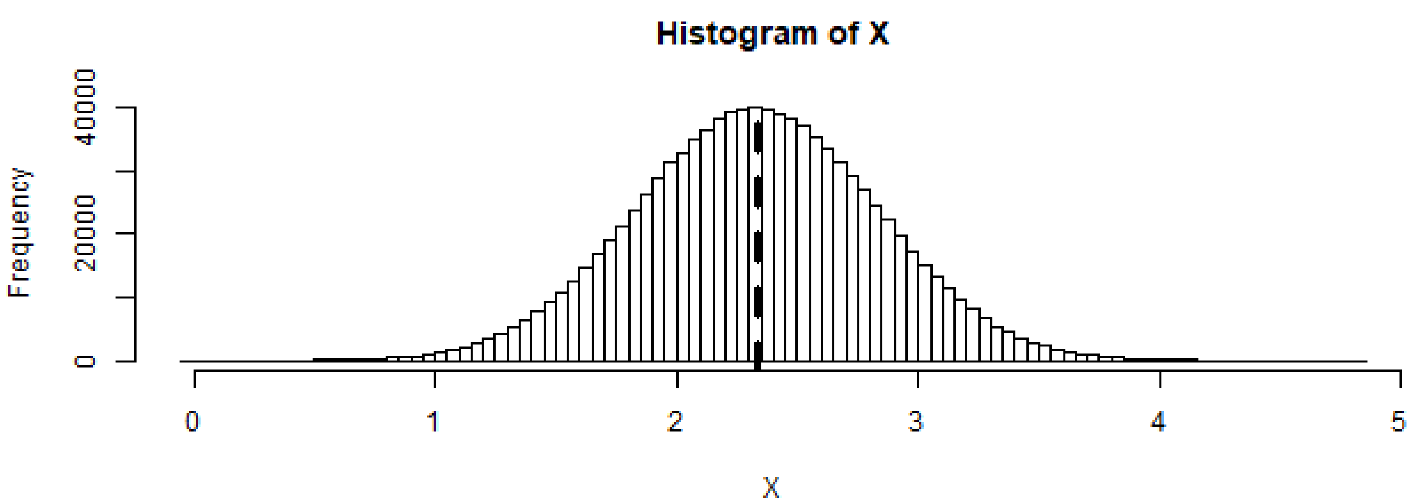
Chapter 4 and 5

Statistics and Probability

```
# set seed for reproducability

set.seed(100)

X <- rnorm(1e6, mean = 2.33, sd = 0.5)
mu <- mean(X)
sd <- sd(X)

par(mfrow = c(1, 1))
hist(X, breaks = 100)
abline(v = mu, lwd = 3, lty = 2)
```




```
set.seed(12)

sample5 <- sample(X, 5, replace = T)
sample10 <- sample(X, 10, replace = T)
sample50 <- sample(X, 50, replace = T)
```

```
sample5
```

```
[1] 1.733402 3.338341 1.975026 2.579086 2.312752
```

```
sample10
```

```
 [1] 2.222012 1.929418 2.378164 1.906355 2.597484 2.083215 1.931390 1.934741
 [9] 2.404611 3.454793
```

```r
sample50
```

```
 [1] 2.434066 2.309894 1.864877 2.832417 2.606947 1.699691 2.387940 3.222830
 [9] 1.352942 1.958900 2.506444 3.122438 3.300801 1.405033 1.273678 1.610530
[17] 2.180945 2.828618 1.999630 2.617088 2.550695 3.108779 2.696634 2.719493
[25] 2.743386 1.987014 2.208630 2.197928 2.174014 1.887448 3.413535 2.302024
[33] 2.082600 2.152472 2.794769 2.865118 1.895745 1.744464 2.409345 2.297630
[41] 3.146226 2.915429 2.098339 2.369153 2.547045 2.216975 2.509051 1.325271
[49] 1.932362 2.719544
```

```r
mean(sample5)
```

```
[1] 2.387721
```

```r
mean(sample10)
```

```
[1] 2.284218
```

```r
mean(sample50)
```

```
[1] 2.350536
```

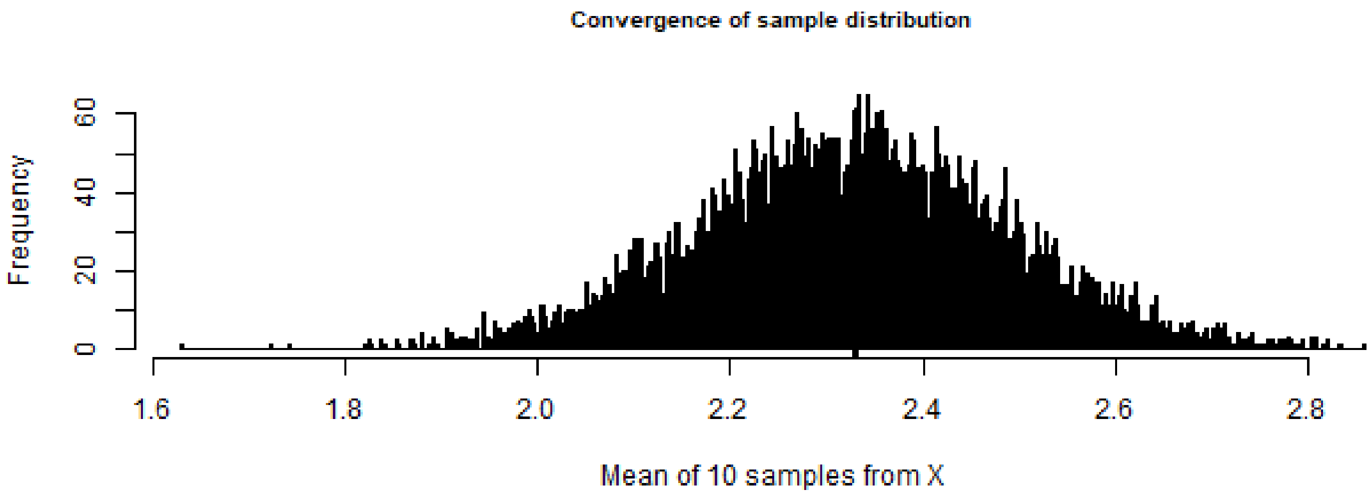
```r
mean(sample(X, 1000, replace = T))
```

```
[1] 2.34422
```

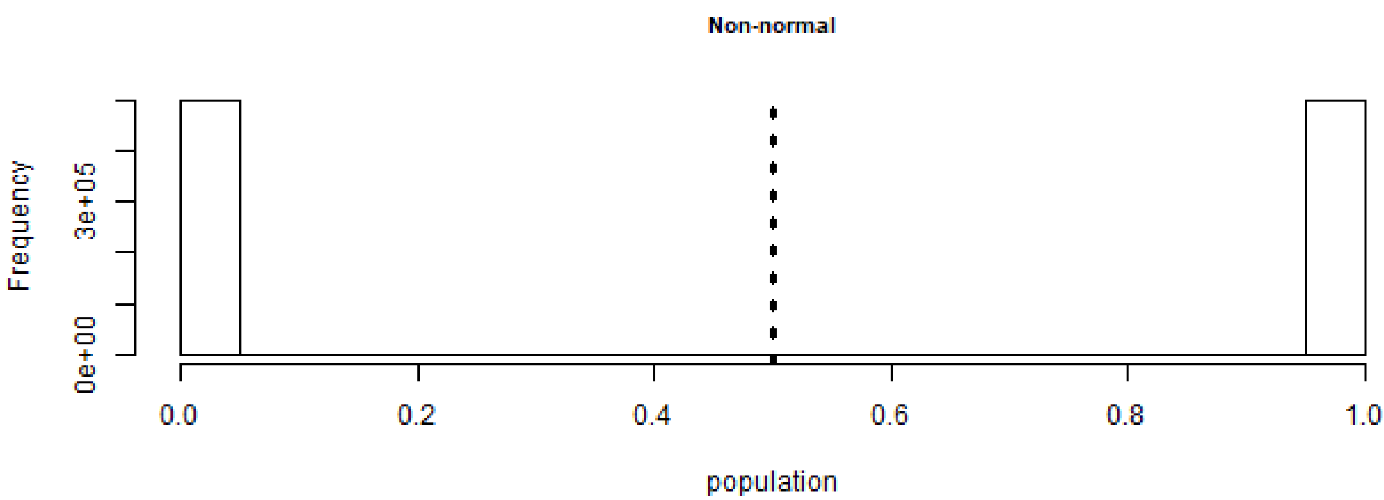
```r
mean(sample(X, 1e5, replace = T))
```

```
[1] 2.32916
```

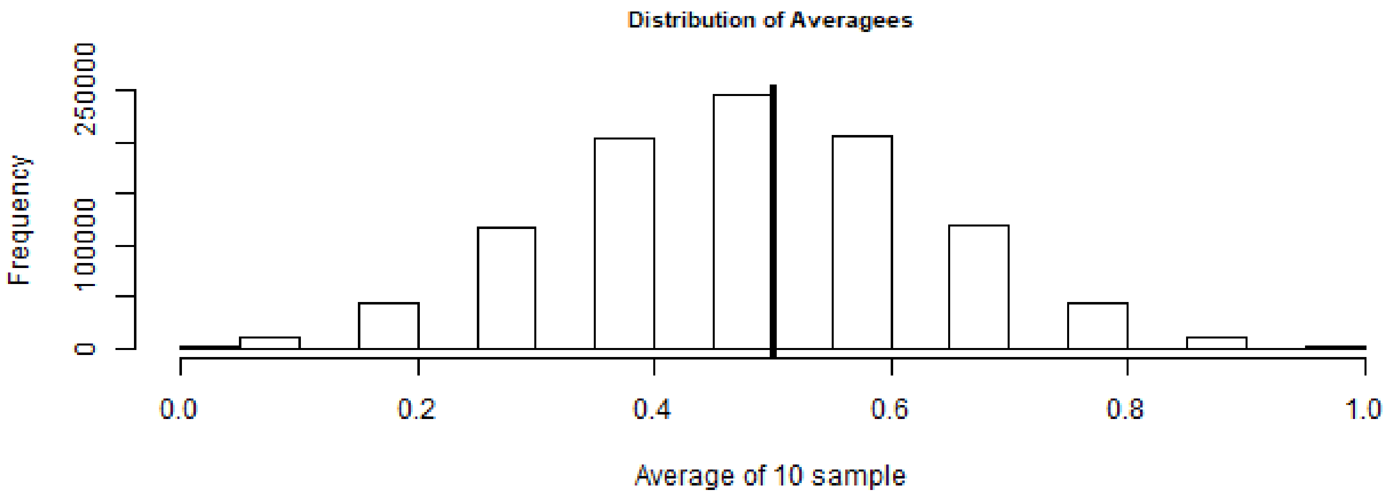
```r
mean_list <- list()

for(i in 1:10000) {
  mean_list[[i]] <- mean(sample(X, 10, replace = T))
}

hist(unlist(mean_list), breaks = 500,
     xlab = "Mean of 10 samples from X",
     main = "Convergence of sample distribution",
     cex.main = 0.8)
abline(v = mu, lwd = 3, col = "black", lty = 2)
```




```r
population <- sample(c(0, 1), 1e6, replace = T)
hist(population, main = "Non-normal", cex.main = 0.8)
abline(v = mean(population), lwd = 3, lty = 3)
```



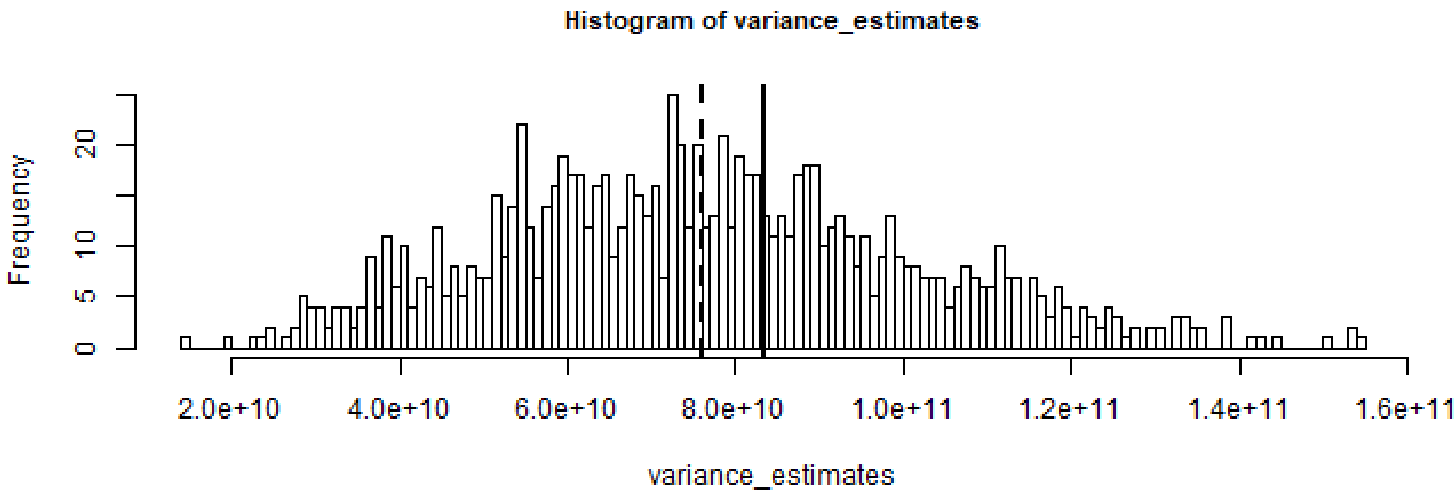

```r
mean_list <- list()
for(i in 1:1e6){
  mean_list[[i]] <- mean(sample(population, 10, replace = T))
}

hist(unlist(mean_list), main = "Distribution of Averagees",
     cex.main = 0.8,
     xlab = "Average of 10 sample")
abline(v = 0.5, lwd = 3)
```




```r
population_variance <- function(x) {
  n <- length(x)
  mu <- sum(x) / n
  sum((x - mu)^2)/n
}
```

```r
population <- as.numeric(1:1e6)

variance <- population_variance(population)
```

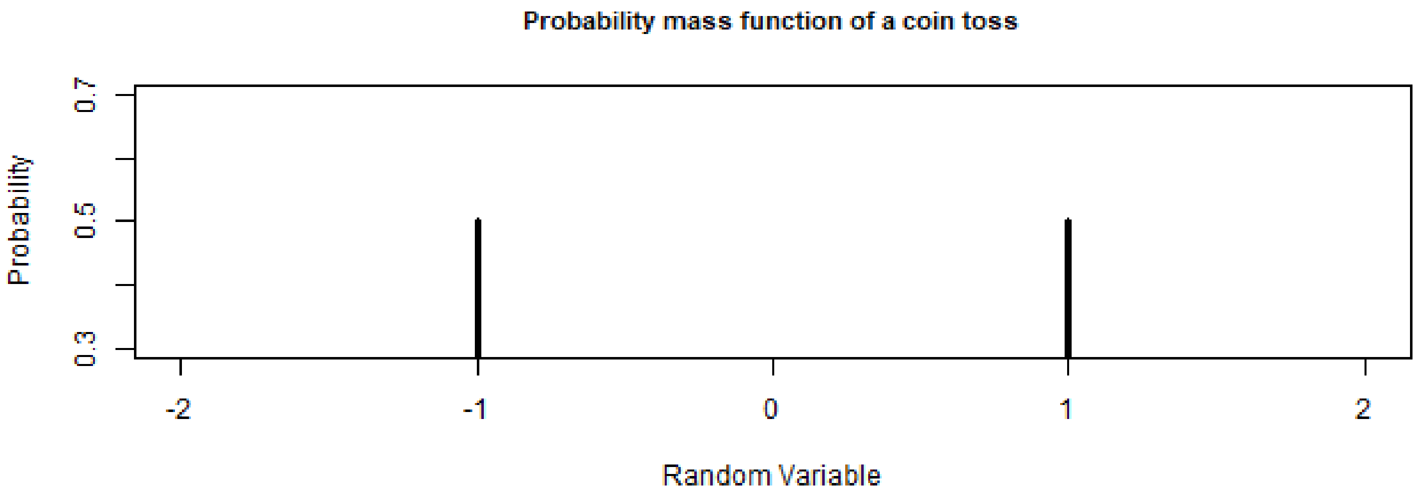
```r
output <- list()

for(i in 1:1000){
  output[[i]] <- population_variance(sample(population, 10, replace = T))
}

variance_estimates <- unlist(output)

hist(variance_estimates, breaks = 100, cex.main = 0.9)
average_variance <- mean(variance_estimates)
abline(v = average_variance, lty = 2, lwd = 2)
abline(v = variance, lwd = 2)
```




```
sample_variance <- function(x) {
  n <- length(x)
  mu <- sum(x) / n
  sum((x - mu)^2) / (n - 1)
}

N <- 1e3
output <- vector(mode = "numeric", length = N)

for(i in 1:N)
{
  output[[i]] <- sample_variance(sample(population, 10, replace = T))
}

sample_variance_estimate <- unlist(output)
average_sample_variance <- mean(sample_variance_estimate)
average_sample_variance
```

```
[1] 85342843826
```

```
plot(c(-1, 1), c(.5, .5), type = "h", lwd = 3,
     xlim = c(-2, 2), main = "Probability mass function of a coin toss",
     ylab = "Probability",
     xlab = "Random Variable",
     cex.main = 0.9)
```




```r
outcomes <- sample(c(0, 1), 1000, replace = T)

set.seed(101)

biased_outcomes <- sample(c(0, 1), 1000, replace = T, prob = c(0.4, 0.6))
```

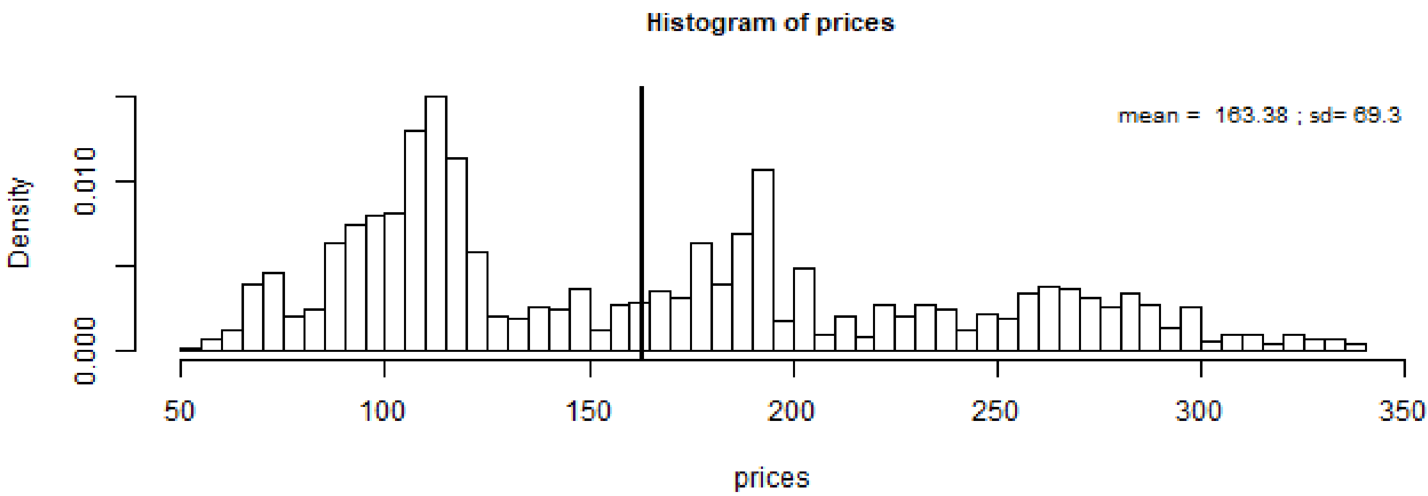
```r
getSymbols("SPY")
```

```
'getSymbols' currently uses auto.assign=TRUE by default, but will
use auto.assign=FALSE in 0.5-0. You will still be able to use
'loadSymbols' to automatically load data. getOption("getSymbols.env")
and getOption("getSymbols.auto.assign") will still be checked for
alternate defaults.

This message is shown once per session and may be disabled by setting
options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
[1] "SPY"
```

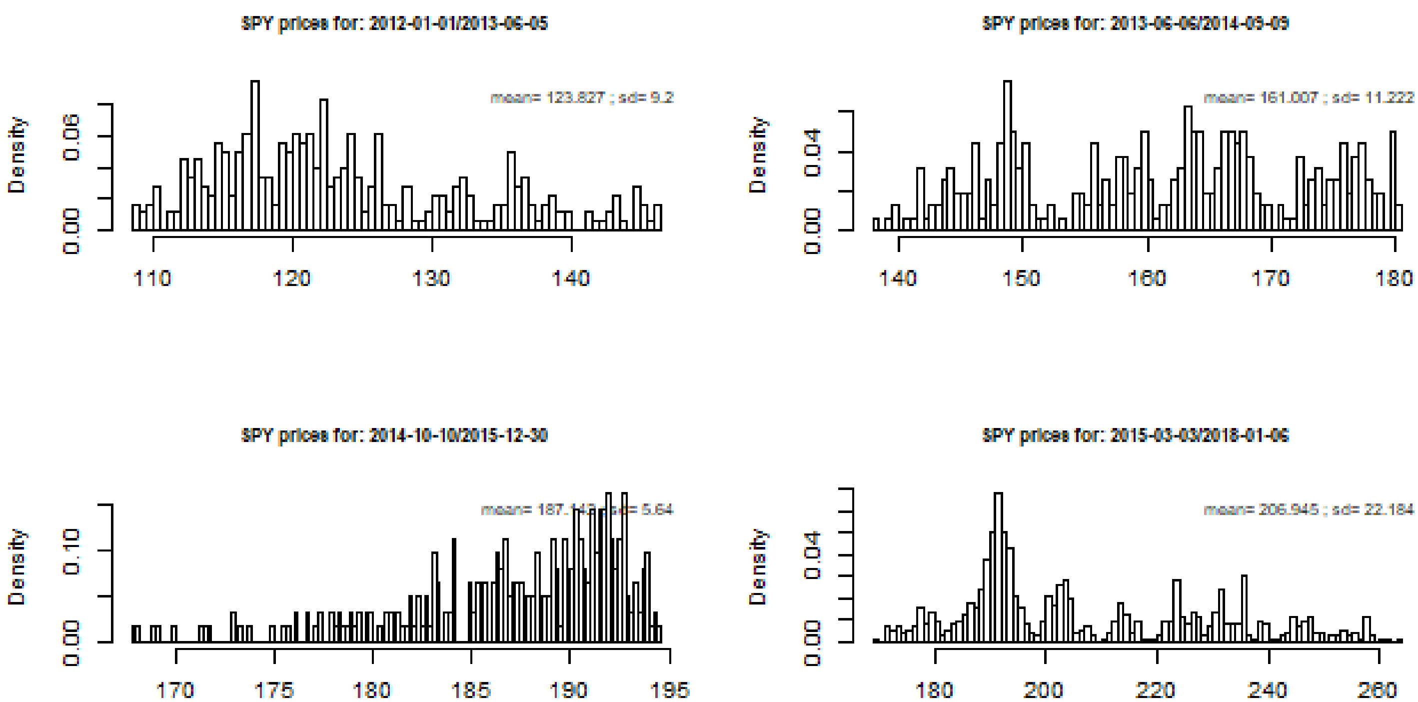
```r
spy <- SPY$SPY.Adjusted

# Extract prices and compute statistics
prices <- spy
mean_prices <- round(mean(prices), 2)
sd_prices <- round(sd(prices), 2)

# Plot the histogram along with a legend
hist(prices, breaks = 100, prob = T, cex.main = 0.9)
abline(v = mean_prices, lwd = 2)
legend("topright", cex = 0.8, border = NULL, bty = "n",
```

```r
      paste("mean = ", mean_prices, "; sd=", sd_prices))
```

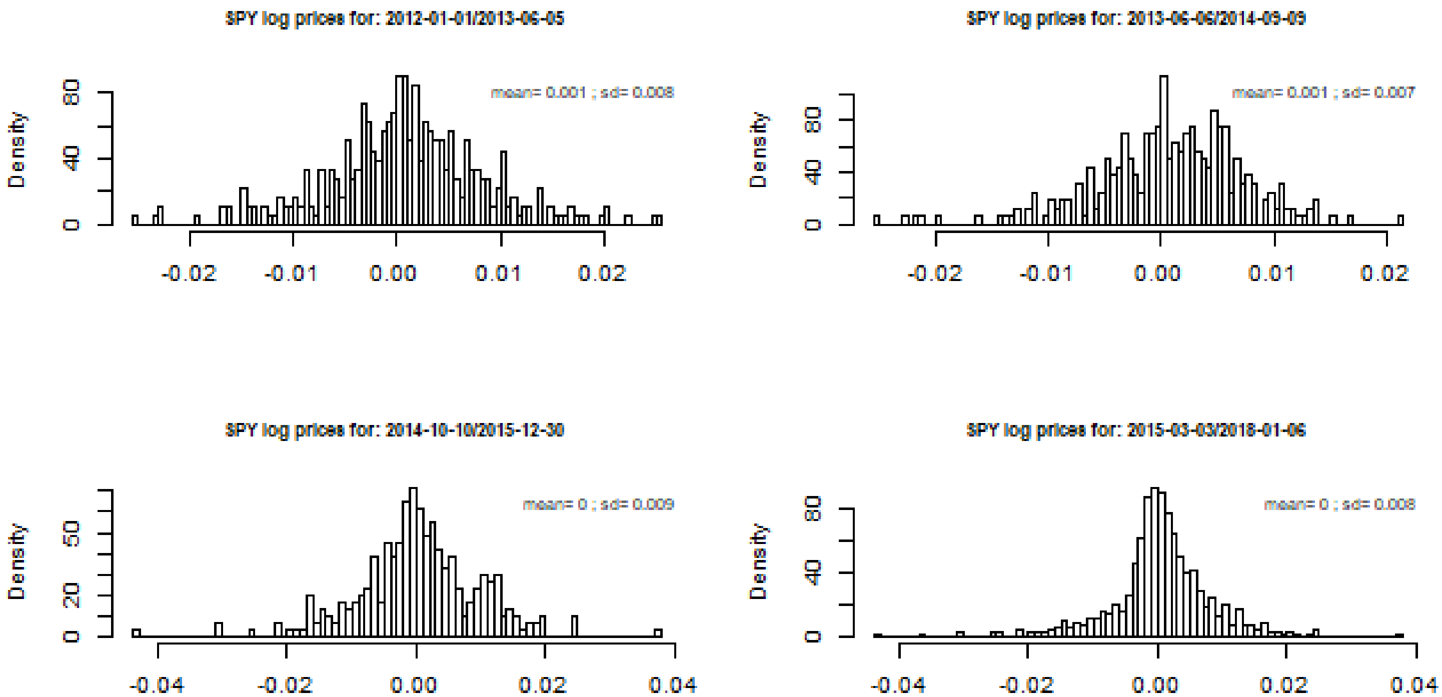



```r
plot_4_ranges <- function(data, start_date, end_date, title) {

  # Set the plot window to be 2 rows and 2 columns
  par(mfrow = c(2, 2))

  for(i in 1:4) {
    # Create a string with the appropriate date range
    range <- paste(start_date[i], "/", end_date[i], sep = "")

    # Create the price vector and necessary statistics
    time_series <- data[range, ]

    mean_data <- round(mean(time_series, na.rm = TRUE), 3)
    sd_data <- round(sd(time_series, na.rm = TRUE), 3)

    # Plot the histogram along with a legend
    hist_title <- paste(title, range)
    hist(time_series, breaks = 100, prob=TRUE,
      xlab = "", main = hist_title, cex.main = 0.8)
    legend("topright", cex = 0.7, bty = 'n',
      paste("mean=", mean_data, "; sd=", sd_data))
  }

  # Reset the plot window
  par(mfrow = c(1, 1))
}
```

```r
# Define start and end dates of interest
begin_dates <- c("2012-01-01", "2013-06-06",
  "2014-10-10", "2015-03-03")
end_dates <- c("2013-06-05", "2014-09-09",
  "2015-12-30", "2018-01-06")

# Create plots
plot_4_ranges(prices, begin_dates,
  end_dates, "SPY prices for:")
```




```r
# Compute log returns
returns <- diff(log(prices))

# Use the same function as before to plot returns rather than prices
plot_4_ranges(returns, begin_dates, end_dates, "SPY log prices for:")
```




```r
test <- ur.kpss(as.numeric(spy))
test
```

```
#######################################
# KPSS Unit Root / Cointegration Test #
#######################################

The value of the test statistic is: 30.3672
```

```r
test@teststat
```

```
[1] 30.36724
```

```r
test@cval
```

```
                10pct  5pct 2.5pct  1pct
critical values 0.347 0.463  0.574 0.739
```

```r
spy_returns <- diff(log(spy))

test.ret <- ur.kpss(as.numeric(spy_returns))
test.ret@teststat
```

```
[1] 0.1639651
```

```
test.ret@cval
```

```
                10pct  5pct 2.5pct  1pct
critical values 0.347 0.463  0.574 0.739
```

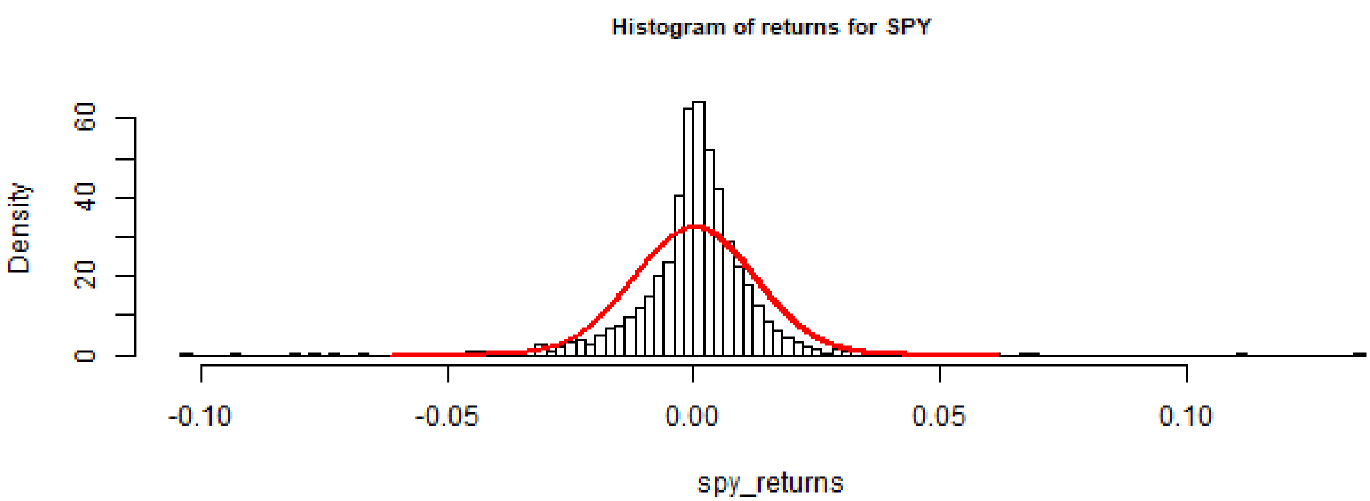
```
test_post_2013 <- ur.kpss(as.numeric(spy_returns['2013::']))

test_post_2013@teststat
```

```
[1] 0.1486586
```

```
test_post_2013@cval
```

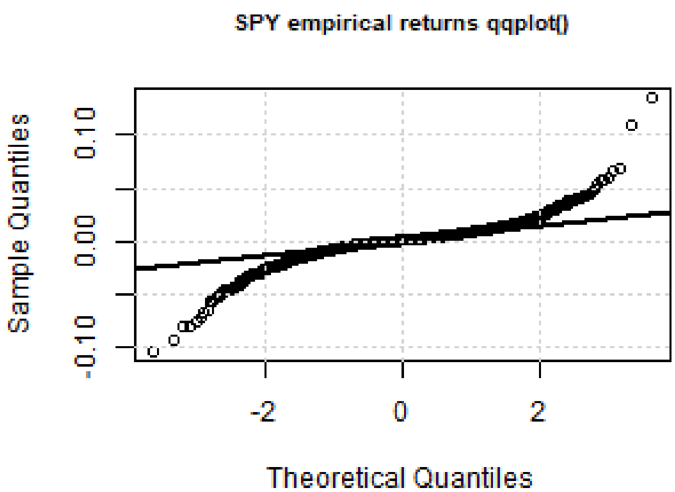
```
                10pct  5pct 2.5pct  1pct
critical values 0.347 0.463  0.574 0.739
```

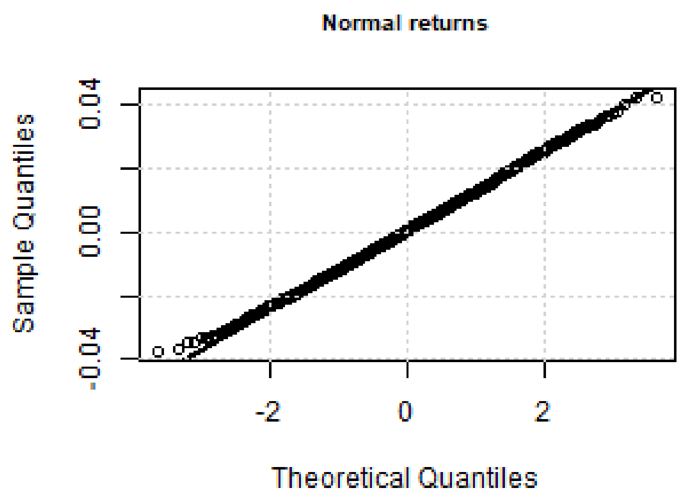
```
# Plot histogram and density
mu <- mean(spy_returns, na.rm = T)
sigma <- sd(spy_returns, na.rm = T)
x <- seq(-5 * sigma, 5 * sigma, length = nrow(spy_returns))

hist(spy_returns, breaks = 100,
     main = "Histogram of returns for SPY",
     cex.main = 0.8, prob = T)
lines(x, dnorm(x, mu, sigma), col = "red", lwd = 2)
```




```
par(mfrow = c(1, 2))

qqnorm(as.numeric(spy_returns),
       main = "SPY empirical returns qqplot()",
       cex.main = 0.8)
```

```
qqline(as.numeric(spy_returns), lwd = 2)
grid()

normal_data <- rnorm(nrow(spy_returns), mean = mu, sd = sigma)

qqnorm(normal_data, main = "Normal returns", cex.main = 0.8)
qqline(normal_data, lwd = 2)
grid()
```






```
answer <- shapiro.test(as.numeric(spy_returns))

answer
```

```

    Shapiro-Wilk normality test

data:  as.numeric(spy_returns)
W = 0.86519, p-value < 2.2e-16
```

```
set.seed(129)

normal_numbers <- rnorm(5000, 0, 1)
ans <- shapiro.test(normal_numbers)

ans
```

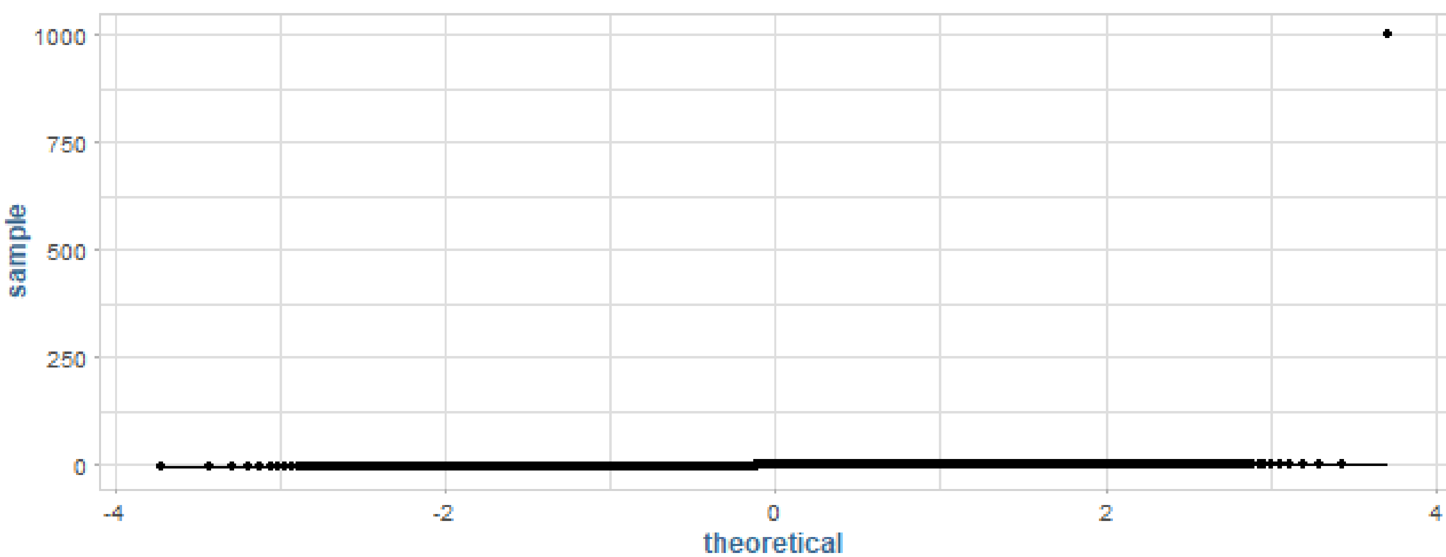
```

    Shapiro-Wilk normality test

data:  normal_numbers
```

```
W = 0.99987, p-value = 0.9964
```

```r
normal_numbers[50] <- 1000

shapiro.test(normal_numbers)
```

```

	Shapiro-Wilk normality test

data:  normal_numbers
W = 0.015741, p-value < 2.2e-16
```

```r
ggplot(data.table(values = normal_numbers), aes(sample = values)) +
  geom_qq() +
  geom_qq_line()
```




```r
getSymbols(c("SPY", "VXX"))
```

```
[1] "SPY" "VXX"
```

```r
date_range <- "2013/1/1::2013/12/31"

spy_prices <- SPY[date_range]$SPY.Close; vxx_prices <- VXX[date_range]$VXX.Close

prices <- merge.xts(spy_prices, vxx_prices, join = "inner")

cor(prices[, c(1, 2)])
```

```
           SPY.Close  VXX.Close
SPY.Close  1.0000000 -0.8289104
VXX.Close -0.8289104  1.0000000
```

```
returns <- data.table(Date = index(prices), SPY = diff(log(prices$SPY.Close)), diff(log(prices$
returns <- returns[-1]

colnames(returns) <- c("Date", "SPY", "VXX")

cor(returns[, c(2, 3)])
```

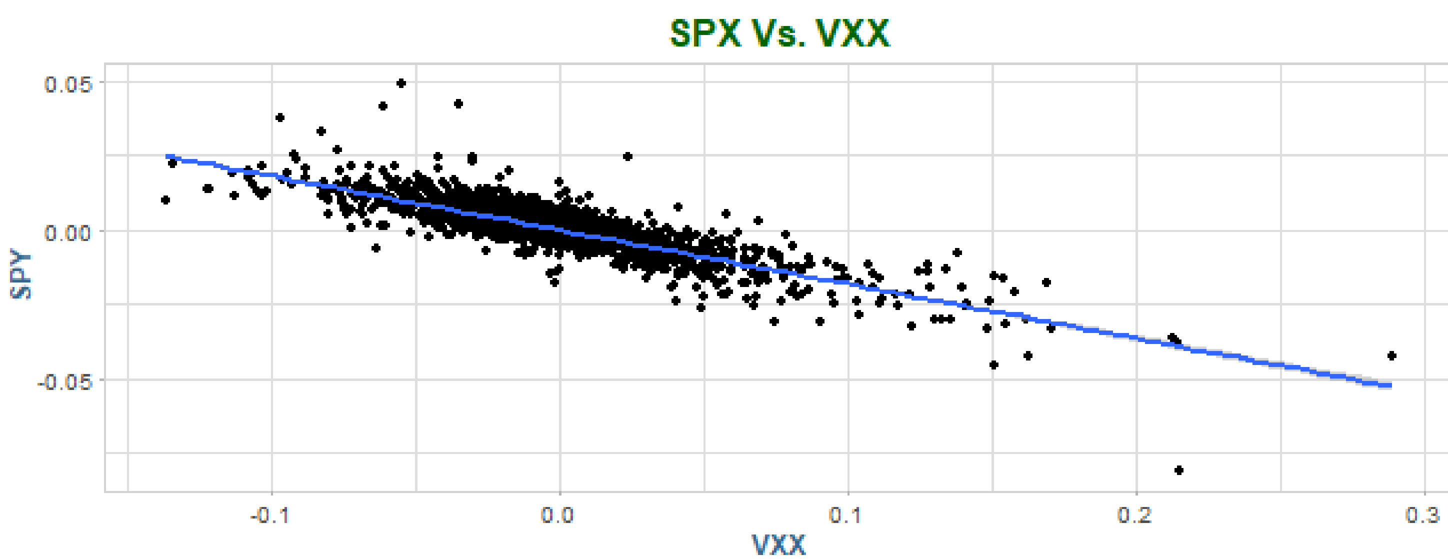
```
           SPY        VXX
SPY  1.0000000 -0.8372323
VXX -0.8372323  1.0000000
```

```
fit1 <- lm(SPY ~ VXX, data = returns)

sqrt(summary(fit1)$r.squared)
```

```
[1] 0.8372323
```

```
ggplot(returns, aes(VXX, SPY)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "SPX Vs. VXX")
```

```
`geom_smooth()` using formula 'y ~ x'
```




```
reg <- lm(SPY.Close ~ VXX.Close - 1, data = prices)
summary(reg)
```

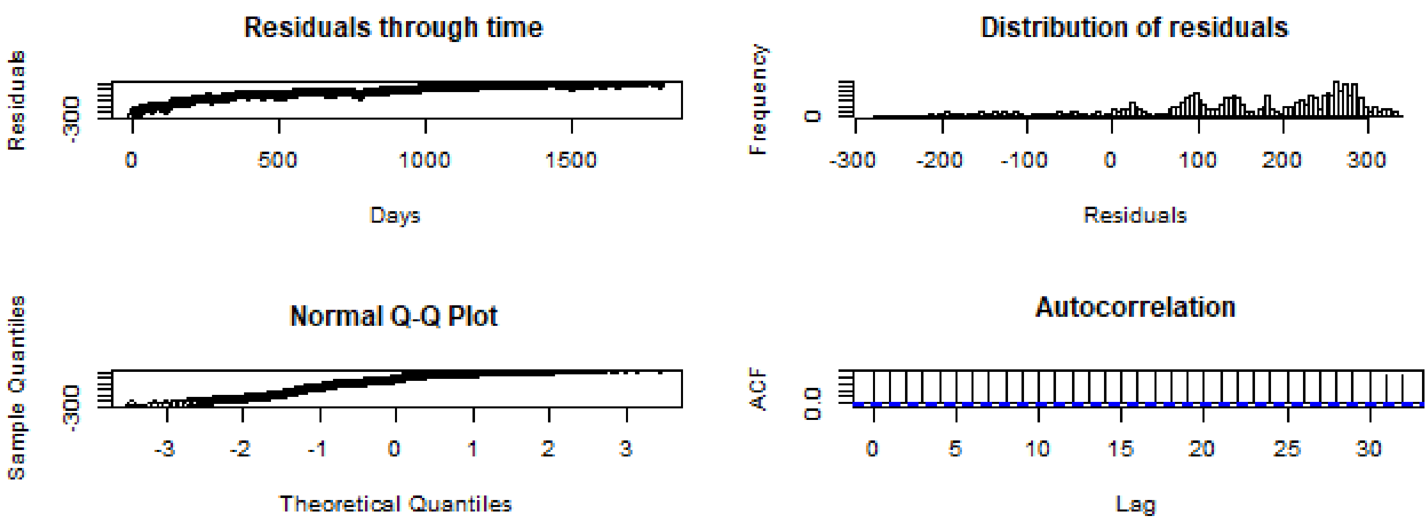
```
Call:
lm(formula = SPY.Close ~ VXX.Close - 1, data = prices)
```

```
Residuals:
    Min      1Q  Median      3Q     Max 
-276.53   87.37  176.37  261.14  335.16 

Coefficients:
          Estimate Std. Error t value Pr(>|t|)    
VXX.Close 0.234546   0.009163    25.6   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 199.8 on 1807 degrees of freedom
Multiple R-squared:  0.2661,    Adjusted R-squared:  0.2657 
F-statistic: 655.2 on 1 and 1807 DF,  p-value: < 2.2e-16
```

```r
# cor
sqrt(summary(reg)$r.squared)
```

```
[1] 0.5158428
```

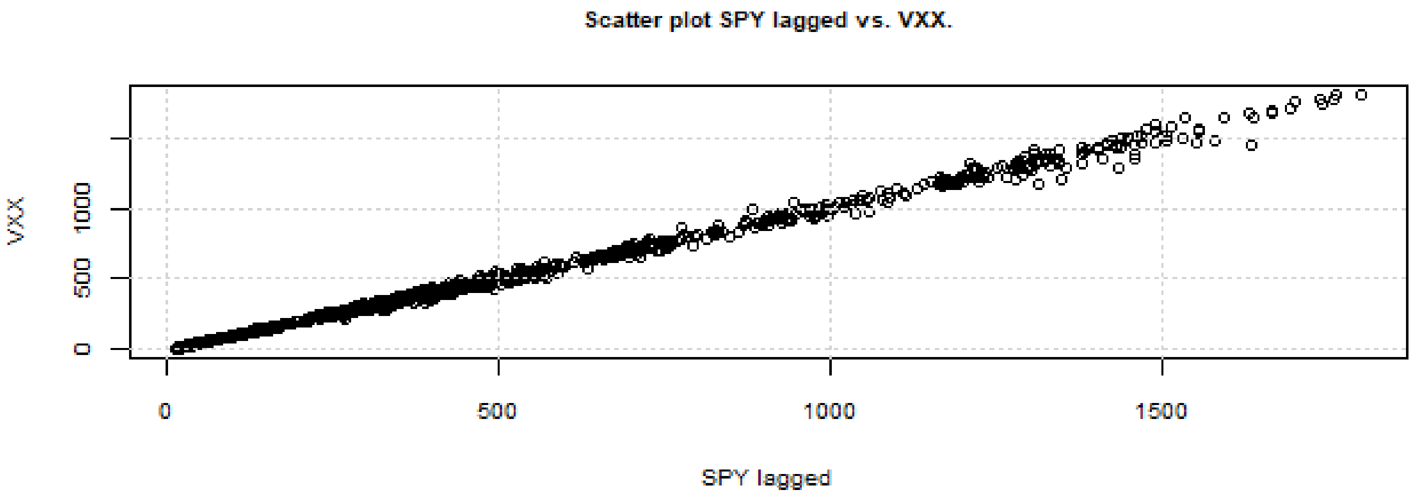
```r
b <- reg$coefficients[1]
a <- reg$coefficients[2]

par(mfrow = c(2, 2))
plot(reg$residuals,
  main = "Residuals through time",
  xlab = "Days", ylab = "Residuals")
hist(reg$residuals, breaks = 100,
  main = "Distribution of residuals",
  xlab = "Residuals")
qqnorm(reg$residuals)
qqline(reg$residuals)
acf(reg$residuals, main = "Autocorrelation")
```



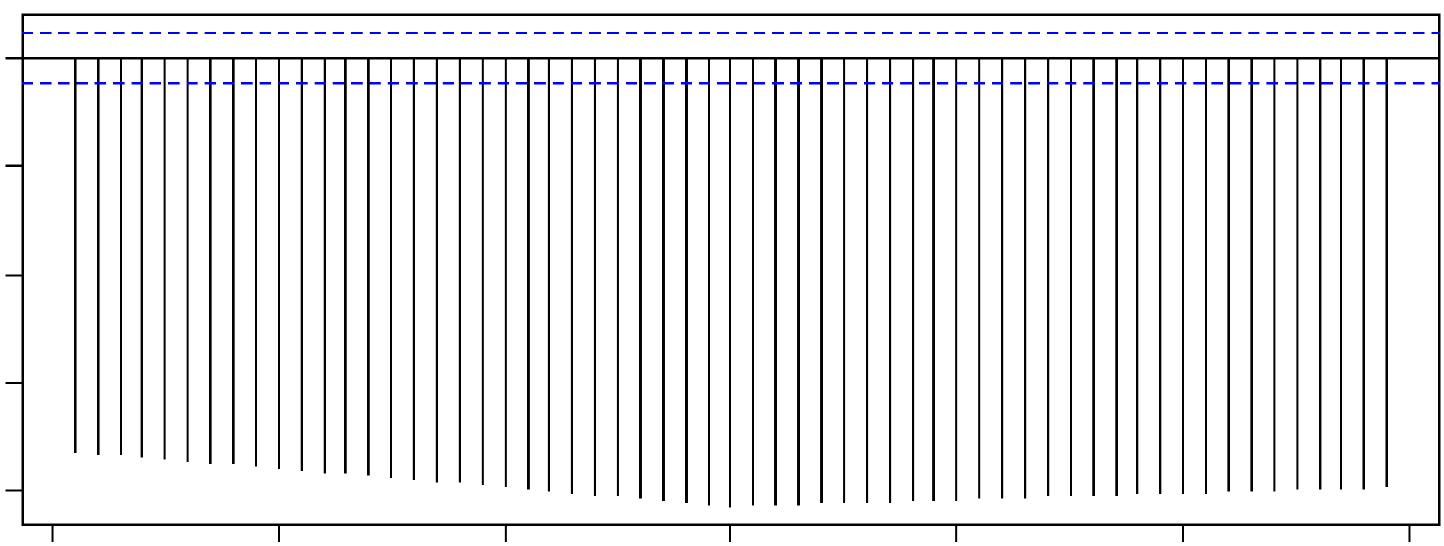

```
vxx_lag_1 <- lag(prices$VXX.Close, k = 1)

head(vxx_lag_1)
```

```
           VXX.Close
2013-01-02        NA
2013-01-03   1795.20
2013-01-04   1800.32
2013-01-07   1763.20
2013-01-08   1761.28
2013-01-09   1739.52
```

```
vxx <- merge(prices$VXX.Close, vxx_lag_1)

par(mfrow = c(1, 1))
# Scatter plot of lagged SPY vs. VXX
plot(as.numeric(vxx[, 1]), as.numeric(vxx[, 2]),
  main = "Scatter plot SPY lagged vs. VXX.",
  xlab = "SPY lagged",
  ylab = "VXX",
  cex.main = 0.8,
  cex.axis = 0.8,
  cex.lab = 0.8)
grid()
```

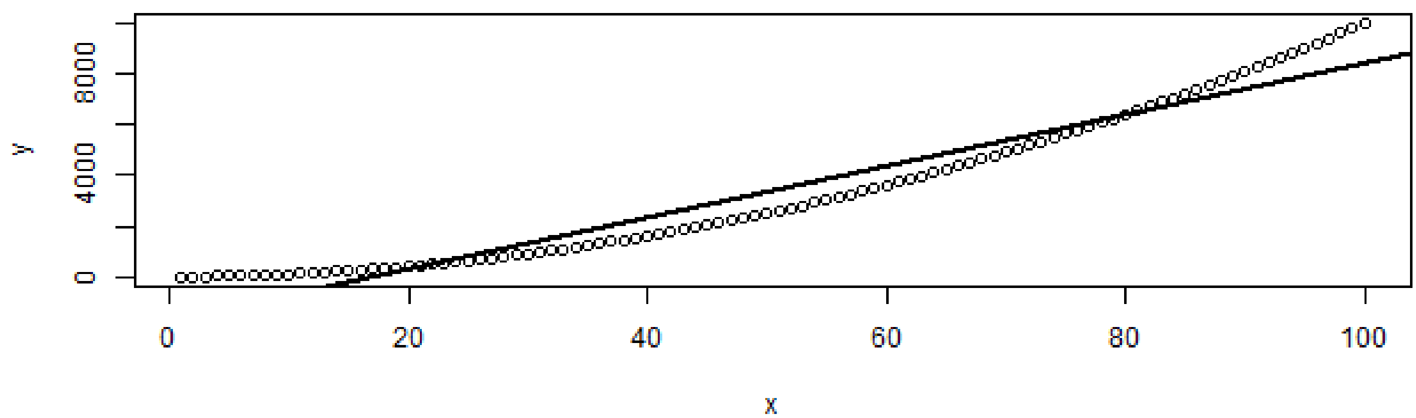



```
par(mfrow = c(1, 1), mar=c(1,1,1,1))

ccf(as.numeric(prices[, 1]), as.numeric(prices[, 2]),
  main = "Cross correlation between SPY and VXX",
  ylab = "Cross correlation", xlab = "Lag", cex.main = 0.8,
  cex.lab = 0.8, cex.axis = 0.8)
```



```
####################################
# The linear in linear regression #
####################################
x <- seq(1:100)
y <- x ^ 2

par(mfrow = c(1, 1))
```

```r
# Generate the plot
plot(x, y)

# Fit the regression
reg_parabola <- lm(y ~ x)

# Superimpose the best fit line on the plot
abline(reg_parabola, lwd = 2)
```



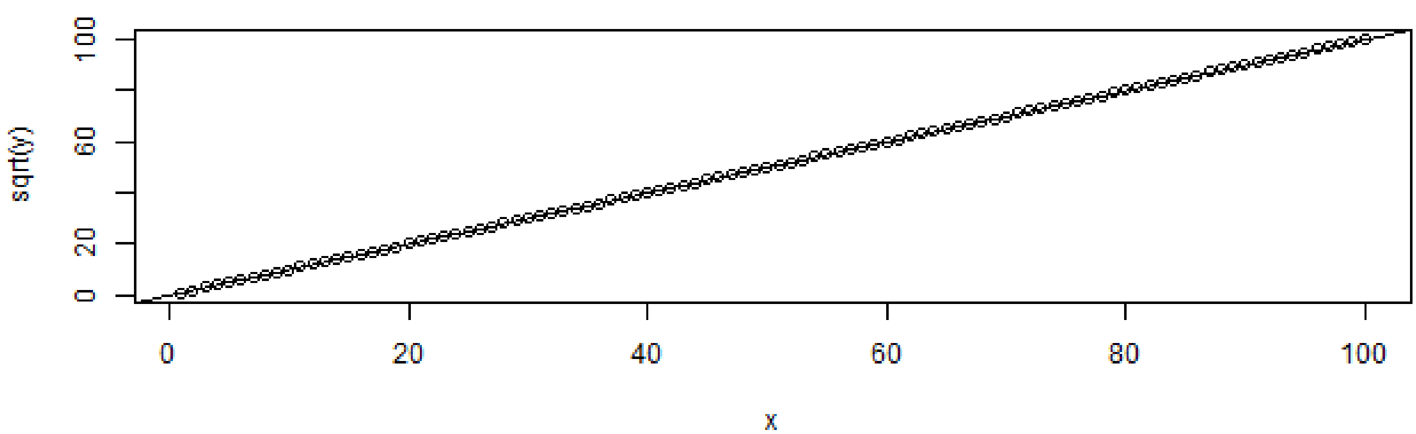

```r
# Look at the results
summary(reg_parabola)
```

```
Call:
lm(formula = y ~ x)

Residuals:
   Min     1Q Median     3Q    Max 
  -833   -677   -208    573   1617 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -1717.000    151.683  -11.32   <2e-16 ***
x             101.000      2.608   38.73   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 752.7 on 98 degrees of freedom
Multiple R-squared:  0.9387,    Adjusted R-squared:  0.9381
```

```
F-statistic:  1500 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
## Coefficients:
##               Estimate   Std. Error t value Pr(>|t|)
## (Intercept)  -1717.000   151.683   -11.32   <2e-16 ***
## x              101.000     2.608    38.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1

## Residual standard error: 752.7 on 98 degrees of freedom
## Multiple R-squared:  0.9387,Adjusted R-squared:  0.9381
## F-statistic:  1500 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
plot(x, sqrt(y))
reg_transformed <- lm(sqrt(y) ~ x)
abline(reg_transformed)
```



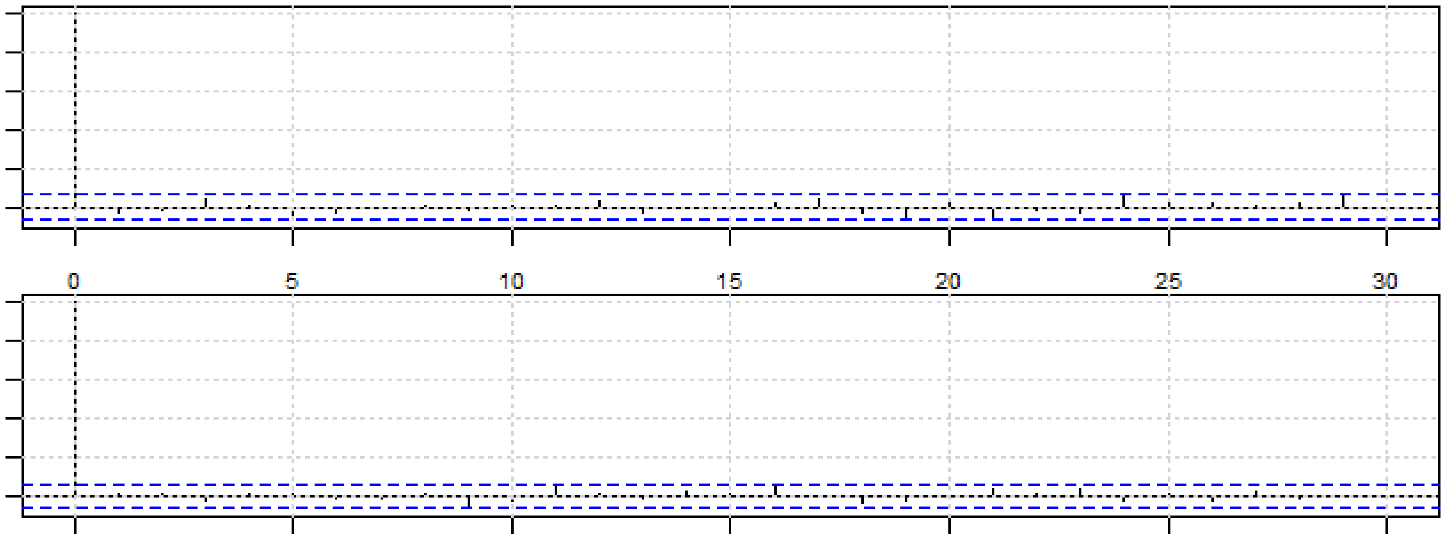

```
summary(reg_transformed)
```

```
Warning in summary.lm(reg_transformed): essentially perfect fit: summary may be
unreliable


Call:
lm(formula = sqrt(y) ~ x)

Residuals:
       Min         1Q     Median         3Q        Max
-2.680e-13 -4.300e-16  2.850e-15  5.302e-15  3.575e-14

Coefficients:
```

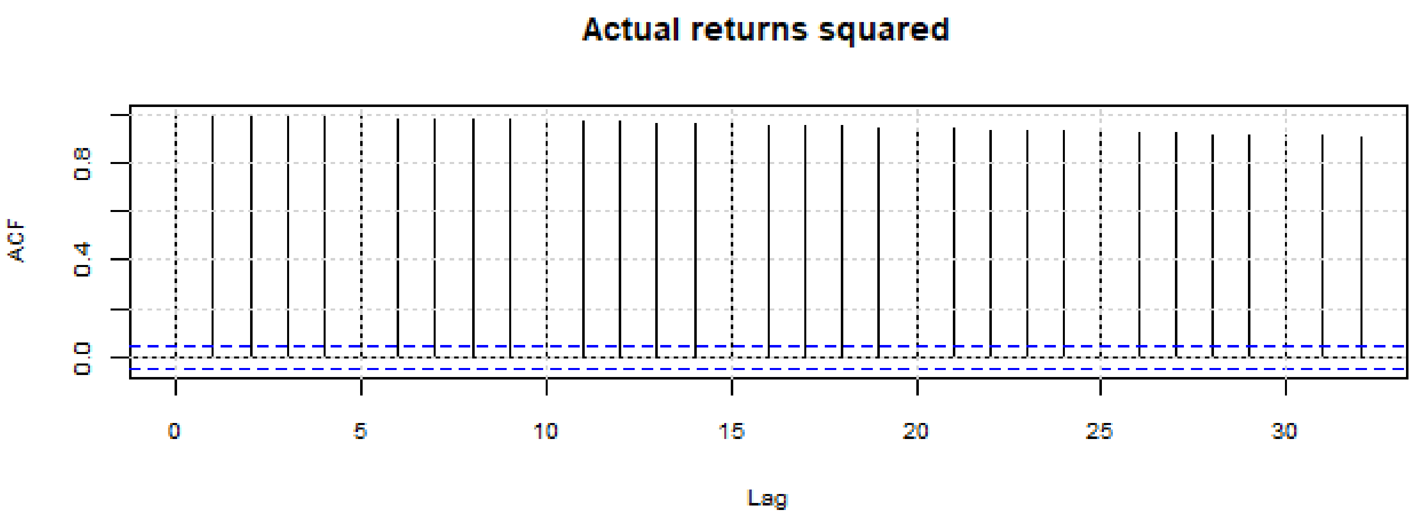
```
              Estimate Std. Error    t value Pr(>|t|)
(Intercept) -5.684e-14  5.598e-15 -1.015e+01   <2e-16 ***
x            1.000e+00  9.624e-17  1.039e+16   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.778e-14 on 98 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:      1
F-statistic: 1.08e+32 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
## Coefficients:
##               Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) -5.684e-14  5.598e-15 -1.015e+01   <2e-16 ***
## x            1.000e+00  9.624e-17  1.039e+16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1

## Residual standard error: 2.778e-14 on 98 degrees of freedom
## Multiple R-squared:      1,Adjusted R-squared:      1
## F-statistic: 1.08e+32 on 1 and 98 DF,  p-value: < 2.2e-16
```

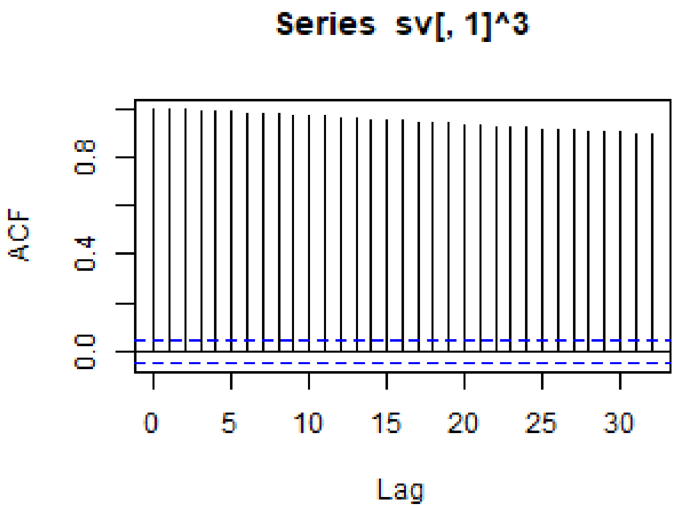
```r
##############
# Volatility #
##############

par(mar=c(1,1,1,1))

# Generate 1000 IID numbers from a normal distribution.
z <- rnorm(1000, 0, 1)

sv <- prices[, c(1, 2)]
# Autocorrelation of returns and squared returns

par(mfrow = c(2, 1))

acf(z, main = "returns", cex.main = 0.8,
  cex.lab = 0.8, cex.axis = 0.8)
grid()

acf(z ^ 2, main = "returns squared",
  cex.lab = 0.8, cex.axis = 0.8)
grid()
```

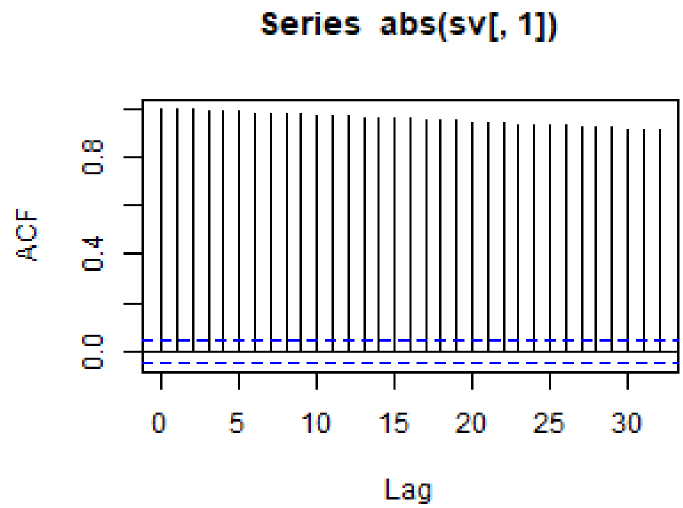



```r
par(mfrow = c(1, 1))
acf(sv[, 1] ^ 2, main = "Actual returns squared",
  cex.main = 0.8, cex.lab = 0.8, cex.axis = 0.8)
grid()
```




```r
par(mfrow = c(1, 2))
acf(sv[, 1]^3)
acf(abs(sv[, 1]))
```


Series sv[, 1]^3
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


Series abs(sv[, 1])
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag