rCharts Documentation

Release 0.1.0

Ramnath Vaidyanathan

April 07, 2014

rCharts is an R package to create, customize and publish interactive javascript visualizations from R using a familiar lattice style plotting interface.

CHAPTER

ONE

QUICK START

You can install rCharts from github using the devtools package

```
require(devtools)
install_github('rCharts', 'ramnathv')
```

The design philosophy behind rCharts is to make the process of creating, customizing and sharing interactive visualizations easy.

Create

rCharts uses a formula interface to specify plots, just like the lattice package. Here are a few examples you can try out in your R console.

```
require(rCharts)

## Example 1 Facetted Scatterplot
names(iris) = gsub("\\.", "", names(iris))
rPlot(SepalLength ~ SepalWidth | Species, data = iris, color = 'Species', type = 'point')

## Example 2 Facetted Barplot
hair_eye = as.data.frame(HairEyeColor)
rPlot(Freq ~ Hair | Eye, color = 'Eye', data = hair_eye, type = 'bar')
```

Customize

rCharts supports multiple javascript charting libraries, each with its own strengths. Each of these libraries has multiple customization options, most of which are supported within rCharts.

Share

rCharts allows you to share your visualization in multiple ways, as a standalone page, embedded in a shiny application, or embedded in a tutorial/blog post.

Publish to Gist/RPubs

```
names(iris) = gsub("\\.", "", names(iris))
r1 <- rPlot(SepalLength ~ SepalWidth | Species, data = iris,
  color = 'Species', type = 'point')
r1$publish('Scatterplot', host = 'gist')
r1$publish('Scatterplot', host = 'rpubs')
```

Use with Shiny

rCharts is easy to embed into a Shiny application using the utility functions renderChart and showOutput. Here is an example of an rCharts Shiny App.

```r
## server.r
require(rCharts)
shinyServer(function(input, output) {
  output$myChart <- renderChart({
    names(iris) = gsub("\\.", "", names(iris))
    p1 <- rPlot(input$x, input$y, data = iris, color = "Species",
      facet = "Species", type = 'point')
    return(p1)
  })
})

## ui.R
require(rCharts)
shinyUI(pageWithSidebar(
  headerPanel("rCharts: Interactive Charts from R using polychart.js"),

  sidebarPanel(
    selectInput(inputId = "x",
     label = "Choose X",
     choices = c('SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth'),
     selected = "SepalLength"),
    selectInput(inputId = "y",
      label = "Choose Y",
      choices = c('SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth'),
      selected = "SepalWidth")
  ),
  mainPanel(
    showOutput("myChart", "polycharts")
  )
))
```

1.1 Credits

Most of the implementation in `rCharts` is inspired by rHighcharts and rVega. I have reused some code from these packages verbatim, and would like to acknowledge the efforts of its author Thomas Reinholdsson, who has since merged his rHighcharts package into rCharts. I would also like to thank @timelyportfolio for adding Dimple JS to rCharts, as well as for his contagious enthusiasm, which has egged me on constantly.

1.2 License

rCharts is licensed under the MIT License. However, the JavaScript charting libraries that are included with this package are licensed under their own terms. All of them are free for non-commercial and commercial use, with the exception of **Polychart** and **Highcharts**, both of which require paid licenses for commercial use. For more details on the licensing terms, you can consult the `License.md` file in each of the charting libraries.

1.3 See Also

There has been a lot of interest recently in creating packages that allow R users to make use of Javascript charting libraries.

- ggvis by RStudio

- clickme by Nacho Caballero

CHAPTER

TWO

GETTING STARTED

You can install rCharts from github using the devtools package

```
require(devtools)
install_github('rCharts', 'ramnathv')
```

The design philosophy behind rCharts is to make the process of creating, customizing and sharing interactive visualizations easy.

2.1 Create

rCharts uses a formula interface to specify plots, just like the lattice package. Here are a few examples you can try out in your R console.

Note: Every example comes with an edit button that allows you to experiment with the code online. The online playground was built using OpenCPU

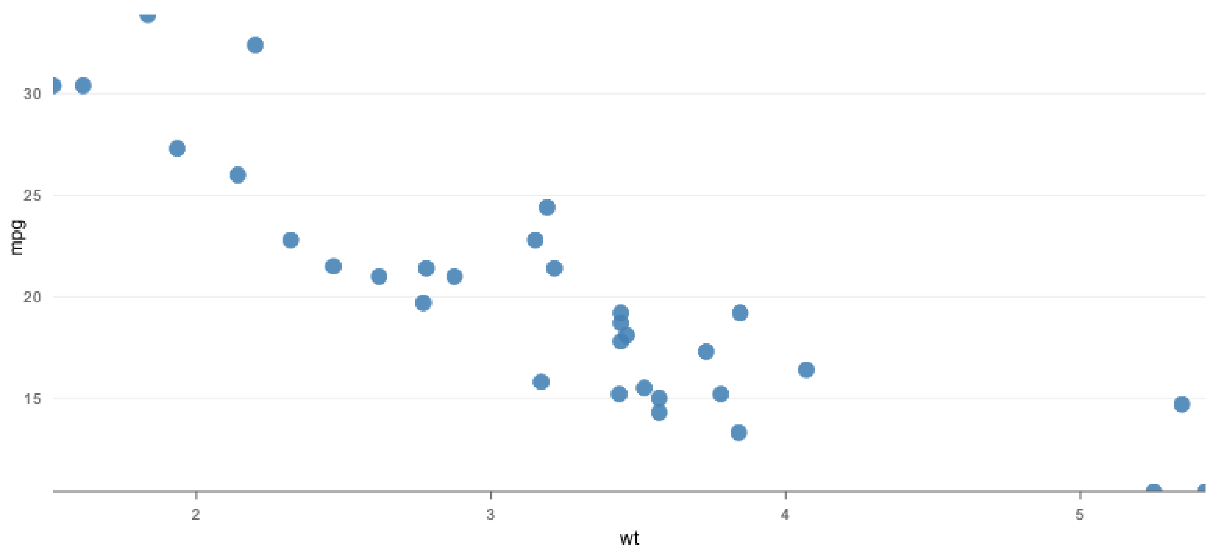
First let us load the `rCharts` package

```
library(rCharts)
```

2.1.1 Polycharts

```
r1 <- rPlot(mpg ~ wt, data = mtcars, type = 'point')
r1
```

```
Standalone
```



2.1.2 NVD3

```
hair_eye = as.data.frame(HairEyeColor)
p2 <- nPlot(Freq ~ Hair, group = 'Eye',
  data = subset(hair_eye, Sex == "Female"),
  type = 'multiBarChart'
)
p2$chart(color = c('brown', 'blue', '#594c26', 'green'))
p2
```

_downloads/intro-nvd3.pdf intro\protect \T1\textunderscore downloads/intro-nvd3.pdf

View Interactive

2.1.3 Morris

```
data(economics, package = "ggplot2")
econ <- transform(economics, date = as.character(date))
m1 <- mPlot(x = "date", y = c("psavert", "uempmed"), type = "Line", data = econ)
m1$set(pointSize = 0, lineWidth = 1)
m1
```

```
Standalone
```

2.1.4 Highcharts

```
h1 <- hPlot(x = "Wr.Hnd", y = "NW.Hnd",
  data = MASS::survey,
  type = c("line", "bubble", "scatter"),
  group = "Clap",
  size = "Age"
)
h1
```

Standalone

2.1.5 Rickshaw

```
usp = reshape2::melt(USPersonalExpenditure)
usp$Var2 <- as.numeric(as.POSIXct(paste0(usp$Var2, "-01-01")))
p4 <- Rickshaw$new()
p4$layer(value ~ Var2, group = "Var1", data = usp, type = "area")
p4$set(slider = TRUE)
p4
```

Standalone

2.1.6 xCharts

```
require(reshape2)
uspexp <- melt(USPersonalExpenditure)
names(uspexp)[1:2] = c('category', 'year')
x1 <- xPlot(value ~ year, group = 'category', data = uspexp,
  type = 'line-dotted')
x1
```

Standalone

2.2 Share

Any visualization is useful only when you are able to share it. rCharts tries to make it really easy to share the visualizations you create. Let us first create a simple interactive scatterplot to illustrate the different sharing mechanisms built into rCharts

- Save
- Publish
- Embed

```
library(rCharts)
r1 <- rPlot(mpg ~ wt, data = mtcars, type = 'point')
```

2.2.1 Save

You can save your chart using the `save` method. The additional parameters passed to the `save` method determine how the js/css assets of the javascript visualization library are served. You can now email your visualization or embed it in a blog post as an iframe.

```
# link js/css assets from an online cdn
r1$save('mychart1.html', cdn = TRUE)
# create standalone chart with all assets included directly in the html file
r1$save('mychart2.html', standalone = TRUE)
```

2.2.2 Publish

Sometimes, you may want to directly publish the visualization you created, without having to bother with the steps of saving it and then uploading it. rChart has you covered here, and provides a `publish` method that combines these two steps. It currently supports publishing to RPubs and Gist and I expect to add more providers over time.

```
# the host defaults to 'gist'
r1$publish("My Chart")
r1$publish("My Chart", host = 'rpubs')
```

Publishing a chart saves the html in a temporary file, uploads it to the specified `host`, and returns a link to where the chart can be viewed. There are many gist viewers out there, and rCharts uses a custom viewer http://rcharts.io/viewer, designed specifically for rCharts, and is a modified version of another excellent gist viewer http://www.pagist.info/. Another popular gist viewer is http://blocks.org, built by Mike Bostock, the creator of d3.js.

If you wish to simply **update** a visualization you have already created and shared, you can pass the gist/rpubs id to the `publish` method, and it will update instead of uploading it as a brand new chart.

```
r1$publish("My Chart", id = 9253202)
```

While using a provider like Gist that allows multiple files to be uploaded, you can use the `extras` argument to add additional files that you want to upload. This is especially useful, if you want to provide a `README.md` or upload external assets like js/css/json files that are required for your chart to render.

```
r1$publish("My Chart", id = 9253202, extras = "README.md")
```

2.2.3 Embed

RMarkdown

Suppose you wish to embed a visualization created using rCharts in an Rmd document.

IFrame

One way to do this would be to use the `save` method to save the chart, and then embed it as an iframe. rCharts saves you the steps by allowing you to use the `show` method and specify that you want the chart to be embedded as an `iframe`.

We need to set the chunk options `comment = NA` and `results = "asis"` so that the resulting html is rendered asis and not marked up (which is the default in knitr).

~~~
```{r results = "asis", comment = NA}
r1$show('iframe', cdn = TRUE)
```
~~~

If you have several charts in your Rmd document, you can set these options globally in a setup chunk. Make sure to set `cache = F` for this chunk so that it is always run.

~~~
```{r setup, cache = F}
options(rcharts.mode = 'iframe', rcharts.cdn = TRUE)
knitr::opts_chunk$set(results = "asis", comment = NA)
```
~~~

You can now rewrite the earlier sourcecode chunk simply as

~~~
```{r}
r1
```
~~~

I prefer this style when writing, since it allows a user to simply copy paste sourcecode from the html and run it in their R console.

IFrame Inline

The `iframe` mode requires users to upload the additional chart html files along with their document. This introduces additional steps, and in the case of some providers like Rpubs, is not even possible. Hence, rCharts provides an additional mode named `iframesrc` that embeds the chart as an inline iframe, which makes your document self contained.

~~~
```{r results = "asis", comment = NA}
r1$show('iframesrc', cdn  = TRUE)
```
~~~

This option has the advantage of keeping the html standalone, but isolating the chart from the html on the page, thereby avoiding css and js conflicts. However, this feature is not supported by IE and Opera.

Inline

A third option to embed an rCharts created visualization is to inline the chart directly. Note that you need to add `include_assets = TRUE`, only the first time you are creating a chart using a specific library.

~~~
```{r chart3}
r1$show('inline', include_assets = TRUE, cdn = TRUE)
```
~~~

This approach should work in all browsers, however, it is susceptible to css and js conflicts.

If you are using Slidify to author your Rmd, then you can specify the charting library as `ext_widgets` in the YAML front matter. Here is a minimal reproducible example.

Note how you did not have to specify `include_assets = TRUE`. This is because slidify uses the `ext_widgets` property to automatically pick up the required assets and include them in the header of the resulting html page.

Shiny

It is easy to embed visualizations created using rCharts into a Shiny application. The main idea is to make use of the utility functions `renderChart()` and `showOutput()`. The shiny application created using the code below, can be seen here

```
## server.r
require(rCharts)
shinyServer(function(input, output) {
  output$myChart <- renderChart({
    names(iris) = gsub("\\.", "", names(iris))
    p1 <- rPlot(input$x, input$y, data = iris, color = "Species",
      facet = "Species", type = 'point')
    p1$addParams(dom = 'myChart')
    return(p1)
  })
})

## ui.R
require(rCharts)
```

```
shinyUI(pageWithSidebar(
  headerPanel("rCharts: Interactive Charts from R using polychart.js"),

  sidebarPanel(
    selectInput(inputId = "x",
     label = "Choose X",
     choices = c('SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth'),
     selected = "SepalLength"),
    selectInput(inputId = "y",
      label = "Choose Y",
      choices = c('SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth'),
      selected = "SepalWidth")
  ),
  mainPanel(
    showOutput("myChart", "polycharts")
  )
))
```

CHAPTER

THREE

LIBRARIES

rCharts supports multiple javascript visualization libraries

3.1 NVD3

NVD3 is an elegant visualization library that provides re-usable chart components based on d3.js. Here is an excerpt directly taken from the NVD3 website.

> "This project is an attempt to build re-usable charts and chart components for d3.js without taking away the power that d3.js gives you. This is a very young collection of components, with the goal of keeping these components very customizeable, staying away from your standard cookie cutter solutions."

3.1.1 Create

The NVD3 library supports most of the common chart types.

- Scatter Chart
- Multibar Chart
- Multibar Horizontal Chart
- Pie Chart
- Donut Chart
- Line Chart
- Line with Focus Chart
- Stacked Area Chart
- Multi Chart

You can create an interactive plot making use of the NVD3 library using the `nPlot()` function.

Argument	Type	Description
x	formula	A formula of the form y ~ x, with column names from the data frame.
data	data frame	A data frame containing the data to be plotted
type	string	The type of chart to plot
group	string	Name of column based on which data should be grouped.

Scatter Chart

```
p1 <- nPlot(mpg ~ wt, group = 'cyl', data = mtcars, type = 'scatterChart')
p1$xAxis(axisLabel = 'Weight')
p1
```

Standalone

Multibar Chart

```
hair_eye = as.data.frame(HairEyeColor)
p2 <- nPlot(Freq ~ Hair, group = 'Eye',
  data = subset(hair_eye, Sex == "Female"),
  type = 'multiBarChart'
)
p2$chart(color = c('brown', 'blue', '#594c26', 'green'))
p2
```

Standalone

Multibar Horizontal Chart

Standalone

Pie Chart

```
p4 <- nPlot(~ cyl, data = mtcars, type = 'pieChart')
p4
```

Standalone

Donut Chart

```
p5 <- nPlot(~ cyl, data = mtcars, type = 'pieChart')
p5$chart(donut = TRUE)
p5
```

Standalone

Line Chart

```
data(economics, package = 'ggplot2')
p6 <- nPlot(uempmed ~ date, data = economics, type = 'lineChart')
p6
```

Standalone

Line with Focus Chart

```
ecm <- reshape2::melt(
  economics[,c('date', 'uempmed', 'psavert')],
  id = 'date'
)
p7 <- nPlot(value ~ date, group = 'variable',
  data = ecm,
  type = 'lineWithFocusChart'
)
p7
```

```
Standalone
```

Stacked Area Chart

```
dat <- data.frame(
  t = rep(0:23, each = 4),
  var = rep(LETTERS[1:4], 4),
  val = round(runif(4*24,0,50))
)
p8 <- nPlot(val ~ t, group =  'var', data = dat,
 type = 'stackedAreaChart', id = 'chart'
)
p8
```

```
Standalone
```

Multi Chart

```
p12 <- nPlot(value ~ date, group = 'variable', data = ecm, type = 'multiChart')
p12$set(multi = list(
  uempmed = list(type="area", yAxis=1),
  psavert = list(type="line", yAxis=2)
))
p12$setTemplate(script = system.file(
  "/libraries/nvd3/layouts/multiChart.html",
  package = "rCharts"
))
p12
```

```
Standalone
```

3.2 dimple

dimple is a flexible and powerful charting library built on d3.js. Strengths of dimple include multivariate plotting, thorough documentation, multiple chart types, numerous examples, and MIT license.

Here is the description directly from the dimple website.

> **Simply Powerful**
>
> The aim of dimple is to open up the power and flexibility of d3 to analysts. It aims to give a gentle learning curve and minimal code to achieve something productive. It also exposes the d3 objects so you can pick them up and run to create some really cool stuff.

3.2.1 Create

The dimple library supports most of the common chart types.

- Scatter/bubble Chart
- Bar Chart
- Bar Horizontal Chart
- Line Chart
- Area Chart
- Pie and Donut Charts (not yet supported)

You can create an interactive plot making use of the dimple library using the `dPlot()` function. We will go through some basic chart types and adjust the parameters only minimally. dimple is capable of much more with opportunity for customization through chart parameters and d3 manipulation. For a more thorough documentation of the API, see API docs. For more examples, see Dimple gallery.

Argument	Type	Description
x	formula or string	formula y ~ x or string x = “” or x=c(“”,””,..) with column names from the data frame
y	string or strings	y = “” or y = c(“”,””,...) with column names from the data frame
data	data frame	A data frame containing the data to be plotted
type	string	The type of chart to plot “line” , “bubble” , “bar” , “area”
z (optional)	string	z = “” with column names from the data frame
groups (optional)	string or strings	groups = “” or groups = c(“”,””,...)

Scatter/bubble Chart

```
#rownames in data.frames are not provided with toJSON
#add names as a column
mtcars.df <- data.frame( car = rownames(mtcars), mtcars )
p1 <- dPlot(mpg ~ wt, groups = c("car","cyl"), data = mtcars.df, type = 'bubble')
#by default dimple rCharts will assign x as a categorical/discrete axis
# and y as a measure/continuous axis
# easily changed as shown in the next step
p1$xAxis( type = "addMeasureAxis" )
p1
```

```
Standalone
```

Bar Chart

```
hair_eye = as.data.frame(HairEyeColor)
p2 <- dPlot(Freq ~ Hair, groups = 'Eye',
  data = subset(hair_eye, Sex == "Female"),
  type = 'bar'
)
p2$defaultColors(c('brown', 'blue', '#594c26', 'green'))
p2
```

```
Standalone
```

Bar Horizontal Chart

```
Standalone
```

Line Chart

```
data(economics, package = 'ggplot2')
#dimple supports a time axis
#for that we need dimple in a d3 date format
economics$date <- format(economics$date, "%Y-%m-%d")
p6 <- dPlot(uempmed ~ date, data = economics, type = 'line')
#here is how we tell dimple the input and output format of the date
p6$xAxis(
  type = "addTimeAxis",
  inputFormat = "%Y-%m-%d",
  outputFormat = "%b %Y"
)
p6

Standalone
```

Area Chart

```
dat <- data.frame(
  t = rep(0:23, each = 4),
  var = rep(LETTERS[1:4], 4),
  val = round(runif(4*24,0,50))
)
p8 <- dPlot(val ~ t, groups =  'var', data = dat,
 type = 'area'
)
p8

Standalone
```

Pie and Donut Charts (not yet supported)

Pie and donut charts are not currently provided in dimple (see issue) .

3.2.2 dimple/rCharts API

In most cases the rCharts/dimple plugin has attempted to map the dimple API on a one-to-one basis while keeping rCharts reference class methods consistent.

- dPlot
- xAxis, yAxis, zAxis
- legend
- storyboard
- lineWeight
- barGap
- aggregate

dPlot

You can create an interactive plot making use of the dimple library using the `dPlot()` function. The minimal parameters are listed in the table below. The primary difference with dimple and other rCharts libraries is that arrays can be provided for x, y, and groups for pivot like functionality.

Argument	Type	Description
x	formula or string	formula y ~ x or string x = "" or x=c("","",..) with column names from the data frame
y	string or strings	y = "" or y = c("","",...) with column names from the data frame
data	data frame	A data frame containing the data to be plotted
type	string	The type of chart to plot "line" , "bubble" , "bar" , "area"
z (optional)	string	z = "" with column names from the data frame
groups (optional)	string or strings	groups = "" or groups = c("","",...)

Beyond this basic specification, there are numerous opportunities for customization as shown with the methods and parameters below.

xAxis, yAxis, zAxis

dimple API: dimple.axis

xAxis, yAxis, and zAxis will gets passed in the JSON spec as opts.xAxis, ... By default rCharts will assign the x axis as addCategoryAxis (categorical/discrete) and y axis as addMeasureAxis (continuous/numeric).

Arguments

Argument	Type	Description
type	string	"addMeasureAxis", "addCategoryAxis", "addTimeAxis", "addPctAxis".
orderRule	string or array	add a rule by which the values in a category axis will be ordered
grouporderRule	string or array	add a rule by which the values in a group on a category axis will be ordered
overrideMin	generally numeric	manually override minimum for the axis
overrideMax	generally numeric	manually override maximum for the axis
inputFormat **only for "addTimeAxis"**	string	d3 time parse format using d3 time formatting
outputFormat	string	formatting for the axis using the d3 time formatting or d3 formatting style

Details

Provide sample usages and an example similar to the gallery/create sections.

legend

dimple API: dimple.chart.legends

Arguments

Details

storyboard

dimple API: dimple.chart.storyboard

Arguments

Details

lineWeight

dimple API: dimple.series.lineWeight

Arguments

Details

barGap

dimple API: dimple.series.barGap

Arguments

Details

aggregate

dimple API: dimple.series.aggregate

Arguments

Details

3.2.3 Gallery

The dimple library offers a wealth of common and uncommon chart types.

- example01_bar_vert.R
- example02_bar_vert_stack.R
- example03_bar_vert_100.R
- example04_bar_vert_grp.R
- example05_bar_vert_stack_grp.R
- example06_bar_vert_stack_100.R
- example07_bar_horiz.R
- example08_bar_horiz_stack.R
- example09_bar_horiz_stack_100.R
- example10_bar_horiz_grp.R
- example11_bar_horiz_stack_grp.R
- example12_bar_horiz_stack_100.R
- example13_marimekko_vert.R
- example14_marimekko_horiz.R
- example15_block_matrix.R
- example16_scatter.R
- example18_lollipop_vert_grp.R
- example19_lollipop_horiz.R
- example20_lollipop_horiz_grp.R
- example21_dot_matrix.R
- example22_bubble.R
- example23_bubble_lollipop_vert.R
- example24_bubble_lollipop_vert_grp.R
- example25_bubble_lollipop_horiz.R
- example26_bubble_lollipop_horiz_grp.R
- example27_bubble_matrix.R
- example28_area.R
- example29_area_stack.R
- example30_area_stack_100.R
- example31_area_grp.R
- example32_area_grp_stack.R
- example33_area_grp_stack_100.R
- example34_area_vert.R
- example35_area_vert_stack.R
- example36_area_vert_stack_100.R
- example37_area_vert_grp.R
- example38_area_vert_grp_stack.R
- example39_area_vert_grp_stack_100.R
- example40_line.R
- example41_line_mult.R
- example42_line_single_grp.R
- example43_line_mult_grp.R
- example44_line_vert.R
- example45_line_vert_mult.R
- example46_line_vert_grp.R
- example47_line_vert_mult_grp.R
- example51_rCharts_issue164.R

We will use the same data source that dimple uses for its example gallery.

```
#get data used by dimple for all of its examples as a first test
data <- read.delim(
  "http://pmsi-alignalytics.github.io/dimple/data/example_data.tsv"
)

### eliminate . to avoid confusion in javascript
colnames(data) <- gsub("[.]","",colnames(data))
```

example01_bar_vert.R

```
### example 1 vt bar
d1 <- dPlot(
  x ="Month" ,
  y = "UnitSales",
  data = data,
  type = "bar"
)
d1$xAxis(orderRule = "Date")
d1
```

```
Standalone
```

example02_bar_vert_stack.R

```
### example 2 vt stacked bar
d1 <- dPlot(
  x ="Month" ,
  y = "UnitSales",
  groups = "Channel",
  data = data,
  type = "bar"
)
d1$xAxis(orderRule = "Date")
d1$legend(
  x = 60,
  y = 10,
  width = 700,
  height = 20,
  horizontalAlign = "right"
)
d1
```

```
Standalone
```

example03_bar_vert_100.R

```
### example 3 vt stacked bar 100%
d1 <- dPlot(
  x ="Month" ,
  y = "UnitSales",
  groups = "Channel",
  data = data,
  type = "bar"
)
d1$xAxis(orderRule = "Date")
d1$yAxis(type = "addPctAxis")
d1$legend(
```

```r
  x = 60,
  y = 10,
  width = 700,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example04_bar_vert_grp.R

```r
### example 4 vertical grouped bar
d1 <- dPlot(
  x = c("PriceTier","Channel"),
  y = "UnitSales",
  groups = "Channel",
  data = data,
  type = "bar"
)
d1$legend(
  x = 60,
  y = 10,
  width = 700,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example05_bar_vert_stack_grp.R

```r
### example 5 vertical stack grouped bar
d1 <- dPlot(
  x = c("PriceTier","Channel"),
  y = "UnitSales",
  groups = "Owner",
  data = data,
  type = "bar"
)
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example06_bar_vert_stack_100.R

```
### example 6 vertical stack grouped 100% bar
d1 <- dPlot(
  x = c("PriceTier","Channel"),
  y = "UnitSales",
  groups = "Owner",
  data = data,
  type = "bar"
)
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1$yAxis(type = "addPctAxis")
d1
```

```
Standalone
```

example07_bar_horiz.R

```
### example 7 horizontal bar
d1 <- dPlot(
  Month ~ UnitSales,
  data = data,
  type = "bar"
)
d1$xAxis(type = "addMeasureAxis")
#good test of orderRule on y instead of x
d1$yAxis(type = "addCategoryAxis", orderRule = "Date")
d1
```

```
Standalone
```

example08_bar_horiz_stack.R

```
### example 8 horizontal stacked bar
d1 <- dPlot(
  Month ~ UnitSales,
  groups = "Channel",
  data = data,
  type = "bar"
)
d1$xAxis(type = "addMeasureAxis")
#good test of orderRule on y instead of x
d1$yAxis(type = "addCategoryAxis", orderRule = "Date")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example09_bar_horiz_stack_100.R

```r
### example 9 horizontal stacked 100% bar
d1 <- dPlot(
  Month ~ UnitSales,
  groups = "Channel",
  data = data,
  type = "bar"
)
d1$xAxis(type = "addPctAxis")
#good test of orderRule on y instead of x
d1$yAxis(type = "addCategoryAxis", orderRule = "Date")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example10_bar_horiz_grp.R

```r
### example 10 horizontal stacked bar
d1 <- dPlot(
  x = "UnitSales",
  y = c("PriceTier","Channel"),
  groups = "Channel",
  data = data,
  type = "bar"
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example11_bar_horiz_stack_grp.R

```
### example 11 horizontal stacked grouped bar
d1 <- dPlot(
  x = "UnitSales",
  y = c("PriceTier","Channel"),
  groups = "Owner",
  data = data,
  type = "bar"
)
d1$xAxis(type = "addMeasureAxis")
#good test of orderRule on y instead of x
d1$yAxis(type = "addCategoryAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1

Standalone
```

example12_bar_horiz_stack_100.R

```
### example 12 horizontal stacked grouped 100% bar
d1 <- dPlot(
  x = "UnitSales",
  y = c("PriceTier","Channel"),
  groups = "Owner",
  data = data,
  type = "bar"
)
d1$xAxis(type = "addPctAxis")
#good test of orderRule on y instead of x
d1$yAxis(type = "addCategoryAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1

Standalone
```

example13_marimekko_vert.R

```
#require(devtools)
#install_github('rCharts', 'ramnathv')
require(rCharts)

#get data used by dimple for all of its examples as a first test
data <- read.delim(
  "http://pmsi-alignalytics.github.io/dimple/data/example_data.tsv"
)
```

```
### eliminate . to avoid confusion in javascript
colnames(data) <- gsub("[.]","",colnames(data))

### example 13 vertical marimekko
d1 <- dPlot(
  UnitSales ~ Channel,
  groups = "Owner",
  data = data,
  type = "bar"
)
d1$xAxis(type = "addAxis", measure = "UnitSales", showPercent = TRUE)
d1$yAxis(type = "addPctAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
#test with storyboard
d1$set(storyboard = "Date")
d1
```

`Standalone`

example14_marimekko_horiz.R

```
### example 14 horizontal marimekko
d1 <- dPlot(
  Channel ~ UnitSales,
  groups = "Owner",
  data = data,
  type = "bar"
)
d1$yAxis(type = "addAxis", measure = "UnitSales", showPercent = TRUE)
d1$xAxis(type = "addPctAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1
```

`Standalone`

example15_block_matrix.R

```
### example 15 block matrix
d1 <- dPlot(
  x = c("Channel","PriceTier"),
  y = "Owner",
  groups = "PriceTier",
  data = data,
  type = "bar"
)
d1$yAxis(type = "addCategoryAxis")
d1$xAxis(type = "addCategoryAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 400,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example16_scatter.R

```
### example 16 Scatter
d1 <- dPlot(
  OperatingProfit~UnitSales,
  groups = c("SKU","Channel"),
  data = subset(data, Date == "01/12/2012"),
  type = "bubble"
)
d1$xAxis( type = "addMeasureAxis" )
d1$yAxis( type = "addMeasureAxis" )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example18_lollipop_vert_grp.R

```
### example 18 Vertical Grouped Lollipop
d1 <- dPlot(
  y = "UnitSales",
  x = c("PriceTier","Channel"),
  groups = "Channel",
  data = data,
  type = "bubble"
)
#defaults to yAxis (Measure) and xAxis (Category)
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example19_lollipop_horiz.R

```r
### example 19 Horizontal Lollipop
d1 <- dPlot(
  x = "UnitSales",
  y = "Month",
  groups = "Channel",
  data = data,
  type = "bubble"
)
d1$xAxis( type = "addMeasureAxis" )
d1$yAxis( type = "addCategoryAxis", orderRule = "Date")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

```
Standalone
```

example20_lollipop_horiz_grp.R

```r
### example 20 Horizontal Grouped Lollipop
d1 <- dPlot(
  x = "UnitSales",
  y = c("PriceTier","Channel"),
  groups = "Channel",
  data = data,
  type = "bubble"
)
d1$xAxis( type = "addMeasureAxis" )
d1$yAxis( type = "addCategoryAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

```
Standalone
```

example21_dot_matrix.R

```r
### example 21 Dot Matrix
d1 <- dPlot(
  y = "Owner",
  x = c("Channel","PriceTier"),
  groups = "PriceTier",
  data = data,
  type = "bubble"
)
```

```
d1$xAxis( type = "addCategoryAxis" )
d1$yAxis( type = "addCategoryAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example22_bubble.R

```
### example 22 Bubble
d1 <- dPlot(
  x = "UnitSalesMonthlyChange",
  y = "PriceMonthlyChange",
  z = "OperatingProfit",
  groups = c("SKU","Channel"),
  data = subset(data, Date == "01/12/2012"),
  type = "bubble"
)
d1$xAxis( type = "addMeasureAxis" )
d1$yAxis( type = "addMeasureAxis" )
d1$zAxis( type = "addMeasureAxis" )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example23_bubble_lollipop_vert.R

```
### example 23 Vertical Bubble Lollipop
d1 <- dPlot(
  x = "Month",
  y = "UnitSales",
  z = "OperatingProfit",
  groups = "Channel",
  data = subset(
    data,
    Date %in% c(
      "01/07/2012",
      "01/08/2012",
      "01/09/2012",
      "01/10/2012",
      "01/11/2012",
      "01/12/2012"
    )
  ),
  type = "bubble"
)
d1$xAxis( type = "addCategoryAxis", orderRule = "Date" )
d1$yAxis( type = "addMeasureAxis" )
d1$zAxis( type = "addMeasureAxis" )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example24_bubble_lollipop_vert_grp.R

```
### example 24 Vertical Grouped Bubble Lollipop
d1 <- dPlot(
  x = c("PriceTier","Channel"),
  y = "UnitSales",
  z = "OperatingProfit",
  groups = "Channel",
  data = subset(
    data,
    Date %in% c(
      "01/07/2012",
      "01/08/2012",
      "01/09/2012",
      "01/10/2012",
      "01/11/2012",
      "01/12/2012"
    )
  ),
  type = "bubble"
)
d1$xAxis( type = "addCategoryAxis" )
d1$yAxis( type = "addMeasureAxis" )
d1$zAxis( type = "addMeasureAxis" )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example25_bubble_lollipop_horiz.R

```
### example 25 Horizontal Bubble Lollipop
d1 <- dPlot(
  y = "Month",
  x = "UnitSales",
  z = "OperatingProfit",
  groups = "Channel",
  data = subset(
    data,
    Date %in% c(
      "01/07/2012",
      "01/08/2012",
      "01/09/2012",
      "01/10/2012",
      "01/11/2012",
      "01/12/2012"
    )
  ),
  type = "bubble"
)
d1$yAxis( type = "addCategoryAxis", orderRule = "Date" )
d1$xAxis( type = "addMeasureAxis" )
d1$zAxis( type = "addMeasureAxis" )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

```
Standalone
```

example26_bubble_lollipop_horiz_grp.R

```
### example 26 Horizontal Grouped Bubble Lollipop
d1 <- dPlot(
  y = c("PriceTier","Channel"),
  x = "UnitSales",
  z = "OperatingProfit",
  groups = "Channel",
  data = subset(
    data,
    Date %in% c(
      "01/07/2012",
      "01/08/2012",
      "01/09/2012",
      "01/10/2012",
      "01/11/2012",
      "01/12/2012"
    )
  ),
  type = "bubble"
)
d1$yAxis( type = "addCategoryAxis" )
d1$xAxis( type = "addMeasureAxis" )
d1$zAxis( type = "addMeasureAxis" )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example27_bubble_matrix.R

```
### example 27 Bubble Matrix
d1 <- dPlot(
  x = c( "Channel", "PriceTier"),
  y = "Owner",
  z = "Distribution",
  groups = "PriceTier",
  data = data,
  type = "bubble",
  aggregate = "dimple.aggregateMethod.max"
)
d1$xAxis( type = "addCategoryAxis" )
d1$yAxis( type = "addCategoryAxis" )
d1$zAxis( type = "addMeasureAxis", overrideMax = 200 )
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example28_area.R

```
### example 28 Area
d1 <- dPlot(
  UnitSales ~ Month,
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area"
)
d1$xAxis(type = "addCategoryAxis", orderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1
```

Standalone

example29_area_stack.R

```
### example 29 Stacked Area
d1 <- dPlot(
  UnitSales ~ Month,
  groups = "Channel",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area"
)
d1$xAxis(type = "addCategoryAxis", orderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example30_area_stack_100.R

```
### example 30 Stacked Area 100%
d1 <- dPlot(
  UnitSales ~ Month,
  groups = "Channel",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area"
)
d1$xAxis(type = "addCategoryAxis", orderRule = "Date")
d1$yAxis(type = "addPctAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example31_area_grp.R

```
### example 31 Grouped Area
d1 <- dPlot(
  y = "UnitSales",
  x = c("Owner","Month"),
  groups = "Owner",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area"
)
d1$xAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example32_area_grp_stack.R

```
### example 32 Grouped Stacked Area
d1 <- dPlot(
  y = "UnitSales",
  x = c("Owner","Month"),
  groups = "SKU",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=70,y=30,height=340,width=330),
  barGap = 0.05,
  lineWeight = 1,
  height = 400,
  width = 590
)
d1$xAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1$legend(
  x = 430,
  y = 20,
  width = 100,
  height = 300,
  horizontalAlign = "left"
)
d1
```

Standalone

example33_area_grp_stack_100.R

```
### example 33 Grouped Stacked Area 100%
d1 <- dPlot(
  y = "UnitSales",
  x = c("Owner","Month"),
  groups = "SKU",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=70,y=30,height=340,width=330),
  barGap = 0.05,
  lineWeight = 1,
  height = 400,
  width = 590
)
d1$xAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$yAxis(type = "addPctAxis")
d1$legend(
  x = 430,
  y = 20,
  width = 100,
  height = 300,
  horizontalAlign = "left"
)
d1
```

Standalone

example34_area_vert.R

```
### example 34 Vertical Area
d1 <- dPlot(
  x = "UnitSales",
  y = "Month",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=80,y=30,height=480,width=330),
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", orderRule = "Date")
d1
```

Standalone

example35_area_vert_stack.R

```
### example 35 Vertical Stacked Area
d1 <- dPlot(
  x = "UnitSales",
  y = "Month",
  groups = "Channel",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=80,y=30,height=480,width=330),
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$legend(
  x = 60,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example36_area_vert_stack_100.R

```
### example 36 Vertical 100% Stacked Area
d1 <- dPlot(
  x = "UnitSales",
  y = "Month",
  groups = "Channel",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=80,y=30,height=480,width=330),
  height = 400,
  width = 590
)
d1$xAxis(type = "addPctAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$legend(
  x = 60,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

```
Standalone
```

example37_area_vert_grp.R

```
### example 37 Vertical Grouped Area
d1 <- dPlot(
  x = "UnitSales",
  y = c("Owner","Month"),
  groups = "Owner",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=90,y=30,height=470,width=330),
  lineWeight = 1,
  barGap = 0.05,
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1
```

```
Standalone
```

example38_area_vert_grp_stack.R

```
### example 38 Vertical Grouped Stacked Area
d1 <- dPlot(
  x = "UnitSales",
  y = c("Owner","Month"),
  groups = "SKU",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=90,y=30,height=320,width=330),
  lineWeight = 1,
  barGap = 0.05,
  height = 400,
  width = 590
)
```

```
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$legend(
  x = 430,
  y = 20,
  width = 100,
  height = 300,
  horizontalAlign = "left"
)
d1
```

Standalone

example39_area_vert_grp_stack_100.R

```
### example 39 Vertical Grouped Stacked Area 100%
d1 <- dPlot(
  x = "UnitSales",
  y = c("Owner","Month"),
  groups = "SKU",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "area",
  bounds = list(x=90,y=30,height=320,width=330),
  lineWeight = 1,
  barGap = 0.05,
  height = 400,
  width = 590
)
d1$xAxis(type = "addPctAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$legend(
  x = 430,
  y = 20,
  width = 100,
  height = 300,
  horizontalAlign = "left"
)
d1
```

Standalone

example40_line.R

```
### example 40 Line
d1 <- dPlot(
  UnitSales ~ Month,
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line"
)
d1$xAxis(type = "addCategoryAxis", orderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1
```

Standalone

example41_line_mult.R

```
### example 41 Multiple Line
d1 <- dPlot(
  UnitSales ~ Month,
  groups = "Channel",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line"
)
d1$xAxis(type = "addCategoryAxis", orderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1$legend(
  x = 200,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

```
Standalone
```

example42_line_single_grp.R

```
### example 42 Grouped Single Line
d1 <- dPlot(
  y = "UnitSales",
  x = c("Owner","Month"),
  groups = "Owner",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line",
  barGap = 0.05
)
d1$xAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1
```

```
Standalone
```

example43_line_mult_grp.R

```
### example 43 Grouped Multiple line
d1 <- dPlot(
  y = "UnitSales",
  x = c("Owner","Month"),
  groups = "Brand",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line",
  bounds = list(x=70,y=30,height=420,width=330),
  barGap = 0.05,
  height = 400,
  width = 590
)
d1$xAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$yAxis(type = "addMeasureAxis")
d1$legend(
  x = 510,
  y = 20,
  width = 100,
  height = 300,
  horizontalAlign = "left"
)
d1
```

Standalone

example44_line_vert.R

```
### example 44 Vertical Line
d1 <- dPlot(
  x = "UnitSales",
  y = "Month",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line",
  bounds = list(x=80,y=30,height=480,width=330),
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", orderRule = "Date")
d1
```

Standalone

example45_line_vert_mult.R

```
### example 45 Vertical Multiple Line
d1 <- dPlot(
  x = "UnitSales",
  y = "Month",
  groups = "Channel",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line",
  bounds = list(x=80,y=30,height=480,width=330),
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", orderRule = "Date")
d1$legend(
  x = 60,
  y = 10,
  width = 500,
  height = 20,
  horizontalAlign = "right"
)
d1
```

Standalone

example46_line_vert_grp.R

```
###e xample 46 Vertical Grouped Line
d1 <- dPlot(
  x = "UnitSales",
  y = c("Owner","Month"),
  groups = "Owner",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line",
  bounds = list(x=90,y=30,height=470,width=330),
  barGap = 0.05,
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1
```

```
Standalone
```

example47_line_vert_mult_grp.R

```
### example 47 Vertical Grouped Multi Line
d1 <- dPlot(
  x = "UnitSales",
  y = c("Owner","Month"),
  groups = "Brand",
  data = subset(data, Owner %in% c("Aperture","Black Mesa")),
  type = "line",
  bounds = list(x=90,y=30,height=320,width=330),
  barGap = 0.05,
  height = 400,
  width = 590
)
d1$xAxis(type = "addMeasureAxis")
d1$yAxis(type = "addCategoryAxis", grouporderRule = "Date")
d1$legend(
  x = 430,
  y = 20,
  width = 100,
  height = 300,
  horizontalAlign = "left"
)
d1
```

```
Standalone
```

example51_rCharts_issue164.R

```
require(rjson)
```

```
Loading required package: rjson
```

```
require(rCharts)
```

```
data <- data.frame(rjson::fromJSON('{"Zscore": [  1.594,  0.024,  5.231,  3.933,  6.222,  0.164,  3.
"Cell": [ "HTR8svn", "Adult_CD4+", "CD14+", "CD20+", "CD34+", "CLL", "CMK", "GM06990", "GM12864", "GM
"Tissue": [ "Blastula", "Blood", "Blood", "Blood", "Blood", "Blood", "Blood", "Blood", "Blood", "Bloo
                         "Colour": [      2,      2,      0,      0,      0,      2,      1,      1,
                         }'))

d1 <- dPlot(
  y = "Zscore",
  x = "Cell",
  groups = "Colour",
  data = data,
  type = "bubble",
  width = 1000,
  height = 600,
  bounds = list(x=90,y=30,height=500,width=850)
)
d1$xAxis( type = "addCategoryAxis", orderRule = "Cell" )
d1$yAxis( type = "addMeasureAxis" )
d1$colorAxis(
  type = "addColorAxis",
  colorSeries = "Zscore",
  palette = c("red","yellow","green") )
d1

Standalone
```

CHAPTER

FOUR

TUTORIALS

4.1 Visualizing Strikeouts

This tutorial explains in detail, how I used `rCharts` to replicate this NY times interactive graphic on strikeouts in baseball. The end result can be seen here as a `shiny` application.

4.1.1 Data

The first step is to get data on strikeouts by team across years. The NY Times graphic uses data scraped from baseball-reference, using the `XML` package in R. However, I will be using data from the R package Lahman, which provides tables from Sean Lahman's Baseball Database as a set of data frames.

The data processing step involves using the plyr package to create two data frames:

1. `team_data` containing `SOG` (strikeouts per game) by `yearID` and team `name`
2. `league_data` containing `SOG` by *yearID* averaged across the league.

```
require(Lahman) ; require(plyr); library(ascii)
dat = Teams[,c('yearID', 'name', 'G', 'SO')]
team_data = na.omit(transform(dat, SOG = round(SO/G, 2)))
league_data = ddply(team_data, .(yearID), summarize, SOG = mean(SOG))
ascii(head(team_data), type = 'rst')
```

	yearID	name	G	SO	SOG
1	1871.00	Boston Red Stockings	31.00	19.00	0.61
2	1871.00	Chicago White Stockings	28.00	22.00	0.79
3	1871.00	Cleveland Forest Citys	29.00	25.00	0.86
4	1871.00	Fort Wayne Kekiongas	19.00	9.00	0.47
5	1871.00	New York Mutuals	33.00	15.00	0.45
6	1871.00	Philadelphia Athletics	28.00	23.00	0.82

4.1.2 Charts

We will start by first creating a scatterplot of *SOG* by *yearID* across all teams. We use the *rPlot* function which uses the PolyChartsJS library to create interactive visualizations. The formula interface specifies the x and y variables, the data to use and the type of plot. We also specify a *size* and *color* argument to style the points. Finally, we pass a *tooltip* argument, which is a javascript function that overrides the default tooltip to display the information we require. You will see below the R code and the resulting chart.

```
require(rCharts)
p1 <- rPlot(SOG ~ yearID, data = team_data,
  type = "point",
  size = list(const = 2),
  color = list(const = "#888"),
  tooltip = "#! function(item){
    return item.SOG + ' ' + item.name + ' ' + item.yearID
   } !#"
)
p1
```

```
Standalone
```

_downloads/chart1.pdf tutorials\protect \T1\textunderscore downloads/chart1.pdf

Now, we need to add a line plot of the average `SOG` for the league by `yearID`. We do this by adding a second layer to the chart, which copies the elements of the previous layer and overrides the `data`, *type*, `color` and `tooltip` arguments. The R code is shown below and you will note that the resulting chart now shows a blue line chart corresponding to the league average `SOG`.

```
p1$layer(data = league_data, type = 'line',
  color = list(const = 'blue'), copy_layer = T, tooltip = NULL)
p1
```

```
Standalone
```

_downloads/chart2.pdf tutorials\protect \T1\textunderscore downloads/chart2.pdf

Finally, we will overlay a line plot of `SOG` by `yearID` for a specific team *name*. Later, while building the shiny app, we will turn this into an input variable that a user can choose from a dropdown menu. We use the layer approach used earlier and this time override the *data* and *color* arguments so that the line plot for the team stands out from the league average.

```
myteam = "Boston Red Sox"
p1$layer(data = team_data[team_data$name == myteam,],
  color = list(const = 'red'),
  copy_layer = T)
p1$set(dom = 'chart3')
p1
```

```
Standalone
```

_downloads/chart3.pdf tutorials\protect \T1\textunderscore downloads/chart3.pdf

Let us add a little more interactivity to the chart. To keep it simple, we will use handlers in PolychartJS to initiate an action when a user clicks on a point. The current handler is a simple one, which just displays the name of the team clicked on. If you are familiar with Javascript event handlers, the code should be self explanatory.

```
p2 <- p1$copy()
p2$setTemplate(afterScript = '
  <script>
    graph_chart3.addHandler(function(type, e) {
      var data;
      data = e.evtData;
      if (type === "click") {
        return alert("You clicked on the team: " + data.name["in"][0]);
      }
    });
  </script>
')
p2

Standalone
```

4.1.3 Application

Now it is time to convert this into a Shiny App. We will throw the data processing code into *global.R* so that it can be accessed both by *ui.R* and *server.R*. For the dropdown menu allowing users to choose a specific team, we will restrict the choices to only those which have data for more than 30 years. Accordingly, we have the following *global.R*.

```
## global.R
require(Lahman); require(plyr)
dat = Teams[,c('yearID', 'name', 'G', 'SO')]
team_data = na.omit(transform(dat, SOG = round(SO/G, 2)))
league_data = ddply(team_data, .(yearID), summarize, SOG = mean(SOG))
THRESHOLD = 30
team_appearances = count(team_data, .(name))
teams_in_menu = subset(team_appearances, freq > THRESHOLD)$name
```

For the UI, we will use a bootstrap page with controls being displayed in the sidebar. Shiny makes it really easy to create a page like this. See the annotated graphic below and the *ui.R* code that accompanies it to understand how the different pieces fit together.

We now need to write the server part of the shiny app. Thankfully, this is the easiest part, since it just involves wrapping the charting code inside *renderChart* and replacing user inputs to enable reactivity. We add a few more lines of code to set the height and title and remove the axis titles, since they are self explanatory.