How to Get Data — An Introduction into **quantmod**

November 29, 2016

1 The S&P 500 index

This vignette gives a brief introduction to obtaining data from the web by using the R package **quantmod**. As example data, the time series of the S&P 500 index is used. This data is also used in Carmona, page 5 ff.

First, we load the **quantmod** package:

```
R> require("quantmod")
```

quantmod provides a very suitable function for downloading financial date from the web. This function is called `getSymbols`. The first argument of this function is a character vector specifying the names of the symbols to be downloaded and the second one specifies the environment where the object is created. The help page of this function (`?getSymbols`) provides more information. By default, objects are created in the workspace. Here, we use a separate environment which we call `sp500` to store the downloaded data. We first create the environment:

```
R> sp500 <- new.env()
```

We can then download the S&P 500 time series (symbol: `^GSPC`) from 1960-01-04 to 2009-01-01 from *yahoo finance* via:

```
R> getSymbols("^GSPC", env = sp500, src = "yahoo",
+             from = as.Date("1960-01-04"), to = as.Date("2009-01-01"))
```

```
[1] "GSPC"
```

The package **quantmod** works with a variety of sources. Current `src` methods available are: *yahoo, google, MySQL, FRED, csv, RData,* and *oanda*. For example, FRED (Federal Reserve Economic Data), is a database of 20,070 U.S. economic time series (see `http://research.stlouisfed.org/fred2/`).

There are several possibilities, to load the variable `GSPC` from the environment `sp500` to a variable in the global environment (also known as the workspace), e.g., via

```
R> GSPC <- sp500$GSPC
R> GSPC1 <- get("GSPC", envir = sp500)
R> GSPC2 <- with(sp500, GSPC)
```

The object `GSPC1` and `GSPC2` are identical to `GSPC` so we can remove them from the workspace with:

```
R> rm(GSPC1)
R> rm(GSPC2)
```

The function `head` shows the first six rows of the data.

```
R> head(GSPC)
```

```
           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
1960-01-04     59.91     59.91    59.91      59.91     3990000         59.91
1960-01-05     60.39     60.39    60.39      60.39     3710000         60.39
1960-01-06     60.13     60.13    60.13      60.13     3730000         60.13
1960-01-07     59.69     59.69    59.69      59.69     3310000         59.69
1960-01-08     59.50     59.50    59.50      59.50     3290000         59.50
1960-01-11     58.77     58.77    58.77      58.77     3470000         58.77
```

This is on OHLC time series with at least the (daily) Open, Hi, Lo and Close prices for the symbol; here, it also contains the traded volume and the closing price adjusted for splits and dividends.

The data object is an "extensible time series" (xts) object:

```
R> class(GSPC)
```

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

Here, it is a multivariate (irregular) time series with 12334 daily observations on 6 variables:

```
R> dim(GSPC)
```

```
[1] 12334     6
```

Such xts objects allow for conveniently selecting single time series using `$`

```
R> head(GSPC$GSPC.Volume)
```

```
           GSPC.Volume
1960-01-04     3990000
1960-01-05     3710000
1960-01-06     3730000
1960-01-07     3310000
1960-01-08     3290000
1960-01-11     3470000
```

as well as very conviently selecting observations according to their time stamp by using a character "row" index in the ISO 8601 date/time format 'CCYY-MM-DD HH:MM:SS', where more granular elements may be left out in which case all observations with time stamp "matching" the given one will be used. E.g., to get all observations in March 1970:

```
R> GSPC["1970-03"]
```

```
           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
1970-03-02     89.50     90.80    88.92      89.71    12270000         89.71
1970-03-03     89.71     90.67    88.96      90.23    11700000         90.23
1970-03-04     90.23     91.05    89.32      90.04    11850000         90.04
1970-03-05     90.04     90.99    89.38      90.00    11370000         90.00
1970-03-06     90.00     90.36    88.84      89.44    10980000         89.44
1970-03-09     89.43     89.43    87.94      88.51     9760000         88.51
1970-03-10     88.51     89.41    87.89      88.75     9450000         88.75
1970-03-11     88.75     89.58    88.11      88.69     9180000         88.69
1970-03-12     88.69     89.09    87.68      88.33     9140000         88.33
```

```
1970-03-13     88.33     89.43    87.29      87.86     9560000         87.86
1970-03-16     87.86     87.97    86.39      86.91     8910000         86.91
1970-03-17     86.91     87.86    86.36      87.29     9090000         87.29
1970-03-18     87.29     88.28    86.93      87.54     9790000         87.54
1970-03-19     87.54     88.20    86.88      87.42     8930000         87.42
1970-03-20     87.42     87.77    86.43      87.06     7910000         87.06
1970-03-23     87.06     87.64    86.19      86.99     7330000         86.99
1970-03-24     86.99     88.43    86.90      87.98     8840000         87.98
1970-03-25     88.11     91.07    88.11      89.77    17500000         89.77
1970-03-26     89.77     90.65    89.18      89.92    11350000         89.92
1970-03-30     89.92     90.41    88.91      89.63     9600000         89.63
1970-03-31     89.63     90.17    88.85      89.63     8370000         89.63
```

It is also possible to specify a range of timestamps using '`/`' as the range separator, where both endpoints are optional: e.g.,

```
R> GSPC["/1960-01-06"]

           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
1960-01-04     59.91     59.91    59.91      59.91     3990000         59.91
1960-01-05     60.39     60.39    60.39      60.39     3710000         60.39
1960-01-06     60.13     60.13    60.13      60.13     3730000         60.13
```

gives all observations up to Epiphany (Jan 6) in 1960, and

```
R> GSPC["2008-12-25/"]

           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
2008-12-26    869.51    873.74   866.52     872.80  1880050000        872.80
2008-12-29    872.37    873.70   857.07     869.42  3323430000        869.42
2008-12-30    870.58    891.12   870.58     890.64  3627800000        890.64
2008-12-31    890.59    910.32   889.67     903.25  4172940000        903.25
```

gives all observations from Christmas (Dec 25) in 2008 onwards.

For OHLC time series objects, **quantmod** also provides convenience (column) extractors and transformers, such as `Cl()` for extracting the closing price, `OpCl()` for the transformation from opening to closing prices, and `ClCl()` for the changes in closing prices:

```
R> head(Cl(GSPC))

           GSPC.Close
1960-01-04      59.91
1960-01-05      60.39
1960-01-06      60.13
1960-01-07      59.69
1960-01-08      59.50
1960-01-11      58.77

R> head(OpCl(GSPC))

           OpCl.GSPC
1960-01-04         0
```

```
1960-01-05         0
1960-01-06         0
1960-01-07         0
1960-01-08         0
1960-01-11         0
```

```
R> head(ClCl(GSPC))
```

```
              ClCl.GSPC
1960-01-04           NA
1960-01-05  0.008012001
1960-01-06 -0.004305316
1960-01-07 -0.007317512
1960-01-08 -0.003183096
1960-01-11 -0.012268908
```

If we are intersted in the daily values of the weekly last-traded-day, we aggregate it by using an appropriate function from the "zoo Quick-Reference" (Shah et al., 2005). The "zoo Quick-Reference" can be found in the web, `cran.r-project.org/web/packages/zoo/vignettes/zoo-quickref.pdf`, and it is strongly recommended to have a look at this vignette since it gives a very good overview of the **zoo** package. Their convenience function `nextfri` computes for each "Date" the next Friday.

```
R> nextfri <- function(x) 7 * ceiling(as.numeric(x - 5 + 4)/7) + as.Date(5 - 4)
```

We get the aggregated data then via

```
R> SP.we <- aggregate(GSPC, nextfri, tail, 1)
```

The function `aggregate` splits the data into subsets — here according to the function `nextfri` — and computes statistics for each, i.e., takes the last value, which is done by `tail`.

This works because the data object is also a "Z's ordered observations" (zoo) object which knows to apply `nextfri()` to the index (timestamps). However, this loses the `xts` class: if this is not desired, one can use

```
R> SP.we <- xts(aggregate(GSPC, nextfri, tail, 1))
```

instead.

(Alternatively, package **quantmod** provides `apply.weekly()`, which uses a slightly different endpoint strategy.)

We can now extract the closing prices for the last trading day in every week:

```
R> SPC.we <- Cl(SP.we)
```

and create a plot of this time series via

```
R> plot(SPC.we)
```

(see Figure 1).

Finally, we can create log-returns "by hand" and visualize these as well

```
R> lr <- diff(log(SPC.we))
R> plot(lr)
```

(see Figure 2).

Alternatively, we could use `periodReturn()` (and relatives, specifically `weeklyReturn()`) from **quantmod** with `type = "log"`. Again, this will give slightly different values.

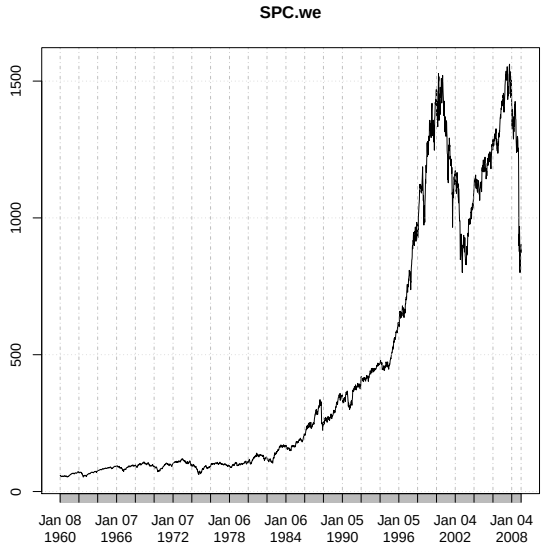

Figure 1: Plot of the weekly S&P 500 index closing values from 1960-01-04 to 2009-01-01.

2 Investigating the NASDAQ-100 index

In this example we want analyze an American stock exchange, the National Association of Securities Dealers Automated Quotations, better known as NASDAQ (see `http://www.nasdaq.com/` for more information). It is the largest electronic screen-based equity securities trading market in the United States.

Accessing `http://www.nasdaq.com/quotes/nasdaq-100-stocks.aspx?render=download` allows to download a .csv file including company symbol and name (note that there are more than 100 entries, as some companies appear with 2 symbols):

```
R> nasdaq100 <-
+     read.csv("nasdaq100list.csv",
+              stringsAsFactors = FALSE, strip.white = TRUE)
R> dim(nasdaq100)

[1] 105   8
```

This has the company symbols and names in variables `Symbol` and `Name`, respectively:

```
R> names(nasdaq100)

[1] "Symbol"           "Name"             "lastsale"         "netchange"
[5] "pctchange"        "share_volume"     "Nasdaq100_points" "X"

R> nasdaq100$Name[duplicated(nasdaq100$Name)]

[1] "Alphabet Inc."                   "Discovery Communications Inc."
[3] "Liberty Global plc"              "Liberty Interactive Corporation"
[5] "Twenty-First Century Fox Inc."
```

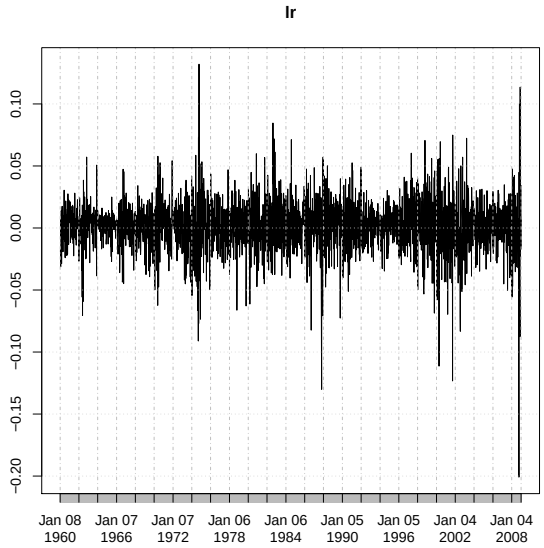

Figure 2: Plot of the weekly S&P 500 index log-returns values from 1960-01-04 to 2009-01-01.

As before we create a new environment for our NASDAQ data and use the function `getSymbols` of the **quantmod** package to download the NASDAQ-100 time series from 2000-01-01 to today.

By using the command `tryCatch` we handle unusual conditions, including errors and warnings. In this case, if the data from a company are not available from yahoo finance, the message `"Symbol ... not downloadable!"` is given. (For simplicity, we only download the symbols starting with 'A'.)

```
R> nasdaq <- new.env()
R> for(i in nasdaq100$Symbol[startsWith(nasdaq100$Symbol, "A")]) {
+     cat("Downloading time series for symbol '", i, "' ...\n",
+         sep = "")
+     status <- tryCatch(getSymbols(i, env = nasdaq, src = "yahoo",
+                                   from = as.Date("2000-01-01")),
+                        error = identity)
+     if(inherits(status, "error"))
+         cat("Symbol '", i, "' not downloadable!\n", sep = "")
+ }

Downloading time series for symbol 'ATVI' ...
Downloading time series for symbol 'ADBE' ...
Downloading time series for symbol 'AKAM' ...
Downloading time series for symbol 'ALXN' ...
Downloading time series for symbol 'AMZN' ...
Downloading time series for symbol 'AAL' ...
Downloading time series for symbol 'AMGN' ...
Downloading time series for symbol 'ADI' ...
Downloading time series for symbol 'AAPL' ...
Downloading time series for symbol 'AMAT' ...
Downloading time series for symbol 'ADSK' ...
```

```
Downloading time series for symbol 'ADP' ...
Downloading time series for symbol 'AVGO' ...
```

E.g., the first values of the Apple time series are

```
R> with(nasdaq, head(AAPL))
```

```
           AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
2000-01-03   104.875  112.5000 101.6875   111.9375   133949200      3.641362
2000-01-04   108.250  110.6250 101.1875   102.5000   128094400      3.334358
2000-01-05   103.750  110.5625 103.0000   104.0000   194580400      3.383153
2000-01-06   106.125  107.0000  95.0000    95.0000   191993200      3.090380
2000-01-07    96.500  101.0000  95.5000    99.5000   115183600      3.236767
2000-01-10   102.000  102.2500  94.7500    97.7500   126266000      3.179838
```

Further, the command `chartSeries` of the package **quantmod** provides the full financial charting abilities to R and allows for an interaction within the charts. E.g., using

```
R> chartSeries(nasdaq$AAPL)
```

gives a chart of the Apple values (see Figure 3) and e.g., with the command `with(nasdaq,addOBV(AAPL))` the On-Balance volume can be visualized in the plot. See the manual of the **quantmod** package (Ryan, 2016) for the whole list of available plot and visualization functions.

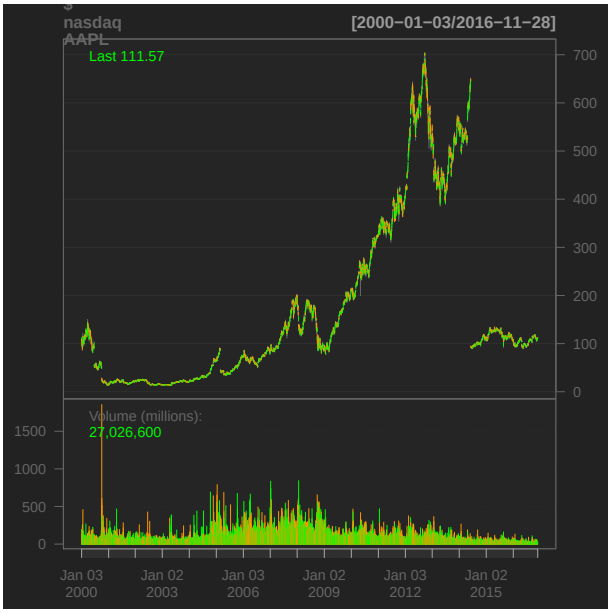

Figure 3: `barChart` of Apple.

E.g., Bollinger bands consist of a center line and two price channels (bands) above and below it. The center line is an exponential moving average; the price channels are the standard deviations of the stock being studied. The bands will expand and contract as the price action of an issue becomes volatile (expansion) or becomes bound into a tight trading pattern (contraction).

We can add the Bollinger Bands to a plot by using the command: `addBBands(n = 20, sd = 2, ma = "SMA", draw = "bands", on = -1)`, where `n` denotes the number of moving average periods, `sd` the number of standard deviations and `ma` the used moving average process.

Have a look at the **quantmod** homepage for further examples and try to reproduce them, `http://www.quantmod.com/examples/intro/`.