PharmaSUG 2023 - Paper AP-151

Real Projects, Real Transition, Really Revolutionary – Transitioning to R for Biometrics Work

Danielle Stephenson, Rebekah Oster, and Alyssa Wittle, Atorus Research

ABSTRACT

Technology is developing quickly and staying on the cutting edge has its challenges. Often, diving into something new is an intimidating idea for programmers, companies, and sponsors – especially when the day-to-day work must continue. The pharmaceutical industry has been entrenched in SAS® for decades, and the time has come to explore open-source and dedicate the time to figure this out. What are the ups and downs of this transition? How can the learning curve become a bit less curvy? Can TFLs and CDISC-compliant datasets be created using R? We will dive into what it looks like for a team to go from SAS® -fluent to multi-lingual in real time, with real projects, and ways to ensure quality while making the transition.

INTRODUCTION

"SAS® vs R" has been the topic of a decade-long debate. When R was first introduced, it was seen as a tool for academics and amateurs. Even the cutting-edge models and statistical graphics developed in R were initially only used as a complement to SAS® analysis. As R has gained popularity, there has been more and more investment into the development of packages specifically designed to support clinical data programming. The discussion then turned to how SAS and R can effectively work together. Recently, the industry is accepting that R can replace SAS® in many cases, from SDTM to ADaM and TFL generation.

BASIC R CONCEPTS

Just as every programming language has its own look and feel, R is no different. We will focus first on a comparison toward traditional SAS® programming. SAS® is full of useful procedures from something as simplistic as a PROC SORT to something as advanced as PROC NLMIXED and everything in between. These are done a bit differently within R. Out of the box functions are contained in "base" R and extensions to the language are contained in packages. A "package" is a collection of R functions with a singular purpose. They must be installed and loaded into the R workspace before they can be used in a program.

```
# Upload packages
base::library(envsetup)
envsetup::library(dplyr)
envsetup::library(haven)
envsetup::library(metacore)
envsetup::library(metatools)
envsetup::library(diffdf)
envsetup::library(xportr)
envsetup::library(tibble)
```

Figure 1: Loading packages into the R workspace

This can loosely be compared to a SAS® macro library in the way that it can contain several functionalities which are produced from a series of specific coding statements and options. Packages can range in purposes from simple string manipulation to large scale data visualization. In SAS®, procedures are created by the software company SAS® and macros are created by users. In R, packages and their functions are created by users and published for open-source use. This is a strength because it puts users on the forefront of innovation. It is also a weakness because packages may not be validated. However, packages are constantly being enhanced, corrected, or adjusted. Due to this, version release notes should be reviewed in detail for impact.

Programmers who are familiar with a Linux or Unix system will find the R programming approach similar in basic format. The structure in R uses commands and functions to edit the output in small increments. SAS® does not operate in a line by line or option by option functionality but rather requires full sets of

data steps or procedures to run. Python can also be considered a similar language in that they are both object-oriented, interpreted, and functional languages. While Python is general purpose, R is designed for statistical analysis.

The most basic unit in R is a vector rather than a single variable. While a vector can hold a single value, it is more commonly an object such as a SAS® array and can even be used like a multi-dimensional array. Vectors in R make recoding and reformatting very simple. This combination of formatting and nested array functionality can greatly simplify data manipulation, especially in creating SDTM datasets where large quantities of raw data variables must be transformed in mostly predictable ways.

THINGS TO CONSIDER BEFORE THE TRANSITION

THE CASE FOR IMMERSION

As a language, R approaches and conceptualizes data very differently than SAS® does. As programmers learn the new language and become fluent, the process is similar to real-life language learning in that immersion is the fastest and most efficient learning approach. Requiring programmers to switch back and forth between R and SAS® during the early days of the transition can result in fluency coming more slowly and can be very frustrating for the programmers. The transition from SAS® to R will likely go more smoothly if programmers are moved in dedicated groups which are able to focus solely on R for a significant period.

R IS POWERFUL AND FLEXIBLE

The fact that R conceptualizes data differently than SAS®, however, is also a great advantage of the language. SAS® programmers who have enjoyed the use of PROC SQL within their SAS® programmers may have some insight into how useful a non-linear processing language can be in rapidly manipulating large quantities of data. While SAS® excels in line-by-line processing logic, R (like SQL) is capable of processing and changing entire datasets and columns in single commands. This ability lends itself to more compact programs with fewer steps required to reach the same results.

IN THE THICK OF IT

RELATE SAS TO R, AND THEN MOVE BEYOND

When making the transition from SAS® to R, it is very helpful to relate new R concepts to similar SAS® concepts so that programmers can understand quickly how to utilize R functions. For example, a vector can be used with the “case_match” function to recode a variable the way that many programmers use SAS® formats. However, when we take this approach, it is important that we do not make the mistake of thinking that these concepts function the same way behind the scenes. This will limit our ability to advance in R and our flexibility in the use of R commands. In this example, SAS® formats define how a variable is printed, so it takes an extra step to actually change the raw data values. On the other hand, in R, “case_match” acts like a vectorized recode function and has much greater functionality than a simple reformat does: it can also accept vectors and single values as input and has arguments that determine how to handle unmatched values.

As we approach fluency in R, our thinking as programmers will need to transition from a SAS® line-by-line or row-based process to the R column-based process. While this can be difficult to get used to, it also allows for more flexible, compact programming when done correctly.

For example, the “by” statement is used in SAS® to process dataset observations in groups. In R, a commonly used equivalent is the “group_by” function from the dplyr package. Moving beyond this simple equivalent opens the door to a whole new world of grouped data processing. Unlike the SAS® data step which is blind to all but the current row, grouped data processing in R can see the whole group.

Figure 2 shows a simple example of grouped data processing, where a total score is derived from four test scores. First, the data frame adft is grouped by key variables using the “group_by” function. Then the “filter” function removes all parameters except the four test scores. Finally, the “summarize” function creates a new data frame that has one row for each combination of grouping variables. This is where the

magic happens! AVAL is derived using if-else logic. The function “n” gives the current group size. Thus, if the current group contains less than four records, AVAL will be NA. Else, AVAL is a sum of the four test scores.

```
total_score <- adft %>%
  dplyr::group_by(STUDYID, USUBJID, AVISIT, AVISITN, ADT) %>% # Create grouped data frame
  dplyr::filter(PARAMCD %in% c("FT0101", "FT0102", "FT0103", "FT0104")) %>% # Keep four test scores
  dplyr::summarize(AVAL = ifelse(dplyr::n()<4,as.numeric(NA),sum(AVAL)), .groups="drop") # Derive total score
```

Figure 2: Example of grouped data processing

Performing this task in SAS® without the benefit of grouped data processing would require an intermediate dataset and at least two steps. This example illustrates how powerful R can be when programmers are given the time and support needed to move beyond simply recreating SAS® code in R.

EMBRACE THE TIDYVERSE

SAS® programmers tend to find comfort in code that is contained between “data” and “run;” or “proc” and “quit;”. In contrast, R is boundless and tends to feel wide open, difficult to nail down and organize. The best tool for this problem is the tidyverse, a collection of packages often described as a dialect of the R language. Tidyverse packages were designed with a shared vision, grammar, and data structures. Within the tidyverse, data frames are created and transformed in defined blocks, just like in SAS®. Each line is connected to the next line with the pipe operator, “%>%”, which pipes a value or data frame forward.

```
vs1 <- vs_raw %>%
    tidyr::pivot_longer(
      c(
        HR_,
        SYSBP_,
        DIABP_,
        RESP_,
        O2SAT_,
        SUPPLEMENTAL_O2_,
        TEMP_,
        WEIGHT_,
        HEIGHT_,
        ULNA_
      ),
      names_to = "TEST_RAW",
      values_to = "RESULT_N"
    ) %>%
  left_join(dm, by = "USUBJID") %>%
  mutate(
    RESULT = case_when(
      !is.na(RESULT_N) ~ as.character(RESULT_N),
      TRUE ~ NA_character_
    )
  ) %>%
  mutate(
    STUDYID = "ABC-101",
    DOMAIN = "VS",
    VSTEST = recode(TEST_RAW, !!!vs_fmt$raw_to_test)
  )
```

Figure 3: Tidyverse VS data transformation

In Figure 3, the data frame vs1 is assigned the value of the entire block of code following the assignment operator “<-“. First the data frame vs_raw is set, followed by the pipe operator %>%, which takes the data frame forward. The “pivot_longer” function transposes the raw data from wide to long. The “left_join” function merges in the demographics data by USUBJID. The “mutate” function is a handy way to create new variables. All these steps are performed sequentially to the same data frame, without intermediates. The pipe operator takes what precedes it and offers it as input for the next line. It can be read as “and then”.

For comparison, the SAS® code in Figure 4 performs the same transformation using a PROC TRANSPOSE and a DATA STEP.

```
proc transpose data=vs_raw out=vs1 (rename=(col1=result_n _name_=test_raw));
     var HR_ SYSBP_ DIABP_ RESP_ O2SAT_ SUPPLEMENTAL_O2_ TEMP_ WEIGHT_ HEIGHT_ ULNA_;
     id usubjid;
run;

data vs1;
     merge vs1 (in=a) dm;
     by usubjid;
     if a;

     if result_n = . then result = put(result_n, best.);
     else result = .;

     studyid = 'ABC-101';
     domain = 'VS';
     vstest = format(test_raw, $raw_to_test.);
run;
```

Figure 4: SAS VS data transformation

Beginner R programmers who stay within the tidyverse will feel more secure and confident in their code.

UTILIZE AVAILABLE TOOLS

In addition to packages and functions, R also has tools available which can be loaded into the environment and utilized to meet specific needs. For example, R does not automatically generate a log file like SAS® does. However, the tool “logrx”, once installed in the RStudio IDE, can be used to automatically generate a log file. Figure 5 shows the user-friendly interface of the logrx tool.

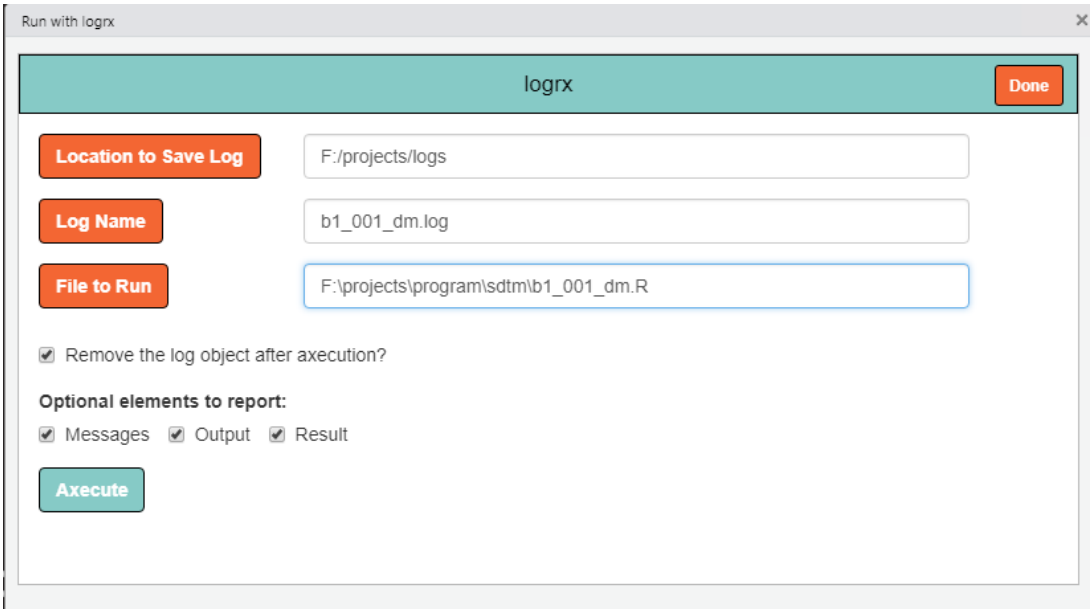


Figure 5: Logrx interface

UNDER THE HOOD

R has capabilities far beyond simple statistical data processing and can be used to create system tools to simplify tasks such as batch runs and compares. As an open-source tool, we are not limited when using R to only the functions which we have created. While user-defined functions and tools may be used often, the real power of R lies in functions from packages such as tidyverse or admiral which offer manipulation of data in pre-defined ways and can cut many steps out of the programming process.

R functions fill a similar place within the R world that SAS® procedures and macros do. An awareness of the differences “under the hood” is extremely important when using open-source packages for statistical analysis, which may have different assumptions and defaults than similar SAS® procedures or macros do. It is important to read the documentation and understand the underlying model assumptions to make sure that we are using what we need for the analysis.

DIFFERENCES IN RESULTS

As the transition progresses, you will likely find some differences in the logic behind similar functions in SAS® and R. One well-documented example of simple differences between the languages in general is that while SAS® rounds numbers ending in “5” up to the next number (2.5 becomes 3), R rounds numbers ending in “5” to the nearest even number (2.5 becomes 2).

A lesser-known difference between the two languages lies in the way that they store numbers in the background. All computing software stores numbers in binary. In many cases, this does not cause a problem and binary and decimal system numbers can be converted back and forth without a problem. However, certain numbers are irrational in binary even though they are rational in the decimal system. An example from fraction and decimal systems can be used to illustrate this idea: 1/3 is an exact number as a fraction, and you can add 1/3 to itself three times and get exactly 1. However, if you convert that fraction to a decimal, you get an infinitely repeating number (.33333…..). You must decide where to truncate this number, and if you add .3333… to itself three times, you will not get exactly 1 – the conversion between fraction and decimal systems causes a discrepancy at the end of your math. Similarly, when a number which is rational in the decimal system and irrational in the binary system is entered into your programming language, the language must decide where to truncate and how to store that number.

Different operating systems will do this slightly differently in order to maximize the range or precision of the numbers which can be stored, so even when you run SAS® on two different computers, when you are running large amounts of precise information you may get minor discrepancies. In addition, SAS® and R also use slightly different assumptions to store these irrational numbers. Most of the time, this does not cause issues, but every once in a while, the minor discrepancies add up and you end up not being able to perfectly match outcomes after “translating” numbers back and forth between decimal numbers, SAS®, and R, and rounding at the end.

Fortunately, it is possible to minimize the effect of these rounding and float point differences by choosing particular rounding functions which account for these known differences in binary number storage. The tplyr package, for example, utilizes a function that minimizes these discrepancies and matches SAS® nearly all of the time.

Another minor difference is that while SAS® sorts null values first in a dataset, R sorts “NA” values last. A simple solution for this problem is to replace all “NA” values with blanks prior to sorting. Blank values are different from “NA” in R, and they are sorted first like in SAS®. Figure demonstrates an easy way to accomplish this with the “across” function which can apply the same transformation to multiple columns. The tidyr “replace_na” function is applied to all character columns in the data frame.

```
dplyr::mutate(across(where(is.character), ~ tidyr::replace_na(., "")))
```

Figure 6: Replace NA values with blanks

All of these small differences can add up even if all larger assumptions within a package match the corresponding SAS® function, which is why it is important to understand exactly what is happening behind the scenes in order to minimize or explain the differences in outcomes.

CDISC COMPLIANCE

SDTM PACKAGES

While it is possible to program SDTMs using base R, packages are being developed to help streamline this process in a way that creates defaults in a CDISC compliant way. the dplyr package, for example, contains functions allowing the programmer to transform individual variables in predictable and familiar ways such as if-then statements or reformatting. It also allows us to filter data, join datasets in ways reminiscent of PROC SQL, and control which columns are output into a data set. The metatools package allows us to check that we have all the required and no extra variables, to check variable values against predetermined codelists and controlled terminology, and to sort datasets by predefined keys.

```
qcdm <- dm1 %>%
  metatools::drop_unspec_vars(dm_spec) %>% #Keep needed variables
  metatools::set_variable_labels(dm_spec) %>% #Set variable labels
  metatools::order_cols(dm_spec) %>% #Reorder variables per spec
  metatools::sort_by_key(dm_spec) %>% #Sort the datasets by key variables
  metatools::check_variables(dm_spec) %>% #Check if all variables present
  metatools::check_ct_col(dm_spec, DOMAIN) %>% #Check controlled Terminology
  metatools::check_ct_col(dm_spec, DTHFL) %>%
  metatools::check_ct_col(dm_spec, AGEU) %>%
  metatools::check_ct_col(dm_spec, SEX) %>%
  metatools::check_ct_col(dm_spec, RACE) %>%
  metatools::check_ct_col(dm_spec, ETHNIC) %>%
  metatools::check_ct_col(dm_spec, ARMCD) %>%
  metatools::check_ct_col(dm_spec, ARM) %>%
```

Figure 7: Metatools package functions used to clean and check a DM dataset

Even the more advanced admiral package has some great uses in SDTM, such as calculation of baseline flags or the currently recommended –LOBXFL.

```
restrict_derivation(
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = vars(USUBJID, FTTESTCD, FTLOC, FTLAT),
    order = vars(VISITNUM),
    new_var = FTLOBXFL,
    mode = "last"
  ),
```

Figure 8: Deriving --LOBXFL using an admiral function

ADAM PACKAGES

The admiral package has much wider uses in ADaM to derive nearly every variable. The metatools package allows manipulation of metadata to be read in and assists in setting labels, keeping specific variables, and sorting variables as needed for each dataset. The xportr package then applies formats and dataset labels. Reading through the full documentation of admiral is important to see its full and wide range of options to simplify many common, complex analysis methods.

```
adpe <- sdtm$pe %>%
  admiral::derive_vars_merged_lookup(dataset_add = param_fmt,
                                     by_vars = vars(PETESTCD),
                                     new_vars = vars(PARAMCD, PARAM, PARAMN),
                                     print_not_mapped = FALSE) %>%
  admiral::derive_vars_dt(new_vars_prefix = "A", dtc = PEDTC) %>% #ADT
  admiral::derive_vars_merged(dataset_add = adams$adsl, new_vars = vars(TRTSDT),
                              by_vars = vars(STUDYID, USUBJID)) %>% #Add TRTSDT
  admiral::derive_vars_dy(reference_date = TRTSDT, source_vars = vars(ADT)) %>% #ADY
  metatools::drop_unspec_vars(adpe_spec) %>% # Keep needed variables
  metatools::set_variable_labels(adpe_spec) %>% # Set variable labels
  metatools::order_cols(adpe_spec) %>% # Reorder variables per spec
  metatools::sort_by_key(adpe_spec) %>% # Sort the datasets by key variables
  metatools::check_variables(adpe_spec) %>% # Check if all variables present
  xportr::xportr_format(adpe_spec, domain = domain) %>% # Apply spec formats
  xportr::xportr_df_label(adpe_spec, domain = domain) # Set dataset label
```

Figure 9: Creating ADaM with admiral, metatools, and xportr

DOCUMENTATION

As we work within CDISC standards, one of our necessary goals must be clear documentation of our metadata and processes. Our programming specifications should evolve to contain the required information for R programming, which may in many cases be different than that needed for SAS programming because of unique defaults within packages and procedures. In addition, the use of packages in R that are loaded into the environment adds a level of needed documentation. We should consider including package information in the ADRG and the cSDRG. In the ADRG, Section 7 (Submission of Programs) would be an appropriate place to include this information, while the cSDRG could include this within domain-specific sections or in an appendix.

ENSURING QUALITY DURING TRANSITION

With these warnings in place, how can we proceed into this transition without compromising the quality of work along the way? In our transition, we have found several practices helpful to ensure that we are confident in the quality and consistency of our work as we move between SAS® and R.

First, we have had great results **practicing independent programming** using SAS® on one side and R on the other. When we do production side in SAS® and QC in R, or vice versa, we are able to clearly see and consider any differences in results between the two languages and decide our approach to resolving these.

Along similar lines, it can be useful to **match existing** SAS® **QC'd results** using R to have predefined stepping stones. If a dataset or table has already been created and QC'd in SAS®, then as the programmer works to do a second QC in R, there is less chance that differences are caused by mistakes on the production SAS® side and the programmer can focus on figuring out methods to match the results in R.

As the transition happens, details required in R that were assumed or not required in SAS® may become apparent. **Determine what of these extra details may need to be added in SAP, specs, reviewers guides, etc.,** to ensure that as programming moves toward R on both sides of the process there are no details missed or assumptions made differently when SAS® no longer is being used as a double check. Defaults in SAS® no longer apply, and R is a much more versatile tool for customization which, while helpful, increases the need for excellent and detailed documentation in the desired assumptions for a project.

Finally, as is often encouraged in SAS® but so often ignored since it is the common language to all in our industry, programmers should be encouraged to **add detailed comments very liberally to their R**

programs. It can be easy to become focused on solving the problem or finishing the dataset in R, but during the learning process especially, thorough commenting can save having to figure out the solution to the same problem from scratch a second time. This is especially important while team members are learning and transitioning through programs so that their logic can not only be documented for use by themselves and others, but also improved upon through review from a more advanced programmer.

FORWARD THINKING APPLICATIONS

Using R can transform our industry into embracing the more technology based reviews being used today. Gone are the days of printing out all of the data for an entire study to review on paper, we have technology at our fingertips, so why should we settle for static tables and figures? RShiny is a tool used to create interactive tables and figures which allow the user to "drill down" the data points. Points could be selected in a data visualization/plot which automatically generates a table and/or listing of only those selected points. A user can also highlight certain groups in a plot by hovering (over a single line in a line plot, for example) to view subject-level values and/or summary statistics. Tables, figures and listings can be sorted, filtered, and more in a click rather than needing to program individual outputs for each situation. In a static environment, a user would have to generate one table or plot per group to accomplish the same high-level data exploration. While static outputs are not completely outdated since they certainly have a use in publication, clinical study reports, etc., the advantages of this level of data investigation for an experimental therapy are immense. Consider the below snapshots of some of these capabilities from RShiny as inspiration for becoming multi-lingual. Don't miss the chance to be on the cutting edge of modern data review and analysis!

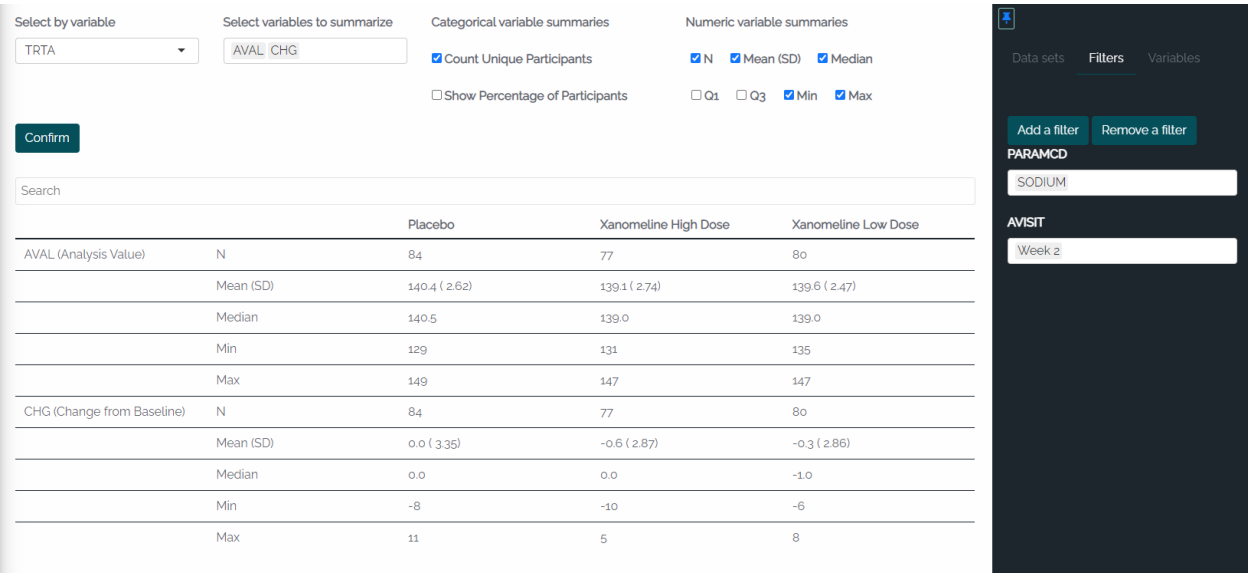
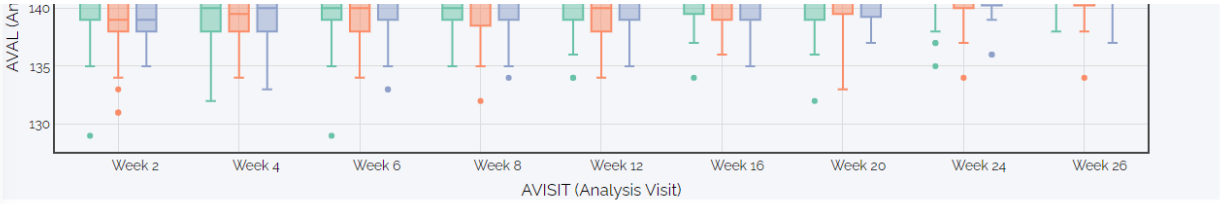


Figure 50: Observed and Change from Baseline summary table for sodium at the week 2 visit (exemplifies the filtering capabilities to see various summaries; TRTA as grouping variable and summarizes AVAL & CHG)

Figure 61: Box plot of observed sodium (AVAL) by AVISIT and TRTA (excluded baseline and unscheduled visits using filters; green boxes on the plot shows summary stats of the particular box being hovered over)



Search

USUBJID	AVISIT	AVAL	TRTA
01-701-1015	Week 2	142	Placebo
01-701-1015	Week 4	140	Placebo
01-701-1015	Week 6	140	Placebo
01-701-1015	Week 8	141	Placebo
01-701-1015	Week 12	139	Placebo
01-701-1015	Week 16	141	Placebo
01-701-1015	Week 20	142	Placebo
01-701-1015	Week 24	140	Placebo

Figure 12: Listing of the data used in the plot (Figure 9) is presented underneath

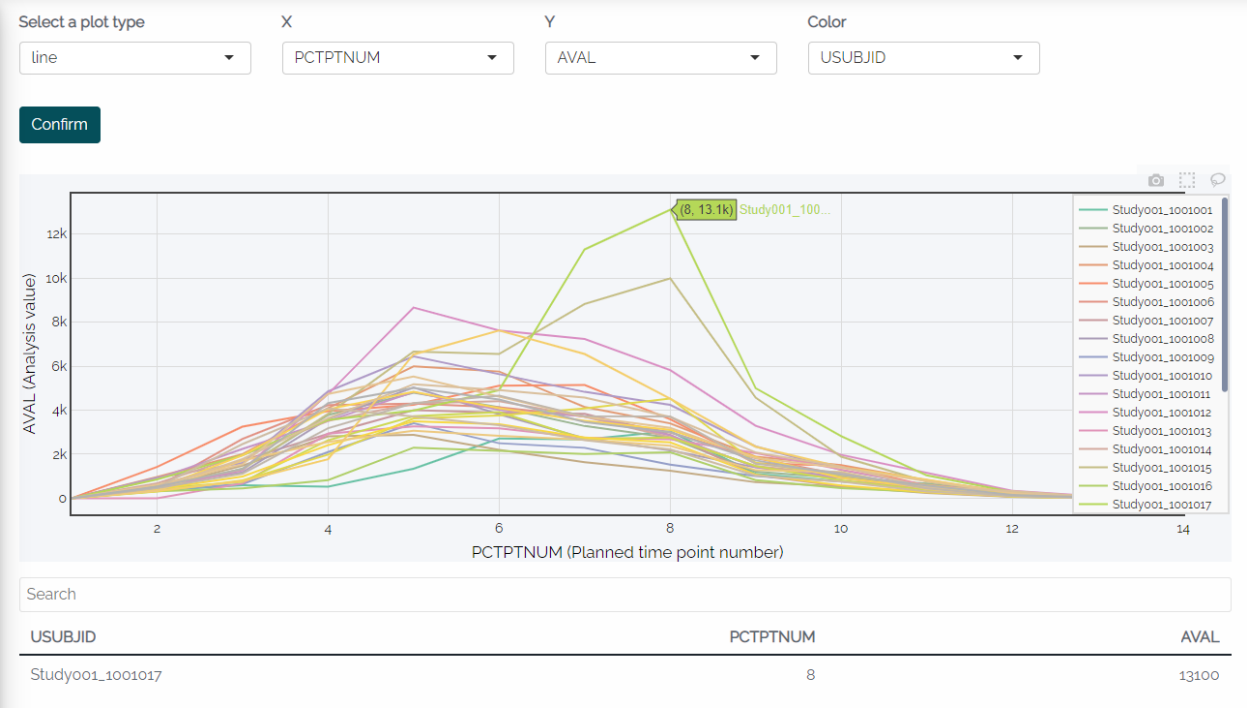


Figure 73: Clicking on the highest point of a spaghetti plot to investigate why Cmax is so much higher than the rest of the subjects, the listing below is filtered just for that point/subject

CONCLUSION

Not everyone is a language aficionado. In fact, many programmers in our industry did not come into it with a computer science degree and have never had to learn multiple programming languages. For that reason, while exciting and adventurous to some, others can find the mere suggestion of an industry diving into a new programming language to be daunting at best. However, each of us does not need to re-invent the wheel in this endeavor. Resources abound to help SAS® programmers explore the world of R. By taking the advice and instruction of others who have already tread this path and asking lots of questions along the way, you may find that R is a more powerful, customizable, and user-friendly tool than SAS® has proved to be. Keep an open mind and keep exploring R capabilities and you will be multi-lingual before you know it.

ACKNOWLEDGMENTS

Most of our SAS programmers learned R in large part from Atorus Academy which is tailored to help SAS programmers transition to R. Instruction covered is widespread: from a base language to specially designed packages, lessons on programming SDTM, ADaM, tables, figures, and more frequently being added.

CONTACT INFORMATION

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

Danielle Stephenson
Atorus Research Inc.
Danielle.Stephenson@AtorusResearch.com

Rebekah Oster
Atorus Research Inc.
Rebekah.Oster@AtorusResearch.com

Alyssa Wittle
Atorus Research Inc.
Alyssa.Wittle@AtorusResearch.com