PharmaSUG China 2021 - Paper SR-079

Interactive web widgets and applications in R

Changming Yang, BeiGene

ABSTRACT

Table, Listing and Figure produced by SAS have been widely used in pharmaceutical industry. But along with the increasing need on data review, exploration and visualization, traditional TLF is not convenient to customize the report format, to change the parameters, to filter or add information without a user-friendly interaction. R is much powerful and flexible on the data processing. We can use R to enhance the interaction and visualization.

This paper will introduce how to design a rich format output by using web widgets within R. A demo on how to generate and customize the output per user's request in an interactive mode will be provided. We will also show a shiny application for exploratory analysis in Fluorescence activated Cell Sorter (FACS).

INTRODUCTION

R is widely used in data exploration and data visualization. There are a lot of packages in R can help you create various output easily and efficiently, including web graphics, maps, tables and so on.

This paper focuses on the interactive web widgets, which is called HTML Widgets in R. The HTML Widgets framework, which is mainly based on the R package **htmlwidgets**, can interface JavaScript libraries and allow user to create interactive applications, such as interactive graphics, maps, and tables.

HTML Widgets can be:

- Used at the R console for data analysis just like conventional R plots.
- Embedded within R Markdown documents.
- Incorporated into Shiny web applications.
- Saved as standalone web pages for ad-hoc sharing via Email and Dropbox, etc.

There are also many R packages such as **plotly**, **leaflet**, and **DT** which are developed based on the HTML widgets framework, can allow R users to create JavaScript applications much easier.

ATTACHING REQUIRED PACKAGES

```
library(plotly)
library(leaflet)
library(DT)
library(crosstalk)
library(dplyr)
library(stringr)
library(htmltools)
library(survival)
library(survminer)

set.seed(233)
```

INTERACTIVE GRAPHICS WITH PLOTLY

plotly is an R package for creating interactive web-based graphics via the open source JavaScript graphing library plotly.js. You can create interactive and dynamic web graphics via plotly in R.

SCATTER PLOT

You can create interactive and dynamic web graphics by a few lines of code:

```
plot_ly(
  data = iris,
  x = ~Sepal.Length,
  y = ~Petal.Length,
  type = 'scatter', mode = 'markers',
  text = ~str_glue(
    'Species: {Species}',
    'Sepal.Length: {Sepal.Length}',
    'Sepal.Width: {Sepal.Width}',
    'Petal.Length: {Petal.Length}',
    'Petal.Width: {Petal.Width}',
    .sep = '\n'
  ),
  hoverinfo = 'text',
  color = ~Species
)
```

Data set iris from package base is used to generate the scatter plot. Note that all the data sets in this paper can be found from the attached packages (including package base), you can use function help() such as help(iris) in RStudio IDE to get more information about the data set.

In Figure 1, you can get the hover information of each point and click to select different group in legend.

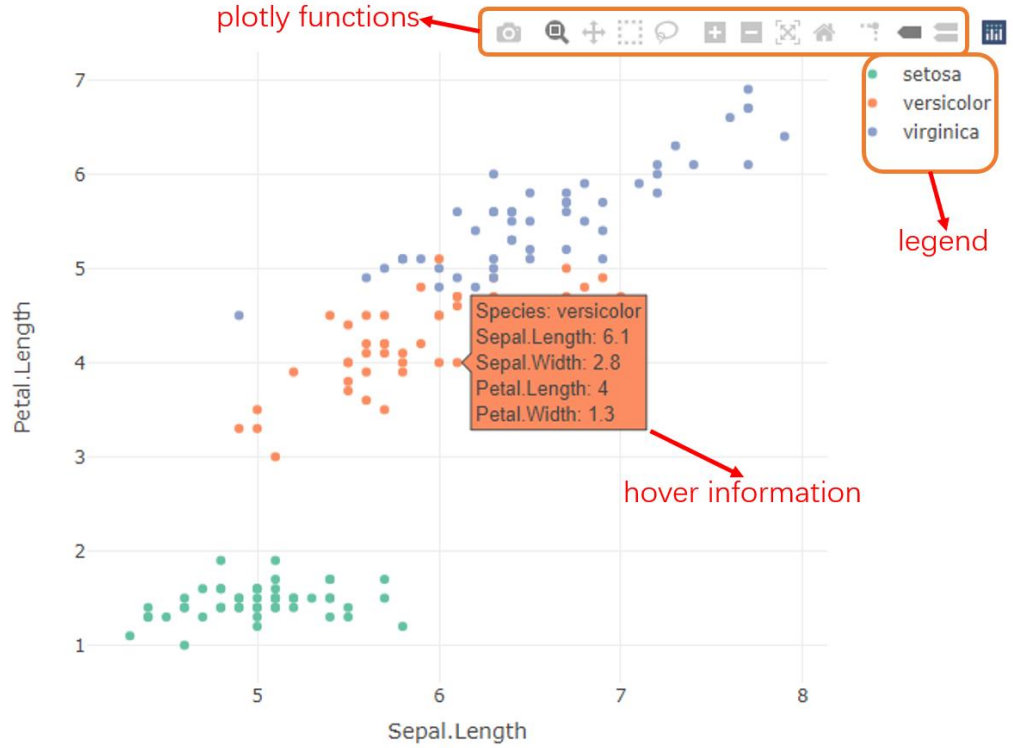


Figure 1: scatter plot

LINE PLOT

You can use function ggplotly() to transform ggplot2 graph into plotly graph:

```
fit <- survfit(Surv(time, status) ~ sex, data = lung)
p <- ggsurvplot(fit, data = lung)

ggplotly(p[[1]])
```

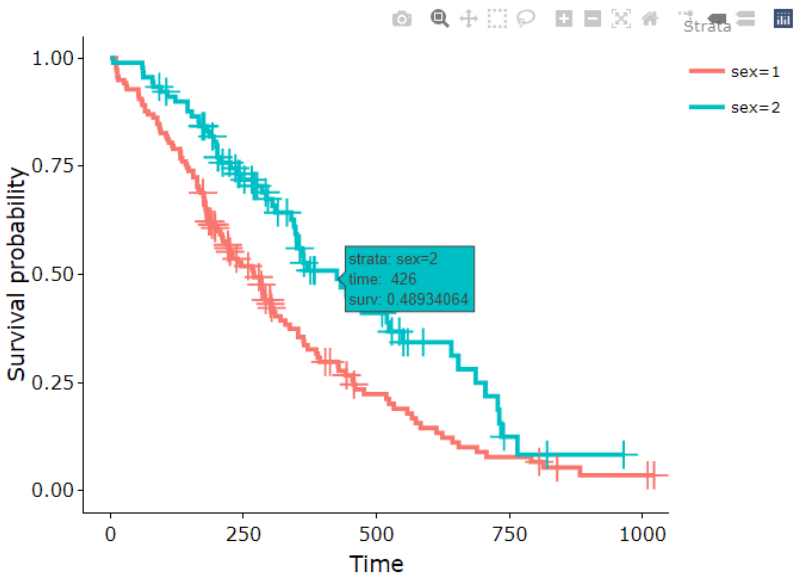


Figure 2: survplot

BAR PLOT

```
data <- diamonds %>% count(cut, clarity)

plot_ly(data = data, x = ~cut, y = ~n, type = 'bar', color = ~clarity)
```



Figure 3: bar plot

PIE PLOT

```
data <- diamonds %>% count(cut)

plot_ly(data = data, labels = ~cut, values = ~n, type = 'pie')
```

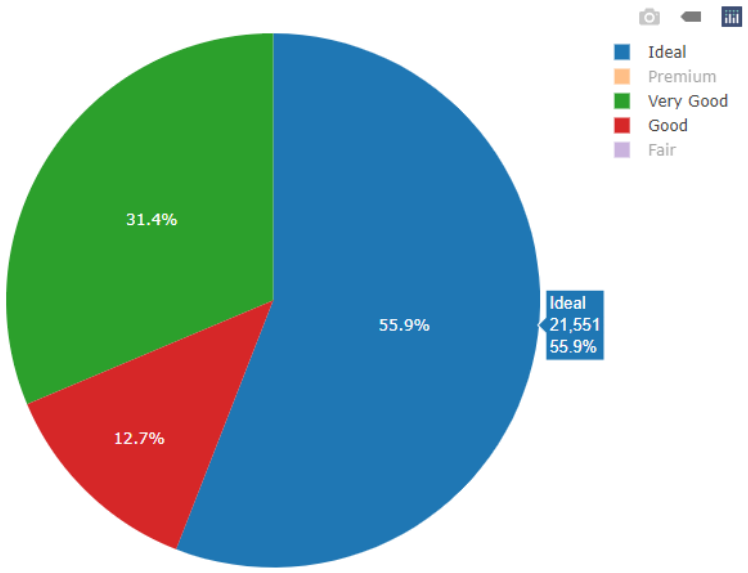


Figure 4: pie plot

3D SCATTER PLOT

```
data <- mtcars %>%
  mutate(
    am = if_else(am == 1, 'Manual', 'Automatic'),
    am = as.factor(am)
  )

plot_ly(
  data = data,
  x = ~wt,
  y = ~hp,
  z = ~qsec,
  type = "scatter3d", mode = "markers",
  text = ~str_glue(
    'Weight: {wt}',
    'Gross horsepower: {hp}',
    '1/4 mile time: {qsec}',
    .sep = '\n'
  ),
  hoverinfo = 'text+name',
  color = ~am, colors = c("#BF382A", "#0C4B8E")
) %>%
  layout(scene = list(
    xaxis = list(title = "Weight"),
    yaxis = list(title = "Gross horsepower"),
    zaxis = list(title = "1/4 mile time")
  ))
```

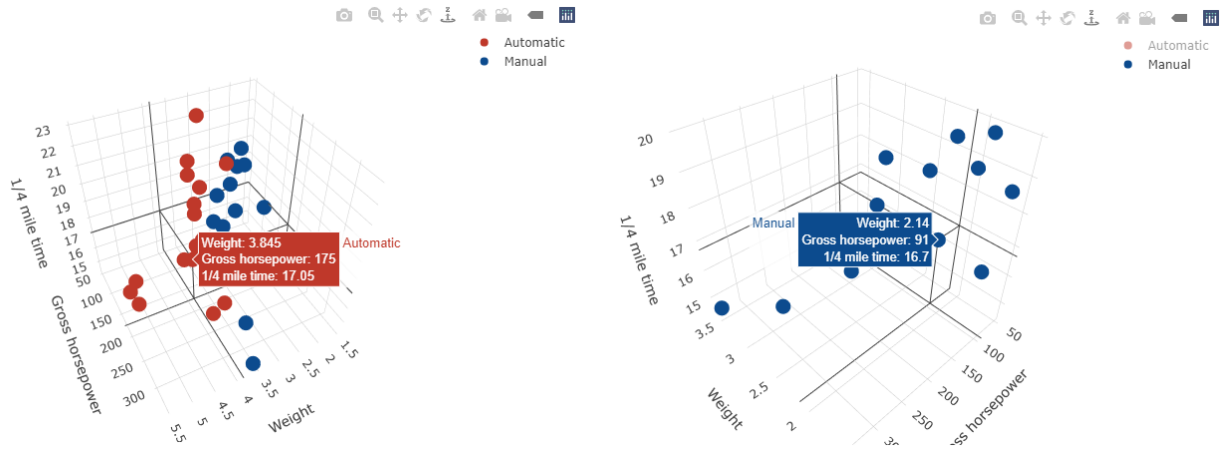


Figure 5: 3D scatter plot

INTERACTIVE MAPS WITH LEAFLET

Leaflet is one of the most popular open-source JavaScript libraries for interactive maps. The R package **leaflet** makes it easy to integrate and control Leaflet maps in R.

MAP WITH SPATIAL DATA

You can mark the data points on the map when the latitude and longitude are given by the data set:

```
leaflet(quakes[1:100, ]) %>%
  addTiles() %>%
  addMarkers(
    lng = ~long, lat = ~lat,
    label = ~as.character(mag)
  )
```

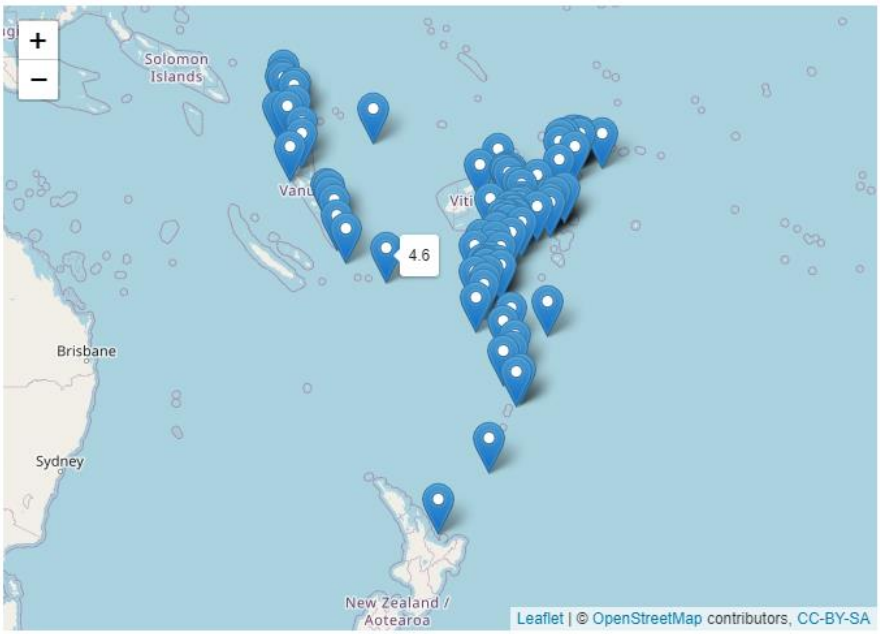


Figure 6: map with spatial data

When there are a large number of markers in the same area, you can cluster them to get a better interaction:

```
leaflet(quakes) %>%
  addTiles() %>%
  addMarkers(
    lng = ~long, lat = ~lat,
    label = ~as.character(mag),
    clusterOptions = markerClusterOptions()
  )
```

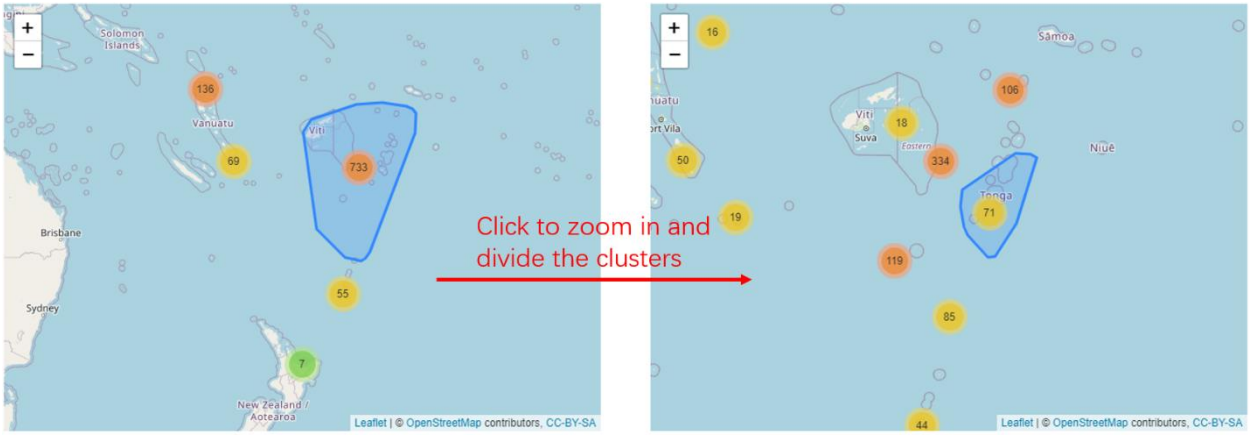


Figure 7: marker clusters

INTERACTIVE TABLES WITH DT

The R package **DT** provides an R interface to the JavaScript library DataTables. R data objects (matrices or data frames) can be displayed as tables on HTML pages, and DataTables provides filtering, pagination, sorting, and many other features in the tables.

BASIC USAGE

Here is a simple datatable with zero configuration, it contains pagination, sorting and searching:

```
datatable(iris)
```

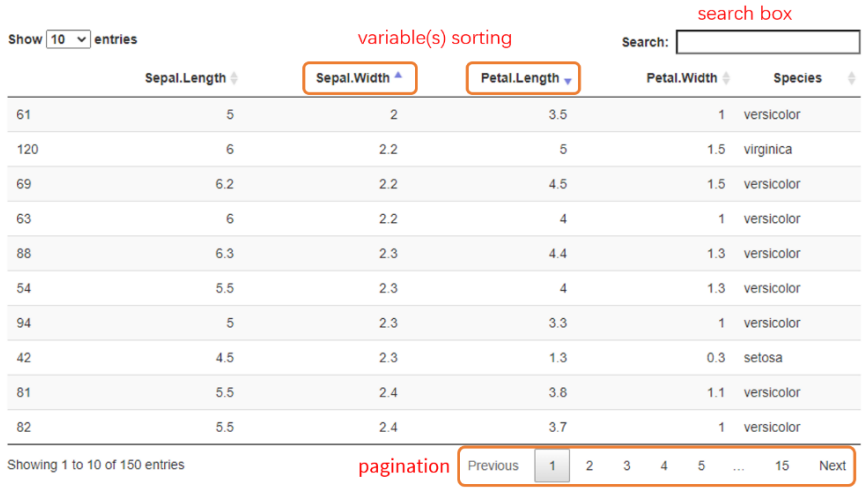


Figure 8: basic datatable

You can add column filters to each variable and highlight search result:

```
datatable(
  iris,
  filter = 'top',
  options = list(
    searchHighlight = TRUE,
    search = list(search = 'sa')
  )
)
```

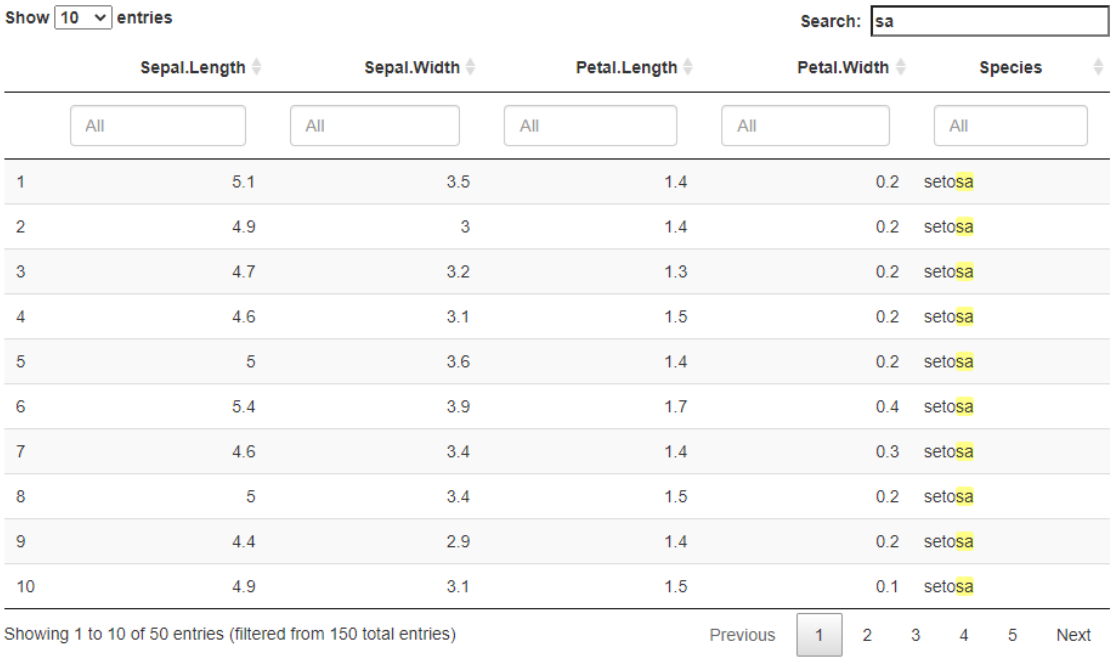


Figure 9: column filters and highlight search

ADD BUTTONS TO TABLE

You can add buttons to copy the table to clipboard and save the table as CSV/XLS/PDF:

```
datatable(
  iris,
  extensions = 'Buttons',
  options = list(
    dom = 'Bfrtip',
    buttons = list(
      'copy',
      list(
        extend = 'collection',
        buttons = c('csv', 'excel', 'pdf'),
        text = 'Download'
      )
    )
  )
)
```



Figure 10: copy and download buttons

Buttons can also be used to select columns:

```
datatable(
  iris, rownames = FALSE,
  extensions = 'Buttons',
  options = list(
    dom = 'Bfrtip',
    buttons = list(
      list(
        extend = 'colvis',
        postfixButtons = list('colvisRestore')
      )
    )
  )
)
```



Figure 11: column selection button

Besides column selection, you can even click and drag the table header to reorder the columns:

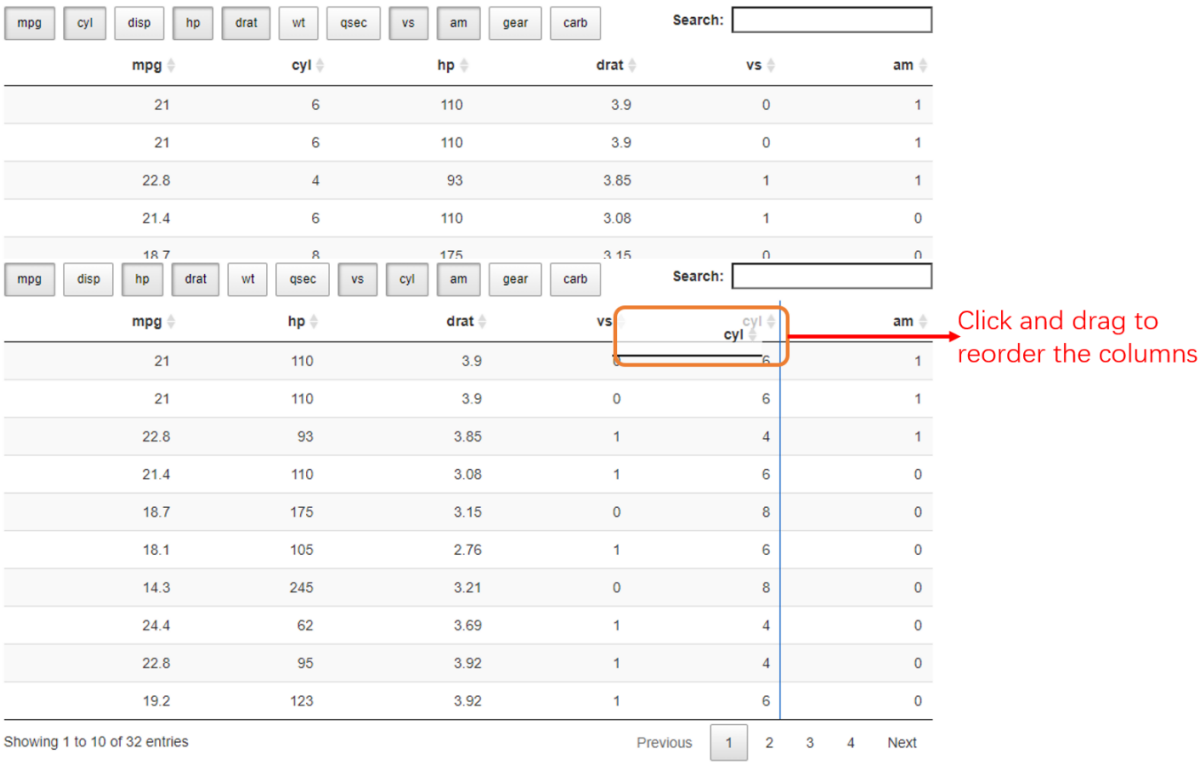


Figure 12: column reorder

SCROLLING LAYOUT

You can choose scrolling layout to insert a big table (with a lot of observations and variables) into a fixed page:

```
datatable(
  mtcars,
  extensions = 'Scroller',
  options = list(
    deferRender = TRUE,
    scrollY = 200,
    scroller = TRUE
  )
)
```

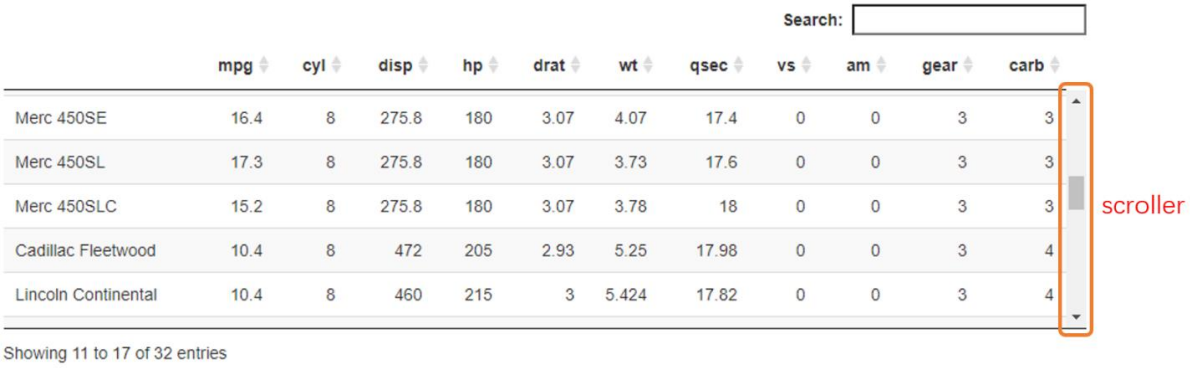


Figure 13: datatable with scrollY

You can also add fixed columns:

```
m = as.data.frame(round(matrix(rnorm(100), ncol = 20), 5))

datatable(
  m, rownames = FALSE,
  extensions = 'FixedColumns',
  options = list(
    dom = 't',
    scrollX = TRUE,
    fixedColumns = list(
      leftColumns = 2,
      rightColumns = 1
    )
  )
)
```



Figure 14: scrollX and fixed columns

CROSS-WIDGET INTERACTION WITH CROSSTALK

Crosstalk is an add-on to the htmlwidgets package. It extends htmlwidgets with a set of classes, functions, and conventions for implementing cross-widget interactions (currently, linked brushing and filtering).

LINKING BETWEEN HTMLWIDGETS

Crosstalk makes it easy to link multiple (Crosstalk-compatible) HTML widgets within an R Markdown page or Shiny app. You can combine a map (from leaflet) and a table (from DT) based on dataset quakes, and add a slider on it:

```
shared_quakes1 <- SharedData$new(quakes[sample(nrow(quakes), 100),])

browsable(tagList(
  filter_slider(
    "mag", "Magnitude",
    shared_quakes1,
    column = ~mag,
    step = 0.1, width = 250
  ),
  bscols(
    leaflet(shared_quakes1) %>%
```

```
      addTiles() %>%
      addMarkers(
        lng = ~long, lat = ~lat,
        label = ~mag,
        options = markerOptions(zIndexOffset = -999)
      ),
    datatable(
      shared_quakes1,
      extensions = "Scroller",
      style = "bootstrap", class = "compact",
      width = "100%",
      options=list(deferRender = TRUE, scrollY = 300, scroller = TRUE)
    )
  )
))
```

You can manipulate the slider to filter the data set and click rows to highlight the data points on the map, or use the selection box in the map to filter the datatable.



Figure 15: linking between map and table

Linking between map and scatter plot:

```
shared_quakes2 <- SharedData$new(quakes[sample(nrow(quakes), 100),])

bscols(
  leaflet(shared_quakes2) %>%
    addTiles() %>%
    addMarkers(
      lng = ~long, lat = ~lat,
      label = ~mag,
      options = markerOptions(zIndexOffset = -999)
    ),
  plot_ly(
    shared_quakes2,
    x = ~depth, y = ~mag,
    type = 'scatter', mode = 'markers'
  ) %>%
    layout(dragmode = 'lasso') %>%
    highlight(on = 'plotly_selected', off = 'plotly_deselect')
)
```

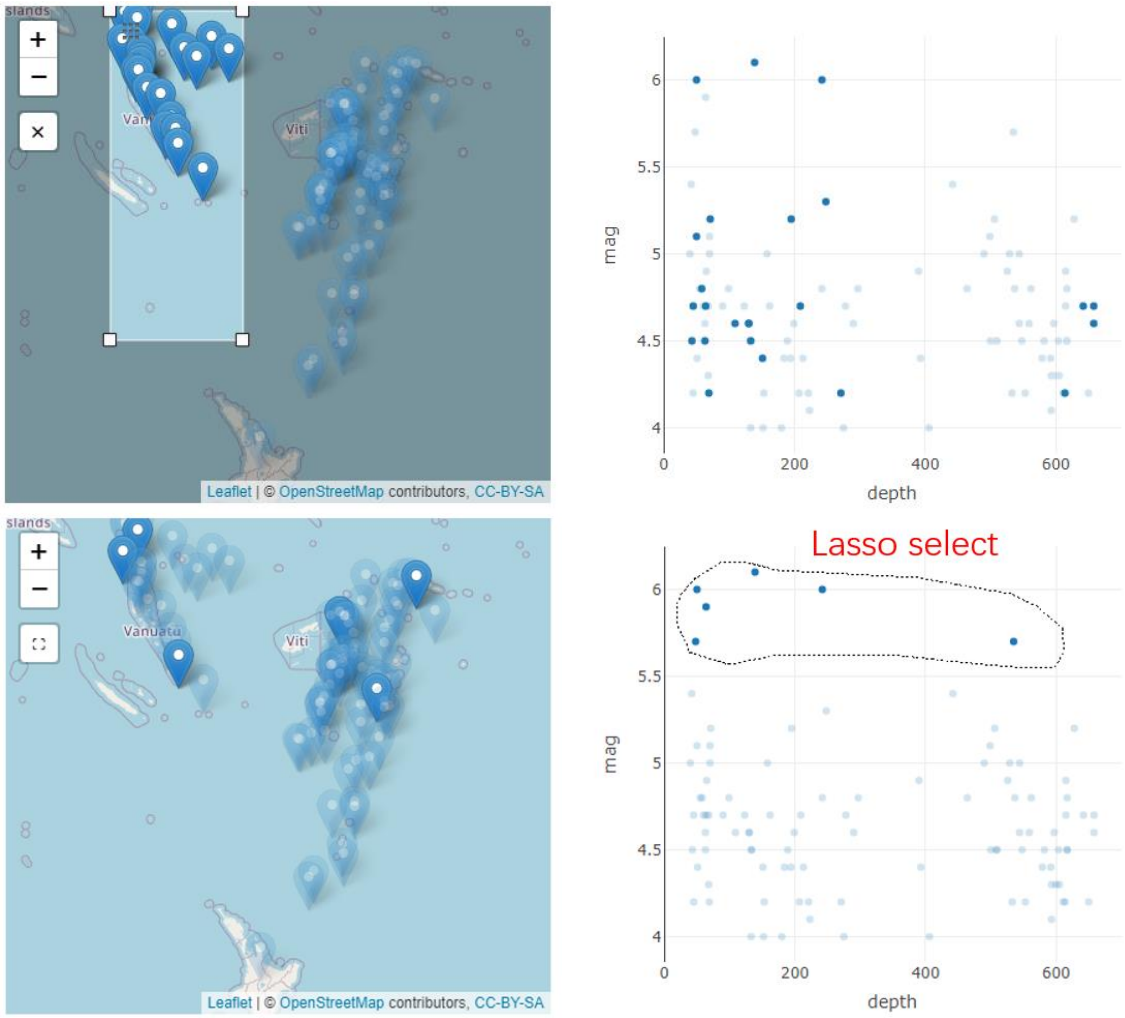


Figure 16: linking between map and plot

Linking between scatter plot and table:

```
shared_iris <- SharedData$new(iris, group = 'row number')

bscols(
  plot_ly(
    shared_iris,
    x = ~Petal.Length, y = ~Petal.Width,
    color = ~Species,
    type = 'scatter', mode = 'markers'
  ) %>%
    highlight(on = 'plotly_selected', off = 'plotly_deselect', selectize = T),
  datatable(
    shared_iris,
    extensions = "Scroller",
    style = "bootstrap", class = "compact",
    width = "100%",
    options=list(scrollX = TRUE, deferRender = TRUE, scrollY = 300, scroller = TRUE)
  )
)
```

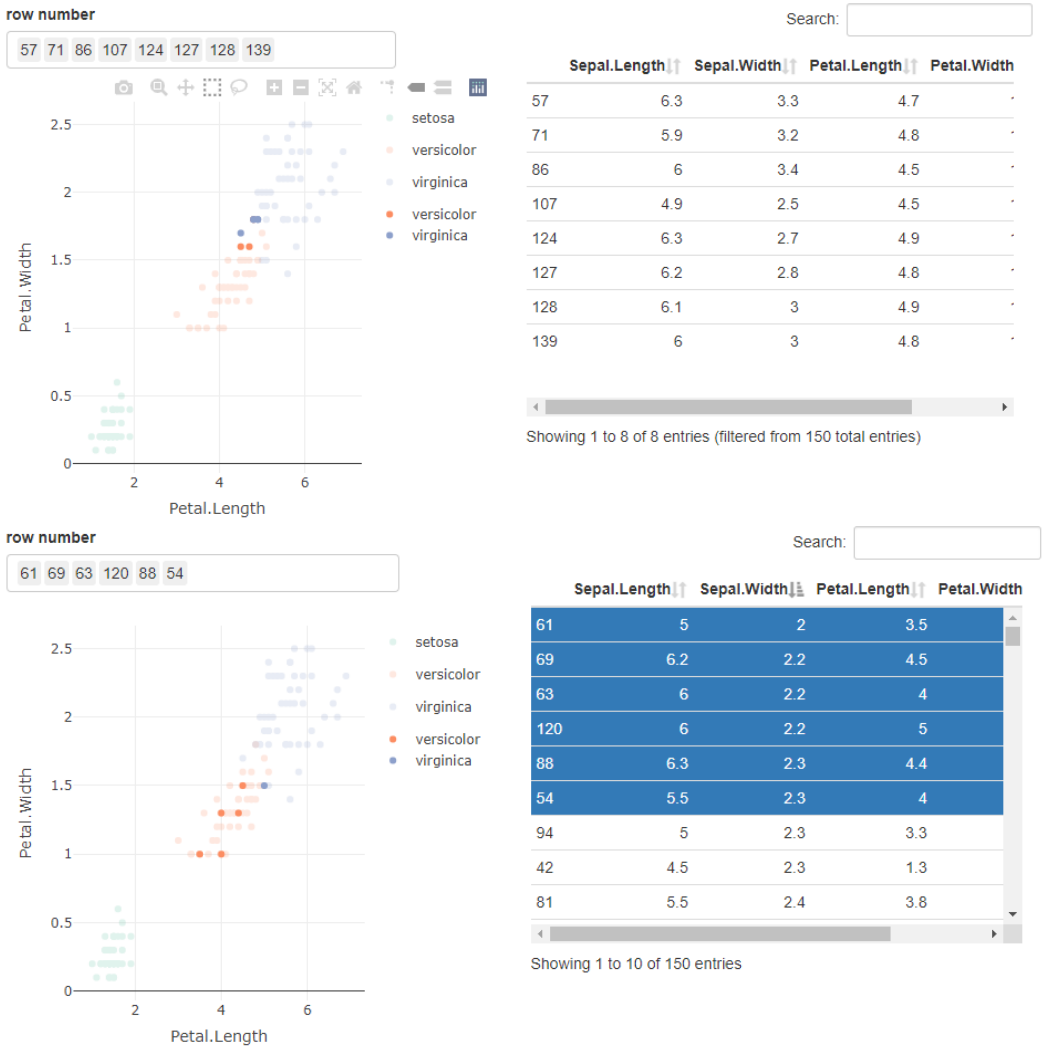


Figure 17: linking between plot and table

LINKING BETWEEN PLOTS

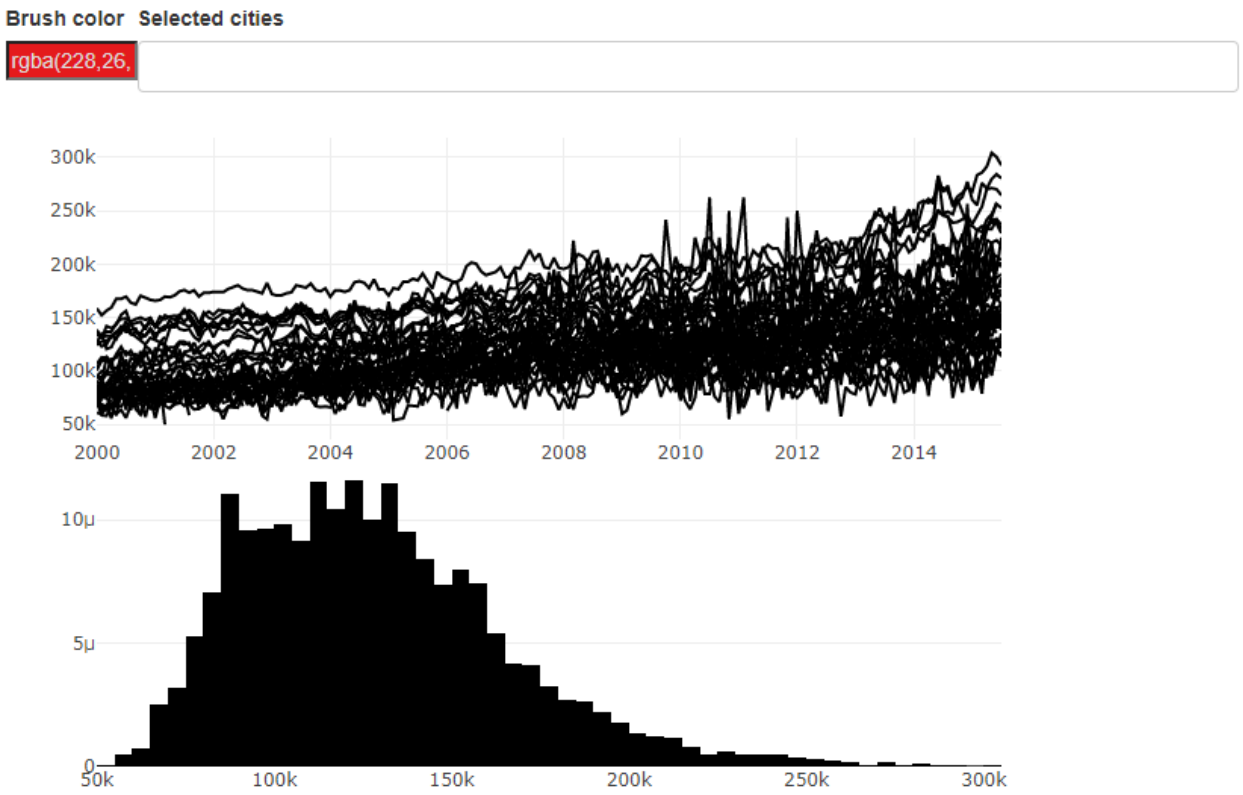
Multi-plot linking is a powerful technique in interactive plot. You can write R code to pose graphical queries that operate entirely client-side in a web browser.

You can combine a line plot with median house price in each city over time, and a histogram plot of median house price in all cities:

```
shared_txhousing <- SharedData$new(txhousing, ~city, group = 'Selected cities')

subplot(
  plot_ly(shared_txhousing, color = I("black")) %>%
    group_by(city) %>%
    add_lines(x = ~date, y = ~median),
  plot_ly(shared_txhousing, color = I("black")) %>%
    group_by(city) %>%
    add_histogram(x = ~median, histnorm = "probability density"),
  nrows = 2
) %>%
  layout(barmode = "overlay", showlegend = FALSE) %>%
  highlight(
    on = 'plotly_click',
    off = 'plotly_doubleclick',
    dynamic = TRUE,
    selectize = TRUE,
    selected = attrs_selected(opacity = 0.3)
  )
```

You can get the histogram of all the data points, when you click one line to select one city of the data, a new highlight sub-histogram of this city will be shown. It's easier for users to compare the city specific distribution to the overall distribution.



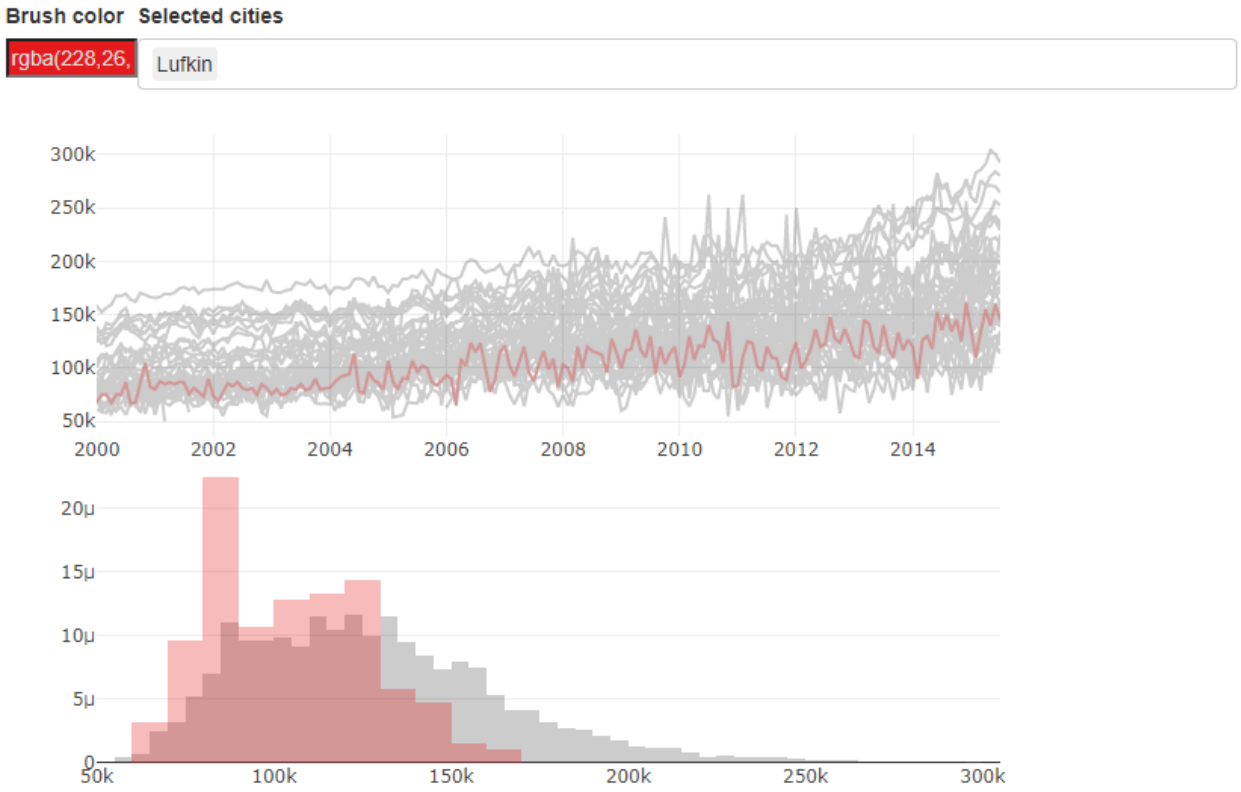


Figure 18: linking house prices as a function of time with their probability density estimates.

Here is an example shows line plots with median house price in four cities over months in each year:

```
cities <- c("Galveston", "Midland", "Odessa", "South Padre Island")

txhousing %>%
  select(city, year, month, median) %>%
  filter(city %in% cities) %>%
  group_by(city) %>%
  do(
    p = SharedData$new(., ~year, group = "txhousing-trellis") %>%
      plot_ly(showlegend = FALSE) %>%
      group_by(year) %>%
      add_lines(
        x = ~month, y = ~median, text = ~year,
        hoverinfo = "text"
      ) %>%
      add_annotations(
        text = ~unique(city),
        x = 0.5, y = 1,
        xref = "paper", yref = "paper",
        xanchor = "center", yanchor = "bottom",
        showarrow = FALSE
      )
  ) %>%
  subplot(
    nrows = 2, margin = 0.05,
    shareY = TRUE, shareX = TRUE, titleY = FALSE
```

```
  ) %>%
  highlight(on = 'plotly_click', off = 'plotly_doubleclick')
```

you can click one line to select a year to compare years both within and across cities.

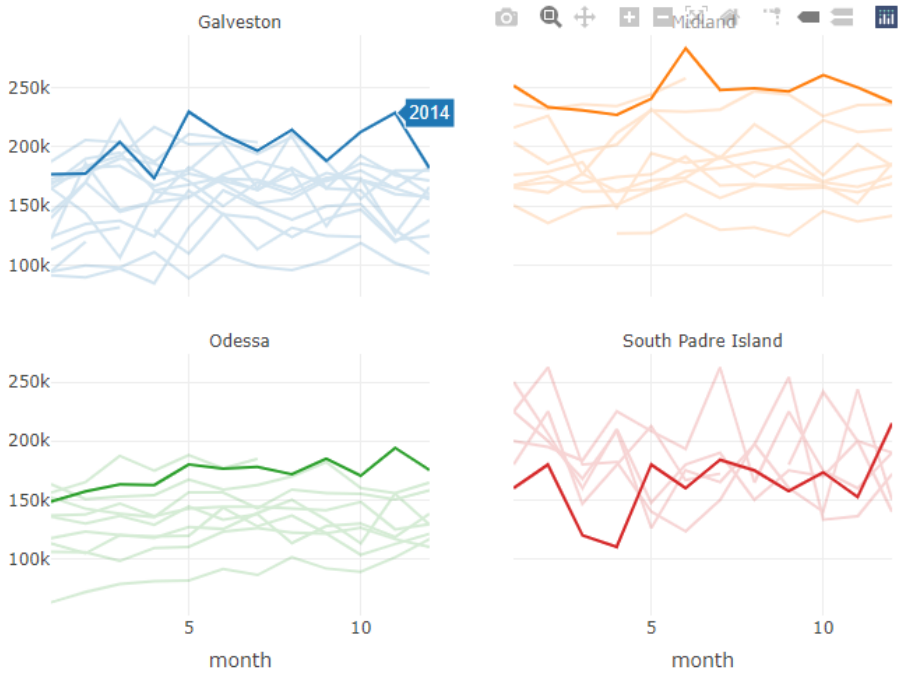


Figure 19: linking between line plots

Another example which demonstrates ways to display binned/aggregated selections:

```
shared_mtcars <- SharedData$new(mtcars)

scatter <- plot_ly(
  shared_mtcars,
  x = ~mpg, y = ~disp,
  height = 300, width = 900
) %>%
  add_markers(color = I("black"))

box <- plot_ly(shared_mtcars, y = ~disp, color = I("black")) %>%
  add_boxplot(name = " ")

violin <- plot_ly(shared_mtcars, y = ~disp, color = I("black")) %>%
  add_trace(type = "violin", name = " ")

hist <- plot_ly(shared_mtcars, x = ~factor(cyl)) %>%
  add_histogram(color = I("black"))

subplot(scatter, box, violin, shareY = TRUE, titleX = TRUE, titleY = TRUE) %>%
  subplot(hist, widths = c(.75, .25), titleX = TRUE, titleY = TRUE) %>%
  layout(
    barmode = "overlay",
    title = "Click and drag scatterplot",
    showlegend = FALSE
  ) %>%
  highlight(on = "plotly_selected", off = 'plotly_deselect')
```

You can click and drag on the scatter plot to filter data, then plotly will help generating the new summary plots on the top of the original summary plots.

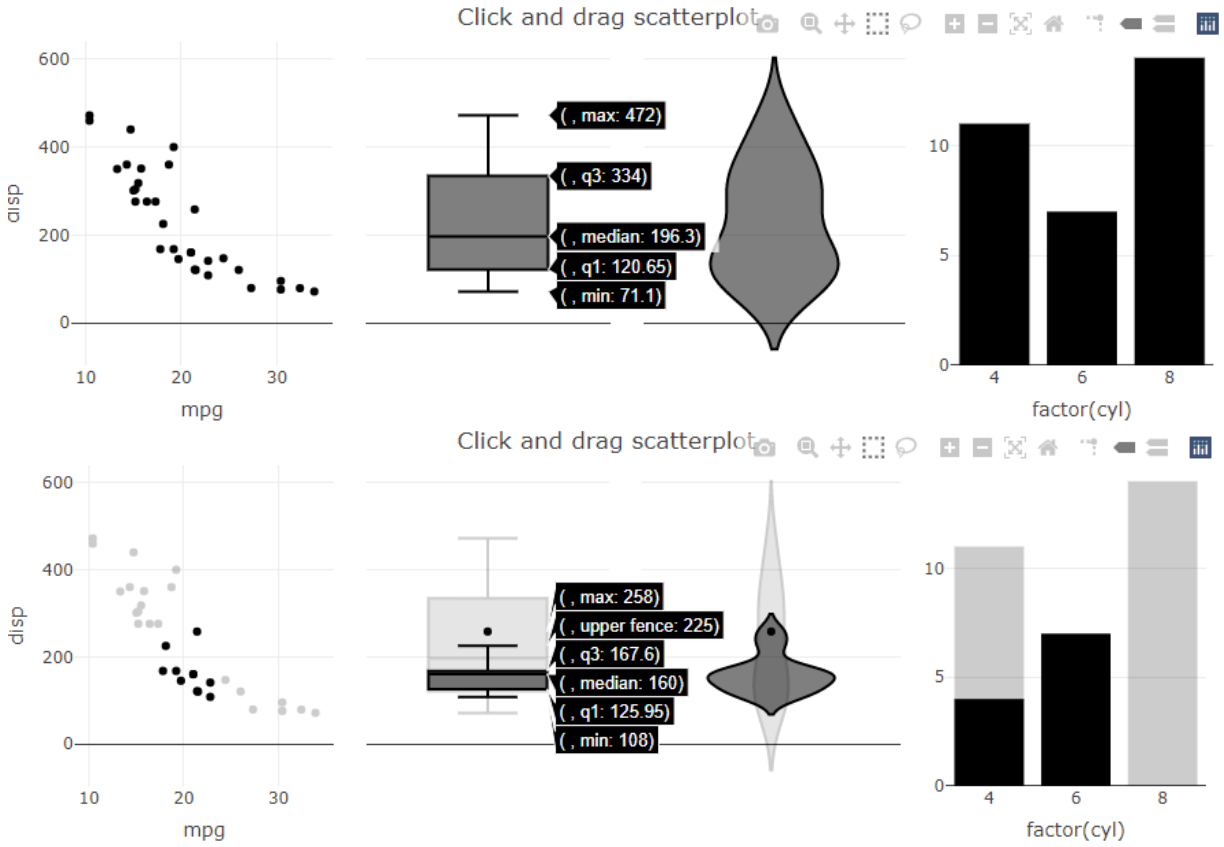


Figure 20: statistical query

INTERACTIVE WEB WIDGETS IN SHINY

Shiny is one of the most famous packages in R to help programmer build web application conveniently. As a web-based output, HTML widgets are great for embedding in shiny applications. Here we provide a part of a shiny application for exploratory analysis in Fluorescence activated Cell Sorter (FACS).

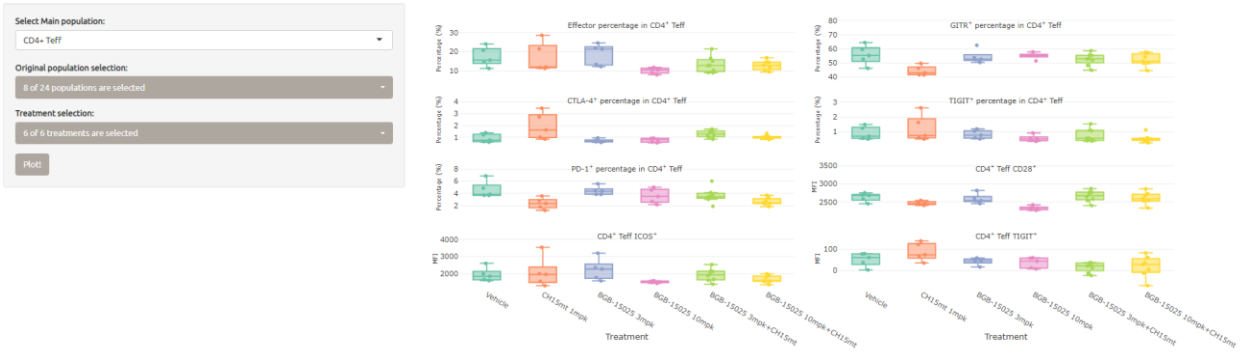


Figure 21: FACS boxplot

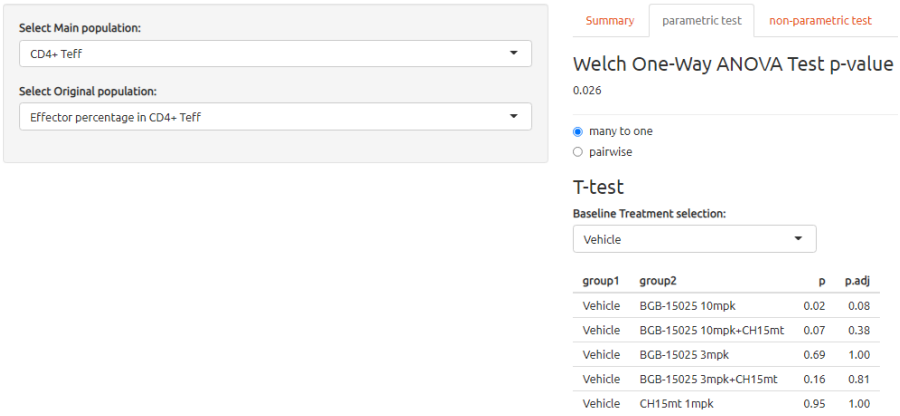


Figure 22: FACS statistical testing

CONCLUSION

Primary objective of this paper is to share the example and usage of HTML widgets. With the support of several R packages, programmer can generate interactive output by writing a simple piece of code without any knowledge of JavaScript.

REFERENCES

- htmlwidgets for R
  http://www.htmlwidgets.org/
- Interactive web-based data visualization with R, plotly, and shiny
  https://plotly-r.com/
- Leaflet for R
  http://rstudio.github.io/leaflet/
- DT: An R interface to the DataTables library
  https://rstudio.github.io/DT/
- Crosstalk
  http://rstudio.github.io/crosstalk/
- Shiny
  https://shiny.rstudio.com/

ACKNOWLEDGMENTS

Thanks for the inspiring and guidance from SRM and Young Voice working group of BeiGene. Thanks to Chun Yi, Jie Liu, Bingting Tao and Luqiong Miao for the encouragement and support.

CONTACT INFORMATION

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

<Name> Changming Yang
<Enterprise> BeiGene
<E-mail> changming.yang@beigene.com