PharmaSUG 2020 - Paper EP-337

Generating ADaM Compliant ADSL Dataset Using R

Vipin Kumpawat, Eliassen Group Life Science, Somerset NJ, USA

Abstract

In this paper we show how to generate an ADaM compliant ADSL dataset using R. R packages such as Sas7bdat, Dplyr, Tidyr, and Hmisc are used to generate the ADSL dataset. A procedure to set-up the R-environment for the process of generating the ADSL dataset is shown. The typical steps used to create the ADSL dataset along with the derivation of numeric variables, flags, treatment variables and trial dates are outlined. R procedures to attach labels to the variables are discussed. A side by side comparison of R and SAS code is presented. A known weakness in R such as attaching labels to the variables [1] has been resolved in this work. The challenges encountered in generating the ADSL dataset using R are discussed.

Introduction

SAS is widely used in clinical trials. Like SAS, R is a language and environment for statistical computing and graphics. R can be considered as a viable alternative to SAS for generating specialized clinical trials datasets, tables, listings and figures. R is freely available and an open source environment which is supported by R foundation of statistical computing. R has many specific packages available for design, monitoring and analysis of clinical trials datasets, these include Sas7bdat, Dplyr, Tidyr, and Hmisc that enable reading, merging, transposing, attaching label's to variables respectively [2].

ADaM and ADSL Background

ADaM datasets are classified in three structures: ADSL (Subject-Level Analysis Dataset), BDS (Basic Data Structure), and OCCDS (Occurrence Data Structure), as shown in Figure 1 [3]. In this work we focus on the ADSL dataset, which contains key data on demographics, exposure and disposition of a clinical trial. Regardless of the type of the clinical trial design, ADSL dataset contains one record per subject. ADSL is a source for subject -level variables used in other ADaM datasets. ADSL dataset contains variables that include information on demographics, randomization factors, planned and actual treatment, sub grouping, subject -level population flags and important trial dates. The structure of ADSL dataset allows merging with other ADaM and SDTM dataset. The ADSL dataset is used to provide the key facts about the subject, facts that facilitate analysis and interpretation of analysis.

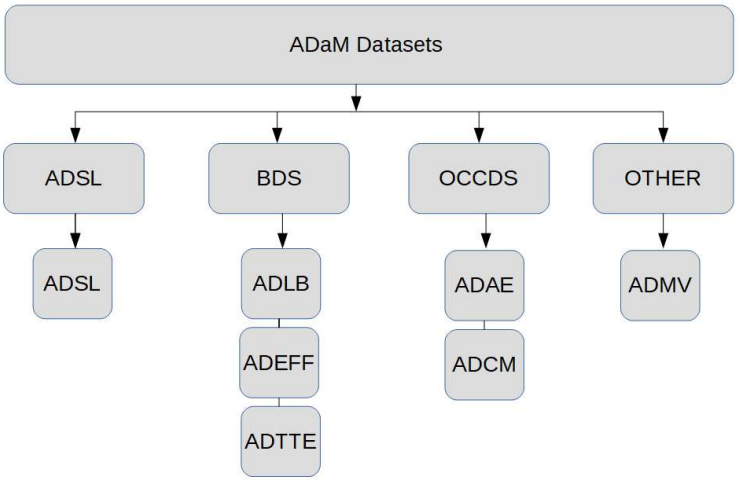

Figure1. Classifications of Analysis Datasets

Steps To Generate ADSL Dataset

The process steps below outline the general steps needed to generate an ADSL dataset, beginning with the reading of the SDTM datasets and ending with the generation of the ADSL dataset.

1: Reading Datasets

SDTM datasets which include DM, SUPPDM, EX, SUPPEX, DS and SUPPDS datasets are imported into the statistical programming environment.

2: Sorting, Transposing and Merging DM and SUPPDM dataset

DM and SUPPDM datasets are sorted, and then SUPPDM dataset is transposed and finally merged with DM dataset.

3: Sorting, Transposing and Merging the DS and SUPPDS dataset.

DS and SUPPDS datasets are sorted, and then SUPPDS dataset is transposed and finally merged with DS dataset.

4: Sorting, Transposing and Merging the EX and SUPPEX dataset.

EX and SUPPEX datasets are sorted, and then SUPPEX dataset is transposed and finally merged with EX dataset.

5: Exposure Variables

Extracting Exposure related variables for period 1 and 2.

6: Combining all the datasets

Exposure and Disposition related variables are merged with DM dataset.

7: Generating Numeric Variables

Numeric variables are generated for RACE and SEX variables.

8: Generating Flags

Subject level population flags such as safety population flag (SAFFL), intent-to treat population flag (ITT) and enrollment population flag (ENRFL) are generated.

9: Generating Treatment Variables

Treatment variables are generated for planned and actual treatment.

10: Generating Trial Dates

Trial dates such as treatment start date and treatment end date are generated.

11: Assigning Labels to the Variables

Labels are assigned to the variables.

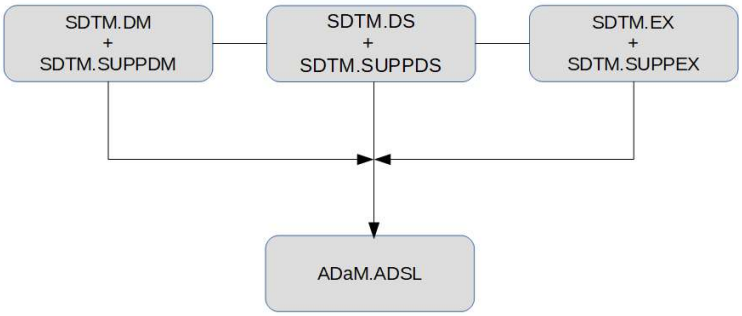

Figure2. ADSL dataset composition

Setting-up the R Environment

The first typical step of using any programming environment is to install the required packages and activate the libraries needed for a program. R packages can be installed in R-Studio through the lower right pane of the R-Studio IDE (4th Quadrant). Figure 3 below shows how to install the Dplyr package; the same procedure can be used to install other R packages namely Sas7bdat, Tidyr and Hmisc shown in Table 1. R Libraries can be installed in R-Studio through console commands as shown in figure 4.

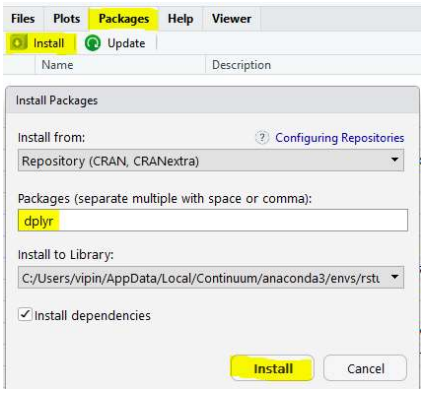

Figure3. R-Studio Snapshot to install R package

```
➔ library(foreign)
➔ library(dplyr)
➔ library(tidyr)
➔ library(hmisc)
```

Figure4. R console command's to install R libraries

Table 1: R Packages and Libraries needed for generating ADSL dataset.

Procedure	R Package	R Library
Read SAS Dataset	Sas7bdat	Foreign
Merge Datasets	Dplyr	Dplyr
Transpose Dataset	Tidyr	Tidyr
Attach Label	Hmisc	Hmsic
Check attributes	Hmisc	Hmsic

R Code for generating the ADSL Dataset

After the R packages and libraries are installed, the R-environment is ready to read and process the input dataset's used to generate the ADSL dataset. Step by step R code to generate the ADSL dataset is shown below. Table 2 summarizes the R functions used in the processes of generating the ADSL dataset.

1. Reading Dm, Suppdm, Ds, Suppds, Ex and Suppex Datasets

```
dm<- read.sas7bdat("c:/sdtm/dm.sas7bdat")
suppdm<- read.sas7bdat("c:/sdtm/suppdm.sas7bdat")
ds<- read.sas7bdat("c:/sdtm/ds.sas7bdat")
suppds<- read.sas7bdat("c:/sdtm/suppds.sas7bdat")
ex<- read.sas7bdat("c:/sdtm/ex.sas7bdat")
suppex<- read.sas7bdat("c:/sdtm/suppex.sas7bdat")
```

2. Combining Dm and Suppdm Dataset by Sorting, Transposing and Merging Steps.

1. Sorting Dm and Suppdm Dataset

```
dm<- dm[order(usubjid), ]
suppdm<- suppdm[order(usubjid), ]
```

2. Transposing Suppdm Dataset

```
suppdm_ <- suppdm %>%
 select(usubjid,qnam,qval) %>%
 spread(.,qnam,qval)
```

3. Merging Dm and Suppdm Dataset

```
dm<- dm[order(usubjid), ]
suppdm_ <- suppdm_[order(usubjid), ]
dmall<- left_join(dm, suppdm_, by = "usubjid")
```

3. Combining Ds and Suppds Dataset by Sorting, Transposing and Merging Steps.

1. Sorting Ds and Suppds Dataset

```
ds<- ds[order(usubjid), ]
suppds<- suppds[order(usubjid), ]
```

2. Transposing Suppds Dataset

```
suppds_ <- suppds %>%
mutate(dsseq = as.numeric(idvarval)) %>%
select(usubjid,qnam,qval) %>%
spread(.,qnam,qval)
```

3. Merging Ds and Suppds Dataset

```
ds<- ds[order(usubjid, dsseq), ]
suppds_ <- suppds_[order(usubjid, dsseq), ]
dsall<- left_join(ds, suppds_, by = c("usubjid", "dsseq")
```

4. Combining Ex and Suppex Dataset by Sorting, Transposing and Merging Steps.

1. Sorting Ex and Suppex Dataset

```
ex<- ex[order(usubjid), ]
suppex<- suppex[order(usubjid), ]
```

2. Transposing Suppex Dataset

```
suppex_ <- suppex %>%
mutate(exseq = as.numeric(idvarval)) %>%
select(usubjid,qnam,qval) %>%
spread(.,qnam,qval)
```

3. Merging Ex and Suppex Dataset

```
ex<- ex[order(usubjid, exseq), ]
suppex_ <- suppex_[order(usubjid, exseq), ]
exall<- left_join(ex, suppex_, by = c("usubjid", "exseq")
```

5. Generating Exposure Variables For Period 1 And Period 2

```
ex1 <- subset(exall, exseq==1)
ex1 <- select(ex1, usubjid, extrt, exdose, exstdtc, exendtc)
ex2 <- subset(exall, exseq==2)
ex2 <- select(ex2, usubjid, extrt, exdose, exstdtc, exendtc)
```

```
ex2<- ex2 %>%
rename(extrt2 = extrt, exdose2 = exdose, exstdtc2 =
exstdtc, exendtc2 = exendtc)
```

6. Combine Exposure and Disposition Datasets With Dm Dataset

```
dmexall<- left_join(dmall, ex1, by = "usubjid")
dmexall<- left_join(dmexall, ex2, by = "usubjid")
adsl<- left_join(dmexall, dsall, by = "usubjid")
```

7. Generate Numeric Variables for Sex and Race Variable

```
adsl$sexn[adsl$sex=="m"]<- 1
adsl$sexn[adsl$sex=="f"]<- 2
adsl$racen[adsl$race=="asian"]<- 1
adsl$racen[adsl$race=="other"]<- 2
```

8. Generate Flags (Saffl, Ittfl, Enrfl)

```
adsl$saffl <- ifelse(!is.na(adsl$exdose) & !is.na(adsl$exstdtc),
      "Y", "N")
adsl$ittfl <- ifelse(!is.na(adsl$armcd), "Y", "N")
adsl$enrfl <- ifelse(!is.na(adsl$rfstdtc) & !is.na(adsl$rficdtc),
      "Y", "N")
```

9. Generate Treatment Variables

1. Treatment Variables for Planned Treatment

```
adsl$trt01p[adsl$armcd=="xx"] <- "refe"
adsl$trt02p[adsl$armcd=="xx"] <- "test"
adsl$trt01pn[adsl$armcd=="xx"] <- 2
adsl$trt02pn[adsl$armcd=="xx"] <- 1
adsl$trt01p[adsl$armcd=="yy"] <- "test"
adsl$trt02p[adsl$armcd=="yy"] <- "refe"
adsl$trt01pn[adsl$armcd=="yy"] <- 1
adsl$trt02pn[adsl$armcd=="yy"] <- 2
```

2. Treatment Variables for Actual Treatment

```
adsl$trt01a[adsl$actarmcd=="xx"] <- "refe"
adsl$trt02a[adsl$actarmcd=="xx"] <- "test"
adsl$trt01an[adsl$actarmcd=="xx"] <- 2
adsl$trt02an[adsl$actarmcd=="xx"] <- 1
adsl$trt01a[adsl$actarmcd=="yy"] <- "test"
adsl$trt02a[adsl$actarmcd=="yy"] <- "refe"
adsl$trt01an[adsl$actarmcd=="yy"] <- 1
adsl$trt02an[adsl$actarmcd=="yy"] <- 2
```

10. Generating Trial Dates

```
adsl$trt01sdt <- adsl$exstdtc
adsl$trt02sdt <- adsl$exstdtc2
```

11. Assigning Labels To The Variables

```
label(adsl$studyid) <- "Study Identifier"
label(adsl$usubjid) <- "Unique Subject Identifier"
```

Similarly attach all the labels to the remaining variables.

12. Checking the Labels

```
describe(adsl)
```

Table 2: R functions used to generate ADSL dataset.

Procedure	R Function
Read SAS Dataset	read.sas7bdat
Merge Datasets	inner_join/left_join/right_ join/full_join
Transpose Dataset	Spread
Attach Label	Label
Variable Selection	Select
Character to Numeric	as.numeric
Numeric to Character	as.character
Check attributes	Content
Sort Dataset	Order
Rename Variable	Rename
Conditional operator	ifelse()
Check Labels	Describe
Check variable type	Class

Comparison of SAS and R Code

There are some key differences between SAS and R programming language . The most basic difference is that SAS is case insensitive while R is case sensitive. There are other key differences [4,5] that pertain to how each language sorts and rounds data and the way each language handle's missing values . SAS and R handle missing values differently in the sorting process. In SAS the missing values will be sorted before the populated value for both numeric and character variable when using the Proc Sort procedure. In R, the character column missing value will be sorted before the populated value while the numeric column missing value will be sorted after populated value when using the order function, which is a Proc Sort equivalent of R. SAS and R handle rounding of numeric values differently resulting in different values, this is due to the difference in the algorithm that each language uses for rounding [4,5].

The Table 3 below compares the code in SAS and R to generate ADSL dataset. Table 3 is a comprehensive procedure step and code reference for generating the ADSL dataset in SAS and R.

Table 3: SAS and R code comparison.

No	Procedure	SAS code	R code

1	Importing and Reading Dataset	`libname sdtm "c:\sdtm";`
		`data dm1;`
		`set sdtm.dm;`
		`run;`
		➔ similarly read suppdm, ds, suppds, ex, suppex
			`library(foreign)`
			`dm <- read.sas7bdat("c:/sdtm/ dm.sas7bdat")`
			➔ similarly read suppdm, ds, suppds, ex, suppex

2	Checking Dataset	`proc contents data = dm;` `run;`	**`library(hmisc)`** `content(dm)`

3	Sorting Dataset	`proc sort data = suppdm;`
		`by usubjid;`
		`run;`
			`suppdm <- suppdm [order(usubjid), ]`

4	Transposing Data	`proc transpose data = suppdm out = suppdmt(drop=_name_ _label_);`
		`by usubjid;`
		`id qnam; var qval;`
		`idlabel qlabel;`
		`run;`
			`library(tidyr)`
			`suppdm_ <- suppdm %>%`
			`select(usubjid, qnam, qval) %>%`
			`spread(., qnam, qval)`

5	Merging Dm, Suppdm	`data dmall;`
		`merge dm1(in=a) suppdmt (in=b);`
		`by usubjid;`
		`if a;`
		`run;`
		➔ similarly dsall, exall datasets generated
			`library(dplyr)`
			`dm <- dm[order(usubjid), ]`
			`suppdm_ <- suppdm_[order(usubjid), ]`
			`dmall <- left_join(dm, suppdm_ , by = "usubjid")`
			➔ similarly dsall, exall datasets generated

6	Ex dataset for Period 1 and 2	`data ex1; set exall;`
		`if exseq = 1;`
		`keep usubjid extrt exdose exstdtc exendtc;`
		`run;`
		
		`data ex2 ; set exall;`
		`if exseq = 2;`
		`extrt2 = extrt;`
		`exdose2 = exdose;`
		`exstdtc2 = exstdtc;`
		`exendtc2 = exendtc;`
		`keep usubjid extrt2 exdose2 exstdtc2 exendtc2; run;`
			`ex1 <- subset(exall, exseq==1)`
			`ex1 <- select(ex1, usubjid, extrt, exdose, exstdtc, exendtc)`
			
			`ex2 <- subset(exall, exseq==2)`
			`ex2 <- select(ex2, usubjid, extrt, exdose, exstdtc, exendtc)`
			
			`ex2 <- ex2 %>% rename(extrt2 = extrt, exdose2 = exdose, exstdtc2 = exstdtc, exendtc2 = exendtc)`

7	Combining Ex, Ds datasets with Dm dataset	`data adsl1;`
		`merge dmall(in=a) ex1 ex2 dsall;`
		`by usubjid;`
		`if a;`
		`run;`
			`dmexall <- left_join(dmall, ex1, by = "usubjid")`
			`dmexall <- left_join(dmexall, ex2, by = "usubjid")`
			`adsl<- left_join(dmexall, dsall, by = "usubjid")`

8	Generating Numeric variable	`data adsl2;`
		`set adsl1;`
		`if sex = "m" then sexn = 1;`
		`else if sex = "f" then sexn = 2;`
		`if race = "asian" then racen =1;`
		`else if race = "other" then racen = 2;run;`
			`adsl$sexn[adsl$sex=="m"]<- 1`
			`adsl$sexn[adsl$sex=="f"]<- 2`
			
			`adsl$racen[adsl$race=="asian"]<- 1`
			`adsl$racen[adsl$race=="other"]<- 2`

9	Generating Safety Flag, ITT Flag, Enrolment Flag	data adsl3; set adsl2; if exdose ^= . and exstdtc ^= '' then saffl = "Y"; else saffl = "N"; if armcd ^= '' then ittfl = 'Y'; else ittfl = 'N'; if rfstdtc ^= " " and rficdtc ^= " " then enrfl = 'Y'; else enrfl = 'N'; run;	adsl$saffl <- ifelse(!is.na(adsl$exdose) & ! is.na(adsl$exstdtc), "Y", "N") adsl$ittfl <- ifelse(!is.na(adsl$armcd), "Y", "N") adsl$enrfl <- ifelse(!is.na(adsl$rfstdtc) & ! is.na(adsl$rficdtc), "Y", "N")

12	Treatment Variable :-
	
	TRT01P
	TRT02P
	TRT01PN
	TRT02PN
	
	TRT01A
	TRT02A
	TRT01AN
	TRT02AN
		data adsl4;
		set adsl3;
		if armcd = "xx" then do;
		trt01p = " refe";
		trt02p = " test ";
		trt01pn = 2;
		trt02pn = 1;
		end;
		if armcd = "yy" then do;
		trt01p = " test";
		trt02p = " refe";
		trt01pn = 1;
		trt02pn = 2;
		end; run;
		Similarly for actual treatment
			Treatment variables for planned treatment
			
			adsl$trt01p[adsl$armcd=="xx"] <- "refe"
			adsl$trt02p[adsl$armcd=="xx"] <- "test"
			
			adsl$trt01pn[adsl$armcd=="xx"] <- 2
			adsl$trt02pn[adsl$armcd=="xx"] <- 1
			
			adsl$trt01p[adsl$armcd=="yy"] <- "test"
			adsl$trt02p[adsl$armcd=="yy"] <- "refe"
			
			adsl$trt01pn[adsl$armcd=="yy"] <- 1
			adsl$trt02pn[adsl$armcd=="yy"] <- 2
			
			Similarly for actual treatment

13	Trial Dates	data adsl5; set adsl4; trt01sdt = exstdtc ; trt02sdt = exstdtc2 ; run;	adsl$trt01sdt <- adsl$exstdtc adsl$trt02sdt <- adsl$exstdtc2
14	Attaching Labels	proc sql ; create table adsl as select studyid "study identifier", . . trt02sdt "P-02 start date " from adsl5; quit;	**library(hmisc)** label(adsl$studyid) <- "Study Identifier" . . label(adsl$trt02sdt) <- "P-02 Start date" describe(adsl) #To check the labels

Results and Discussion

The key issue that this paper addresses is attaching labels to the variables. In R, labels can be attached to the variable by using "Hmisc" package. After loading Hmisc package we start calling Hmisc library by following code.

```
library(hmisc)
```

Using label function in Hmisc, we add the label to each variable by following code.

```
label(adsl$studyid) <- "Study Identifier"
label(adsl$usubjid) <- "Unique Subject Identifier"
```

"Describe(adsl)" code can be used to check the labels attached to the variables. Below is the snapshot of the dataset which display the labels with the variable name.

STUDYID Study Identifier	USUBJID Unique Subject Identifier	SUBJID Subject Identifier for the Study	SITEID Study Site Identifier	COUNTRY Country	INVNAM Investigator Name

Two key challenges faced when coding in R were (1) R does not provide a log like SAS, so code debugging is difficult in R compared to SAS. (2) We were able to attach label to the variables but were not able to attach label to the dataset.

Conclusion

In this paper we generated an ADaM compliant ADSL dataset using the R programming language. We demonstrate that R can be used as an effective alternative to create the clinical trial dataset. We provided a step by step process to set-up the R environment and the R code for reading the input dataset and processing the data to generate the ADSL dataset. We also compared the SAS and R code for the process, and discussed challenges encountered and addressed issues like attaching labels to variables.

Reference

1. Prasanna Murugesan, 2018, "Clinical Trial Datasets (CDISC - SDTM/ADaM) Using R", Phuse US Connect.
2. Monika M. Wahi, Peter Seebach, 2018, "Analyzing Health Data in R for SAS Users", Boca Raton, Florida, Taylor & Francis.
3. CDISC Analysis Data Model Team, 2016, "Analysis Data Model Implementation Guide Version 1.1".
4. Ali Dootson, 2020,TFL Programming in R versus SAS, d-wise, https://www.d-wise.com/blog/tfl-programming-in-r-versus-sas
5. Amol Waykar, Kevin Kramer, Kalyani Komarasetti, Andrew Miskell, 2020, Generating TFLs in R - Challenges and Successes compared to SAS, Phuse US Connect.

Acknowledgement

I would like to thank Nagadip Rao, Associate Director of Eliassen Group Life Science for reviewing this paper and providing valuable comments. I would also like to thank Lalitkumar Bansal of Statum Analytics for technical discussions and editorial inputs to this paper.

Contact Information

Vipin Kumpawat
Eliassen Group Life Science
Somerset New Jersey USA
vipin.kb09@gmail.com
vkumpawat@eliassen.com