

Why SASSY?

David J. Bosak, r-sassy.org

ABSTRACT

R has several weaknesses in regards to its use in the pharmaceutical industry. It lacks many of the basic capabilities of SAS®. It is also numerically inconsistent with SAS. R is furthermore conceptually very different from SAS, and quite difficult for a SAS programmer to learn and use. Lastly, the stability and backward compatibility enjoyed by SAS programmers generally does not exist in R. Most R package developers do not give priority to these values. The purpose of this paper is to explain how the SASSY system of R packages can solve all of these problems. The paper will first clarify the above problems. It will then give a quick overview of the SASSY ecosystem. Finally, the paper will elaborate on how SASSY addresses the above issues in a very simple and direct way. The end result is a small, stable set of R packages that replicates the basic functionality of SAS, without any of the weaknesses normally associated with R.

INTRODUCTION

As an analytical/statistical language, R offers several advantages. R is free and open source. It has a huge repository of contributed packages. It has excellent graphical capabilities. It has good development tools. It has cutting edge statistical methods. Best of all, it has a very energetic community of developers and users, who are willing to share their knowledge and skills with others.

Nevertheless, SAS programmers coming to R are often hit with a series of unpleasant surprises. It was these unpleasant surprises that motivated development of the **sassy** system. So before exploring the **sassy** system further, let's first understand some of these problems with R.

PROBLEMS WITH R

LACKING CAPABILITIES

SAS programmers coming to R are often shocked to see that the most basic capabilities of SAS do not exist natively in R. The following capabilities are all, to the surprise of many SAS programmers, missing from Base R:

- No log
- No formats
- No format catalog
- No libname
- No datastep
- No suitable reporting system

How is anyone supposed to work in R without these basic things? But wait! It gets much worse!

NUMERIC INCONSISTENCY

Next, SAS programmers are very surprised to see that the statistical output from R functions doesn't always match SAS. Here are just a few of the R statistical measures that do not match SAS by default:

- Quartiles
- Skewness
- Kurtosis
- Fisher's Odds Ratio

- Cochran-Mantel-Haenszel

Some of the above discrepancies can be fixed by setting the function parameters properly. Some (like the CMH) do not match no matter how you set the function parameters. These discrepancies greatly slow down development, and cause a lot of stress for the statisticians.

While researching such discrepancies, SAS programmers then realize a few more unpleasanties while working in R.

HARD TO USE/LEARN

SAS documentation is really great. It is very thorough, with extensive examples, sample code, sample data, equations, references, etc. Coming from the SAS world, programmers are surprised to discover the following about R:

- Documentation is not great
- Examples are generally terrible
- No tech support
- Too many functions
- Too many packages
- Built for specialists, not business users
- Mostly built for interactive use, not production programming

In SAS, when you are stuck, you can always call tech support, or refer to the excellent documentation. With R, it is a lot more trial and error, and the support is much less formal.

The next surprise comes after about three months. That is when you realize the R programs you wrote three months ago don't work anymore! What happened?

UNSTABLE

SAS Institute is 100% in charge of the SAS language. Because of this, the SAS Institute is able maintain backward compatibility and stability of the language over decades. SAS programs written in the 1980s will probably still work in the 2020s.

R, on the other hand, is supported by thousands of volunteer contributors. These contributors have different levels of commitment to backward compatibility. Some are very committed. Some not at all.

R therefore has many characteristics that most SAS programmers wouldn't expect:

- Low backward compatibility
- Breaking changes
- Deprecated functions
- Deprecated parameters
- Packages removed from CRAN
- Maintainers that stop supporting their package

Compared to SAS, the R world can seem very chaotic. It is out of this chaos that the idea for SASSY arose.

WHAT IS SASSY?

The **sassy** system is an integrated ecosystem of R packages inspired by ideas from SAS. The system allows you to write programs in R similar to how you would write them in SAS. The system is a direct response to all the problems with R discussed above.

FOCUSED ECOSYSTEM

The **sassy** system is a focused system, with a very small number of packages. As of this writing, the **sassy** system contains only seven packages:

- **logr**: A package to create a log that somewhat resembles a SAS log.
- **fmtr**: Has formatting capabilities that mimic PROC FORMAT. You can also create a FORMAT CATALOG.
- **libr**: Supports a LIBNAME function, a DATASTEP function, and a data DICTIONARY.
- **reporter**: A multi-format reporting system, similar to PROC REPORT and ODS.
- **procs**: A package that contains replicas of common SAS procedures like PROC FREQ, PROC MEANS, PROC REG, PROC TRANSPOSE, and PROC SORT.
- **common**: A set of general utilities used across the **sassy** system.
- **sassy**: The **sassy** package itself is a wrapper for all of the above packages. It allows you to install and load the whole set at once.

That's it! To replicate your SAS programs in R, you don't need dozens of packages and months of training. You just need SASSY!

Let's see how it works.

SASSY IN ACTION

Below is a small sample program. This program creates a simple table using the **sassy** packages. Notice the following about this example:

- The `libname()` function works just like a SAS LIBNAME.
- `proc_freq()`, `proc_means()`, `proc_transpose()`, and `proc_sort()` provide the basic capabilities of their SAS counterparts.
- The `value()` function operates like a VALUE statement in SAS PROC FORMAT. It allows you to create a user-defined format to perform lookup-style mapping.
- The `create_table()`, `create_report()`, and `write_report()` functions from the **reporter** package allow you to output reports in several file formats, including TXT, RTF, DOCX, HTML, and PDF.
- The `log_open()`, `log_close()`, and `put()` functions from the **logr** package make it easy to create a SAS-style log in R.
- Above all, notice that a **sassy** program *looks like a SAS program*. Any SAS programmer will be able to read and understand this program with *no prior training in R or sassy*. And if you can read it with no prior training, it will only take a short time before you can write it.

Here is the program:

```
library(sassy)

# Prepare Log -----

options("logr.autolog" = TRUE,
        "logr.on" = TRUE,
        "logr.notes" = FALSE,
        "procs.print" = FALSE)

# Get temp directory
tmp <- tempdir()

# Open log
lf <- log_open(file.path(tmp, "AGE_table.log"))
```

```

# Prepare formats -----

sep("Prepare formats")

put("Age categories")
agecat <- value(condition(x >= 18 & x <= 29, "18 to 29"),
                condition(x >=30 & x <= 39, "30 to 39"),
                condition(x >=40 & x <=49, "40 to 49"),
                condition(x >= 50, ">= 50"),
                as.factor = TRUE)

put("Compile format catalog")
fc <- fcat(MEAN = "%.1f", STD = "(%.2f)",
          Q1 = "%.1f", Q3 = "%.1f",
          MIN = "%d", MAX = "%d",
          CNT = "%2d", PCT = "(%.1f%%)",
          AGECAT = agecat)

# Load and Prepare Data -----

sep("Prepare Data")

put("Create sample ADSL data.")
adsl <- read.table(header = TRUE, text = '
SUBJID  ARM    SEX  RACE    AGE
"001"   "ARM A" "F"   "WHITE" 19
"002"   "ARM B" "F"   "WHITE" 21
"003"   "ARM C" "F"   "WHITE" 23
"004"   "ARM D" "F"   "BLACK" 28
"005"   "ARM A" "M"   "WHITE" 37
"006"   "ARM B" "M"   "WHITE" 34
"007"   "ARM C" "M"   "WHITE" 36
"008"   "ARM D" "M"   "WHITE" 30
"009"   "ARM A" "F"   "WHITE" 39
"010"   "ARM B" "F"   "WHITE" 31
"011"   "ARM C" "F"   "BLACK" 33
"012"   "ARM D" "F"   "WHITE" 38
"013"   "ARM A" "M"   "BLACK" 37
"014"   "ARM B" "M"   "WHITE" 34
"015"   "ARM C" "M"   "WHITE" 36
"016"   "ARM A" "M"   "WHITE" 40')

put("Categorize AGE")
adsl$AGECAT <- fapply(adsl$AGE, agecat)

put("Log starting dataset")
put(adsl)

put("Get ARM population counts")
proc_freq(adsl, tables = ARM,
          output = long,
          options = v(nopercent, nonobs)) -> arm_pop

# Age Summary Block -----

sep("Create summary statistics for age")

put("Call means procedure to get summary statistics for age")
proc_means(adsl, var = AGE,
          stats = v(n, mean, std, median, q1, q3, min, max),
          by = ARM,
          options = v(notype, nofreq)) -> age_stats

```

```

put("Combine stats")
datastep(age_stats,
  format = fc,
  drop = find.names(age_stats, start = 4),
  {
    `Mean (SD)` <- fapply2(MEAN, STD)
    Median <- MEDIAN
    `Q1 - Q3` <- fapply2(Q1, Q3, sep = " - ")
    `Min - Max` <- fapply2(MIN, MAX, sep = " - ")
  }) -> age_comb

put("Transpose ARMs into columns")
proc_transpose(age_comb,
  var = names(age_comb),
  copy = VAR, id = BY,
  name = LABEL) -> age_block

# Age Group Block -----

sep("Create frequency counts for Age Group")

put("Get age group frequency counts")
proc_freq(adsl,
  table = AGECAT,
  by = ARM,
  options = nonobs) -> ageg_freq

put("Combine counts and percents and assign age group factor for sorting")
datastep(ageg_freq,
  format = fc,
  keep = v(VAR, LABEL, BY, CNTPCT),
  {
    CNTPCT <- fapply2(CNT, PCT)
    LABEL <- CAT
  }) -> ageg_comb

put("Sort by age group factor")
proc_sort(ageg_comb, by = v(BY, LABEL)) -> ageg_sort

put("Tranpose age group block")
proc_transpose(ageg_sort,
  var = CNTPCT,
  copy = VAR,
  id = BY,
  by = LABEL,
  options = noname) -> ageg_block

put("Combine blocks into final data frame")
datastep(age_block,
  set = ageg_block,
  {}) -> final

# Report -----

sep("Create and print report")

var_fmt <- c("AGE" = "Age", "AGECAT" = "Age Group")

# Create Table
tbl <- create_table(final, first_row_blank = TRUE) |>
  column_defaults(from = `ARM A`, to = `ARM D`, align = "center", width = 1.1) |>
  stub(vars = c("VAR", "LABEL"), "Variable", width = 2.5) |>

```

```

define(VAR, blank_after = TRUE, dedupe = TRUE, label = "Variable",
       format = var_fmt, label_row = TRUE) |>
define(LABEL, indent = .25, label = "Demographic Category") |>
define(`ARM A`, label = "Placebo", n = arm_pop["ARM A"]) |>
define(`ARM B`, label = "Drug 50mg", n = arm_pop["ARM B"]) |>
define(`ARM C`, label = "Drug 100mg", n = arm_pop["ARM C"]) |>
define(`ARM D`, label = "Competitor", n = arm_pop["ARM D"]) |>
titles("Table 1.0", "Summary Analysis of AGE",
       "Safety Population", bold = TRUE) |>
footnotes("Program: AGE_Table.R",
          "NOTE: Denominator based on number of non-missing responses.")

rpt <- create_report(file.path(tmp, "AGE_Table.pdf"),
                    output_type = "PDF",
                    font = "Arial") |>
page_header("Sponsor: Company", "Study: ABC") |>
set_margins(top = 1, bottom = 1) |>
add_content(tbl) |>
page_footer("Date Produced: {Sys.Date()}", right = "Page [pg] of [tpg]")

put("Write out the report")
res <- write_report(rpt)

# Clean Up -----

sep("Clean Up")

put("Close log")
log_close()

# Display report
file.show(res$modified_path)

# Display log
file.show(lf)

```

Here is the output for the above program:

Sponsor: Company

Study: ABC

Table 1.0
Summary Analysis of AGE
Safety Population

Variable	Placebo (N=5)	Drug 50mg (N=4)	Drug 100mg (N=4)	Competitor (N=3)
Age				
N	5	4	4	3
Mean (SD)	34.4 (8.71)	30.0 (6.16)	32.0 (6.16)	32.0 (5.29)
Median	37.0	32.5	34.5	30.0
Q1 - Q3	37.0 - 39.0	26.0 - 34.0	28.0 - 36.0	28.0 - 38.0
Min - Max	19 - 40	21 - 34	23 - 36	28 - 38
Age Group				
18 to 29	1 (20.0%)	1 (25.0%)	1 (25.0%)	1 (33.3%)
30 to 39	3 (60.0%)	3 (75.0%)	3 (75.0%)	2 (66.7%)
40 to 49	1 (20.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
>= 50	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)

Program: AGE_Table.R

NOTE: Denominator based on number of non-missing responses.

Date Produced: 2025-03-23

Page 1 of 1

And here is the log for the above program:

```
=====
Log Path: C:/Users/dbosa/AppData/Local/Temp/RtmpU5CzJJ/log/AGE_Table.log
Program Path: C:/packages/Testing/PharmaSUG/sassy.R
Working Directory: C:/packages/Testing
User Name: dbosa
R Version: 4.4.2 (2024-10-31 ucrt)
Machine: SOCRATES x86-64
Operating System: Windows 10 x64 build 26100
Base Packages: stats graphics grDevices utils datasets methods base
Other Packages: tidylog_1.1.0 procs_1.0.7 reporter_1.4.4 libr_1.3.4 logr_1.3.8
fmtr_1.6.5 common_1.1.3 sassy_1.2.5
Log Start Time: 2025-03-23 10:42:47.348675
=====
```

```
=====
Prepare formats
=====
```

Age categories

```
# A user-defined format: 4 conditions
- as.factor: TRUE
  Name Type      Expression      Label Order
1  obj     U x >= 18 & x <= 29 18 to 29    NA
2  obj     U x >= 30 & x <= 39 30 to 39    NA
3  obj     U x >= 40 & x <= 49 40 to 49    NA
4  obj     U              x >= 50  >= 50    NA
```

Compile format catalog

```
# A format catalog: 9 formats
- $MEAN: type S, "%.1f"
- $STD: type S, "(%.2f)"
- $Q1: type S, "%.1f"
- $Q3: type S, "%.1f"
- $MIN: type S, "%d"
- $MAX: type S, "%d"
- $CNT: type S, "%2d"
- $PCT: type S, "(%5.1f%%)"
- $AGECAT: type U, 4 conditions
```

```
=====
Prepare Data
=====
```

Create sample ADSL data.

Categorize AGE

Log starting dataset

	SUBJID	ARM	SEX	RACE	AGE	AGECAT
1	1	ARM A	F	WHITE	19	18 to 29
2	2	ARM B	F	WHITE	21	18 to 29
3	3	ARM C	F	WHITE	23	18 to 29
4	4	ARM D	F	BLACK	28	18 to 29
5	5	ARM A	M	WHITE	37	30 to 39
6	6	ARM B	M	WHITE	34	30 to 39
7	7	ARM C	M	WHITE	36	30 to 39
8	8	ARM D	M	WHITE	30	30 to 39
9	9	ARM A	F	WHITE	39	30 to 39

```

10      10 ARM B   F WHITE  31 30 to 39
11      11 ARM C   F BLACK  33 30 to 39
12      12 ARM D   F WHITE  38 30 to 39
13      13 ARM A   M BLACK  37 30 to 39
14      14 ARM B   M WHITE  34 30 to 39
15      15 ARM C   M WHITE  36 30 to 39
16      16 ARM A   M WHITE  40 40 to 49

```

Get ARM population counts

```

proc_freq: input data set 16 rows and 6 columns
           tables: ARM
           output: long
           view: TRUE
           output: 1 datasets

```

```

VAR STAT ARM A ARM B ARM C ARM D
1 ARM CNT      5      4      4      3

```

```

=====
Create summary statistics for age
=====

```

Call means procedure to get summary statistics for age

```

proc_means: input data set 16 rows and 6 columns
            by: ARM
            var: AGE
            stats: n mean std median q1 q3 min max
            view: TRUE
            output: 1 datasets

```

```

      BY VAR N MEAN      STD MEDIAN Q1 Q3 MIN MAX
1 ARM A AGE 5 34.4 8.706320  37.0 37 39  19  40
2 ARM B AGE 4 30.0 6.164414  32.5 26 34  21  34
3 ARM C AGE 4 32.0 6.164414  34.5 28 36  23  36
4 ARM D AGE 3 32.0 5.291503  30.0 28 38  28  38

```

Combine stats

datastep: columns decreased from 10 to 7

```

      BY VAR N Mean (SD) Median Q1 - Q3
1 ARM A AGE 5 34.4 (8.71)  37.0 37.0 - 39.0
2 ARM B AGE 4 30.0 (6.16)  32.5 26.0 - 34.0
3 ARM C AGE 4 32.0 (6.16)  34.5 28.0 - 36.0
4 ARM D AGE 3 32.0 (5.29)  30.0 28.0 - 38.0
Min - Max
1  19 - 40
2  21 - 34
3  23 - 36
4  28 - 38

```

Transpose ARMs into columns

```

proc_transpose: input data set 4 rows and 7 columns
                var: BY VAR N Mean (SD) Median Q1 - Q3 Min - Max
                id: BY
                copy: VAR
                name: LABEL
                output dataset 5 rows and 6 columns

```

```

VAR      LABEL      ARM A      ARM B      ARM C

```

```

1 AGE          N          5          4          4
2 AGE Mean (SD) 34.4 (8.71) 30.0 (6.16) 32.0 (6.16)
3 AGE      Median      37.0      32.5      34.5
4 AGE      Q1 - Q3    37.0 - 39.0 26.0 - 34.0 28.0 - 36.0
5 AGE Min - Max      19 - 40      21 - 34      23 - 36
      ARM D
1          3
2 32.0 (5.29)
3          30.0
4 28.0 - 38.0
5          28 - 38

```

```

=====
Create frequency counts for Age Group
=====

```

Get age group frequency counts

```

proc_freq: input data set 16 rows and 6 columns
           tables: AGECAT
           by: ARM
           view: TRUE
           output: 1 datasets

```

	BY	VAR	CAT	CNT	PCT
1	ARM A	AGECAT 18 to 29	1	20.00000	
2	ARM A	AGECAT 30 to 39	3	60.00000	
3	ARM A	AGECAT 40 to 49	1	20.00000	
4	ARM A	AGECAT >= 50	0	0.00000	
5	ARM B	AGECAT 18 to 29	1	25.00000	
6	ARM B	AGECAT 30 to 39	3	75.00000	
7	ARM B	AGECAT 40 to 49	0	0.00000	
8	ARM B	AGECAT >= 50	0	0.00000	
9	ARM C	AGECAT 18 to 29	1	25.00000	
10	ARM C	AGECAT 30 to 39	3	75.00000	
11	ARM C	AGECAT 40 to 49	0	0.00000	
12	ARM C	AGECAT >= 50	0	0.00000	
13	ARM D	AGECAT 18 to 29	1	33.33333	
14	ARM D	AGECAT 30 to 39	2	66.66667	
15	ARM D	AGECAT 40 to 49	0	0.00000	
16	ARM D	AGECAT >= 50	0	0.00000	

Combine counts and percents and assign age group factor for sorting

datastep: columns decreased from 5 to 4

	VAR	LABEL	BY	CNTPCT
1	AGECAT 18 to 29	ARM A	1	(20.0%)
2	AGECAT 30 to 39	ARM A	3	(60.0%)
3	AGECAT 40 to 49	ARM A	1	(20.0%)
4	AGECAT >= 50	ARM A	0	(0.0%)
5	AGECAT 18 to 29	ARM B	1	(25.0%)
6	AGECAT 30 to 39	ARM B	3	(75.0%)
7	AGECAT 40 to 49	ARM B	0	(0.0%)
8	AGECAT >= 50	ARM B	0	(0.0%)
9	AGECAT 18 to 29	ARM C	1	(25.0%)
10	AGECAT 30 to 39	ARM C	3	(75.0%)
11	AGECAT 40 to 49	ARM C	0	(0.0%)
12	AGECAT >= 50	ARM C	0	(0.0%)
13	AGECAT 18 to 29	ARM D	1	(33.3%)
14	AGECAT 30 to 39	ARM D	2	(66.7%)
15	AGECAT 40 to 49	ARM D	0	(0.0%)
16	AGECAT >= 50	ARM D	0	(0.0%)

Sort by age group factor

```
proc_sort: input data set 16 rows and 4 columns
           by: BY LABEL
           keep: VAR LABEL BY CNTPCT
           order: a a
           output data set 16 rows and 4 columns
```

	VAR	LABEL	BY	CNTPCT
1	AGECAT	18 to 29	ARM A	1 (20.0%)
2	AGECAT	30 to 39	ARM A	3 (60.0%)
3	AGECAT	40 to 49	ARM A	1 (20.0%)
4	AGECAT	>= 50	ARM A	0 (0.0%)
5	AGECAT	18 to 29	ARM B	1 (25.0%)
6	AGECAT	30 to 39	ARM B	3 (75.0%)
7	AGECAT	40 to 49	ARM B	0 (0.0%)
8	AGECAT	>= 50	ARM B	0 (0.0%)
9	AGECAT	18 to 29	ARM C	1 (25.0%)
10	AGECAT	30 to 39	ARM C	3 (75.0%)
11	AGECAT	40 to 49	ARM C	0 (0.0%)
12	AGECAT	>= 50	ARM C	0 (0.0%)
13	AGECAT	18 to 29	ARM D	1 (33.3%)
14	AGECAT	30 to 39	ARM D	2 (66.7%)
15	AGECAT	40 to 49	ARM D	0 (0.0%)
16	AGECAT	>= 50	ARM D	0 (0.0%)

Transpose age group block

```
proc_transpose: input data set 16 rows and 4 columns
                by: LABEL
                var: CNTPCT
                id: BY
                copy: VAR
                name: NAME
                output dataset 4 rows and 6 columns
```

	VAR	LABEL	ARM A	ARM B	ARM C
1	AGECAT	18 to 29	1 (20.0%)	1 (25.0%)	1 (25.0%)
2	AGECAT	30 to 39	3 (60.0%)	3 (75.0%)	3 (75.0%)
3	AGECAT	40 to 49	1 (20.0%)	0 (0.0%)	0 (0.0%)
4	AGECAT	>= 50	0 (0.0%)	0 (0.0%)	0 (0.0%)
ARM D					
1			1 (33.3%)		
2			2 (66.7%)		
3			0 (0.0%)		
4			0 (0.0%)		

Combine blocks into final data frame

```
datastep: columns started with 6 and ended with 6
```

	VAR	LABEL	ARM A	ARM B
1	AGE	N	5	4
2	AGE	Mean (SD)	34.4 (8.71)	30.0 (6.16)
3	AGE	Median	37.0	32.5
4	AGE	Q1 - Q3	37.0 - 39.0	26.0 - 34.0
5	AGE	Min - Max	19 - 40	21 - 34
6	AGECAT	18 to 29	1 (20.0%)	1 (25.0%)
7	AGECAT	30 to 39	3 (60.0%)	3 (75.0%)
8	AGECAT	40 to 49	1 (20.0%)	0 (0.0%)
9	AGECAT	>= 50	0 (0.0%)	0 (0.0%)
ARM C ARM D				

```

1          4          3
2 32.0 (6.16) 32.0 (5.29)
3          34.5          30.0
4 28.0 - 36.0 28.0 - 38.0
5      23 - 36      28 - 38
6 1 ( 25.0%) 1 ( 33.3%)
7 3 ( 75.0%) 2 ( 66.7%)
8 0 (  0.0%) 0 (  0.0%)
9 0 (  0.0%) 0 (  0.0%)

```

```

=====
Create and print report
=====

```

```

Write out the report

```

```

# A report specification: 1 pages
- file_path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpU5CzJJ\AGE_Table.pdf'
- output_type: PDF
- units: inches
- orientation: landscape
- margins: top 1 bottom 1 left 1 right 1
- line size/count: 9/41
- page_header: left=Sponsor: Company right=Study: ABC
- page_footer: left=Date Produced: 2025-03-23 center= right=Page [pg] of [tpg]
- content:
# A table specification:
- data: data.frame 'final' 9 rows 6 cols
- show_cols: all
- use_attributes: all
- title 1: 'Table 1.0'
- title 2: 'Summary Analysis of AGE'
- title 3: 'Safety Population'
- footnote 1: 'Program: AGE_Table.R'
- footnote 2: 'NOTE: Denominator based on number of non-missing responses.'
- stub: VAR LABEL 'Variable' width=2.5 align='left'
- define: VAR 'Variable' dedupe='TRUE'
- define: LABEL 'Demographic Category'
- define: ARM A 'Placebo'
- define: ARM B 'Drug 50mg'
- define: ARM C 'Drug 100mg'
- define: ARM D 'Competitor'

```

```

=====
Clean Up
=====

```

```

Close log

```

```

=====
Log End Time: 2025-03-23 10:42:47.716228
Log Elapsed Time: 0 00:00:00
=====

```

WHY SASSY?

Reviewing the above code, output, and log, it should be immediately apparent *why* the **sassy** system was developed; this set of packages give you capabilities like no other. The system retains all the strengths of R, and fixes all the weaknesses. Let's review:

SAS-LIKE CAPABILITIES

The **sassy** system adds all the basic features of SAS that are missing from R: procedures, reporting, logging, formatting, libnames, etc. These missing features can be a major impediment to doing pharmaceutical programming in R.

The **sassy** system covers the full suite of activities normally performed in pharmaceutical research:

- Dataset creation
- Tables
- Listing
- Figures
- Multiple output types

And it is all there in a single library!

For information on the **sassy** system as a whole, see the web site at <https://r-sassy.org/>. Don't miss the report gallery at <https://r-sassy.org/gallery.html#gallery>. This page has many more examples of **sassy** in action.

NUMERIC CONSISTENCY

Getting R to match SAS can be a painful activity. Researching these discrepancies can burn many hours, and create a significant barrier to R adoption.

With **sassy**, however, all the research and validation has been done for you. For common statistics, numeric consistency with SAS comes out of the box. Furthermore, the **sassy** statistical procedures are familiar and easy to use.

For more information on **sassy** statistical procedures, see the **procs** package documentation here: <https://procs.r-sassy.org/index.html>.

For questions regarding the validity of **sassy** statistics, see the **procs** validation document here: https://r-sassy.org/validation/Procs_Validation.pdf.

EASY TO USE AND LEARN

To a SAS programmer, it often feels that everything in R is backwards. R operates column by column instead of row by row. SAS has a few large procedures, while R has hundreds of small functions. These fundamental differences can be very disorienting to SAS programmers.

The **sassy** system brings R much closer to SAS. The functions are instantly recognizable and easy for a SAS programmer to understand. **sassy** functions leverage the mental patterns of SAS programmers, and allow them to transfer directly to R.

To get started with logging, see the **logr** web site: <https://logr.r-sassy.org/index.html>.

To get started with libnames and datasteps, see the **libr** web site: <https://libr.r-sassy.org/index.html>.

To get started with formatting, see the **fmtr** web site: <https://fmtr.r-sassy.org/index.html>.

To get started with reporting, see the reporter web site: <https://reporter.r-sassy.org/index.html>.

STABLE

SAS is famous for its stability and backward compatibility. R is famous for its instability and breaking changes. The **sassy** system aims to bring SAS-like stability to R.

A stable base language is a basic necessity for production pharmaceutical work. We expect our programs to run properly for the length of the study and several years afterward. It is totally unacceptable to have programs breaking every few months.

The developers of the **sassy** system recognized this problem right from the outset. They are committed to the philosophy of no breaking changes. This commitment to stability is a distinguishing feature of the **sassy** system.

ACCELERATES R ADOPTION

Aligning the conceptual model in SAS and R has many advantages in terms of accelerating R adoption. With the **sassy** system:

- The code is easy to read and learn
- R training can take days or weeks instead of months and years
- Fewer R packages make R repository management and validation much simpler
- Existing code can be easily translated from SAS to R
- The time spent chasing statistical discrepancies is significantly reduced
- The packages won't change/break every few months

All these factors together mean that R adoption can get going very quickly with the **sassy** system.

CONCLUSION

The **sassy** system is a set of packages designed to fix many of the problems associated with R. The system intelligently blends the best qualities of R with the best qualities of SAS. The result is a free, Open-Source language that also has powerful capabilities, ease of use, and validated numeric consistency with SAS. The system maps the existing mental model of SAS programmers onto R, and allows them to write R programs in the same way they write SAS programs. It provides all the most basic structures that SAS programmers use every day: formats, a datastep, and powerful statistical procedures. The system makes it easy to generate a SAS-style log. It gives reporting capabilities that far exceed anything previously available in R. All of this is provided in a small, stable set of R packages that can be installed and loaded in two lines of code. It is for all these reasons that **sassy** is the ideal solution to the problems with using R in production pharmaceutical programming.

REFERENCES

Bosak, David J. June 2021. "The SASSY System: Making R Easier for SAS Programmers." *Proceedings of the PharmaSUG Conference 2021*: AD-175. Available at <https://pharmasug.org/proceedings/2021/AD/PharmaSUG-2021-AD-175.pdf>.

Bosak, David J. "Introduction to SASSY", r-sassy.org. June 2024. Available at <https://sassy.r-sassy.org/index.html>.

ACKNOWLEDGMENTS

The author would like to acknowledge several contributors whose ideas and feedback were instrumental to the development of the **sassy** system: Brian Varney, Kevin Kramer, Duong Tran, and Kevin Putschko.

CONTACT INFORMATION

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

David J. Bosak
Archytas Clinical Solutions
dbosak01@gmail.com
www.r-sassy.org

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