D3 Visualizations in R Cheat Sheet

By *Zheng Wu & Qiran Li*

The main reference is based on the official guideline from rstudio `https://rstudio.github.io/r2d3/`.

Why D3

The main reason is to **improve and extend the interactive/dynamic ability.** When you want to go further in data visualization, the Javascript library D3 will provide a comprehensive support for R.

How to use D3 in R?

The `r2d3` package provides a suitable of tools for using D3 visualizations with R. Make sure you use the html output and start by,

```
---
output: html_document
---
```{r}
# install and import the required library
if(!require('r2d3')) install.packages('r2d3')
library('r2d3')
```
```

Include D3 in RMarkdown

1. Call `r2d3` function in `R`.

```
```{r}
r2d3(data=c(0.3, 0.6, 0.8, 0.95, 0.40,
    0.20), script = "barchart.js")
```
```

2. First include the data in R.

```
```{r setup}
bars <- c(10, 20, 30)
```
```

And then use directly the `d3` visualization code to draw the data.

```
```{d3 data=bars, options=list(color = '
    orange')}
svg.selectAll('rect')
  .data(data)
  .enter()
    .append('rect')
      .attr('width', function(d) {
          return d * 10; })
      .attr('height', '20px')
      .attr('y', function(d, i) { return
          i * 22; })
      .attr('fill', options.color);
```
```

Combine with Shiny

Apart from the previous two ways, we could also combine the d3 style with the shiny framework to generate dynamic plotting in R. The `renderD3()` and `d3Output()` functions enable you to include D3 visualizations within Shiny applications.

```
library(shiny)
library(r2d3)

ui <- fluidPage(
  inputPanel(
    sliderInput("bar_max", label = "Max:",
                min = 0.1, max = 1.0, value =
                    0.2, step = 0.1)
  ),
  d3Output("d3")
)

server <- function(input, output) {
  output$d3 <- renderD3({
    r2d3(
      runif(5, 0, input$bar_max),
      script = system.file("examples/baranims
          .js", package = "r2d3")
    )
  })
}

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

Sizing

1. Change the the `width` and `height` variables as is defined by the htmlwidget.

```
var barHeight = Math.floor(height / data.
    length);

svg.selectAll('rect')
  .data(data)
  .enter().append('rect')
    .attr('width', function(d) { return d *
        width; })
    .attr('height', barHeight)
    .attr('y', function(d, i) { return i *
        barHeight; })
    .attr('fill', 'steelblue');
```

2. via the `sizing` argument to `r2d3()`.

D3 Variables

Variable Name	Physical Meaning
data	The R data converted to JavaScript
svg	The svg container for the visualization
width	The current width of the container
height	The current height of the container
options	Additional options provided by the user
theme	Colors for the current theme

R to D3-friendly Data Conversion

R objects provided D3 visualizations conversion to JSON using the jsonlite::toJSON() function, same default serialization behavior as Shiny and htmlwidgets:

```
vjsonlite::toJSON(
    dataframe = "columns", null = "null", na
        = "null", auto_unbox = TRUE, digits =
         getOption("shiny.json.digits", 16),
        use_signif = TRUE, force = TRUE,
        POSIXt = "ISO8601", UTC = TRUE,
        rownames = FALSE, keep_vec_names =
        TRUE, json_verabitm = TRUE
    )
```

Here is an example:

```
{
  "Sepal.Length": [5.1, 4.9, 4.7],
  "Sepal.Width": [3.5, 3, 3.2],
  "Petal.Length": [1.4, 1.4, 1.3],
  "Petal.Width": [0.2, 0.2, 0.2],
  "Species": ["setosa", "setosa", "setosa"]
}
```

After HTMLWidgets.dataframeToD3(), it is:

```
[
  { "Sepal.Length": 5.1, "Sepal.Width": 3.5,
      "Petal.Length": 1.4, "Petal.Width":
      0.2, "Species": "setosa"},
  { "Sepal.Length": 4.9, "Sepal.Width": 3, "
      Petal.Length": 1.4, "Petal.Width": 0.2,
       "Species": "setosa" },
  { "Sepal.Length": 4.7, "Sepal.Width": 3.2,
      "Petal.Length": 1.3, "Petal.Width":
      0.2, "Species": "setosa" }
]
```