Paper DV06

Constructing Multilingual Visualization Toolset for Clinical Study Reports and Submission

Bingjun Wang, Jane Liao, Yulia Sidi, Nan Xiao
Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

With the increasing demand for insightful data visualization deliverables for clinical study reports and submissions, robust and powerful data visualization tools are needed. In this paper, we illustrate development and implementation of a multilingual visualization toolset. The system includes a graphic repository with visualization standards, a visualization gallery, internally developed R packages, and SAS macros. Visualization standards define the figure layout, recommendations, and standard themes to promote operational efficiency and consistency across various clinical trials within the organization. The visualization gallery provides code in both SAS and R for commonly used figures. Additionally, R packages and SAS macros are developed based on our organization standards and branding for frequently requested figures. This toolkit enables efficient and consistent generation of figures that enhance visual storytelling in clinical study reports and provide improved data comprehension.

INTRODUCTION

There is an increasing demand for insightful data visualization and a growing trend of using multiple programming languages in clinical trial analysis & reporting (A&R). In our organization, we have clearly defined processes and standards for tables, listings, and figures (TLFs) so that our deliverables across clinical trials have a consistent format and theme. Our primary focus remains on upholding compliance by ensuring the consistency, quality, and reproducibility of A&R deliverables. With the growing trend of using multiple programming languages in clinical trial development, our organization explored different software languages, in addition to SAS, such as R and Python to improve our A&R capabilities. In this paper, we will focus on the development of data visualization standards and the implementation of a standard toolset using SAS and R in our organization.

FRAMEWORK OF THE VISUALIZATION TOOLSET

The framework of the visualization toolset is shown below in Figure 1. Standard Operating Procedures (SOPs) help ensure compliance, traceability, and reproducibility of A&R deliverables. The manual of TLF formatting standards sets formatting expectations of TLF deliverables. Visualization standards were developed based on the manual of TLF formatting standards to define the figure layout and recommendations on themes to promote operational efficiency and consistency in A&R across clinical trials within the organization. This sets the foundation for development of standard programs. Based on our experience, it is not easy to develop a standard program that meets all the stakeholder needs for figure generation. For frequently requested figures, standard programs (SAS macros and R packages) are developed. Customizable programs include template programs that were developed to provide flexibility beyond what is offered by the standard programs. The visualization gallery showcases various types of figures that the standard programs support along with example programs. Training materials were developed internally to help the study programmers use standard programs to generate figures for a given study. Details about all the components of this framework will be provided in the following sections.

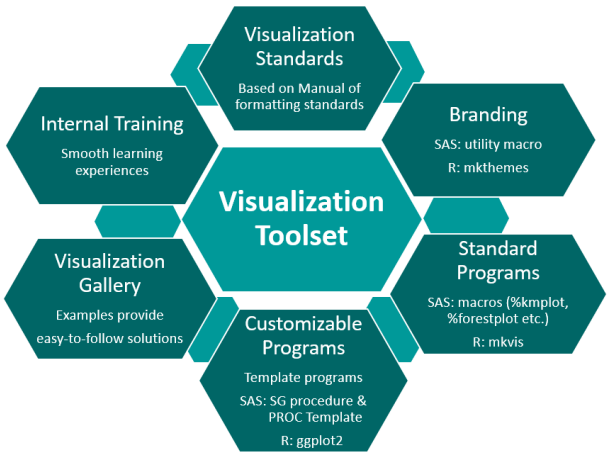

Figure 1. Diagram of Visualization Toolset within Our Organization

VISUALIZATION STANDARDS AND BRANDING

Lack of uniform standards and branding results in a noticeable inconsistency across study deliverables. The figures below show outputs without a standard theme (Figure 2) vs. the ones with a standard theme applied (Figure 3). The

implementation of a standardized theme is crucial for maintaining the branding of our deliverables, thereby enhancing their recognizability and consistency in a professional context.

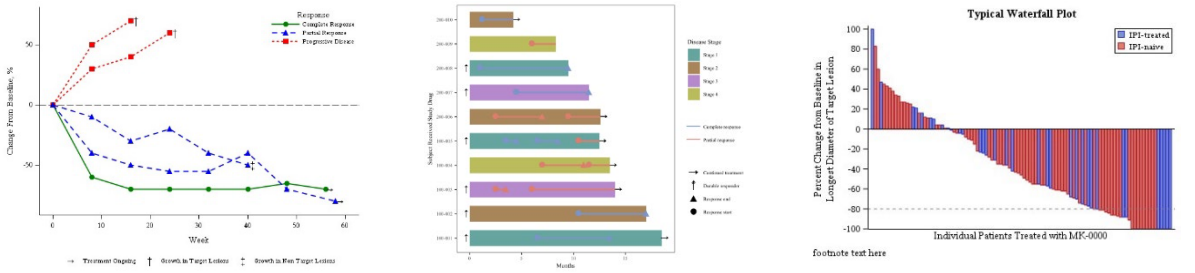

Figure 2. Example Graphic Outputs without a Standard Theme

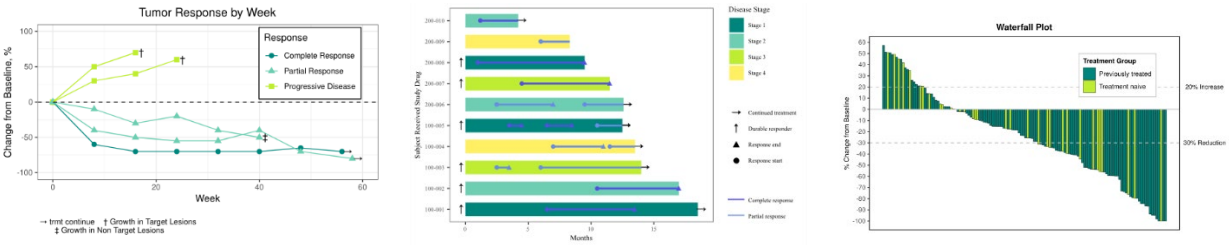

Figure 3. Example Graphic Outputs with a Standard Theme

The manual of TLF formatting standards sets the foundation for visualization standards. Visualization standards also provides the best practices when developing programs/mockups to support analysis and reporting deliverables. A standard SAS macro %graph0param0init and an internally developed R package mkthemes are designed to enforce standardized formatting of the output with consistent font type, font size, figure elements (symbol, line pattern, color used if need), etc. With the use of %graph0param0init for SAS, mkthemes for R, branded deliverables from our organization are easily recognized both internally and externally. In addition, having consistency in figures across clinical trials will help make internal and external review more efficient.

STANDARD VISUALIZATION TOOLSET

We have developed a standard toolset to generate figures using SAS and R. This multilingual toolset is developed based on internally defined visualization standards and branding discussed in the previous paragraph.

SAS GRAPHIC REPOSITORY

In our organization, we have a mature SAS graphics repository. Figure 4 shows the structure of this repository. Most of the plots used in clinical trial A&R can be generated with Statistical Graphics (SG) procedures and using the Graph Template Language (GTL) for customization as needed. The SG procedures and GTL programs are simple and straightforward. The code tends to be easy to follow and can be easily adapted to address specific A&R requirements. Commonly used figures are evaluated and standardized by a cross functional team involving statistical programmers, statisticians, clinical team members and medical writers. Standard macros are developed wherever possible for use across clinical trials. Template programs are made available to provide flexibility when a standard macro doesn't meet all the stakeholder needs. Template programs provide a better starting point to the users and help to shorten their learning curve for SAS graphics programming in scenarios when standard macros are not sufficient.

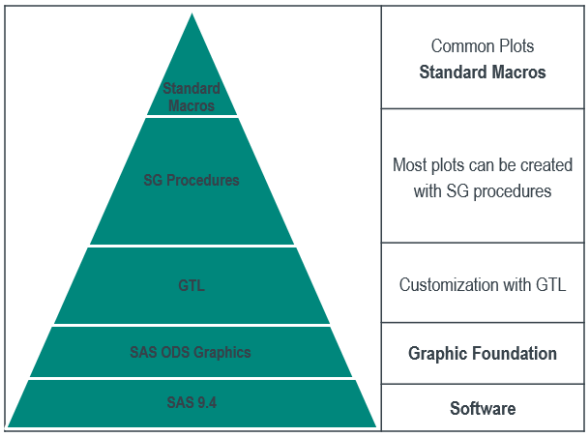

Figure 4. SAS Graphic Repository Structure

STANDARD PROGRAMS

Standard macros are developed for common A&R figures such as Kaplan-Meier (KM) plot, rainfall plot, forest plot, and volcano plot. Standard macros may be complicated and have several pages of user documentation, especially for a generic macro that addresses all scenarios. It can be challenging for a user to debug or review code when there is an error or if additional customization is needed.

Example macro call code (only listing key macro parameters) for our global standard graphic macro %kmplot is shown below. The output of this call program is shown in Figure 5. The options for the macro may not fit all customization needed for a specific request. In such scenarios, corresponding template programs are made available for study programmers to adopt and make changes as needed. The template code is often open SAS code with less than 100 lines of code.

```
%kmplot(
 input_dataset     = ADTTE
,time_var          = AVAL
,censor_var        = CNSR
,group_var         = trtp
,plot              = surv
,censor_indicator  = 1
,x_origin_offset   = 0
,legend_location   = TOPRIGHT
,display_stats     = Y|1 2 3 4 5
,display_tab1      = NBRISK
,display_tab2      = NBLEFT
);
```

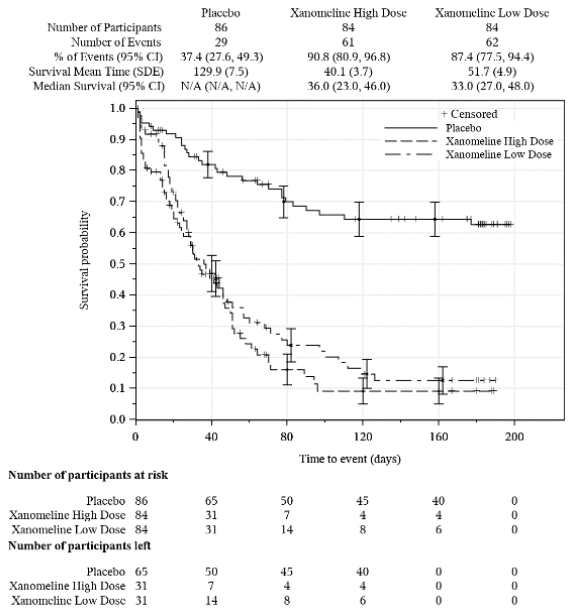

Figure 5. KM Plot Example from SAS Standard Macro

As you can see, simple macro call can generate a comprehensive KM plot based on internal standards.

CUSTOMIZABLE PROGRAMS

Template programs with GTL and SG procedure are available to the users, and they are easy-to-follow. These offers a better starting point for the users to adopt when a standard macro doesn't meet the stakeholder needs. The SG procedure is simple and provides a straight-forward way to generate figures, but it may not be flexible enough to generate the required figure. Below we show two different approaches to create a spider plot (Figure 6). One is using SGPLOT and another is using PROC TEMPLATE.

```
*** Attribute map data sets are used in the procedures to associate data values with
visual attributes;
data _attrmap_;
input ID $5. VALUE $7-34 LINEPATTER $36-44 LINECOLOR $46-50 MARKERSYMBOL $52-65
MARKERCOLOR $66-71;
datalines;
rspid Complete Response            solid     black circlefilled   black
```

```
rspid Partial Response             dash      black trianglefilled black
rspid Progressive Disease          shortdash black squarefilled   black
sttid Treatment Ongoing                              rightallow    black
sttid Growth in Target Lesions                       dagger        black
sttid Growth in Non Target Lesions                   ddagger       black
run;

*** SG procedure to generate the plot;
proc sgplot data= data_for_plot dattrmap=_attrmap;
   symbolchar name=rightallow char='2192'x / scale=1;
   symbolchar name=dagger     char='2020'x / scale=1;
   symbolchar name=ddagger    char='2021'x / scale=1;
   refline 0 / lineattrs=(pattern=shortdash);
   series x=week   y=change / group=subject grouplp=response grouplc=response
                              groupms=response groupmc=response
                              markers markerattrs=(size=10) lineattrs=(thickness=2)
                              lpattrid=rspid lcattrid=rspid msattrid=rspid
                              mcattrid=rspid name='a';
   scatter x=maxwk y=change / group=status markerattrs=(size=20) nomissinggroup
                              attrid=sttid name='b';
   keylegend 'a' / title='Response' type=linecolor valueattrs=(size=8) location=inside
                   position=topright across=1;
   keylegend 'b' / valueattrs=(size=8) noborder;
   xaxis label='Week';
run;
```

GTL is more comprehensive and can be used to generate figures that require more complex customization. It provides a lot of flexibility to meet stakeholder needs. For e.g. The plot in Figure 5 has multiple components within a plot - the KM survival curve, statistical summary statistics, and two counts tables.

There are two steps when using GTL.

- Step 1: Define a figure template with PROC TEMPLATE
- Step 2: Render with PROC SGRENDER to generate the figure.

The below code snippet generates a spider plot using GTL. It provides ability to the users to define every detail on each line, symbol, and legend in the plot. A user can easily identify which statement is associated with different components of the figure. It is easy to make changes as needed.

```
proc template;
define statgraph spiderplot;
begingraph / collation=binary;

*** define customized symbols;
symbolChar char='2021'x name=ddagger    / scale=2;
symbolChar char='2020'x name=dagger     / scale=2;
symbolChar char='2192'x name=rightallow / scale=2;
DiscreteAttrVar attrvar=status var=status attrmap="STTID_ATTRMAP";
DiscreteAttrMap name="STTID_ATTRMAP" /;
   Value "Growth in Non Target Lesions" / markerattrs=(color=black symbol=dagger);
   Value "Growth in Target Lesions"     / markerattrs=(color=black symbol=ddagger);
   Value "Treatment Ongoing"            / markerattrs=(color=black symbol=rightallow);
EndDiscreteAttrMap;

*** legenditem controls the item to display at legend;
legenditem type=marker name='s1' / label="Growth in Non Target Lesions"
                                   markerattrs=(color=black symbol=dagger);
legenditem type=marker name='s2' / label="Growth in Target Lesions"
                                   markerattrs=(color=black symbol=ddagger);
legenditem type=marker name='s3' / label="Treatment Ongoing"
                                   markerattrs=(color=black symbol=rightallow);
```

```
*** response attribute: CX688CE8(blue) CXF68D2E(orange) CX799A0E(green);
DiscreteAttrVar attrvar=grpid var=series_grp attrmap="grpid_map";
DiscreteAttrMap name="grpid_map" /;
    Value "Complete Response"   / markerattrs=(symbol=circlefilled)
                                  lineattrs=(pattern=solid);
    Value "Partial Response"    / markerattrs=(symbol=trianglefilled)
                                  lineattrs=(pattern=dash);
    Value "Progressive Disease" / markerattrs=(symbol=squarefilled)
                                  lineattrs=(pattern=shortdash);
EndDiscreteAttrMap;
*** legenditem controls the item to display at legend;
legenditem type=markerline name='c' / label="Complete Response"
                                      markerattrs=(symbol=circlefilled)
                                      lineattrs=(pattern=solid thickness=2);
legenditem type=markerline name='p' / label="Partial Response"
                                      markerattrs=(symbol=trianglefilled)
                                      lineattrs=(pattern=dash thickness=2);
legenditem type=markerline name='d' / label="Progressive Disease"
                                      markerattrs=(symbol=squarefilled)
                                      lineattrs=(pattern=shortdash thickness=2);

layout overlay / xaxisopts=(label="Week" LABELATTRS=(Style=Normal Weight=Bold)
                            type=linear)
                 yaxisopts=(label="Change From Baseline, %" labelFitPolicy=Split
                            LABELATTRS=(Style=Normal Weight=Bold));
    ***Display change from baseline;
    seriesplot x=time y=Change / group=series_id display=(markers)
                                 markerattrs=(size=9) lineattrs=(thickness=2)
                                 LineColorGroup=grpid LinePatternGroup=grpid
                                 MarkerColorGroup=grpid MarkerSymbolGroup=grpid;
    ***Display subject status;
    scatterplot x=maxtm y=Change / subpixel=off includemissinggroup=false
                                   primary=true group=status;
    ***Display reference line for 0 change from baseline;
    referenceLine y=0 / clip=true lineattrs=(pattern=4 color=black thickness=1);
    layout gridded / AUTOALIGN=(topright) border=false;
           ***Legend for response group;
           discretelegend 'c' 'p' 'd' / location=inside across=1 valign=top
                                        halign=left title="Response"
                                        titleattrs=(Style=Normal) border=no
                                        outerpad=(left=10pct);
           ***Legend for status;
           discretelegend 's3' 's2' 's1' / location=inside across=1 valign=top
                                           halign=left title="Status"
                                           titleattrs=(Style=Normal) border=no
                                           outerpad=(left=10pct top=4pct);
    endlayout;
endlayout;
endgraph;
end;
run;

*** Render the plot using pre-defined template;
proc sgrender data=data_for_plot template=spiderplot;
run;
```

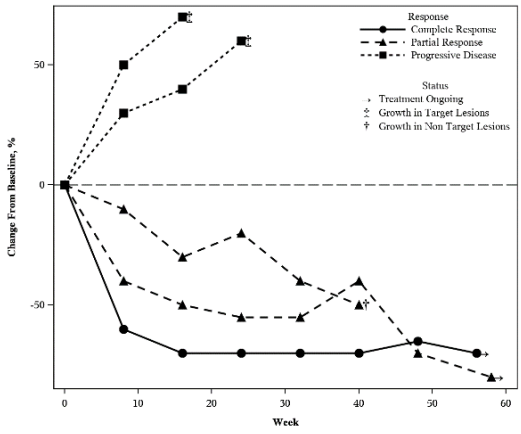

Figure 6. Spider Plot

VISUALIZATION GALLERY

Various figures used in different functional areas are compiled in the visualization gallery. Corresponding sample programs used to generate these figures with sample data and example outputs are stored on our computing platform. This is a place to share various innovative approaches to customize plots using SAS ODS Graphics. These programs are not complicated for others to adopt for their project-specific deliverable needs, i.e., study team can copy sample code to the project area as a starting point for figure development.

INTERNAL TRAINING

Internally developed graphic training sessions are offered to help statistical programmers and statisticians gain the unique skillset of SAS graphic programming. The sessions are described below.

Session 1: Basics and gallery – This session covers the basics of ODS Graphics, such as single-cell plots, multiple-cell plots, general plot enhancement options, and tips. This session also provides an introduction to the plots in the gallery and explains how to adopt the program for further customization through examples.

Session 2: Standard Macros – This session covers how to use the standard macros for creating common plots using parameters to provide enhanced flexibility.

Session 3: GTL and ODS Graphic Designer (OGD) – This session covers how to write GTL templates and covers an introduction to GTL, including GTL process steps and abstract framework, layout, and plot statements. It also covers GTL supporting tools such as SG procedure TMPLOUT option, automatic graph template, and the use of the OGD user interface to write GTL without any coding.

BENEFITS OF SAS GRAPHIC REPOSITORY

The SAS graphic repository in our organization is a comprehensive hub for qualified graphic programs, offering a streamlined approach to the creation and customization of plots. This graphic repository is instrumental in ensuring standardized deliverables are generated in an efficient manner across various projects within different therapeutic areas. It boasts of an extensive collection of template SAS programs, standard macros, and tailored solutions, thereby guaranteeing consistency, operational efficiency and high-quality in graphical deliverables. This repository not only simplifies the accessibility of a diverse array of programming tools but also promotes uniformity in data visualization.

R GRAPHIC REPOSITORY

With the success of the SAS Graphic Repository, we follow a similar design idea to build a graphics repository using R. Experienced R programmers have developed internal R packages to support data visualization and a visualization gallery. This gallery includes example programs showing how to use internal R packages along with external R packages such as ggplot2 to generate figures that are commonly used in clinical trial A&R. Internally developed training helps upskill users with R specific graphic knowledge and provides support to explore the possibilities of using R to generate figures. Advanced R programming expertise is required to develop and maintain R graphic repository the structure of which is provided in Figure 7.

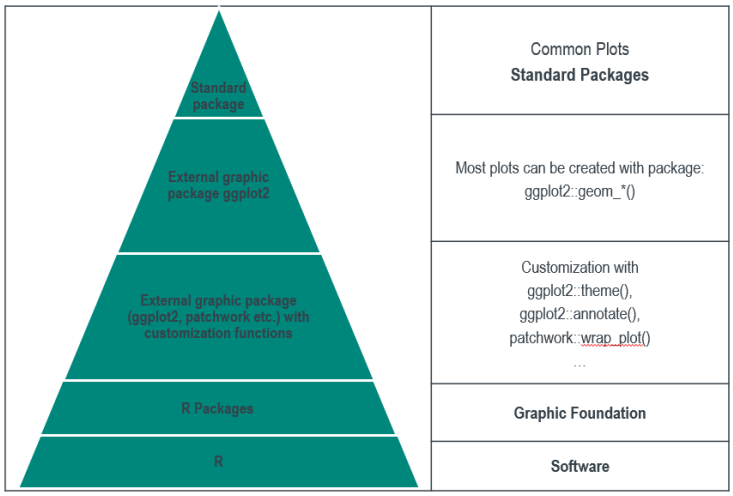

Figure 7. R Graphic Repository Structure

STANDARD PROGRAMS

The internally developed R package "mkvis" is used to generate commonly used static figures such as KM plot, rainfall plot, forest plot, etc. Figures created with mkvis share the same input data structure and output style. The input data for plotting are expected to be summarized and ready to be displayed visually so that functions in mkvis are focused on visualization rather than data manipulations. Output from these functions is a ggplot2 object that can use `+` operator to incrementally add layers or components to the output.

The general workflow of mkvis is as follow:
Step 1: Data manipulation – this includes reading and preparing data for plotting, i.e., filter, arrange, and factorize data as needed. Functions are developed as needed to tidy up data ready to be passed into plotting functions described in step 2.
Step 2: Generate a plot using one of the multiple plotting functions.

Function Name	Description
plot_box()	To create box plot
plot_dot()	To create dot plot (scatter plot)
plot_errorbar()	To display difference with horizontal high-low bar plot
plot_km()	To create KM plot
plot_line()	To create line plot (ex: for summary of change)
plot_volcano()	To create volcano plot
table_panel()	To create a table to display data value (number of event, risk difference etc.)

Step 3: Further plot customization using the following functions.

Function Name	Description
arrange_panel()	To arrange plot with multiple panels, e.g., rainfall plot and forest plot
arrange_facet()	To arrange plots in multiple rows or columns within a page, e.g., summary of change plot (line plot) and box plot;
annotate_volcano_bottom()	To insert annotations on bottom of volcano plot
annotate_volcano_top()	To insert annotations on top of volcano plot
theme_panel()	To define theme (axis ticks, text annotation for axis, plot margins etc.) for multiple panel plots.
theme_facet()	To define theme (axis ticks, text annotation for axis, plot margins etc.) for plots with multiple facets.

We also leveraged the capability of R to generate interactive figures. We have developed standard packages forestly and boxly to create interactive forest plot and box plot. See more details in the paper [1].

VISUALIZATION GALLERY

A visualization gallery provides figure mockups with template programs using both internal and external R packages to fulfill stakeholder needs. The gallery showcases an easy-to-follow solution with R to generate commonly used figures. The R packages ggplot2 and mkvis are mainly used for this purpose. The concept of standardization in this context does not imply a rigid, one-size-fits-all approach. Instead, it emphasizes the creation of standardized outputs that are accepted across various functional areas allowing some flexibility. For instance, a standardized KM plot, as included in the mkvis package, is designed to offer some flexibility for customization.

The template code in visualization gallery could always help the user to quickly generate commonly used figures. For example, here is the template code of a KM plot shown in Figure 8.

The following code snippet assign data values of legend title, label for x-axis and y-axis, and upper limit for x-axis tick value to be displayed in the plot.

```
legend_title <- "||| Censored"
x_label <- "Event or censored time"
y_label <- "Survival Probability (%)"
x_upper_limit <- ceiling(max(kmdata0os$TIME))
```

The code snippet below creates the KM plot ggplot2 object - kmplot1. It first to creates the initial KM plot which includes survival curve and censor points. It then applies further customization and an additional layer using the '+' operator.

```
kmplot1 <-
  ggplot2::ggplot(data =  dplyr::filter(kmdata0os, !is.na(SURVP)),
                  ggplot2::aes(group = trt01p, colour = trt01p)) +
  ggplot2::geom_step(ggplot2::aes(x = TIME, y = SURVP,
                                  linetype = trt01p,
                                  colour = trt01p),
                     direction = "hv",
                     linewidth = 1,
                     show.legend = TRUE) +
  ggplot2::geom_point(data = subset(kmdata0os, CENSOR >= 1),
                      ggplot2::aes(x = TIME, y = SURVP),
                      shape = 3,
                      size = 1,
                      show.legend = FALSE) +
```

Following code snippet allows to further customize the theme for the KM plot created with above statements, i.e., to show tick marks and label, define the margins, legend position, etc.

```
  mkvis::theme_panel(
    show_ticks = TRUE,
    show_text = TRUE,
    plot_margin = ggplot2::margin(t = 1, r = 0, b = 0, l = 0, unit = "lines")) +
  ggplot2::theme(
    legend.position = c(0.9, 0.8),
    legend.title.align = 0.5,
    panel.grid = ggplot2::element_line(color = "grey85",
                                       linewidth = 0.5,
                                       linetype = 1),
    axis.title.x = ggplot2::element_text(colour = "black"),
    axis.title.y = ggplot2::element_text(colour = "black")) +
```

We can add statements to work on the details of the legend, x-axis, and y-axis, such as the title, color manual of the legend, the label of x-axis and y-axis, tick mark values as shown below.

```
# Define the legend for the plot
  ggplot2::guides(
    group = ggplot2::guide_legend(title = legend_title),
    colour = ggplot2::guide_legend(title = legend_title),
    linetype = ggplot2::guide_legend(title = legend_title)) +
  ggplot2::scale_color_manual(values = c("#00857C", "#66203A", "#F68D2E")) +
  ggplot2::xlab(tools::toTitleCase(x_label)) +
  ggplot2::ylab(tools::toTitleCase(y_label)) +
  ggplot2::coord_cartesian(clip = "off") +

  # Add minor ticks using annotation_ticks()
  mkvis::annotation_ticks("lb", ticks_per_base = c(2, 2)) +

  # Control x-axis and y-axis breaks
```

```
ggplot2::scale_x_continuous(breaks = seq(0, x_upper_limit, 5),
                            limits = c(0, x_upper_limit),
                            expand = c(0.05, 0)) +
ggplot2::scale_y_continuous(breaks = seq(0, 100, 10),
                            limits = c(0, 100),
                            expand = c(0, 0)) +
```

New figure layer can be added with '+' operator, the programs below are adding error bar of confidence interval on the KM plot.

```
ggplot2::geom_errorbar(
  data = kmdata0os |> subset(TIME %in% seq(0, x_upper_limit, 5)),
  ggplot2::aes(x = TIME,
               y = SURVP,
               group = trt01p,
               colour = trt01p,
               ymin = LCI_SURVP,
               ymax = UCI_SURVP),
  width = 0.6,
  position = ggplot2::position_dodge(0.6),
  show.legend = FALSE)
```

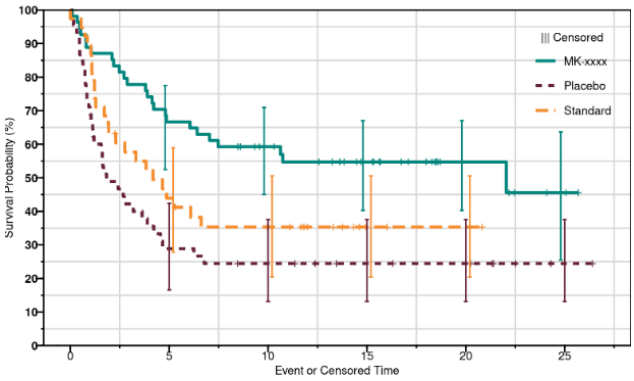

Figure 8. KM Plot Example Generated by Above Code

Our graphic repository primarily focuses on visualization rather than data analysis or data manipulation. Therefore, the input data is assumed to be summarized and ready to be displayed visually. To assist programmers in our organization to use R for their deliverables, we have included examples in our visualization gallery that demonstrate the process of transforming ADaM data for visualization purposes, as well as data processing techniques for mkvis.

INTERNAL TRAINING

The statistical programming staff in our organization have diverse skillsets and experience with varying levels of expertise in generating figures in SAS and R. Appropriate training is very important for programmers. There are several external trainings available (with or without cost), especially for R. Besides the general-purpose R training within our organization [2], we also developed our trainings to specifically suit the needs of visualization work. There are four main sessions:

Session 1: Basics and tools: Introduces the essential tools when building a graph using R: ggplot2, patchwork, and esquisse. This session focuses on external graphic R packages, introducing basic R graphic programming language.

Session 2: Standards: Sets the standards and recommendations for visualization deliverables and provides the best practices when developing programs/mockups to support A&R deliverables.

Session 3: Gallery: Provides easy-to-follow solutions using R to create common figures. Examples show workflows using both internal and external visualization packages.

Session 4: mkvis: A training for use of the internal graphic R package mkvis.

BENEFITS OF R GRAPHIC REPOSITORY

Over several years, our organization has dedicated considerable effort to developing standard programs for generating commonly used plots in R. The following examples (Figure 9 and Figure 10) showcase the output of these programs.

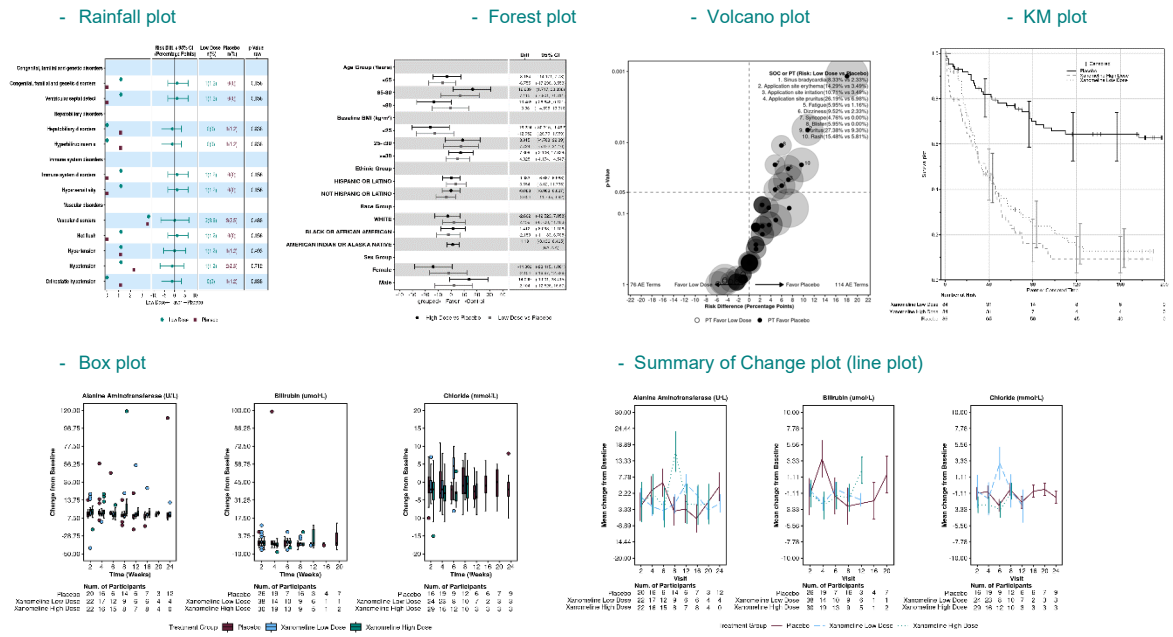

Figure 9. Examples of Figures Generated with Internally Developed R Package: mkvis

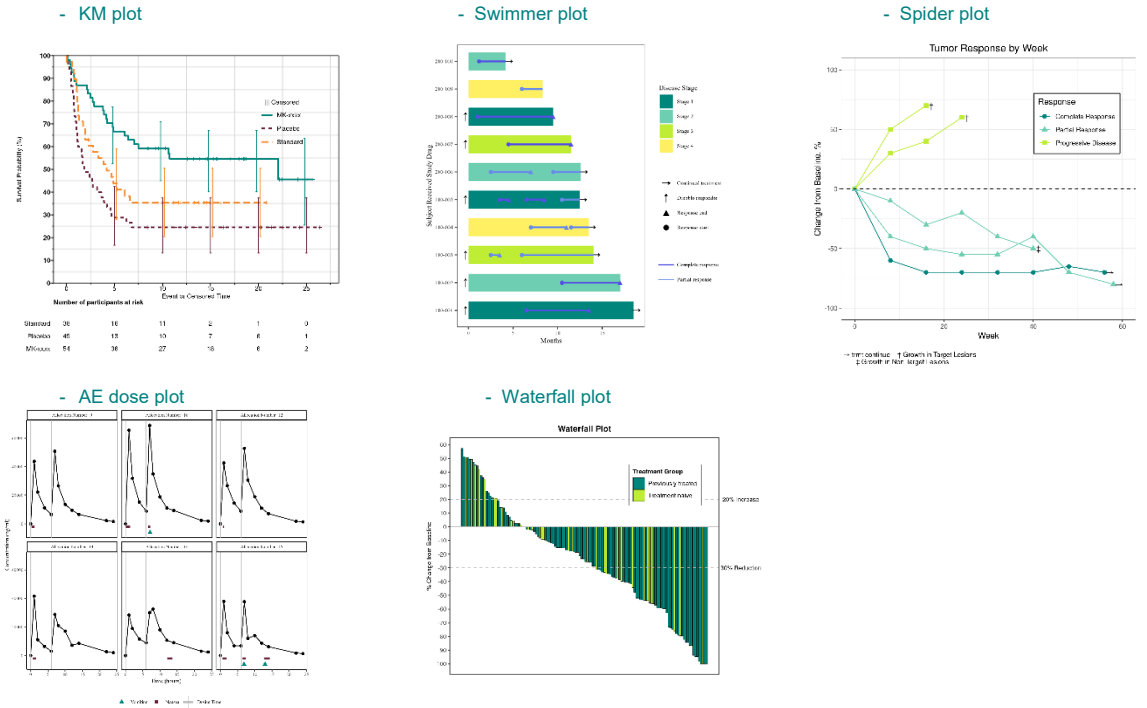

Figure 10. Examples of Figures Generated with General Data Visualization R Package: ggplot2

In our approach to visualization using R, we leverage the advanced capabilities that R offers to complement the already existing SAS graphic repository. Since R is an open-source programming language, combined with its extensive range of packages, it means that we can often find elegant solutions for various tasks. For instance, consider the scenario where we want to add a “number at risk” table beneath the KM plot we previously generated. In R, we can easily create this table using ggplot2, and then seamlessly integrate it with the plot using the R package ‘patchwork’. The function wrap_plots() from patchwork provides a straightforward method to adjust the relative height proportions between the plot and the appended table (Figure 11), showcasing the convenience and flexibility of R in creating complex, multi-component visualizations.

```
# Create count table to append
append_table <-
  ggplot2::ggplot(data = dplyr::filter(kmdata0os, !is.na(TABTIME)),
    ggplot2::aes(x = TABTIME, y = trt01p,
                 label = as.character(NBRISK))) +
```

```
  ggplot2::geom_text(
    lineheight = 1,
    size = 8 / ggplot2::.pt,
    show.legend = FALSE) +
  ggplot2::xlab("") +
  ggplot2::ylab("") +
  ggplot2::ggtitle("Number of participants at risk \n") +
  ggplot2::coord_cartesian(clip = "off") +
  mkthemes::theme_mk(
    font_size = 8, rel_small = 1, rel_tiny = 1,
    rel_large = 1, border_type = "none") +
  ggplot2::theme(
    strip.background = ggplot2::element_blank(),
    plot.margin = ggplot2::margin(t = 0, r = 0, b = 1, l = 1, unit = "lines"),
    axis.text.x = ggplot2::element_blank(),
    axis.ticks = ggplot2::element_blank(),
    axis.text.y = ggplot2::element_text(colour = "black")) +
  ggplot2::scale_x_continuous(
    breaks = seq(0, x_upper_limit, 5),
    limits = c(0, ceiling(max(kmdata0os$TIME))),
    expand = c(0.05, 0))

# Append count table under the plot
kmplot2 <-
  patchwork::wrap_plots(kmplot1, append_table, ncol = 1, heights = c(8.5, 1.5))
```

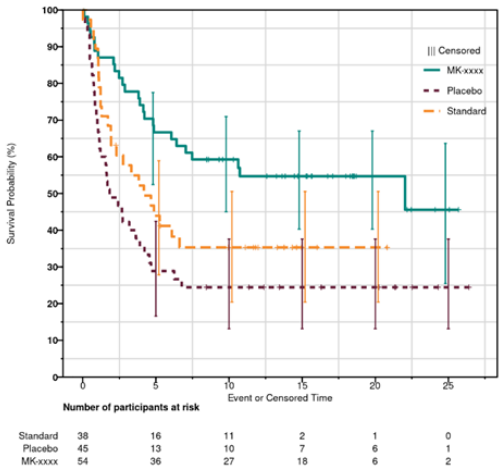

Figure 11. Examples of KM Plot with Number at Risk Table

The ggplot2 object in R offers a straightforward and flexible way to customize plots, which differs from the approach used in SAS figure generation. In SAS, modifying an existing figure often requires updating the existing program. In contrast, with R's ggplot2 object, each customization is added incrementally using the '+' operator, functioning like independent modules that each contribute a specific element to the plot without having to update the existing standard program. For instance, in the provided KM plot example, every '+' statement adds a distinct component/layer. This modular approach means that to further customize the plot, such as the KM plot (kmplot2), there's no need to spend time to understand the existing standard program. Instead, one can directly build upon the existing kmplot2 object. For example, by adding new elements like a horizontal reference line or labeling survival values at selected time points (Figure12), we don't need to have the statements creating kmplot1 again, we can use the ggplot2 object kmplot1 returned previously and stich it with the newly created counts tables. In this case if using SAS, we need to generate the entire figure at once. It demonstrates the flexibility and efficiency of using ggplot2 as solution of data visualization in R.

```
# The first component of kmplot2 object is the original kmplot without appended table
kmplot2[[1]] <- kmplot2[[1]] +

# Add horizontal reference line at survival percent equal to 50
```

```
    geom_hline(yintercept = 50,
               linetype = "dashed",
               color = "black") +
# Add label of survival value for time point 5
    geom_text(data = dplyr::filter(kmdata0os, TIME == 5),
              aes(x = TIME, y = SURVP, label = round(SURVP, 2)),
              vjust = -0.5,
              show.legend = FALSE)
```

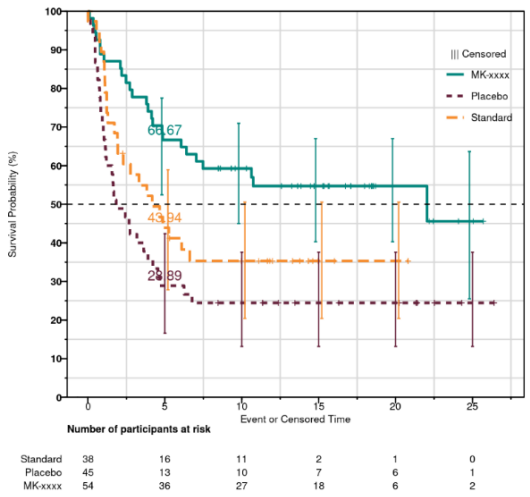

Figure 12. Examples of KM Plot with Further Customization

R Markdown and the R package knitr are powerful tools for creating dynamic, interactive documentation. They allow the integration of R code directly into documents, automating data processing and ensuring that results are always up-to-date and reproducible. With knitr package, these documents can be easily converted into various formats like HTML, PDF, and Word, enhancing the accessibility and sharing of findings. These tools are used to generate user-friendly and informative documentation in our graphic repository. For example, R visualization gallery offer an interactive experience where users can view and copy the code behind each plot by clicking the arrow at 'Click to view the code for creating plot' as shown in Figure 13.

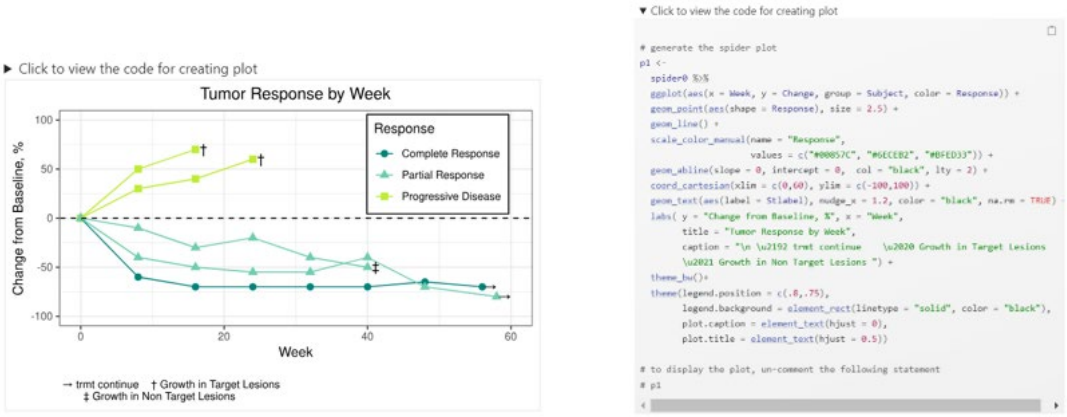

Figure 13. Example Document Generated by R Markdown and R package knitr to View Code

CONCLUSION

In conclusion, when selecting a statistical analysis software for clinical study reports and submissions, it's crucial to consider the unique advantages and limitations of each tool. SAS, as a commercial product, offers stability, reliability, and excellent customer support, with a strong Software Development Life Cycle (SDLC) and well-structured documentation. However, its high cost, slower adoption of advanced methods, and limitations in interactive figure generation are notable drawbacks.

On the other hand, R, as an open-source tool, allows rapid integration of advanced methods due to its community-driven development. The use of R Markdown ensures that documentation is consistently updated, reflecting the

current status of content. Yet, R requires internal qualification to guarantee package quality and additional efforts to maintain traceability and reproducibility, given the frequent updates in R and its packages. See more details on qualification of R packages papers [3] and [4].

It's essential to focus on the value that each tool can add, rather than merely reproducing existing capabilities while choosing the right tool. A multilingual visualization toolset is achievable through well-defined processes and standards. By effectively implementing a visualization system that harnesses the strengths of both SAS and R, we can meet the goal of producing efficient, reliable, and versatile reports to support drug and vaccine submissions. This synergy of SAS and R not only addresses the individual limitations of each tool but also leverages their combined strengths to enhance overall analytical capabilities.

REFERENCES

[1] Bingjun Wang, Yujie Zhao, Yilong Zhang "forestly – R Package to Generate Interactive Forest Plots" 2023. https://www.lexjansen.com/phuse-us/2023/dv/PAP_DV07.pdf
[2] Jeff Cheng, Abhilash Chimbirithy, Amy Gillespie, Yilong Zhang "Implementing and Achieving Organizational R Training Objectives" 2022. https://www.lexjansen.com/pharmasug/2022/SI/PharmaSUG-2022-SI-055.pdf
[3] Jane Liao, Fansen Kong, Yilong Zhang "External R Package Qualification Process in Regulated Environment" 2022. https://www.lexjansen.com/pharmasug/2022/SI/PharmaSUG-2022-SI-057.pdf
[4] Paul Bernecki, Nicole Jones, Uday Preetham Palukuru, Abhilash Vasu Chimbirithy "R Package Qualification: Automation and Documentation in a Regulated" 2023. https://www.pharmasug.org/proceedings/2023/SI/PharmaSUG-2023-SI-212.pdf

ACKNOWLEDGMENTS

The authors would like to thank management teams from Merck & Co., Inc., Rahway, NJ, USA, for their advice on this paper/presentation and want to thank Suhas Ramesh Sanjee from Merck & Co., Inc., Rahway, NJ, USA, for valuable inputs on the paper. Shuping Zhang for his contributions on the development of SAS graphic repository.

RECOMMENDED READING

1. Chang, Winston. 2012. R Graphics Cookbook: Practical Recipes for Visualizing Data. O'Reilly Media. https://www.cookbook-r.com/Graphs/.
2. Wilke, Claus O. 2019. Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures. O'Reilly Media. https://clauswilke.com/dataviz/.

CONTACT INFORMATION

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

Author Name: Bingjun Wang
Company: Merck & Co., Inc.
Address: Rahway, NJ, USA
Work Phone: 215-283-6035
Email: benjamin.wang@merck.com

Author Name: Jane Liao
Company: Merck & Co., Inc.
Address: Rahway, NJ, USA
Work Phone: 267-305-7514
Email: Jane.liu2@merck.com

Author Name: Yulia Sidi
Company: Merck & Co., Inc.
Address: Rahway, NJ, USA
Work Phone: 732-594-5815
Email: yulia.sidi@merck.com

Author Name: Nan Xiao
Company: Merck & Co., Inc.
Address: Rahway, NJ, USA
Work Phone: 267-305-3578
Email: nan.xiao1@merck.com