Paper PP03

Risk Management Plan (RMP) Tables using R

Diana Tai, Bristol Myers Squibb, Berkeley Heights, NJ, USA
Marckenley Mercie, Bristol Myers Squibb, Berkeley Heights, NJ, USA

ABSTRACT

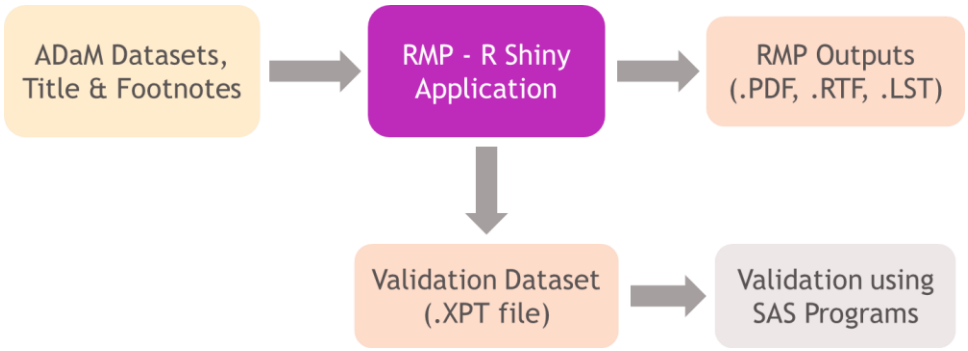
The European Medicines Agency (EMA) requires a Risk Management Plan (RMP) submission when applying for marketing authorization. Companies must provide details of how authorized indications and target populations have developed during the life cycle for the products within this RMP. Based on the EMA Guidance on the format of the RMP, duration and adverse event tables must be provided for each indication. Traditionally, SAS has been used to develop the tables required for the RMP submission. In most cases, every table would have a corresponding production and validation program. Using R Shiny and powerful data manipulation packages found in the tidyverse, we have created a user-friendly web application to generate these tables. This has increased the efficiency of our table development and accelerated our EMA submission timelines.

INTRODUCTION

SAS has been the software traditionally used to develop tables in a Risk Management Plan, as required in submissions to the European Medicines Agency. R is an open-source programming language that has been gaining popularity in the statistical field as an alternative to SAS. The base functions of the language are augmented by many freely available user-developed packages that are constantly being updated. When brainstorming on ways to bring RMP table creation over to R, we kept in mind that many SAS programmers do not have the R knowledge to immediately transfer their workflow to a new language. Thus, we have developed a user-friendly Shiny application that takes advantage of the capabilities of R while providing a streamlined application for users to quickly create the tables needed for RMP submissions.

OVERVIEW OF THE APPLICATION

The goal of creating this R Shiny application was to allow SAS programmers to generate the tables required for an EMA RMP submission without dealing with the potential hassles associated with production programming. The application utilizes a variety of R packages, with Shiny providing the basis of the interactivity. We also looked at packages that have been developed for the purpose of extending Shiny, such as adding new input controls or displaying interactive tables. Below is a flow chart of the process for generating the tables required for an RMP submission using the R Shiny Application.



First, the application takes in the ADaM data sets ADAE, ADSL, and ADEX, as well as a tracking file, as inputs. For this step we make the following assumptions:

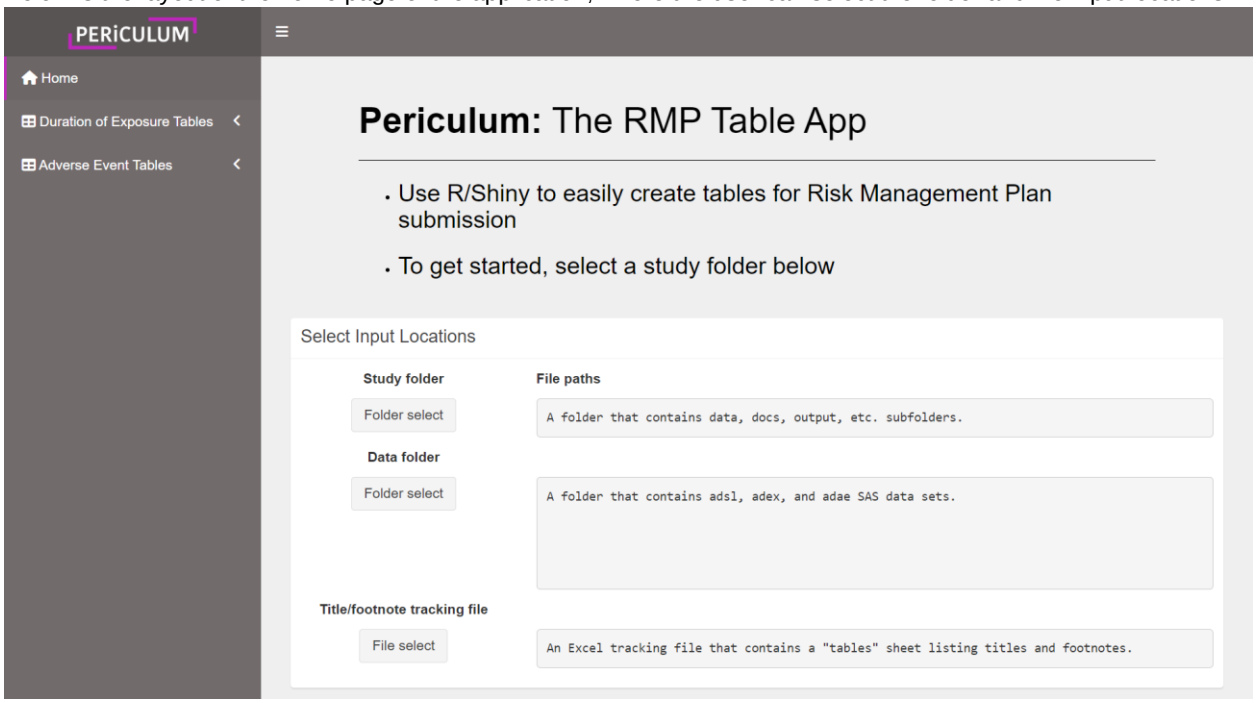
- ADAE, ADSL, and ADEX are CDISC compliant.
- ADEX is in BDS structure with parameters for treatment duration and the number of doses per subject.
- The tracking file is an Excel file containing the titles and footnotes for each table.

After the data has been inputted from the front end of the application, the back end will output tables in PDF, RTF, and LST formats. From there the user can download the necessary outputs for the RMP submission. Alongside generating the table outputs, the application has the option for the user to download an XPT file of the table. This will then be used to validate the table outputs using SAS programs.

READING IN DATA

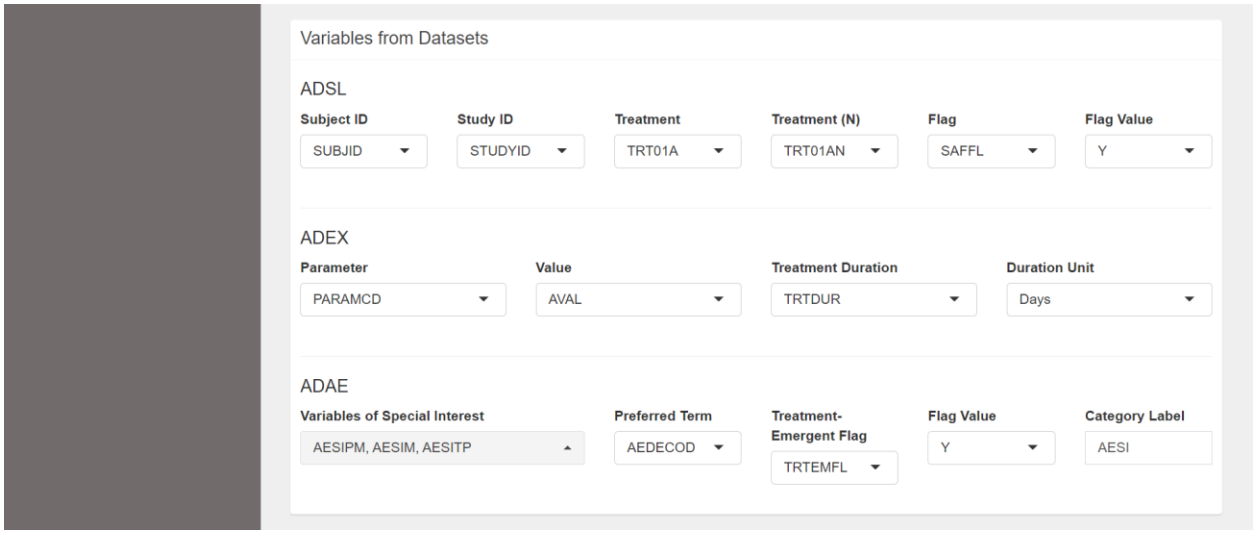
For the RMP tables to be generated, the ADaM data sets and tracking file must be accessible in the R environment. Usually, the ADaM data sets are in *.sas7bdat* format. Fortunately, the *haven* package has facilitated a way to load *.sas7bdat* data in R via the *read_sas()* function. As stated before, the tracking file is an Excel file containing the titles and footnotes for the table outputs. The *read_excel()* function found in the *readxl* package reads in this file into the R environment. In order to access the required folders and files, we took advantage of the Shiny extension package *shinyFiles*. This package allows us to build a UI that lets the user select the folders in which the data is stored. Then on the back end, we created reactive functions that utilize *read_sas()* and *read_excel()* to load the data in the environment. These files give the application the necessary information to generate the tables.

Below is the layout of the home page of the application, where the user can select the folder and file input locations.



To allow for greater flexibility, the variables defined as subject ID, treatment, etc. are not hard coded into the app. They are user-controlled with *selectInput()* functions, where the list of possible values comes directly from the variables in the uploaded data sets. Thus, even if the variables are different from expected, or if the user wishes to construct a table deviating from a standard shell, the app is still able to accommodate for these cases.

Scrolling down on the home page, here is a snapshot of the variables the user is allowed to select, after the data sets have been loaded in.



To use these reactive inputs in *dplyr* functions, we made use of the *.data* pronoun. For example, say the treatment variable comes from a *selectInput()* with the inputID 'treatment'. To use this in a *select()* function, with the data frame defined as 'df', write:

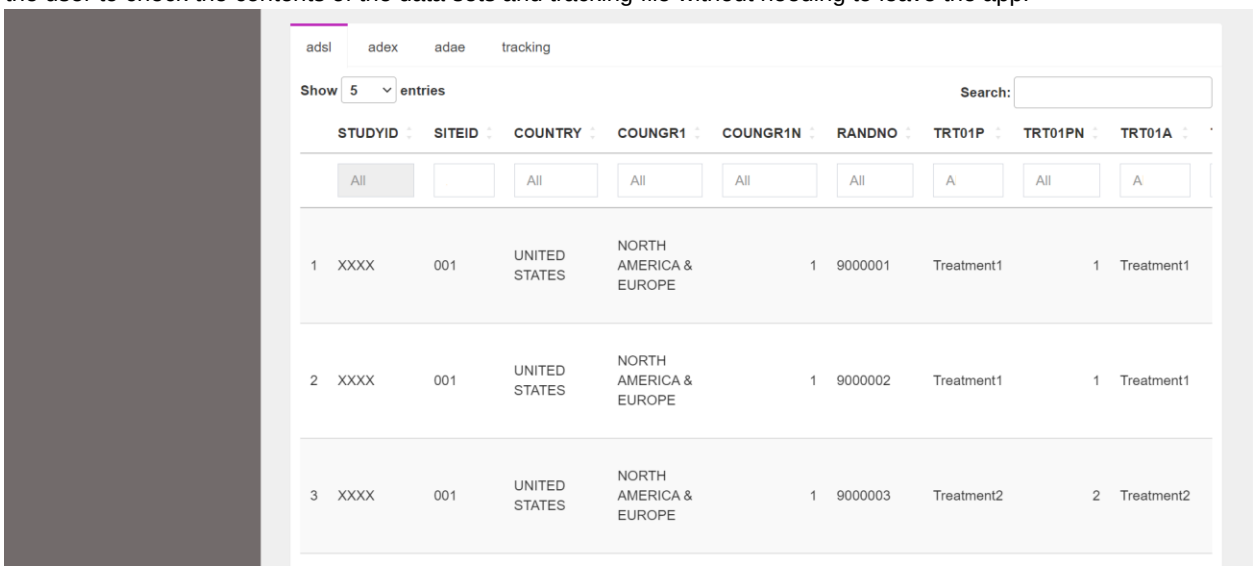
```
select(df, .data[[input$treatment]])
```

If the selection for 'subjid' was 'TRT01A' as in the above image, this would be equivalent to:

```
select(df, TRT01A)
```

For the rest of this paper, we will ignore these reactive elements and will simply use the variable names for ease of reading. Additionally, there are a few other input fields on the home page that let the user define the dates for data extract and database cutoff, as well as the protocol, etc., in order to fill in the headers and footers of the tables.

At the bottom of the home page, previews of the uploaded files are displayed with use of the *DT* package. This allows the user to check the contents of the data sets and tracking file without needing to leave the app.



adsl | adex | adae | tracking

Show 5 entries Search:

	STUDYID	SITEID	COUNTRY	COUNGR1	COUNGR1N	RANDNO	TRT01P	TRT01PN	TRT01A
	All		All	All	All	All	A	All	A
1	XXXX	001	UNITED STATES	NORTH AMERICA & EUROPE	1	9000001	Treatment1	1	Treatment1
2	XXXX	001	UNITED STATES	NORTH AMERICA & EUROPE	1	9000002	Treatment1	1	Treatment1
3	XXXX	001	UNITED STATES	NORTH AMERICA & EUROPE	1	9000003	Treatment2	2	Treatment2

TABLE CREATION

The EMA has provided guidance in which an RMP should be submitted. In this guidance document there are formats for state of exposure reporting. To transform the input data into the structure required by each table, we used several data manipulation packages, especially those found in the *tidyverse* collection, including *dplyr*, *tidyr*, and *purrr*. We will break down the steps taken to create two tables, one exposure table and one adverse event (AE) table.

STATE OF EXPOSURE

In an RMP there are subsets of duration of exposure tables. These include duration of exposure by indication, by dose level, by age group and gender, by ethnic origin, and by race. Below is a standard table shell similar to the one provided by the EMA for the duration of exposure by indication.

Table 0.6
Duration of Exposure
Safety Population

	Trt1 (N=XXX)		Trt2 (N=XXX)	
Duration of Exposure (at least)	Persons	Person Year	Persons	Person Year
1 month	xx	xx.x	xx	xx.x
3 months	xx	xx.x	xx	xx.x
6 months	xx	xx.x	xx	xx.x
12 months etc.	xx	xx.x	xx	xx.x
Total person time	xx	xx.x	xx	xx.x

Before we get into calculating the Persons and Person Years, the data has to be prepared and the appropriate population of subjects must be obtained. This is where we take advantage of several functions available in the *dplyr* package.

First, we *filter()* the ADSL data frame to reflect the safety population. Then we use *inner_join()* to combine the ASDL and ADEX data frames with USUBJID as the 'by' argument. In R, merge and join functions come with a particular quirk. They will duplicate and distinguish common variables shared by both data sets not listed as a 'by' argument with the suffixes '.x' and '.y'. We want to ensure they are joined by subject ID, but do not necessarily want to list every variable the data frames have in common. We can get around this by defining the first value in the suffix argument under *inner_join()* to be an empty character string, while keeping the second value as the default '.y'. After dropping variables with the '.y' suffix, using *select()* and the *tidyselect* function *ends_with()*, no duplicated columns will remain.

```
adsl <- adsl %>%
  filter(SAFFL == "Y")
ex <- adsl %>%
  inner_join(adex, by = "USUBJID", suffix = c("", ".y"))%>%
  select(-ends_with(".y"))
```

The next step is to *filter()* the rows so only the Treatment Duration parameter remains. Then new 'months' and 'years' columns are added using the *mutate()* function with calculations based on the analysis value, which is the treatment duration, measured in days in this case.

```
ex <- ex %>%
  filter(PARAMCD == "TRTDUR")) %>%
  mutate(months = AVAL/30.4375,
         years = AVAL/365.25)
```

Next, we define the different intervals for duration of exposure, e.g., the subjects that fall under at least one month of exposure, three months of exposure, and so on. Vectors of both the numerical breaks, and the corresponding character labels are created, with up to five years of exposure being the limit defined here. The base R function *findInterval()* is used to determine where the maximum number of months from the data falls in the defined duration intervals.

```
dur_breaks <- c(0, 1, 3, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60)
dur_labels <- c("Total person time",
                "1 month", "3 months",
                "6 months", "12 months",
                "18 months", "24 months",
                "30 months", "36 months",
                "42 months", "48 months",
                "54 months", "60 months")
dur_max <- findInterval(max(ex$months), dur_breaks)
```

Now we are ready to calculate Persons and Person Years. The subjects are divided into exposure groups with *cut()*. The function takes in the 'months' column, the vector of duration breaks (up to the maximum), the vector of duration labels, and the 'right' argument is set as false to indicate the intervals are closed on the left and open on the right. Next, only the necessary columns are kept with *select()* and NA values are dropped with *drop_na()* from the *tidyr* package. After grouping by both the treatment and the exposure groups with *group_by()*, we *summarize()* to determine the count of Persons in each group with *n()* and the Persons Years by calculating the *sum()* of the years.

```
ex %>%
  mutate(group = cut(months,
                     c(dur_breaks[1:dur_max], Inf),
                     dur_labels[1:dur_max],
                     right = FALSE)) %>%
  select(TRT01A, group, years) %>%
  drop_na() %>%
  group_by(TRT01A, group) %>%
  summarize(Persons = n(),
            Person_Year = sum(years))
```

The result:

	group	Persons_Treatment1	Person_Year_Treatment1	Persons_Treatment2	Person_Year_Treatment2
1	1 month	429	514.7	109	125.5
2	3 months	421	513.8	105	124.7
3	6 months	407	508.4	102	123.6
4	12 months	335	442.1	79	102.7
5	18 months	60	96.7	12	19.0
6	Total person time	433	515.0	109	125.5

ADVERSE EVENTS

When submitting an RMP to the EMA, adverse events must be reported. The EMA is primarily interested in seeing data on adverse events of special interest. Below is an example of a table that reports these AEs of special interest.

Table 1
Treatment-emergent Adverse Events of Special Interest (AESI) by Treatment, AESI Category and Preferred Term
Safety Population

AESI Category [a] Preferred Term [b]	Trt1 (N=XXX) n (%)	Trt2 (N=XXX) n (%)
Subjects With at Least One AESI	xx (xx.x)	xx (xx.x)
AESI Category Name #1	xx (xx.x)	xx (xx.x)
Preferred Term #1	xx (xx.x)	xx (xx.x)
Preferred Term #2	xx (xx.x)	xx (xx.x)
AESI Category Name #2	xx (xx.x)	xx (xx.x)
Preferred Term #1	xx (xx.x)	xx (xx.x)
Preferred Term #2	xx (xx.x)	xx (xx.x)

The initial process for preparing the data sets is similar to the one for the duration of exposure table previously discussed, with a few added steps. The first difference is that we need the population counts for each treatment group to calculate the percentages for n (%). To do so, we use the *group_by()* function to group the data by the treatment. Next, the *mutate()* function will create a new variable 'N' which has the population counts for each treatment group. To get the subjects in the safety population with AEs, we use *inner_join()* to combine ASDL with the ADAE data frame this time. Then, we use *filter()* to reflect the treatment-emergent AEs.

```
adsl <- adsl %>%
  filter(SAFFL == "Y") %>%
  group_by(TRT01A) %>%
  mutate(N = n())
ae <- adsl %>%
  inner_join(adae, by = "USUBJID", suffix = c("", ".y"))%>%
  select(-ends_with(".y")) %>%
  filter(TRTEMFL == "Y")
```

Now that we have our data, we can start calculating the contents of the table. The first line of the table displays how many subjects had at least one AE of special interest. Depending on the study, AEs of special interest can be represented in many ways. Let's assume for now that each AESI has its own variable. This could be a bit difficult to get counts for if there is more than one AESI category. Therefore, we need to process the data more. In this example ADAE data set we have three AESI categories which are:

- AESITP: Thrombophlebitis
- AESIM: Malignancies
- AESIPM: Premalignant Disorders

We wish to combine these categories into one variable to make counting simpler. To start, we declare a vector which contains the names of the AESI categories. In the app, of course, this would be a user-controlled selection. Then we pass our data set from before into *tidyr*'s *pivot_longer()* function. This will create a data frame with the AESI categories as variables in one column and their values in another. From there we *select()* the columns needed and *filter()* out the blank rows.

```
aesicat <- c("AESITP", "AESIM", "AESIPM")
aesi <- ae %>%
  pivot_longer(cols = all_of(aesicat),
               names_to = "AESI_Categories",
               values_to = "AESICAT") %>%
  select(SUBJID, AESICAT, AEDECOD, TRT01A, TRT01AN, N) %>%
  filter(AESICAT != "")
```

The result:

	SUBJID	AESICAT	AEDECOD Dictionary-Derived Term	TRT01A Actual Treatment for Period 01	TRT01AN Actual Treatment for Period 01 (N)	N
1	0011001	Premalignant Disorders	Abdominal pain	Treatment1	1	433
2	0011001	Malignancies	Nausea	Treatment1	1	433
3	0011001	Premalignant Disorders	Bone pain	Treatment1	1	433
4	0011002	Premalignant Disorders	Abdominal pain	Treatment1	1	433
5	0011002	Premalignant Disorders	Abdominal pain	Treatment1	1	433
6	0011002	Premalignant Disorders	Abdominal pain	Treatment1	1	433
7	0011002	Malignancies	Back pain	Treatment1	1	433
8	0011002	Malignancies	Back pain	Treatment1	1	433
9	0011002	Premalignant Disorders	Bone pain	Treatment1	1	433
10	0011002	Premalignant Disorders	Dizziness	Treatment1	1	433
11	0011003	Malignancies	Back pain	Treatment2	2	109
12	0011004	Premalignant Disorders	Bone pain	Treatment1	1	433

To calculate the first line of the table, we pass the data set created in the previous block to the *group_by()* function. Then, after grouping by treatment, we use *summarize()* to apply the *n_distinct()* function on the subject ID, which calculates distinct counts for each group. The percentage is also calculated using both 'N' and the newly created 'n'. From there, we use *pivot_longer()* to give a record for every 'n' and 'pct'. The function *unite()* creates a variable that is a concatenation of two or more existing variables. In this case, we concatenate the treatment group with 'var' created in the previous function call, that has 'n' and 'pct' as values. Finally, we use *pivot_wider()*, which outputs a wide data set with each treatment group count and percentage as separate columns.

```
atleastone <- aesi %>%
  group_by(TRT01A) %>%
  summarize(AESICAT = "Subjects with at Least one AESI",
            n = n_distinct(SUBJID),
            pct = n / unique(N) * 100) %>%
  pivot_longer(cols = c(n, pct),
               names_to = "var",
               values_to = "value") %>%
  unite(temp, TRT01A, var) %>%
  pivot_wider(names_from = temp,
              values_from = value,
              values_fill = 0)
```

The result:

	AESICAT	Treatment1_n	Treatment1_pct	Treatment2_n	Treatment2_pct
1	Subjects with at Least one AESI	258	59.5843	55	50.45872

Calculating the 'n' and 'pct' at the AESI and Preferred Term levels are similar in nature. The difference will be in the *group_by()* statement, which takes the corresponding additional grouping variables.

```
aesicounts <- aesi %>%
  group_by(TRT01A, AESICAT) %>%
  summarize(n = n_distinct(SUBJID),
            pct = n / unique(N) * 100) %>%
  pivot_longer(cols = c(n, pct),
               names_to = "var",
               values_to = "value") %>%
  unite(temp, TRT01A, var) %>%
  pivot_wider(names_from = temp,
              values_from = value,
              values_fill = 0)
```

The code above produces this result:

	AESICAT	Treatment1_n	Treatment1_pct	Treatment2_n	Treatment2_pct
1	Malignancies	146	33.71824	34	31.19266
2	Premalignant Disorders	166	38.33718	26	23.85321
3	Thrombophlebitis	88	20.32333	30	27.52294

The resulting data frames are then combined in the required order, and additional formatting is applied to create the final AE table.

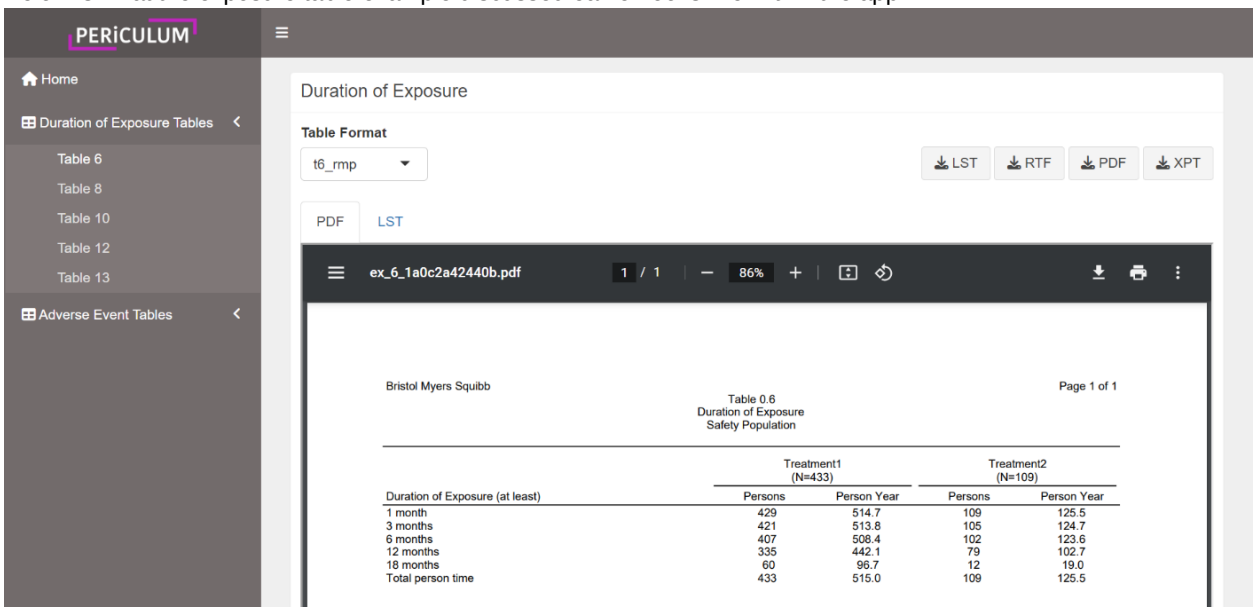
	SpecialInterest	Treatment1_n	Treatment1_pct	Treatment2_n	Treatment2_pct
1	Subjects With at Least One AESI	258	(59.6)	55	(50.5)
2	Malignancies	146	(33.7)	34	(31.2)
3	Back pain	120	(27.7)	32	(29.4)
4	Human chorionic gonadotropin increased	2	(0.5)	0	(0.0)
5	Nausea	40	(9.2)	6	(5.5)
6	Premalignant Disorders	166	(38.3)	26	(23.9)
7	Abdominal pain	36	(8.3)	7	(6.4)
8	Bone pain	85	(19.6)	9	(8.3)
9	Chest discomfort	6	(1.4)	0	(0.0)
10	Cough	64	(14.8)	12	(11.0)
11	Dizziness	49	(11.3)	5	(4.6)
12	Thrombophlebitis	88	(20.3)	30	(27.5)
13	Deep vein thrombosis	6	(1.4)	0	(0.0)
14	Phlebitis	0	(0.0)	1	(0.9)

DOWNLOAD AND VALIDATION

The *reporter* package was used to create the final RMP tables outputs. The *create_report()* function starts off the report. The table is formed with *create_table()*, and then it, as well as any headers, titles, and footnotes necessary are added with the corresponding functions found in the package.

The sidebar in the app allows the user to navigate to the desired table. On each page, there is an additional *selectInput()* for table format, which selects from the corresponding titles and footnotes from the Excel tracking file. A preview of the table is displayed, and the user can download it in LST, RTF, and PDF formats with the corresponding buttons located at the upper right of the page.

Below is what the exposure table example discussed earlier looks like within the app.



In addition to creating tables in the three display formats, the application can also output the table data in SAS transport format (XPORT) using *write_xpt()* from the *haven* package. There is a corresponding XPT download button located on each page in the app. This allows for the transfer of data to SAS to validate the tables created.

CONCLUSION

The application streamlines the process of creating RMP tables for EMA submission. R has the capability to create all the required tables that have traditionally been programmed with SAS. With the numerous open-source packages available for download, one can perform all the data manipulation and output formatting necessary. Finally, with the benefits of Shiny, users of the app do not require prior knowledge of R, and can easily regenerate tables if updates to the data sets are made.

REFERENCES

Wickham H, François R, Henry L, Müller K (2022). *dplyr: A Grammar of Data Manipulation*. https://dplyr.tidyverse.org, https://github.com/tidyverse/dplyr.

Wickham H, Girlich M (2022). *tidyr: Tidy Messy Data*. https://tidyr.tidyverse.org, https://github.com/tidyverse/tidyr.

Wickham H, Miller E, Smith D (2022). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. https://haven.tidyverse.org, https://github.com/tidyverse/haven, https://github.com/WizardMac/ReadStat.

Bosak D (2022). *reporter: Creates Statistical Reports*. R package version 1.3.8, https://reporter.r-sassy.org.

ACKNOWLEDGMENTS

We would like to thank our Bristol Myers Squibb colleagues Mahesh Swaminathan and Baole Fan for their collaborative work on the application, as well as Derek Morgan for his insights.

CONTACT INFORMATION

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

Diana Tai
Bristol Myers Squibb
Email: diana.tai@bms.com

Marckenley Mercie
Bristol Myers Squibb
Email: marckenley.mercie@bms.com