Downloading and Exploring Financial Time series Data

Ms. Matabel Odin

9/18/2023

The following R packages can used to download financial data: library(timeSeries) library(quantmod) or Download directly from Yahoo Finance or any other source and save as an xls, xlsx or csv file

Lets load some data

```
MSFT_data=read.csv(file="D:/undergraduate_notes/fts_sta2420/MSFT.csv") #Load
data
summary(MSFT_data)

##      Date              AdjClose     
##  Length:121         Min.   : 20.59  
##  Class :character   1st Qu.: 35.48  
##  Mode  :character   Median : 55.75  
##                     Mean   : 91.06  
##                     3rd Qu.:127.06  
##                     Max.   :301.30  
```

The data corresponds to monthly MSFT adjusted closing prices from November 2011 to August 2021.

EXPLORATORY DATA ANALYSIS.

Load some packages for investigating properties of financial data

```
library(fBasics)

## Warning: package 'fBasics' was built under R version 4.0.5

## Loading required package: timeDate

## Warning: package 'timeDate' was built under R version 4.0.5

## Loading required package: timeSeries

## Warning: package 'timeSeries' was built under R version 4.0.5

library(tseries)

## Warning: package 'tseries' was built under R version 4.0.5

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

1. Get a feel of the data

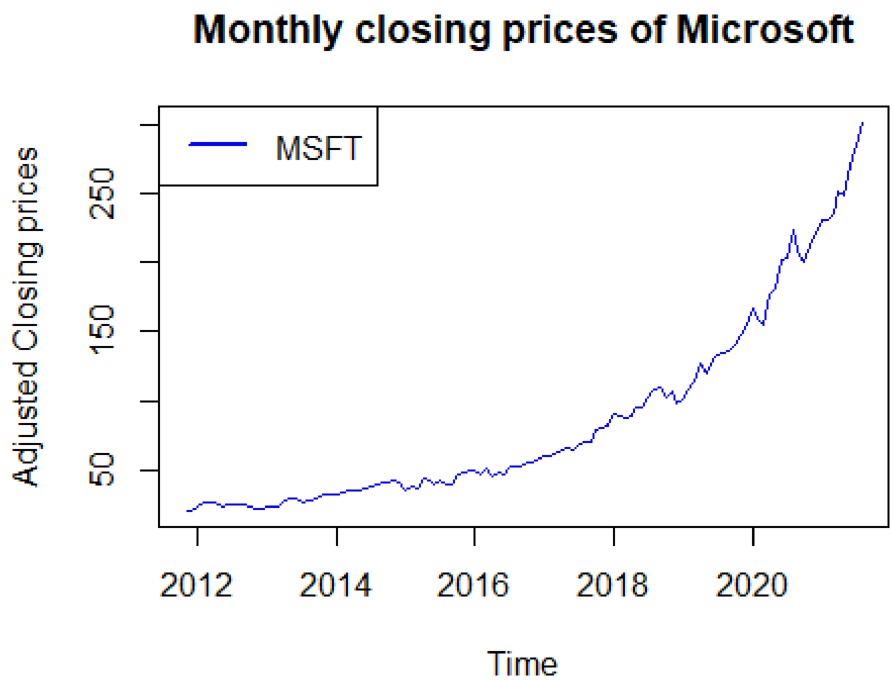
```
str(MSFT_data) # structure of data

## 'data.frame':    121 obs. of  2 variables:
##  $ Date    : chr  "11/1/2011" "12/1/2011" "1/1/2012" "2/1/2012" ...
##  $ AdjClose: num  20.6 21.1 23.9 25.7 26.3 ...

class(MSFT_data)# check the class of the R object

## [1] "data.frame"

dim(MSFT_data) #check dimensions of the data

## [1] 121   2

head(MSFT_data)# view first few rows of the data

##        Date AdjClose
## 1 11/1/2011 20.58750
## 2 12/1/2011 21.05067
## 3  1/1/2012 23.94554
## 4  2/1/2012 25.73761
## 5  3/1/2012 26.33147
## 6  4/1/2012 26.13558

tail(MSFT_data) # view the last few rows of the data

##           Date AdjClose
## 116  6/1/2021 270.3824
## 117  7/1/2021 284.3656
## 118  8/1/2021 301.3032
## 119  9/1/2021 281.9200
## 120 10/1/2021 294.8500
## 121 10/8/2021 294.8500
```

2. Convert data into a time series object

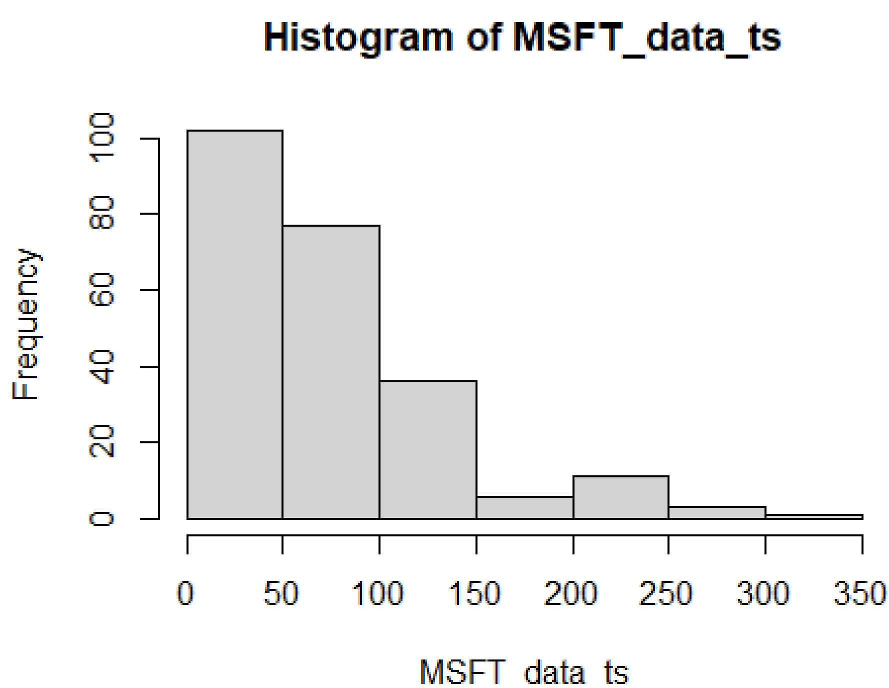
```
MSFT_data_ts=ts(MSFT_data, start=c(2011,11),end=c(2021,8), frequency=12)
```

3. Plotting the price data

```
plot(MSFT_data_ts[,2], col="blue", ylab="Adjusted Closing prices", xlab="Time
",
        main="Monthly closing prices of Microsoft")
legend(x = 'topleft', legend = 'MSFT', lty = 1, lwd = 2, col = 'blue')
```



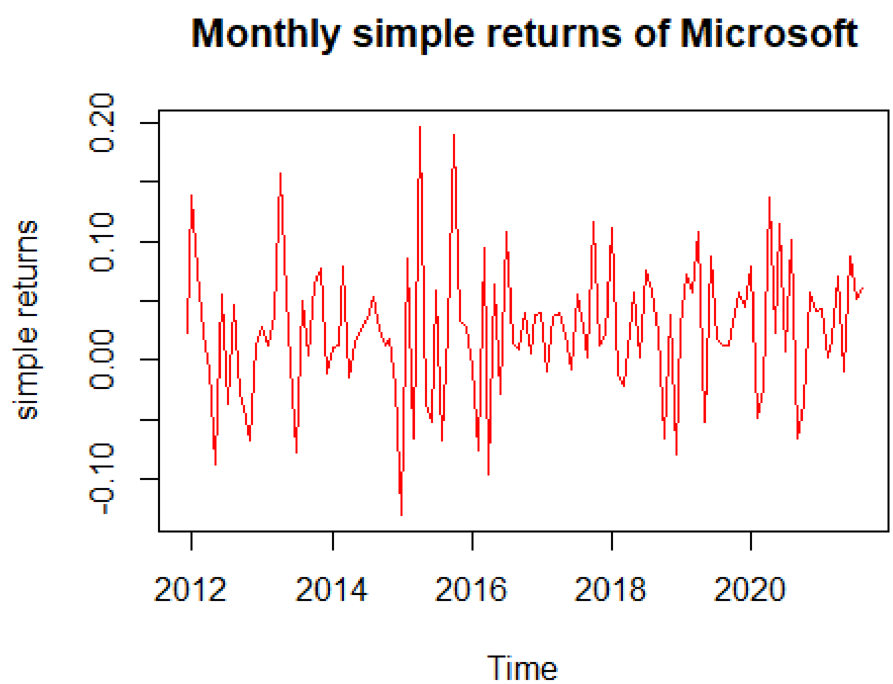

```r
hist(MSFT_data_ts) # plot histogram
```



CALCULATION OF RETURNS

1. Simple returns

```
n <- nrow(MSFT_data) #obtain the number of rows
close_prices <- MSFT_data$AdjClose #extract the closing prices
close_prices=as.data.frame(close_prices)
msft_ret <- ((close_prices[2:n, 1] - close_prices[1:(n-1), 1])/close_prices[1
:(n-1), 1])
# Add dates and plot
msft_ret1=ts(msft_ret, start=c(2011,12),end=c(2021,8), frequency=12)
plot(msft_ret1, col="red", ylab="simple returns", xlab="Time",
     main="Monthly simple returns of Microsoft")
```



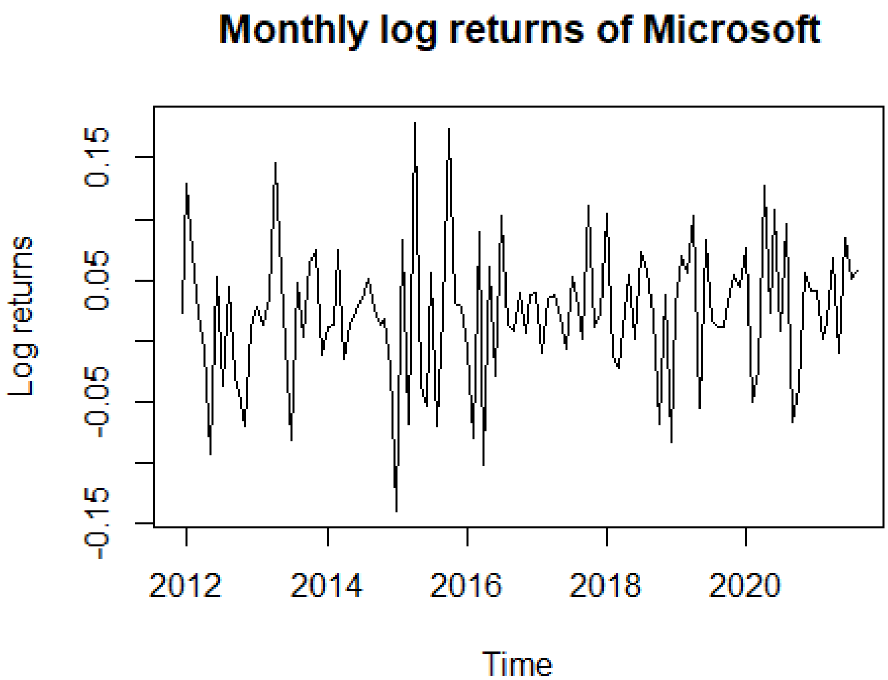
Alternatively use:

simplereturns<-returns(MSFT_data_ts, method="simple")

ts.plot(simplereturns)

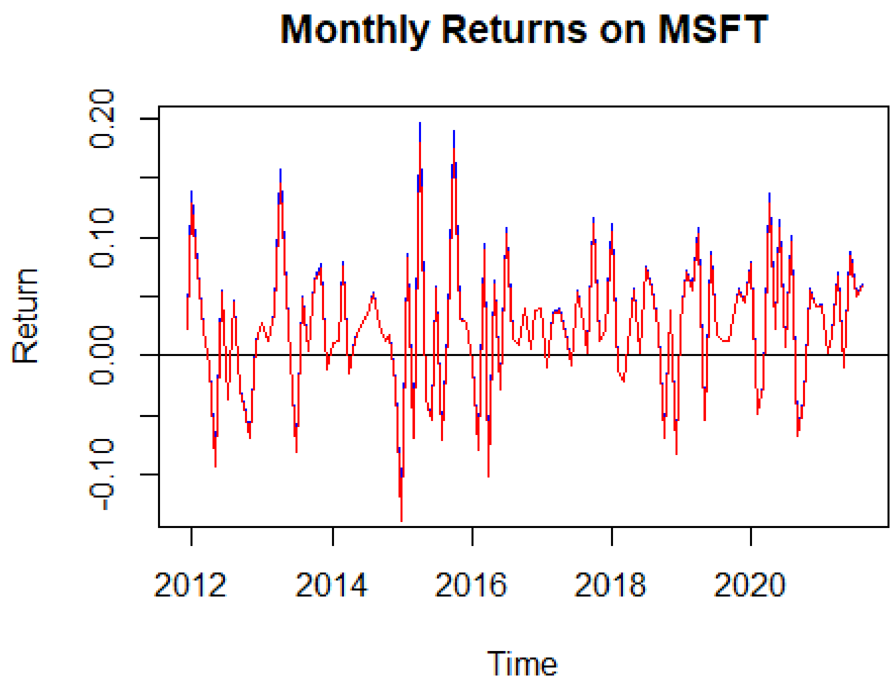
2. Continuously compounded returns/log returns/geometric returns

```
msft_cret <- log(close_prices[2:n, 1]) - log(close_prices[1:(n-1), 1])
msft_ret2<-ts(msft_cret, start=c(2011,12),end=c(2021,8), frequency=12)
plot(msft_ret2, ylab="Log returns", xlab="Time",
     main="Monthly log returns of Microsoft")
```



Compare simple and continuous returns

```
head(cbind(msft_ret, msft_cret))

##          msft_ret    msft_cret
## [1,]  0.022497486  0.022248151
## [2,]  0.137519110  0.128849672
## [3,]  0.074839502  0.072171350
## [4,]  0.023073861  0.022811685
## [5,] -0.007439462 -0.007467273
## [6,] -0.088382153 -0.092534404

#plot on the same graph
plot(msft_ret1, col = "blue", ylab = "Return",
     main = "Monthly Returns on MSFT")
# Add horizontal line at zero
abline(h = 0)
# Add the continuously compounded returns
lines(msft_ret2, col = "red") # you can add legend
```



Descriptive statistics of returns

Use basicStats in fBasics

```
basicStats(msft_ret) # simple returns

##                  msft_ret
## nobs           120.000000
## NAs              0.000000
## Minimum         -0.130247
## Maximum          0.196262
## 1. Quartile     -0.007389
## 3. Quartile      0.056124
## Mean             0.024012
## Median           0.023392
## Sum              2.881475
## SE Mean          0.005223
## LCL Mean         0.013671
## UCL Mean         0.034353
## Variance         0.003273
## Stdev            0.057210
## Skewness         0.157667
## Kurtosis         0.571776

basicStats(msft_cret) #compound returns

##                 msft_cret
## nobs           120.000000
```

```
## NAs            0.000000
## Minimum       -0.139546
## Maximum        0.179201
## 1. Quartile   -0.007416
## 3. Quartile    0.054605
## Mean           0.022182
## Median         0.023123
## Sum            2.661783
## SE Mean        0.005101
## LCL Mean       0.012081
## UCL Mean       0.032282
## Variance       0.003123
## Stdev          0.055881
## Skewness      -0.052434
## Kurtosis       0.460632
```

Alternatively have commands for individual tests

```
mean(msft_cret)

## [1] 0.02218152

var(msft_cret)

## [1] 0.003122672

stdev(msft_cret)

## [1] 0.05588087

skewness(msft_cret)

## [1] -0.05243391
## attr(,"method")
## [1] "moment"

length(msft_cret)

## [1] 120
```

Some common tests on return series

Q. Write the null and alternative hypothesis for each case

```
t.test(msft_cret) #testing mean return=0

##
##  One Sample t-test
##
## data:  msft_cret
## t = 4.3483, df = 119, p-value = 2.911e-05
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
```

```
##  0.01208063 0.03228241
## sample estimates:
##  mean of x
## 0.02218152
```

```r
adf.test(msft_cret)#testing for stationarity
```

```
## Warning in adf.test(msft_cret): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  msft_cret
## Dickey-Fuller = -7.1404, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```r
normalTest(msft_ret,method="jb") # testing for normality assumption
```

```
##
## Title:
##  Jarque - Bera Normalality Test
##
## Test Results:
##   STATISTIC:
##     X-squared: 2.5073
##   P VALUE:
##     Asymptotic p Value: 0.2855
##
## Description:
##  Wed Sep 20 14:45:11 2023 by user: ADMIN
```

```r
Box.test(msft_cret,lag=12, type="Ljung")
```

```
##
##  Box-Ljung test
##
## data:  msft_cret
## X-squared = 17.343, df = 12, p-value = 0.1372
```

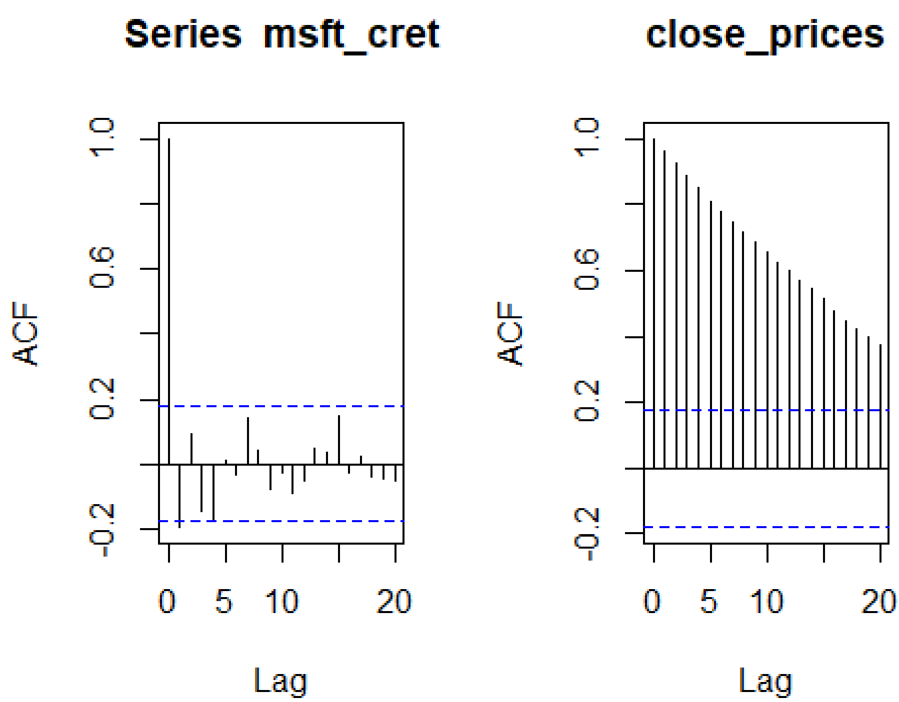
```r
# testing for serial autocorrelation
Box.test(msft_cret^2,lag=12, type="Ljung") # squared return series. Compare w
ith price series
```

```
##
##  Box-Ljung test
##
## data:  msft_cret^2
## X-squared = 12.949, df = 12, p-value = 0.3728
```

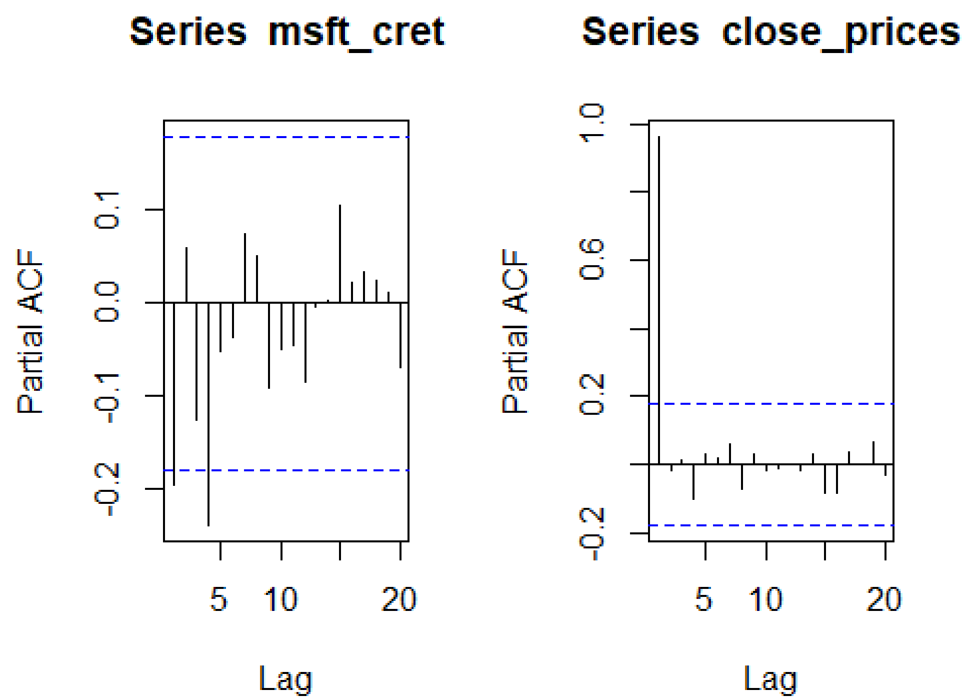
In practice, the choice of m may affect the performance of the Q(m) statistic. Several values of m are often used. Simulation studies suggest that the choice of $m \approx \ln(T)$ provides better power performance. This general rule needs modification in analyzing seasonal time series for which autocorrelations with lags at the multiples of the seasonality are more important. For instance, lags 12 and 24 are important for monthly time series

You can also check for serial autocorrelation using the acf and pacf plots

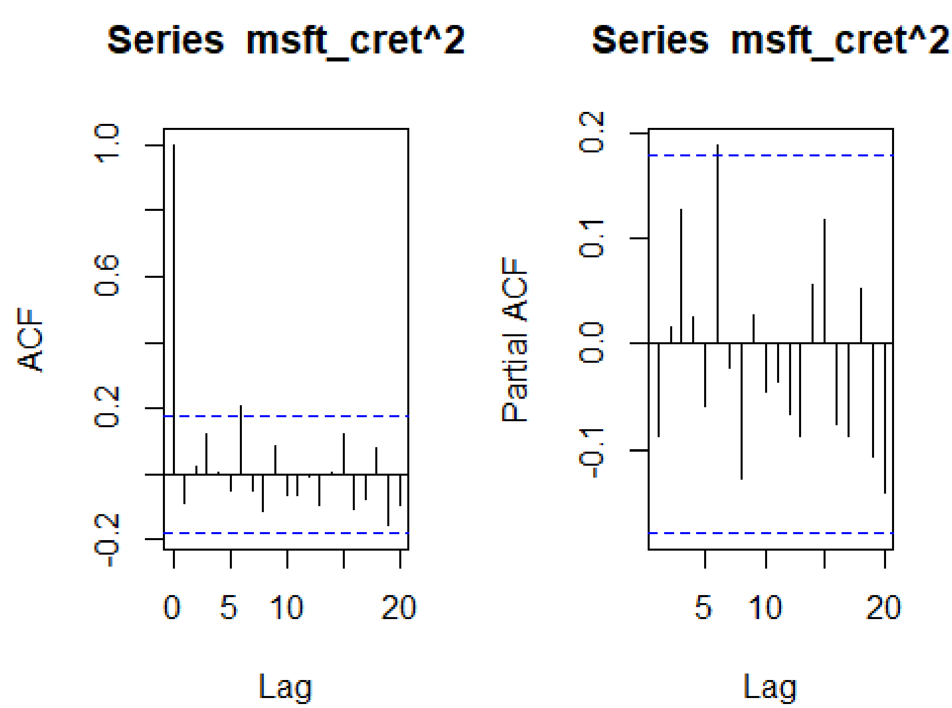
```
par(mfrow=c(1,2))
acf(msft_cret)
acf(close_prices) # check how the spikes pass through the confidence bands
```




```
pacf(msft_cret)
pacf(close_prices)
```




```
acf(msft_cret^2)
pacf(msft_cret^2)
```



Comment on the plots