

Tarea 2: Series temporales II

En esta actividad se propone la realización de un análisis completo de series temporales utilizando las herramientas vistas en clase.

Tenéis que **elegir una serie temporal** y analizarla. Es necesario indicar de donde se obtiene la serie temporal y poner el código R utilizado.

Se valorará tanto el análisis, como las explicaciones y conclusiones obtenidas. Es muy importante la redacción y presentación de la información.

Objetivos

- Decidir el tipo de modelo adecuado (ARCH, GARCH y sus variantes) dependiendo de las características de la serie.
- Aplicar el modelo y extraer conclusiones a partir del mismo.
- Comunicar dichas conclusiones.

Se puede aplicar varios modelos y compararlos.

Tarea 2: Series Temporales II

Análisis de Series temporales Modelos ARCH y GARCH

4º Grado Matemáticas

Análisis serie temporal

Introducción

Para mi segunda tarea de Series temporales II voy a analizar una serie temporal con datos tomados de internet y voy a determinar las características de la misma, para poder determinar si podemos aplicar los conocimientos adquiridos en el tema 4 sobre los modelos ARCH y GARCH que capturan la volatilidad condicional de los residuos del modelo seleccionado para mi serie temporal. Todo mi análisis será realizado con el software de Rstudio y sus múltiples librerías.

1.Definición de los datos y análisis inicial de la serie temporal

En primer lugar, antes de empezar a cargar los datos para generar mi serie temporal voy a importar todas las librerías que voy a necesitar durante el análisis de mi serie:

```
#--  
#Importamos las librerías que vamos a usar  
library(forecast)  
library(tseries)  
library(FinTS)  
library(dynlm)  
library(ggplot2)  
library(rugarch)
```



```
# Librerías que he tenido que instalar porque no las tenía instaladas previamente  
#install.packages("dynlm")  
#install.packages("FinTS")
```

A continuación, una vez que hemos cargado las librerías vamos a importar los datos que emplearemos en la creación de la serie temporal;

```
#Para construir nuestra serie temporal vamos a tomar datos financieros,  
#los cuales vamos a extraer de la página Yahoo Finance.  
datos <- "C:/Users/Rubén Garrido/Desktop/Series Temporales II/Modelos ARCH y GARCH/ETH-EUR.csv"  
cambio <- read.csv(datos)  
head(cambio)  
str(cambio)  
show(cambio)
```



```
#Datos(descripción) Tenemos los datos del precio de la criptomoneda Ethereum  
#con respecto al euro. En mi caso tomaré las columnas de cambio$date y cambio$High,  
#con esas dos estudiaria que modelo se ajusta mejor a los datos
```



```
datos_seleccionados <- cambio[, c("Date", "High")]
```

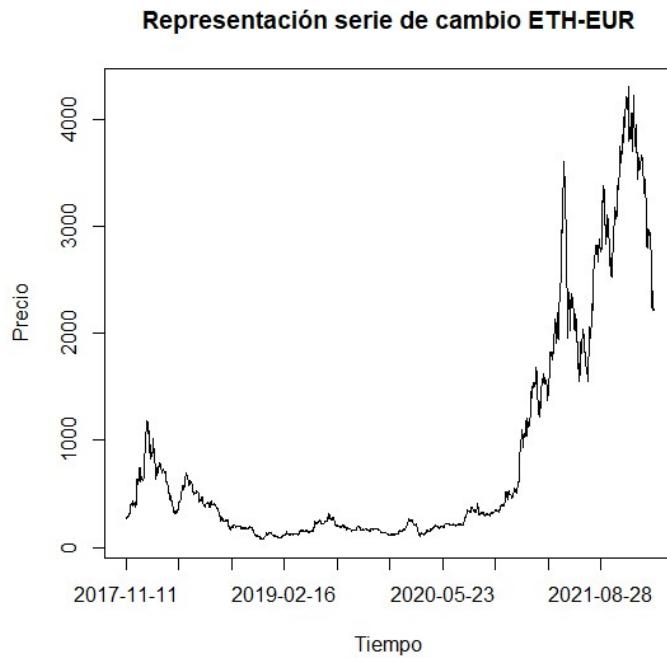
Descripción de los datos: Los datos que hemos importado han sido extraídos de la página de Yahoo Finance. Estos muestran el valor de la criptomoneda Ethereum con respecto al precio del euro todos los días desde el 11/11/2017 hasta el 28/01/2022. Este conjunto de datos posee

diferentes columnas, de las cuales yo solo voy a emplear dos para construir mi serie temporal.

Emplearé las columnas de datos cambio\$Date y cambio\$High correspondientes a la fecha de registro de los datos y el precio máximo alcanzado de la moneda Ethereum con respecto al euro cada día.

Seguidamente construimos una serie temporal con estos datos y la representamos para ir analizando cómo ha ido variando el precio del Ethereum con respecto del euro a lo largo del periodo establecido en los datos:

```
st_eth <- ts(datos_seleccionados$High, frequency = 1)
plot(st_eth,
      type = "l",
      xaxt = "n",
      main = "Representación serie de cambio ETH-EUR",
      xlab = "Tiempo",
      ylab = "Precio")
axis(1, at = seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10)),
     labels = datos_seleccionados>Date[seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10))]
```



Análisis de la representación de la serie temporal: A simple vista vemos que la serie posee cierta tendencia, una tendencia negativa al principio y una tendencia positiva al final. Es decir, desde finales de 2017 hasta 2019 el precio del Ethereum con respecto al euro decreció significativamente, pero tras mantenerse estable durante un año, donde sufre pequeñas fluctuaciones, es a partir de mitad de 2020, puede que posiblemente debido a factores externos como la pandemia. La cual cambió la vida de las personas y produjo grandes cambios en la economía durante un largo

periodo de tiempo, cuando el precio del ethereum se disparó llegando a cuadriplicar su valor inicial. Como conclusión podemos decir que la serie posee tendencia y por lo tanto no es estacionaria ni en media ni en varianza.

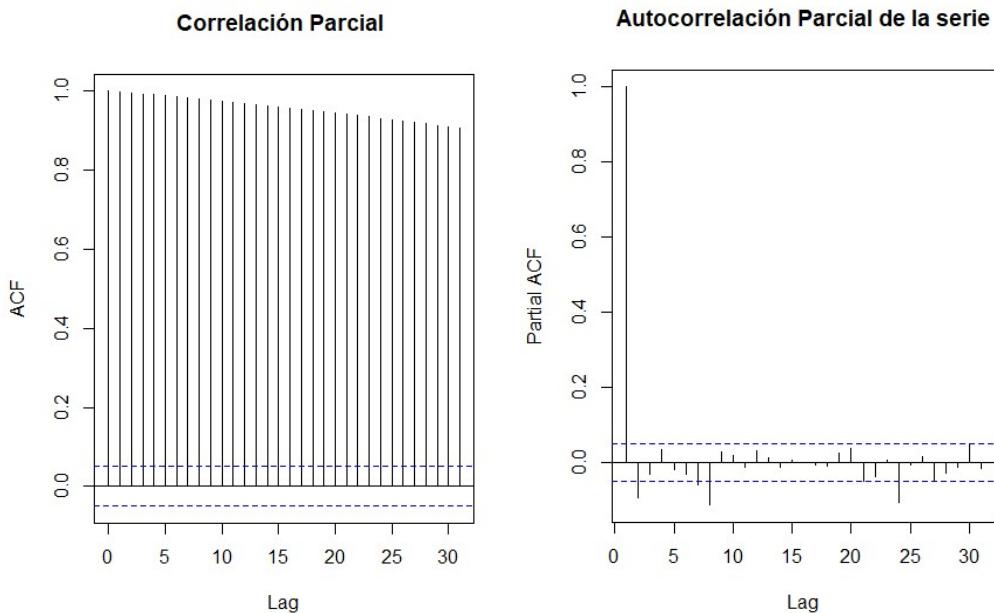
Como queremos determinar qué modelo se ajusta mejor a nuestra serie temporal si un modelo ARMA o ARIMA, para ello necesitamos que nuestra serie temporal sea estacionaria, es decir no posea tendencia.

Transformación serie estacionaria: Vamos a verificar la no estacionariedad de la serie por el test estadístico de Dicker-Fuller, para asegurar que es no estacionaria pese a poder verlo en la gráfica anterior:

```
#Una vez representada la serie temporal vemos que no va a ser estacionaria:  
#Dicker Fuller para verificar si mi serie es estacionaria  
adf_results <- adf.test(st_eth)  
print(adf_results)  
  
#Augmented Dickey-Fuller Test  
  
#data: st_eth  
#Dickey-Fuller = -1.6696, Lag order = 11, p-value = 0.7182  
#alternative hypothesis: stationary  
  
#El p-value es mayor que 0.05 y no podemos rechazar la hipótesis nula.|
```

Vemos que gracias a este test hemos verificado que la serie no es estacionaria.

También podemos analizar las gráficas de autocorrelación parcial y correlación parcial para poder ver la no estacionariedad de la serie.

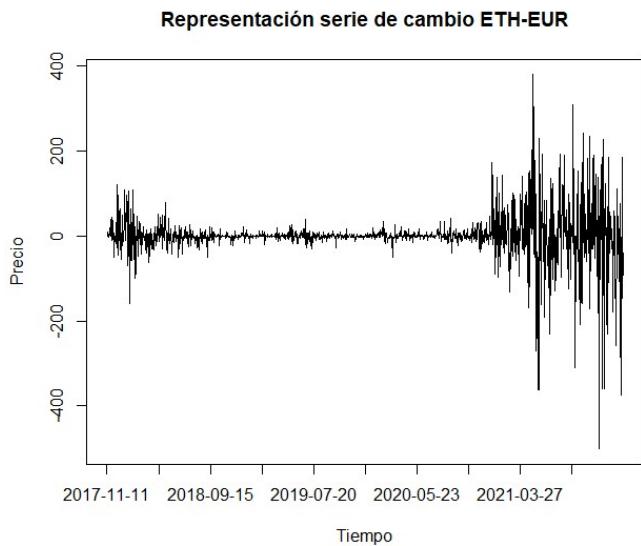


Por lo tanto, vamos a realizar transformaciones en la serie para que esta sea estacionaria en media y varianza.

Tras probar con el logaritmo de la serie he visto que no me sirve así que antes de diferenciar la serie, primero realizamos una transformación de BoxCox sobre los datos para que la serie sea estacionaria en varianza, para posteriormente diferenciar la serie y que sea estacionaria en media.

Hago esto porque si diferenciamos la serie sin realizar la transformación de BoxCox tenemos una serie que solo es estacionaria en media, pero no en varianza:

```
dif_eth <- diff(st_eth)
plot(dif_eth,
      type = "l",
      xaxt = "n",
      main = "Representación serie de cambio ETH-EUR",
      xlab = "Tiempo",
      ylab = "Precio")
axis(1, at = seq(1, length(datos_seleccionados$date), by = round(length(datos_seleccionados$date) / 10)),
     labels = datos_seleccionados$date[seq(1, length(datos_seleccionados$date), by = round(length(datos_seleccionados$date) / 10))])
```



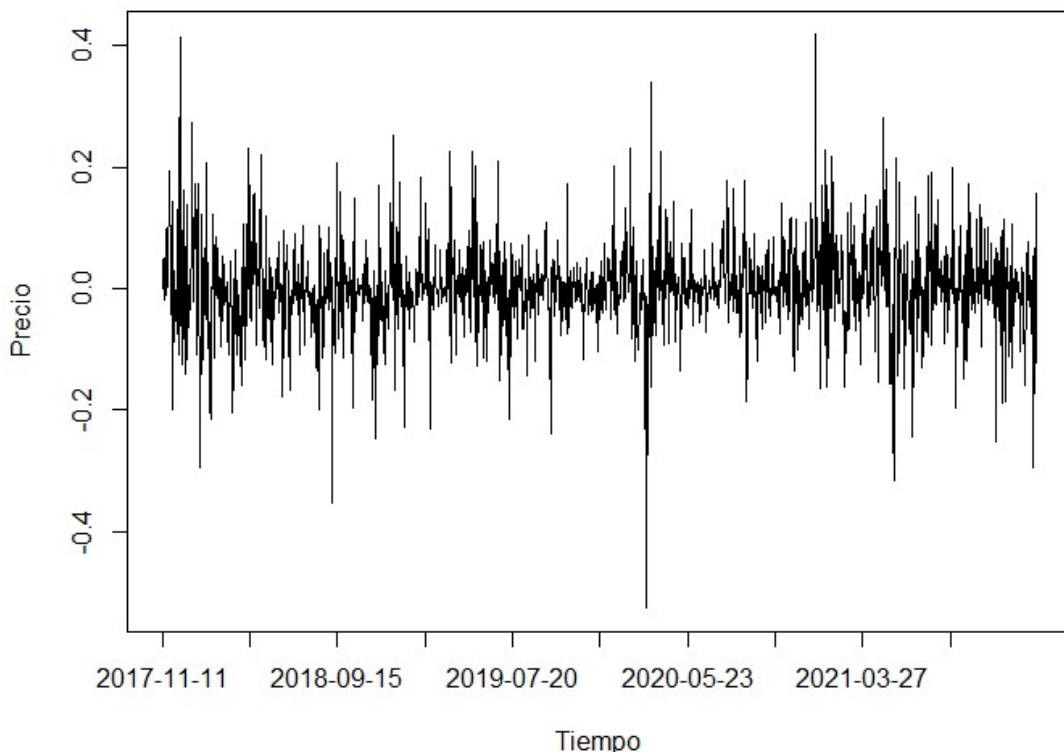
Por lo tanto realizamos la transformación de BoxCox y diferenciamos la serie:

```
#Realizamos una transformación de BoxCox para hacer que la serie sea estacionaria en varianza
lambda_optimo <- BoxCox.lambda(datos_seleccionados$High)
print(lambda_optimo)

datos_transformados <- BoxCox(datos_seleccionados$High, lambda_optimo)
print(datos_transformados)

#Una vez realizado la transformación de BoxCox sobre los datos
#vamos a realizar la representación de la serie con esos datos
st_eth_box <- ts(datos_transformados, frequency = 1)
plot(st_eth_box)
#Diferenciamos estos datos
dif_box <- diff(st_eth_box)
plot(dif_box,
      type = "l",
      xaxt = "n",
      main = "Representación serie de cambio ETH-EUR",
      xlab = "Tiempo",
      ylab = "Precio")
axis(1, at = seq(1, length(datos_seleccionados$date), by = round(length(datos_seleccionados$date) / 10)),
     labels = datos_seleccionados$date[seq(1, length(datos_seleccionados$date), by = round(length(datos_seleccionados$date) / 10))])
```

Representación serie de cambio ETH-EUR



Obteniendo una serie estacionaria, que como vemos en la representación gráfica es estacionaria en media y varianza pues los valores oscilan entre [-0.5,0.5] un intervalo contenido en [-1,1] y están centrados en cero.

Como hicimos previamente verificamos que la serie es estacionaria con el test de Dicker-Fuller:

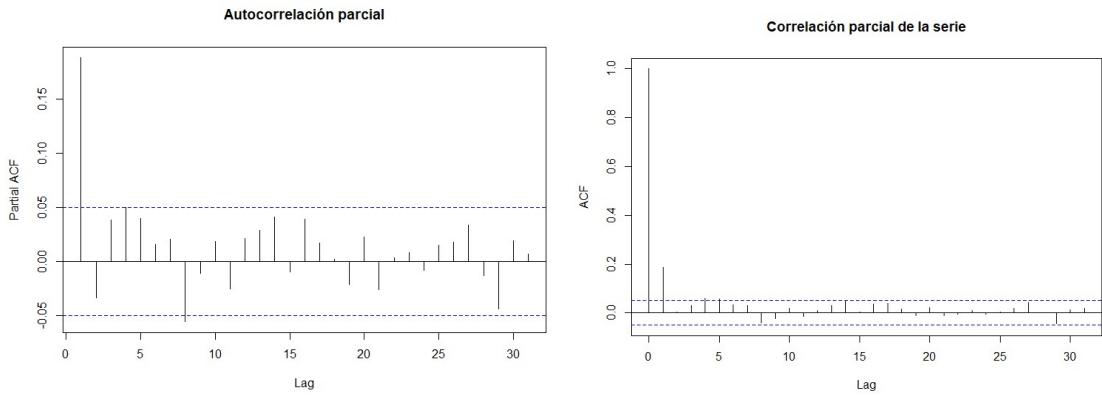
```
#Aunque lo hemos calculado con los datos de la serie original diferenciada es necesario hacerlo con los datos
#tras haber sufrido una transformación de Box-Cox
adf_results_box <- adf.test(dif_box)
print(adf_results_box)
#Resultado:
#Augmented Dickey-Fuller Test

#data: dif_box
#Dickey-Fuller = -10.797, Lag order = 11, p-value = 0.01
#alternative hypothesis: stationary

#p-value bueno rechazamos hipótesis nula y por lo tanto podemos confirmar que la serie es estacionaria.
```

Este test nos permite corroborar la estacionariedad de nuestra serie temporal.

Volvemos a representar sus gráficas de correlación y autocorrelación parcial para ver cómo cambian y usarlas posteriormente para poder determinar los coeficientes p y q de modelo que se ajuste mejor a los datos:



Una vez realizados estos cambios y representaciones procedemos a establecer el modelo que capture mejor los datos de mi serie temporal.

2. Selección de modelo:

Vemos que podemos seleccionar entre dos tipos de modelo ARMA (p, q) o un ARIMA (p, d, q).

Vamos a decantarnos por un ARIMA (p, d, q), pues hemos tenido que diferenciar una vez la serie por lo tanto $d=1$. Para estimar los otros parámetros debemos analizar los gráficos anteriores del ACF y el PACF, los cuales nos pueden mostrar los valores de q y p respectivamente.

Analizando los gráficos anteriores vemos que en cuanto al gráfico de autocorrelación parcial el valor de p es 1 y con respecto al grafico de correlación parcial vemos que el valor de q puede ser 1 u 2.

Por lo tanto, consecuentemente podemos predecir que el modelo puede ser una ARIMA (1,1,1) o ARIMA (1,1,2).

Una vez realizado este análisis vamos a emplear la función auto.arima para terminar de aclarar qué modelo captura mejor los datos:

```
auto.arima(st_eth, trace = TRUE)
best_model <- auto.arima(st_eth)
best_model
#best_model será el que posee un AIC Corregido menor, el cual es:
#ARIMA(2,1,2) with drift : 16616.54
#ARIMA(0,1,0) with drift : 16634.04
#ARIMA(1,1,0) with drift : 16611.87
#ARIMA(0,1,1) with drift : 16611.1
#ARIMA(0,1,0) : 16632.88
#ARIMA(1,1,1) with drift : 16613.88
#ARIMA(0,1,2) with drift : 16612.92
#ARIMA(1,1,2) with drift : Inf
#ARIMA(0,1,1) : 16609.76
#ARIMA(1,1,1) : 16612.54
#ARIMA(0,1,2) : 16611.57
#ARIMA(1,1,0) : 16610.51
#ARIMA(1,1,2) : Inf
```

```

ARIMA(0,1,1) : 16617.75
Best model: ARIMA(0,1,1)

```

Podemos observar que los modelos que yo he deducido están presentes como posibles modelos, pero el modelo que posee un AIC menor y por lo tanto es el mejor modelo es ARIMA (0,1,1)

```

Series: st_eth
ARIMA(0,1,1)

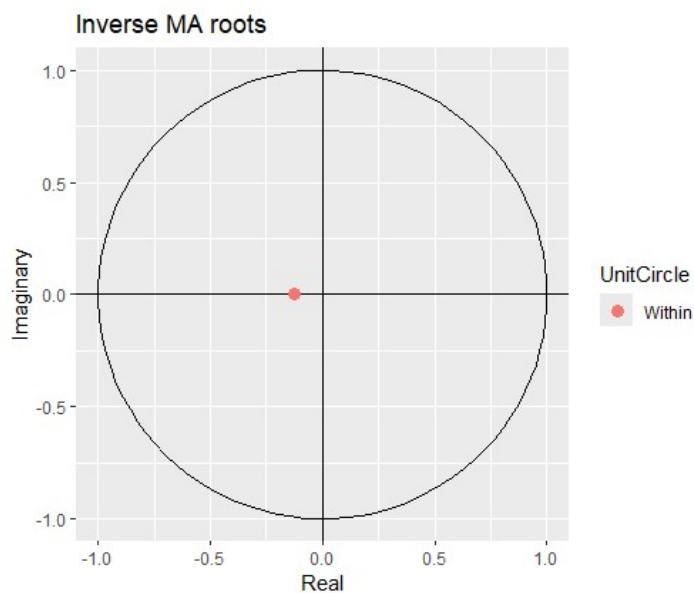
Coefficients:
    ma1
    0.1268
  s.e.  0.0250

sigma^2 = 2858: log likelihood = -8306.87
AIC=16617.74  AICc=16617.75  BIC=16628.42

```

Podemos representar las raíces del modelo en la circunferencia compleja. En este caso serán las raíces del proceso de medias móviles MA(q) pues q=1 y p=0:

```
autoplot(best_model)
```



Una vez hemos seleccionado el modelo ARIMA (0,1,1) vamos a verificar todos los supuestos de los residuos para ver cómo se comportan y si son ruido blanco:

Vemos un resumen del error cuadrático medio y otras medidas estadísticas importantes.

```
Series: st_eth
ARIMA(0,1,1)

Coefficients:
    ma1
    0.1268
s.e.  0.0250

sigma^2 = 2858: log likelihood = -8306.87
AIC=16617.74   AICc=16617.75   BIC=16628.42

Training set error measures:
      ME      RMSE      MAE      MPE      MAPE      MASE      ACF1
Training set 1.115953 53.42657 24.64246 0.03961969 2.883131 0.9889624 0.001075261
```

Verificamos la **normalidad** de los residuos empleando el test de Jarque Bera sobre los residuos del modelo:

```
#Verificamos la normalidad de los residuos
jarque.bera.test(best_model$residuals)
# Resultado
#> Jarque Bera Test
#> data: best_model$residuals
#> X-squared = 1411.9, df = 2, p-value < 2.2e-16
#Podemos rechazar la hipótesis nula y podemos afirmar que los residuos se distribuyen con una una distibución normal
```

Probando así que los residuos se distribuyen con una distribución normal

Probamos la **incorrelación de los residuos** y la **homocedasticidad de los mismos**:

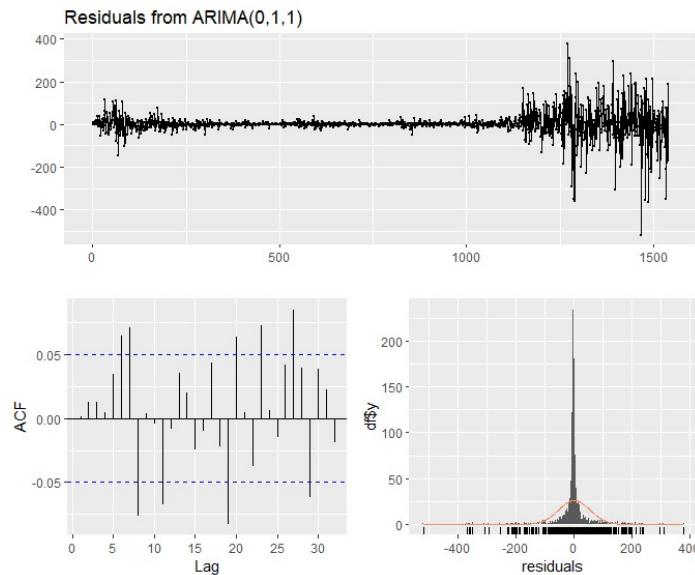
```
#Verificamos la incorrelación de los residuos
Box.test(best_model$residuals,lag=2, type = "Ljung-Box")
#Box-Ljung test
# data: best_model$residuals
#X-squared = 0.0046287, df = 2, p-value = 0.9977
#El no cumplir esta propiedad nos hace planteranos que el modelo pueda ser otro

#Homocedasticidad de los residuos:
Box.test(best_model$residuals^2,lag=2,type = "Ljung-Box")
#Box-Ljung test

#data: best_model$residuals^2
#X-squared = 7.4182, df = 2, p-value = 0.0245
```

Vemos que los residuos por el Box.test sobre los residuos al cuadrado no presentan homocedasticidad de forma general, pero por el mismo test sobre los residuos sin elevarlos al cuadrado vemos que están relacionados entre sí. La ausencia de la homocedasticidad de la varianza de los residuos en nuestro caso es positiva, pues implica que la varianza de los residuos cambia su valor a lo largo del análisis y presenta heterocedasticidad de la varianza.

Todo esto se puede reducir a implementar únicamente la función `Checkresiduals` sobre nuestro modelo (`best_model`):



```
> checkresiduals(best_model)

Ljung-Box test

data: Residuals from ARIMA(0,1,1)
Q* = 25.878, df = 9, p-value = 0.002139

Model df: 1. Total lags used: 10
```

Este comando nos devuelve todas las gráficas y el resultado del Ljung-Box Test el cual devuelve un p-value que podemos rechazar y suponer la incorrección de los residuos.

Como resultado general podemos concluir que los datos no son ruido blanco porque no cumplen todas las hipótesis necesarias y pueden tener heterocedasticidad condicional.

El siguiente paso es determinar si el modelo posee heterocedasticidad condicional.

Heterocedasticidad condicional:

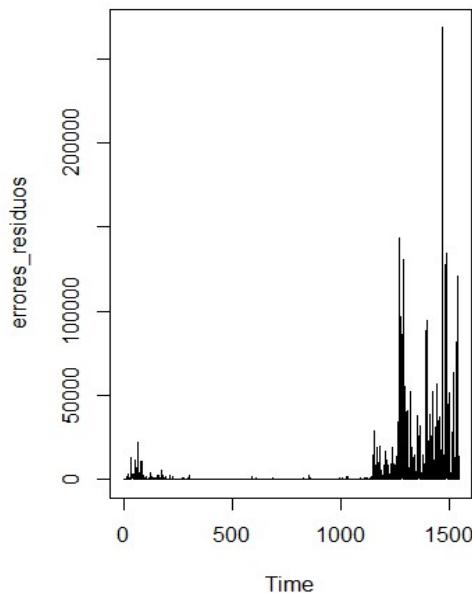
Vamos a calcular el error al cuadrado de los residuos de nuestro modelo para posteriormente estudiar si hay presencia de heterocedasticidad condicional:

```

errores_residuos <- best_model$residuals ^2
plot(errores_residuos, main = "Representación residuos al cuadrado ")

```

Representación residuos al cuadrado



Podemos observar en esta gráfica que la varianza de los residuos al cuadrado no es constante, por lo tanto, existe heterocedasticidad condicional.

Para continuar con la verificación vamos a construir una regresión lineal con los errores al cuadrado para poder ver si hay presencia de efecto ARCH o GARCH en los residuos en función del p-value y el R^2 (bondad de ajuste)

```

Regresion_1 = dynlm(errores_residuos ~ L(errores_residuos))
summary(Regresion_1)

> Regresion_1 = dynlm(errores_residuos ~ L(errores_residuos))
> summary(Regresion_1)

Time series regression with "ts" data:
Start = 2, End = 1540

Call:
dynlm(formula = errores_residuos ~ L(errores_residuos))

Residuals:
    Min     1Q Median     3Q    Max 
-23875 -2526 -2508 -2171 263735 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2.525e+03 3.397e+02 7.435 1.73e-13 ***
L(errores_residuos) 1.158e-01 2.534e-02 4.571 5.24e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13020 on 1537 degrees of freedom
Multiple R-squared:  0.01341, Adjusted R-squared:  0.01277 
F-statistic: 20.9 on 1 and 1537 DF,  p-value: 5.235e-06

```

El p-value y el R-Square adjusted (bondad de ajuste) indican que existe presencia de efecto ARCH en los residuos, es decir hay presencia de

autocorrelación parcial en los residuos. Esto es algo que verificamos anteriormente y era previsible obtener este resultado.

Para verificar la presencia de efecto ARCH en los residuos podemos emplear la prueba del Multiplicador de Lagrange, el cual es el siguiente test de hipótesis:

H0: No hay efectos ARCH si el p-value es mayor que 0.05

H1: Sí hay efectos ARCH si el p-value es menor que 0.05

```
# Realizar la prueba del Multiplicador de Lagrange
arch_test_result <- ArchTest(best_model$residuals, lag = 1) # lag = 1 es un valor común, puedes ajustarlo según convenga

# Ver los resultados de la prueba
print(arch_test_result)

#ARCH LM-test; Null hypothesis: no ARCH effects

#data: best_model$residuals
#Chi-squared = 20.643, df = 1, p-value = 5.533e-06

#El p-value es inferior a 0.05 por lo tanto hay efecto ARCH con un rezago, es decir existe volatilidad en el primer rezago.
```

Conclusión: Al ser p-value: 5.235e-06 aceptamos la H1 y por lo tanto podemos decir que hay volatilidad condicional en el primer rezago.

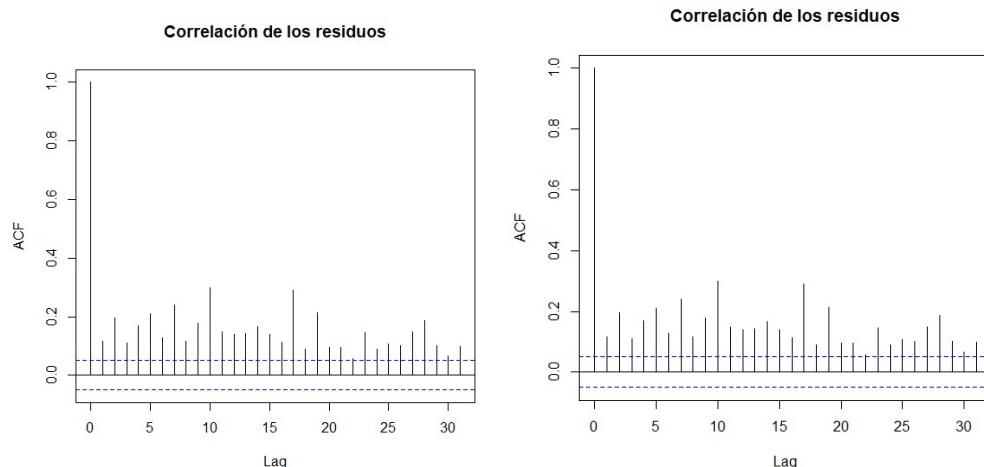
Una vez que hemos probado que hay efecto ARCH vamos a analizar la correlación y autocorrelación parcial de los residuos al cuadrado.

He implementado dos opciones:

1. Con los comandos usuales:

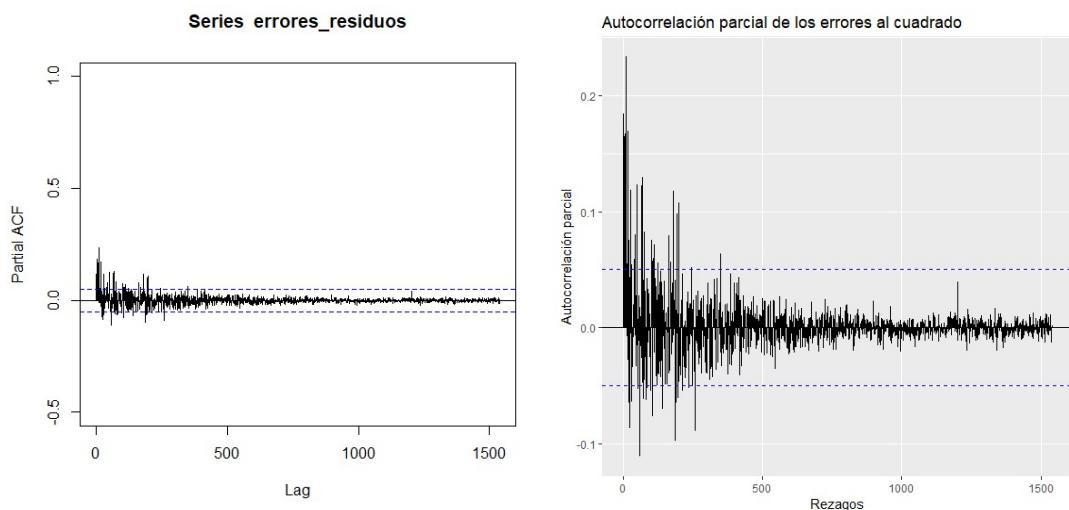
```
#opción 1
corr_resid <- acf(errores_residuos, main = "Correlación de los residuos")
autocorr_resid <- pacf(errores_residuos, main = "Autocorrelación Parcial")

#Se salen ambos de sus límites lo que indica que hay ruido blanco
```



2. Con la función autoplot, la cual a mi criterio hace más visual los resultados

```
#opción 2
autoplot(acf(errores_residuos, lag.max = 5394, ylim = c(-0.5,1))) +
  labs(title = "Autocorrelación de los errores al cuadrado") +
  xlab("Rezagos") +
  ylab("Autocorrelación")
 
autoplot(pacf(errores_residuos, lag.max = 5394, ylim = c(-0.5,1))) +
  labs(title = "Autocorrelación parcial de los errores al cuadrado") +
  xlab("Rezagos") +
  ylab("Autocorrelación parcial")
#Se salen ambos de sus límites lo que indica que hay ruido blanco
```



En ambas implementaciones podemos observar que tanto en la correlación y autocorrelación parcial las gráficas llegan a salir del límite establecidos, podemos decir que hay ruido blanco. En otras palabras, hay heterocedasticidad en la varianza.

3. Generación del modelo GARCH:

El siguiente paso será generar nuestro modelo GARCH de los residuos a partir del modelo ARIMA (0,1,1) ya definido. Esto lo hacemos para poder manejar el efecto ARCH de los residuos y así poder gestionar la heterocedasticidad condicional de los residuos del modelo.

Tenemos diferentes modelos para poder implementar, en mi caso analizando mi serie temporal, voy a optar por seleccionar un modelo GARCH (1,1), el cual se implementa en modelos econométricos y es uno de los más comunes.

Antes de construir el modelo, voy a estimar el valor del modelo ARCH(q), esto lo haré mediante el uso del ya empleado test del multiplicador de Lagrange:

```

results <- sapply(1:10, function(q) {
  test <- ArchTest(best_model$residuals, lags = q)
  return(c(q = q, p_value = test$p.value))
})
|
# Convertir los resultados a un data frame
results_df <- as.data.frame(t(results))
colnames(results_df) <- c("q", "p_value")

# Ver los resultados
print(results_df)

# De estos resultados debemos tomar el q que minimice el p-value y cumpla que p-value sea < 0.05
# q = 1

> print(results_df)
      q     p_value
1   1 5.533318e-06
2   2 2.183612e-16
3   3 2.808930e-17
4   4 2.462446e-21
5   5 6.828215e-29
6   6 6.250269e-29
7   7 5.277471e-37
8   8 1.185765e-36
9   9 1.738701e-37
10 10 1.719012e-53

```

De esos resultados debemos tomar aquel q que minimice el p-value y que sea menos que 0.05, por lo tanto, q=1.

A continuación, vamos construir el modelo GARCH (1,1) y el modelo EGARCH (1,1) y vamos a comparar sus resultados para ver cuál de los dos será el adecuado, es decir cuál de las dos capturas mejore la volatilidad condicional del modelo.

Simulación modelo GARCH (1,1):

```

# Hacemos la simulación para un modelo GARCH(1,1)
# Definimos el modelo GARCH(1,1)
modelo_garch <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                           mean.model = list(armaOrder = c(0, 1), include.mean = TRUE),
                           distribution.model = "norm") # Distribución normal
# Ajustamos el modelo GARCH(1,1)
fit_garch <- ugarchfit(spec = modelo_garch, data = best_model$residuals)

# Mostramos el resumen del modelo ajustado
show(fit_garch)

# Mostramos los coeficientes del modelo GARCH (1,1)
coef(fit_garch)

```

```

> show(fit_garch)
*-----*
*      GARCH Model Fit      *
*-----*

Conditional Variance Dynamics
-----
GARCH Model   : sGARCH(1,1)
Mean Model    : ARFIMA(0,0,1)
Distribution   : norm

Optimal Parameters
-----
Estimate Std. Error t value Pr(>|t|)
mu     -0.043902  0.250967 -0.17493 0.861132
ma1     0.082358  0.029083  2.83187 0.004628
omega   1.083863  0.279593  3.87657 0.000106
alphal  0.087999  0.010344  8.50755 0.000000
beta1   0.911001  0.010353 87.99397 0.000000

Robust Standard Errors:
Estimate Std. Error t value Pr(>|t|)
mu     -0.043902  0.259371 -0.16926 0.865588
ma1     0.082358  0.033069  2.49050 0.012757
omega   1.083863  0.580269  1.86786 0.061781
alphal  0.087999  0.020411  4.31133 0.000016
beta1   0.911001  0.021033 43.31208 0.000000

LogLikelihood : -6721.641

Information Criteria
-----
Akaike       8.7359
Bayes        8.7532
Shibata      8.7359
Hannan-Quinn 8.7423

Weighted Ljung-Box Test on Standardized Residuals
-----
statistic p-value
Lag[1]          1.063  0.3024
Lag[2*(p+q)+(p+q)-1][2] 1.095  0.6815
Lag[4*(p+q)+(p+q)-1][5] 3.664  0.2841
d.o.f=1
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
-----
statistic p-value
Lag[1]          0.5422 0.4615
Lag[2*(p+q)+(p+q)-1][5] 0.9358 0.8740
Lag[4*(p+q)+(p+q)-1][9] 2.3119 0.8647
d.o.f=2

Weighted ARCH LM Tests
-----
Statistic Shape Scale P-Value
ARCH Lag[3]    0.3022 0.500 2.000 0.5825
ARCH Lag[5]    0.5762 1.440 1.667 0.8609
ARCH Lag[7]    2.1289 2.315 1.543 0.6900

Nyblom stability test
-----
Joint Statistic: 0.7943
Individual Statistics:
mu     0.20633
ma1    0.08688
omega  0.03066
alphal 0.23512
beta1  0.15929

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic: 1.28 1.47 1.88
Individual Statistic: 0.35 0.47 0.75

Sign Bias Test
-----
t-value prob sig
Sign Bias      0.3110 0.7558
Negative Sign Bias 0.1323 0.8947
Positive Sign Bias 1.0539 0.2921
Joint Effect    1.8144 0.6118

Adjusted Pearson Goodness-of-Fit Test:
-----
group statistic p-value(g-1)
1    20      244.4    4.277e-41
2    30      263.4    1.267e-39
3    40      277.2    1.095e-37
4    50      299.2    1.288e-37

```

```
> coef(fit_garch)
      mu        ma1        omega      alpha1      beta1
-0.04390241  0.08235828  1.08386286  0.08799917  0.91100083
```

Cuando ejecutamos el código que define el modelo GARCH (1,1) y lo ejecutamos obtenemos los resultados adjuntados anteriormente.

Vamos a analizar estos resultados, esto lo vamos a hacer respetando el orden del código que hemos obtenido:

Los coeficientes obtenidos en la primera tabla de resultados son

	Estimate	Std. Error	t value	Pr(> t)
mu	-0.043902	0.250967	-0.17493	0.861132
ma1	0.082358	0.029083	2.83187	0.004628
omega	1.083863	0.279593	3.87657	0.000106
alpha1	0.087999	0.010344	8.50755	0.000000
beta1	0.911001	0.010353	87.99397	0.000000

Son los que podemos verlos de forma más clara con el comando `coef(fit_garch)`. De estos vamos a fijarnos en el valor de `alpha1` y `beta1`.

- $\text{Alpha1} = 0.0879917$ es un valor muy pequeño que nos indica que, aunque la volatilidad depende de los errores pasados la persistencia de este efecto no es tan fuerte en valores posteriores.
- $\text{Beta1} = 0.911001$ indica que la varianza futura está muy influenciada por la varianza pasada. Es decir, hay una gran persistencia de la volatilidad.

Lo que nos permite afirmar que el modelo creado captura la dinámica de la volatilidad con alta persistencia, es decir, la volatilidad futura será igual que la volatilidad de los datos ya estudiados.

Estudiamos los valores del AIC y BIC:

```
LogLikelihood : -6721.641
Information Criteria
-----
Akaike       8.7359
Bayes        8.7532
Shibata      8.7359
Hannan-Quinn 8.7423
```

Vemos que los valores de AIC y BIC son muy bajos, pues $AIC = 8.7359$ y $BIC = 8.7532$, lo cual es señal del buen ajuste del modelo.

Estudiamos si hemos conseguido capturar la volatilidad del modelo y eso lo podemos ver examinando los siguientes resultados:

Weighted ARCH LM Tests				
	Statistic	Shape	Scale	P-Value
ARCH Lag[3]	0.3022	0.500	2.000	0.5825
ARCH Lag[5]	0.5762	1.440	1.667	0.8609
ARCH Lag[7]	2.1289	2.315	1.543	0.6900

Los p-value = [0.5825,0.8609,0.6900] nos permiten afirmar que no hay evidencias de efecto ARCH que no haya sido modelado, es decir, nos permite afirmar que hemos capturado toda la volatilidad del modelo.

Por último, para verificar que el modelo se ajusta bien a los datos vamos a examinar el valor de dos coeficientes la bondad de ajuste, que se corresponde con el coeficiente de Pearson ajustado. El valor de este es de 4.277e-4

1

Podemos verlo en el siguiente trozo de código:

Adjusted Pearson Goodness-of-Fit Test:			
group	statistic	p-value(g-1)	
1	20	244.4	4.277e-41
2	30	263.4	1.267e-39
3	40	277.2	1.095e-37
4	50	299.2	1.288e-37

Este resultado tan bajo a diferencia de los otros valores anteriores indica que puede que el modelo no se ajuste del todo bien a los datos. Por lo tanto, esto nos hace plantearnos la posibilidad de tener que seleccionar otro modelo para capturar la volatilidad.

Siguiendo la gráfica de la correlación parcial de los datos expuesta anteriormente vamos a tomar $p = 2$, es decir, vamos a probar a emplear un modelo GARCH (2,1).

```
# Definimos el modelo GARCH(2,1)
modelo_garch <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(2, 1)),
                           mean.model = list(armaOrder = c(0, 1), include.mean = TRUE),
                           distribution.model = "norm") # Distribución normal
# Ajustamos el modelo GARCH(1,1)
fit_garch <- ugarchfit(spec = modelo_garch, data = best_model$residuals)

# Mostramos el resumen del modelo ajustado
show(fit_garch)
```

Este modelo posee unos resultados similares al GARCH (1,1), pero con la cualidad de que posee un coeficiente de Pearson ajustado más pequeño. Sus parámetros son:

```

> coef(fit_garch)
      mu        ma1        omega     alpha1     alpha2       beta1
-4.407410e-02 8.209790e-02 1.076175e+00 8.756619e-02 1.393531e-07 9.114337e-01

Information Criteria
-----
Akaike      8.7377
Bayes       8.7585
Shibata     8.7377
Hannan-Quinn 8.7454

Weighted Ljung-Box Test on Standardized Residuals
-----
statistic p-value
Lag[1]          1.082 0.2982
Lag[2*(p+q)+(p+q)-1][2] 1.114 0.6699
Lag[4*(p+q)+(p+q)-1][5] 3.679 0.2813
d.o.f=1
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
-----
statistic p-value
Lag[1]          0.5446 0.4605
Lag[2*(p+q)+(p+q)-1][8] 1.9977 0.8592
Lag[4*(p+q)+(p+q)-1][14] 3.7114 0.9061
d.o.f=3

Weighted ARCH LM Tests
-----
Statistic Shape Scale P-Value
ARCH Lag[4]    0.2243 0.500 2.000  0.6358
ARCH Lag[6]    0.5923 1.461 1.711  0.8659
ARCH Lag[8]    2.5755 2.368 1.583  0.6255

Nyblom stability test
-----
Joint Statistic: 1.1034
Individual Statistics:
mu      0.20663
ma1     0.08578
omega   0.03068
alpha1  0.23449
alpha2  0.23407
beta1   0.15905

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      0.3122 0.7550
Negative Sign Bias 0.1300 0.8966
Positive Sign Bias 1.0524 0.2928
Joint Effect    1.8107 0.6126

Adjusted Pearson Goodness-of-Fit Test:
-----
group statistic p-value(g-1)
1     20      245.6    2.393e-41
2     30      266.0    3.991e-40
3     40      280.4    2.829e-38
4     50      302.6    3.092e-38

```

Si resumimos los valores que nos interesan estudiar tendremos:

Alpha1= 8.756619e-02

beta1= 9.114337e-01

AIC = 8.7377

BIC = 8.7585

R^2 = 2.393e-41

p-values LM test = [0.6358,0.8659,0.6255]

Vamos a realizar una tabla para comparar los coeficientes de los modelos implementados

Coeficientes	GARCH (1,1)	GARCH(2,1)
Alpha1	0.0879917	8.756619e-02
Beta1	0.911001	9.114337e-01
AIC	8.7359	8.7377
BIC	8.7532	8.7585
R^2	4.277e-41	2.393e-41
p-value LM test	[0.5825,0.8609,0.6900]	[0.6358,0.8659,0.6255]

Análisis de la tabla: Los coeficientes del modelo GARCH (2,1) son similares a los del modelo GARCH (1,1), pero en este caso el coeficiente de Pearson y los valores de alpha1 y beta1 son menores. Esto nos permite entender que el modelo al tener un coeficiente de Pearson ajustado mucho más pequeño va a indicar que el modelo GARCH (2,1) se ajusta mucho mejor a los datos y captura mucha más volatilidad de los datos que el modelo GARCH (1,1). Por lo tanto, podemos decantarnos por el modelo GARCH (2,1) como el modelo seleccionado para capturar la volatilidad de los residuos de la serie.

Podríamos también probar con un modelo EGARCH (1,1) pero obtenemos los siguientes resultados:

```

#Modelo EGARCH
modelo_egarch <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(1, 1)),
                           mean.model = list(armaOrder = c(0, 0), include.mean = TRUE),
                           distribution.model = "norm") # Distribución normal
# Ajustar el modelo EGARCH(1,1) a los datos de mi serie temporal
fit_egarch <- ugarchfit(spec = modelo_egarch, data = best_model$residuals)

# Mostrar el resumen del modelo ajustado
show(fit_egarch)

# Mostramos los coeficientes del modelo GARCH (1,1)
coef(fit_egarch)

Warning message:
In .egarchfit(spec = spec, data = data, out.sample = out.sample, :
ugarchfit-->warning: solver failed to converge.

```

Obtengo este error que me indica que el modelo es muy complejo para usar esta variante de GARCH, pues falla la convergencia, pese a que he intentado ponerle una tolerancia al modelo sigue sin poder ajustar bien los datos y por lo tanto no puede converger. Eso se traduce en que no puedo analizar los coeficientes de los resultados, por lo tanto, este modelo no es óptimo.

4. Predicción del modelo GARCH

Por último, tras capturar la volatilidad con nuestro modelo GARCH (2,1) y validar el modelo vamos a realizar una predicción del precio de la criptomonedra Ethereum con respecto al euro treinta días después desde la fecha del último dato recopilado, es decir un mes después del 28/01/2022

```

ug_forecast = ugarchforecast(fit_garch_2, n.ahead = 30)

ug_forecast

```

0-roll forecast [T0=1540-01-01]:		
	Series	Sigma
T+1	-0.26861	126.3
T+2	-0.04407	126.2
T+3	-0.04407	126.1
T+4	-0.04407	126.1
T+5	-0.04407	126.0
T+6	-0.04407	126.0
T+7	-0.04407	125.9
T+8	-0.04407	125.9
T+9	-0.04407	125.8
T+10	-0.04407	125.7
T+11	-0.04407	125.7
T+12	-0.04407	125.6
T+13	-0.04407	125.6
T+14	-0.04407	125.5
T+15	-0.04407	125.4
T+16	-0.04407	125.4
T+17	-0.04407	125.3
T+18	-0.04407	125.3
T+19	-0.04407	125.2
T+20	-0.04407	125.1
T+21	-0.04407	125.1
T+22	-0.04407	125.0
T+23	-0.04407	125.0
T+24	-0.04407	124.9
T+25	-0.04407	124.9
T+26	-0.04407	124.8
T+27	-0.04407	124.7
T+28	-0.04407	124.7
T+29	-0.04407	124.6
T+30	-0.04407	124.6

Análisis de los resultados:

Vemos que la columna serie nos devuelve el valor estimado de la media del precio de la criptomoneda Ethereum con respecto al euro en los próximos 30 días vemos que la media de la serie cambia a lo largo de las iteraciones, en este caso decrece en la segunda iteración.

En la primera iteración el valor es de -0.26861 y en la segunda iteración es de -0.04407. A partir de la segunda iteración se mantiene el valor fijo del precio, esto nos sugiere que la serie no cambia a lo largo del tiempo, concretamente podemos decir que el precio de las criptomonedas Ethereum no cambia prácticamente a lo largo de un mes, llegando a la conclusión de que el precio en el periodo de un mes es estable en cuanto a la predicción de su media.

La columna sigma representa la volatilidad condicional. De esta columna vemos que se mantienen casi constante, vemos que la poca variación de la volatilidad es señal del grado de variabilidad o de incertidumbre de los precios. En nuestro caso vemos como la tendencia de la volatilidad de los precios es baja disminuye de un 126.3 a un 124.6 en cuestión de 30 días.

Esto nos permite deducir que las fluctuaciones en el precio de las monedas Ethereum con respecto al euro disminuirá y provocarán que se estabilice el precio con el tiempo.

Conclusión: El precio del Ethereum con respecto al euro se va a volver menos volátil en un periodo futuro de 30 días a partir de la última estimación tomada para el análisis de la serie temporal. Además, la tendencia del precio de la criptomoneda Ethereum con respecto al euro es decreciente en febrero de 2022

Representación gráfica:

Primero convertimos los datos de la predicción en un data frame, para poder usarlo en la creación de la gráfica

```
#Convertimos los datos de la predicción en un data frame, para poder usarlo en la creación de la gráfica
# Coeficientes de la predicción
valores_pred <- fitted(ug_forecast)

volatilidad_predicha <- sigma(ug_forecast)

# Creamos un rango de fechas para las predicciones
# La última fecha en la serie histórica es "2022-01-28"
comienzo <- as.Date("2022-01-28")
datos_forecast <- seq(comienzo + 1, by = "day", length.out = length(valores_pred))

# Combinamos los resultados en un DataFrame
tabla_datos_prediccion <- data.frame(
  Fecha = datos_forecast,
  Prediccion = as.numeric(valores_pred),
  Volatilidad = as.numeric(volatilidad_predicha)
)
```

A continuación, empleamos estos datos para representar la serie original junto con los valores predichos:

```
# Graficamos la serie histórica ETH-EUR con los valores de la predicción
st_eth <- ts(datos_seleccionados$High, frequency = 1)

plot(st_eth,
  type = "l",
  xaxt = "n", main = "Predicción serie de cambio ETH-EUR", xlab = "Tiempo",
  ylab = "Precio")|>

# Etiquetas personalizadas del eje X
axis(1, at = seq(1, length(datos_seleccionados$date), by = round(length(datos_seleccionados$date) / 10)),
  labels = datos_seleccionados$date[seq(1, length(datos_seleccionados$date), by = round(length(datos_seleccionados$date) / 10))])

# Fechas de la predicción
comienzo <- as.Date(tail(datos_seleccionados$date), 1) # Última fecha histórica
datos_forecast <- seq(comienzo + 1, by = "day", length.out = length(tabla_datos_prediccion$Prediccion))

# Predicciones
lines(x = length(st_eth) + 1:length(tabla_datos_prediccion$Prediccion),
  y = tabla_datos_prediccion$Prediccion,
  col = "red",
  lwd = 2) # Línea roja para las predicciones

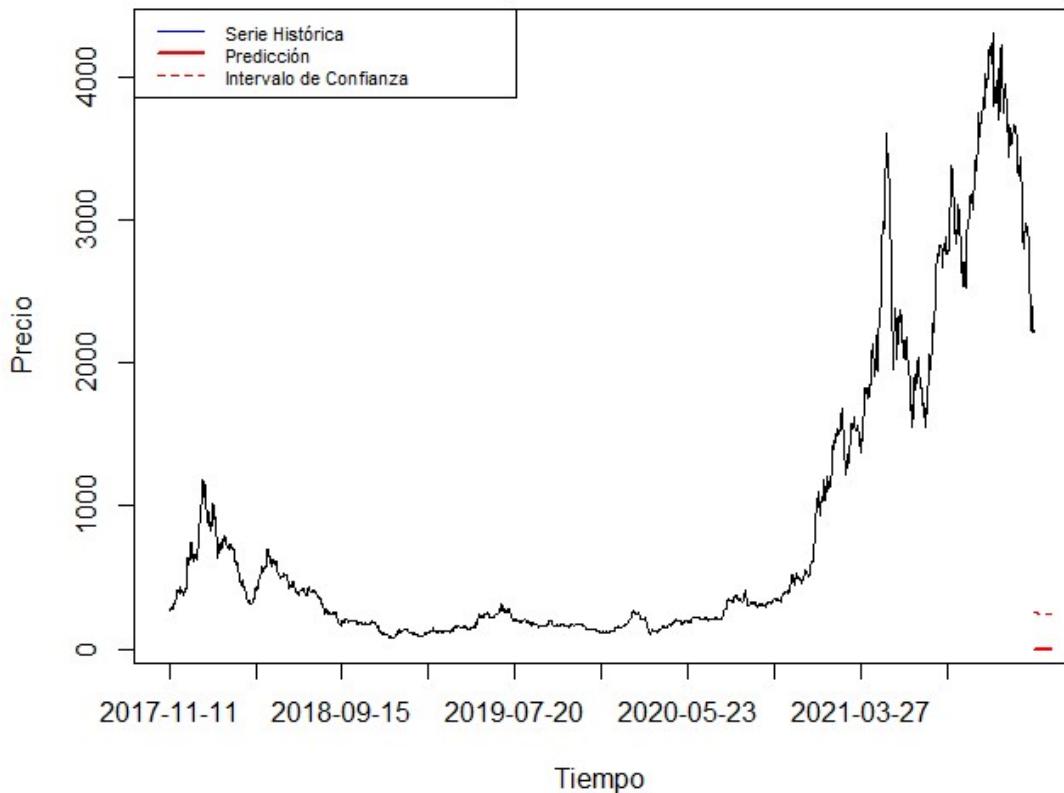
# Intervalo de confianza
lines(x = length(st_eth) + 1:length(tabla_datos_prediccion$Prediccion),
  y = tabla_datos_prediccion$Prediccion + 2 * tabla_datos_prediccion$Volatilidad,
  col = "red",
  lty = 2)

lines(x = length(st_eth) + 1:length(tabla_datos_prediccion$Prediccion),
  y = tabla_datos_prediccion$Prediccion - 2 * tabla_datos_prediccion$Volatilidad,
  col = "red",
  lty = 2)

# Leyenda
legend("topleft",
  legend = c("Serie Histórica", "Predicción", "Intervalo de Confianza"),
  col = c("blue", "red", "red"),
  lty = c(1, 1, 2),
  lwd = c(1, 2, 1),
  cex = 0.7)
```

Lo que podemos deducir de la predicción es como ya hemos visto al analizar las predicciones anteriores que el precio del Ethereum con respecto al euro en el mes de febrero de 2022 va a decaer significativamente.

Predicción serie de cambio ETH-EUR



Lo que podemos deducir de la predicción es como ya hemos visto al analizar las predicciones anteriores que el precio del Ethereum con respecto al euro en el mes de febrero de 2022 va a decaer significativamente.

Adjunto a continuación todo el código empleado con las explicaciones y notas sobre el mismo:

```

# Autor: Rubén Garrido Hidalgo
# Asignatura: Series temporales II
# Grado: Matemáticas
# Fecha: 16/01/2025
#-----
#Importamos las librerías que vamos a usar
library(forecast)
library(tseries)
library(FinTS)
library(dynlm)
library(ggplot2)
library(rugarch)

# Librerías que he tenido que instalar porque no las tenía instaladas previamente
#install.packages("dynlm")
#install.packages("FinTS")
#####
#####Para construir nuestra serie temporal vamos a tomar datos financieros,
#los cuales vamos a extraer de la página Yahoo Finance.
#Para construir nuestra serie temporal vamos a tomar datos financieros,
#los cuales vamos a extraer de la página Yahoo Finance.
datos <- "C:/Users/Rubén Garrido/Desktop/Series Temporales II/Modelos ARCH y GARCH/ETH-EUR.csv"
cambio <- read.csv(datos)
head (cambio)
str(cambio)
show(cambio)

#Datos(descripción) Tenemos los datos del precio de la criptomoneda Ethereum
#con respecto al euro. En mi caso tomaré las columnas de cambio$date y cambio$High,
#con esas dos estudiaria que modelo se ajusta mejor a los datos

datos_seleccionados <- cambio[, c("Date", "High")]

#El primer paso para analizar nuestra serie temporal es determinar el tipo de modelo que sigue la serie si es
#ARMA(p,q) o ARIMA(p,d,q)

#Por lo tanto vamos a determinar el modelo de nuestra serie temporal,
#para ello primero realizamos una representación gráfica de los datos:

st_eth <- ts(datos_seleccionados$High, frequency = 1)
plot(st_eth,
      type = "l",
      xaxt = "n",
      main = "Representación serie de cambio ETH-EUR",
      xlab = "Tiempo",
      ylab = "Precio")
axis(1, at = seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10)),
     labels = datos_seleccionados>Date[seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10))])

#####
#Una vez representada la serie temporal vemos que no va a ser estacionaria:
#Dicker Fuller para verificar si mi serie es estacionaria
adf_results <- adf.test(st_eth)
print(adf_results)

#Augmented Dickey-Fuller Test

#data: st_eth
#Dickey-Fuller = -1.6696, Lag order = 11, p-value = 0.7182
#alternative hypothesis: stationary

#El p-value es mayor que 0.05 y no podemos rechazar la hipótesis nula.

#####
#Realizamos transformaciones para que sea estacionaria en varianza y media
#####
#Vamos a diferenciarla la intentar eliminar la estacionalidad:
dif_eth <- diff(st_eth)
plot(dif_eth,
      type = "l",
      xaxt = "n",
      main = "Representación serie de cambio ETH-EUR",
      xlab = "Tiempo",
      ylab = "Precio")
axis(1, at = seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10)),
     labels = datos_seleccionados>Date[seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10))])

#####
#Realizamos la representación gráfica de la autocorrelación y la autocorrelación parcial
#para estudiar la estacionariedad de la serie
auto_parcial <- pacf(st_eth, main = "Autocorrelación Parcial de la serie" )
correla_parcial <- acf(st_eth, main = "Correlación Parcial ")
#Sería recomendable realizar dos intervenciones, pues en varianza no es estacional.
#####
#Realizamos una transformación de BoxCox para hacer que la serie sea estacionaria en varianza
lambda_optimo <- BoxCox.lambda(datos_seleccionados$High)
print(lambda_optimo)

datos_transformados <- BoxCox(datos_seleccionados$High, lambda_optimo)
print(datos_transformados)

#Una vez realizado la transformación de BoxCox sobre los datos
#vamos a realizar la representación de la serie con esos datos
st_eth_box <- ts(datos_transformados, frequency = 1)
plot(st_eth_box)
#Diferenciamos estos datos
dif_box <- diff(st_eth_box)

```

```

plot(dif_box,
      type = "l",
      xaxt = "n",
      main = "Representación serie de cambio ETH-EUR",
      xlab = "Tiempo",
      ylab = "Precio")
axis(1, at = seq(1, length(datos_seleccionados$Date), by = round(length(datos_seleccionados$Date) / 10)),
     labels = datos_seleccionados$Date[seq(1, length(datos_seleccionados$Date), by = round(length(datos_seleccionados$Date) / 10))])

#Tras relizar la transformación BoxCox sobre los datos hemos podido ver que al diferenciar esa serie ya tenemos una serie que
#es estacionaria en varianza y en media,
#a continuación aplicamos Dicker-Fuller para verificar la estacionariedad de la serie
#####
#Volvemos a realizar el test de Dicker Fuller para verificar la estacionariedad de la serie
adf_results <- adf.test(dif_eth)
print(adf_results)

#Augmented Dickey-Fuller Test

#data: dif_eth
#Dickey-Fuller = -11.464, Lag order = 11, p-value = 0.01
#alternative hypothesis: stationary

#EL p-value es < 0.05 por lo tanto rechazamos la hipótesis nula y podemos decir que la serie es estacionaria.
auto_parcial <- pacf(dif_eth)
correla_parcial <- acf(dif_eth)
#####
#Aunque lo hemos calculado con los datos de la serie original diferenciada es necesario hacerlo con los datos
#tras haber sufrido una transformación de Box-Cox
adf_results_box <- adf.test(dif_box)
print(adf_results_box)
#Resultado:
#Augmented Dickey-Fuller Test

#data: dif_box
#Dickey-Fuller = -10.797, Lag order = 11, p-value = 0.01
#alternative hypothesis: stationary

#p-value bueno rechazamos hipótesis nula y por lo tanto podemos confirmar que la serie es estacionaria.
#####
#podemos estudiar el acf y pacf de esta serie para poder estimar el modelo de ARIMA(p,d,q) que sigue
autoc_st_box <- acf(dif_box, main = "Correlación parcial de la serie")
pacf_st_box <- pacf(dif_box, main = "Autocorrelación parcial")
#El acf y el pacf son bastante claros y podemos ver que en el gráfico de la autocorrelación de la serie se puede observar que
#el valor de q puede ser 1 u 2 en función del lag que se sale de los límites establecidos.
#En cuanto al gráfico de autocorrelación parcial vemos que el valor de p puede ser también de 1.
#Modelo sugerido ARIMA(1,1,1)
#####
#Pese a este estudio vamos a emplear la función auto.arima

auto.arima(st_eth, trace = TRUE)
best_model <- auto.arima(st_eth)
best_model
#best_model será el que posee un AIC Corregido menor, el cual es:
#ARIMA(2,1,2) with drift : 16616.54
#ARIMA(0,1,0) with drift : 16634.04
#ARIMA(1,1,0) with drift : 16611.87
#ARIMA(0,1,1) with drift : 16611.1
#ARIMA(0,1,0) : 16632.88
#ARIMA(1,1,1) with drift : 16613.88
#ARIMA(0,1,2) with drift : 16612.92
#ARIMA(1,1,2) with drift : Inf
#ARIMA(0,1,1) : 16609.76
#ARIMA(1,1,1) : 16612.54
#ARIMA(0,1,2) : 16611.57
#ARIMA(1,1,0) : 16610.51
#ARIMA(1,1,2) : Inf

#ARIMA(0,1,1) : 16617.75

#Best model: ARIMA(0,1,1)
summary(datos_seleccionados)

autoplot(best_model) #representación de las raíces del modelo en la circunferencia compleja;
summary(best_model)
#####
#Tendremos un modelo ARIMA(0,1,1), vamos ahora a emplear un test de hipótesis para analizar
#el p-value y el Coeficiente de determinación R^2 para ver si hay efecto ARCH en los residuos.

#Estudiamos los valores de los residuos del best_model para ver el comportamiento de los mismos y así verificar el modelo
#Representamos los valores de la autocorrelación y correlación parcial
auto_parcial <- pacf(dif_eth)
correla_parcial <- acf(dif_eth)
#Medidas estadísticas importantes:
summary(best_model)
#Verificamos la normalidad de los residuos
jarque.bera.test(best_model$residuals)
# Resultado
#Jarque-Bera Test
#data: best_model$residuals
#X-squared = 1411.9, df = 2, p-value < 2.2e-16
#Podemos rechazar la hipótesis nula y podemos afirmar que los residuos se distribuyen con una distribución normal

#Verificamos la incorrelación de los residuos
Box.test(best_model$residuals, lag=2, type = "Ljung-Box")

```

```

#Box-Ljung test
# data: best_model$residuals
#X-squared = 0.0046287, df = 2, p-value = 0.9977
#El no cumplir esta propiedad nos hace planteranos que el modelo pueda ser otro

#Homocedasticidad de los residuos:
Box.test(best_model$residuals^2,lag=2,type = "Ljung-Box")
#Box-Ljung test

#data: best_model$residuals^2
#X-squared = 7.4182, df = 2, p-value = 0.0245
## Los residuos pueden parecer ruido blanco, pero pueden presentar heterocedasticidad condicional
#####
##### Todo esto se puede reducir a implementar únicamente la función Checkresiduals
checkresiduals(best_model)
#Nos devuelve todas las gráficas y el resultado del Ljung-Box Test
#el cual devuelve un p-value que podemos rechazar y suponer la incorrelación de los residuos

#####
#A continuación vamos a calcular el error al cuadrado de los resiudos de nuestro modelo para
#posteriormente estudiar si hay heterocedasticidad condicional en nuestro modelo:
errores_residuos <- best_model$residuals ^2
plot(errores_residuos, main = "Representación residuos al cuadrado ")
#La varianza no es constante existe heterocedasticidad

#A continuación vamos a construir una regresión lineal con los errores al cuadrado para poder ver si hay presencia
#de un modelo ARCH o GARCH en función del p-value y el R^2 (bondad de ajuste)

Regresion_1 = dynlm(errores_residuos ~ L(errores_residuos))

summary(Regresion_1)

#El p-value y el R-Square adjusted indican que existe presencia de efecto ARCH en los residuos,es decir hay presencia
#de autocorrelación parcial en los residuos.

#####
#Test de hipótesis multiplicador de Lagrange
#H0: No hay efectos ARCH si el p-value es mayor que 0.05
#H1: Sí hay efectos ARCH si el p-value es menor que 0.05
#Conclusión: Al ser p-value: 5.235e-06 aceptamos la H1
#####
# Realizar la prueba del Multiplicador de Lagrange
arch_test_result <- ArchTest(best_model$residuals, lag = 1) # lag = 1 es un valor común, puedes ajustarlo según convenga

# Ver los resultados de la prueba
print(arch_test_result)

#ARCH LM-test; Null hypothesis: no ARCH effects

#data: best_model$residuals
#Chi-squared = 20.643, df = 1, p-value = 5.533e-06

#El p-value es inferior a 0.05 por lo tanto hay efecto ARCH con un rezago, es decir existe volatilidad en el primer rezago.

#####
#Por lo tanto con eso hemos determinado que si hay un efecto ARCH en la varianza de los errores al cuadrado de nuestros
#residuos
#Para prosseguir con el análisis vamos a analizar la correlacion y la autocorrelación parcial de los errores
#de los residuos al cuadrado:
#opción 1
corr_resid <- acf(errores_residuos, main = "Correlación de los residuos")
autocorr_resid <- pacf(errores_residuos, main = "Autocorrelación Parcial")

#Se salen ambos de sus límites lo que indica que hay ruido blanco
#####
#opción 2
autoplot(acf(errores_residuos, lag.max = 5394, ylim = c(-0.5,1))) +
  labs(title = "Autocorrelación de los errores al cuadrado") +
  xlab("Rezagos") +
  ylab("Autocorrelación")

autoplot(pacf(errores_residuos, lag.max = 5394, ylim = c(-0.5,1))) +
  labs(title = "Autocorrelación parcial de los errores al cuadrado") +
  xlab("Rezagos") +
  ylab("Autocorrelación parcial")
#Se salen ambos de sus límites lo que indica que hay ruido blanco
#####

#Tanto en la autocorrelación y autocorrelación parcial llegan
#a salir de los límites establecidos en azul, podemos decir que hay ruido blanco,
#en otras palabras,hay heterocedacistida en la varianza
#####
#El siguiente paso será generar nuestro modelo GARCH de los residuos a partir de la ARIMA(0,1,1)
#Tengo que un modelo ARIMA(0,1,1) posee efecto ARCH en sus residuos, debo ajustar un modelo GARCH para poder manejar
#esta heterocedasticidad condicional de los residuos

#####
#Cargamos al libreria para usar el modelo GARCH, específicamente un modelo GARCH(1,1), es decir con un componente
autorregresivo de orden 1
#y con un componenete de media móvil de orden 1.
#install.packages("rugarch")
library(rugarch)

#Para estimar el posible valor de un ARCH(q) vamos s usar la prueba del multiplicador de Lagrange

```

```

# Realizar la prueba ARCH-LM para diferentes valores de q
results <- sapply(1:10, function(q) {
  test <- ArchTest(best_model$residuals, lags = q)
  return(c(q = q, p_value = test$p.value))
})

# Convertir los resultados a un data frame
results_df <- as.data.frame(t(results))
colnames(results_df) <- c("q", "p_value")

# Ver los resultados
print(results_df)

# De estos resultados debemos tomar el q que minimice el p-value y cumpla que p-value sea < 0.05
# q = 1

#A continuación ajustamos el modelo GARCH(p,q)
q <- 1

#####
#Hacemos la simulación para un modelo GARCH(1,1)
# Definimos el modelo GARCH(1,1)
modelo_garch <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                            mean.model = list(armaOrder = c(0, 1), include.mean = TRUE),
                            distribution.model = "norm") # Distribución normal
# Ajustamos el modelo GARCH(1,1)
fit_garch <- ugarchfit(spec = modelo_garch, data = best_model$residuals)

# Mostramos el resumen del modelo ajustado
show(fit_garch)

# Mostramos los coeficientes del modelo GARCH (1,1)
coef(fit_garch)

# Para poder decir que el modelo está bien ajustado debemos fijarnos en varios datos del resultado.

# En primer lugar vemos que
#Information Criteria
-----
#Akaike      8.7398
#Bayes       8.7536

#El AIC y el BIC son muy bajos, lo que indica que el modelo tiene un buen ajuste.
#####

# En cuanto a los otros coeficientes del modelo obtenidos en coef(fit_garch),
#vemos que un valor alpha1 = 0.086562 tan pequeño nos
#indica que aunque la volatilidad depende de los errores pasados la persistencia
#de este efecto no es tan fuerte en valores posteriores.

#El beta1= 0.91243757 indica que la varianza futura está muy influenciada por la varianza pasada.
#Es decir hay una gran persistencia de la volatilidad.

#Lo que nos permite afirmar que el modelo creado capture la dinámica de la volatilidad con alta persistencia, es decir, la
volatilidad futura será igual que la volatilidad de los datos ya estudiados.

#Podemos realizar la prueba del multiplicador de Lagrange, pero si observamos los resultados vemos que ya tenemos los
coeficientes de esta prueba

-----
#Weighted ARCH LM Tests
-----
# Statistic Shape Scale P-Value
#ARCH Lag[3]    0.3022 0.500 2.000  0.5825
#ARCH Lag[5]    0.5762 1.440 1.667  0.8609
#ARCH Lag[7]    2.1289 2.315 1.543  0.6900

#Los p-value [0.5825,0.8609,0.6900] nos permiten afirmar que no hay evidencias de efecto ARCH que no hayan sido modelados,
es decir
#conseguimos capturar toda la volatilidad condicional del modelo.

#Para verificar que el modelo está bien ajustado podemos ver el valor de la bondad de ajuste, es decir del Coeficiente de
Pearson:
#En el resultado del ajuste del modelo en la última ristra de datos podemos ver que se muestra el
#coeficiente de Pearson ajustado, concretamente su p-value = 8.408e-49, que a diferencia de los otros valores anteriores este
indica que puede que le modelo
# no se ajuste del todo bien a los datos.

#####
#Conclusión: Aunque emplear el modelo GARCH (1,1) no es del todo una mala decisión debemos tener en cuenta que
# gracias al R^2 ajustado vemos que el modelo no se ajusta del todo bien a los datos y por ello podríamos probar con otro
# modelo como un EGARCH o un GARCH(2,1)
#####
# Definimos el modelo GARCH(2,1)
modelo_garch_2 <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(2, 1)),
                             mean.model = list(armaOrder = c(0, 1), include.mean = TRUE),
                             distribution.model = "norm") # Distribución normal
# Ajustamos el modelo GARCH(1,1)
fit_garch_2 <- ugarchfit(spec = modelo_garch_2, data = best_model$residuals)

# Mostramos el resumen del modelo ajustado
show(fit_garch_2)

# Mostramos los coeficientes del modelo GARCH (2,1)
coef(fit_garch_2)
#####
#Modelo EGARCH
modelo_egarch <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(1, 1)),

```

```

mean.model = list(armaOrder = c(0, 0), include.mean = TRUE,
                  distribution.model = "norm") # Distribución normal
# Ajustar el modelo EGARCH(1,1) a los datos de mi serie temporal
fit_egarch <- ugarchfit(spec = modelo_egarch, data = best_model$residuals)

# Mostrar el resumen del modelo ajustado
show(fit_egarch)

# Mostramos los coeficientes del modelo GARCH (1,1)
coef(fit_egarch)
# Obtengo este error que me indica que el modelo es muy complejo para usar esta variante de GARCH, pues falla la convergencia,
# pese a que he intentado ponerle una tolerancia al modelo sigue sin poder ajustar bien los datos y por lo tanto no puede
converger. Eso se
# traduce en que no puedo analizar los coeficientes de los resultados, por lo tanto ese modelo no es óptimo.
#####
# Por último tras capturar la volatilidad con nuestro modelo GARCH(2,1) y validar el modelo vamos a realizar una predicción
# del precio de la criptomonedas Ethereum con respecto al euro un mes después desde la fecha del último dato recopilado,
# decir un mes después del 28/01/2022
ug_forecast = ugarchforecast(fit_garch_2, n.ahead = 30)

ug_forecast

#Análisis de los resultados:

# Vemos que la columna serie nos devuelve el valor estimado de la media del precios de la criptomonedas Ethereum con respecto
al euro en los próximos 30 días
# vemos que la media de la serie cambia a lo largo de las iteraciones, en este caso decrece en la segunda iteración.
# En la primera iteración el valor es de -0.26861 y en la segunda iteración es de -0.04407. A partir de la segunda iteración
se mantiene el valor fijo del precio, esto nos sugiere que la serie
# no cambia a lo largo del tiempo, concretamente podemos decir que el precio de las monedas Ethereum no cambia prácticamente
a lo largo de un mes, llegando a la conclusión de que el precio en el periodo de un mes
# es estable en cuanto a la predicción de su media.

# La columna sigma representa la volatilidad condicional. De esta columna vemos que se mantienen casi constantes,
# vemos que la poca variación de la volatilidad es señal del grado de variabilidad o de incertidumbre de los precios.
# En nuestro caso vemos como la tendencia de la volatilidad de los precios es baja disminuye de un 126.3 a un 124.6 en
cuestión de 30 días.
# Esto nos permite deducir que las fluctuaciones en el precio de las monedas Ethereum con respecto al euro disminuirán y
provocarán que se estabilice el precio con el tiempo.

# Conclusión: El precio del Ethereum con respecto al euro se va a volver menos volátil en un periodo futuro de 30 días a
partir de la última estimación tomada para el análisis de la serie temporal. Además la tendencia
# del precio de la criptomonedas Ethereum con respecto al euro es decreciente en febrero de 2022
#####
#Representación gráfica de la predicción:

# Convertimos los datos de la predicción en un data frame, para poder usarlo en la creación de la gráfica
# Coeficientes de la predicción
valores_pred <- fitted(ug_forecast)

volatilidad_predicha <- sigma(ug_forecast)

# Creamos un rango de fechas para las predicciones

# La última fecha en la serie histórica es 20/01/2022
comienzo <- as.Date("2022-01-28")
datos_forecast <- seq(comienzo + 1, by = "day", length.out = length(valores_pred))

# Combinamos los resultados en un DataFrame
tabla_datos_prediccion <- data.frame(
  Fecha = datos_forecast,
  Predicción = as.numeric(valores_pred),
  Volatilidad = as.numeric(volatilidad_predicha)
)

print(head(tabla_datos_prediccion))

# Graficamos la serie histórica ETH-EUR con los valores de la predicción

st_eth <- ts(datos_seleccionados$High, frequency = 1)

plot(st_eth,
      type = "l",
      xaxt = "n", main = "Predicción serie de cambio ETH-EUR", xlab = "Tiempo",
      ylab = "Precio")

# Eje X
axis(1, at = seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10)),
     labels = datos_seleccionados>Date[seq(1, length(datos_seleccionados>Date), by = round(length(datos_seleccionados>Date) / 10))])

# Fechas de la predicción
comienzo <- as.Date(tail(datos_seleccionados>Date, 1)) # Última fecha histórica
datos_forecast <- seq(comienzo + 1, by = "day", length.out = length(tabla_datos_prediccion$Predicción))

# Predicciones
lines(x = length(st_eth) + 1:length(tabla_datos_prediccion$Predicción),
      y = tabla_datos_prediccion$Predicción,
      col = "red",
      lwd = 2) # Línea roja para las predicciones

# Intervalo de confianza
lines(x = length(st_eth) + 1:length(tabla_datos_prediccion$Predicción),
      y = tabla_datos_prediccion$Predicción + 2 * tabla_datos_prediccion$Volatilidad,
      col = "red",
      lty = 2)

```

```
lines(x = length(st_eth) + 1:length(tabla_datos_prediccion$Prediccion),
      y = tabla_datos_prediccion$Prediccion - 2 * tabla_datos_prediccion$Volatilidad,
      col = "red",
      lty = 2)

# Leyenda
legend("topleft",
       legend = c("Serie Histórica", "Predicción", "Intervalo de Confianza"),
       col = c("blue", "red", "red"),
       lty = c(1, 1, 2),
       lwd = c(1, 2, 1),
       cex = 0.7)
```