PharmaSUG 2020 - Paper DV-057

R for Clinical Reporting, Yes - Let's Explore It!

Hao Meng, Yating Gu, and Yeshashwini Chenna, Seattle Genetics, Inc., Bothell WA

ABSTRACT

R is an interpreted programming language-based software application which can be an ideal platform for statistical analysis and data visualization. For biostatisticians and programmers in the pharmaceutical and biotech industry, it offers a wide and rapidly growing range of user-developed packages containing functions which can efficiently manipulate complex data sets and create tables, figures, and listings. While SAS® remains a critical tool in these industries, the popularity of RStudio® in both academia and the clinical industry increased exponentially over the last decade since it is free and open-source, has powerful statistical support and advanced visualization through its huge user base and extension packages.

As RStudio® is gaining popularity and given that regulatory agencies have not endorsed any particular software for clinical trial analysis and submission, understanding the competency of R and being well-positioned to use it in a clinical data reporting environment is a worthy endeavor.

This paper introduces a pragmatic application of RStudio® by describing an actual use case of clinical trial data manipulation and export (e.g., SDTM and ADaM to XPORT format), creation of tables, figures, and listings, and a simulation use case. We will share the RStudio® packages used as well as pros and cons between RStudio® and SAS®. This paper also provides a brief background to the RStudio® platform and our programming environment set-up, along with relevant statistical programming details.

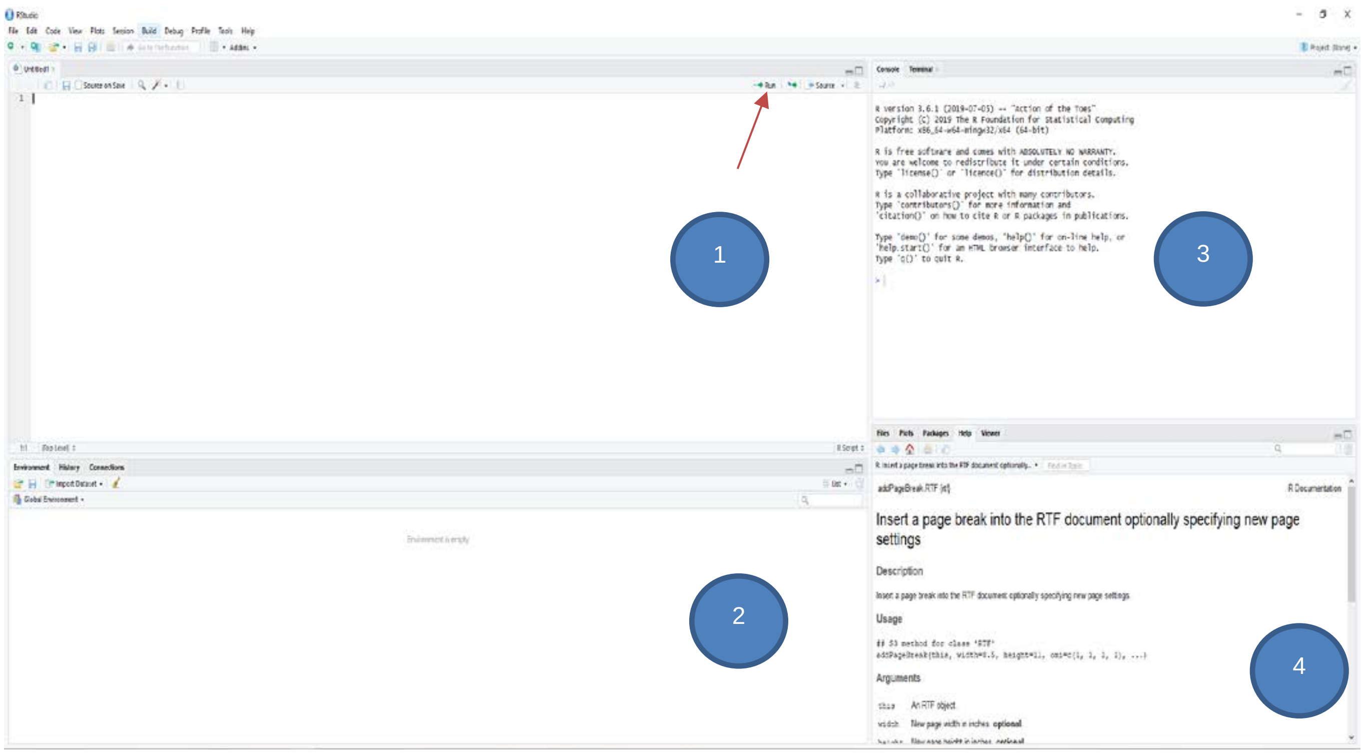
INTRODUCTION

With new technological advancements in pharma industry, we have learned that R/ RStudio® can be a powerful tool for our daily life. There have been many papers/presentations demonstrating the power of R in supporting innovative statistical methodologies and advanced visualization. On the other hand, for a statistical programmer working in pharma industry, creating SDTM, ADaM and Table, Listing, Figure (TFL) takes a huge part in our day-to-day work. This type of work has been predominantly done using SAS®. There have not been many discussions on how competent R/RStudio® utilization in this particular area. In this paper, we will share our experience in exploring the potential of R/RStudio® in handling SDTM, ADaM and TFL tasks which usually done by SAS®, and assessing the pros and cons of handling such task in R over SAS®.

In this paper, we also discuss the use of RStudio®. RStudio® is an integrated development environment (IDE) for the R language, offering a code editor and development environment. Similar to Spyder® to Python^{TM} and SAS® Studio to SAS®. If R is your GoPro, then RStudio® brings the accessories! Without it you'll be fine, but with it you'll be a lot better. For example, RStudio® provides the ability to call up potential syntax options with the tab key which makes it easier to write new scripts, offers a convenient interface to view and interact with objects stored in the environment, exposes click buttons to set the working directory and more accessible graphics such as zoom in the output plots, export figures in PDF/PNG by simply clicking button etc.

A BRIEF INTRODUCTION OF RSTUDIO®

Figure 1 shows a typical view of the RStudio® interface while working on a given program (script).



```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

addPageBreak.RTF {rtf} R Documentation

Insert a page break into the RTF document optionally specifying new page settings

Description

Insert a page break into the RTF document optionally specifying new page settings

Usage

```
## S3 method for class 'RTF'
addPageBreak(this, width=8.5, height=11, omi=c(1, 1, 1, 1), ...)
```

Arguments

this	An RTF object
width	New page width in inches. **optional**
height	New page height in inches. **optional**

Figure 1 RStudio interface

1. Script source window:
   - Area for code development
   - Code will not be evaluated or executed until you hit 'Run' button

2. Environment / History window
   - Lists objects that exist in the working space
   - View command history (like SAS® log)

3. Console
   - Here code from the script source is evaluated by R
   - Also allow you to perform quick calculations that you don't need to save

4. Files/ Plots/ Packages/ Help
   - Here you can see file folders, plot output, browse and install available package, access R help

To evaluate the capability of creating SDTM, ADaM and TFL outputs by R/RStudio® , we selected a completed study and aimed to reproduce all its SDTM and ADaM datasets, TFLs (as shown in Figure 2), and to conduct one simulation example using RStudio®. This paper will summarize the results of our experiments, as well as the pros and cons between SAS® and R/RStudio® in these use cases.

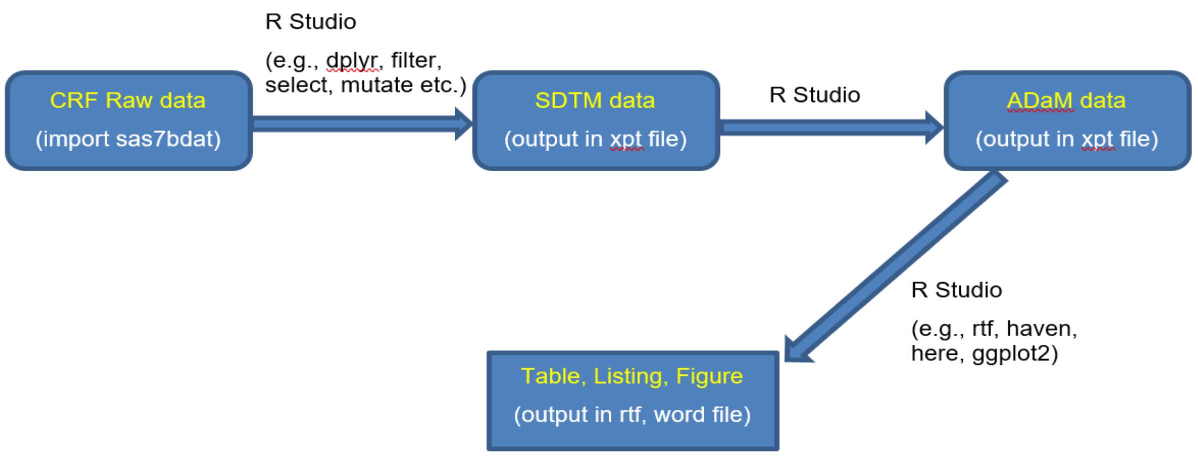


Figure 2 Diagram of steps

USING RSTUDIO® FOR SDTM AND ADAM DATA SETS

For a programmer who is a seasoned SAS® user in pharma/biotech industry but new to R/RStudio® application, the following details will be helpful in understanding the basics of R. There are several R packages that are widely used to provide different functionalities to import SAS® datasets into R session. “Packages” in R is similar to procedures in SAS. After R is installed, there are several R packages come as default, however often time we would need to install our self. A good starting point to decide which package serves our job better is to Google ‘R package for XXX job’. It is important to make sure we have the required packages available for programming during an active R session.

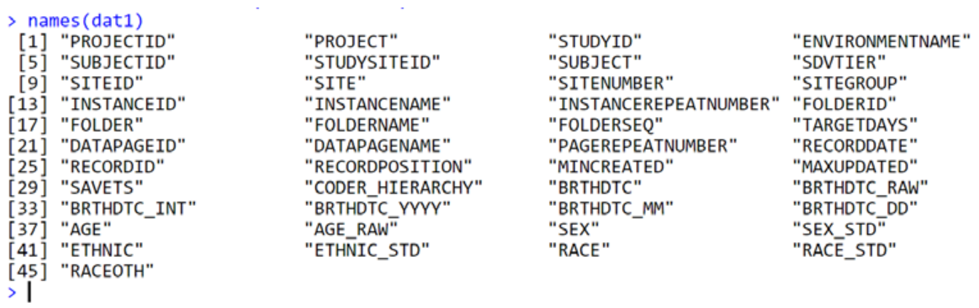
1. The code snippet below installs an R-package using the install.packages() function and loading the packages for any current R session using the library() function. Please note in R, packages need to be installed at least once, after the first time installation, we can load the specific package in future unless the package got deleted afterwards.

```
install.packages(“SASxport”)
library(SASxport)
library(sas7bdat)
library(dplyr)
```

   NOTE: It is easy to get confused between library() in R and libref in SAS, library() here is mainly loading the procedures to be used

2. Using the read.sas7bdat() function we can read in a SAS® dataset; we’ll use a standard SDTM demographics data set (dm.sas7bdat) as an example here and assign it to a memory construct we chose to name “dat1.” Once we have the dataset in R’s current session identified by dat1, we can use the names() function to list all the variables that are present in the dataset. The Rstudio® Console window will then show the output presented in Figure 3.

```
dat1 <- read.sas7bdat(“dm.sas7bdat”)
names(dat1)
```



```
> names(dat1)
 [1] "PROJECTID"         "PROJECT"           "STUDYID"               "ENVIRONMENTNAME"
 [5] "SUBJECTID"         "STUDYSITEID"       "SUBJECT"               "SDVTIER"
 [9] "SITEID"            "SITE"              "SITENUMBER"            "SITEGROUP"
[13] "INSTANCEID"        "INSTANCENAME"      "INSTANCEREPEATNUMBER"  "FOLDERID"
[17] "FOLDER"            "FOLDERNAME"        "FOLDERSEQ"             "TARGETDAYS"
[21] "DATAPAGEID"        "DATAPAGENAME"      "PAGEREPEATNUMBER"      "RECORDDATE"
[25] "RECORDID"          "RECORDPOSITION"    "MINCREATED"            "MAXUPDATED"
[29] "SAVETS"            "CODER_HIERARCHY"   "BRTHDTC"               "BRTHDTC_RAW"
[33] "BRTHDTC_INT"       "BRTHDTC_YYYY"      "BRTHDTC_MM"            "BRTHDTC_DD"
[37] "AGE"               "AGE_RAW"           "SEX"                   "SEX_STD"
[41] "ETHNIC"            "ETHNIC_STD"        "RACE"                  "RACE_STD"
[45] "RACEOTH"
> |
```

Figure 3

3. Next, we can use the select() function to keep selected variables from the main dataset in dat1.

```
dat2 <-
select(dat1,”PROJECT”,”SUBJECT”,”SITEID”,”RACE”,”ETHNIC”,”SEX”,”AGE”)
```

4. We can apply the distinct() function to select records with unique values in our dataset. For example, when we read in vendor data, sometimes each subject can have more than one record. If our analysis requires that we retain only 1 record per subject per treatment, we will want to subset the dataset by picking a unique record for each subject.

```
dset <- read.sas7bdat(“lab_vendor.sas7bdat”)
dat3 %>% arrange(PATIENT_NUMBER, TREATMENT_DECODE, TREATMENT_CODE ,
desc(RECORDID)) %>%
distinct(dset,”PATIENT_NUMBER”,”TREATMENT_DECODE”,”TREATMENT_CODE”,
keep_all = TRUE)
```

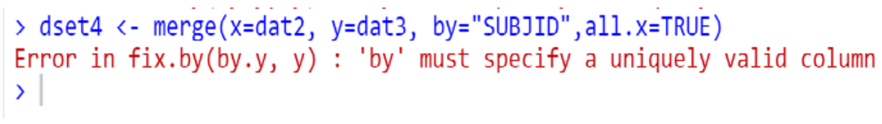
   The distinct() function in R is similar to the NODUPKEY option on PROC SORT in SAS. Variables listed in parentheses are used for unique key(s), the output dataset contains all variables by specifying keep_all = TRUE, which returns variables used for unique key and variables not used for unique key. Also, to decide which duplicated records to keep, we can use arrange() function to sort first and the first row will be kept.

5. Merging datasets: The “merge” function is straightforward and simple to use to merge 2 datasets. Taking the example of the 2 datasets that are created above:

```
dm1 <- merge(x=dat2,y=dat3, by=”SUBJECT”,all.x=TRUE)
```

   Here we are specifying x is the first table and y is the second table, with all.x = TRUE, this merging will keep all records from x=dat2 table, similar to a left join. Note that for merging 2 datasets in R, similar in SAS, the variable(s) or the column(s) used to merge should be present in both datasets.

```
dset4 <- merge(x=dat2,y=dat3,by=”SUBJID”,all.x=TRUE)
```

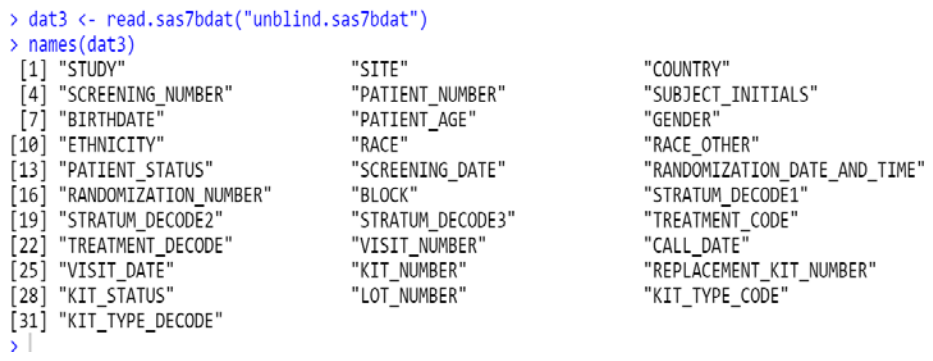


```
> dset4 <- merge(x=dat2, y=dat3, by="SUBJID",all.x=TRUE)
Error in fix.by(by.y, y) : 'by' must specify a uniquely valid column
> |
```

   This error happens here because variable SUBJID is not present in dat2.

6. Assigning column names by using the colnames() function is similar to creating new variables in SAS®. In the example shown in Figure 4 below, we are assigning the column name “SUBJID” in the dat1 dataset

and dat3 datasets. The number reference in the colnames() functions is for the position of the variable listed in the dataset when we use the names() function.



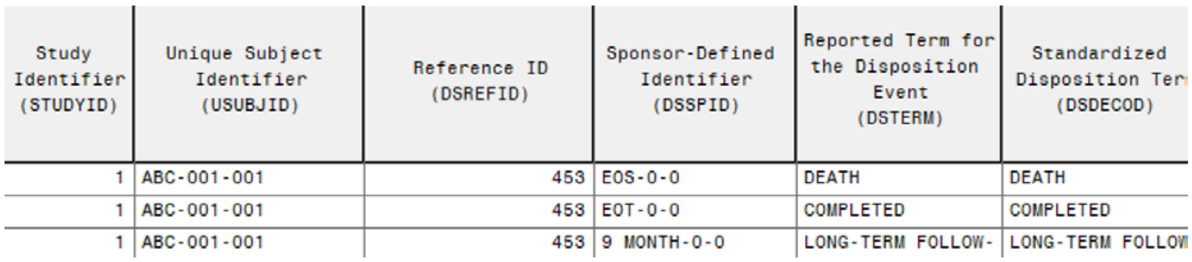
```
> dat3 <- read.sas7bdat("unblind.sas7bdat")
> names(dat3)
 [1] "STUDY"                  "SITE"                "COUNTRY"
 [4] "SCREENING_NUMBER"       "PATIENT_NUMBER"      "SUBJECT_INITIALS"
 [7] "BIRTHDATE"              "PATIENT_AGE"         "GENDER"
[10] "ETHNICITY"              "RACE"                "RACE_OTHER"
[13] "PATIENT_STATUS"         "SCREENING_DATE"      "RANDOMIZATION_DATE_AND_TIME"
[16] "RANDOMIZATION_NUMBER"   "BLOCK"               "STRATUM_DECODE1"
[19] "STRATUM_DECODE2"        "STRATUM_DECODE3"     "TREATMENT_CODE"
[22] "TREATMENT_DECODE"       "VISIT_NUMBER"        "CALL_DATE"
[25] "VISIT_DATE"             "KIT_NUMBER"          "REPLACEMENT_KIT_NUMBER"
[28] "KIT_STATUS"             "LOT_NUMBER"          "KIT_TYPE_CODE"
[31] "KIT_TYPE_DECODE"
>
```

Figure 4

```
colnames(dat3)[5] <- “SUBJID”
```

Assigning column labels by using the label() function, as shown in Figure 5:

```
label(ds$STUDYID) <- "Study Identifier"
label(ds$USUBJID) <- "Unique Subject Identifier"
label(ds$DSREFID) <- "Reference ID"
```

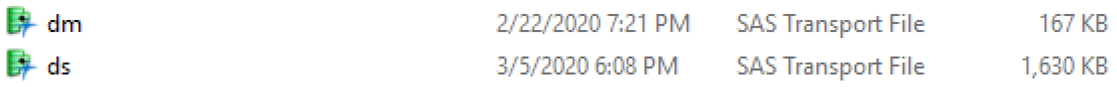


Study Identifier (STUDYID)	Unique Subject Identifier (USUBJID)	Reference ID (DSREFID)	Sponsor-Defined Identifier (DSSPID)	Reported Term for the Disposition Event (DSTERM)	Standardized Disposition Ter (DSDECOD)
1	ABC-001-001	453	EOS-0-0	DEATH	DEATH
1	ABC-001-001	453	EOT-0-0	COMPLETED	COMPLETED
1	ABC-001-001	453	9 MONTH-0-0	LONG-TERM FOLLOW-	LONG-TERM FOLLOW

Figure 5

7. Exporting the final dataset in xpt format. Using write.xport(), we can export the final dataset created in R to SAS® in xpt format as shown in Figure 6.

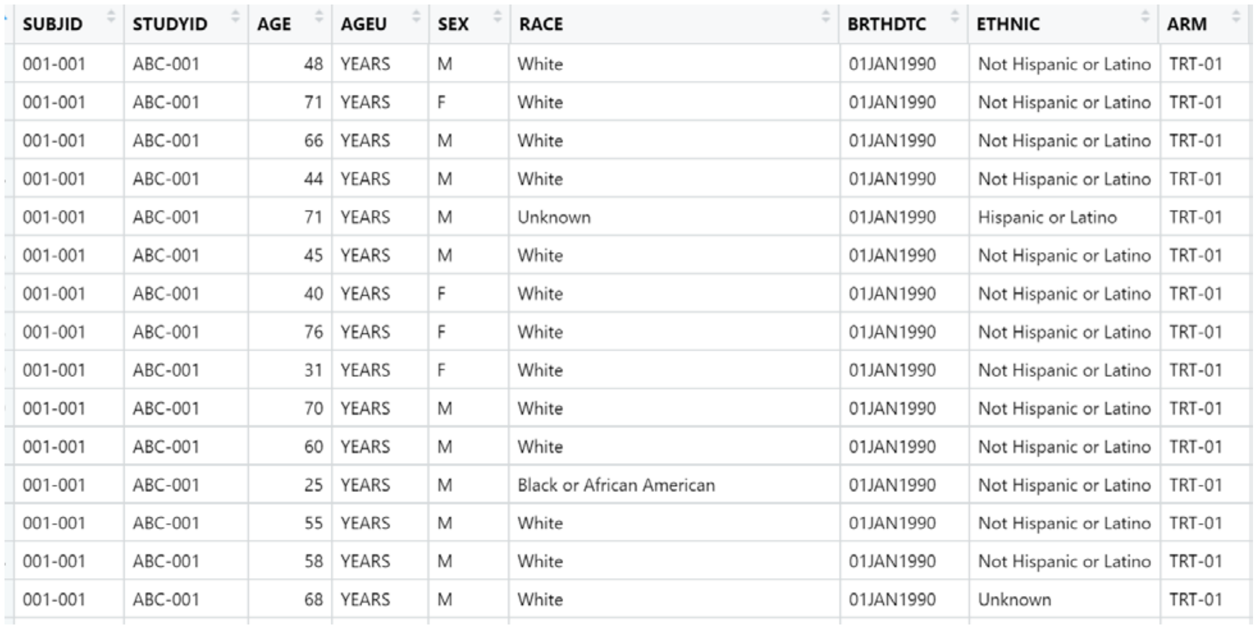
```
write.xport(dm_f2,file=paste(getwd(), "dm.xpt", sep="/"),autogen.formats =
FALSE
```



dm	2/22/2020 7:21 PM	SAS Transport File	167 KB
ds	3/5/2020 6:08 PM	SAS Transport File	1,630 KB

Figure 6

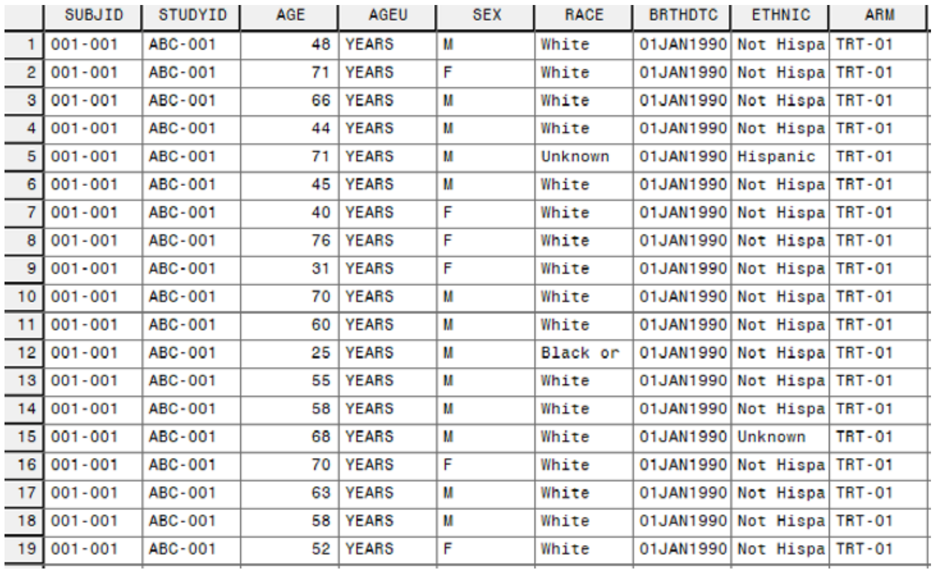
8. Our final dataset is partially shown in Figure 7 as displayed in the RStudio®:



SUBJID	STUDYID	AGE	AGEU	SEX	RACE	BRTHDTC	ETHNIC	ARM
001-001	ABC-001	48	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	71	YEARS	F	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	66	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	44	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	71	YEARS	M	Unknown	01JAN1990	Hispanic or Latino	TRT-01
001-001	ABC-001	45	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	40	YEARS	F	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	76	YEARS	F	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	31	YEARS	F	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	70	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	60	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	25	YEARS	M	Black or African American	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	55	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	58	YEARS	M	White	01JAN1990	Not Hispanic or Latino	TRT-01
001-001	ABC-001	68	YEARS	M	White	01JAN1990	Unknown	TRT-01

Figure 7

9. Figure 8 is our final XPT dataset as shown in SAS® Viewer:



	SUBJID	STUDYID	AGE	AGEU	SEX	RACE	BRTHDTC	ETHNIC	ARM
1	001-001	ABC-001	48	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
2	001-001	ABC-001	71	YEARS	F	White	01JAN1990	Not Hispa	TRT-01
3	001-001	ABC-001	66	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
4	001-001	ABC-001	44	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
5	001-001	ABC-001	71	YEARS	M	Unknown	01JAN1990	Hispanic	TRT-01
6	001-001	ABC-001	45	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
7	001-001	ABC-001	40	YEARS	F	White	01JAN1990	Not Hispa	TRT-01
8	001-001	ABC-001	76	YEARS	F	White	01JAN1990	Not Hispa	TRT-01
9	001-001	ABC-001	31	YEARS	F	White	01JAN1990	Not Hispa	TRT-01
10	001-001	ABC-001	70	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
11	001-001	ABC-001	60	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
12	001-001	ABC-001	25	YEARS	M	Black or	01JAN1990	Not Hispa	TRT-01
13	001-001	ABC-001	55	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
14	001-001	ABC-001	58	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
15	001-001	ABC-001	68	YEARS	M	White	01JAN1990	Unknown	TRT-01
16	001-001	ABC-001	70	YEARS	F	White	01JAN1990	Not Hispa	TRT-01
17	001-001	ABC-001	63	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
18	001-001	ABC-001	58	YEARS	M	White	01JAN1990	Not Hispa	TRT-01
19	001-001	ABC-001	52	YEARS	F	White	01JAN1990	Not Hispa	TRT-01

Figure 8

NOTE: Unlike SAS®, the dataset, variable, and function names in R are case-sensitive.

USING RSTUDIO® FOR TABLES AND LISTINGS

CREATE TABLES AND LISTINGS FROM ADAM DATASETS IN R

The first step is to read in the ADaM dataset using the function sasxport.get() from package “Hmisc” in RStudio®:

```
library(Hmisc)
adsl <- sasxport.get("adsl.xpt", lowernames=FALSE)
```

An alternative way to read in SAS® transport format is to use the function read_xpt() from package “haven”, which is not shown further in this paper. The following code snippets show how we subset data to select variables and observations:

```
ds <- adsl[c("SUBJID", "TRT01P", "TRT01A", "SAFFL", "ITTFL")]
mydata <- ds[which(adsl$ITTFL=="Y"),]
```

We can define the column headers in final listing output by renaming variables and here is an example how to rename variables “SUBJID” and “TRT01P”:

```
colnames(mydata)[colnames(mydata)=="SUBJID"] <- "Subject Number"
colnames(mydata)[colnames(mydata)=="TRT01P"] <- "Treatment Randomized"
```

In order to batch run all outputs in the future, we save all the meta data of outputs in a centralized EXCEL sheet, with a separate tab for each output (see Table 1.). A separate R program is set up to capture all the output programs that we want to batch run and each output can be added by using the below format of the Adverse event table and Disposition listing (i.e., t_ad.R, l_ds.R):

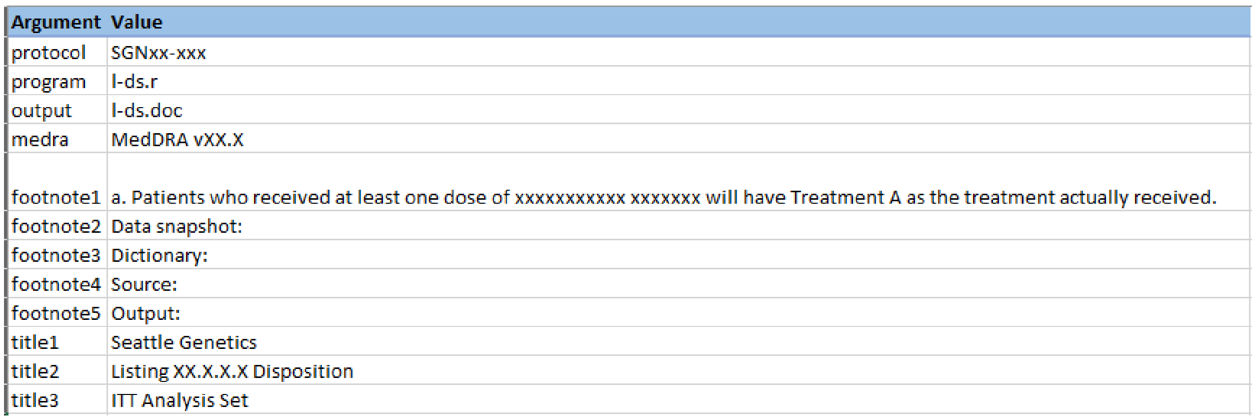
```
setwd("O:/ClinProg/Initiatives/R evaluation/pgm/TFL")

script <- "t_ad.R"
sheet <- 1
source(script)

script <- "l_ds.R"
sheet <- 2
source(script)
```

Table 1 displays the metadata information within the Disposition listing tab of the centralized EXCEL file:

Table 1 Metadata of Disposition Listing Saved in a Centralized Excel File



Argument	Value
protocol	SGNxx-xxx
program	l-ds.r
output	l-ds.doc
medra	MedDRA vXX.X
footnote1	a. Patients who received at least one dose of xxxxxxxxxxx xxxxxxx will have Treatment A as the treatment actually received.
footnote2	Data snapshot:
footnote3	Dictionary:
footnote4	Source:
footnote5	Output:
title1	Seattle Genetics
title2	Listing XX.X.X.X Disposition
title3	ITT Analysis Set

The next step is to read in the metadata so that it can later be used when exported to an RTF file.

The below sample code snippets show how the metadata is read in:

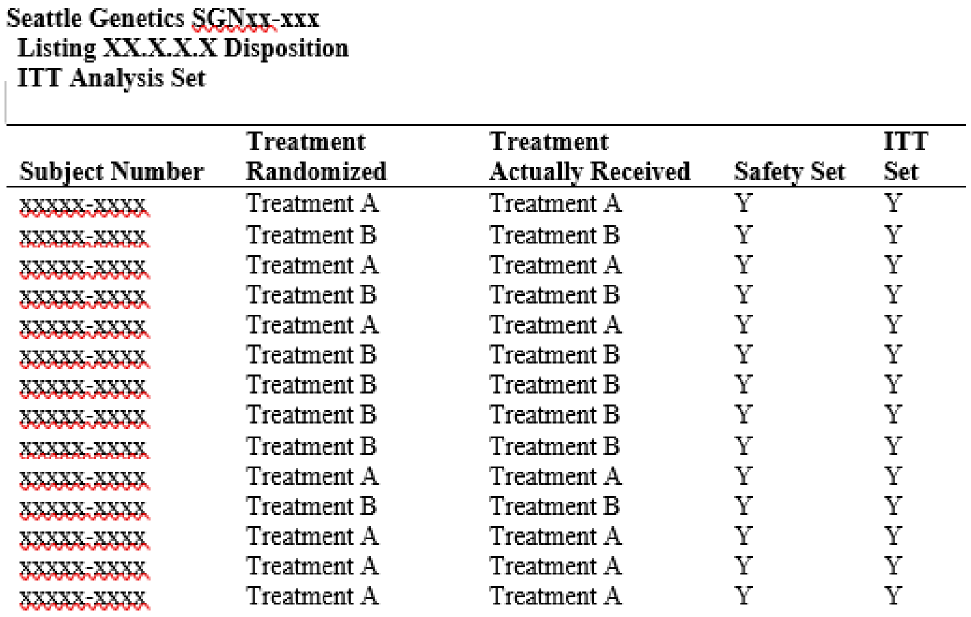
```
library(readxl)
df <- data.frame(read_excel("centralized text file.xlsx", sheet = sheet))
tlf <- toString(df$Value[df$Argument=="output"])
pgm <- toString(df$Value[df$Argument=="program"])
protocol <- toString(df$Value[df$Argument=="protocol"])
```

To export results to a Rich Text Format (RTF) file output, we use package “rtf” by calling a set of its R functions (details of this code are clarified in the text that follows it):

```
library(rtf)
rtf <- RTF(tlf)
nrow = dim(mydata)[1]
line = 43
i = 1
j = i + line
while (j <= nrow)
{
    addHeader(rtf, title=paste(title1, protocol, "\n", title2, "\n",
    title3))
    addTable(rtf, as.data.frame(mydata[i:j,]))
    addNewLine(rtf)
    addParagraph(rtf, footnote1)
    addParagraph(rtf, paste(footnote2, " ", snapshot_c, ";", " ",
    footnote3, " ", dic, sep=''))
    addParagraph(rtf, paste(footnote4, " ", getwd(),  "/", pgm, " ",
    footnote5, " ", tlf, " (", Sys.time(), ")", sep=''))
    i = j + 1
    j = i + line
    if (i < nrow & j > nrow)
    {
          j = nrow
    }
}
done(rtf)
```

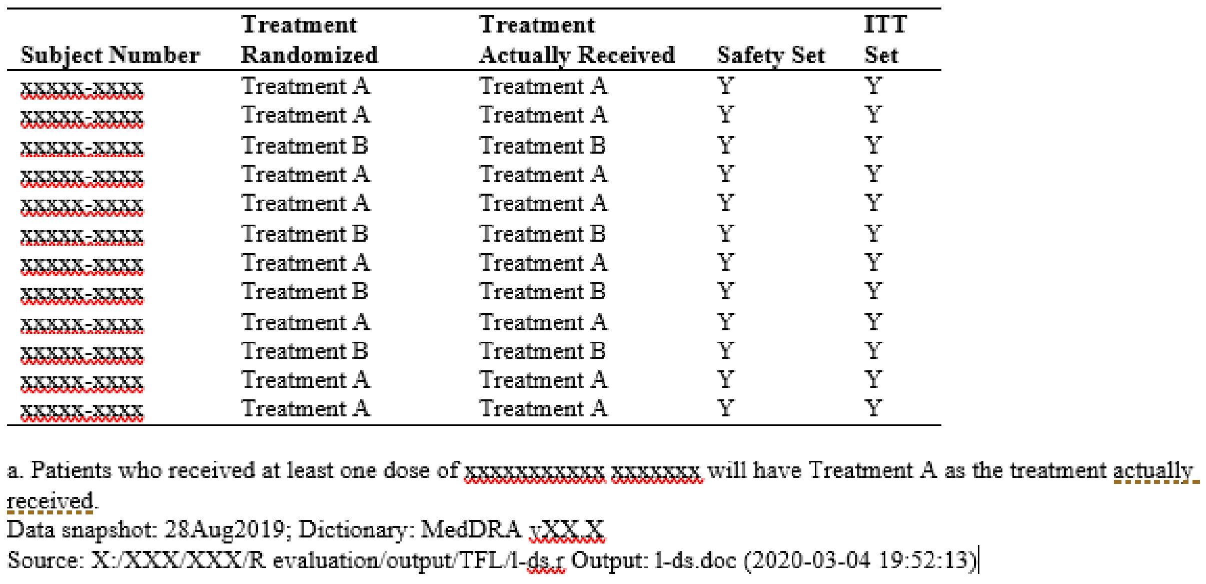
The RTF document can be created using the RTF() function. Text can be added as a title using the R function addHeader(). The main content of the listing can be inserted using the R function addTable(). We use the R function addNewLine() to create a blank line after adding contents. Text can be added as a footnote using the R function addParagraph(). There is a R function addPageBreak() which is designed to insert a page break into the RTF document, however, it can only split the main listing contents, and the titles and footnotes will be displayed once only at the very beginning (page 1) and the very end (which could be page 100 for example), respectively.

Example shown in Table 2 and Table 3:


Seattle Genetics SGNxx-xxx
Listing XX.X.X.X Disposition
ITT Analysis Set

Subject Number	Treatment Randomized	Treatment Actually Received	Safety Set	ITT Set
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y

Table 2. A Failure Page Break Example by Using the R Function addPageBreak() as Footnotes Are Missing in the First Page of the Disposition Listing



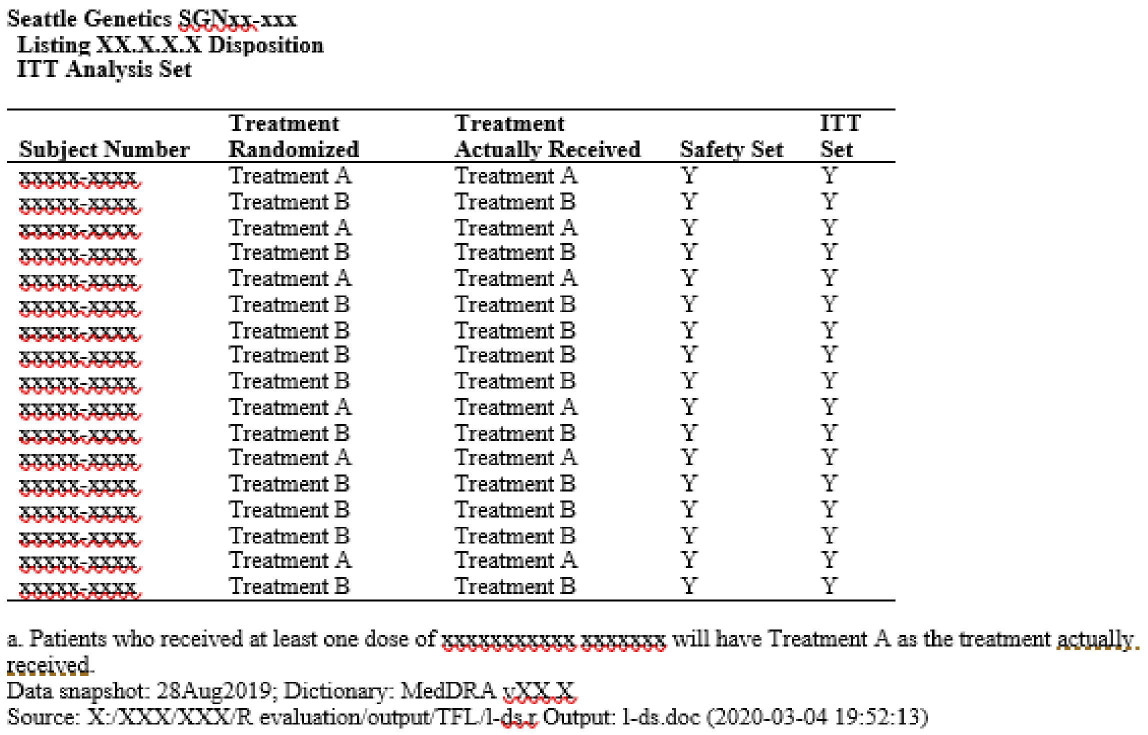
Subject Number	Treatment Randomized	Treatment Actually Received	Safety Set	ITT Set
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y

a. Patients who received at least one dose of xxxxxxxxxxx xxxxxxx will have Treatment A as the treatment actually received.
Data snapshot: 28Aug2019; Dictionary: MedDRA vXX.X
Source: X:/XXX/XXX/R evaluation/output/TFL/l-ds.r Output: l-ds.doc (2020-03-04 19:52:13)

Table 3. A Failure Page Break Example by Using the R Function addPageBreak() as Titles Are Missing in the Last Page of the Disposition Listing

In order to appropriately display the metadata in each page, a WHILE loop is used instead. Firstly, we use the R function dim() to find the total number of rows in the listing data. For this particular disposition listing, we arbitrarily set the number of rows displayed per page to be 43. Then we use a WHILE loop to display 43 lines (records) per page as well as adding titles and footnotes to each page.

Table 4 shows part of the Disposition listing output:



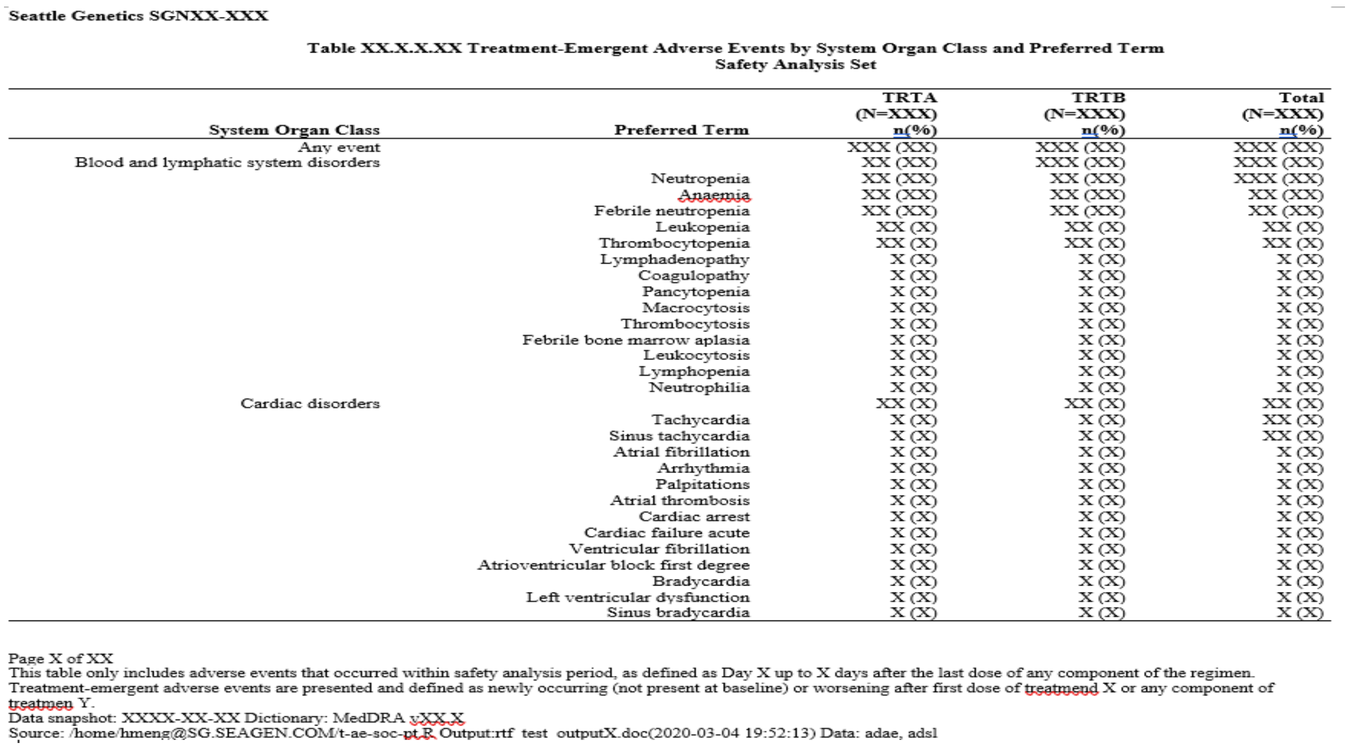
Seattle Genetics SGNxx-xxx
Listing XX.X.X.X Disposition
ITT Analysis Set

Subject Number	Treatment Randomized	Treatment Actually Received	Safety Set	ITT Set
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y
xxxxx-xxxx	Treatment A	Treatment A	Y	Y
xxxxx-xxxx	Treatment B	Treatment B	Y	Y

a. Patients who received at least one dose of xxxxxxxxxxx xxxxxxx will have Treatment A as the treatment actually received.
Data snapshot: 28Aug2019; Dictionary: MedDRA vXX.X
Source: X:/XXX/XXX/R evaluation/output/TFL/l-ds.r Output: l-ds.doc (2020-03-04 19:52:13)

Table 4. Sample Disposition Listing

We can create an adverse event subject incidence table in a similar manner as shown in Table 5:



Seattle Genetics SGNXX-XXX

Table XX.X.X.XX Treatment-Emergent Adverse Events by System Organ Class and Preferred Term
Safety Analysis Set

System Organ Class	Preferred Term	TRTA (N=XXX) n(%)	TRTB (N=XXX) n(%)	Total (N=XXX) n(%)
Any event		XXX (XX)	XXX (XX)	XXX (XX)
Blood and lymphatic system disorders		XX (XX)	XXX (XX)	XXX (XX)
	Neutropenia	XX (XX)	XX (XX)	XXX (XX)
	Anaemia	XX (XX)	XX (XX)	XX (XX)
	Febrile neutropenia	XX (XX)	XX (XX)	XX (XX)
	Leukopenia	XX (X)	XX (X)	XX (X)
	Thrombocytopenia	XX (X)	XX (X)	XX (X)
	Lymphadenopathy	X (X)	X (X)	X (X)
	Coagulopathy	X (X)	X (X)	X (X)
	Pancytopenia	X (X)	X (X)	X (X)
	Macrocytosis	X (X)	X (X)	X (X)
	Thrombocytosis	X (X)	X (X)	X (X)
	Febrile bone marrow aplasia	X (X)	X (X)	X (X)
	Leukocytosis	X (X)	X (X)	X (X)
	Lymphopenia	X (X)	X (X)	X (X)
	Neutrophilia	X (X)	X (X)	X (X)
Cardiac disorders		XX (X)	XX (X)	XX (X)
	Tachycardia	X (X)	X (X)	XX (X)
	Sinus tachycardia	X (X)	X (X)	XX (X)
	Atrial fibrillation	X (X)	X (X)	X (X)
	Arrhythmia	X (X)	X (X)	X (X)
	Palpitations	X (X)	X (X)	X (X)
	Atrial thrombosis	X (X)	X (X)	X (X)
	Cardiac arrest	X (X)	X (X)	X (X)
	Cardiac failure acute	X (X)	X (X)	X (X)
	Ventricular fibrillation	X (X)	X (X)	X (X)
	Atrioventricular block first degree	X (X)	X (X)	X (X)
	Bradycardia	X (X)	X (X)	X (X)
	Left ventricular dysfunction	X (X)	X (X)	X (X)
	Sinus bradycardia	X (X)	X (X)	X (X)

Page X of XX
This table only includes adverse events that occurred within safety analysis period, as defined as Day X up to X days after the last dose of any component of the regimen.
Treatment-emergent adverse events are presented and defined as newly occurring (not present at baseline) or worsening after first dose of treatmend X or any component of treatmen Y.
Data snapshot: XXXX-XX-XX Dictionary: MedDRA vXX.X
Source: /home/hmeng@SG.SEAGEN.COM/t-ae-soc-pt.R Output:rtf test outputX.doc(2020-03-04 19:52:13) Data: adae, adsl

Table 5. Sample Table for Adverse Event Subject Incidence

FIGURES

Statistical programmers routinely create graphical representation and visualization of clinical data and analyses. Both SAS® and R have powerful engines for data visualization, and here we will use two basic examples frequently seen in a clinical trial analysis setting to compare the two applications in the areas of:

- Coding effort
- Convenience in generating the plot
- Flexibility in customize our figure

SAS® output

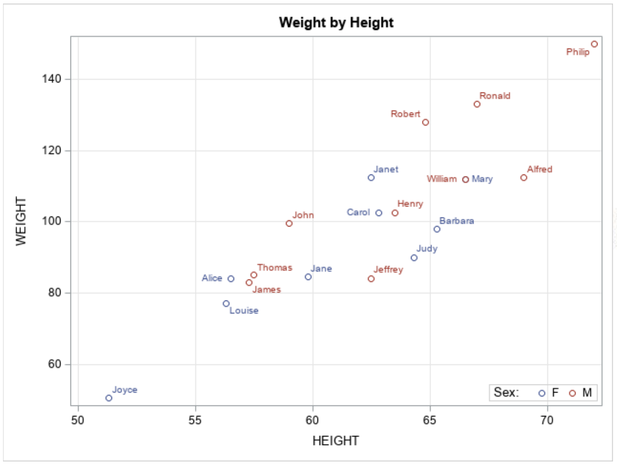


Figure 9

R output

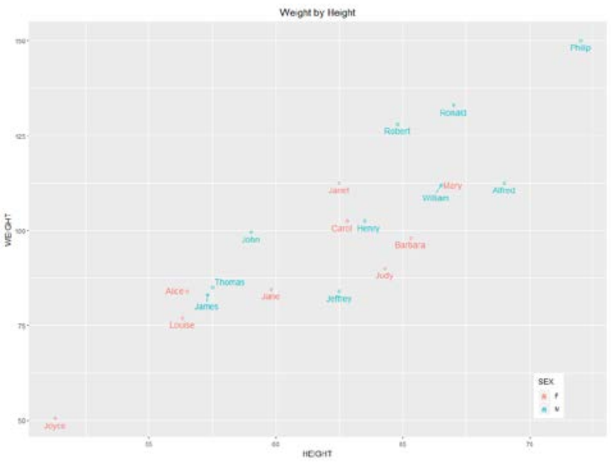


Figure 10

As can be seen from the SAS® and R code below, R is relatively shorter to code. SAS® breaks down to 2 code submissions, designing a plot template and a subsequent application of that template. In contrast, R realizes designing the template while plotting it.

```
SAS® code:
proc template;
      define statgraph Fig_training_test;
       begingraph;
            entrytitle 'Weight by Height';
            layout overlay/ xaxisopts=(griddisplay=on)
                                  yaxisopts=(griddisplay=on);
                  scatterplot x= height y=weight/ group=sex datalabel=name
name='a';
                  discretelegend 'a'/ location=inside title='Sex:'
                              halign=right valign=bottom;
            endlayout;
       endgraph;
      end;
run;

proc sgrender data=sashelp.class template=Fig_training_test;
run;
```

```
R code:
myplot<- ggplot(ds, aes(x=HEIGHT, y=WEIGHT, color=SEX)) +
 geom_point(shape=1) +
 geom_text_repel(label=NAME, vjust = 1.5)+
 labs(title="Weight by Height")+
 theme(legend.position = c(0.9, 0.1), plot.title = element_text(hjust = 0.5))
```

As a second example, we'll use a Kaplan-Meier (KM) plot to compare SAS® vs. R:

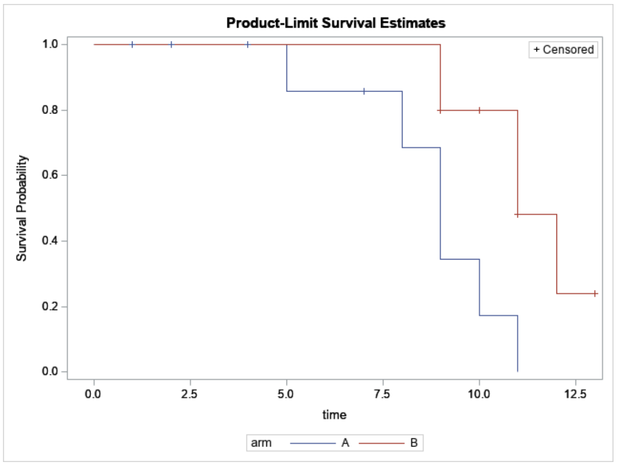


Figure 11

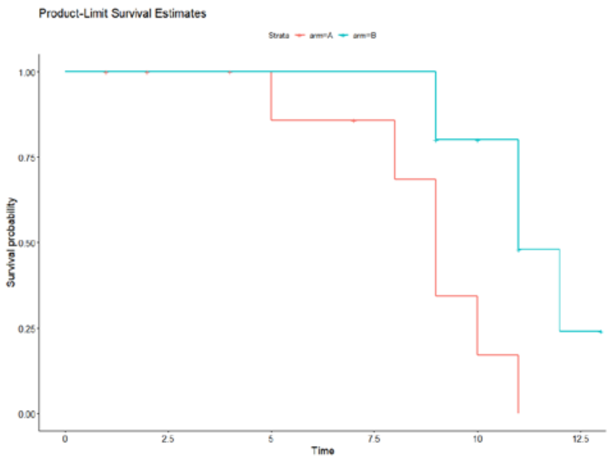


Figure 12

Comparing the SAS® and R code below, R codes are shorter than SAS codes. Also, we can find that the SAS® KM plot is produced as a default from PROC LIFETEST, whereas the fitted model and the plot request in R are separated. That means it depends on the R user's knowledge and judgement whether to include/request a relevant plot to support their analysis.

```
SAS® code:
ods graphics on;
proc lifetest data=a method=km alpha=0.05;
  time time * cnsr(1);
  strata arm /diff=control('B') test= (logrank) adjust=bon;
run;


R code:
fit <- survfit(Surv(time, cnsr2) ~ arm, data = ds2)
myplot2<- ggsurvplot(fit, data = ds2 , title= "Product-Limit Survival
Estimates")
```

Most likely we want the flexibility to change things like titles, axes, and legends. In SAS®, these elements need to be manipulated by updating the STATGRAPH template. Changing this in the Graph Template Language (GTL) is not particularly difficult but a bit tedious. Luckily, R offers many flexible options for graphics plotting without needing to understand detailed template settings. Using the same survival data, the example (Figure 13) below demonstrates an improved version of the KM plot by specifying flexible options.

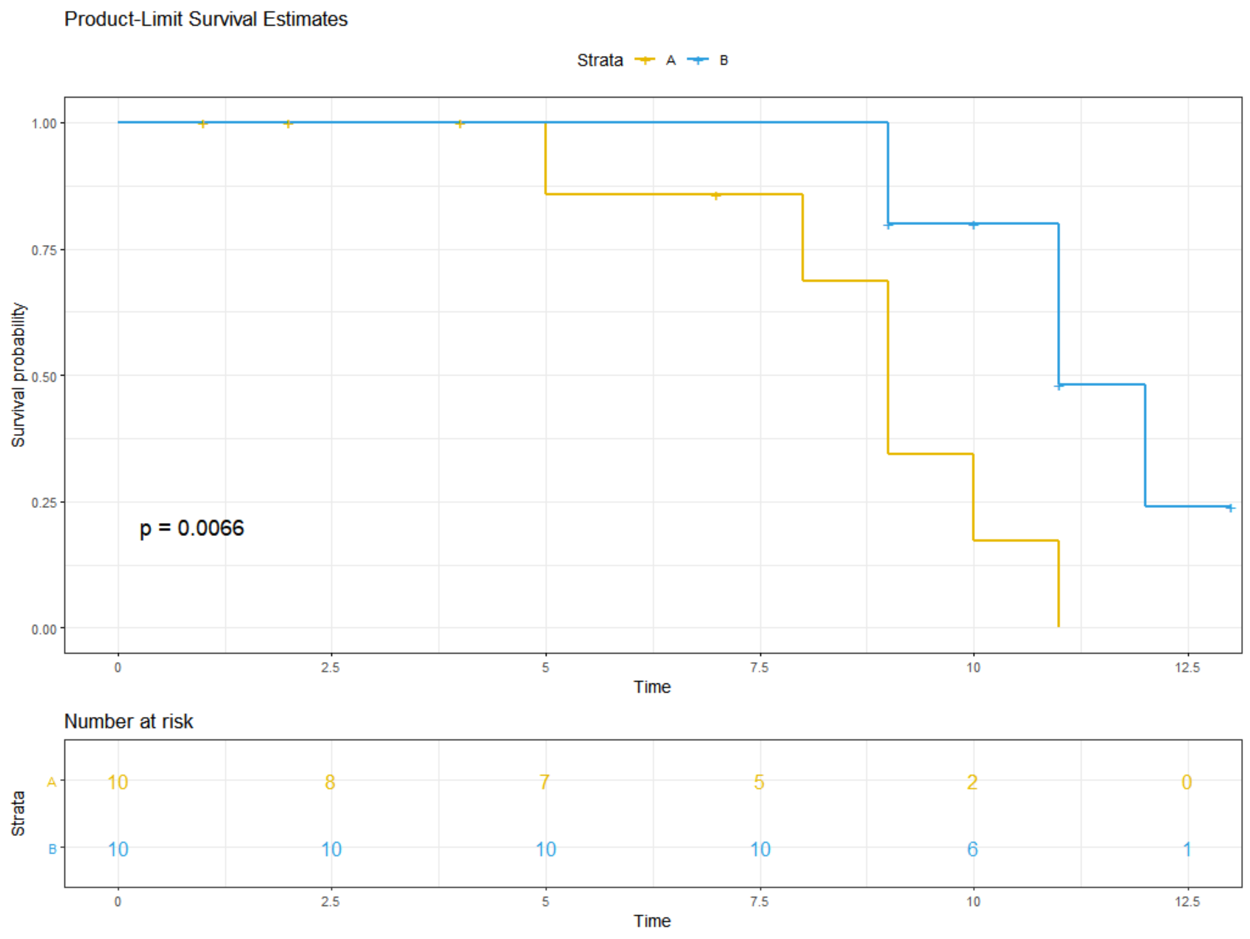


Figure 13

R code:

```
myplot3<-ggsurvplot(
  fit,
  data = ds2,
  size = 1,                 # change line size
  palette =
    c("#E7B800", "#2E9FDF"),# custom color palettes
  conf.int = FALSE,         # Add confidence interval
  pval = TRUE,              # Add p-value
  risk.table = TRUE,        # Add risk table
  risk.table.col = "strata",# Risk table color by groups
  legend.labs =
    c("A", "B"),    # Change legend labels
  risk.table.height = 0.25, # Useful to change when you have multiple groups
  ggtheme = theme_bw()      # Change ggplot2 theme
)
```

SIMULATION

R and RStudio® are open-source software which grows as new methodology needs arise. While SAS® is a commercial and closed-source application, it is catching up quickly. Below we use one simulation use case available in both SAS® and R to compare their performance in different scenarios.

In this example, we simulate data from a logistic regression on local SAS® 9.4 and RStudio® 3.6.1. In local PC SAS®, we do this within a data step by using a do-loop X times:

```
%let _sdtm=%sysfunc(datetime());             #Set start runtime for later calculation
data test;
intercept = 0;                               #Assume an intercept of 0 for linear
predictor
beta = .5;                                   #Assume a slope of 0.5 for linear
predictor
do i = 1 to X;                               #Do loop
   xtest = normal(12345);                    #Generate a sample from normal
distribution
   linpred = intercept + (xtest * beta);     #Calculate linear predictor
   prob = exp(linpred)/ (1 + exp(linpred));  #Take inverse to calculate probability of
success
   ytest = uniform(0) lt prob;               #Test the resulting expit against a
random unif()
   output;
   end;
run;
%let _edtm=%sysfunc(datetime());             #Set end runtime for later calculation
%let _runtm=%sysfunc(putn(&_edtm - &_sdtm, 12.4));   #Calculating runtime
%put It took &_runtm second to run the program;
```

In the local PC RStudio®, we generate X number of random normal variates, calculate the linear predictor and expit for each, then test against X number of random uniform variates.

```
start_time <- Sys.time()                # Set start runtime for later calculation
intercept = 0                                # Set linear predictor parameters
beta = 0.5
xtest = rnorm(X,1,1)                         # Draw sample from normal distribution
linpred = intercept + xtest*beta        # Linear predictor
prob = exp(linpred)/(1 + exp(linpred))       # Expit linear predictor
runis = runif(X,0,1)                         # Draw sample from uniform distribution
ytest = ifelse(runis < prob,1,0)             #Testing the resulting expit against a
random unif()
end_time <- Sys.time()                       #Set end runtime for later calculation
end_time - start_time                        #Calculating runtime
```

Software Runtime (in sec) / Sample size (X)	**SAS**	**RStudio®**
10^3	0.0070	0.0632019
10^4	0.0090	0.06385112
10^5	0.0310	0.08033586
10^6	0.2320	0.6481841
10^7	2.3290	3.866677
10^8	24.3830	25.69618
10^9	247.5530	Failed (Error: cannot allocate vector of size 2.9 Gb)

Table 6

As can be seen from the summary in Table 6 under different scenarios, the run time performance does not have a significant difference. It seems SAS® in general processes this simulation faster, however, it may vary depending on code efficiency and the analytical platform (the results here show the runtimes on a local SAS® 9.4 install vs. a local RStudio® 3.6.1 install on the same PC). It was observed that when the simulated sample size increased to 10^9, there was a memory limitation in RStudio®, which typically allows up to 2 Gb of memory for processing data.

CONCLUSION

Our findings from this research suggest that R is capable to perform the tasks that are typically done in SAS in handling data manipulation, reporting tables and listings, and plotting figures. When it comes to figures, we find R offers a shorter and more straight-forward coding style and provides greater and easier flexibility in modifying the plot (as compared to SAS® GTL).

By applying a simple simulation use case, we find there is no significant difference in runtime between local SAS® and RStudio®. However, given the different ways SAS® and RStudio® load data for processing and limitations in processing RAM, we can see RStudio®'s limitation in this regard. Setting aside this processing limitation, many new statistical methodologies will be implemented and added as ready-to-use packages on R community portals as the need arises. In contrast, with SAS®, users will have to wait for periodic updates in SAS® STAT which in part for this reason may be considered less agile compared to R. On the flipside of that, because the R community is open-source based, anyone can upload their self-created package, therefore user discretion is strongly suggested; while SAS® is considered validated software and would not have this particular concern.

Lastly, it is worth noting that software cost is important to any company, including pharma and biotech ones. We may think that since R and RStudio® are free and SAS® requires an annual license, using R instead of SAS may help lowering cost. That is not always true. For example, since the cost of software is only one part of the equation. To be used in a highly regulated industry such as ours, software validation, maintenance and support are also critical and the cost of these also need to be considered. Although R is free and open-source, it will have a steep learning curve, there is no direct support from the company; instead there are many R support communities. Also, R programmers are in relatively short supply compared to those familiar with SAS®.

In conclusion, with the interactivity provided by RStudio®/Shiny® and potential automation benefit by R Markdown (not covered in this paper), as well as the flexibility and easy coding of popular R packages like ggplot2, we see a definite benefit of performing data visualization using RStudio®. In terms of data manipulation and reporting, staying with SAS® for now will probably be a better choice. As for runtime performance, there is no significant difference using local SAS® vs. RStudio® in our specific case being demonstrated here, however local RStudio® will have a 2 Gb RAM processing limitation. In cases where R offers a significant benefit over SAS® (e.g., limma for bioconductors or R2jags for Bayesian modeling). And when computation effort is challenging in local RStudio®, switching to a server based RStudio® may well be a viable and preferred option, where more processing memory is allowed.

ACKNOWLEDGMENTS

We would like to take this opportunity to thank our manager Adarsh Gunukula, manager in statistical programming, Shefalica Chand, associate director of statistical programming and Wei Qian, director of statistical programming, for reviewing the paper and providing valuable inputs. We also would like to thank Balavenkata R. Pitchuka, Yang Wang, managers in statistical programming for their support in writing this paper.

REFERENCES

https://cran.r-project.org/web/packages/rtf/rtf.pdf

http://sas-and-r.blogspot.com/p/simulation-examples.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors at:

Hao Meng
Seattle Genetics, Inc.
21823 - 30th Drive S.E.
Bothell, WA 98021
hmeng@seagen.com

Yeshashwini Chenna
Seattle Genetics, Inc.
21823 - 30th Drive S.E.
Bothell, WA 98021
ychenna@seagen.com

Yating Gu
Seattle Genetics, Inc.
21823 - 30th Drive S.E.
Bothell, WA 98021
ygu@seagen.com