MQIM R BOOTCAMP

Financial Data & R Class, Object and functions

Monday, Dec 7, 2020

Section 1

Bootcamp Week Schedule

Table of Content

Section 2

Getting Financial Data in R

Obtaining Financial Data in R

- Many R packages have built in functionality for downloading financial data from either free sources (Yahoo!Finance, etc.) or commercial vendors (Bloomberg or FactSet, for example).
- We'll primarily use free data sources in this class (or I will provide data sets when free sources won't cover our needs) but for those with Bloomberg access here at UNB or at work, the packages *Rbbg* and *Rblpapi* can be very useful.
- Note that Bloomberg does not suport either of these packages - they appear to be OK with users accessing the API in this way, but will of course offer absolutely no support for Rblpapi users experiencing issues. Do NOT ask the Bloomberg Help Desk for assistance with Rblpapi problems.

Free Sources

- In general, Yahoo!Finance is a pretty good source of equity pricing data, and Quandl is a good source of pricing and fundamentals (both free and non free) for equities and other securities, as well as some economic data.
- Oanda has some FX data, and Google Finance can also be used to obtain some equity pricing.
- The packages we'll use for accessing these sources are *tseries*, *quantmod*, *Quandl*, in addition to *Rblpapi* for Bloomberg data.

tseries

The *tseries* package can download data from Yahoo!Finance or Oanda:

```
> library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

The function *get.hist.quote()* (refer to *?get.hist.quote* for usage details) can download data into a time series object in R - lets get a few years of S&P 500 prices from Yahoo!Finance:

tseries

```
> get.hist.quote("^GSPC","2019-12-31","2020-10-31",
+ provider="yahoo",retclass="zoo")
```

```
## '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.e
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by set
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for detai

## time series ends   2020-10-30

##                 Open    High     Low   Close
## 2019-12-31 3215.18 3231.72 3212.03 3230.78
## 2020-01-02 3244.67 3258.14 3235.53 3257.85
## 2020-01-03 3226.36 3246.15 3222.34 3234.85
## 2020-01-06 3217.55 3246.84 3214.64 3246.28
## 2020-01-07 3241.86 3244.91 3232.43 3237.18
```

quantmod

quantmod is a package for quantitative modeling of financial data and includes a variety of functions to obtain, process, and plot financial time series from multiple sources. Let's use the *getSymbols* function in the *quantmod* package to get the past 5 years of USDCAD exchange rates:

```
> library(quantmod)
> USDCAD <- getSymbols("CAD=X",src = "yahoo",auto.assign = F)
```

```
## Warning: CAD=X contains missing values. Some functions will not w
## contain missing values in the middle of the series. Consider usin
## na.approx(), na.fill(), etc to remove or replace them.
```

```
> tail(USDCAD,2) # show final two data points
```

```
##            CAD=X.Open CAD=X.High CAD=X.Low CAD=X.Close CAD=X.Volu
## 2020-12-07    1.27814    1.28320   1.27760     1.27796
## 2020-12-08    1.28037    1.28159   1.27696     1.28020
##            CAD=X.Adjusted
## 2020-12-07        1.27796
## 2020-12-08        1.28020
```

quantmod

```
> library(quantmod)
> tail(USDCAD,2) # show final two data points
```

```
##            CAD=X.Open CAD=X.High CAD=X.Low CAD=X.Close CAD=X.Volu
## 2020-12-07    1.27814    1.28320   1.27760     1.27796
## 2020-12-08    1.28037    1.28159   1.27696     1.28020
##            CAD=X.Adjusted
## 2020-12-07        1.27796
## 2020-12-08        1.28020
```

Quandl

Quandl has become one of the most full-featured sources of free financial data on the web (www.quandl.com) and have been offering non-free premium data as well. R package titled *Quandl*:

```
> library(Quandl)
```

For limited usage (less than 50 calls per day), just install the package and use it. If you plan to download larger amounts of data, you will need to register for an authentication token (free) and register that in your R session.

Quandl

The official Quandl documentation has the following example to download a time series of oil prices from the NYSE and save it to an *xts* object:

```
> mytimeseries <- Quandl("FRED/GDP", type="xts")
> tail(mytimeseries,5)
```

```
> # Quandl.api_key("NfEvd1uysf11ToVR7dbh")
> # mytimeseries <- Quandl("FRED/GDP")
> # head(mytimeseries,5)
```

Rblpapi

Rblpapi uses the Bloomberg API to allow easy importing of Bloomberg data into R. Load the package with the *library()* command and connect to the Bloomberg API (on a terminal computer) with the *blpConnect* command:

```
> library(Rblpapi)
```

Essentially all of the API functionality (whatever you're familiar with from the Bloomberg Excel addin) should be available in Rblpapi - including *bdp*, *bds*, *bdh* functions. Start a connection to the API with

```
> con <- blpConnect()
```

Rblpapi - bdp()

Get current data points (prices, fundamentals) for one or more tickers with *bdp()*:

```
> tickers <- c("RY CN Equity","TD CN Equity","CM CN Equity","BNS CN
> data.items <- c("PX_LAST","DIVIDEND_YIELD","EQY_BETA")
> bdp(tickers,data.items)
```

```
##                 PX_LAST DIVIDEND_YIELD  EQY_BETA
## RY CN Equity    105.81       4.054437 0.8756356
## TD CN Equity     71.52       4.348434 0.9012201
## CM CN Equity    111.62       5.214119 1.0073590
## BNS CN Equity    68.21       5.277819 0.8888354
```

Rblpapi - bdh()

Get historical data with *bdh()*:

```
> # historial prices for the XIU etf since Dec 31, 2019
> my.data <- bdh("XIU CN Equity","PX_LAST",
+                start.date=as.Date("2019-12-31"),
+                end.date=as.Date("2020-11-30"))
> head(my.data)
```

```
##         date PX_LAST
## 1 2019-12-31 25.5600
## 2 2020-01-02 25.6300
## 3 2020-01-03 25.5600
## 4 2020-01-06 25.6508
## 5 2020-01-07 25.7100
## 6 2020-01-08 25.7680
```

Rblpapi - bds()

Download bulk data with *bds()*:

```
> # current constituents of the s&p/tsx 60 index
> tsx.memb <- bds("SPTSX INDEX","INDX_MEMBERS")
> head(tsx.memb)
```

```
##   Member Ticker and Exchange Code
## 1                          ABX CT
## 2                           AC CT
## 3                          ACB CT
## 4                        ACO/X CT
## 5                          AEM CT
## 6                          AGI CT
```

Rblpapi

Other functionality exists, including the *getBars()* and *getTicks* functions, as well as a function for searching available fields (*fieldSearch*):

```
> search.res <- fieldSearch("volatility")
> head(search.res, 2)
```

```
##      Id                   Mnemonic                                Descriptio
## 1 RQ295                IVOL_MID_RT  Implied Volatility Mid (Realtime
## 2 RQ438 VOLATILITY_STRIKE_PX_RT Volatility Strike Price (Realtime
```

```
> dim(search.res)
```

```
## [1] 55  3
```

Note that *fieldSearch()* returns a data frame object.

Rblpapi

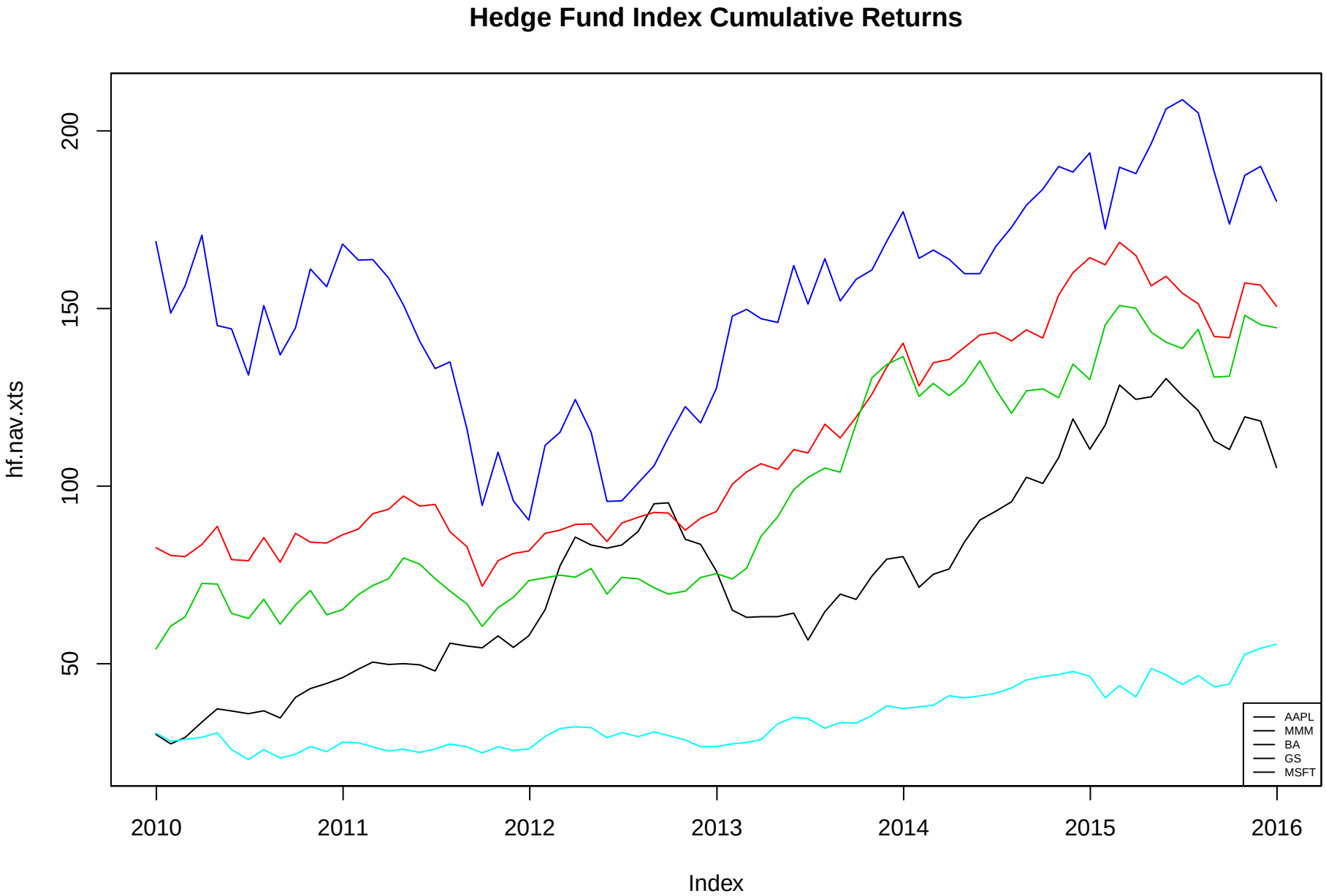
Don't forget to close the connection

```
> blpDisconnect(con)
```

Importing Data from Files

```
> hf.nav <- read.table(file = "data/stock_data.csv",
+                      header=T,sep=",",
+                      stringsAsFactors=FALSE)
> hf.nav.xts <- xts(hf.nav[,-1],
+                   order.by=as.Date(hf.nav[,1],
+                                    format="%Y-%m-%d"))
> plot.zoo(hf.nav.xts,plot.type='single',
+          lty=1,main="Hedge Fund Index Cumulative Returns",
+          col=seq(1:ncol(hf.nav.xts)))
> legend("bottomright", colnames(hf.nav.xts), lty = 1, cex = 0.5)
```

Importing Data from Files



Recap

- R has easy and convenient methods for importing data from both free and non-free online sources
- We can also import files (and interact with databases) to import data that we already own
- Many packages are available to make this process easier - *quantmod* is one of the originals and is very full featured, while *Quandl* allows access to Quandl data sets
- Commercial vendors often provide unofficial support (Bloomberg) or official support (FactSet) for development of R packages for subscribers to access their data

Section 3

R Class, Object and functions

R Objects (frequently used)

In R, all types of data are treated as objects.

- Vectors

Type	Example
Doubles	die <- c(1:6)
Characters	text <- c('R', 'Workshop')
Logicals	logic <- c(TRUE, FALSE, TRUE)

- Matrices

```
> die <- c(1:6)
> mtx <- matrix(die, nrow =2, byrow = TRUE)
```

- Arrarys

```
> ary <- array(mtx, dim=c(2,3,3))
```

R Objects (frequently used)

- Lists

```
> list <- list(die, mtx, ary)
```

- DataFrame is the two-dimensional version of a list. It is a very useful storage structure for data analysis. You can think of a dataframe as R's equivalent to the Excel spreadsheet.

```
> df <- data.frame(face = c("ace","king","queen"),
+                  suit = c("heart","spades","diamonds"),
+                  value = c(1,13,12))
```

R Class

However, objects are not simply collections of data. An object is a data structure having some attributes and methods which act on its attributes.

Class is a blueprint for the object. We can think of class like a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house.

Let's build a simple slot machine, with the *play* class with *symbols* and *score* methods, that you can play by running the class. When you're finished, you'll be able to play it

like this:

```
> play()
```

```
## [1] "0" "0" "B"
```

```
## [1] "You win $0"
```

R Class

The **play** function will need to do two things. First, it will need to randomly generate three symbols; and, second, it wil need to calculate a prize based on those symbols.

R function

1. Define your slot machine symbols and then randomly generate three symbols with the **sample** function.

The wheel, or all the possible outcomes are DD, 7,BBB,BB,B,0, now please take a few minute to think about how to write this *get_symbols* function to randomly select 3 symbols from the wheel.

steps: 1. set up the general function format for *get_symbols* 2. need a selection funcion for the output

R function

1. Define your slot machine symbols and then randomly generate three symbols with the **sample** function.

```
> get_symbols <- function(){
+   wheel <- c("DD","7","BBB","BB","B","0")
+   sample(wheel, size = 3, replace = TRUE,
+          prob = c(0.03, 0.03, 0.06 ,0.1, 0.27, 0.51))
+ }
> get_symbols()
```

```
## [1] "0"  "0"  "BB"
```

R function

2. Write a program that can take the output of get_symbols and calculate the correct prize based on the prize rule, the payout scheme is set as:

Combination	Prize
DD DD DD	100
7 7 7	80
BBB BBB BBB	40
BB BB BB	25
B B B	10
Any combo of bars	5

We call this function **score**, such that *score(c("DD","DD","DD"))* will give you $100.

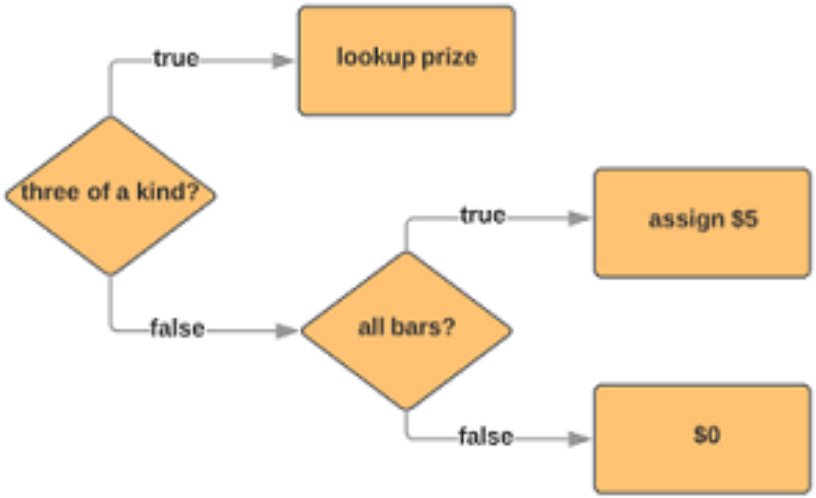
R function

3. Put them together to create the full slot machine, like:

```
> play <- function(){
+   symbols <- get_symbols()
+   print(symbols)
+   score(symbols)
+ }
```

R function

Now let's write the score function, the flow chart below describes the payout scenarios, diamond shape symbolize an *if else* decision.



R function

Please take a few minutes to think about how to write this *score* function.

steps: 1. general function format setup 2. identify case (ifelse statement and test for equality) 3. assign prize for each case

R function

```
> score <- function(symbols){
+   # identify case
+   same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]
+   bars <- symbols %in% c("B","BB","BBB")
+   # get prize
+   if(same){
+     payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25,
+                  "B" = 10, "C" = 10, "0" = 0)
+     prize <- unname(payouts[symbols[1]])
+   } else if(all(bars)){
+     prize <- 5
+   } else {
+     prize <- 0
+   }
+   # return prize
+   prize <- paste0("You Win $", prize)
+   return(prize)
+ }
```

R function

Once the **score** function is defined, the **play** class will work as well:

```
> play <- function(){
+   symbols <- get_symbols()
+   print(symbols)
+   score(symbols)
+ }
> play()
```

```
## [1] "B" "0" "0"
```

```
## [1] "You Win $0"
```