PharmaSUG China 2023 - Paper AP-149

Extract Data from Graph with R

Dan Zhao, Jiale Zhao, Pfizer (China) Research and Development Co.,Ltd. China

ABSTRACT

Over the last few years, R has been more popular in pharmaceutical industry. For clinical trial reporting, the results must be verified to be consistent across tables, listings and figures. For all the listings and summary tables, the results can be compared in HTML or RTF format. However, for the graphs, manually checking would be needed on figures to affirm the results are consistent with in corresponding source datasets, listings, or summary tables, which would lower the efficiency and accuracy.

To overcome the deficiency, a program has been developed in this paper, by using R, to extract data from graphs using image recognition, which could improve the efficiency and accuracy from naked eye recognition. The goal could be achieved by using R packages, “imager” and “dbscan” cooperate with frequently used analysis packages “ggplot”, “tidyverse”, etc. Besides, the presentation would introduce another method on the way to generate interactive graphs, by using R package “plotly” and “highcharter”, which would be easier to get the background data sets on the graph. To further simplify, package “jsonlite” would be introduced for batching extracting JSON data precisely from interactive graphs.

INTRODUCTION

In pharmaceutical industries, the results of clinical trial reports must be verified to be consistent across tables, listings and figures, where tables and listings can be easily compared in HTML or RTF format, by using software, but for figures, the comparing would be more complicated. The background data needs to be compared, and manual check would be needed to affirm all the data points are presented correctly on the graphs, which would lower the efficiency and accuracy. In this case, a way to extract data from graphs could fill in to improve the efficiency and accuracy.

In this paper, three ways of extracting data from graphs, by using different R packages, would be introduced:

1. “imager” and “dbscan” cooperate with frequently used analysis packages.

   In this method, graphs would be imported as a ‘cimg’ and then be transformed into a data frame, which contains the images information, including positions and colors.

   For line graph, the morphological closing by a square element would be used to erode/dilate the line and only the scatter plot would be kept.

   Point pixels and coordinate axis could be located by analyzing the dataset. After that, clustering would be used to further confirm the points’ values.

2. “plotly” and “highcharter” can be used to convert a ggplot2::ggplot() object to the interactive graphs, which would be easier to check the background data sets on the graph.

3. The R package “jsonlite” would be introduced for batching extracting JSON data precisely from interactive graphs. The interactive plot would be converted to jsonedit firstly, then transform to json string, finally the precisely original data can be derived through data tidying.

The three methods have their own advantages and limitations and could be used in different situations.

Audience would need to have basic knowledge for data cleaning and visualization with R.

METHOD 1. EXTRACT DATA FROM GRAPHS BY USING IMAGE RECOGNITION

There are different types of graphs can be used for analysis. In method 1, the two most frequently used types of graphs, scatter plot and line plot, have been investigated. Two kinds of frequently used themes, gray background and black frame, have been studied. The graphs used were plotted from the built-in datasets from R, “imager”, “dbscan” and some frequently used R packages have been used in this method.

TYPE 1: BLACK AND WHITE SCATTER PLOT

The process of extracting data will be shown in 7 steps.

Step 1: Import Graph

Graph need to be imported and assigned to an object with class “cimg”, which could be done by using load.image(). The code could be written as following:

```
library(imager)
img <- imager::load.image("myplot.png")
plot(img)
```

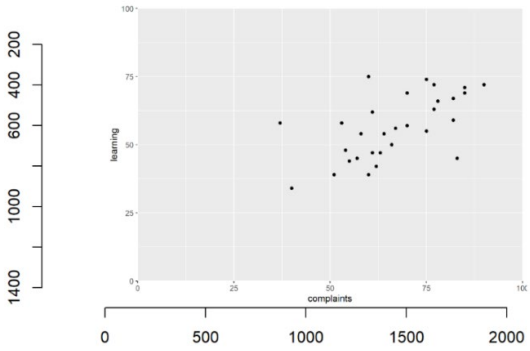


Figure 1. Graph Loaded

Step 2: Flip the Image

From Figure 1, Y-axis has been noted running downwards, which is different from the traditional coordinate system. Therefore, the figure would need to be flipped to make the directions of the pixel coordinate and origin coordinate the same, which could be realized by using mirror(), where the first argument is object and the second is axis:

```
flip_img <- imager::mirror(img,"y")
plot(flip_img)
```

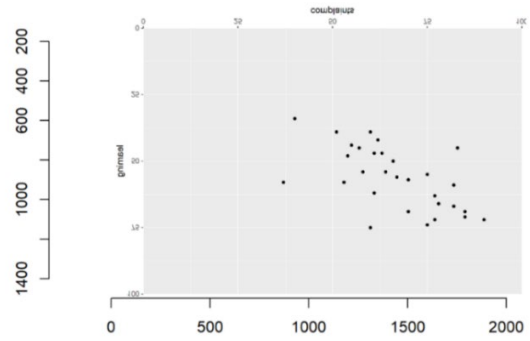


Figure 2. Flipped Image

Step 3: Transform the "cimg" Object to Data Frame

The figure is now ready to be converted into data frame by using function as.data.frame(). All the information of the graph, including coordinates and color, is stored in the data frame. The data frame will include four variables, where x and y to describe the location of the pixel, cc is color channel and value (darkness) to describe color:

```
library(dplyr)
library(tidyverse)
df_img <- as.data.frame(flip_img)
```

Step 4: Crop Out the Image

To compute the X and Y coordinates of the points, the main graph needs to be cropped out. Therefore, the X and Y coordinates can be calculated from the pixel coordinates by scaling.

To crop out the main graph, the elements of the graph need to be investigated. In this figure, the way to recognize the main graph is to identify the gray background. Then the edge of main graph, maximum and minimum of X-axis and Y-axis could be determined. Besides, the gray background should have the most frequent "value". Therefore, frequency of the values has been calculated and compared. The most frequent value, the gray color, has been used to filter out the gray point pixels. To get the edge of the main graph, minimum and maximum of both x and y values have been calculated and used to crop the flipped image. The result has been shown in Figure 3

```
library(plyr)
freq <- plyr::count(df_img, "value")
freq1 <- freq %>%
  summarize(freq=max(freq)) %>%
  left_join(freq)
v <- freq1[1,"value"]

df_img_edge <- df_img %>%
  dplyr::filter(value==v) %>% #filter out background
  summarise(minx=min(x),
            maxx=max(x),
            miny=min(y),
            maxy=max(y)) #get edge
minx <- df_img_edge[1,"minx"] + 0 #+0 is to keep numeric
maxx <- df_img_edge[1,"maxx"] + 0
miny <- df_img_edge[1,"miny"] + 0
maxy <- df_img_edge[1,"maxy"] + 0

cimg3 <- imsub(flip_img, minx <= x & x <= maxx, miny <= y & y <= maxy)
plot(cimg3)
```

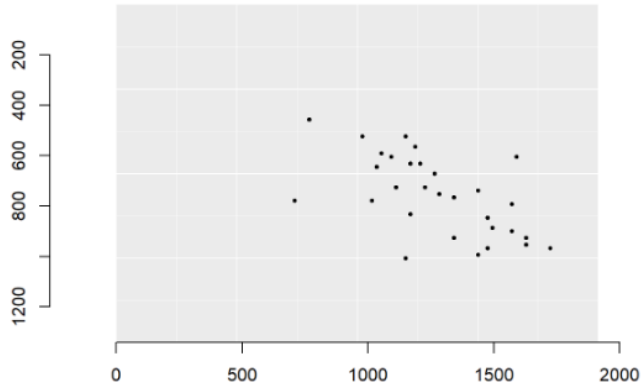


Figure 3. Cropped Graph

Step 5: Filter Out Black Pixels

The main graph has been cropped out. The data points needed to be extracted are formed by gathering of black pixel points. Therefore, the main graph needs to be converted into data fame, and black pixel points need to be filtered out for further analysis. The column “value” describes the darkness of the pixel point: the smaller the number is, the darker the point will be with range of 0 - 1. Therefore, black pixel points would have value 0 or a small value close to 0:

```
df_img2 <- as.data.frame(cimg3) %>%
  filter(value==0) %>%
  select(x,y)

ggplot(data = df_img2) + geom_point(mapping = aes(x = x, y = y))
```

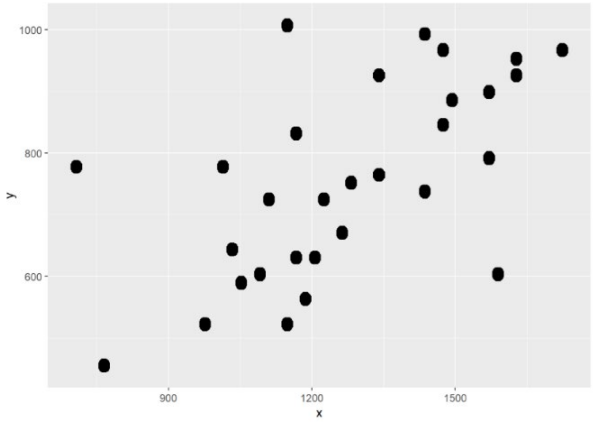


Figure 4. Graph for Black Pixels

Step 6: Cluster Pixels

As shown in the Figure 4, each black point consists of lots of pixel point. We use the R package “dbscan” for clustering, which is the density-based spatial clustering algorithm using a kd-tree. The X and Y coordinates of the black points can be derived by calculating the average of each cluster:

```
library(dbscan)
df_matrix <- as.matrix(df_img2)
dbscan_res <- dbscan::dbscan(df_matrix,eps = 2,minPts = 10)
```

```
df_img2$cluster_label <- dbscan_res$cluster
ggplot(data = df_img2) +
   geom_point(mapping = aes(x = x, y = y, color=cluster_label))

df_cluster <- df_img2 %>%
   group_by(cluster_label) %>%
   summarize(mean_x=mean(x), mean_y=mean(y))
df_cluster_c <- df_cluster %>%
   mutate(mean_x1=(mean_x)/(maxx-minx)*100) %>%
   mutate(mean_y1=(mean_y)/(maxy-miny)*100)

ggplot(df_cluster_c,aes(mean_x1,mean_y1)) + geom_point()
```

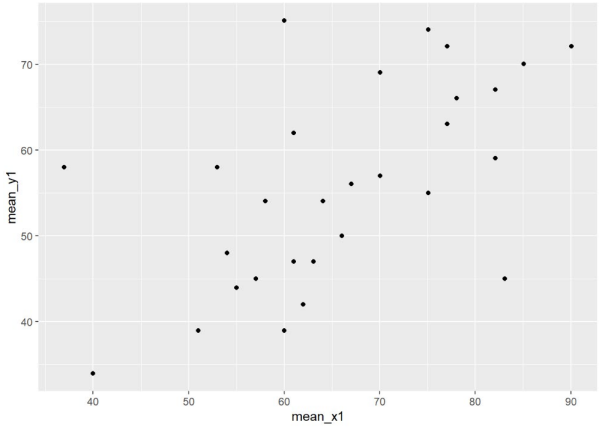


Figure 5. Clustering Scatter Plot

Step 7: Compare the Output

Compare the derived data set with the R built-in data set “attitude” and set the accuracy to 0.1:

```
library(diffdf)
base <- attitude %>%
   rename(x=complaints, y=learning) %>%
   select(x,y) %>%
   arrange(x,y)

qc <- df_cluster_c %>%
   rename(x=mean_x1, y=mean_y1) %>%
   select(x,y) %>%
   arrange(x,y)

diffdf(base,qc,tolerance = 0.1)
```

TYPE 2: COLORED SCATTER PLOT

The only difference between black scatter plot and the colored scatter plot is that the points in colored scatter plot are grouped by colors. Therefore, the only difference between extracting data from the two kinds of graphs is adding the way of recognizing and grouping the points by color.

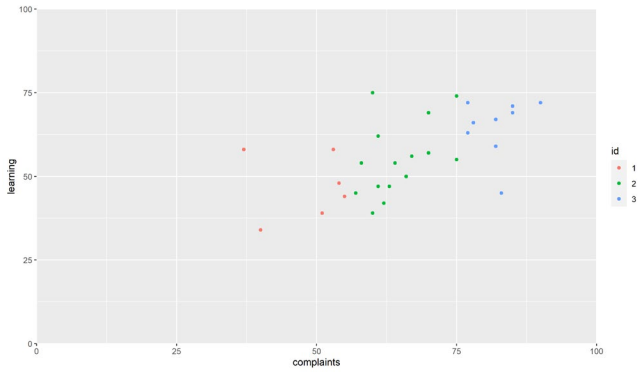


Figure 6. Colored Scatter Plot

After cropping out the main graph, convert the image into data frame with a "wide" format, by setting wide= "c". This argument would transpose the dataset by column "cc", which have values 1, 2 and 3, stands for red, green and blue. After transposing, colors could be recognized by using rgb(red, green, blue), and pixel points could be grouped:

```
df2_2 <-as.data.frame(im1,wide="c") %>% mutate(rgb.val=rgb(c.1,c.2,c.3))
```

Figure 7 shows the first two rows of the data frame converted from graph with the color information, where hexadecimal color #EBEBEB stands for bright gray.

	x	y	c.1	c.2	c.3	rgb.val
1	1	1	0.9215686	0.9215686	0.9215686	#EBEBEB
2	2	1	0.9215686	0.9215686	0.9215686	#EBEBEB

Figure 7. Data Frame for Colored Scatter Plot

In the main graph, the only two colors besides the data points are white and gray, which should be filtered out of the dataset (#FFFFFF is white and #EBEBEB is gray):

```
df3<- df2_2%>%
  filter(rgb.val != "#FFFFFF" & rgb.val != "#EBEBEB")
```

Figure 8: "ggplot" is used to check if the data is correct (Group #00BA38 is id=2, #619CFF is id=3, #F8766D is id=1.)

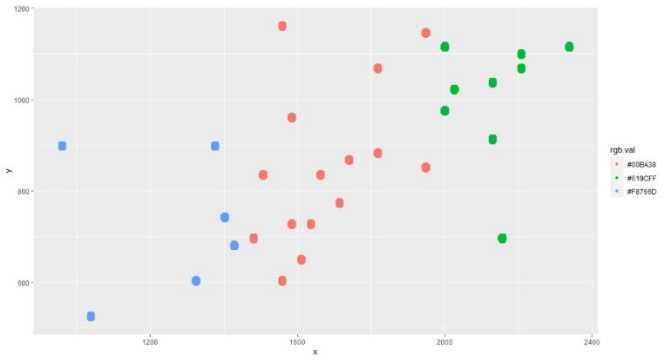


Figure 8. Graph for Colored Pixels

TYPE 3: LINE GRAPH

The only difference between line graph and scatter plot is the line. The morphological closing by a square element would be used to erode/dilate (dilation followed by erosion) the line and then only the scatter points would be kept. imager::mclosing_square() will be used in this section, where the first argument is the object and the second is the size of the structuring element (adjust according to the line size):

```
library(imager)
library(ggplot2)
img <- imager::load.image("../line_graph.png")
closing_img <- mclosing_square(img,size=9)
plot(img)
```

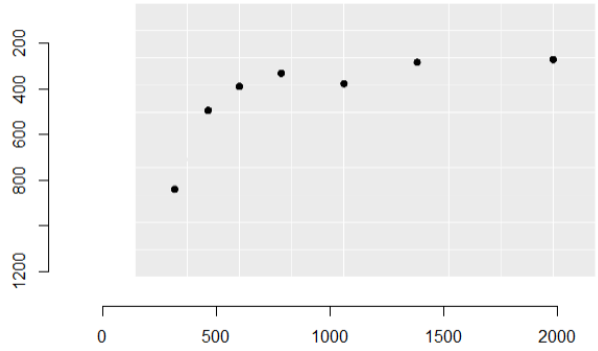


Figure 9. Morphological Closing Graph

TYPE 4: GRAPHS WITH WHITE BACKGROUND AND BLACK BORDER

To crop this kind of graph, no background could be used to filter out the graph area. Therefore, the black border, and the position of tickers need to be found.

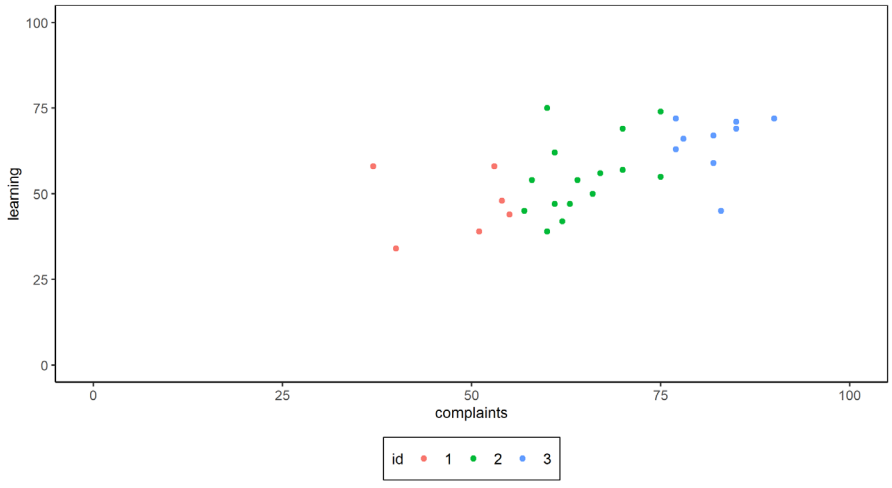
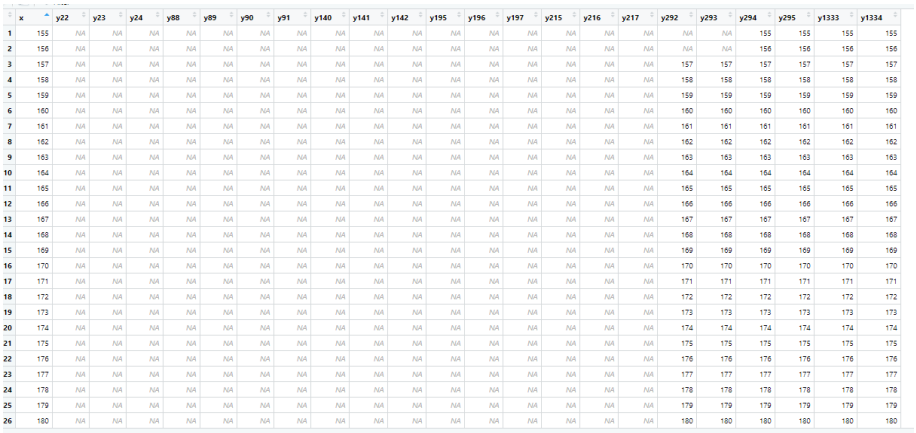


Figure 10. Graph with Black Border

Find the Position of the Upper and Lower Borders

There are following elements in black color that pass through the median of x (the middle line of the figure): upper and lower border for the main graph and legend, X-axis label, and maybe some lines or points in the graph. Therefore, median of x could be used to filter out most of the data in the converted data frame. After filtering, the data frame would be transposed by y with values from X-axis. The data frame would look like the figure, where the column names are the Y-axis and the column x is the X-axis, as shown in Figure 11.

Figure 11 shows that in the data frame. From the column names, the thickness of upper border is 2 pixel points (y = 1333 and 1334), where the thickness of the lower border is 4 pixel points (y = 292 - 295)



	x	y22	y23	y24	y88	y89	y90	y91	y140	y141	y142	y195	y196	y197	y215	y216	y217	y292	y293	y294	y295	y1333	y1334
1	155	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	155	155	155	155
2	156	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	156	156	156	156
3	157	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	157	157	157	157	157	157
4	158	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	158	158	158	158	158	158
5	159	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	159	159	159	159	159	159
6	160	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	160	160	160	160	160	160
7	161	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	161	161	161	161	161	161
8	162	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	162	162	162	162	162	162
9	163	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	163	163	163	163	163	163
10	164	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	164	164	164	164	164	164
11	165	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	165	165	165	165	165	165
12	166	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	166	166	166	166	166	166
13	167	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	167	167	167	167	167	167
14	168	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	168	168	168	168	168	168
15	169	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	169	169	169	169	169	169
16	170	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	170	170	170	170	170	170
17	171	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	171	171	171	171	171	171
18	172	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	172	172	172	172	172	172
19	173	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	173	173	173	173	173	173
20	174	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	174	174	174	174	174	174
21	175	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	175	175	175	175	175	175
22	176	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	176	176	176	176	176	176
23	177	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	177	177	177	177	177	177
24	178	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	178	178	178	178	178	178
25	179	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	179	179	179	179	179	179
26	180	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	180	180	180	180	180	180

Figure 11. Data Frame for Border

In the elements that would pass through the median, the only ones that have the largest differences of x values are left and right border. Therefore, the differences of maximum and minimum x values of each y values need to be calculated and compared. Then the y values that have the largest x differences are selected, and the largest and smallest y value selected should be the upper and lower borders of the main graph.

However, the thickness of the borders could be different, and all the black borders need to be cropped off, so the different thickness need to be determined. This could be done by calculating the differences between the adjacent y values filtered out, by using diff(). The differences larger than 1 shows the pixel points from the two y values are not connected with each other, which could not be both on the upper border or lower border. The index of the differences larger than 1 have been collected, and the first index number should be the thickness of the lower border (y pixel axis is downward). The thickness of upper axis could not be determined by the last index number. The thickness will be the difference between length of the list (total number of y values with the largest x differences) and the last index value:

```
framey <- yvalue[ran1:ran2]
diffy <- diff(framey)
thicky <- which(diffy > 1)
thicktop <- thicky[1] + 0
thickbot <- length(framey) - thicky[length(thicky)]
```

After calculating y values, x values need to be calculated for left and right border. The process is simpler since y values are determined. The only elements with black color and pass-through median y, and one of the upper and lower borders' x values are left and right borders (Figure 12).

Figure 12 shows the origin data frame is filtered by black color, median Y and one of the Y values that have largest X differences.

	x	y
1	155	294
2	156	294
3	157	294
4	158	294
5	2457	294
6	2458	294
7	2459	294

Figure 12. Data Frame for X border

From filtered data, the x values left are from left and right borders. The left and right border of the main graph and the thickness could be calculated with the same method as upper and lower border. Since there are only x values from left and right border, the thickness could be calculated directly:

```
xvalue <- df_bx[,1]
minx <- min(xvalue)+0
maxx <- max(xvalue)+0
diffx <- diff(xvalue)
thickleft <- which(diffx > 1)[1]+0
thickright <- length(xvalue)-thickleft
```

All the values of black borders are collected. The flipped image could be cropped:

```
im1 <- imsub(flip_im, (minx+thickleft) <= x & x <= (maxx-thickright),
(miny+thicktop) <= y & y <= (maxy-thickbot))
```

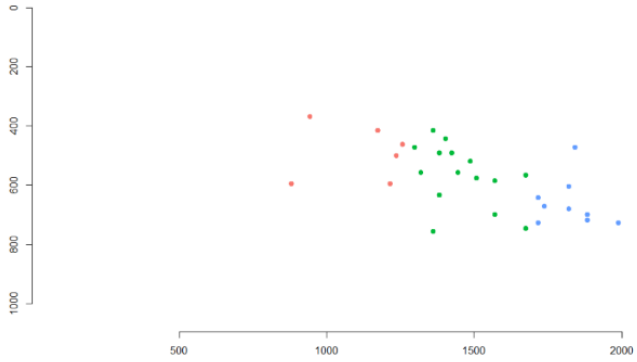


Figure 13. Cropped Graph

Find the Position of the First and Last Tickers

Since the x and y values for the data points are calculated by corresponding ratio of the origin x and y coordinate and the x and y pixel coordinate. The position of start points and the end points of the origin X and Y-axis need to be calculated.

In the origin data frame converted from flipped image, the positions of tickers must appear on pixels with x slightly smaller than left border and y slightly smaller than lower border (In flipped image is on the top). Therefore, the data frame would be filtered by color black, and respectively filtered by x = minx – 1 and

y = miny -1, for y and x axis tickers:

```
t_x<- minx-1
df_t_y <- df%>%
  filter(x==t_x, value <0.3) %>%
  distinct(x, y)
tick_y <- df_t_y[["y"]]
```

Since tickers also have thickness, the same method as calculating borders' thickness has been used. However, the usage of ticker is for calculating the ratio, which needs to be one value, but not the edge and thickness. Therefore, the middle points of the first and last tickers are calculated:

```
t_diff_y <- diff(tick_y)
t_thick_y <- which(t_diff_y>1)
#get pix value at ticker
ticker_y_min <- mean(tick_y[1:t_thick_y[1]])
ticker_y_max <-
   mean(tick_y[(t_thick_y[length(t_thick_y)]+1):length(tick_y)])
```

The position of the tickers (the start and the end of axis), in the origin image has been calculated. Since the new graph is a subset from origin graph, the length of its x and y pixel axis is the same as when it was in the origin graph, except the axis are started with 0. Therefore, the positions of the tickers in the new cropped graph could be calculated by subtracting x value and thickness of left border or y value and thickness of lower border (top border in the flipped image):

```
#pix value in new graph
tick1_y_max1 <- ticker_y_max - (miny + thicktop) + 1
tick1_y_min1 <- ticker_y_min - (miny + thicktop) + 1
tick1_x_max1 <- ticker_x_max - (minx + thickleft) + 1
tick1_x_min1 <- ticker_x_min - (minx + thickleft) + 1
```

tick1_x_max1	2195.5
tick1_x_min1	103.5
tick1_y_max1	991.5
tick1_y_min1	45.5

Figure 14 Final Result for Starts and Ends of X and Y Axis in Cropped Image.

METHOD 2. CONVERT TO AN INTERACTIVE GRAPH

R can also generate interactive graphs that can be placed on web pages.

CONVERT WITH THE R PACKAGE "PLOTLY"

"plotly" is both a commercial service and open-source product for creating high end interactive visualizations. The package plotly allows you to create plotly interactive graphs from within R. In addition, any ggplot2 graph can be turned into a plotly graph.

The code for conversion is shown below:

```
library(tidyverse)
library(ggplot2)
library(plotly)

p <- ggplot(data = CO2) +
    geom_point(mapping = aes(x = conc, y = uptake, color=Plant), size=3) +
    geom_line(mapping = aes(x = conc, y = uptake, color=Plant),size=0.5) +
    scale_x_continuous(expand=c(0,0), limits=c(0,1100)) +
    scale_y_continuous(expand=c(0,0),limits=c(0,50))


pp <- ggplotly(p)
htmlwidgets::saveWidget(pp,"../interactive_plotly.html")
```

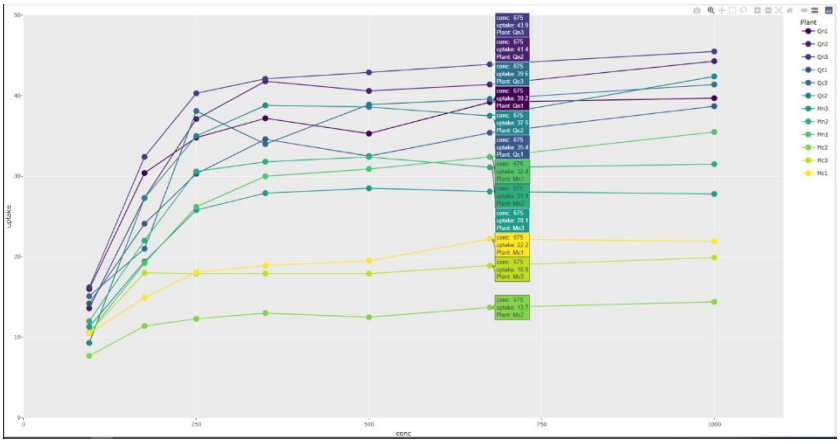


Figure 15. Interactive Graph Generated by "plotly"

CONVERT WITH THE R PACKAGE “HIGHCHARTER”

“highcharter” provides access to the Highcharts JavaScript graphics library, which can create another interactive line chart.

The code for conversion is shown below:

```
library(tidyverse)
library(highcharter)

CO2_1 <- CO2 %>%
  select(conc, Plant, uptake)

plotdata <- spread(CO2_1, Plant, uptake)

h <- highchart() %>%
  hc_xAxis(categories = plotdata$conc) %>%
  hc_add_series(name = "Qn1",
                data = plotdata$Qn1) %>%
  hc_add_series(name = "Qn2",
                data = plotdata$Qn2) %>%
  hc_add_series(name = "Qn3",
                data = plotdata$Qn3) %>%
  hc_add_series(name = "Qc1",
                data = plotdata$Qc1) %>%
  hc_add_series(name = "Qc2",
                data = plotdata$Qc2) %>%
  hc_add_series(name = "Qc3",
                data = plotdata$Qc3)%>%
  hc_add_series(name = "Mn1",
                data = plotdata$Mn1) %>%
  hc_add_series(name = "Mn2",
                data = plotdata$Mn2) %>%
  hc_add_series(name = "Mn3",
                data = plotdata$Mn3) %>%
  hc_add_series(name = "Mc1",
                data = plotdata$Mc1) %>%
  hc_add_series(name = "Mc2",
                data = plotdata$Mc2) %>%
  hc_add_series(name = "Mc3",
                data = plotdata$Mc3)
h <- h %>%
  hc_tooltip(crosshairs = TRUE,
             backgroundColor = "#FCFFC5",
             shared = TRUE,
             borderWidth = 4) %>%
  hc_legend(align = "left",
            verticalAlign = "top",
            layout = "vertical",
            x = 0,
            y = 100) %>%
  hc_exporting(enabled = TRUE)

htmlwidgets::saveWidget(h,"../interactive_highcharter.html")
```

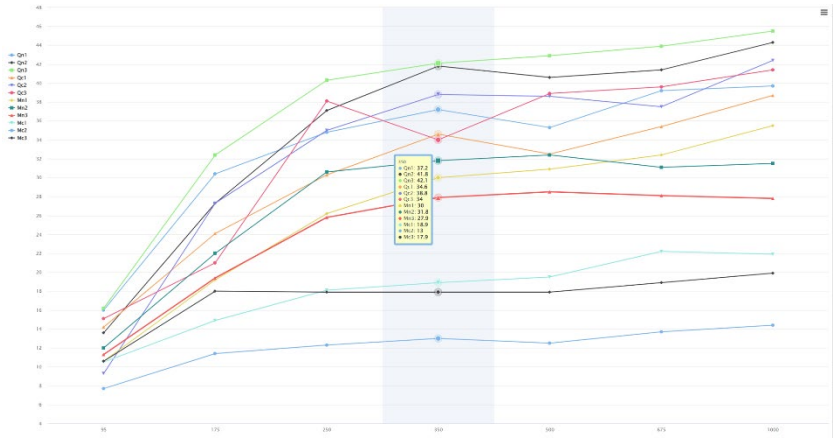


Figure 16. Interactive Graph Generated by "highcharter"

METHOD 3. EXTRACT JSON DATA PRECISELY FROM INTERACTIVE GRAPHS

The background data is stored in the interactive graph with RDS format, so the graph can be converted into the JSONEDIT object, following by the HTML Widget, and then into the format of JSON string. Finally, it's easy to compute the X and Y coordinates of the points from the JSON string.

The process of extracting data will be shown in 6 steps.

STEP 1: READ THE INTERACTIVE PLOT IN RDS FORMAT WITH R

The interactive graphs need to be read in the RDS format, which contains the most complete information:

```
library(tidyverse)
library(plotly)
library(htmlwidgets)
library(diffdf)

p <- readRDS("../interactive_myplot.rds")
```

STEP 2: CONVERT INTERACTIVE PLOT TO JSONEDIT OBJECT

The jsonedit object is shown below:

```
p_json <- plotly_json(p)
```

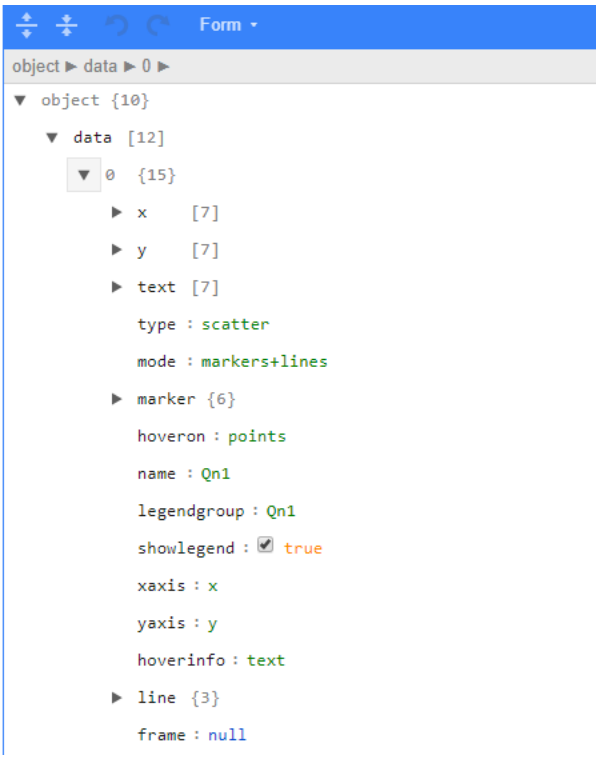


Figure 17. JSONEDIT Object

STEP 3: CONVERT JSONEDIT OBJECT TO HTML WIDGET

The code for conversion is shown below:

```
p_widget <- createWidget("plotly",p_json, width="100%", height="400px")
```

STEP 4: CONVERT HTML WIDGET TO JSON STRING

The code for conversion is shown below:

```
p_json_string <- p_widget[["x"]][["x"]][["data"]]
```

STEP 5: DERIVE THE DATA FROM JSON STRING

First, get the list from JSON string, then the X and Y coordinates and legend group of each point can be obtained through regularization and data tidying:

```
p_data <- fromJSON(p_json_string)

text_length <- length(p_data$data$text)
element_new <- data.frame()
for (i in 1:text_length) {
  text <- p_data$data$text[[i]]
  elements <- strsplit(text,"<br />")
  split_elements <- lapply(elements, function(x) lapply(x, function(y)
strsplit(y, ":\\s+")[[1]]))
  for (j in 1:length(split_elements)) {
```

```
    element <- as.data.frame(split_elements[j])
    element_name <- as.character(element[1,])
    element2 <- element[2,]
    names(element2) <- element_name
    element_new <- bind_rows(element_new, element2)
  }
}
qc <- element_new %>%
  mutate(conc=as.numeric(conc),uptake=as.numeric(uptake)) %>%
  arrange(conc,uptake,Plant)
```

STEP 6: COMPARE THE OUTPUT

Compare the derived data set with the R built-in data set "CO2":

```
base <- CO2 %>%
  select(conc,uptake,Plant) %>%
  mutate(Plant=as.character(Plant)) %>%
  arrange(conc,uptake,Plant)

diffdf(base,qc)
```

CONCLUSION

This paper is to show three ways of extracting data from graphs, which will simplify the programming process and improve the efficiency and accuracy when compare the results across tables, listings and figures. On the other hand, it is helpful for us to further explore the application of automation in clinical trials.

REFERENCES

Barthelmé, Simon "Getting started with imager" May 9, 2023. https://cran.r-project.org/web/packages/imager/vignettes/gettingstarted.html#plotting-and-loading-images

CONTACT INFORMATION

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

Name: Dan Zhao
Enterprise: Pfizer (China) Research and Development Co.,Ltd. China
E-mail: dan.zhao2@pfizer.com