Code

R Notebook for Computational Finance 2021, Homework 1 2021

Introduction to Notebooks

Coding notebooks are the standard tool for data science. Notebooks allow you to combine text and code in a single document. For work in R (e.g., homework labs, class project), we will use the Rstudio notebook (see https://rstudio.com/resources/webinars/introducing-notebooks-with-r-markdown/ (https://rstudio.com/resources/webinars/introducing-notebooks-with-r-markdown/)) which is integrated into Rstudio.

In an Rstudio notebook, text is formatted using R Markdown (see **Help/Markdown Quick Reference** from the Rstudio menu for a brief overview of R Markdown syntax)

Executable R code is placed in code chunks (which can be inserted using the keyboard shortcut Ctrl-Alt-I)

Hide

```
# this is a code chunk
```

Any valid R code can be placed in a code chunk. The code is executed by pressing the green triangle at the right side of the code chunk. For example, running the code chunck below prints "Hello world!":

Hide

```
print("Hello world!")
```

```
[1] "Hello world!"
```

Notebooks are a great tool to organize work and facilitate reproducable research. In Rstudio, notebooks can be output to a number of formats including html, pdf and Microsoft Word. This can be done using the **Preview** dropdown on the notebook toolbar. When you click **Preview**, there are 4 main options:

- Preview Notebook
- Knit to html
- Knit to pdf
- Knit to Word

Selecting "Preview to Notebook" shows the formatted notebook in an Rstudio html viewer. This is the prefered way to view the notebook as you are creating it. You can keep the viewer window open and it will automatically update when you save the notebook.

Selecting "Knit to html" will show the output in your default web browser and create a .html file.

Selecting "Knit to pdf" will show the output in a pdf viewer and create a .pdf file. This will be our default for submitting class assignments.

Try out these different Preview options now on this notebook.

Read data in R from text file

Here, we illustrate how to get data into R from a text file in comma separated value (.csv) format. We will use the base R function `read.csv()` .

The file `sbuxPrices.csv` contains monthly adjusted closing price data on sbux from March, 1993 through March 2008. It is assumed to be in the directory `D:\\NCKU\\courses\\2021\\CompFin\\Homework\\hw1` (which is my computer). Use the `setwd()` function to change to the appropriate directory where you have saved the data. Edit the code chunk below and change the path to the path where the `sbuxPrices.csv` is on your computer.

Hide

```
setwd("C:\\Users\\Aaron Lin\\Desktop\\Comp_Finance")
```

Read the Starbucks prices into a `data.frame` object. First look at the online help file for `read.csv()`

Hide

```
?read.csv
```

You should see the help for `read.table` in the **Help** tab of Rstudio (on the right).

Now read in the data using `read.csv()` :

Hide

~~~
```r
sbux.df = read.csv(file=\sbuxPrices.csv\,
                   header=TRUE, stringsAsFactors=FALSE)
~~~

~~~

<!-- rnb-source-end -->

<!-- rnb-chunk-end -->

<!-- rnb-text-begin -->

The object `sbux.df` is a `data.frame` object, which are rectangular data objects with observati
ons in rows and variables in columns. The `class()` function tells you the class of an R object:

<!-- rnb-text-end -->

<!-- rnb-chunk-begin -->

<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuY2xhc3Moc2J1eC5kZilcbmBgYCJ9 -->

```r
class(sbux.df)
~~~

```
[1] "data.frame"
```

The `str()` function gives you information about the structure of the object:

Hide

```
str(sbux.df)
```

```
'data.frame':	181 obs. of  2 variables:
 $ Date     : chr  "3/31/1993" "4/1/1993" "5/3/1993" "6/1/1993" ...
 $ Adj.Close: num  1.13 1.15 1.43 1.46 1.41 1.44 1.63 1.59 1.32 1.32 ...
```

The `head()` function returns the first observations of the object:

Hide

```
head(sbux.df)
```

	Date <chr>	**Adj.Close** <dbl>
1	3/31/1993	1.13
2	4/1/1993	1.15
3	5/3/1993	1.43
4	6/1/1993	1.46
5	7/1/1993	1.41
6	8/2/1993	1.44

6 rows

The `tail()` function shows you the last observations:

Hide

~~~
```r
tail(sbux.df)
~~~

~~~
<!-- rnb-source-end -->

<!-- rnb-chunk-end -->


<!-- rnb-text-begin -->


`colnames()` extract column names:


<!-- rnb-text-end -->


<!-- rnb-chunk-begin -->


<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuYGBgclxuY29sbmFtZXMoc2J1eC5kZilcbmBgYFxuYGBgIn0= -->

```r
```r
colnames(sbux.df)
~~~

~~~
<!-- rnb-source-end -->

<!-- rnb-chunk-end -->


<!-- rnb-text-begin -->


The "Date" column contains character values:


<!-- rnb-text-end -->


<!-- rnb-chunk-begin -->


<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuYGBgclxuY2xhc3Moc2J1eC5kZiREYXRlKVxuYGBgXG5gYGAifQ== -
->

```r
```r
class(sbux.df$Date)
~~~

~~~
<!-- rnb-source-end -->

<!-- rnb-chunk-end -->


<!-- rnb-text-begin -->


The "Adj.Close" column contains numeric values:


<!-- rnb-text-end -->


<!-- rnb-chunk-begin -->


<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuYGBgclxuY2xhc3Moc2J1eC5kZiRBZGouQ2xvc2UpXG5gYGBcbmBgYC
J9 -->

```r
```r
class(sbux.df$Adj.Close)
~~~

~~~
<!-- rnb-source-end -->

<!-- rnb-chunk-end -->


<!-- rnb-text-begin -->


Notice how dates are not the end of month dates. This is Yahoo!'s default when
you download monthly data. Yahoo! doesn't get the dates right for the monthly adjusted
close data.

# Working with `data.frame` objects

Extract the first 5 rows of the price data.


<!-- rnb-text-end -->


<!-- rnb-chunk-begin -->


<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuc2J1eC5kZlsxOjUsIFwiQWRqLkNsb3NlXCJdXG5gYGAifQ== -->

```r
sbux.df[1:5, "Adj.Close"]
~~~

```
[1] 1.13 1.15 1.43 1.46 1.41
```

Hide

```
sbux.df[1:5, 2]
```

```
[1] 1.13 1.15 1.43 1.46 1.41
```

Hide

```
sbux.df$Adj.Close[1:5]
```

```
[1] 1.13 1.15 1.43 1.46 1.41
```

In the above operations, the dimension information was lost. To preserve the dimension information use `drop=FALSE`

Hide

```
sbux.df[1:5, "Adj.Close", drop=FALSE]
```

	Adj.Close <dbl>
1	1.13
2	1.15
3	1.43
4	1.46
5	1.41

5 rows

Find indices associated with the dates 3/1/1994 and 3/1/1995:

Hide

```
idx1 = which(sbux.df$Date == "3/1/1994")
idx2 = which(sbux.df$Date == "3/1/1995")
```

Now, extract prices between 3/1/1994 and 3/1/1995 by subsetting using the extracted indices:

Hide

```
sbux.df[c(idx1, idx2),]
```

	Date <chr>	Adj.Close <dbl>
13	3/1/1994	1.45
25	3/1/1995	1.43

2 rows

Next, create a new `data.frame` containing the price data with the dates as the row names (using `drop = FALSE` prevents dropping the row and column dimensions)

Hide

```
sbuxPrices.df = sbux.df[, "Adj.Close", drop=FALSE]
rownames(sbuxPrices.df) = sbux.df$Date
head(sbuxPrices.df)
```

	Adj.Close <dbl>
3/31/1993	1.13
4/1/1993	1.15
5/3/1993	1.43
6/1/1993	1.46

	Adj.Close <dbl>
7/1/1993	1.41
8/2/1993	1.44

6 rows

With dates as rownames, you can subset directly on the dates. For example, find prices associated with the dates 3/1/1994 and 3/1/1995

Hide

```
sbuxPrices.df[c("3/1/1994", "3/1/1995"), 1, drop = FALSE ]
```

	Adj.Close <dbl>
3/1/1994	1.45
3/1/1995	1.43

2 rows

To show the rownames use `drop=FALSE`

Hide

```
sbuxPrices.df["3/1/1994", 1, drop=FALSE]
```

	Adj.Close <dbl>
3/1/1994	1.45

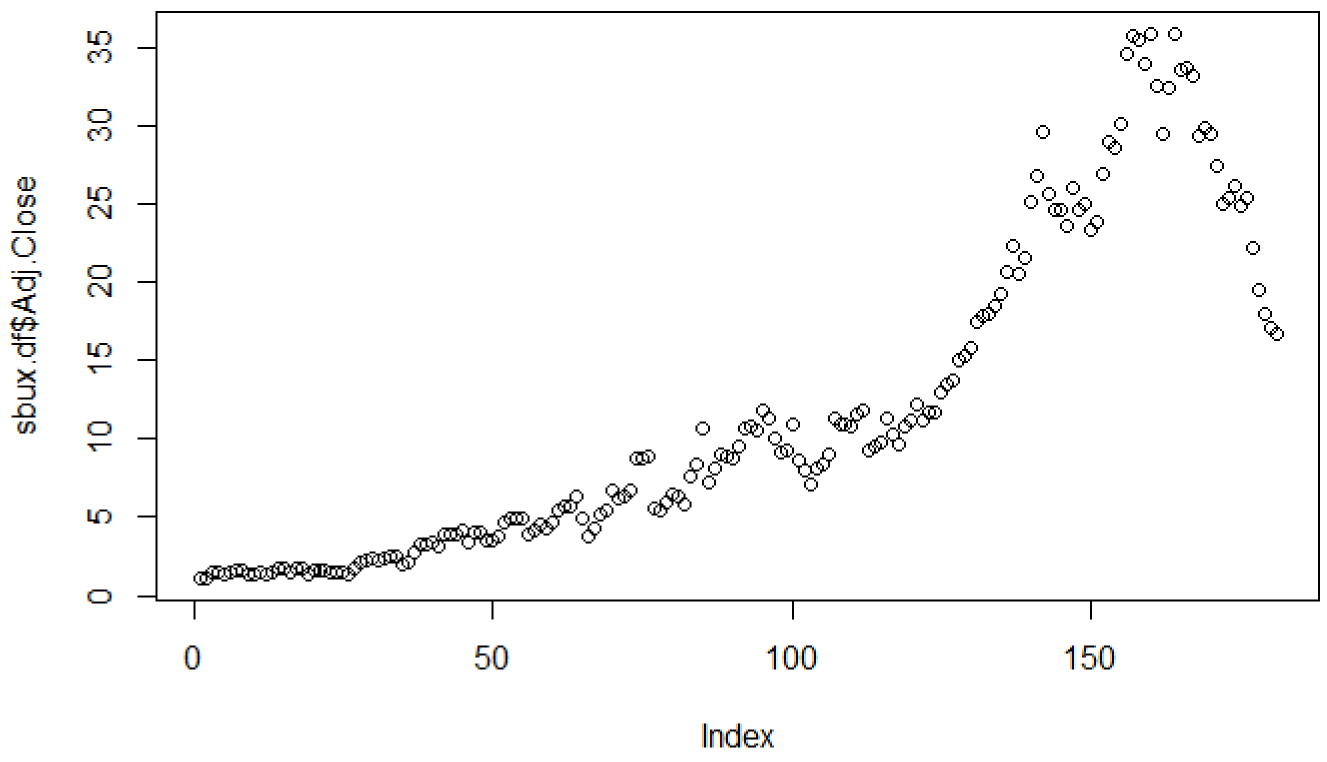
1 row

Plotting data

The base R function 'plot()` can be used for simple plots. The default plot is a "points" (e.g. x-y) plot

Hide

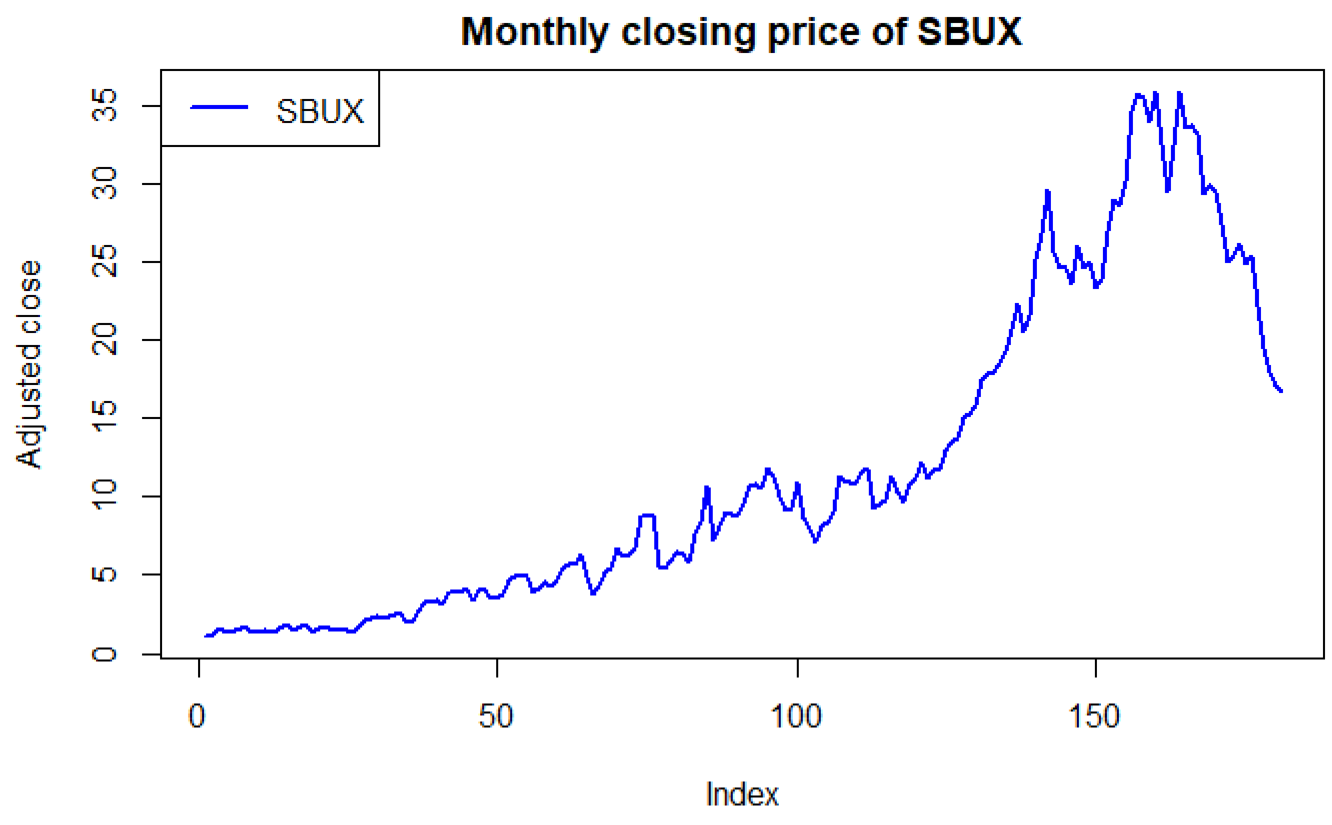
```
plot(sbux.df$Adj.Close)
```



Let’s make a better plot. Set `type="l"` to specify a line plot, `col="blue"` to set the line color to blue, `lwd=2` to double the line thickness, `ylab="Adjusted close"` to add a y axis label, and `main="Monthly closing price of SBUX"` to add a title. Also, use `legend()` to add a legend to the plot.

Hide

```
plot(sbux.df$Adj.Close, type="l", col="blue",
     lwd=2, ylab="Adjusted close",
     main="Monthly closing price of SBUX")
legend(x="topleft", legend="SBUX",
       lty=1, lwd=2, col="blue")
```



Notice there are no dates on the x-axis. We will see how to fix this later using data stored in specialized “time series” objects.

Computing returns

First, compute simple 1-month returns from prices

Hide

```
n = nrow(sbuxPrices.df)
sbux.ret = (sbuxPrices.df[2:n,1] - sbuxPrices.df[1:(n-1),1])/sbuxPrices.df[1:(n-1),1]
head(sbux.ret)
```

```
[1]  0.01770  0.24348  0.02098 -0.03425  0.02128  0.13194
```

Notice that the object `sbux.ret` is not a `data.frame`, but a numeric vector.

Hide

```
class(sbux.ret)
```

```
[1] "numeric"
```

Now add dates as names to the vector.

Hide

```
names(sbux.ret) = rownames(sbuxPrices.df)[2:n]
head(sbux.ret)
```

```
4/1/1993 5/3/1993 6/1/1993 7/1/1993 8/2/1993 9/1/1993
 0.01770  0.24348  0.02098 -0.03425  0.02128  0.13194
```

To ensure that `sbux.ret` is a `data.frame` use `drop=FALSE` when computing returns

Hide

```
sbux.ret.df = (sbuxPrices.df[2:n,1,drop=FALSE] - sbuxPrices.df[1:(n-1),1,drop=FALSE])/
        sbuxPrices.df[1:(n-1),1,drop=FALSE]
head(sbux.ret.df)
```

	Adj.Close <dbl>
4/1/1993	0.01770
5/3/1993	0.24348
6/1/1993	0.02098
7/1/1993	-0.03425
8/2/1993	0.02128
9/1/1993	0.13194

6 rows

Next, compute continuously compounded 1-month returns

Hide

```
sbux.ccret = log(1 + sbux.ret)
head(sbux.ccret)
```

```
4/1/1993 5/3/1993 6/1/1993 7/1/1993 8/2/1993 9/1/1993
 0.01754  0.21791  0.02076 -0.03485  0.02105  0.12394
```

Alternatively,

Hide

```
sbux.ccret = log(sbuxPrices.df[2:n,1]) - log(sbuxPrices.df[1:(n-1),1])
names(sbux.ccret) = rownames(sbuxPrices.df)[2:n]
head(sbux.ccret)
```

```
4/1/1993 5/3/1993 6/1/1993 7/1/1993 8/2/1993 9/1/1993
 0.01754  0.21791  0.02076 -0.03485  0.02105  0.12394
```

Compare the simple and cc returns

Hide

```
head(cbind(sbux.ret, sbux.ccret))
```

```
         sbux.ret sbux.ccret
4/1/1993  0.01770    0.01754
5/3/1993  0.24348    0.21791
6/1/1993  0.02098    0.02076
7/1/1993 -0.03425   -0.03485
8/2/1993  0.02128    0.02105
9/1/1993  0.13194    0.12394
```

Plot the simple and cc returns in separate graphs. Split screen into 2 rows and 1 column using the `par()` function:

Hide

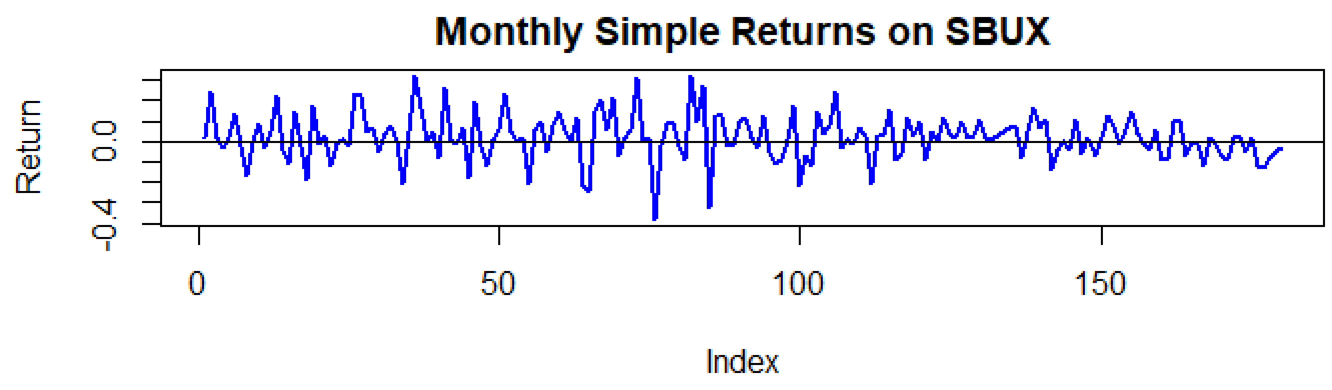
```
par(mfrow=c(2,1))
# plot simple returns first
plot(sbux.ret, type="l", col="blue", lwd=2, ylab="Return",
     main="Monthly Simple Returns on SBUX")
abline(h=0)
```

Hide

```
# next plot the cc returns
plot(sbux.ccret, type="l", col="blue", lwd=2, ylab="Return",
     main="Monthly Continuously Compounded Returns on SBUX")
abline(h=0)
```

Hide

```
# reset the screen to 1 row and 1 column
par(mfrow=c(1,1))
```



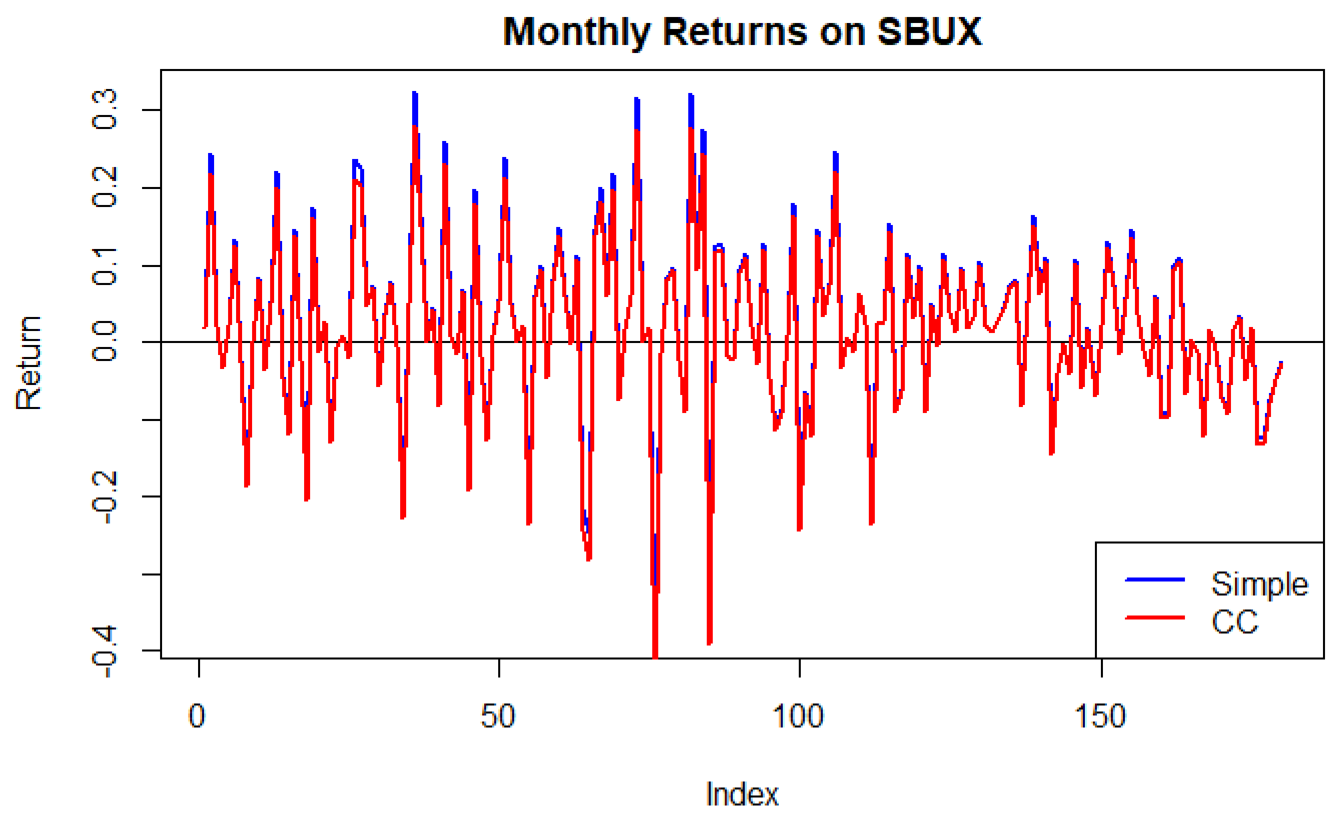
Next, plot the returns on the same graph

Hide

```
plot(sbux.ret, type="l", col="blue", lwd=2, ylab="Return",
     main="Monthly Returns on SBUX")
# add horizontal line at zero
abline(h=0)
```

Hide

```
# add the cc returns
lines(sbux.ccret, col="red", lwd=2)
# add a legend
legend(x="bottomright", legend=c("Simple", "CC"),
       lty=1, lwd=2, col=c("blue","red"))
```



Notice that the cc returns are different from the simple returns when the simple returns are either big positive or negative values.

Calculate growth of $1 invested in SBUX

First, compute gross returns

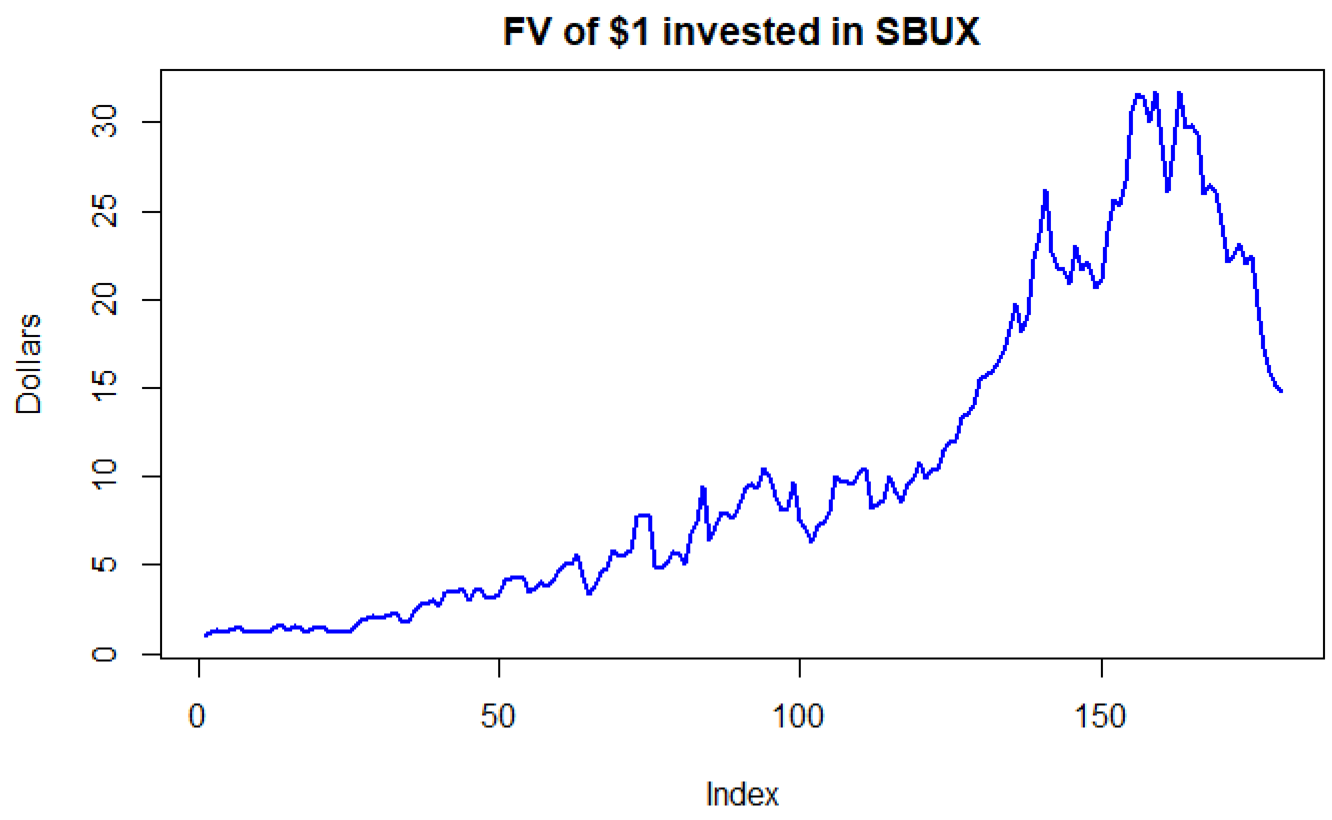
Hide

```
sbux.gret = 1 + sbux.ret
```

Next, compute future values using the cumulative product of the gross returns:

Hide

```
sbux.fv = cumprod(sbux.gret)
plot(sbux.fv, type="l", col="blue", lwd=2, ylab="Dollars",
     main="FV of $1 invested in SBUX")
```



Here we see that $1 invested grows to about $15 over the sample

Dynamic JavaScript web graphics

See the examples at https://rstudio.github.io/dygraphs/ (https://rstudio.github.io/dygraphs/).

In R you can create interactive web graphics based on a number of JavaScript graphics libraries (e.g. D3, Hightcharts, dygraphs, etc). Here we will use the dygraphs JavaScript library through the R package **dygraphs**. You will also use the R packages **xts** and **zoo**. Make sure all of these packages are installed before running the code below.

First we need to create a specialized time series object called an “xts” object using the R package **xts**.

Hide

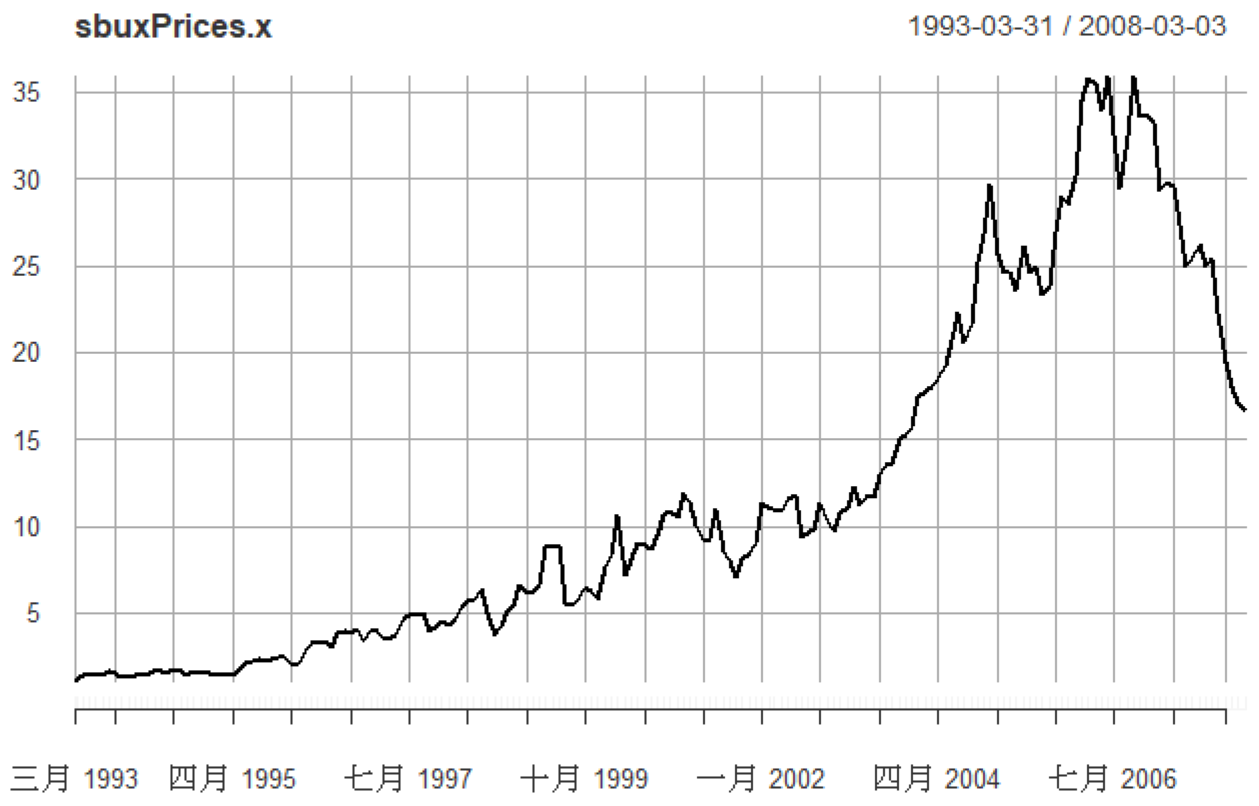
```
suppressPackageStartupMessages(library(xts))
sbuxPrices.x = xts(sbuxPrices.df, as.Date(rownames(sbuxPrices.df), format="%m/%d/%Y"))
sbuxRet.x = xts(sbux.ret, as.Date(names(sbux.ret), format="%m/%d/%Y"))
head(sbuxRet.x)
```

```
               [,1]
1993-04-01  0.01770
1993-05-03  0.24348
1993-06-01  0.02098
1993-07-01 -0.03425
1993-08-02  0.02128
1993-09-01  0.13194
```

The plot method for `xts` objects shows dates on the axes:

Hide

```
plot(sbuxPrices.x)
```



We can also create dynamic graph of prices - a graph will be displayed in Rstudio viewer pane that you can interact with

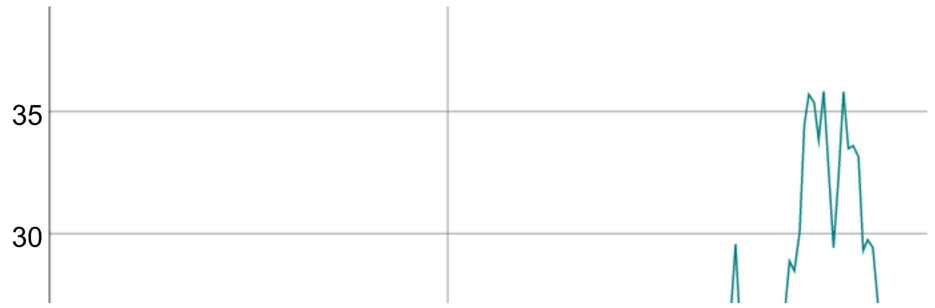
Hide

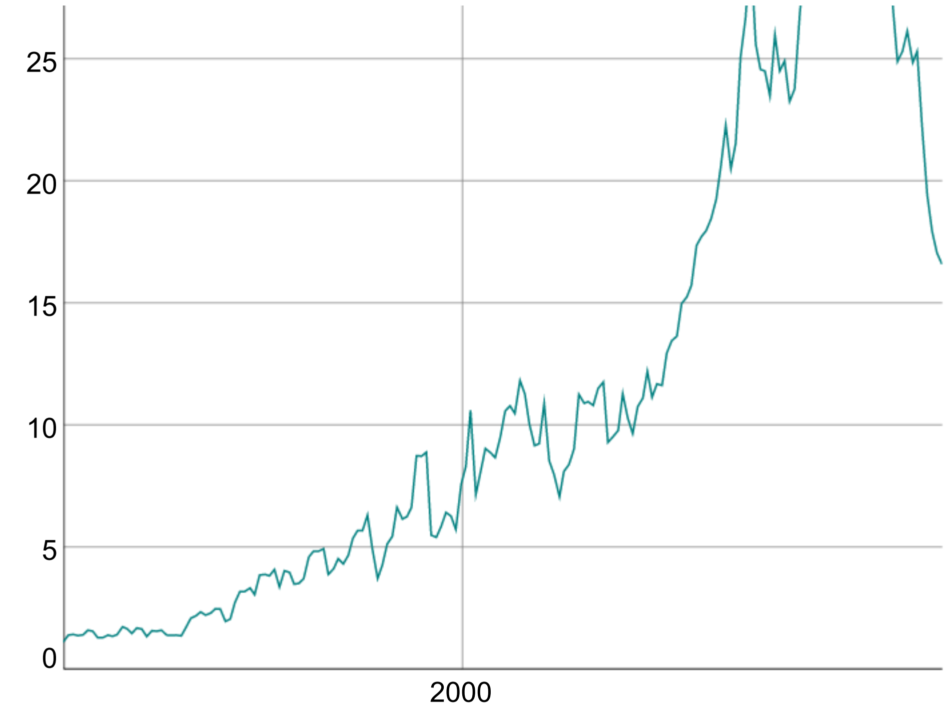
```
library(dygraphs)
```

```
Registered S3 method overwritten by 'htmlwidgets':
  method           from
  print.htmlwidget tools:rstudio
```

Hide

```
dygraph(sbuxPrices.x)
```

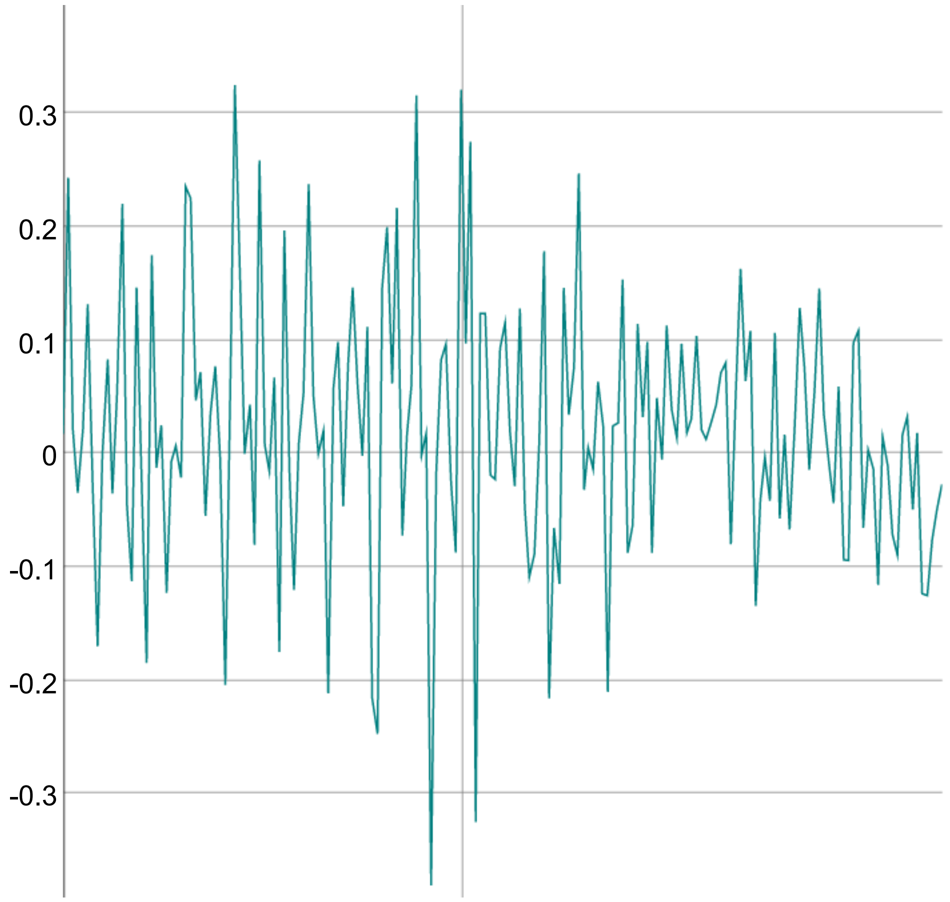


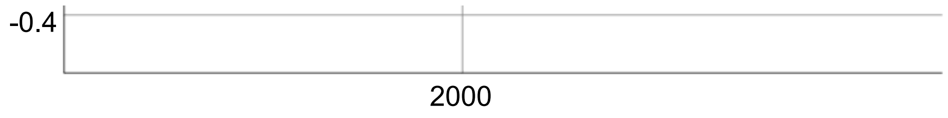


You can interact with the graph. Put your cursor on the graph and see how it iteracts.

Create dynamic graph for returns

```
dygraph(sbuxRet.x)
```




-0.4
2000