04 Importing and Managing Financial Data

Boni

9/3/2020

Use getSymbols()

```r
getSymbols(Symbols = "AAPL", src = "av", api.key = "OST2PMQENLUXD5YT")
```

```
## '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] "AAPL"
```

```r
# alphavantage OST2PMQENLUXD5YT
# src can be alphavantage (av), google, yahoo, fred
first(AAPL, 5)
```

```
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume
## 2020-04-16    287.38    288.20   282.35     286.69    39281300
## 2020-04-17    284.69    286.95   276.86     282.80    53812500
## 2020-04-20    277.95    281.68   276.85     276.93    32503800
## 2020-04-21    276.28    277.25   265.43     268.37    45247900
## 2020-04-22    273.61    277.90   272.20     276.10    29264300
```

```r
getSymbols("GDP", src="FRED")
```

```
## [1] "GDP"
```

```r
first(GDP, 5)
```

```
##                GDP
## 1947-01-01 243.164
## 1947-04-01 245.968
## 1947-07-01 249.585
## 1947-10-01 259.745
## 1948-01-01 265.742
```

Use Quandl()

```
dgs10 <- Quandl::Quandl(code = "FRED/DGS10", type = "xts")
first(dgs10, 5)
```

```
##            [,1]
## 1962-01-02 4.06
## 1962-01-03 4.03
## 1962-01-04 3.99
## 1962-01-05 4.02
## 1962-01-08 4.03
```

Get currency from Oanda

```
# Get available currency in Oanda
head(quantmod::oanda.currencies)
```

```
##     oanda.df.1.length.oanda.df...2....1.
## USD                            US Dollar
## AFN                  Afghanistan Afghani
## ALL                         Albanian Lek
## DZD                       Algerian Dinar
## ADF                       Andorran Franc
## ADP                      Andorran Peseta
```

```
# Create a currency_pair object
currency_pair <- "GBP/CAD"

# Load British Pound to Canadian Dollar exchange rate data
getSymbols(currency_pair, src = "oanda")
```

```
## [1] "GBP/CAD"
```

```
# Examine object using str()
str(GBPCAD)
```

```
## An ’xts’ object on 2020-03-11/2020-09-05 containing:
##   Data: num [1:179, 1] 1.77 1.76 1.73 1.69 1.7 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "GBP.CAD"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "oanda"
##  $ updated: POSIXct[1:1], format: "2020-09-06 11:24:56"
```

```r
# Try to load data from 190 days ago
getSymbols(currency_pair, from = Sys.Date() - 190, to = Sys.Date(), src = "oanda")
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Oanda only
## provides historical data for the past 180 days. Symbol: GBP/CAD
```

```
## [1] "GBP/CAD"
```

Unemployment Rate from FRED

```r
# Create a series_name object
series_name <- "UNRATE"

# Load the data using getSymbols
getSymbols(series_name, src = "FRED", type = "xts")
```

```
## [1] "UNRATE"
```

```r
head(UNRATE)
```

```
##            UNRATE
## 1948-01-01    3.4
## 1948-02-01    3.8
## 1948-03-01    4.0
## 1948-04-01    3.9
## 1948-05-01    3.5
## 1948-06-01    3.6
```

```r
# Create a quandl_code object
quandl_code <- "FRED/UNRATE"

# Load the data using Quandl
unemploy_rate <- Quandl(quandl_code, type = "xts")
head(unemploy_rate)
```

```
##          [,1]
## Jan 1948  3.4
## Feb 1948  3.8
## Mar 1948  4.0
## Apr 1948  3.9
## May 1948  3.5
## Jun 1948  3.6
```

Extract OHLC

```
# Important method to extract OHLC information: Op(), Hi(), Lo(), Cl(), Vo(), Ad(), and OHLC()
# Learn more: help("OHLC.Transformations")

# Get Symbols
getSymbols("TSLA", src = "yahoo", type = "xts")
```

```
## [1] "TSLA"
```

```
# Extract the close column
tsla_close <- Cl(TSLA)

# Look at the head of dc_close
head(tsla_close)
```

```
##            TSLA.Close
## 2010-06-29      4.778
## 2010-06-30      4.766
## 2010-07-01      4.392
## 2010-07-02      3.840
## 2010-07-06      3.222
## 2010-07-07      3.160
```

```
# Extract the volume column
tsla_volume <- Vo(TSLA)

# Look at the head of dc_volume
head(tsla_volume)
```

```
##            TSLA.Volume
## 2010-06-29    93831500
## 2010-06-30    85935500
## 2010-07-01    41094000
## 2010-07-02    25699000
## 2010-07-06    34334500
## 2010-07-07    34608500
```

```
# Extract the high, low, and close columns
tsla_hlc <- HLC(TSLA)

# Look at the head of dc_hlc
head(tsla_hlc)
```

```
##            TSLA.High TSLA.Low TSLA.Close
## 2010-06-29     5.000    3.508      4.778
## 2010-06-30     6.084    4.660      4.766
## 2010-07-01     5.184    4.054      4.392
## 2010-07-02     4.620    3.742      3.840
## 2010-07-06     4.000    3.166      3.222
## 2010-07-07     3.326    2.996      3.160
```

```r
# Extract the open, high, low, close, and volume columns
tsla_ohlcv <- OHLCV(TSLA)

# Look at the head of dc_ohlcv
head(tsla_ohlcv)
```

```
##            TSLA.Open TSLA.High TSLA.Low TSLA.Close TSLA.Volume
## 2010-06-29     3.800     5.000    3.508      4.778    93831500
## 2010-06-30     5.158     6.084    4.660      4.766    85935500
## 2010-07-01     5.000     5.184    4.054      4.392    41094000
## 2010-07-02     4.600     4.620    3.742      3.840    25699000
## 2010-07-06     4.000     4.000    3.166      3.222    34334500
## 2010-07-07     3.280     3.326    2.996      3.160    34608500
```

Extract the Close column from many instruments

```r
Cl <- function (x)
{
    if (has.Cl(x))
        return(x[, grep("Close", colnames(x), ignore.case = TRUE)])
    stop("subscript out of bounds: no column name containing \"Close\"")
}

# Symbols
symbols <- c("AAPL", "MSFT", "IBM")

# Create new environment
data_env <- new.env()

# Load symbols into data_env
getSymbols(symbols, env = data_env)
```

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

```r
# Extract the close column from each object and combine into one xts object
close_data <- do.call(merge, eapply(data_env, Cl))

# View the head of close_data
head(close_data)
```

```
##            AAPL.Close IBM.Close MSFT.Close
## 2007-01-03   2.992857     97.27      29.86
## 2007-01-04   3.059286     98.31      29.81
## 2007-01-05   3.037500     97.42      29.64
## 2007-01-08   3.052500     98.90      29.93
## 2007-01-09   3.306072    100.07      29.96
## 2007-01-10   3.464286     98.89      29.66
```

Change default getSymbols()

```
# Look at getSymbols.yahoo arguments
args(getSymbols.yahoo)
```

```
## function (Symbols, env, return.class = "xts", index.class = "Date",
##     from = "2007-01-01", to = Sys.Date(), ..., periodicity = "daily",
##     curl.options = list())
## NULL
```

```
# Set default ’from’ value for getSymbols.yahoo
setDefaults(getSymbols.yahoo, from = "2000-01-01")

# Confirm defaults were set correctly
getDefaults(name = "getSymbols.yahoo")
```

```
## $from
## [1] "’2000-01-01’"
```

Handling instrument symbols that clash or are not valid R names

```
# get BRK-A
getSymbols("BRK-A")
```

```
## [1] "BRK-A"
```

```
# Assign the result to BRK.A
BRK.A <- get("BRK-A")

head(BRK.A)
```

```
##            BRK-A.Open BRK-A.High BRK-A.Low BRK-A.Close BRK-A.Volume
## 2000-01-03      56100      56100     53800       54800        36000
## 2000-01-04      53700      53800     52000       52000        44000
## 2000-01-05      51700      54700     51700       53200        51000
## 2000-01-06      53300      55000     53100       55000        57000
## 2000-01-07      55600      56500     55200       56500        67000
## 2000-01-10      57300      58000     56000       56000        31000
##            BRK-A.Adjusted
## 2000-01-03          54800
## 2000-01-04          52000
## 2000-01-05          53200
## 2000-01-06          55000
## 2000-01-07          56500
## 2000-01-10          56000
```

Another way to handle invalid names (1)

```
# Create BRK.A object
BRK.A <- getSymbols("BRK-A", auto.assign = FALSE)

# Create col_names object with the column names of BRK.A
col_names <- colnames(BRK.A)

# Set BRK.A column names to syntactically valid names
colnames(BRK.A) <- make.names(col_names)

head(BRK.A)
```

```
##            BRK.A.Open BRK.A.High BRK.A.Low BRK.A.Close BRK.A.Volume
## 2000-01-03      56100      56100     53800       54800        36000
## 2000-01-04      53700      53800     52000       52000        44000
## 2000-01-05      51700      54700     51700       53200        51000
## 2000-01-06      53300      55000     53100       55000        57000
## 2000-01-07      55600      56500     55200       56500        67000
## 2000-01-10      57300      58000     56000       56000        31000
##            BRK.A.Adjusted
## 2000-01-03          54800
## 2000-01-04          52000
## 2000-01-05          53200
## 2000-01-06          55000
## 2000-01-07          56500
## 2000-01-10          56000
```

Another day to handle invalid names (2)

```
# Set name for BRK-A to BRK.A
setSymbolLookup(BRK.A = list(name = "BRK-A"))

# Set name for T (AT&T) to ATT
setSymbolLookup(ATT = list(name = "T"))

# Load BRK.A and ATT data
getSymbols(c("BRK.A", "ATT"))
```

```
## [1] "BRK.A" "ATT"
```

Aggregate daily data and merge with monthly data

Sometimes two series have the same periodicy, but use different conventions to represent a timestamp. For example, monthly series may be timestamped with the first or last date of the month. The different timestamp convention can cause many NA when series are merged. The yearmon class from the zoo package helps solve this problem.

```
getSymbols(c("FEDFUNDS", "DFF"), src = "FRED")
```

```
## [1] "FEDFUNDS" "DFF"
```

```
# Aggregate DFF to monthly
monthly_fedfunds <- apply.monthly(DFF, mean)

# Convert index to yearmon
index(monthly_fedfunds) <- as.yearmon(index(monthly_fedfunds))

# Merge FEDFUNDS with the monthly aggregate
merged_fedfunds <- merge(FEDFUNDS, monthly_fedfunds)

# Look at the first few rows of the merged object
head(merged_fedfunds)
```

```
##            FEDFUNDS       DFF
## 1954-07-01     0.80 0.7993548
## 1954-08-01     1.22 1.2206452
## 1954-09-01     1.06 1.0666667
## 1954-10-01     0.85 0.8487097
## 1954-11-01     0.83 0.8336667
## 1954-12-01     1.28 1.2777419
```

Align series to first and last day of month

Sometimes you may not be able to use convenience classes like yearmon to represent timestamps. This exercise will teach you how to manually align merged data to the timestamp representation you prefer.

First you merge the lower-frequency data with the aggregate data, then use na.locf() to fill the NA forward (or backward, using fromLast = TRUE). Then you can subset the result using the index of the object with the representation you prefer.

```
# Fill NA forward
merged_fedfunds_locf <- na.locf(merged_fedfunds)

# Extract index values containing last day of month
aligned_last_day <- merged_fedfunds_locf[index(monthly_fedfunds)]

# Fill NA backward
merged_fedfunds_locb <- na.locf(merged_fedfunds, fromLast = TRUE)

# Extract index values containing first day of month
aligned_first_day <- merged_fedfunds_locb[index(FEDFUNDS)]
```

Aggregate to weekly, ending on Wednesdays

In this exercise, you will learn a general aggregation technique to aggregate daily data to weekly, but with weeks ending on Wednesdays. This is often done in stock market research to avoid intra-week seasonality.

You can supply your own end points to period.apply() (versus using endpoints()). Recall endpoints() returns locations of the last observation in each period specified by the on argument. The first and last elements of the result are always zero and the total number of observations, respectively. The end points you pass to period.apply() must follow this convention.

```r
# Extract index weekdays
index_weekdays <- .indexwday(DFF)

# Find locations of Wednesdays
wednesdays <- which(index_weekdays == 3)

# Create custom end points
end_points <- c(0, wednesdays, nrow(DFF))

# Calculate weekly mean using custom end points
weekly_mean <- period.apply(DFF, end_points, mean)
```

Merge intraday data with different TZ

```r
# Create merged object with a Europe/London timezone
# tz_london <- merge(london, chicago)

# Look at tz_london structure
# str(tz_london)

# Create merged object with a America/Chicago timezone
# tz_chicago <- merge(chicago, london)

# Look at tz_chicago structure
# str(tz_chicago)
```

Download split and dividend data

In the previous exercise, you used adjustOHLC() to adjust raw historical OHLC prices for splits and dividends, but it only works for OHLC data. It will not work if you only have close prices, and it does not return any of the split or dividend data it uses.

You need the dates and values for each split and dividend to adjust a non-OHLC price series, or if you simply want to analyze the raw split and dividend data.

You can download the split and dividend data from Yahoo Finance using the quantmod functions getSplits() and getDividends(), respectively. The historical dividend data from Yahoo Finance is adjusted for splits. If you want to download unadjusted dividend data, you need to set split.adjust = FALSE in your call to getDividends().

```r
# Download AAPL split data
splits <- getSplits("AAPL")

# Download AAPL dividend data
dividends <- getDividends("AAPL")

# Look at the first few rows of dividends
head(dividends)
```

```
##                AAPL.div
## 1987-05-11 2.410714e-06
## 1987-08-10 4.821429e-06
## 1987-11-17 6.339286e-06
## 1988-02-12 6.339286e-06
## 1988-05-16 6.339286e-06
## 1988-08-15 6.339286e-06
```

```
# Download unadjusted AAPL dividend data
raw_dividends <- getDividends("AAPL", split.adjust = FALSE)

# Look at the first few rows of raw_dividends
head(raw_dividends)
```

```
##            AAPL.div
## 1987-05-11  0.00054
## 1987-08-10  0.00054
## 1987-11-17  0.00071
## 1988-02-12  0.00071
## 1988-05-16  0.00071
## 1988-08-15  0.00071
```

Adjust univariate data for splits and dividends

If you only have close prices, you can adjust them with adjRatios(). It has 3 arguments: splits, dividends, and close. It returns an xts object with split and dividend adjustment ratios in columns "Split" and "Div", respectively.

You need to provide split data via the splits argument to calculate the split ratio. To calculate the dividend ratio, you need to provide raw dividends and raw prices via the dividends and close arguments, respectively.

Once you have the split and dividend adjustment ratios, you calculate the adjusted price multiplying the unadjusted price by both the split and dividend adjustment ratios.

```
getSymbols("AAPL")
```

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

```
# Calculate split and dividend adjustment ratios
ratios <- adjRatios(splits = splits, dividends = raw_dividends, close = Cl(AAPL))

# Calculate adjusted close for AAPL
aapl_adjusted <- Cl(AAPL) * ratios[, "Split"] * ratios[, "Div"]

# Look at first few rows of Yahoo adjusted close
head(Ad(AAPL))
```

```
##            AAPL.Adjusted
## 2000-01-03      0.863657
## 2000-01-04      0.790842
## 2000-01-05      0.802415
## 2000-01-06      0.732975
## 2000-01-07      0.767695
## 2000-01-10      0.754193
```

```r
# Look at first few rows of aapl_adjusted
head(aapl_adjusted)
```

```
##            AAPL.Close
## 2000-01-03 0.007711222
## 2000-01-04 0.007061088
## 2000-01-05 0.007164415
## 2000-01-06 0.006544418
## 2000-01-07 0.006854420
## 2000-01-10 0.006733865
```