Code

CW2

03 October, 2022

Load libraries

```
library(quantmod)
library(tidyverse)
library(PerformanceAnalytics)
library(timeSeries)
library(tseries)
library(roll)
library(car)
library(MASS)
library(extraDistr)
library(rugarch)
library(rmgarch)
library(BEKKs)
library(QRM)
library(dplyr)
library(rmarkdown)
```

Get stock data

```
rm(list=ls())
ENV.CW2 <- new.env() # Create environment where data are stored

Stocks <- c('SP500', 'JPM') # Stock names
tickers <- c('^GSPC', 'JPM') # Stock tickers
tickers_cleaned <- c('GSPC', 'JPM')
tickers_cleaned <- as.vector(sapply(tickers_cleaned,
                                    FUN = function(x) paste(x, '.Adjusted',
                                                            sep = '')))
# Function merge violates the orders of columns; tickers_cleaned is used to restore the order

# Get the data from Yahoo Finance
Symbols <- getSymbols(Symbols = tickers, src = 'yahoo',
                      from = "1995-01-01",
                      to = "2022-09-20",
                      env = ENV.CW2)

# Create one XTS object containing adjusted prices of all stocks
Adjusted_Stock_Prices <- do.call(merge, eapply(env = ENV.CW2, Ad))
Adjusted_Stock_Prices <- Adjusted_Stock_Prices[, tickers_cleaned] # Restore the right order of columns
names(Adjusted_Stock_Prices) <- Stocks # Change names from tickers to real names
```

Convert into log returns

```
log_returns <- diff(log(Adjusted_Stock_Prices)) # Compute daily log
        returns
log_returns <- na.omit(log_returns) # Remove rows containing na's
# Computing Statistics
AvgRet = colMeans(log_returns)
StdDevRet = colSds(log_returns)
MaxRet = colMaxs(log_returns)
MinRet = colMins(log_returns)
SkewRet = colSkewness(log_returns)
KurtRet = colKurtosis(log_returns)
DailyStats <- as.table(rbind(AvgRet, StdDevRet, MaxRet, MinRet, Ske
        wRet, KurtRet))
knitr::kable(DailyStats, digits=4)
```


	SP500	JPM
AvgRet	0.0003	0.0004
StdDevRet	0.0121	0.0237
MaxRet	0.1096	0.2239
MinRet	-0.1277	-0.2323
SkewRet	-0.4259	0.2069
KurtRet	10.2764	12.7092

Fat Tails

Histogram

Returns are not normally distributed. Financial data exhibits fat tails, which means that extreme values, both positive and negative, are seen more frequently than what we would expect if the data followed a normal distribution. Also, in financial data most days are uneventful, so we see a higher frequency of points around zero than in a normal distribution.

Plotting returns histogram against normal distribution

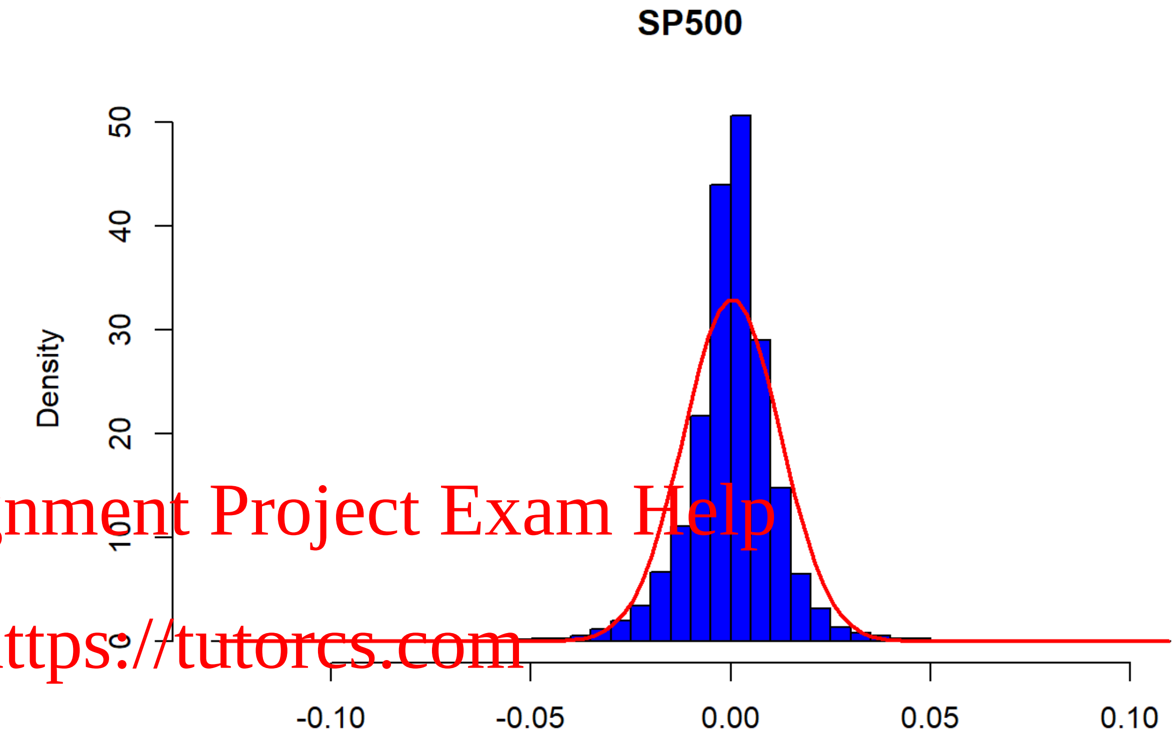
To show this graphically, we will plot the histogram of returns and overlay a normal distribution with the same mean and standard deviation:


```
for (i in 1:2) {
  par(mfrow=c(1,1))
  seq_curve <- seq(min(log_returns[, i]), max(log_returns[, i]), length = 100)
  normal_density <- dnorm(x = seq_curve, mean = AvgRet[i], sd = StdDevRet[i])
  hist(x = log_returns[, i], prob = TRUE, breaks = 80,
       main = Stocks[i], col = 'blue', xlab = '')
  lines(seq_curve, normal_density, lwd = 2, col = 'red')
}
```

Fat Tails

Volatility Clusters

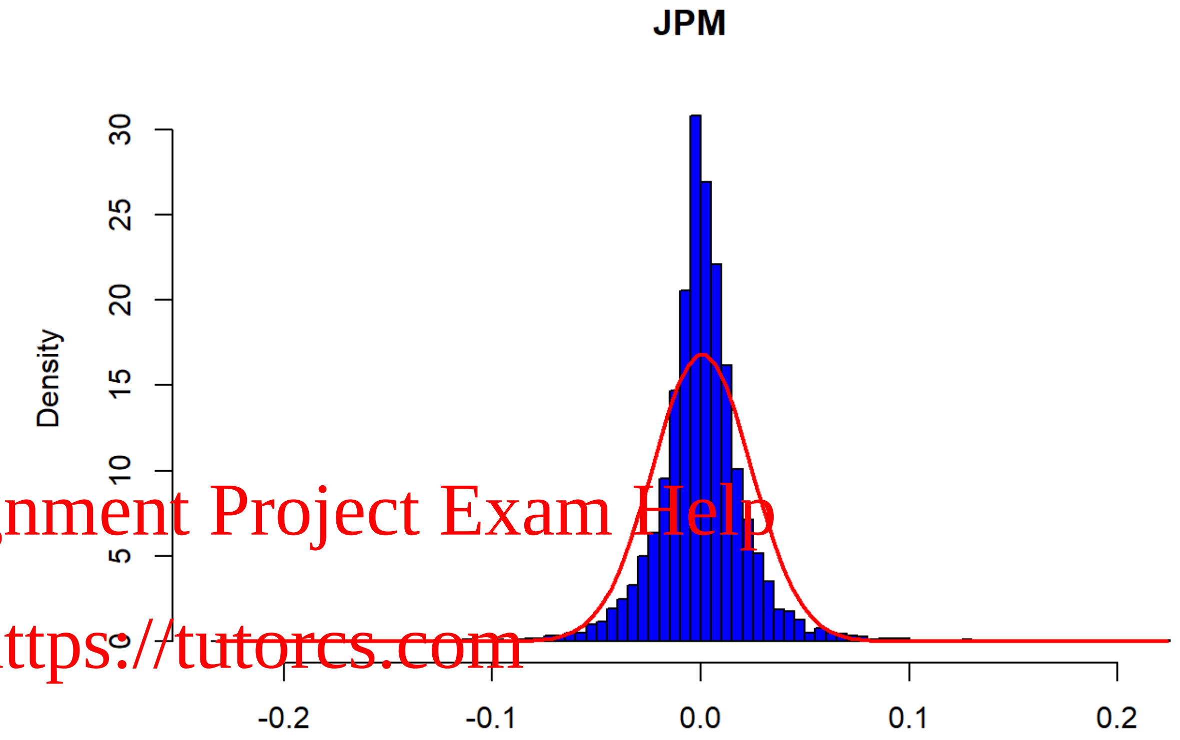
Conditional volatility model: MA



Fat Tails

Volatility Clusters

Conditional volatility model: MA



Plotting returns histogram against a generalized t-distribution (fatter tails)

Other distributions that exhibit fat tails might be more suitable to work with. A common choice is the Student-t distribution.

Fat Tails

Volatility Clusters

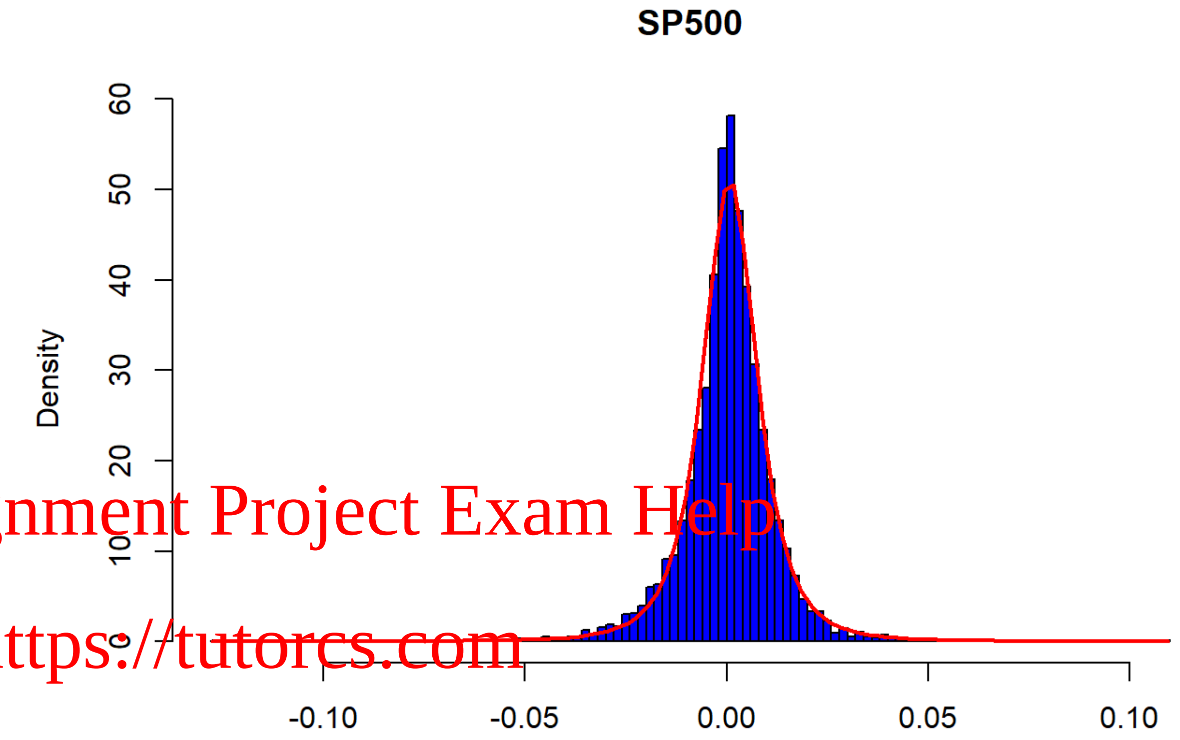
Conditional volatility model: MA

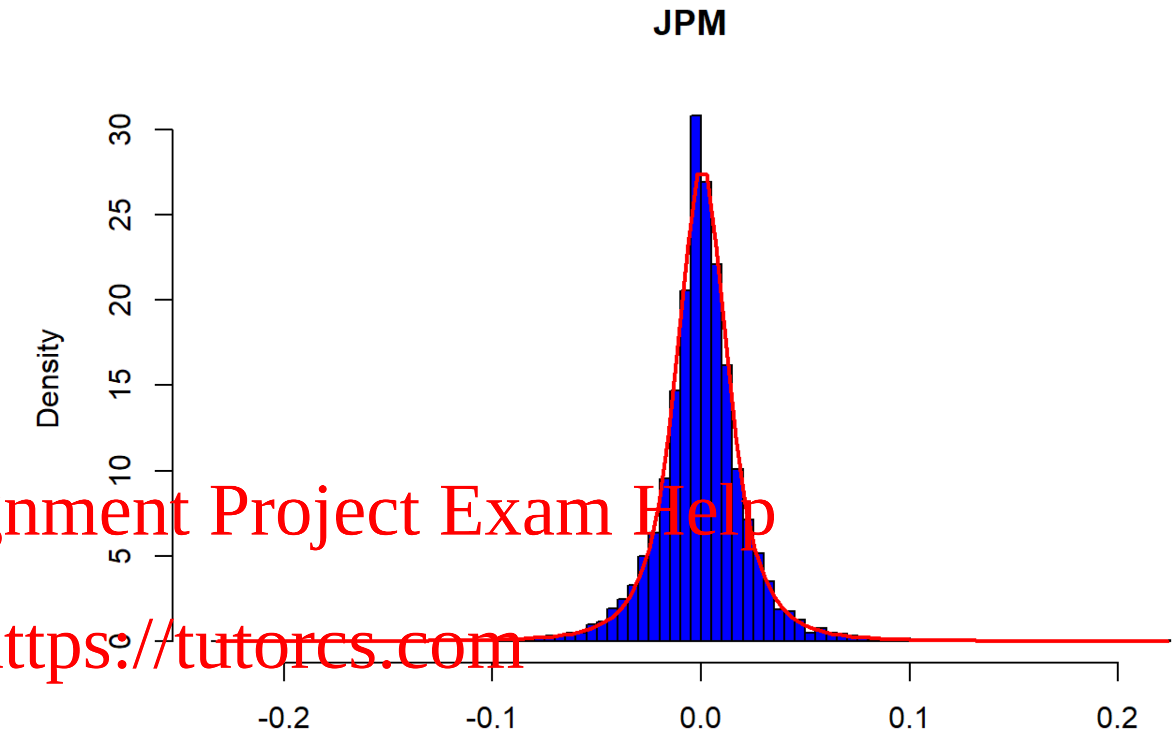
```
for (i in 1:2) {
  fit_t <- fitdistr(log_returns[, i], densfun = 't') # Find t Location-Scale Distribution parameters that best fit data
  mu <- fit_t$estimate['m']
  sigma <- fit_t$estimate['s']
  df <- fit_t$estimate['df']
  seq_curve <- seq(min(log_returns[, i]), max(log_returns[, i]), length = 100)
  tscaled_density <- dlst(x = seq_curve, df = df, mu = mu, sigma = sigma) # Create Location-Scale Distribution density
  hist(x = log_returns[, i], prob = TRUE, breaks = 100,
       main = Stocks[i], col = 'blue', xlab = '')
  lines(x = seq_curve, y = tscaled_density, type = 'l', lwd = 2, col = 'red')
}
```

Fat Tails

Volatility Clusters

Conditional volatility model: MA





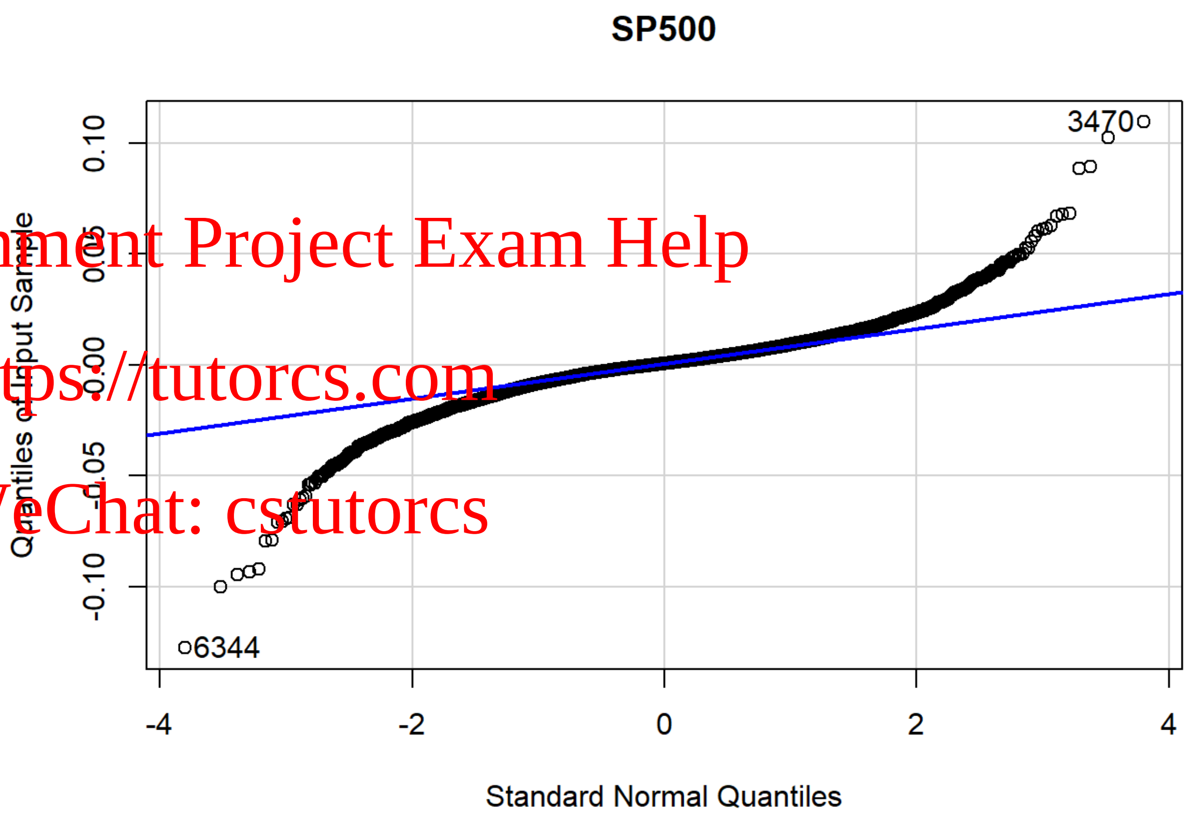
QQ Plot

A graphical way to see how our data fits different distributions is by using a Quantile-Quantile Plot. This method plots the quantiles of our data against the quantiles of a specified distribution. If the distribution is a good fit for our data, we will see the data points aligning with a diagonal line.

To build QQ plots, we will use the qqPlot function from the car package:

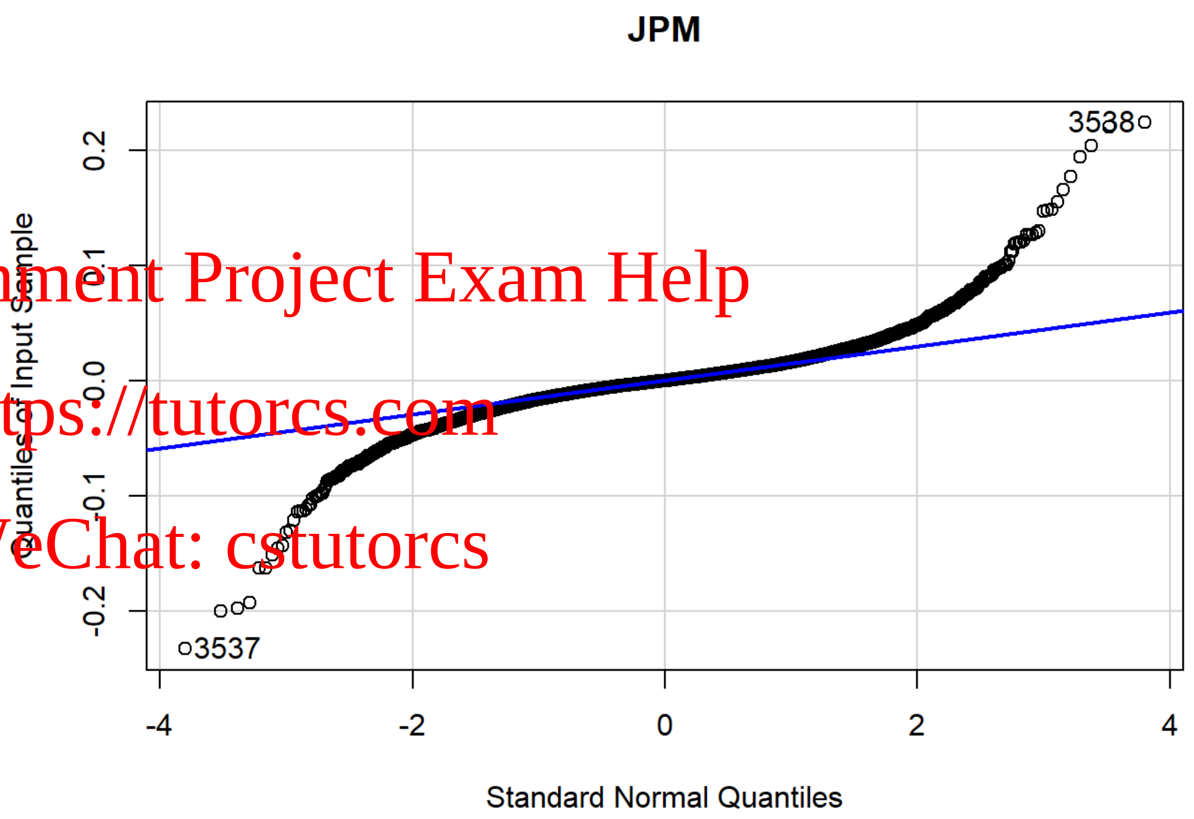
- Fat Tails
- Volatility Clusters
- Conditional volatility model: MA

```
qqPlot(as.vector(log_returns[, 1]), xlab = 'Standard Normal Quantiles',
       ylab = 'Quantiles of Input Sample', main = Stocks[1],
       envelope = FALSE) # Be careful, data format needs to be numeric, hence: as.vector()
```




```
## [1] 6344 3470
```

```
qqPlot(as.vector(log_returns[, 2]), xlab = 'Standard Normal Quantil
        es',
       ylab = 'Quantiles of Input Sample', main = Stocks[2],
       envelope = FALSE)
```




```
## [1] 3537 3538
```

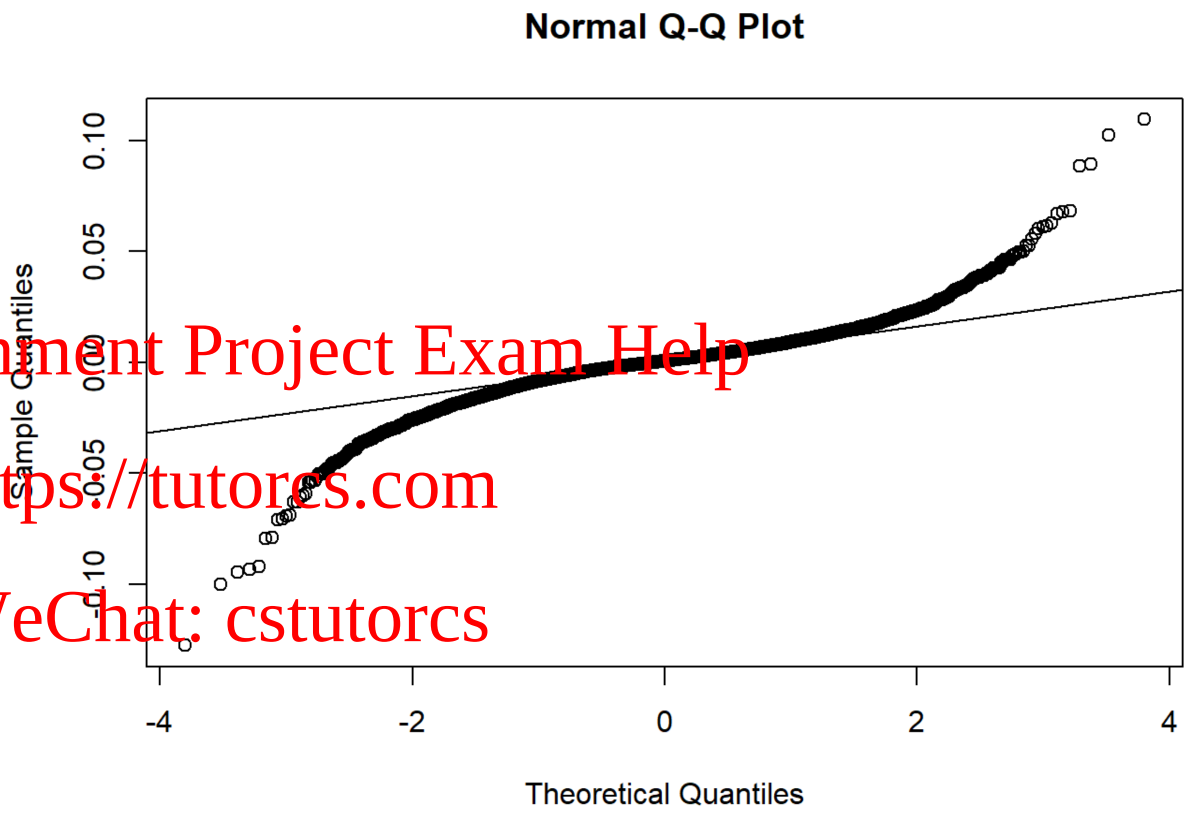
Alternative approach:

Fat Tails

Volatility Clusters

Conditional volatility model: MA

```
qqnorm((log_returns[, 1]))
qqline(log_returns[, 1])
```



Statistical test for normality

It is visually obvious that the returns do not follow a normal distribution with matched moments. However, visually obvious is not a rigorous statement, so we should perform a statistical test to prove this.

- Fat Tails
- Volatility Clusters
- Conditional volatility model: MA

To see if a vector of numbers could have been drawn from a normal distribution, we will use the Jarque-Bera test, which uses skewness and kurtosis. The test statistic of the Jarque-Bera test asymptotically follows a chi-square distribution with two degrees of freedom. The null hypothesis, H_0, of the test is that the skewness and excess kurtosis of a distribution are jointly zero, which is the equivalent of a Normal Distribution. This test can be directly implemented with the jarque.bera.test() function from the tseries package:

```
jarque.bera.test(log_returns[, 1])
```



```
##
##  Jarque Bera Test
##
## data:  log_returns[, 1]
## X-squared = 30934, df = 2, p-value < 2.2e-16
```

```
jarque.bera.test(log_returns[, 2])
```

```
##
##  Jarque Bera Test
##
## data:  log_returns[, 2]
## X-squared = 47040, df = 2, p-value < 2.2e-16
```

```
CV = qchisq(p = 0.95, df = 2)
CV
```

```
## [1] 5.991465
```

To understand the output of a test, we can look at the p-value. A p-value below 0.05 tells us that we have enough evidence to reject the null hypothesis H0 with a confidence level of 95%. Statistically, the p-value is the inverse of the test-statistic under the CDF of the asymptotic distribution.

The p-value of this test is basically zero, which means that we have enough evidence to reject the null hypothesis that our data has been drawn from a Normal distribution.

Volatility Clusters

The autocorrelation function shows us the linear correlation between a value in our time series with its different lags. For example, if tomorrow's return can be determined with today's value, we would expect to have a significant autocorrelation of lag one.

The acf function plots the autocorrelation of an ordered vector. The horizontal lines are confidence intervals, meaning that if a value is outside of the interval, it is considered significantly different from zero.

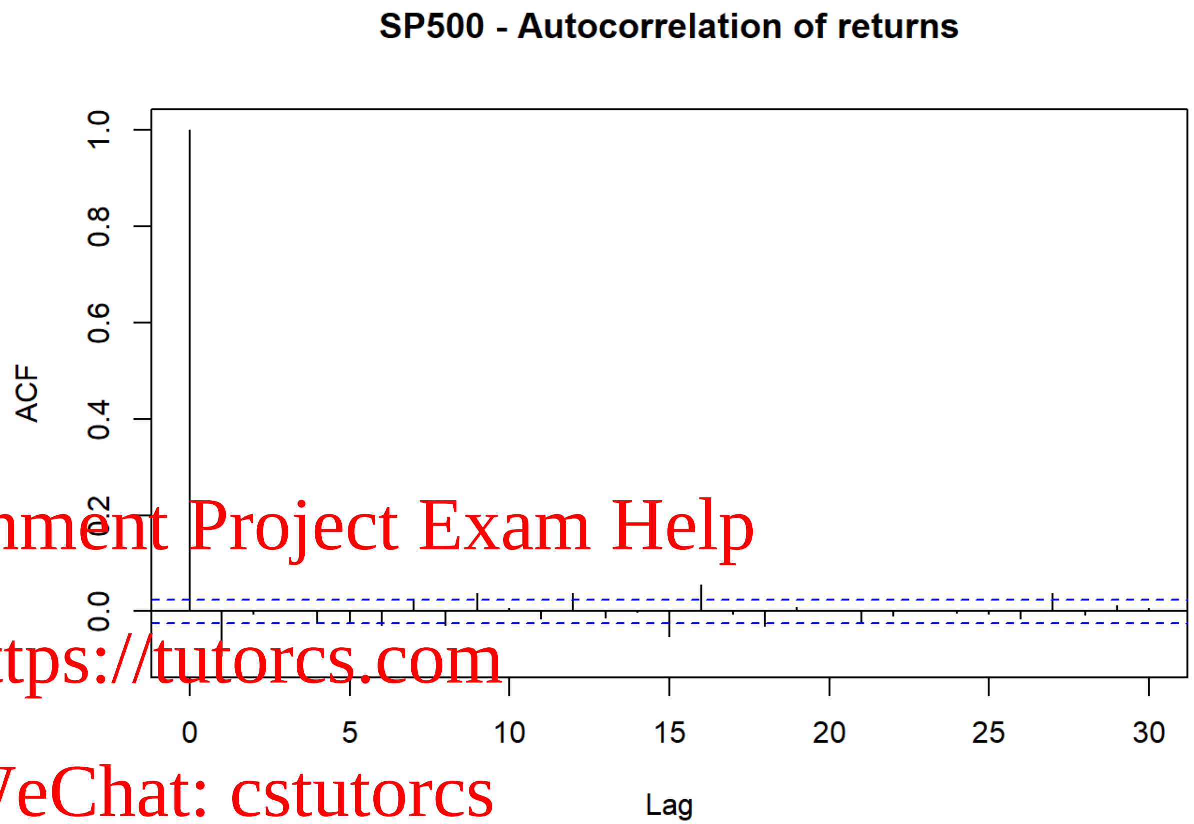
Acf of returns


```
acf(x = log_returns[, 1], lag.max = 30,
    main = paste(Stocks[1], '- Autocorrelation of returns'))
```

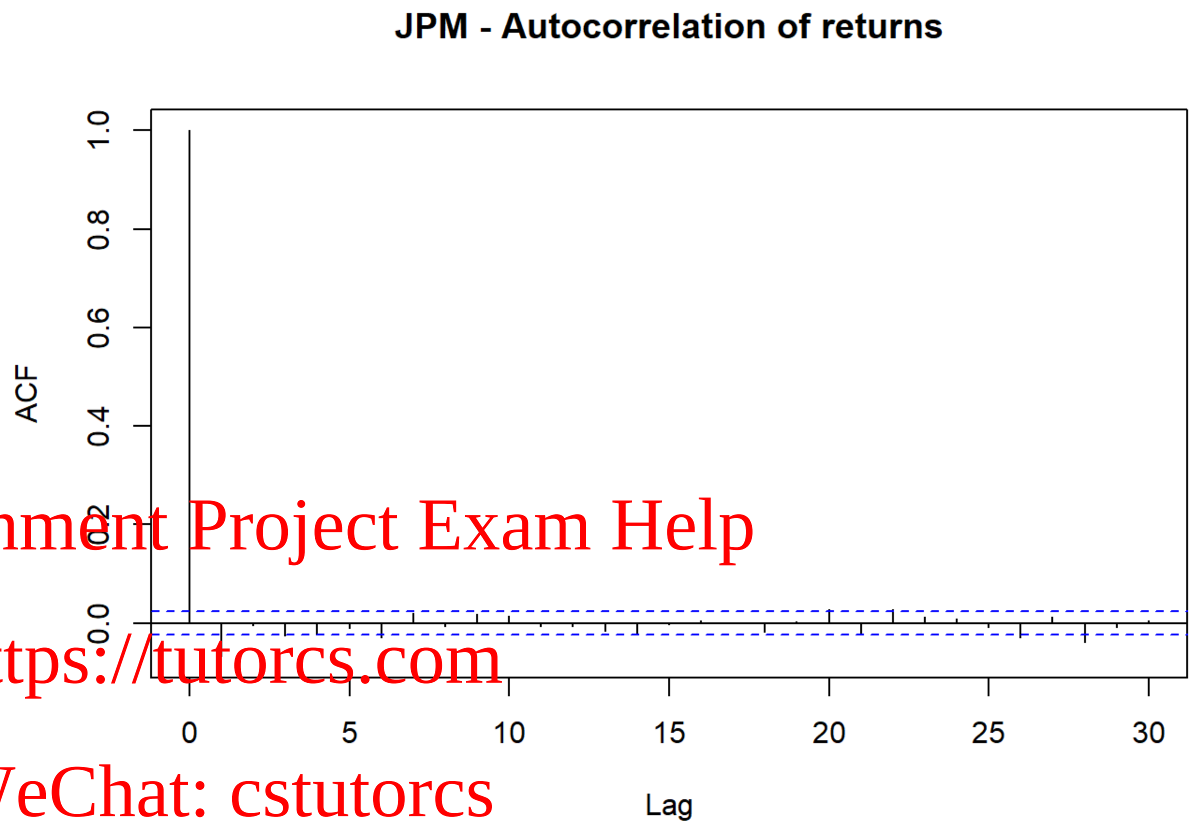
Fat Tails

Volatility Clusters

Conditional volatility model: MA




```
acf(x = log_returns[, 2], lag.max = 30,
    main = paste(Stocks[2], '- Autocorrelation of returns'))
```



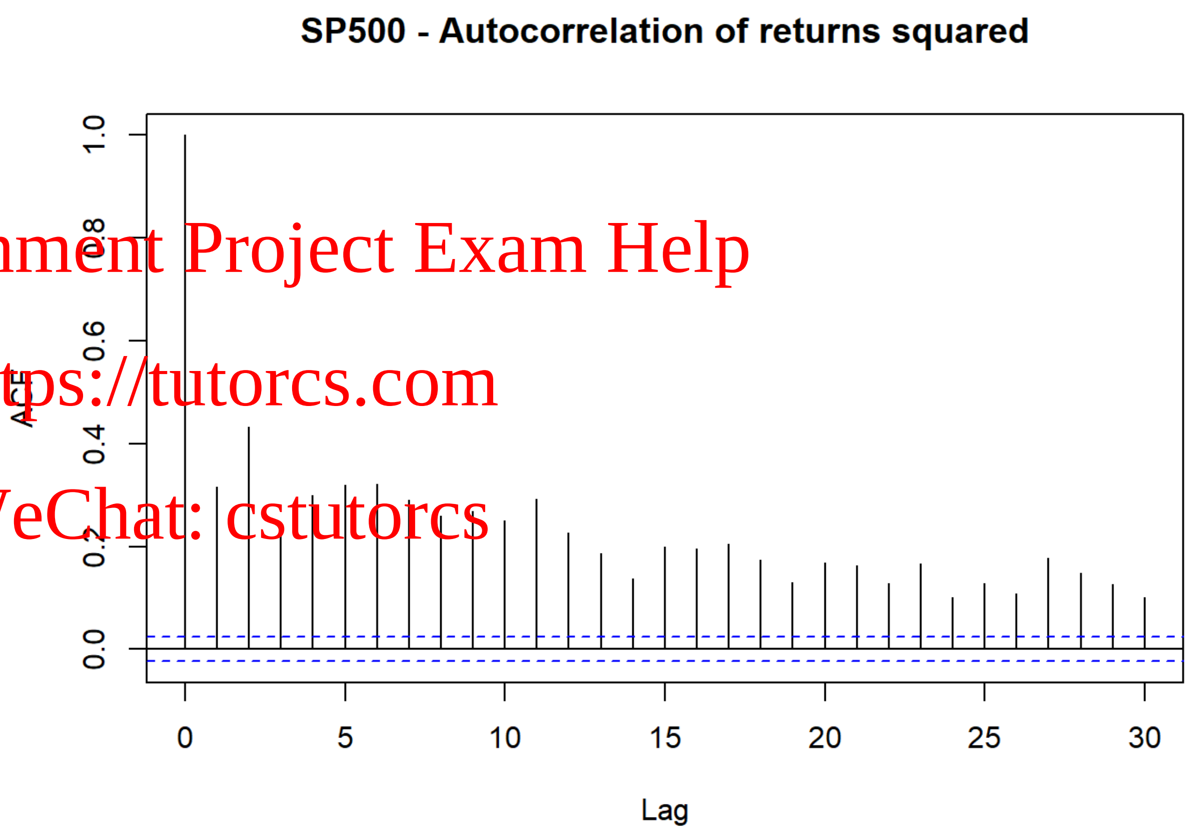
The autocorrelation of lag 0 is always 1 (every value is perfectly correlated with itself), but apart from that, we see no significant values. Remember than with a 95% confidence interval, we would expect to see 1 out of every 20 values to show significance out of pure chance.

This shows us good evidence that you cannot easily forecast stock prices. If there was a clear autocorrelation, you could build trading strategies to create profit, but the market makes sure this is not the case.

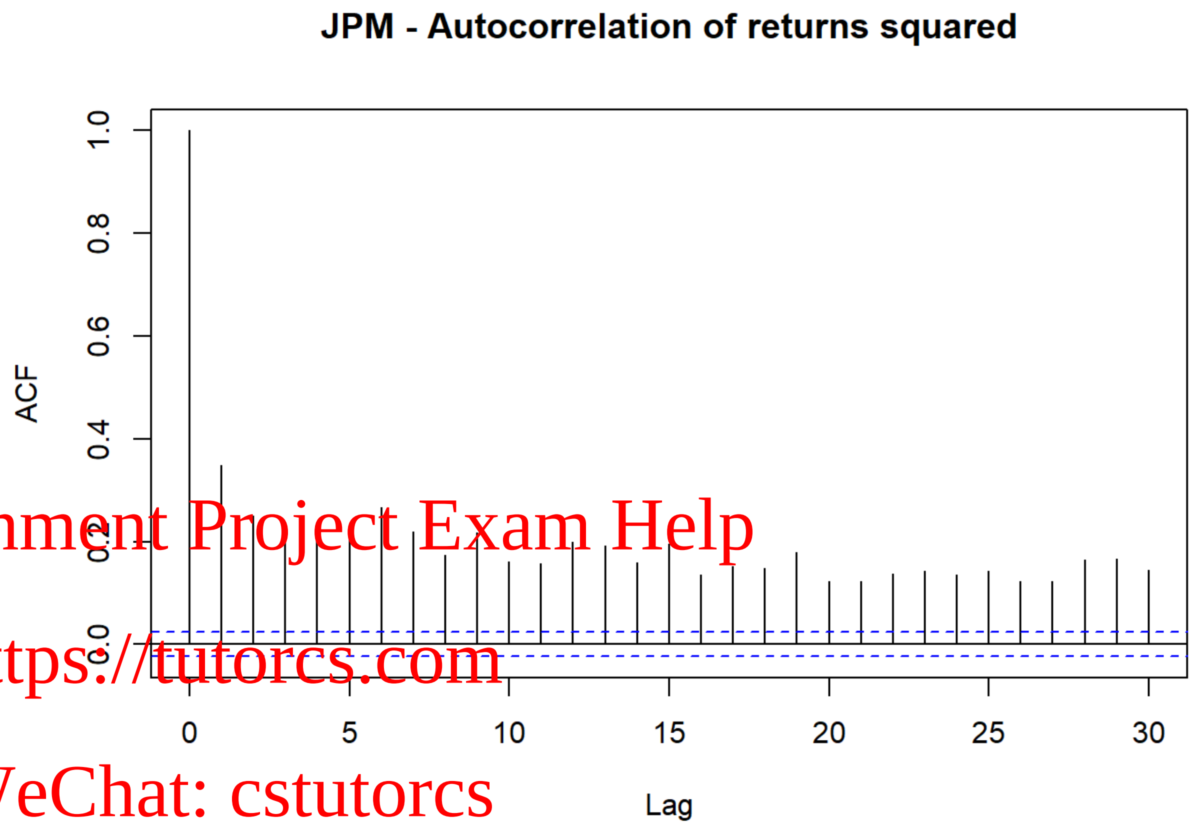
Even if returns show no autocorrelation, let's see what happens with returns squared, which are our estimate for volatility:

Acf of returns squared

```
acf(x = log_returns[, 1]^2, lag.max = 30,
    main = paste(Stocks[1], '- Autocorrelation of returns square
        d'))
```




```
acf(x = log_returns[, 2]^2, lag.max = 30,
    main = paste(Stocks[2], '- Autocorrelation of returns square
        d'))
```



We clearly see there is a strong positive autocorrelation for all shown lags. This pattern is normally seen in long memory time series. Recall that in a GARCH model, the size of $\alpha + \beta$ determines the memory of the time series. We will discuss more of the GARCH properties in lecture.

fpp2 library's ggAcf function can also provide the acf plot with ggplot features. ggAcf could be a better looking alternative.

Statistical test to detect volatility clusters

We can use the Ljung-Box test to test for serial correlation. This is a statistical tesf of whether any autocorrelations of a time series are different from zero. The null hypothesis H0 is that the data are independently distributed, while H1 is that the data exhibit serial correlation. The test statistic is distributed as a chi-square. The function Box.test can perform this test:

```
Box.test(x = log_returns[, 1], lag = 20, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  log_returns[, 1]
## X-squared = 151.62, df = 20, p-value < 2.2e-16
```



```
Box.test(x = log_returns[, 2], lag = 20, type = "Ljung-Box")
```



```
##
##  Box-Ljung test
##
## data:  log_returns[, 2]
## X-squared = 68.008, df = 20, p-value = 3.839e-07
```



```
Box.test(x = log_returns[, 1]^2, lag = 20, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  log_returns[, 1]^2
## X-squared = 9171.6, df = 20, p-value < 2.2e-16
```

```
Box.test(x = log_returns[, 2]^2, lag = 20, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  log_returns[, 2]^2
## X-squared = 5835.9, df = 20, p-value < 2.2e-16
```

```
print(paste('Critical Value is:', qchisq(p = 0.95, df = 20, lower.t
        ail=TRUE)))
```



```
## [1] "Critical Value is: 31.4104328442309"
```


Conditional volatility model: MA

Running the MA model for two lengths of estimation window for SP500

```
log_returns_demean <- sweep(x = log_returns, MARGIN = 2, STATS = Av
        gRet) # De-mean returns
we <- c(20, 60) # Estimation windows we = 20 and we = 60
sigma <- xts(x = matrix(nrow = dim(log_returns_demean)[1],
                            ncol = length(we)),
                 order.by = index(log_returns_demean)) # Pre-alloca
        tion of estimated volatility matrix

for (i in 1:2) {
  sigma[, i] <- rollapply(data = log_returns_demean[, 'SP500'],
                          width = we[i],
                          FUN = function(x) sd(x) * sqrt(we[i] - 1)
        / sqrt(we[i]))
  sigma[, i] <- lag(sigma[, i], k = 1, na.pad = TRUE) # lagging by
        1 to ensure that previous observation from t = 1 to t = we
        predict volatility at t = we + 1
}
```

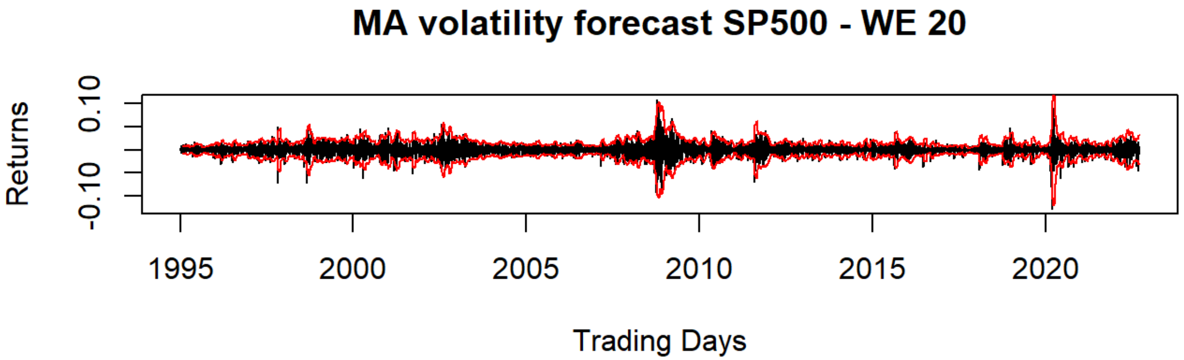

Plot Moving Average model for SP500

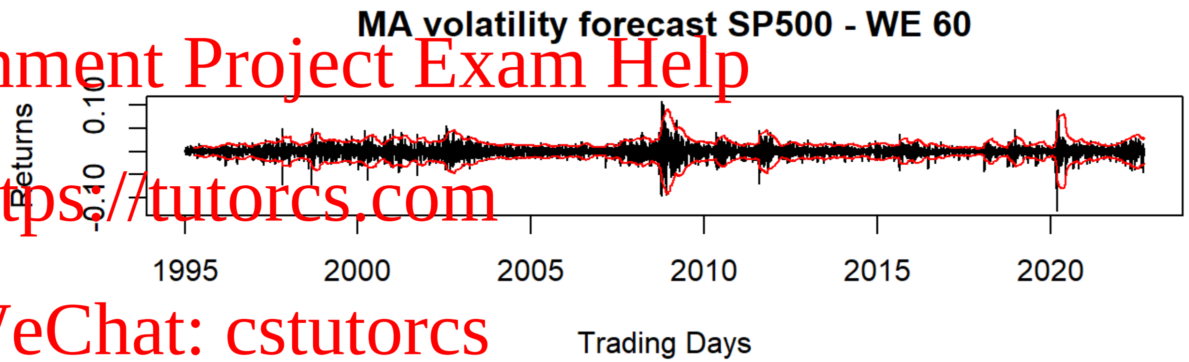
```
par(mfrow=c(2,1))
for (i in 1:2) {
  plot(x = index(sigma), y = log_returns_demean[, 'SP500'], type =
        'l',
       main = paste('MA volatility forecast SP500 - WE', we[i]),
       xlab = 'Trading Days', ylab = 'Returns')
  lines(x = index(sigma), y = 2 * sigma[, i], col = 'red')
  lines(x = index(sigma), y = -2 * sigma[, i], col = 'red')
}
```

Fat Tails

Volatility Clusters

Conditional volatility model: MA





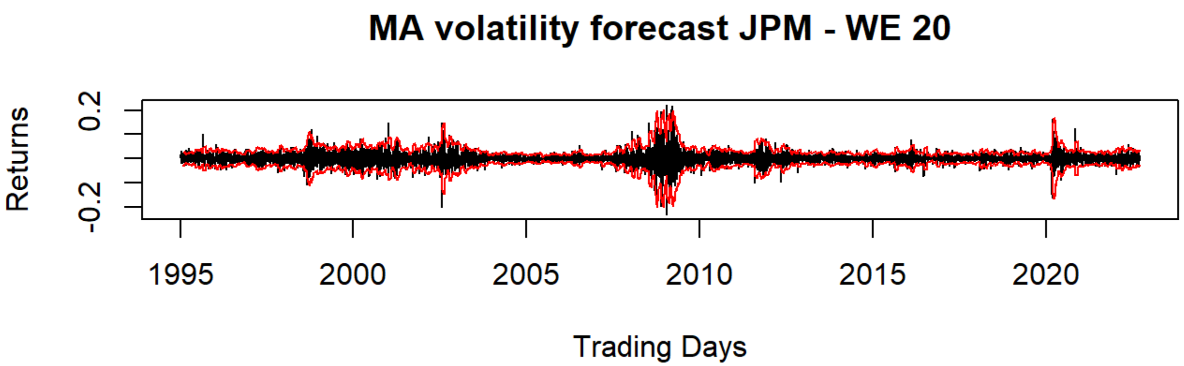
Running the MA model for two lengths of estimation window for JPMorgan

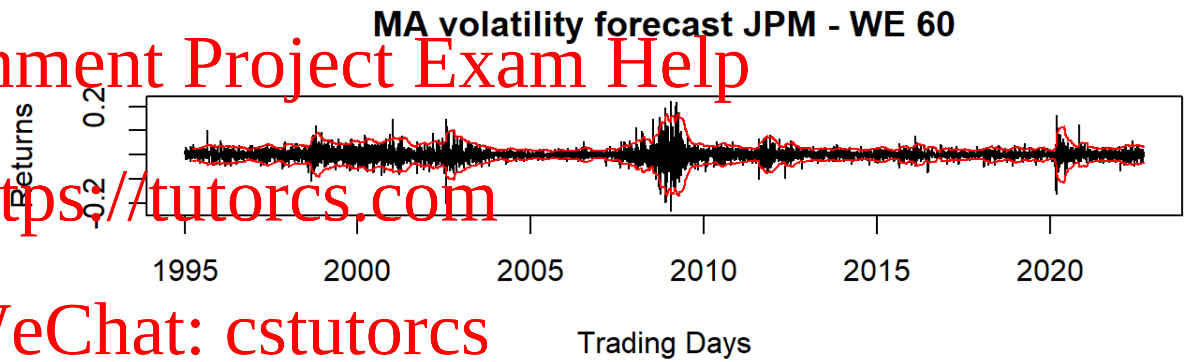
```
for (i in 1:2) {
  sigma[, i] <- rollapply(data = log_returns_demean[, 'JPM'],
                          width = we[i],
                          FUN = function(x) sd(x) * sqrt(we[i] - 1) / sqrt(we[i]))
                          # function sd normalizes by N-1 instead of N
  sigma[, i] <- lag(sigma[, i], k = 1, na.pad = TRUE) # lagging by 1 to ensure that previous observation from t = 1 to t = we predict volatility at t = we + 1
}
```


Note that sigma gets overwritten, one could also define a new sigma object for each stock, for example sigma_SP500 and sigma_JPM.

Plot Moving Average model for JPMorgan

```
par(mfrow=c(2,1))
for (i in 1:2) {
  plot(x = index(sigma), y = log_returns_demean[, 'JPM'], type = 'l',
       main = paste('MA volatility forecast JPM - WE', we[i]),
       xlab = 'Trading Days', ylab = 'Returns')
  lines(x = index(sigma), y = 2 * sigma[, i], col = 'red')
  lines(x = index(sigma), y = -2 * sigma[, i], col = 'red')
}
```






```
dev.off()
```

```
## null device
##           1
```

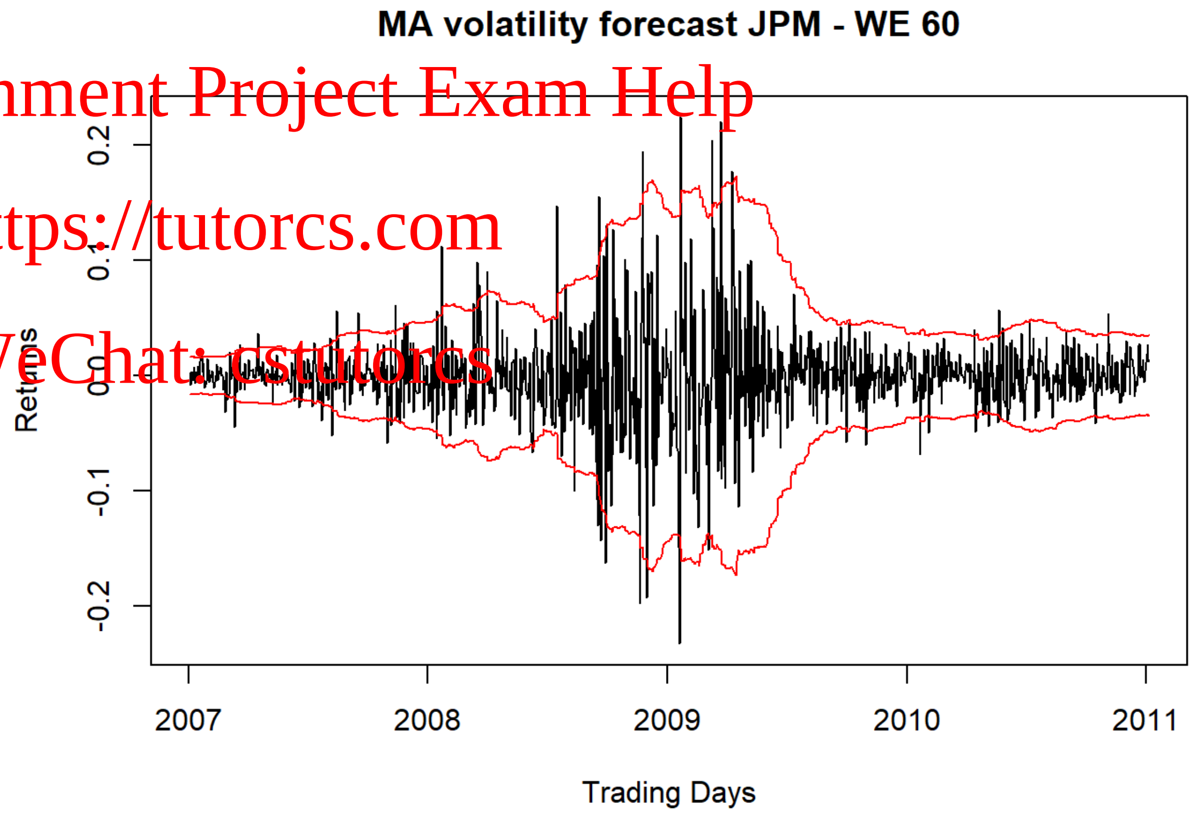
Some extra code

Zoom into the period of the Financial Crisis, JPMorgan stock

- Fat Tails
- Volatility Clusters
- Conditional volatility model: MA

```
log_returns_zoom <- log_returns['2007-01-03/2011-01-05',2]
sigma_zoom <- sigma['2007-01-03/2011-01-05',2]

plot(x = index(sigma_zoom), y = log_returns_zoom, type = 'l',
       main = paste('MA volatility forecast JPM - WE', we[2]),
       xlab = 'Trading Days', ylab = 'Returns')
lines(x = index(sigma_zoom), y = 2 * sigma_zoom, col = 'red')
lines(x = index(sigma_zoom), y = -2 * sigma_zoom, col = 'red')
```



Finding the date with the lowest S&P500 daily return

Fat Tails

Volatility Clusters

Conditional volatility model: MA

```
# Finding the lowest return
min(log_returns[, 'SP500'])
```

```
## [1] -0.1276522
```

```
# We can find the date when this happened in two different but equi
valent ways
# 1. Filtering the dates when the stock return was at its minimum
log_returns$SP500[log_returns[, 'SP500'] == min(log_returns[, 'SP50
0'])]
```



```
##                  SP500
## 2020-03-16 -0.1276522
```



```
# 2. Find the index where the minimum value is reached, and use it
as a filter
log_returns$SP500[which.min(log_returns$SP500),]
```



```
##                  SP500
## 2020-03-16 -0.1276522
```