Paper PP07

Plunging into Pools: Dive into a Comparative Analysis of Pooled TLFs with R

Sophie Khan, J&J Innovative Medicine, High Wycombe, UK

ABSTRACT

As we delve into the realm of innovative programming methodologies beyond SAS, we venture into the ever-evolving landscape of R programming. R emerges as a versatile tool, offering enhanced efficiency in data analysis and visualization, and opening new avenues of exploration and discovery in the pharmaceutical industry.

This paper presents a comparative analysis of pooled TLFs using R, focusing on a pooling report prepared for regulatory submission that integrates findings from multiple studies across many indications. The methodology involves importing RTF documents to construct dataframes, extracting related R code from an Excel document, and utilizing the 'compare_df' package for dataframe comparisons. Results include a comprehensive suite of HTML outputs for each pooled TLF, facilitating informed decision-making. The pooled TLFs were segmented into distinct components, with data comparisons revealing identical matches across individual study TLFs.

This innovative approach not only boosts the quality assessment of pooled TLFs but also significantly streamlines workflow efficiency, ultimately accelerating review processes.

BACKGROUND

A team faced a lengthy and resource-intensive task involving manual comparison of pooled TLFs against individual study TLFs. This process required a small number of employees working over a few weeks to ensure accuracy, and this highlighted the need for a more efficient solution. Building on this experience, an effort was initiated to automate this comparison process using R for our next pooled project to streamline our workflow. The goal was to significantly reduce the time required, improve consistency, and explore new applications of R within our project.

This was a Safety Clinical Summary (SCS) pooling project focused on evaluating the safety profile of an investigational drug across multiple indications with data from up to 14 phase 2 and 3 studies.

INTRODUCTION

The objective of this pooling effort was to provide a comprehensive safety overview across these indications and study populations, enabling a robust safety assessment of the investigational drug. This integrated analysis supported regulatory submissions. This was achieved by conducting a comprehensive and efficient comparative analysis, scrutinizing each pooled TLF against its respective individual study TLFs whenever feasible.

METHODOLOGY

An excel document is created which contains information about each pooled TLF and its associated individual study TLFs which is imported and processed in an overall QC R Program. The input file contains information about the location, contents, and R code for each TLF preparing them into dataframes which are then used in the QC program for analysis and comparison.

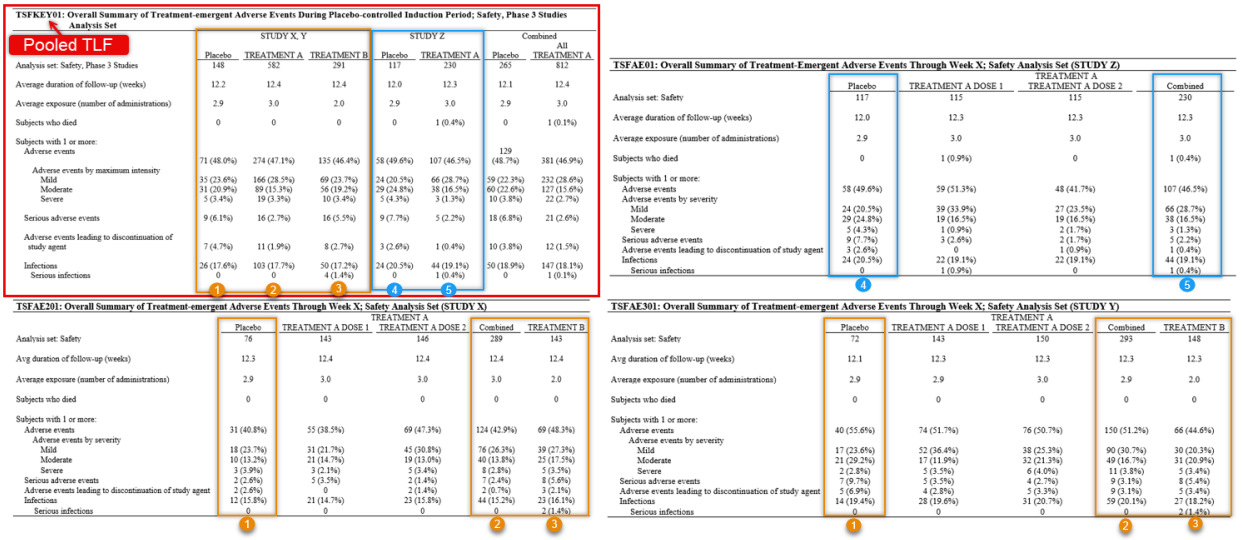

Figure 1. Display of the RTF documents of a pooled TLF and its individual study TLFs, and the plan to split the data ready to compare outlined

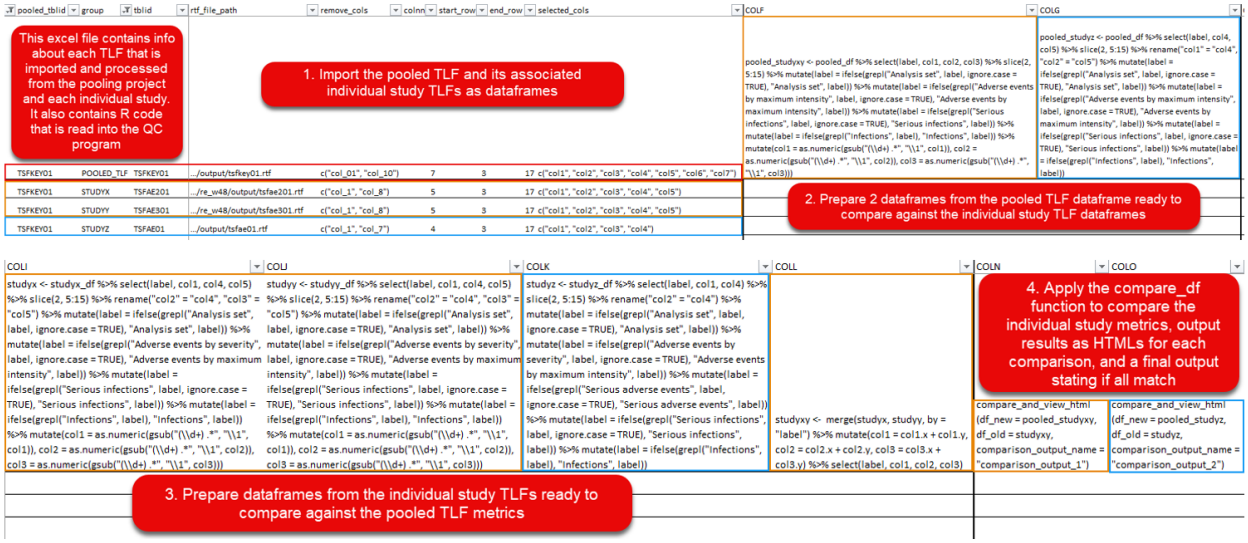

Figure 2. Display of the Excel document containing R coding to extract for each pooled TLF

R QC POOLED PROGRAM

An overall QC program in R was created which incorporates 3 R functions wrapped in one overall R function to run for each pooled TLF.

The first R function named process_rtf_data.r reads in the excel document, and in turn uses that information to import all the TLF RTFs as individual dataframes manipulating each dataframe keeping the necessary information ready for comparison of the individual study TLF dataframes against the pooled TLF dataframe.

Below is a display of the **process_rtf_data** R function that imports an RTF document and programmatically cleans for use as a dataframe.

```
# ------------------------------------- START -------------------------------
# ---------- 1. Overview and Descriptions ------------------------------

# An Excel file in the Input folder contains info about each TLF that requires importing and
processing from Pooling, and Studies XYZ.

# rtf_file_path           # rtf filename and location
# row_start               # delimiter at the beginning of each row
# row_end                 # delimiter at the end of each row
```

```
# cell_end                # delimiter at the end of each cell
# delimiter               # delimiter in-between each cell
# coln                    # number of columns in rtf
# remove_cols             # column names to remove before renaming to label, col1-colx. If there
are more than 9 columns then col_1 is named col_01 by default
# colnn                   # total number of columns (x from colx) excluding label
# start_row               # row number to start from to keep
# end_row                 # row number to end at to keep
# selected_cols           # column names to keep
# tblid_cd                # name of Pooling TLF to compare, used in condition
# tblid                   # name of TLF to pull in from either Pooling, Studies XYZ, used in
condition

# Test outside of macro/function

# Example
# tblid_cd <- "TSFKEY01"
# tblid <- "TSFAE201"

process_rtf_data <- function(tblid_cd, tblid) {

  # ---------- 2. Read in Excel file -------------------------------

  # Import the entire Excel file
  excel_data <- read_excel(paste0(dpspath[environ], "/pooledinput.xlsx"))

  # To output dataframe in a function
  excel_data <<- excel_data

  # Define condition based on columns tblid_cd and tblid in the Excel file
  condition <- excel_data$tblid_cd == tblid_cd & excel_data$tblid == tblid

  # ---------- 3. Read in rtf -------------------------------

  # Input TLF info from the Excel file
  rtf_file_path <- excel_data$rtf_file_path[condition]
  row_start <- "*|"
  row_end <- "*|"
  cell_end <- "|"
  delimiter <- "*|"
  coln <- 2
  remove_cols <- eval(parse(text = excel_data$remove_cols[condition]))
  colnn <- excel_data$colnn[condition]
  start_row <- excel_data$start_row[condition]
  end_row <- excel_data$end_row[condition]
  selected_cols <- eval(parse(text = excel_data$selected_cols[condition]))

  # Parses an RTF file and extracts plain text as character vector
  y <- read_rtf(file = rtf_file_path,
                verbose = FALSE,
                row_start = row_start,
                row_end = row_end,
                cell_end = cell_end,
                ignore_tables = FALSE,
                check_file = TRUE)

  # ------ 4. Add manipulation to tidy up and then convert string to a dataframe -----

  # Filters the vector y to include only those elements where the count of the string "|" is
equal to the value of coln.
  # The result is stored in the variable col.
  col <- y[str_count(y, fixed(delimiter)) == coln]

  # Filters the vector col to include only those elements that contain at least one alphanumeric
character
  col <- col[grepl("[a-zA-Z0-9]", col)]

  # Replaces all newline characters in the vector or column col with space characters and assigns
the modified result back to the variable col
  col <- gsub("\n", " ", col)

  # RTF pulled in with all info in one column before separation and tidy up
  z1 <- data.frame(col)

  z1 <<- z1
```

```
  # Split each string into columns separated by |
  df0 <- cSplit(indt = z1,
                splitCols = "col",
                sep = "|",
                direction = "wide",
                fixed = TRUE,
                drop = TRUE,
                stripWhite = TRUE,
                makeEqual = NULL,
                type.convert = FALSE)

  df0 <<- df0

  # Remove empty first & last columns
  df1 <- df0 %>% select(-all_of(remove_cols))

  df1 <<- df1

  # First column of titles/subtitles in tbl
  colnames <- c("label")

  # Create loop to rename columns col1-colx
  for (i in 1:(colnn)) {
    colnames <- append(colnames, paste0("col", i))
  }

  # Rename remaining columns col1-colx per standard tbl layout
  colnames(df1) <- colnames

  df1 <<- df1

  # ----- 5. Tbl manipulation - rename analysis set row, customise rows & columns to keep -------

  result_data <- df1 [start_row:end_row,] %>%
    mutate(label = ifelse(grepl("Analysis set", label), "Analysis set", label)) %>%
    select(label, !!!selected_cols)

}

# ------- 6. Run function (Macro Sample Call) ------------------------------

# Pooling
# pooled_df <- process_rtf_data (tblid_cd = "TSFKEY01", tblid = "TSFKEY01")

# STUDY X
# studyx_df <- process_rtf_data (tblid_cd = "TSFKEY01", tblid = "TSFAE201")

# STUDY Y
# studyy_df <- process_rtf_data (tblid_cd = "TSFKEY01", tblid = "TSFAE301")

# STUDY Z
# studyz_df <- process_rtf_data (tblid_cd = "TSFKEY01", tblid = "TSFAE01")

# ---------------------------------- END ------------------------------------------
```

Following that, the dataframes are cleaned via the code read in from the excel document, ready for comparisons which are output in HTML documents.

```
# ---------------------------------- START ----------------------------------------
# ---------- Overview and Descriptions -------------------------------

# compareDF::compare_df
# Compare Two dataframes. Do a git style comparison between two data frames of similar columnar
structure

# Test outside of macro/function

# Example
# df_new <- studyxy
# df_old <- pooled_studyxy
# comparison_output_name <- "comparison_output_1"

compare_and_view_html <- function(df_new, df_old, comparison_output_name) {
```

```
  comparison_output <- compare_df(
    df_new = df_new,
    df_old = df_old,
    group_col = "label",
    # exclude = c("col3", "col4", "col5", "col6", "col9", "col10"),
    tolerance = 0.0000001,
    tolerance_type = "ratio",
    stop_on_error = FALSE,
    keep_unchanged_rows = TRUE, # uncomment if wish to display matches and unmatches
    keep_unchanged_cols = TRUE,
    change_markers = c("+", "-", "="),
    round_output_to = 3
  )

  # Assign the comparison_output to a variable
  assign(comparison_output_name, comparison_output, envir = .GlobalEnv)

  # View the comparison output
  view_html(comparison_output)
}

# ------- Run function (Macro Sample Call) ------------------------------

# compare_and_view_html (df_new = studyxy, df_old = pooled_studyxy, comparison_output_name =
"comparison_output_1")
# compare_and_view_html (df_new = studyz, df_old = pooled_studyz, comparison_output_name =
"comparison_output_2")

# ------------------------------------ END ----------------------------------------
```

Above, a function named **compare_and_view_html** is created utilizing the compare_df function and outputting the results in a HTML document.

It was recognized that with this pooling project, the majority of the individual study TLFs had a very similar, if not the same, layout and design as the pooled TLF, and this was utilized by creating a final R function named print_comparisons which looked at the comparison groupings of the individual study TLFs against the metrics in the pooled TLF, with a third and final comparison between the 2 groupings to confirm the metrics from the each individual study TLF matched those present in the pooled TLF. A final HTML document is created looking at all comparisons of a pooled TLF vs. its individual TLFs and the evaluation is displayed in the output.

An R function named **print_comparisons** is displayed below to look at all possible individual study metric groupings against the pooled TLF and a final assessment of the results is presented in a final HTML output.

```
# ------------------------------------ START ----------------------------------------
# ---------- Overview and Descriptions ------------------------------

#  Output comparisons in html files based on results from applying function
compare_and_view_html.r

# Test outside of macro/function

# Example
# tblid_cd <- "TSFKEY01"
# tblid <- "TSFAE201"

print_comparisons <- function(tblid_cd, tblid, output1_filename, output2_filename) {

  if (exists("comparison_output_1") && dim(comparison_output_1$comparison_df)[1] > 0) {
    create_output_table(
      comparison_output_1,
      output_type = "html",
      file_name = paste0(opath[environ], "/QC_", tblid_cd, "_", output1_filename, ".html"),
      limit = 100,
      color_scheme = c(addition = "#52854C", removal = "#FC4E07", unchanged_cell = "#999999",
unchanged_row = "#293352"),
      headers = NULL,
      change_col_name = "chng_type",
      group_col_name = "label"
    )
  }
```

```
  if (exists("comparison_output_2") && dim(comparison_output_2$comparison_df)[1] > 0) {
    create_output_table(
      comparison_output_2,
      output_type = "html",
      file_name = paste0(opath[environ], "/QC_", tblid_cd, "_", output2_filename, ".html"),
      limit = 100,
      color_scheme = c(addition = "#52854C", removal = "#FC4E07", unchanged_cell = "#999999",
unchanged_row = "#293352"),
      headers = NULL,
      change_col_name = "chng_type",
      group_col_name = "label"
    )
  }

  if ((exists("comparison_output_1") && any(comparison_output_1$comparison_df$chng_type != "="))
||
      (exists("comparison_output_2") && any(comparison_output_2$comparison_df$chng_type != "=")))
{

    # Print mismatches from comparison_output_1
    if (exists("comparison_output_1")) {
      cat("Mismatches from comparison_output_1:\n")
      print(comparison_output_1$comparison_df[comparison_output_1$comparison_df$chng_type != "=",
])
    }

    # Print mismatches from comparison_output_2
    if (exists("comparison_output_2")) {
      cat("\nMismatches from comparison_output_2:\n")
      print(comparison_output_2$comparison_df[comparison_output_2$comparison_df$chng_type != "=",
])
    }
  }

  if ((exists("comparison_output_1") && any(comparison_output_1$comparison_df$chng_type != "="))
&&
      (exists("comparison_output_2") && all(comparison_output_2$comparison_df$chng_type == "=")))
{

    sink(paste0(opath[environ], "/QC_", tblid_cd, "_comparison_output.html"))
    print("Please check output_1.")
    sink()
  }

  if ((exists("comparison_output_1") && all(comparison_output_1$comparison_df$chng_type == "="))
&&
      (exists("comparison_output_2") && any(comparison_output_2$comparison_df$chng_type != "=")))
{

    sink(paste0(opath[environ], "/QC_", tblid_cd, "_comparison_output.html"))
    print("Please check output_2.")
    sink()
  }

  if ((exists("comparison_output_1") && any(comparison_output_1$comparison_df$chng_type != "="))
&&
      (exists("comparison_output_2") && any(comparison_output_2$comparison_df$chng_type != "=")))
{

    sink(paste0(opath[environ], "/QC_", tblid_cd, "_comparison_output.html"))
    print("Please check output_1 and output_2.")
    sink()
  }

  if ((exists("comparison_output_1") && all(comparison_output_1$comparison_df$chng_type == "="))
&&
      (exists("comparison_output_2") && all(comparison_output_2$comparison_df$chng_type == "=")))
{

    sink(paste0(opath[environ], "/QC_", tblid_cd, "_comparison_output.html"))
    print("No unequal values found! Good job.")
```

```
    sink()
  }

}

# ------- Run function (Macro Sample Call) -----------------------------

# print_comparisons (tblid_cd = "TSFKEY01", tblid = "TSFAE201")

# ---------------------------------- END -------------------------------------------
```

A final QC R program is created pulling in each of these R functions so this comparison can be rolled out across >200 pooled TLFs.

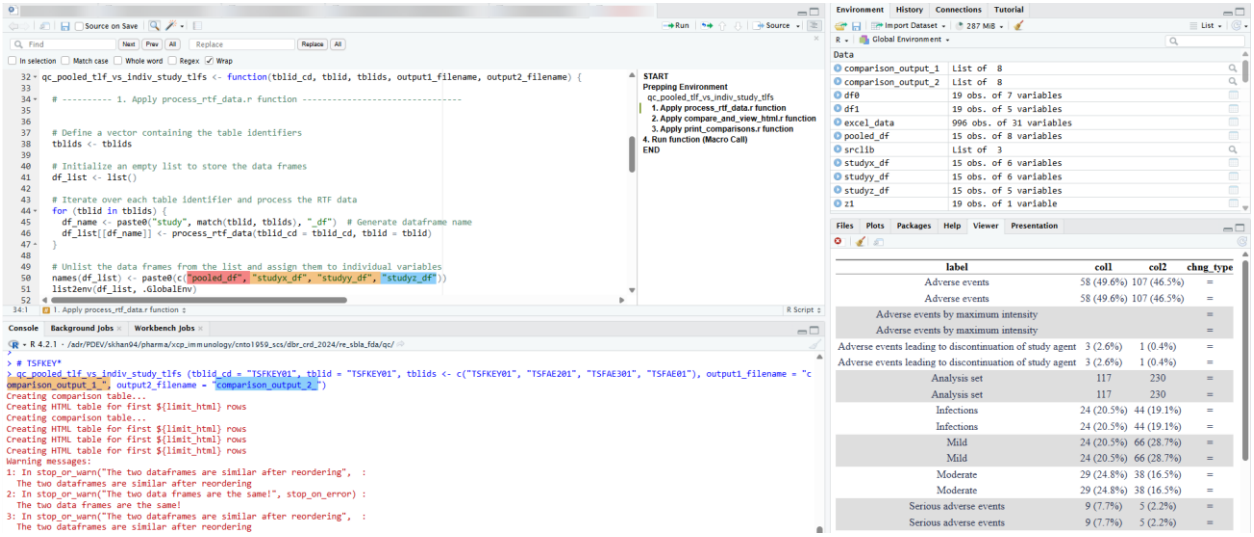

Figure 3. Display of the QC R program as it is run for a pooled TLF.

Below is the final code combining the 3 R functions to wrap all the above steps in an overall QC R program named **qc_pooled_tlf_vs_indiv_study_tlfs** to apply across all pooled TLFs. At the end of the program, it listed >200 pooled TLFs compared against their individual study TLFs. Only one example of a pooled TLF, TSFKEY01, has been displayed in this paper.

```
# ---------------------------------- START -------------------------------------------

qc_pooled_tlf_vs_indiv_study_tlfs <- function(tblid_cd, tblid, tblids, output1_filename,
output2_filename) {

# ---------- 1. Apply process_rtf_data.r function -----------------------------

# Define a vector containing the table identifiers
tblids <- tblids

# Initialize an empty list to store the data frames
df_list <- list()

# Iterate over each table identifier and process the RTF data
for (tblid in tblids) {
  df_name <- paste0("gal", match(tblid, tblids), "_df")  # Generate dataframe name
  df_list[[df_name]] <- process_rtf_data(tblid_cd = tblid_cd, tblid = tblid)
}

# Unlist the data frames from the list and assign them to individual variables
names(df_list) <- paste0(c("pooled_df", "studyx_df", "studyy_df", "studyz_df"))
list2env(df_list, .GlobalEnv)

# ---------- 2. Apply compare_and_view_html.r function -------------------------

# Define condition based on columns tblid_cd and tblid in the Excel file
```

```
condition <- excel_data$tblid_cd == tblid_cd & excel_data$tblid == tblid_cd

excel_data <- read_excel(paste0(dpspath[environ], "/pooledinput.xlsx"))

# Run code from individual cells from pooledinput.xlsx
# Extract the relevant columns based on the condition
columns <- excel_data[condition, c("COLF", "COLG", "COLH", "COLI", "COLJ", "COLK", "COLL",
"COLM")]

# Create a list to store the resulting data frames
result_dataframes <- list()

# Evaluate expressions from each column
for (col_name in names(columns)) {
  result_dataframes[[col_name]] <- eval(parse(text = columns[[col_name]]))
}

# Output the resulting data frames
result_dataframes

# Evaluate comparisons using compare_df package

eval(parse(text = excel_data$COLN[condition])) # comparison_output_1

eval(parse(text = excel_data$COLO[condition])) # comparison_output_2

# ---------- 3. Apply print_comparisons.r function ----------------------------

# Output comparisons in html files based on results

print_comparisons (tblid_cd = tblid_cd, tblid = tblid_cd, output1_filename = output1_filename,
output2_filename = output2_filename)

}

# ------- 4. Run function (Macro Call) ------------------------------

# TSFKEY*
qc_pooled_tlf_vs_indiv_study_tlfs (tblid_cd = "TSFKEY01", tblid = "TSFKEY01", tblids <-
c("TSFKEY01", "TSFAE201", "TSFAE301", "TSFAE01"), output1_filename = "comparison_output_1",
output2_filename = "comparison_output_2")

# TSFAE*, TSFSAE*, TSFAESV*, TSFDCAE*, TSFINFE*, TSFSINFE*, TSFMAL*, TSFVTE*, TSFMACE*, TSFTB*,
TSFHEP*, TSFISR*, TSFIR*, TSFLAB*, TSIDEM*, TSFVIT*
# ---------------------------------- END ------------------------------------------
```

Result from Individual Studies X&Y vs. Pooled TLF Comparison 1

label	col1	col2	col3	chng_type
Adverse events	71	274	135	=
Adverse events	71	274	135	=
Adverse events by maximum intensity				=
Adverse events by maximum intensity				=
Adverse events leading to discontinuation of study agent	7	11	8	=
Adverse events leading to discontinuation of study agent	7	11	8	=
Analysis set	148	582	291	=
Analysis set	148	582	291	=
Infections	26	103	50	=
Infections	26	103	50	=
Mild	35	166	69	=
Mild	35	166	69	=
Moderate	31	89	56	=
Moderate	31	89	56	=
Serious adverse events	9	16	16	=
Serious adverse events	9	16	16	=
Serious infections	0	0	4	=
Serious infections	0	0	4	=
Severe	5	19	10	=
Severe	5	19	10	=
Subjects who died	0	0	0	=
Subjects who died	0	0	0	=
Subjects with 1 or more:				=
Subjects with 1 or more:				=

Result from Individual Study Z vs. pooled TLF Comparison 2

label	col1	col2	chng_type
Adverse events	58 (49.6%)	107 (46.5%)	=
Adverse events	58 (49.6%)	107 (46.5%)	=
Adverse events by maximum intensity			=
Adverse events by maximum intensity			=
Adverse events leading to discontinuation of study agent	3 (2.6%)	1 (0.4%)	=
Adverse events leading to discontinuation of study agent	3 (2.6%)	1 (0.4%)	=
Analysis set	117	230	=
Analysis set	117	230	=
Infections	24 (20.5%)	44 (19.1%)	=
Infections	24 (20.5%)	44 (19.1%)	=
Mild	24 (20.5%)	66 (28.7%)	=
Mild	24 (20.5%)	66 (28.7%)	=
Moderate	29 (24.8%)	38 (16.5%)	=
Moderate	29 (24.8%)	38 (16.5%)	=
Serious adverse events	9 (7.7%)	5 (2.2%)	=
Serious adverse events	9 (7.7%)	5 (2.2%)	=
Serious infections	0	1 (0.4%)	=
Serious infections	0	1 (0.4%)	=
Severe	5 (4.3%)	3 (1.3%)	=
Severe	5 (4.3%)	3 (1.3%)	=
Subjects who died	0	1 (0.4%)	=
Subjects who died	0	1 (0.4%)	=
Subjects with 1 or more:			=
Subjects with 1 or more:			=

Figure 4. Display of the HTML outputs showing a visual of the comparisons.

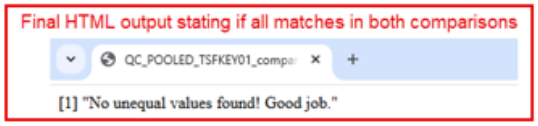

Figure 5. Display of the HTML outputs showing the final result of comparing the pooled TLF vs. its individual study TLFs.

CONCLUSION

Incorporating the power of R, this program represents an original approach towards the quality assessment of examining pooled TLFs against their individual study TLFs, offering a streamlined option that significantly reduces time and effort. This tool optimizes workflow efficiency, ultimately advancing the pace of decision-making processes within the pharmaceutical industry.

Recommendations for enhancing the examination of pooled TLFs against their corresponding individual study counterparts include the implementation of a comprehensive summary document aggregating final evaluations from each pooled TLF. Additionally, refinement of the R coding to enhance efficiency to optimize this process further as we continue to consider other tools in the pharmaceutical industry. Finally, utilizing the latest developments in R of new packages and functions available to continually develop and improve this quality assessment of pooled TLFs.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.
Contact the author at:
Author Name: Sophie Khan
Company: J&J Innovative Medicine
Address: 50-100 Holmers Farm Way, High Wycombe, Bucks, HP12 4DP, UK
Work Phone: 07784150824
Email: skhan94@its.jnj.com
Website: www.linkedin.com/in/sophina-sophie-khan-ab215975