Paper DV07

Make your data come alive

Babych Oleksandr, Intego Group LLC, Kharkiv, Ukraine

ABSTRACT

Data visualization enables you to observe patterns, summarize and interpret complex tabular data efficiently at a glance. But nowadays static graphs often do not provide the necessary amount of information to the data scientist. The industry trend, which is becoming a standard, lies in making data visualizations interactive. The implementation of interactivity can be profoundly powerful. It allows to pose multiple questions per graph, focus on details, zoom in on various deviations and increase chances to discover insights. Fortunately, there are two charting JavaScript libraries which are available for Python and R: Highcharts and Plotly. Both of them can produce stunning and smooth publication-quality animated graphs. The main goal of this paper is to demonstrate how to produce common clinical trials graphs in R and make them interactive.

INTRODUCTION

Figures are part of any submission to a regulatory authority in clinical trials, but usually they do not contain interactive clinical data visualizations which has recently become quite popular among data scientists.

There are lots of Graphical User Interface (GUI) systems for creating interactive graphs, but almost all of them have certain disadvantages such as the inability to repeat the result without human intervention (for example, if the data was changed). Unlike GUI systems, the code-based approach allows to repeat a graph on any data, easily customize it, and integrate with another system[1].

Implementation of interactive clinical data visualizations can add tremendous value during the conduct of a clinical trial. It is a powerful tool to investigate the data and interactively change the level of detail you need to see. It helps to identify data issues, enhances the quality of clinical reporting, and can extend the amount of information on the graph without increasing its complexity. Interactivity allows to create clean looking data visualizations that summarize a large amount of information and provide a tool for viewers to explore data as much as they're interested.

Interactive visualizations enable the following benefits:

- **More information**
  Interactivity provides an opportunity to include much more information (and avoid overplotting) than any static graph, show and hide the information on demand by using tooltips or click-events, and the possibility to filter data.

- **Pose multiple questions**
  Since interactivity allows us to summarize more data on the plot, therefore the number of questions it can answer is highly increased.

- **Focus on details and promotes exploration**
  Interactivity enables users to zoom into a specific part of the data visualization, find intriguing variations, and quickly identify observations that stand out and deserve further investigation. Responsive data encourages deeper exploration and, therefore, leads to obtaining more insights.

- **Highlight the key points**
  Interactive data visualization can be useful in highlighting the key points. It instantly draws viewers' attention and displays accurate and essential information.

- **Enhance user experience**
  Interactive graphs are visually attractive and provide an easier perception of complex data. They can easily grab users' attention and engage them into interaction (for instance, if added in an email). Interactivity provides more enjoyment and fun.

There are plenty of charting libraries written in pure JavaScript offering easy ways of creating publication-quality interactive web-based graphs, but in this paper, I will focus on two of them: Highcharts and Plotly. One of the main advantages of these two JavaScript libraries is their binding to popular languages like R and Python. Since they are both available in R, it enables the user to combine their functionality with a leading computing environment for data analysis and statistics. Both libraries generate web-based data visualizations that can be easily shared between

colleagues via email or a website, which improves the conversation and speeds up the analysis. Highcharts and plotly were both inspired by a well-known Hadley Wickham's ggplot2 package for data visualization and the concept of Grammar of Graphics. So if you are familiar with the ggplot2 R package, it will be easy to get acquainted with these ones and start getting advantage from it. Going forward, it is possible to test all data visualizations shown in this paper at https://babych.me/slides.

PLOTLY PACKAGE

The plotly R package enables us to create a variety of interactive web graphics using the open-source JavaScript graphing library plotly.js. Here is a little explanation on how R code is transformed into an HTML chart which is based on JavaScript. Firstly, when the plotly object is printed, the plotly_build() function is applied to this object producing a specific R list that follows the syntax that plotly.js understands. Plotly uses JavaScript Object Notation (JSON) syntax to represent and render web graphics. After that, the R list is transformed into a JSON object by plotly_json() function. In the last step (so-called, render step) plotly.js transforms the JSON object to an HTML chart.

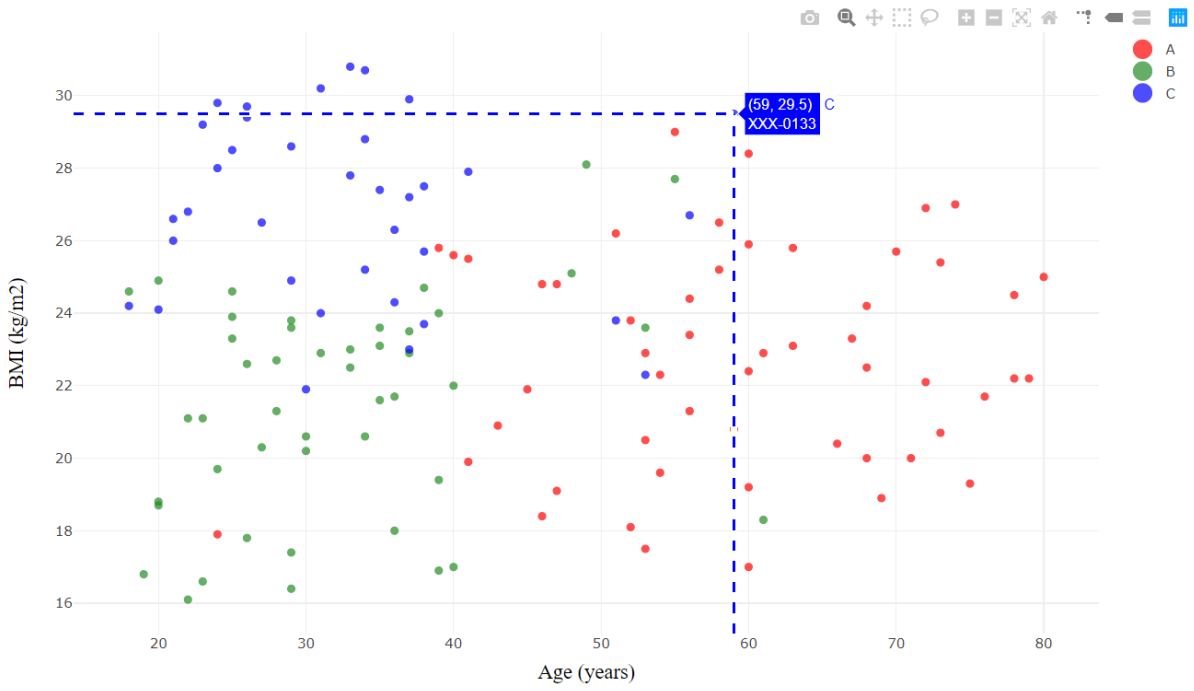
There are two main ways to create a plotly object:

- Directly set up a plotly object using the plot_ly() function.
- Transform a ggplot2 graph into a plotly object using the ggplotly() function.

Both approaches have their benefits and limitations, but the Grammar of Graphics was implemented into both of them. Let's start with the direct initialization of plotly object via plot_ly() function.

First of all, to install (from CRAN) and load plotly package it is necessary to run:

```
install.packages("plotly")
library(plotly)
```




```
plot_ly(ADSL, x = ~AGE, y = ~BMI, color = ~ARMCD, text = ~USUBJID, size = I(20),
        colors = c("red", "forestgreen","blue")) %>%
    add_trace(type = "scatter",  mode = "markers") %>%
    layout(xaxis =list(title = "Age (years)"), yaxis =list(title = "BMI (kg/m2)")))
```

FIGURE 1 – SCATTER PLOT

PLOT_LY () FUNCTION

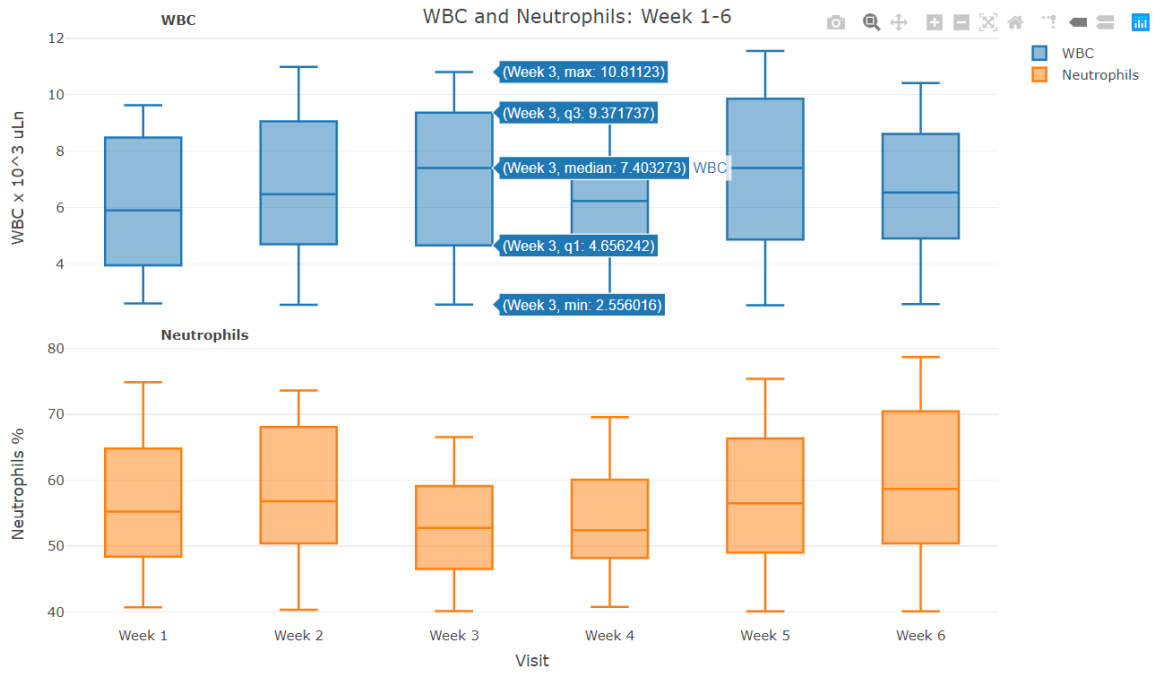
The syntax of the plot_ly() function is very similar to ggplot2() function. To create a simple graph we need to define a dataset as the first argument of plot_ly() function and then assign variable names to visual properties (for example, x, y, color, text, etc) also inside the plot_ly(). By default plot_ly() tries to find an appropriate geometrical representation for these mapping of data variables, but to avoid issues it is better to define geometric objects by yourself. To do so

you can use the family of add*_() functions (e.g., add_markers(), add_bars(), add_lines(), etc) which add the geometrical layer of the plot. As it is shown in Figure 1, to create a basic scatter plot using plot_ly() we simply defined the dataset (ADSL), mapped the AGE variable onto the x-axis, and the BMI variable onto the y-axis. Using the "%>%" operator we have added a geometrical layer using the add_trace() function where *type* and *mode* parameters were defined. In fact, we could simply use the add_markers() function and the result would be the same. So, add_trace(type= , mode=) is a general function for adding a geometrical layer and add_markers() is its particular representation that creates a scatter plot by adding points.

Also, in Figure 1 it is shown that the ARMCD variable was mapped to a *color* argument which colored points depending on the value of the ARMCD variable and automatically added a legend. The *colors* argument defined the color of the points, and by setting the *size* argument the size of the points was enlarged. The layout() function modifies the layout of plotly visualization, e.g., plot title, axis title, etc.

Now, let's look at which interactivity plotly provides and then it will make it clear why the USUBJID variable was mapped into the *text* argument inside the plot_ly() function. On the right top corner of Figure 1 you can see a plotly dropdown menu which provides the following functionality:

- Download plot as png (export static image)
- Zoom into the selected part of the plot, zoom in and out
- Pan
- Box select or lasso select
- Autoscale
- Reset axes
- Show closest data on hover
- Compare data on hover
- Toggle spike lines
- Scroll axes



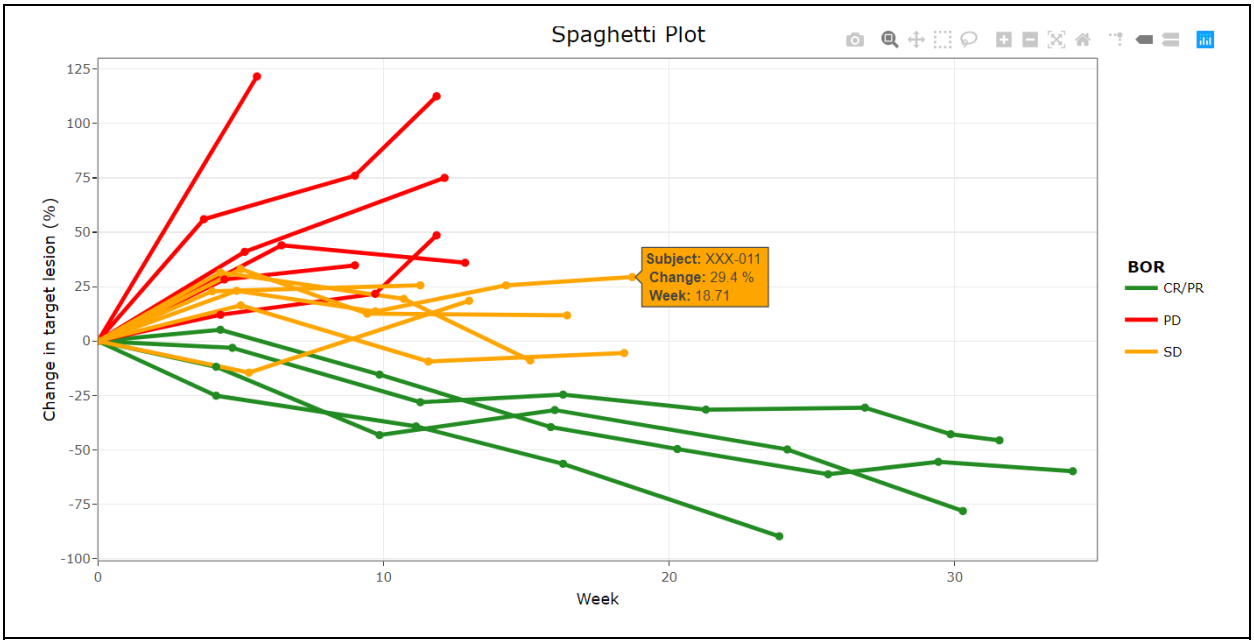

```
fig1 <- plot_ly(ADLB_1, y = ~AVAL, x = ~VISIT, type = "box", name = "WBC") %>%
          layout(yaxis = list(title = 'WBC x 10^3 uLn'),
          annotations = list(x = 0.1 , y = 1.1, text = "<b>WBC</b>",
                             showarrow = F, xref='paper', yref='paper'))
fig2 <- plot_ly(ADLB_2, y = ~AVAL, x = ~VISIT, type="box", name="Neutrophils") %>%
          layout(yaxis = list(title = 'Neutrophils %'),
          annotations = list(x = 0.1 , y = 1.03, text = "<b>Neutrophils</b>",
                             showarrow = F, xref='paper', yref='paper'))
subplot(fig1, fig2, nrows = 2, shareX = TRUE, titleY = TRUE) %>%
    layout(title = "WBC and Neutrophils: Week 1-6", xaxis = list(title = "Visit"))
```

FIGURE 2 – TWO BOXPLOTS ON THE ONE GRAPH

Of course, it is possible to remove the modebar buttons or add custom ones. Figure 1 shows that three options of plotly mode bar are on. The “Zoom” and the “Show closest data on hover” options are on by default and the third option “Toggle spike lines” was turned on manually. So, when we point at the dot the dashed lines and the tooltip with information about this dot appear. The first line holds precise information about variables that were assigned to *x* (AGE), *y* (BMI), and *color* (ARMCD) arguments. On the second line, there is a subject identifier (USUBJID) which was represented by this dot. It appeared because the USUBJID variable was mapped to the *text* argument of plot_ly() function. Of course, this tooltip may look slightly confusing (due to being produced by default) at the first glance, but it is possible to customize it whatever you like.

In Figure 2 we can see two box plots that represent the changing of the statistical parameters over time in two laboratory tests: WBC and Neutrophils. The code under the graph shows how to set up a box plot via plotly package and how two plots can be placed on the single graph. So, the creation of a box plot is pretty similar to Figure 1, except for the geometrical layers that were defined inside the plot_ly() function by assigning the *type* argument. Note that ADLB_1 and ADLB_2 datasets represent two subsets of ADLB dataset that includes only one specific parameter. Annotations (which is an argument of a layout function) helps to add a custom text to the graph. In this example, it is the name of the parameter.

To make a good looking code that can be easily read we have saved two box plots as objects called “fig1” and “fig2”. To merge several plotly objects into a single one the subplot() function is used. The definition of the subplot() function is fairly straightforward: it is required to set up figures that must be placed together, the *nrows* argument sets the number of rows for laying out plots in a grid-like structure, and “shareX = TRUE” means that x-axis must be shared among the subplots.




```
# Spaghetti Plot by ggplot2
ggplot(ARS_S, aes(x = Week, y = AVAL, group = USUBJID, color = BOR)) +
  geom_line(size = 1) +
  geom_point(aes(text=paste('<b>Subject:</b>',USUBJID,
                            '<br>','<b>Change:</b>', AVAL, '%',
                            '<br>', '<b>Week:</b>', Week))) +
  scale_color_manual(name = "", values = c('forestgreen','red','orange')) +
  scale_x_continuous(name="Week",limits=c(0,35),breaks=seq(0,40,10),expand=c(0, 0))+
  scale_y_continuous(name = "Change in target lesion (%)", limits = c(-101,130),
                     breaks = seq(-100,125,25), expand=c(0, 0)) +
  theme_bw()

# ggplot2 plot converted into plotly object using the ggplotly()
ggplotly(p = ggplot2::last_plot(), tooltip = c("text")) %>%
    layout(legend = list(x = 100, y =0.5, title = list(text = "<b> BOR </b>")),
           title = list(text = "Spaghetti Plot"))
```

FIGURE 3 – SPAGHETTI PLOT FROM GGPLOT2 TO PLOTLY

The default tooltip of the box plot shows the VISIT value (the variable that is mapped to the x-axis), the names of statistical parameters (min, max, median, q1, q3), and their exact values. Showing information on demand helps to avoid overplotting and is a great advantage provided by interactive graphs.

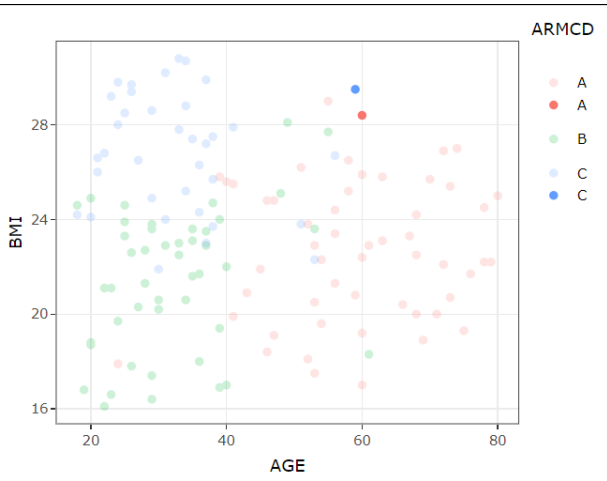
GGPLOTLY() FUNCTION

Since ggplot2 is a well-known and preferred tool of choice for constructing graphs in R, it means that it was used to create a large amount of clinical trial graphs. The plotly ability to transform graphs produced by ggplot2 to plotly object makes this package very unique and outstanding. So, the second way to create a plotly object consists in using the ggplotly() function which simply takes a ggplot2 object as an argument.

Let's look at the spaghetti plot shown in Figure 3 which displays a percent change in target lesion over time (in weeks). Firstly, we need to create a ggplot2 graph by defining data and aesthetics layers inside the ggplot() function and add a geometrical layer. In this case, two geometrical layers were used: geom_point() to displays observations of the dataset as points and geom_line() to connect these points (separately for each subject). Lines and dots were colored depending on the values of the Best Overall Response (BOR) parameter as shown in the legend. Three scale_*() functions change the appearance of the graph by defining axis labels, setting ticks, axis labels, and colors. When the ggplot2 graph is ready, it is now a good choice to save the graphs as an object (which can be later used as an argument for ggplotly() function). Or it is possible to identify the last graph produced by ggplot2 and take it as an argument as shown in the code under the graph:

```
ggplotly(p = ggplot2::last_plot())
```

The *tooltip* argument defines which information appears when you point at the dot. In Figure 3 the tooltip shows values from y and x-axes (AVAL and WEEK) and the USUBJID to which this dot refers. So, you can see how easy it is to add some interactivity to a ggplot2 graph – you simply need one line of code and that's all.



Show 10 entries

	USUBJID	ARMCD	SEX	AGE
122	XXX-0041	A	F	60
129	XXX-0133	C	F	59

Showing 1 to 2 of 2 entries (filtered from 135 total entries)

```
adsl_table <- highlight_key(ADSL)
plot <- ggplot(adsl_table, aes(AGE, BMI, color = ARMCD, text = USUBJID)) +
            + geom_point()
final_plot <- highlight(ggplotly(plot), "plotly_selected")
crosstalk::bscols(final_plot, DT::datatable(adsl_table))
```

FIGURE 4 – SCATTER PLOT WITH A DATATABLE

Another ability of the plotly package in the combination with the crosstalk package consists in attaching a dataset to a plotly object. In Figure 4 we can see the same graph as in Figure 1, but in this case it was created using the ggplot2 package and then converted to a plotly object using the ggplotly() function. Highlight() and highlight_key() functions are designed to link multiple plotly graphs or objects. In the final row, the bscols() function form the crosstalk package was used. The bscols() function helps to put HTML elements side by side, it takes as an arguments the plotly object called "final plot" and the dataset that was used to create a plot. As a result, the datatable appears on the right side of the graph and, moreover, this datatable is also interactive. As shown in Figure 4, two points were selected using the lasso select option and the datatable was immediately filtered to show only these two observations from the ADSL dataset.

HIGHCHARTS

Highcharts is a modern SVG-based, multi-platform charting library written in pure JavaScript. Highcharts is very flexible and customizable and has many of wrappers for most popular languages, such as R, Python, .Net, and others. The main benefit of Highcharts is its' beautiful graphics which makes plots look awesome. In this paper, we will consider the R wrapper for highcharts called highcharter.

```
install.packages("highcharter")
library(highcharter)
```

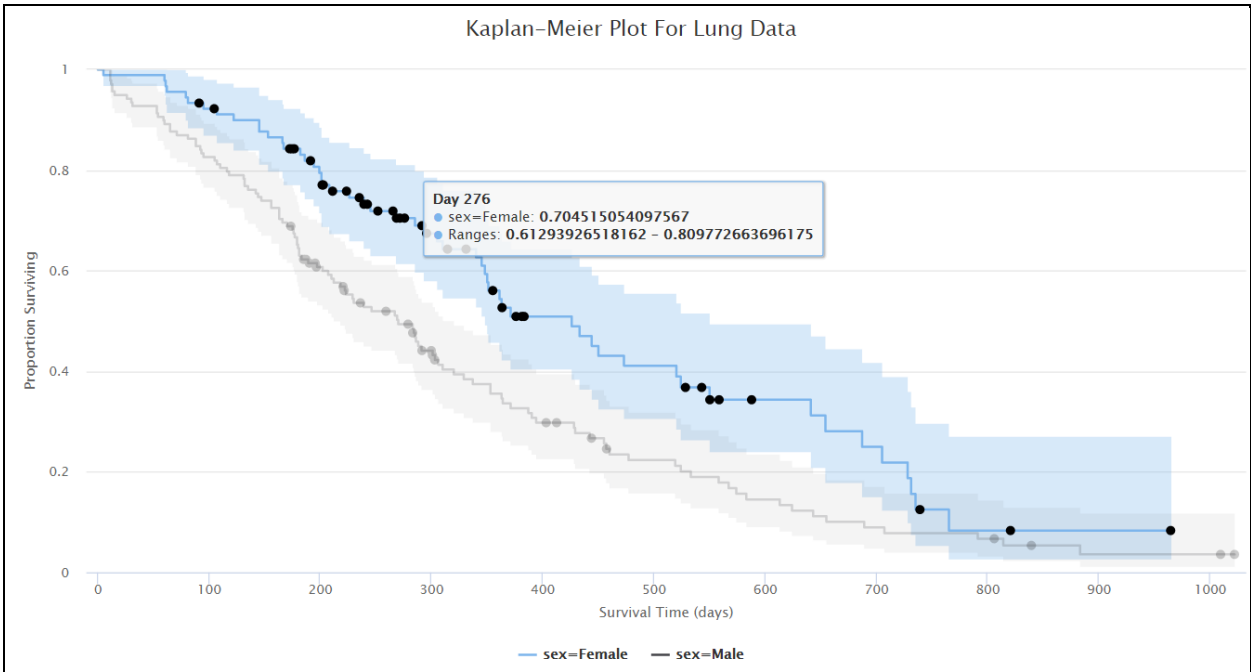
The main features of the highcharter package are its ability to create various charts with the same style and support various R objects. It also has a pipping style that is loved by all R users and a wide variety of themes which look amazing. Highcharter has two main functions to create a chart:

- highchart() : creates a highchart chart using htmlwidgets.
- hchart(): a generic function which creates a highchart object (chart) from a particular data type.

Also, there are two essential highcharter functions called hc_add_series() and hcaes(). Hc_add_series() function is a generic function which adds data to an existing highchart object and hcaes() function defines aesthetic mappings. All these four functions that I have mentioned before were inspired by the ggplot2 package:

- highchart() is very similar to ggplot2's ggplot() function where it is possible to add further geometrical layers on top of the base ggplot object. In the same manner, once the highchart() function has been defined, further highchart elements can be added on top of it.
- hchart() is very similar to qplot() function from the ggplot2 package.
- hc_add_series() works in a similar way to ggplot2's geom_*() functions
- hcaes() works the same way as ggplot2's aes() function

The main difference with ggplot2 is that it is required to define the data and the aesthetic layers in every highcharts function. Let's start to explore highcharts on various examples.




```
library(survival)
data(lung)
lung <- mutate(lung, sex = ifelse(sex == 1, "Male", "Female"))
fit <- survfit(Surv(time, status) ~ sex, data = lung)
hchart(fit, ranges = TRUE,
       tooltip = list(headerFormat = "<b>Day {point.key} </b> <br>")) %>%
    hc_title(text = "Kaplan-Meier Plot For Lung Data") %>%
    hc_xAxis(title = list(text = "Survival Time (days)")) %>%
    hc_yAxis(title = list(text = "Proportion Surviving"))
```

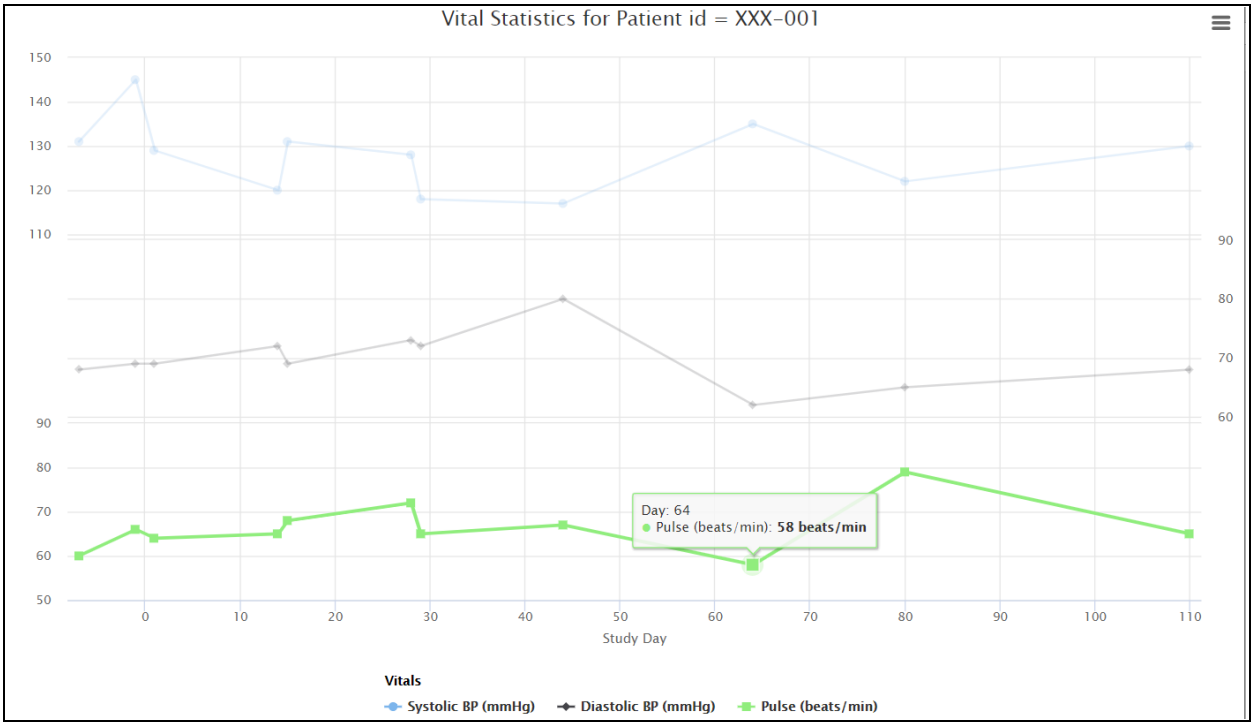
FIGURE 5 – KM PLOT FOR LUNG DATA

In Figure 5 there is the Kaplan-Meier plot for lung data. The dataset “lung” was taken from the survival package and contains data for lung cancer patients. More information about the dataset can be found with:

```
help(lung)
```

I am not focusing on the survival analysis in R in this paper. But in a few words, the surv() function created a survival object for lung data, and survit() function creates survival curves from it[2]. The hchart() function takes the “fit” object (in which we saved survival curves) as an argument and creates an interactive Kaplan-Meier plot. So, when you point at any part of the curve, the tooltip with information about this point appears. Also, using the tooltip option inside hchart() function the word “Day” was added to the tooltip header to make it more clear. Moreover, you can see that we can elaborate the graph using the pipe “%>%” operator which makes the syntax amicable for tidyverse users. In this way, the plot title and the axes titles were added.

It seems to me that by looking even at this first highcharts plot, you have already discovered that the highcharts graphics is much better than anything else and really looks awesome. Anyway, it will become obvious to you in a few minutes later.




```
highchart() %>%
  hc_add_series(ADVS_t, type = "line", hcaes(x = ADY, y = SYSBP), pointWidth = 20,
                yAxis = 0, name = "Systolic BP (mmHg)",
                tooltip = list(headerFormat = "Day: {point.key} <br>",
                               valueSuffix=" mmHg")) %>%
  hc_add_series(ADVS_t, type = "line", hcaes(x = ADY, y = DIASBP), yAxis = 1,
                name = "Diastolic BP (mmHg)",
                tooltip = list(headerFormat = "Day: {point.key} <br>",
                               valueSuffix = " mmHg")) %>%
  hc_add_series(ADVS_t, type = "line", hcaes(x = ADY, y = PULSE), yAxis = 2,
                name = "Pulse (beats/min)",
                tooltip = list(headerFormat = "Day: {point.key} <br>",
                               valueSuffix = " beats/min")) %>%
  hc_yAxis_multiples(create_yaxis(naxis = 3, title = list(text = NULL ) ) ) %>%
  hc_plotOptions(line = list(marker=list(enabled=TRUE))  ) %>%
  hc_title(text = "Vital Statistics for Patient id = XXX-001") %>%
  hc_xAxis(title = list(text="Study Day"), gridLineWidth = list(width = 1)) %>%
  hc_legend(title = list(text="Vitals")) %>%
  hc_exporting(enabled = TRUE)
```

FIGURE 6 – LINE PLOTS FOR VITAL SIGNS

Let's have a look at the next example. Figure 6 displays three separate line plots showing an amendment of vital signs parameters (pulse, systolic, and diastolic blood pressure) over time, which were merged into one graph. To generate this interactive data visualization firstly we have created an empty highcharts object using the highchart() function. After setting up the highcharts object we may add as many highcharts elements as needed on top of it. By looking at the code it is easy to find out that each line plot was added separately with the help of the hc_add_series() function where the aesthetics layer was defined using the hcaes() function. The hc_yAxis_multiples() function assists in establishing three Y axes that are allowed to merge line plots into one graph. The *yAxis* argument inside the hc_add_series() function configures the sequence in which these plots show up. The *tooltip* argument inside the hc_add_series() function helps to customize the tooltip appearance, so the word "Day:" was added to the header, and units were added as a suffix after the value which is shown in Figure 6. Using the hc_plotOptions() function the markers were added to the plot, so it is easy to observe which points lie at the core of these line plots. At the bottom of the code there is an hc_exporting() function that creates the menu icon at the top right corner of the graph. This menu enables us to save images in various formats (PNG, JPEG, PDF, SVG) and also provides an opportunity to save the data (in CSV and XLS format) that underlie the graph. I believe that this can be a useful feature during the sharing of graphs between colleagues. Moreover, it is possible to customize and upgrade this menu to add or remove various options.

HIGHCHARTS TOOLTIPS

Highcharts tooltips are one of the most underrated features. In the tooltips, you can easily render HTML, which implies that it is entirely possible to embed images, tables, and eventually charts. So the possibilities are almost limitless[3]. Truly saying it is not that easy to develop advanced tooltips, but it is not impossible, therefore, let's explore the next graph.

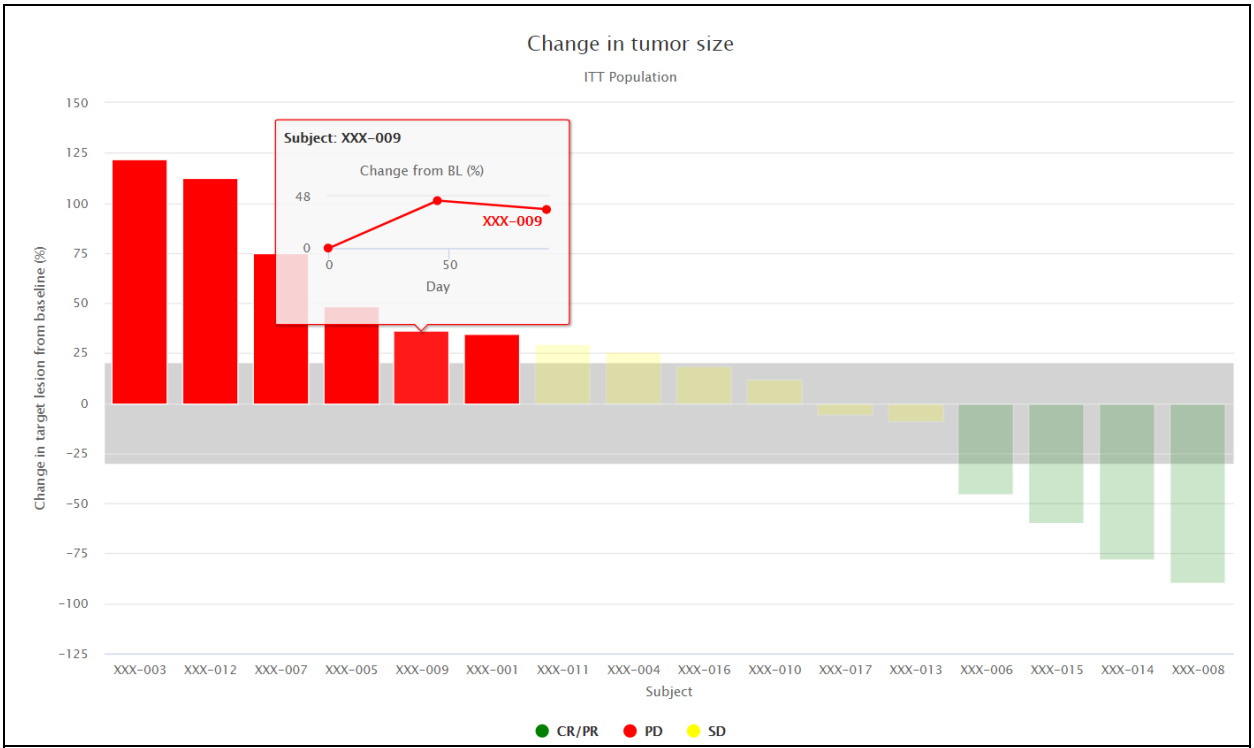
Figure 7 displays a waterfall plot that shows the change in target lesion from baseline among different subjects. Moreover, the tooltip that appears when we point at a column contains a line chart that shows the amendment of a target lesion over time. Also, when you click on some value in the legend you will "turn it off", so all bars that refer to this value will disappear and then the scale of the graph will be updated automatically to be in line with the columns that are still displayed. This feature looks really magnificent and you can check it at the https://babych.me/slides. So, let's figure out how to create such graph and insert a chart in the tooltip.

First of all, I will start with the waterfall plot itself. In the hchart() function, we need to define the dataset that will be used to create graphs (ADRS_f i.e. ADRS final), the type of the graph (type = "column") and assign variables to x and y axes (inside haes() function). By assigning the BOR (best overall response) variable to group aesthetic and adding the hc_colors() function, the columns were colored according to the value of the group variable. Using the hc_yAxis() function we added a grey zone on the chart from -30 to +20 and aligned columns. As you can see, it is quite easy to create a waterfall plot, so now let's consider how a chart can be inserted in the tooltip. Initially, it is necessary to prepare data for the all tooltip charts, which is done at the beginning of the code. So, we change the ADRS dataset in a specific way to create ADRS_tt (ADRS tooltip) dataset which looks as follows:

	USUBJID	ttdata
1	XXX-001	*list(list(x = 0, y = 0), list(x = 31, y = 28.3), list(x = 63, y = 34.8))*
2	XXX-003	*list(list(x = 0, y = 0), list(x = 39, y = 121.6))*
3	XXX-004	*list(list(x = 0, y = 0), list(x = 28, y = 22.9), list(x = 79, y = 25.6))*
4	XXX-005	*list(list(x = 0, y = 0), list(x = 30, y = 12.1), list(x = 68, y = 21.7...*
5	XXX-006	*list(list(x = 0, y = 0), list(x = 33, y = -3.1), list(x = 79, y = -28....*
6	XXX-007	*list(list(x = 0, y = 0), list(x = 36, y = 41), list(x = 85, y = 75))*
7	XXX-008	*list(list(x = 0, y = 0), list(x = 29, y = -25.1), list(x = 78, y = -39...*
8	XXX-009	*list(list(x = 0, y = 0), list(x = 45, y = 44), list(x = 90, y = 36))*
9	XXX-010	*list(list(x = 0, y = 0), list(x = 35, y = 33.1), list(x = 66, y = 12.6...*

PICTURE 1 – DATASET FOR TOOLTIP THAT CONTAINS CHART

The dataset is pretty simple. It includes the USUBJID variable and a ttdata column that contains the R list with data for the tooltip chart. The next step, it is necessary to merge the dataset with tooltip data to the initial one. That is how ADRS_f (ADRS final) dataset was created. To insert a chart in the tooltip we need to add this data to the highcharts object using the hc_tooltip() option and specify two essential arguments: `useHTML = TRUE` and `pointFormatter = tooltip_chart(accesor = "ttdata")`. Now the tooltip chart has been added to the plot. The rest of the code slightly enhances the visual appearance of the tooltip.



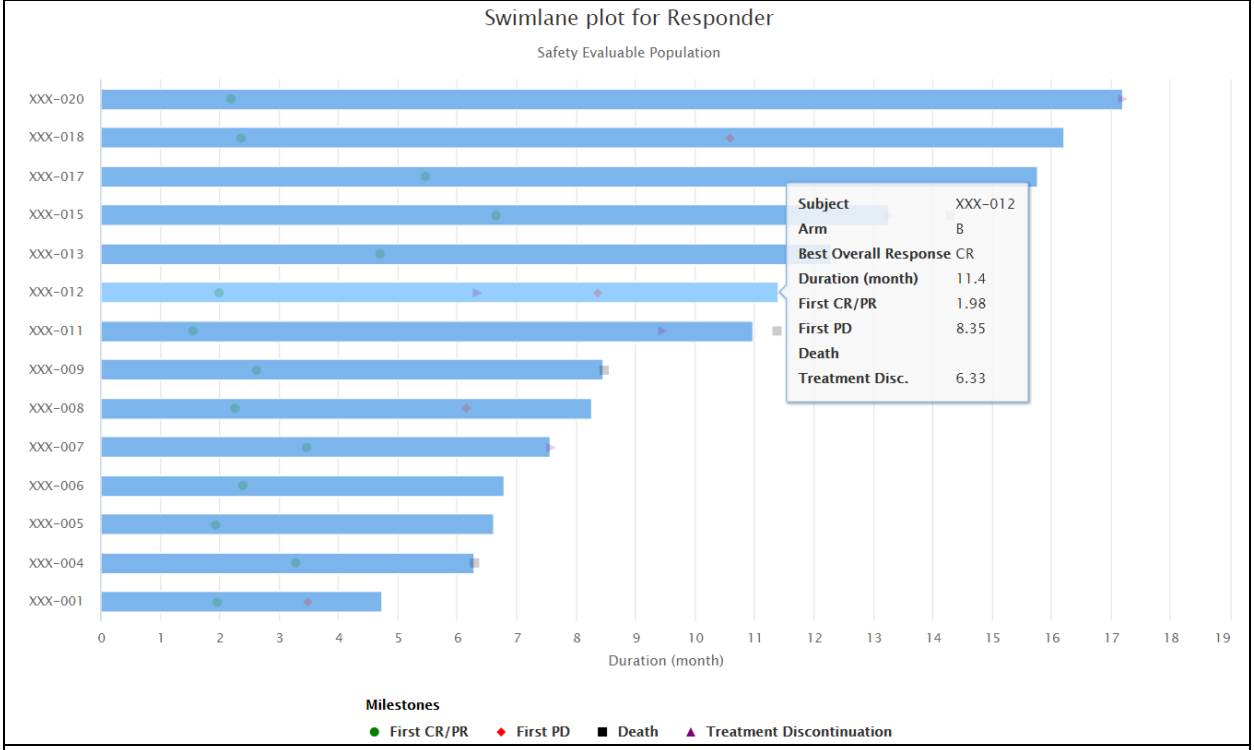

```
# Create dataset ADRS_tt that contains data for tooltip chart
ADRS_tt <- ADRS %>%
    select(USUBJID, ADY, AVAL) %>%
    nest(-USUBJID) %>%
    mutate(data = map(data, mutate_mapping, hcaes(x = ADY, y = AVAL), drop = TRUE),
           data = map(data, list_parse)) %>%
    rename(ttdata = data)

ADRS_f <- left_join(ADRS, ADRS_tt, by = "USUBJID")

hchart(ADRS_f, type = "column", hcaes(x = USUBJID, y = AVAL, group = BOR) ) %>%
    hc_xAxis(categories = ADRS_f$USUBJID, title = list(text = "Subject")) %>%
    hc_yAxis(title = list(text = "Change in target lesion from baseline (%)"),
             plotBands = list(list(color = "lightgrey",from = -30,to = 20))) %>%
    hc_colors(c("green","red","yellow")) %>%
    hc_plotOptions(column = list(pointWidth = 25, centerInCategory = T)) %>%
    hc_title(text = "Change in tumor size") %>%
    hc_subtitle(text = "ITT Population") %>%
    hc_tooltip(useHTML = TRUE, headerFormat = "<b>Subject: {point.key}</b>",
               pointFormatter = tooltip_chart(accesor = "ttdata",
                                hc_opts = list(series =  list(list(name = "point.name")),
                                subtitle = list(text = "Change from BL (%)"),
                                xAxis = list(title = list(text = "Day")))))
```

FIGURE 7 – WATERFALL PLOT WITH A CAHRT IN THE TOOLTIP

The final graph that is considered in this paper is a swimlane plot shown in Figure 8. This type of plot is frequently used in oncology studies where several symbols (i.e. milestones) are added to the bars to indicate a specific event. Let's take a look at how this graph can be created and how interactivity can enhance it.

In general, the coding of the swimlane plot is quite simple: it is necessary to define the dataset, the type of the plot (type = "bars" tells highchart to create a horizontal "columns" while type = "column" means vertical columns) and set the aesthetic mapping. Basically, the swimlane plot is ready and the next step is the addition of milestones which is done with the help of the hc_add_series() function. There are four milestones on the chart, which means that we should use the hc_add_series() function four times to add all of them, but to shorten the code I put an ellipsis instead of repeating the identical code. Inside the hc_add_series() function it is necessary to define the appropriate variable, assign the name to specify how it will show up in the legend, define the symbol and the color and certainly define the tooltip for this type of markers. In Figure 8 it is not so easy to distinguish markers because they were dimmed when we pointed at the subjects (XXX-012) bar. But when we point at any marker, another tooltip will appear with the

information about this specific event.
Now, let's discuss the tooltip that is shown in Figure 8. In Figure 7 we have inserted a chart into the tooltip and in Figure 8, in the same manner, we have inserted a table. At the beginning of the code, two vectors x and y were created. Vector x contains the description text of the parameters that would be shown in the tooltip and vector y contains variable names. Using the tooltip_table() function, these two vectors were combined into one object which is passed as an argument to the `pointFormat` option of the `hc_tooltip()` function. The tooltip is created, so you can see that in this case there was nothing complicated, but the result is effective and efficient. The rest of the code improves the visual appearance of the graph and seems obvious (due to its syntax).




```
x <- c("Subject", "Arm", "Best Overall Response", "Duration (month)", "First CR/PR",
       "First PD", "Death", "Treatment Disc.")
y <- sprintf("{point.%s}", c("USUBJID", "ARMCD", "RESPONSE", "DURM", "FSTCRPR",
             "FSPD", "DEATH", "LDD"))
tltip <- tooltip_table(x, y)

hchart(ADRS_t, "bar", hcaes(x = USUBJID, y = DURM)) %>%
    hc_add_series(ADRS_t$FSTCRPR, type="scatter", name="First CR/PR", color="green",
                  marker = list(symbol = "circle"), showInLegend = TRUE,
                  tooltip = list(headerFormat = "<b>Subject {point.key} </b> <br>",
                                 pointFormat = "First CR/PR: {point.y}",
                                 valueSuffix = " month")) %>%
...
    hc_xAxis(categories = ADRS_t$USUBJID, title = list(text=NULL)) %>%
    hc_yAxis(title = list(text = "Duration (month)")) %>%
    hc_plotOptions(bar = list(pointWidth = 18, centerInCategory = T)) %>%
    hc_tooltip(useHTML = TRUE, headerFormat = "", pointFormat = tltip) %>%
    hc_title(text = "Swimlaneplot for Responder") %>%
    hc_subtitle(text = "Safety Evaluable Population") %>%
    hc_legend(enabled = TRUE, title = list(text = "Milestones"))
```

FIGURE 8 – SWIMLANE PLOT WITH ELABORATED TOOLTIP

As a corollary, the syntax of the highcharts is straightforward especially for those who are familiar with ggplot2 and can be learned quickly. Indeed, some advanced points require further investigations, but there are lots of articles and free courses on the internet that can provide you with necessary knowledge. As a final point, you could find all interactive data visualizations mentioned in this paper at https://babych.me/slides and test them out by yourself.

CONCLUSION

Plotly and highcharts JavaScript libraries and especially their R wrappers can play a significant role during the analysis of a clinical study or when the analysis is finished and it is necessary to present visually appealing graphs. Implementation of interactivity can enhance the workflow, it provides much more information on the graph maintaining its visual attraction, which consequently leads to enlarging the number of insights. Plotly ability to easily transform static ggplot2 graphs into interactive ones can be extremely valuable and efficient. Highcharts has almost unlimited tooltip potential and its fascinating and stunning graphics can easily draw viewers' attention. At a certain point, the syntax of these packages is convenient and obvious, so it is quite easy to start using them and the results will pay off.

REFERENCES

1. Carson Sievert, 2019, "Interactive web-based data visualization with R, plotly, and shiny". Available at https://plotly-r.com/index.html
2. Michael Eagle, 2016, "Survival Analysis". Available at https://rpubs.com/mjeagle/Surv
3. Joshua Kunst, 2019,. "Using tooltips in unexpected ways". Available at https://jkunst.com/blog/posts/2019-02-04-using-tooltips-in-unexpected-ways/

RECOMMENDED READING

1. Carson Sievert, 2019, "Interactive web-based data visualization with R, plotly, and shiny". Available at https://plotly-r.com/index.html
2. "Plotly R Open Source Graphing Library". Available at https://plotly.com/r/
3. "Create Interactive Web Graphics via 'plotly.js'". Available at https://www.rdocumentation.org/packages/plotly
4. Joshua Kunst, 2020, "First steps on highcharter package" (https://jkunst.com/highcharter/index.html)
5. Nishant Singh, 2018, "Data Visualization with Highcharter in R". (https://www.datacamp.com/community/tutorials/data-visualization-highcharter-r)
6. "Beginners guide to Highchart Visual in R ". (https://www.kaggle.com/nulldata/beginners-guide-to-highchart-visual-in-r/)
7. "Highcharts Demos". (https://www.highcharts.com/demo)
8. Nana Boateng, "Highcharter". (https://rstudio-pubs-static.s3.amazonaws.com/304105_70f2ad540827454e934117e3d90f6c1a.html)
9. Babych Oleksandr, All data visualization shown in this paper available at https://babych.me/slides

CONTACT INFORMATION

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

Oleksandr Babych
Intego Group LLC
19 Hromadyanska Street
Kharkiv 61057, Ukraine
Work Phone: +1 407.512.1006 (ext. 2443) | +380 44.500.7020 (ext. 2449)
Email: oleksandr.babych@intego-group.com, oleksandr.babych1@gmail.com
Web: www.intego-group.com