PharmaSUG 2022 - Paper AD-62

A Multilingual Shiny App for Drug Labelling in Worldwide Submission

Jinchun Zhang, Aiming Yang, Yiwen Luo, Nan Xiao, Yilong Zhang

Merck & Co., Inc., Kenilworth, NJ, USA

ABSTRACT

In preparing a successful worldwide submission for a drug or vaccine, one of the critical steps is to provide drug labeling in different languages and different formatting requirements to worldwide regulatory agencies. If a drug label contains figures, there is a challenge to create and maintain drug labels in different languages programmatically. For example, in oncology studies, Kaplan-Meier Plot (K-M plot) is frequently required for drug labelling worldwide. An application that can generate the figure in the required languages with controlled labeling process is highly desired. It ensures accuracy, consistency, and security of the data. In this paper, we demonstrate a user-friendly Shiny app to simplify the labeling process in collaboration with local regulatory teams who need to update drug labels in local languages. The Shiny app simplifies the manual steps to re-create K-M plots in different languages with different formatting requirements and enables advanced plot rendering. We explain the implementation of the Shiny app with code examples.

INTRODUCTION

For clinical statisticians and programmers working on pharmaceutical clinical trials, it is critical to fulfill the submission requirements successfully. Global submissions are common practice in the pharmaceutical industry. One challenge is to provide figures for drug labelling in different languages, programmatically. However, editing the figures manually is error-prone because (1) it exposes not only the text labeling but also other plot components that must be the same across languages, (2) it renders plots on individual systems which might differ from the original system that generated such plots, (3) different countries may use different languages and different writing systems, thus requiring a typeface for all languages.

Shiny is an open-source framework for building web applications in R. With Shiny, developers can implement and streamline the interactive data analysis process on a web page. Shiny apps can be running on local machines or published to a server for sharing. Behind the scenes, a Shiny app has a UI (user interface) component and a server component, while the users of the applications do not need to learn the programming language or the logic behind. The app can also provide an opportunity to maintain end-to-end processes with controlled data access and traceability.

In this paper, we discuss the idea of building a Shiny app to tackle the challenges in manual K-M plot labeling. We present this application from both the user's and developer's perspective. The first part demonstrates the business need addressed by the Shiny app. It is from the user perspective, or people who have years of clinical programming experience other than R but would like to gain the benefit of using R or Shiny app. The second part is dedicated to discussing the workflow and the corresponding coding behind the scenes. We will go through the input, update, and output process of the Shiny app. This part intends to share the work with R Shiny developers.

BENEFITS

By building a Shiny app as shown in Figure 1, we addressed the current challenges of manually labeling the K-M plot with additional benefits:

- User-friendly: minimal knowledge of R or programming required. Users only need to upload the original K-M plot data and choose the language desired to update the K-M plot languages and text formatting.

• Process-controlled: the labeling process is well-controlled and minimizes human errors. This app exposes only the text components that need to be updated per regulatory labeling standards. The remaining content in the plot is presented as read-only; thus, data consistency, accuracy, and integrity are achieved.

• Flexible: this app uses the Noto fonts that covers over 1,000 languages and more than 150 writing systems for typesetting the text labels. It provides a generic typeface solution for worldwide submissions.

• Efficient: timesaving by reducing manual efforts in rendering plots on different systems. This Shiny app uses higher performance and higher quality graphic devices , to achieve advanced plot rendering features such as access to system fonts, right-to-left (RTL) text, and system independent rendering.

PART I: USER GUIDE, HOW DOES THIS APP ADDRESS THE BUSINESS NEEDS FOR WORLDWIDE SUBMISSIONS?

In this part, we will introduce the Shiny app from the user perspective step by step. With the initial landing page displayed in Figure 1, users can load the data directly into the Shiny app. After the required K-M data is loaded, the original K-M plot in English will be displayed on the right side. Then, the users can type in desired treatments, legends, title, and choose the appropriate decimal separator for specific languages. After all input areas have been updated per required language, users can download the finalized local-language K-M plot.

Step 1. Display Landing Page as in Figure 1

The app has the title “K-M plot Labeling App”, with links to the “Home” and “About” pages in the navigation menu. The navigation bar is created via a slightly customized Shiny “navbarPage()” function to implement internal branding guidelines. The top-left panel asks for the input K-M data.

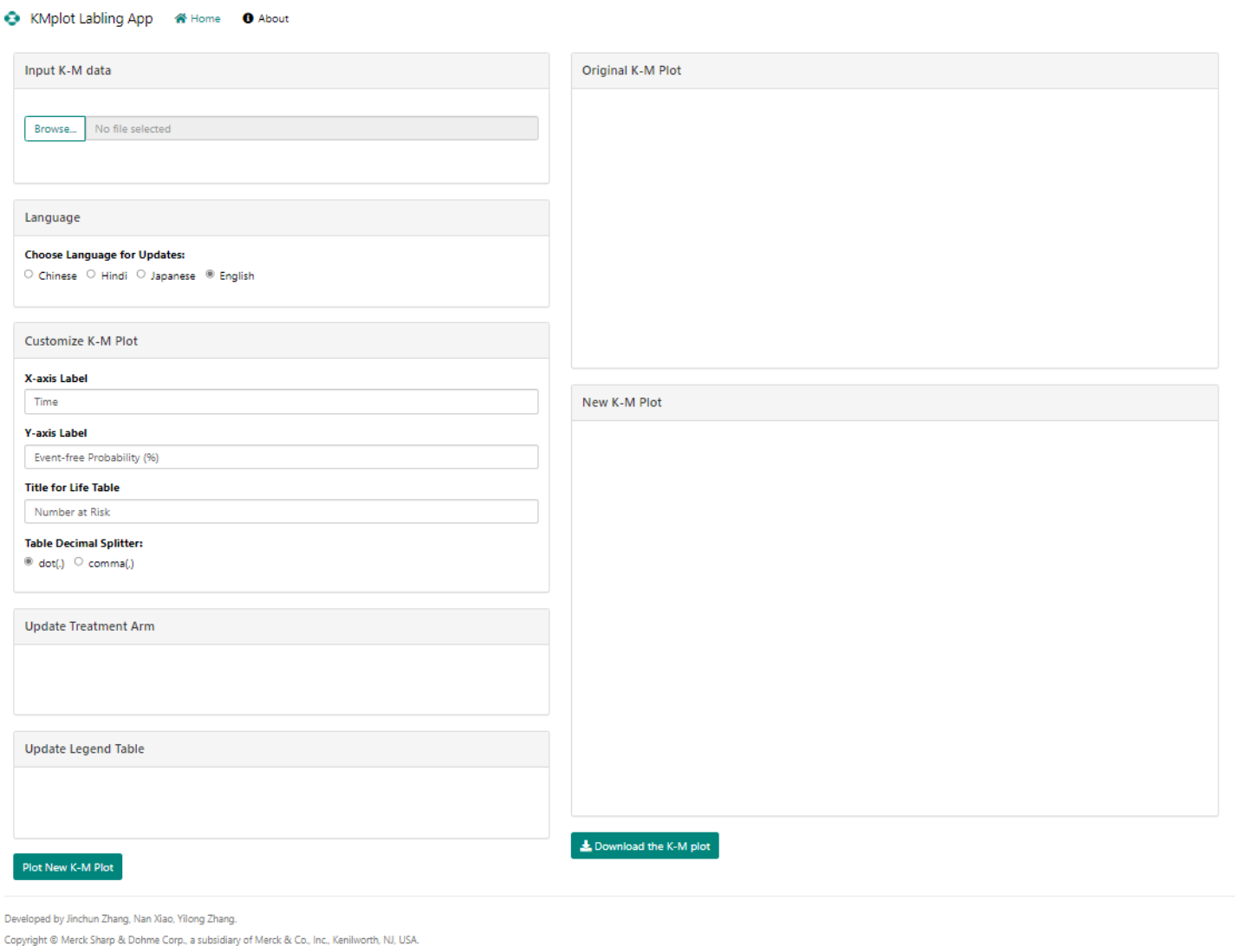

Figure 1. The landing page of the Shiny app.

Step 2. Load Data and Render Initial K-M Plot as in Figure 2

After the data upload is completed, the generated K-M plot in English will appear in the first panel on the right side.

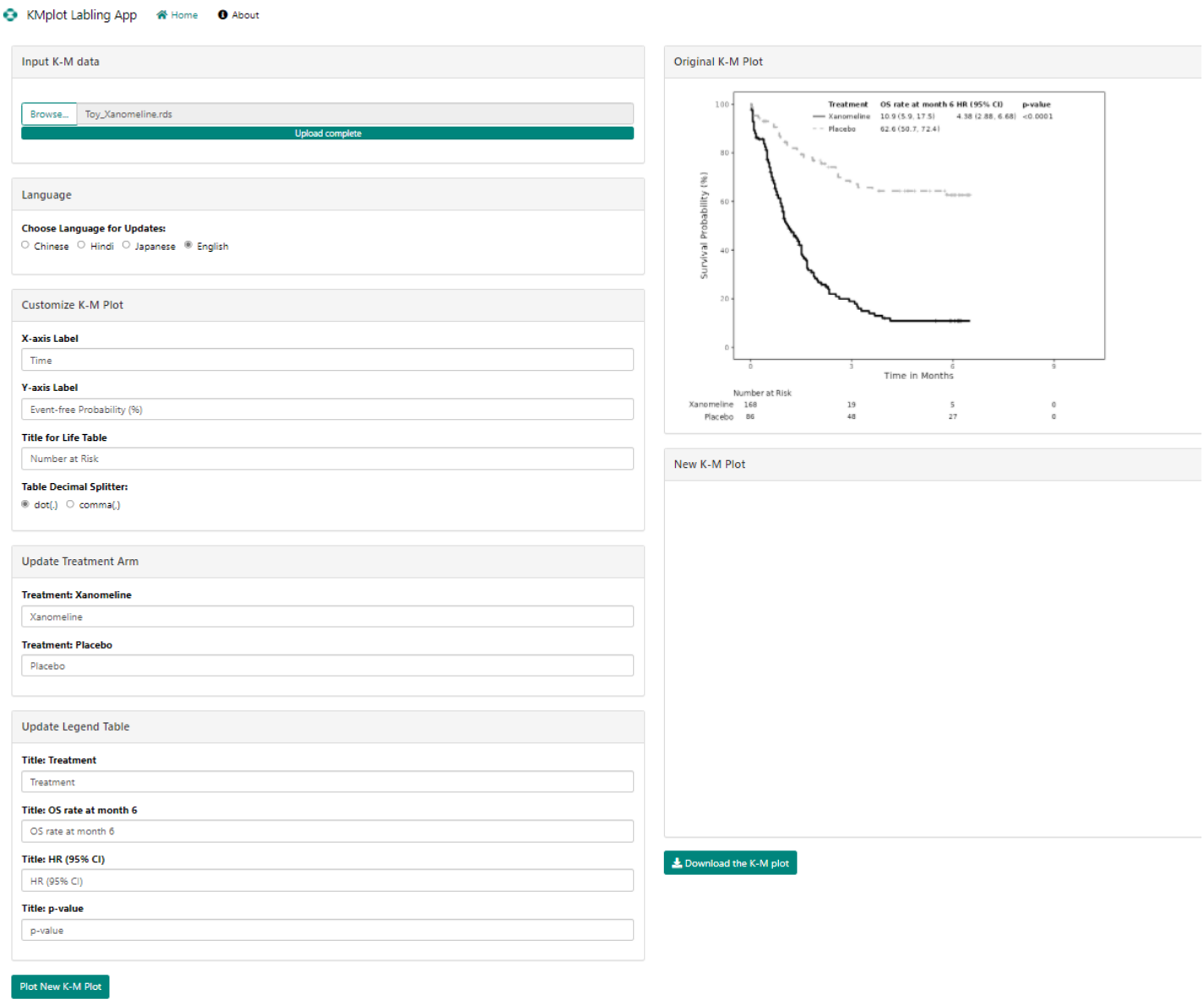

Figure 2. The original K-M plot annotated in English as the data upload completes.

Step 3. Update in Desired Language and Appropriate Format

To generate K-M plot in different languages, the following items may be updated. As we can see in Figure 2, the user can choose a different language, for example, Chinese. Then the user can enter the X-axis and Y-axis label in Chinese. For the table underneath the plot, we can replace the label “Number at risk” with the corresponding Chinese label. In some countries like the U.S., dot is used as the decimal separator, while in some European countries, comma maybe used instead, therefore the app provides an option to customize the decimal separator. The treatment arm and legend in Chinese may be entered as well. After all the updates are completed as shown

in Figure 3, the user can click the “Plot New K-M Plot” button to generate the K-M plot in the desired language, rendered inside the “New K-M Plot” panel in Figure 3.

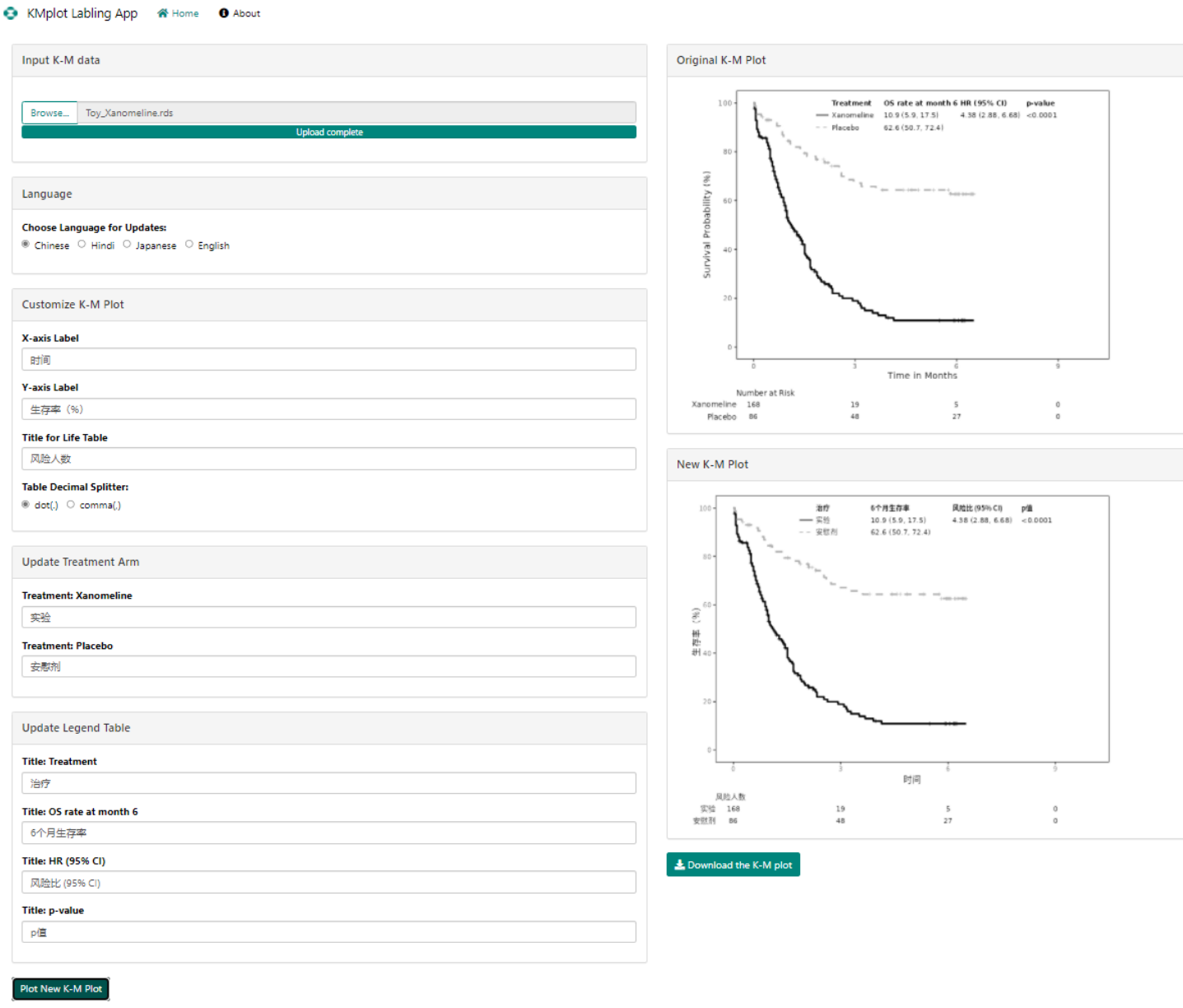

Figure 3. The updated K-M plot with the title, axis labels, and table legends in Chinese.

Step 4. Download the Updated K-M Plot.

Click the “Download the K-M plot” button, a PNG file will be downloaded with the standard name format of kmplot_yyyy-mm-dd.png. Figure 4 is a final K-M plot in Chinese.

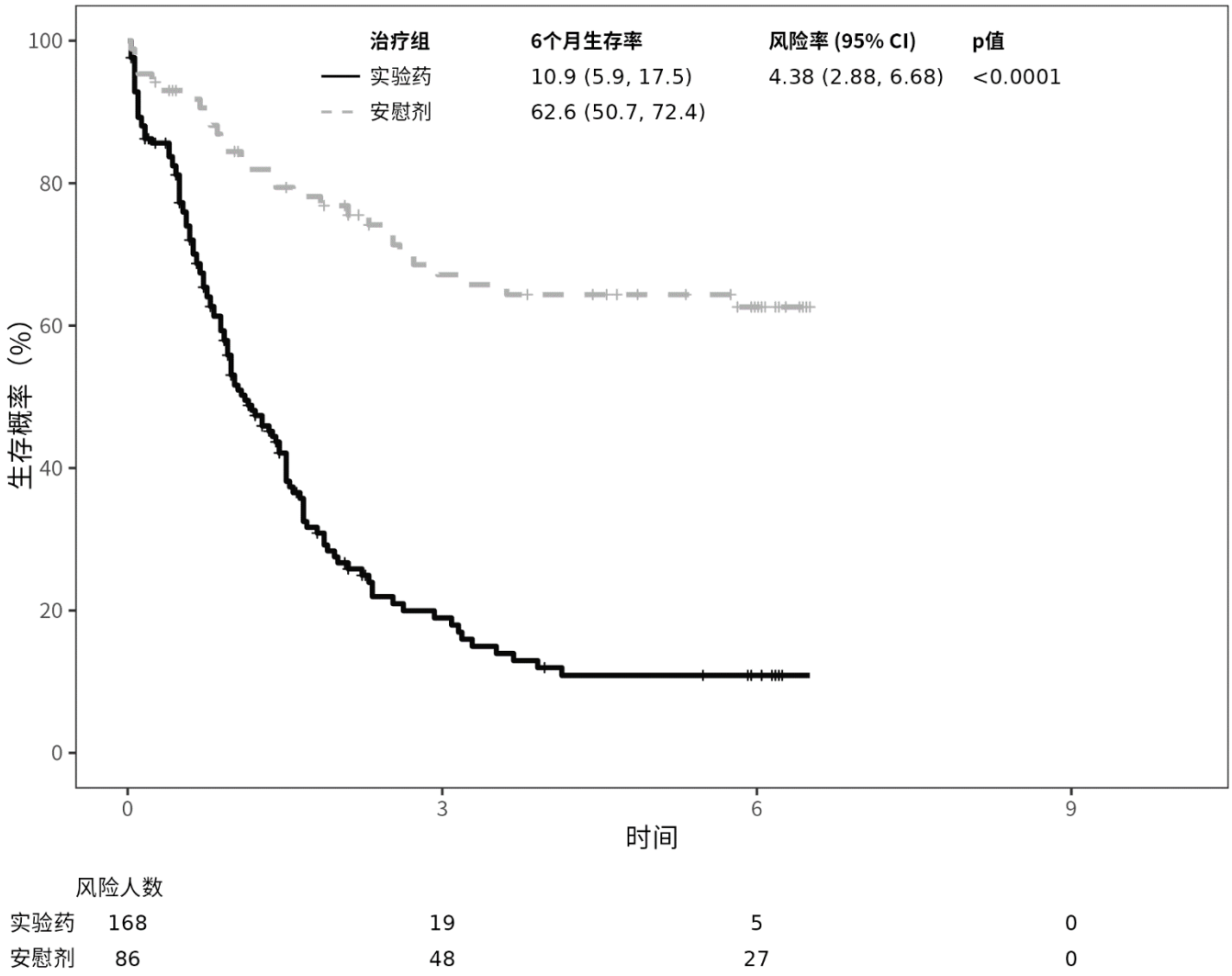

Figure 4. The K-M plot in Chinese generated by the Shiny app.

PART II: DEVELOPER GUIDE, WHAT IS THE WORKFLOW FOR THIS APP?

The application's workflow is illustrated in Figure 5. Like other Shiny apps, the process includes input, update, and output. The input process is comprised of uploading, dynamic UI generation for treatment arm and legend table title, and initial plot rendering. The key R functions used include ragg::agg_png() which draws to PNG (Portable Network Graphic) format file, grid::grid.draw() which produces graphical output from a graphical object, and a function for survival analysis. The update process includes selecting the typeface for the chosen language, updating the axis labels, customizing the decimal separator (as needed), updating the treatment arm labels, and updating the legend table titles. The majority of the server logic for updating the K-M plot and the lifetable is implemented as R functions to encourage code reuse and iterative development. For the output process, plot renderers and download handlers are two main Shiny components used. Key R functions for the output process are again ragg::agg_png(), grid::grid.draw(), and a function for survival analysis. We will go through the main Shiny components constructed in the app and R functions developed for generating the original and localized K-M plots.

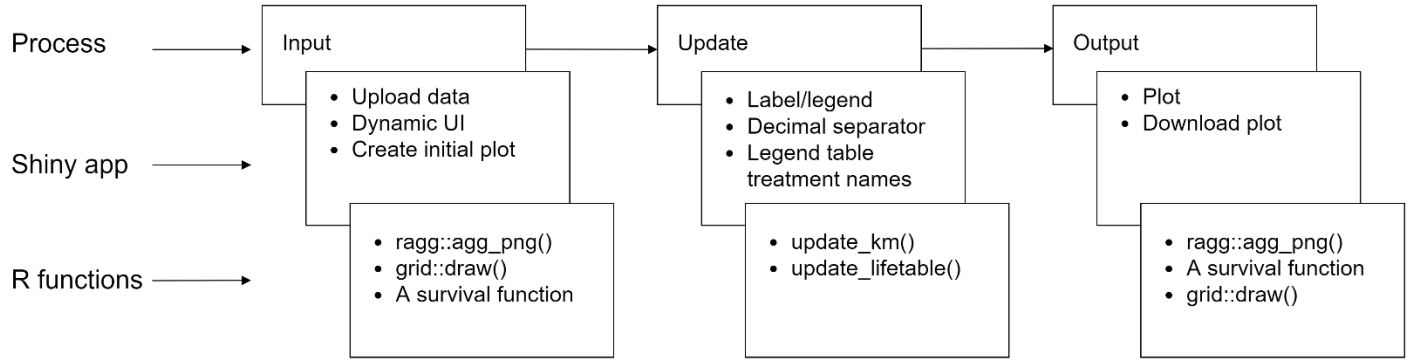

Figure 5. Process flow overview.

INPUT PROCESS

Input Process – Upload

For Shiny, file uploading is simple to implement on the UI side by using shiny::fileInput(). In our example, the code looks like below. The fileInput ID is defined as “file_survKM”, with a label to guide the user to load the right file. The argument accept is a suggestion to the web browser to limit the possible input file types to .rds files (for storing serialized R object) only.

```
fileInput(
  "file_survKM",
  label = tagList("Input K-M data"),
  accept = ".rds"
)
```

On the server side, a reactive expression is implemented to read the data. The shiny::req() call is used to make sure that the server waits until the file is uploaded.

```
# Reactive to uploaded data
filedata <- reactive({
  req(input$file_survKM)
  infile <- input$file_survKM
  if (is.null(infile)) {
    return(NULL)
  }
  readRDS(infile$datapath)
})
```

Input Process - Dynamic UI for Treatment Arm and Legend Table Title

The shiny::renderUI() function, which is used in server.R in conjunction with the shiny::uiOutput() function in ui.R, enables us to render UI components dynamically and make the results appear in a predetermined place in the UI.

In ui.R:

```
headingPanel(
```

```
  title = "Update Treatment Arm",
  style = "min-height: 120px;",
  uiOutput("medname")
),

headingPanel(
  title = "Update Legend Table",
  style = "min-height: 120px;",
  uiOutput("tbtitle")
)
```

In server.R:

```
# Send med name to UI based on input
output$medname <- renderUI({
  md <- medlevel()
  myInput <- lapply(md, function(i) {
    textInput(
      inputId = glue("{i}"),
      label = glue("Treatment: {i}"),
      value = glue("{i}")
    )
  })
})

# Send legend table title to UI based on input
output$tbtitle <- renderUI({
  indat <- filedata()
  lyr <- sapply(indat$km$layers, function(j) j$aes_params$label)
  tmp <- sapply(lyr, function(j) sum(grepl("Treatment", j)) + sum(grepl("p-va
lue", j)) * 2)
  idx <- c(0, 0)
  idx[1] <- which(tmp == 1)
  idx[2] <- which(tmp == 2)

  myInput <- lapply(seq(idx[1], idx[2], by = 2), function(i) {
    name <- lyr[[i]]
    textInput(
      inputId = glue("{name}"),
      label = glue("Title: {name}"),
      value = glue("{name}")
    )
  })
})
```

Input Process - Create the original plot

In the dynamic UI, shiny::imageOutput() is used to change the UI using code executed in the server function, including shiny::renderImage().

In ui.R:

```
headingPanel(
  title = "Original K-M Plot",
  style = "min-height: 400px;",
```

```
  imageOutput("OrigPlot", height = "100%", width = "100%")
)
```

In server.R:

```
# Start updating K-M plot only if button is clicked
observeEvent(input$plotKM, {
  v$plot <- plotInput()
})

output$plot <- renderImage({
  outfile <- tempfile(fileext = ".png")
  newplot <- v$plot
})
```

Input Process - Key Functions: agg_png(), a survival function, and grid.draw()

The PNG file is generated by the ragg::agg_png() function, which opens the AGG PNG device as shown below. In addition, a call to dev.off() is used to close the graphic device after the plotting code.

```
# Generate the PNG
agg_png(outfile, width = 96 * 7.25, height = 96 * 5.6, res = 96)
```

An internal survival analysis package is used to generate the K-M plot.

UPDATE PROCESS

The update process involves updating the original language to the registration-needed local language, which also includes customizing plot details such as the decimal separator. The Noto fonts for specific languages needed are downloaded from Google Fonts and stored in the app repository. The plot customization and treatment arm/legend table updates are achieved by functions update_KM() and update_LifeTable() below. Specifically, we updated the label and font family of aes_params in the corresponding layers of the ggplot2 object.

Function update_KM():

```
update_KM <- function(km, xlab_name = NULL, ylab_name = NULL,
                      medname = "", decimal_splitter = ".",
                      textfamily = "Noto Sans") {
  km <- km + theme(text = element_text(family = textfamily))

  # Change ylab and xlab
  km <- km + ylab(ylab_name) + xlab(xlab_name)

  # Update legend title
  lyr <- sapply(km$layers, function(j) j$aes_params$label)
  tmp <- sapply(
    lyr,
    function(j) sum(grepl("Treatment", j)) + sum(grepl("p-value", j)) * 2
  )
  idx <- c(0, 0)
  idx[1] <- which(tmp == 1)
```

```
  idx[2] <- which(tmp == 2)

  for (i in seq(idx[1], idx[2], by = 2)) {
    tb_old <- lyr[[i]]
    tb_new <- input[[glue("{tb_old}")]]

    km$layers[[i]]$aes_params$label <- tb_new
    km$layers[[i]]$aes_params$family <- textfamily
  }

  # Update decimal split
  if (decimal_splitter == ",") {
    for (i in 7:length(km$layer)) {
      km$layers[[i]]$aes_params$label <- str_replace(
        km$layers[[i]]$aes_params$label, "\\.", ","
      )
    }
  }

  # Change the treat name in legend table
  tmp <- km$layers[[5]]$aes_params$label
  for (i in medname) {
    mi <- input[[glue("{i}")]]
    if (mi != "") {
      tmp <- str_replace(tmp, i, mi)
    }
  }

  km$layers[[5]]$aes_params$label <- tmp
  km$layers[[5]]$aes_params$family <- textfamily

  return(km)
}
```

Function update_LifeTable():

```
update_LifeTable <- function(lifetable, medname = NULL, title = NULL,
                             textfamily = "Noto Sans") {
  tmp <- lifetable$data$strata
  for (i in medname) {
    mi <- input[[glue("{i}")]]
    if (mi != "") {
      tmp <- str_replace(tmp, i, mi)
    }
  }
  lifetable$data$strata <- tmp

  if (!is.null(title)) {
    lifetable <- lifetable + ggtitle(eval(title))
  }
  lifetable <- lifetable + theme(text = element_text(family = textfamily))
  return(lifetable)
}
```

OUTPUT PROCESS

A Shiny download handler generates the output of the process. Again, key R functions includes ragg::agg_png(), a survival function, and grid::grid.draw(). The corresponding functions used in the server logic to take inputs to update the plot are plotInput() and shiny::downloadHandler().

Function used for plot output:

```
plotInput <- function() {
  indat <- filedata()
  smname <- medlevel()
  xlab <- input$xlab
  ylab <- input$ylab
  ltb_title <- input$lftitle
  decimal_splitter <- ifelse(input$decimal == 1, ".", ",")
  textfamily <- ifelse(input$lang == 1, "Noto Sans CJK SC",
    ifelse(input$lang == 2, "Noto Sans Devanagari",
      ifelse(input$lang == 3, "Noto Sans CJK JP", "Noto Sans")
    )
  )

  newKM <- update_KM(indat$km,
    xlab_name = xlab, ylab_name = ylab,
    medname = smname, decimal_splitter = decimal_splitter,
    textfamily = textfamily
  )

  newLTb <- update_LifeTable(
    indat$lifetable,
    medname = smname,
    title = ltb_title,
    textfamily = textfamily
  )

  return(list(newKM, newLTb))
}
```

Function: plot download handler

```
output$downloadPlot <- downloadHandler(
  filename = function() {
    paste0("KMplot_", Sys.Date(), ".png")
  },
  content = function(file) {
    newplot <- v$plot
    out <- mksurv:::append_table(newplot[[1]], newplot[[2]], position_leg = "
none")
    agg_png(file, width = 96 * 7.25, height = 96 * 5.6, res = 96)
    grid.newpage()
    grid.draw(out)
    dev.off()
  }
)
```

CONCLUSION

In preparing a successful worldwide submission for a drug or vaccine, the proposed Shiny app can assist critical drug labeling processes targeting different languages and different formatting requirements from regulatory agencies worldwide. An application that can generate the commonly used Kaplan-Meier Plot (K-M plot) in the required language based on a single data source is highly desired. It ensures accuracy, consistency, and security of the data. The Shiny app simplifies the manual steps to re-create a K-M plot in different languages and different formatting requirements. Shiny, as a framework for building web applications in R, can enable the essential interactive user experience to fulfill such requirements. Users do no need to know any programming languages or understand the backend logic to implement the labeling process.

REFERENCES

Hadley Wickham (2021). Mastering Shiny. O'Reilly Media, Inc. https://mastering-shiny.org/

Hadley Wickham (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York. https://ggplot2.tidyverse.org

Noto: A typeface for the world. Retrieved March 8, 2022, from https://fonts.google.com/noto

Thomas Lin Pedersen and Maxim Shemanarev (2021). ragg: Graphic Devices Based on AGG, https://ragg.r-lib.org

Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert and Barbara Borges (2021). shiny: Web Application Framework for R. R package version 1.7.1. https://shiny.rstudio.com/

ACKNOWELEGEMENTS

We are grateful for our colleague Simiao Ye kindly providing a reference flow chart prototype. We are thankful for the support and reviewing by Shailaja Suryawanshi and Christine Gause.

CONTACT INFO

Your comments and questions are valuable and appreciated. The authors can be reached at

Jinchun Zhang, Ph.D.
Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Email: jinchun.zhang@merck.com

Aiming Yang, Ph.D.
Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Email: aiming_yang@merck.com

Yiwen Luo, Ph.D.
Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Email: yiwen.luo@merck.com

Nan Xiao, Ph.D.

Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Email: nan.xiao1@merck.com

Yilong Zhang, Ph.D.
Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Email: yilong.zhang@merck.com