Pratical 1,2 and 3 for Group 1 (In full)

Lodrik Adam, Sophie Daya, Jeff Macaraeg, Julien Perini, Alexandra Boado

December 10, 2024

Table of contents

Introduction

This documents is the annex to our final report for group 1. It contains the full code for the practicals 1, 2 and 3 with the results and some discussions. This allows anyone to reproduce our results and to understand the code we used.

Note:

Certain sections of the code have been omitted from the generated PDF and HTML versions of this annex to enhance readability. Specifically, the code for data loading, package installation, and plot generation has been excluded. The full code is available in the QMD files located in the "code" folder of our GitHub repository.

Practical 1: Financial returns and normality

Part 1: Financial returns and normality

The working directory is set to: /Users/lodrik/Documents/GitHub/RA_Praticals

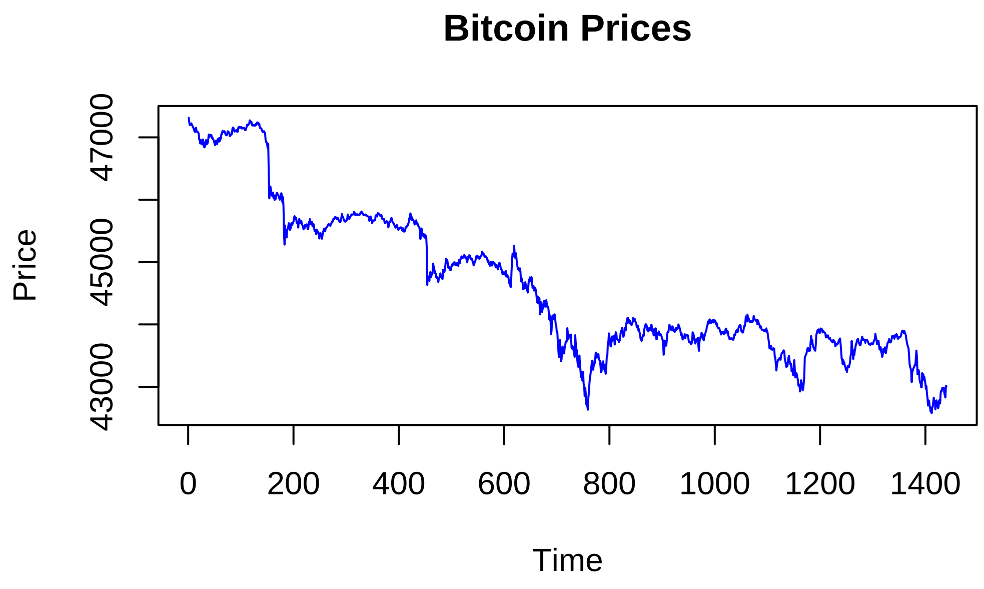
a) Load Bitcoin data and assess price stationarity

Question

Read in the Bitcoin data from file Crypto data.csv. Then, assess the stationarity of the (raw) Bitcoin prices.

First, let's take a look at the Bitcoin Prices on a plot.

```r
## Step 1: Extract the Bitcoin prices
bitcoin_prices <- crypto_data$Bitcoin

## Step 2: Plot the Bitcoin prices
plot(bitcoin_prices,
     type="l",
     col="blue",
     main="Bitcoin Prices",
     xlab="Time",
     ylab="Price")
```



The graph of the raw Bitcoin prices suggest that the series might not be stationary.

Let's perform the Augmented Dickey-Fuller test to check if the raw Bitcoin prices are stationary.

```r
## Step 3: test for stationarity
adf.test(crypto_data$Bitcoin)
```

```

	Augmented Dickey-Fuller Test

data:  crypto_data$Bitcoin
Dickey-Fuller = -2.4484, Lag order = 11, p-value = 0.3885
alternative hypothesis: stationary
```

Since the p-value is significantly bigger than 0.05, we can not reject the null hypothesis and therefore, we can conclude that the raw Bitcoin prices are non-stationary.

b) Create and plot Bitcoin negative log returns, assess stationarity

Question

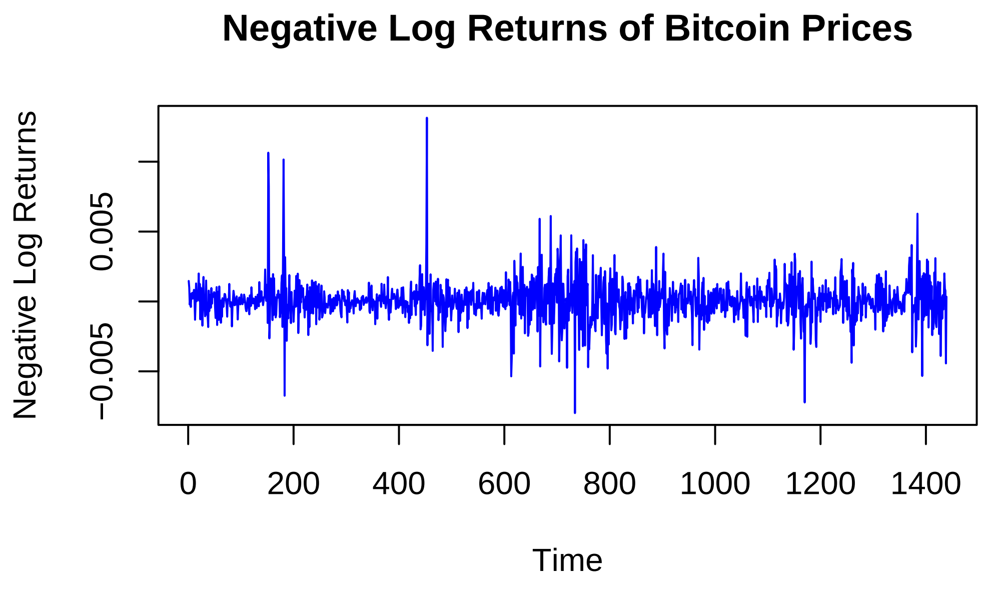
Create a function to transform the Bitcoin prices into their negative log returns counterparts. Plot the latter series and assess their stationarity. To compare the series, also plot the negative log returns on a common scale.

Let's create a function to compute the negative log returns of a given price series. We will then apply this function to the Bitcoin prices to compute the negative log returns.

```r
## Step 1: Create a function to compute negative log returns
negative_log_returns <- function(prices) {
  return(-diff(log(prices)))
}

## Step 2: Use the fucntion on Bitcoin prices
neg_log_returns_bitcoin <- negative_log_returns(bitcoin_prices)
```

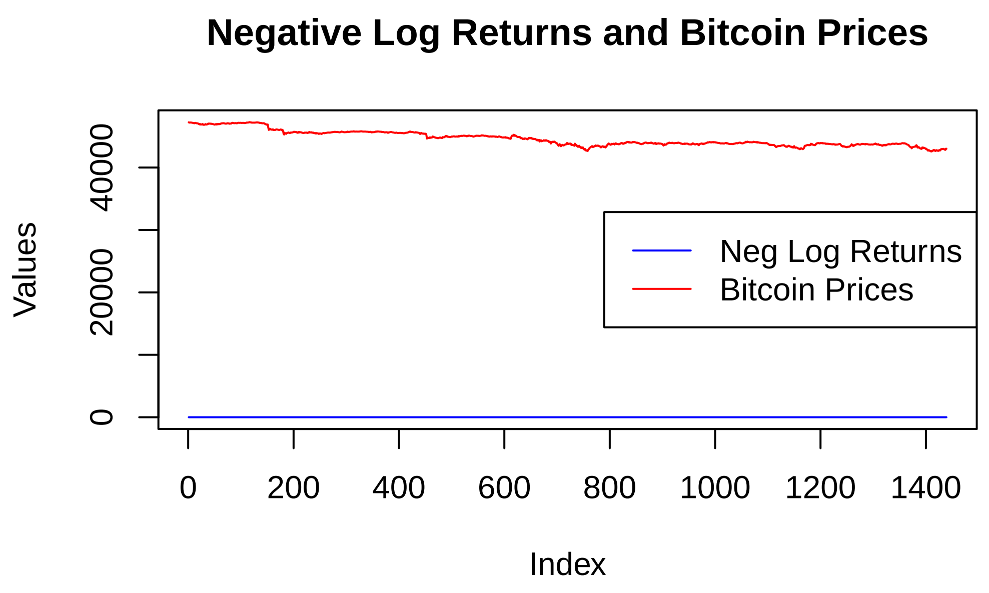
We can now plot the negative log returns series and the raw Bitcoin prices to compare.

```r
## Step 3: Plot the negative log returns series
plot(neg_log_returns_bitcoin,
     type="l",
     col="blue",
     main="Negative Log Returns of Bitcoin Prices",
     xlab="Time",
     ylab="Negative Log Returns")
```



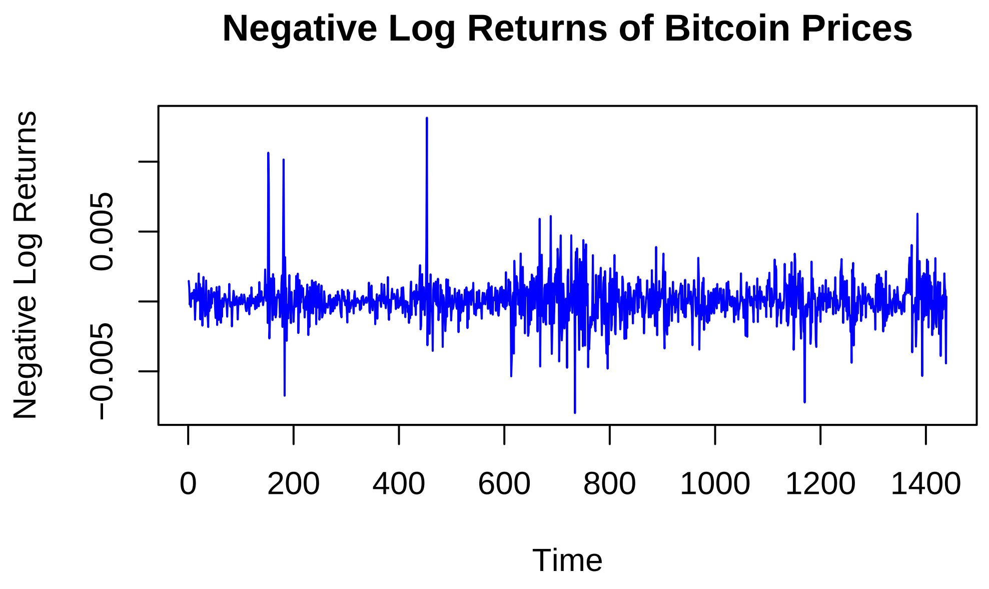
If we scale bitcoin prices and negative log returns, we can compare both time series on a plot with a common scale.

```r
# Top Plot: Plot both time series on the same graph
trimmed_bitcoin_prices <- bitcoin_prices[-1]  # Make sure lengths match
plot(neg_log_returns_bitcoin,
     type = "l",
     col = "blue",
     ylab = "Values",
     xlab = "Index",
     main = "Negative Log Returns and Bitcoin Prices",
     ylim = range(c(neg_log_returns_bitcoin, trimmed_bitcoin_prices)))

# Add Bitcoin prices on the same plot
lines(trimmed_bitcoin_prices, col = "red")

# Add a legend
legend("right",
       legend = c("Neg Log Returns", "Bitcoin Prices"),
       col = c("blue", "red"),
       lty = 1)
```

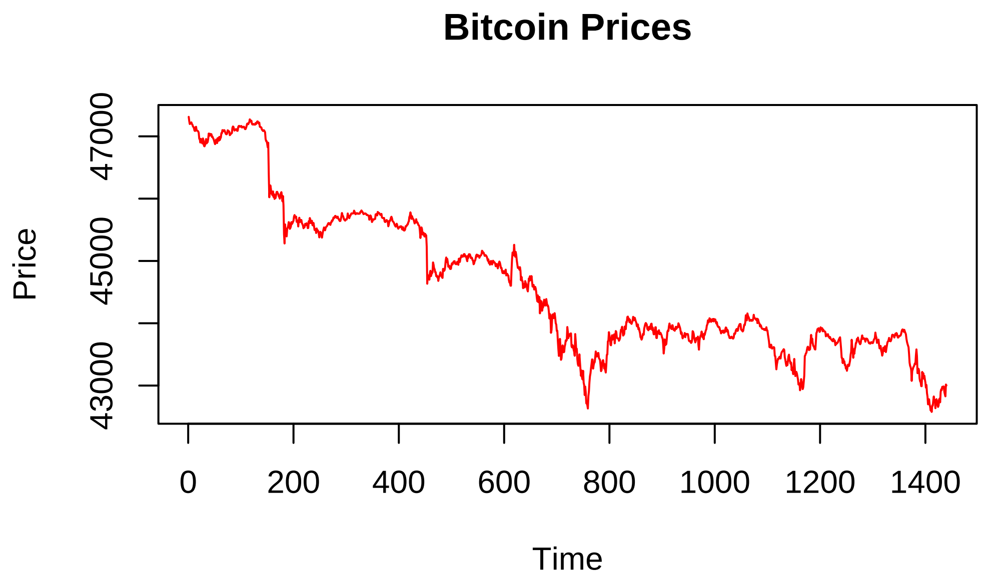


```
# Reset the plotting area to default settings (optional, for future plots)
par(mfrow=c(1,1))
```

```
# Set up the plotting area to have 3 rows and 1 column
par(mfrow=c(1,1))

# Bottom Left Plot: Plot the negative log returns series
plot(neg_log_returns_bitcoin,
     type = "l",
     col = "blue",
     main = "Negative Log Returns of Bitcoin Prices",
     xlab = "Time",
     ylab = "Negative Log Returns")
```



```r
# Bottom Right Plot: Plot the Bitcoin prices
plot(bitcoin_prices,
     type = "l",
     col = "red",
     main = "Bitcoin Prices",
     xlab = "Time",
     ylab = "Price")
```



Visually, the negative log returns series does not appear to indicate a clear trend or seasonality. The variance, although it fluctuates in the middle, seems relatively constant. This observation suggests that the series may be stationary. To confirm this, we will perform the Augmented Dickey-Fuller test to assess the stationarity of the negative log returns.

```r
## Step 5: Test the stationarity of the negative log returns with the Augmented Dickey-Fuller tes
adf.test(neg_log_returns_bitcoin)
```

```
	Augmented Dickey-Fuller Test

data:  neg_log_returns_bitcoin
Dickey-Fuller = -11.035, Lag order = 11, p-value = 0.01
alternative hypothesis: stationary
```

Since the p-value is significantly smaller than 0.05, we can reject the null hypothesis and conclude that the negative log returns series is stationary.

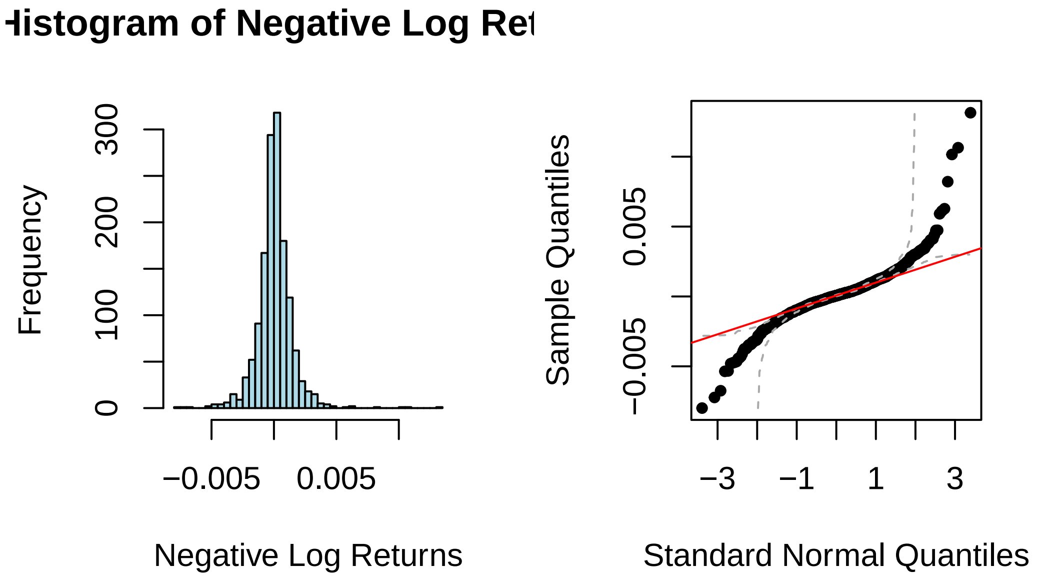
c) Check negative log returns normality with histograms, QQ-plots, Anderson-Darling

Question

Are the negative log returns normally distributed? Draw histograms, check QQ-plots and use an Anderson-Darling testing procedure to answer this question.

Let's first plot the histogram and QQ-plot of the negative log returns to visually assess the normality.

```r
## Step 1: Plot the histogram and QQ-plot of the negative log returns
par(mfrow=c(1, 2))

# Plot the histogram of the negative log returns
hist(neg_log_returns_bitcoin,
     breaks=50,
     col="lightblue",
     main="Histogram of Negative Log Returns",
     xlab="Negative Log Returns")

# Plot the QQ-plot of the negative log returns
qqnorm(neg_log_returns_bitcoin)
qqline(neg_log_returns_bitcoin,
       col="red")
```



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

The histogram of the negative log returns suggests that the data may follow a normal distribution. However, we need to perform a formal test to confirm this.

```r
## Step 2: Perform Anderson-Darling test for normality
ad.test(neg_log_returns_bitcoin)
```

```

	Anderson-Darling normality test

data:  neg_log_returns_bitcoin
A = 26.277, p-value < 2.2e-16
```

Even though the Histogram suggest that the negative log returns follows a normal distribution, the p-value when performing the Andersen-Darling test is smaller than 5%. It indicates that the data does not follow a normal distribution. The Normal Q-Q plot suggest also that the data does not follow a normal distribution.

d) Fit t-distribution, compare with Normal via QQ-plot analysis

Question

Fit a t-distribution to the negative log returns using `fitdistr()`. Using a QQ-plot, decide whether the fit is better than with a Normal distribution, based on your answer in (c).

Let's fit a t-distribution to the negative log returns and compare it with the normal distribution using a QQ-plot.

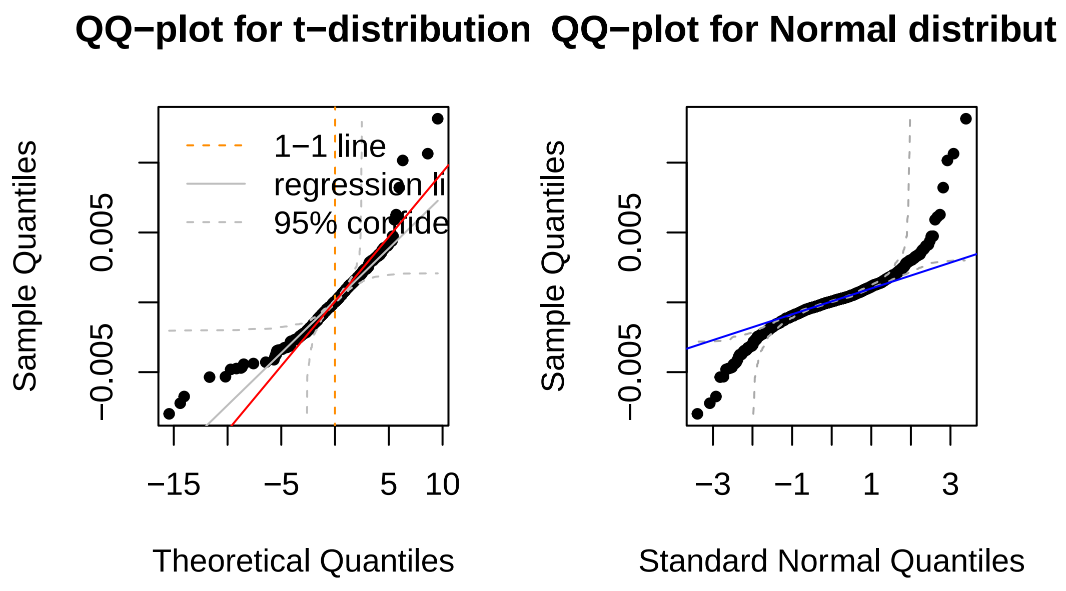
```r
## Step 1: Fit a t-distribution to the negative log returns
fit_t <- fitdistr(scaling_factor * neg_log_returns_bitcoin, "t") # Multiply by 100000 to avoid nu
```

The t-distribution fit parameters are:

- mean: 5.65
- standard deviation: 84.15
- degrees of freedom: 2.77

We can now compare the QQ-plot of the t-distribution with the QQ-plot of the normal distribution of question c).

```r
## Step 2: Create a QQ-plot for the t-distribution and the normal distribution
par(mfrow = c(1, 2))

# Generate QQ-plot for t-distribution
df_t <- fit_t$estimate[3]  # Degrees of freedom from the fit
qqplot(rt(length(neg_log_returns_bitcoin),
          df=df_t),
       neg_log_returns_bitcoin,
       main="QQ-plot for t-distribution",
       xlab="Theoretical Quantiles",
       ylab="Sample Quantiles")
qqline(neg_log_returns_bitcoin,
       col="red")

# Generate QQ-plot for normal distribution
qqnorm(neg_log_returns_bitcoin,
       main="QQ-plot for Normal distribution")
qqline(neg_log_returns_bitcoin,
       col="blue")
```



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

As we can see, the QQ-plot for the t-distribution is closer to the 45-degree line than the QQ-plot for the normal distribution. This suggests that the t-distribution is a better fit for the negative log returns than the normal distribution.

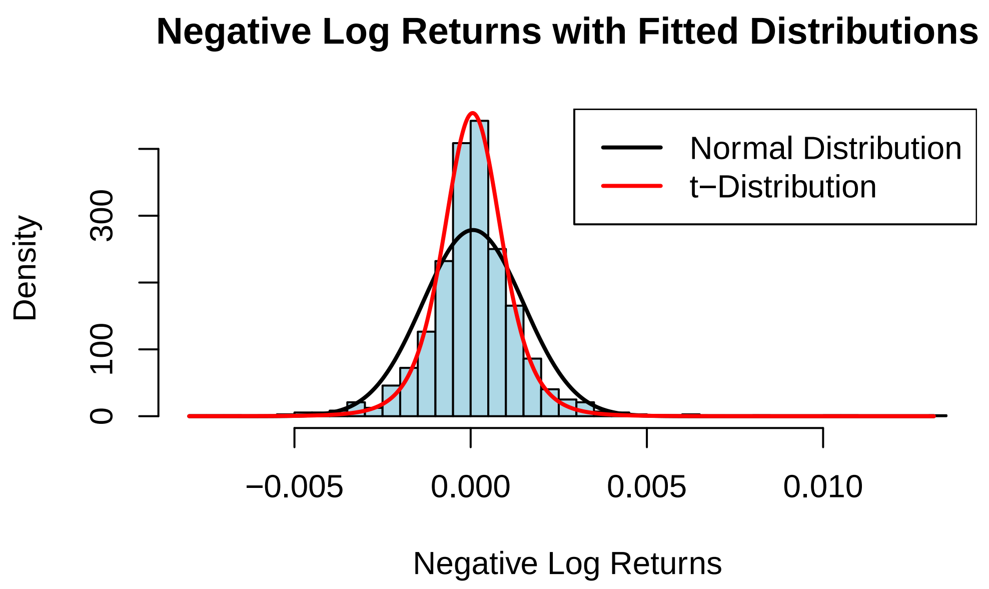
e) Compare t-distribution and normal tails

Question

Compare the tails of the density of the t-distribution and the normal distribution. Can we expect more extreme, unexpected events in t-distribution or in normal distribution? What can you conclude about the extreme events of our bitcoin data?

To compare the tails of the t-distribution and the normal distribution, we will plot the density functions of both distributions and visually assess the differences.

```r
## Step 1: Fit the normal distribution to the negative log returns
fit_norm <- fitdistr(scaling_factor * neg_log_returns_bitcoin, "normal")

# Generate a sequence of values for the x-axis (log returns)
x <- seq(min(neg_log_returns_bitcoin), max(neg_log_returns_bitcoin), length = 1000)

## Step 2: Scale back the mean and sd for plotting (for both normal and t-distributions)
# For normal distribution
scaled_mean_norm <- fit_norm$estimate[1] / scaling_factor
scaled_sd_norm <- fit_norm$estimate[2] / scaling_factor

# For t-distribution
scaled_mean_t <- fit_t$estimate[1] / scaling_factor
scaled_sd_t <- fit_t$estimate[2] / scaling_factor

# Density for the normal distribution using the scaled mean and sd
dens_norm <- dnorm(x, mean = scaled_mean_norm, sd = scaled_sd_norm)

# Density for the t-distribution using the scaled parameters
dens_t <- dt((x - scaled_mean_t) / scaled_sd_t, df = fit_t$estimate[1]) / scaled_sd_t

## Step 3 : Plot the histogram of negative log returns
hist(neg_log_returns_bitcoin,
     breaks = 50,
     col = "lightblue",
     freq = FALSE,  # For density plot
     main = "Negative Log Returns with Fitted Distributions",
     xlab = "Negative Log Returns")

# Add the normal distribution curve
lines(x,
      dens_norm,
      col = "black",
      lwd = 2,
      lty = 1)

# Add the t-distribution curve
lines(x,
      dens_t,
      col = "red",
       lwd = 2,
       lty = 1)

# Add a legend
legend("topright",
       legend = c("Normal Distribution", "t-Distribution"),
       col = c("black", "red"),
       lty = c(1, 1),
       lwd = 2)
```



Visually, the tails of the t-distribution are heavier than those of the normal distribution. This means that the t-distribution assigns more probability to extreme events than the normal distribution. Therefore, we can expect more extreme, unexpected events in the t-distribution than in the normal distribution. This observation is consistent with the QQ-plot analysis in question d), where the t-distribution was a better fit for the negative log returns than the normal distribution.

Part 2: Financial time series heteroskedasticity and the random walk hypothesis

Another crucial hypothesis in asset pricing is the so-called homoscedasticity, i.e. constant variance of theresiduals. We would also like to check this assumption. We use the same Bitcoin data as in Part 1.

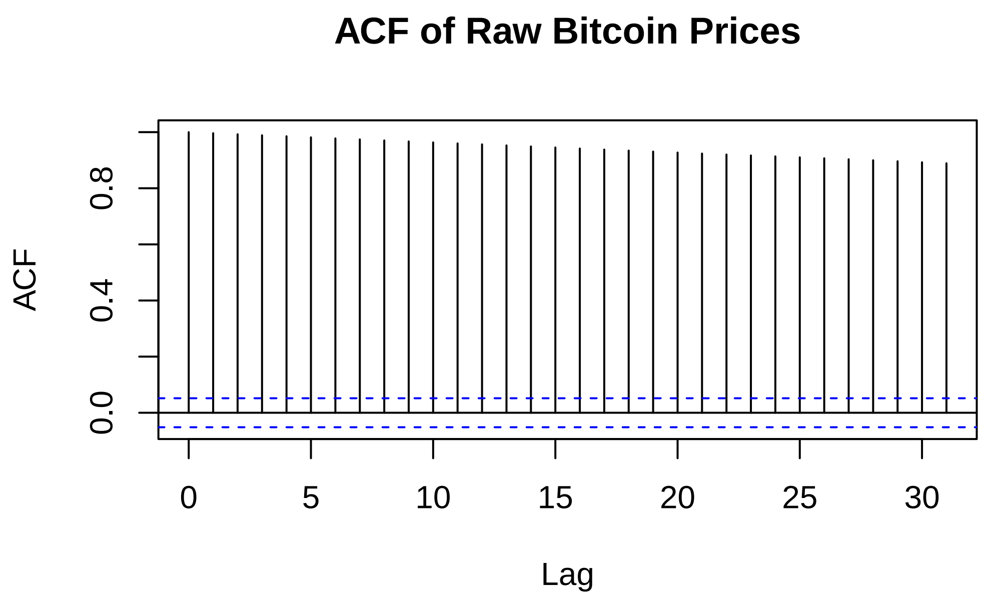
```r
# load the required packages and install them if they are not.
source(here::here("code","setup.R"))

# getiing the working directory
wd <- here::here()

# Loading the data
crypto_data <- read.csv(here("data", "crypto_data.csv"))

# Extract the Bitcoin prices
bitcoin_prices <- crypto_data$Bitcoin

# Create a function to compute negative log returns
negative_log_returns <- function(prices) {
  return(-diff(log(prices)))
}

# Use the fucntion on Bitcoin prices
neg_log_returns_bitcoin <- negative_log_returns(bitcoin_prices)
```

a) ACF & negative log returns

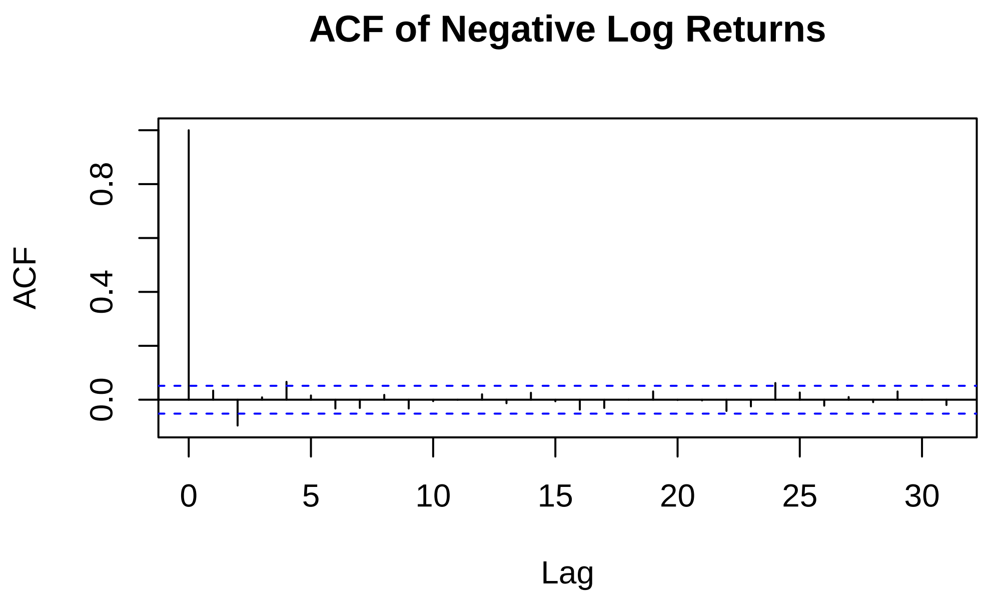
Question

Plot the ACF of the raw series as well as the negative log returns. Which one do you think are easier to model?

```r
# Tracer l'ACF de la série brute des prix du Bitcoin
acf(bitcoin_prices, main = "ACF of Raw Bitcoin Prices")
```



```r
# Tracer l'ACF des rendements logarithmiques négatifs (bitcoin_log_returns)
acf(neg_log_returns_bitcoin, main = "ACF of Negative Log Returns")
```



1. ACF or Raw Bitcoin Prices: The ACF plot for the Raw Bictoin Prices shows strong autocorrelation. The values are strongly correlated with their past values. This indicates that the raw series is non-stationary and has a long-term dependency.
2. ACF of Negative Log Returns: The ACF of Negative Log Returns shows that most of the correlations at higher lags fall withing the confidence interval. It implies that the negative log returns are more likely to be stationary and have less long-term dependence.

Conclusion: The Negative Log Returns are likely easier to model due to their more stationary nature and lack of significant autocorrelation.

b) Ljung-Box procedure

Question

Use a Ljung-Box procedure to formally test for (temporal) serial dependence in the raw series and in the negative log return series. What is your conclusion?

```r
# Apply Ljung-Box test on raw Bitcoin prices
ljung_box_raw <- Box.test(bitcoin_prices, lag = 20, type = "Ljung-Box")

# Apply Ljung-Box test on negative log returns
ljung_box_returns <- Box.test(neg_log_returns_bitcoin, lag = 20, type = "Ljung-Box")

# Print results
print(ljung_box_raw)
```

```

    Box-Ljung test

data:  bitcoin_prices
X-squared = 26873, df = 20, p-value < 2.2e-16
```

```r
print(ljung_box_returns)
```

```

    Box-Ljung test

data:  neg_log_returns_bitcoin
X-squared = 33.356, df = 20, p-value = 0.03082
```

The Ljung-Box test checks for serial dependence (autocorrelation) in the series. If the p-value is small (typically < 0.05), it suggests that there is serial dependence, meaning the series is not independent over time.

For the raw series: Since price data tends to show trends, we often expect serial dependence. For the negative log returns: These are typically expected to be more random (i.e., closer to white noise), so the test might indicate less serial dependence.

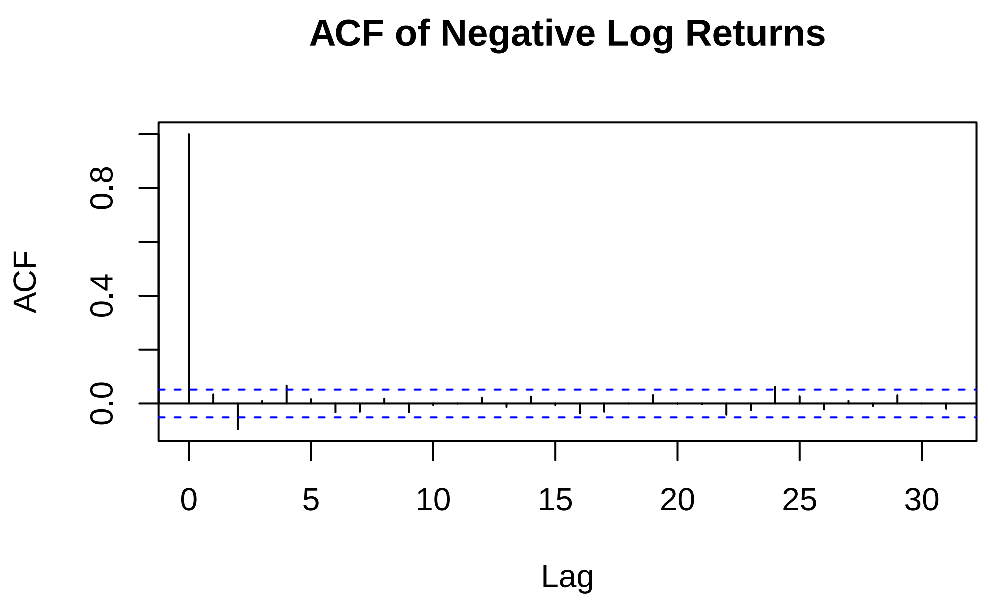
Based on the results of the Ljung-Box tests: For the raw Bitcoin prices: - Raw Bictoin Prices: p-value $< 2.2\text{e-}16$, the p-value is extremely small, which mean that we reject the null hypothesis of no autocorrelation in the raw Bictoin prices. The values are highly dependent on previous values, it confirms that the series is non-stationary. - Negative Log Returns: p-value $= 0.03082$, the p-value is also small, but higher than the raw prices. It indicates that there is still some autocorrelation in the series, although it is less pronounced compared to the raw Bitcoin prices. Ideally, negative log returns should behave more likke white noise, meaning no serial dependence

c) ARIMA models for the negative log returns series

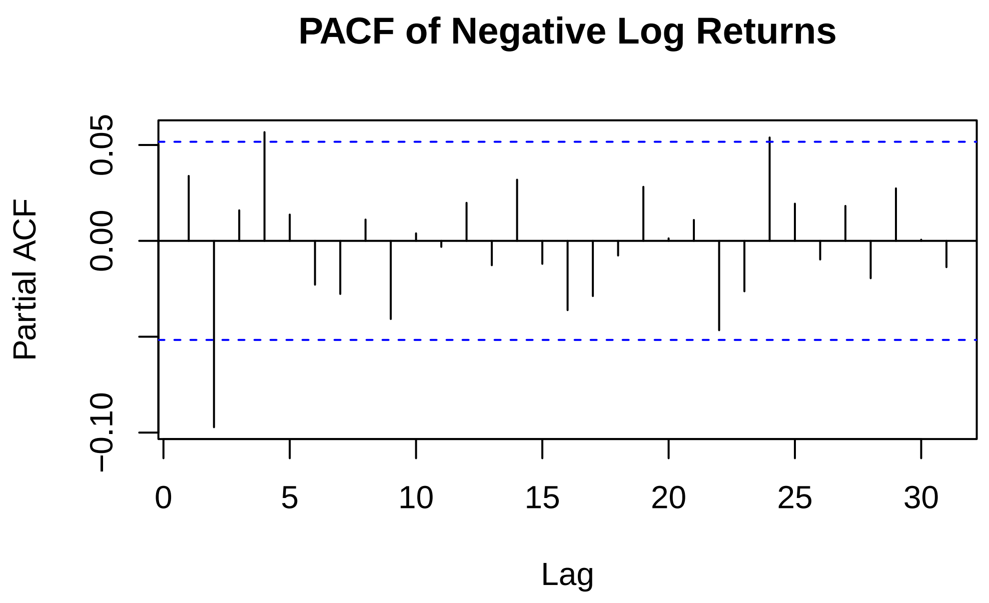
Question

Propose ARIMA models for the negative log returns series, based on visualization tools (e.g. ACF, PACF). Select an ARIMA model using auto.arima() (forecast package) for the negative log returns series. Comment on the difference. Assess the residuals of the resulting models.

```r
# Step 1: Visualize ACF and PACF for negative log returns
acf(neg_log_returns_bitcoin, main = "ACF of Negative Log Returns")
```



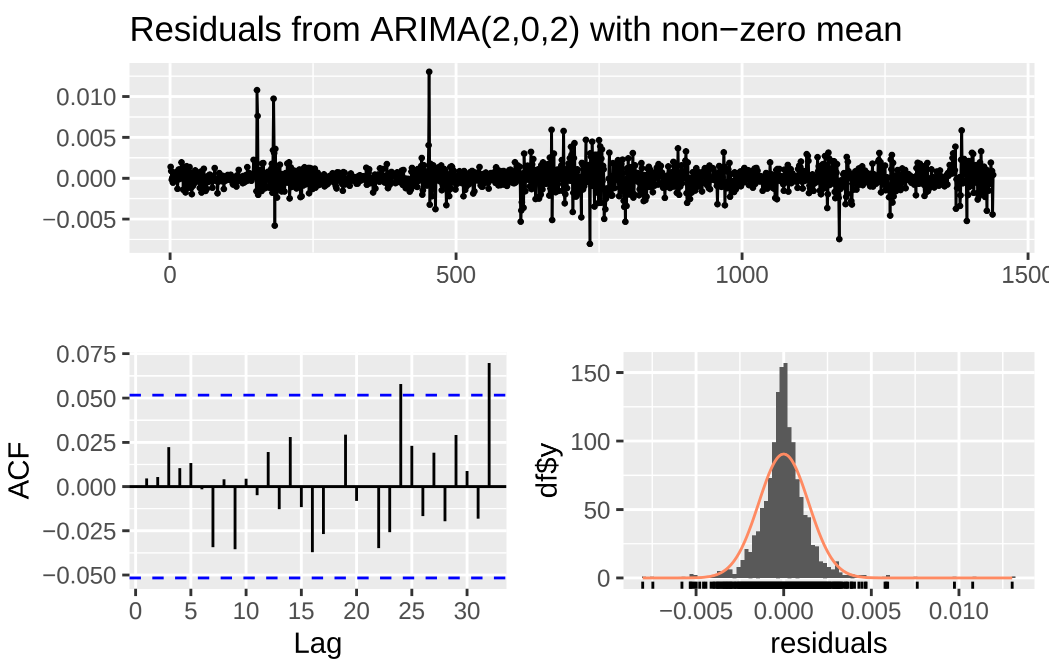
```r
pacf(neg_log_returns_bitcoin, main = "PACF of Negative Log Returns")
```



```
# Step 2: Use auto.arima() to find the best ARIMA model for negative log returns
auto_arima_model <- auto.arima(neg_log_returns_bitcoin)
summary(auto_arima_model)
```

```
Series: neg_log_returns_bitcoin
ARIMA(2,0,2) with non-zero mean

Coefficients:
          ar1      ar2     ma1     ma2   mean
      -0.0520  -0.5415  0.0853  0.4479  1e-04
s.e.   0.1717   0.1664  0.1824  0.1773  0e+00

sigma^2 = 2.029e-06:  log likelihood = 7391.82
AIC=-14771.65   AICc=-14771.59   BIC=-14740.02

Training set error measures:
                        ME        RMSE          MAE     MPE     MAPE      MASE
Training set -1.965777e-07 0.001421946 0.0009423239 100.013 131.3896 0.7133069
                   ACF1
Training set 0.00455059
```

```
# Step 3: Plot residuals of the ARIMA model to assess the goodness of fit
checkresiduals(auto_arima_model)
```



```
    Ljung-Box test

data:  Residuals from ARIMA(2,0,2) with non-zero mean
Q* = 4.7774, df = 6, p-value = 0.5727

Model df: 4.   Total lags used: 10
```

```r
# Additional: Ljung-Box test on residuals to check if they are white noise
Box.test(residuals(auto_arima_model), type="Ljung-Box")
```

```
    Box-Ljung test

data:  residuals(auto_arima_model)
X-squared = 0.029861, df = 1, p-value = 0.8628
```

The results from the ARIMA model fitting for the negative log returns of Bitcoin and the residual analysis suggest the following:

ARIMA Model: The selected ARIMA model is ARIMA(2,0,2), meaning:

AR(2): Two autoregressive terms are included. MA(2): Two moving average terms are included. d = 0: No differencing was applied, indicating that the series is already stationary (which aligns with the fact that negative log returns tend to be stationary). Coefficients:

The AR1 and AR2 coefficients are -0.0520 and -0.5415, respectively. The MA1 and MA2 coefficients are 0.0853 and 0.4479, respectively. The mean of the series is very close to zero (1e-04). Error metrics:

RMSE (Root Mean Square Error): 0.00142, which is relatively low, indicating that the model fits the data well. MAE (Mean Absolute Error): 0.000942, which is also quite low. ACF1 of residuals: 0.00455, suggesting that the residuals do not exhibit significant autocorrelation. Ljung-Box Test: The Ljung-Box test on residuals gives a p-value of 0.8628, which is much larger than 0.05. This indicates that there is no significant autocorrelation left in the residuals, implying that the model fits the data well. Conclusion: The ARIMA(2,0,2) model selected by auto.arima() seems to be a good fit for the negative log returns of Bitcoin, as evidenced by the low RMSE and MAE, as well as the results of the Ljung-Box test. The residuals behave like white noise, meaning that the model has successfully captured the patterns in the data. There is no significant temporal dependence left in the residuals, which supports the adequacy of this ARIMA model for the series. Overall, the ARIMA model chosen by auto.arima() fits the data well and leaves no significant autocorrelation in the residuals.

d) GARCH models

Question

Fit GARCH models to the negative log returns with both normal and standardized t-distributions, with order (1, 1), using the garchFit() function from the fGarch library. Assess the quality of the fit by evaluating the residuals.

```r
# Fit GARCH(1,1) model with normal distribution
garch_normal <- garchFit(~ garch(1, 1), data = neg_log_returns_bitcoin, cond.dist = "norm")

# Summary of the model
summary(garch_normal)

# Fit GARCH(1,1) model with standardized t-distribution
garch_t <- garchFit(~ garch(1, 1), data = neg_log_returns_bitcoin, cond.dist = "std")

# Summary of the model
summary(garch_t)

# Residuals from the normal GARCH model
residuals_normal <- residuals(garch_normal)

# Residuals from the t-distribution GARCH model
residuals_t <- residuals(garch_t)

# Plot residuals for the normal GARCH model
plot(residuals_normal, main = "Residuals of GARCH(1,1) with Normal Distribution", type = "l")
```

```r
# Plot residuals for the t-distribution GARCH model
plot(residuals_t, main = "Residuals of GARCH(1,1) with t-Distribution", type = "l")
```

```r
# ACF of residuals for the normal GARCH model
acf(residuals_normal, main = "ACF of Residuals (Normal GARCH Model)")
```

```r
# ACF of residuals for the t-distribution GARCH model
acf(residuals_t, main = "ACF of Residuals (t-Distribution GARCH Model)")
```

```r
# Ljung-Box test for normal GARCH model residuals
Box.test(residuals_normal, lag = 20, type = "Ljung-Box")

# Ljung-Box test for t-distribution GARCH model residuals
```

```r
Box.test(residuals_t, lag = 20, type = "Ljung-Box")
```

The results for fitting GARCH(1,1) models with both normal and standardized t-distributions to the negative log returns are as follows:

GARCH(1,1) with Normal Distribution:

The log-likelihood value is 7632.108. The coefficients for the GARCH model (omega, alpha1, and beta1) are significant (p-values < 0.05), indicating that the model is well-fitted. The Ljung-Box test for the residuals shows a p-value of 0.3419 for 10 lags, which indicates no significant autocorrelation in the residuals, meaning the model fits well in terms of residual serial dependence. GARCH(1,1) with Standardized t-Distribution:

The log-likelihood value is 7736.355, which is slightly better than the normal distribution model, indicating a potentially better fit. The coefficients are also significant (p-values < 0.05), with the shape parameter of the t-distribution (shape $= 4.28$) indicating a heavier tail than the normal distribution. The Ljung-Box test for the residuals shows a p-value of 0.3507 for 10 lags, similar to the normal model, suggesting that there is no significant autocorrelation in the residuals. Conclusion: Both the GARCH(1,1) models (with normal and t-distributions) provide a good fit, with no significant residual autocorrelation based on the Ljung-Box test. However, the GARCH model with the standardized t-distribution has a higher log-likelihood and captures heavier tails (as indicated by the shape parameter), suggesting that it may be a better fit for the data due to the presence of tail risk or more extreme variations in the negative log returns of Bitcoin.

e) Residual serial correlation

Question

Residual serial correlation can be present when fitting a GARCH directly on the negative log returns. Hence, in order to circumvent this problem, it is possible to use the following two-step approach:
- Fit an ARIMA(p, d, q) on the negative log returns with the choices p, d and q from part (c);
- Fit a GARCH(1, 1) on the residuals of the ARIMA(p, d, q) fit.

Proceed with the above recipe. Assess the quality of the above fit.

```r
# Step 1: Fit an ARIMA(p, d, q) model on the negative log returns
# From part (c), we decided ARIMA(2, 0, 2)
arima_model <- arima(neg_log_returns_bitcoin, order = c(2, 0, 2))

# Extract the residuals from the ARIMA model
arima_residuals <- residuals(arima_model)

# Step 2: Fit a GARCH(1,1) model on the residuals from the ARIMA model
garch_model_arima_residuals <- garchFit(~ garch(1, 1), data = arima_residuals, cond.dist = "norm"
```

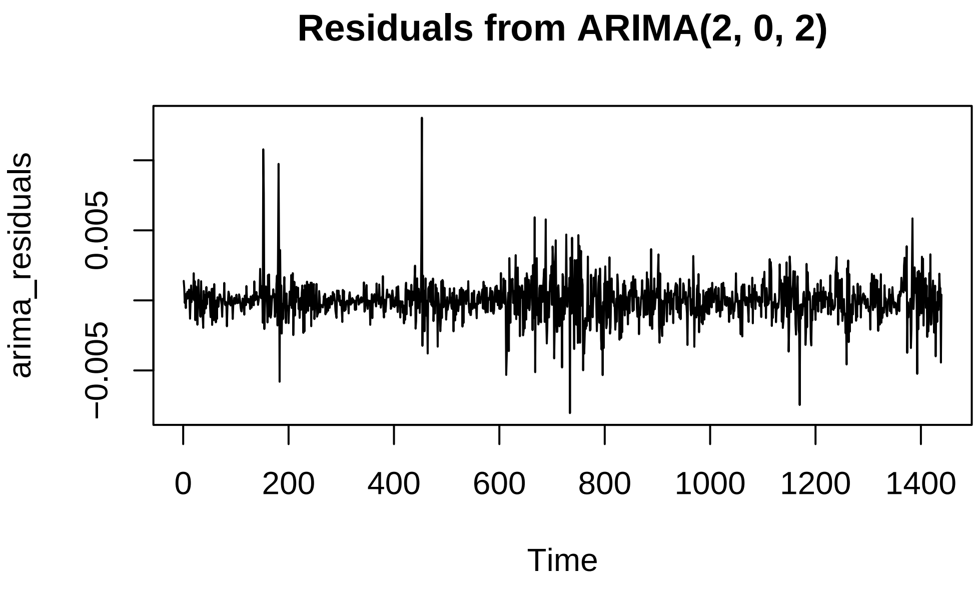
```
Series Initialization:
 ARMA Model:                arma
 Formula Mean:              ~ arma(0, 0)
 GARCH Model:               garch
 Formula Variance:          ~ garch(1, 1)
 ARMA Order:                0 0
 Max ARMA Order:            0
 GARCH Order:               1 1
 Max GARCH Order:           1
 Maximum Order:              1
 Conditional Dist:           norm
 h.start:                    2
 llh.start:                  1
 Length of Series:           1439
 Recursion Init:             mci
 Series Scale:               0.001422441

Parameter Initialization:
 Initial Parameters:          $params
 Limits of Transformations:   $U, $V
 Which Parameters are Fixed?  $includes
 Parameter Matrix:
                     U            V        params includes
    mu     -0.001381975 1.381975e-03 -0.0001381975     TRUE
    omega   0.000001000 1.000000e+02  0.1000000000     TRUE
    alpha1  0.000000010 1.000000e+00  0.1000000000     TRUE
    gamma1 -0.999999990 1.000000e+00  0.1000000000    FALSE
    beta1   0.000000010 1.000000e+00  0.8000000000     TRUE
    delta   0.000000000 2.000000e+00  2.0000000000    FALSE
    skew    0.100000000 1.000000e+01  1.0000000000    FALSE
    shape   1.000000000 1.000000e+01  4.0000000000    FALSE
 Index List of Parameters to be Optimized:
    mu  omega alpha1  beta1
     1      2      3      5
 Persistence:                  0.9


--- START OF TRACE ---
Selected Algorithm: nlminb

R coded nlminb Solver:

  0:     1877.5170: -0.000138197 0.100000 0.100000 0.800000
  1:     1855.7714: -0.000138198 0.0726602 0.109431 0.786204
  2:     1839.1695: -0.000138198 0.0680171 0.139755 0.795457
  3:     1829.7062: -0.000138198 0.0532235 0.143295 0.790116
  4:     1810.0200: -0.000138198 0.0267149 0.199381 0.807723
  5:     1809.1646: -0.000138198 0.0219331 0.198987 0.805030
  6:     1808.7267: -0.000138198 0.0249406 0.202100 0.801633
  7:     1808.3177: -0.000138199 0.0257164 0.207720 0.792203
  8:     1807.4361: -0.000138210 0.0315621 0.224320 0.778984
  9:     1806.9719: -0.000138218 0.0251042 0.230515 0.781780
 10:     1806.9152: -0.000138230 0.0268312 0.234701 0.782044
 11:     1806.6955: -0.000138240 0.0271249 0.235641 0.777616
 12:     1806.4274: -0.000138242 0.0278457 0.249636 0.766085
 13:     1806.3620: -0.000138242 0.0290636 0.249980 0.766517
 14:     1806.3497: -0.000138251 0.0292906 0.250326 0.765248
 15:     1806.3450: -0.000138323 0.0305057 0.251714 0.763384
 16:     1806.3146: -0.000138526 0.0299426 0.253707 0.762550
 17:     1806.3008: -0.000139110 0.0300663 0.256681 0.761179
 18:     1806.2912: -0.000139772 0.0303903 0.258077 0.759281
 19:     1806.2891: -0.000140480 0.0304056 0.259412 0.758500
 20:     1806.2883: -0.000141953 0.0305967 0.259976 0.757903
 21:     1806.2879: -0.000145229 0.0306869 0.260363 0.757537
 22:     1806.2873: -0.000152139 0.0307685 0.260698 0.757217
 23:     1806.2854: -0.000177344 0.0309333 0.261376 0.756569
 24:     1806.2808: -0.000251181 0.0312215 0.262556 0.755442
 25:     1806.2700: -0.000436287 0.0316540 0.264316 0.753760
 26:     1806.2435: -0.000905201 0.0323112 0.266960 0.751230
 27:     1806.1898: -0.00138197 0.0322376 0.266573 0.751590
 28:     1806.1562: -0.00138197 0.0309672 0.261581 0.756386
 29:     1806.1541: -0.00138197 0.0305619 0.259903 0.757984
 30:     1806.1541: -0.00138197 0.0305834 0.259961 0.757926
 31:     1806.1541: -0.00138197 0.0305829 0.259962 0.757926

Final Estimate of the Negative LLH:
 LLH:  -7627.039    norm LLH:  -5.300236
           mu         omega        alpha1         beta1
-1.965777e-06  6.187958e-08  2.599616e-01  7.579261e-01

R-optimhess Difference Approximated Hessian Matrix:
                  mu         omega        alpha1         beta1
mu     -1.554923e+09  2.507936e+11  9.975053e+04 -2.026553e+04
omega   2.507936e+11 -1.709771e+16 -6.864037e+09 -1.486634e+10
alpha1  9.975053e+04 -6.864037e+09 -6.383094e+03 -9.576435e+03
beta1  -2.026553e+04 -1.486634e+10 -9.576435e+03 -1.823892e+04
attr(,"time")
Time difference of 0.007724047 secs

--- END OF TRACE ---


Time to Estimate Parameters:
 Time difference of 0.02834296 secs
```

```
# Summary of the GARCH(1,1) model
summary(garch_model_arima_residuals)
```

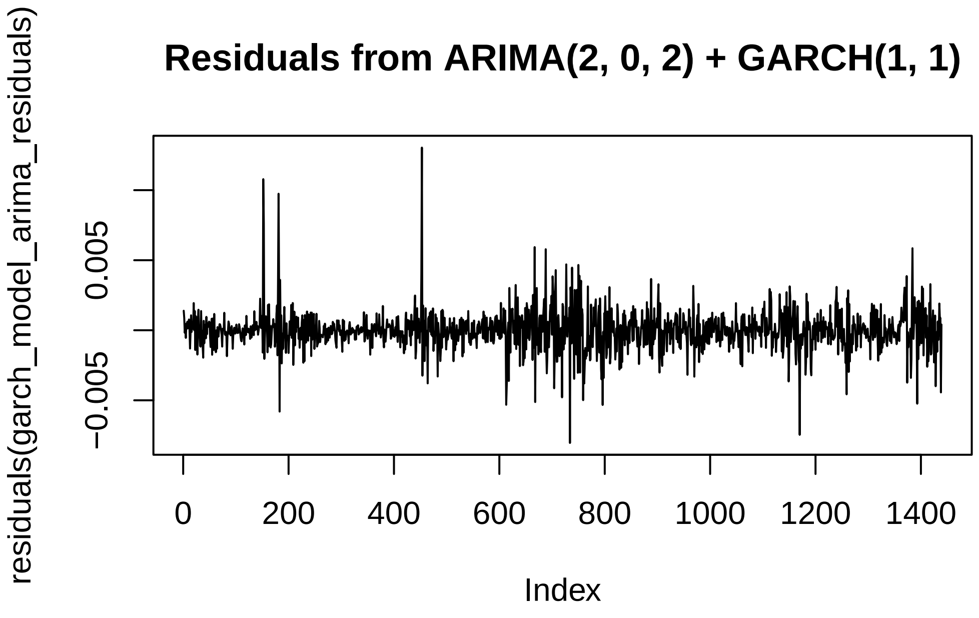
```
Title:
 GARCH Modelling

Call:
 garchFit(formula = ~garch(1, 1), data = arima_residuals, cond.dist = "norm")

Mean and Variance Equation:
 data ~ garch(1, 1)
<environment: 0x126dd1388>
 [data = arima_residuals]

Conditional Distribution:
 norm
```

```
Coefficient(s):
        mu        omega       alpha1        beta1
-1.9658e-06   6.1880e-08   2.5996e-01   7.5793e-01

Std. Errors:
 based on Hessian

Error Analysis:
        Estimate  Std. Error  t value Pr(>|t|)
mu     -1.966e-06   2.566e-05   -0.077    0.939
omega   6.188e-08   1.534e-08    4.034 5.49e-05 ***
alpha1  2.600e-01   2.936e-02    8.854  < 2e-16 ***
beta1   7.579e-01   2.434e-02   31.134  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log Likelihood:
 7627.039    normalized:  5.300236

Description:
 Tue Dec 10 18:44:45 2024 by user:


Standardised Residuals Tests:
                                  Statistic   p-Value
 Jarque-Bera Test   R    Chi^2  2537.823714 0.0000000
 Shapiro-Wilk Test  R    W         0.945703 0.0000000
 Ljung-Box Test     R    Q(10)    10.890175 0.3661383
 Ljung-Box Test     R    Q(15)    11.971102 0.6812151
 Ljung-Box Test     R    Q(20)    13.626059 0.8489384
 Ljung-Box Test     R^2  Q(10)    12.237348 0.2694857
 Ljung-Box Test     R^2  Q(15)    13.290578 0.5798648
 Ljung-Box Test     R^2  Q(20)    14.030759 0.8289330
 LM Arch Test       R    TR^2     12.413093 0.4130994

Information Criterion Statistics:
      AIC       BIC       SIC      HQIC
-10.59491 -10.58026 -10.59493 -10.58944
```

```r
# Plot the residuals from the ARIMA + GARCH model
plot(arima_residuals, main = "Residuals from ARIMA(2, 0, 2)", type = "l")
```



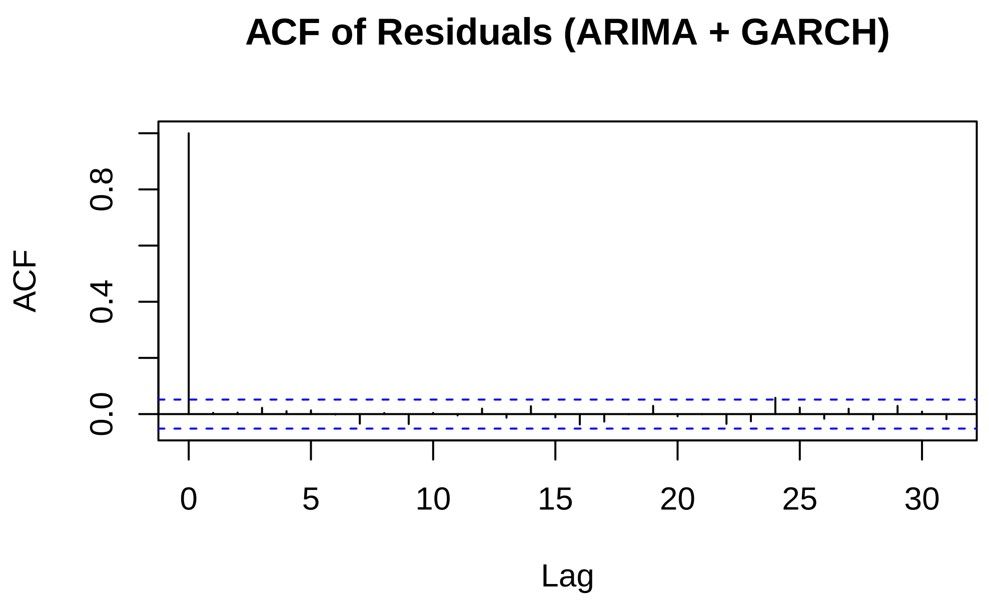
```
plot(residuals(garch_model_arima_residuals), main = "Residuals from ARIMA(2, 0, 2) + GARCH(1, 1)"
```



```
# Assess the quality of the fit using the Ljung-Box test on the GARCH model residuals
ljung_box_test <- Box.test(residuals(garch_model_arima_residuals), lag = 20, type = "Ljung-Box")
print(ljung_box_test)
```

```

	Box-Ljung test

data:  residuals(garch_model_arima_residuals)
X-squared = 11.355, df = 20, p-value = 0.9365
```

```r
# Plot the ACF of the residuals to visually assess if there's still serial correlation
acf(residuals(garch_model_arima_residuals), main = "ACF of Residuals (ARIMA + GARCH)")
```



Quality Assessment of the ARIMA + GARCH(1,1) Fit: Parameter Estimates:

The fitted ARIMA + GARCH(1,1) model has significant coefficients for the GARCH components: Alpha1 (0.2599) and Beta1 (0.7579) are both highly significant (p-value $<$ 2e-16), indicating that the GARCH(1,1) model has captured the volatility clustering effectively. The intercept (Mu) is not significant, suggesting that the mean of the residuals is close to zero, which is expected in a well-fitted model. Log-Likelihood and AIC:

The log-likelihood of the model is high (7627.039), and the AIC (-10.59491) and BIC (-10.58026) indicate a good fit for the model. Lower AIC/BIC values suggest a better model fit. Residuals Analysis:

Ljung-Box Test for Serial Correlation: The p-value of the Ljung-Box test ($p = 0.9365$) for the residuals of the ARIMA + GARCH(1,1) model is very high, indicating no significant serial correlation. This suggests that the model has captured the dependencies in the data well, and the residuals are essentially white noise. Jarque-Bera and Shapiro-Wilk Tests for Normality: Both tests indicate non-normality of the residuals (p-value = 0.000). This is common in financial time series data, as they often exhibit heavy tails and skewness. Ljung-Box Test on Squared Residuals: The p-values for the Ljung-Box test on squared residuals are also high, indicating no remaining conditional heteroscedasticity in the model, suggesting that the GARCH model has effectively modeled the conditional variance. Conclusion:

The ARIMA + GARCH(1,1) model fits the data well, capturing both the autocorrelation in the returns and the conditional heteroscedasticity (volatility clustering). The residuals show no significant autocorrelation, and the GARCH model appears to have adequately handled the volatility. The model might still exhibit non-normality, but this is expected in financial data due to extreme returns or fat tails.

f) Model Comparison

Question

Compare the three models from the previous parts. Which is more suitable? In which of these models is the homoscedasticity assumption violated?

Comparison of the Three Models:

Model 1: GARCH(1,1) with Normal Distribution

Fit: This model captured the volatility clustering well. The Ljung-Box test on residuals and squared residuals showed no significant serial correlation, indicating that the model accounted for the time-varying volatility. Homoscedasticity: By definition, a GARCH model assumes heteroscedasticity, meaning that the volatility changes over time. Therefore, the assumption of constant variance (homoscedasticity) is explicitly violated in this model. Conclusion: This model is good for capturing conditional heteroscedasticity (changing volatility), and thus it is appropriate for financial time series data where volatility clustering is observed.

Model 2: GARCH(1,1) with Standardized t-Distribution

Fit: Similar to the GARCH with normal distribution, this model accounts for volatility clustering and captures extreme returns better due to the heavy tails of the t-distribution. Homoscedasticity: The GARCH structure still assumes heteroscedasticity, and therefore the homoscedasticity assumption is violated in this model as well. Conclusion: This model is better suited for capturing heavy-tailed data, like extreme price movements in financial markets, while still accounting for changing volatility. It can outperform the GARCH(1,1) with a normal distribution for financial data with fat tails.

Model 3: ARIMA + GARCH(1,1) (Two-Step Approach)

Fit: This two-step model first applies an ARIMA to remove autocorrelation from the log returns and then applies a GARCH(1,1) to model the remaining volatility clustering. The Ljung-Box tests on residuals showed no significant serial correlation, suggesting that the ARIMA model handled the autoregressive components well, and the GARCH captured the volatility. Homoscedasticity: As in the other GARCH models, this model also assumes heteroscedasticity, meaning the homoscedasticity assumption is violated here as well. Conclusion: This model is the most comprehensive because it first removes any autocorrelation in the series (via ARIMA) before modeling the volatility. It is typically more suitable for financial time series where both autocorrelation and volatility clustering are present.

Conclusion: Most Suitable Model: The ARIMA + GARCH(1,1) model is the most suitable because it addresses both the autocorrelation in the log returns and the heteroscedasticity (changing volatility). It captures both the time-varying nature of volatility and any serial dependence in the data. Homoscedasticity Violation: In all three models, the homoscedasticity assumption is violated since all models incorporate the GARCH(1,1) structure, which models conditional heteroscedasticity (changing volatility over time).

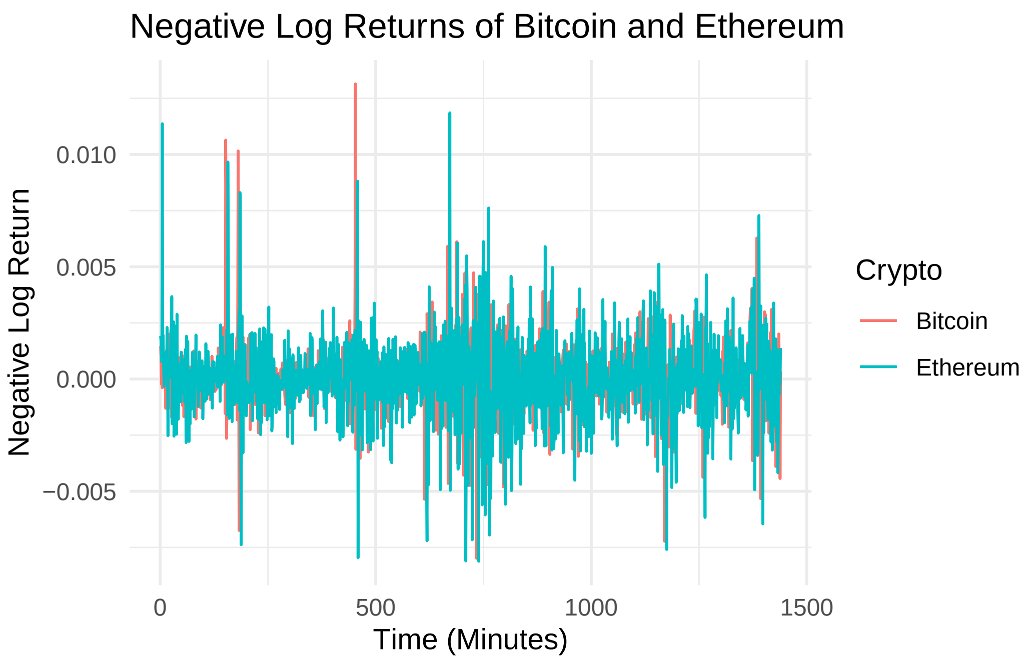
Part 3: Financial returns and normality

a) Check Bitcoin-ETH dependence using correlation

Question

Are the negative log returns of Bitcoin and ETH dependent? Compute the correlation using `cor.test()` function. Can we conclude that these series are independent?

To answer this question, we will first plot both negative log returns and then perform a correlation test.



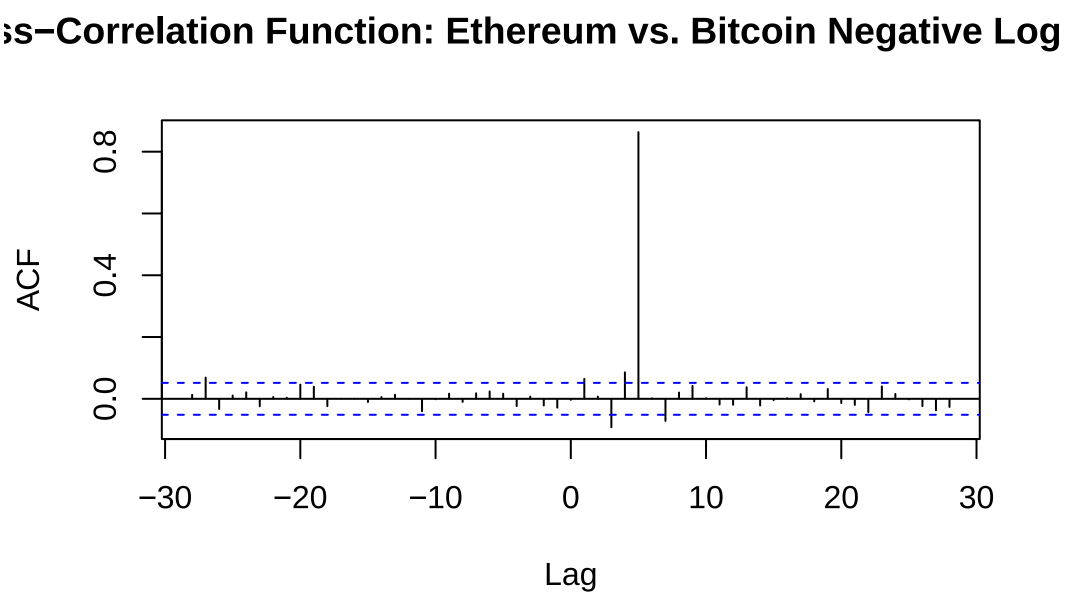
The graph could suggest that the negative log returns of Bitcoin and Ethereum are dependent. However, we will perform a correlation test to confirm this.

The computed correlation between the negative log returns of Bitcoin and Ethereum using the `cor.test()` function is -0.0031 and is not statistically significant (p-value = 0.905). However, the lack of significance in the correlation test does not necessarily imply independence, as there may be a non-linear relationship or other factors influencing the association between the two variables.

b) Analyze Bitcoin-ETH cross-correlation function

Question

Calculate the cross-correlation function (CCF) between the negative log returns of Bitcoin and ETH. What do you observe?



We can clearly observe a significant spike at lag 5 indicates that there is a notable correlation between Bitcoin's returns and Ethereum's returns when Ethereum's returns are shifted by 5 minutes. Ethereum's returns are correlated with past values of Bitcoin's returns and thus may predict Etherum's returns with a 5 minutes delay. The spike indicates the strength of the correlation. In our case, the spike at lag 5 crosses the significance threshold (horizontal dashed lines). We can say with confidence that it is statistically significant and not likely due to random noise. However, it is important to keep in mind that cross-correlation does not imply causation. Other factors or indirect relationships might also explain this pattern.

c) Assess predictive power with Granger test

Question

Is one of the time series good predictor of the second? Assess whether there is any predictive power between the negative log returns of Bitcoin and ETH. You can use `grangertest()` in the `lmtest` package with carefully chosen hyperparameter `order`. What is your conclusion?

We performed here two grander test using the `grandertest()`function. The first tests if Ethereum's past values do not Granger-cause Bitcoin's returns. With a p-value of 0.7132, we cannot reject the null hypothesis. Therefore, there is no evidence to suggest that Ethereum's past values have predictive power for Bitcoin's returns.

The second test examines if Bitcoin's past values Granger-cause Ethereum's returns. With a p-value of 0, Bitcoin's past values significantly Granger-cause Ethereum's returns, meaning there is strong evidence of predictive power.

d) Predict mutual impacts of drops

Question

Based on your answer in (c), answer the following questions:

d.1) Predict ETH reaction to Bitcoin drop

Question

We observe an extreme sudden drop in Bitcoin stocks. What should we expect that will happen with ETH stocks?

Given our results in question c), the sudden drop in Bitcoin stocks should have a significant impact on the Etherum stocks price. Given our analysis, the Ethereum stocks should drop 5 minutes (lag of 5 / question b)) after the Bitcoin stocks drop with a value close to 80% of the Bitcoin drop.

d.2) Predict Bitcoin reaction to ETH drop

Question

We observe an extreme sudden drop in ETH stocks. What should we expect that will happen with Bitcoin stocks?

Based on the Granger causality test results from question c), Ethereum's past values do not Granger-cause Bitcoin's returns, suggesting no significant predictive relationship. If Ethereum experiences an extreme sudden drop, it is less likely that Bitcoin will exhibit a direct or immediate reaction based on historical relationships. However, Bitcoin and Ethereum share common market factors and investor sentiment, so an indirect reaction (e.g., due to market-wide panic) is still possible, though it is not strongly supported by the causality analysis.

Practical 2: Weather

Part 1: Block maxima approach

a) Best fit distribution

Question

Read in the data. Draw an histogram of the daily precipitation values. Which distribution would best fit the data ?

b) Best fit distribution on yearly maxima

Question

Extract the yearly maximum values. Draw their histogram. Which distribution would best fit the data ?

c) Linear model approach

Question

Fit a linear model to the yearly maximum precipitation values and predict the values for the next 10 years. Provide confidence intervals for your predictions and plot it. Do you think that this a reasonable approach?

d) GEV distribution

Question

Fit a GEV with constant parameters to the historical yearly max values. We recommend using fevd function in `extRemes` library or `gev.fit` function in `ismev` library. Fit a second GEV model with time varying location parameter. Compare the two models using AIC or BIC. Which one do you recommend using?

e) Diagnostic plots

Question

Draw diagnostic plots of your GEV fit (for example, using `gev.diag` function). Is it a good fit?

f) Return levels prediction

Question

Using the model chosen in the previous parts, predict the 10-year return level. Draw your predictions of the 10-year return levels together with your data.

g) Interpretation

Question

Broadly speaking, each year, there is a chance of 1/10 that the observed value is above the 10-year return level. Comment on the results for both the linear model prediction (from (c)) and the GEV approach (from (f)). How many historical values were above this 10-year return level? Answer the same question with 20, 50 and 85-year return level.

h) Return period 100 mm precipitation

Question

Using the fitted model, compute the return period of 100 mm of precipitation.

i) Probability of exceeding 150 mm

Question

Using the fitted model, compute the probability that there will be a day in the next year when the precipitation exceeds 150 mm.

Part 2: Peaks-over-threshold approach

a) Time series plot

Question

Display a time series plot of the daily precipitation across the data range.

b) Mean Residual Life Plot

Question

We want to model the high precipitation levels using the POT approach. First step is choosing a threshold. Draw Mean Residual Life Plot (for example using `mrlplot` in `POT` library) for the full range of your data. Choose a reasonable threshold. In the plot from part (a)) highlight the data that exceeds this threshold.

c) Fit a GPD

Question

Fit a GPD for the data exceeding the threshold and draw a diagnostic plot. Is it a reasonable fit? (Hint: if not, you may reconsider the choice of the threshold)

d) Return levels prediction

Question

Using the fitted model, compute the 10-year, 20-year, 50-year and 85-year return levels.

e) Return period 100 mm precipitation

Question

Using the fitted model, compute the return period of 100 mm of precipitation.

f) Probability of exceeding 150 mm

Question

Using the fitted model, compute the probability that there will be a day in the next year when the precipitation exceeds 150 mm.

g) Comparison with block maxima

Question

Compare the results with the block maxima method. Explain the drawbacks and advantages of using the POT approach compared to the block maxima method. Which method do you prefer?

Part 2

Part 3

Practical 3: Heat wave in Switzerland

Part 3: Comparing GEV and POT approaches

Data Loading and Preparation

```r
# load the required packages and install them if they are not.
source(here::here("code","setup.R"))

# Load the data
data_temp <- read.csv(here("data", "Cleaned_Stations_Data.csv"))

# Filter data for all cities and remove any rows with NA in TMAX
# Geneva
data_geneva <- data_temp %>%
  filter(!is.na(TMAX)) %>%
  filter(NAME == "Genève") %>%
  dplyr::select(DATE, TMAX) %>%
  mutate(DATE = as.Date(DATE))

# Santïs
data_santis <- data_temp %>%
  filter(!is.na(TMAX)) %>%
  filter(NAME == "Saentis") %>%
  dplyr::select(DATE, TMAX) %>%
  mutate(DATE = as.Date(DATE))

# Lugano
data_lugano <- data_temp %>%
  filter(!is.na(TMAX)) %>%
  filter(NAME == "Lugano") %>%
  dplyr::select(DATE, TMAX) %>%
  mutate(DATE = as.Date(DATE))
```

We ensure that we have the necessary data for the analysis by filtering the temperature data for all three cities and removing any rows with missing values. We removed 395 rows.

Block Maxima Approach (GEV Distribution)

Question

Can a Generalized Extreme Value (GEV) distribution accurately model annual maximum temperatures in Switzerland?

In this analysis, we investigate whether a Generalized Extreme Value (GEV) distribution can accurately model annual maximum temperatures in Switzerland. Additionally, we compare the results of the block Maxima approach with the Peaks-over-Threshold (POT) approach using declustering for extreme temperatures.

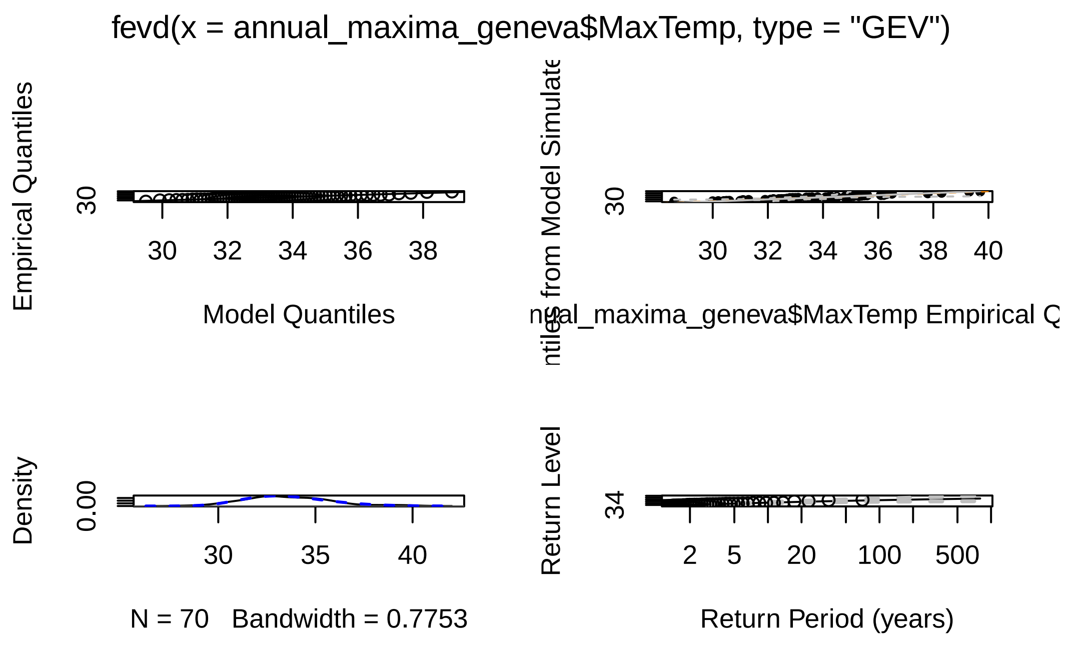
We calculate the annual maximum temperatures for Geneva, Santïs and Lugano and fit a GEV distribution to these maxima.

```r
# Calculate annual maximum temperatures
# Geneva
```

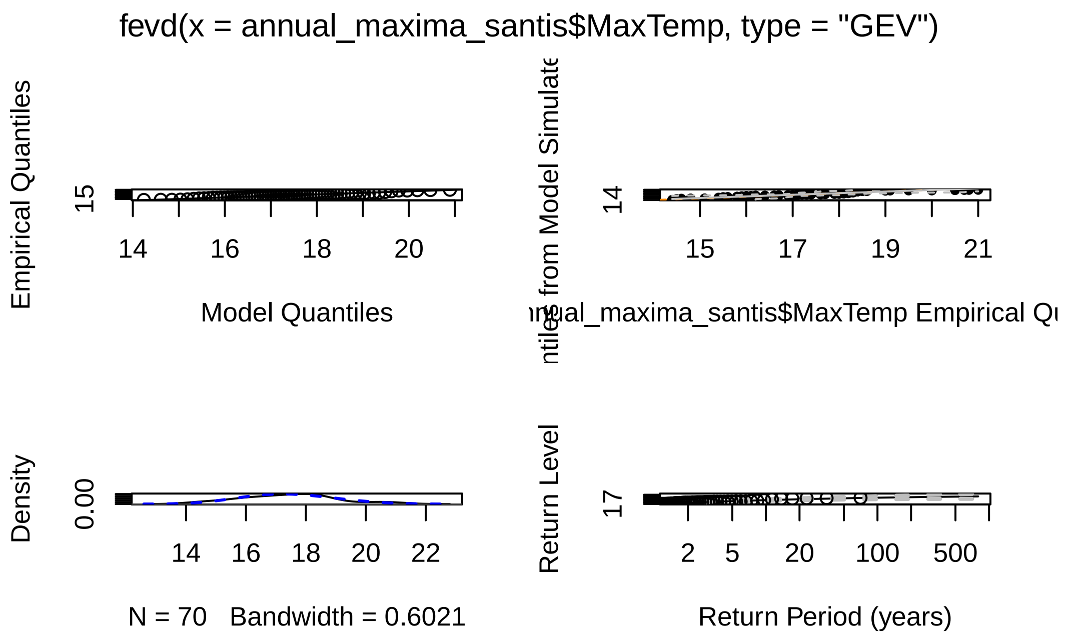
```
annual_maxima_geneva <- data_geneva %>%
  mutate(Year = year(DATE)) %>%
  group_by(Year) %>%
  summarize(MaxTemp = max(TMAX), .groups = 'drop')

# Santïs
annual_maxima_santis <- data_santis %>%
  mutate(Year = year(DATE)) %>%
  group_by(Year) %>%
  summarize(MaxTemp = max(TMAX), .groups = 'drop')

# Lugano
annual_maxima_Lugano <- data_lugano %>%
  mutate(Year = year(DATE)) %>%
  group_by(Year) %>%
  summarize(MaxTemp = max(TMAX), .groups = 'drop')
```

```
# Fit a GEV distribution to the annual maxima
# Genève
gev_fit_geneva <- fevd(annual_maxima_geneva$MaxTemp, type = "GEV")

# Santïs
gev_fit_santis <- fevd(annual_maxima_santis$MaxTemp, type = "GEV")

# Lugano
gev_fit_Lugano <- fevd(annual_maxima_Lugano$MaxTemp, type = "GEV")
```

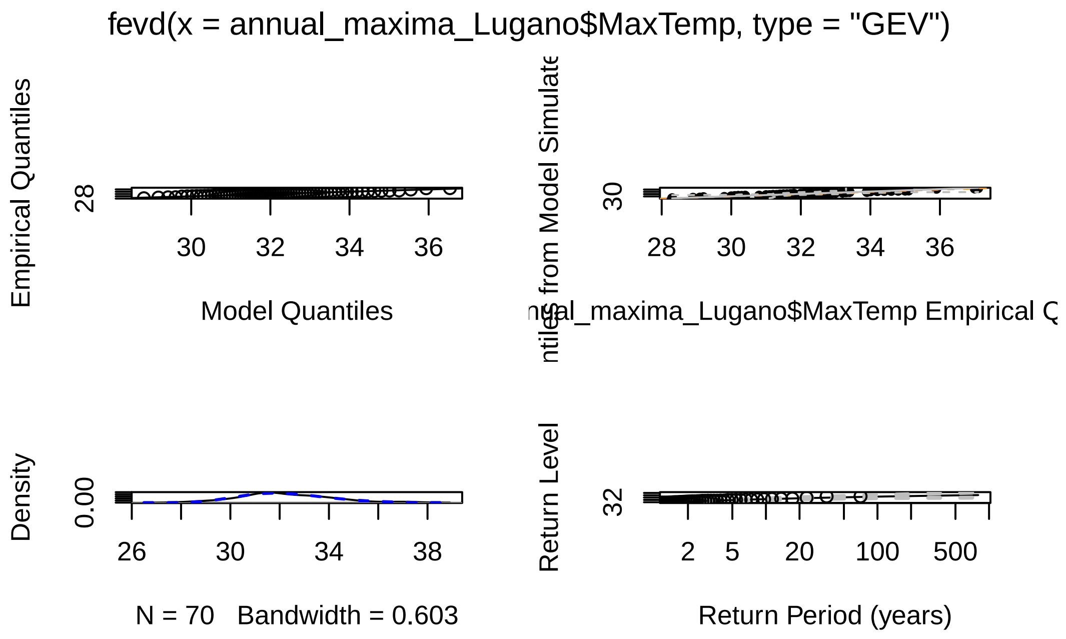
```
# Diagnostic plots for the GEV fit
# Geneva
par(mfrow = c(2, 2))
plot(gev_fit_geneva)
```



```r
# Santïs
par(mfrow = c(2, 2))
plot(gev_fit_santis)
```



```r
# Lugano
par(mfrow = c(2, 2))
plot(gev_fit_Lugano)
```



We have here the diagnostic plots for the GEV fit, which include the quantile-quantile plot, the return level plot, the probability plot, and the density plot. We see that overal the fit is good.

```r
# Calculate return levels for 10, 50, and 100-year return periods
# Genève
gev_return_levels_geneva <- return.level(gev_fit_geneva, return.period = c(10, 50, 100))

# Santïs
gev_return_levels_santis <- return.level(gev_fit_santis, return.period = c(10, 50, 100))

# Lugano
gev_return_levels_Lugano <- return.level(gev_fit_Lugano, return.period = c(10, 50, 100))
```

Comparison with Peaks-over-Threshold Approach (GPD Distribution)

Question

How do the results of the block maxima approach compare with the Peaks-over-Threshold (POT) approach for temperature extremes?

Peaks-over-Threshold Approach

```r
# Filter the data for the summer months (June to September)
# Geneva
geneva_summer <- subset(data_geneva, format(DATE, "%m") %in% c("06", "07", "08", "09"))

# Santis
santis_summer <- subset(data_santis, format(DATE, "%m") %in% c("06", "07", "08", "09"))

# Lugano
lugano_summer <- subset(data_lugano, format(DATE, "%m") %in% c("06", "07", "08", "09"))
```

```r
# Define a threshold at the 95th percentile
# Geneva
threshold_geneva <- quantile(geneva_summer$TMAX, 0.95)

# Santïs
threshold_santis <- quantile(santis_summer$TMAX, 0.95)

# Lugano
threshold_lugano <- quantile(lugano_summer$TMAX, 0.95)
```

```r
# Number of exceedances over the threshold
# Geneva
num_exceedances_geneva <- sum(geneva_summer$TMAX > threshold_geneva)

# Santïs
num_exceedances_santis <- sum(santis_summer$TMAX > threshold_santis)

# Lugano
```

```r
num_exceedances_Lugano <- sum(lugano_summer$TMAX > threshold_lugano)
```

We now apply the Peaks-over-Threshold (POT) approach using a suitable threshold. With a 95% threshold, we have 413 exceedances over the threshold for Genève, 413 for Santïs, and 406 for Lugano.

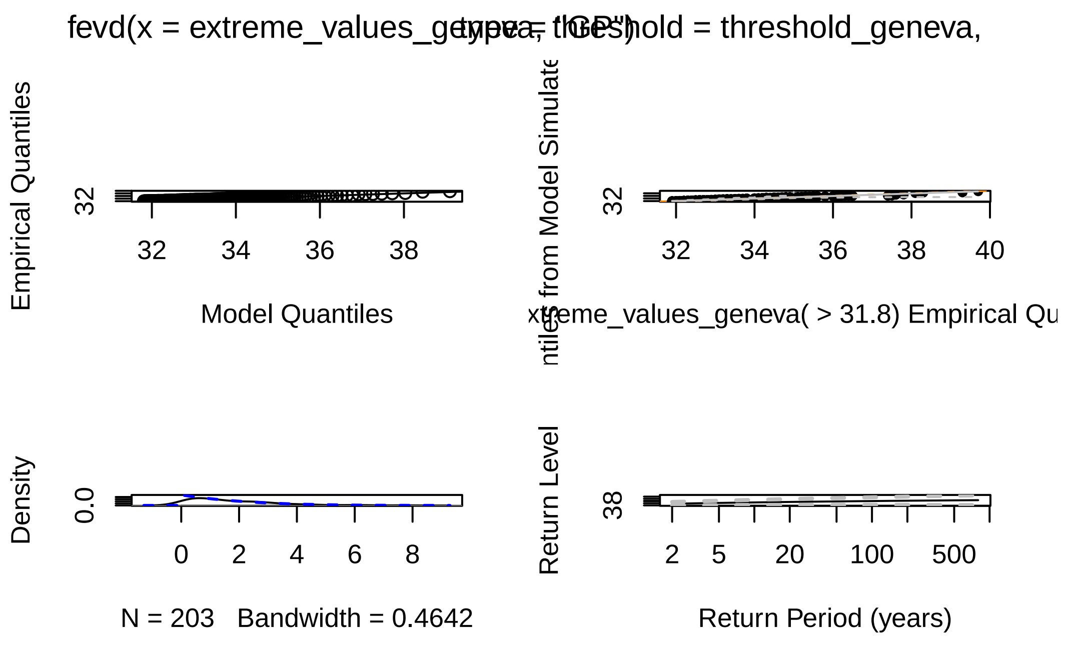
```r
# Decluster the data using the chosen threshold for each station
# Genava
geneva_declustered <- extRemes::decluster(geneva_summer$TMAX, threshold = threshold_geneva, run.l

# Santïs
santis_declustered <- extRemes::decluster(santis_summer$TMAX, threshold = threshold_santis, run.l

# Lugano
lugano_declustered <- extRemes::decluster(lugano_summer$TMAX, threshold = threshold_lugano, run.l
```

```r
# Add the declustered data to the corresponding datasets
# Geneva
geneva_summer$declustered <- ifelse(geneva_summer$TMAX >= threshold_geneva, geneva_declustered, N

# Santïs
santis_summer$declustered <- ifelse(santis_summer$TMAX >= threshold_santis, santis_declustered, N

# Lugano
lugano_summer$declustered <- ifelse(lugano_summer$TMAX >= threshold_lugano, lugano_declustered, N
```

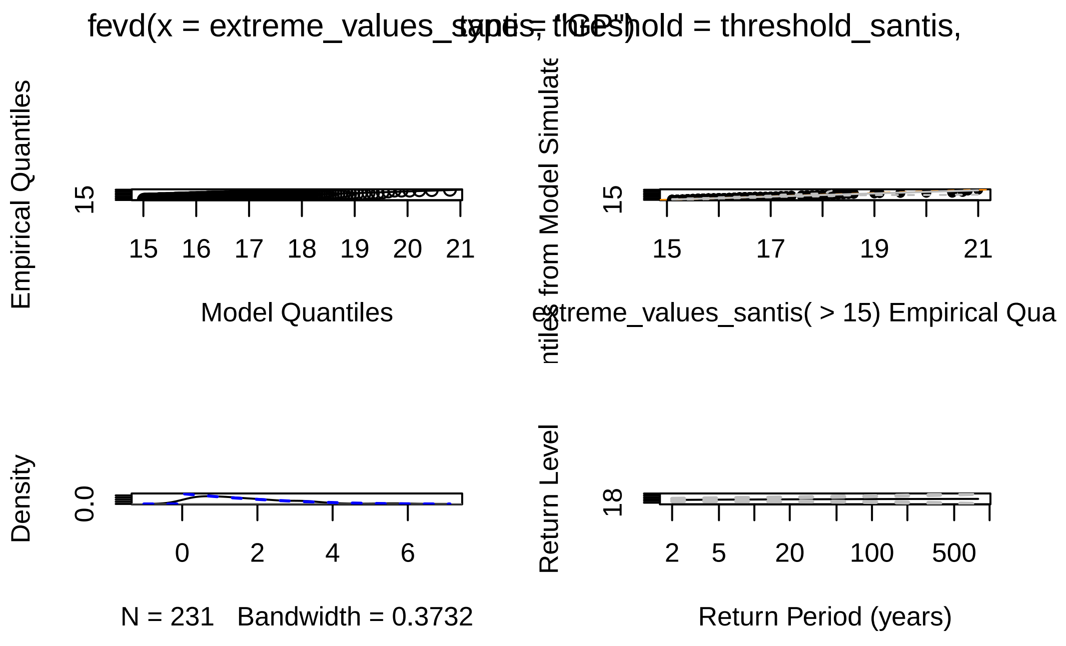
```r
# Extract declustered extreme values
# Geneva
extreme_values_geneva <- geneva_summer[["declustered"]][!is.na(geneva_summer[["declustered"]])]

# Santïs
extreme_values_santis <- santis_summer[["declustered"]][!is.na(santis_summer[["declustered"]])]

# Lugano
extreme_values_lugano <- lugano_summer[["declustered"]][!is.na(lugano_summer[["declustered"]])]
```

```r
# Fit a GPD to the exceedances
# Genève
gpd_fit_geneva <- fevd(extreme_values_geneva, threshold = threshold_geneva, type = "GP")

# Santïs
gpd_fit_santis <- fevd(extreme_values_santis, threshold = threshold_santis, type = "GP")

# Lugano
gpd_fit_lugano <- fevd(extreme_values_lugano, threshold = threshold_lugano, type = "GP")
```

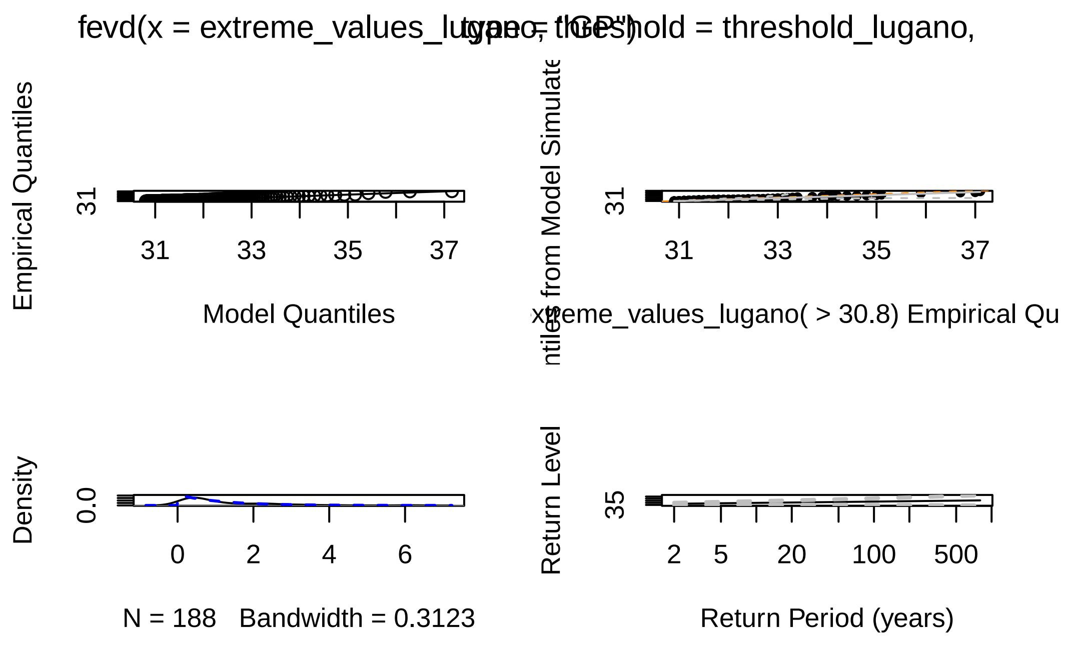
```r
# Diagnostic plots for the GPD fit
# Genève
par(mfrow = c(2, 2))
plot(gpd_fit_geneva)
```



```r
# Santïs
par(mfrow = c(2, 2))
plot(gpd_fit_santis)
```



```r
# Lugano
par(mfrow = c(2, 2))
plot(gpd_fit_lugano)
```



Again, we have the diagnostic plots for the GPD fit, which include the quantile-quantile plot, the return level plot, the probability plot, and the density plot. We see that overal the fit is good.

```r
# Calculate return levels for 10, 50, and 100-year return periods
# Geneva
gpd_return_levels_geneva <- return.level(gpd_fit_geneva, return.period = c(10, 50, 100))

# Santïs
gpd_return_levels_santis <- return.level(gpd_fit_santis, return.period = c(10, 50, 100))

# Lugano
gpd_return_levels_Lugano <- return.level(gpd_fit_lugano, return.period = c(10, 50, 100))
```

Our results

Block Maxima Approach (GEV Distribution):

Parameter Estimates:

Station	Elevation	location parameter	scale parameter	shape parameter
Geneva	375m	32.69	1.97	-0.15
Santïs	2500m	16.84	1.5	-0.23
Lugano	275m	31.51	1.65	-0.17

GEV Return Levels:

Station	Elevation	10-year RL (°C)	50-year RL (°C)	100-year RL (°C)
Geneva	375m	36.45	38.51	39.23
Santïs	2500m	19.47	20.69	21.08

Station	Elevation	10-year RL (°C)	50-year RL (°C)	100-year RL (°C)
Lugano	275m	34.61	36.24	36.81

Peaks-over-Threshold Approach (GPD Distribution):

Threshold Selection:

The threshold was set at the 95th percentile, which is 31.8 degrees Celsius for Geneva

The threshold was set at the 95th percentile, which is 15 degrees Celsius for Santïs.

The threshold was set at the 95th percentile, which is 30.8 degrees Celsius for Lugano.

Number of Exceedances:

There are 413 exceedances over the threshold for Geneva.

There are 413 exceedances over the threshold for Santïs.

There are 406 exceedances over the threshold for Lugano.

GPD Return Levels:

The return levels are:

Station	Elevation	10-year RL (°C)	50-year RL (°C)	100-year RL (°C)
Geneva	375m	40.72	41.63	41.96
Santïs	2500m	21.52	21.84	21.94
Lugano	275m	39.88	42.01	42.95

Comparison of Approaches:

In comparing the block maxima approach (GEV) to the Peaks-over-Threshold approach (GPD) for modeling temperature extremes, we find that the POT method often provides more nuanced information about the tail behavior of the distribution. While the GEV model uses only the annual maximum values, which can lead to relatively large uncertainty due to a limited amount of extreme data, the POT approach utilizes all temperature values above a chosen high threshold. This generally results in more data points to characterize the tail, potentially yielding more stable and reliable estimates of parameters and return levels. However, the POT approach depends on careful threshold selection and declustering, which introduces additional steps not required by the simpler block maxima method. In practice, the POT approach can lead to different (often more precise) return level estimates, giving a potentially more accurate picture of the frequency and magnitude of extreme temperature events.

Part 4: Return Period & Probabilities of New Record High Temperatures

In this part, we will discuss the return periods & probability analysis of having a new record of heatwaves in the cities of Geneva, Säntis and Lugano. As seen when comparing the Block Maxima and the Peaks-over-Threshol approaches, the Block Maxima one seems to be more accurate to use. Thus, we will only use this approach when calculating the return period and the probabilities of obtaining a new record of temperature for 1, 3 and 10 years. To begin with, we will take the code from part 3 to obtain the GEV distribution for all cities.

Block Maxima Approach (GEV Distribution)

See part 3 for the code.

Peaks-over-Threshold Approach (GPD Distribution)

See part 3 for the code.

Question

Can a Generalized Extreme Value (GEV) distribution accurately model annual maximum temperatures in Switzerland?

Return Period

Now, we will analyse the return period for each city. The return period is the expected time (here, in years) between occurrences of an extreme event to be at least as extreme as a given value.

```r
# Load required library
library(extRemes)

# Extract GEV parameters for Geneva
location_geneva <- gev_fit_geneva$results$par["location"]
scale_geneva <- gev_fit_geneva$results$par["scale"]
shape_geneva <- gev_fit_geneva$results$par["shape"]

# Extract GEV parameters for Säntis
location_santis <- gev_fit_santis$results$par["location"]
scale_santis <- gev_fit_santis$results$par["scale"]
shape_santis <- gev_fit_santis$results$par["shape"]

# Extract GEV parameters for Lugano
location_lugano <- gev_fit_Lugano$results$par["location"]
scale_lugano <- gev_fit_Lugano$results$par["scale"]
shape_lugano <- gev_fit_Lugano$results$par["shape"]

# Set the temperature threshold
temp_threshold <- 35
temp_threshold_santis <- 19

# Function to calculate return period
```

```r
calculate_return_period <- function(location, scale, shape, temp_threshold) {
  cdf_value <- pevd(temp_threshold, loc = location, scale = scale, shape = shape, type = "GEV")
  return_period <- 1 / (1 - cdf_value)
  return(return_period)
}

# Calculate return periods
return_period_geneva <- calculate_return_period(location_geneva, scale_geneva, shape_geneva, temp
return_period_santis <- calculate_return_period(location_santis, scale_santis, shape_santis, temp
return_period_lugano <- calculate_return_period(location_lugano, scale_lugano, shape_lugano, temp

# Print the results
cat("The return period for 35°C in Geneva is approximately", round(return_period_geneva, 2), "yea
```

```
The return period for 35°C in Geneva is approximately 4.16 years.
```

```r
cat("The return period for 35°C in Säntis is approximately", round(return_period_santis, 2), "yea
```

```
The return period for 35°C in Säntis is approximately 6.29 years.
```

```r
cat("The return period for 35°C in Lugano is approximately", round(return_period_lugano, 2), "yea
```

```
The return period for 35°C in Lugano is approximately 14.04 years.
```

As a result, we can observe that the return period is around 4.16 years in Geneva, Inf for Säntis and 14.04 years for Lugano.

Probability of New Record High Temperature

Using the previously fitted model

```r
# Historical record highs for each city (replace with actual values)
record_high_geneva <- max(annual_maxima_geneva$MaxTemp)
record_high_santis <- max(annual_maxima_santis$MaxTemp)
record_high_lugano <- max(annual_maxima_Lugano$MaxTemp)

# Function to calculate the probability of a new record high
calculate_new_record_probability <- function(location, scale, shape, record_high, years) {
  # Probability of not exceeding the record high in 1 year
  prob_not_exceed <- pevd(record_high, loc = location, scale = scale, shape = shape, type = "GEV"
  # Probability of exceeding the record high in the given time horizon
  prob_exceed <- 1 - (prob_not_exceed^years)
  return(prob_exceed)
}

# Calculate probabilities for Geneva
```

```r
prob_new_record_geneva_1yr <- calculate_new_record_probability(location_geneva, scale_geneva, sha
prob_new_record_geneva_3yr <- calculate_new_record_probability(location_geneva, scale_geneva, sha
prob_new_record_geneva_10yr <- calculate_new_record_probability(location_geneva, scale_geneva, sh

# Calculate probabilities for Säntis
prob_new_record_santis_1yr <- calculate_new_record_probability(location_santis, scale_santis, sha
prob_new_record_santis_3yr <- calculate_new_record_probability(location_santis, scale_santis, sha
prob_new_record_santis_10yr <- calculate_new_record_probability(location_santis, scale_santis, sh

# Calculate probabilities for Lugano
prob_new_record_lugano_1yr <- calculate_new_record_probability(location_lugano, scale_lugano, sha
prob_new_record_lugano_3yr <- calculate_new_record_probability(location_lugano, scale_lugano, sha
prob_new_record_lugano_10yr <- calculate_new_record_probability(location_lugano, scale_lugano, sh

# Print the results
cat("Probability of a new record high in Geneva:\n",
    "1 year:", round(prob_new_record_geneva_1yr, 4), "\n",
    "3 years:", round(prob_new_record_geneva_3yr, 4), "\n",
    "10 years:", round(prob_new_record_geneva_10yr, 4), "\n\n")
```

```
Probability of a new record high in Geneva:
 1 year: 0.0061
 3 years: 0.0183
 10 years: 0.0596
```

```r
cat("Probability of a new record high in Säntis:\n",
    "1 year:", round(prob_new_record_santis_1yr, 4), "\n",
    "3 years:", round(prob_new_record_santis_3yr, 4), "\n",
    "10 years:", round(prob_new_record_santis_10yr, 4), "\n\n")
```

```
Probability of a new record high in Säntis:
 1 year: 0.0116
 3 years: 0.0343
 10 years: 0.1098
```

```r
cat("Probability of a new record high in Lugano:\n",
    "1 year:", round(prob_new_record_lugano_1yr, 4), "\n",
    "3 years:", round(prob_new_record_lugano_3yr, 4), "\n",
    "10 years:", round(prob_new_record_lugano_10yr, 4), "\n")
```

```
Probability of a new record high in Lugano:
 1 year: 0.0068
 3 years: 0.0201
 10 years: 0.0655
```

Results

As a result, we can observe that ...