PharmaSUG China 2024 - Paper DV-10003

# Alluvial and Sankey Plots for Clinical Data Analysis – Implementation in SAS, R and Python

Ke Yu, Pfizer (China) Research and Development Center

## ABSTRACT

Alluvial and Sankey plots are both useful data visualization tools. They have some superficial features in common but internally they have different use. Alluvial plot shows how a population of facts is allocated across categorical dimensions. While Sankey plot can exhibit the quantity of flows from one stage to another. Both can also be used in data analysis of clinical trials. E.g., present the onset and severity change of adverse events of special interest, check the balance of characteristics between test and control arms at baseline.

Here we will look back on the different solutions to implement such figures in SAS ®, including a set of SAS ® macros, HTML output by JSON and STREAM procedure, and using SAS ® Visual Analytics.

With the arrival of open-source language era, the arsenal of statistical programmers has been greatly augmented. Both R and Python delivered impressive performances in this circumstance. In R language area, we would like to introduce a 'ggplot2' extensional package 'ggalluvial'. While for Python programming part, we will introduce a powerful data visualization library 'plotly'. These tools can help us to generate alluvial and Sankey plots nice and easy. The output can have a lot of user interactive functions, which is different with the traditional static figure. We will also share the application of such techniques in our daily work in pharmaceutical industry.

## INTRODUCTION

### SANKEY PLOT

The Sankey plot was firstly created by an Irish captain Henry Riall Sankey in 1898 to show how the energy flows among components of a steam engine. Then this plot named after him was widely used to present flows move through difference stages. A Sankey plot consist of 3 types of information: Nodes cluster observations in the same flow on a given stage. Flows are the connections between different nodes, showing where the flow goes. The width of flow is often scaled by the number of observations. Stages generally represent the status in time sequence.

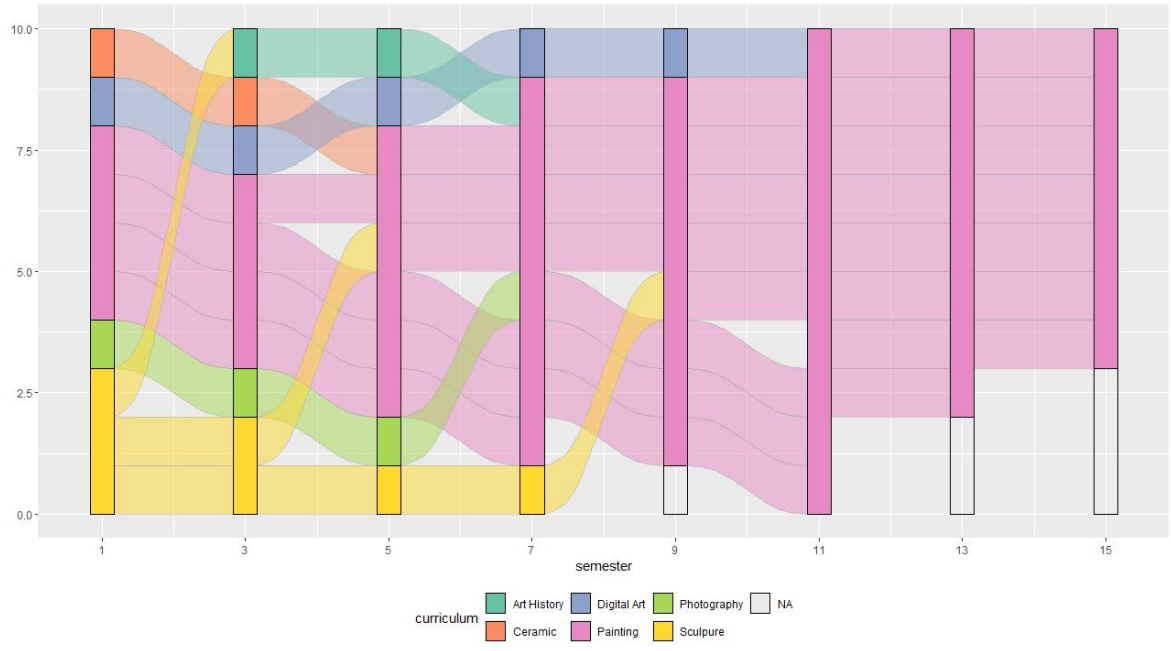

**Figure 1. Sankey Plot - Students Curricula Across Semesters**

## ALLUVIAL PLOT

The alluvial plot appeared much later, which was introduced in a paper in 2010 to map the citation changes across scientific fields. But then the alluvial plot was used to show proportions of observations across different categories. Visually, an alluvial plot is made up of stacked bars, and between those bars are links. Each set of stacked bars represents a categorical field. The separations within the stacked bars represent different categories. The links between the bars conveys the set of observations shared between one categorical field and the next.

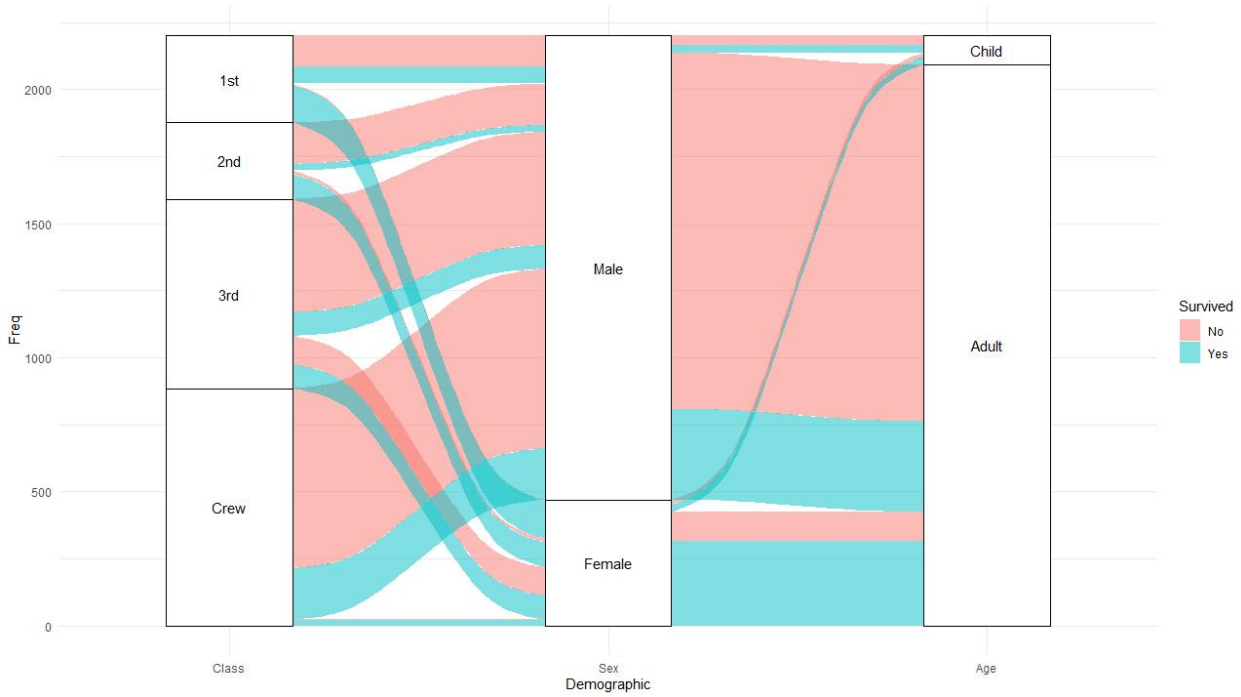


**Figure 2. Alluvial Plot - Survival Rate of Titanic Passengers by Demographics**

## SIMILARITIES AND DIFFERENCES

On first glance, the Sankey plot and alluvial plot are almost the same in the superficial features. Both have bars and links with varying widths. But the logics are different in kernel. An alluvial plot is used to convey information about categories, whereas a Sankey plot is used to convey information of flows.

In a Sankey plot, the order of stages does matter, since there is a logical sequencing to how one observation moves from one stage to the next. (e.g., In figure 1, a student must complete the course in the semester 1 and 2 before starting the semester 3) While in an *alluvial plot, the order of the* stacked bars does not have special meaning, as they just represent different categories. (e.g., In figure 2, not must to put Sex before Age and vice versa)

## DATA STRUCTURE

There are two kinds of data structure to make Sankey and alluvial plot:

1. Wide (alluvia) format: The wide format reflects the visual arrangement of an alluvial plot. Each row corresponds to a cohort of observations that take a specific value at each variable, and each variable has its own column. May have an additional column contains the quantity of each row, e.g., the number of observational units in the cohort, which may be used to control the heights of the strata.

| Class | Sex | Age | Survived | Freq |
|---|---|---|---|---|
| 1st | Female | Adult | No | 4 |
| 2nd | Female | Adult | No | 13 |
| 3rd | Female | Adult | No | 89 |
| Crew | Female | Adult | No | 3 |
| 1st | Male | Child | Yes | 5 |
| 2nd | Male | Child | Yes | 11 |
| 3rd | Male | Child | Yes | 13 |
| Crew | Male | Child | Yes | 0 |

**Display 1. Long data format**

2. Long (lode) format: The long format contains one row per node and can be understood as the result of "pivoting" the axis columns of a dataset in the wide format into a key-value pair of columns encoding the axis as the key and the stratum as the value. This format requires an additional indexing column that links the rows corresponding to a common cohort.

| Survived | Freq | alluvium | Demographic | stratum |
|---|---|---|---|---|
| Yes | 140 | 29 | Class | 1st |
| Yes | 80 | 30 | Class | 2nd |
| Yes | 76 | 31 | Class | 3rd |
| Yes | 20 | 32 | Class | Crew |
| No | 0 | 1 | Sex | Male |

**Display 2. Wide data format**

Some of the macro and packages below provide special functions to convert raw data between these structures. E.g. function *to_lodes_form()* and *to_alluvia_form()* in R package ggalluvial.

## SAS

There is no SAS ® procedure to generate Sankey and alluvial plots directly. However, since GTL (Graph Template Language) was introduced into SAS ® ODS graphics, some new features in SGPLOT procedure can help user to create elements of such figures. E.g., the bar segment can be drawn with HIGHLOW statement and BAND statement can create the part of flows. In PharmSUG 2015, Shane Rosanbalm published a macro %SankeyBarChart, in which he integrated them together to create longitudinal bar charts with Sankey-style overlays.

Next year in SESUG 2016, Yichen Zhong further improved this macro to give an intuitive solution to handle missing/terminated data, which is common in medical industry. From then on, this macro was widely used and performed steadily. You can download the source code and related papers from Github repository.

Figure 3 was plotted using SAS ® macro %SankeyBarChart. Data is sourced from a research about the switch of therapy regimens among patients with small cell lung cancer. Patients might not appear in all lines of therapy even with the assumption of no data missed. For instance, a patient was continuing first-line therapy since no progression observed at the time point of analysis. Or a patient discontinued from first-line therapy and then dead. So no chance to receive second-line treatment. The bald parts on the right sides of the rectangles represent these cases.

```
%sankeybarchart
   (data=adcm, subject=USUBJID, yvar=CMDECOD,
    yvarord=%quote(Crizotinib,Docetaxel,Nivolumab,Pemetrexed),
    xvar=CMREFID, barwidth=0.4, completecases=no, stat=percent);
```

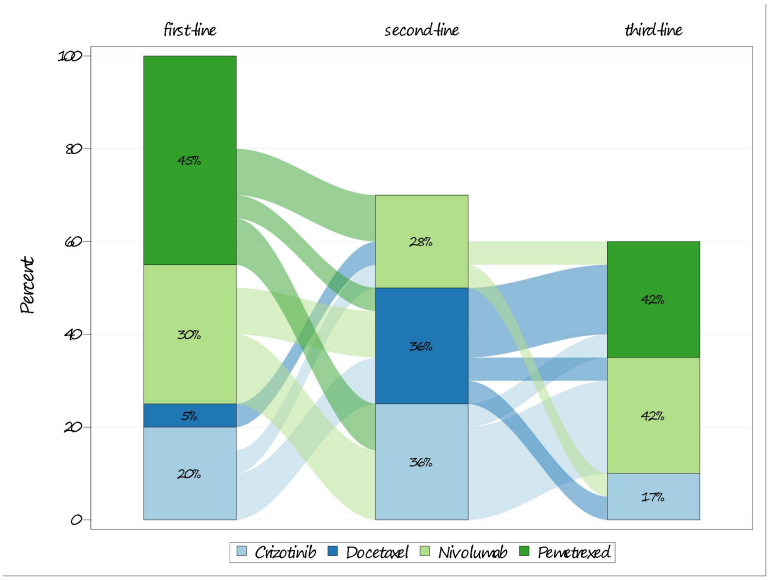


**Figure 3. Pattern of Treatment Changes for Small Cell Lung Cancer Patients**

Besides, there are other different solutions. In PharmSUG 2018, Anni Weng and Huanxue Zhou shared their exploration to generate web-based Sankey plot in HTML format by JSON and STREAM procedure. Additionally, SAS® Visual Analytics is a complete platform for data visualization and exploration. Sankey plot was included in it to do pathway analysis.

## R

ggplot2 may be the most popular data visualization package in R programming world. However, like in SAS, there is no internal function to directly draw alluvial plot and Sankey plot in basic ggplot2 package. Benefited by the open-source ecosystem of R language, several extension packs were developed by the enthusiastic R users. Among them, ggalluvial is a good choice. This is a ggplot2 extension that means you can use almost every piece of tooling available through ggplot2 seamlessly as well. Meanwhile, ggalluvial follows the same grammar and conventions of ggplot2. So it's really friendly to the traditional R user who will have a smooth learning curve.

### DRAW AE PROFILE USING GGALLUVIAL

This is a example of Sankey plot using ggalluvial. Data is from a clinical trial about an immunotherapy for cancer. According to study design, patients received study drug every week. The severity of cytokine release syndrome (CRS) was assessed according to the ASTCT CRS Consensus Grading system. In the interim analysis, the authority has a concerned about the prevalence of CRS. So we prepared Sankey plots for total patients and by dose levels.

The R data frame before graphing, which was manipulated from ADaM dataset ADAE and ADCM, has the structure as below. Column ASTCTGR is the CRS grade. For patient discontinued from treatment before week 6 dose, the value of ASTCTGR was imputed as “No Dose” for the untreated planned weeks after discontinuation. Tocilizumab, a direct anti-cytokine drug, was used to treat CAR-T associated CRS in this study. Study team want to present the CRS outcome after treatment of Tocilizumab. So, for these cases, we derived the value of ASTCTGR as “Grade n + Tocilizumab”.

| | COHORT | WEEK | ASTCTGR | SUBJID |
|---|---|---|---|---|
| 1 | Dose Level 3 | 1 | GRADE 1 | 10031001 |
| 2 | Dose Level 3 | 2 | GRADE 0 | 10031001 |
| 3 | Dose Level 3 | 3 | GRADE 0 | 10031001 |
| 4 | Dose Level 3 | 4 | GRADE 0 | 10031001 |
| 5 | Dose Level 3 | 5 | GRADE 0 | 10031001 |
| 6 | Dose Level 3 | 6 | GRADE 0 | 10031001 |

**Display 3. Data Structure for Drawing the CRS Sankey Plot**

This is the R code to generate Figure 4. The skeleton of ggplot was still used. The ggalluvial specific aesthetic elements, stratum and alluvium, have been integrated into the argument *mappings = aes()* together with traditional aesthetics x and fill. The ggalluvial function *geom_stratum* plotted rectangles for nodes (represent different CRS grades here), while *geom_flow* drew the flows between nodes on adjacent stages (mean week n treatment here). *geom_text* is a ggplot2 function. Now it can accept new stat argument stratum, which helps to display the proportion of patients with CRS onset for each grade. In brief, you can find the ggluvial package have been deeply incorporated into the ggplot2 framework and shared the same tidyverse grammar, data structure and underlying design.

```
ggplot(data, aes(x=WEEK, stratum=ASTCTGR, alluvium=SUBJID, fill=ASTCTGR)) +
  scale_x_discrete(expand=c(.1,.1)) +
  scale_y_continuous(breaks=seq(0, 18, 1)) +
  scale_fill_brewer(type = "qual", palette = "Set3") +
  geom_stratum() +
  geom_text(stat = "stratum", size=3,
    aes(label = after_stat(paste0(round(prop*100, 0), "%")))) +
  theme_minimal()+
  theme(legend.position="bottom") +
  guides(fill = guide_legend(nrow = 1)) +
  geom_flow(stat="alluvium", lode.guidance="forward", color="black") +
  labs(y='Number of participants', x='Week', fill='ASTCT Grade: ')
```

By adding category of “No Dose”, figure 4 provided information on the patients who did not complete 6 weeks treatment. The denominator is the total patients received week 1 dose. Different with figure 3, this is another alternative way to handle the censored data.

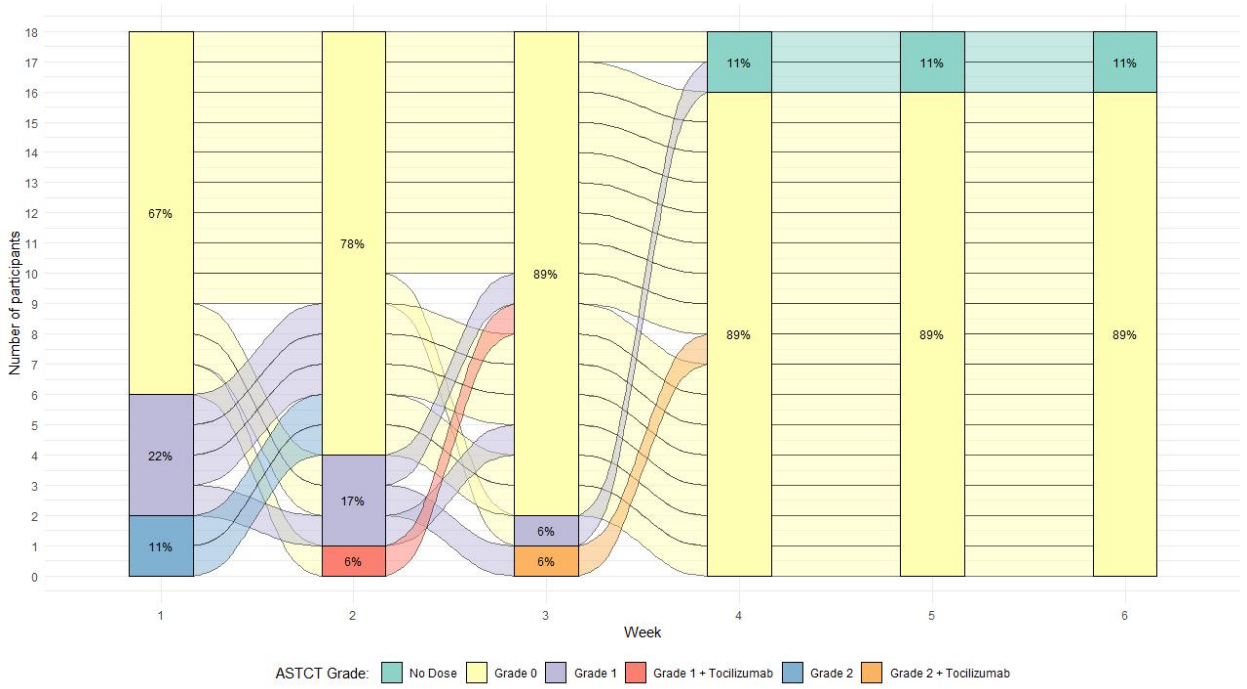


**Figure 4. CRS Profile in the First 6 Weeks for Patients in Total**

Since ggalluvial returns the same object of ggplot2, it's easy to just call ggplot2 function *facet_wrap* to split the plot into the panels of dose levels (figure 5).

```
...
  facet_wrap(~COHORT, nrow=3) +
  theme(strip.text = element_text(face = "bold"))
...
```

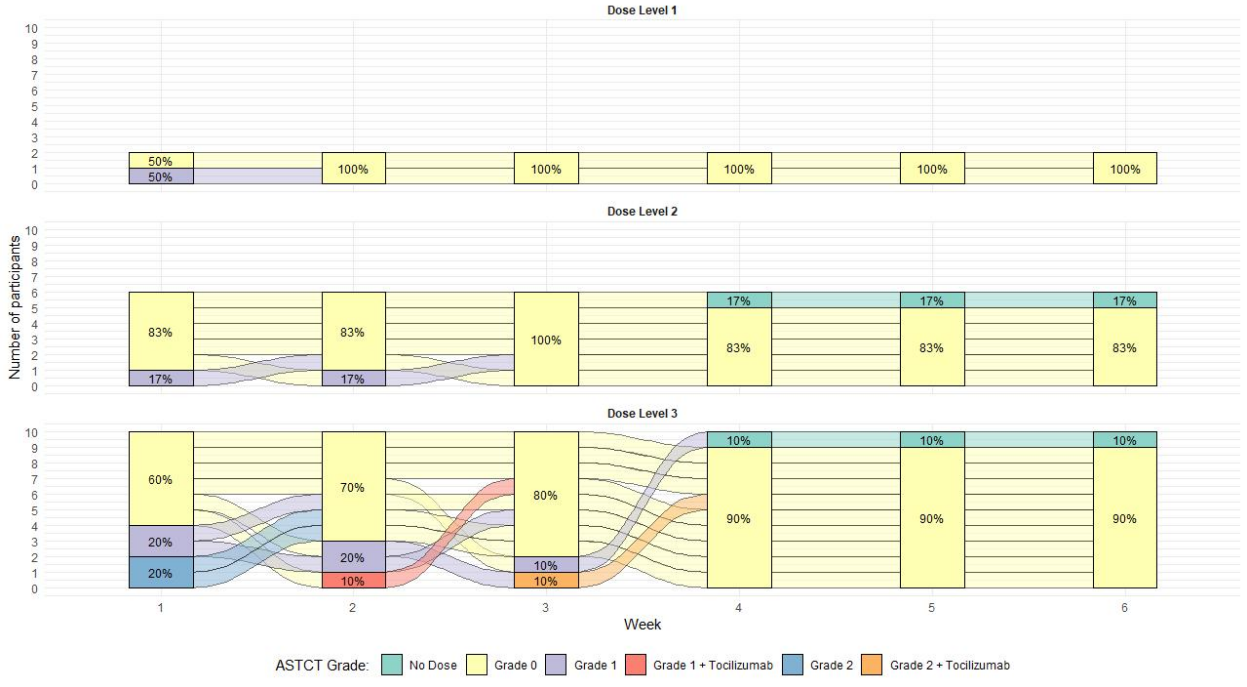

**Figure 5. CRS Profile in the First 6 Weeks by Dose Level**

## ADD INTERACTIVE ELEMENTS IN SHINY APP

The above mentioned 2 figures are both static. Let's create a Shiny app to make it more interactive. Our aim is to add a pop-up tooltip which appears when the user hovers over elements of the plot. The tooltips can display more detailed information to give a visually cleaner and better interpretation. However, the ggalluvial package does not natively include this functionality. We need use some other packages. The code was shown below with explanations in R comments the main steps.

```
library(ggalluvial)
library(shiny)
library(htmltools)
library(sp)

# Offset, in pixels, for location of tooltip relative to mouse cursor,
# in both x and y direction.
offset <- 0
# Width of node boxes
node_width <- 1/5
# Width of alluvia
alluvium_width <- 1/8

# Draw plot and extract coordinates
p <-...
  geom_flow(stat="alluvium", lode.guidance="forward", color="black",
            width = alluvium_width) +
  geom_stratum(width = node_width) +
...

# Build the plot.
pbuilt <- ggplot_build(p)

# Use built plot data to recalculate the locations of the flow polygons:

# Add width parameter, and then convert built plot data to xsplines
data_draw <- transform(pbuilt$data[[1]], width = alluvium_width)
groups_to_draw <- split(data_draw, data_draw$group)
group_xsplines <- lapply(groups_to_draw, data_to_alluvium)

# Convert xspline coordinates to grid object.
xspline_coords <- lapply(
  group_xsplines,
  function(coords) grid::xsplineGrob(x = coords$x, y = coords$y,
                                     shape = coords$shape, open = FALSE)
)

# Use grid::xsplinePoints to draw the curve for each polygon
xspline_points <- lapply(xspline_coords, grid::xsplinePoints)

# Define the x and y axis limits in grid coordinates (old) and plot
# coordinates (new)
xrange_old <- range(unlist(lapply(
  xspline_points,
  function(pts) as.numeric(pts$x)
)))
yrange_old <- range(unlist(lapply(
```

```
  xspline_points,
  function(pts) as.numeric(pts$y)
)))
xrange_new <- c(1 - alluvium_width/2, max(pbuilt$data[[1]]$x) +
alluvium_width/2)
yrange_new <- c(0, sum(pbuilt$data[[2]]$count[pbuilt$data[[2]]$x == 1]))

# Define function to convert grid graphics coordinates to data coordinates
new_range_transform <- function(x_old, range_old, range_new) {
  (x_old - range_old[1])/(range_old[2] - range_old[1]) *
    (range_new[2] - range_new[1]) + range_new[1]
}

# Using the x and y limits, convert the grid coordinates into plot
coordinates.
polygon_coords <- lapply(xspline_points, function(pts) {
  x_trans <- new_range_transform(x_old = as.numeric(pts$x),
                                 range_old = xrange_old, range_new =
xrange_new)
  y_trans <- new_range_transform(x_old = as.numeric(pts$y),
                                 range_old = yrange_old, range_new =
yrange_new)
  list(x = x_trans, y = y_trans)
})

# User interface
ui <- fluidPage(
  fluidRow(tags$div(
    style = "position: relative;",
    plotOutput("alluvial_plot", height = "600px",
               hover = hoverOpts(id = "plot_hover")
    ),
    htmlOutput("tooltip")))
)

server <- function(input, output, session) {

  # Draw plot
  output$alluvial_plot <- renderPlot(p, res = 200)

  # Display tooltip
  output$tooltip <- renderText(
    if(!is.null(input$plot_hover)) {
      hover <- input$plot_hover
      x_coord <- round(hover$x)

      if(abs(hover$x - x_coord) < (node_width / 2)) {
        # Display node information if cursor is over a stratum box.

        # Determine stratum name from x and y coord, and the n.
        node_row <- pbuilt$data[[2]]$x == x_coord &
          hover$y > pbuilt$data[[2]]$ymin &
          hover$y < pbuilt$data[[2]]$ymax
        node_label <- pbuilt$data[[2]]$stratum[node_row]
        node_n <- pbuilt$data[[2]]$count[node_row]

        # Render tooltip
```

```
        renderTags(
          tags$div(
            node_label, tags$br(),
            "n =", node_n,
            style = paste0(
              "position: absolute; ",
              "top: ", hover$coords_css$y + offset, "px; ",
              "left: ", hover$coords_css$x + offset, "px; ",
              "background: gray; ",
              "padding: 3px; ",
              "color: white; "
            )
          )
        )$html
      } else {
        # Display flow information if cursor is over a flow polygon: what
        # alluvia does it pass through?

        # Calculate whether coordinates of hovering cursor are inside one
of the
        # polygons.
        hover_within_flow <- sapply(
          polygon_coords,
          function(pol) point.in.polygon(point.x = hover$x,
                                         point.y = hover$y,
                                         pol.x = pol$x,
                                         pol.y = pol$y)
        )
        if (any(hover_within_flow)) {
          # Find the alluvium that is plotted on top. (last)
          coord_id <- rev(which(hover_within_flow == 1))[1]
          # Find the strata labels and n corresponding to that alluvium in
the data.
          flow_label <- paste(groups_to_draw[[coord_id]]$stratum, collapse
= ' -> ')
          flow_label <- paste(unique(groups_to_draw[[coord_id]]$alluvium),
flow_label, collapse = ': ')
          flow_n <- groups_to_draw[[coord_id]]$count[1]

          # Render tooltip
          renderTags(
            tags$div(
              flow_label, tags$br(),
              "n =", flow_n,
              style = paste0(
                "position: absolute; ",
                "top: ", hover$coords_css$y + offset, "px; ",
                "left: ", hover$coords_css$x + offset, "px; ",
                "background: gray; ",
                "padding: 3px; ",
                "color: white; "
              )
            )
          )$html
        }
      }
    }
```

```
  )
}

shinyApp(ui = ui, server = server)
```

The result is shown in figure 6. There are two kinds of tooltips. When the cursor is hovering over a stratum, the stratum name and the count n are presented. While when the cursor is moved to a flow, a label is popped up displaying the patient ID and severity changes in the entire 6 weeks. This update can facilitate the readers to review the CRS change of each patient separately.

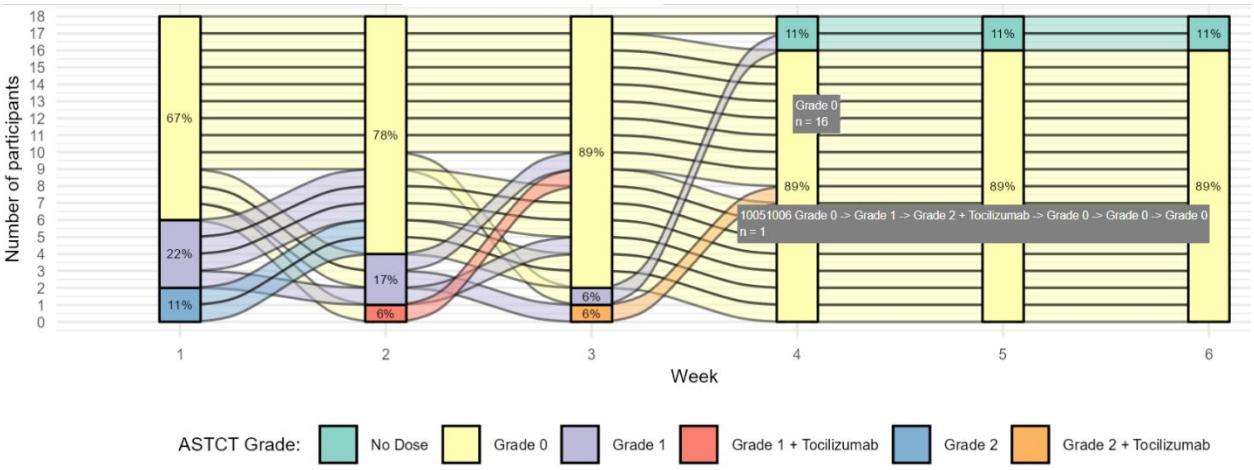


**Figure 6. Tooltips for CRS Profile in Shiny App**

## ANOTHER VARIANT SANKEY BUMP PLOT

In the previous cases, the y-axis always represents small n or proportion. Putting a continuous variable to y-axis, ggalluvial allow user to produce another variant Sankey bump plot. In this case, the strata contain no more information than the alluvia and often are removed. The width of alluvia represents the y-variable while height represents the ranking order. So when a group becomes larger than another, it bumps above it. This technique can be used in the analysis of biospecimen data. Let's take a look at the example from an early phase vaccine study. The aim is to compare the diversity and seasonal dynamics of microbiotas related to lower respiratory infection (LRI). 127 tracheal aspirates across four consecutive meteorological seasons (quarters) were collected from 40 patients, of whom 20 developed LRIs (YLNR) and 20 remained healthy (NLNR). After 16S rRNA-based gene sequencing, the taxonomic composition of the microbiome was obtained together with the abundances.

| | genera | Quarter | LRICAT | abundant |
|---|---|---|---|---|
| 1 | Acinetobacter | 1 | YLNR | 132 |
| 2 | Acinetobacter | 1 | NLNR | 29537 |
| 3 | Acinetobacter | 2 | YLNR | 3685 |
| 4 | Acinetobacter | 2 | NLNR | 10655 |

**Display 4. Data of the Microbial Taxonomic Results for Sankey Bump Plot**

The most abundant genera in microbiome were filtered out to draw the Sankey bump plot. In the R code below, the key steps are putting the abundances to y-axis and using alluvium with different color to represent each genus.

```
ggplot(data, aes(x = Quarter, y = abundant, alluvium = genera)) +
  geom_alluvium(aes(fill = genera, colour = genera),
                alpha = .75, decreasing = FALSE) +
  scale_x_continuous(breaks = seq(1, 4, 1)) +
  scale_fill_brewer(type = "qual", palette = "Set3") +
  scale_color_brewer(type = "qual", palette = "Set3") +
  facet_wrap( ~ LRICAT, scales = "fixed") +
  labs(fill = 'Pathogenic bacteria', color = 'Pathogenic bacteria') +
  theme(axis.text.y = element_blank(),
        axis.title.y = element_blank())
```

The final output exhibited the variation of genera abundances between microbiotas. In both YLRI and NLRI patients, *Streptococcus* are almost the dominant genus especially in summer and fall. By comparison between YLRI and NLRI patients, we noticed that the abundances of *Neusseria* from YLRIs is significantly higher than that from NLRI patients. It implies the potential association with LRI for this genus.

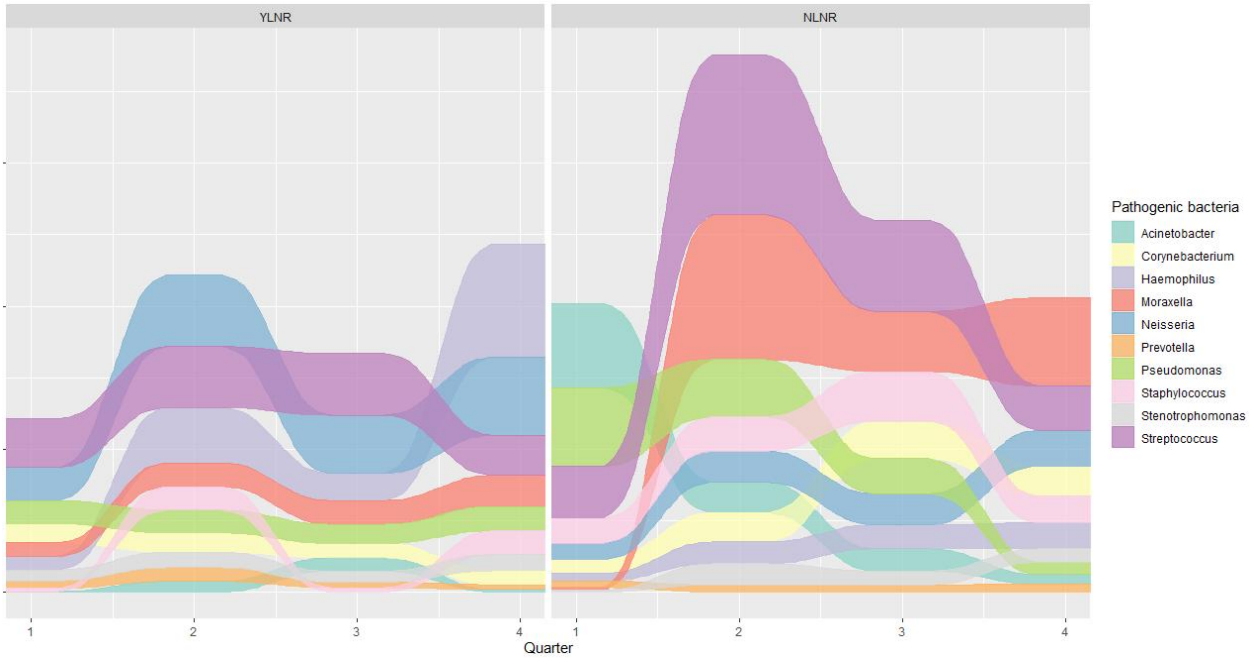


**Figure 7. Sankey Bump Plot of Abundances of Pathogenic Bacteria by Quarter**

## OTHER USEFUL R PACKAGES

ggsankey is another potential option. Similar to ggalluvial, it also developed upon the ggplot2 system. But at this time, only development version is available. And it does feel like this package requires some more polishing.

Besides the two packages mentioned above, networkD3 is a mature package to create D3 network graphs (including Sankey plot) in the htmlwidgets framework. The most impressive feature is the interactive output.

The cross-platform package plotly offers the similar interactive figures. We will further discuss it in the section of Python.

# PYTHON

Python is another powerful open-source language with a huge number of fans around the world. Supported by several plotting library, such as matplotlib and seaborn, Python has wonderful performance in data visualization. Here we will introduce the hottest library plotly and show the application in clinical data visualization.

Built on top of the plotly JavaScript library, plotly in Python enables users to create hight-quality, interactive, web-based graphs, which can be displayed in Jupyter notebooks, saved to standalone HTML files, or served as part of pure Python-built web applications using Dash.

## A MONITOR OF LIVER INJURY BASED ON PLOTLY

The data frame below is from a study about drug-induced liver injury (DILI). Column ALT_H and TBili_H are the peak serum alanine transaminase (ALT) and total bilirubin respectively, divided by their upper limit of normal range. The rest 3 columns are the potential risk factors: AGEGR1 is the age category. bimcat is the category of BMI index. tobacco is the tobacco usage history. The gene polymorphism of Cytochrome P450 is another important risk factor of DILI. But we did not include it in the plot to simplify the illustration.

| | SUBJID | ALT_H | TBili_H | AGEGR1 | bmicat | tobacco |
|---|---|---|---|---|---|---|
| **0** | 10011002 | 0.477612 | 0.348039 | 18-44 | <18.5 | Never |
| **1** | 10011003 | 0.567164 | 0.426471 | 45-64 | 24-<28 | Never |
| **2** | 10021002 | 2.590361 | 0.780000 | 45-64 | 18.5-<24 | Former |
| **3** | 10021003 | 1.089552 | 0.290476 | 45-64 | 18.5-<24 | Never |
| **4** | 10021004 | 1.566265 | 0.509524 | 45-64 | 24-<28 | Never |

**Display 5. Data of Lab Tests for Liver Function and Other Risk Factors**

This is the Python code to generate figure 8. The *on_selection* and *on_click* callbacks were used to implement linked brushing between the alluvial plot and the scatter plot. Firstly, *go.Scatter* and *go.Parcats* were called to create objects of scatter plot and alluvial plot (named as parallel categorical diagram in plotly) respectively. Then, in order for the callback functions to be executed, wrap the two figures together to build a FigureWidget object. Next, call *update_layout* to set format and define the mode of drag and hover interactions by mouse. Finally, define a function to control the color changes in interaction.

Benefited from the inherently elegant syntax, Python code is clear at a glance. The chaos caused by the nested parentheses are common in R code, which often make other programmer (even the author himself a few days later) frustrating and maddening. Unlike R, Python's dot-separated syntax makes itself much easier to understand and maintain.

```
import plotly.graph_objects as go
from ipywidgets import widgets
import pandas as pd
import numpy as np

adsl = pd.read_csv(r'C:\Users\...\adsl.csv')

# Build parcats dimensions
dim = ['AGEGR1', 'bmicat', 'tobacco']
label = ['Age', 'BMI', 'Tobacco Usage']

dimensions = ([dict(values=adsl[dim], label=label)
               for dim, label in zip(dim, label)])


# Build colorscale
```

```
color = np.zeros(len(adsl), dtype='uint8')
colorscale = [[0, 'gray'], [1, 'firebrick']]

# Build figure as FigureWidget
fig = go.FigureWidget(
    data=[go.Scatter(x=adsl.altmhi, y=adsl['tbilimhi'],
                     marker={'color': 'gray'}, mode='markers',
                     selected={'marker': {'color': 'firebrick'}},
                     unselected={'marker': {'opacity': 0.3}}),
          go.Parcats(domain={'y': [0, 0.4]}, dimensions=dimensions,
                     line={'colorscale': colorscale, 'cmin': 0, 'cmax': 1,
                           'color': color, 'shape': 'hspline'})
    ])

fig.update_layout(height=800, xaxis={'title': 'Peak Serum ALT(/ULN)'},
                  yaxis={'title': 'Peak Total Bilirubin(/ULN)',
                         'domain': [0.5, 1]},
                  dragmode='lasso', hovermode='closest')

# Update color callback
def update_color(trace, points, state):
    # Update scatter selection
    fig.data[0].selectedpoints = points.point_inds

    # Update parcats colors
    new_color = np.zeros(len(adsl), dtype='uint8')
    new_color[points.point_inds] = 1
    fig.data[1].line.color = new_color

# Register callback on scatter selection...
fig.data[0].on_selection(update_color)
# and parcats click
fig.data[1].on_click(update_color)

fig
```

The web-based output is interactive. The reader can drag categories horizontally to reorder dimensions as well as vertically to reorder categories within the dimension. When hovering mouse on an interested flow or category, a tooltip popped up displaying count and percentage information. Furthermore, it allows the reader to select the interested data points by box or lasso in the scatterplot, meanwhile the corresponding flows will be highlighted in red in the alluvial plot to display the categorical distribution. As shown below, patients with peak ALT > 3 × ULN were selected in the scatter plot part. That's the potential Temple's Corollary range and Hy's Law range suggesting cholestatic liver injury and severe DILI respectively. At the same time, risk factors' distribution of these patients was highlighted in the alluvial plot. According to which subgroups the patients fall in, investigators can estimate the potential risk and prepare the target therapeutic regimen.

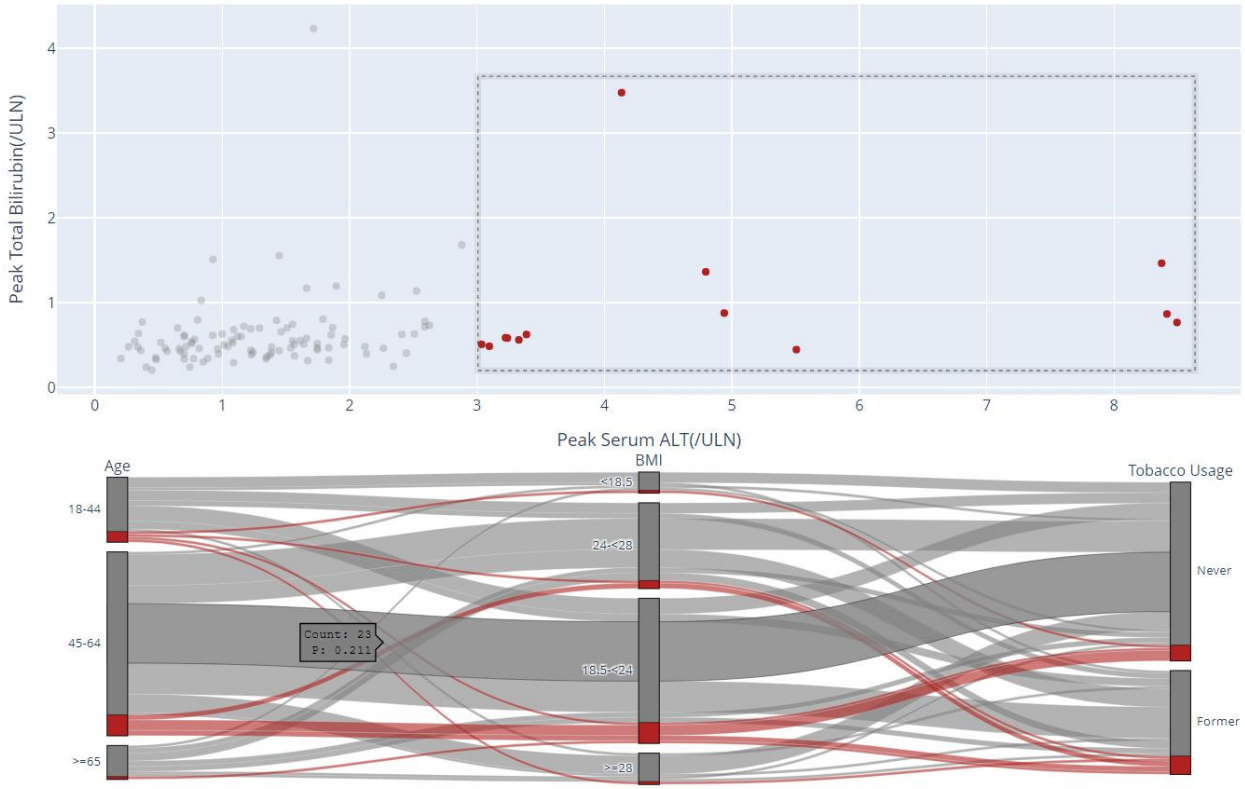


**Figure 8. eDISH Plot Linked Risk Factors by Bushing**

Besides of go.Parcats,  plotly have go.Sankey to make Sankey plot.

## THE FULL PICTURE OF A RHEUMATOID ARTHRITIS STUDY

The aim of this study was to investigate sustained remission following the discontinuation of methotrexate in patients with rheumatoid arthritis (RA). RA disease activity was defined by Clinical Disease Activity Index (CDAI), which is a questionnaire assessed on scheduled visits (week 0 to 52, and week 52 to 104). Scores of 22.1–76.0 are labelled as high activity, 10.1–22.0 as moderate activity, 2.9–10.0 as low activity and 0.0–2.8 as remission. At week 52, patients who achieved remission then stop to receive the treatment of methotrexate. In the following weeks, the RA activity was observed among these patients.

Display 6 is the data after our manipulation. The levels of RA activity have been categorized into high, moderate, low and remission. Patients with ID 33 and 35 were not achieved remission at week 52. The data after then was censored.

| Subject ID | Week 0 | Week 1 | Week 12 | Week 26 | Week 38 | Week 52 | Week 56 | Week 64 | Week 78 | Week 90 | Week 104 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | Moderate | Moderate | Moderate | Remission | Remission | Low | nan | nan | nan | nan | nan |
| 34 | Moderate | Moderate | Low | Low | Low | Remission | Moderate | Moderate | Moderate | Moderate | Moderate |
| 35 | Moderate | Moderate | Low | Low | Low | Low | nan | nan | nan | nan | nan |
| 36 | Moderate | Moderate | Low | Low | Low | Remission | Low | Low | Low | Low | Moderate |
| 37 | Moderate | Moderate | Low | Low | Low | Remission | Remission | Moderate | Moderate | Moderate | Moderate |
| 38 | Moderate | Moderate | Low | Low | Low | Remission | Remission | Low | Moderate | Moderate | Moderate |

**Display 6. Data Structure of Categorized CDAI for RA by Weeks**

In plotly, Sankey plot is defined by three lists: source, target, and values. Just as their name implies, source and target represent the source node and target node respectively. And the list of values set the flow volume. Then plotly indexes each node using a number starting from 0 to the total number of nodes

minus one. The code to call *go.Sankey* is really simple. Actually, the hardest part is configuring these three lists. Unlike the ggalluvial package in R, plotly in Python does not provide any function to convert input data into long format to fit Sankey plot. As the code shown below, we used a large section to deal with the dataframe with multiple visits in columns.

```
import plotly.graph_objects as go
import pandas as pd

df = pd.read_csv(r'C:\User\...\RA.csv')

# all the combinations of 11 visits
groups = df.groupby(['Week 0','Week 1','Week 12','Week 26','Week 38','Week
52','Week 56','Week 64','Week 78','Week 90','Week 104'],
dropna=False).agg({'Subject ID':'count'}).reset_index()

# all the combinations of results between 2 consecutive visits
week_0 = groups.groupby(['Week 0','Week 1']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_1 = groups.groupby(['Week 1','Week 12']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_12 = groups.groupby(['Week 12','Week 26']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_26 = groups.groupby(['Week 26','Week 38']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_38 = groups.groupby(['Week 38','Week 52']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_52 = groups.groupby(['Week 52','Week 56']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_56 = groups.groupby(['Week 56','Week 64']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_64 = groups.groupby(['Week 64','Week 78']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_78 = groups.groupby(['Week 78','Week 90']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})
week_90 = groups.groupby(['Week 90','Week 104']).agg({'Subject
ID':'sum'}).reset_index().rename(columns={'Subject ID':'counts'})

#list_ contains all these dataframes
list_=[week_0,week_1,week_12,week_26,week_38,week_52,week_56,week_64,week_7
8,week_90]

names = [] # all nodes, i.e Low_V2 = low RA activity at visit 2.
count_dict = {} # value list
source_list = [] # source list
target_list = [] # target list

for i in range(0, len(list_)):
   cols =list_[i].columns
   for x,y,z in zip(list_[i][cols[0]],list_[i][cols[1]],list_[i][cols[2]]):
   #Iterates over x(source),y(target),z(counts)
            names.append(x+'_V'+str(i+1))
            count_dict[x+'_V'+str(i+1),y+'_V'+str(i+2)] =z
            source_list.append(x+'_V'+str(i+1))
            target_list.append(y+'_V'+str(i+2))

for v in target_list:
```

```
    if v not in names:
        names.append(v)

# the index for every nodes
all_numerics = {}
for i in range(0,len(names)):
    all_numerics[names[i]] = i


# color for nodes
color_dict =
{'High':'rgba(252,65,94,0.7)','Moderate':'rgba(255,162,0,0.7)','Low':'rgba(
55,178,255,0.7)', 'Remission':'turquoise'}
# color for flows.
color_dict_link = {'High':'
rgba(252,65,94,0.4)','Moderate':'rgba(255,162,0,0.4)','Low':'rgba(55,178,25
5,0.4)', 'Remission':'paleturquoise'}


# Draw Sankey plot
fig = go.Figure(data=[go.Sankey(
    arrangement = 'perpendicular',
    node = dict(
        thickness = 5, align="left",
        line = dict(color = None, width = 0.01),
        label = [x.split('_')[0] for x in names],
        #Adding node colors,have to split to remove the added suffix
        color = [color_dict[x.split('_')[0]] for x in names],),
    link = dict(
        #use all_numerics to transform labels to index
        source = [all_numerics[x] for x in source_list],
        target = [all_numerics[x] for x in target_list],
        #Use count_dict to get value for each link
        value = [count_dict[x,y] for x,y in zip(source_list,target_list)],
        #Adding link colors,have to split to remove the added suffix
        color = [color_dict_link[x.split('_')[0]] for x in target_list]
        ),)])

# Adding title, size, margin
fig.update_layout(title_text="<b>Sankey plot describing proportion of
patients per CDAI disease activity category over time</b><br>Methotrexate
arm",
                  font_size=15,width=1200,height=800,
margin=dict(t=210,l=90,b=20,r=30),
                  showlegend=True, xaxis_visible=False, yaxis_visible=False,
plot_bgcolor='white'
                  )

for x_coordinate, column_name in
enumerate(["0W","1W","12W","26W",'38W','52W','56W','64W','78W','90W','104W']
):
    fig.add_annotation(
          x=x_coordinate,#Plotly recognizes 0-5 to be the x range.
          y=1.075,#y value above 1 means above all nodes
          xref="x",
          yref="paper",
          text=column_name,#Text
```

```
            showarrow=False,
            font=dict(
                family="Tahoma",
                size=16,
                color="black"
                ),
            align="left",
            )

fig.show()
```

Again, the Plotly produced output is interactive. The nodes are free to move (though meaningless here). As shown in figure 9, hovering the cursor on the node, the reader can get several information: 22 patients with low RA level at week 26. They came from 3 different groups at week 12 and then were distributed into 4 groups at week 52. While if the reader hovers the cursor on a flow, the corresponding source, target and value are displayed, e.g., one patient had RA worsen from remission to moderated activity at week 64.

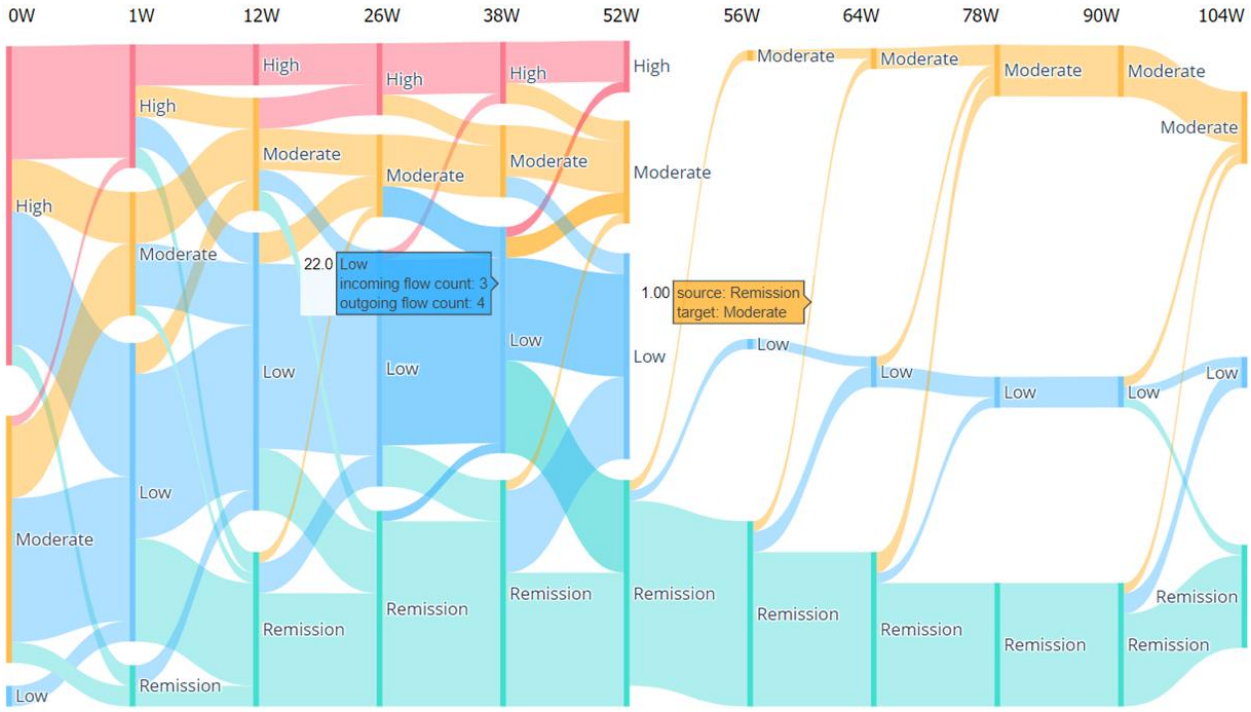


**Figure 9. Sankey plot present the change of RA Activity in Treatment and discontinuation Period**

## CONCLUSION

Alluvial and Sankey plots have special advantages in categorical and longitudinal data visualization. This paper discussed the similarities and differences between them and provided the solutions to generate such plots in SAS, R and Python environment. We also summarized the relative merits and the most suitable scenarios for these 3 languages in table 1. Readers can choose anyone according to their actual demands even mix them together in programming.

| SAS | R | Python |
|---|---|---|
| The algorithm is validated and accepted by authorities in a long history. | Free, open-source language.<br>Is enough to produce the static figures.<br>Can be added in Shiny | Free, open-source language.<br>Particularly good at complex interactive figures.<br>Elegant syntax. |

| SAS | R | Python |
|---|---|---|
| | application | Low code amount.<br>Can be added in Dash application |

**Table 1. Comparison among SAS, R and Python in Data Visualization**

For instance, to generate the figure 9, we spent a large amount of code to develop a function in Python to extract the key lists of “source”, “target” and “value” from the raw data. Actually, R package ggsankey have a ready-make function *make_long* to achieve the same goal.

| Week.0 | Week.1 | Week.12 | Week.26 | Week.38 | Week.52 | n |
|---|---|---|---|---|---|---|
| High | Low | Moderate | Low | Remission | Low | 2 |

source target value

| x | node | next_x | next_node | value |
|---|---|---|---|---|
| Week.0 | High | Week.1 | Low | 2 |
| Week.1 | Low | Week.12 | Moderate | 2 |
| Week.12 | Moderate | Week.26 | Low | 2 |
| Week.26 | Low | Week.38 | Remission | 2 |
| Week.38 | Remission | Week.52 | Low | 2 |
| Week.52 | Low | Week.56 | | 2 |

```
library(ggsankey)
weeks <- names(df)[1:6]
df %>%
  make_long(weeks, value=n)
```

**Display 7. Data Structure Transformation using *make_long* Function in ggsankey**

## REFERENCES


Shane Rosanbalm, Ryan Bailey. “A set of SAS macros for creating longitudinal bar charts with Sankey-style overlays”. GitHub. 2016-12-07. Available at https://github.com/RhoInc/sas-sankeybarchart.

Anni Weng, Huanxue Zhou. 2018. “Graphic Visualization for Treatment Switch Pattern” PharmSUG 2018, Paper DV-16.

Jason Cory Brunson and Quentin D. Read. “Alluvial Plots in ggplot2 • ggalluvial”. GitHub. 2023-02-03. Available at https://corybrunson.github.io/ggalluvial/.

David Sjoberg, Henning Lenz, Richard Latham, Salim B. “Make sankey, alluvial and sankey bump plots in ggplot”. GitHub. 2024-04-04. Available at https://github.com/davidsjoberg/ggsankey.

Plotly Open Source Graphing Libraries. “Parallel Categories Diagram in Python”. Plotly. 2024. Available at https://plotly.com/python/parallel-categories-diagram/.


## ACKNOWLEDGMENTS


I would like to thank my manager, Zhang, Wei (Tony) and Wang, Man (Maggie), for reviewing this paper and providing valuable comments and suggestions. They also guide me on journey of clinical programming and provided inspirations for my little try in new languages.


## CONTACT INFORMATION

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

Ke Yu
Pfizer (China) Research and Development Center
13817763834
ke.yu@pfizer.com