PharmaSUG China 2024 - Paper CC-10010

Compare and batch validation in R for mixed QC

Perphy ZHAO, Sanofi;
Melody YU, Q2BI;
Michael WANG, Sanofi;
Qian SU, Sanofi

Abstract

Double programming is highly important and necessary for clinical data analysis. This paper aims to introduce several different R functions to validate for clinical data analysis, and batch validation for data or output when both R and SAS are used to validate the result in the projects. There are often many options for doing something as R is a diverse and flexible statistical programming language, and the comparison is no exception. The advantages and disadvantages of each comparison function are summarized by examples, the functions with logical errors are corrected, and finally create a customed function to batch validation of the data or output in a project to check the compare status.

Introduction

Statistical analysis of clinical trial data has always been double program, which refers to two independent programmers' programming, and then comparing and validating the results. Its importance lies in ensuring the accuracy and reliability of analytical results to eliminate human errors and enhance the scientific validity and credibility of data analysis and helps identify and rectify discrepancies, thereby minimizing the risk of false conclusions. Therefore, whatever tools used to the analysis, it should and must be double programming to ensure the statistical quality. As a shared, ideal and popular programming language in recent years, R has great strengths in statistical analysis, data wrangling and visualization, and more and more programmers in pharmaceutical industry are moving from SAS to R programming. The compare procedure in SAS is highly familiar by programmers, in order to help ones to know about comparison in R, this paper will list the advantages and disadvantages of each comparison function by examples and summary, then create a customed function to batch validation of the data or output in a project to check the compare status quickly.

Function illustration

Here will be a brief introduction for each function.

diffdf() in diffdf

Description: Compares 2 data frames and outputs any differences.

Usage:

```
diffdf(
  base,
  compare,
  keys = NULL,
  suppress_warnings = FALSE, # Suppress warnings?
  strict_numeric = TRUE, # Flag for strict numeric to numeric comparisons
  strict_factor = TRUE, # Flag for strict factor to character comparisons
  file = NULL, # Location and name of a text file to output the results to
  tolerance = sqrt(.Machine$double.eps), # Set tolerance for numeric comparisons
  scale = NULL # Comparisons fail if (x-y)/scale > tolerance
)
```

Example:

In order to check easily the difference between to data frames, we dummied two data frames by the internal data named Iris in R, as figure 1.

iris1

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	7.0	3.2	4.7	1.4	versicolor
4	6.4	3.2	4.5	1.5	versicolor
5	6.3	3.3	6.0	2.5	virginica
6	5.8	2.7	5.1	1.9	virginica

iris2

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.3	3.5	1.4	0.2	versicolor
2	4.9	3.0	1.4	0.2	setosa
3	7.0	3.2	4.7	1.4	versicolor
4	6.4	3.2	4.5	1.5	versicolor
5	6.3	3.3	6.0	2.5	virginica
6	5.8	2.7	5.1	1.9	virginica

Figure 1

The returned value is a list by diffdf(), which will be return null when two data frames are exactly matched, however, it will return a non-null list when two data frames are unmatched, including attributes, the number of difference and detail column difference between two data frames, detail information returned as figure 2.

```
> dif_diffdf1 <- diffdf(iris1,iris1)
> names(dif_diffdf1)
NULL
> length(dif_diffdf1)
[1] 0
```

```
> dif_diffdf <- diffdf(iris1,iris2)
Warning message:
In diffdf(iris1, iris2) :
Not_all_Values_Compared_Equal
> names(dif_diffdf)
[1] "NumDiff"              "VarDiff_Sepal.Length" "VarDiff_Species"
> dif_diffdf$NumDiff
# A tibble: 2 × 2
  Variable     `No of Differences`
* <chr>                      <int>
1 Sepal.Length                   1
2 Species                        1
> dif_diffdf$VarDiff_Sepal.Length
# A tibble: 1 × 4
  VARIABLE     ..ROWNUMBER..  BASE COMPARE
* <chr>               <int> <dbl>   <dbl>
1 Sepal.Length            1   5.1     5.3

> dif_diffdf <- diffdf(iris1,iris2,suppress_warnings=T)
>
```

Figure 2

The warning message above is not a real warning, which only highlight these two data frames are unmatched, surpress_warnings = TRUE can avoid this warning in the chatty box, if you would not like to this warning message, the other options can be also chosen when necessary.

compare_df() in compareDF

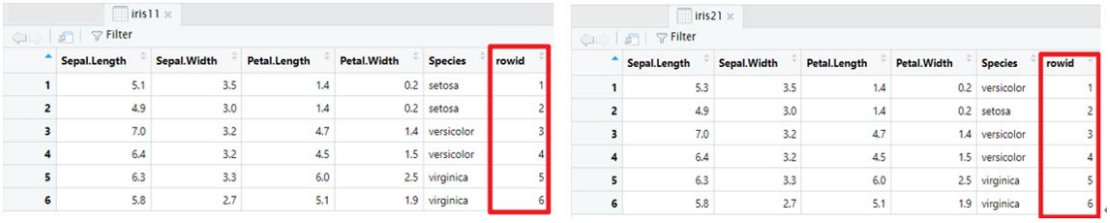
Description: Do a git style comparison between two data frames of similar columnar structure.

Usage:

```
compare_df(
  df_new,
  df_old,
  group_col,
  exclude = NULL, # The columns which should be excluded from the comparison
  tolerance = 0,
  tolerance_type = "ratio", # Type of comparison for numeric var(s), 'ratio' | 'difference'
  stop_on_error = TRUE, # Whether to stop on acceptable errors on not
  keep_unchanged_rows = FALSE, # whether to preserve unchanged values or not
  keep_unchanged_cols = TRUE, # whether to preserve unchanged values or not
  change_markers = c("+", "-", "="), # what the different change_type nomenclature should be
  round_output_to = 3 # Number of digits to round the output to
)
```

Example:

We mutated rowid variable in two data frames, which is key variable and can identify each record uniquely, as figure 3.



iris11

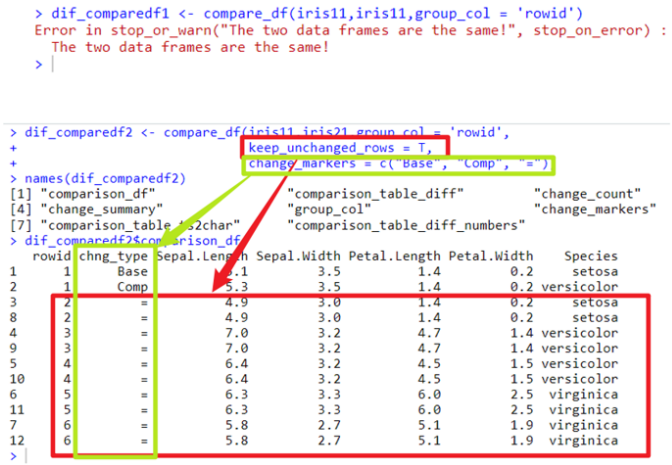
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
1	5.1	3.5	1.4	0.2	setosa	1
2	4.9	3.0	1.4	0.2	setosa	2
3	7.0	3.2	4.7	1.4	versicolor	3
4	6.4	3.2	4.5	1.5	versicolor	4
5	6.3	3.3	6.0	2.5	virginica	5
6	5.8	2.7	5.1	1.9	virginica	6

iris21

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
1	5.3	3.5	1.4	0.2	versicolor	1
2	4.9	3.0	1.4	0.2	setosa	2
3	7.0	3.2	4.7	1.4	versicolor	3
4	6.4	3.2	4.5	1.5	versicolor	4
5	6.3	3.3	6.0	2.5	virginica	5
6	5.8	2.7	5.1	1.9	virginica	6

Figure 3

The returned value is a list by compare_df(), which will return nothing, but print an error message to highlight the two data frames matched when two data frames are exactly matched, we can use tryCatch() to wrap the compare_df() to avoid the error at the time, however, it will return a non-null list when two data frames are unmatched. The detail unmatched records can be found in the element called ‘comparision_df’ of the returned list as figure 4.



```
> dif_comparedf1 <- compare_df(iris11,iris11,group_col = 'rowid')
Error in stop_or_warn("The two data frames are the same!", stop_on_error) :
  The two data frames are the same!
>
```

```
> dif_comparedf2 <- compare_df(iris11,iris21,group_col = 'rowid',
+                              keep_unchanged_rows = T,
+                              change_markers = c("Base", "Comp", "="))
> names(dif_comparedf2)
[1] "comparison_df"                "comparison_table_diff"        "change_count"
[4] "change_summary"               "group_col"                    "change_markers"
[7] "comparison_table_diff2char"   "comparison_table_diff_numbers"
> dif_comparedf2$comparison_df
   rowid chng_type Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1      1      Base          5.1         3.5          1.4         0.2     setosa
2      1      Comp          5.3         3.5          1.4         0.2 versicolor
3      2         =          4.9         3.0          1.4         0.2     setosa
8      2         =          4.9         3.0          1.4         0.2     setosa
4      3         =          7.0         3.2          4.7         1.4 versicolor
9      3         =          7.0         3.2          4.7         1.4 versicolor
5      4         =          6.4         3.2          4.5         1.5 versicolor
10     4         =          6.4         3.2          4.5         1.5 versicolor
6      5         =          6.3         3.3          6.0         2.5  virginica
11     5         =          6.3         3.3          6.0         2.5  virginica
7      6         =          5.8         2.7          5.1         1.9  virginica
12     6         =          5.8         2.7          5.1         1.9  virginica
>
```

Figure 4

The most useful information of unmatched records is contained in ‘comparision_df’, the other

elements returned can be a reference when check the difference. The options in the function can control some detail comparison result when used.

dfCompare() in dfComapre

Description: Return data frames for added, changed, and deleted records structure.

Usage:

dfCompare(dfOld,dfNew,key)

Example:

We filtered some records from two data frames to keep some different records, this is an exceptional example, there is only one record merged by the 'rowid' variable, as figure 5.

iris3

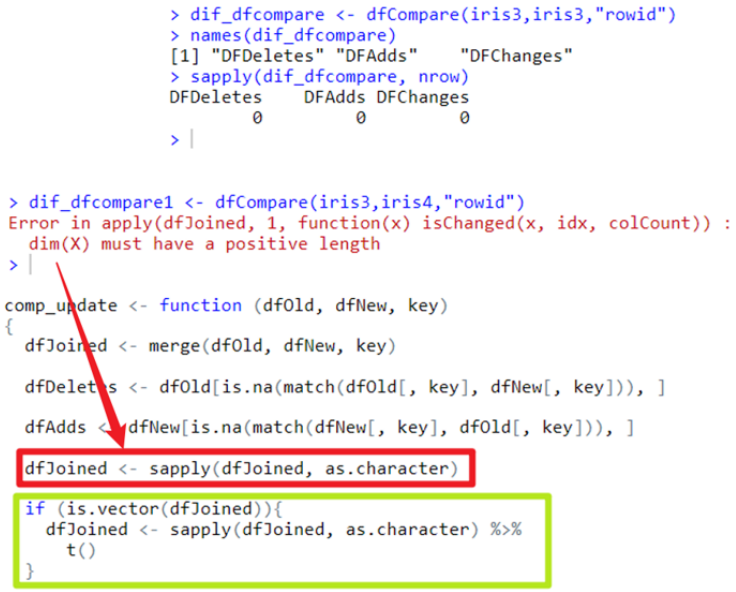
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
1	5.1	3.5	1.4	0.2	setosa	1
2	6.3	3.3	6.0	2.5	virginica	5
3	5.8	2.7	5.1	1.9	virginica	6

iris4

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
1	5.3	3.5	1.4	0.2	versicolor	1
2	7.0	3.2	4.7	1.4	versicolor	3
3	6.4	3.2	4.5	1.5	versicolor	4

Figure 5

The returned value is a list by dfCompare() whenever the two data frames are matched or unmatched. The two data frames are completely matched when the number of rows each element in the list is zero. There will be an error for only one record when the two data frames merged by key variables. The statement in the red box creates a matrix when the records are more than 1 after merging by key variables, but a vector when only 1 record after merging, so it's necessary to update this function in green box, translate the vector into matrix when only one record after merging, as figure 6.



```
> dif_dfcompare <- dfCompare(iris3,iris3,"rowid")
> names(dif_dfcompare)
[1] "DFDeletes" "DFAdds"    "DFChanges"
> sapply(dif_dfcompare, nrow)
DFDeletes    DFAdds DFChanges
        0         0         0
> |
```

```
> dif_dfcompare1 <- dfCompare(iris3,iris4,"rowid")
Error in apply(dfJoined, 1, function(x) isChanged(x, idx, colCount)) :
  dim(X) must have a positive length
> |

comp_update <- function (dfOld, dfNew, key)
{
  dfJoined <- merge(dfOld, dfNew, key)

  dfDeletes <- dfOld[is.na(match(dfOld[, key], dfNew[, key])), ]

  dfAdds <- dfNew[is.na(match(dfNew[, key], dfOld[, key])), ]

  dfJoined <- sapply(dfJoined, as.character)

  if (is.vector(dfJoined)){
    dfJoined <- sapply(dfJoined, as.character) %>%
      t()
  }
```

Figure 6

When using the updated function called comp_update() to compare two data frames, the result is as in figure 7.

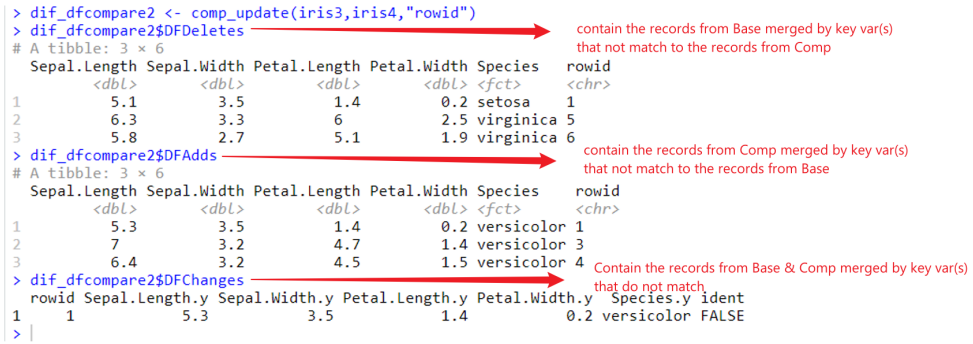


Figure 7

compareEqual() in compare

Description: Compare two objects and allow for various minor differences between them.

Usage:

```
compareEqual(model, # The "correct" object
             comparison, # The object to be compared with the model
             transform=character(), # A character vector containing any transformations
             # that have been performed on the objects prior to this comparison
             ignoreCase=FALSE,
             trim=FALSE,
             dropLevels,
             ignoreLevelOrder,
             ignoreColOrder,
             ignoreNameCase
)
```

Example:

The returned value is a list by compareEqual() whenever the two data frames are matched or unmatched, and the result is so simple that we can only know which column is matched or unmatched, as figure 8.

Figure 8

diff_data() in daff

Description: Find differences with a reference data set. To store the difference for documentation purposes using write_diff() or to visualize the difference using render_diff().

Usage:

```
diff_data(
  data_ref,
  data,
  always_show_header = TRUE,
  always_show_order = FALSE,
  ids = c(),
  ignore_whitespace = FALSE,
  never_show_order = FALSE,
  ordered = TRUE,
  padding_strategy = c("auto", "smart", "dense", "sparse"),
  show_unchanged = FALSE,
  show_unchanged_columns = FALSE
)
```

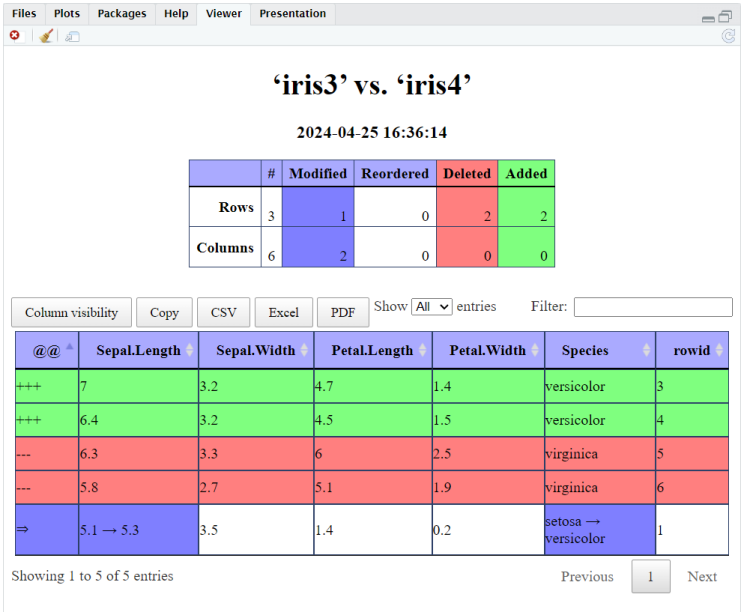
Example:

The returned value is a list by diff_data() whenever the two data frames are matched or unmatched, as figure 9, but the result need to store using write_diff() or to visualize the difference using render_diff(), here we use reander_diff() to create an interactive interface, witch easy to check the difference, as figure 10. There will be no record when the two data frames are completely matched.

```
> dif_diff_data1<- diff_data(iris3, iris3)
> names(dif_diff_data1)
 [1] "set_data"   "get_data"   "var_name"   "raw"        "to_csv"     "from_csv"
 [7] "get_matrix" "ctx"        "mode"       "is_factor"  "levels"
> render_diff(
+   dif_diff_data1,
+   file = tempfile(fileext = ".html"),
+   view = interactive(),
+   fragment = FALSE,
+   pretty = TRUE,
+   summary = TRUE,
+   use.DataTables = TRUE
+ )
>

> dif_diff_data2<- diff_data(iris3, iris4)
> names(dif_diff_data2)
 [1] "set_data"   "get_data"   "var_name"   "raw"        "to_csv"     "from_csv"
 [7] "get_matrix" "ctx"        "mode"       "is_factor"  "levels"
> render_diff(
+   dif_diff_data2,
+   file = tempfile(fileext = ".html"),
+   view = interactive(),
+   fragment = FALSE,
+   pretty = TRUE,
+   summary = TRUE,
+   use.DataTables = TRUE
+ )
>
```

Figure 9



	#	Modified	Reordered	Deleted	Added
Rows	3	1	0	2	2
Columns	6	2	0	0	0

@@	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
+++	7	3.2	4.7	1.4	versicolor	3
+++	6.4	3.2	4.5	1.5	versicolor	4
---	6.3	3.3	6	2.5	virginica	5
---	5.8	2.7	5.1	1.9	virginica	6
⇒	5.1 → 5.3	3.5	1.4	0.2	setosa → versicolor	1

Figure 10

diffdfs() in diffdfs

Description: Returns a data frame describing the modifications required to transform old_df into new_df.

Usage:

```
diffdfs(new_df,
        old_df = NA,
        key_cols = NA,
        verbose = FALSE, # Should the processing be chatty?
)
```

Example:

The returned value is a data frame. The two data frames are completely matched if the row number of returned data frame is equal to 0. However, they are unmatched when the number of rows is more than 0, as figure 11.

```
> dif_diffdfs1<- diffdfs(
+   iris3,
+   iris3,
+   key_cols = 'rowid',
+   verbose = TRUE
+ )
Computing diff dataframe...
Processing new_df...
Checking that key column rows are unique
Processing old_df...
Checking that key column rows are unique
>
```

dif_diffdfs1

Filter

operation	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
No data available in table						

```
> dif_diffdfs2<- diffdfs(
+   iris3,
+   iris4,
+   key_cols = 'rowid',
+   verbose = FALSE
+ )
>
```

dif_diffdfs2

Filter

	operation	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	rowid
1	new	6.3	3.3	6.0	2.5	virginica	5
2	new	5.8	2.7	5.1	1.9	virginica	6
3	modified	5.1	3.5	1.4	0.2	setosa	1
4	deleted	7.0	3.2	4.7	1.4	versicolor	3
5	deleted	6.4	3.2	4.5	1.5	versicolor	4

Figure 11

compare() in versus

Description: compare() creates a representation of the differences between two tables, along with a shallow copy of the tables. This output is used as the comparison argument when exploring the differences further with other versus functions e.g., slice_*() and weave_*().

Usage:

```
compare(
  table_a,
  table_b,
  by,
  allow_both_NA = TRUE,
  coerce = TRUE # If FALSE and columns from the input tables have differing classes, then throws an error
  )
```

Example:

The returned value is a list whenever the two data frames are matched or unmatched. The two data frames are completely matched if the row number of elements (intersection, matched_cols, matched_rows) returned data frame are equal to 0 after unnesting, as figure 12.

```
> dif_vsscomp1 <- versus::compare(
+   iris3,
+   iris3,
+   by='rowid'
+ )
> names(dif_vsscomp1)
[1] "tables"         "by"             "intersection"   "unmatched_cols" "unmatched_rows"
[6] "input"
> sapply(dif_vsscomp1, nrow, simplify = T)
$tables
[1] 2

$by
[1] 1

$intersection
[1] 5

$unmatched_cols
[1] 0

$unmatched_rows
[1] 0

$input
NULL

> dif_vsscomp1$intersection %>%
+   unnest(diff_rows)
# A tibble: 0 × 6
# … with 6 variables: column <chr>, n_diffs <int>, class_a <chr>, class_b <chr>, row_a <int>,
#   row_b <int>
>
```

```
> dif_vsscomp1$intersection
# A tibble: 5 × 5
  column       n_diffs class_a class_b diff_rows
  <chr>          <int> <chr>   <chr>   <list>
1 Sepal.Length       0 numeric numeric <tibble [0 × 2]>
2 Sepal.Width        0 numeric numeric <tibble [0 × 2]>
3 Petal.Length       0 numeric numeric <tibble [0 × 2]>
4 Petal.Width        0 numeric numeric <tibble [0 × 2]>
5 Species            0 factor  factor  <tibble [0 × 2]>
>
```

Figure 12

We can also use the silice_*(), value_*() and weave_*() functions to translate the list returned to the data frames to check the difference, as figure 13.

```
> dif_vsscomp2 <- versus::compare(
+   iris3,
+   iris4,
+   by='rowid'
+ )
> # slice_diffs(dif_vsscomp2,'b',everything())
> slice_unmatched_both(dif_vsscomp2)
# A tibble: 4 × 7
  table rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <chr> <chr>        <dbl>       <dbl>        <dbl>       <dbl> <fct>
1 a     5              6.3         3.3          6           2.5 virginica
2 a     6              5.8         2.7          5.1         1.9 virginica
3 b     3              7           3.2          4.7         1.4 versicolor
4 b     4              6.4         3.2          4.5         1.5 versicolor
> value_diffs_stacked(dif_vsscomp2, column = everything())
ℹ Columns converted to character: val_a, val_b
# A tibble: 2 × 4
  column       val_a  val_b      rowid
  <chr>        <chr>  <chr>      <chr>
1 Sepal.Length 5.1    5.3        1
2 Species      setosa versicolor 1
> weave_diffs_long(dif_vsscomp2,everything())
# A tibble: 2 × 7
  table rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <chr> <chr>        <dbl>       <dbl>        <dbl>       <dbl> <fct>
1 a     1              5.1         3.5          1.4         0.2 setosa
2 b     1              5.3         3.5          1.4         0.2 versicolor
>
```

Figure 13

rCompare() in dataCompareR

Description: Compare two data frames (or objects coercible to data frames) and produce a dataCompareR object containing details of the matching and mismatching elements of the data.

Usage:

```
rCompare(
  dfA,
  dfB,
  keys = NA,
  roundDigits = NA,
  mismatches = NA, # The max number of mismatches to assess
  trimChars = FALSE
)
```

Example:

The returned value is a list whenever the two data frames are matched or unmatched. We can use summary() to check the compare result, which is highly similar with result created by compare procedure from SAS, as figure 14.

```
> dif_dcompr1 <- dataCompareR::rCompare(iris3,iris3, keys = 'rowid')
Running rCompare...
Coercing input data to data.frame
> names(dif_dcompr1)
[1] "meta"         "colMatching"  "rowMatching"  "cleaninginfo" "mismatches"   "matches"
> summary(dif_dcompr1)
dataCompareR is generating the summary...
Data Comparison
===============

Date comparison run: 2024-04-27 13:15:28
Comparison run on R version 4.2.1 (2022-06-23)
With dataCompareR version 0.1.4


Meta Summary
============


|Dataset Name |Number of Rows |Number of Columns |
|:------------|:--------------|:-----------------|
|iris3        |3              |6                 |
|iris3        |3              |6                 |


Variable Summary
================

Number of columns in common: 6
Number of columns only in iris3: 0
Number of columns only in iris3: 0
Number of columns with a type mismatch: 0
Match keys : 1   - ROWID

Row Summary
===========

Total number of rows read from iris3: 3
Total number of rows read from iris3: 3
Number of rows in common: 3
Number of rows dropped from iris3: 0
Number of rows dropped from  iris3: 0


Data Values Comparison Summary
==============================

Number of columns compared with ALL rows equal: 5
Number of columns compared with SOME rows unequal: 0
Number of columns with missing value differences: 0

Columns with all rows equal : PETAL.LENGTH, PETAL.WIDTH, SEPAL.LENGTH, SEPAL.WIDTH, SPECIES

>
```

Figure 14

We can also translate the list returned to the data frames to check and save the difference, as figure 15.

```
> dif_dcompr2 <- dataCompareR::rCompare(iris3,iris4, keys = 'rowid')
Running rCompare...
Coercing input data to data.frame
> names(dif_dcompr2)
[1] "meta"         "colMatching"  "rowMatching"  "cleaninginfo" "mismatches"   "matches"
> (mismatches <- as.data.frame(do.call(rbind,dif_dcompr2$mismatches)))
             ROWID valueA     valueB     variable     typeA     typeB diffAB
SEPAL.LENGTH     1    5.1        5.3 SEPAL.LENGTH    double    double   -0.2
SPECIES          1 setosa versicolor      SPECIES character character
> (rowMatching <- as.data.frame(do.call(rbind,dif_dcompr2$rowMatching)) %>%
+   mutate(
+     indata=rownames(as.data.frame(do.call(rbind,dif_dcompr2$rowMatching)))
+   ))
          ROWID    indata
matchKeys ROWID matchKeys
inboth        1    inboth
inA.1         5     inA.1
inA.2         6     inA.2
inB.1         3     inB.1
inB.2         4     inB.2
> (colMatching <-  as.data.frame(do.call(cbind,dif_dcompr2$colMatching)))
        inboth
1 PETAL.LENGTH
2  PETAL.WIDTH
3         ROWID
4 SEPAL.LENGTH
5  SEPAL.WIDTH
6      SPECIES
>
```

Figure 15

Batch vallidation

Prepare the reflist

For reflist, the following columns must be contained in reflist.

pgmname	filename	tit CSR	Programmer Name	Internal Reviewer Name	QC program Name	SAS or R
ran batch 1	ran batch 1	Listing of participants receiving i	Icey Yu	Perphy ZHAO	v ran batch 1	R
ran randoscheme 1	ran randoscheme 1	Participant Randomization List	Icey Yu	Perphy ZHAO	v ran randoscheme 1	R
dis dispo r t	dis dispo r t	Participant disposition	Perphy Zhao	Qian SU	v dis dispo r t	R
dis ana pop a t	dis ana pop a t	Analysis populations	Perphy Zhao	Qian SU	v dis ana pop a t	R

pgmname: leading program name

filename: output name, must be unique

tit_CSR: title for each output, must be unique

Programmer Name: leader programmer name

Internal Reviewer Name: QC programmer name

SAS or R: indicate whether the output is validated by SAS or R

Individual QCprogram

For individual TLG, QCer should save two permanent data prefixed with 'v_' and 'd_' whichever programming language you choose, dem_demo_r_t as an example as below:

For SAS:

```
proc compare base=prod compare=qcd.v_dem_demo_r_t out=qcd.d_dem_demo_r outbase outcomp outnoeq outdiff ;
run;
```

For R: use save() function to save the two data.

```
save(final, file = file.path(MISCQCD, paste('v_',filename,'.RData'))) # R data

save(dif, file = paste0(MISCQCD,"/d_",filename,".RData")) # R data
```

Compare function

diffdf() function in diffdf package and compare_df() function in compareDF package to compare individual TLG. Sample code as below

```
# Compare and review the comparison ----------------------------------------
dif_i <- diffdf(base, final,strict_numeric = FALSE, strict_factor = FALSE,
                suppress_warnings = TRUE)
if (length(dif_i) == 0) {
    dif <- dif_i %>%
        unlist() %>%
        as.data.frame() %>%
        rownames_to_column()
} else {
    (dif0 <- compare_df(
        df_new = final,
        df_old = base,
        group_col = c('col0'),
        exclude = NULL,
        tolerance = 0,
        tolerance_type = "difference",
        stop_on_error = FALSE,
        keep_unchanged_rows = FALSE,
        keep_unchanged_cols = FALSE,
        change_markers = c("qc", "base", "="),
        round_output_to = 3
    )  )
    # Convert to data frame ---------------------------------------------------
    dif1 <- dif0 %>%
        unlist() %>%
        as.data.frame() %>%
        rownames_to_column()
    # Save comparison output --------------------------------------------------
    dif <- as.data.frame(dif0$comparison_df)
}
```

Batch validation function

Check the result of the validation of each output simply.

```r
dif_chk <- function(x=NULL){
  if (file.exists(file.path(rqcfilepath, paste0('d_',trimws(x),'.RData')))){

    x_temp <- load(file.path(rqcfilepath, paste0('d_',trimws(x),'.RData')))

    if (nrow(assign(x, eval(parse(text = x_temp))))>0){
      return('Not pass')
    }else{
      return('Pass')
    }
  }
  else if (file.exists(file.path(sasqcfilepath, paste0('d_',trimws(x),'.sas7bdat')))){
    x_temp <- haven::read_sas(file.path(sasqcfilepath, paste0('d_',trimws(x),'.sas7bdat')))

    if (nrow(x_temp)>0){
      return('Not pass')
    }else{
      return('Pass')
    }
  }
  else{
    return('Not compare')
  }
}
```

```r
data_dif <- reflist %>%
  mutate(
    `compare status temp`=lapply(filename, dif_chk)
  )
```

Get the information of programs and outputs by leading & QC programmer.

```r
get_info <- function(x=NULL ,filepath=NULL){
  file_info <- file.info(file.path(filepath, trimws(x))) %>%
    mutate(
      file=x,
    )
  row.names(file_info) <- NULL
  return(file_info)
}
```

```r
pgm_info <- reflist %>%
  select(filename,tit_CSR,pgmname,`QC program Name`,`SAS or R`) %>%
  distinct(tit_CSR,.keep_all = T) %>%
  filter(!is.na(pgmname)) %>%
  mutate(
    info_group=lapply(paste0(pgmname,'.sas'), get_info,filepath=pgmpath)
  ) %>%
  unnest(info_group) %>%
  rename_with(~paste0('pgm_',.x),-c(filename,pgmname,tit_CSR))
```

Check the result by the file information completely.

```r
final_excel <- reduce(list(data_dif, pgm_info,file_info,qcpgm_info,qcfile_v_info,qcfle_d_info),
                      left_join,by=c('filename','pgmname','tit_CSR')) %>%
  rowwise() %>%
  mutate(
    `program status`=case_when(
      is.na(pgm_ctime) ~ 'Not start',
      is.na(file_ctime) ~ 'Programming',
      max(file_ctime,file_mtime)<max(pgm_ctime,pgm_mtime) ~'Not meet time logic, Please re-run',
      .default = 'Ready for compare'
    ),
    `qc program status`=case_when(
      is.na(qcpgm_ctime) ~ 'Not start',
      is.na(qcvfile_ctime) | is.na(qcdfile_ctime) ~ 'Programming',
      max(qcdfile_ctime,qcdfile_mtime,qcvfile_ctime,qcvfile_mtime)<max(qcpgm_ctime,qcpgm_mtime) ~
        'Not meet time logic, Please re-run',
      .default = 'Ready for compare'
    ),
    `compare status`=case_when(
      min(qcdfile_ctime,qcdfile_mtime,qcvfile_ctime,qcvfile_mtime)>max(file_ctime,file_mtime) &
        `compare status temp`=='Pass' ~`compare status temp`,
      min(qcdfile_ctime,qcdfile_mtime,qcvfile_ctime,qcvfile_mtime)<=max(file_ctime,file_mtime) &
        `compare status temp`=='Pass' ~ 'Pass, but QC-time<=Source-time, Please re-run',
      is.na(qcdfile_ctime) | is.na(file_ctime) ~ 'Not compare',
      .default = 'Not pass'
    )
  )
```

Pgm Create time	Pgm Midify time	File Create time	File Midify time	QC Pgm Create time	QC Pgm Modify time	QC file Create time	QC file Modify time	Validation file Create time	Validation file Modify t	program status	qc program status	compare status
4/14/2024 0:38:21	4/7/2024 15:25:13									Programming	Not start	Not compare
4/14/2024 0:38:20	4/7/2024 15:37:39	4/17/2024 3:42:42	4/17/2024 3:42:42	4/14/2024 0:38:20	4/11/2024 3:42:39	4/18/2024 6:52:57	4/18/2024 6:52:57	4/18/2024 6:52:57	4/18/2024 6:52:57	Ready for compare	Ready for compare	Pass
4/14/2024 0:38:20	4/7/2024 1:09:27	4/15/2024 7:01:03	4/15/2024 7:01:03	4/14/2024 0:38:20	4/7/2024 6:15:04	4/15/2024 7:19:58	4/15/2024 7:19:58	4/15/2024 7:19:58	4/15/2024 7:19:58	Ready for compare	Ready for compare	Pass
4/14/2024 0:38:21	4/3/2024 3:48:52	4/15/2024 7:00:54	4/15/2024 7:00:54	4/14/2024 0:38:20	4/6/2024 6:18:07	4/15/2024 7:19:49	4/15/2024 7:19:49	4/15/2024 7:19:49	4/15/2024 7:19:49	Ready for compare	Ready for compare	Pass

Function parameters illustration

```
validation_tlg(
    reflistname = 'reflist.xlsx', # reflist name
    pgmpath = REPP, # leading program path
    filepath = REPD, # leading tlg data path
    rqcpgmpath = MISCQCP, # R QC program path
    sasqcpgmpath = QCP, # SAS QC program path
    rqcfilepath = MISCQCD, # R QC data path
    sasqcfilepath =QCD, # SAS QC data path
    validationfilepath = QCO, # validation file path
    validationfile = 'R_validation' # validation file name
)
```

Potential error

R is case-sensitive, please pay attention to the columns' name when create reflist.

Reference

[1] compareDF: Do a Git Style Diff of the Rows Between Two Data frames with Similar Structure (r-project.org)

[2] dataCompareR: Compare Two Data Frames and Summarise the Difference (r-project.org)

[3] dfCompare: Compare Two Data frames and Return Adds, Changes, and Deletes (r-project.org)

[4] dataCompareR: Compare Two Data Frames and Summarise the Difference (r-project.org)

[5] compare: Comparing Objects for Differences (r-project.org)

[6] diffdfs: Compute the Difference Between Data Frames (r-project.org)

[7] diffdf: Dataframe Difference Tool (r-project.org)

[8] versus: Compare Data Frames (r-project.org)

ACKNOWLEGEMENTS

The authors would like to take this opportunity to thank Sanofi and my team for facilitating and supporting this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors at:

Perphy ZHAO
perphyzhao@163.com

Melody YU
yjlcq2009@163.com

Michael WANG
michael3.wang@sanofi.com

Qian SU
Qian.su@sanofi.com