PharmaSUG 2025 - Paper SD-054

Automated Word Report Generation for Standardized CMC PC Study with Programmatically Inserted Contents in R

Song Liu and Jiannan Kang, Merck & Co., Inc., Upper Gwynedd, PA, USA

ABSTRACT

Biologics process characterization (BPC) study reports have prescriptive formats and contents. An internally developed BPC-Stat tool (JMP add-in) can produce Microsoft Word reports containing statistical analysis results. But this output does not have the reference linking and required outline structure. Manual efforts and additional contents such as description of the model effects are still required from the project scientists.

This paper demonstrates a new tool to use R/R Markdown/R shiny to automatically generate the Word report file with standard format, consistent language, auto-populated interpretation texts, and auto-generated TOC, TOT, TOF, header/footer, content-driven cross-references, and prompts. With this tool, PC Study Report is automatically generated with a few user inputs of document number, document title, etc. and one click of the download button from the R shiny app which shared by the developer. This tool is a great way to reduce the likelihood of human errors that often arise while performing numerous repetitive tasks and divert resources from boring copy/paste type of exercises to challenging problem solving.

INTRODUCTION

The Research Chemistry, Manufacturing, and Control Statistics (RCMCS) group at Merck needs to generate tens to hundreds of Process Characterization (PC) study report documents every year on BPC Stat study results.

As part of a separate automation project, PC analyses within Merck have been automated via a JMP tool called BPC-Stat. The results of the analyses are saved to several output files that can be utilized as inputs to the automated reporting tool.

Even with the help of BPC-Stat, this process involved heavy manpower to write PC study reports with scientific interpretation and formatting into a consistent and standardized format. This tedious work includes a lot of manual copying and pasting, as well as complex descriptions of results, both of which are time-consuming and error-prone.

Automating such routines can help scientists generate the reports in the predefined format programmatically. It is widely accepted that report automation improves work efficiency and report accuracy.

BACKGROUND

The PC Study reports are generated in Microsoft Word format, composed of several key components that define their underlying structure and content organization. Understanding these components is essential for effectively working with Word documents programmatically.

The main parts of a Word document include the body, headers, and footers. Headers are located at the top of each page and can contain information such as document titles and logos. They may differ for the first page of a section. Footers, situated at the bottom of each page, can contain information like page numbers and may also feature different content on the first page of a section.

The body serves as the main content area, containing paragraphs, tables, figures, and other elements that make up the textual and visual content. Additional important features of Word include calculated fields and styles. Calculated fields are dynamic elements that automatically update data, such as dates, page numbers, and table or figure references. Tables of contents are typically generated using these calculated fields, which reference titles and page numbers.

Styles are formatting features that apply to text, paragraphs, headings, and other elements in a Word document. They define the appearance of various components and provide a quick and efficient way to maintain consistent formatting throughout the document.

The R language is widely used among statisticians for developing data analysis software. R Markdown is a tool that aids in creating reproducible, dynamic reports with R. An R Markdown document is written in Markdown—a user-friendly plain text format—and contains chunks of embedded R code. R Markdown relies primarily on the knitr and Pandoc packages: knitr executes embedded code and converts R Markdown to Markdown, while Pandoc renders Markdown into the desired output format (such as PDF, HTML, or Word).

Two primary packages are designed to automate report generation in Word: Officer and Officedown. Officer enables the creation of Word reports from within R; however, it is not intuitive for reading and writing and has limitations in aspects like cross-references and support for Flextable output. Officedown, on the other hand, incorporates most of Officer's features into R Markdown documents and is much more user-friendly. This project primarily utilizes Officedown while incorporating Officer for specific scenarios.

PROGRAM FLOW

The high-level design of the report automation tool is illustrated below.

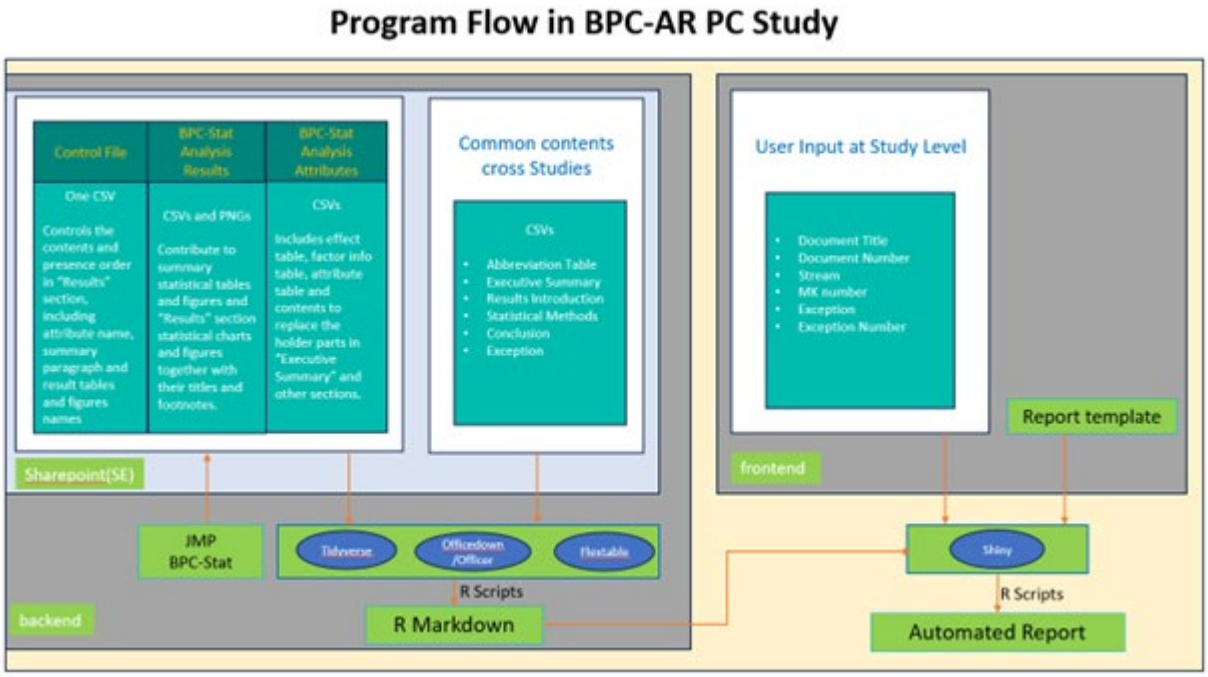

Figure 1. Program Flow in BPC-AR PC Study

This R Shiny app is built around a robust backend R Markdown file, which is the core component of the tool. It accepts both common files shared across studies and unique files specific to individual PC studies as input. All these files are stored on SharePoint and can be accessed and downloaded through the R Shiny app hosted on Merck's server via the HTTP protocol. By integrating user inputs on the frontend with the Word report template, the app generates a downloadable report document.

The flowchart detailing the backend R Markdown code is presented below, describing the entire document generation process in greater detail. Further explanations will be provided in the following sections.

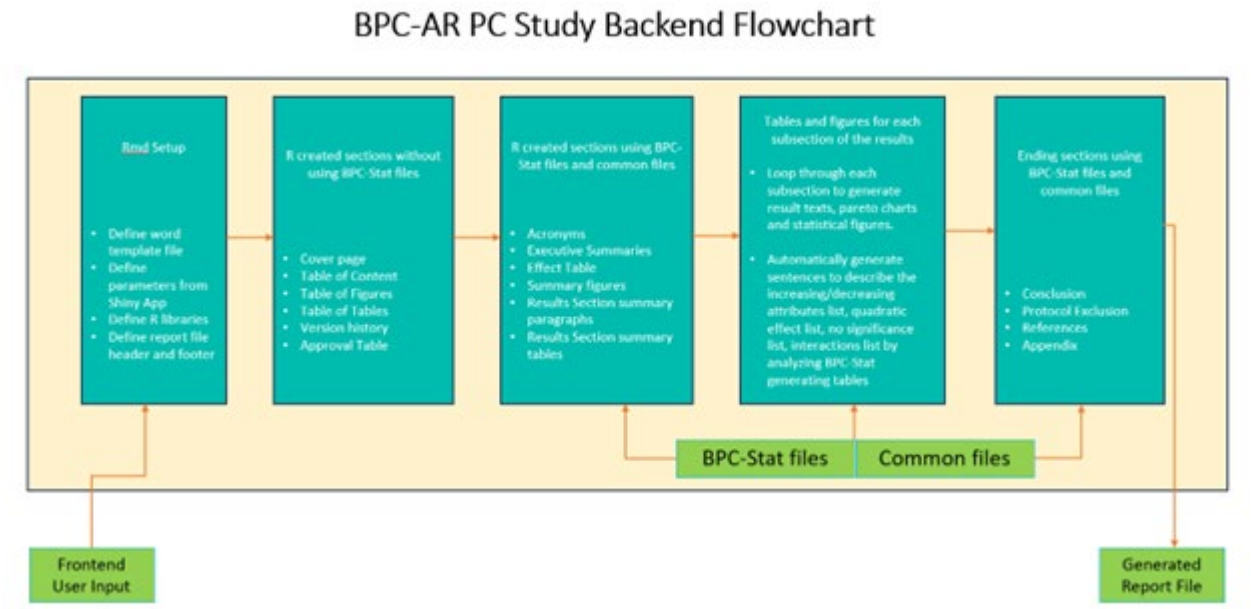

Figure 2. BPC-AR PC Study Backend Flowchart

PROGRAM WALKTHROUGH

INTRODUCTION OF THE INPUT FILES

All input files are either .csv or .png format and fall into two categories. The first group consists of predefined .csv files shared across studies containing standard content, such as abbreviation tables, executive summaries, and conclusions. The second group comprises individual study results exported from BPC Stat, including tables, figures, result texts, and related control files that describe analysis model types and result structures.

STRUCTURE OF THE PC REPORT AND THE R MARKDOWN FILE

Like other common reports, the PC report includes a cover page, a “Table of Contents,” a “Table of Figures,” a “Table of Tables,” a “Version History Table,” an “Approval Table,” an “Acronyms Table,” “References,” and an “Appendix.” It also contains various sections specific to PC analysis, such as “Executive Summary,” “Introduction,” “Materials and Methods,” “Results,” “Conclusion,” and “Protocol Exception Table.” The “Results” section is particularly important, as it presents the statistical analysis results and scientific interpretations for each attribute from the effect table.

The structure of the R Markdown file is designed to reflect the structure of the PC report. Each code chunk corresponds to a specific section in the report. The R Markdown file begins with metadata (YAML metadata), written between a pair of three dashes. A simplified example of the metadata section is as follows:

```
---
output:
  officedown::rdocx_document:
     reference_docx: "template.docx"
params:
  docName: ""
  docNumber: ""
---
```

Program 1. R Markdown metadata section

In the metadata definition, there are two parts to define the output: “rdocx_document,” which indicates that the R Markdown file will create a Word document, and “reference_docx,” which points to “template.docx” as the reference template. This reference document is used to customize formatting characteristics in the generated .docx file.

Additionally, metadata parameters are defined using the “params” field within the YAML metadata section. Here, we can specify one or more parameters, which can be overridden by a custom list from the R Shiny interface.

The body of the R Markdown document follows the metadata. The first body chunk is the setup section, specifying the R Markdown knit options, required R packages, and any user-defined functions. The “knitr” package provides numerous chunk options to customize nearly all components of code chunks, such as the source code, text output, and plots. For example, the “echo” option (TRUE/FALSE) determines whether to display the source code in the output document. An example of the setup section is as follows:

````
```{r setup, include=FALSE}

knitr::opts_chunk$set(echo = FALSE,warning = FALSE, message = FALSE,comment = NA)

library(officedown)

library(officer)

library(flextable)

library(knitr)

library(tidyverse)
```
````

Program 2. R Markdown setup section

Section headers are marked with one or more pound signs; for example, one pound sign denotes a first-level header, while two pound signs indicate a second-level header.

Most tasks in this project are accomplished across different code chunks. Typically, each chunk is crafted to generate a section of the report. Each chunk can have its own options. For instance, the chunk option “eval” controls whether to evaluate a code chunk, allowing different chunks to be conditionally evaluated based on various conditions. In the PC study report, different statistical models necessitate distinct executive summaries and analysis results. By utilizing the “eval” option, the program can select the appropriate chunks based on the model types. This is illustrated as follows:

````
\newpage

# ACRONYMS

```{r, abbreviation}

source("abbr.R")

```

\newpage

# EXECUTIVE SUMMARY

```{r, summary1, results = 'asis', eval=summary1Flag}

source("summary1.R")

```

```{r, summary2, results = 'asis', eval=summary2Flag}

source("summary2.R")
````
````

```
```

Results

```{r, results1, results = 'asis', eval=results1Flag}

source("results1.R")

```

```{r, results2, results = 'asis', eval=results2Flag}

source("results2.R")

```

Conclusion

```{r, conclusion}

source("conclusion.R")

```
```

**Program 3. R Markdown section headers and code chunks illustration**

## GENERATING TEXTS USING R OFFICEDOWN

In Officedown, the primary functions for generating texts include ftext(), fp_text(), fpar(), fp_par(), and knit_print_block(). The ftext() and fpar() functions are used to create multiple sentences and paragraphs, respectively, while fp_text() and fp_par() facilitate text and paragraph formatting properties.

If a loop is employed to generate multiple paragraphs, two actions are necessary: first, set results='asis' as the chunk option; second, within the loop, utilize the knit_print_block() function to wrap the fpar() codes.

For example, a section of paragraphs stored in the file “appendix_test.csv” looks like this:

| | A |
|---|---|
| 1 | Appendix paragraph 1 |
| 2 | Appendix paragraph 2 |
| 3 | Appendix paragraph 3 |

**Figure 3. appendix_test.csv**

To write these paragraphs, the R Markdown chunk might look like this:

```
```{r, appendix, results = 'asis'}

appendix_file <- "appendix_test.csv"

appendixTexts <- read_csv(appendix_file,col_names = FALSE)

propAp <- fp_text(color = "BLACK",

                        font.family = "Times New Roman",font.size = 12)

for(i in 1:3){

  knit_print_block(

    fpar(

      ftext(unlist(appendixTexts[i,1]), propAp),

      fp_p = fp_par(
```

```
        text.align = "justify",
        line_spacing = 1.5)
    ))
}
```

Program 4. R Markdown codes to generate texts

The output Word document will display the generated text and paragraph properties as specified in the code chunk as shown below.

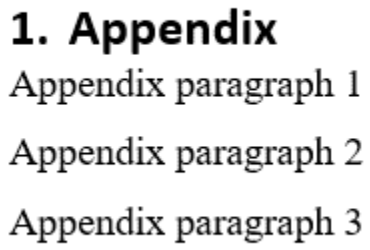

Figure 4. text paragraph output

GENERATING TABLES USING R FLEXTABLE

The Flextable package simplifies table creation for reports and publications, allowing integration within R Markdown Word documents. Users can create tables, modify their content, and format them easily. Flextable also supports merging cells, adding header and footer rows, changing formats, and specifying how data appears in cells. To ensure consistent formatting, it is recommended to define formatting properties in advance, such as font type, border color, and the number of decimal places to display using the set_flextable_defaults() function.

Many functions are available to modify table layouts, add or change headers/footers, adjust cell dimensions, and merge cells.

For example, consider a .csv table file shown below, where the first three columns represent the table data, the fourth column is the title, and the fifth column provides a footnote.

	A	B	C	D	E
1	Column 1	Column 2	Column 3	Title	Footnote
2	Attribute 1	type 1	999.9	table title	table footnote
3		type2	999.9		
4		type 3	999.9		
5	Attribute 2	type 1	999.9		
6		type 2	999.9		
7		type 3	999.9		
8	Attribute 3	type 1	999.9		
9		type 2	999.9		
10		type 3	999.9		

Figure 5. An example table file

If we also want to merge three consecutive rows in the first column into one cell, the following R chunk illustrates how to implement this:

```
```{r, tableTest, results = 'asis', eval=FALSE}
tableTest <- read_csv("table_test.csv",col_names = TRUE,
                      col_types = cols(.default = col_character()))


table_cap <- unlist(tableTest[1,ncol(tableTest)-1])
table_foot <- unlist(tableTest[1,ncol(tableTest)])
table_data <- tableTest[ ,seq(1,ncol(tableTest)-2)]
set_flextable_defaults(
  font.size = 12,
  bold = FALSE,
  line_spacing = 1.0,
  font.family = "Times New Roman",
  theme_fun = "theme_vanilla")
nn <- floor(nrow(table_data)/3)
merge_custom <- function(ft){
      for(rr in 1:nn){
        ft <- merge_at(ft, i = seq(3*rr-2,3*rr), j = 1)
        ft <- valign(ft, i = 3*rr-2, valign = "top")
      }
  ft
}
cat("\n  <br>  \n")
tab_num <- run_autonum(seq_id = "Table", pre_label = "Table ",
                       bkm = "Tabletest")
knit_print_block(block_caption(table_cap,
                               style = "caption", autonum = tab_num))
table_data %>%
  flextable() %>%
    set_table_properties(width = 1, layout = "autofit") %>%
    align(j=1:3,align=c("left","left","right"),part="all") %>%
    merge_custom %>%
    add_footer_lines(table_foot)%>%
    fontsize(part = "footer", size = 10) %>%
```
```

```
    knit_print() %>%
    cat()
cat('\n')
```

Program 5. R Markdown codes to generate flextable

Using the read_csv() function, we can define the col_types argument as character to ensure numeric values are correctly interpreted. A user-defined function, merge_custom, will merge three consecutive rows into a single cell in the first column. The functions set_flextable_defaults(), set_table_properties(), and align() are employed to define the table's formatting, while add_footer_lines() adds a footer to the table. Cross-reference functions establish a basis for the table title, which will be discussed in the next section.

After knitting, the output table in the Word file will appear as follows:

Table 1: table title

Column 1	Column 2	Column 3
Attribute 1	type 1	999.9
	type2	999.9
	type 3	999.9
Attribute 2	type 1	999.9
	type 2	999.9
	type 3	999.9
Attribute 3	type 1	999.9
	type 2	999.9
	type 3	999.9

table footnote

Figure 6. Flextable output

LOOPING THROUGH RESULT SECTIONS TO GENERATE TEXTS, TABLES, AND FIGURES

This component of the report is crucial for programmatically generating texts, tables, and figures with their references in the Word report. Here, we conduct repeated analyses on multiple attributes without prior knowledge of how many sections, tables, or figures the report will contain. The solution is to create R Markdown code using a for loop, setting the chunk option results="asis".

Key steps to create cross-references for each section, table, and figure include:

- Creating a bookmark using the run_autonum() function
- Adding a caption using the block_caption() function
- Wrapping the block_caption() function into knit_print_block() function
- Managing citations in the text using the run_reference() function

To programmatically create subsections within the analysis section, use the chunk option results="asis". This option instructs “knitr” not to wrap the text output in verbatim code blocks (as it does with normal chunks) but treat it as raw Markdown content. In these chunks:

- Headings can be created using cat("\n## HeadingName \n"), where the \n characters are essential for proper formatting.
- Figures are outputted with cat().
- Flextables must be outputted using both the knit_print() and cat() functions.
- Captions (block_caption()) and references (run_reference()) must be wrapped in the knit_print_block() function.

For example, if there are two figures to present, their data is stored in g001.csv and g002.csv, respectively, as shown below.

	A	B	C
1	Title	Footnote	image_path
2	figure 1 title	figure 1 footnote	figure_test1.png
3			

Figure 7. g001.csv

	A	B	C
1	Title	Footnote	image_path
2	figure 2 title	figure 2 footnote	figure_test2.png
3			

Figure 8. g002.csv

Using the figures_test.csv file as a control file shown below, the looped figure generation and citation code chunk might look like this:

	A
1	figures
2	g001.csv
3	g002.csv

Figure 9. control csv file

```
```{r, Figures, results = 'asis'}

Fig_file <- "figures_test.csv"

Fig_tab <- read_csv(Fig_file)

for(i in 1:2){

  current_fig_name <- unlist(Fig_tab[i,1])

  current_Fig <- read_csv(current_fig_name,col_names = TRUE)

  current_Fig_cap <- unlist(current_Fig[1,1])

  current_Fig_foot <- unlist(current_Fig[1,2])

  current_Fig_data <- unlist(current_Fig[1,3])
```
```

```
  cat("\n  <br>  \n")

  Fig_num  <-  run_autonum(seq_id  =  "Figure",  pre_label  =  "Figure  ",  bkm  =
paste0("FigData",as.character(i)))

  knit_print_block(block_caption(current_Fig_cap, style = "caption", autonum = Fig_num))

  image_file_Fig <- paste('\n![](', current_Fig_data,')\n',sep="")

  cat(image_file_Fig)

  cat("\n")

  cat(current_Fig_foot)

  cat("\n  <br>  \n")

}

cat("\n  <br>  \n")

cat("\n  <br>  \n")

cat("\n  <br>  \n")

cat("\n  <br>  \n")

block_list(

  fpar(

       ftext("This is the reference to figure "),

       run_reference(id = sprintf("FigData%s", 1)),

       ftext(". ")

  ),

  fpar(

       ftext("This is the reference to figure "),

       run_reference(id = sprintf("FigData%s", 2)),

       ftext(". ")

      )

)
```

Program 6. R Markdown codes to generate figures in a loop

After knitting, the output in the Word file will show the cross-referenced figure. Hovering over “1” after “This is the reference to figure” displays the cross-reference name “FigData1.” By pressing “Ctrl + Click,” the cursor will navigate to Figure 1.

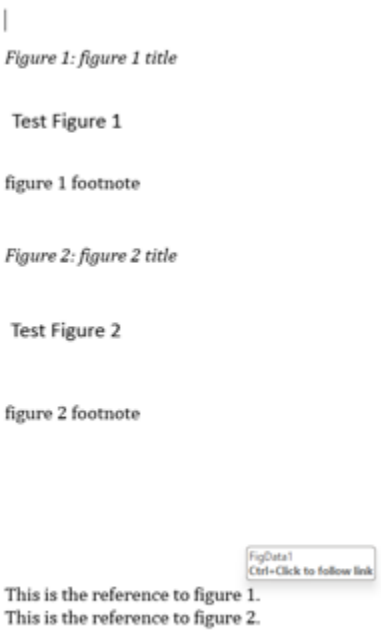

Figure 10. Output of figures generated in a loop and cross-reference illustration

AUTOMATICALLY GENERATING WORD DOCUMENT HEADER/FOOTER, TOC, TOT, AND TOF

In Officedown, the header and footer are set using the block_section() function. The arguments within this function are defined by the prop_section() function, which specifies header and footer templates, page orientation (landscape or portrait), and section types (continuous or odd pages). Header and footer templates for the first page can differ from those on subsequent pages and may be empty for the first page.

An example code chunk is presented below:

```
```{r, include=FALSE}
ft_foot <- fp_text_lite(color = "BLACK", bold = FALSE,
                        font.family = "Times New Roman",font.size = 9)
prop_line1 <- fp_text_lite(color = "BLACK", bold = TRUE,
                           font.family = "Times New Roman",font.size = 12)
prop_line2 <- fp_text_lite(color = "BLACK", bold = FALSE,
                           font.family = "Times New Roman",font.size = 10)
prop_footer <- fp_text_lite(color = "BLACK", bold = FALSE,
                           font.family = "Times New Roman",font.size = 12)
header_template <-  block_list(
    fpar(ftext("header_line1",prop_line1),fp_p = fp_par(
                     line_spacing = 1.3,
                     text.align = "center")),
    fpar(ftext("header_line2",prop_line2),fp_p = fp_par(
```
```

```
                      line_spacing = 1.3,
                      text.align = "left"))
  )
footer_template <-  block_list(
    fpar(ftext("Page ", prop_footer),
        run_word_field(field = "PAGE  \\* MERGEFORMAT"),
        ftext(" of ", prop_footer),
        run_word_field(field = "NUMPAGES  \\* MERGEFORMAT"),
        fp_p = fp_par(text.align = "left")
        )
  )
header_first_template <- block_list(fpar(ftext("")))
footer_first_template <- block_list(fpar(ftext("")))
sec1_pr <- prop_section(
  header_default = header_template,
  footer_default = footer_template,
  header_first = header_first_template,
  footer_first = footer_first_template,
  page_size = page_size(orient = "portrait"),
  type = "continuous"
)
```

Program 7. R Markdown codes to generate header/footer

The run_word_field() function is part of the Officer package and allows for the insertion of calculated fields (such as the current page number and the total number of pages) into Word documents. When executed, this code chunk will omit the header and footer on the first page. Starting from the second page, the header will display “header_line1” and “header_line2,” while the footer will follow the format “Page x of xxx.”

The Table of Contents (TOC) is a computed field in Word. The TOC field collects entries using heading styles or other specified styles.

In Officedown, the block_toc() function generates the Table of Contents. We can use a single style for both table and figure captions, necessitating the creation of separate tables of contents for tables and figures. In this scenario, the block_toc() function sets the seq_id argument to “Table” for table captions and “Figure” for figure captions.

The respective chunks for TOC, TOT, and TOF are illustrated below:

```
```{r}
block_toc()
```
```

Program 8. R Markdown codes to generate TOC

```
```{r}
block_toc(seq_id = "Figure")
```
```

Program 9. R Markdown codes to generate TOF.

```
```{r}
block_toc(seq_id = "Table")
```
```

Program 10. R Markdown codes to generate TOT

MIXING LANDSCAPE AND PORTRAIT PAGES AND MANAGING THEIR HEADER/FOOTER

In the PC study report, page orientation may vary between portrait and landscape. To mix these orientations, we need to define multiple section properties using the prop_section() function.

The arguments for prop_section() include "header default," "footer default," "header_first," "footer_first," "page_size," and "type." Header and footer templates for all but the first page in a specific section are assigned to “header default” and “footer default.” Conversely, header and footer templates for the first page of a specific section are assigned to "header_first" and "footer_first." The orientation property is set through the “orient” argument within the page_size() function.

For instance, if the report consists of three sections, where the first section is portrait, the second is landscape, and the third is again portrait, we must define three separate section properties, as shown below:

```
```{r}
sec1_pr <- prop_section(
  header_default = header_template,
  footer_default = footer_template,
  header_first = header_first_template,
  footer_first = footer_first_template,
  page_size = page_size(orient = "portrait"),
  type = "continuous"
)


sec2_pr <- prop_section(
  header_default = header_template,
  footer_default = footer_template,
  page_size = page_size(orient = "landscape"),
  type = "continuous"
)
```
```

```
sec3_pr <- prop_section(
  header_default = header_template,
  footer_default = footer_template,
  header_first = header_template,
  footer_first = footer_template,
  page_size = page_size(orient = "portrait"),
  type = "continuous"
)
```

Program 11. R Markdown codes to mix landscape and portrait pages

After these declarations, the functions block_section(sec1_pr), block_section(sec2_pr), and block_section(sec3_pr) must be placed at the end of each section to ensure the output document adheres to the required format.

WRAPPING THE R MARKDOWN FILE INTO THE R SHINY APP

All R Shiny apps begin with an app.R file, which determines both the appearance and behavior of the app. A simple example of an app.R file is as follows:.

```
library("shiny")
# UI
ui <- function() {
  source("ui-main.R")
}
# Server
server <- function(input, output, session) {
  source("server-main.R")
}

shinyApp(ui, server)
```

Program 12. R Shiny App illustration

The ui-main.R file defines the user interface, while the server-main.R file specifies the app's behavior.

For this project, the user interface, which is shown below, requires users to input data and click the “Generate Report” button. Following this action, the output document will be downloaded to the user's local computer.

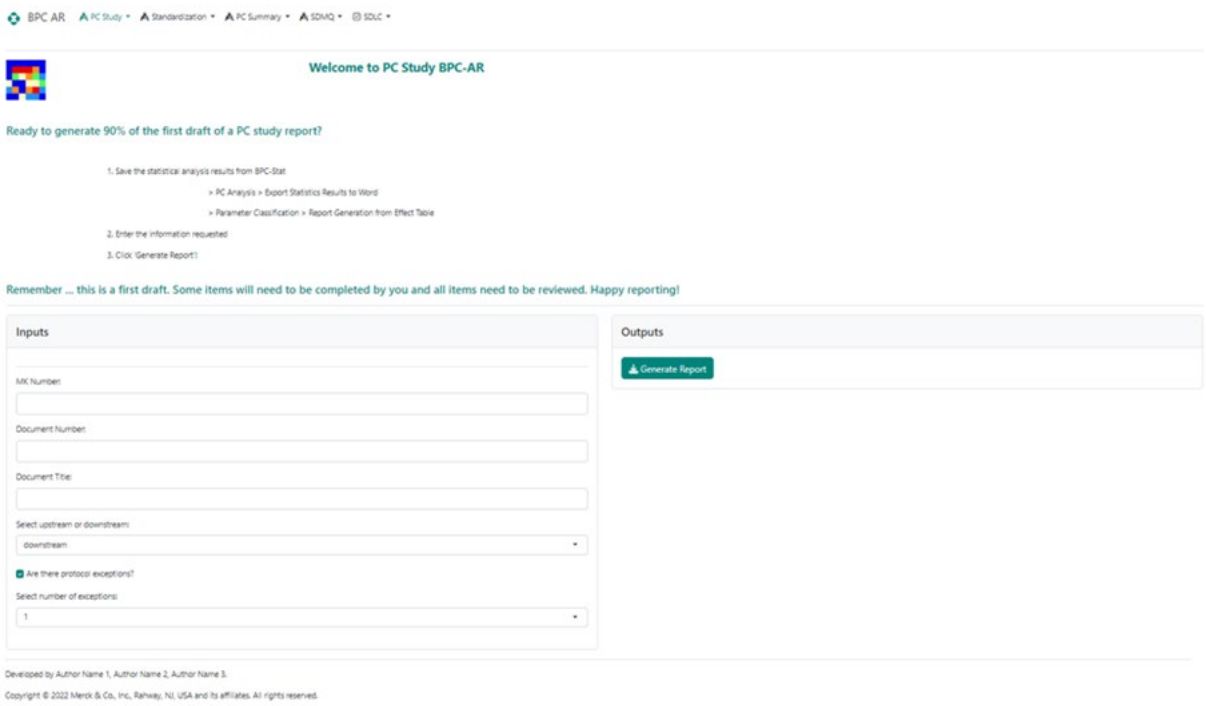

Figure 11. R Shiny interface

The code in the server-main.R file that handles the download button is shown below.

```
output$report <- downloadHandler(
  filename = function() {
    paste0(input$name_Doc, ".docx")
  },
  content = function(file) {
    rmarkdown::render(
      input = "BPC_AR_study_report.Rmd",
      output_file = file,
      params = list(
        docName = input$name_Doc,
        docNumber = as.character(input$num_Doc)
      ),
      envir = new.env(parent = globalenv())
    )
  }
```

Program 13. R Shiny code to handle downloading

It utilizes the rmarkdown::render() function to knit the R Markdown file "BPC_AR_study_report.Rmd." The parameter list defined in this function is passed to the .Rmd file, overwriting any values previously defined therein.

CONCLUSION

The report automation tool introduced in this document significantly enhances the ability of RCMCS scientists and statisticians to generate study reports in a predefined format, which, in turn, improves both work efficiency and report accuracy.

However, the current version is still in draft form and requires further refinement to ensure greater robustness and broader applicability. Moving forward, we will explore the feasibility of incorporating AI to extend the functionality of this tool beyond PC studies.

REFERENCES

Xie, Y., Allaire, J. J., & Grolemund, G. *R Markdown: The Definitive Guide.* Available at https://bookdown.org/yihui/rmarkdown/.

Grolemund, G. *officeverse.* Available at https://ardata-fr.github.io/officeverse/index.html.

Grolemund, G. *Using the flextable R package.* Available at https://ardata-fr.github.io/flextable-book/index.html.

Layton, R. *Happy collaboration with Rmd to docx.* Available at https://rmarkdown.rstudio.com/articles_docx.html.

Wickham, H. *Mastering Shiny.* Available at https://mastering-shiny.org/index.html.

Clark S., Matzke M., Karl A., Lucas R., Gardner S., Asherman G., Rushing H., Christopher, D. 2022. *BPC-Stat: Biologics Process Characterization – Statistics: Accelerating process characterization with agile development of an automation tool set*. Invited presentation, JMP Discovery Summit.

ACKNOWLEDGMENTS

The authors wish to extend their sincere gratitude to the Research Chemistry, Manufacturing & Controls Statistics (RCMCS) group at Merck & Co., Inc., Rahway, NJ, USA, particularly Seth Clark and Melissa Matzke. Without their support, the successful development of this tool would not have been possible.

Additionally, we appreciate the valuable reviews of this paper provided by David Christopher, Seth Clark, and Melissa Matzke.

CONTACT INFORMATION

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

Song Liu
Merck & Co., Inc., Upper Gwynedd, PA, USA
Song.liu3@merck.com

Jiannan Kang
Merck & Co., Inc., Upper Gwynedd, PA, USA
jiannan_kang@merck.com