Post 2: Some Financial Applications of R

Athan Diep

December 2, 2017

Introduction and Motivation

R has become an increasingly popular programming language for use in financial applications over the years in part due to its wide capabilities for working with and analyzing data. Its open-source nature has also allowed R's range of capabilities within finance to widen as more packages, functions, and products are created by developers and provided for public use. The language has made itself popular among the data scientists of the finance industry, often called quantitative analysts or "quants". In this post, I will explore various packages and functions in R for importing, wrangling, visualizing, and analyzing financial data.

We will import and work with daily pricing data for a few company stocks and introduce various functions and tools that might be useful for a common analysis by a finance professional.

Loading a Few Useful Finance Packages

There are many packages to chose from for working with financial data, but I am going to feature a few of the most popular ones.

```
#Load Quantmod, a package with many functions for analyzing and importing stock data
library(quantmod)
#Load xts, a useful package for manipulating time-series data
library(xts)
#Load ggplot2 to visualize our data
library(ggplot2)
#Load lubridate to make it easier to work with dates
library(lubridate)
```

Importing Financial Data using 'quantmod'

Quantmod is a useful package made for importing stock data into R. It includes a helpful function, *getSymbols* for importing stock data into R. It takes a single or a vector of character stock ticker symbols as input as well as other optional parameters such as "from" and "to" that allow you to specify the timespan of stock price data that you would like. "src" can also be specified which is the source of the data (default is "yahoo"). The function creates xts object(s) in your global environment (or whichever you specify) that includes the daily open, close, high, low, and adjusted price as well as volume.

I chose Advance Auto Parts and O'Reilly Automotive, two auto-part retailers, to load into my environment. Their ticker symbols are AAP and ORLY, respectively (see sources for ticker symbol reference). The function will, by default, create the new objects with the same name as the ticker symbols.

```
#Loading daily price data for the stock of Advance Auto Parts
getSymbols(c("AAP", "ORLY"), from = "2010-1-1", to = "2017-12-1")
```

```
## [1] "AAP"  "ORLY"
```

Let's inspect the new xts objects:

```
#Check the class of the new objects
class(ORLY)
```

```
## [1] "xts" "zoo"
```

Xts and Zoo are both packages used for time series data.

```
#View head of xts object
head(ORLY)
```

```
##            ORLY.Open ORLY.High ORLY.Low ORLY.Close ORLY.Volume
## 2010-01-04     39.24     39.24    38.10      38.50     1278200
## 2010-01-05     38.35     38.57    38.07      38.56     1134100
## 2010-01-06     38.44     38.75    38.18      38.39     1105000
## 2010-01-07     38.22     38.58    38.15      38.27      695300
## 2010-01-08     38.11     38.31    37.89      37.93      875000
## 2010-01-11     37.93     38.17    37.55      38.13      778200
##            ORLY.Adjusted
## 2010-01-04         38.50
## 2010-01-05         38.56
## 2010-01-06         38.39
## 2010-01-07         38.27
## 2010-01-08         37.93
## 2010-01-11         38.13
```

```
#Examine the structure of ORLY
str(ORLY)
```

```
## An 'xts' object on 2010-01-04/2017-11-30 containing:
##   Data: num [1:1993, 1:6] 39.2 38.3 38.4 38.2 38.1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "ORLY.Open" "ORLY.High" "ORLY.Low" "ORLY.Close" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "yahoo"
##  $ updated: POSIXct[1:1], format: "2017-12-03 22:00:39"
```

Calculating Financial Metrics

When analyzing a potential investment, its usually necessary to examine some different metrics for a given stock. A common measure is the daily return of a stock. This is calculated as (current day's closing price - previous day's closing price) / previous day's closing price. We might use this to have a normalized performance metric that is comparable between different securities. Let's add a new column to our stock price tables with daily returns.

```
#Creating a function that will return a vector of returns (starting on the second day) given a vector of closing p
rices
calculate_returns <- function(prices){
  returns <- rep(NA, length(prices))
  prices <- as.numeric(prices)
  for (i in 2:length(prices)){
    returns[i] <- (prices[i] - prices[i-1]) / (prices[i-1])
  }
  return(returns)
}

#Calculate and Add Daily Return Column to the xts tables
ORLY$returns <- calculate_returns(ORLY$ORLY.Adjusted)
AAP$returns <- calculate_returns(AAP$AAP.Adjusted)
```

The *dailyReturn* function from *quantmod* essentially completes the same task as this function with additional features such as log returns, another useful measure of periodic returns that is suitable under the assumption that the returns follow a lognormal distribution. Let's also add another column to each table with lognormal returns using quantmod's dailyReturn function with the parameter "type" set to "log". Note that dailyReturn is one of the family of periodReturn functions in quantmod. There is also weeklyReturn, monthlyReturn, annualReturn, etc.

```
ORLY$log_returns <- dailyReturn(ORLY$ORLY.Adjusted, type = 'log')
AAP$log_returns <- dailyReturn(AAP$AAP.Adjusted, type = 'log')
```

Another common metric that is useful is the moving average of the prices. We can use a standard 10 or 30 day window for calculating the price average and add it as another column to our xts tables. A stock's 30 day average is simply the average price for the last 30 days. It is often important to see how this average changes over time and to compare it to a moving average of a shorter timespan or of another stock.

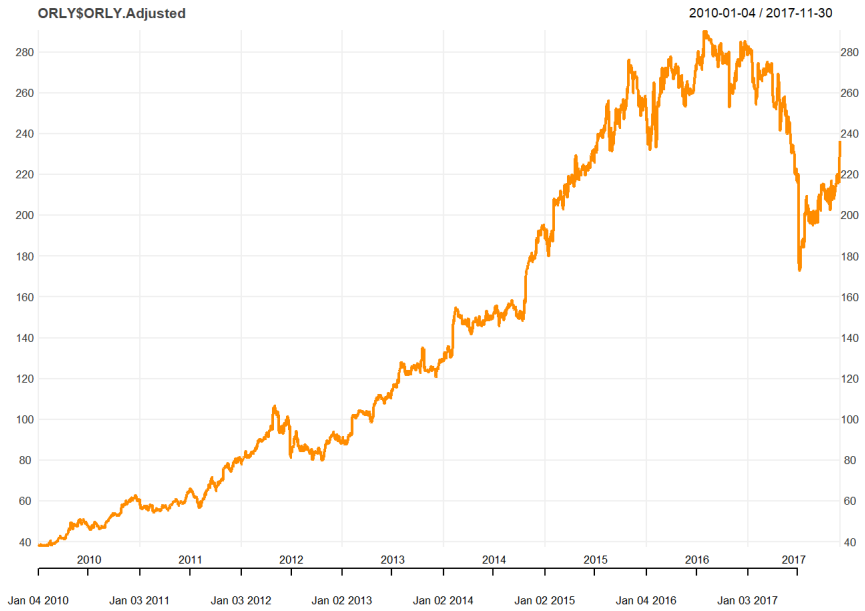
```
#Creating a function that will return a vector of the moving averages given a vector of prices
calculate_averages <- function(prices, timespan = 30){
  averages <- rep(NA, length(prices))
  prices <- as.numeric(prices)
  if (timespan > length(prices)){
    print("Error: price vector is not long enough")
  }else {
    for (i in timespan:length(prices)){
      averages[i] <- mean(prices[(i - timespan + 1):i])
    }
    return(averages)
  }
}

#Calculate and Add 10-day and 30-day Moving Average Column to both tables
ORLY$average_thirty <- calculate_averages(ORLY$ORLY.Adjusted, 30)
AAP$average_thirty <- calculate_averages(AAP$AAP.Adjusted, 30)
ORLY$average_ten <- calculate_averages(ORLY$ORLY.Adjusted, 10)
AAP$average_ten <- calculate_averages(AAP$AAP.Adjusted, 10)
```

Visualizing Data

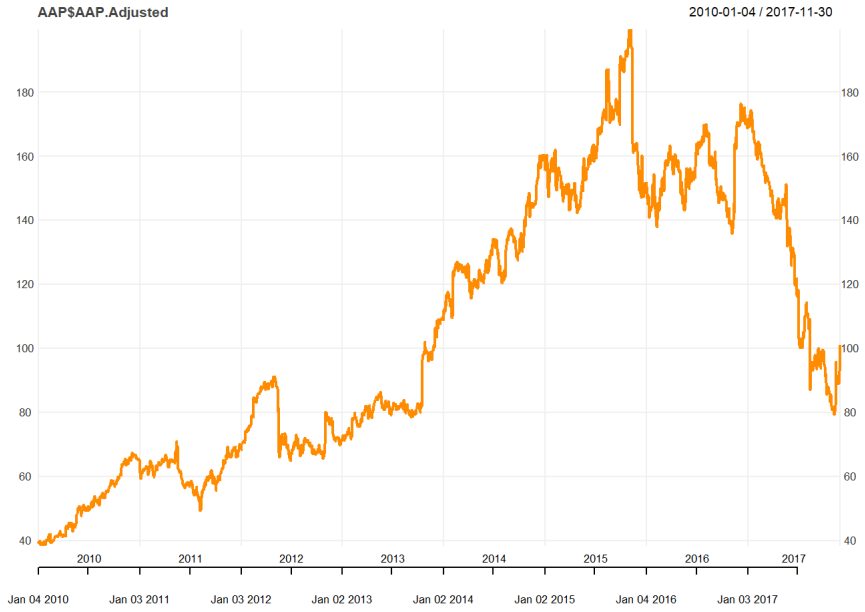
Visualizing data is a key part of financial analysis, especially to view how prices and differnt metrics change over time. There are a few options for visualizing the xts data that we have imported. *quantmod* has some useful features for visualizing the data in addition to *ggplot*.

The *chart_Series* function from *quantmod* has many cool features for charting financial data. Let's first chart the closing adjusted prices for both stocks.

```
#Plot closing adjusted price for O'Reilly
chart_Series(ORLY$ORLY.Adjusted)
```




```
#Plot closing adjusted prices for Advance Auto Parts
chart_Series(AAP$AAP.Adjusted)
```



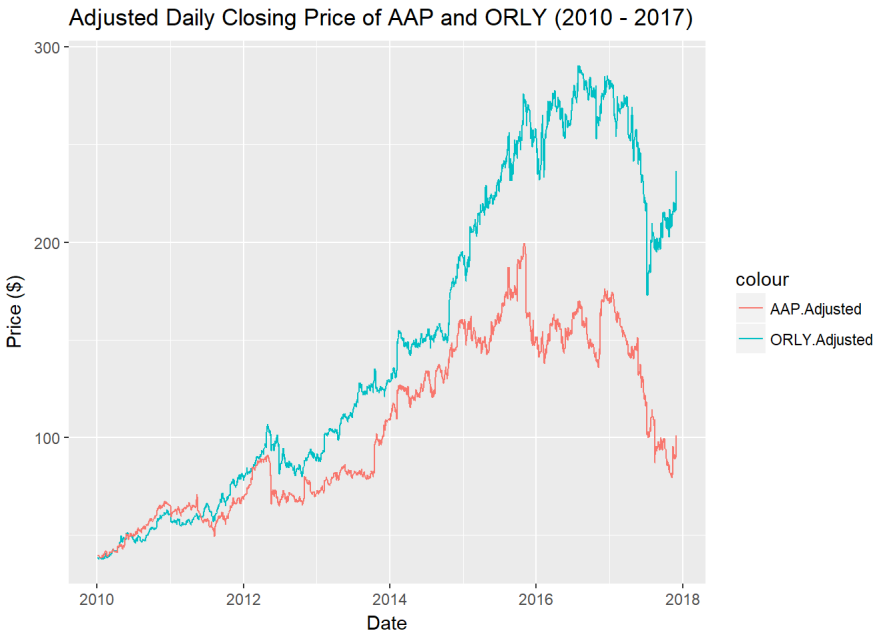
Another common visualization for stock prices is the candlestick chart, a chart which shows each day's high, opening, low, and closing price. Quantmod includes a *candleChart* function for this which also allows us to add a few more features to the chart such as:
* subset: restrict time span displayed
* Bollinger Bands: bars that display two standard deviations above and below the price of the stock
* Volume: the total trades for the day
* theme: color theme of the chart

Note: To add the Bollinger Bands and volume, we have to specify the "TA" parameter with "addBBands()" and "addVo()" separated by a semicolon. See below.

```
#Charting AAP in 2016 with new added features for more in-depth analysis
candleChart(AAP, TA='addBBands();
                     addVo()',
                subset='2016',
                theme="black"
                )
```

We can also try using the ggplot package to plot the prices and returns of the two stocks on the same or separate charts for a better visual comparison.

```
#Plot the prices of both AAP and ORLY on the same chart. We need to combine and align both datasets into the same dataframe first.

df_closing <- data.frame(date = index(ORLY), ORLY = ORLY$ORLY.Adjusted, AAP = AAP$AAP.Adjusted)

ggplot(df_closing, aes(x = date)) +
  geom_line(aes(y = ORLY.Adjusted, colour = "ORLY.Adjusted")) +
  geom_line(aes(y = AAP.Adjusted, colour = "AAP.Adjusted")) +
  ggtitle("Adjusted Daily Closing Price of AAP and ORLY (2010 - 2017)") +
  xlab("Date") +
  ylab("Price ($)")
```

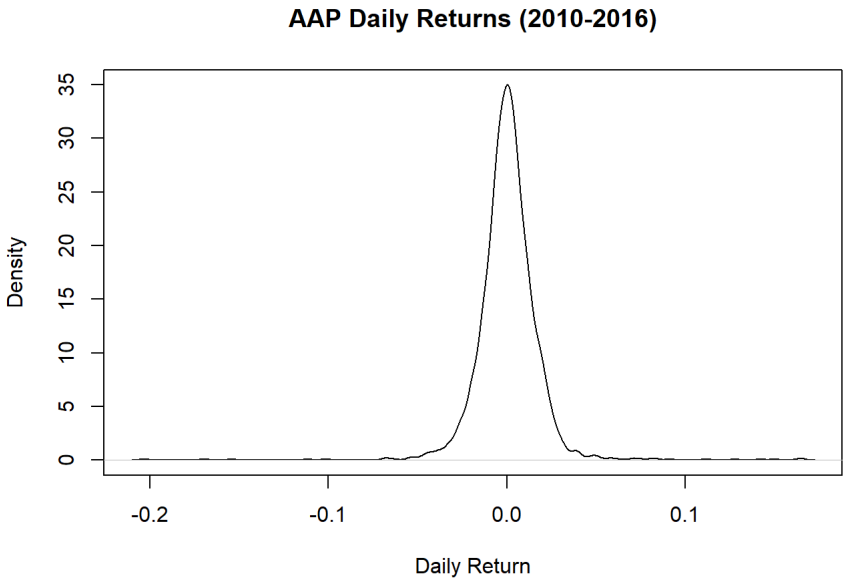


Simulating Financial Data

Now that we have collected, calculated, and visualized our data, we can take things a step further. As in the interest of any investment professional who is making a decision on whether to invest in a particular security or not, how the price of the stock will behave in the future is very important. As such, some form of simulation or time-series analysis is done to make a reasonable prediction on where the stock price will go in the future.

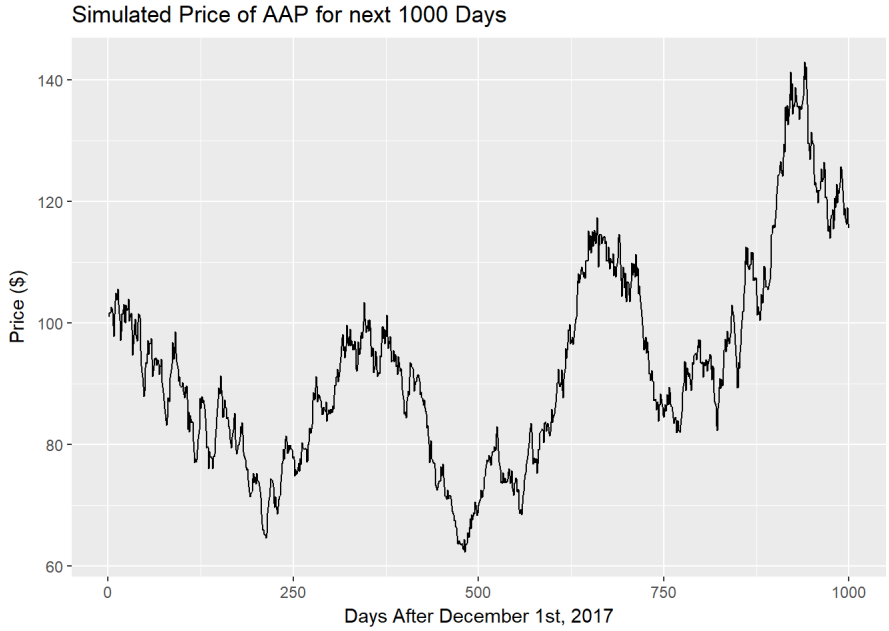
While the process of simulating a random variable can be overwhelmingly complex, we can make a simple and naive simulation of the stock price under the assumption that the returns, not the actual price, of the stock is normally distributed. Let's take a look at the distribution of returns for AAP by plotting its density:

```
#Plot the density of AAP returns with base package plot function. Because the first value for returns is missing, we start with the second.
plot(density(AAP$returns[2:nrow(AAP)]), main = "AAP Daily Returns (2010-2016)", xlab = "Daily Return")
```



While it is naive to assume the distribution of daily returns is normal for real applications, it is an adequate assumption for our purposes of simply introducing simulation basics. Thus, we can use the sample mean and standard deviation for estimating our distribution's parameters. We are also making several other assumptions such as that this process is stationary, meaning that the mean and variance will not change over time.

```
#Figuring out the parameters
N <- 1000
days <- 1:N
AAP_return_mean <- mean(AAP$returns[2:nrow(AAP)])
AAP_return_sd <- sd(AAP$returns[2:nrow(AAP)])

#Setting the last day's price as our starting price for simulating the future
AAP_initial <- as.numeric(AAP$AAP.Adjusted[nrow(AAP)])

#Simulate the price using set.seed() and rnorm() function to sample randomly from the normal distribution
set.seed(268)
AAP_price <- c(AAP_initial, rep(NA, N-1))

#Now we calculate the daily price using our sampled returns
for (i in 2:N){
  AAP_price[i] <- AAP_price[i-1] * (1 + rnorm(1, mean = AAP_return_mean, sd = AAP_return_sd))
}

AAP_simulated <- as.data.frame(cbind(days, AAP_price))

#Visualize the simulated price for the next 1000 days
ggplot(AAP_simulated, aes(x = days)) +
  geom_line(aes(y = AAP_price)) +
  ggtitle("Simulated Price of AAP for next 1000 Days") +
  xlab("Days After December 1st, 2017") +
  ylab("Price ($)")
```



Conclusion

In this post, we explored a few of the many possible applications of R to finance. Just as in other domains, data analysis within finance typically follows the standard lifecycle that begins with data acquisition and preparation. Thankfully, many R packages made for finance and stocks make it really easy to import the clean data into our environment often in an already usable form. As we saw, it is possible to make our own functions to make new metrics from the data, and there is also many open-source packages out there that include several more different functions and packages out there for doing different analyses. Visualizing data is also made easy with the quantmod and ggplot packages that plot time series data very conveniently. Lastly, I also demonstrated the very basics of applying simulation tools in R to stocks.

Sources

Overview of R's finance applications: https://www.slideshare.net/LiangChengZhang/r-in-finance-introduction-to-r-and-its-applications-in-finance

Stock Analysis in R: https://www.r-bloggers.com/quantitative-stock-analysis-tutorial-screening-the-returns-for-every-sampp500-stock-in-less-than-5-minutes/

List of finance-related R packages: https://cran.r-project.org/web/views/Finance.html

Monte Carlo Simulation of a portfolio: https://www.r-bloggers.com/random-portfolios-versus-monte-carlo/

Quantmod documentation: https://www.quantmod.com/

Ticker Symbol Lookup Tool: https://www.marketwatch.com/tools/quotes/lookup.asp

Explanation on log returns: https://quantivity.wordpress.com/2011/02/21/why-log-returns/

Explanation on stationarity: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm