Trading Geeks

World Markets

.DJI 16,102.38 -272.38 -1.66% HSI 21,259.04 +675.52 3.28%

Technical Analysis with R

Technical Analysis with R

Posted on July 14, 2014 by TradingGeek — 6 Comments ↓

Table Of Content

In this post we'll take a look at how a trader could use R to calculate some basic Technical Analysis indicators. R is a free open-source statistical analysis environment and programming language. It is available for Windows, Mac OS, and Linux operating systems. Installation is easy and quick. For download and installation instructions go to: http://cran.r-project.org.

When developing a trading strategy it's useful to be able to analyze and visualize data and to be able to test your trade-generation rules and their variations and models quickly and with minimum turn-around. While many trading platforms, such as Interactive Brokers, etc.. provide access to historical data via API or straight file download – analyzing that data and prototyping trading strategies often requires writing hundreds of lines of code in programming languages such as Java or C++, or writing cumbersome difficult-to-test formulas in Excel. This requires a significant time investment, regardless of how experience programmer you are. By contrast, a higher-level programming language such as R or Matlab, coupled with their interactive programming environments, allow their users to slice, dice, and analyze data within a fraction of time it takes with C++, C#, or Java. The amount of code required to develop a trading strategy in R is typically an order of magnitude less as well.

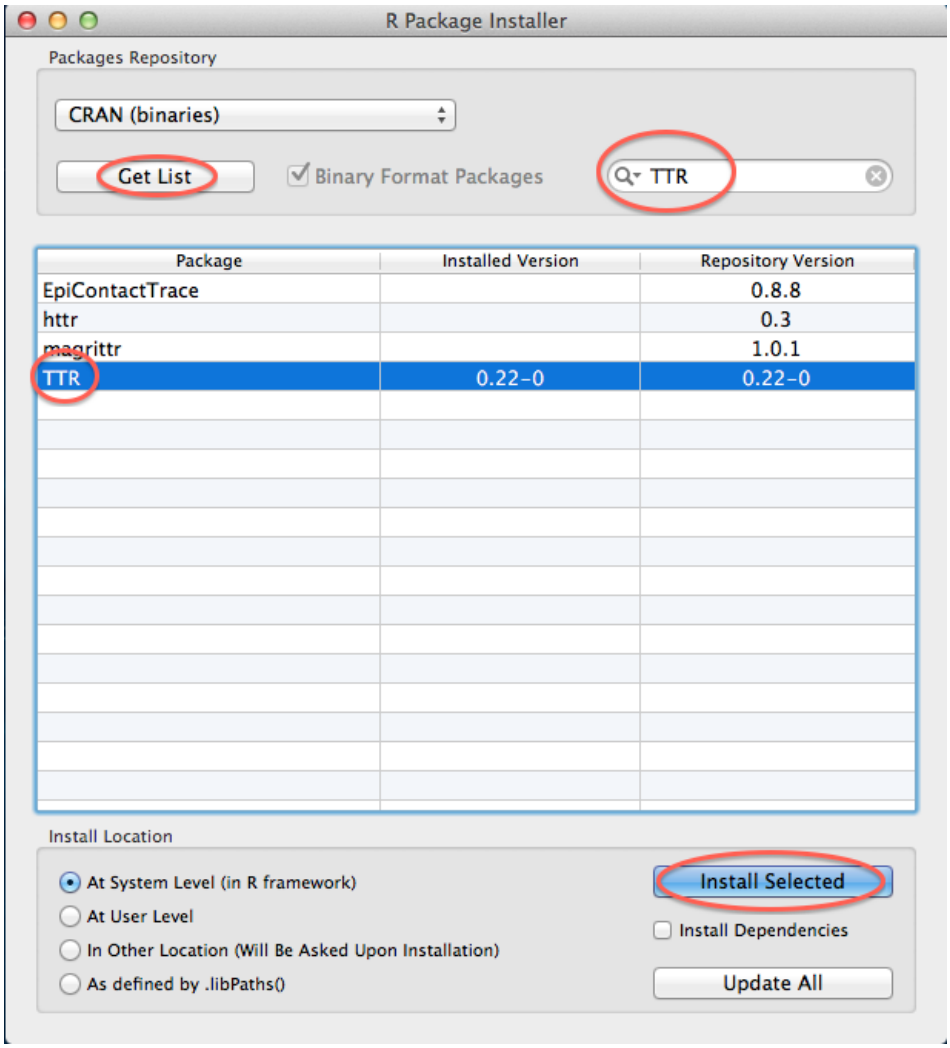
In this example we'll use a simple comma-separate file containing open, high, low, and close price columns (a.k.a. OHLC), along with volume and timestamp values for SPY ETF. In this post we'll demonstrate how to use a free R library to calculate Simple Moving Average (SMA), Exponential Moving Average (EMA), Bollinger Bands (BBands), RSI, and MACD technical analysis indicators. We will append calculated indicators as new columns to our input file so that it can be used for further analysis or trading strategy prototyping in Excel, R, or any other CSV-friendly software package of your choice.

Installing Technical Analysis library for R

1. To calculate Technical Analysis with R we will be using a free open-source library called "TTR" (Technical Trading Rules). This step includes instructions for installing TTR library, assuming you already have installed R on your computer. This steps only needs to be performed once per R installation on a computer.

To install the library on your computer:

1) Start R environment on your computer, then in the menu select: Packages & Data -> Package Installer
2) In Package Installer type "TTR" in the Package Search field, and click "Get List" button.
3) Select package "TTR" and click "Install Selected".



Loading Historical Data (Input)

For demo purposes we will use daily historical prices for SPY ETF from September 2013 through May 2014. Click here to download the data file. This input file for this example was generated using IB Historical Data Downloader.

2. We are going to start off by opening R shell and loading "TTR" library, which is a free R extension that contains functions for calculating some of the most common indicators.

```
> library("TTR")
```

3. The next step is to import our data file with historical prices into R environment. We will load data from sample CSV file into R environment and store it a "data frame", which an R variable type for storing data in table format in memory.

```
> data = read.csv(file="spy_historical_data.txt")
```

To display first few rows of the "data" table:

```
> head(data)
```

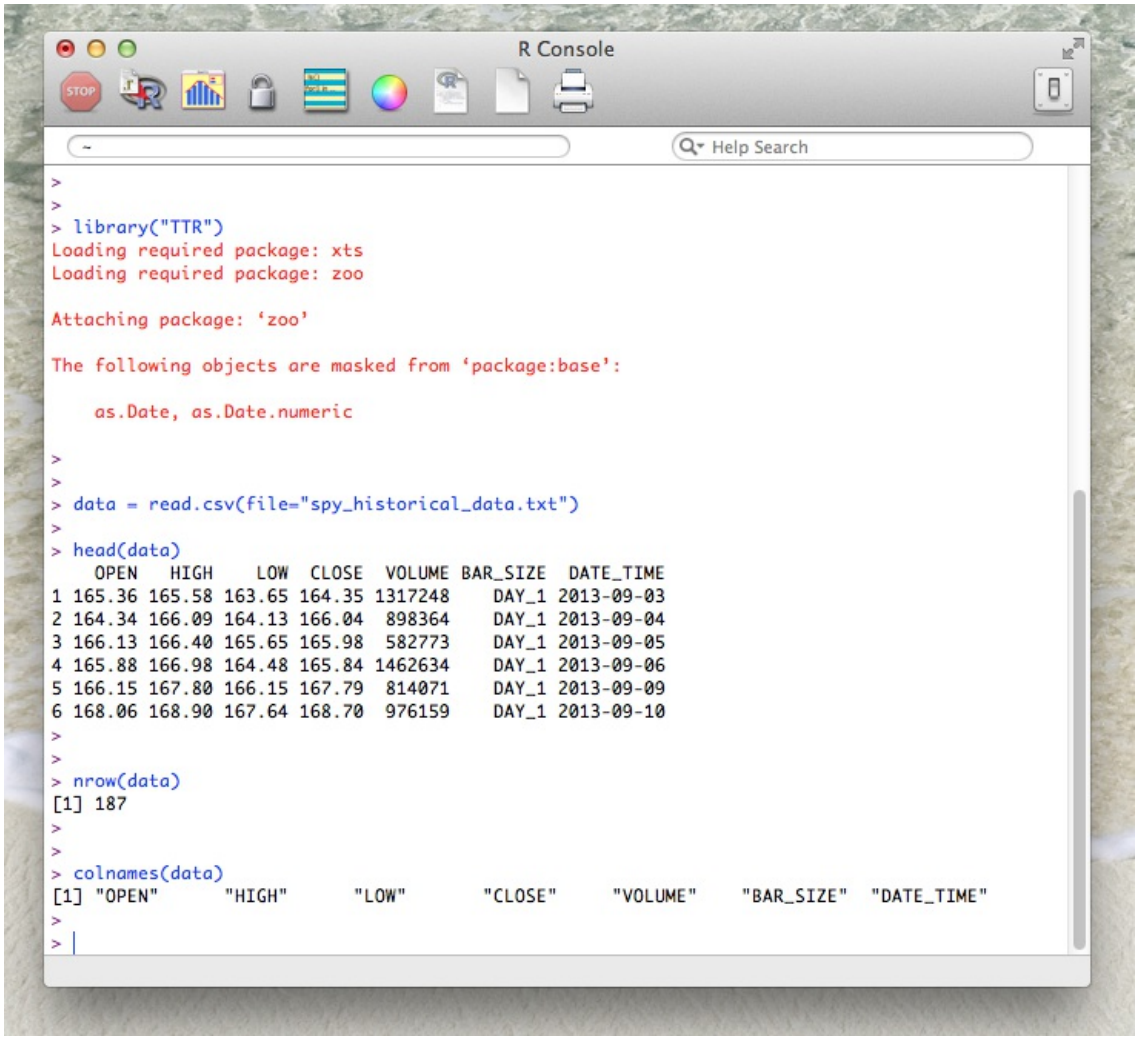
This by default shows first 6 rows of data along with column names (table header). To see how many rows you have in the "data" table:

```
> nrow(data)
[1] 187
```

This shows we have 187 data records in our SPY data file, for 187 trading days between Sep 3, 2013 – May 31, 2014.

We can also list table column names using "colnames" functions as follows:

```
> colnames(data)
```



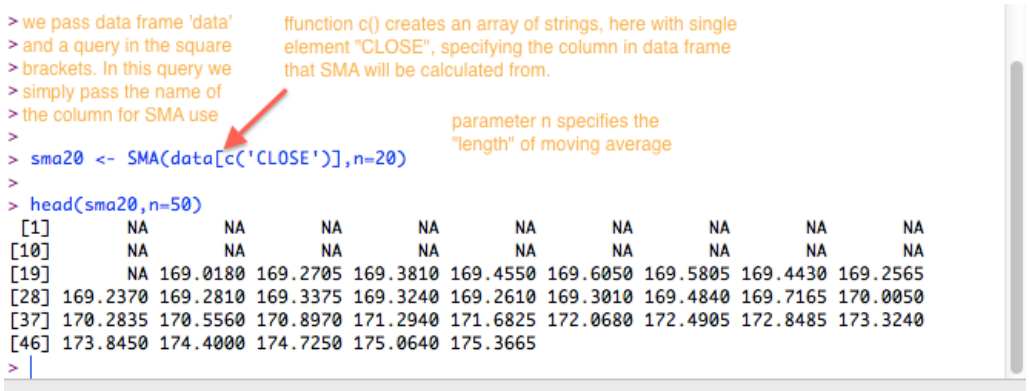
Moving Averages

4. Let's now calculate 20-day Simple Moving Average (SMA) of the CLOSE price column using TTR library's R function "SMA":

```
> sma20 < - SMA(data,n=20)
```

Now, let's see first 50 values of the "sma20" array:

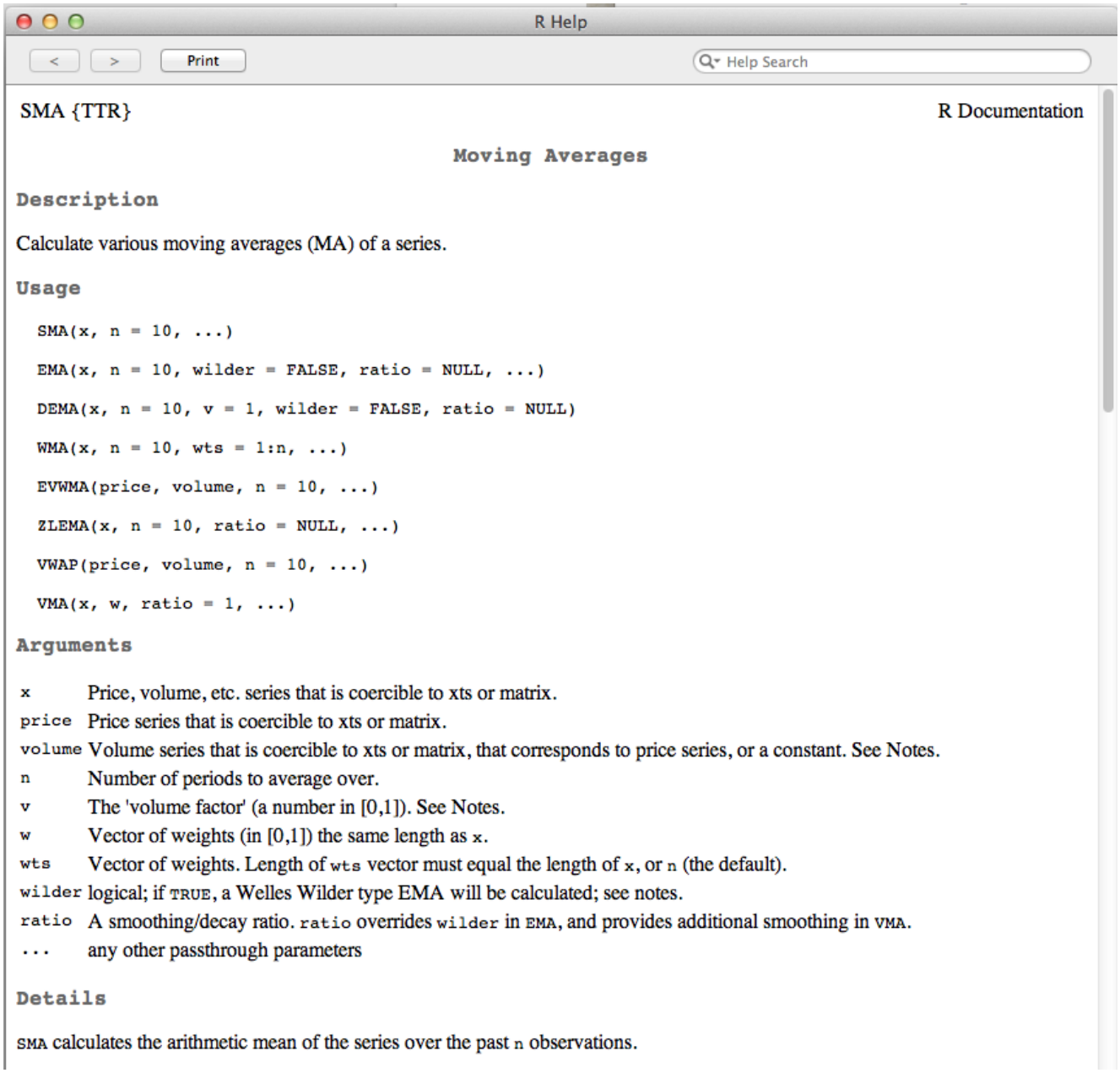
```
> head(sma20, n=50)
```



Here we used function SMA from TTR library we loaded above, telling it to calculate 20-day average (value of parameter "n"), of the "CLOSE" column from data frame "data". The function returns an array of SMA values and stores it in a new variable called "sma20".

You can bring up the help with a detailed description of the function and it's parameters using ? followed by the function name, as below. It is always a good idea to read help pages for the functions you are using, since they will list all optional parameters that you can use to tweak the output. Also, many functions have variations or related functions, which could be helpful in various circumstances and will be listed on the help page.

```
> ?SMA
```



R Help

Print | Help Search

SMA {TTR} R Documentation

Moving Averages

Description

Calculate various moving averages (MA) of a series.

Usage

```
SMA(x, n = 10, ...)

EMA(x, n = 10, wilder = FALSE, ratio = NULL, ...)

DEMA(x, n = 10, v = 1, wilder = FALSE, ratio = NULL)

WMA(x, n = 10, wts = 1:n, ...)

EVWMA(price, volume, n = 10, ...)

ZLEMA(x, n = 10, ratio = NULL, ...)

VWAP(price, volume, n = 10, ...)

VMA(x, w, ratio = 1, ...)
```

Arguments

x	Price, volume, etc. series that is coercible to xts or matrix.
price	Price series that is coercible to xts or matrix.
volume	Volume series that is coercible to xts or matrix, that corresponds to price series, or a constant. See Notes.
n	Number of periods to average over.
v	The 'volume factor' (a number in [0,1]). See Notes.
w	Vector of weights (in [0,1]) the same length as x.
wts	Vector of weights. Length of wts vector must equal the length of x, or n (the default).
wilder	logical; if TRUE, a Welles Wilder type EMA will be calculated; see notes.
ratio	A smoothing/decay ratio. ratio overrides wilder in EMA, and provides additional smoothing in VMA.
...	any other passthrough parameters

Details

SMA calculates the arithmetic mean of the series over the past n observations.

5. Calculating Exponential Moving Average is similarly easy, just use a different function, this time EMA(). Notice that we calculate EMA for 14-period length

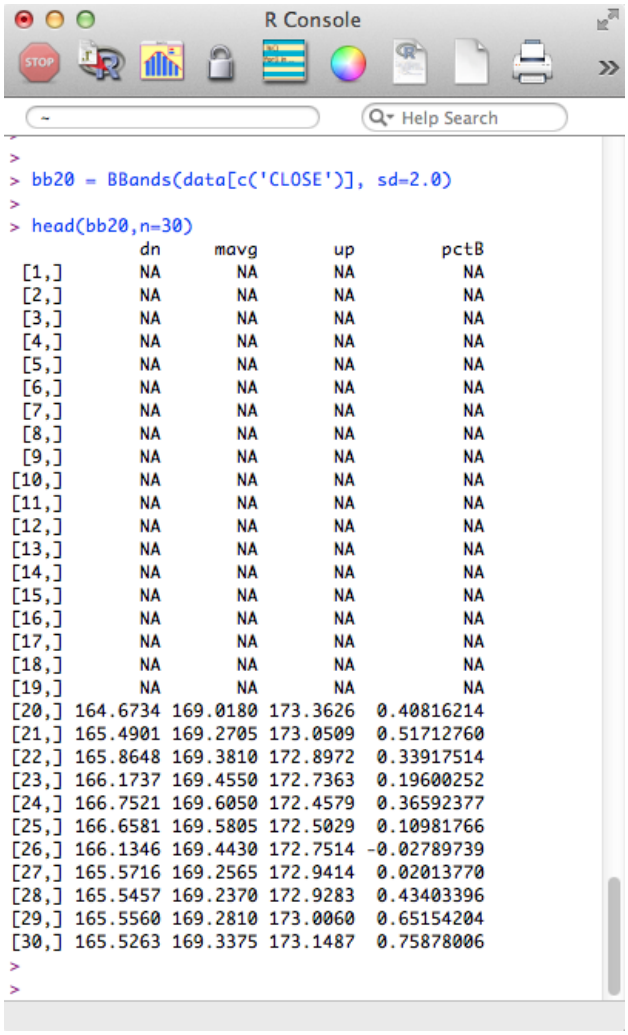
```
>
>
> ema14 = EMA(data[c('CLOSE')],n=14)
>
> head(ema14,n=20)
 [1]       NA       NA       NA       NA       NA       NA       NA       NA       NA       NA
[10]       NA       NA       NA       NA 168.9107 169.0160 169.0872 169.1009 169.2128
[19] 169.1857 169.0570
>
>
> |
```

Bollinger Bands

6. To calculate Bollinger Bands indicator we use the BBands function. There is a number of optional parameters that it takes, so we'll provide several examples. In the example below we call BBands passing it data frame 'data' with a query that specifies that we want to use values from 'CLOSE' column, just as we've been doing above to SMA and EMA calculations above. Second parameter 'sd' takes the number of standard deviations for upper and lower bands. Since we don't pass value for 'n' – BBands uses 20-period moving average by default. The output contains several columns: 'dn' for "lower" band, 'mavg' for the moving average, 'up' for the "upper" band, and pctB, which quantifies a security's price relative to the upper and lower Bollinger Band, a detailed description of it can be found here.

- %B equals 1 when price is at the upper band
- %B equals 0 when price is at the lower band
- %B is above 1 when price is above the upper band
- %B is below 0 when price is below the lower band
- %B is above .50 when price is above the middle band (20-day SMA)
- %B is below .50 when price is below the middle band (20-day SMA)

```
> bb20 = BBands(data, sd=2.0)
```



6.1 Now we'd like to create a new data frame containing all input data from the 'data' frame, plus Bollinger Bands data we just calculated.

```
> dataPlusBB = data.frame(data,bb20)
```

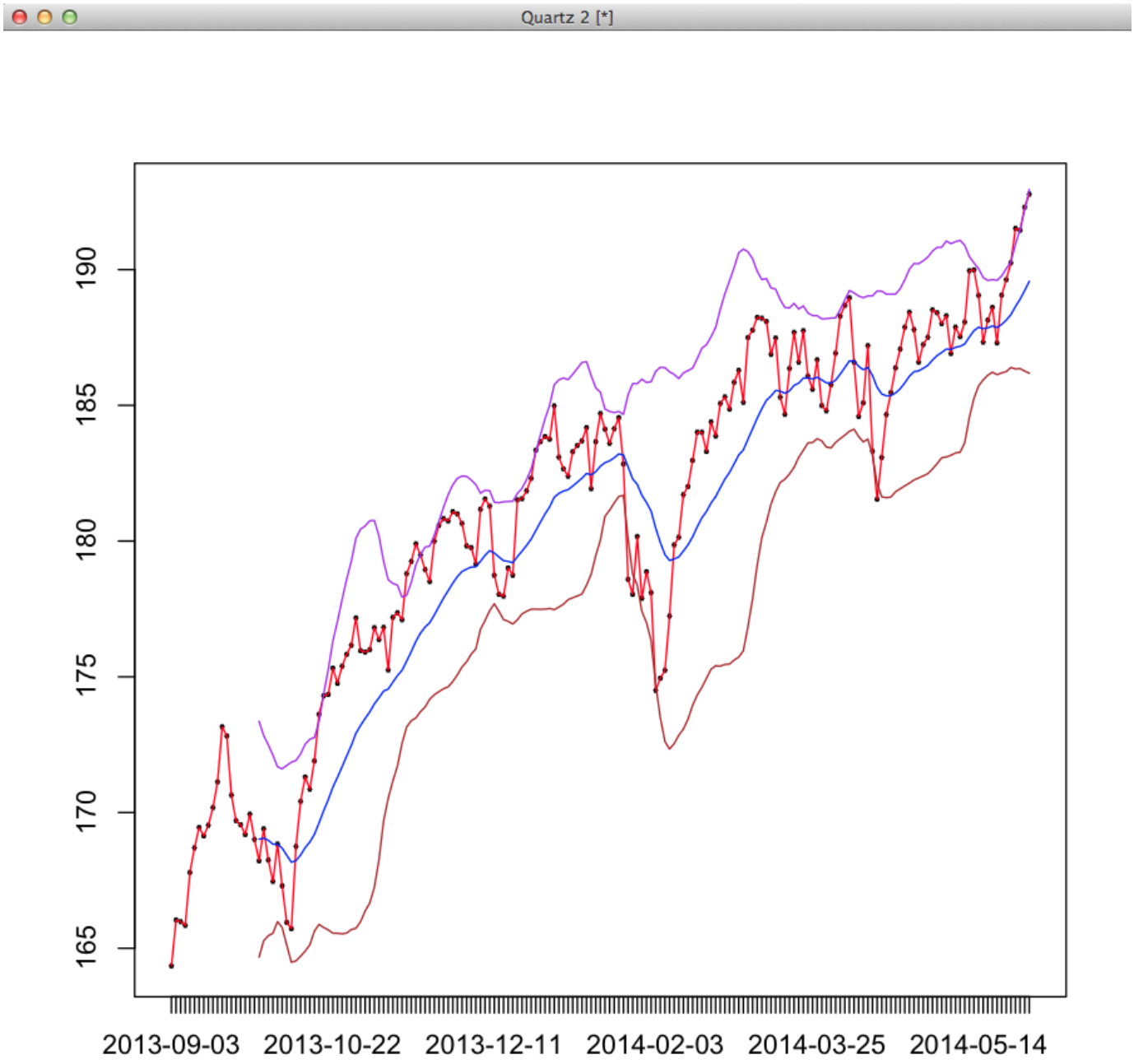
The data.frame() function takes any number of data frames and joins them row-wise into a new data frame, so that elements from corresponding rows are "joined" together in the result.

6.2 Bollinger Bands plot:

> plot(dataPlusBB$DATE_TIME,allData$CLOSE)
> lines(dataPlusBB$CLOSE, col = 'red')
> lines(dataPlusBB$up, col = 'purple')
> lines(dataPlusBB$dn, col = 'brown')
> lines(dataPlusBB$mavg, col = 'blue')

R Console

```
> dataPlusBB = data.frame(data,bb20)
>
>
> head(dataPlusBB, n=30)
     OPEN   HIGH    LOW  CLOSE  VOLUME BAR_SIZE  DATE_TIME       dn     mavg       up      pctB
1  165.36 165.58 163.65 164.35 1317248    DAY_1 2013-09-03       NA       NA       NA        NA
2  164.34 166.09 164.13 166.04  898364    DAY_1 2013-09-04       NA       NA       NA        NA
3  166.13 166.40 165.65 165.98  582773    DAY_1 2013-09-05       NA       NA       NA        NA
4  165.88 166.98 164.48 165.84 1462634    DAY_1 2013-09-06       NA       NA       NA        NA
5  166.15 167.80 166.15 167.79  814071    DAY_1 2013-09-09       NA       NA       NA        NA
6  168.06 168.90 167.64 168.70  976159    DAY_1 2013-09-10       NA       NA       NA        NA
7  168.64 169.61 168.35 169.45  826807    DAY_1 2013-09-11       NA       NA       NA        NA
8  169.35 169.74 168.72 169.14  752439    DAY_1 2013-09-12       NA       NA       NA        NA
9  169.03 169.59 168.58 169.53  657959    DAY_1 2013-09-13       NA       NA       NA        NA
10 171.00 171.61 169.34 170.18 1079380    DAY_1 2013-09-16       NA       NA       NA        NA
11 170.10 171.22 170.05 171.13  695315    DAY_1 2013-09-17       NA       NA       NA        NA
12 171.11 173.52 170.58 173.16 1703056    DAY_1 2013-09-18       NA       NA       NA        NA
13 173.46 173.86 172.59 172.82 1164763    DAY_1 2013-09-19       NA       NA       NA        NA
14 172.12 172.50 170.57 170.64 1236445    DAY_1 2013-09-20       NA       NA       NA        NA
15 171.04 171.17 169.39 169.70  936418    DAY_1 2013-09-23       NA       NA       NA        NA
16 169.65 170.53 169.21 169.55  859227    DAY_1 2013-09-24       NA       NA       NA        NA
17 169.33 169.98 168.89 169.19 1102872    DAY_1 2013-09-25       NA       NA       NA        NA
18 169.33 170.17 169.05 169.94  742390    DAY_1 2013-09-26       NA       NA       NA        NA
19 169.65 169.66 168.47 169.01  909434    DAY_1 2013-09-27       NA       NA       NA        NA
20 167.98 168.97 167.15 168.22 1365546    DAY_1 2013-09-30 164.6734 169.0180 173.3626 0.4081621
21 168.65 169.69 167.97 169.40 1107371    DAY_1 2013-10-01 165.2739 169.0544 172.8348 0.5457114
22 168.65 169.34 167.83 168.25 1015636    DAY_1 2013-10-02 165.4615 168.9778 172.4940 0.3965128
23 168.47 169.23 166.84 167.46 1611752    DAY_1 2013-10-03 165.5519 168.8332 172.1145 0.2907486
24 167.65 169.06 167.53 168.84  877451    DAY_1 2013-10-04 165.9810 168.8339 171.6867 0.5010746
25 167.41 168.90 167.01 167.30  877383    DAY_1 2013-10-07 165.7654 168.6878 171.6101 0.2625566
26 167.42 167.64 165.16 165.95 1641395    DAY_1 2013-10-08 165.1186 168.4270 171.7355 0.1256440
27 165.98 166.29 164.53 165.72 1570025    DAY_1 2013-10-09 164.4843 168.1692 171.8541 0.1676676
28 166.43 169.26 165.57 168.75 1758254    DAY_1 2013-10-10 164.5332 168.2245 171.9158 0.5711753
29 168.95 170.57 168.70 170.41  978501    DAY_1 2013-10-11 164.7076 168.4327 172.1577 0.7654092
30 169.54 171.42 168.96 171.31 1045432    DAY_1 2013-10-14 164.8956 168.7067 172.5179 0.8415357
>
>
> plot(dataPlusBB$DATE_TIME,allData$CLOSE)
> lines(dataPlusBB$CLOSE, col = 'red')
> lines(dataPlusBB$up, col = 'purple')
> lines(dataPlusBB$dn, col = 'brown')
> lines(dataPlusBB$mavg, col = 'blue')
>
>
```



6.3 Alternatively, we can specify explicitly what type of moving average should be used as the basis for Bollinger Bands using function parameter 'maType', which simply take a moving average function name. Refer to ?SMA help page to see different types of moving averages supported in TTR library. For example, if you'd like to calculate an EMA Bollinger Bands, you can pass EMA to maType. Notice that in this example we are overriding default length parameter for moving average, using 14-period average this time.

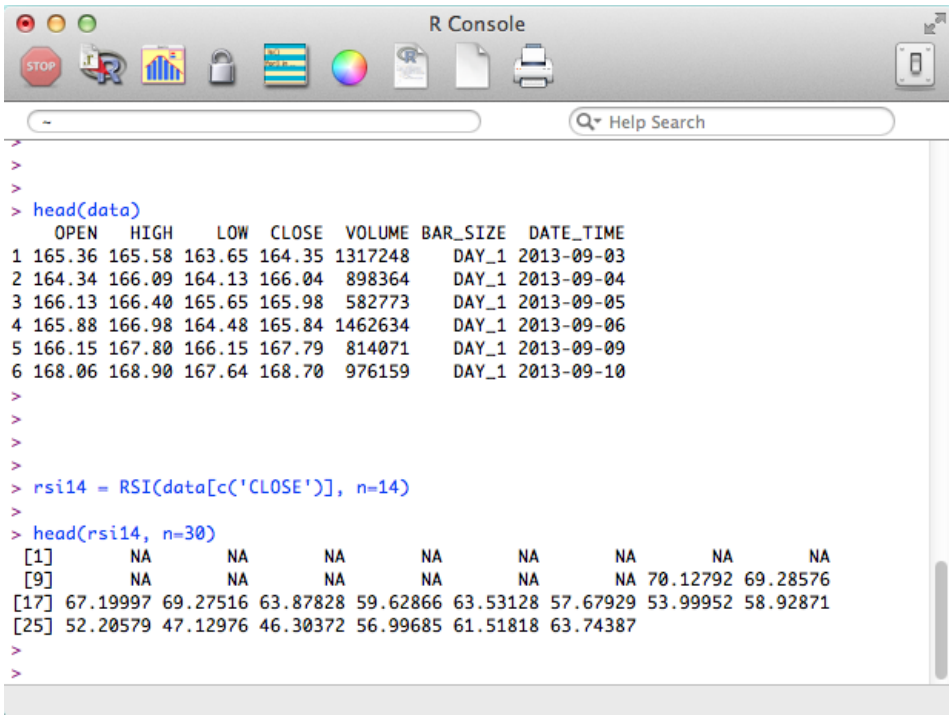
```
> bbEMA = BBands(data, sd=2.0, n=14, maType=EMA)
```

RSI – Relative Strength Indicator

7. RSI. To calculate RSI we use the RSI() function. You can use ?RSI command in R shell to get details for the function parameters. Basically, it's very similar to the functions we used above to generate moving averages. It has two required parameters: time series (such as 'CLOSE' column from our 'data' data frame, and 'n' integer value for the "length" of the RSI indicator.

```
> rsi14 = RSI(data, n=14)
```

Here the first parameter to RSI function is: data, which is a statement that says "take column named 'CLOSE' from the 'data' table, and return it as a list of values, and the second parameter is n=14, where the parameter name is 'n', and the value 14 indicates that we want to calculate 14-day RSI values on the close prices.



MACD

8. The MACD function takes several arguments:

1. input data series (such as 'CLOSE' price)
2. number of periods for "fast" moving average
3. number of periods for "slow" moving average
4. number of periods for the "signal" line

You can also optionally specify moving average function you want to use for MACD moving averages. See a screenshot of the help page below (you can also use ?MACD command in R shell to open the help page yourself):

R Help

< > Print Help Search

MACD {TTR} R Documentation

MACD Oscillator

Description

The MACD was developed by Gerald Appel and is probably the most popular price oscillator. The MACD function documented in this page compares a fast moving average (MA) of a series with a slow MA of the same series. It can be used as a generic oscillator for any univariate series, not only price.

Usage

```
MACD(x, nFast = 12, nSlow = 26, nSig = 9, maType,
  percent = TRUE, ...)
```

Arguments

x	Object that is coercible to xts or matrix; usually price, but can be volume, etc.
nFast	Number of periods for fast moving average.
nSlow	Number of periods for slow moving average.
nSig	Number of periods for signal moving average.
maType	Either: 1. A function or a string naming the function to be called. 2. A *list* with the first component like (1) above, and additional parameters specified as *named* components. See Examples.
percent	logical; if TRUE, the percentage difference between the fast and slow moving averages is returned, otherwise the difference between the respective averages is returned.
...	Other arguments to be passed to the maType function in case (1) above.

Let's calculate a standard (12,26,9) MACD indicator using this function. We'll be using standard simple moving averages, so, we'll specify SMA function in 'maType' parameter:

```
> macd = MACD(data, nFast=12, nSlow=26, nSig=9, maType=SMA)
```

R Console

~ Help Search

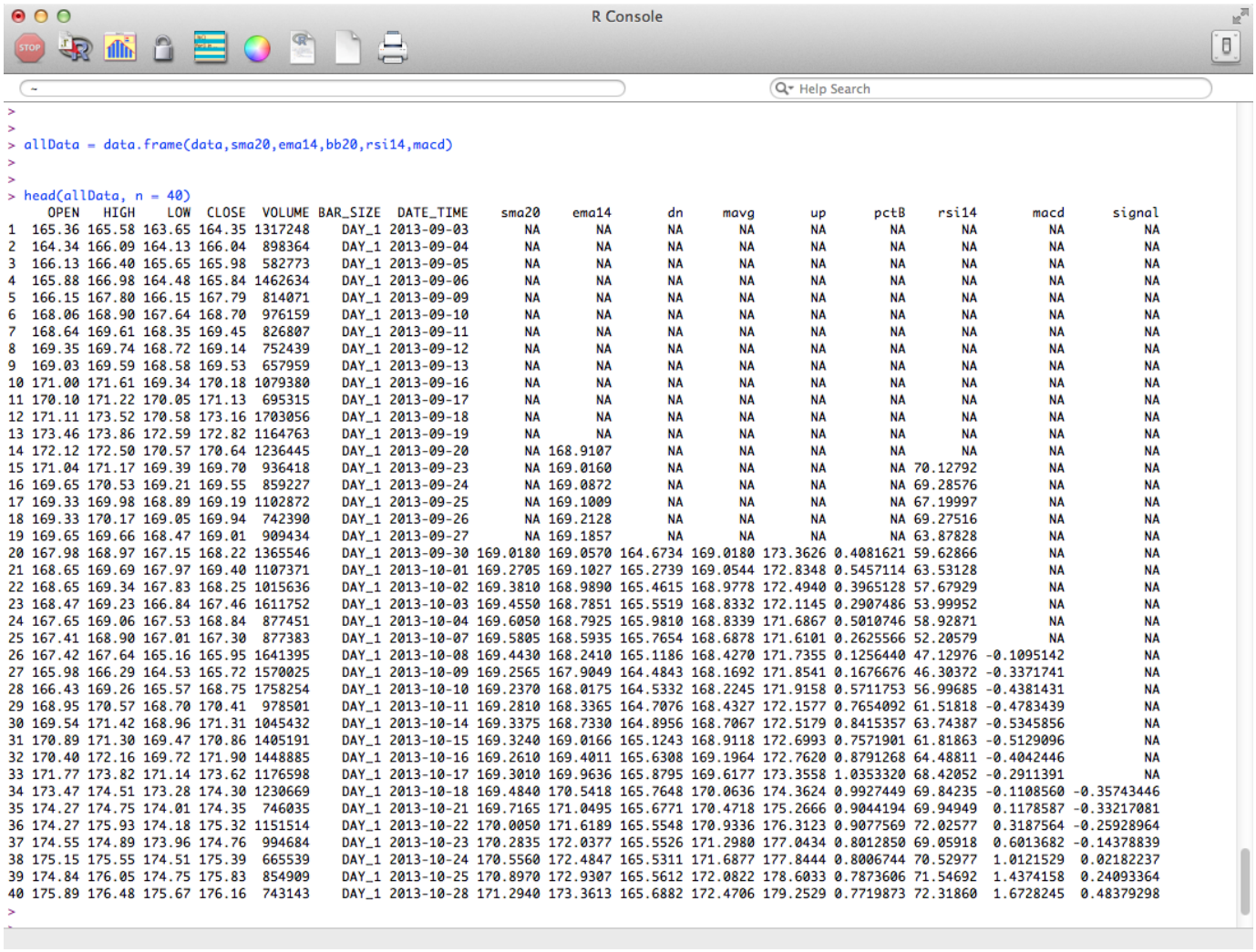
```
>
>
>
> macd = MACD(data[c('CLOSE')], nFast=12, nSlow=26, nSig=9, maType=SMA)
>
>
> tail(macd, n =10)
            macd    signal
[178,] 0.6311686 0.6339976
[179,] 0.5305574 0.6303503
[180,] 0.3802553 0.6081955
[181,] 0.2910297 0.5761129
[182,] 0.3096272 0.5464777
[183,] 0.3165816 0.5091552
[184,] 0.3878241 0.4707256
[185,] 0.4475288 0.4402350
[186,] 0.4603753 0.4172164
[187,] 0.4944879 0.4020297
>
>
>
```

Join All Data Together

9. Now, we join all of the indicators calculated above with the original input data into a single data frame:

```
> allData = data.frame(data,sma20,ema14,bb20,rsi14,macd)
```

The data.frame() function takes any number of data frames and joins them row-wise, so that elements from corresponding rows are “glued” together in the resulting data.frame ‘allData’.




```
> allData = data.frame(data,sma20,ema14,bb20,rsi14,macd)
>
>
> head(allData, n = 40)
     OPEN   HIGH    LOW  CLOSE  VOLUME BAR_SIZE  DATE_TIME     sma20     ema14       dn     mavg       up      pctB     rsi14       macd      signal
1  165.36 165.58 163.65 164.35 1317248    DAY_1 2013-09-03        NA        NA       NA       NA       NA        NA        NA         NA          NA
2  164.34 166.09 164.13 166.04  898364    DAY_1 2013-09-04        NA        NA       NA       NA       NA        NA        NA         NA          NA
3  166.13 166.40 165.65 165.98  582773    DAY_1 2013-09-05        NA        NA       NA       NA       NA        NA        NA         NA          NA
4  165.88 166.98 164.48 165.84 1462634    DAY_1 2013-09-06        NA        NA       NA       NA       NA        NA        NA         NA          NA
5  166.15 167.80 166.15 167.79  814071    DAY_1 2013-09-09        NA        NA       NA       NA       NA        NA        NA         NA          NA
6  168.06 168.90 167.64 168.70  976159    DAY_1 2013-09-10        NA        NA       NA       NA       NA        NA        NA         NA          NA
7  168.64 169.61 168.35 169.45  826807    DAY_1 2013-09-11        NA        NA       NA       NA       NA        NA        NA         NA          NA
8  169.35 169.74 168.72 169.14  752439    DAY_1 2013-09-12        NA        NA       NA       NA       NA        NA        NA         NA          NA
9  169.03 169.59 168.58 169.53  657959    DAY_1 2013-09-13        NA        NA       NA       NA       NA        NA        NA         NA          NA
10 171.00 171.61 169.34 170.18 1079380    DAY_1 2013-09-16        NA        NA       NA       NA       NA        NA        NA         NA          NA
11 170.10 171.22 170.05 171.13  695315    DAY_1 2013-09-17        NA        NA       NA       NA       NA        NA        NA         NA          NA
12 171.11 173.52 170.58 173.16 1703056    DAY_1 2013-09-18        NA        NA       NA       NA       NA        NA        NA         NA          NA
13 173.46 173.86 172.59 172.82 1164763    DAY_1 2013-09-19        NA        NA       NA       NA       NA        NA        NA         NA          NA
14 172.12 172.50 170.57 170.64 1236445    DAY_1 2013-09-20        NA 168.9107       NA       NA       NA        NA        NA         NA          NA
15 171.04 171.17 169.39 169.70  936418    DAY_1 2013-09-23        NA 169.0160       NA       NA       NA        NA 70.12792         NA          NA
16 169.65 170.53 169.21 169.55  859227    DAY_1 2013-09-24        NA 169.0872       NA       NA       NA        NA 69.28576         NA          NA
17 169.33 169.98 168.89 169.19 1102872    DAY_1 2013-09-25        NA 169.1009       NA       NA       NA        NA 67.19997         NA          NA
18 169.33 170.17 169.05 169.94  742390    DAY_1 2013-09-26        NA 169.2128       NA       NA       NA        NA 69.27516         NA          NA
19 169.65 169.66 168.47 169.01  909434    DAY_1 2013-09-27        NA 169.1857       NA       NA       NA        NA 63.87828         NA          NA
20 167.98 168.97 167.15 168.22 1365546    DAY_1 2013-09-30 169.0180 169.0570 164.6734 169.0180 173.3626 0.4081621 59.62866         NA          NA
21 168.65 169.69 167.97 169.40 1107371    DAY_1 2013-10-01 169.2705 169.1027 165.2739 169.0544 172.8348 0.5457114 63.53128         NA          NA
22 168.65 169.34 167.83 168.25 1015636    DAY_1 2013-10-02 169.3810 168.9890 165.4615 168.9778 172.4940 0.3965128 57.67929         NA          NA
23 168.47 169.23 166.84 167.46 1611752    DAY_1 2013-10-03 169.4550 168.7851 165.5519 168.8332 172.1145 0.2907486 53.99952         NA          NA
24 167.65 169.06 167.53 168.84  877451    DAY_1 2013-10-04 169.6050 168.7925 165.9810 168.8339 171.6867 0.5010746 58.92871         NA          NA
25 167.41 168.90 167.01 167.30  877383    DAY_1 2013-10-07 169.5805 168.5935 165.7654 168.6878 171.6101 0.2625566 52.20579         NA          NA
26 167.42 167.64 165.16 165.95 1641395    DAY_1 2013-10-08 169.4430 168.2410 165.1186 168.4270 171.7355 0.1256440 47.12976 -0.1095142          NA
27 165.98 166.29 164.53 165.72 1570025    DAY_1 2013-10-09 169.2565 167.9049 164.4843 168.1692 171.8541 0.1676676 46.30372 -0.3371741          NA
28 166.43 169.26 165.57 168.75 1758254    DAY_1 2013-10-10 169.2370 168.0175 164.5332 168.2245 171.9158 0.5711753 56.99685 -0.4381431          NA
29 168.95 170.57 168.70 170.41  978501    DAY_1 2013-10-11 169.2810 168.3365 164.7076 168.4327 172.1577 0.7654092 61.51818 -0.4783439          NA
30 169.54 171.42 168.96 171.31 1045432    DAY_1 2013-10-14 169.3375 168.7330 164.8956 168.7067 172.5179 0.8415357 63.74387 -0.5345856          NA
31 170.89 171.30 169.47 170.86 1405191    DAY_1 2013-10-15 169.3240 169.0166 165.1243 168.9118 172.6993 0.7571901 61.81863 -0.5129096          NA
32 170.40 172.16 169.72 171.90 1448885    DAY_1 2013-10-16 169.2610 169.4011 165.6308 169.1964 172.7620 0.8791268 64.48811 -0.4042446          NA
33 171.77 173.82 171.14 173.62 1176598    DAY_1 2013-10-17 169.3010 169.9636 165.8795 169.6177 173.3558 1.0353320 68.42052 -0.2911391          NA
34 173.47 174.51 173.28 174.30 1230669    DAY_1 2013-10-18 169.4840 170.5418 165.7648 170.0636 174.3624 0.9927449 69.84235 -0.1108560 -0.35743446
35 174.27 174.75 174.01 174.35  746035    DAY_1 2013-10-21 169.7165 171.0495 165.6771 170.4718 175.2666 0.9044194 69.94949  0.1178587 -0.33217081
36 174.27 175.93 174.18 175.32 1151514    DAY_1 2013-10-22 170.0050 171.6189 165.5548 170.9336 176.3123 0.9077569 72.02577  0.3187564 -0.25928964
37 174.55 174.89 173.96 174.76  994684    DAY_1 2013-10-23 170.2835 172.0377 165.5526 171.2980 177.0434 0.8012850 69.05918  0.6013682 -0.14378839
38 175.15 175.55 174.51 175.39  665539    DAY_1 2013-10-24 170.5560 172.4847 165.5311 171.6877 177.8444 0.8006744 70.52977  1.0121529  0.02182237
39 174.84 176.05 174.75 175.83  854909    DAY_1 2013-10-25 170.8970 172.9307 165.5612 172.0822 178.6033 0.7873606 71.54692  1.4374158  0.24093364
40 175.89 176.48 175.67 176.16  743143    DAY_1 2013-10-28 171.2940 173.3613 165.6882 172.4706 179.2529 0.7719873 72.31860  1.6728245  0.48379298
>
```

Write to text file

And, finally, we write contents of ‘allData’ data frame to a comma-separated values file. We use write.table() function, which contains a large number of optional parameters. A detailed help page is available using command “?write.table” in R shell.

```
> write.table(allData, file="spy_with_indicators.csv", na="", sep=",", row.names = FALSE)
```

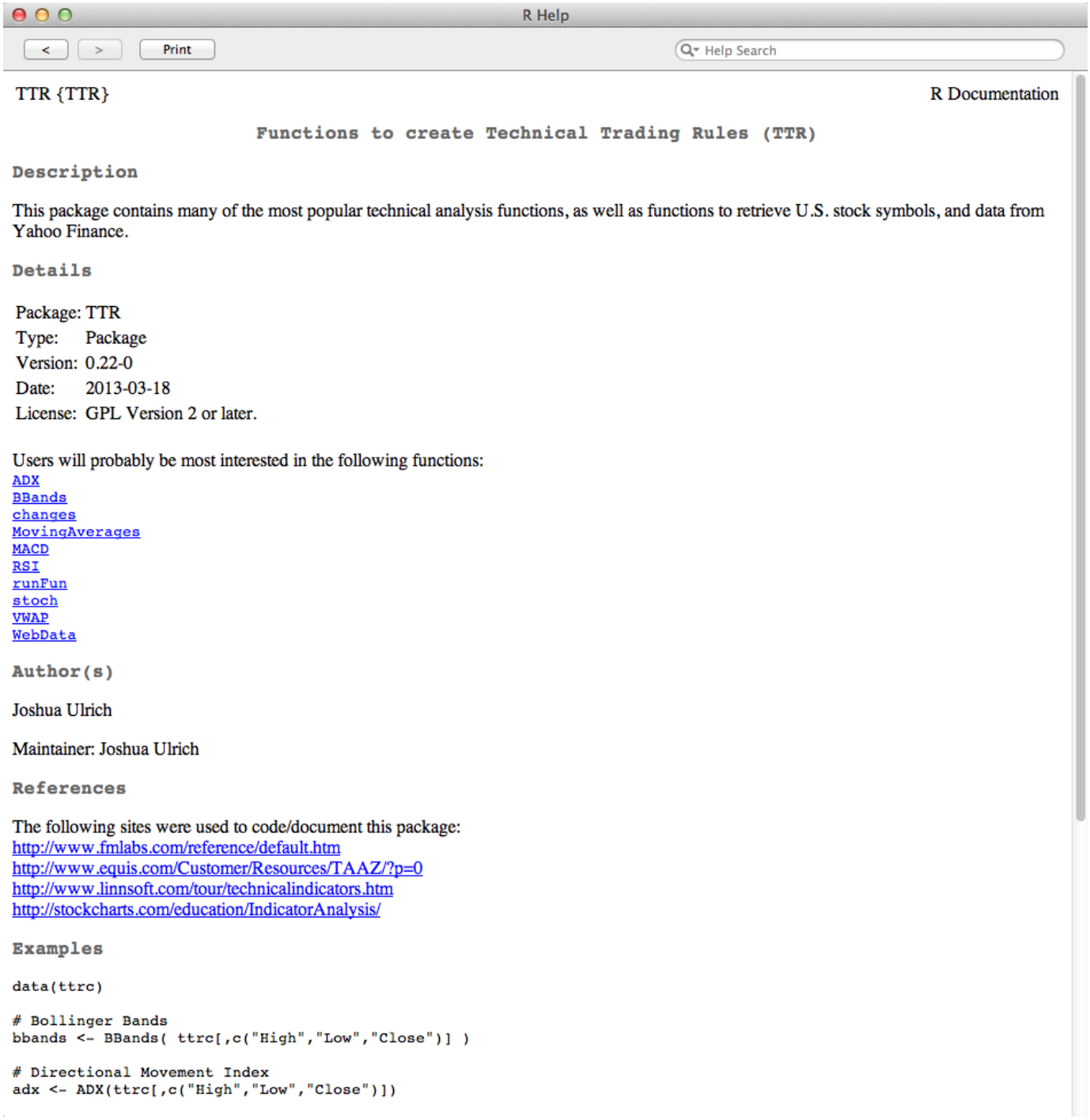
When we call write.table() function we pass the following arguments:

- allData – this is simply a reference to the data frame containing data to be written to the output file.
- file = “…” – this is the path and name of the file we are creating.
- na = “” – makes sure that cells in the data frame that contain R value “NA” will contain empty values in the output file. Some cells have NA for rows where there were not enough data to generate a corresponding indicator value (for example first 19 rows for 20-day SMA).
- sep = “,” – sets column separator to comma (hence comma-separated values file). To create a tab-separated file (really a preferred format for serious software systems) – use: sep = “\t”.
- row.names = FALSE – it is important to set this value, otherwise first column in the output file will contain row numbers.

The resulting file is available here. Right-click and select “Save Linked File As…” Downloaded file can be opened in Excel or text editor.

10. There are more functions and features available in the “TTR” library. You can find out more by bringing up TTR’s help page:

```
> ?TTR
```



R Help

Print

Help Search

TTR {TTR} R Documentation

Functions to create Technical Trading Rules (TTR)

Description

This package contains many of the most popular technical analysis functions, as well as functions to retrieve U.S. stock symbols, and data from Yahoo Finance.

Details

Package: TTR
Type: Package
Version: 0.22-0
Date: 2013-03-18
License: GPL Version 2 or later.

Users will probably be most interested in the following functions:
ADX
BBands
changes
MovingAverages
MACD
RSI
runFun
stoch
VWAP
WebData

Author(s)

Joshua Ulrich

Maintainer: Joshua Ulrich

References

The following sites were used to code/document this package:
http://www.fmlabs.com/reference/default.htm
http://www.equis.com/Customer/Resources/TAAZ/?p=0
http://www.linnsoft.com/tour/technicalindicators.htm
http://stockcharts.com/education/IndicatorAnalysis/

Examples

```
data(ttrc)

# Bollinger Bands
bbands <- BBands( ttrc[,c("High","Low","Close")] )

# Directional Movement Index
adx <- ADX(ttrc[,c("High","Low","Close")])
```

CONCLUSION

R provides a convenient and versatile environment for data analysis and calculations. In addition to thousands of free open-source statistical, mathematical libraries and algorithms, R contains a great number of functions and libraries for reading and writing data to/from files, databases, URLs, Web Services, etc… That, combined with the conciseness of the language, is a powerful combination that can help traders save precious time. Traders can significantly cut down the time required to prototype and backtest trading strategies using R. There are also methods to integrate R with mainstream programming languages such as Java and C++. Don't hesitate to post a comment or send as a message via Contact Us form if you have any questions regarding this material.

Finally, we'd like to mention a couple of books that have been very helpful in our development efforts. The first book – **"Quantitative Trading with R"** is a great mix of financial data analysis insights and application of R to backtesting, data exploration, and analysis. It has a number of great code examples and goes over a number of useful R packages. This is a good intro-to-intermediate level book for people who would like to build and backtest their own trading strategies.

The second book – **"Mastering R for Quantitative Finance"** – is a real gem. It contains more advanced information for traders with a good understanding of derivatives instruments and stronger mathematical background. We found that this book is a great follow up for the "Quantitative Trading with R". In addition to great R code samples and packages it contains overviews of a number of advanced (and practical!) quantitative finance models and algorithms, and lets you get your feet wet with R code straight away.

Trading Geeks provides consulting services in trading strategy and software development for independent traders, partnerships, and hedge funds. Please inquire for more information or a free quote for your project via Contact Us form on the right.

‹ Technical Analysis in Excel: Part II – MACD and RSI

Calculating Correlations of Forex Currency Pairs in Python ›

Posted in Data Analysis, Trading Tools

6 comments on "Technical Analysis with R"

Egor Ufimtsev says:
November 18, 2014 at 12:26 am

Great post! Thank you.

Reply

Bill Toomey says:
November 25, 2014 at 7:20 am

1) can you use the downloaded data to make charts, with the indicators or oscillators?
2) can other perameters be used to screen for the right candidates? I don't want a thousand stocks to sift through.
3) is this a search screen or stocks have to be entered manually?
4) will all search criteria be updated automatically?
5) million other questions, but these seem the most relevant at this time.
You did a hell of a job doing all this work.
Is there a possibility that I could have you tweak a couple of things in the MACD?

Reply

TradingGeek says:
May 6, 2015 at 12:58 pm

Bill,

Yes, you can definitely plot any time series data in R, including indicators, similarly to Bollinger Bands plot example in my post.

Reply

Joe Toh says:
April 17, 2015 at 6:11 pm

wow this is really great better than a lot of other stuff i have read trying to understand how to build my own trading platform i can have control over. Would be great if there was a back testing guide as well

Reply

TradingGeek says:
May 6, 2015 at 1:01 pm

Joe,

Thank you! I'll be happy to discuss backtesting and answer your questions if you drop me a line via the Contact Us form on the right.

Best Regards

Reply

ea-builder.net says:
July 1, 2015 at 1:53 am

Thank you for sharing the link to the tutorial page, educative post here by the way!

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

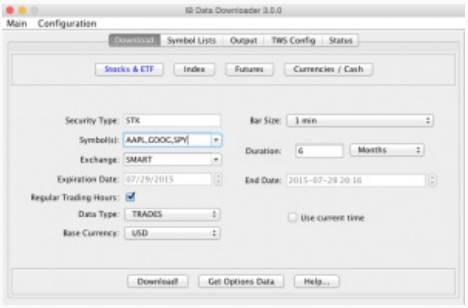
Name *

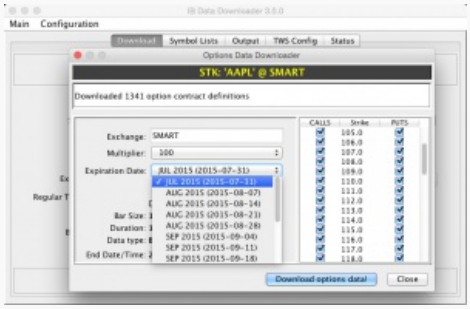
E-mail *

Website

Comment

Post Comment





More Information

---

*Older version of IB Data Downloader (2.1) without support for options historical data available here for a reduced price.

Recent Posts

- Calculating Correlations of Forex Currency Pairs in Python
- Technical Analysis with R
- Technical Analysis in Excel: Part II – MACD and RSI
- Interactive Brokers TWS Performance Optimization
- Technical Analysis in Excel: Part I – SMA, EMA, Bollinger Bands

Contact Us!

Please enter your contact details and a short message below and we will respond to your message shortly.

Name: Your Name

Email Address: Your Email Address

Message: Your Message

Send Message

search here …

Go

↑