Configuração do ambiente

In [ ]:
```
install.packages(c("ggfortify","fpp2","forecast","astsa","ggplot2","dplyr
required_packages=c("ggfortify","fpp2","forecast","astsa","ggplot2","dply
lapply(required_packages, library, character.only=TRUE)
```

Importação dos dados e pré-processamento dos dados

In [ ]:
```
raw_df = read.csv("./gold_price_data.csv")
raw_df$Date = as.Date(raw_df$Date)
#Create a parallel dataframe with the complete dates getting start and en
complete_dates = data.frame(
    Date = seq(min(raw_df$Date), max(raw_df$Date), by = "days")
)
#merge both dataframes to create NA values into the missing date values a
merged_withNA = merge(raw_df, complete_dates, by = "Date", all=TRUE)

#Checking the amount of missing values in the DF
sum(is.na(merged_withNA$Value))

#creating a start date object to use as reference
date1980s = as.Date("1980-01-01")
#filtering the gold price observations after the reference date (1980)
after1980s = merged_withNA[merged_withNA$Date >= date1980s, ]

#resampling the dataframe for weekly frequency from the observations star
weekly_data = seq(min(after1980s$Date), max(after1980s$Date), by = "7 day
weekly_data_withValues = merged_withNA[merged_withNA$Date %in% weekly_dat
```

In [ ]:
```
start_date = as.Date(min(weekly_data_withValues$Date))
start_date_forTS = c(year(start_date),isoweek(start_date))
end_date = max(weekly_data_withValues$Date)
end_date_forTS = c(year(end_date),isoweek(end_date))

goldPrice_TS = ts(weekly_data_withValues[,"Value"],
    start = start_date_forTS,
    end = end_date_forTS,
    frequency = 52)
```

Dataframe primario

In [ ]:
```
plot = ggplot(raw_df, aes(x= Date, y = Value)) +
              geom_point() +
              labs(x = "Date", y = "Value") +  # Add axis labels
              ggtitle("Scatter Plot of Date vs. Value") # Add a title
              theme_minimal(base_size = 16) +  # Adjust the base size of
              theme(plot.title = element_text(size = 20),
              main = "Série temporal do preço do ouro - Pré tratamento")
# Set the size of the plot
options(repr.plot.width = 15, repr.plot.height = 15)  # Adjust the width
# Print the scatter plot
```

```
print(plot)
```

Dataframe pos processamento

In [ ]:
```
ts.plot(goldPrice_TS, xlab= "Year", main="Série temporal do preço do ouro
# Customize the x-axis labels to show a yearly grid using time(diff_goldP
years = seq(as.integer(year(start_date)), as.integer(year(end_date)))
axis(1, at = years)
```

Estudo do Retorno

In [ ]:
```
returnTS_gold = diff(goldPrice_TS) / stats::lag(goldPrice_TS, -1)
sqd_returnTS_gold = returnTS_gold^2
kurt_return = kurtosis(returnTS_gold)
print(kurt_return)
```

In [ ]:
```
par(mfrow = c(2,1), mar = c(4, 4, 2, 2), oma = c(0, 0, 1, 0), mex = 0.8)
norm_dist_reference = seq(min(returnTS_gold),max(returnTS_gold),by=0.001)
plot(returnTS_gold, main = "Série temporal do retorno semanal do preço do
hist(returnTS_gold, freq= FALSE, col = "lightblue", main= "Distribuição d
lines(density(returnTS_gold))
lines(norm_dist_reference,dnorm(norm_dist_reference,mean(returnTS_gold),s
legend("topleft", legend = "Return Distribution", col = "black", lty = 1,
legend("topright", legend = paste("Kurtosis:", round(kurt_return, 3)), co
```

In [ ]:
```
# Perform Shapiro-Wilk test
shapiro_test_result = shapiro.test(returnTS_gold)
print(shapiro_test_result)
```

```
	Shapiro-Wilk normality test

data:  returnTS_gold
W = 0.90733, p-value < 2.2e-16
```

In [ ]:
```
resid_AR = resid(sarima(returnTS_gold, 1,0,0, details=FALSE)$fit )
plot(resid_AR)
```

Validação da existência do efeito ARCH nos retornos.

In [ ]:
```
ArchTest(returnTS_gold)
```

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

data:  returnTS_gold
Chi-squared = 188.33, df = 12, p-value < 2.2e-16
```

Residuos do modelo AR sobre os retornos

In [ ]:
```
par(mfrow = c(4,1))
acf(resid_AR)
pacf(resid_AR)
acf(resid_AR^2)
```

```
pacf(resid_AR^2)
summary(garchFit(resid_AR,formula = ~garch(1,1)))
```

Estudo da autocorrelação nos retornos e retornos²

In [ ]:
```
par(mfrow = c(3,1))
plot(returnTS_gold, main = "Return Time Series")
acf(returnTS_gold, main= "ACF Return")
pacf(returnTS_gold, main= "PACF Return")
```

In [ ]:
```
par(mfrow = c(3,1))
plot(sqd_returnTS_gold, main = "Squared Returns Time Series")
acf(sqd_returnTS_gold, main= "ACF Squared Returns")
pacf(sqd_returnTS_gold, main= "PACF Squared Returns")
```

Testando modelos GARCH e parametros

In [ ]:
```
fit11 = garchFit(~garch(1,1), data=returnTS_gold)
fit21 = garchFit(~garch(2,1), data=returnTS_gold)
fit22 = garchFit(~garch(2,2), data=returnTS_gold)
```

In [ ]:
```
summary(fit11)
```

```
Title:
 GARCH Modelling

Call:
 garchFit(formula = ~garch(1, 1), data = returnTS_gold)

Mean and Variance Equation:
 data ~ garch(1, 1)
<environment: 0xdb69238>
 [data = returnTS_gold]

Conditional Distribution:
 norm

Coefficient(s):
        mu       omega      alpha1       beta1
5.9853e-04  1.0537e-05  6.7443e-02  9.1421e-01

Std. Errors:
 based on Hessian

Error Analysis:
        Estimate  Std. Error  t value Pr(>|t|)
mu     5.985e-04   4.313e-04    1.388    0.165
omega  1.054e-05   2.293e-06    4.595 4.32e-06 ***
alpha1 6.744e-02   8.966e-03    7.522 5.37e-14 ***
beta1  9.142e-01   1.033e-02   88.502  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Log Likelihood:
 5004.622    normalized:  2.394556

Description:
 Thu Feb  1 23:04:40 2024 by user: dbsantos


Standardised Residuals Tests:
                                Statistic   p-Value
 Jarque-Bera Test   R    Chi^2  9330.7848178 0.0000000
 Shapiro-Wilk Test  R    W         0.9410989 0.0000000
 Ljung-Box Test     R    Q(10)     9.5307574 0.4825788
 Ljung-Box Test     R    Q(15)    11.4864547 0.7174103
 Ljung-Box Test     R    Q(20)    17.2700779 0.6353731
 Ljung-Box Test     R^2  Q(10)     2.1278883 0.9952627
 Ljung-Box Test     R^2  Q(15)     2.8077599 0.9997340
 Ljung-Box Test     R^2  Q(20)     3.2375142 0.9999921
 LM Arch Test       R    TR^2      1.8309125 0.9996245

Information Criterion Statistics:
      AIC       BIC       SIC      HQIC
-4.785284 -4.774480 -4.785291 -4.781326
```

In [ ]:

```
summary(fit21)
summary(fit22)
```

GARCH(1,1) estudo dos resíduos

In [ ]:
```
par(mfrow = c(4,4))
plot(fit11, which = "all")
```

In [ ]:
```
residuals_stdzided = residuals(fit11, standardize =TRUE)
```

Validação da existencia de efeito ARCH nos resíduos padronizados do GARCH(1,1)

In [ ]:
```
shapiro.test(residuals_stdzided)
ArchTest(residuals_stdzided)
```

```
        Shapiro-Wilk normality test

data:  residuals_stdzided
W = 0.9411, p-value < 2.2e-16
        ARCH LM-test; Null hypothesis: no ARCH effects

data:  residuals_stdzided
Chi-squared = 1.8309, df = 12, p-value = 0.9996
```

In [ ]:
```
par(mfrow = c(4,1))
acf(residuals_stdzided, main = "Standardized Residuals of GARCH(1,1) ACF"
pacf(residuals_stdzided, main = "Standardized Residuals of GARCH(1,1) ACF
plot(fit11, which = 13)
x = seq(min(residuals_stdzided),max(residuals_stdzided),by=0.001)
plot(density(residuals_stdzided), main = "Distribuiçao de densidade das f
lines(x,dnorm(x,mean(residuals_stdzided),sd(residuals_stdzided)),col = "r
plot(fit11, which= 3)
```

Predição | Análise Janela Móvel

Predição

In [ ]:
```
garch_model = garchFit(~garch(1, 1), data = returnTS_gold)

# Predict volatility on the test set
predicted_ = predict(garch_model, n.ahead = 100, plot=TRUE)

# Extract the conditional standard deviation (volatility) from the predic
predicted_volatility = sqrt(predicted_$standardDeviation)

# Plot the original returns and predicted volatility
par(mfrow = c(2, 1))
plot(returnTS_gold, type = "l", col = "blue", main = "Retornos Originais"
plot(predicted_volatility, type = "l", col = "red", main = "Volatilidade

# Adicionar rótulos de data em alguns pontos específicos (por exemplo, a
pontos_rotulos = seq(5, length(predicted_volatility), by = 5)
text(index(predicted_)[pontos_rotulos], predicted_volatility[pontos_rotul
```

Janela Móvel

In [ ]:
```
spec = ugarchspec(variance.model = list(model="fGARCH" , submodel="GARCH"
    , mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
distribution.model = "norm")
```

```
#Definiçoes da janela movel
window_size_ = 700
start_date = window_size_

rolling = ugarchroll(spec, returnTS_gold, n.start= start_date,
 refit.every = 1, refit.window = 'moving', window.size = window_size_,
  calculate.VaR = TRUE, keep.coef = TRUE)


result_rolllingWindow = as.data.frame(rolling)

#Ajustar indices das datas do dataframe
gold_ts_dates = as.numeric((time(returnTS_gold)))
gold_ts_dates = as.Date(date_decimal(gold_ts_dates))

#alterar somente após primeira janela (n=700)
dates_after_firstWindow = gold_ts_dates[(window_size_ + 1) :length(gold_t
row.names(result_rolllingWindow) = dates_after_firstWindow

#Extrair valores
predVolatility_sigma = xts(rslt$Sigma, order.by = as.Date(rownames(rslt))
ground_truth_return = xts(rslt$Realized, order.by = as.Date(rownames(rslt
```

In [ ]:

```
plot(ground_truth_return)
lines(predVolatility_sigma, col= "red")
```