




Reproducible Finance with R: The Sharpe Ratio

📅 2016-11-09

by Jonathan Regenstein


Financial applications were an early driving force behind the adoption of the R language, but as data science becomes increasingly critical to banks, hedge funds, investment managers, data providers, [EXCHANGES](#) , etc., R is becoming even more important to Finance. We are excited and inspired by what the future holds in the brave new world of data-driven financial institutions. In this first post of what we hope will be a regular series about Finance, R and RStudio, we present a classic use case: import stock data, build a portfolio, and calculate the Sharpe Ratio. We use the new RStudio IDE Notebooks tool to emphasize reproducibility.

By way of a brief roadmap, we'll start with a function that grabs monthly stock returns and saves those monthly returns as an xts object in the global environment. With that function, we will create three xts objects of monthly returns, and merge those three xts objects into one object, before passing that merged object to dygraphs to peek at the individual stocks. Then, we move on to build a portfolio by selecting asset weights and calculating the portfolio monthly returns. Next, we will calculate the growth of a dollar invested in that portfolio (which is what matters to us) over time, and save the results to an xts object. Dygraphs will come in handy again for the portfolio visualizations. Finally, we will calculate the Sharpe Ratio.

Before we get to the fun stuff - the code itself! - I will note that these posts aim to be useful to R coders who either work in the financial industry or want to work in the financial industry. At times the posts will contain ideas or paradigms that are good practice in the institutional context but perhaps not directly relevant to an R user on the lookout for one-off coding tricks. That is a good segue into the structure and content of this Notebook, which could be fit comfortably into one setup chunk and one substantive code chunk, instead of my five chunks.


I chose to use five chunks to demonstrate how this document might fit into the workflow of a team of analysts with varying levels of R knowledge. Team workflow +

reproducibility is going to be a recurrent theme in this series, as it is a recurrent theme at financial institutions using the RStudio toolchain.

Five chunks breaks this code into simple steps: build an import function, choose/visualize the stocks, choose portfolio weights/visualize returns, and calculate the Sharpe Ratio. From a team workflow perspective, this structure could be an introductory template for analysts that are just learning R, or it could serve as the starting point for advanced users who want to use the data import and return functions before they add complexity. For example, calculating the Sharpe Ratio is just the beginning of risk-to-volatility analysis, and most [SHOPS](#)  have their own proprietary algorithms/models to advance the analysis. Beginners or advanced users could start with a common Notebook template like this one, to standardize the data import and basic visualization workflow.

Now, onwards to the setup code chunk, where we load three packages: quantmod to download the data, PerformanceAnalytics to run portfolio calculations, and dygraphs to graph time series objects. We will also create a function to import stock data.

```
library(PerformanceAnalytics)
library(quantmod)
library(dygraphs)

# Function to calculate monthly returns on a stock
monthly_stock_returns <- function(ticker, start_year) {
  # Download the data from YAHOO  finance
  symbol <- getSymbols(ticker, src = 'yahoo', auto.assign = FALSE, warnings = FALSE)
  # Tranform it to monthly returns using the periodReturn function from quantmod
  data <- periodReturn(symbol, period = 'monthly', subset=paste(start_year, ":", s
    type = 'log')

  # Let's rename the column of returns to something intuitive because the column na
  # will eventually be displayed on the time series graph.
  colnames(data) <- as.character(ticker)

  # We want to be able to work with the xts objects that result from this function
  # so let's explicitly put them to the global environment with an easy to use
  # name, the stock ticker.
  assign(ticker, data, .GlobalEnv)
}
```

The `monthly_stock_returns` function above takes 2 parameters, a stock symbol and a year. Note that we could have included a third parameter called something like 'period' if we wanted the ability to grab periods other than monthly returns. For example, we can envision a desire to look at annual, weekly or daily returns. Here, I force monthly returns because I don't want to allow different period options. It's a choice driven by the purpose of this Notebook - which here is focused on monthly returns.

In the next chunk, we choose three stock tickers and a starting year argument for the `monthly_stock_returns` function. Then, we merge them into one xts object and graph

their individual performances over time.

```
# Choose the starting year and assign it to the 'year' variable. How about 2010?
```

```
year <- 2010
```

```
# Use the function the monthly returns on 3 stocks, and pass in the 'year' value
```

```
# Let's choose Google, JP Morgan and AMAZON ; after you run these functions, have
```

```
# a look at the global environment and make sure your three xts objects are there
```

```
monthly_stock_returns('GOOG', year)
```

```
## As of 0.4-0, 'getSymbols' uses env=parent.frame() and
```

```
## auto.assign=TRUE by default.
```

```
##
```

```
## This behavior will be phased out in 0.5-0 when the call will
```

```
## default to use auto.assign=FALSE. getOption("getSymbols.env") and
```

```
## getOptions("getSymbols.auto.assign") are now 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 more details.
```

```
monthly_stock_returns('JPM', year)
```

```
monthly_stock_returns('AMZN', year)
```

```
# Merge the 3 monthly return xts objects into 1 xts object.
```

```
merged_returns <- merge.xts(GOOG, JPM, AMZN)
```

```
# Before we combine these into a portfolio, graph the individual returns and
```

```
# see if anything jumps out as unusual. It looks like something happened to
```

```
# GOOGLE  in March of 2014, but that something didn't affect JP Morgan or Amazon.
```

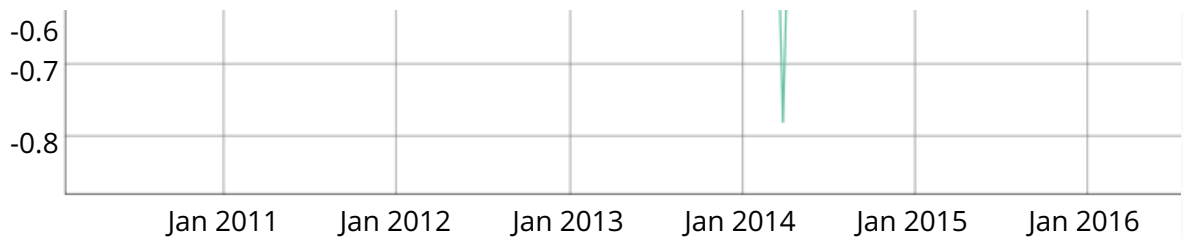
```
dygraph(merged_returns, main = "Google v JP Morgan v Amazon") %>%
```

```
  dyAxis("y", label = "%") %>%
```

```
  dyOptions(colors = RColorBrewer::brewer.pal(3, "Set2"))
```

Google v JP Morgan v Amazon



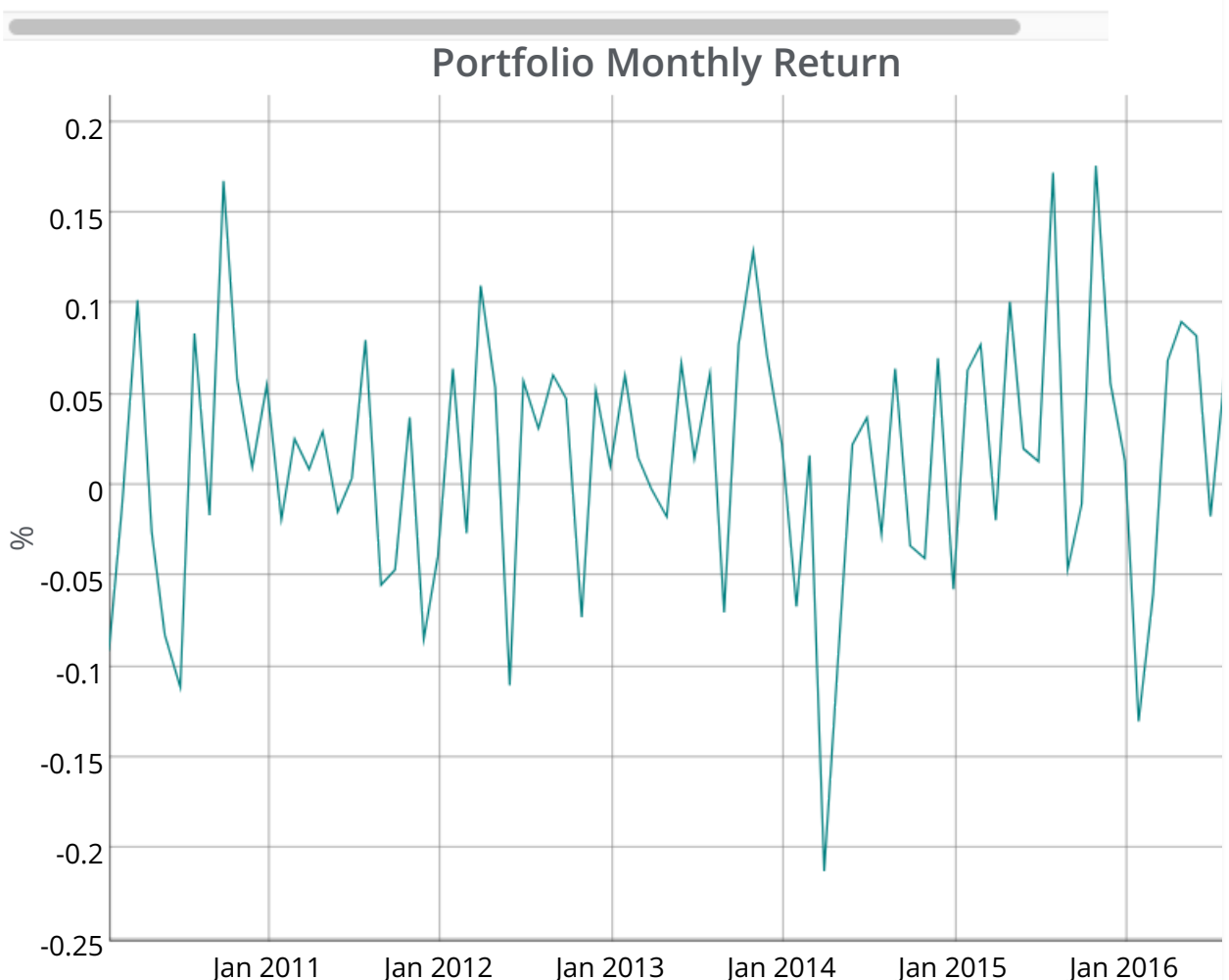


Nothing earth-shattering thus far: we have an xts object of three time series and have seen that one of them had weird behavior in April of 2014 (a [GOOGLE](#) stock split). We'll ignore that behavior for this example and go on to constructing a portfolio, which means finding the monthly returns of a weighted combination of assets. Unsurprisingly, we start out by choosing those weights.

```
# We have the 3 monthly returns saved in 1 object.
# Now, let's choose the respective weights of those 3.
# Here we'll allocate 25% to Google, 25% to JP Morgan and 50% to Amazon.
w <- c(.25, .25, .50)

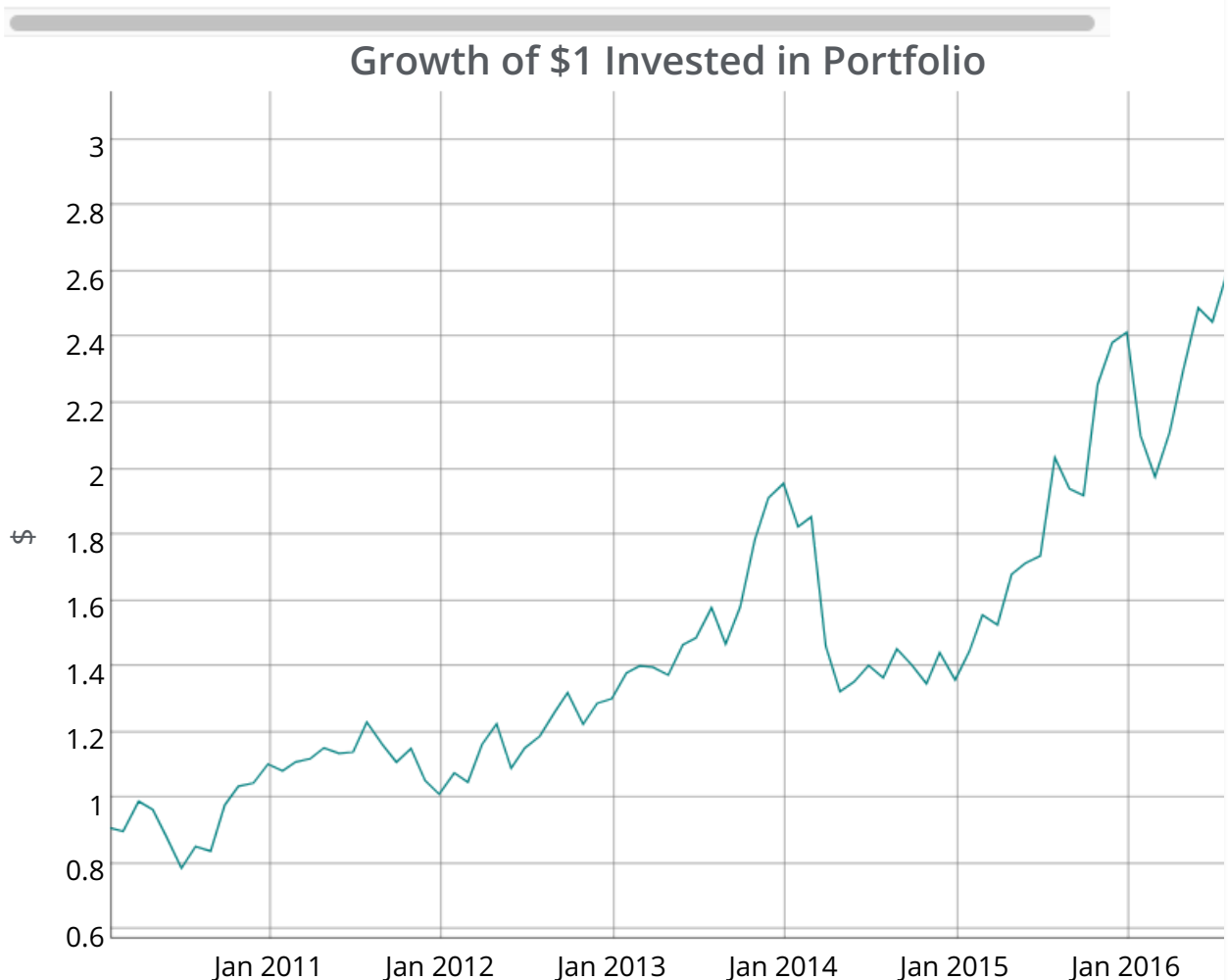
# Now use the built in PerformanceAnalytics function Return.portfolio
# to calculate the monthly returns on the portfolio, supplying the vector of weight
portfolio_monthly_returns <- Return.portfolio(merged_returns, weights = w)

# Use dygraphs to chart the portfolio monthly returns.
dygraph(portfolio_monthly_returns, main = "Portfolio Monthly Return") %>%
  dyAxis("y", label = "%")
```



Now, instead of looking at monthly returns, let's look at how \$1 would have grown in this portfolio.

```
# Add the wealth.index = TRUE argument and, instead of returning monthly returns,  
# the function will return the growth of $1 invested in the portfolio.  
dollar_growth <- Return.portfolio(merged_returns, weights = w, wealth.index = TRUE)  
  
# Use dygraphs to chart the growth of $1 in the portfolio.  
dygraph(dollar_growth, main = "Growth of $1 Invested in Portfolio") %>%  
  dyAxis("y", label = "$")
```



A dollar would have grown quite nicely in this portfolio - about 2.5x - fantastic. Now let's look at the risk/reward of this portfolio by calculating the Sharpe Ratio. Briefly, the Sharpe Ratio is the mean of the excess monthly returns above the risk-free rate, divided by the standard deviation of the excess monthly returns above the risk-free rate. This is the formulation of the Sharpe Ratio as of 1994; if we wished to use the original formulation from 1966 the denominator would be the standard deviation of portfolio monthly returns. Learn more [here](#).

In other words, the Sharpe Ratio measures excess returns per unit of volatility, where we take the standard deviation to represent portfolio volatility. The Sharpe Ratio was brought to us by Bill Sharpe - arguably the most important economist for modern investment management as the creator of the Sharpe Ratio, CAPM and Financial Engines, a forerunner of today's robo-advisor movement.

In the code chunk below, we'll calculate the Sharpe Ratio in two ways.

First, we'll use the `Return.excess` function from `PerformanceAnalytics` to calculate a time series of monthly excess returns. Two arguments need to be supplied: the time series of returns and the risk-free rate. The function will return a time series of excess returns, and we'll take the mean of that time series to get the numerator of the Sharpe Ratio. Then we'll divide by the standard deviation of the that time series to get the Sharpe Ratio.

Our second method is a bit easier. We'll use the `SharpeRatio` function in `PerformanceAnalytics`, for which we'll supply two arguments: a time series of monthly returns and risk-free rate.

For both methods I use a risk-free rate of .03% as the approximate mean of the 1-month Treasury bill rate since 2010. I'll cover a quick way to grab this and other data via Quandl in a future post.

```
# Method 1: use the Return.excess function from PerformanceAnalytics,
# then calculate the Sharpe Ratio manually.
portfolio_excess_returns <- Return.excess(portfolio_monthly_returns, Rf = .0003)
sharpe_ratio_manual <- round(mean(portfolio_excess_returns)/StdDev(portfolio_excess_returns), 4)

# If we wanted to use the original, 1966 formulation of the Sharpe Ratio, there is
# change to the code in Method 1.
# sharpe_ratio_manual <- round(mean(portfolio_excess_returns)/StdDev(portfolio_monthly_returns), 4)

# Method 2: use the built in SharpeRatio function in PerformanceAnalytics.
sharpe_ratio <- round(SharpeRatio(portfolio_monthly_returns, Rf = .0003), 4)
```

Using the `Return.excess` function and then dividing by the standard deviation of excess returns, the Sharpe Ratio is `sharpe_ratio_manual[,1] = 0.211`.

Using the built in `SharpeRatio` function, the Sharpe Ratio is `sharpe_ratio[1,] = 0.211`.

Alright, we have built a portfolio and calculated the Sharpe Ratio - and also set up some nice reusable chunks for data import, portfolio construction and visualization. We haven't done anything terribly complex but this can serve as a useful paradigm to any collaborators, including our future selves, who want to reproduce this work, learn from this work, or expand upon this work.

In the next post, we will create a Shiny app to let users enter their own stock tickers, starting year, portfolio weights and risk-free rate, and then calculate a portfolio Sharpe Ratio. See you next time!

You may leave a comment below or discuss the post in the forum community.rstudio.com.