11/28/2022

FINAL PROJECT

Financial Econometrics

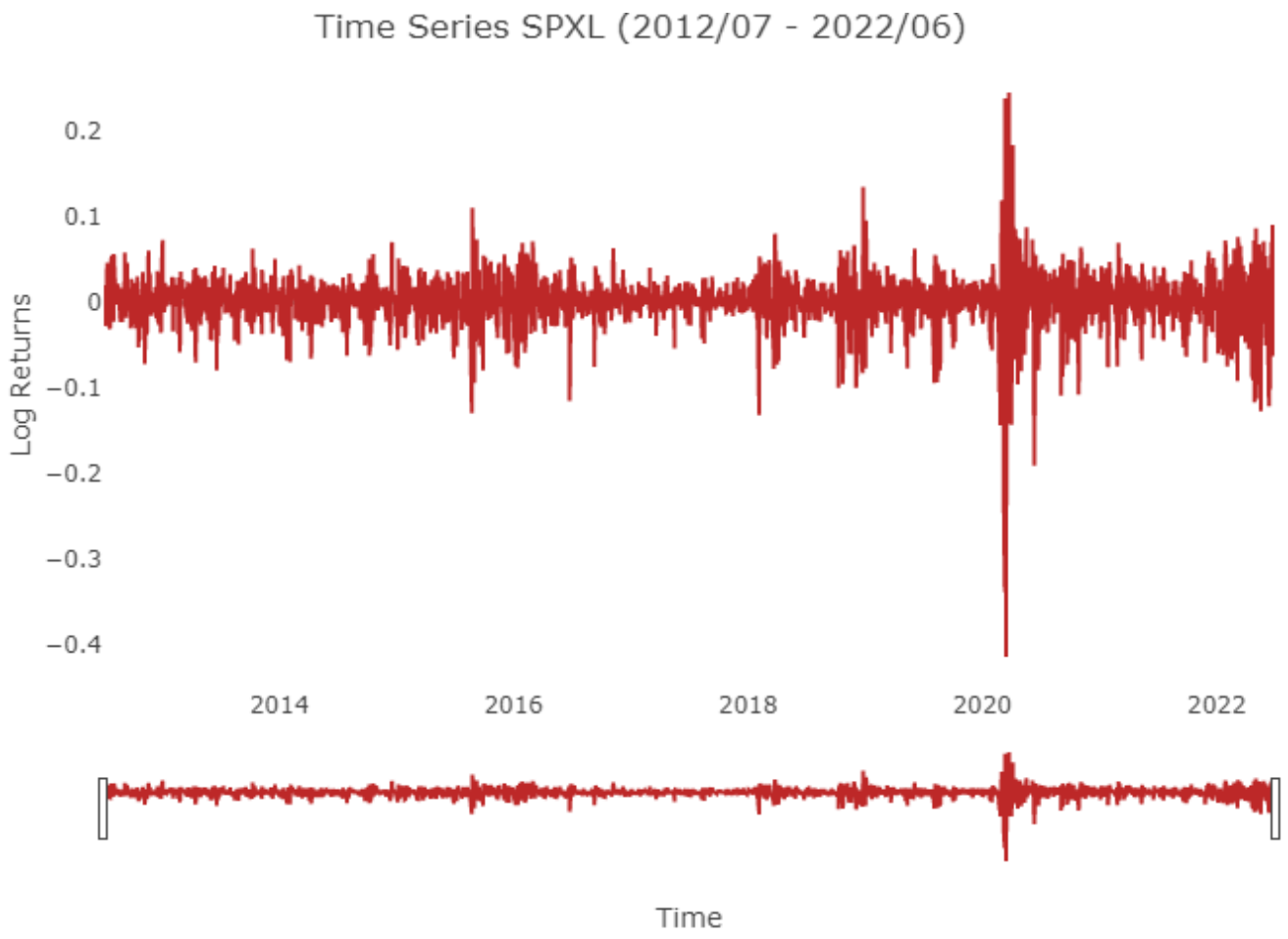
Fahad Alhodaithy

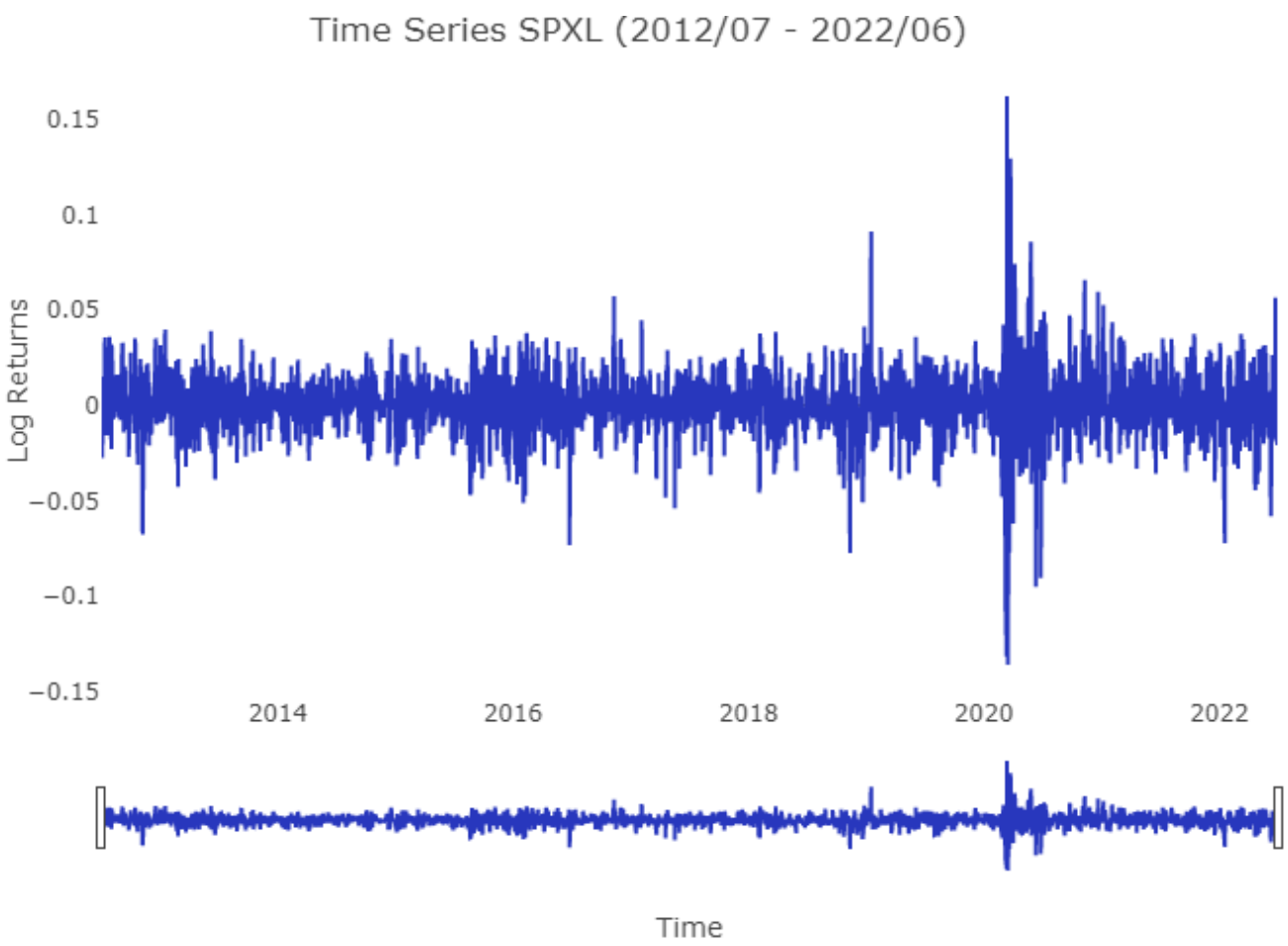
Contents

Introduction

This project aims to cover the concepts of risk management and time series to forecast variance for Direxion Daily S&P 500 Bull 3X Shares and Goldman Sachs to be used for calculating multiple risk metrics. It is essential to first understand the data and have a background of our dataset. The objective securities of this report are SPXL and Goldman Sachs. We will be first exploring their distribution and trend for the data from past 10 years. Such an exploratory analysis would steer us into the right direction of our analysis and provide the initial hypothesis to be further tested. The analysis could show different kurtosis, skewness, dispersion, variance etc. We would need to run test for each statistic to arrive at a conclusion. Such an analysis is also a building block for time series analysis and fitting various trend models and extracting stationarity. A little introduction about our securities at hand.

SPXL stock seeks daily investment results, fees, and expenses of 300%, or 300% of the inverse (or opposite), of the performance of the S&P 500 Index. It does not seek to achieve its stated investment objective over a period greater than one day. Since this stock seeks daily goals, it is riskier than the alternatives that do not use leverage. These types of stock are not suitable for all investors and should be utilized only by investors who understand leverage risk and who actively manage their investments. It is perfect for performing a risk analysis for SPXL to predict the daily outcome of the stock. One interesting fact about SPXL is that it was created in 2009 after the Great Recession.





Goldman Sachs is an American multinational investment bank and financial services company. Founded in 1869, Goldman Sachs is headquartered at 200 West Street in Lower Manhattan, with regional headquarters in London, Warsaw, Bangalore, Hong Kong, Tokyo, Dallas and Salt Lake City, and additional offices in other international financial centers. Goldman Sachs is the second largest investment bank in the world by revenue and is ranked 57th on the Fortune 500 list of the largest United States corporations by total revenue. It is considered a systemically important financial institution by the Financial Stability Board. Based on the time series plot of GS, we can say the log returns for GS are lesser volatile than SPXL because SPXL as an ETF invests in the volatile stocks.

Both stocks are extracted from Yahoo Finance using the 'quantmod' R package. The goal is to look at the daily stock prices ten years back. So, the data is from Fiscal year 2012 to 2022. As mentioned above, GS is ticker for Goldman Sachs and SPXL for Direxion Daily S&P. Below are the summary tables for both stocks. The tables show the stock volume, opening stock price, highest, lowest, closing, and adjusted price.

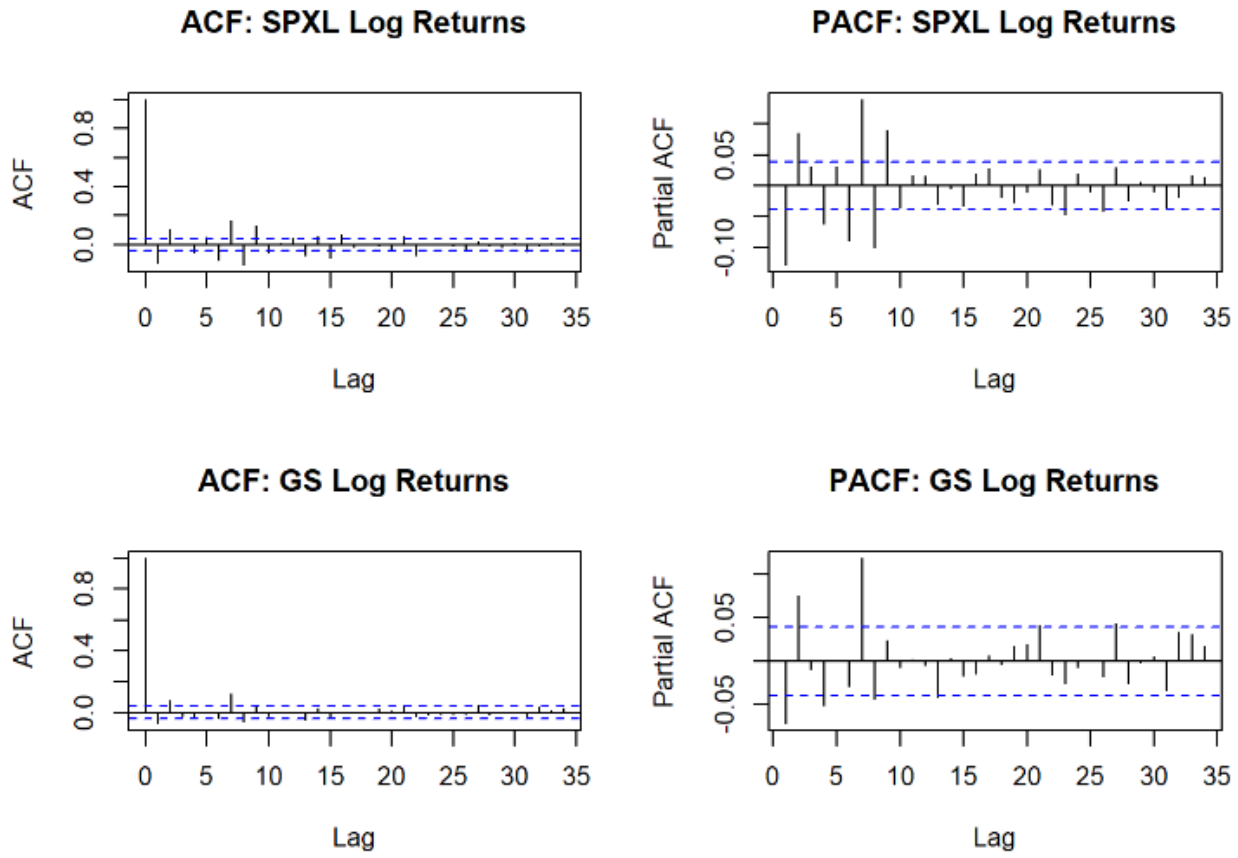
My dataset consists of the price data for both the tickers and the adjusted closing price for everyday is of importance to us for calculating the log returns. I made two different datasets consisting of daily log returns and removed first row to avoid NaNs for seamless analysis. Starting of with a distribution analysis on my dataset, I see that SPXL has a range of 0.66, while GS has a range of 0.3. The first measure of dispersion (Range) confirms that closing prices for SPXL are more volatile than GS. The distribution for Log returns of both stocks is negatively skewed because Mean < Median.

Index	SPXL.Adjusted
Min. :2012-07-03	Min. :-0.4135766
1st Qu.:2015-01-03	1st Qu.:-0.0101632
Median :2017-07-03	Median : 0.0019937
Mean :2017-07-02	Mean : 0.0009503
3rd Qu.:2020-01-01	3rd Qu.: 0.0157773
Max. :2022-06-30	Max. : 0.2453454

Index	GS.Adjusted
Min. :2012-07-03	Min. :-0.1358806
1st Qu.:2015-01-03	1st Qu.:-0.0082340
Median :2017-07-03	Median : 0.0006690
Mean :2017-07-02	Mean : 0.0005093
3rd Qu.:2020-01-01	3rd Qu.: 0.0096007
Max. :2022-06-30	Max. : 0.1619513

Statistical Analysis

ACF Analysis



Starting off by ACF analysis, the ACF plots show stationarity in log returns, with non-constant variance and a constant mean. The PACF further buttresses the fact for non-constant variance, and we can see some spikes for some lags.

As we stated before that SPXL is more volatile than GS because of a wider range, it is further buttressed by the fact that SPXL's distribution is platykurtic with a kurtosis = 25.39 and GS with a kurtosis = 13. Both distributions are Platykurtic but SPXL's returns are more spread out. We know that our distributions are negatively skewed, and this has been confirmed by running the skewness test.

```
"SPXL Log Returns skewness =  -1.61288292656291"

"Goldman Sachs Log Returns skewness =  -0.239871420369225"

"SPXL Log Returns kurtosis =  25.3863253408893"

"Goldman Sachs Log Returns kurtosis =  13.0006362335318"
```

Serial-Correlation Test

Next, I ran a Box-Ljung test for checking serial correlation in Log-returns for both datasets. Based on the P-value, we can reject the null hypothesis that that the observations are independent and not serially correlated; hence there is serial correlation for SPXL and GS Log-returns.

```
 Box-Ljung test

data:  spxl_log_ret
X-squared = 81.037, df = 5, p-value = 5.551e-16

 Box-Ljung test

data:  gs_log_ret
X-squared = 34.337, df = 5, p-value = 2.04e-06
```

Stationarity Test:

To further confirm for stationarity, we reject the null hypothesis of non-stationarity because the p value is in the rejection region for both SPXL and GS. Hence, our log-normal returns are stationary for both datasets; this would not have been the case for adjusted closing prices for these stocks because taking log-differencing removes trend and seasonality from the data points.

```
 Augmented Dickey-Fuller Test

data:  spxl_log_ret
Dickey-Fuller = -13.265, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary

 Augmented Dickey-Fuller Test

data:  gs_log_ret
Dickey-Fuller = -13.536, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary
```

Time Series Models

I fit the AR(1)-GARCH(1,1) to log-returns of SPXL and GS to the series of log returns as per the code below. The summary for both AR-GARCH Fit is shown in the Knit file. The P Values for AR and garch coefficients indicate that those are statistically significant for both models.

```
SPXLfit=garchFit(formula=~arma(1,0)+garch(1,1),data=spxl_log_ret,cond.dist="norm")
```

```
GSfit=garchFit(formula=~arma(1,0)+garch(1,1),data=gs_log_ret,cond.dist="norm"
```

The mean of error(residuals) and standard deviation of error(residuals) are fit to t-distribution and multiple copulas such as t-copula, Gaussian copula, Gumbel Copula, and Clayton copula.

Extract residuals and Fit T-Distribution

```
# Residuals from AR-GARCH model

SPXL_res=residuals(SPXLfit)
GS_res=residuals(GSfit)

# Standardized Residuals from AR-GARCH model

SPXL_res_sd=residuals(SPXLfit,standardize=TRUE)
GS_res_sd=residuals(GSfit,standardize=TRUE)

#Fit t-distribution

#Fit t-distribution to residuals

std.SPXL_res = as.numeric(fitdistr(SPXL_res,"t")$estimate)
std.GS_res = as.numeric(fitdistr(GS_res,"t")$estimate)

std.SPXL_res[2] = std.SPXL_res[2] * sqrt(std.SPXL_res[3] / (std.SPXL_res[3]-2))
std.GS_res[2] = std.GS_res[2] * sqrt(std.GS_res[3] / (std.GS_res[3]-2))

#Fit t-distribution to standardized residuals

std.SPXL_res_sd = as.numeric(fitdistr(SPXL_res_sd,"t")$estimate)
std.GS_res_sd = as.numeric(fitdistr(GS_res_sd,"t")$estimate)

std.SPXL_res_sd[2] = std.SPXL_res_sd[2] * sqrt(std.SPXL_res_sd[3] / (std.SPXL_res_sd[3]-2))
std.GS_res_sd[2] = std.GS_res_sd[2] * sqrt(std.GS_res_sd[3] / (std.GS_res_sd[3]-2))
```

The above chunk shows how I took the residuals and the standardized residuals for both series. As specified in the requirements, these are fit to t-distribution and the t-copula, Gaussian copula, Gumbel Copula, and Clayton copula, in order to calculate the AIC values from residuals and standardized residuals for choosing our best copula.

Fitting Copulas to Residuals

```
# Estimate of the correlation coefficient in the t-copula using Kendall's tau

cor_tau = cor(SPXL_res, GS_res, method = "kendall", use="pairwise.complete.obs")
round(cor_tau,2)
omega = sin((pi/2)*cor_tau) #estimator for rho
round(omega,2)

# Combining datasets:

data1 = cbind(pstd(SPXL_res, std.SPXL_res[1], std.SPXL_res[2], std.SPXL_res[3]),
              pstd(GS_res, std.GS_res[1], std.GS_res[2], std.GS_res[3]))

# Fit t-copula
cop_t_dim2 = tCopula(omega, dim = 2, dispstr = "un", df = 4) #define t copula
Ct=fitCopula(cop_t_dim2, data1, method="ml", start=c(omega,4) )
Ct@estimate
lst_value <- loglikCopula(param=Ct@estimate,u=data1,copula=tCopula(dim=2));#compute loglikelihood function
AIC_Ct <- (-2) * lst_value + 2*length(Ct@estimate) #compute AIC

# Fit Gaussian copula
Cgauss=fitCopula(copula=normalCopula(dim=2),data=data1, method="ml")
Cgauss@estimate
lst_value <- loglikCopula(param=Cgauss@estimate,u=data1,copula=normalCopula(dim=2))
AIC_gauss <- (-2)*lst_value+2*length(Cgauss@estimate)#compute AIC

# Fit Gumbel copula
Cgu=fitCopula(copula=gumbelCopula(3,dim=2),data=data1,method="ml")
Cgu@estimate
lst_value <- loglikCopula(param=Cgu@estimate,u=data1,copula=gumbelCopula(dim=2))
AIC_Cgu <- (-2)*lst_value+2*length(Cgu@estimate)

# Fit Clayton copula
Ccl=fitCopula(copula=claytonCopula(1,dim=2),data=data1,method="ml")
Ccl@estimate
lst_value <- loglikCopula(param=Ccl@estimate,u=data1,copula=claytonCopula(dim=2))
AIC_Ccl <- (-2)*lst_value+2*length(Ccl@estimate)

AIC_Values_ <- rbind(AIC_Ccl,AIC_Cgu, AIC_gauss, AIC_Ct)
colnames(AIC_Values) <- c("AIC")
AIC_Values <- AIC_Values %>%
  data.frame()
AIC_Values
```

```
               AIC
AIC_Ccl   -1367.121
AIC_Cgu   -1666.950
AIC_gauss -1800.126
AIC_Ct    -1895.333
```

The t-copula for our residuals is the best copula based on the AIC values, as it has the lowest AIC value.

Fitting Copulas to Standardized Residuals

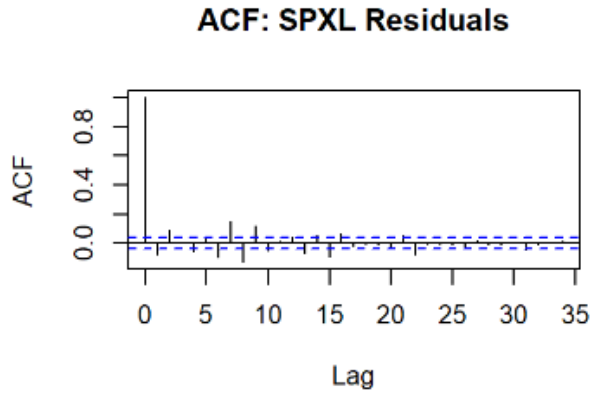
```
# Estimate of the correlation coefficient in the t-copula using Kendall's tau

cor_tau = cor(SPXL_res_sd, GS_res_sd, method = "kendall", use="pairwise.complete.obs")
round(cor_tau,2)
omega = sin((pi/2)*cor_tau) #estimator for rho
round(omega,2)

# Combining datasets:

data1 = cbind(pstd(SPXL_res_sd, std.SPXL_res_sd[1], std.SPXL_res_sd[2], std.SPXL_res_sd[3]),
              pstd(GS_res_sd, std.GS_res_sd[1], std.GS_res_sd[2], std.GS_res_sd[3]))

# Fit t-copula
cop_t_dim2 = tCopula(omega, dim = 2, dispstr = "un", df = 4) #define t copula
Ct=fitCopula(cop_t_dim2, data1, method="ml", start=c(omega,4) )
Ct@estimate
lst_value <- loglikCopula(param=Ct@estimate,u=data1,copula=tCopula(dim=2));#compute loglikelihood function
AIC_Ct <- (-2) * lst_value + 2*length(Ct@estimate) #compute AIC

# Fit Gaussian copula
Cgauss=fitCopula(copula=normalCopula(dim=2),data=data1, method="ml")
Cgauss@estimate
lst_value <- loglikCopula(param=Cgauss@estimate,u=data1,copula=normalCopula(dim=2))
AIC_gauss <- (-2)*lst_value+2*length(Cgauss@estimate)#compute AIC

# Fit Gumbel copula
Cgu=fitCopula(copula=gumbelCopula(3,dim=2),data=data1,method="ml")
Cgu@estimate
lst_value <- loglikCopula(param=Cgu@estimate,u=data1,copula=gumbelCopula(dim=2))
AIC_Cgu <- (-2)*lst_value+2*length(Cgu@estimate)

# Fit Clayton copula
Ccl=fitCopula(copula=claytonCopula(1,dim=2),data=data1,method="ml")
Ccl@estimate
lst_value <- loglikCopula(param=Ccl@estimate,u=data1,copula=claytonCopula(dim=2))
AIC_Ccl <- (-2)*lst_value+2*length(Ccl@estimate)

AIC_Values_sd <- rbind(AIC_Ccl,AIC_Cgu, AIC_gauss, AIC_Ct)
colnames(AIC_Values_sd) <- c("AIC")
AIC_Values_sd <- AIC_Values_sd %>%
  data.frame()
AIC_Values_sd
```

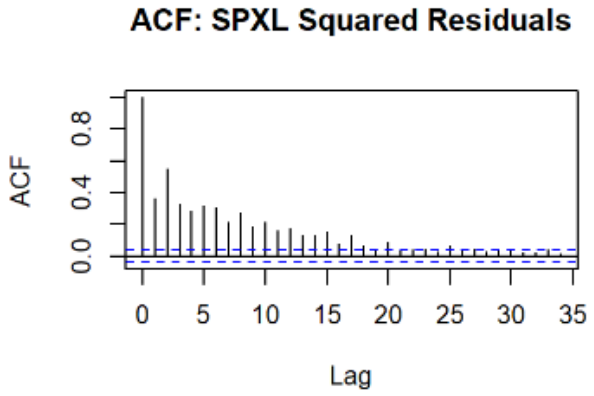
```
               AIC
AIC_Ccl   -1146.843
AIC_Cgu   -1503.134
AIC_gauss -1593.214
AIC_Ct    -1665.424
```

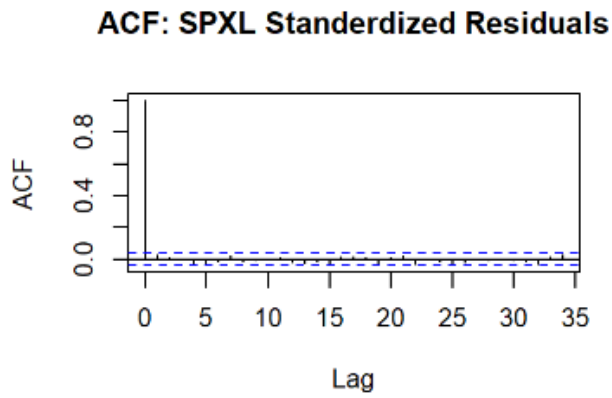
The t-copula for our standardized residuals is the best copula based on the AIC values, as it has the lowest AIC value.

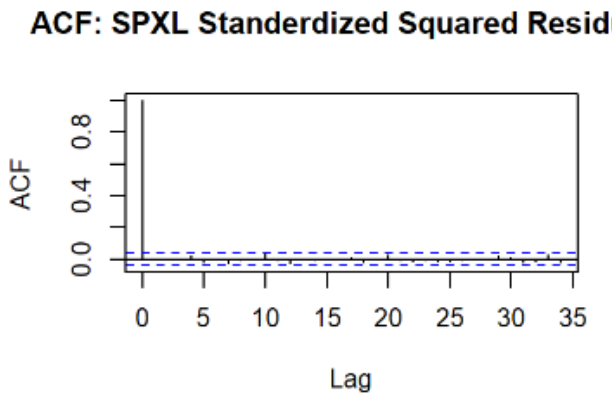
Residual Analysis

Direxion Daily S&P





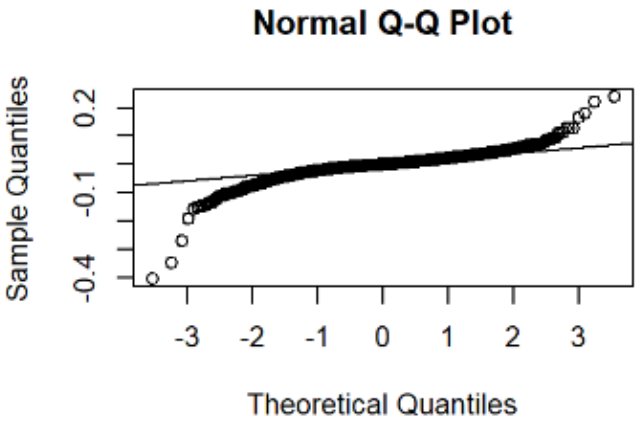


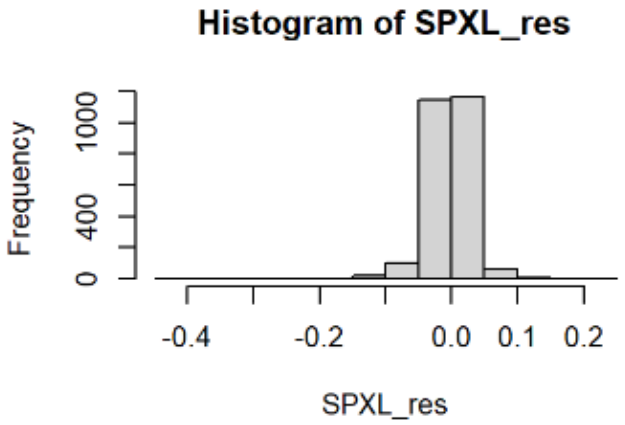


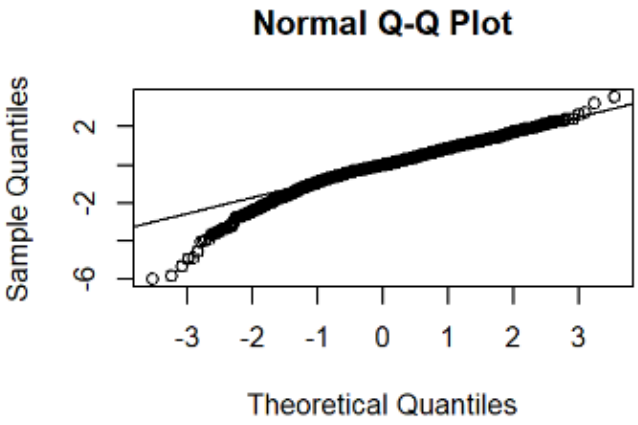

```
 Box-Ljung test

data:  SPXL_res
X-squared = 50.063, df = 5, p-value = 1.346e-09

 Box-Ljung test

data:  SPXL_res_sd
X-squared = 6.2073, df = 5, p-value = 0.2866

 Augmented Dickey-Fuller Test

data:  SPXL_res
Dickey-Fuller = -13.242, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary

 Augmented Dickey-Fuller Test

data:  SPXL_res_sd
Dickey-Fuller = -14.386, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary
```

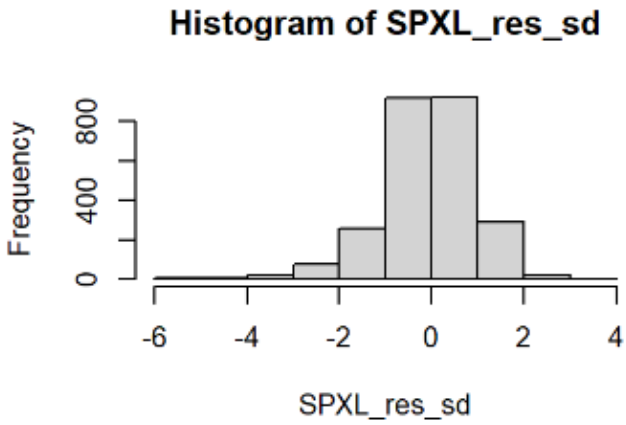
The residual analysis for SPXL shows that ε_t(residuals) is not stationary and serially correlated based on our ACF plots for residuals further confirmed by ACF of Squared Residuals. In order to check for stationarity; I conducted Dicky-Fuller test and a Box Ljung test for Serial Correlation. We fail to reject the null hypothesis for no serial correlation and conclude that there is serial correlation. $\tilde{\varepsilon}_t$ (Standardized residuals) is not serial correlated and stationary as indicated by the ACF of residuals and Squared residuals. This is further confirmed by the Dicky fuller test not Stationarity where we reject the null hypothesis for non-stationarity based on the p value and serial correlation is tested by Box-Ljung test, where p value confirms that the residuals for SD are not serially correlated.

I tried multiple P-Values to eliminate the serial correlation in our time series, it seems that it was not possible for P Values 1 to 20. It would be counterproductive to keep trial and error for 251 trading days of a year and find the right p order. Finding p order through the AIC iteration makes more sense but it is going to be computationally demanding for a daily data set as the range is going to be from 1 to 251. It is easier to transform the data in order to eliminate any trend or seasonality, and avoid the serial correlation in residuals altogether.



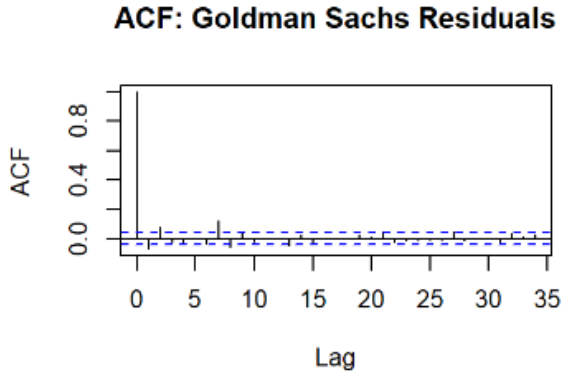


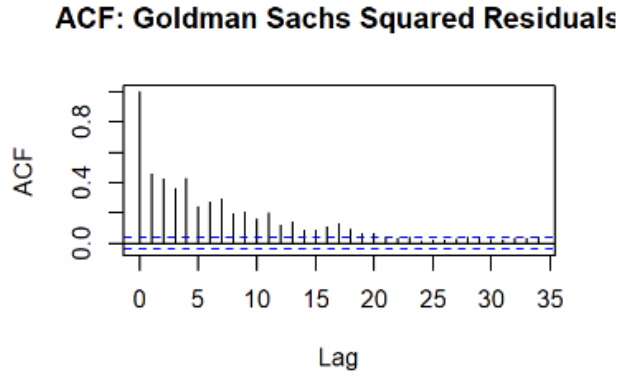


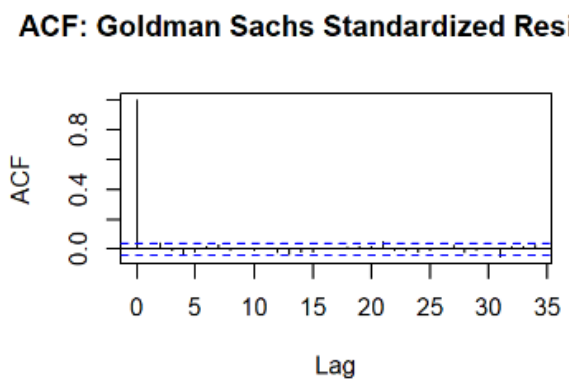


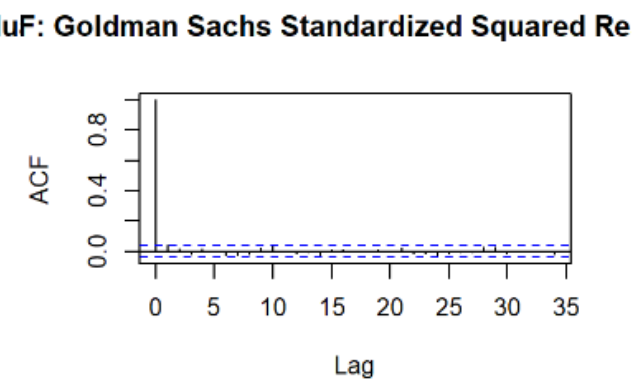
Based on the QQ plot and Histogram analysis for the distribution of residuals and standardized residuals, we confirm that residuals and the standardized residuals are normally distributed with some outliers only.

Goldman Sachs





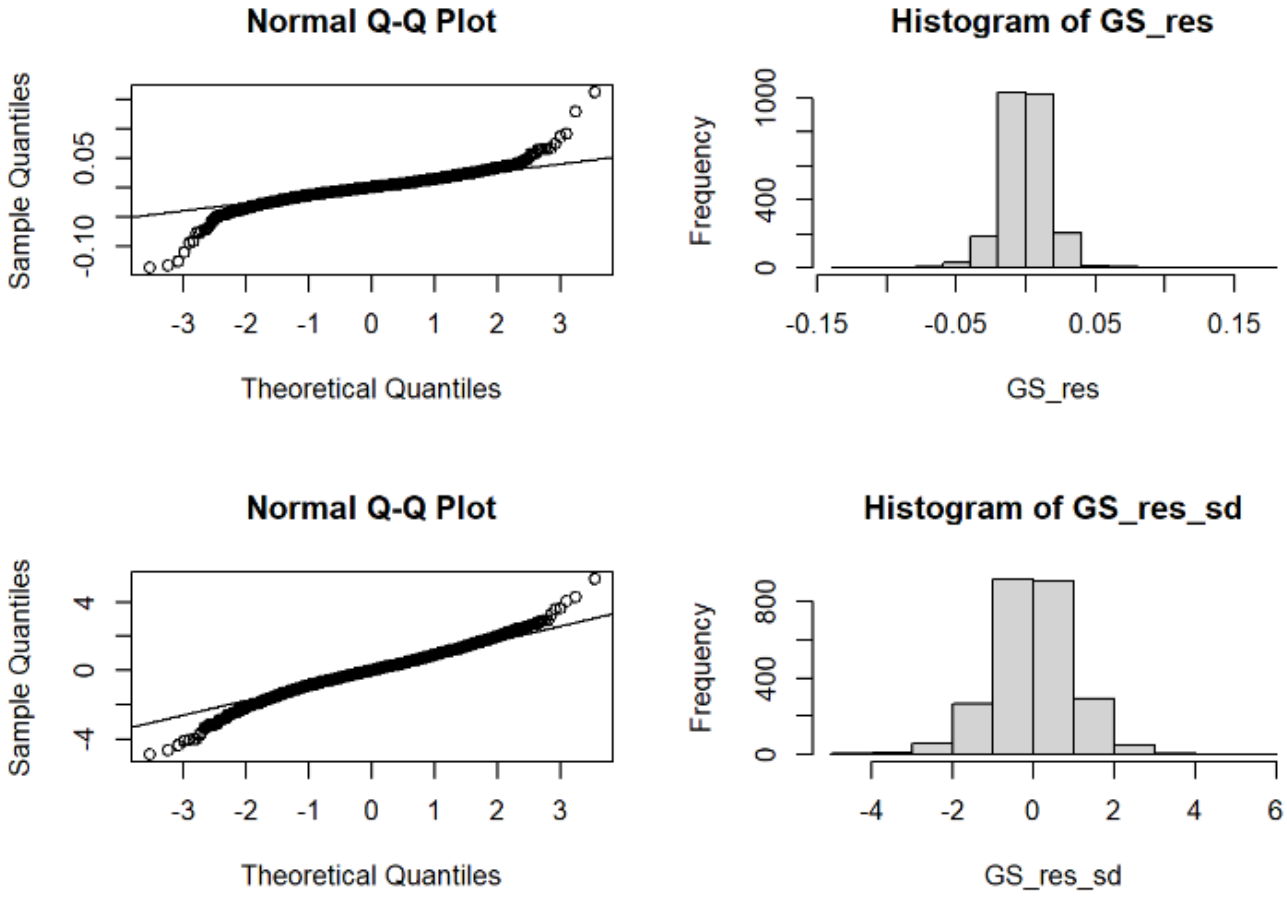





```
 Box-Ljung test

data:  GS_res
X-squared = 34.113, df = 5, p-value = 2.261e-06

 Box-Ljung test

data:  GS_res_sd
X-squared = 9.3712, df = 5, p-value = 0.09514

 Augmented Dickey-Fuller Test

data:  GS_res
Dickey-Fuller = -13.538, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary

 Augmented Dickey-Fuller Test

data:  GS_res_sd
Dickey-Fuller = -13.986, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary
```

The residual analysis for GS shows that ε_t(mean of residuals) is not stationary and serially correlated based on our ACF plots for residuals of mean further confirmed by ACF of Squared Residuals of mean. In order to check for stationarity; I conducted Dicky-Fuller test and a Box Ljung test for Serial Correlation. We fail to reject the null hypothesis for no serial correlation and conclude that there is serial correlation. (Residuals for standard deviation) $\tilde{\varepsilon}_t$ is not serial correlated and stationary as indicated by the ACF of residuals and Squared residuals. This is further confirmed by the Dicky fuller test not Stationarity where we reject the null hypothesis for non-stationarity based on the p value and serial correlation is tested by Box-Ljung test, where p value confirms that the residuals for SD are not serially correlated.

I tried multiple P-Values to eliminate the serial correlation in our time series, it seems that it was not possible for P Values 1 to 20. It would be counterproductive to keep trial and error for 251 trading days of a year and find the right p order. Finding p order through the AIC iteration makes more sense but it is going to be computationally demanding for a daily data set as the range is going to be from 1 to 251. It is easier to transform the data in order to eliminate any trend or seasonality, and avoid the serial correlation in residuals altogether.



Based on the QQplot and Histogram analysis for the distribution of residuals of mean and standard deviation, we confirm that residuals for both are normally distributed with some outliers only.

Risk Calculation

The aim of the whole project was to arrive at Value at Risk with 99% Confidence interval using the coefficient and degrees of freedom from our best copula. Further on, we have to manipulate the following equation to arrive at sigma (σ):

$$\begin{aligned} 0.99 &= P(\rho X_{n+1} + (1-\rho)\tilde{X}_{n+1} \leq x | \mathcal{F}_n) \\ &= P\left(\rho\{\mu + \phi X_n + \sigma_{n+1}\varepsilon_{n+1}\} + (1-\rho)\{\tilde{\mu} + \tilde{\phi}\tilde{X}_n + \tilde{\sigma}_{n+1}\tilde{\varepsilon}_{n+1}\} \leq x | \mathcal{F}_n\right) \end{aligned}$$

The probabilities for both series are generated by using the coefficient of t-copula and assigning dimensions with degrees of freedom; based on required number of series. I made a choice of generating a random sample of 10000 probabilities, to achieve central limit theorem based on the law of large numbers. Such a distribution would be close to a real-world depiction VaR statistics for Log Returns from past 10 years. These probabilities would be used to make distributions for both series, based on their respective mean and standard deviation of T-distribution residuals. Such a process is called simulation, where we are generating random probabilities and fitting on a distribution based on actual mean and standard deviation. The code for simulation is shared below.

```
# Drawing a Random Sample
rho <- coef(Ct)

random_samples <- rCopula(10000,tCopula(rho[1], dim=2, df=rho[2]))

# Transform Data

random_samples <- random_samples %>%
  data.frame()

colnames(random_samples) <- c("SPXL.prob", "GS.prob")

random_samples$SPXL.e.std <- qstd(random_samples$SPXL.prob, mean = std.SPXL_res[1], sd = std.SPXL_res[2],
                        nu = std.SPXL_res[3])
random_samples$GS.e.std <- qstd(random_samples$GS.prob, mean = std.GS_res[1], sd = std.GS_res[2],
                      nu = std.GS_res[3])


# Fit a AR(1)-GARCH(1,1) model to the returns

ar1 <- function(mu, phi, x_prev) {
  x = mu*(1-phi) + phi * x_prev
  return(as.numeric(x))
}

garch1.1 <- function(omega, alpha, beta, e_prev, sigma_prev) {
  sigma_sq <- omega + alpha*(e_prev^2) + beta*(sigma_prev^2)
  sqrt(sigma_sq) %>% as.numeric() %>% return()
}
```

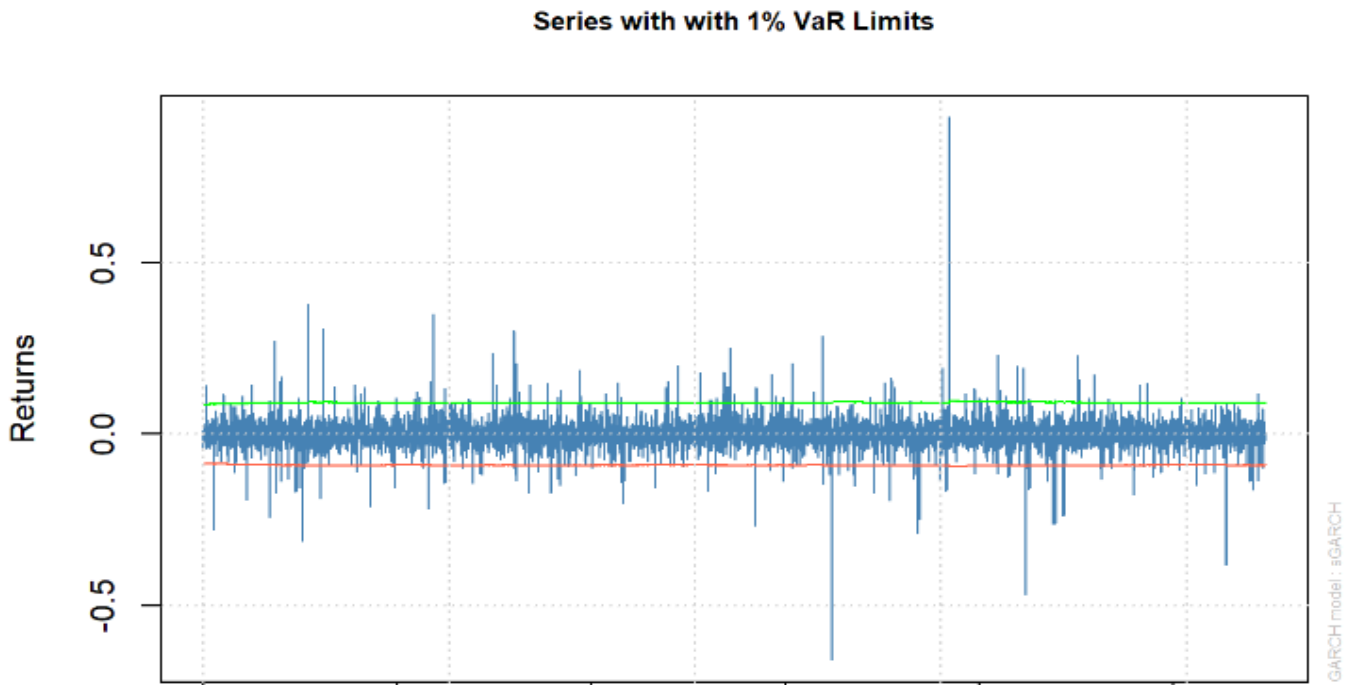
The above distributions are used to fit AR- GARCH model and run the forecast for our distribution from random sample of 10000 observations. I have created the above function to run such forecasts based on the formula provided in the guidelines.

```
#SPXL

spxl.params <- coef(SPXLfit)


random_samples$spxl.next <-
  ar1(mu = spxl.params[1], phi = spxl.params[2], x_prev = spxl_log_ret$SPXL.Adjusted[nrow(spxl_log_ret)]) +
  garch1.1(omega = spxl.params[3], alpha = spxl.params[4], beta = spxl.params[5],
           e_prev = SPXL_res[nrow(spxl_log_ret)], sigma_prev = SPXL_res_sd[nrow(spxl_log_ret)]) * random_samples
$SPXL.e.std

#GS

gs.params <- coef(GSfit)

random_samples$gs.next <-
  ar1(mu = gs.params[1], phi = gs.params[2], x_prev = gs_log_ret$GS.Adjusted[nrow(gs_log_ret)]) +
  garch1.1(omega = gs.params[3], alpha = gs.params[4], beta = gs.params[5],
           e_prev = GS_res[nrow(gs_log_ret)], sigma_prev = GS_res_sd[nrow(gs_log_ret)]) * random_samples$GS.e.std

#Apply AR(1,0)-GARCH(1,1)

garch.SPXL = ugarchspec(mean.model=list(armaOrder=c(1,0)),
                        variance.model=list(garchOrder=c(1,1)),
                        distribution.model = "std")
SPXL.garch.fit = ugarchfit(data=random_samples$SPXL.e.std, spec=garch.SPXL)

garch.GS = ugarchspec(mean.model=list(armaOrder=c(1,0)),
                        variance.model=list(garchOrder=c(1,1)),
                        distribution.model = "std")
GS.garch.fit = ugarchfit(data=random_samples$GS.e.std, spec=garch.GS)

show(SPXL.garch.fit)
```

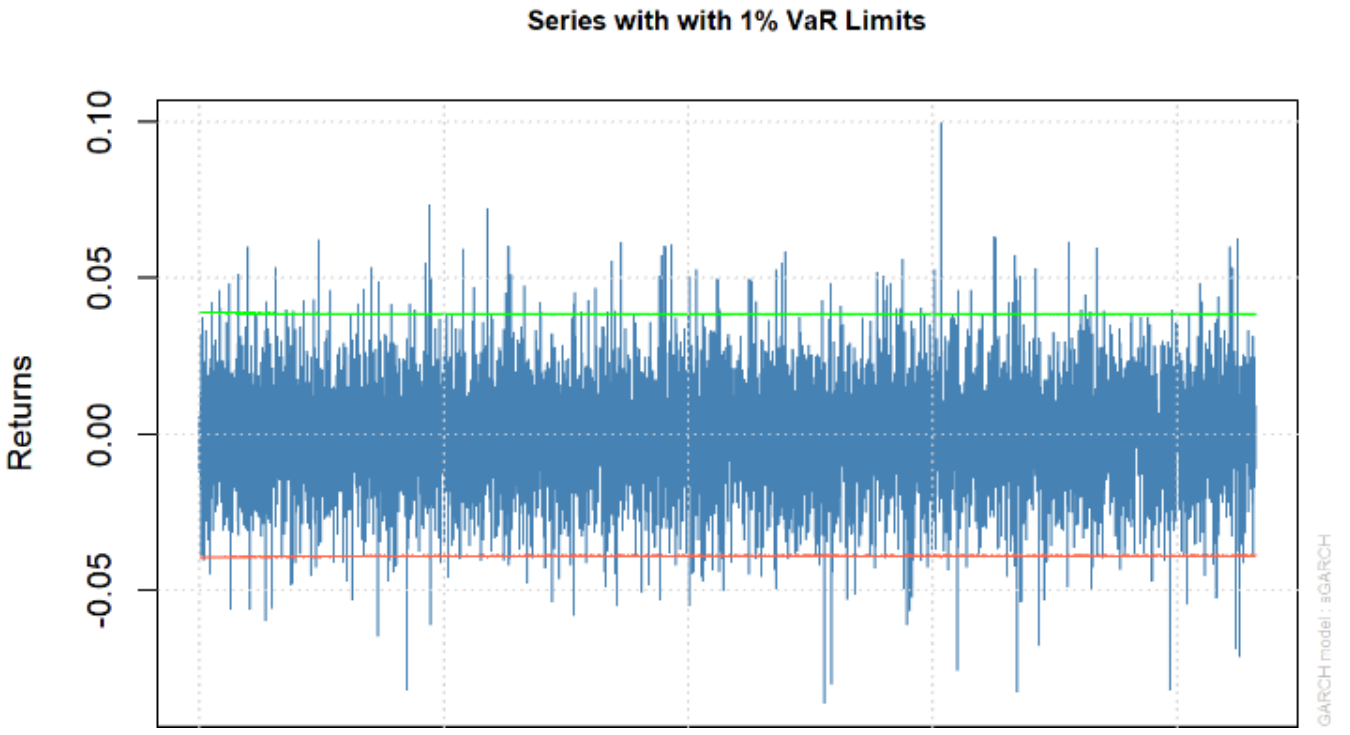
The above code shows how I fit my simulated distribution on the time series models and shown the results for each fit in my appendix. These models are used to plot the VaR for 99% confidence interval and we can clearly see the number of instances, where our returns have been below our VaR levels for 10000 samples. There are equal chances of good and bad times in such a case. Following plot for SPXL and Goldman Sachs is shared below.

The plots below are SPXL series with 1% VaR Limits on the top, and the GS series with 1% VaR Limits on the bottom. We can see that SPXL has more returns that are lower than VaR, indicating that SPXL is riskier than GS. On the other hand, GS has fewer returns that are lower than VaR.

SPXL VaR Plot



Goldman Sachs VaR Plot

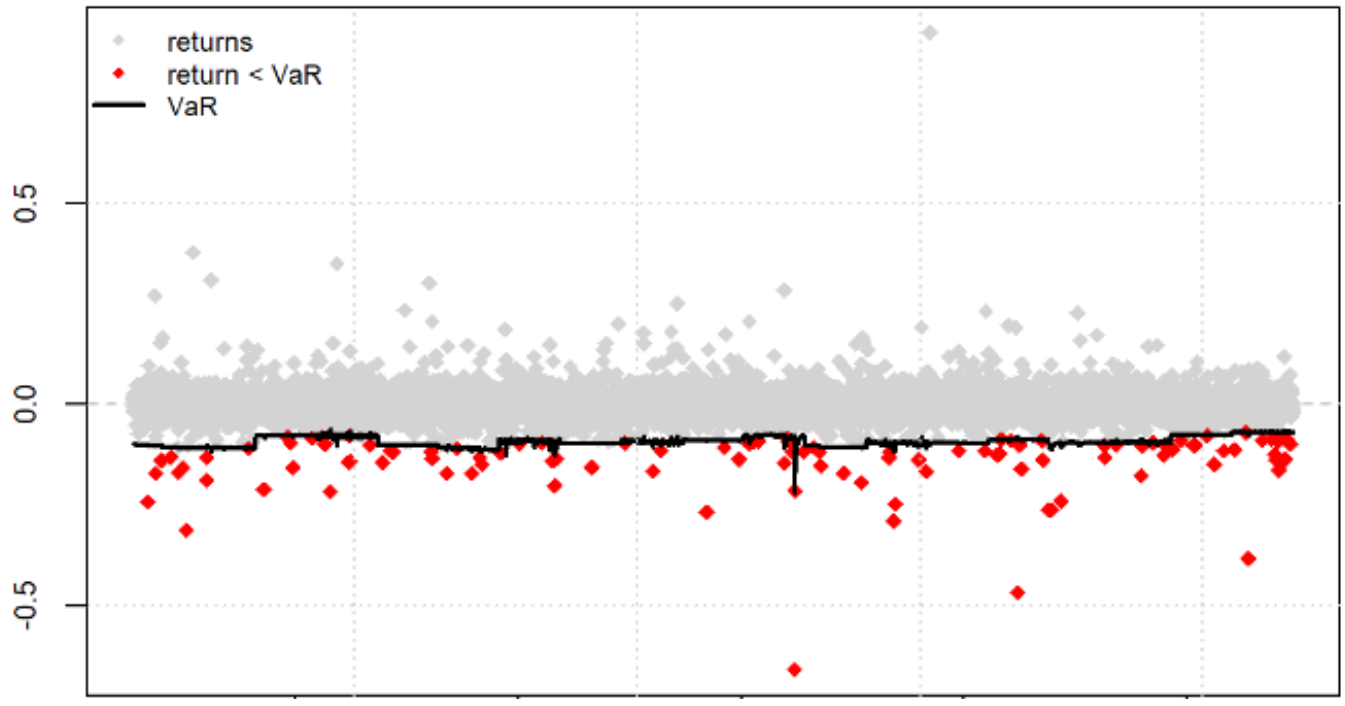


Rolling Forecasts

```
# Calculate one step forecasts

SPXL_cond_roll <- ugarchroll(garch.SPXL, random_samples$SPXL.e.std, n.start = 500, refit.every = 500,
                             refit.window = "moving", solver = "hybrid",
                             calculate.VaR = TRUE, VaR.alpha = 0.01, keep.coef = TRUE,
                             solver.control = list(tol = 1e-7, delta = 1e-9),
                             fit.control = list(scale = 1))
plot(SPXL_cond_roll, which = 4)

GS_cond_roll <- ugarchroll(garch.GS, random_samples$GS.e.std, n.start = 500, refit.every = 500,
                             refit.window = "moving", solver = "hybrid",
                             calculate.VaR = TRUE, VaR.alpha = 0.01, keep.coef = TRUE,
                             solver.control = list(tol = 1e-7, delta = 1e-9),
                             fit.control = list(scale = 1))
plot(GS_cond_roll, which = 4)
```

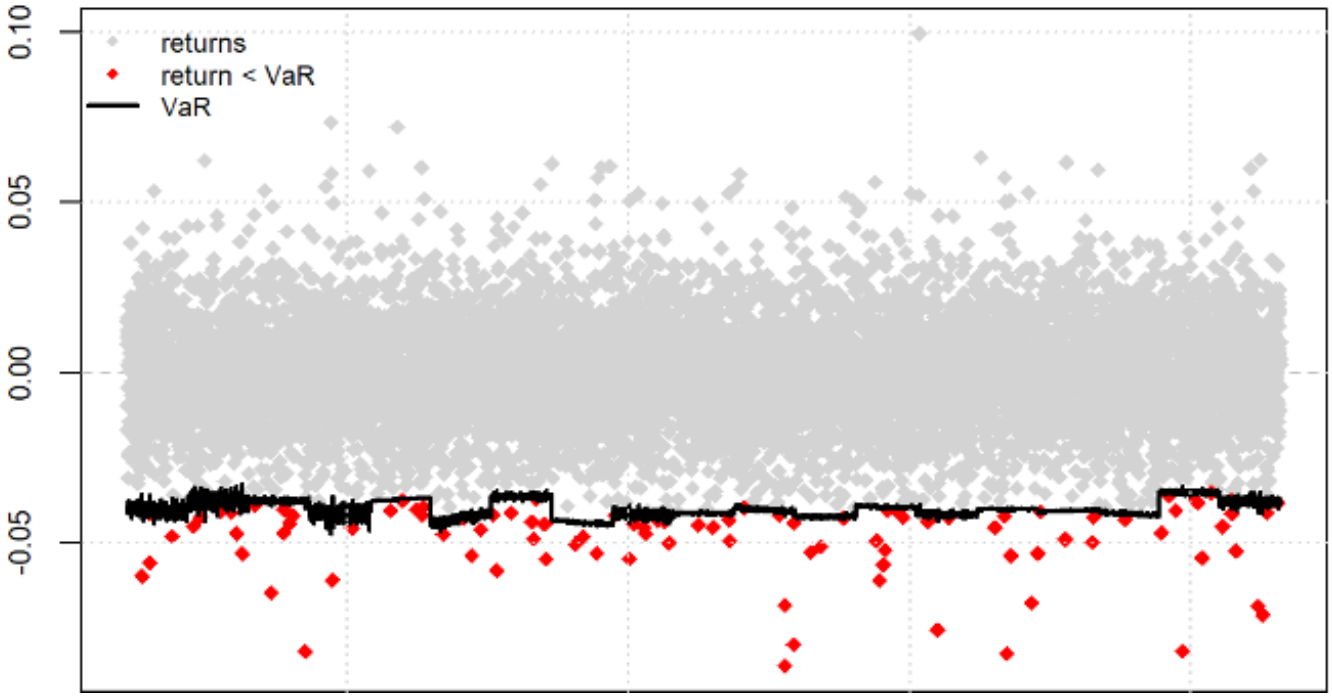
The above code is used to run the conditional rolling forecast for both of our series, based on refitting our model after every 500 observations for our 10000 random samples. Below are the two conditional rolling forecasts SPXL and BAC.

SPXL Rolling forecast



As our initial hypothesis, the log returns for SPXL have a very wide range and are quite spread out in comparison to the range for log-returns of GS below. This indicates a lager variance and fatter tails. It also increases chances of very huge sudden losses and definitely increase the expected shortfall.

Goldman Sachs Rolling forecast



As our initial hypothesis, the log returns for Goldman Sachs have a very small range and are not to spread out. This indicates a smaller variance and thinner tails. It also reduces chances of very huge sudden losses and definitely reduces the expected shortfall.

Finally, we predict the VaR and the predicted VaR is -0.124 for a portfolio of both securities.

Various Rho's

rho	VaR
0.1	0.01813
0.2	0.01786
0.3	0.01784
0.4	0.01806
0.5	0.01849
0.6	0.01911
0.7	0.01991
0.8	0.02085
0.9	0.02191

Following table shows the VaR for different shares of the SPXL in the portfolio. Portfolios can reduce risk by decreasing the portfolio's exposure to SPXL. For example, if the portfolio is 90% SPXL and 10% Goldman Sachs, there is a 1% chance of losing more than 2.191 cents per dollar invested. If the portfolio is 10% SPXL and 90% Goldman Sachs, there is a 1% chance of losing more than 1.813 cents per dollar invested. However, it should be noted that VaR discriminates against diversifying stock portfolios.

Conclusion

We started off by calculating log returns and exploring our data, in order to understand the nature and dispersion. We concluded that SPXL has a higher variance based on kurtosis. The skewness test also revealed that its returns were more skewed towards left. The AR(1,0)-Garch(1,1) model is used to model our mean and variance. We want to see if our residuals are not serially correlated with and independent and therefore, we conducted an in-depth residual analysis to conclude that is the case for our standardized residuals. We fit the t-distribution to our residuals, which would be used later for generating random samples and risk calculations. The copulas use these residuals to come up with Rho, to be utilized for probabilities and making a simulation.

This build up has been for our main aim of the project, to calculate the the VaR for various Rho's, indicating various weight of both securities in the portfolio. Such an analysis was not possible without determining if our variance is homoscedastic or heteroscedastic, errors are independent distributed or serially correlated and has zero conditional mean. This analysis is important before forecasting variance and calculating the VaR and observation below VaR. The VaR could have initially been calculated for our base case data, but the base data has trend and seasonality, and we don't know if variance is independent and not serially correlated.

Based on my analysis, SPXL is a risker security to invest in comparison to Goldman Sachs and a higher composition of SPXL increases our VaR and Expected shortfall. The exploratory data analysis, fitting AR-GARCH models, diagnosing time series models and estimating risk measures are all indicating us that SPXL has higher VaR.

Appendix

Financial Econometrics - Final project

Fahad Alhodaithy

2022-11-05

##Loading Libraries

```
library(quantmod)
library(ggplot2)
library(fitdistrplus)
library(moments)
library(Rcpp)
library(knitr)
library(kableExtra)
library(rugarch)
library(QRM)
library(fGarch)
library(LambertW)
library(copula)
library(sn)
library(ks)
library(base.rms)
library(TSstudio)
library(tidyverse)
library(forecast)
library(htmlTable)
library(tseries)
```

Reading & Cleaning the data

```
# Past ten years' stock prices for SPXL and Goldman Sachs from Yahoo Finance

spxl <- getSymbols("SPXL", src = "yahoo", from = "2012-07-01", to = "2022-07-01", auto.assign =
FALSE)
gs <- getSymbols("GS", src = "yahoo", from = "2012-07-01", to = "2022-07-01", auto.assign = FALS
E)

# Calculating Log returns

spxl_log_ret <- (diff(log(spxl[,6])))[-1,]
gs_log_ret <- (diff(log(gs[,6])))[-1,]
```

Exploring the data to determine necessity for modeling time dependence and volatility of log returns.

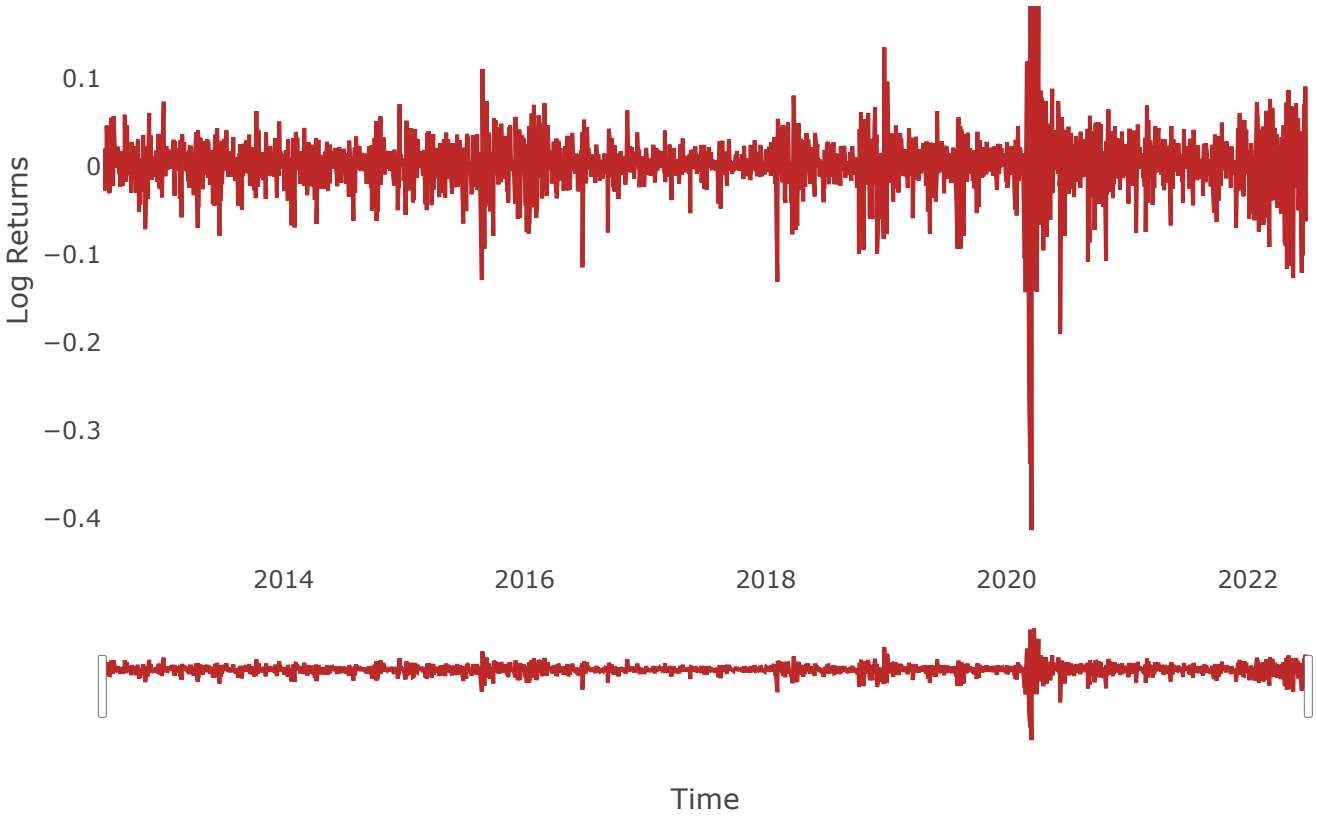
```
summary(spxl_log_ret) %>%
  kbl() %>%
  kable_styling()
```

Index	SPXL.Adjusted
Min. :2012-07-03	Min. :-0.4135766
1st Qu.:2015-01-03	1st Qu.:-0.0101632
Median :2017-07-03	Median : 0.0019941
Mean :2017-07-02	Mean : 0.0009503
3rd Qu.:2020-01-01	3rd Qu.: 0.0157774
Max. :2022-06-30	Max. : 0.2453456

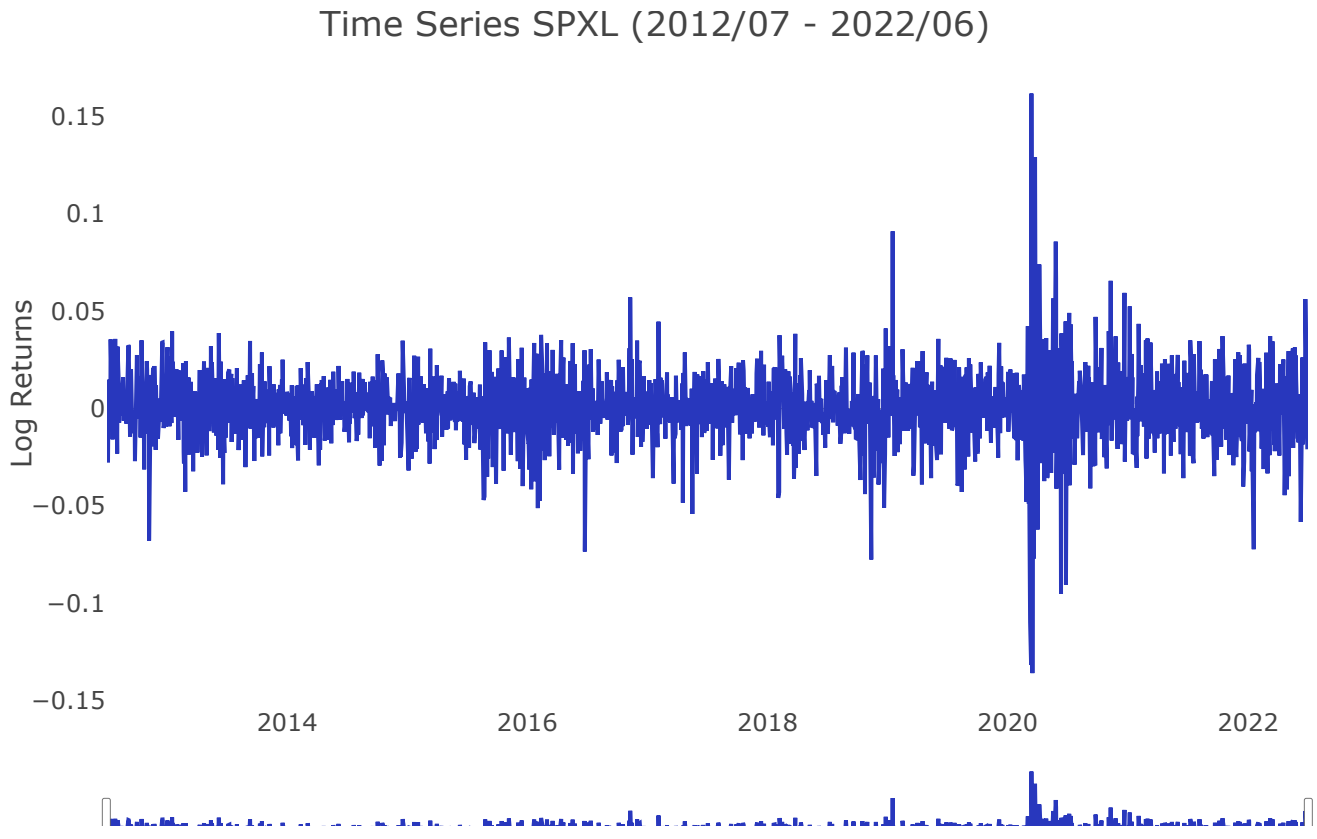
```
summary(gs_log_ret) %>%
  kbl() %>%
  kable_styling()
```

Index	GS.Adjusted
Min. :2012-07-03	Min. :-0.1358805
1st Qu.:2015-01-03	1st Qu.:-0.0082341
Median :2017-07-03	Median : 0.0006480
Mean :2017-07-02	Mean : 0.0004976
3rd Qu.:2020-01-01	3rd Qu.: 0.0095834
Max. :2022-06-30	Max. : 0.1619514

```
# Time Series Plots

ts_plot(spxl_log_ret,
        title = "Time Series SPXL (2012/07 - 2022/06)",
        Xtitle = "Time",
        Ytitle = "Log Returns",
        color = "#bd2828",
        slider = TRUE)
```

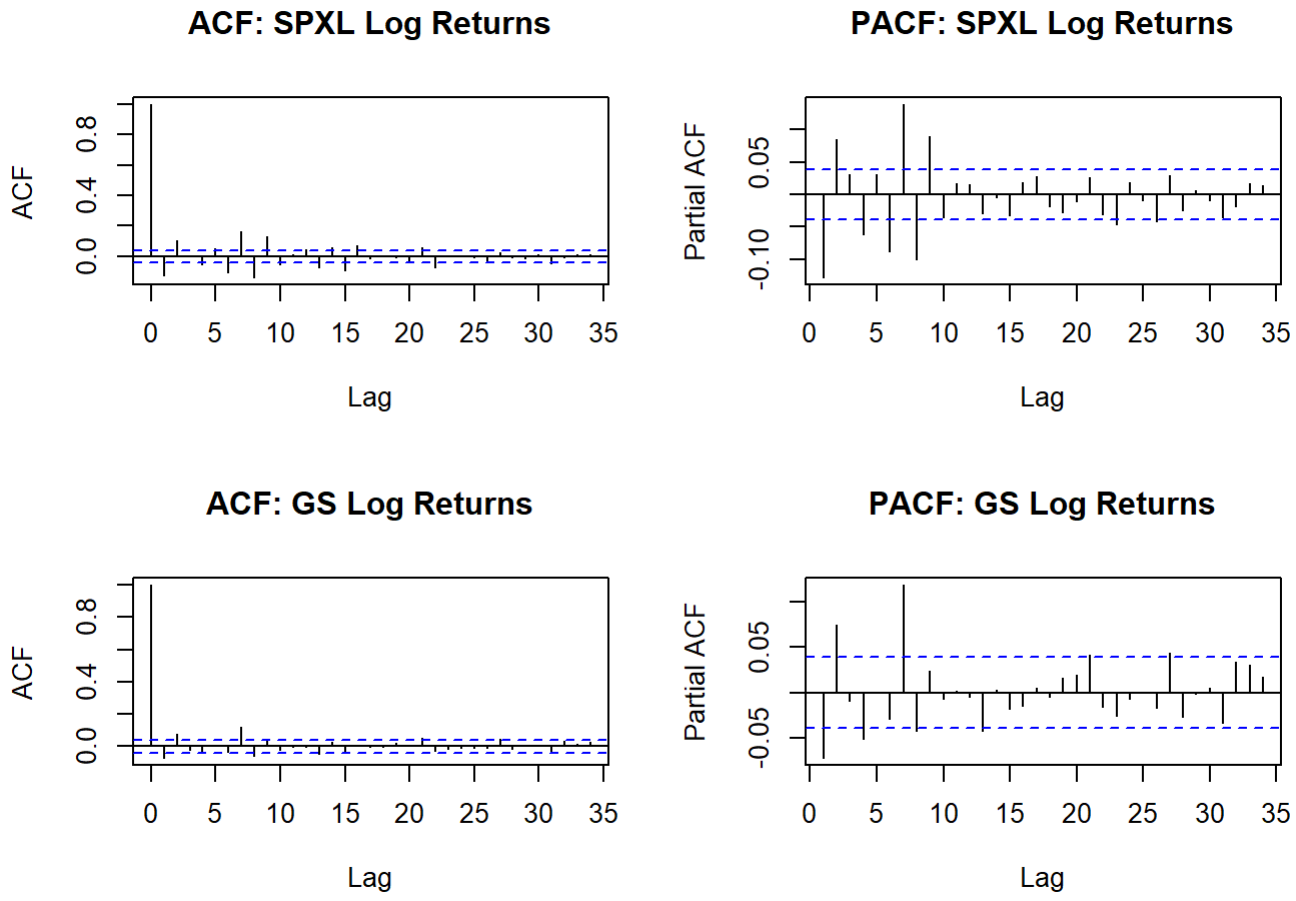



```
ts_plot(gs_log_ret,
        title = "Time Series SPXL (2012/07 - 2022/06)",
        Xtitle = "Time",
        Ytitle = "Log Returns",
        color =  "#2837bd",
        slider = TRUE)
```



Time

```
# ACF & PACF Plots

par(mfrow=c(2,2))
acf(spxl_log_ret, main = "ACF: SPXL Log Returns")
pacf(spxl_log_ret, main = "PACF: SPXL Log Returns")
acf(gs_log_ret, main = "ACF: GS Log Returns")
pacf(gs_log_ret, main = "PACF: GS Log Returns")
```




```
#Check for Skewness and Kurtosis

sk1 <- skewness(spxl_log_ret)
print(paste("SPXL Log Returns skewness = ", sk1))
```

```
## [1] "SPXL Log Returns skewness =  -1.61288243148022"
```

```
sk2 <- skewness(gs_log_ret)
print(paste("Goldman Sachs Log Returns skewness = ", sk2))
```

```
## [1] "Goldman Sachs Log Returns skewness =  -0.238138046597843"
```

```
k1 <- kurtosis(spxl_log_ret)
print(paste("SPXL Log Returns kurtosis = ", k1))
```

```
## [1] "SPXL Log Returns kurtosis =  25.386326637088"
```

```
k2 <- kurtosis(gs_log_ret)
print(paste("Goldman Sachs Log Returns kurtosis = ", k2))
```

```
## [1] "Goldman Sachs Log Returns kurtosis =  13.0008395025853"
```

```
#Test for serial correlation

Box.test(x=spxl_log_ret,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  spxl_log_ret
## X-squared = 81.037, df = 5, p-value = 5.551e-16
```

```
Box.test(x=gs_log_ret,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  gs_log_ret
## X-squared = 34.305, df = 5, p-value = 2.071e-06
```

```
adf.test(spxl_log_ret)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  spxl_log_ret
## Dickey-Fuller = -13.265, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(gs_log_ret)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  gs_log_ret
## Dickey-Fuller = -13.535, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

Building Time Series Models

```
# Fit AR(1)-GARCH(1,1) model

SPXLfit=garchFit(formula=~arma(1,0)+garch(1,1),data=spxl_log_ret,cond.dist="norm")
```

```
##
## Series Initialization:
##  ARMA Model:                arma
##  Formula Mean:              ~ arma(1, 0)
##  GARCH Model:               garch
##  Formula Variance:          ~ garch(1, 1)
##  ARMA Order:                1 0
##  Max ARMA Order:            1
##  GARCH Order:               1 1
##  Max GARCH Order:           1
##  Maximum Order:             1
##  Conditional Dist:          norm
##  h.start:                   2
##  llh.start:                 1
##  Length of Series:          2515
##  Recursion Init:            mci
##  Series Scale:              0.03234779
##
## Parameter Initialization:
##  Initial Parameters:          $params
##  Limits of Transformations:   $U, $V
##  Which Parameters are Fixed?  $includes
##  Parameter Matrix:
##                      U          V      params includes
##     mu     -0.29378395   0.293784  0.02939386     TRUE
##     ar1    -0.99999999   1.000000 -0.12850736     TRUE
##     omega   0.00000100 100.000000  0.10000000     TRUE
##     alpha1  0.00000001   1.000000  0.10000000     TRUE
##     gamma1 -0.99999999   1.000000  0.10000000    FALSE
##     beta1   0.00000001   1.000000  0.80000000     TRUE
##     delta   0.00000000   2.000000  2.00000000    FALSE
##     skew    0.10000000  10.000000  1.00000000    FALSE
##     shape   1.00000000  10.000000  4.00000000    FALSE
##  Index List of Parameters to be Optimized:
##     mu    ar1  omega alpha1  beta1
##      1      2      3      4      6
##  Persistence:                  0.9
##
##
## --- START OF TRACE ---
## Selected Algorithm: nlminb
##
## R coded nlminb Solver:
##
##   0:     3005.0321: 0.0293939 -0.128507 0.100000 0.100000 0.800000
##   1:     2928.0110: 0.0293952 -0.126767 0.0723505 0.0979605 0.783461
##   2:     2896.9822: 0.0293987 -0.122725 0.0458178 0.115650 0.780001
##   3:     2893.2776: 0.0294043 -0.117861 0.0583013 0.143269 0.790144
##   4:     2864.8828: 0.0294198 -0.106707 0.0384140 0.158312 0.772861
##   5:     2858.4341: 0.0294666 -0.0806869 0.0434185 0.176260 0.768560
##   6:     2856.3934: 0.0295729 -0.0521524 0.0305840 0.179941 0.774856
##   7:     2853.5103: 0.0297338 -0.0348436 0.0428334 0.191169 0.753891
```

```
##   8:     2852.6114: 0.0300064 -0.0567449 0.0413549 0.209441 0.741915
##   9:     2852.2809: 0.0300372 -0.0515582 0.0387815 0.215075 0.745326
##  10:     2852.2055: 0.0300906 -0.0449833 0.0400190 0.212522 0.740488
##  11:     2852.1830: 0.0303034 -0.0431898 0.0424417 0.216524 0.739822
##  12:     2851.9092: 0.0304056 -0.0422167 0.0407495 0.217790 0.738412
##  13:     2851.4178: 0.0326753 -0.0341337 0.0402903 0.230088 0.734146
##  14:     2849.0263: 0.0468923 -0.0392528 0.0334985 0.203826 0.764783
##  15:     2847.8000: 0.0611191 -0.0479747 0.0480447 0.207554 0.731985
##  16:     2847.1425: 0.0611199 -0.0478370 0.0442255 0.208252 0.731736
##  17:     2846.7927: 0.0611507 -0.0476663 0.0452839 0.210957 0.734100
##  18:     2846.4738: 0.0613648 -0.0477325 0.0426801 0.211796 0.733894
##  19:     2846.2442: 0.0618163 -0.0479609 0.0435346 0.213844 0.735037
##  20:     2846.0075: 0.0627286 -0.0485435 0.0418117 0.215266 0.734476
##  21:     2845.7410: 0.0645564 -0.0496498 0.0425924 0.217920 0.734185
##  22:     2845.4120: 0.0682141 -0.0510577 0.0415838 0.220951 0.731714
##  23:     2845.0796: 0.0755283 -0.0419040 0.0422067 0.223395 0.730389
##  24:     2844.8992: 0.0785523 -0.0509610 0.0409920 0.229575 0.729341
##  25:     2844.8822: 0.0779098 -0.0487582 0.0403409 0.229509 0.730790
##  26:     2844.8760: 0.0779179 -0.0475404 0.0406912 0.231269 0.729725
##  27:     2844.8753: 0.0779197 -0.0476959 0.0406309 0.231213 0.729654
##  28:     2844.8753: 0.0779239 -0.0477318 0.0406209 0.231152 0.729725
##  29:     2844.8753: 0.0779233 -0.0477254 0.0406226 0.231161 0.729716
##
## Final Estimate of the Negative LLH:
##  LLH:  -5784.617    norm LLH:  -2.300046
##            mu           ar1         omega        alpha1         beta1
##  2.520645e-03 -4.772542e-02  4.250671e-05  2.311606e-01  7.297157e-01
##
## R-optimhess Difference Approximated Hessian Matrix:
##                  mu          ar1         omega        alpha1         beta1
## mu      -6243310.06 -17010.56467 -3.292262e+07  1.043603e+04 -9.234850e+03
## ar1       -17010.56  -1987.33540  3.126423e+05 -3.594699e+01  2.572742e+01
## omega  -32922615.56 312642.29140 -1.158706e+11 -2.320147e+07 -4.329593e+07
## alpha1     10436.03    -35.94699 -2.320147e+07 -1.104536e+04 -1.434097e+04
## beta1      -9234.85     25.72742 -4.329593e+07 -1.434097e+04 -2.338020e+04
## attr(,"time")
## Time difference of 0.03624201 secs
##
## --- END OF TRACE ---
##
##
## Time to Estimate Parameters:
##  Time difference of 0.4380729 secs
```

```
GSfit=garchFit(formula=~arma(1,0)+garch(1,1),data=gs_log_ret,cond.dist="norm")
```

```
##
## Series Initialization:
##  ARMA Model:                arma
##  Formula Mean:              ~ arma(1, 0)
##  GARCH Model:               garch
##  Formula Variance:          ~ garch(1, 1)
##  ARMA Order:                1 0
##  Max ARMA Order:            1
##  GARCH Order:               1 1
##  Max GARCH Order:           1
##  Maximum Order:             1
##  Conditional Dist:          norm
##  h.start:                   2
##  llh.start:                 1
##  Length of Series:          2515
##  Recursion Init:            mci
##  Series Scale:              0.01756295
##
## Parameter Initialization:
##  Initial Parameters:          $params
##  Limits of Transformations:   $U, $V
##  Which Parameters are Fixed?  $includes
##  Parameter Matrix:
##                      U           V      params includes
##     mu     -0.28332336   0.2833234  0.02834268     TRUE
##     ar1    -0.99999999   1.0000000 -0.07157075     TRUE
##     omega   0.00000100 100.0000000  0.10000000     TRUE
##     alpha1  0.00000001   1.0000000  0.10000000     TRUE
##     gamma1 -0.99999999   1.0000000  0.10000000    FALSE
##     beta1   0.00000001   1.0000000  0.80000000     TRUE
##     delta   0.00000000   2.0000000  2.00000000    FALSE
##     skew    0.10000000  10.0000000  1.00000000    FALSE
##     shape   1.00000000  10.0000000  4.00000000    FALSE
##  Index List of Parameters to be Optimized:
##     mu    ar1  omega alpha1  beta1
##      1      2      3      4      6
##  Persistence:                 0.9
##
##
## --- START OF TRACE ---
## Selected Algorithm: nlminb
##
## R coded nlminb Solver:
##
##   0:     3273.0024: 0.0283427 -0.0715707 0.100000 0.100000 0.800000
##   1:     3263.8671: 0.0283430 -0.0699908 0.0896139 0.0974579 0.793761
##   2:     3256.2485: 0.0283470 -0.0520408 0.0582254 0.123848 0.799168
##   3:     3252.4837: 0.0283473 -0.0510715 0.0636831 0.126905 0.803869
##   4:     3251.7622: 0.0283491 -0.0456472 0.0586935 0.125381 0.806219
##   5:     3249.4446: 0.0283534 -0.0344847 0.0586391 0.120743 0.816347
##   6:     3247.8534: 0.0283592 -0.0268135 0.0522670 0.112873 0.825688
##   7:     3246.4817: 0.0283671 -0.0268015 0.0482368 0.107961 0.840119
```

```
##   8:     3245.4874: 0.0283787 -0.0283583 0.0404585 0.101442 0.852081
##   9:     3244.3575: 0.0284084 -0.0164028 0.0399916 0.0954203 0.860338
##  10:     3243.8363: 0.0284890 -0.0159628 0.0303705 0.0943491 0.872450
##  11:     3243.3092: 0.0286322 -0.00581854 0.0328654 0.0846383 0.876888
##  12:     3243.2660: 0.0286324 -0.00575051 0.0330376 0.0850405 0.877284
##  13:     3243.2341: 0.0286326 -0.00563796 0.0324798 0.0852042 0.877335
##  14:     3243.2035: 0.0286534 -0.00583287 0.0323233 0.0852819 0.878231
##  15:     3243.1707: 0.0287035 -0.00638170 0.0314769 0.0846371 0.879275
##  16:     3243.1327: 0.0288091 -0.00387397 0.0318377 0.0858317 0.878339
##  17:     3243.0822: 0.0289959 -0.00727006 0.0297780 0.0831393 0.883232
##  18:     3243.0703: 0.0289971 -0.00658250 0.0287068 0.0837081 0.883596
##  19:     3243.0462: 0.0290348 -0.00646421 0.0290139 0.0838797 0.884014
##  20:     3243.0365: 0.0290750 -0.00627725 0.0288969 0.0837796 0.883922
##  21:     3243.0240: 0.0291521 -0.00540078 0.0288864 0.0838030 0.884286
##  22:     3243.0062: 0.0293147 -0.00492211 0.0288312 0.0836258 0.884096
##  23:     3242.9236: 0.0313734 -0.00139557 0.0295754 0.0829034 0.883871
##  24:     3242.8344: 0.0334324 -0.000909928 0.0278578 0.0821016 0.886714
##  25:     3242.8035: 0.0354930 -0.00221920 0.0282319 0.0828025 0.885768
##  26:     3242.7730: 0.0375537 -0.000781277 0.0278673 0.0823308 0.886670
##  27:     3242.7693: 0.0383737 -0.000783128 0.0279263 0.0824638 0.886224
##  28:     3242.7650: 0.0391934 -0.000188366 0.0277302 0.0822038 0.886806
##  29:     3242.7648: 0.0394127 -6.93234e-05 0.0277166 0.0821736 0.886895
##  30:     3242.7648: 0.0394114 -7.05029e-05 0.0277142 0.0821712 0.886900
##
## Final Estimate of the Negative LLH:
##  LLH:  -6922.773    norm LLH:  -2.752594
##             mu           ar1         omega        alpha1         beta1
##  6.921799e-04 -7.050295e-05  8.548638e-06  8.217120e-02  8.868996e-01
##
## R-optimhess Difference Approximated Hessian Matrix:
##                   mu         ar1         omega        alpha1         beta1
## mu     -12276283.254  -9788.1392 -3.695413e+07  6.801283e+03 -2.137485e+03
## ar1        -9788.139  -2173.7887 -1.415934e+04 -1.366369e+02 -2.944649e+02
## omega  -36954131.430 -14159.3403 -2.503949e+12 -3.708905e+08 -5.001332e+08
## alpha1      6801.283   -136.6369 -3.708905e+08 -8.115929e+04 -8.976307e+04
## beta1      -2137.485   -294.4649 -5.001332e+08 -8.976307e+04 -1.120974e+05
## attr(,"time")
## Time difference of 0.04440093 secs
##
## --- END OF TRACE ---
##
##
## Time to Estimate Parameters:
##  Time difference of 0.145443 secs
```

```
summary(SPXLfit)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~arma(1, 0) + garch(1, 1), data = spxl_log_ret,
##     cond.dist = "norm")
##
## Mean and Variance Equation:
##  data ~ arma(1, 0) + garch(1, 1)
## <environment: 0x000000002a1cfb18>
##  [data = spxl_log_ret]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu          ar1        omega       alpha1        beta1
##  2.5206e-03  -4.7725e-02   4.2507e-05   2.3116e-01   7.2972e-01
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##          Estimate  Std. Error  t value Pr(>|t|)
## mu      2.521e-03   4.092e-04    6.159 7.31e-10 ***
## ar1    -4.773e-02   2.273e-02   -2.100   0.0357 *
## omega   4.251e-05   5.711e-06    7.443 9.81e-14 ***
## alpha1  2.312e-01   2.296e-02   10.068  < 2e-16 ***
## beta1   7.297e-01   2.156e-02   33.853  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  5784.617    normalized:  2.300046
##
## Description:
##  Fri Nov 25 19:42:52 2022 by user: fahad
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  904.9919  0
##  Shapiro-Wilk Test  R    W      0.9648744 0
##  Ljung-Box Test     R    Q(10)  8.763817  0.5546554
##  Ljung-Box Test     R    Q(15)  15.51002  0.4153412
##  Ljung-Box Test     R    Q(20)  18.92452  0.5267373
##  Ljung-Box Test     R^2  Q(10)  9.467648  0.4883698
##  Ljung-Box Test     R^2  Q(15)  11.85707  0.6898137
##  Ljung-Box Test     R^2  Q(20)  14.90005  0.782098
##  LM Arch Test       R    TR^2   11.68866  0.4709966
##
```

```
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -4.596117 -4.584526 -4.596125 -4.591910
```

```
summary(GSfit)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~arma(1, 0) + garch(1, 1), data = gs_log_ret,
##     cond.dist = "norm")
##
## Mean and Variance Equation:
##  data ~ arma(1, 0) + garch(1, 1)
## <environment: 0x000000003b47d7f0>
##  [data = gs_log_ret]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu          ar1        omega       alpha1        beta1
##  6.9218e-04  -7.0503e-05   8.5486e-06   8.2171e-02   8.8690e-01
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##          Estimate  Std. Error  t value Pr(>|t|)
## mu      6.922e-04   2.860e-04    2.420 0.015514 *
## ar1    -7.050e-05   2.157e-02   -0.003 0.997392
## omega   8.549e-06   2.377e-06    3.596 0.000323 ***
## alpha1  8.217e-02   1.287e-02    6.385 1.71e-10 ***
## beta1   8.869e-01   1.889e-02   46.951  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  6922.773    normalized:  2.752594
##
## Description:
##  Fri Nov 25 19:42:52 2022 by user: fahad
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  342.6092  0
##  Shapiro-Wilk Test  R    W      0.984021  3.266981e-16
##  Ljung-Box Test     R    Q(10)  13.63311  0.1903939
##  Ljung-Box Test     R    Q(15)  18.44976  0.2397588
##  Ljung-Box Test     R    Q(20)  21.30093  0.379614
##  Ljung-Box Test     R^2  Q(10)  14.65039  0.1453375
##  Ljung-Box Test     R^2  Q(15)  17.19789  0.3071727
##  Ljung-Box Test     R^2  Q(20)  17.73353  0.6049572
##  LM Arch Test       R    TR^2   15.24154  0.2284926
##
```

```
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -5.501212 -5.489621 -5.501219 -5.497005
```

```
# Mean Residuals from AR-GARCH model

SPXL_res=residuals(SPXLfit)
GS_res=residuals(GSfit)

# SD Residuals from AR-GARCH model

SPXL_res_sd=residuals(SPXLfit,standardize=TRUE)
GS_res_sd=residuals(GSfit,standardize=TRUE)

#Fit t-distribution

#Fit t-distribution to Mean residuals

std.SPXL_res = as.numeric(fitdistr(SPXL_res,"t")$estimate)
std.GS_res = as.numeric(fitdistr(GS_res,"t")$estimate)

std.SPXL_res[2] = std.SPXL_res[2] * sqrt(std.SPXL_res[3] / (std.SPXL_res[3]-2))
std.GS_res[2] = std.GS_res[2] * sqrt(std.GS_res[3] / (std.GS_res[3]-2))

#Fit t-distribution to SD residuals

std.SPXL_res_sd = as.numeric(fitdistr(SPXL_res_sd,"t")$estimate)
std.GS_res_sd = as.numeric(fitdistr(GS_res_sd,"t")$estimate)

std.SPXL_res_sd[2] = std.SPXL_res_sd[2] * sqrt(std.SPXL_res_sd[3] / (std.SPXL_res_sd[3]-2))
std.GS_res_sd[2] = std.GS_res_sd[2] * sqrt(std.GS_res_sd[3] / (std.GS_res_sd[3]-2))
```

Fitting Copulas to Mean Residuals:

```
# Estimate of the correlation coefficient in the t-copula using Kendall's tau

cor_tau = cor(SPXL_res, GS_res, method = "kendall", use="pairwise.complete.obs")
round(cor_tau,2)
```

```
## [1] 0.51
```

```
omega = sin((pi/2)*cor_tau) #estimator for rho
round(omega,2)
```

```
## [1] 0.71
```

```
# Combining datasets:

data1 = cbind(pstd(SPXL_res, std.SPXL_res[1], std.SPXL_res[2], std.SPXL_res[3]),
              pstd(GS_res, std.GS_res[1], std.GS_res[2], std.GS_res[3]))

# Fit t-copula
cop_t_dim2 = tCopula(omega, dim = 2, dispstr = "un", df = 4) #define t copula
Ct=fitCopula(cop_t_dim2, data1, method="ml", start=c(omega,4) )
Ct@estimate
```

```
## [1] 0.7162246 5.1624755
```

```
lst_value <- loglikCopula(param=Ct@estimate,u=data1,copula=tCopula(dim=2));#compute loglikelihoo
d function
AIC_Ct <- (-2) * lst_value + 2*length(Ct@estimate) #compute AIC

# Fit Gaussian copula
Cgauss=fitCopula(copula=normalCopula(dim=2),data=data1, method="ml")
Cgauss@estimate
```

```
## [1] 0.7106091
```

```
lst_value <- loglikCopula(param=Cgauss@estimate,u=data1,copula=normalCopula(dim=2))
AIC_gauss <- (-2)*lst_value+2*length(Cgauss@estimate)#compute AIC

# Fit Gumbel copula
Cgu=fitCopula(copula=gumbelCopula(3,dim=2),data=data1,method="ml")
Cgu@estimate
```

```
## [1] 1.946992
```

```
lst_value <- loglikCopula(param=Cgu@estimate,u=data1,copula=gumbelCopula(dim=2))
AIC_Cgu <- (-2)*lst_value+2*length(Cgu@estimate)

# Fit Clayton copula
Ccl=fitCopula(copula=claytonCopula(1,dim=2),data=data1,method="ml")
Ccl@estimate
```

```
## [1] 2.046358
```

```
lst_value <- loglikCopula(param=Ccl@estimate,u=data1,copula=claytonCopula(dim=2))
AIC_Ccl <- (-2)*lst_value+2*length(Ccl@estimate)
```

Fitting Copulas to SD Residuals:

```
# Estimate of the correlation coefficient in the t-copula using Kendall's tau

cor_tau = cor(SPXL_res_sd, GS_res_sd, method = "kendall", use="pairwise.complete.obs")
round(cor_tau,2)
```

```
## [1] 0.49
```

```
omega = sin((pi/2)*cor_tau) #estimator for rho
round(omega,2)
```

```
## [1] 0.69
```

```
# Combining datasets:

data1 = cbind(pstd(SPXL_res_sd, std.SPXL_res_sd[1], std.SPXL_res_sd[2], std.SPXL_res_sd[3]),
              pstd(GS_res_sd, std.GS_res_sd[1], std.GS_res_sd[2], std.GS_res_sd[3]))

# Fit t-copula
cop_t_dim2 = tCopula(omega, dim = 2, dispstr = "un", df = 4) #define t copula
Ct=fitCopula(cop_t_dim2, data1, method="ml", start=c(omega,4) )
Ct@estimate
```

```
## [1] 0.6915893 6.1749177
```

```
lst_value <- loglikCopula(param=Ct@estimate,u=data1,copula=tCopula(dim=2));#compute loglikelihoo
d function
AIC_Ct <- (-2) * lst_value + 2*length(Ct@estimate) #compute AIC

# Fit Gaussian copula
Cgauss=fitCopula(copula=normalCopula(dim=2),data=data1, method="ml")
Cgauss@estimate
```

```
## [1] 0.6855186
```

```
lst_value <- loglikCopula(param=Cgauss@estimate,u=data1,copula=normalCopula(dim=2))
AIC_gauss <- (-2)*lst_value+2*length(Cgauss@estimate)#compute AIC

# Fit Gumbel copula
Cgu=fitCopula(copula=gumbelCopula(3,dim=2),data=data1,method="ml")
Cgu@estimate
```

```
## [1] 1.887085
```

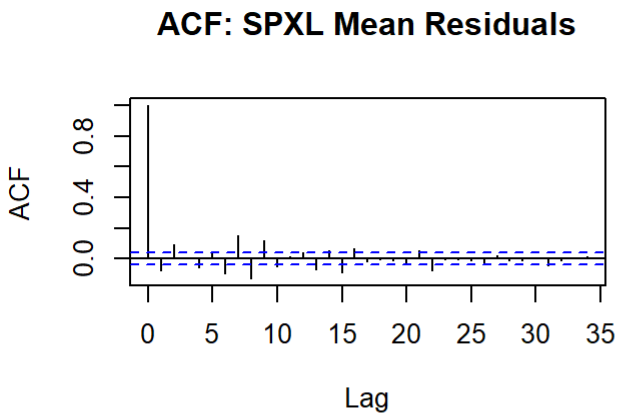
```
lst_value <- loglikCopula(param=Cgu@estimate,u=data1,copula=gumbelCopula(dim=2))
AIC_Cgu <- (-2)*lst_value+2*length(Cgu@estimate)

# Fit Clayton copula
Ccl=fitCopula(copula=claytonCopula(1,dim=2),data=data1,method="ml")
Ccl@estimate
```

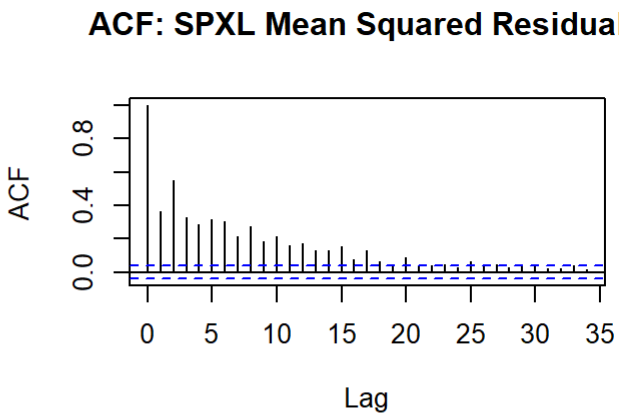
```
## [1] 1.913672
```

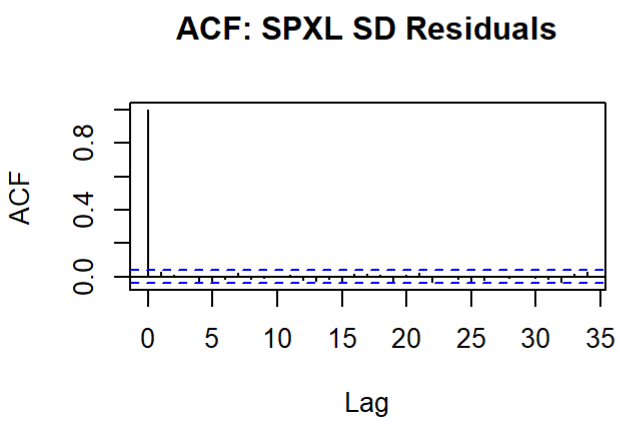
```
lst_value <- loglikCopula(param=Ccl@estimate,u=data1,copula=claytonCopula(dim=2))
AIC_Ccl <- (-2)*lst_value+2*length(Ccl@estimate)
```

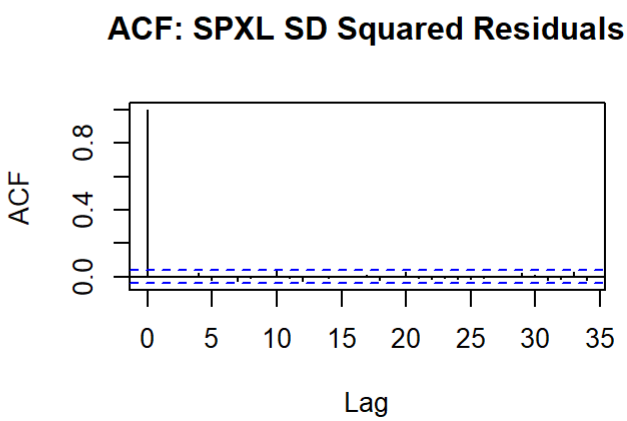
Residual Analysis

```
## SPXL

# ACF Plots

par(mfrow=c(2,2))
acf(SPXL_res, main = "ACF: SPXL Mean Residuals")
acf(SPXL_res^2, main = "ACF: SPXL Mean Squared Residuals")
acf(SPXL_res_sd, main = "ACF: SPXL SD Residuals")
acf(SPXL_res_sd^2, main = "ACF: SPXL SD Squared Residuals")
```










```
#Test for serial correlation

Box.test(x=SPXL_res,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  SPXL_res
## X-squared = 50.063, df = 5, p-value = 1.345e-09
```

```
Box.test(x=SPXL_res_sd,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  SPXL_res_sd
## X-squared = 6.2073, df = 5, p-value = 0.2866
```

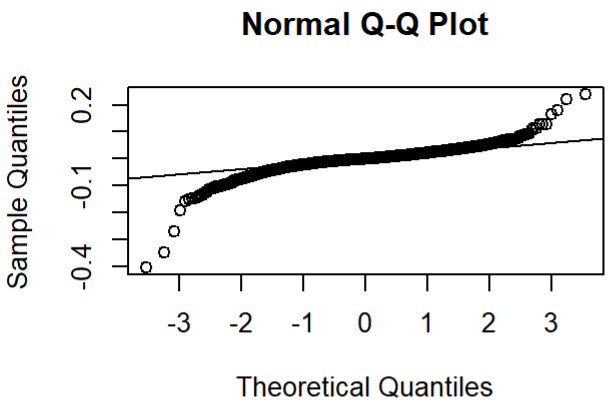
```
adf.test(SPXL_res)
```

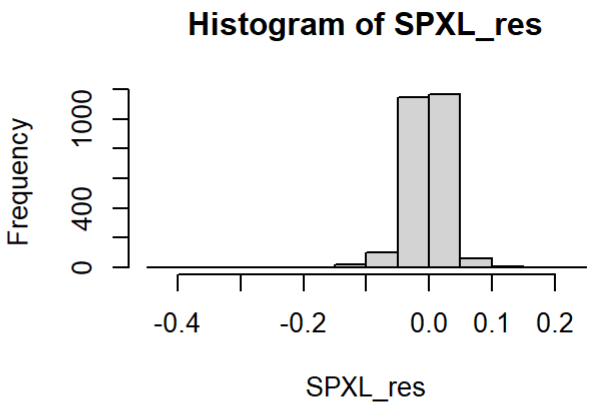
```
##
##  Augmented Dickey-Fuller Test
##
## data:  SPXL_res
## Dickey-Fuller = -13.242, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

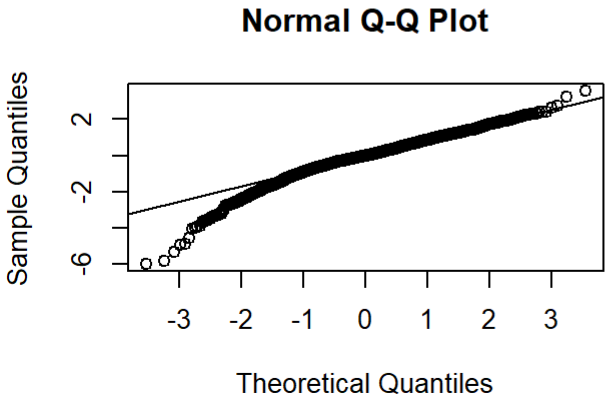
```
adf.test(SPXL_res_sd)
```

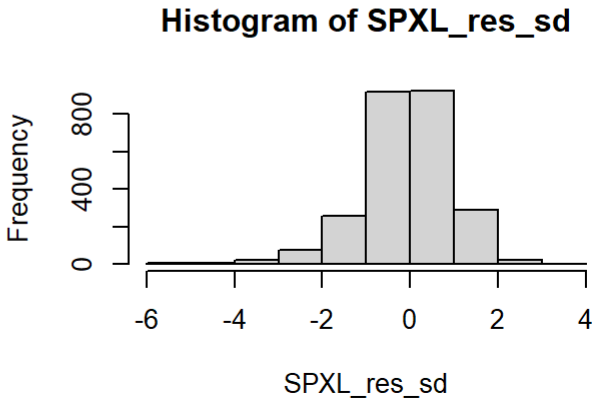
```
##
##  Augmented Dickey-Fuller Test
##
## data:  SPXL_res_sd
## Dickey-Fuller = -14.386, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

```
# Check Normal Distribution

qqnorm(SPXL_res)
qqline(SPXL_res)
hist(SPXL_res)

qqnorm(SPXL_res_sd)
qqline(SPXL_res_sd)
hist(SPXL_res_sd)
```

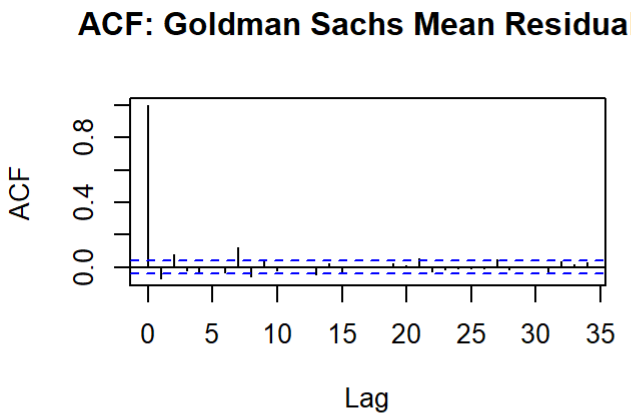


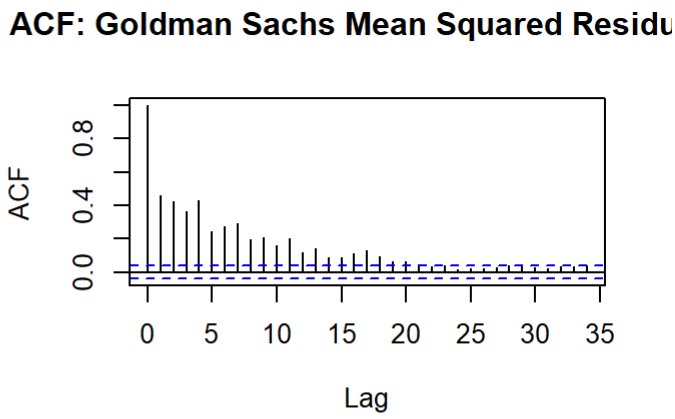


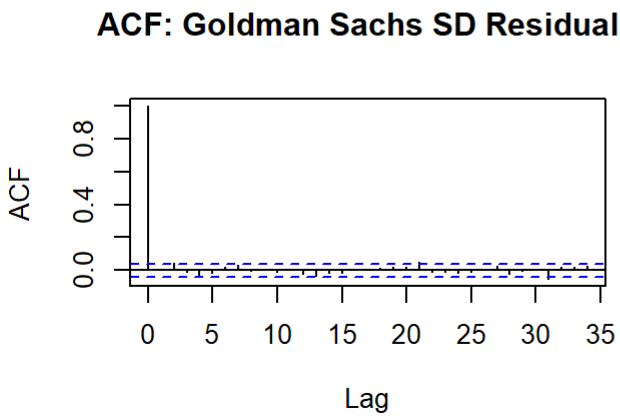


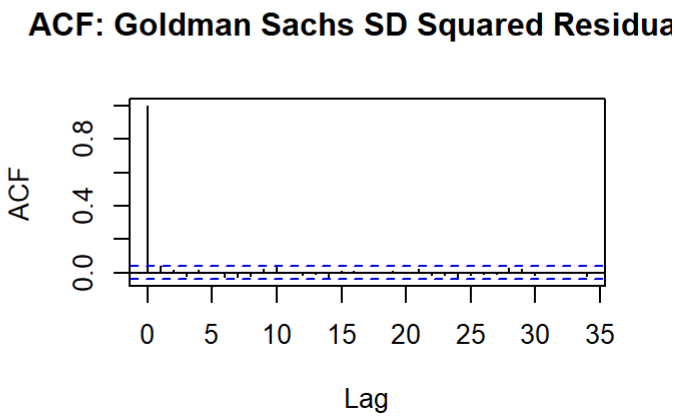



```
## GS

# ACF Plots

acf(GS_res, main = "ACF: Goldman Sachs Mean Residuals")
acf(GS_res^2, main = "ACF: Goldman Sachs Mean Squared Residuals")
acf(GS_res_sd, main = "ACF: Goldman Sachs SD Residuals")
acf(GS_res_sd^2, main = "ACF: Goldman Sachs SD Squared Residuals")
```

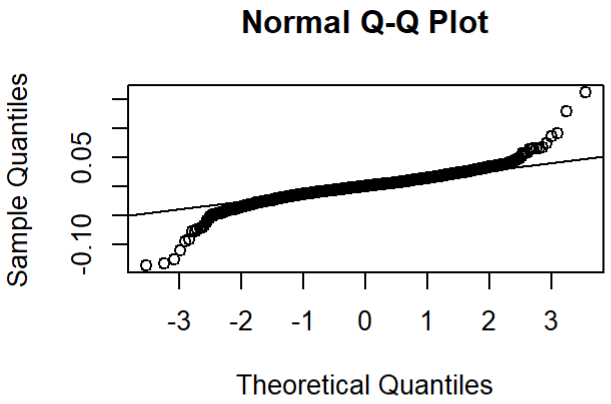


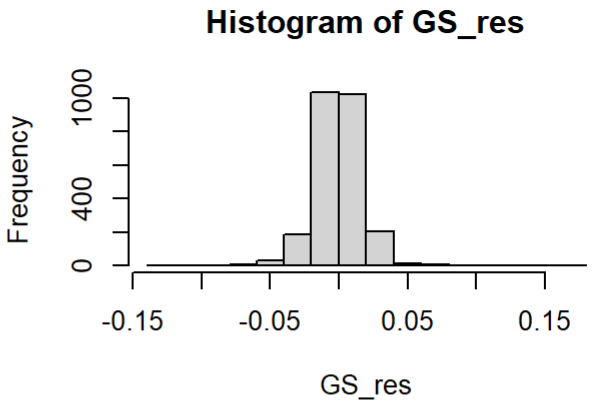


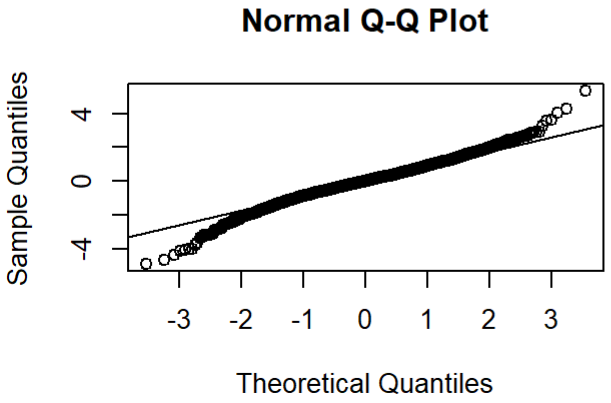


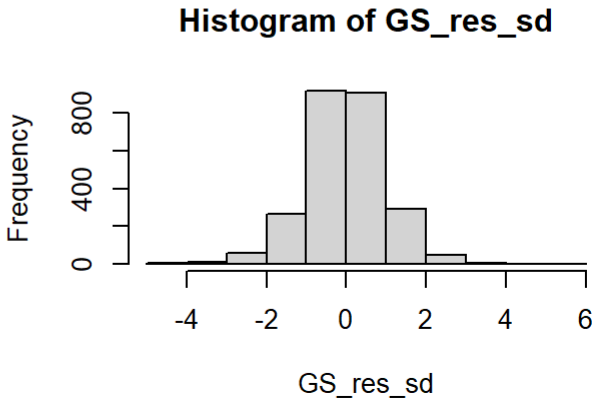



```
# Check Normal Distribution

qqnorm(GS_res)
qqline(GS_res)
hist(GS_res)

qqnorm(GS_res_sd)
qqline(GS_res_sd)
hist(GS_res_sd)
```










```
#Test for serial correlation

Box.test(x=GS_res,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  GS_res
## X-squared = 34.113, df = 5, p-value = 2.261e-06
```

```
Box.test(x=GS_res_sd,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  GS_res_sd
## X-squared = 9.3712, df = 5, p-value = 0.09514
```

```
adf.test(GS_res)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  GS_res
## Dickey-Fuller = -13.538, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(GS_res_sd)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  GS_res_sd
## Dickey-Fuller = -13.986, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

Risk Calculation

```
# Drawing a Random Sample
rho <- coef(Ct)

random_samples <- rCopula(10000,tCopula(rho[1], dim=2, df=rho[2]))

# Transform Data

random_samples <- random_samples %>%
  data.frame()

colnames(random_samples) <- c("SPXL.prob", "GS.prob")

random_samples$SPXL.e.std <- qstd(random_samples$SPXL.prob, mean = std.SPXL_res[1], sd = std.SPX
L_res[2],
                        nu = std.SPXL_res[3])
random_samples$GS.e.std <- qstd(random_samples$GS.prob, mean = std.GS_res[1], sd = std.GS_res[2
],
                      nu = std.GS_res[3])


# Fit a AR(1)-GARCH(1,1) model to the returns

ar1 <- function(mu, phi, x_prev) {
  x = mu*(1-phi) + phi * x_prev
  return(as.numeric(x))
}

garch1.1 <- function(omega, alpha, beta, e_prev, sigma_prev) {
  sigma_sq <- omega + alpha*(e_prev^2) + beta*(sigma_prev^2)
  sqrt(sigma_sq) %>% as.numeric() %>% return()
}

#SPXL

spxl.params <- coef(SPXLfit)


random_samples$spxl.next <-
  ar1(mu = spxl.params[1], phi = spxl.params[2], x_prev = spxl_log_ret$SPXL.Adjusted[nrow(spxl_l
og_ret)]) +
  garch1.1(omega = spxl.params[3], alpha = spxl.params[4], beta = spxl.params[5],
           e_prev = SPXL_res[nrow(spxl_log_ret)], sigma_prev = SPXL_res_sd[nrow(spxl_log_ret)])
 * random_samples$SPXL.e.std

#GS

gs.params <- coef(GSfit)

random_samples$gs.next <-
  ar1(mu = gs.params[1], phi = gs.params[2], x_prev = gs_log_ret$GS.Adjusted[nrow(gs_log_ret)])
 +
  garch1.1(omega = gs.params[3], alpha = gs.params[4], beta = gs.params[5],
```

```
          e_prev = GS_res[nrow(gs_log_ret)], sigma_prev = GS_res_sd[nrow(gs_log_ret)]) * random
_samples$GS.e.std

#Apply AR(1,0)-GARCH(1,1)

garch.SPXL = ugarchspec(mean.model=list(armaOrder=c(1,0)),
                        variance.model=list(garchOrder=c(1,1)),
                        distribution.model = "std")
SPXL.garch.fit = ugarchfit(data=random_samples$SPXL.e.std, spec=garch.SPXL)

garch.GS = ugarchspec(mean.model=list(armaOrder=c(1,0)),
                        variance.model=list(garchOrder=c(1,1)),
                        distribution.model = "std")
GS.garch.fit = ugarchfit(data=random_samples$GS.e.std, spec=garch.GS)

show(SPXL.garch.fit)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error    t value Pr(>|t|)
## mu      0.000361    0.000212 1.6984e+00 0.089426
## ar1     0.004795    0.006452 7.4319e-01 0.457365
## omega   0.000002    0.000000 8.1735e+02 0.000000
## alpha1  0.000053    0.000002 3.4242e+01 0.000000
## beta1   0.998699    0.000022 4.5074e+04 0.000000
## shape   2.557152    0.018248 1.4014e+02 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error    t value Pr(>|t|)
## mu      0.000361    0.000217     1.6624 0.096424
## ar1     0.004795    0.006413     0.7478 0.454579
## omega   0.000002    0.000000   774.0241 0.000000
## alpha1  0.000053    0.000004    14.4714 0.000000
## beta1   0.998699    0.000012 80024.8340 0.000000
## shape   2.557152    0.013500   189.4135 0.000000
##
## LogLikelihood : 22396.6
##
## Information Criteria
## ------------------------------------
##
## Akaike       -4.4781
## Bayes        -4.4738
## Shibata      -4.4781
## Hannan-Quinn -4.4767
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.4020  0.5261
## Lag[2*(p+q)+(p+q)-1][2]    0.4531  0.9744
## Lag[4*(p+q)+(p+q)-1][5]    0.7240  0.9794
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
```

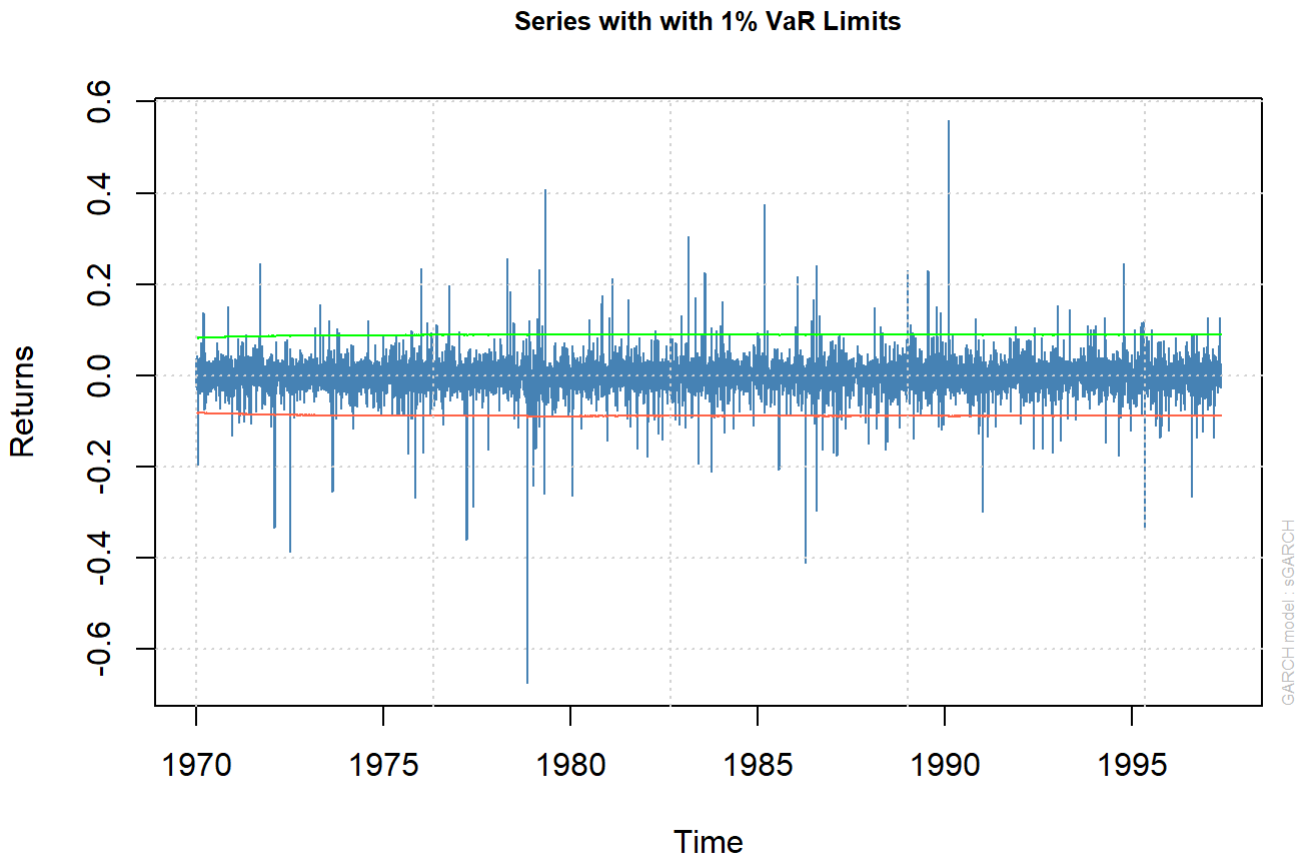
```
## Lag[1]                     0.1854  0.6668
## Lag[2*(p+q)+(p+q)-1][5]    0.3529  0.9778
## Lag[4*(p+q)+(p+q)-1][9]    0.7933  0.9933
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.2394 0.500 2.000  0.6246
## ARCH Lag[5]    0.2887 1.440 1.667  0.9433
## ARCH Lag[7]    0.6064 2.315 1.543  0.9678
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1305.916
## Individual Statistics:
## mu       0.25100
## ar1      0.12314
## omega  210.42806
## alpha1   0.14528
## beta1    0.08275
## shape    0.11578
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.49 1.68 2.12
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.8297 0.4067
## Negative Sign Bias  1.1336 0.2570
## Positive Sign Bias  1.2184 0.2231
## Joint Effect        2.8155 0.4210
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     18.78       0.4713
## 2    30     22.60       0.7943
## 3    40     33.10       0.7350
## 4    50     43.89       0.6799
##
##
## Elapsed time : 1.316154
```

```
summary(SPXL.garch.fit)
```

```
##    Length     Class      Mode
##         1 uGARCHfit        S4
```

```
plot(SPXL.garch.fit, which = 2, VaR.alpha=0.01)
```

```
##
## please wait...calculating quantiles...
```




```
show(GS.garch.fit)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error     t value Pr(>|t|)
## mu     -0.000061    0.000149 -4.0781e-01  0.68341
## ar1     0.000007    0.009446  7.7300e-04  0.99938
## omega   0.000000    0.000000  1.4208e+00  0.15538
## alpha1  0.000771    0.000082  9.3814e+00  0.00000
## beta1   0.997958    0.000023  4.2865e+04  0.00000
## shape   7.077278    0.451558  1.5673e+01  0.00000
##
## Robust Standard Errors:
##         Estimate  Std. Error     t value Pr(>|t|)
## mu     -0.000061    0.000154   -0.395219  0.69268
## ar1     0.000007    0.009431    0.000774  0.99938
## omega   0.000000    0.000005    0.063956  0.94900
## alpha1  0.000771    0.001555    0.495767  0.62006
## beta1   0.997958    0.000111 9025.613216  0.00000
## shape   7.077278    1.093649    6.471252  0.00000
##
## LogLikelihood : 27470.01
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.4928
## Bayes        -5.4885
## Shibata      -5.4928
## Hannan-Quinn -5.4913
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.06433  0.7998
## Lag[2*(p+q)+(p+q)-1][2]   0.22708  0.9980
## Lag[4*(p+q)+(p+q)-1][5]   0.86589  0.9649
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
```

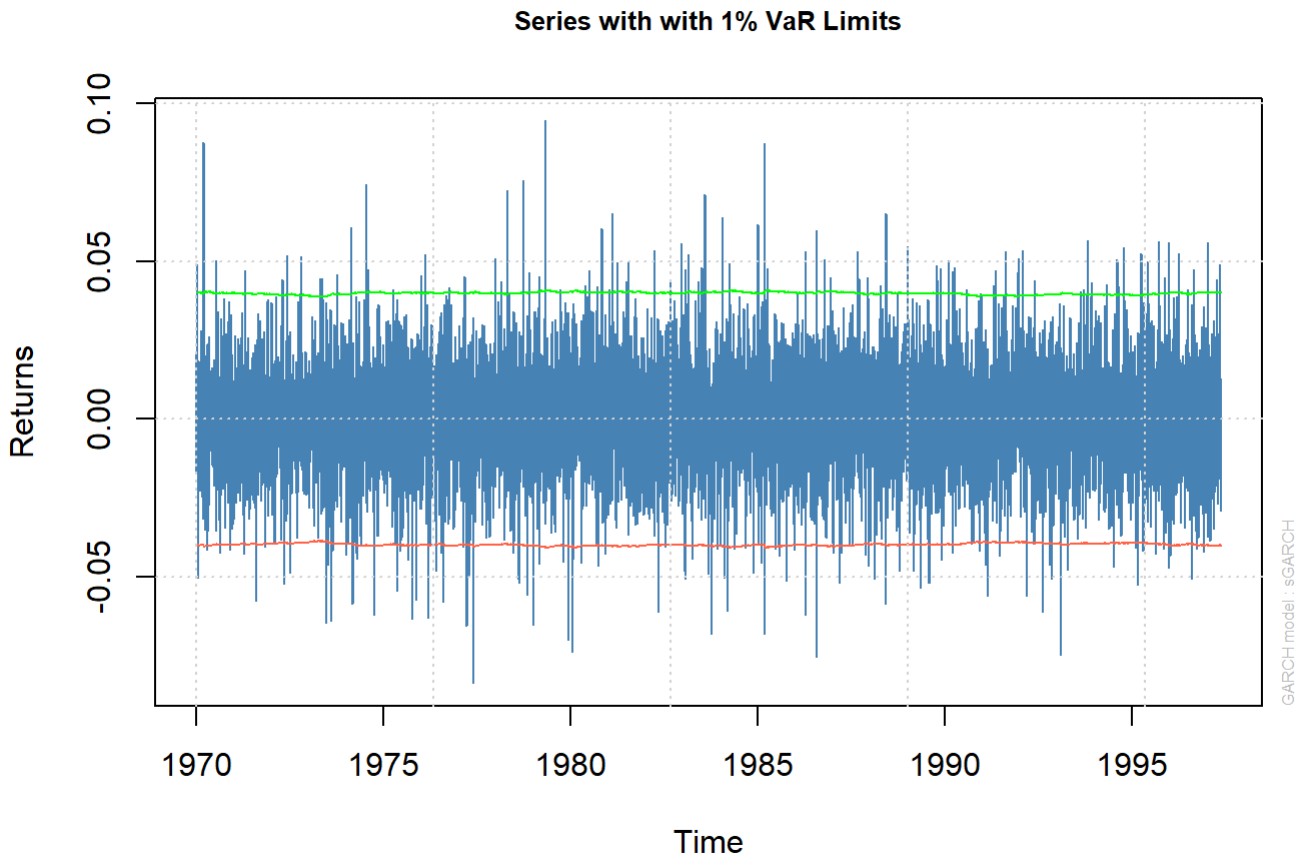
```
## Lag[1]                     0.2798  0.5969
## Lag[2*(p+q)+(p+q)-1][5]    1.4994  0.7401
## Lag[4*(p+q)+(p+q)-1][9]    2.2438  0.8740
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.2672 0.500 2.000  0.6052
## ARCH Lag[5]    0.2845 1.440 1.667  0.9444
## ARCH Lag[7]    0.7381 2.315 1.543  0.9521
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  3227.046
## Individual Statistics:
## mu       0.61243
## ar1      0.07886
## omega  711.98282
## alpha1   0.11296
## beta1    0.14401
## shape    0.06570
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.49 1.68 2.12
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.7562 0.4496
## Negative Sign Bias  0.2635 0.7922
## Positive Sign Bias  0.3273 0.7435
## Joint Effect        2.9174 0.4045
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     16.44       0.6275
## 2    30     26.90       0.5771
## 3    40     44.73       0.2438
## 4    50     47.23       0.5451
##
##
## Elapsed time : 1.770091
```

```
summary(GS.garch.fit)
```

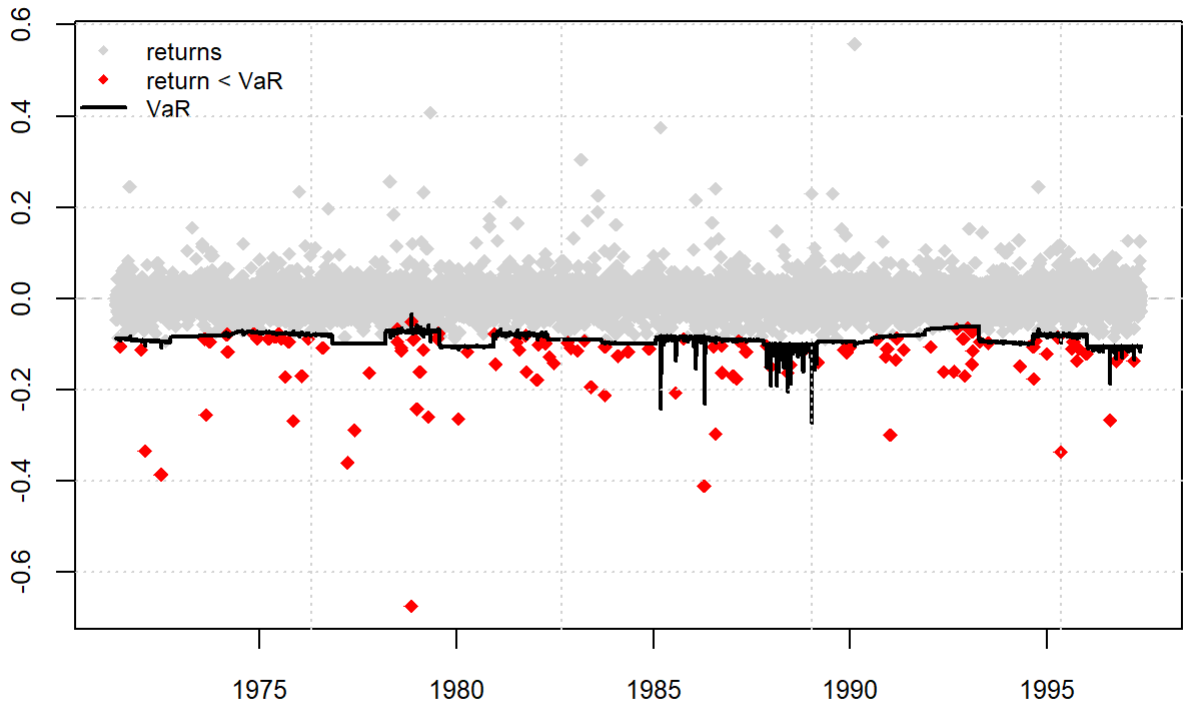
```
##    Length     Class      Mode
##         1 uGARCHfit        S4
```

```
plot(GS.garch.fit, which = 2, VaR.alpha=0.01)
```

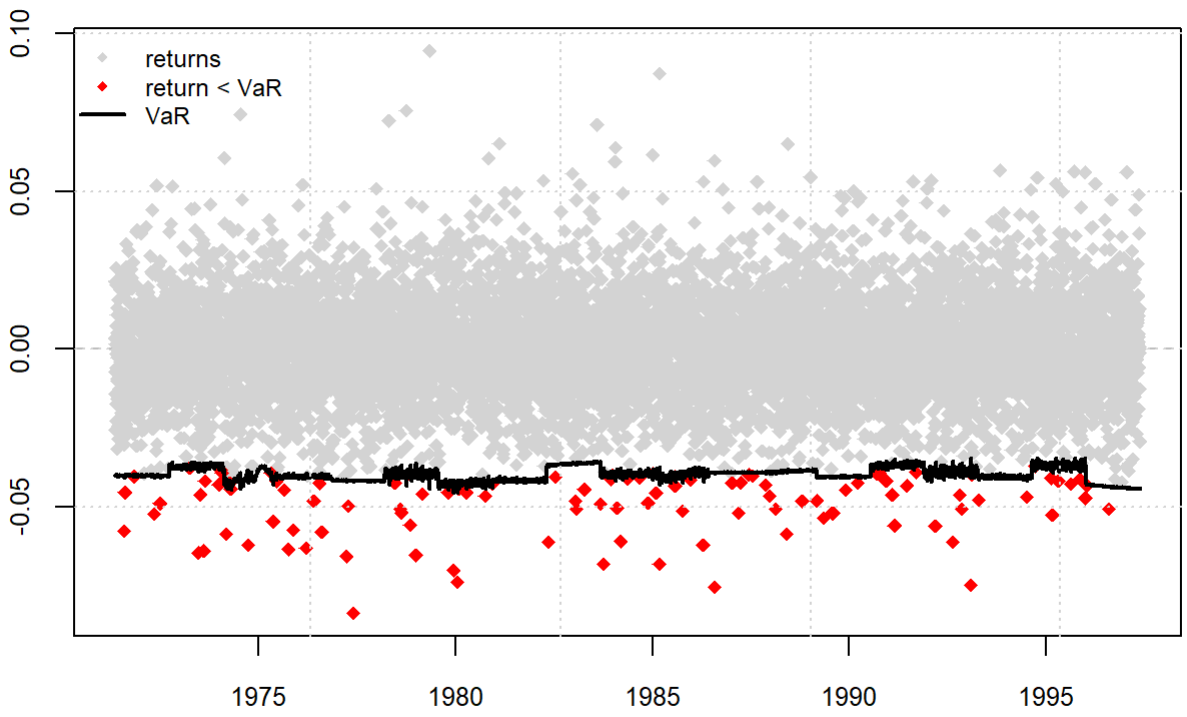
```
##
## please wait...calculating quantiles...
```




```
# Calculate one step forecasts

SPXL_cond_roll <- ugarchroll(garch.SPXL, random_samples$SPXL.e.std, n.start = 500, refit.every =
500,
                             refit.window = "moving", solver = "hybrid",
                             calculate.VaR = TRUE, VaR.alpha = 0.01, keep.coef = TRUE,
                             solver.control = list(tol = 1e-7, delta = 1e-9),
                             fit.control = list(scale = 1))
plot(SPXL_cond_roll, which = 4)
```




```
GS_cond_roll <- ugarchroll(garch.GS, random_samples$GS.e.std, n.start = 500, refit.every = 500,
                           refit.window = "moving", solver = "hybrid",
                           calculate.VaR = TRUE, VaR.alpha = 0.01, keep.coef = TRUE,
                           solver.control = list(tol = 1e-7, delta = 1e-9),
                           fit.control = list(scale = 1))
plot(GS_cond_roll, which = 4)
```




```
# Calculate mu, sigma, VaR, ES

SPXL_cond_pred = ugarchforecast(SPXL.garch.fit, data=random_samples$SPXL.e.std, n.ahead=1)
SPXL_mu.predict <- fitted(SPXL_cond_pred)
SPXL_sig.predict <- sigma(SPXL_cond_pred)
SPXL_VaR.predict <- as.numeric(SPXL_mu.predict + SPXL_sig.predict * qnorm(0.01))

GS_cond_pred = ugarchforecast(GS.garch.fit, data=random_samples$GS.e.std, n.ahead=1)
GS_mu.predict <- fitted(GS_cond_pred)
GS_sig.predict <- sigma(GS_cond_pred)
GS_res = fitdistr(random_samples$GS.e.std,"t")
GS_VaR.predict = GS_mu.predict + GS_sig.predict * qdist(distribution='std', shape=GS_res$estimat
e['df'], p=0.01)

print(paste("VaR Conditional Prediction: ", round(SPXL_VaR.predict + GS_VaR.predict,3)))
```

```
## [1] "VaR Conditional Prediction:  -0.125"
```

```
#Rho Calculation

mu.next <- dplyr::select(random_samples, spxl.next, gs.next) %>% as.matrix()
rho <- matrix(rep(NA,18), ncol = 2)
rho[,1] <- seq(.1,.9,.1)
rho[,2] <- 1-rho[,1]

portfolios <- mu.next %*% t(rho)
VaR <- qnorm(.1, mean = colMeans(portfolios), sd = apply(portfolios,2,sd)) * -1

VaRs <- cbind(rho[,1], VaR)
colnames(VaRs) <- c("rho","VaR")
VaRs[,2] <- VaRs[,2] %>% round(5)
VaRs %>% htmlTable
```

rho	VaR
0.1	0.01801
0.2	0.01748
0.3	0.01712
0.4	0.01693
0.5	0.01692
0.6	0.01707
0.7	0.01737
0.8	0.0178
0.9	0.01836