```
# load the "quantmod" package
require('quantmod')
```

Get Apple stock quotes (daily).

```
# get quotes
AAPL <- getSymbols('AAPL', auto.assign = FALSE)
```

Returns

$$R_t = \frac{P_t-P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}}-1$$

Note the use of the `lag` function in the following code snippet. The function shifts the time base of a `xts` object back by a given number of observations. **Warning**: the function behaves differently for `zoo`, `numeric` or other object classes.

```
# shift the series back by one period and compute returns
r <- AAPL$AAPL.Close/lag(AAPL$AAPL.Close, k = 1) - 1
```

Log-Returns

$$r_t = ln(P_t)-ln(P_{t-1})$$

```
# shift the series back by one period and compute log-returns
r.log <- log(AAPL$AAPL.Close) - log(lag(AAPL$AAPL.Close, k = 1))
```

Returns cannot be computed for the first observation, as there is no previous observation to use. R returns `NA` in this case. Clean the data.

```
# drop NA from data
r <- na.omit(r)
r.log <- na.omit(r.log)
```

Log-returns are approximately equal to standard returns when they are small. Note: **approximately** does not mean **equal**.
$$r_t=ln(P_t)-ln(P_{t-1})=ln\Bigl(\frac{P_t}{P_{t-1}}\Bigl)=ln\Bigl(\frac{P_t-P_{t-1}+P_{t-1}}{P_{t-1}}\Bigl)=ln(1+R_t) \approx R_t$$
where the last passage can be easily obtained by Taylor expansion. Compare the returns.

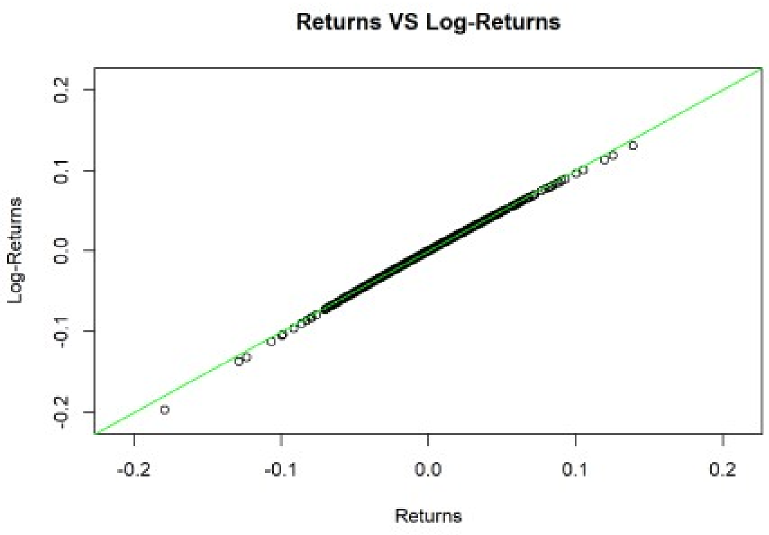
```
# build a data.frame containing ret and log-ret
x <- data.frame(r, r.log)
colnames(x) <- c('ret', 'log.ret')
head(x)
```

```
##                     ret      log.ret
## 2007-01-04  0.022195679  0.021952941
## 2007-01-05 -0.007121188 -0.007146665
## 2007-01-08  0.004938272  0.004926118
## 2007-01-09  0.083070106  0.079799700
## 2007-01-10  0.047855589  0.046745780
## 2007-01-11 -0.012371021 -0.012448179
```

Plot the returns. For high quality charts, refer to the `ggplot2` package.

```
# base R plot function
plot(x$log.ret ~ x$ret, main = 'Returns VS Log-Returns', ylab = 'Log-Returns',
xlab = 'Returns', xlim = c(-0.21, 0.21), ylim = c(-0.21, 0.21))
# add the line y = x
```

```
abline(b = 1, a = 0, col = 'green')
```



Multi-Period Returns

\[R_{t,s}=\frac{P_t}{P_s}-1=\frac{P_t}{P_{t-1}}\frac{P_{t-1}}{P_s}-1=\frac{P_t}{P_{t-1}}\frac{P_{t-1}}{P_{t-2}}…\frac{P_{s+1}}{P_s}-1=\\
=(R_{t}+1)(R_{t-1}+1)…(R_{s+1}+1) – 1=\\
=\Bigl(\prod_{i=s+1}^{t}(R_i+1)\Bigl)-1\]

```
# direct calculation
AAPL$AAPL.Close['2018-12-31'][[1]]/AAPL$AAPL.Close['2015-12-31'][[1]] - 1
```

```
## [1] 0.498575
```

```
# compound calculation
r.i <- window(r, start = '2016-01-01', end = '2018-12-31')  # extract returns
prod(r.i+1) - 1
```

```
## [1] 0.498575
```

Multi-Period Log-Returns

\[r_{t,s}=ln(P_t)-ln(P_s)=ln(P_t)-ln(P_s)=ln(P_t)-ln(P_{t-1})+ln(P_{t-1})-ln(P_s)=\\
=ln(P_t)-ln(P_{t-1})+ln(P_{t-1})-ln(P_{t-2}) … +\;ln(P_{s+1})-ln(P_s)=\\
=r_t+r_{t-1}+…\;+r_{s+1}=\sum_{i=s+1}^tr_i\]

```
# direct calculation
log(AAPL$AAPL.Close['2018-12-31'][[1]]) - log(AAPL$AAPL.Close['2015-12-
31'][[1]])
```

```
## [1] 0.4045146
```

```
# compound calculation
r.i <- window(r.log, start = '2016-01-01', end = '2018-12-31')  # extract
returns
sum(r.i)
```

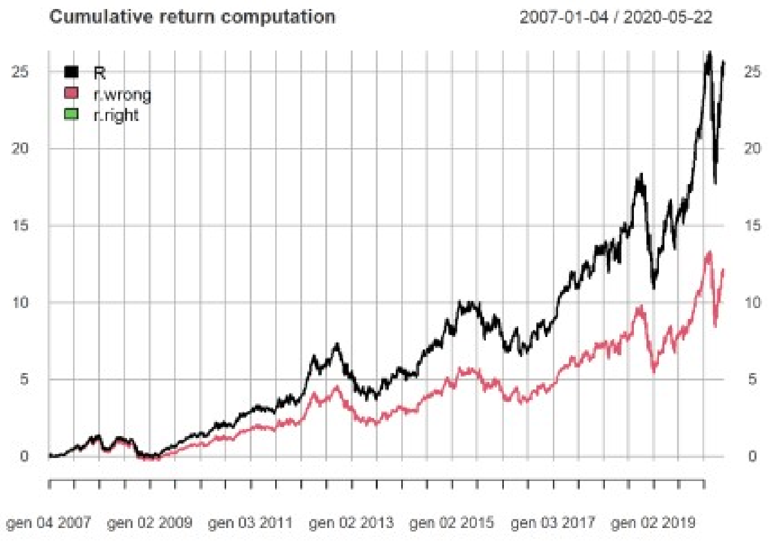
```
## [1] 0.4045146
```

Cumulative Returns

Compute the standard returns with respect to the first observation at each point in time (\(R_{t,0}\) for every \(t\)). Use the following approaches:

- (RIGHT) cumulate standard daily returns
- (RIGHT) cumulate log-returns and convert to standard return using the following property: \(r=ln(1+R) \rightarrow R=exp(r)-1\)
- (WRONG) cumulate log-returns as if they were standard returns. This holds approximately, and the error becomes larger and larger when cumulating more and more returns

```
# cumulate standard returns
r.cum <- cumprod(1+r) - 1

# cumulate log-returns and convert to standard returns
r.log.cum.right <- exp(cumsum(r.log)) - 1

# cumulate log-returns as if they were standard returns
r.log.cum.wrong <- cumprod(1+r.log) -1

# create a unique xts object
x <- merge(r.cum, r.log.cum.wrong, r.log.cum.right)
colnames(x) <- c('R', 'r.wrong', 'r.right')

# plot and compare
plot(x, legend.loc = 'topleft', main = 'Cumulative return computation')
```



The green line coincides with the black one. This can be checked printing the values.

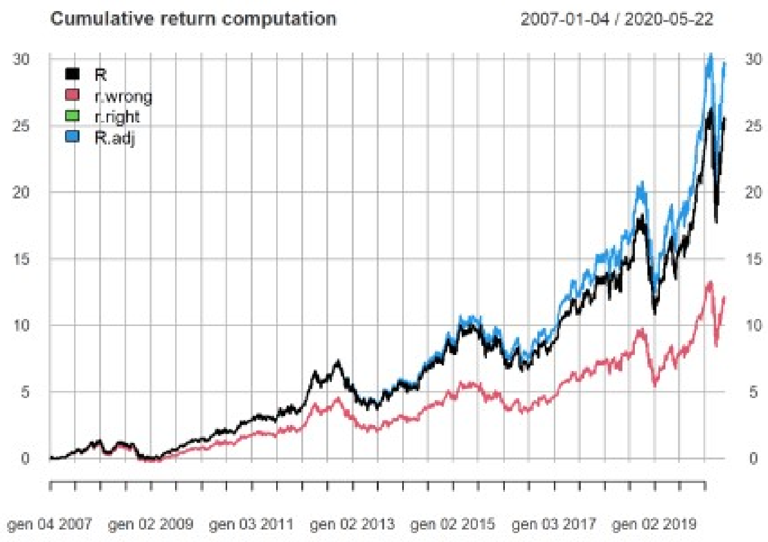
```
# print values
head(x)
```

```
##                     R    r.wrong    r.right
## 2007-01-04 0.02219568 0.02195294 0.02219568
## 2007-01-05 0.01491643 0.01464939 0.01491643
## 2007-01-08 0.01992836 0.01964767 0.01992836
## 2007-01-09 0.10465392 0.10101525 0.10465392
## 2007-01-10 0.15751779 0.15248306 0.15751779
## 2007-01-11 0.14319811 0.13813675 0.14319811
```

Adjusted Price

The closing price is the ‘raw’ price which is just the cash value of the last transacted price before the market closes. Adjusted closing price amends a stock’s closing price to accurately reflect that stock’s value after accounting for any corporate actions (e.g. dividends, splits). It is considered to be the true price of that stock and is often used when examining historical returns or performing a detailed analysis of historical returns. Compute returns using the adjusted price.

```
# compute returns using adjusted prices
r.adj <- AAPL$AAPL.Adjusted/lag(AAPL$AAPL.Adjusted, k = 1) - 1

# drop NA
r.adj <- na.omit(r.adj)

# cumulate returns
r.adj.cum <- cumprod(1+r.adj) - 1

# add adjusted returns to the ones computed in the previous section
x <- merge(x, r.adj.cum)
colnames(x)[4] <- 'R.adj'

# plot and compare
plot(x, legend.loc = 'topleft', main = 'Cumulative return computation')
```



Central Limit Theorem

The central limit theorem establishes that when **independent and identically distributed** (i.i.d.) random variables are added, their properly normalized sum tends toward a normal distribution even if the original variables themselves are not normally distributed.

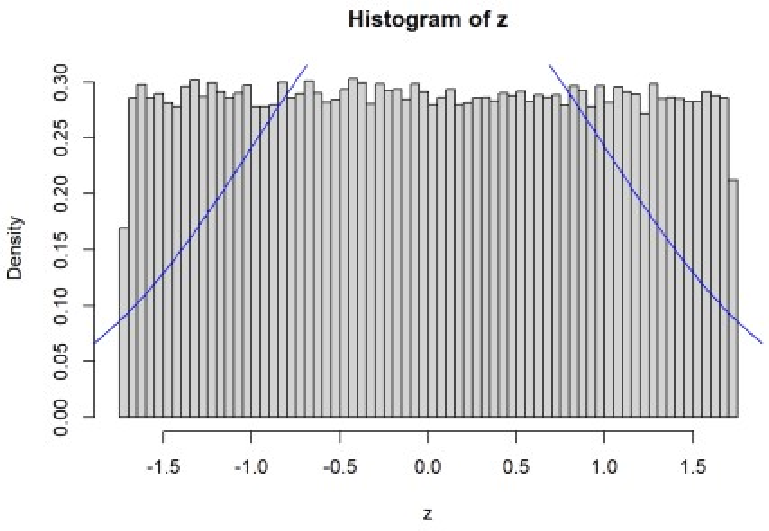
$$\frac{S_n-\mu_n}{\sigma_n}\rightarrow N(0,1)$$

Define the following function to test the Central Limit Theorem

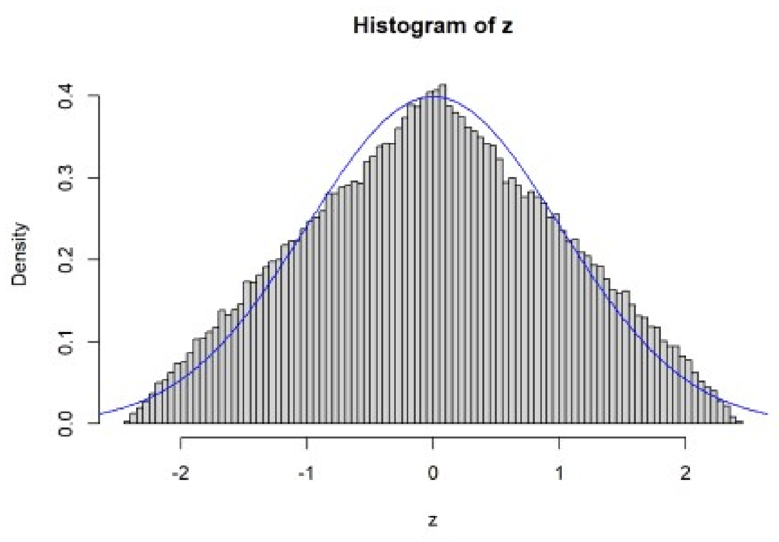
```
# n:    number of iid random variables to sum up
# plot: plotting results? Default FALSE
CLT <- function(n, plot = FALSE){

  # number of trials
  N <- 100000

  # build a matrix with
  #  - 'N' rows (numer of trials)
  #  - 'n' columns (number of iid random variables to sum up)
  #  - 'n*N' uniform random variables: 'runif' function
  u <- matrix(data = runif(n = n * N), ncol = n, nrow = N)

  # for each row, sum up all columns -> generate Sn
  z <- rowSums(u)

  # normalize according to the theorem
  z <- (z - mean(z))/sd(z)

  # if plot
  if(plot){

    # generate histogram of Sn
    hist(z, freq = FALSE, breaks = 100)

    # add normal distribution
    x <- seq(-3,3,0.01)
    lines(x = x, y = dnorm(x = x), col = 'blue')

  }

  # else return Sn values
  else{
    return(z)
  }

}
```

Run the function with several values for `n`. The distribution of Sn converges to a Normal distribution as n increases.

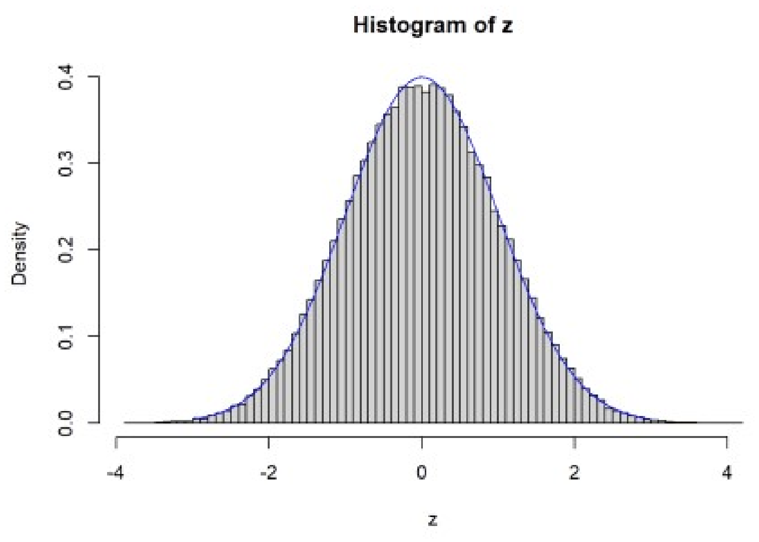
```
CLT(n = 1,  plot = TRUE)
```



```
CLT(n = 2,  plot = TRUE)
```



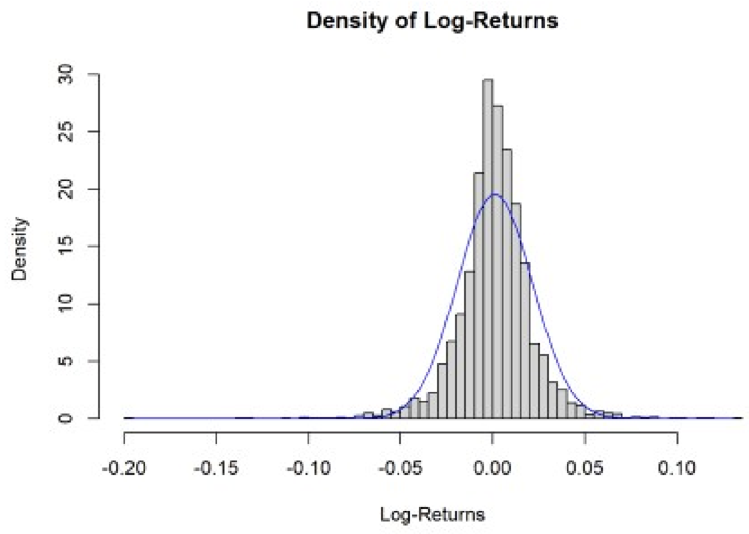
```
CLT(n = 10, plot = TRUE)
```



Normality of financial returns

Plot AAPL log-returns. Are they Normal?

```
# histogram of AAPL log-returns
hist(r.log, breaks = 50, main = "Density of Log-Returns", xlab = 'Log-Returns',
freq = FALSE)

# mean
r.log.mean <- mean(r.log)

# standard deviation
r.log.sd    <- sd(r.log)

# density of N(mu, sigma)
x <- seq(min(r.log), max(r.log), by = 0.001)
y <- dnorm(x = x, mean = r.log.mean, sd = r.log.sd)

# add fitted normal distribution
lines(x = x, y = y, col = 'blue')
```



AAPL daily log-returns, and stock returns in general, are not Normal (Fama 1965, Journal of Business, Nobel price 2013). On the other hand, daily log-returns can be thought as the cumulative sum of intraday log-returns and, according to the central limit theorem, they should be normally distributed. This means that intraday stock returns are **not** iid and the central limit theorem does not hold in this case.

Law of Large Numbers

The law of large numbers is a theorem that describes the result of performing the same experiment a large number of times. According to the law, the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed. In other words, the mean is a good estimator of the expected value.

$$\bar{X_n} \rightarrow E[X]$$
Estimate the expected value of a uniform random variable that takes values in [0,1].

```
# generate 100 uniform random variables and compute the mean
mean(runif(n = 100, min = 0, max = 1))
```

```
## [1] 0.4919098
```

```
# generate 100000 uniform random variables to increase precision
mean(runif(n = 100000, min = 0, max = 1))
```

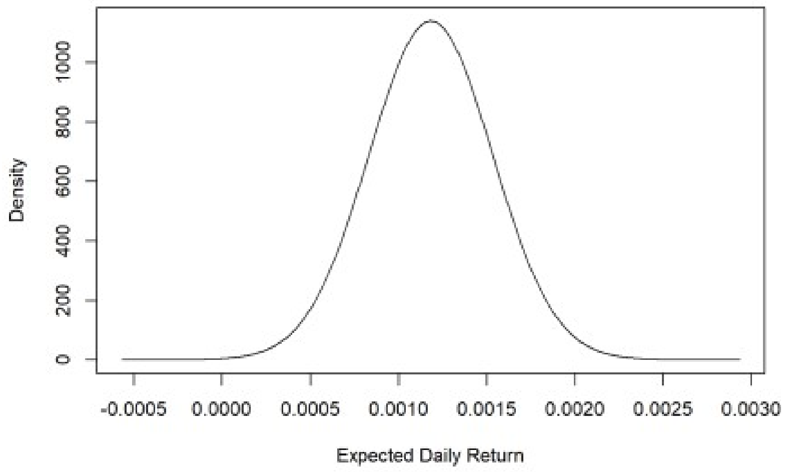
```
## [1] 0.5003542
```

For simulated data it is possible to increase the precision of the estimation by increasing the number of trials. For real data, where the sample size is finite, the precision cannot be increased arbitrarily and we need to understand how uncertain the estimation is. In other words, we need an estimation of the precision, together with the estimation of the mean. This is usually called *standard error* or *standard deviation of the mean*. When realized returns are iid, the standard error of the mean can be easily computed by $\sigma_n/\sqrt{n}$, where σ_n is the standard deviation of the realized returns and n is the sample size. When realized returns are not iid, it is under certain conditions (“stationarity of returns”) still possible to compute the standard error, but the required formula is slightly more complicated. Moreover, the estimation of the mean, for large n, is normally distributed according to the central limit theorem.

$$\bar{X_n} \rightarrow N\Bigl(\bar{X_n}, \frac{\sigma_n}{\sqrt{n}}\Bigl)$$
Estimate the expected daily return for AAPL and its distribution.

```
# estimatation of the expected daily return
mu     <- mean(r)
# estimation of the standard deviation of the sample
sigma  <- sd(r)
```

```
# sample size
n      <- length(r)
# standard devation of the mean
st.err <- sigma/sqrt(n)
# plot the estimated distribution of the expcted return
x <- seq(mu-5*st.err, mu+5*st.err, length.out = 1000)
plot(x = x, y = dnorm(x = x, mean = mu, sd = st.err), type = 'l', ylab =
'Density', xlab = 'Expected Daily Return')
```



```
# which is the most likely expected return?
mu
```

```
## [1] 0.001181849
```

```
# how likely is the expected return to be less than 0.05%?
pnorm(q = 0.0005, mean = mu, sd = st.err, lower.tail = TRUE)
```

```
## [1] 0.02580614
```

Remark. The law of large numbers holds when performing the **same** experiment a large number of times. This is not the case for financial and economic data. Assuming returns coming from the same experiment every day ignores the different economic conditions, sentiment and environment of the different periods.

Stochastic Process

A stochastic process can be defined as a collection of random variables.

Brownian Motion

Increments are normally distributed:

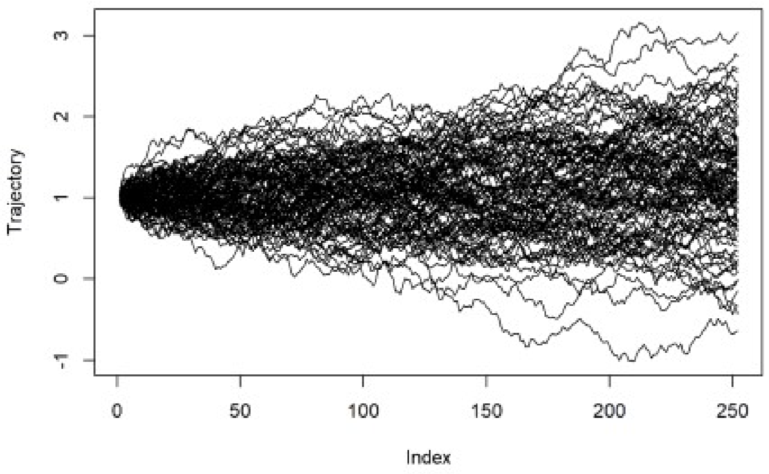
\[\Delta X_t \sim N(\mu, \sigma)\]

The following code snippet defines a function to simulate trajectories of a Brownian Motion.

```
# n:     number of trajectories to simulate
# t:     number of periods for a single trajectory
# mu:    mean of the increments
# sigma: standard deviation of the increments
# x.0:   initial value
# plot:  plot trajectories? default TRUE
BM.sim <- function(n, t = 252, mu = 0.001, sigma = 0.05, X.0 = 1, plot = TRUE){
  # generate increments and store in a matrix with:
  #  - 't' rows (numer of time points for each trajectory)
  #  - 'n' columns (number of trajectories to simulate)
  r <- matrix(data = rnorm(n = n*t, mean = mu, sd = sigma), ncol = n, nrow = t)

  # for each column (trajectory), sum increments starting from x.0
  X.t <- apply(r, MARGIN = 2, function(x) X.0 + cumsum(x))

  # plot and return
  if(plot) plot(as.zoo(X.t), screens = 'single', ylab = 'Trajectory')
  return(X.t)
}
```

Simulate the Brownain Motion.

```
# simulate 100 trajectories
bm <- BM.sim(100)
```



The Brownian Motion can assume negative values. This is not well-suited for financial applications.

Geometric Brownian Motion

Returns are normally distributed. More precisely, log returns are normally distributed and simple returns are log-normal distributed:

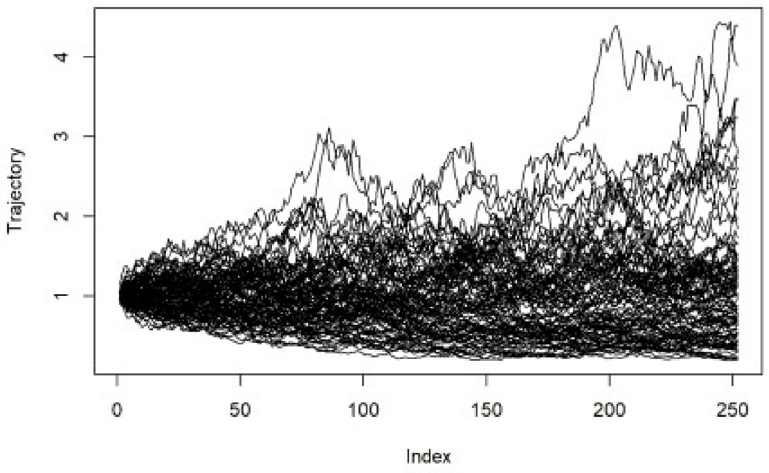
\[\frac{\Delta X_t}{X_t} \sim N(\mu, \sigma)\]
The following code snippet defines a function to simulate trajectories of a Geometric Brownian Motion.

```
# n:         number of trajectories to simulate
# t:         number of periods for a single trajectory
# mu:        mean of the returns
# sigma:     standard deviation of the returns
# x.0:       initial value
# plot:      plot trajectories? default TRUE
# log.scale: plot on a log-scale? default FALSE
GBM.sim <- function(n, t = 252, mu = 0.001, sigma = 0.05, X.0 = 1, log.scale =
FALSE, plot = TRUE){

  # generate returns and store in a matrix with:
```

```
  #  - 't' rows (numer of time points for each trajectory)
  #  - 'n' columns (number of trajectories to simulate)
  r <- matrix(data = rnorm(n = n*t, mean = mu, sd = sigma), ncol = n, nrow = t)

  # for each column (trajectory), cumulate returns starting from x.0
  X.t <- apply(r, MARGIN = 2, function(x) X.0 * cumprod(1+x))

  # plot and return
  if(plot){
    if(log.scale) plot(as.zoo(X.t), screens = 'single', log = 'y',  ylab =
'Trajectory')
    else plot(as.zoo(X.t), screens = 'single',  ylab = 'Trajectory')
  }
  return(X.t)
}
```

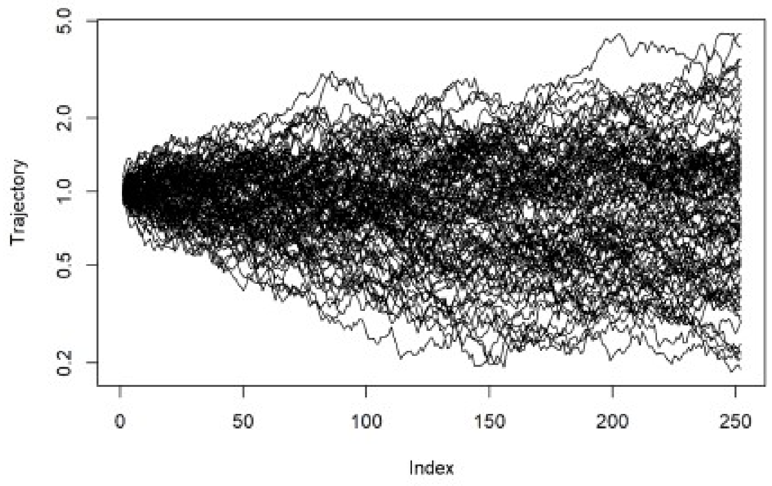
Simulate the Geometric Brownain Motion.

```
# set RNG seed
set.seed(123)
# simulate 100 trajectories
gbm <- GBM.sim(100)
```



The Geometric Brownian Motion is positive defined. Plot the trajectories on a log-scale and check.

```
# set the same RNG seed
set.seed(123)
# simulate 100 trajectories and plot on a log-scale
gbm <- GBM.sim(100, log.scale = TRUE)
```



Exercise

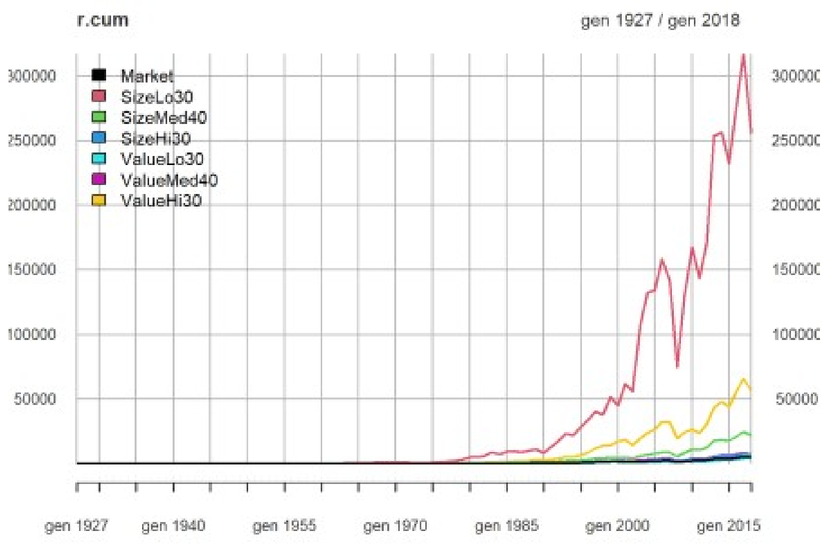
Compute the mean, standard deviation, skewness, and kurtosis for both returns and log-returns for each of the seven investment strategies that can be found in this file. The file contains stock data from the website of Kenneth R. French. Which is your favourite strategy?

```
# import data
data <- read.csv('https://storage.guidotti.dev/course/asset-pricing-unine-2019-2020/basic-statistical-concepts-for-finance.csv', sep = ';', stringsAsFactors = FALSE)

# convert data to xts
data <- xts(data[,-1], order.by = as.yearmon(data[,1]))

# returns and log-returns
r <- data/lag(data,1)-1
r.log <- log(data)-log(lag(data,1))

# drop NA
r <- na.omit(r)
r.log <- na.omit(r.log)

# skewness function
skewness <- function(x){
  mean((x-mean(x))^3)/sd(x)^3
}

# kurtosis function
kurtosis <- function(x){
  mean((x-mean(x))^4)/var(x)^2 - 3
}

# mean: returns
apply(r, MARGIN = 2, mean)
```

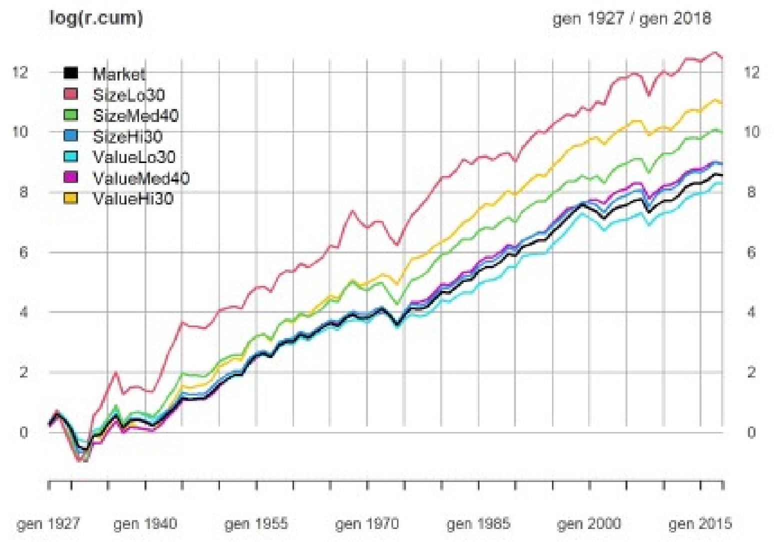
```
##     Market   SizeLo30  SizeMed40   SizeHi30  ValueLo30 ValueMed40   ValueHi30
##  0.1171537  0.2089651  0.1490341  0.1239627  0.1138287  0.1238843  0.1592684
```

```
# mean: log-returns
apply(r.log, MARGIN = 2, mean)
```

```
##     Market   SizeLo30  SizeMed40   SizeHi30  ValueLo30 ValueMed40  ValueHi30
## 0.09315030 0.13532657 0.10865334 0.09704191 0.09021366 0.09727667 0.11893181

# sd: returns
apply(r, MARGIN = 2, sd)

##     Market   SizeLo30  SizeMed40   SizeHi30  ValueLo30 ValueMed40  ValueHi30
##  0.2003840  0.4149276  0.2766021  0.2131912  0.2012226  0.2102552  0.2709719

# sd: log-returns
apply(r.log, MARGIN = 2, sd)

##     Market   SizeLo30  SizeMed40   SizeHi30  ValueLo30 ValueMed40  ValueHi30
##  0.1946990  0.3334365  0.2536980  0.2071972  0.1933582  0.2078262  0.2490666

# skewness: returns
apply(r, MARGIN = 2, skewness)

##      Market    SizeLo30   SizeMed40    SizeHi30   ValueLo30  ValueMed40
ValueHi30
## -0.39025418  1.40580597  0.38599846 -0.27243948 -0.31865990 -0.08623006
0.38710970

# skewness: log-returns
apply(r.log, MARGIN = 2, skewness)

##     Market   SizeLo30  SizeMed40   SizeHi30  ValueLo30 ValueMed40  ValueHi30
## -0.9599914 -0.1495494 -0.6381693 -1.0442356 -0.7637205 -1.4922693 -0.8256848

# kurtosis: returns
apply(r, MARGIN = 2, kurtosis)

##      Market    SizeLo30   SizeMed40    SizeHi30   ValueLo30  ValueMed40
ValueHi30
## -0.08238205  4.81097118  1.53081600  0.44629946 -0.49069275  2.11458192
1.94780192

# kurtosis: log-returns
apply(r.log, MARGIN = 2, kurtosis)

##     Market   SizeLo30  SizeMed40   SizeHi30  ValueLo30 ValueMed40  ValueHi30
##  1.0559344  0.5799876  0.8924158  1.7487555  0.2067589  5.5715657  1.9045128

# cumulative returns for each strategy
r.cum <- apply(r, MARGIN = 2, function(x) cumprod(1+x))
# convert to xts
r.cum <- xts(r.cum, as.yearmon(rownames(r.cum)))
# plot
plot(r.cum, legend.loc = 'topleft')
```



```
# plot on a log-scale
plot(log(r.cum), legend.loc = 'topleft')
```


log(r.cum)
gen 1927 / gen 2018
Market
SizeLo30
SizeMed40
SizeHi30
ValueLo30
ValueMed40
ValueHi30
gen 1927
gen 1940
gen 1955
gen 1970
gen 1985
gen 2000
gen 2015