



COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

R Programming for Quantitative Finance

Guy Yollin

Applied Mathematics
University of Washington

Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with `quantmod`
- 5 Financial data access with other packages
- 6 Technical indicators and TTR
- 7 Intra-day data example

Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with quantmod
- 5 Financial data access with other packages
- 6 Technical indicators and TTR
- 7 Intra-day data example

Lecture references

- quantmod package
 - Jeffrey Ryan's quantmod website:
<http://www.quantmod.com/>
 - quantmod project page on R-forge:
<http://r-forge.r-project.org/projects/quantmod>
- TTR package
 - Joshua Ulrich's Foss Trading blog:
<http://blog.fosstrading.com/>
 - TTR project page on R-forge:
<http://r-forge.r-project.org/projects/ttr/>
- xts package
 - xts project page on R-forge:
<http://r-forge.r-project.org/projects/xts/>

Quantitative analysis package hierarchy

Application Area	R Package
Performance metrics and graphs	PerformanceAnalytics - Tools for performance and risk analysis
Portfolio optimization and quantitative trading strategies	PortfolioAnalytics - Portfolio analysis and optimization
	quantstrat – Rules-based trading system development
	blotter – Trading system accounting infrastructure
Data access and financial charting	quantmod - Quantitative financial modeling framework
	TTR - Technical trading rules
Time series objects	xts - Extensible time series
	zoo - Ordered observation

The zoo package

The zoo package provides an infrastructure for regularly-spaced and irregularly-space time series

Key functions:

<code>zoo</code>	create a zoo time series object
<code>merge</code>	merges time series (automatically handles of time alignment)
<code>aggregate</code>	create coarser resolution time series with summary statistics
<code>rollapply</code>	calculate rolling window statistics
<code>read.zoo</code>	read a text file into a zoo time series object

Authors:

- Achim Zeileis
- Gabor Grothendieck

The xts package

The `xts` package extends the `zoo` time series class with fine-grained time indexes, interoperability with other R time series classes, and user defined attributes

Key functions:

<code>xts</code>	create an xts time series object
<code>align.time</code>	align time series to a coarser resolution
<code>to.period</code>	convert time series data to an OHLC series
<code>[.xts</code>	subset time series

Authors:

- Jeffrey Ryan
- Josh Ulrich

- R-forge is a hosting platform for R package development which includes SVN, daily build and check, mailing lists, bug tracking, message boards etc.
- Almost 2000 R packages are hosted on R-forge including all of the trading system development packages mentioned earlier
- It is common for new packages to be developed on R-forge and for mature packages to be maintained on R-forge even after being hosted on CRAN

Installing the latest quantitative analysis packages

```
#  
# install these packages from CRAN (or r-forge)  
#  
install.packages("xts", dependencies=TRUE)  
install.packages("PerformanceAnalytics", dependencies=TRUE)  
#  
# Install these package from r-forge  
#  
install.packages("quantmod", repos = "http://R-Forge.R-project.org")  
install.packages("TTR", repos = "http://R-Forge.R-project.org")
```

- R-Forge packages can be installed by setting the repos argument to `http://R-Forge.R-project.org`

Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with quantmod
- 5 Financial data access with other packages
- 6 Technical indicators and TTR
- 7 Intra-day data example

The quantmod package

The quantmod package for R is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models.

Key functions:

`getSymbols` load or download price data

- Yahoo Finance / Google Finance
- FRED
- Oanda
- csv, RData
- MySQL, SQLite

`chartSeries` charting tool to create standard financial charts

Author:

- Jeffrey Ryan

The `getSymbols` function

The `getSymbols` function loads (downloads) historic price data

Usage:

```
getSymbols(Symbols = NULL, env = parent.frame(), src = "yahoo",  
  auto.assign = getOption('getSymbols.auto.assign', TRUE), ...)
```

Main arguments:

- Symbols** a vector of ticker symbols
- env** where to create objects. Setting `env=NULL` is equal to `auto.assign=FALSE`
- src** source of data (yahoo)
- auto.assign** should results be returned or loaded to `env`
- ...** additional parameters

Return value:

an object of type `return.class` depending on `env` and `auto.assign`

The `getSymbols.yahoo` function

The `getSymbols.yahoo` function downloads historic price data from `finance.yahoo.com`

Usage:

```
getSymbols.yahoo(Symbols, env, return.class = 'xts', index.class = 'Date',  
  from = "2007-01-01", to = Sys.Date(), ...)
```

Main arguments:

`return.class` class of returned object
`index.class` class of returned object index (xts only)
... additional parameters

Return value:

an object of type `return.class` depending on `env` and `auto.assign`

The `getSymbols` function

```
library(quantmod)
ls()
```

```
## [1] "filename"
```

```
getSymbols("~GSPC")
```

```
ls()
```

```
## [1] "filename" "GSPC"
```

```
class(GSPC)
```

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

```
class(index(GSPC))
```

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

```
dim(GSPC)
```

```
## [1] 2267    6
```

- By default, the symbol was *auto-assigned* to the parent environment

The getSymbols function

```
tail(GSPC,4)
```

```
##           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
## 2015-12-29 2060.5400 2081.5601 2060.5400 2078.3601 2542000000 2078.3601
## 2015-12-30 2077.3401 2077.3401 2061.9700 2063.3601 2367430000 2063.3601
## 2015-12-31 2060.5901 2062.5400 2043.6200 2043.9399 2655330000 2043.9399
## 2016-01-04 2038.2000 2038.2000 1989.6801 2012.6600 4304880000 2012.6600
```

```
tail(C1(GSPC),4)
```

```
##           GSPC.Close
## 2015-12-29 2078.3601
## 2015-12-30 2063.3601
## 2015-12-31 2043.9399
## 2016-01-04 2012.6600
```

```
tail(Ad(GSPC),4)
```

```
##           GSPC.Adjusted
## 2015-12-29 2078.3601
## 2015-12-30 2063.3601
## 2015-12-31 2043.9399
## 2016-01-04 2012.6600
```

- Note that the symbol is prepended to columns names of the xts object; use extractor functions to access column data (e.g. C1(GSPC))

xts extractor functions

The `xts` package includes a number of functions to extract specific series from an `xts` object of market data:

Function	Description
<code>Op(x)</code>	Get Open
<code>Hi(x)</code>	Get High
<code>Lo(x)</code>	Get Low
<code>Cl(x)</code>	Get Close
<code>Vo(x)</code>	Get Volume
<code>Ad(x)</code>	Get Adjusted Close
<code>HLC(x)</code>	Get High, Low, and Close
<code>OHLC(x)</code>	Get Open, High, Low, and Close

The chartSeries function

The chartSeries function creates financial charts

Usage:

```
chartSeries(x, type = c("auto", "candlesticks", "matchsticks",  
  "bars", "line"), subset = NULL, show.grid = TRUE, name = NULL,  
  time.scale = NULL, log.scale = FALSE, TA = "addVo()", TAsep = ";",  
  line.type = "l", bar.type = "ohlc", theme = chartTheme("black"),  
  layout = NA, major.ticks = "auto", minor.ticks = TRUE, yrange = NULL,  
  plot = TRUE, up.col, dn.col, color.vol = TRUE, multi.col = FALSE, ...)
```

Main arguments:

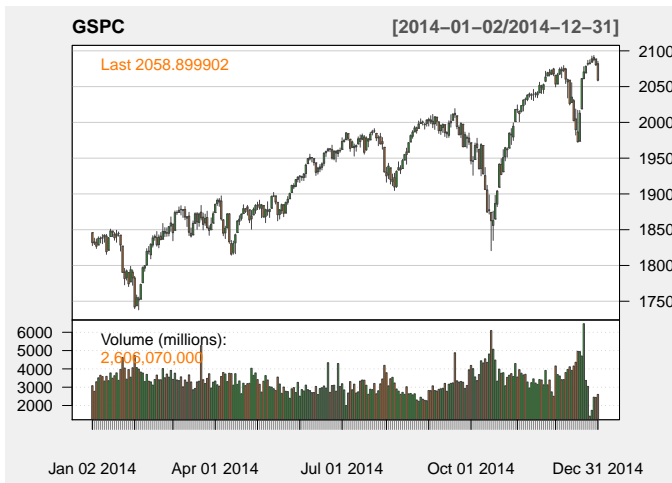
- x** an OHLC object
- type** style of chart to draw
- theme** a chart.theme object
- subset** xts style date subsetting argument
- TA** a vector of technical indicators and params

Return value:

a chob object

The chartSeries function

```
chartSeries(GSPC,subset="2014",theme="white")
```



Customize a chartSeries plot

```
whiteTheme <- chartTheme("white")
names(whiteTheme)

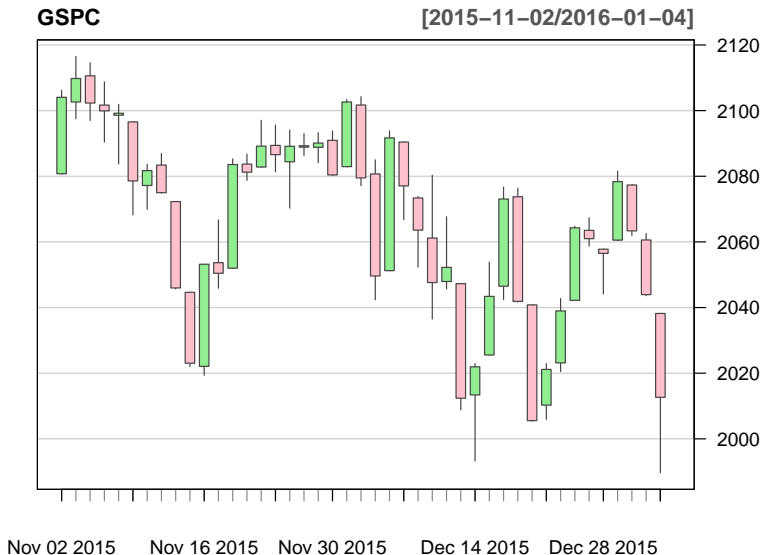
## [1] "fg.col"      "bg.col"      "grid.col"    "border"      "minor.tick"
## [6] "major.tick"  "up.col"      "dn.col"      "dn.up.col"   "up.up.col"
## [11] "dn.dn.col"   "up.dn.col"   "up.border"   "dn.border"   "dn.up.border"
## [16] "up.up.border" "dn.dn.border" "up.dn.border" "main.col"    "sub.col"
## [21] "area"        "fill"        "Expiry"      "theme.name"

whiteTheme$bg.col <- "white"
whiteTheme$dn.col <- "pink"
whiteTheme$up.col <- "lightgreen"
whiteTheme$border <- "lightgray"
x <- chartSeries(GSPC, subset="last 3 months", theme=whiteTheme, TA=NULL)
class(x)

## [1] "chob"
## attr(,"package")
## [1] "quantmod"
```

- subset to last 3 months
- totally white background
- no volume sub-graph (TA=NULL)

A chartSeries plot



Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with quantmod
- 5 Financial data access with other packages
- 6 Technical indicators and TTR
- 7 Intra-day data example

Time series data

Time series

A *time series* is a sequence of *ordered* data points measured at specific points in time

Time series object

A time series object in R is a *compound data structure* that includes a data matrix as well as a vector of associated time stamps

class	package	overview
ts	stats	regularly spaced time series
mts	stats	multiple regularly spaced time series
zoo	zoo	reg/irreg and arbitrary time stamp classes
xts	xts	an extension of the zoo class

Components of an xts object

An xts object is composed of 3 components:

- Index** a Data class or Time-Date class used for the time-stamp of observations
- Matrix** the time series observations (univariate or multivariate)
 - can be numeric, character, logical, etc. but must be homogeneous
- Attr** hidden attributes and user attributes
 - class of the index
 - format of the index
 - time zone

Date class

A Date object is stored internally as the number of days since 1970-01-01

```
myStr <- "7/4/2014"
class(myStr)

## [1] "character"

args(getS3method("as.Date", "character"))

## function (x, format, ...)
## NULL

myDate <- as.Date(myStr, format="%m/%d/%Y")
myDate

## [1] "2014-07-04"

class(myDate)

## [1] "Date"

as.numeric(myDate)

## [1] 16255
```


Date format string for `as.Date` and `format.Date`

- `%Y` Year with century
- `%y` Year without century (00-99)
- `%m` Month as decimal number (01-12)
- `%d` Day of the month as decimal number (01-31)

```
format(myDate, "%m/%d/%Y")
```

```
## [1] "07/04/2014"
```

```
format(myDate, "%m/%d/%y")
```

```
## [1] "07/04/14"
```

```
format(myDate, "%Y%m%d")
```

```
## [1] "20140704"
```

- For comprehensive list of date/time conversion specifications, see help for `strptime` function

Time-Date formatting strings

- `%a` Abbreviated weekday name in the current locale
- `%A` Full weekday name in the current locale
- `%b` Abbreviated month name in the current locale
- `%B` Full month name in the current locale
- `%c` Date and time. Locale-specific on output
- `%C` Century (00–99): the integer part of the year divided by 100.
- `%d` Day of the month as decimal number (01–31).
- `%D` Date format such as `%m/%d/%y`: ISO C99 says it should be that exact format.
- `%e` Day of the month as decimal number (1–31)
- `%F` Equivalent to `%Y-%m-%d` (the ISO 8601 date format)
- `%g` The last two digits of the week-based year (see `%V`)
- `%G` The week-based year (see `%V`) as a decimal number
- `%h` Equivalent to `%b`
- `%H` Hours as decimal number (00–23)
- `%I` Hours as decimal number (01–12)
- `%j` Day of year as decimal number (001–366)
- `%m` Month as decimal number (01–12)
- `%M` Minute as decimal number (00–59)

Time-Date formatting strings (continued)

- `%n` Newline on output, arbitrary whitespace on input
- `%p` AM/PM indicator in the locale. Used in conjunction with `%I` and not with `%H`
- `%r` The 12-hour clock time (using the locale's AM or PM)
- `%R` Equivalent to `%H:%M`
- `%S` Second as decimal number (00–61)
- `%T` Equivalent to `%H:%M:%S`
- `%u` Weekday as a decimal number (1–7, Monday is 1)
- `%U` Week of the year as decimal number (00–53) using Sunday as the first day 1 of the week
- `%V` Week of the year as decimal number (00–53) as defined in ISO 8601
- `%w` Weekday as decimal number (0–6, Sunday is 0)
- `%W` Week of the year as decimal number (00–53) using Monday as the first day of week
- `%x` Date. Locale-specific on output, `"%y/%m/%d"` on input
- `%X` Time. Locale-specific on output, `"%H:%M:%S"` on input
- `%y` Year without century (00–99)
- `%Y` Year with century
- `%z` Signed offset in hours and minutes from UTC, so `-0800` is 8 hours behind UTC
- `%Z` (Output only.) Time zone abbreviation as a character string (empty if not available)

Date-Time classes

- A POSIXct object is a Date-Time object internally stored as the number of seconds since 1970-01-01
- A POSIXlt object is a Date-Time object internally stored as 9 calendar and time components

```
d <- Sys.time()
class(d)

## [1] "POSIXct" "POSIXt"

unclass(d)

## [1] 1452026635

sapply(unclass(as.POSIXlt(d)), function(x) x)

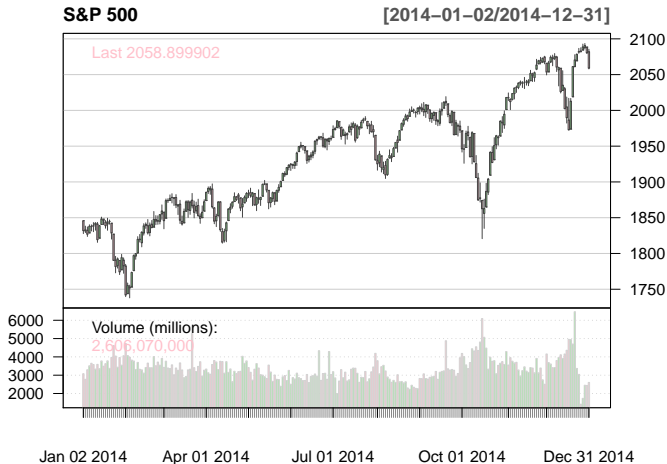
##           sec           min           hour           mday
## "55.2089378833771"      "43"      "12"      "5"
##           mon           year           wday           yday
##           "0"      "116"      "2"      "4"
##           isdst           zone           gmtoff
##           "0"      "PST"      "-28800"
```

xts time series objects can be easily subset:

- Date and time organized from most significant to least significant
 - CCYY-MM-DD HH:MM:SS[.s]
- Separators can be omitted
 - CCYYMMDDHHMMSS
- Intervals can be designated with the "/" or "::"
 - 2010/2011
 - 2011-04::2011-07
 - ::Sys.time()
 - 2000::

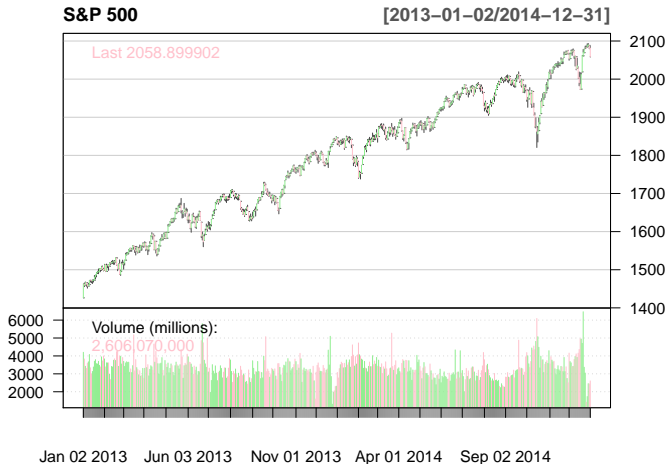
xts subsetting

```
chartSeries(GSPC["2014"], theme=whiteTheme, name="S&P 500")
```



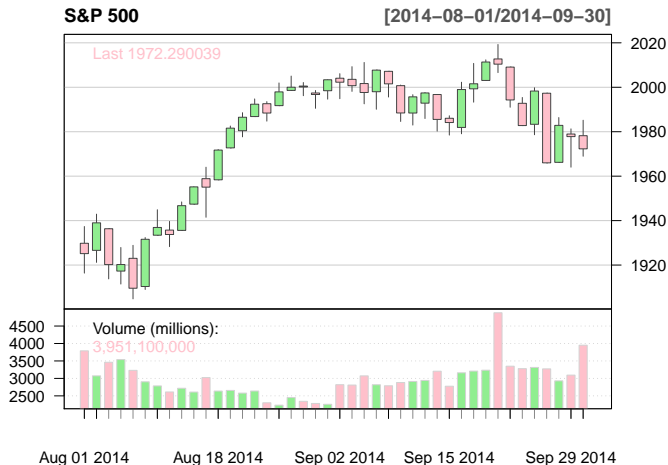
xts subsetting

```
chartSeries(GSPC["2013/2014"], theme=whiteTheme, name="S&P 500")
```



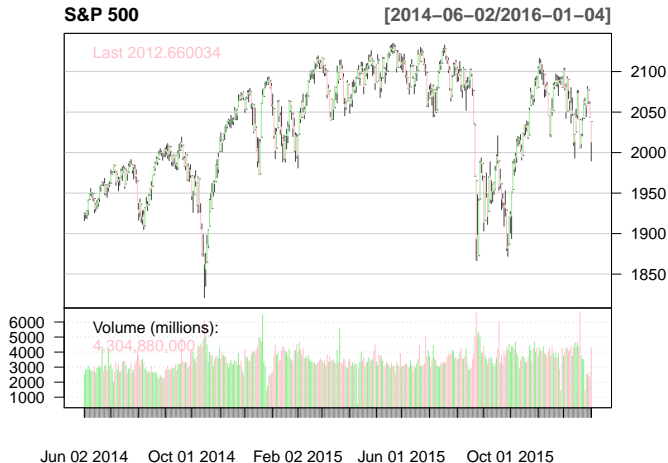
xts subsetting

```
chartSeries(GSPC["2014-08::2014-09"], theme=whiteTheme, name="S&P 500")
```



xts subsetting

```
chartSeries(GSPC["201406::"], theme=whiteTheme, name="S&P 500")
```



Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with quantmod**
- 5 Financial data access with other packages
- 6 Technical indicators and TTR
- 7 Intra-day data example

Download historic quotes: specifying the start date

```
getSymbols("SPY",from="2000-01-01")
class(SPY)

## [1] "xts" "zoo"

head(SPY)

##           SPY.Open SPY.High   SPY.Low SPY.Close SPY.Volume SPY.Adjusted
## 2000-01-03 148.2500 148.2500 143.87500 145.4375   8164300   108.09784
## 2000-01-04 143.5312 144.0625 139.64059 139.7500   8089800   103.87055
## 2000-01-05 139.9375 141.5312 137.25000 140.0000  12177900   104.05636
## 2000-01-06 139.6250 141.5000 137.75000 137.7500   6227200   102.38403
## 2000-01-07 140.3125 145.7500 140.06250 145.7500   8066500   108.33011
## 2000-01-10 146.2500 146.9062 145.03120 146.2500   5741700   108.70173

head(index(SPY))

## [1] "2000-01-03" "2000-01-04" "2000-01-05" "2000-01-06" "2000-01-07" "2000-01-10"

class(index(SPY))

## [1] "Date"
```

Download historic quotes: specifying the time stamp class

```
getSymbols("SBUX",index.class="POSIXct",from="2000-01-01")  
  
class(SBUX)  
  
## [1] "xts" "zoo"  
  
head(SBUX)  
  
##           SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted  
## 2000-01-03 23.875040 24.687519 23.250000 24.656160    24232000    2.833263  
## 2000-01-04 24.062481 24.875040 23.750000 23.875040    21564800    2.743503  
## 2000-01-05 23.937519 24.624960 23.687519 24.187519    28206400    2.779411  
## 2000-01-06 24.000000 25.624960 24.000000 25.062481    30825600    2.879953  
## 2000-01-07 24.750000 25.000000 24.250000 24.937519    26044800    2.865594  
## 2000-01-10 25.875040 26.750000 25.812481 26.000000    29031200    2.987684  
  
head(index(SBUX))  
  
## [1] "2000-01-03 UTC" "2000-01-04 UTC" "2000-01-05 UTC" "2000-01-06 UTC"  
## [5] "2000-01-07 UTC" "2000-01-10 UTC"  
  
class(index(SBUX))  
  
## [1] "POSIXct" "POSIXt"  
  
chartSeries(SBUX,theme=whiteTheme,minor.ticks=FALSE)
```

Unadjusted SBUX data



Get stock split history

```
(spl <- getSplits("SBUX"))
```

```
##           SBUX.spl
## 1993-09-30      0.5
## 1995-12-04      0.5
## 1999-03-22      0.5
## 2001-04-30      0.5
## 2005-10-24      0.5
## 2015-04-09      0.5
```

```
class(spl)
```

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

Get stock dividend history

```
(div <- getDividends("SBUX"))
```

```
##           [,1]
## 2010-04-05 0.050
## 2010-08-02 0.065
## 2010-11-16 0.065
## 2011-02-07 0.065
## 2011-05-09 0.065
## 2011-08-08 0.065
## 2011-11-15 0.085
## 2012-02-06 0.085
## 2012-05-07 0.085
## 2012-08-06 0.085
## 2012-11-13 0.105
## 2013-02-05 0.105
## 2013-05-07 0.105
## 2013-08-06 0.105
## 2013-11-12 0.130
## 2014-02-04 0.130
## 2014-05-06 0.130
## 2014-08-05 0.130
## 2014-11-10 0.160
## 2015-02-03 0.160
## 2015-05-05 0.160
## 2015-08-04 0.160
## 2015-11-09 0.200
```

```
class(div)
```

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

The adjustOHLC function

Adjusts all columns of an OHLC object for splits and dividends

Usage:

```
adjustOHLC(x, adjust = c("split", "dividend"), use.Adjusted = FALSE,  
  ratio = NULL, symbol.name=deparse(substitute(x)))
```

Main arguments:

- x** an OHLC object
- use.Adjusted** calculated from dividends/splits, or used Adjusted price column
- symbol.name** used if x is not named the same as the symbol adjusting

Return value:

An object of the original class, with prices adjusted for splits and dividends

Using `use.Adjusted = TRUE` will be less precise than the method that employs actual split and dividend information

Adjust for split and dividend

```
head(SBUX)
```

```
##           SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 23.875040 24.687519 23.250000 24.656160 24232000 2.833263
## 2000-01-04 24.062481 24.875040 23.750000 23.875040 21564800 2.743503
## 2000-01-05 23.937519 24.624960 23.687519 24.187519 28206400 2.779411
## 2000-01-06 24.000000 25.624960 24.000000 25.062481 30825600 2.879953
## 2000-01-07 24.750000 25.000000 24.250000 24.937519 26044800 2.865594
## 2000-01-10 25.875040 26.750000 25.812481 26.000000 29031200 2.987684
```

```
adj.exact <- adjustOHLC(SBUX)
```

```
head(adj.exact)
```

```
##           SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 2.7435032 2.8368659 2.6716792 2.8332624 24232000 2.833263
## 2000-01-04 2.7650422 2.8584141 2.7291347 2.7435032 21564800 2.743503
## 2000-01-05 2.7506827 2.8296772 2.7219550 2.7794104 28206400 2.779411
## 2000-01-06 2.7578624 2.9445881 2.7578624 2.8799531 30825600 2.879953
## 2000-01-07 2.8440456 2.8727734 2.7865902 2.8655936 26044800 2.865594
## 2000-01-10 2.9733250 3.0738675 2.9661363 2.9876843 29031200 2.987684
```

- An article that describes how Yahoo calculates the adjusted close can be found here:

http://help.yahoo.com/kb/index?locale=en_US&y=PROD_ACCT&page=content&id=SLN2311

Compare adjustment methods

```
head(adj.exact)
```

	SBUX.Open	SBUX.High	SBUX.Low	SBUX.Close	SBUX.Volume	SBUX.Adjusted
## 2000-01-03	2.7435032	2.8368659	2.6716792	2.8332624	24232000	2.833263
## 2000-01-04	2.7650422	2.8584141	2.7291347	2.7435032	21564800	2.743503
## 2000-01-05	2.7506827	2.8296772	2.7219550	2.7794104	28206400	2.779411
## 2000-01-06	2.7578624	2.9445881	2.7578624	2.8799531	30825600	2.879953
## 2000-01-07	2.8440456	2.8727734	2.7865902	2.8655936	26044800	2.865594
## 2000-01-10	2.9733250	3.0738675	2.9661363	2.9876843	29031200	2.987684

```
adj.approx <- adjustOHLC(SBUX, use.Adjusted=TRUE)  
head(adj.approx)
```

	SBUX.Open	SBUX.High	SBUX.Low	SBUX.Close	SBUX.Volume	SBUX.Adjusted
## 2000-01-03	2.7435038	2.8368665	2.6716798	2.833263	24232000	2.833263
## 2000-01-04	2.7650420	2.8584139	2.7291345	2.743503	21564800	2.743503
## 2000-01-05	2.7506833	2.8296778	2.7219555	2.779411	28206400	2.779411
## 2000-01-06	2.7578623	2.9445880	2.7578623	2.879953	30825600	2.879953
## 2000-01-07	2.8440460	2.8727738	2.7865905	2.865594	26044800	2.865594
## 2000-01-10	2.9733247	3.0738672	2.9661360	2.987684	29031200	2.987684

```
chartSeries(adj.exact, theme=whiteTheme, name="SBUX", minor.ticks=FALSE)
```

Adjusted SBUX plot



Download historic quotes with adjust=TRUE

```
getSymbols("SBUX",index.class="POSIXct",from="2000-01-01",adjust=TRUE)
```

```
## [1] "SBUX"
```

```
head(SBUX)
```

```
##           SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03  2.7435032 2.8368659 2.6716792  2.8332624    24232000      2.833263
## 2000-01-04  2.7650422 2.8584141 2.7291347  2.7435032    21564800      2.743503
## 2000-01-05  2.7506827 2.8296772 2.7219550  2.7794104    28206400      2.779411
## 2000-01-06  2.7578624 2.9445881 2.7578624  2.8799531    30825600      2.879953
## 2000-01-07  2.8440456 2.8727734 2.7865902  2.8655936    26044800      2.865594
## 2000-01-10  2.9733250 3.0738675 2.9661363  2.9876843    29031200      2.987684
```

```
head(adj.exact)
```

```
##           SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03  2.7435032 2.8368659 2.6716792  2.8332624    24232000      2.833263
## 2000-01-04  2.7650422 2.8584141 2.7291347  2.7435032    21564800      2.743503
## 2000-01-05  2.7506827 2.8296772 2.7219550  2.7794104    28206400      2.779411
## 2000-01-06  2.7578624 2.9445881 2.7578624  2.8799531    30825600      2.879953
## 2000-01-07  2.8440456 2.8727734 2.7865902  2.8655936    26044800      2.865594
## 2000-01-10  2.9733250 3.0738675 2.9661363  2.9876843    29031200      2.987684
```

The adjust parameter of getSymbols is undocumented

Merging xts objects together

```
symbols = c("SPY", "MDY", "AGG", "IEF", "TLT")

getSymbols(symbols,from="2010-01-01",to="2014-12-31",adjust=TRUE)

## [1] "SPY" "MDY" "AGG" "IEF" "TLT"

prices0 <- Cl(get(symbols[1]))
for(i in 2:length(symbols))
  prices0 <- merge(prices0,Cl(get(symbols[i])))
colnames(prices0) <- symbols
head(prices0)
```

##		SPY	MDY	AGG	IEF	TLT
##	2010-01-04	102.39366	126.62025	89.278714	79.123061	76.266918
##	2010-01-05	102.66471	126.94164	89.684881	79.470482	76.759457
##	2010-01-06	102.73699	127.63170	89.633032	79.149784	75.731922
##	2010-01-07	103.17067	128.23668	89.529327	79.149784	75.859304
##	2010-01-08	103.51400	129.04016	89.581184	79.247776	75.825335
##	2010-01-11	103.65856	128.78495	89.512047	79.301224	75.409227

Using adjustOHLC with a symbol list

```
# download not using the adjust option  
getSymbols(symbols,from="2010-01-01",to="2014-12-31")
```

```
for(symbol in symbols)  
  assign(x=symbol,value=adjustOHLC(get(symbol),symbol.name=symbol))  
  
prices2 <- NULL  
for(i in 1:length(symbols))  
  prices2 <- cbind(prices2,C1(get(symbols[i])))  
colnames(prices2) <- symbols  
all.equal(prices0,prices2,check.attributes=FALSE)  
  
## [1] TRUE  
  
prices <- xts()  
for(i in 1:length(symbols))  
  prices <- merge(prices,C1(get(symbols[i])))  
colnames(prices) <- symbols  
all.equal(prices2,prices,check.attributes=FALSE)  
  
## [1] TRUE
```

Federal reserve economic data

The function `getSymbols` can also be used to access data from the Federal Reserve Economic Data (FRED) database



<http://research.stlouisfed.org/fred2/>

Download interest rate data from FRED

```
if( Sys.info()['sysname'] == "Windows" )  
  setInternet2(TRUE)  
getSymbols('DTB3',src='FRED')  
  
first(DTB3,'1 week')  
  
##           DTB3  
## 1954-01-04 1.33  
  
last(DTB3,'1 week')  
  
##           DTB3  
## 2015-12-28 0.23  
## 2015-12-29 0.23  
## 2015-12-30 0.21  
## 2015-12-31 0.16  
  
chartSeries(DTB3,theme="white",minor.ticks=FALSE)
```

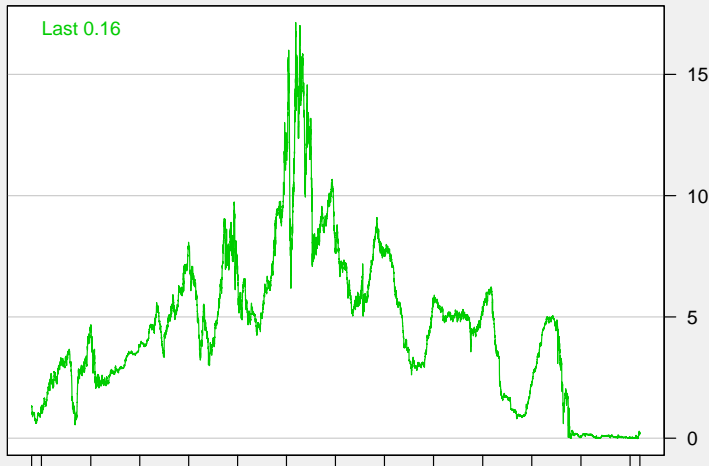
- The xts functions `first` and `last` are more powerful than `head` and `tail` when working with xts objects

Three-month U.S. Treasury bill rate

DTB3

[1954-01-04/2015-12-31]

Last 0.16



Jan 04 1954

Jan 02 1970

Jan 02 1985

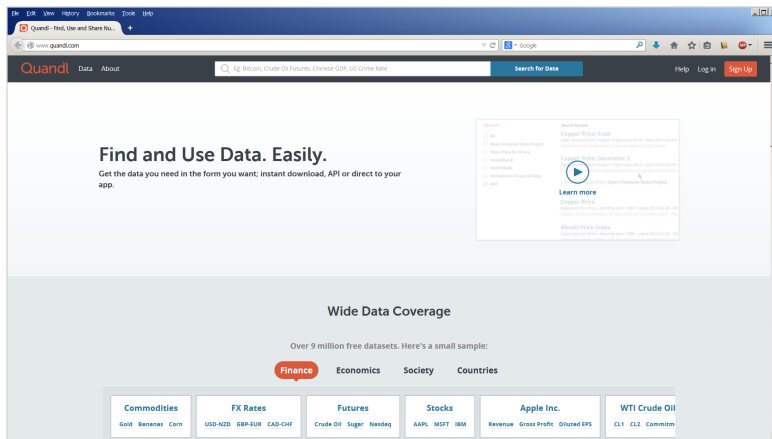
Jan 03 2000

Jan 02 2015

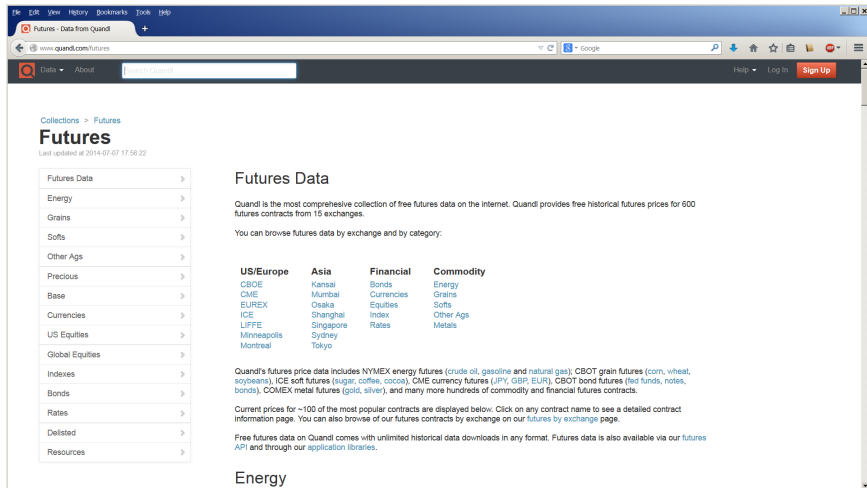
Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with `quantmod`
- 5 Financial data access with other packages**
- 6 Technical indicators and TTR
- 7 Intra-day data example

Tammer Kamel the founder of `www.quandl.com` wants to:
"do to Bloomberg what Wikipedia did to Britannica"



<http://www.quandl.com/>



The screenshot shows a web browser window displaying the 'Futures - Data from Quandl' page. The browser's address bar shows 'www.quandl.com/futures'. The page has a navigation bar with 'Data' and 'About' links, and a search bar. Below the navigation bar, the page title is 'Futures' with a sub-header 'Collections > Futures'. A sidebar on the left lists various data categories: Futures Data, Energy, Grains, Softs, Other Ags, Precious, Base, Currencies, US Equities, Global Equities, Indexes, Bonds, Rates, Delisted, and Resources. The main content area is titled 'Futures Data' and contains a paragraph explaining that Quandl provides free historical futures prices for 600 futures contracts from 15 exchanges. It also mentions that users can browse futures data by exchange and by category. Below this, there are four columns of links: US/Europe (CBOE, CME, EUREX, ICE, LIFFE, Minneapolis, Montreal), Asia (Kansai, Mumbai, Osaka, Shanghai, Singapore, Sydney, Tokyo), Financial (Bonds, Currencies, Equities, Index, Rates), and Commodity (Energy, Grains, Softs, Other Ags, Metals). A paragraph follows, stating that Quandl's futures price data includes NYMEX energy futures (crude oil, gasoline and natural gas), CBOT grain futures (corn, wheat, soybeans), ICE soft futures (sugar, coffee, cocoa), CME currency futures (JPY, GBP, EUR), CBOT bond futures (fed funds, notes, bonds), COMEX metal futures (gold, silver), and many more hundreds of commodity and financial futures contracts. Another paragraph mentions that current prices for ~100 of the most popular contracts are displayed below, and users can click on any contract name to see a detailed contract information page. It also states that users can browse futures contracts by exchange on the 'futures by exchange' page. A final paragraph notes that free futures data on Quandl comes with unlimited historical data downloads in any format, and that futures data is also available via the 'futures API' and through their 'application libraries'. The page ends with the word 'Energy'.

Collections > Futures

Futures

Last updated at 2014-07-07 17:56:22

- Futures Data >
- Energy >
- Grains >
- Softs >
- Other Ags >
- Precious >
- Base >
- Currencies >
- US Equities >
- Global Equities >
- Indexes >
- Bonds >
- Rates >
- Delisted >
- Resources >

Futures Data

Quandl is the most comprehensive collection of free futures data on the internet. Quandl provides free historical futures prices for 600 futures contracts from 15 exchanges.

You can browse futures data by exchange and by category:

US/Europe	Asia	Financial	Commodity
CBOE	Kansai	Bonds	Energy
CME	Mumbai	Currencies	Grains
EUREX	Osaka	Equities	Softs
ICE	Shanghai	Index	Other Ags
LIFFE	Singapore	Rates	Metals
Minneapolis	Sydney		
Montreal	Tokyo		

Quandl's futures price data includes NYMEX energy futures (crude oil, gasoline and natural gas); CBOT grain futures (corn, wheat, soybeans), ICE soft futures (sugar, coffee, cocoa), CME currency futures (JPY, GBP, EUR), CBOT bond futures (fed funds, notes, bonds), COMEX metal futures (gold, silver), and many more hundreds of commodity and financial futures contracts.

Current prices for ~100 of the most popular contracts are displayed below. Click on any contract name to see a detailed contract information page. You can also browse of our futures contracts by exchange on our [futures by exchange](#) page.

Free futures data on Quandl comes with unlimited historical data downloads in any format. Futures data is also available via our [futures API](#) and through our [application libraries](#).

Energy

The Quandl package

The Quandl package interacts directly with the Quandl API to offer data in a number of formats usable in R

Key functions:

`Quandl` Pulls data from the Quandl API

`Quandl.auth` Query or set Quandl API token[†]

`Quandl.search` Search the Quandl database

Authors:

- Raymond McTaggart

[†]Anonymous API calls are limited to 50 requests per day; signed up users receive an authorization token that allows them to get 500 API calls per day; see <http://www.quandl.com/help/r>

The Quandl function

The Quandl function pulls data from the Quandl API

Usage:

```
Quandl(code, type = c("raw", "ts", "zoo", "xts"), start_date, end_date,  
       transformation = c("", "diff", "rdiff", "normalize", "cumul", "rdiff_from"),  
       collapse = c("", "weekly", "monthly", "quarterly", "annual"),  
       sort = c("desc", "asc"), meta = FALSE, authcode = Quandl.auth(), ...)
```

Main arguments:

code	Dataset code on Quandl specified as a string
type	Type of data returned ('raw', 'ts', 'zoo' or 'xts')
start_date	Start date
end_date	End date
collapse	Frequency of data

Return value:

time series data in the specified format

Downloading data from Quandl

```
library(Quandl)

c11 = Quandl("OFDP/FUTURE_CL1",type="xts")
class(c11)

## [1] "xts" "zoo"

class(index(c11))

## [1] "Date"

first(c11)

##           Open   High    Low Settle Volume Open Interest
## 1983-03-30 29.01 29.56 29.01   29.4     949           470

last(c11)

##           Open   High    Low Settle Volume Open Interest
## 2016-01-04 37.6 38.39 36.33  36.76 426831           437108
```

Downloading data from Quandl

```
c11 <- c11[,c("Open", "High", "Low", "Settle", "Volume")]  
colnames(c11) <- c("Open", "High", "Low", "Close", "Volume")  
sum(is.na(coredata(c11)))
```

```
## [1] 0
```

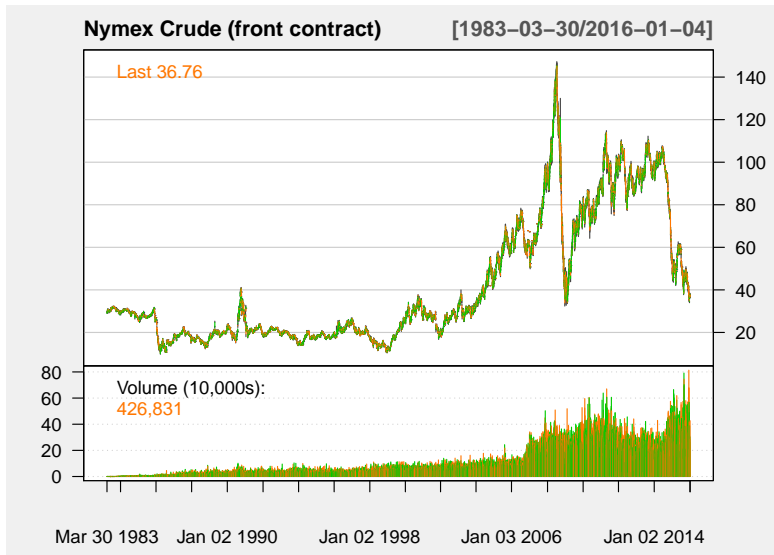
```
sum(coredata(c11)<0.01)
```

```
## [1] 25
```

```
c11[c11 < 0.1] <- NA  
c11 <- na.approx(c11)  
chartSeries(c11, name="Nymex Crude (front contract)",  
  theme=chartTheme("white"), minor.ticks=FALSE)
```

- Required data cleaning: replace zero values with NA and then use the function `na.approx` to interpolate

Historic futures data from Quandl



The tseries package

The tseries package is a collection of functions for time series analysis and computational finance

Key functions:

- | | |
|-------------------------------|---|
| <code>get.hist.quote</code> | Download historical financial data from a given data provider over the WWW (Yahoo, Oanda) |
| <code>jarque.bera.test</code> | Tests the null of normality for x using the Jarque-Bera test statistic |
| <code>adf.test</code> | Computes the Augmented Dickey-Fuller test for the null that x has a unit root |
| <code>po.test</code> | Computes the Phillips-Ouliaris test for the null hypothesis that x is not cointegrated |

Authors:

- Adrian Trapletti
- Kurt Hornik

Downloading data from Yahoo

The `get.hist.quote` function from the package `tseries` is able to download data directly from Yahoo Finance into a zoo object

```
library(tseries)
args(get.hist.quote)

## function (instrument = "^gdax", start, end, quote = c("Open",
##      "High", "Low", "Close"), provider = c("yahoo", "oanda"),
##      method = NULL, origin = "1899-12-30", compression = "d",
##      retclass = c("zoo", "its", "ts"), quiet = FALSE, drop = FALSE)
## NULL
```

Main arguments:

<code>instrument</code>	Yahoo ticker
<code>quote</code>	Open, High, Low, Close, Volume, Adjusted Close (O,H,L,C,V,A)
<code>start</code>	Starting date
<code>end</code>	Ending date

Downloading data from Yahoo

```
BA <- get.hist.quote(instrument="BA",start="2009-01-01",quote="A")
SPY <- get.hist.quote(instrument="SPY",start="2009-01-01",quote="A")
BA.ret <- diff(log(BA))
SPY.ret <- diff(log(SPY))
```

```
class(SPY)
```

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

```
class(time(SPY))
```

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

```
tail(SPY,3)
```

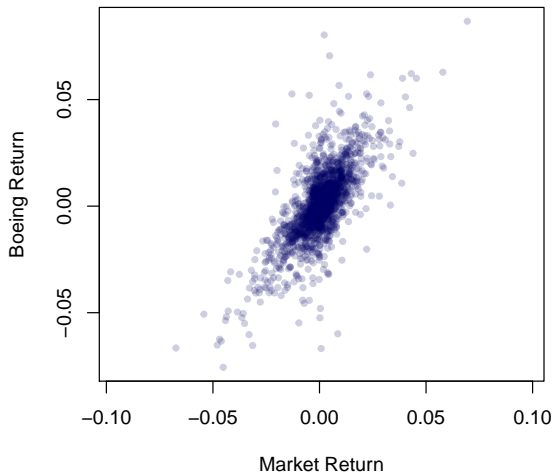
```
##           AdjClose
## 2015-12-30 205.92999
## 2015-12-31 203.86999
## 2016-01-04 201.02000
```

```
plot(x=SPY.ret, y=BA.ret, pch=20, asp=1, xlab="Market Return", ylab="Boeing Return",
     main="Boeing Return versus Market Return",
     col=rgb(0,0,100,50,maxColorValue=255))
```

- Color specification `rgb(0,0,100,50,maxColorValue=255)` makes dense over-plotted areas darker

Historic market data via `get.hist.quote`

Boeing Return versus Market Return



Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with `quantmod`
- 5 Financial data access with other packages
- 6 Technical indicators and TTR**
- 7 Intra-day data example

The TTR package

The TTR package is a comprehensive collection of technical analysis indicators for R

Key features:

- moving averages
- oscillators
- price channels
- trend indicators

Author:

- Joshua Ulrich

Selected technical analysis indicators in TTR

Function	Description	Function	Description
stoch	stochastic oscillator	ADX	Directional Movement Index
aroon	Aroon indicator	ATR	Average True Range
BBands	Bollinger bands	CCI	Commodity Channel Index
chaikinAD	Chaikin Acc/Dist	chaikinVolatility	Chaikin Volatility
ROC	rate of change	momentum	momentum indicator
CLV	Close Location Value	CMF	Chaikin Money Flow
CMO	Chande Momentum Oscillator	SMA	simple moving average
EMA	exponential moving average	DEMA	double exp mov avg
VWMA	volume weighted MA	VWAP	volume weighed avg price
DonchianChannel	Donchian Channel	DPO	Detrended Price Oscillator
EMV	Ease of Movement Value	volatility	volatility estimators
MACD	MA converge/diverge	MFI	Money Flow Index
RSI	Relative Strength Index	SAR	Parabolic Stop-and-Reverse
TDI	Trend Detection Index	TRIX	Triple Smoothed Exponential Osc
VHF	Vertical Horizontal Filter	williamsAD	Williams Acc/Dist
WPR	William's % R	ZigZag	Zig Zag trend line

see Technical Analysis from A to Z by Steven Achelis

Calculate and plot Bollinger bands

```
b <- BBands(HLC=HLC(SBUX["2014"]), n=20, sd=2)
tail(b,10)
```

```
##           dn      mavg      up      pctB
## 2014-12-17 38.328746 40.000876 41.673006 0.32146876
## 2014-12-18 38.551758 40.063650 41.575542 0.36123607
## 2014-12-19 38.707521 40.101907 41.496293 0.23020905
## 2014-12-22 38.766436 40.121735 41.477034 0.31952505
## 2014-12-23 38.811348 40.151024 41.490700 0.53678612
## 2014-12-24 38.854216 40.176776 41.499335 0.50886441
## 2014-12-26 38.952065 40.227785 41.503505 0.57303657
## 2014-12-29 38.974273 40.253865 41.533458 0.61534743
## 2014-12-30 39.003108 40.282579 41.562049 0.60606667
## 2014-12-31 39.069004 40.332518 41.596032 0.64754956
```

```
chartSeries(SBUX,TA='addBBands();addBBands(draw="p");addVo()',
  subset='2014',theme="white")
```

$$\text{pctB} = \frac{\text{Close} - \text{LowerBand}}{\text{UpperBand} - \text{LowerBand}}$$

Bollinger bands



Moving Average Convergence-Divergence (MACD)

```
macd <- MACD( Cl(SBUX), 12, 26, 9, maType="EMA" )  
tail(macd)
```

```
##           macd      signal  
## 2015-12-24 -0.59447108 -0.48628765  
## 2015-12-28 -0.55342642 -0.49971540  
## 2015-12-29 -0.39060792 -0.47789391  
## 2015-12-30 -0.29963488 -0.44224210  
## 2015-12-31 -0.32925085 -0.41964385  
## 2016-01-04 -0.58332710 -0.45238050
```

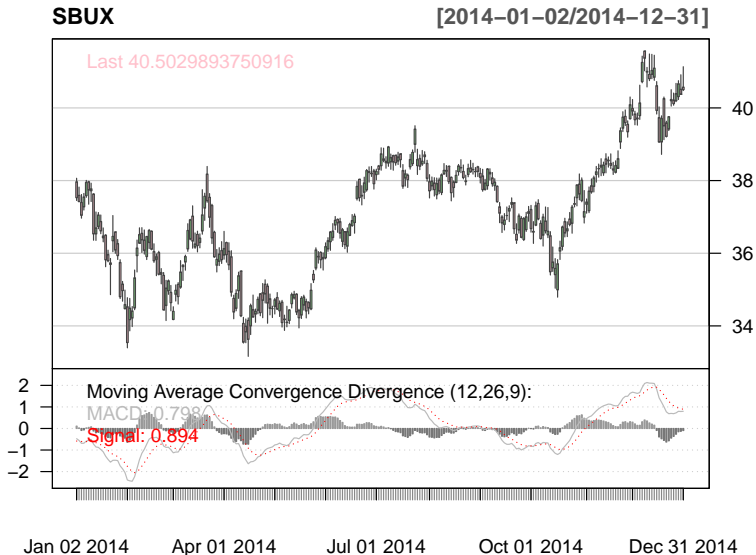
```
chartSeries(SBUX, TA = "addMACD()", subset="2014", theme=whiteTheme)
```

MACD line = $\text{EMA}(\text{Close}, 12) - \text{EMA}(\text{Close}, 26)$

Signal line = $\text{EMA}(\text{MACD line}, 9)$

MACD histogram = MACD line – Signal line

Moving Average Convergence-Divergence (MACD)



Relative Strength Index (RSI)

```
rsi <- RSI( Cl(SBUX), n = 14 )  
tail(rsi)
```

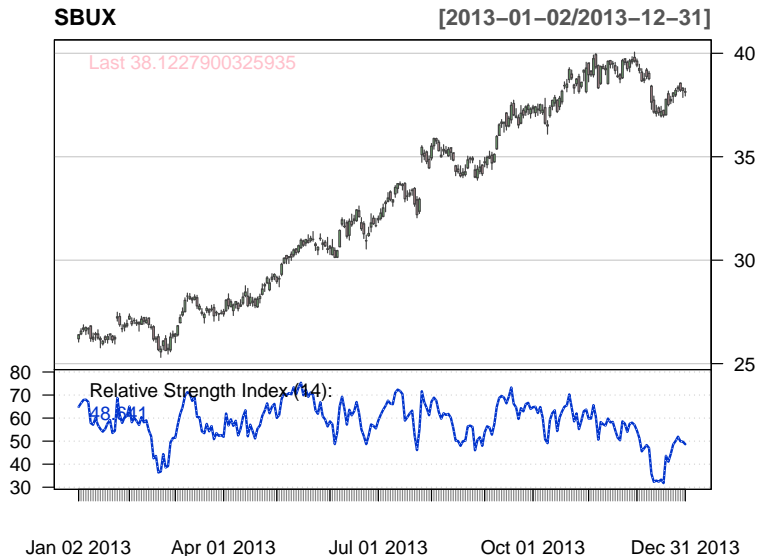
```
##                EMA  
## 2015-12-24 48.371990  
## 2015-12-28 47.596473  
## 2015-12-29 53.412585  
## 2015-12-30 51.387027  
## 2015-12-31 46.543019  
## 2016-01-04 37.918528
```

```
chartSeries(SBUX, TA = "addRSI()", subset="2013", theme=whiteTheme)
```

$$RSI = 100 - \frac{100}{1 + RS}$$

$$RS = \frac{\text{average of up changes}}{\text{average of down changes}}$$

Relative Strength Index (RSI)



Experimental chart_Series chart

The chartSeries functions are in the process of being updated; quantmod functions incorporating an underscore in their name are experimental version 2 functions:

- chart_Series
- add_TA
- chart_Theme

```
myTheme<-chart_theme()  
myTheme$col$up.col<-'lightgreen'  
myTheme$col$dn.col<-'pink'  
#  
chart_Series(SBUX["2013"],TA='add_BBands(lwd=2)',theme=myTheme,name="SBUX")
```

Experimental chart_Series chart



Outline

- 1 Introduction
- 2 Data download and charting
- 3 More about xts objects
- 4 More data retrieval with quantmod
- 5 Financial data access with other packages
- 6 Technical indicators and TTR
- 7 Intra-day data example

Working with intra-day data

- Intra-day time series require formal time-based classes for indexing
 - intra-day bars
 - tick data
- Recommended classes for high-frequency time series:
 - xts class
 - POSIXct/POSIXlt date-time class for indexing

Class	Daily data	Intra-day data
time series class	xts	xts
time index class	Date	POSIX1t

The `strptime` function

The `strptime` function converts character strings to POSIXlt date-time objects

Usage:

```
strptime(x, format, tz = "")
```

Main arguments:

- `x` vector of character strings to be converted to POSIXlt objects
- `format` date-time format specification
- `tz` timezone to use for conversion

Return value:

a POSIXlt object

Read GBPUSD 30-minute bars

```
fn1 <- "GBPUSD.txt"
dat <- read.table(file=fn1,sep=",",header=T,as.is=T)
head(dat)

##           Date Time   Open   High    Low  Close Up Down
## 1 10/20/2002 2330 1.5501 1.5501 1.5481 1.5482 0  0
## 2 10/21/2002   0 1.5481 1.5483 1.5472 1.5472 0  0
## 3 10/21/2002  30 1.5471 1.5480 1.5470 1.5478 0  0
## 4 10/21/2002 100 1.5477 1.5481 1.5471 1.5480 0  0
## 5 10/21/2002 130 1.5480 1.5501 1.5479 1.5493 0  0
## 6 10/21/2002 200 1.5492 1.5497 1.5487 1.5492 0  0

tm <- strptime(
  paste(dat[, "Date"], sprintf("%04d", dat[, "Time"])),
  format="%m/%d/%Y %H%M")
class(tm)

## [1] "POSIXlt" "POSIXt"

head(tm)

## [1] "2002-10-20 23:30:00 PDT" "2002-10-21 00:00:00 PDT" "2002-10-21 00:30:00 PDT"
## [4] "2002-10-21 01:00:00 PDT" "2002-10-21 01:30:00 PDT" "2002-10-21 02:00:00 PDT"
```

- Use paste with sprintf to format a Date-Time string
- Use the format argument of strptime to specify the formatting

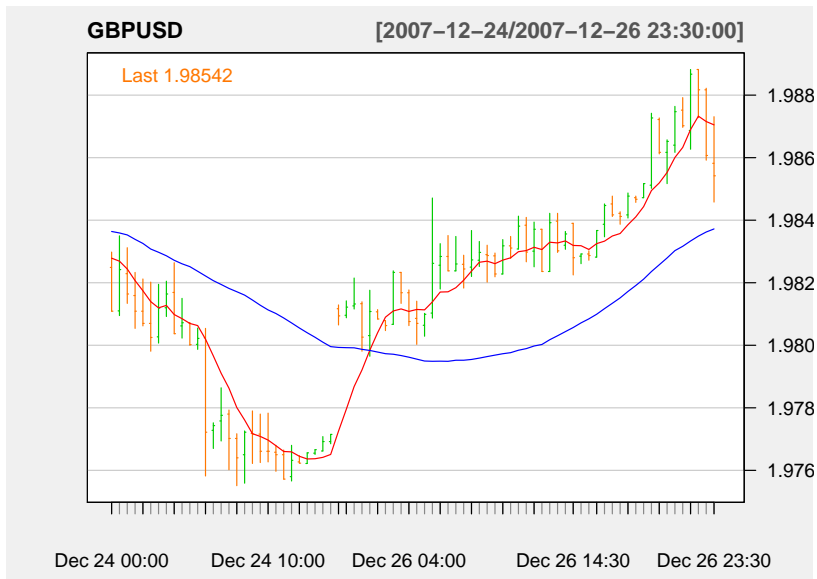
Create and plot xts object

```
GBP <- xts(x=dat[,c("Open", "High", "Low", "Close")], order.by=tm)
GBP <- GBP['2007']
first(GBP, '4 hours')
```

##		Open	High	Low	Close
##	2007-01-01 17:30:00	1.9649	1.9650	1.9644	1.9645
##	2007-01-01 18:00:00	1.9646	1.9648	1.9641	1.9644
##	2007-01-01 18:30:00	1.9645	1.9653	1.9645	1.9650
##	2007-01-01 19:00:00	1.9651	1.9652	1.9647	1.9650
##	2007-01-01 19:30:00	1.9651	1.9658	1.9651	1.9654
##	2007-01-01 20:00:00	1.9655	1.9657	1.9650	1.9654
##	2007-01-01 20:30:00	1.9653	1.9656	1.9651	1.9655

```
barChart(GBP, TA='addSMA(n = 7, col = "red");addSMA(n = 44, col = "blue")',
  subset='2007-12-24/2007-12-26', theme="white", name="GBPUSD")
```

GBPUSD crossover example



GBPUSD crossover example with annotation

```
# make candle stick plot with moving averages
#
chart_Series(GBP, subset='2007-12-24/2007-12-26', theme=myTheme, name="GBPUSD",
  TA='add_SMA(n=7,col="red",lwd=2);add_SMA(n=44,col="blue",lwd=2)')
#
# find cross-over bar
#
fastMA <- SMA(C1(GBP),n=7)
slowMA <- SMA(C1(GBP),n=44)
co <- fastMA > slowMA
x <- which(co['2007-12-24/2007-12-26'])[1]
#
# identify cross-over bar
#
ss <- GBP['2007-12-24/2007-12-26']
add_TA(ss[x,"Low"]-0.0005,pch=17,type="p",col="red", on=1,cex=2)
#
text(x=x,y=ss[x,"Low"]-0.0005,"Crossover\nbar",pos=1)
```

GBPUSD crossover example with annotation



W COMPUTATIONAL FINANCE & RISK MANAGEMENT
UNIVERSITY *of* WASHINGTON
Department of Applied Mathematics

`http://depts.washington.edu/compfin`