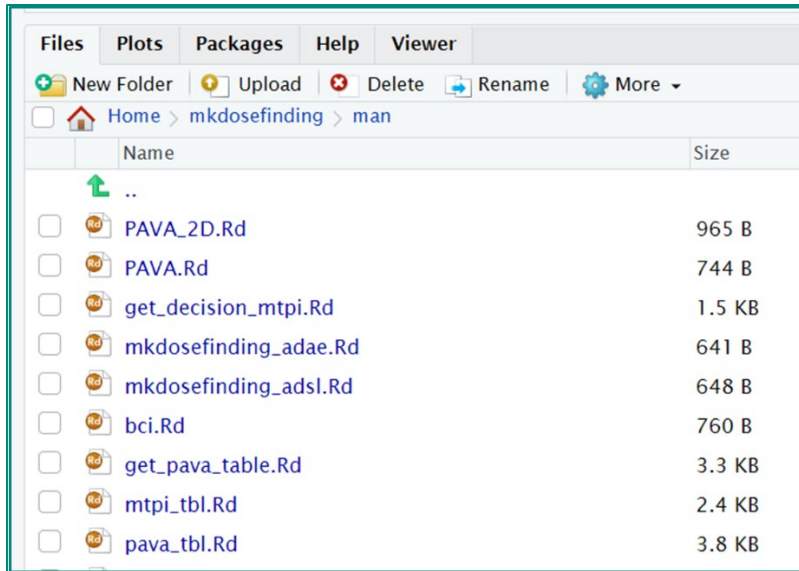


Figure 5. The .Rd files in the subfolder man/

## PACKAGE METADATA

Populate the content of the following two package metadata files: DESCRIPTION and README.md.

These package metadata files can provide high-level description of the package, author list, dependencies, and optionally, basic examples. Note that the information on the metadata files will be automatically parsed and displayed in the package website when it is created.

```

1 Package: mkdosefinding
2 Type: Package
3 Title: Dose-finding Trial with mTPI Design
4 Version: 0.1.0
5 Authors@R: c(
6   person("Heng", "Zhou", email = "heng.zhou@merck.com", role = c("aut", "cre")),
7   person("Huei-Ling", "Chen", email = "huei-ling_chen@merck.com", role = "aut"),
8   person("Zhen", "Zeng", role = "aut"),
9   person("Lily", "Zhang", role = "ctb"),
10  person("BARDS, Merck & Co., Inc.", role = c("cph"))
11 )
12 Description: BARDS Project Specific R package "mkdosefinding" to create early
13 oncology 'Summary of Dose Limiting Toxicity' table and mtpi table.
14 Depends: R (>= 3.6.0)
15 License: file LICENSE
16 Imports:
17   iso,
18   dplyr,
19   magrittr,
20   r2rtf,
21   tibble
22 Suggests:
23   covr,
24   haven,
25   kableExtra,
26   knitr,
27   pkgdown,
28   readxl,
29   rmarkdown,
30   testthat (>= 3.0.0)
31 Encoding: UTF-8
32 LazyData: true
33 VignetteBuilder: knitr
34 Config/testthat/edition: 3
35 Roxygen: list(markdown = TRUE)
36 RoxygenNote: 7.1.2

```

Figure 6. The DESCRIPTION file

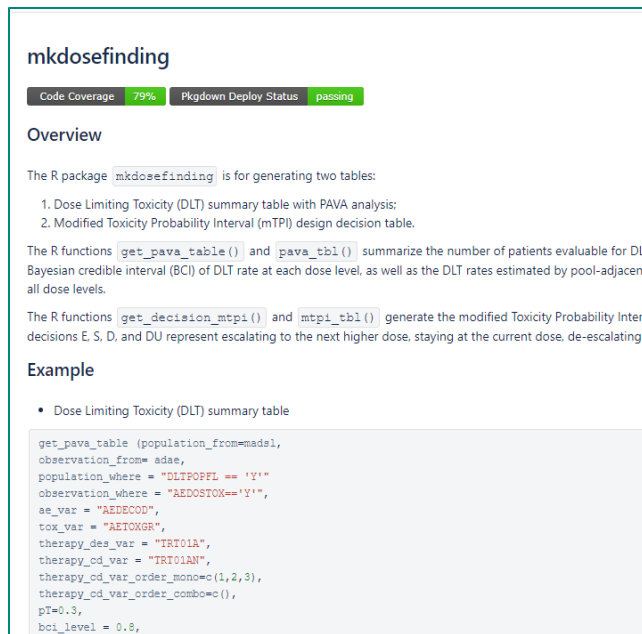


Figure 7. The README.md file

## EXAMPLE DATA

Example data is an optional component for the package. Including example datasets in the package can guide users on how to use the package, especially when the package is for data analysis and reporting.

Following the guidance from Wickham, H. (2015), the dataset '.rda' files are saved in the `data/` directory.

The command below uses the `usethis::use_data()` function to convert the R object into a '.rda' file and save the file in the `data/` directory.

```
usethis::use_data(mkdosefinding_adsl)
```

Figure 8 is the screen shot of the subfolder 'data/' for the package.

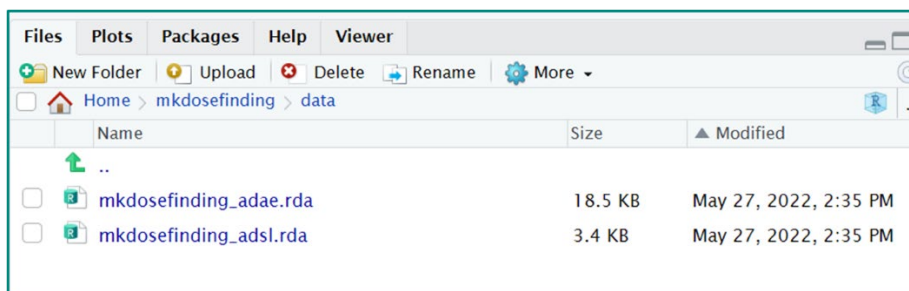


Figure 8. The .rda files created in the subfolder data/.

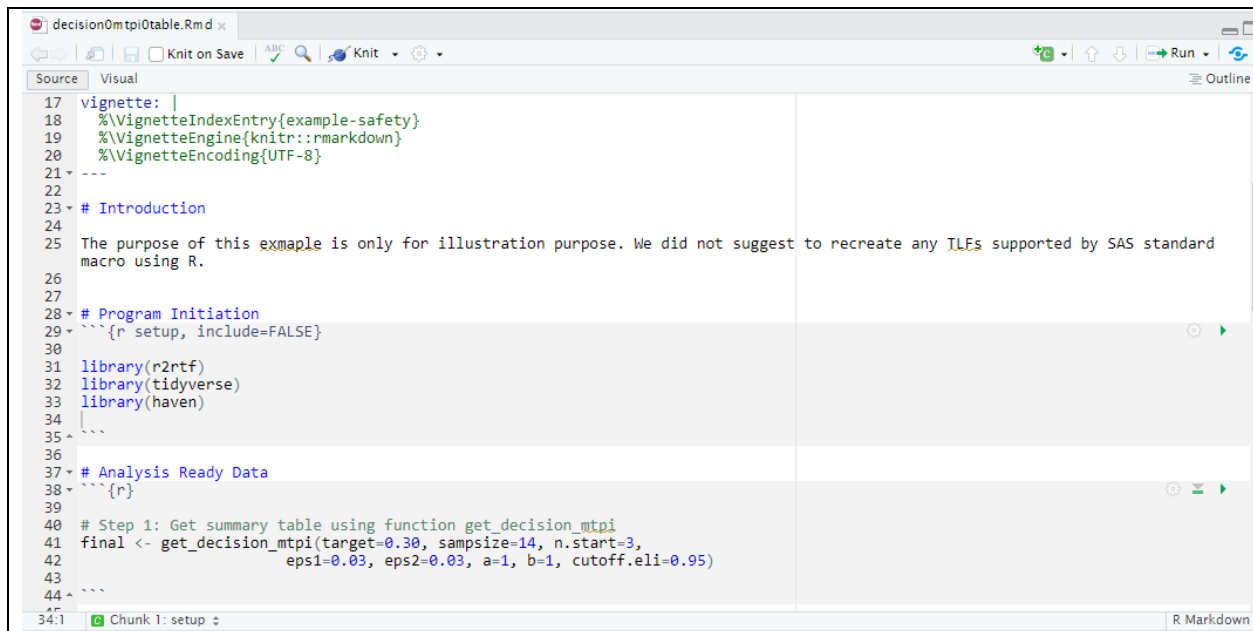
## R MARKDOWN VIGNETTES

The directory `vignettes/` stores long-form documentation to guide users on how to use the package. Many existing packages have vignettes for this purpose. We can create vignettes with R code examples for this guiding purpose and save them in the package `vignettes/` directory. These documentations use the R Markdown (.Rmd) file format. In brief, the R Markdown files stored in

vignettes/ can provide reproducible R code to execute the R functions, similar to a SAS call program to execute SAS macros.

Wu et al. (2021) proposed a workflow for generating analysis and reporting deliverables in clinical trials using R. They suggested users to utilize the R Markdown format instead of the plain R scripts for the reporting. The R Markdown file can include both the natural language descriptions of the analysis, the R code, and the computation results. The R Markdown file can be rendered into PDF or HTML outputs as a documentation for the package.

Figure 9 is a screenshot of a partial R Markdown vignette.



```
17 vignette: |
18   %\VignetteIndexEntry{example-safety}
19   %\VignetteEngine{knitr::rmarkdown}
20   %\VignetteEncoding{UTF-8}
21 - ---
22
23 # Introduction
24
25 The purpose of this example is only for illustration purpose. We did not suggest to recreate any TLEs supported by SAS standard
26 macro using R.
27
28 # Program Initiation
29 ```{r setup, include=FALSE}
30
31 library(r2ntf)
32 library(tidyverse)
33 library(haven)
34
35 - ---
36
37 # Analysis Ready Data
38 ```{r}
39
40 # Step 1: Get summary table using function get_decision_mtpi
41 final <- get_decision_mtpi(target=0.30, sampsize=14, n.start=3,
42   eps1=0.03, eps2=0.03, a=1, b=1, cutoff.eli=0.95)
43
44 - ---
45
34:1 [OK] Chunk 1: setup c
```

Figure 9. The R Markdown file created in the subfolder 'vignettes/'

The R Markdown files provide reproducible R code examples to execute the R functions for package users. Programmers only need to make minimal, necessary changes to the example workflow and can create the mTPI table or the DLT analysis summary table easily.

## CHECK AN R PACKAGE

It is important to conduct compliance checking when all components of a package are completed. The first useful metric is from the R CMD check results where more than 50 individual checks for common issue are presented. An ideal goal is to have 0 errors, 0 warnings, and 0 notes from the check results. The other metric, test coverage, is often used to understand how much of the R code is tested by the unit tests. Although it cannot reflect whether the quality of the tests, a higher percentage number for test coverage often indicates the package is more comprehensively tested.

Leveraging the comprehensive checks provided by the helper functions in devtools, we can frequently check the package during development to maintain its quality. An example checking flow could involve loading the package with devtools::load\_all(), run all tests or individual test files in the package with devtools::test() or devtools::test\_active\_file(), compute test coverage for the package with devtools::test\_coverage(), and run automatic checks with devtools::check().

## CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY

When developing an R package collaboratively, it is imperative to integrate the team's work frequently, in a version controlled repository. The repository should be configured with effective branch management



strategies and permission models. For example, the main branch of a Git repository should be protected by preventing changes without a pull request. A list of default reviewers who will be automatically assigned to new pull requests could facilitate the prompt review of contributions. Merge checks, such as requiring at least one reviewer's approval and passing all continuous integration workflows, are also frequently applied.

In particular, continuous integration (CI) workflows for building and checking the package should be triggered by each the creation of new pull request and any follow-up commits. The CI workflow checking report linked in each pull request helps the developers and reviewers to identify obvious technical issues early. The content of the CI workflows can be added to the R package and tailored for specific needs. For instance, running `usethis::use_github_actions()` will add GitHub Actions workflows as YAML files, which can be further modified to adding missing system dependencies, or adding more code style checks.

The continuous delivery (CD) of R packages in an organization can be automated with appropriate infrastructure. For example, a Git repository can be connected to a package or artifact management system, which will pull the source package after each new pull request is merged to the main branch. The system can then build the package and publish to an internal CRAN-like repository for testing purposes but with easier installation. Production releases of the package are manually submitted following an internal SDLC and are published to a separate internal CRAN-like repository after explicit stakeholder approval, similar to the CRAN review and inspection processes.

## OPTIONAL: CREATE A PKGDOWN WEBSITE

Although recommended, it is optional and up to the package developer's decision whether to create a `pkgdown` website for an internal package. The function `pkgdown::build_site()` builds a `pkgdown` website. The rendered website is saved in the `docs/` folder in your R package by default.

## USE THE PACKAGE TO CREATE A MTPI TABLE

When statisticians or programmers carry out an early oncology dose confirmation study, they can install the internal R package `mkdosefinding` into their project folder. To install the package, use the command below.

```
install.packages("mkdosefinding", repos = "https://cran-like-repo-url/")
```

Then, they can copy the content of the R Markdown vignettes and make necessary changes, e.g., output file name, argument value, etc. Finally, run the R Markdown file and the output file will be created in the designated output folder immediately. Figure 10 is the snapshot of an mTPI table output.

### Dose-finding Rules per mTPI Design

Number of Participants with at least 1 DLT	Number of Participants Evaluable for DLT at Current Dose											
	3	4	5	6	7	8	9	10	11	12	13	14
0	E	E	E	E	E	E	E	E	E	E	E	E
1	S	S	S	E	E	E	E	E	E	E	E	E
2	D	S	S	S	S	S	S	S	E	E	E	E
3	DU	DU	D	S	S	S	S	S	S	S	S	S
4		DU	DU	DU	D	D	S	S	S	S	S	S
5			DU	DU	DU	DU	DU	D	S	S	S	S
6				DU	DU	DU	DU	DU	DU	D	S	S
7					DU	DU	DU	DU	DU	DU	DU	D
8						DU	DU	DU	DU	DU	DU	DU
9							DU	DU	DU	DU	DU	DU
10								DU	DU	DU	DU	DU
11									DU	DU	DU	DU
12										DU	DU	DU
13											DU	DU
14												DU

Abbreviations: D = De-escalate to the next lower dose; DU= The current dose is unacceptably toxic; E = Escalate to the next higher dose; S = Stay at the current dose.  
 Target toxicity rate= 30%  
 Flat noninformative prior Beta (1,1) is used as a prior and  $\epsilon_1=\epsilon_2=0.03$  [Ji Y, Li Y, Bekele BN 2007] [Ji, Y. and Wang, S.-J. 2013] [Ji, Y., et al 2010]

Figure 10. Example mTPI table output.

## CONCLUSION

Like SAS, R programming works used in clinical trials need to be robust, reusable, and well validated. A R package would serve these purposes. This paper lists the essential and optional components of an R package: R functions, testing code, documentation, package metadata, example data, R Markdown vignettes, and package website. A step-by-step instruction explains how to build these components of an R package using the RStudio Integrated Development Environment (IDE). We demonstrate this work by using a package developed for early oncology dose confirmation study as an example. Instructions on how to leverage the package to perform the analysis and generate the final report are also provided.

## REFERENCES

### R packages creation tutorial

- Wickham, H. (2015). R packages: organize, test, document, and share your code. O'Reilly Media, Inc.
- R Packages: [R Packages \(2e\) \(r-pkgs.org\)](#)
- [Build websites for R packages • pkgdown \(r-lib.org\)](#)

### Proceedings

- Palukuru, P., Li, R., Patel, N., & Shi, C. (2022). "Exploring R as a validation tool for statistical programming in clinical trials" Proceedings of PharmaSUG 2022. <https://www.pharmasug.org/proceedings/2022/AD/PharmaSUG-2022-AD-076.pdf>
- Zhu, Y., Jajoo, R., Bai, C., Nepal, S., Woodie, D., Anderson, K., Zhang, Y., (2020). "R Package Oriented Software Development Life Cycle in Regulated Clinical Trial Environments" Proceedings of PHUSE US 2020. <https://www.lexjansen.com/phuse-us/2020/tt/TT12.pdf>
- Yang, A., Zhu, Y., & Zhang, Y. (2021). "A Strategy to Develop Specification for R Functions in Regulated Clinical Trial Environments" Proceedings of PharmaSUG 2021. <https://www.pharmasug.org/proceedings/2021/SI/PharmaSUG-2021-SI-074.pdf>
- Wu, P., Palukuru, P., Luo, Y., Nepal, S., & Zhang, Y. (2021). "Analysis and Reporting in Regulated Clinical Trial Environment using R" Proceedings of PharmaSUG 2021. <https://www.pharmasug.org/proceedings/2021/AD/PharmaSUG-2021-AD-079.pdf>

- Ginnaram, M., Ye, S., Zhu, Y., & Zhang, Y. (2021). "A Process to Validate Internal Developed R Package under Regulatory Environment" Proceedings of PharmaSUG 2021.  
<https://www.pharmasug.org/proceedings/2021/SI/PharmaSUG-2021-SI-084.pdf>

## ACKNOWLEDGMENTS

The authors would like to thank subject matter experts from the R strategic initiative and management teams from Merck & Co., Inc., Kenilworth, NJ, USA, for their advice on this paper/presentation.

## CONTACT INFORMATION

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

Huei-Ling Chen, Ph.D.  
Merck & Co., Inc., Rahway, NJ, USA  
e-mail: [huei-ling\\_chen@merck.com](mailto:huei-ling_chen@merck.com)

Heng Zhou, Ph.D.  
Merck & Co., Inc., Rahway, NJ, USA  
e-mail: [heng.zhou@merck.com](mailto:heng.zhou@merck.com)

Nan Xiao, Ph.D.  
Merck & Co., Inc., Rahway, NJ, USA  
e-mail: [nan.xiao1@merck.com](mailto:nan.xiao1@merck.com)

## TRADEMARK

SAS and all other SAS Institute Inc. products or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.