R package dcr documentation

Xiangdong Gu

November 18, 2015

Contents

Introduction

Dc.js and crossfilter.js are useful for constructing interactive visualizations and exploratory analysis. http://dc-js.github.io/dc.js/ provides documentations and a lot of examples about this javascript library.

This project integrates their features and capabilities into the Shiny to build interactive multi-dimensional shiny apps. One example shiny app built with this package to reproduce some of the charts found in the homepage of dc.js is https://huguyi10.shinyapps.io/stocks. You can slice and dice the data by clicking slices of the pie-charts, clicking bars of the row charts or brushing on the bar charts.

Getting Started

Installation

We can install the R package from our github page. We should also install R package shiny if it has not been insalled.

```
if (!require(devtools)) install.packages("devtools")
devtools:::install_github("massmutual/dcr")
```

Build dc Charts

We demonstrate the use of the package by a simple example.

```
library(dcr)
# create chart object
mydcr <- dcr(mtcars)

# pie chart counting by number of cylinders
chart_cyl <- dcrchart(type = "pieChart", id = "chart1", dimension = "cyl",
                      reduce = reduceCount(), width = 250, height = 250)

# line chart calculation average horse power by number of forward gears
chart_gear <- dcrchart(type = "lineChart", id = "chart2", dimension = "gear",
                    reduce = reduceMean("hp"), width = 400, height = 250)
```

```
# row chart couting by number of carburetors
chart_carb <- dcrchart(type = "rowChart", id = "chart3", dimension = "carb",
                       reduce = reduceCount(), width = 300, height = 250)

# view the charts and interatively slice and dice to explore the data
mydcr + chart_cyl + chart_gear + chart_carb
```

There are two essential functions we need to use to build dc charts. `dcr()` function is used to load the data and create a dcr chart object. `dcrchart()` function is used to create a chart that can be added to the dcr chart object using "+" operator as we see in the example. In the minimum, we should supply the following arguments to the `dcrchart()` function call:

- `type`: the type of the chart, for example barChart, pieChart, rowChart, lineChart
- `id`: the id of the chart. This is used in the ui.R in shiny to layout the charts.
- `dimension`: variable name in the data used for dimension. This is corresponding to the key in map-reduce. For example in the chart_gear in above example, it does counting for each unique value of the dimension variable gear.
- `reduce`: the reduce function in map-reduce for each value (key) of the dimension variable. Commonly used reduce functions are `reduceCount()`, `reduceSum()`, `reduceMean()`.
- `width`: the width of the chart in pixel
- `height`: the height of the chart in pixel

Use with R Shiny

The dc chart object is ready to be incorporated in the R Shiny app.

We first render the dc charts using `render_Chart` function in the `server.R`. Here is an example,

```
library(shiny)
library(dcr)
shinyServer(function(input, output) {
  output$example <- renderChart({
    mydcr <- dcr(mtcars)
    chart_cyl <- dcrchart(type = "pieChart", id = "chart1", dimension = "cyl",
                          reduce = reduceCount(), width = 250, height = 250)
    chart_gear <- dcrchart(type = "lineChart", id = "chart2", dimension = "gear",
                           reduce = reduceMean("hp"), width = 400, height = 250)
    chart_carb <- dcrchart(type = "rowChart", id = "chart3", dimension = "carb",
                           reduce = reduceCount(), width = 300, height = 250)
    mydcr + chart_cyl + chart_gear + chart_carb
  })
})
```

We layout each chart in `ui.R`. We first output the dc chart object using `chartOutput` function. For each chart, we create a division using `div` function with id corresponding to the chart id that we want to place. We can arrange the charts using shiny's layout funcitons such as `fluidRow` and `column`. Below is an example,

```
library(shiny)
library(dcr)
shinyUI(fluidPage(
  chartOutput("example"),
```

```
  fluidRow(column(6, div(id = "chart1")),
           column(6, div(id = "chart3"))),
  fluidRow(column(6, div(id = "chart2")))
))
```

Reference

Dimension variable type

Chart types

As of version 0.1, `pieChart`, `barChart`, `rowChart`, and `lineChart` are well supported. Other chart types can also be produced by this package with some addional workaround.

Specify javascript function as a chart option

In many situations, we need specify a javascript function as a chart option. This can be achieved by wrapping the string of javascript function definition with `dc_code` function. For example, `dc_code("function(d) {return d.key;}")`.

Chart options

Below are chart options we can specify in `dcrchart` function to customize the chart. More details of description about these options can be found at https://github.com/dc-js/dc.js/blob/master/web/docs/api-1.6.0.md. Here we focus on how to specify the options in the R function call.

keyAccessor

> Set or get the key accessor function. Key accessor function is used to retrieve key value in crossfilter group. Key values are used differently in different charts, for example keys correspond to slices in pie chart and x axis position in grid coordinate chart.

We need to specify a javascript function.

```
keyAccessor = dc_code("function(d) {return d.key;}")
```

valueAccessor

> Set or get the value accessor function. Value accessor function is used to retrieve value in crossfilter group. Group values are used differently in different charts, for example group values correspond to slices size in pie chart and y axis position in grid coordinate chart.

We need to specify a javascript function.

```
valueAccessor = dc_code("function(d) {return d.value.percentGain;}")
```

label

Set or get the label function. Chart class will use this function to render label for each child element in the chart, i.e. a slice in a pie chart or a bubble in a bubble chart. Not every chart supports label function for example bar chart and line chart do not use this function at all.

We need to specify a javascript function. A helper function `label_keyvalue` can be used to generate some commonly used labels.

```
label = dc_code("function(d) {return d.key + '(' + d.value + ')';}")
```

renderLabel

Turn on/off label rendering

It takes a value either TRUE or FALSE.

```
renderLabel = FALSE
```

title

Set or get the title function. Chart class will use this function to render svg title(usually interrupted by browser as tooltips) for each child element in the chart, i.e. a slice in a pie chart or a bubble in a bubble chart. Almost every chart supports title function however in grid coordinate chart you need to turn off brush in order to use title otherwise the brush layer will block tooltip trigger.

We need to specify a javascript function.

```
title = dc_code("function(d) {return d.key + '(' + d.value + ')';}")
```

renderTitle

Turn on/off title rendering

It takes a value either TRUE or FALSE.

```
renderTitle = FALSE
```

legend

Attach dc.legend widget to this chart. Legend widget will automatically draw legend labels based on the color setting and names associated with each group.

We use `dc_legend` function to create a legend.

```
legend = dc_legend(x = 400, y = 10, itemHeight = 13, gap = 5)
```

Helper Functions

Loading Time Optimization