# Post02: Introduction to package 'dygraphs'

*Pinshuo Ye*

*2017/12/3*

# Post 02: Introduction to package ***dygraphs***

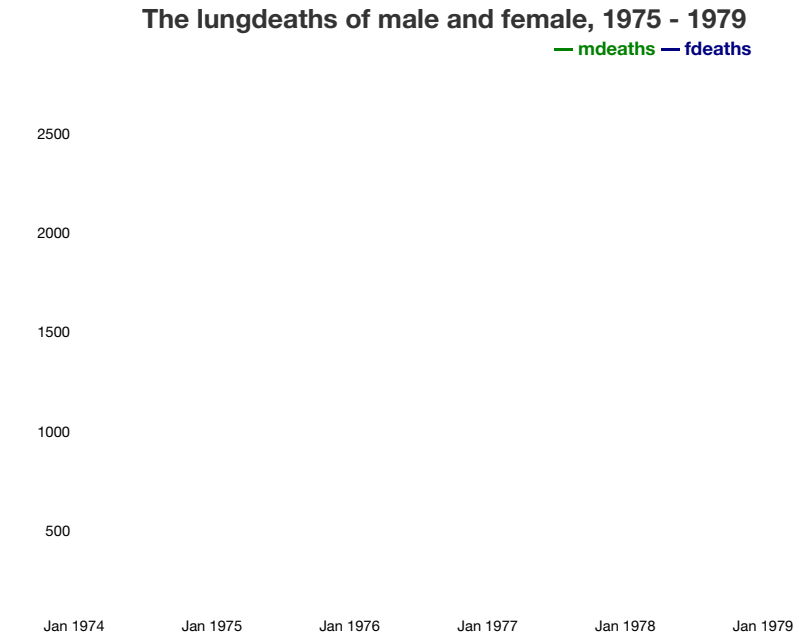
-The basic use of the functions in **dygraph**

## Introduction & Motivation

The package **dygraphs** is an elegant and useful tool to draw graphs and presenting data. The data visualization of this package is similar to the package **ggplot2**, but dygraphs can do more things on time series and shows more details than ggplot2 does. The package dygraphs can "create HTML/JavaScript graphs of time series - one-line command if your data is an xts object". The graph drawn by the functions in dygraph can show the detail of every point on the graph by putting the cursor on specific point.

Here is an brief example of the function **dygraph**:

```
library(dygraphs)
lungDeaths <- cbind(mdeaths, fdeaths)
dygraph(lungDeaths, main = "The lungdeaths of male and female, 1975 - 1979")
```




This graph takes the value of a data frame, which is the lung death of male and female from 1975 to 1979. The graph is clear to show the tendency of the death through the time changes, and it is also easy to trace every point on the graph, just by putting the cursor on it. So this package is really a powerful and useful tool of data visualization.

So using the package, we can do more things about analyzing data and presenting the data in a variety of ways.

## Data Preparation

To deal with the data by using dygraphs, we have to find the data first, and preparing the data:

```
# reading the table of the stock price of alphabet company
google <- read.csv("../data/GOOG.csv")
```

This is the stock price of goole from 2015 to 2017(Alphabet company), the data is daily stock price, with the price of opening, closing, high, low, and adjusted price. Dygraph package is a great way to deal with the time series in this data frame.

```
# loading the file of the personal stats of Lebron James
lbj03 <- read.csv("../data/lbj03.csv")
lbj04 <- read.csv("../data/lbj04.csv")
lbj05 <- read.csv("../data/lbj05.csv")
lbj06 <- read.csv("../data/lbj06.csv")
lbj07 <- read.csv("../data/lbj07.csv")
lbj08 <- read.csv("../data/lbj08.csv")
lbj09 <- read.csv("../data/lbj09.csv")
lbj10 <- read.csv("../data/lbj10.csv")
lbj11 <- read.csv("../data/lbj11.csv")
lbj12 <- read.csv("../data/lbj12.csv")
lbj13 <- read.csv("../data/lbj13.csv")
lbj14 <- read.csv("../data/lbj14.csv")
lbj15 <- read.csv("../data/lbj15.csv")
lbj16 <- read.csv("../data/lbj16.csv")
lbj17 <- read.csv("../data/lbj17.csv")
lbj_career <- read.csv("../data/lbj_career.csv")
```

There are too many data frames, so we have to put them together first, and then we can extract any data we want, to do the graphing:

```
library(stringr)
# Put the data frames together
lbj03 <- lbj03[ , 2:30]
lbj04 <- lbj04[ , 2:30]
lbj05 <- lbj05[ , 2:30]
lbj06 <- lbj06[ , 2:30]
lbj07 <- lbj07[ , 2:30]
lbj08 <- lbj08[ , 2:30]
lbj09 <- lbj09[ , 2:30]
lbj10 <- lbj10[ , 2:30]
lbj11 <- lbj11[ , 2:30]
lbj12 <- lbj12[ , 2:30]
lbj13 <- lbj13[ , 2:30]
lbj14 <- lbj14[ , 2:30]
lbj15 <- lbj15[ , 2:30]
lbj16 <- lbj16[ , 2:30]
lbj17 <- lbj17[ , 2:30]

lbj <- rbind(lbj03, lbj04, lbj05,
             lbj06, lbj07, lbj08,
             lbj09, lbj10, lbj11,
             lbj12, lbj13, lbj14,
             lbj15, lbj16, lbj17)
# Standardize data
lbj$Date <- str_replace_all(lbj$Date, pattern = '/', replacement = '-')
for(i in 1:1126){
  if(nchar(lbj$Date[i]) == 8){
    lbj$Date[i] <-
      paste0(substr(lbj$Date[i], 1, 5), "0", substr(lbj$Date[i], 6, 7), "0", substr(lbj$Date[i], 8, 8))
  }else if(nchar(lbj$Date[i]) == 9){
    if(substr(lbj$Date[i], 8, 8) == "-"){
      lbj$Date[i] <- paste0(substr(lbj$Date[i], 1, 8), "0", substr(lbj$Date[i], 9, 9))
    }else{
      lbj$Date[i] <- paste0(substr(lbj$Date[i], 1, 5), "0", substr(lbj$Date[i], 6, 9))
    }
  }
}
```

After loading the data, we need to make the data clean:

```
#Cleaning data
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
lbj15 <- slice(lbj15, 1:82)
```

```
## Warning in slice_impl(.data, dots): '.Random.seed' is not an integer vector
## but of type 'NULL', so ignored
```

```
lbj16 <- slice(lbj16, 1:82)
```

We can also take a peak at the first few line of some data frame:

```
# head of the data frame
head(google)
```

```
##         Date   Open    High     Low  Close Adj.Close  Volume
## 1 2015-12-02 768.90 775.955 758.960 762.38    762.38 2230400
## 2 2015-12-03 766.01 768.995 745.630 752.54    752.54 2590600
## 3 2015-12-04 753.10 768.490 750.000 766.81    766.81 2757300
## 4 2015-12-07 767.77 768.730 755.090 763.25    763.25 1812300
## 5 2015-12-08 757.89 764.800 754.200 762.37    762.37 1829500
## 6 2015-12-09 759.17 764.230 737.001 751.61    751.61 2700000
```

```
head(lbj06)
```

```
##   G       Date    Age  Tm X Opp     X.1 GS       MP FG FGA   FG. X3P X3PA
## 1 1  2006/11/1 21-306 CLE   WAS  W (+3)  1 40:38:00 11  24 0.458   2    4
## 2 2  2006/11/3 21-308 CLE @ SAS  W (+7)  1 41:53:00 14  26 0.538   1    2
## 3 3  2006/11/4 21-309 CLE @ CHA  L (-4)  1 38:06:00  3  13 0.231   1    3
## 4 4  2006/11/7 21-312 CLE   ATL  L (-9)  1 47:17:00 13  26 0.500   3    8
## 5 5  2006/11/9 21-314 CLE   CHI W (+19)  1 37:50:00  6  13 0.462   0    1
## 6 6 2006/11/11 21-316 CLE   BOS  W (+1)  1 43:32:00  9  17 0.529   1    4
##    X3P. FT FTA   FT. ORB DRB TRB AST STL BLK TOV PF PTS GmSc X....
## 1 0.500  2   6 0.333   3   7  10   5   0   2   5  2  26 15.3  +3\\
## 2 0.500  6  11 0.545   1   9  10   4   1   1   2  3  35 25.1  +6\\
## 3 0.333  9  10 0.900   4   5   9   7   0   1   2  0  16 15.6  -3\\
## 4 0.375  5  11 0.455   0   7   7   6   2   1   2  1  34 25.2  -3\\
## 5 0.000  7   8 0.875   0   4   4  12   3   2   3  0  19 22.9 +26\\
## 6 0.250 19  23 0.826   1   7   8   5   3   0   2  4  38 33.8  +6\\
```

```
head(lbj12)
```

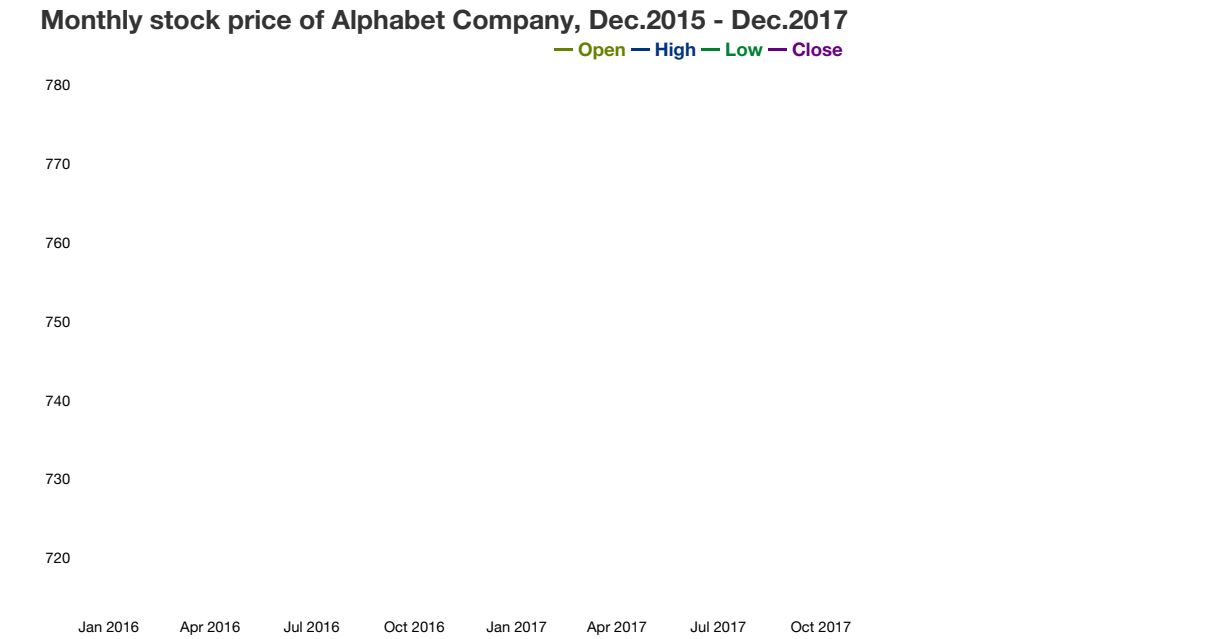
```
##   G       Date    Age  Tm X Opp     X.1 GS       MP FG FGA   FG. X3P X3PA
## 1 1 2012/10/30 27-305 MIA   BOS W (+13)  1 28:52:00 10  16 0.625   2    4
## 2 2  2012/11/2 27-308 MIA @ NYK L (-20)  1 36:41:00  8  16 0.500   2    3
## 3 3  2012/11/3 27-309 MIA   DEN  W (+3)  1 39:12:00  8  17 0.471   0    1
## 4 4  2012/11/5 27-311 MIA   PHO W (+25)  1 30:11:00 10  17 0.588   2    3
## 5 5  2012/11/7 27-313 MIA   BRK W (+30)  1 29:59:00  7  12 0.583   2    3
## 6 6  2012/11/9 27-315 MIA @ ATL  W (+6)  1 35:40:00 10  17 0.588   0    2
##    X3P. FT FTA   FT. ORB DRB TRB AST STL BLK TOV PF PTS GmSc X....
## 1 0.500  4   5 0.800   1   9  10   3   2   0   0  2  26 25.1 +12\\
## 2 0.667  5   6 0.833   0   7   7   5   0   3   5  2  23 16.5 -21\\
## 3 0.000  4   4 1.000   1   8   9  11   0   2   0  2  20 22.7  +7\\
## 4 0.667  1   3 0.333   3   8  11   1   0   0   3  2  23 15.7 +25\\
## 5 0.667  4   4 1.000   4   8  12   8   1   0   2  1  20 23.8 +18\\
## 6 0.000  1   3 0.333   2   9  11   9   1   0   2  2  21 20.9   0\\
```

After dealing with the data, we can use dygraph to visualize the data.

## Graphing

To begin with, we can create a data frame with several variables, and then draw a simple graph like the first one:

```
dat1 <- ts(data = select(google, Open, High, Low, Close), start = c(2015, 12), end = c(2017, 12), frequency = 12)
dygraph(dat1, main = "Monthly stock price of Alphabet Company, Dec.2015 - Dec.2017")
```




This is the monthly stock price of google company within two years, and the data is from the yahoo finance website. To construct this kind of time series graph, the variable of x-axis a special variable called **timeSeries xts** variable. It present the monthly stock price with the order of time.

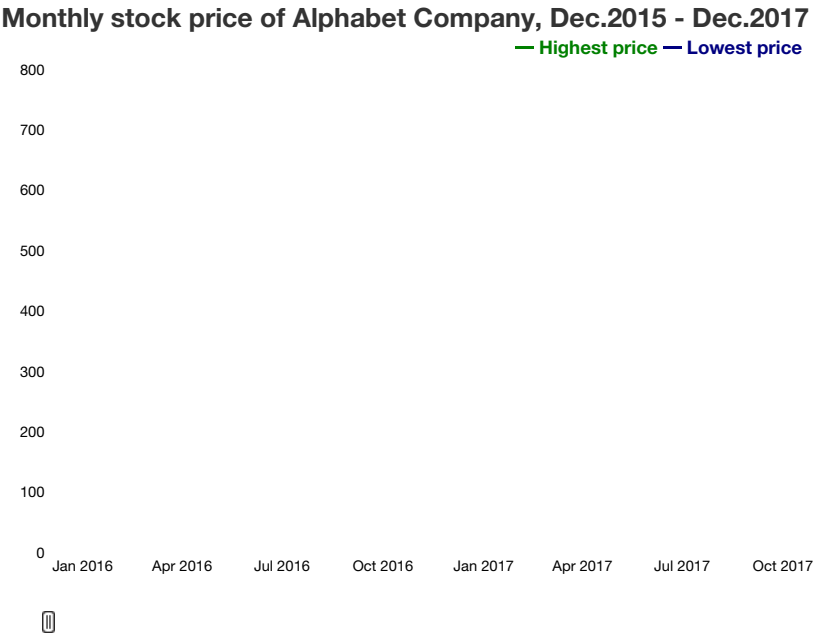
This graph is quiet easy to show the details of the data, but when it come to large size of the data, the cursor cannot get every point accurately, so we need something to zoom in & out, and dygraphs provides this kind of tool. Note that it is similar as the function dplyr, using the **%>%** symbol to combine the seperate element of the graph.

```
# putting a selector on the graph
dygraph(dat1, main = "Monthly stock price of Alphabet Company, Dec.2015 - Dec.2017") %>% dyRangeSelector()
```


As we can see, with the tool of range selector, we can choose any part of the graph to inspect the data details, which is powerful and convenient.

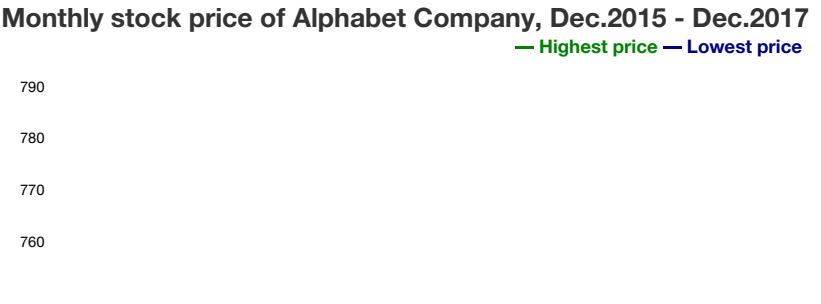
The way of using the symbol **%>%** is very important in this package dygraphs, since we can add multiple layers on the graph, so that adding more element on graph is a lot more easier.

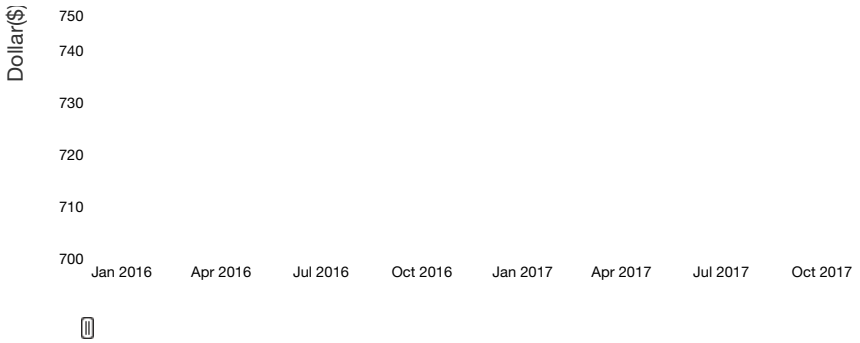
```
# Creating new graph with multiple layers
dat1 <- select(google, High, Low)
dat1$High <- dat1$High - dat1$Low
dat1 <- ts(data = dat1, start = c(2015, 12), end = c(2017, 12), frequency = 12)
dygraph(dat1, main = "Monthly stock price of Alphabet Company, Dec.2015 - Dec.2017") %>%
  dySeries("High", label = "Highest price") %>%
  dySeries("Low", label = "Lowest price") %>%
  dyRangeSelector() %>%
  dyOptions(stackedGraph = TRUE)
```




The graph is not elegant enough since the relative difference between high price and low price is too small, so we need to change the range of the axis:

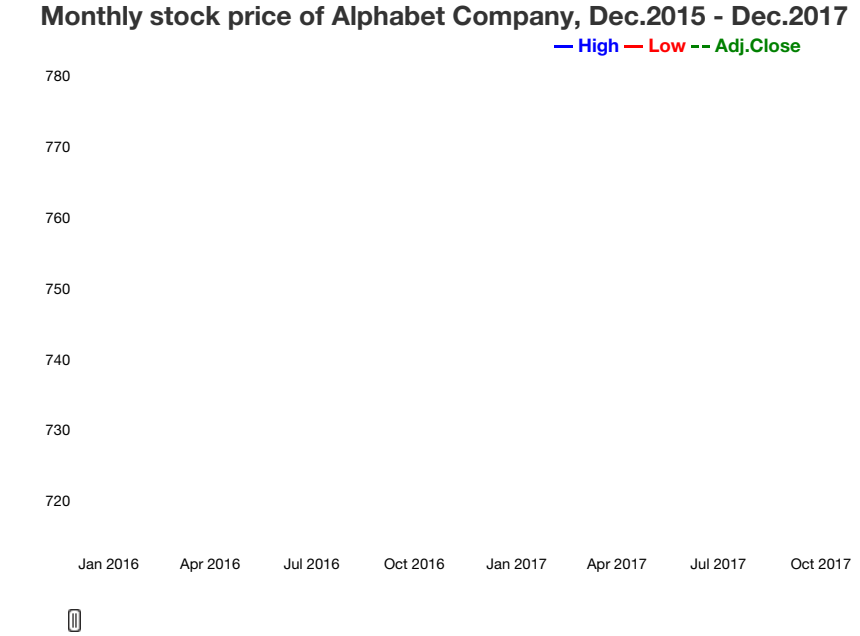
```
# change the range of the axis
dygraph(dat1, main = "Monthly stock price of Alphabet Company, Dec.2015 - Dec.2017") %>%
  dySeries("High", label = "Highest price") %>%
  dySeries("Low", label = "Lowest price") %>%
  dyRangeSelector() %>%
  dyOptions(stackedGraph = TRUE) %>%
  dyAxis("y", label = "Dollar($)", valueRange = c(700, 800))
```





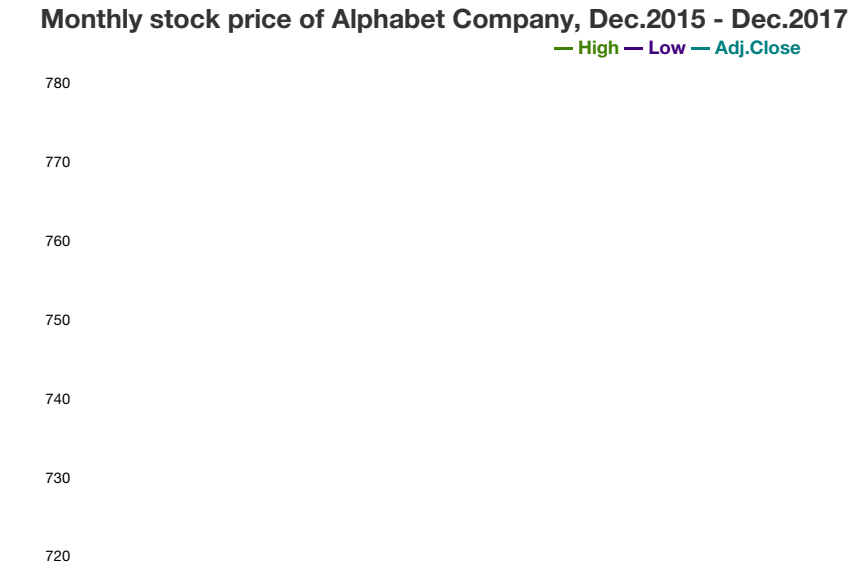

Actually, there are multiple options to operate the graph, like step graph, filling, different shape of points and lines. * We can change the graph to step graphs. * The color below the graph can be filled. * The line can be changed to any shaple, in the given example, it is dashed line. * The point of the graph can be presented as any shape of point.

```
# putting a selector on the graph
dat1 <- ts(data = select(google, High, Low, Adj.Close), start = c(2015, 12), end = c(2017, 12), frequency = 12)
dygraph(dat1, main = "Monthly stock price of Alphabet Company, Dec.2015 - Dec.2017") %>%
  dyRangeSelector() %>%
  dySeries("High", drawPoints = TRUE, color = "blue", pointSize = 4) %>%
  dySeries("Low", stepPlot = TRUE, fillGraph = TRUE, color = "red") %>%
  dySeries("Adj.Close", strokeWidth = 2, strokePattern = "dashed", color = "green")
```



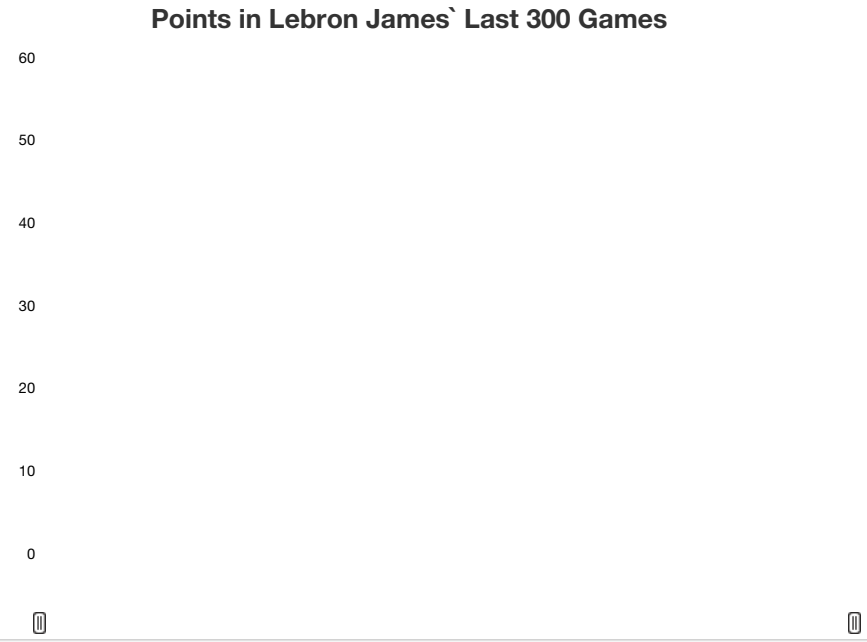

Users and viewers are always want to use the mouse to see the details of the point, so knowing the position of the mouse is also very important. Here is an useful tool that could highlight the graph when the mouse is putting on it. In the following example, the graph is shown as normal when the mouse is not putting on it; and when the mouse is putting on it, the selected graph will be highlighted, and other graphs are weakened.

```
# Creating a highlighted graph
dygraph(dat1, main = "Monthly stock price of Alphabet Company, Dec.2015 - Dec.2017") %>%
  dyHighlight(highlightSeriesOpts = list(strokeWidth = 3))
```

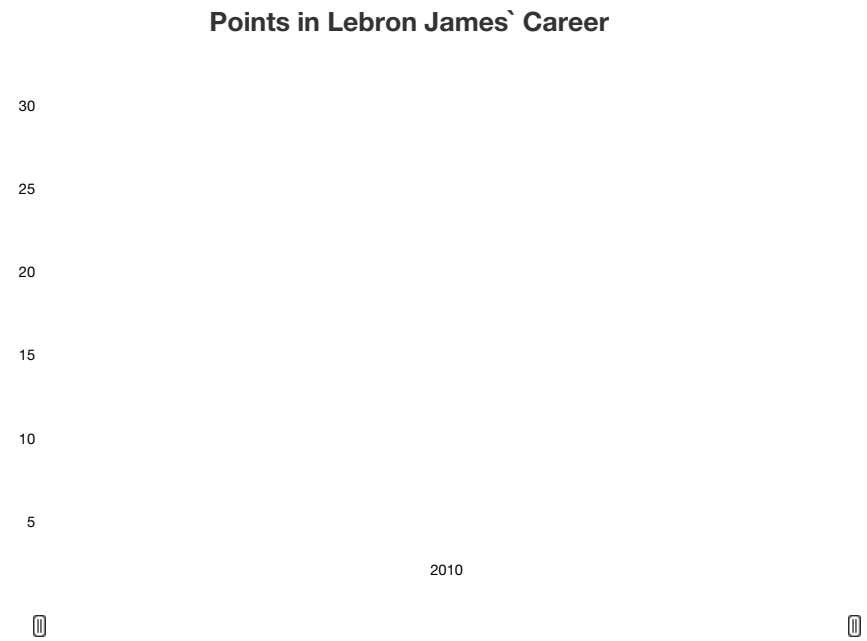


There are still a lot of parameters that is related to the axis and the shape of the graph, and it is easy and interesting to explore. Here is some more useful tools, when people want to see the data more directly using the mouse, we can let the numbers follow the mouse using the function **dyLegend**.

```
# following the mouse
dat2 <- select(lbj, PTS, TRB, AST)
dat2 <- as.matrix(dat2)
rownames(dat2) <- lbj$Date
dat2 <- ts(dat2[816:1126, ])

dygraph(dat2, main = "Points in Lebron James` Last 300 Games") %>%
  dySeries("PTS", label = "Points") %>%
  dyLegend(show = "follow") %>%
  dyRangeSelector(strokeColor = "")
```




```
dat2 <- ts(data = select(lbj_career, PTS, TRB, AST), start = 2003, end = 2017)
dygraph(dat2, main = "Points in Lebron James` Career") %>%
  dySeries("PTS", label = "Points") %>%
  dyLegend(show = "follow") %>%
  dyRangeSelector(strokeColor = "")
```




As we can see that the mouse can show the points, assists and rebounds by Lebron James, and as the mouse move, the data moves as well.

Here is also a very powerful tool for data analysis, especially for financial analysis and stock market price. Using the data of google, the price of everyday, including open, high, low and close is shown clearly by the function **dyCandlestick**

```
library(xts)
```

```
## Loading required package: zoo
```

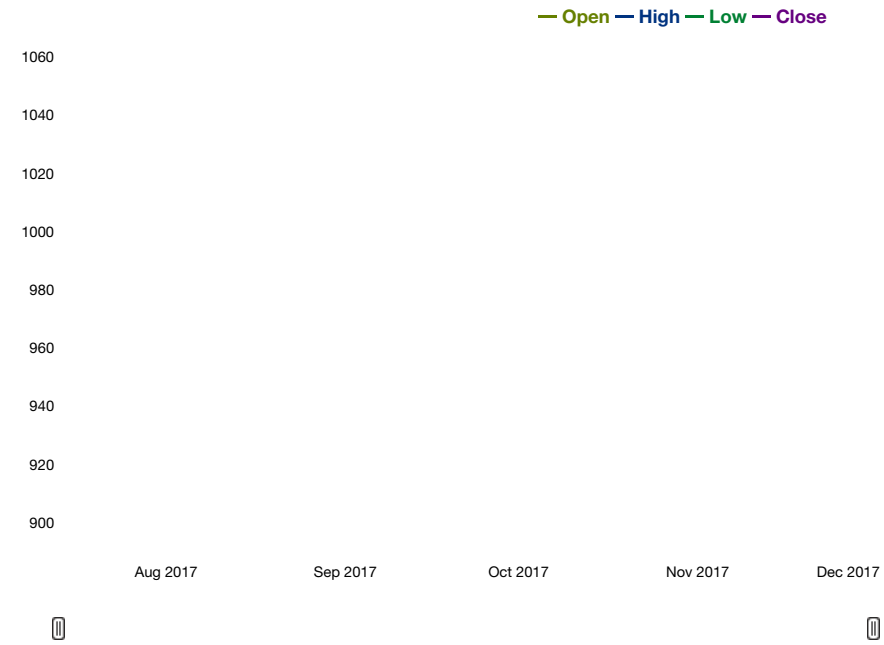
```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
dat3 <- select(google, Open, High, Low, Close)
dat3 <- as.matrix(dat3)
rownames(dat3) <- google$Date
dat3 <- tail(dat3, n = 100)
dygraph(dat3)%>%
  dyCandlestick()%>%
  dyRangeSelector()
```




Using the candle bars is quite easy to show the tendency of the price as well as the daily high and low price.

Last but not the least, we could use dygraph functions to create a barchart, even if the package is not included.

```
library(quantmod)
```

```
## Loading required package: TTR
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

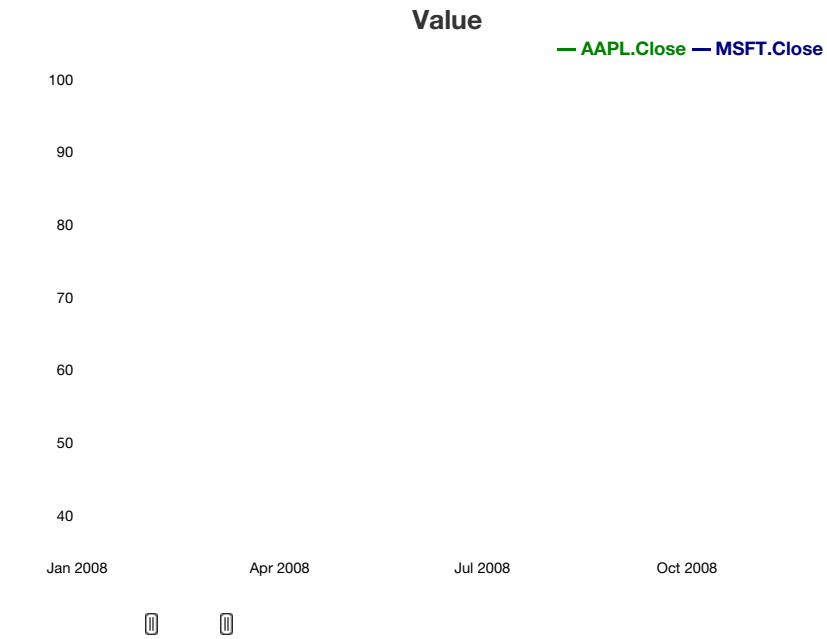
```
tickers <- c("AAPL", "MSFT")
getSymbols(tickers)
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

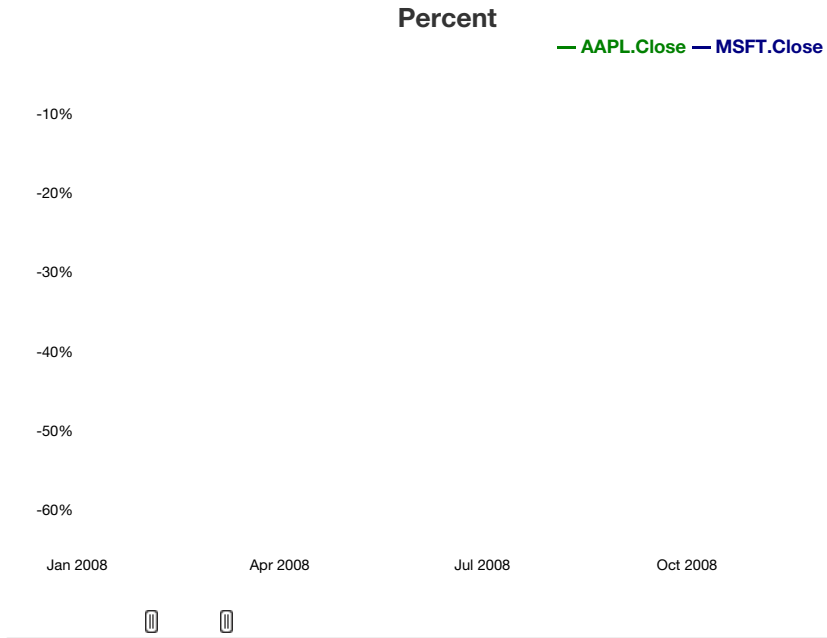
```
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

```
## [1] "AAPL" "MSFT"
```

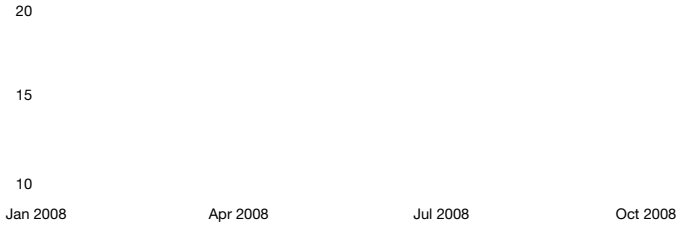
```
closePrices <- do.call(merge, lapply(tickers, function(x) Cl(get(x))))
dateWindow <- c("2008-01-01", "2009-01-01")

dygraph(closePrices, main = "Value", group = "stock") %>%
  dyRebase(value = 100) %>%
  dyRangeSelector(dateWindow = dateWindow)
```




```
dygraph(closePrices, main = "Percent", group = "stock") %>%
  dyRebase(percent = TRUE) %>%
  dyRangeSelector(dateWindow = dateWindow)
```




```
dygraph(closePrices, main = "None", group = "stock") %>%
  dyRangeSelector(dateWindow = dateWindow)
```




The three graphs, representing different meanings of the stock, can be manipulated using the range selector at the same time, so we can easily check the informations.

## Conclusion

- The package **dygrphs** is a great package of data visualizations, especially using the time series objects
- The functions in the packages is added as layer, just as the package 'ggplot2'.
- Before using the package to create the graph, users have to be familiar with the time series object, which is a multiple vector with x-value time.
- The color and the path of the graph is also similar to 'ggplot2'.
- For me the most powerful tool in the package is the range selecter, and the data information changed by mouse.

## Reference


- Great R packages for data import, wrangling and visualization. https://www.computerworld.com/article/2921176/business-intelligence/great-r-packages-for-data-import-wrangling-visualization.html
- Tutorial: https://rstudio.github.io/dygraphs/gallery-series-options.html
- Package 'dygraphs'https://cran.r-project.org/web/packages/dygraphs/dygraphs.pdf
- Data source: yahoo finance finance.yahoo.com
- Data source: basketball reference https://www.basketball-reference.com
- Time Series object: https://www.statmethods.net/advstats/timeseries.html
- Time Series object: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/ts.html
- dygraphs https://github.com/rstudio/dygraphs