COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

Introduction to Trading Systems

Guy Yollin

Applied Mathematics
University of Washington

Outline

Outline

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:

zoo	create a zoo time series object
merge	merges time series (automatically handles of time alignment)
aggregate	create coarser resolution time series with summary statistics
rollapply	calculate rolling window statistics
read.zoo	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:

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

Authors:

- Jeffrey Ryan
- Josh Ulrich

R-forge

- 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

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] 2083    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
## 2015-04-08   2076.94   2086.69  2073.30    2081.90  3265330000       2081.90
## 2015-04-09   2081.29   2093.31  2074.29    2091.18  3172360000       2091.18
## 2015-04-10   2091.51   2102.61  2091.51    2102.06   536200000       2102.06
## 2015-04-13   2102.03   2107.65  2092.33    2092.43  2908420000       2092.43
```

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

```
##            GSPC.Close
## 2015-04-08    2081.90
## 2015-04-09    2091.18
## 2015-04-10    2102.06
## 2015-04-13    2092.43
```

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

```
##            GSPC.Adjusted
## 2015-04-08       2081.90
## 2015-04-09       2091.18
## 2015-04-10       2102.06
## 2015-04-13       2092.43
```

- Note that the symbol is prepended to columns names of the xts object; use extractor functions to access column data (e.g. `Cl(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
Op(x)	Get Open
Hi(x)	Get High
Lo(x)	Get Low
Cl(x)	Get Close
Vo(x)	Get Volume
Ad(x)	Get Adjusted Close
HLC(x)	Get High, Low, and Close
OHLC(x)	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:

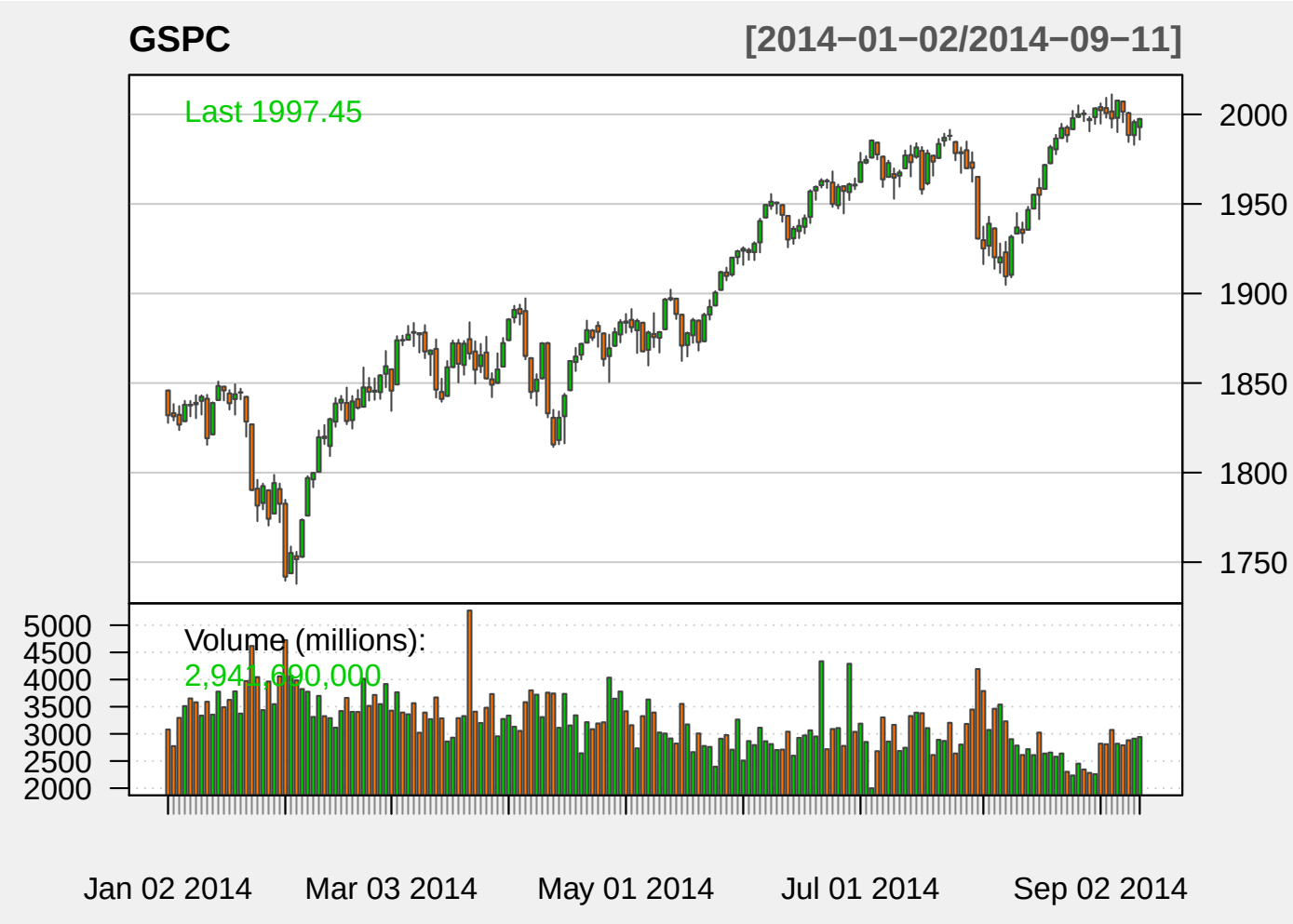
x	an OHLC object
type	style of chart to draw
theme	a chart.theme object
subset	xts style date subsetting argument
TA	a vector of technical indicators and params

Return value:

a chob object

The chartSeries function

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



Customize a chartSeries plot

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

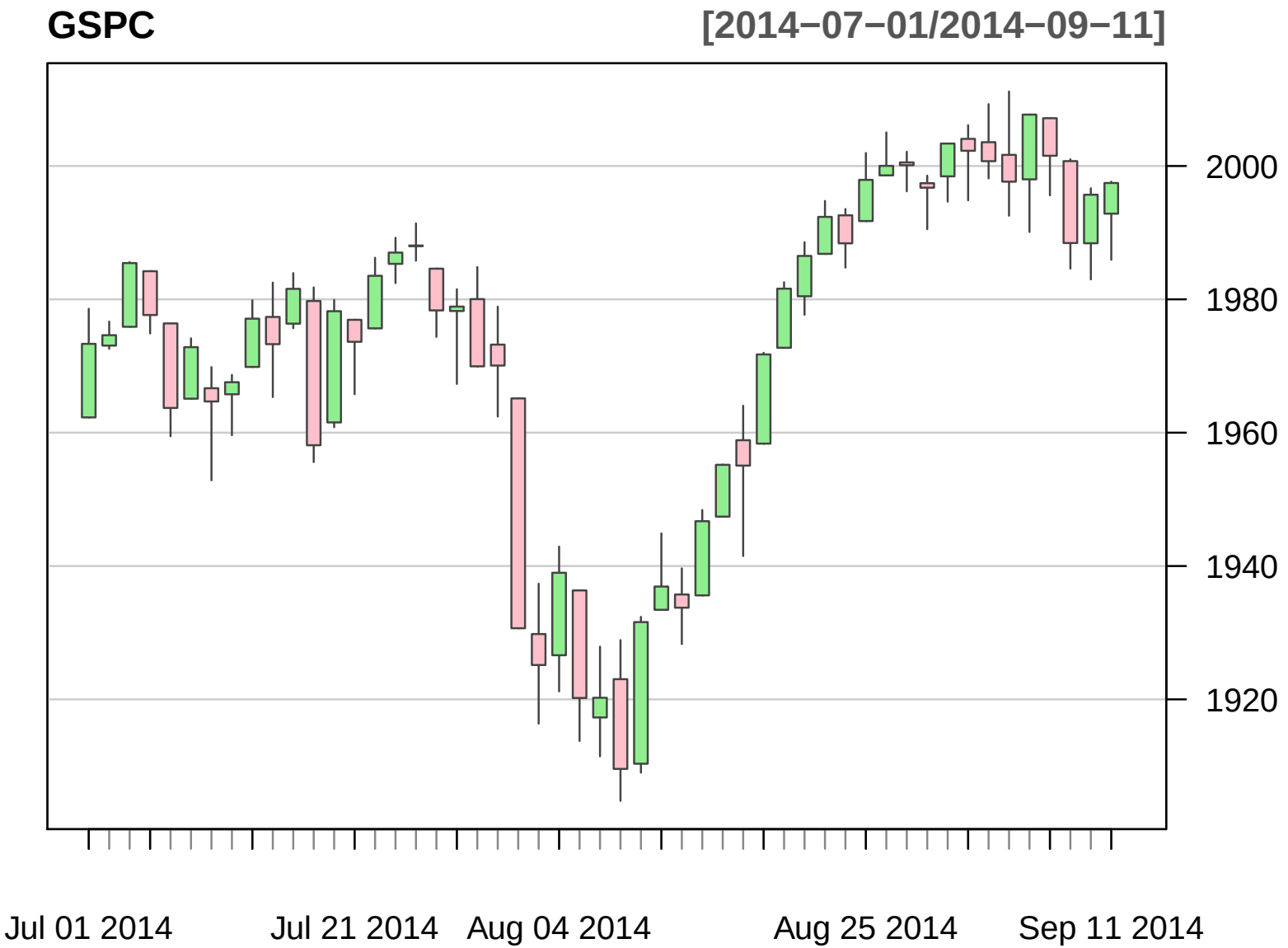
```
##  [1] "fg.col"       "bg.col"       "grid.col"     "border"       "minor.tick"
##  [6] "major.tick"   "up.col"       "dn.col"       "dn.up.col"    "up.up.col"
## [11] "dn.dn.col"    "up.dn.col"    "up.border"    "dn.border"    "dn.up.border"
## [16] "up.up.border" "dn.dn.border" "up.dn.border" "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

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] 1429041661
```

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

```
##                 sec                 min                hour                mday
## "0.775288105010986"                 "1"                "13"                "14"
##                 mon                year                wday                yday
##                 "3"               "115"                 "2"               "103"
##               isdst                zone               gmtoff
##                 "1"               "PDT"           "-25200"
```

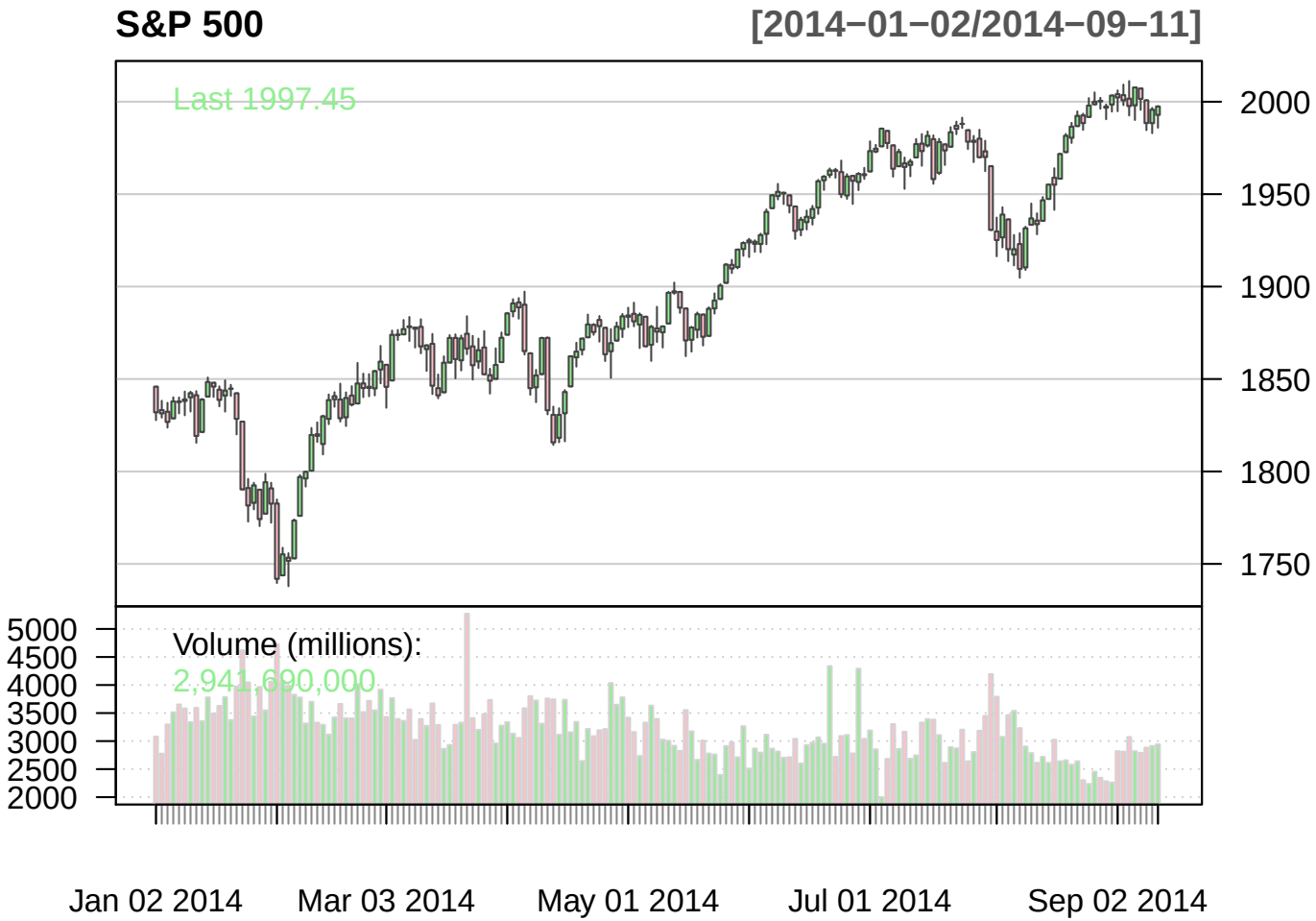
xts subsetting

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::

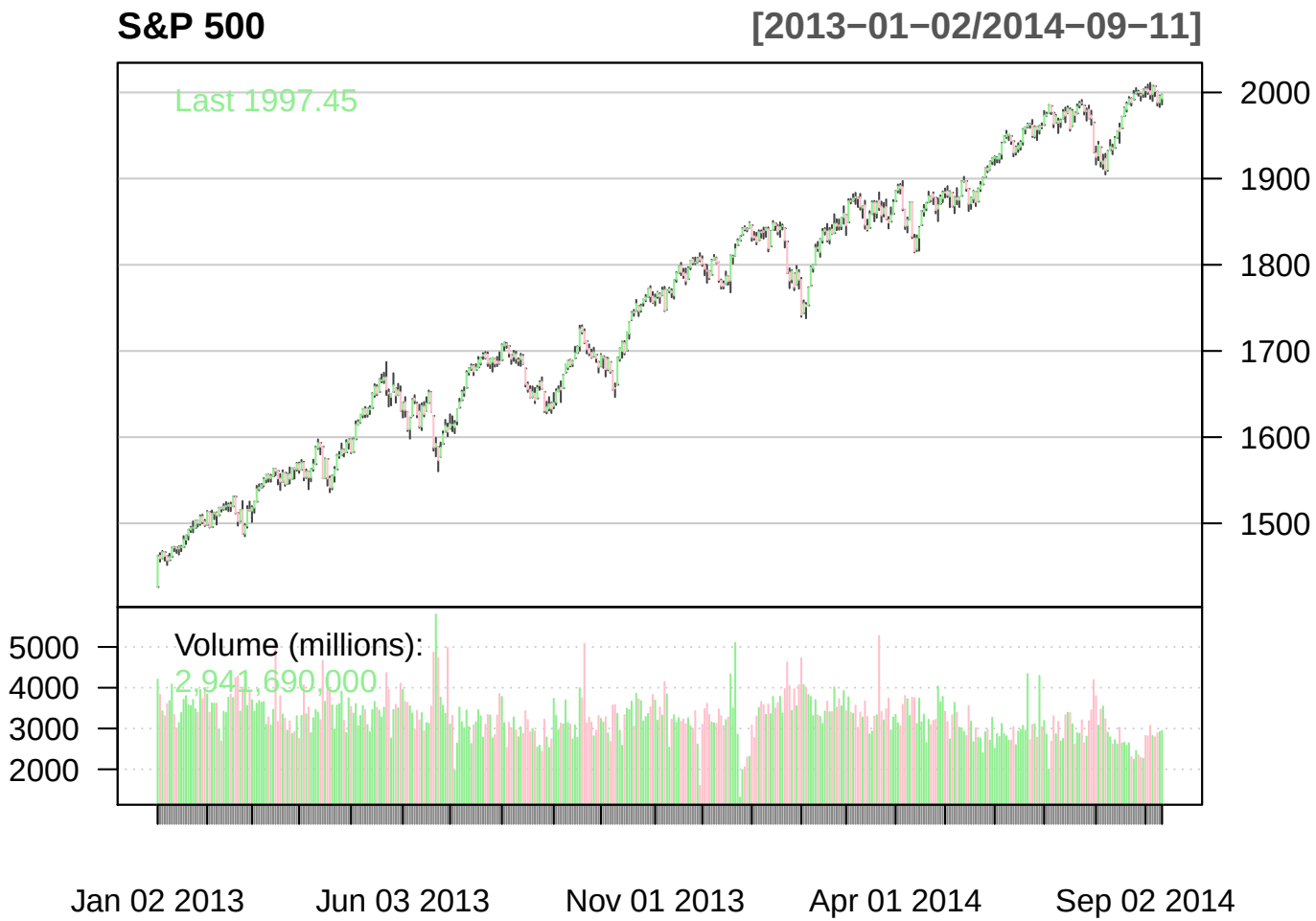
xts subsetting

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



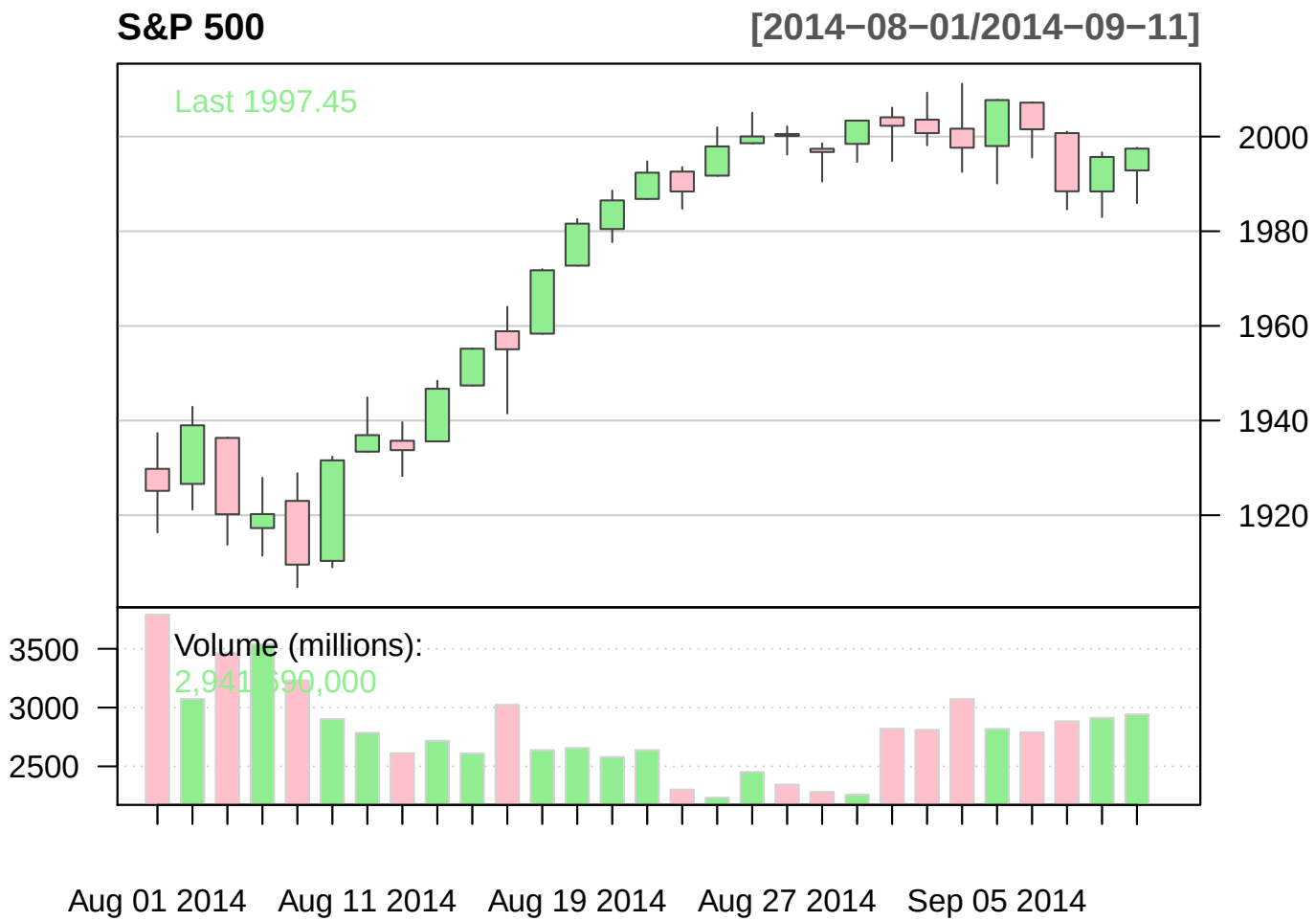
xts subsetting

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



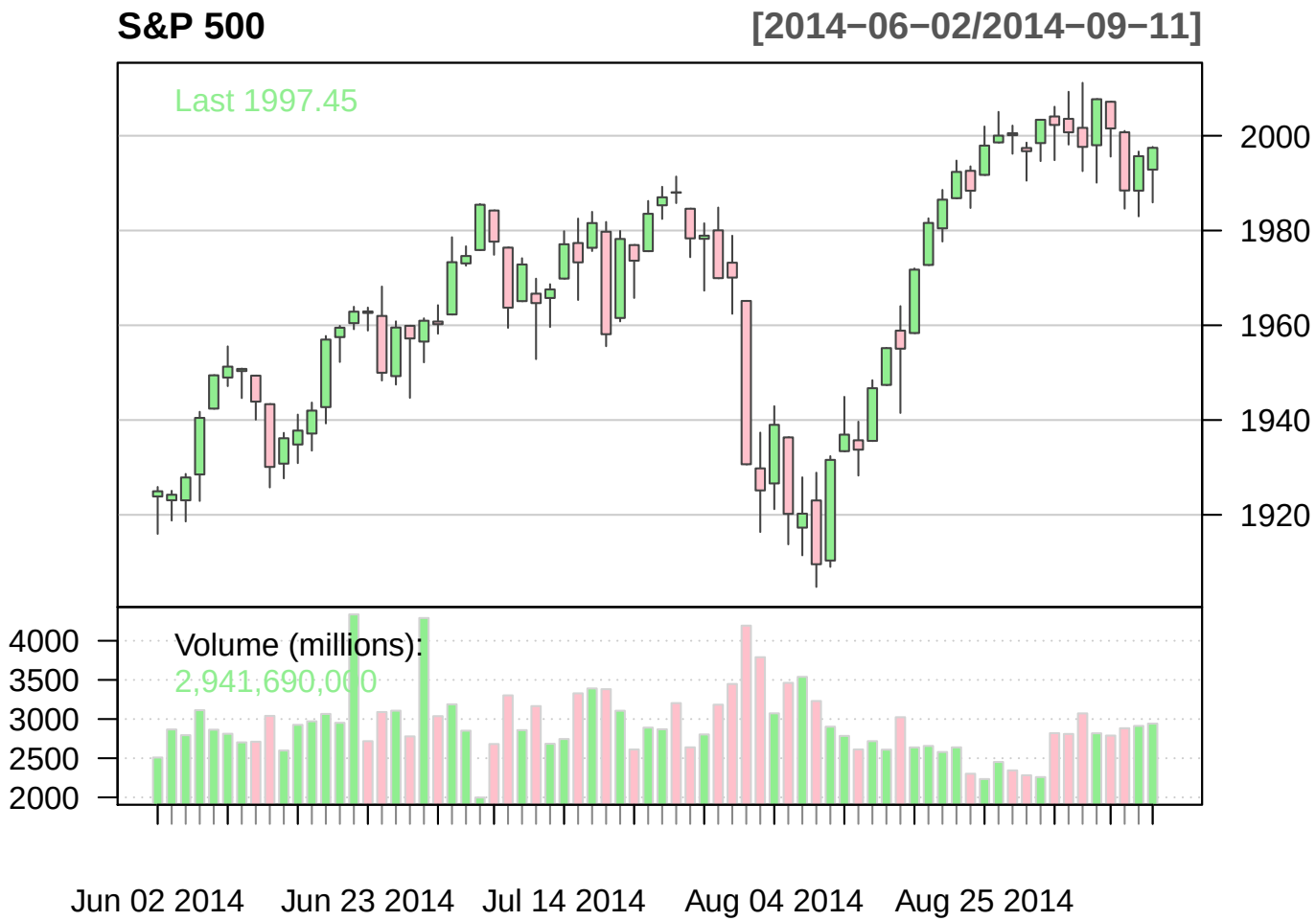
xts subsetting

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



xts subsetting

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



Outline

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   148.25   148.25  143.88    145.44    8164300       109.84
## 2000-01-04   143.53   144.06  139.64    139.75    8089800       105.54
## 2000-01-05   139.94   141.53  137.25    140.00   12177900       105.73
## 2000-01-06   139.62   141.50  137.75    137.75    6227200       104.03
## 2000-01-07   140.31   145.75  140.06    145.75    8066500       110.07
## 2000-01-10   146.25   146.91  145.03    146.25    5741700       110.45
```

```
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     23.88     24.69    23.25      24.66    24232000          2.86
## 2000-01-04     24.06     24.88    23.75      23.88    21564800          2.77
## 2000-01-05     23.94     24.62    23.69      24.19    28206400          2.80
## 2000-01-06     24.00     25.62    24.00      25.06    30825600          2.91
## 2000-01-07     24.75     25.00    24.25      24.94    26044800          2.89
## 2000-01-10     25.88     26.75    25.81      26.00    29031200          3.02
```

```
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
```

```
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     23.88     24.69    23.25      24.66    24232000          2.86
## 2000-01-04     24.06     24.88    23.75      23.88    21564800          2.77
## 2000-01-05     23.94     24.62    23.69      24.19    28206400          2.80
## 2000-01-06     24.00     25.62    24.00      25.06    30825600          2.91
## 2000-01-07     24.75     25.00    24.25      24.94    26044800          2.89
## 2000-01-10     25.88     26.75    25.81      26.00    29031200          3.02
```

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

```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 2.7693312 2.8632658 2.6962710  2.8597867    24232000          2.86
## 2000-01-04 2.7902056 2.8852998 2.7542553  2.7693312    21564800          2.77
## 2000-01-05 2.7762893 2.8551480 2.7472972  2.8052815    28206400          2.80
## 2000-01-06 2.7832474 2.9711166 2.7832474  2.9061742    30825600          2.91
## 2000-01-07 2.8702239 2.8992161 2.8122396  2.8922580    26044800          2.89
## 2000-01-10 3.0012685 3.1021612 2.9931507  3.0151847    29031200          3.02
```

- 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 2.7693312 2.8632658 2.6962710  2.8597867    24232000          2.86
## 2000-01-04 2.7902056 2.8852998 2.7542553  2.7693312    21564800          2.77
## 2000-01-05 2.7762893 2.8551480 2.7472972  2.8052815    28206400          2.80
## 2000-01-06 2.7832474 2.9711166 2.7832474  2.9061742    30825600          2.91
## 2000-01-07 2.8702239 2.8992161 2.8122396  2.8922580    26044800          2.89
## 2000-01-10 3.0012685 3.1021612 2.9931507  3.0151847    29031200          3.02
```

```
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 2.7695377 2.8634793 2.6964720       2.86    24232000          2.86
## 2000-01-04 2.7908794 2.8859966 2.7549204       2.77    21564800          2.77
## 2000-01-05 2.7710624 2.8497726 2.7421248       2.80    28206400          2.80
## 2000-01-06 2.7869114 2.9750279 2.7869114       2.91    30825600          2.91
## 2000-01-07 2.8679832 2.8969527 2.8100441       2.89    26044800          2.89
## 2000-01-10 3.0060615 3.1071154 2.9979308       3.02    29031200          3.02
```

```
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 2.7693312 2.8632658 2.6962710  2.8597867    24232000          2.86
## 2000-01-04 2.7902056 2.8852998 2.7542553  2.7693312    21564800          2.77
## 2000-01-05 2.7762893 2.8551480 2.7472972  2.8052815    28206400          2.80
## 2000-01-06 2.7832474 2.9711166 2.7832474  2.9061742    30825600          2.91
## 2000-01-07 2.8702239 2.8992161 2.8122396  2.8922580    26044800          2.89
## 2000-01-10 3.0012685 3.1021612 2.9931507  3.0151847    29031200          3.02
```

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

```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 2.7693312 2.8632658 2.6962710  2.8597867    24232000          2.86
## 2000-01-04 2.7902056 2.8852998 2.7542553  2.7693312    21564800          2.77
## 2000-01-05 2.7762893 2.8551480 2.7472972  2.8052815    28206400          2.80
## 2000-01-06 2.7832474 2.9711166 2.7832474  2.9061742    30825600          2.91
## 2000-01-07 2.8702239 2.8992161 2.8122396  2.8922580    26044800          2.89
## 2000-01-10 3.0012685 3.1021612 2.9931507  3.0151847    29031200          3.02
```

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 89.278716 79.123061 76.266920
## 2010-01-05 102.66471 126.94165 89.684882 79.470483 76.759458
## 2010-01-06 102.73699 127.63170 89.633031 79.149786 75.731922
## 2010-01-07 103.17067 128.23668 89.529329 79.149786 75.859302
## 2010-01-08 103.51400 129.04017 89.581180 79.247777 75.825334
## 2010-01-11 103.65856 128.78494 89.512045 79.301226 75.409225
```

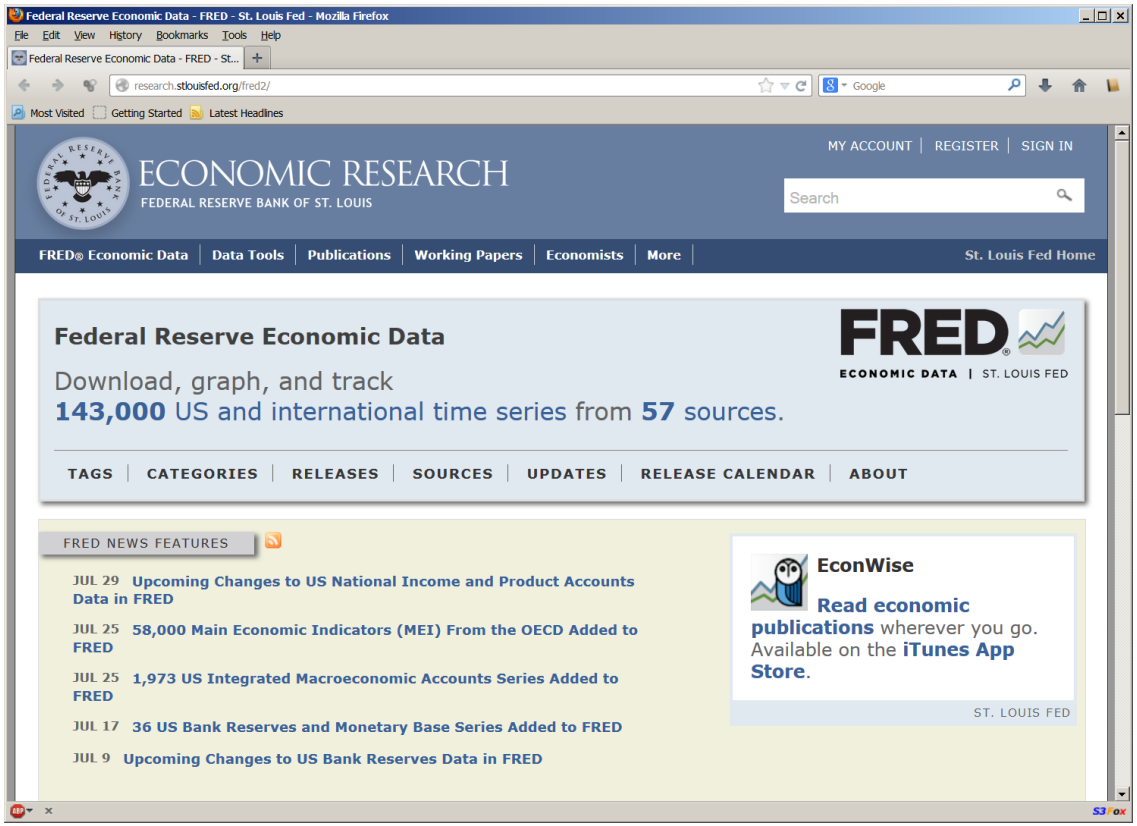
Using adjustOHLC with a symbol list

```
# 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,Cl(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,Cl(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



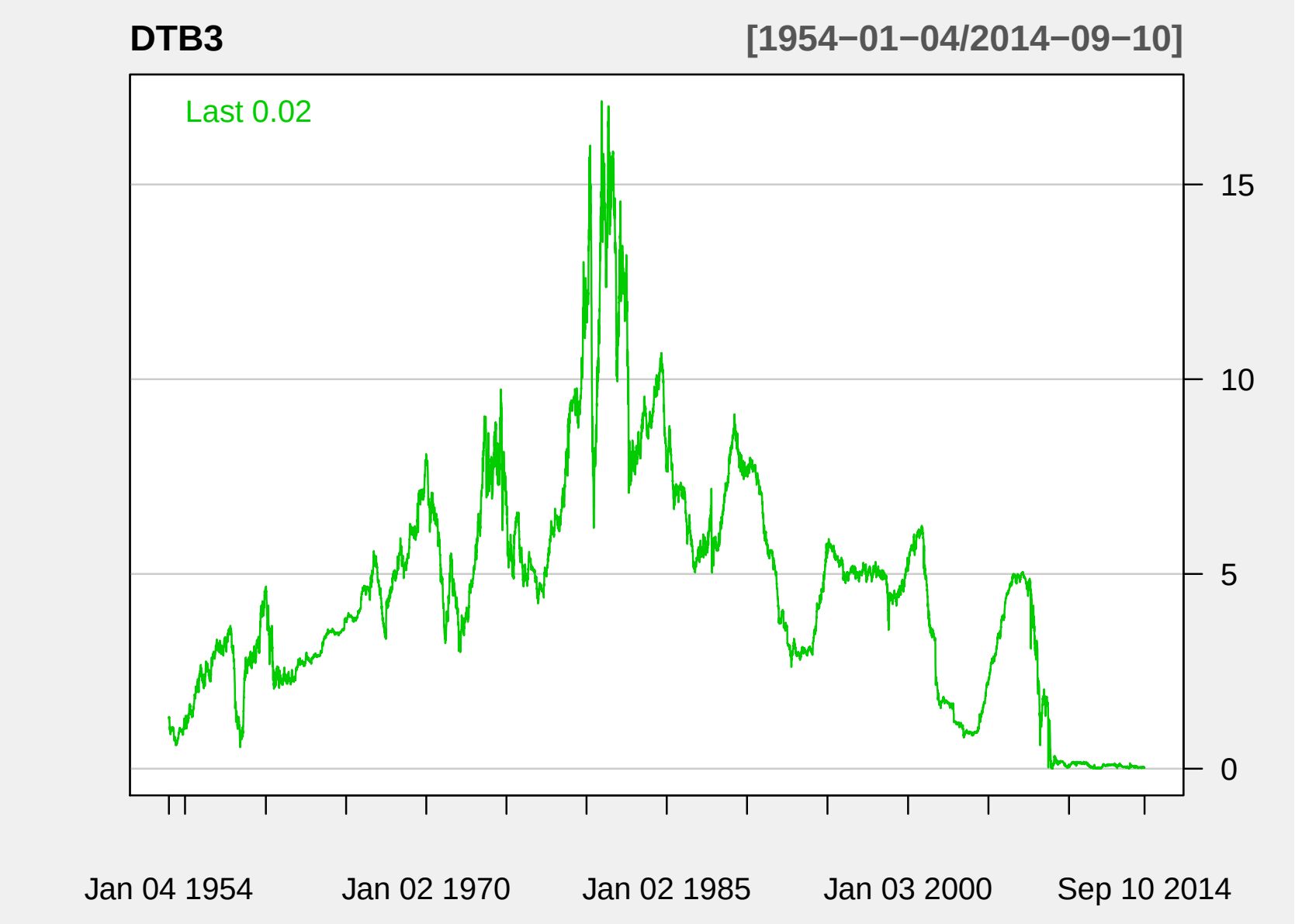
`http://research.stlouisfed.org/fred2/`

Download interest rate data from FRED

```
getSymbols('DTB3',src='FRED')
first(DTB3,'1 week')

##            DTB3
## 1954-01-04 1.33

last(DTB3,'1 week')

##            DTB3
## 2015-04-06 0.03
## 2015-04-07 0.02
## 2015-04-08 0.03
## 2015-04-09 0.03
## 2015-04-10 0.02

chartSeries(DTB3,theme="white",minor.ticks=FALSE)
```

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

Three-month U.S. Treasury bill rate

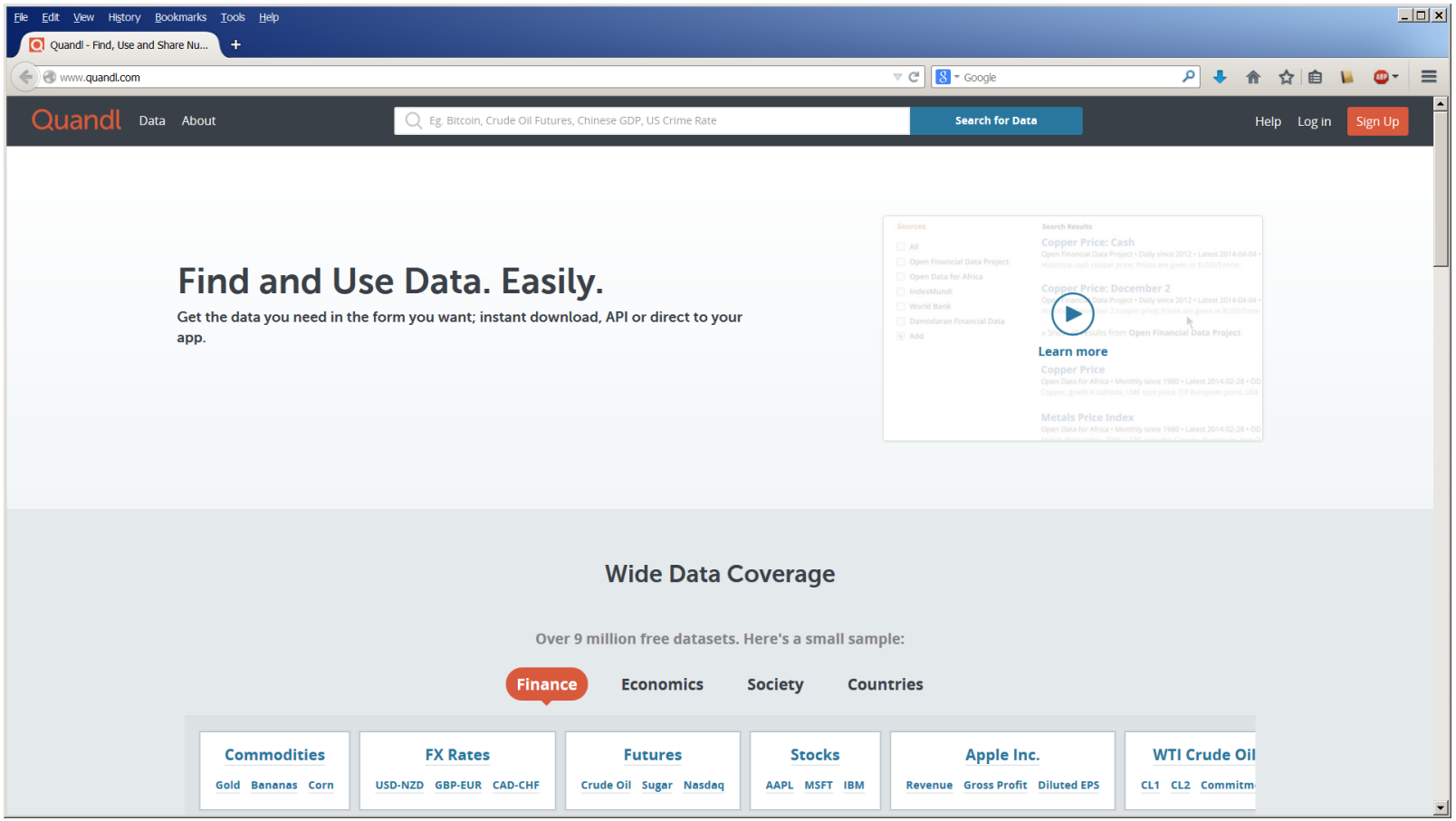


Outline

Quandl

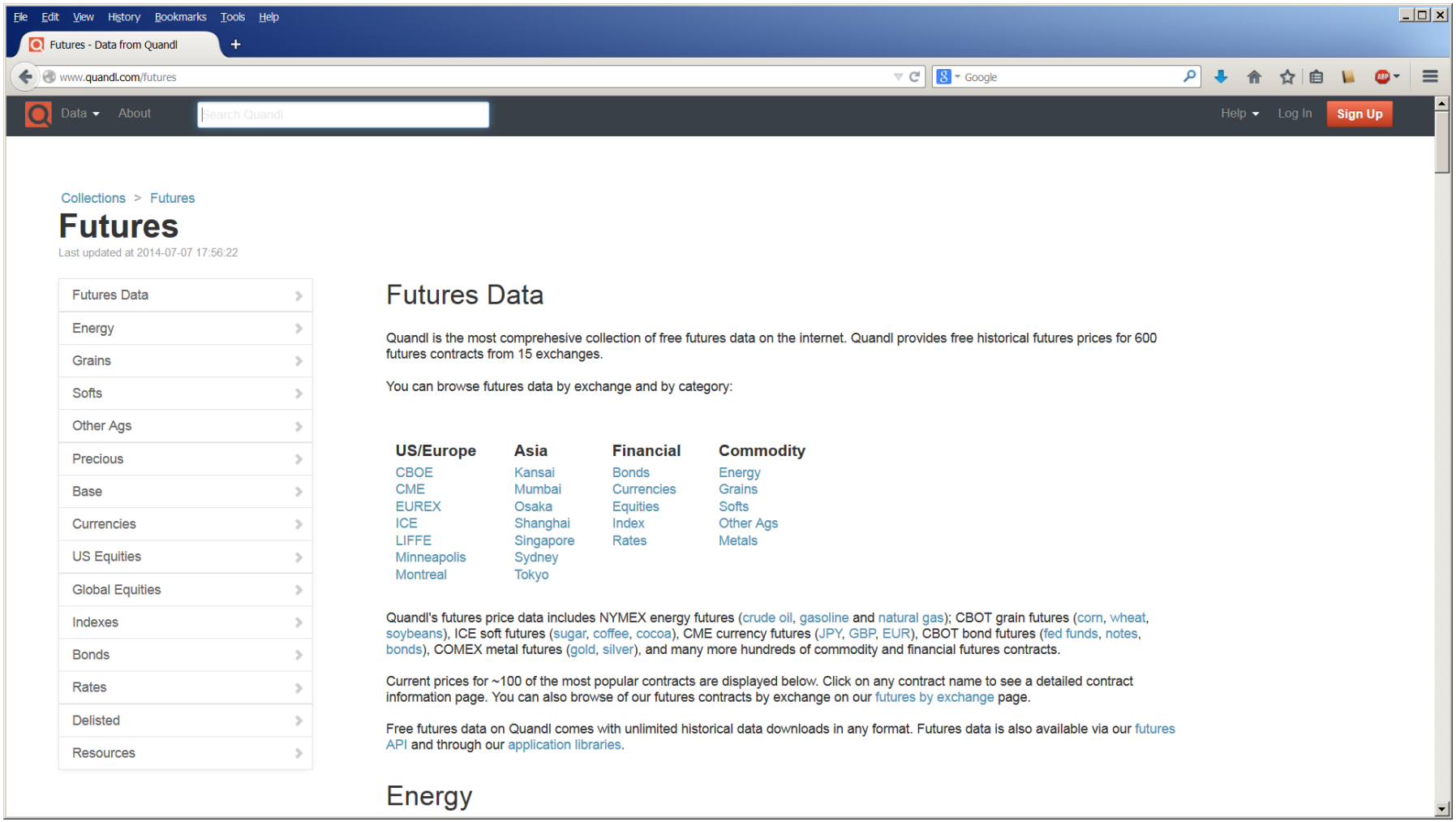
Tammer Kamel the founder of `www.quandl.com` wants to:

"do to Bloomberg what Wikipedia did to Britannica"



`http://www.quandl.com/`

Quandl futures



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

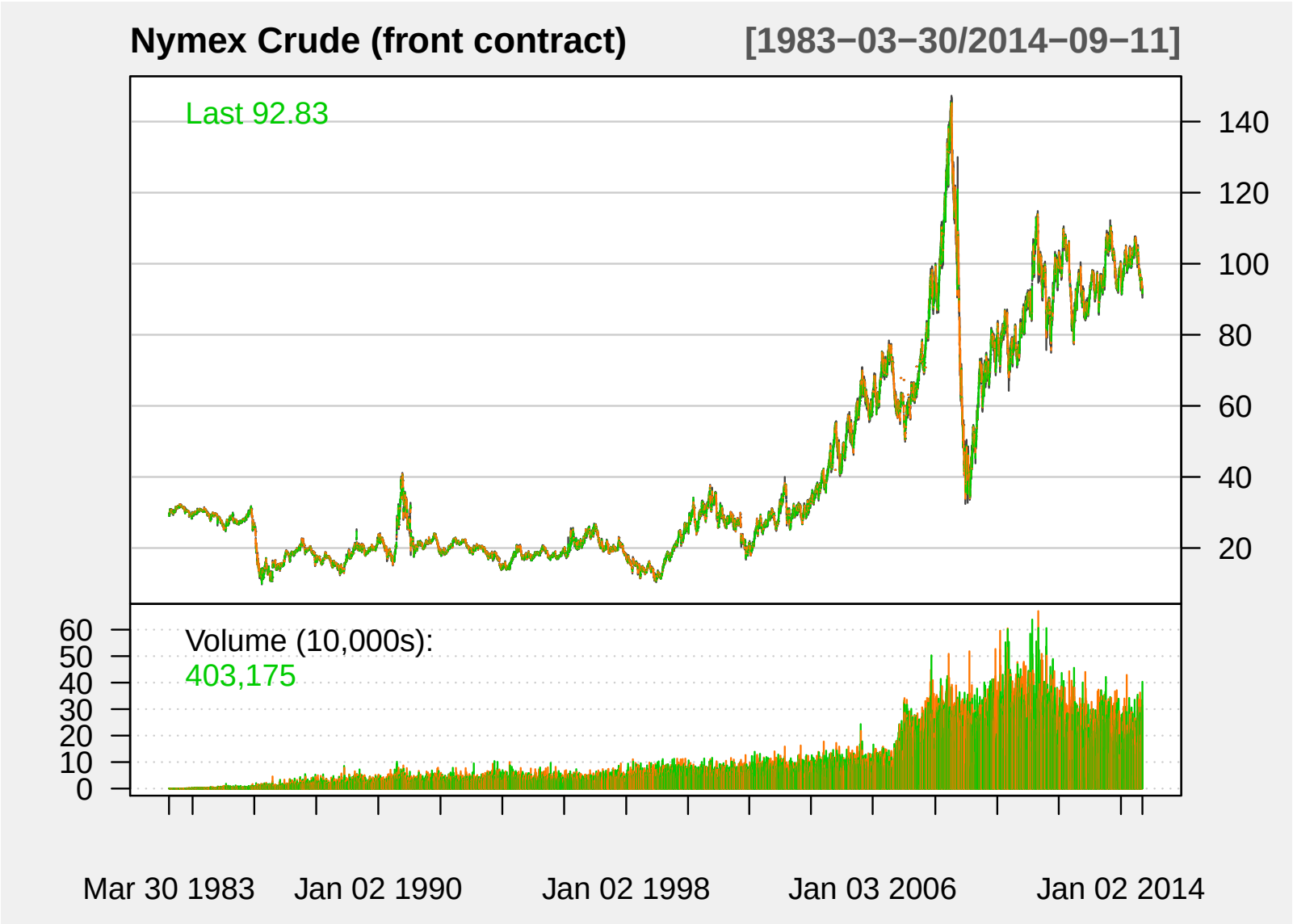
```
library(Quandl)
cl1 = Quandl("OFDP/FUTURE_CL1",type="xts")
class(cl1)

## [1] "xts" "zoo"

class(index(cl1))

## [1] "Date"

first(cl1)

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

last(cl1)

##             Open High   Low Settle Volume Open Interest
## 2015-04-13 51.81 53.1 51.47  51.91 375534        232250
```

Downloading data from Quandl

```
cl1 <- cl1[,c("Open","High","Low","Settle","Volume")]
colnames(cl1) <-  c("Open","High","Low","Close","Volume")
sum(is.na(coredata(cl1)))

## [1] 0

sum(coredata(cl1)<0.01)

## [1] 21

cl1[cl1 < 0.1] <- NA
cl1 <- na.approx(cl1)
chartSeries(cl1, 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:

get.hist.quote	Download historical financial data from a given data provider over the WWW (Yahoo, Oanda)
jarque.bera.test	Tests the null of normality for x using the Jarque-Bera test statistic
adf.test	Computes the Augmented Dickey-Fuller test for the null that x has a unit root
po.test	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:

`instrument`	Yahoo ticker
`quote`	Open, High, Low, Close, Volume, Adjusted Close (O,H,L,C,V,A)
`start`	Starting date
`end`	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"
```

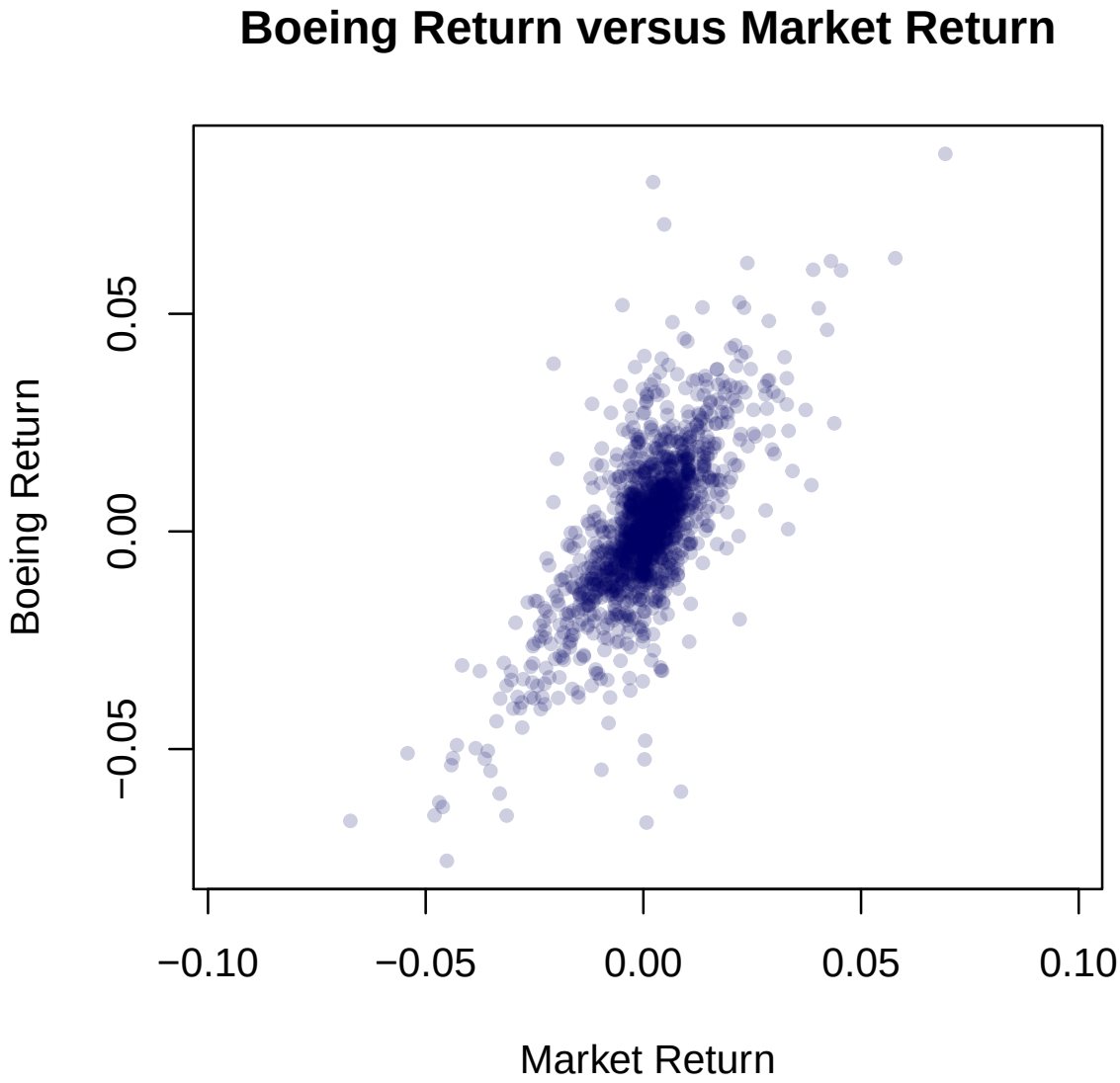
```
tail(SPY,3)
```

```
##            AdjClose
## 2015-04-09   208.90
## 2015-04-10   210.04
## 2015-04-13   209.09
```

```
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`



Outline

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

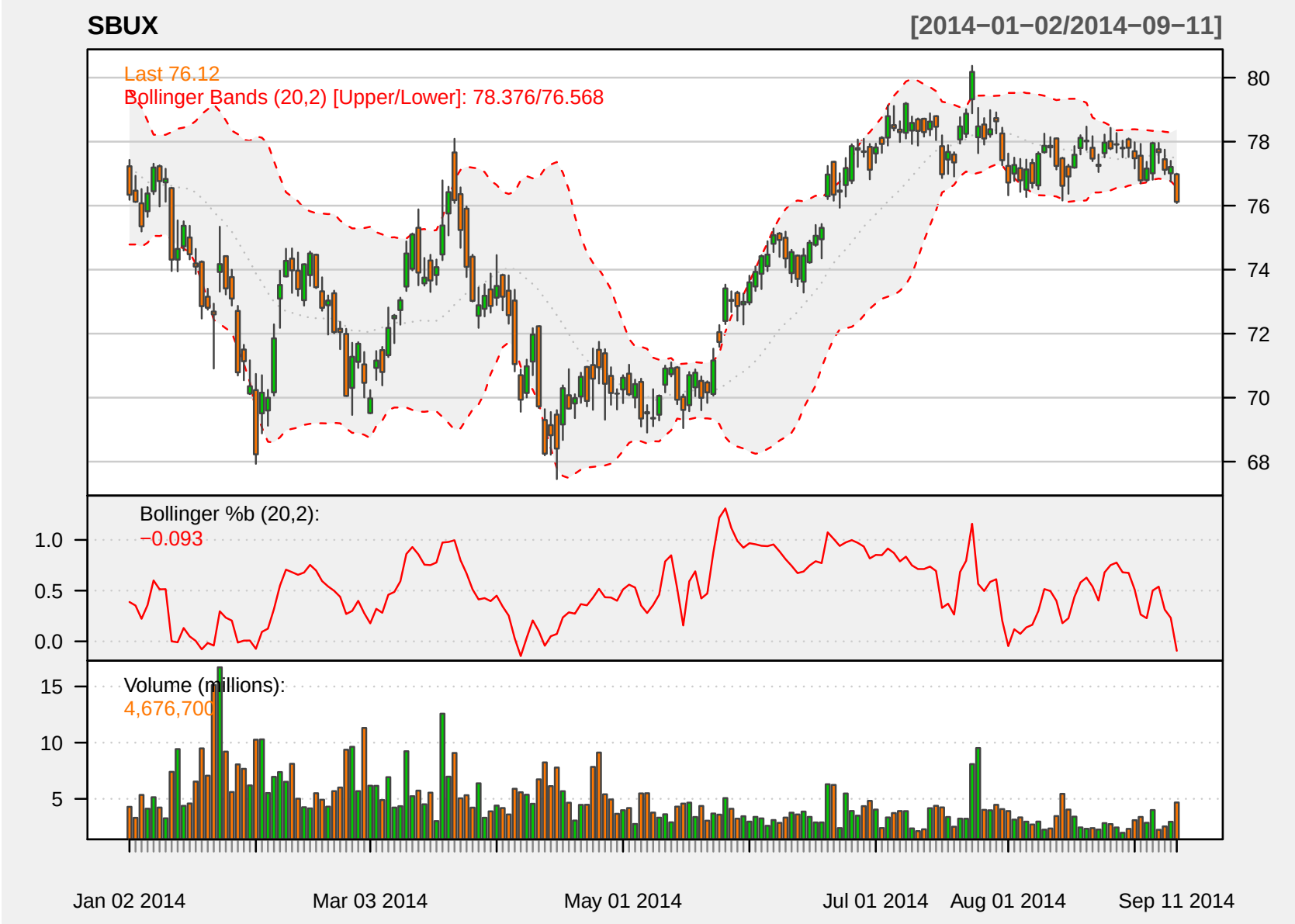
```
b <- BBands(HLC=HLC(SBUX["2014"]), n=20, sd=2)
tail(b,10)
```

```
##                    dn      mavg        up       pctB
## 2014-12-17 38.681546 40.369067 42.056589 0.32146872
## 2014-12-18 38.906612 40.432420 41.958227 0.36123600
## 2014-12-19 39.063809 40.471029 41.878249 0.23020859
## 2014-12-22 39.123265 40.491039 41.858813 0.31952497
## 2014-12-23 39.168590 40.520598 41.872605 0.53678614
## 2014-12-24 39.211853 40.546586 41.881320 0.50886455
## 2014-12-26 39.310603 40.598065 41.885527 0.57303654
## 2014-12-29 39.333015 40.624386 41.915756 0.61534746
## 2014-12-30 39.362116 40.653363 41.944611 0.60606675
## 2014-12-31 39.428619 40.703763 41.978907 0.64754929
```

```
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)

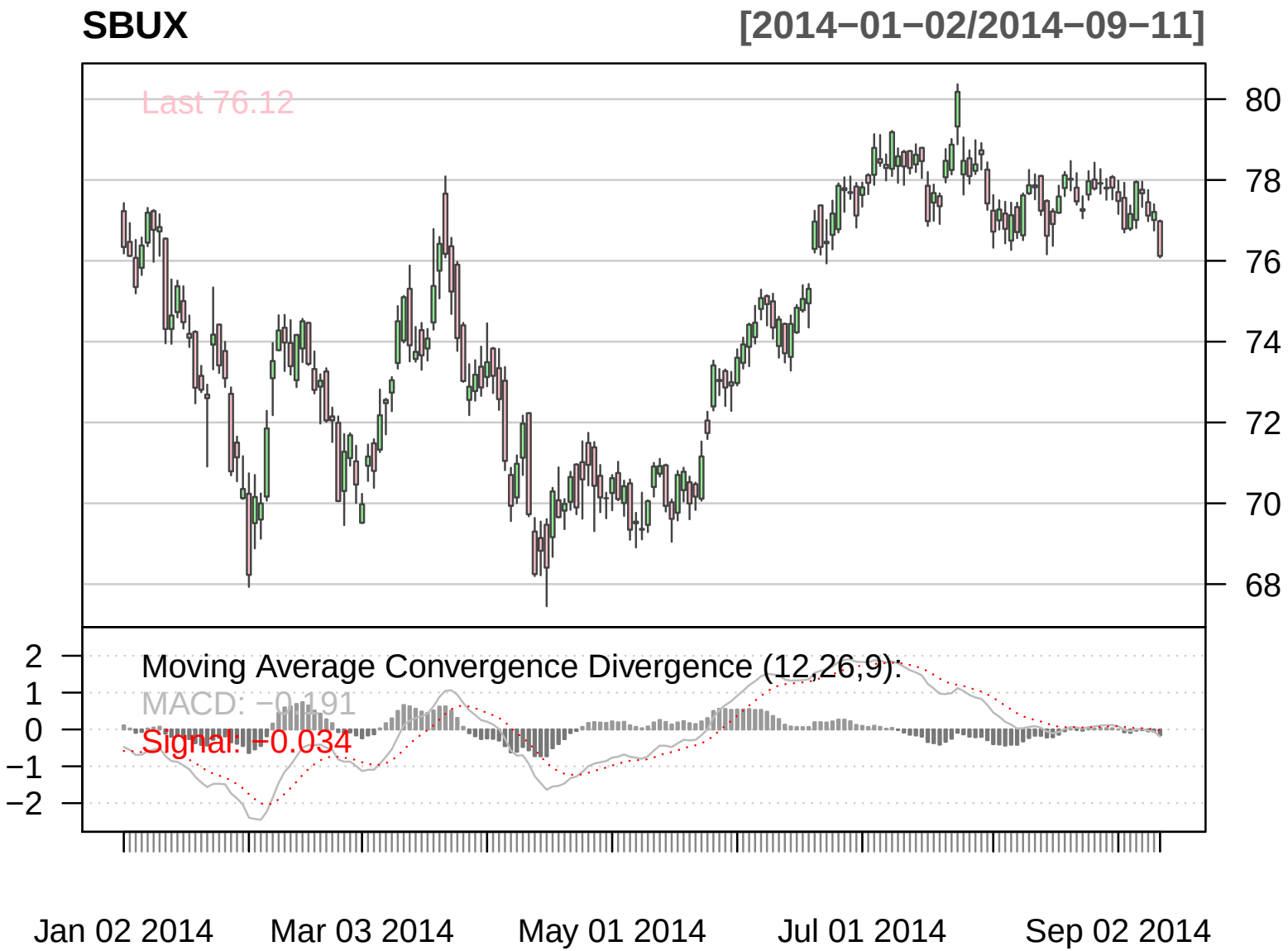
```
macd  <- MACD( Cl(SBUX), 12, 26, 9, maType="EMA" )
tail(macd)

##                  macd     signal
## 2015-04-06 0.73056871 1.19986211
## 2015-04-07 0.60791984 1.08147366
## 2015-04-08 0.60259954 0.98569884
## 2015-04-09 0.64950324 0.91845972
## 2015-04-10 0.71386620 0.87754101
## 2015-04-13 0.81106429 0.86424567

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

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

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

$$\text{MACD histogram} = \text{MACD line} - \text{Signal line}$$

Moving Average Convergence-Divergence (MACD)



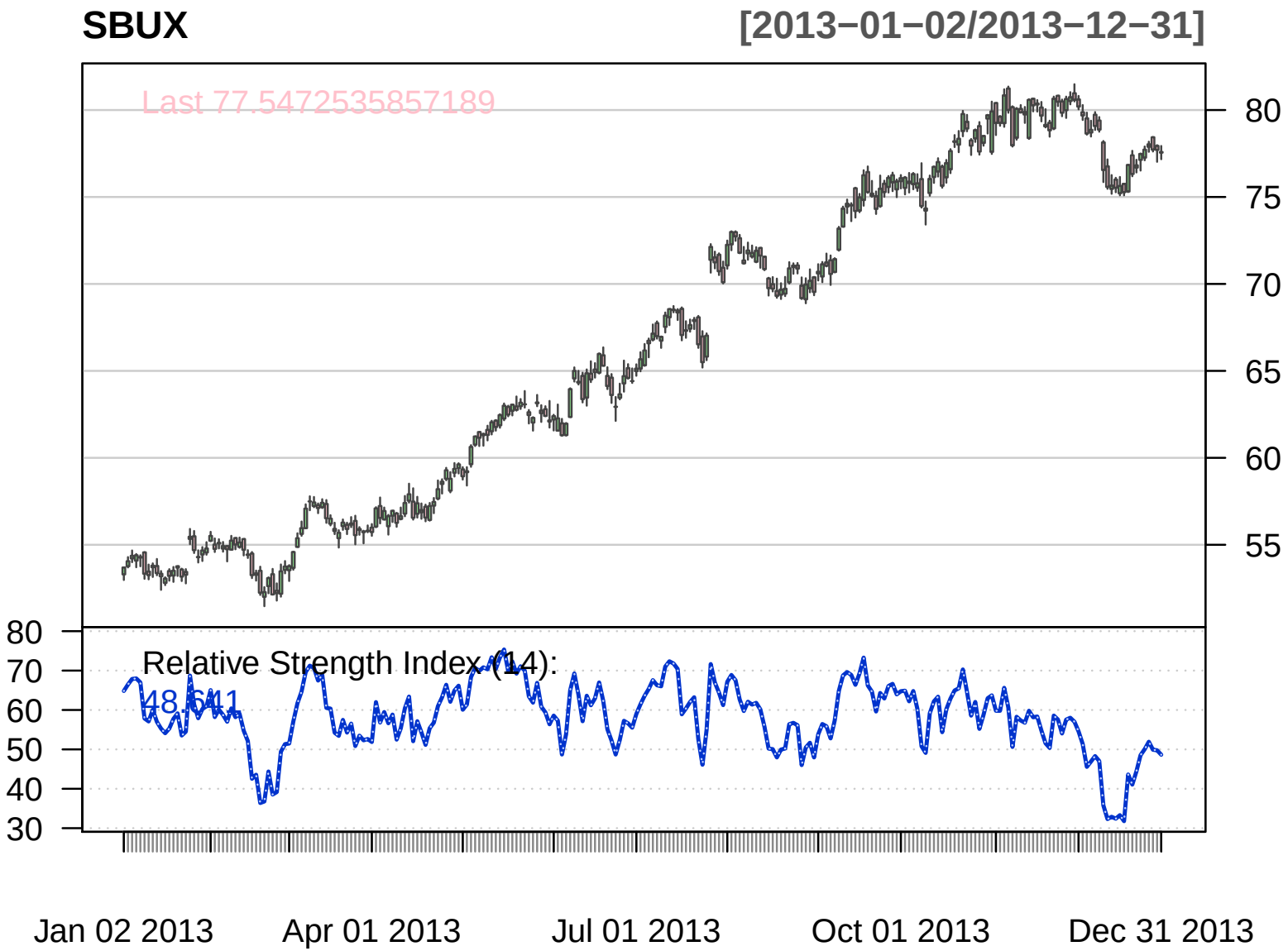
Relative Strength Index (RSI)

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

##            SBUX.Close.EMA.14
## 2015-04-06         51.802377
## 2015-04-07         49.828378
## 2015-04-08         54.628082
## 2015-04-09         57.248005
## 2015-04-10         58.807225
## 2015-04-13         61.201869

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

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

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

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

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`	`POSIXlt`

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)
```

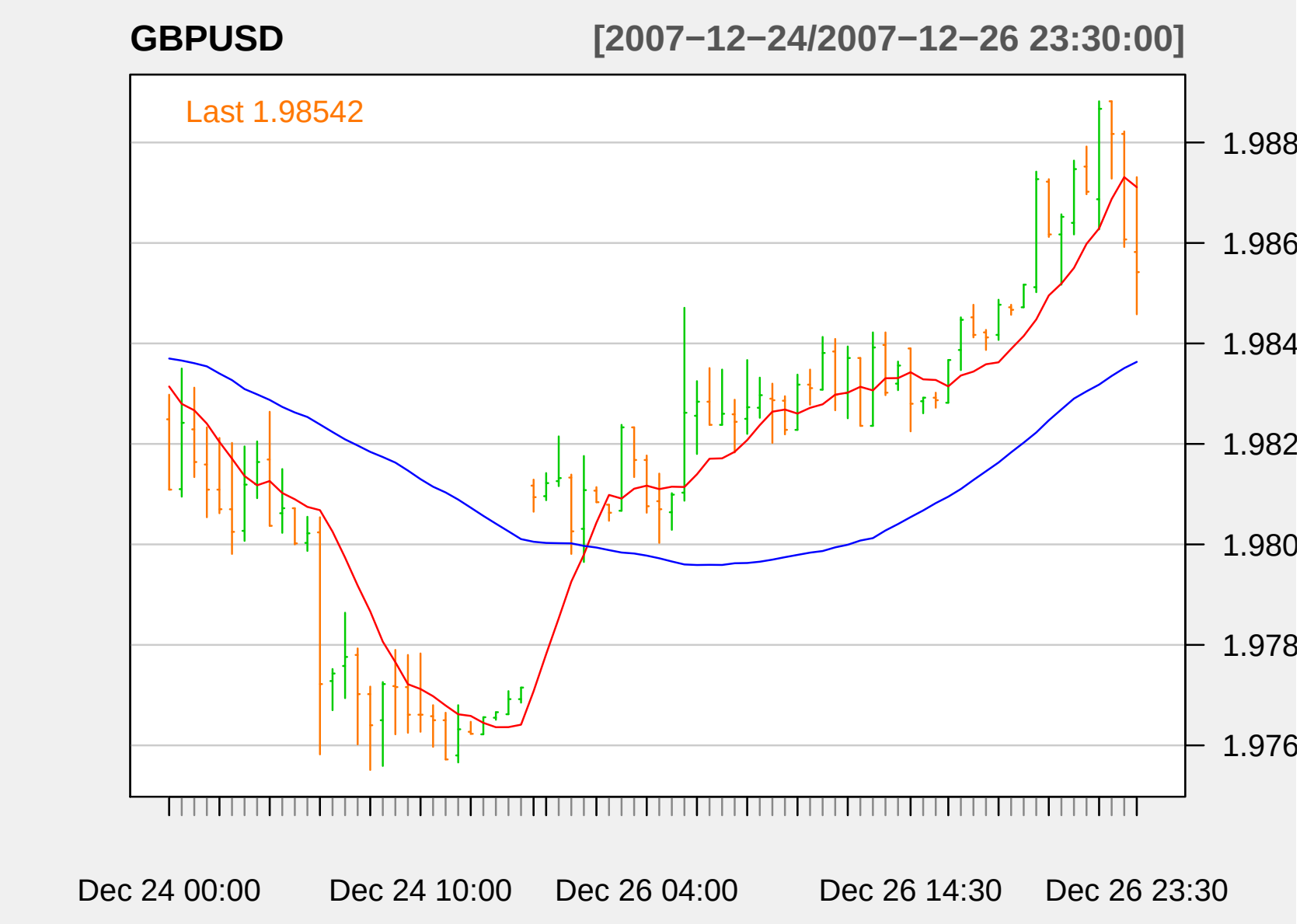
```
## [1] "2002-10-20 23:30:00 PDT" "2002-10-21 00:00:00 PDT" "2002-10-21 00:30:00 PDT"
## [4] "2002-10-21 01:00:00 PDT" "2002-10-21 01:30:00 PDT" "2002-10-21 02:00:00 PDT"
```

- 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(Cl(GBP),n=7)
slowMA <- SMA(Cl(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

Outline

R code in lecture notes

- The lecture slides are prepared with using knitr:

  $$\text{knitr} = \mathbf{R} + \LaTeX$$

- All of the code in the slides actually ran and produced the output shown in the lecture slides
- The associated R files can be run to reproduce all of the results, graphs, tables, etc.
- Please download and study these R scripts yourself

Wrap up

- Homework
  - Assignment #2 due Thursday
  - Assignment #3 will be posted Wednesday
- Reading
  - Tomasini/Jaekle Chapter 1-2
  - Dark Pools Chapters 18-25
- Next lecture
  - Introduction to trading systems
- Questions, comments, concerns
  - Post to the discussion forum on Canvas
  - Xin, chenx26@uw.edu
  - Guy, gyollin@uw.edu

COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

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