# 02-Intermediate-R-for-Finance

Boni

9/2/2020

## Working with Date Time

```
# Today
x <- Sys.Date()
y <- Sys.time()
x
```

```
## [1] "2020-09-02"
```

```
y
```

```
## [1] "2020-09-02 20:00:00 CDT"
```

```
class(x) # Calendar date class
```

```
## [1] "Date"
```

```
class(y) # POSIX class, with POSIXct and POSIXlt
```

```
## [1] "POSIXct" "POSIXt"
```

```
# Convert string to Date
date <- "2020-09-01" # Use the standard ISO 8601 Standard: year-month-day or year/month/day
date_as_date <- as.Date(date)
class(date_as_date)
```

```
## [1] "Date"
```

```
as.numeric(date_as_date) # Find number of days since January 1, 1970
```

```
## [1] 18506
```

```
# Explicit conversion
as.Date("May 2020 09", format = "%b %Y %d")
```

```
## [1] "2020-05-09"
```

```r
# char_dates
char_dates <- c("1jan17", "2jan17", "3jan17", "4jan17", "5jan17")

# Create dates using as.Date() and the correct format
dates <- as.Date(char_dates, format="%d%b%y")

# Use format() to go from "2017-01-04" -> "Jan 04, 17"
format(dates, format = "%b %d, %y")
```

```
## [1] "Jan 01, 17" "Jan 02, 17" "Jan 03, 17" "Jan 04, 17" "Jan 05, 17"
```

```r
# Use format() to go from "2017-01-04" -> "01,04,2017"
format(dates, format = "%m,%d,%Y")
```

```
## [1] "01,01,2017" "01,02,2017" "01,03,2017" "01,04,2017" "01,05,2017"
```

```r
# Dates
dates <- as.Date(c("2017-01-01", "2017-01-02", "2017-01-03"))

# Create the origin
origin <- as.Date("1970-01-01")

# Use as.numeric() on dates
as.numeric(dates)
```

```
## [1] 17167 17168 17169
```

```r
# Find the difference between dates and origin
dates - origin
```

```
## Time differences in days
## [1] 17167 17168 17169
```

### Extract information from Date Time

```r
# dates
dates <- as.Date(c("2017-01-02", "2017-05-03", "2017-08-04", "2017-10-17"))

# Extract the months
months(dates)
```

```
## [1] "January" "May"     "August"  "October"
```

```r
# Extract the quarters
quarters(dates)
```

```
## [1] "Q1" "Q2" "Q3" "Q4"
```

```r
# dates2
dates2 <- as.Date(c("2017-01-02", "2017-01-03", "2017-01-04", "2017-01-05"))

# Assign the weekdays() of dates2 as the names()
names(dates2) <- weekdays(dates2)

# Print dates2
dates2
```

```
##       Monday      Tuesday    Wednesday     Thursday
## "2017-01-02" "2017-01-03" "2017-01-04" "2017-01-05"
```

## Relational operators

```r
today <- 23.45
yesterday <- 34.23
today > yesterday
```

```
## [1] FALSE
```

```r
one <- 1
one == TRUE
```

```
## [1] TRUE
```

## Working with loops

```r
# Using repeat
i <- 0

repeat {
  i <- i + 1
  print(i)
  if (i == 9) {
    break
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

```r
# while loop
stock_price <- 53.89

while (stock_price < 60) {
  stock_price <- stock_price * runif(1, 0.8, 1.2)
  print(stock_price)
}
```

```
## [1] 61.84263
```

```r
# For loops
x <- 1:5
for (n in x) {
  print(n)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

### Working with function

```r
x <- runif(9, 0.3, 1.4)
x
```

```
## [1] 0.8210138 1.0062643 0.5465855 1.1715344 0.4770098 0.4231284 0.4879805
## [8] 0.6231299 0.9156655
```

```r
diff(x) # X_i - X_(i-t)
```

```
## [1]  0.18525052 -0.45967877  0.62494884 -0.69452460 -0.05388136  0.06485212
## [7]  0.13514941  0.29253555
```

```r
# Arithmetic return
prices <- c(23.4, 23.8, 22.3)

arith_return <- function(x) {
  diff(x) / x[-length(x)]
}

arith_return(prices) # (X_i - X_(i-t)) / X_(i-t)
```

```
## [1]  0.01709402 -0.06302521
```

### Functions and packages related to financial data

```
library(tidyquant)
```

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following object is masked _by_ '.GlobalEnv':
##
##     origin

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

## Loading required package: PerformanceAnalytics

## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

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

##
## Attaching package: 'PerformanceAnalytics'

## The following object is masked _by_ '.GlobalEnv':
##
##     prices

## The following object is masked from 'package:graphics':
##
##     legend

## Loading required package: quantmod

## Loading required package: TTR

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

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

## == Need to Learn tidyquant? ====================================================================
## Business Science offers a 1-hour course - Learning Lab #9: Performance Analysis & Portfolio Optimiza
## </> Learn more at: https://university.business-science.io/p/learning-labs-pro </>
```

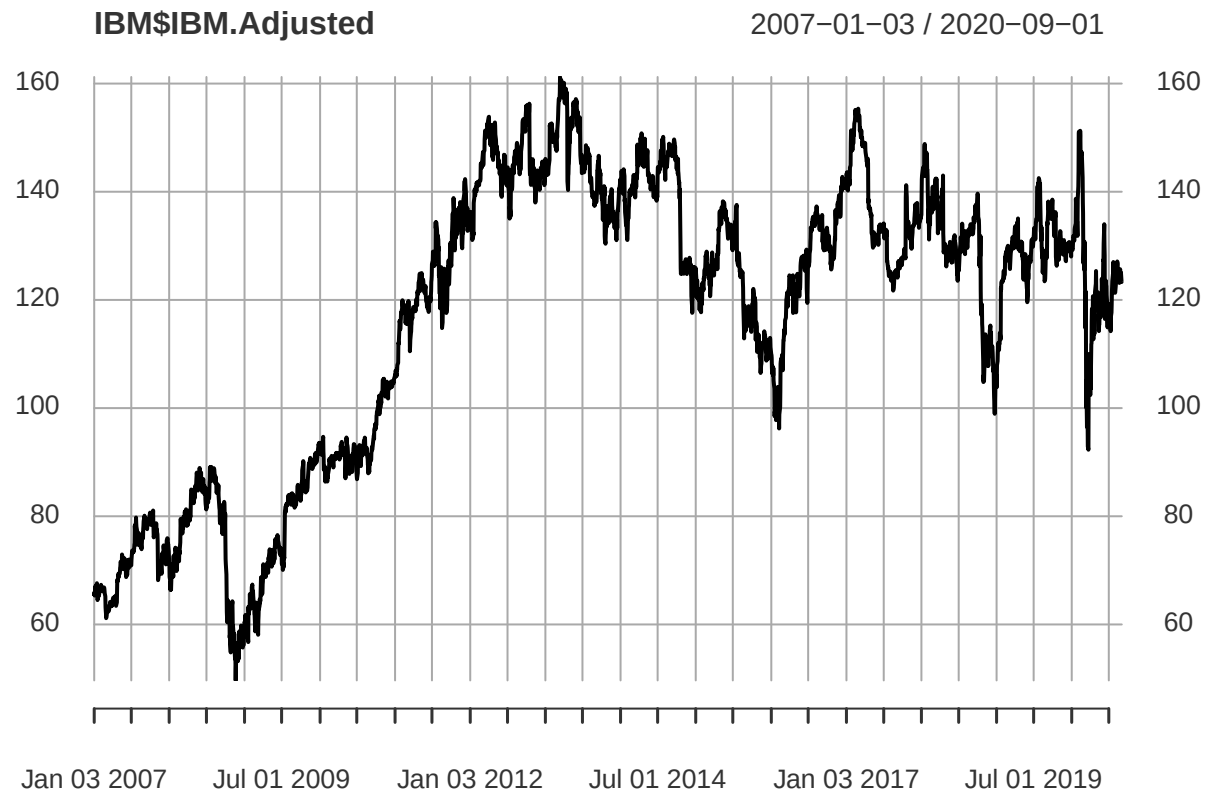
```
library(quantmod)

quantmod::getSymbols(c("IBM", "MSFT"))
```

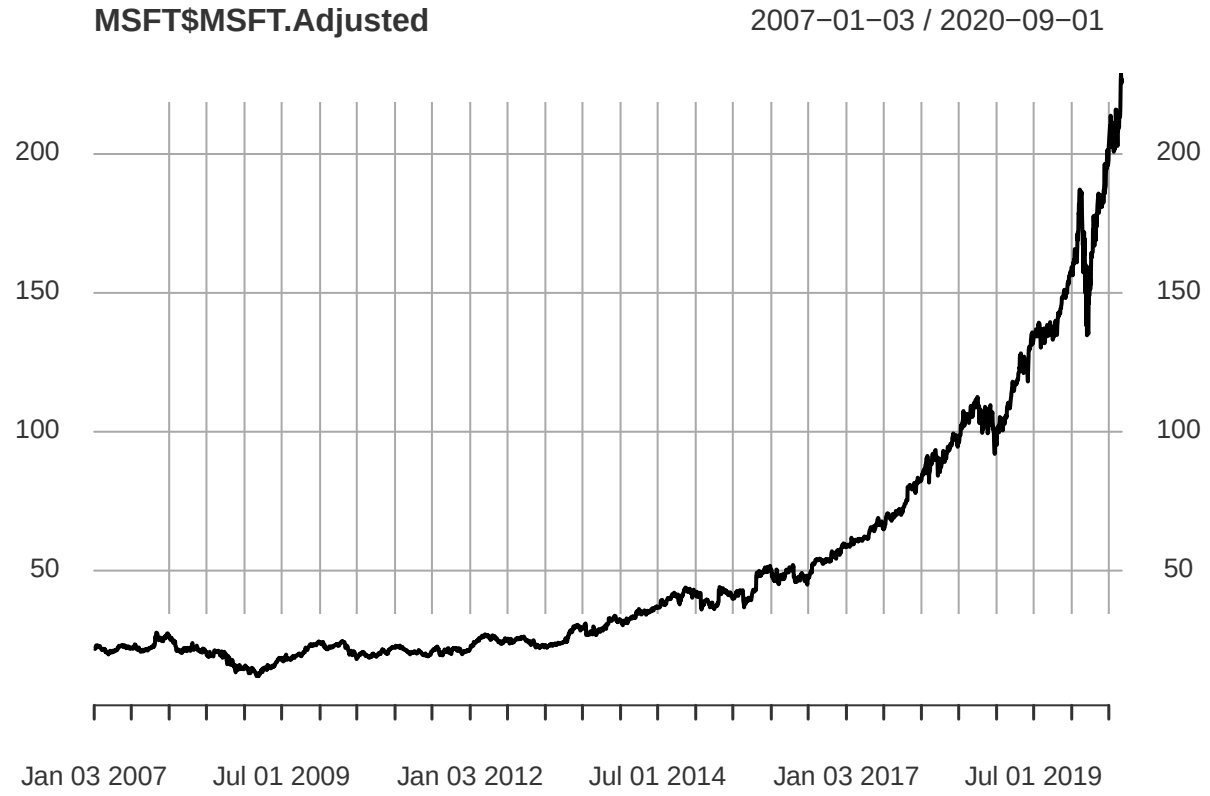
```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.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

```
## [1] "IBM"  "MSFT"
```

```
plot(IBM$IBM.Adjusted)
```




```
plot(MSFT$MSFT.Adjusted)
```

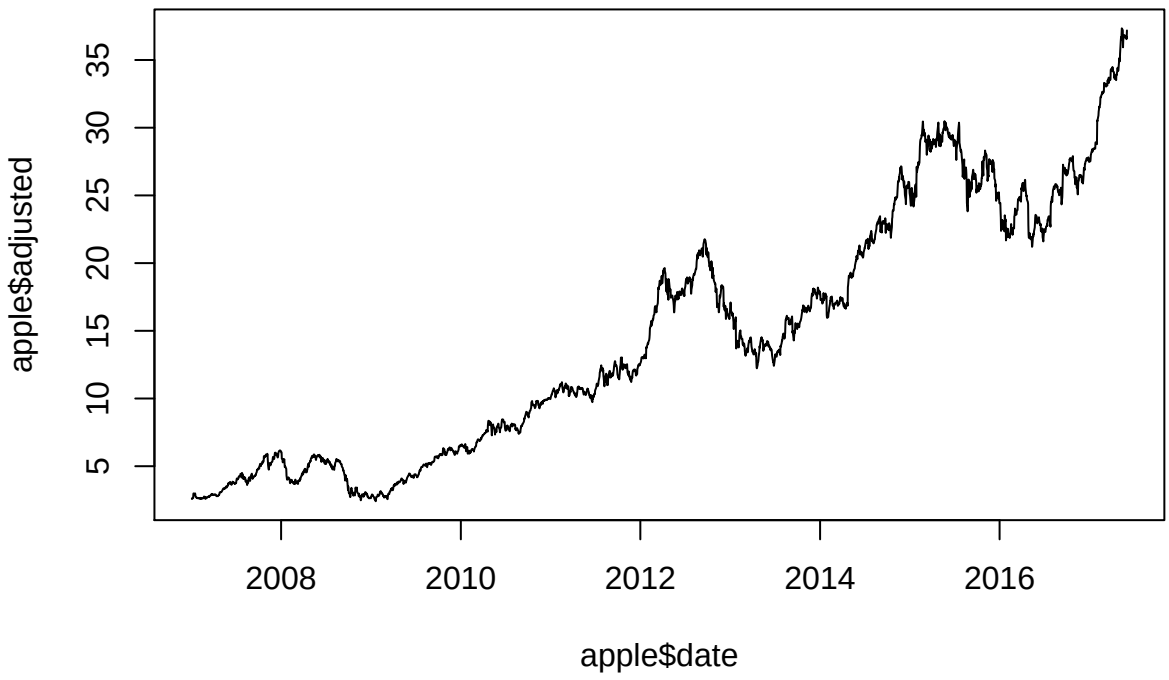



```
# Pull Apple stock data
apple <- tq_get("AAPL", get = "stock.prices",
                from = "2007-01-03", to = "2017-06-05")

# Take a look at what it returned
head(apple)
```

```
## # A tibble: 6 x 8
##   symbol date        open  high   low close     volume adjusted
##   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>      <dbl>    <dbl>
## 1 AAPL   2007-01-03  3.08  3.09  2.92  2.99 1238319600     2.59
## 2 AAPL   2007-01-04  3.00  3.07  2.99  3.06  847260400     2.64
## 3 AAPL   2007-01-05  3.06  3.08  3.01  3.04  834741600     2.62
## 4 AAPL   2007-01-08  3.07  3.09  3.05  3.05  797106800     2.64
## 5 AAPL   2007-01-09  3.09  3.32  3.04  3.31 3349298400     2.86
## 6 AAPL   2007-01-10  3.38  3.49  3.34  3.46 2952880000     2.99
```

```
# Plot the stock price over time
plot(apple$date, apple$adjusted, type = "l")
```

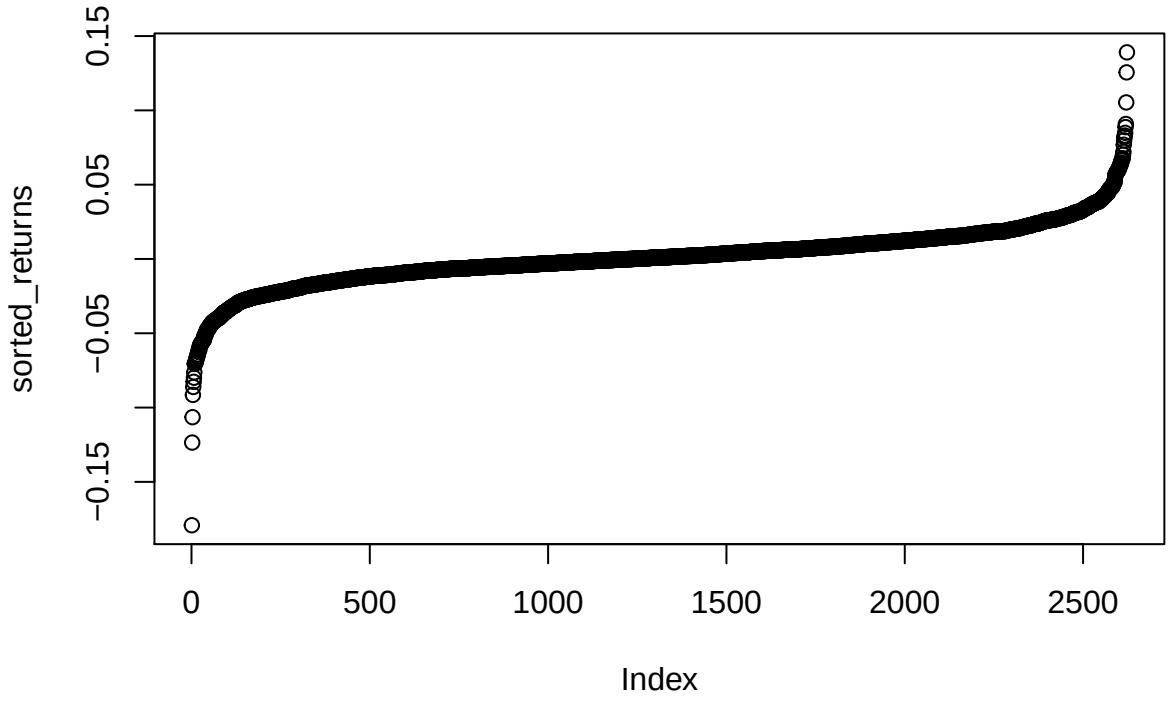



```
# Calculate daily stock returns for the adjusted price
apple <- tq_mutate(data = apple,
                   ohlc_fun = Ad,
                   mutate_fun = dailyReturn)
```

```
## Warning: Argument ohlc_fun is deprecated; please use select instead.
```

```
# Sort the returns from least to greatest
sorted_returns <- sort(apple$daily.returns)

# Plot them
plot(sorted_returns)
```




### Use apply function

There are many apply family:

- apply (Apply function over array margins)
- **lapply** (Apply a function over a list or vector)
- eapply (Apply a function over values in environment)
- mapply (Apply a function to multiple lists or vector arguments)
- rapply (Recursively apply a function to a list)
- tapply (Apply a function over ragged array)
- **sapply** (Simplify the result from lapply)
- **vapply** (Strictly simplify the result from lapply)

```
# lapply in list
list1 <- list("Boni", 31, "University of Chicago", TRUE)
names(list1) <- c("name", "age", "university", "handsome")
lapply(list1, class)
```

```
## $name
## [1] "character"
##
## $age
## [1] "numeric"
##
```

```
## $university
## [1] "character"
##
## $handsome
## [1] "logical"
```

```
# lapply in data frame
a <- c(1,2)
b <- c(3,4)
df <- data.frame(a, b)
lapply(df, sum) # lapply will create summary of each columns
```

```
## $a
## [1] 3
##
## $b
## [1] 7
```

Many times lapply works better than sapply because sapply simplify many things that might be not expected.

```
# sapply in list
list1 <- list("Boni", 31, "University of Chicago", TRUE)
names(list1) <- c("name", "age", "university", "handsome")
sapply(list1, class)
```

```
##        name         age  university    handsome
## "character"   "numeric" "character"   "logical"
```

```
# sapply in data frame
a <- c(1,2)
b <- c(3,4)
df <- data.frame(a, b)
sapply(df, sum) # sapply will create summary of each columns
```

```
## a b
## 3 7
```

```
# Market crash with as.POSIXct()
market_crash2 <- list(dow_jones_drop = 777.68,
                      date = as.POSIXct("2008-09-28"))

# Find the classes with sapply()
sapply(market_crash2, class)
```

```
## $dow_jones_drop
## [1] "numeric"
##
## $date
## [1] "POSIXct" "POSIXt"
```

```r
# Find the classes with vapply()
# vapply(market_crash2, class, FUN.VALUE = character(1))

df <- data.frame(ibm = runif(10, 3, 9), apple = rnorm(10, 3, 1))
vapply(df, FUN = function(x) {c(max = max(x), min = min(x))}, FUN.VALUE = numeric(2))
```

```
##          ibm     apple
## max 8.876193 5.0648881
## min 3.191875 0.6053163
```