ASML Stock Price Analysis

Project of Time Series Analysis for Economic and Financial Data

Daniele Tambone

2026-01-28

Table of contents

Preface

GitHub Repository

View Slides

Download PDF

This project aims to conduct an in-depth time series analysis of the stock prices for **ASML Holding NV (ASML)**, a global leader in the manufacturing of lithography systems for the semiconductor industry. The primary objective is to explore the dynamics of historical prices, volatility, and underlying trends of the stock.

Data Source

We retrieve historical daily data using the `quantmod` R package, focusing on the **Adjusted Closing Price** to account for dividends and stock splits.

- **Ticker**: ASML
- **Source**: Yahoo Finance
- **Start Date**: 2000-01-01
- **End Date**: 2025-12-31

Data Loading

The dataset structure is shown below:

```
datatable(
  tail(asml, 100),
  rownames = FALSE,
  options = list(
    pageLength = 10,
    order = list(list(0, 'desc'))
  )
)
```

Show 10 entries Search:

Date	ASML.Open	ASML.High	ASML.Low	ASML.Close	ASML.Volume	ASML.Adjusted
2025-12-30	1084.579956054688	1086.06005859375	1070.420043945312	1072.140014648438	807300	1072.140014648438
2025-12-29	1064.579956054688	1073.650024414062	1061.069946289062	1066	424300	1066
2025-12-26	1066.25	1076.089965820312	1063.079956054688	1072.75	363800	1072.75
2025-12-24	1059.819946289062	1066.800048828125	1057.43994140625	1065.52001953125	229800	1065.52001953125
2025-12-23	1057.93994140625	1064.75	1055.68994140625	1061.839965820312	543600	1061.839965820312
2025-12-22	1066	1067.160034179688	1050	1056.97998046875	703700	1056.97998046875
2025-12-19	1042.56005859375	1062.680053710938	1042.56005859375	1056.02001953125	2319600	1056.02001953125
2025-12-18	1047.969970703125	1051.81005859375	1035.150024414062	1036.31005859375	1645400	1036.31005859375
2025-12-17	1060.77001953125	1065.119995117188	1010.010009765625	1015.429992675781	2140900	1015.429992675781
2025-12-16	1081.530029296875	1088.06005859375	1065.93994140625	1076.050048828125	956300	1076.050048828125

Showing 1 to 10 of 100 entries Previous 1 2 3 4 5 … 10 Next

1 Exploratory Data Analysis

1.1 Adjusted closing prices

Let us plot the adjusted closing prices...

```
fig_asml <- plot_ly(data = asml, x = ~Date, y = ~ASML.Adjusted,
  type = 'scatter', mode = 'lines', name = 'ASML Adjusted',
  line = list(color = 'darkblue', width = 1.5))

fig_asml <- layout(
  fig_asml,
  title = "ASML Stock Price Evolution",
  xaxis = list(
    title = "Date",
    rangeslider = list(visible = TRUE),
    rangeselector = list(
      buttons = list(
        list(count=1, label="1m", step="month", stepmode="backward"),
        list(count=6, label="6m", step="month", stepmode="backward"),
        list(count=1, label="YTD", step="year", stepmode="todate"),
        list(count=1, label="1y", step="year", stepmode="backward"),
        list(step="all")
      )
    )
  )
)

fig_asml
```

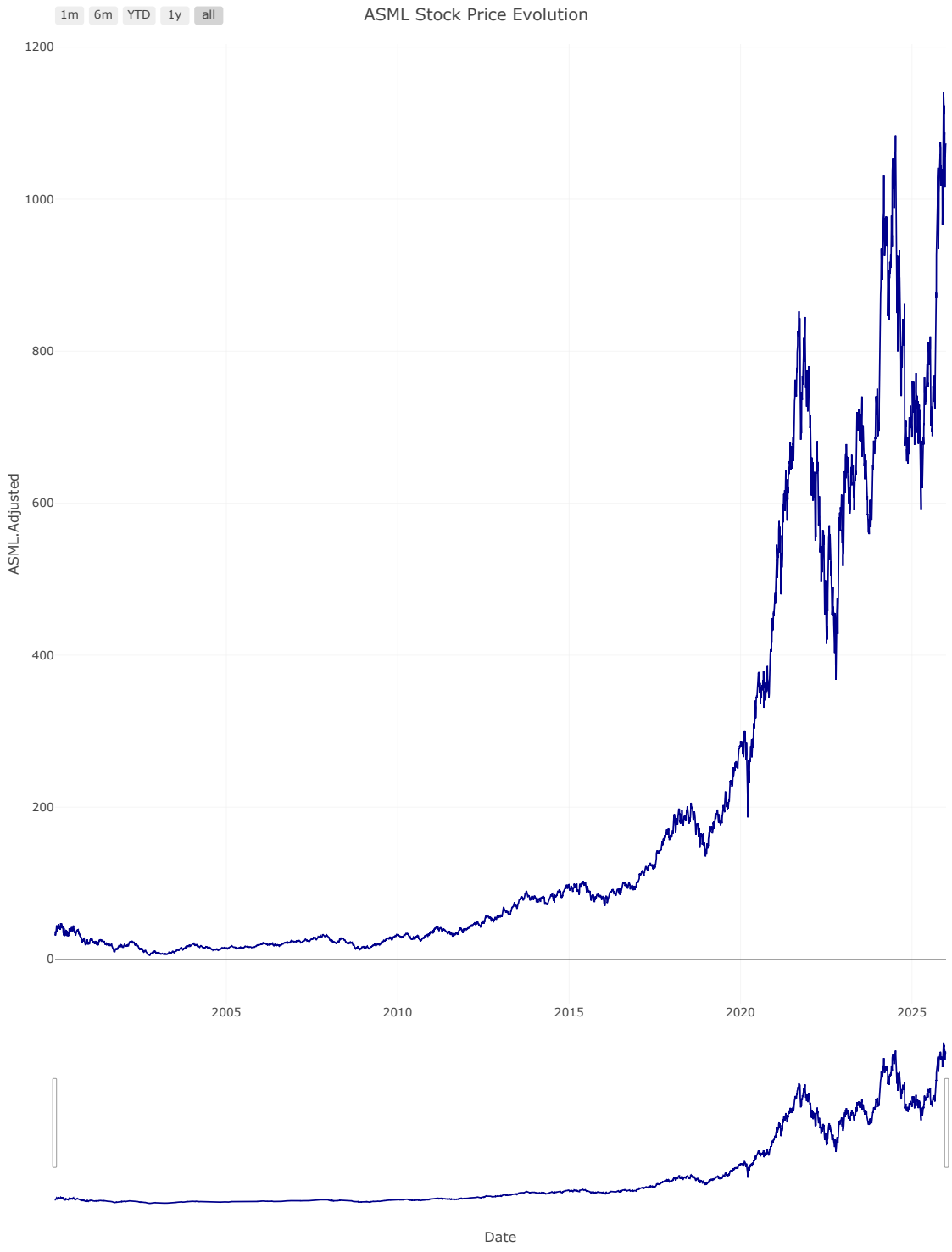


Figure 1.1: ASML Daily Adjusted Closing Prices

The graph clearly shows that the price series is **non-stationary**: the mean is not constant (it exhibits trends), and the variance appears to fluctuate depending on the price level. Consequently, standard statistical inference cannot be directly applied to prices.

1.2 Log returns

To obtain a stationary process, we transform prices into **log-returns**. Let P_t be the price at time t. The simple net return is defined as:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

However, in financial econometrics, **log-returns** (r_t) are preferred due to their time-additivity property. The log-return is defined as the natural logarithm of the gross return:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) = \ln(P_t) - \ln(P_{t-1})$$

We compute the log-returns for ASML in R:

```
log_ret_vec <- diff(log(asml$ASML.Adjusted))
log_ret_vec <- log_ret_vec[is.finite(log_ret_vec)]

log_returns <- data.frame(
  Date = asml$Date[-1],
  LogReturns = as.numeric(log_ret_vec)
)
```

Let's visualize the log-returns of ASML:

```
fig_log <- plot_ly(data = log_returns, x = ~Date, y = ~LogReturns, type = 'scatter',
  mode = 'lines', name = 'Log Returns', line = list(color = 'darkred', width = 1))

fig_log <- layout(
  fig_log,
  title = "ASML Log-Returns",
  xaxis = list(title = "Date"),
  yaxis = list(title = "Log Return"),
  shapes = list(
    list(
      type = "line",
      x0 = min(log_returns$Date),
      x1 = max(log_returns$Date),
      y0 = 0,
```

```
      y1 = 0,
      line = list(color = "black", width = 1)
    )
  )
)

fig_log
```

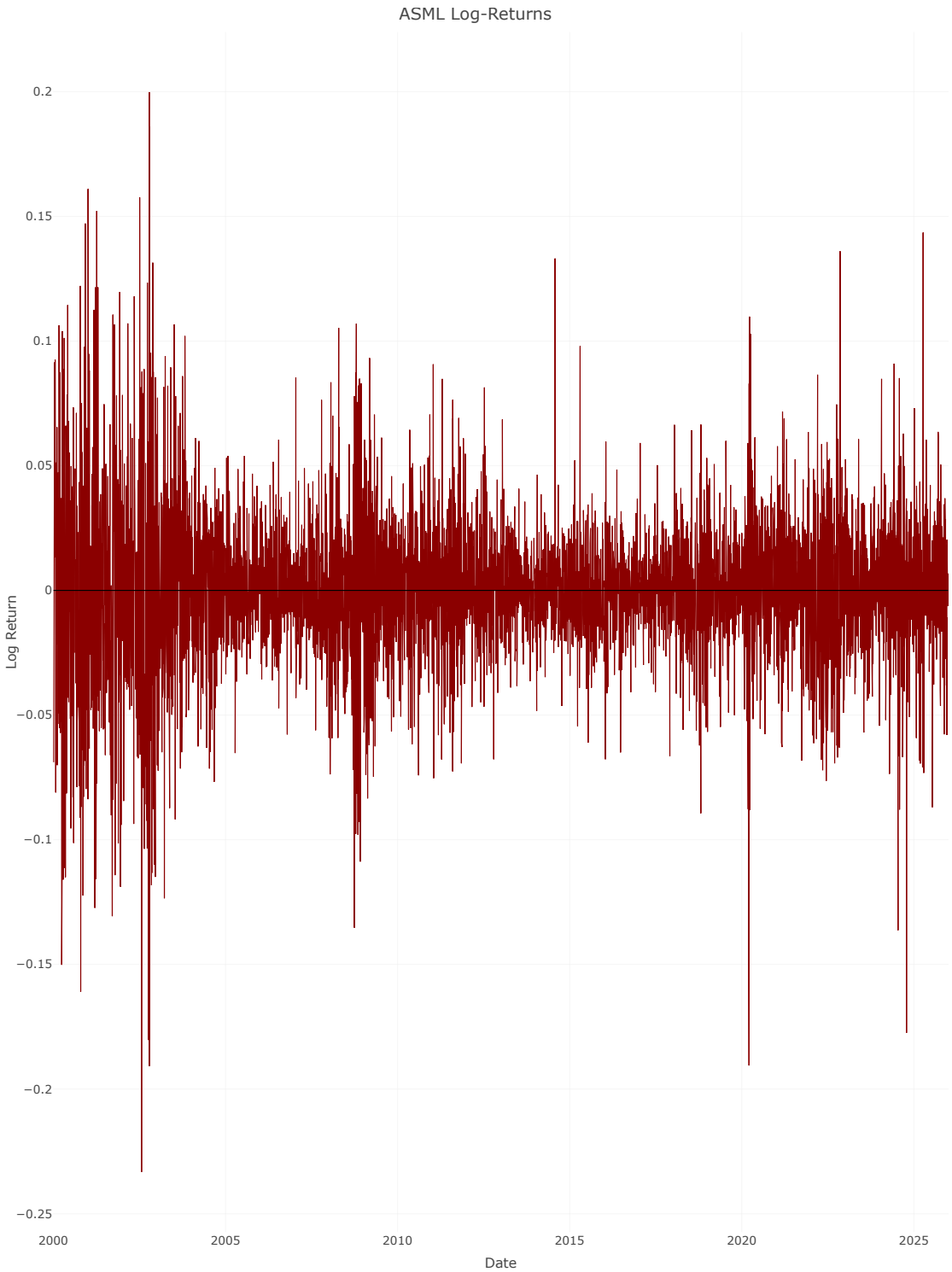


Figure 1.2: ASML Daily Log-Returns

Unlike prices, the log-returns oscillate around a constant mean (close to zero). This behavior suggests that the return series is **stationary**, satisfying the conditions for further econometric analysis.

1.3 Distribution analysis

We analyze the distribution of log-returns to check for deviations from normality (e.g., fat tails or asymmetry).

1.3.1 Graphical representation

1.3.1.1 Histogram

The graphical representation using a histogram of the frequency distribution of returns observed over a given sample period provides an initial indication of the characteristics of the probability distribution that generated them.

```
mu_ret <- mean(log_returns$LogReturns)
sd_ret <- sd(log_returns$LogReturns)

t_fit_fGarch <- stdFit(log_returns$LogReturns * 100)
t_params <- t_fit_fGarch$par
t_label <- sprintf("Student-t (df=%.2f)", t_params["nu"])

fig_dist <- plot_ly(data = log_returns, x = ~LogReturns,
  type = "histogram", name = "Log Returns", histnorm = "probability density",
  marker = list(color = "lightgray", line = list(color = "gray", width = 1)),
  opacity = 0.7)

x_seq <- seq(min(log_returns$LogReturns), max(log_returns$LogReturns), length.out = 500)

y_norm <- dnorm(x_seq, mean = mu_ret, sd = sd_ret)
y_t <- dstd(
  x_seq,
  mean = t_params["mean"]/100,
  sd = t_params["sd"]/100,
  nu = t_params["nu"]
)

fig_dist <- fig_dist %>%
  add_lines(
    x = x_seq,
    y = y_norm,
    name = "Normal Distribution",
```

```
    line = list(color = "#E74C3C", width = 2, dash = "dash"),
    inherit = FALSE
  ) %>%

  add_lines(
    x = x_seq,
    y = y_t,
    name = t_label,
    line = list(color = "#2E86C1", width = 2),
    inherit = FALSE
  ) %>%

  layout(
    title = "Distribution of ASML Log-Returns",
    xaxis = list(title = "Log Return"),
    yaxis = list(title = "Density"),
    legend = list(x = 0.8, y = 0.9),
    hovermode = "x unified"
  )

fig_dist
```

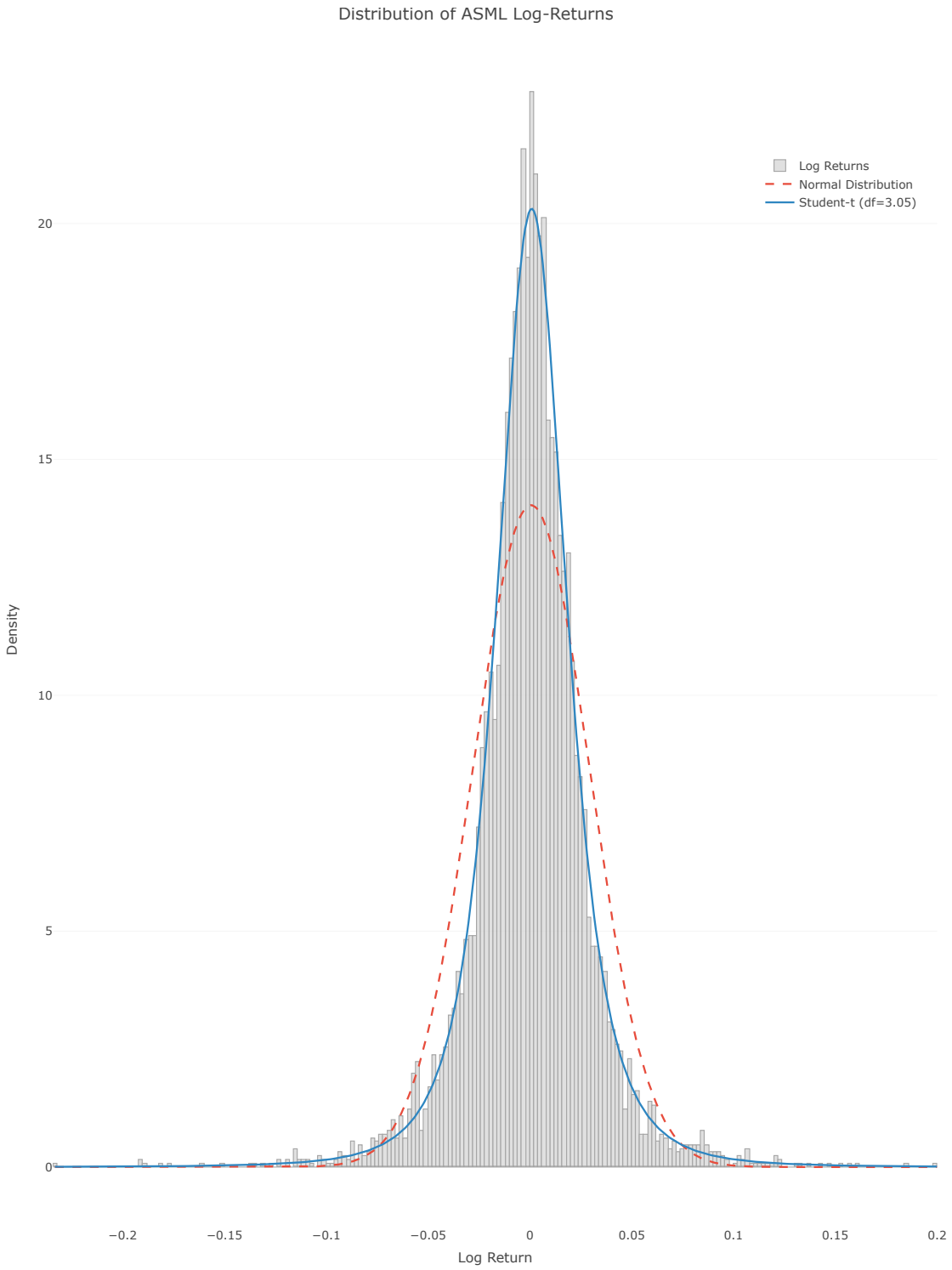


Figure 1.3: Histogram of ASML Log-Returns

The histogram indicates that the distribution of log-returns is closer to a Student-t distribution than to a Normal distribution.

1.3.1.2 Q-Q plot

```
vec_ret <- log_returns$LogReturns

qq_vals <- qqnorm(vec_ret, plot.it = FALSE)
qq_data <- data.frame(
  Theoretical = qq_vals$x,
  Sample = qq_vals$y
)

y <- quantile(vec_ret, c(0.25, 0.75), names = FALSE)
x <- qnorm(c(0.25, 0.75))
slope <- diff(y)/diff(x)
int <- y[1L] - slope * x[1L]

fig_qq <- plot_ly(data = qq_data, x = ~Theoretical, y = ~Sample, type = 'scatter',
  mode = 'markers', marker = list(size = 3, color = '#2E86C1', opacity = 0.6),
  name = "Returns")

fig_qq <- fig_qq %>%
  add_lines(
    x = ~Theoretical,
    y = ~Theoretical * slope + int,
    line = list(color = "#E74C3C", width = 2, dash = "dash"),
    name = "Normal Reference",
    inherit = FALSE
    ) %>%

    layout(
      title = "Q-Q Plot: Normal vs Empirical",
      xaxis = list(title = "Theoretical Quantiles (Normal)"),
      yaxis = list(title = "Sample Quantiles (ASML)")
    )

fig_qq
```

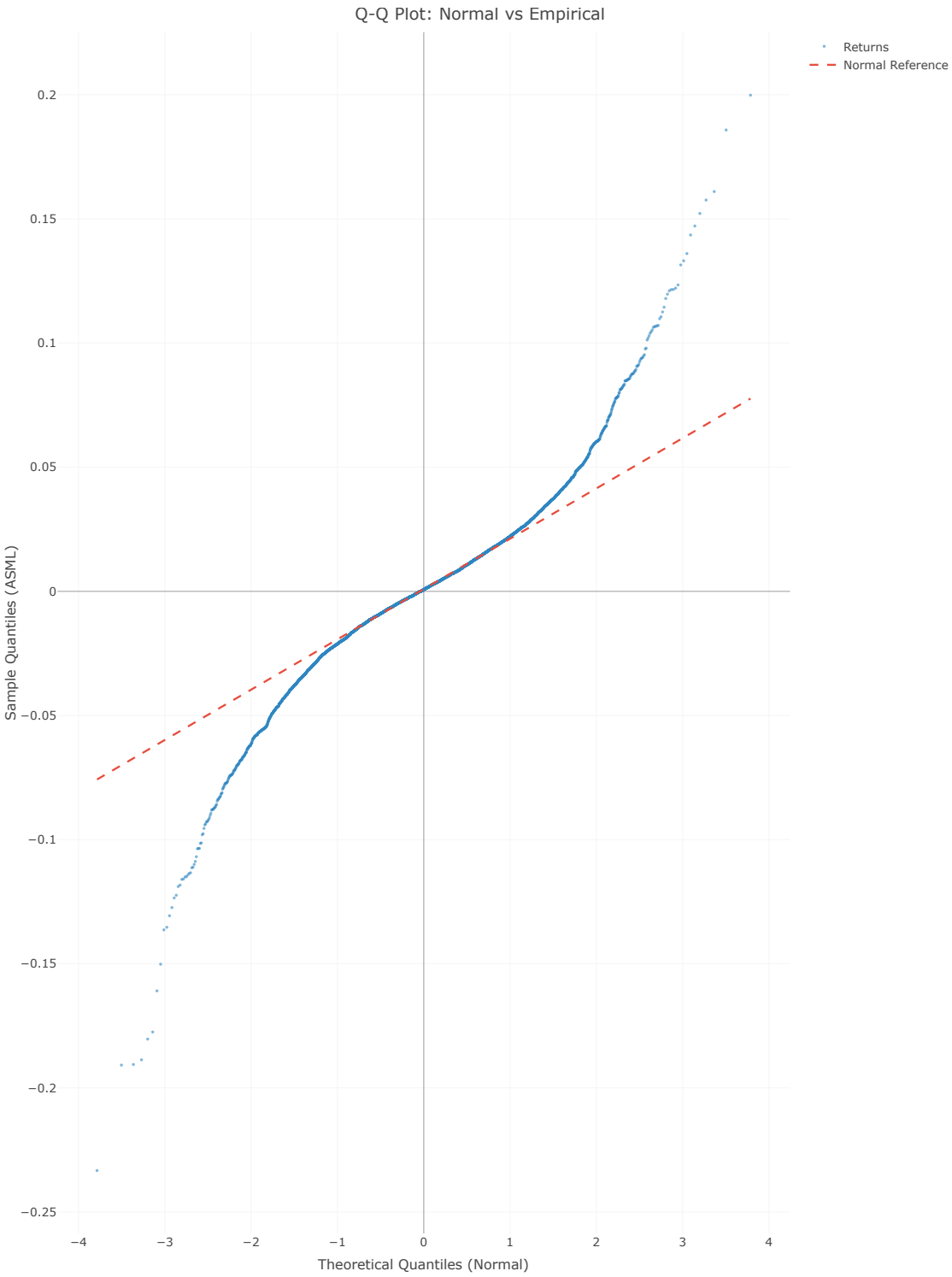


Figure 1.4: Q-Q Plot of ASML Log-Returns

The **Q-Q plot** confirms the findings from the histogram. While the central observations (the body of the distribution) align well with the theoretical normal line (in red), the extreme values at both ends deviate significantly, forming an '*S-shape*'. This provides clear visual evidence of fat tails (leptokurtosis), reinforcing the stylized fact that extreme market movements occur more frequently than predicted by a Gaussian model.

1.3.2 Synthetic indicators

```
desc_stats <- data.frame(
  Metric = c("Mean", "Std. Dev.", "Skewness", "Kurtosis"),
  Value = c(
    mean(log_returns$LogReturns),
    sd(log_returns$LogReturns),
    skewness(log_returns$LogReturns),
    kurtosis(log_returns$LogReturns)
  )
)

datatable(
  desc_stats,
  options = list(dom = 't', paging = FALSE),
  rownames = FALSE
  ) %>% formatRound('Value', digits = 5)
```

Metric	Value
Mean	0.00052
Std. Dev.	0.02843
Skewness	-0.18021
Kurtosis	8.67864

As we can see, the distribution has a mean close to zero, slight negative skewness, and significant leptokurtosis (fat tails). These characteristics align with the well-known **stylized facts** of financial log-returns, indicating a departure from normality.

1.3.3 Test of normality

To formally test the normality hypothesis, we employ the **Jarque-Bera test**, which is based on skewness and kurtosis matching a normal distribution.

$$H_0 : \text{The data is normally distributed (Skewness=0, Kurtosis=3)}$$

$$H_1 : \text{The data is not normally distributed}$$

```
jb_test <- jarque.bera.test(log_returns$LogReturns)

jb_res <- data.frame(
  Test = "Jarque-Bera",
  Statistic = round(jb_test$statistic, 2),
  P_Value = ifelse(jb_test$p.value < 0.001, "< 0.001", round(jb_test$p.value, 4)),
  Result = ifelse(jb_test$p.value < 0.05, "Reject H0", "Fail to Reject H0")
)
datatable(jb_res, options = list(dom = 't'), rownames = FALSE)
```

Test	Statistic	P_Value	Result
Jarque-Bera	8818.66	< 0.001	Reject H0

Since the p-value is virtually zero (<0.001), we strongly reject the null hypothesis of normality. This confirms that ASML returns are not normally distributed.

2 Return Analysis

The objective of this chapter is to identify the optimal ARMA(p,q) model to describe the dynamics of ASML log-returns. To determine the most suitable stochastic process, we adopt a systematic approach structured in the following stages:

1. **Exploratory Analysis:** Analysis of autocorrelation functions **(ACF/PACF)** to assess temporal dependency and detect potential seasonality or distinct patterns.
2. **Model Selection (Grid Search):** Estimation of various order combinations, residual analysis (plots and **Ljung-Box test**), and selection of the "winning" models based on the minimization of **AIC (Akaike Information Criterion)** and **BIC (Bayesian Information Criterion)**.
3. **Out-of-Sample Validation:** Comparison of the predictive performance of the selected models against a benchmark (historical mean) using a test set withheld from the estimation process.

2.1 ACF and PACF Analysis

Analyzing correlograms is the preliminary step to identify potential orders for the **AR (AutoRegressive)** and **MA (Moving Average)** components.

```
acf_res <- acf(log_returns$LogReturns, plot = FALSE, lag.max = 20)
pacf_res <- pacf(log_returns$LogReturns, plot = FALSE, lag.max = 20)

ci <- qnorm((1 + 0.95)/2)/sqrt(length(log_returns$LogReturns))

df_acf <- data.frame(
  lag = as.numeric(acf_res$lag)[-1],
  acf = as.numeric(acf_res$acf)[-1])

df_pacf <- data.frame(
  lag = as.numeric(pacf_res$lag),
  pacf = as.numeric(pacf_res$acf))

fig_acf <- plot_ly(df_acf, x = ~lag, y = ~acf, type = 'bar', name = 'ACF',
  marker = list(color = '#1f77b4')) %>%

  add_segments(x = min(df_acf$lag), xend = max(df_acf$lag), y = ci, yend = ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  add_segments(x = min(df_acf$lag), xend = max(df_acf$lag), y = -ci, yend = -ci,
```

```
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  layout(yaxis = list(title = "ACF"))

fig_pacf <- plot_ly(df_pacf, x = ~lag, y = ~pacf, type = 'bar', name = 'PACF',
  marker = list(color = '#ff7f0e')) %>%

  add_segments(x = min(df_pacf$lag), xend = max(df_pacf$lag), y = ci, yend = ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  add_segments(x = min(df_pacf$lag), xend = max(df_pacf$lag), y = -ci, yend = -ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  layout(yaxis = list(title = "PACF"))

subplot(fig_acf, fig_pacf, nrows = 1, margin = 0.05, titleY = TRUE) %>%
  layout(title = "Autocorrelation & Partial Autocorrelation", margin = list(t = 50))
```

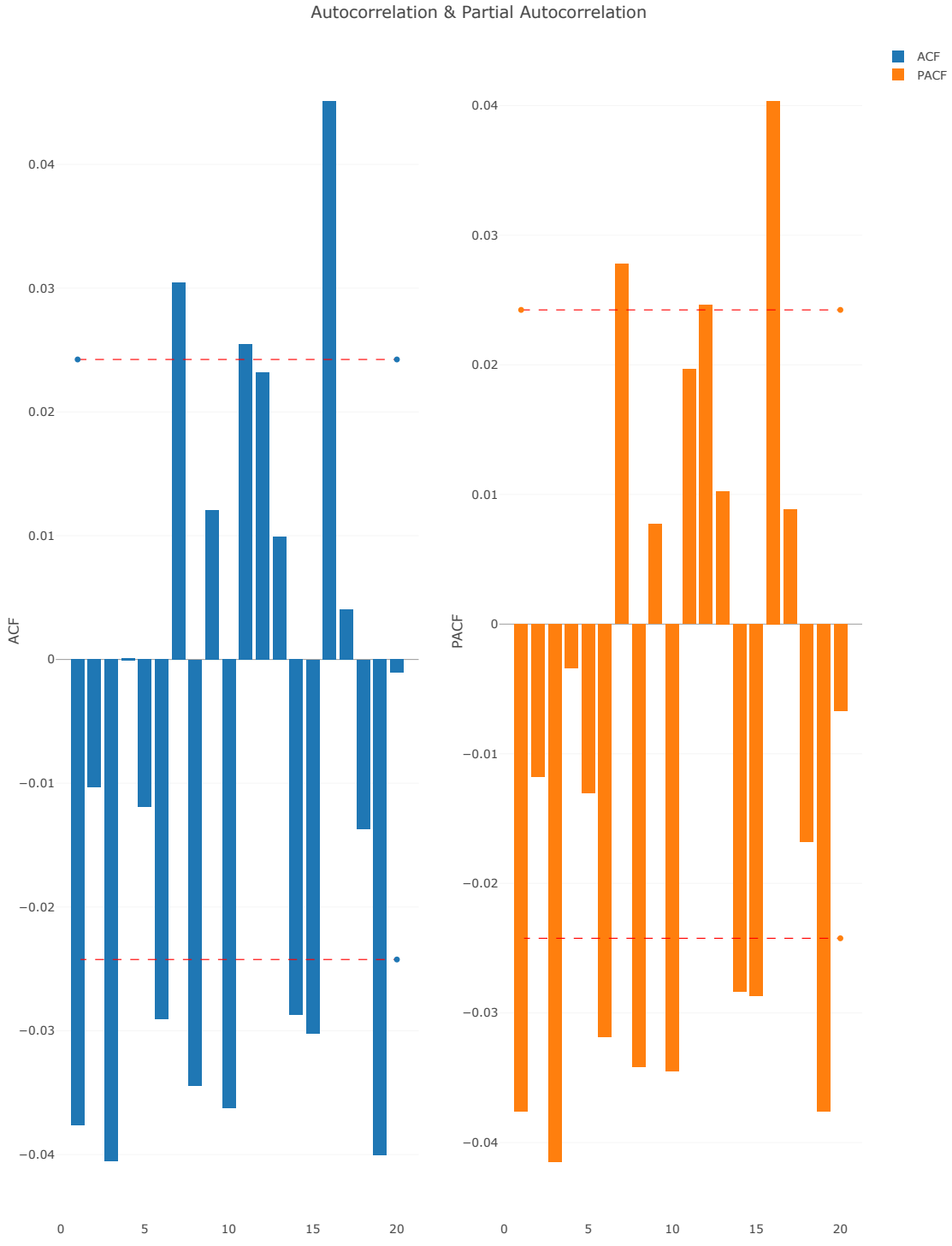


Figure 2.1: Log-Returns ACF and PACF

Observations: The autocorrelation at lag-1 is statistically significant, yet its magnitude is limited (approximately -0.04). Neither the ACF nor the PACF exhibits a clear decay or a sharp cutoff. This ambiguity makes it difficult to visually identify a pure AR or MA process, suggesting instead a mixed ARMA structure.

A classical iterative approach ("Box-Jenkins") would proceed by analyzing the **residuals**:

- If the residuals' ACF shows a sharp cutoff → add an MA term.
- If the residuals' PACF shows a sharp cutoff → add an AR term.
- If both plots decay slowly or are unclear → increase both orders.

However, given the ambiguity of the observed patterns and considering that high-order lags (> 3) are uncommon in financial contexts, we opted for a more transparent and systematic approach. We will perform a comprehensive search, testing all possible combinations within a limited range.

2.2 Models

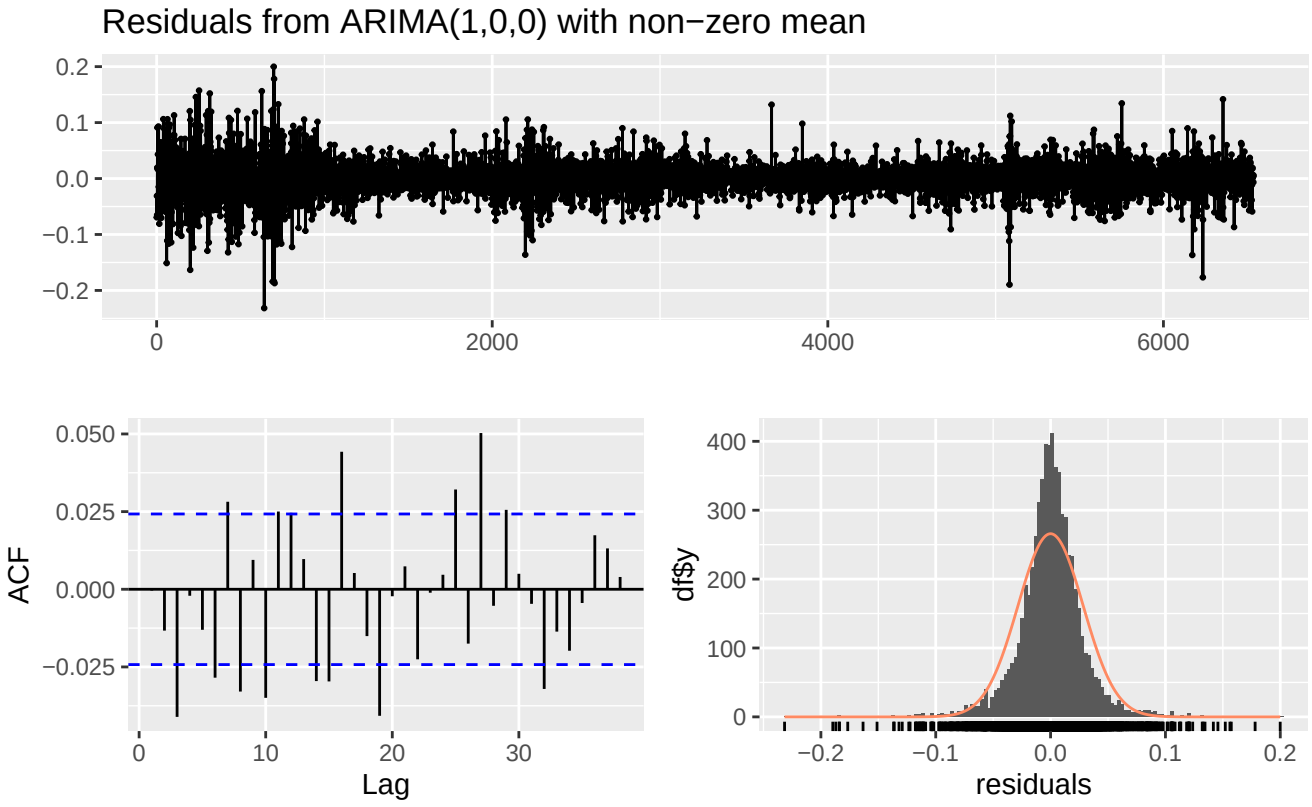
To assess **the goodness of fit** for each configuration, we analyze two fundamental diagnostic outputs:

- **Residual Analysis (`checkresiduals`)**: This function allows us to verify the absence of autocorrelation in the residuals. In addition to a visual inspection of the plots, the **Ljung-Box Test** is performed.
  - H_0 (Null Hypothesis): The residuals are independently distributed (White Noise).
  - Interpretation: A p-value > 0.05 indicates that there is insufficient evidence to reject the null hypothesis. Therefore, the model has adequately captured the temporal structure of the data.
- **Parameter Significance (`coeftest`)**: This reports the estimated coefficients (ar and ma) along with their Standard Errors. Using the z-test, we verify whether the coefficients are statistically different from zero (p-value < 0.05).

```
set.seed(123)
```

2.2.1 AR(1)

```
arma_10 <- Arima(log_returns$LogReturns, order=c(1,0,0))
checkresiduals(arma_10)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(1,0,0) with non-zero mean
Q* = 39.515, df = 9, p-value = 9.299e-06

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

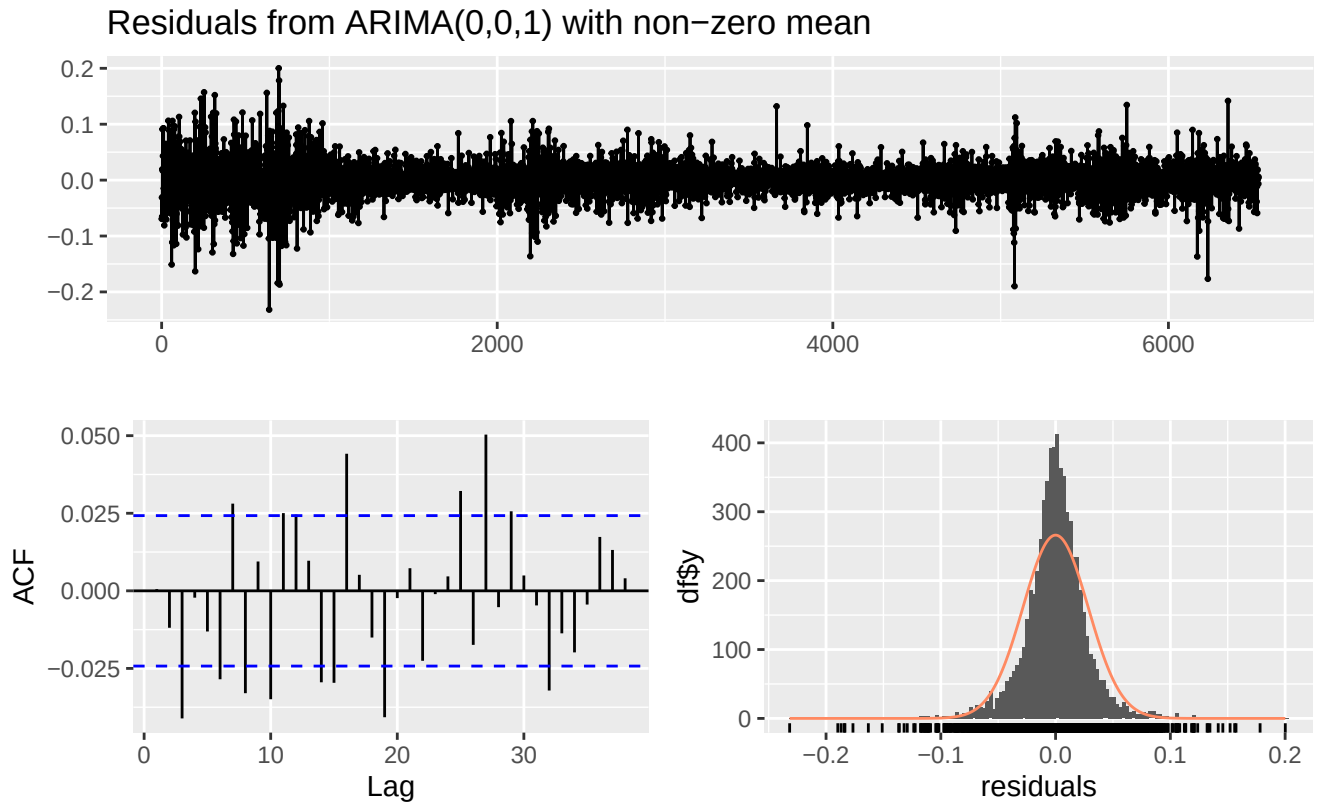
```
coeftest(arma_10)
```

```
z test of coefficients:

              Estimate  Std. Error z value Pr(>|z|)
ar1       -0.03766328  0.01236477 -3.0460 0.002319 **
intercept  0.00051722  0.00033887  1.5263 0.126937
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.2 MA(1)

```
arma_01 <- Arima(log_returns$LogReturns, order=c(0,0,1))
checkresiduals(arma_01)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(0,0,1) with non-zero mean
Q* = 39.316, df = 9, p-value = 1.01e-05

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

```
coeftest(arma_01)
```

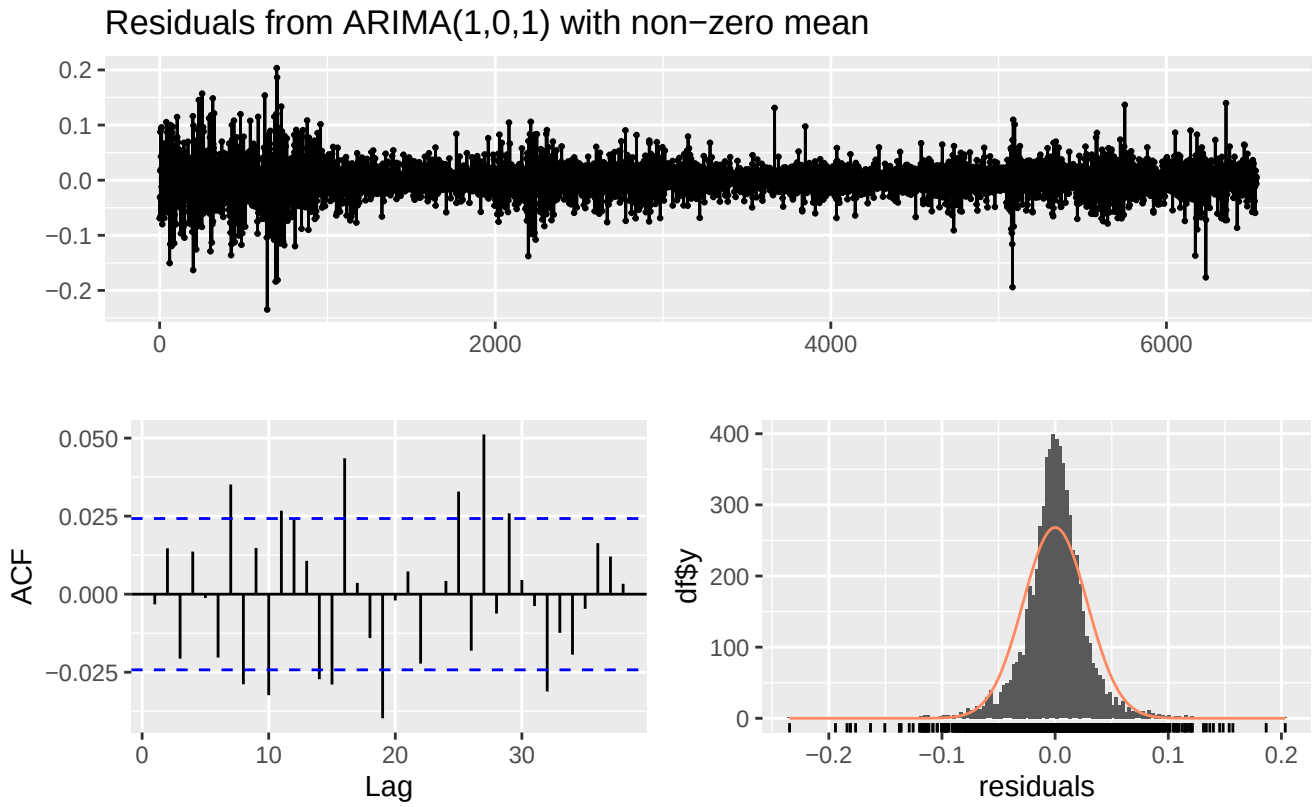
```
z test of coefficients:

             Estimate  Std. Error z value Pr(>|z|)
ma1       -0.03870356  0.01255525 -3.0827 0.002052 **
```

```
intercept  0.00051675  0.00033802  1.5288 0.126326
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.3 ARMA(1,1)

```
arma_11 <- Arima(log_returns$LogReturns, order=c(1,0,1))
checkresiduals(arma_11)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(1,0,1) with non-zero mean
Q* = 30.019, df = 8, p-value = 0.0002097

Model df: 2.   Total lags used: 10
```

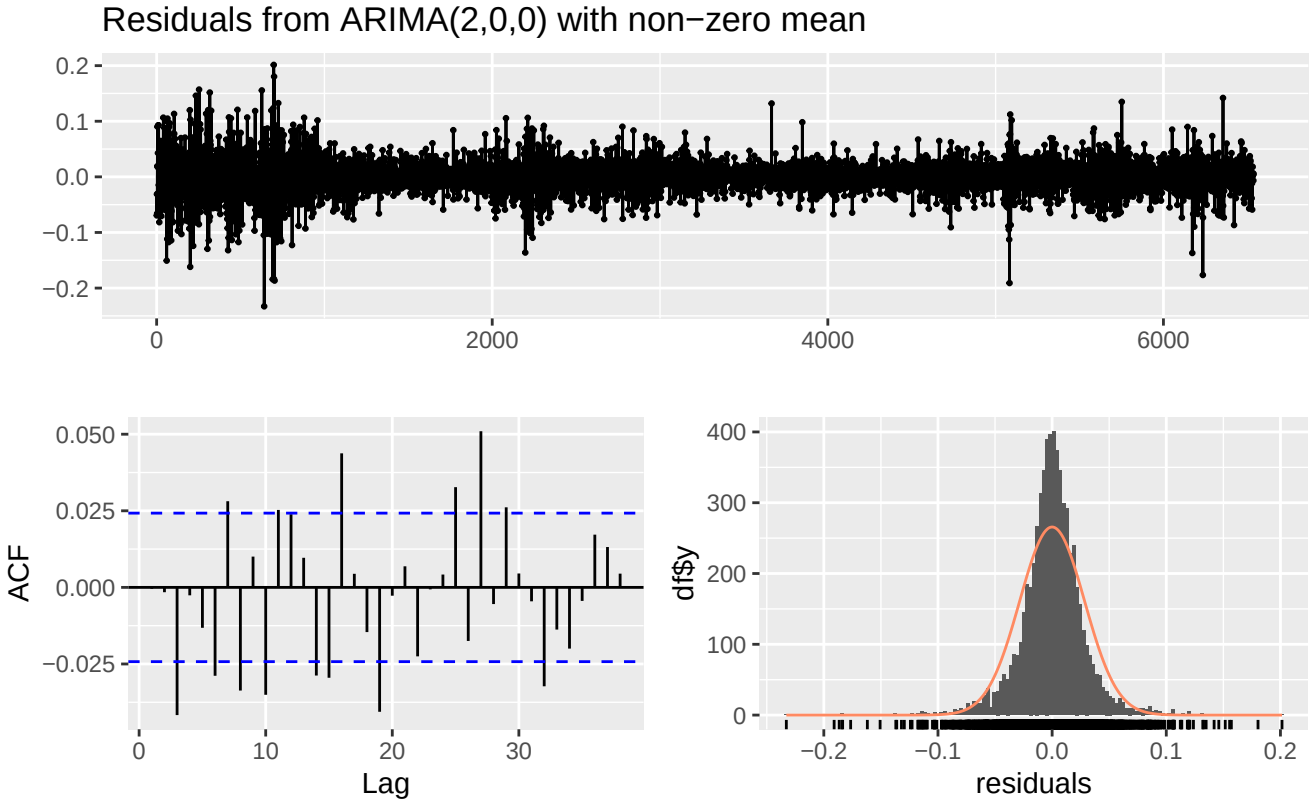
```
coeftest(arma_11)
```

```
z test of coefficients:

             Estimate  Std. Error z value  Pr(>|z|)
ar1         0.74803078  0.09296953  8.0460 8.556e-16 ***
ma1        -0.78463504  0.08698476 -9.0204 < 2.2e-16 ***
intercept   0.00049195  0.00030041  1.6376    0.1015
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.4 AR(2)

```
arma_20 <- Arima(log_returns$LogReturns, order=c(2,0,0))
checkresiduals(arma_20)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(2,0,0) with non-zero mean
Q* = 39.327, df = 8, p-value = 4.275e-06
```

```
Model df: 2.   Total lags used: 10
```

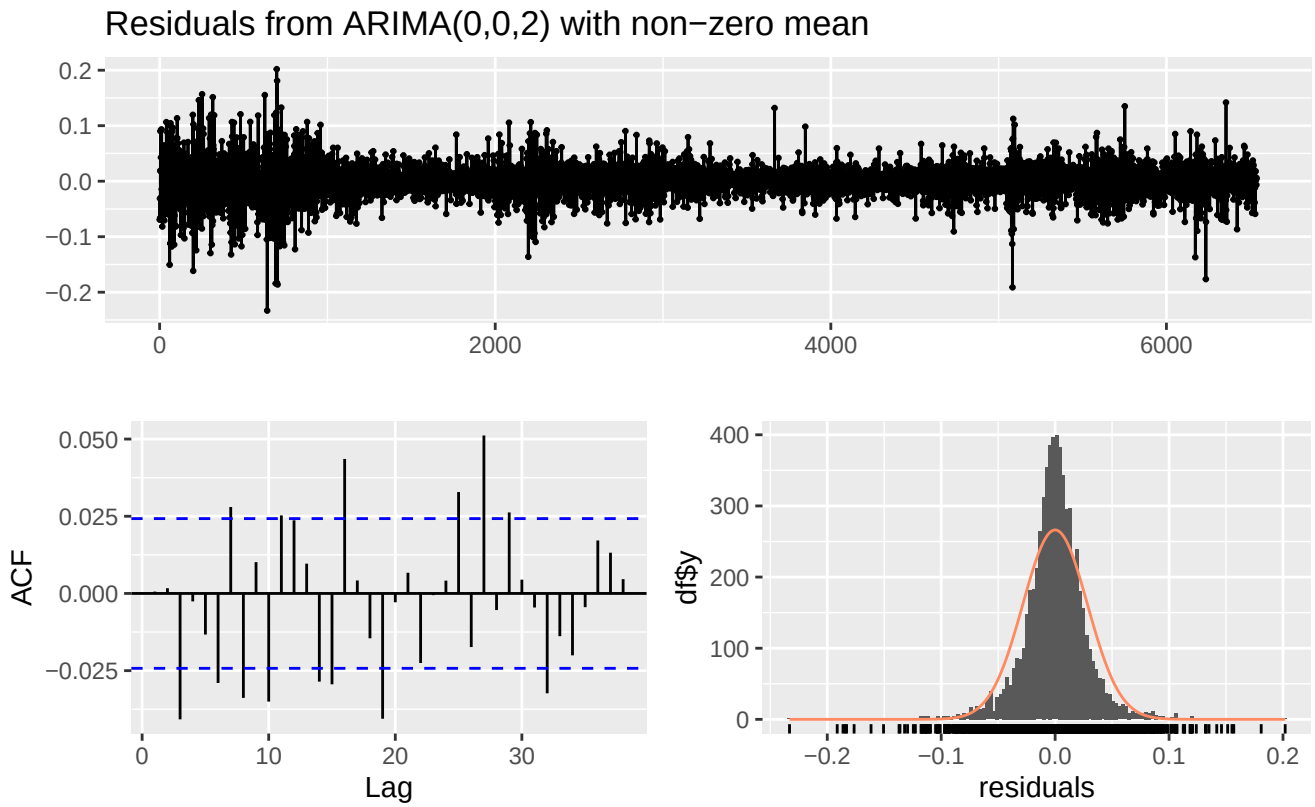
```
coeftest(arma_20)
```

```
z test of coefficients:

              Estimate  Std. Error z value Pr(>|z|)
ar1       -0.03811219  0.01237289 -3.0803 0.002068 **
ar2       -0.01178765  0.01237261 -0.9527 0.340731
intercept  0.00051767  0.00033491  1.5457 0.122176
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.5 MA(2)

```
arma_02 <- Arima(log_returns$LogReturns, order=c(0,0,2))
checkresiduals(arma_02)
```



```
    Ljung-Box test

data:  Residuals from ARIMA(0,0,2) with non-zero mean
Q* = 38.929, df = 8, p-value = 5.066e-06

Model df: 2.   Total lags used: 10
```

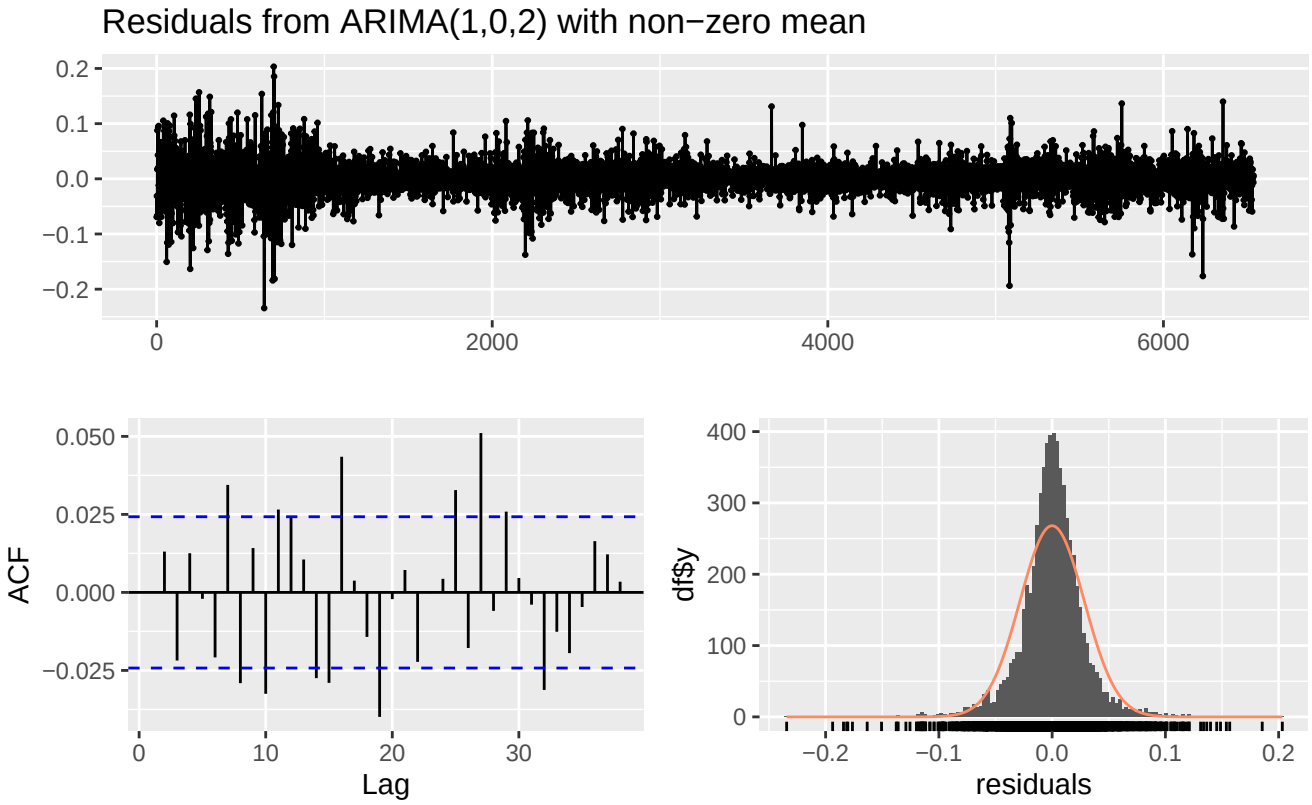
```r
coeftest(arma_02)
```

```
z test of coefficients:

             Estimate  Std. Error z value Pr(>|z|)
ma1       -0.03931359  0.01239570 -3.1716 0.001516 **
ma2       -0.01363511  0.01243057 -1.0969 0.272685
intercept  0.00051676  0.00033299  1.5519 0.120694
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.6 ARMA(1,2)

```r
arma_12 <- Arima(log_returns$LogReturns, order=c(1,0,2))
checkresiduals(arma_12)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(1,0,2) with non-zero mean
Q* = 29.667, df = 7, p-value = 0.0001093

Model df: 3.   Total lags used: 10
```

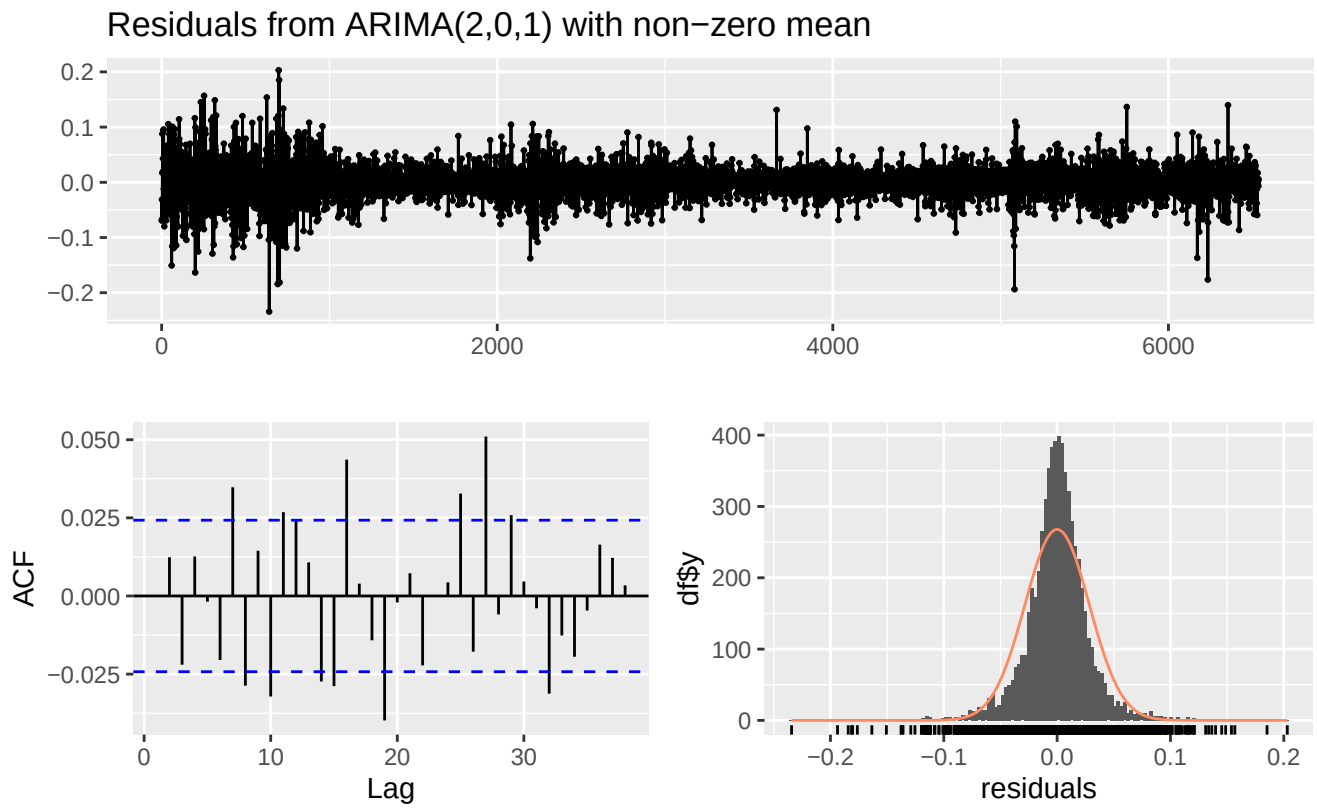
```
coeftest(arma_12)
```

```
z test of coefficients:

             Estimate  Std. Error z value  Pr(>|z|)
ar1        0.74660442  0.12291149  6.0743 1.245e-09 ***
ma1       -0.78624421  0.12342306 -6.3703 1.886e-10 ***
ma2        0.00387487  0.01562497  0.2480    0.8041
intercept  0.00052359  0.00030184  1.7346    0.0828 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.7 ARMA(2,1)

```
arma_21 <- Arima(log_returns$LogReturns, order=c(2,0,1))
checkresiduals(arma_21)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(2,0,1) with non-zero mean
Q* = 29.433, df = 7, p-value = 0.0001206

Model df: 3.   Total lags used: 10
```

```
coeftest(arma_21)
```

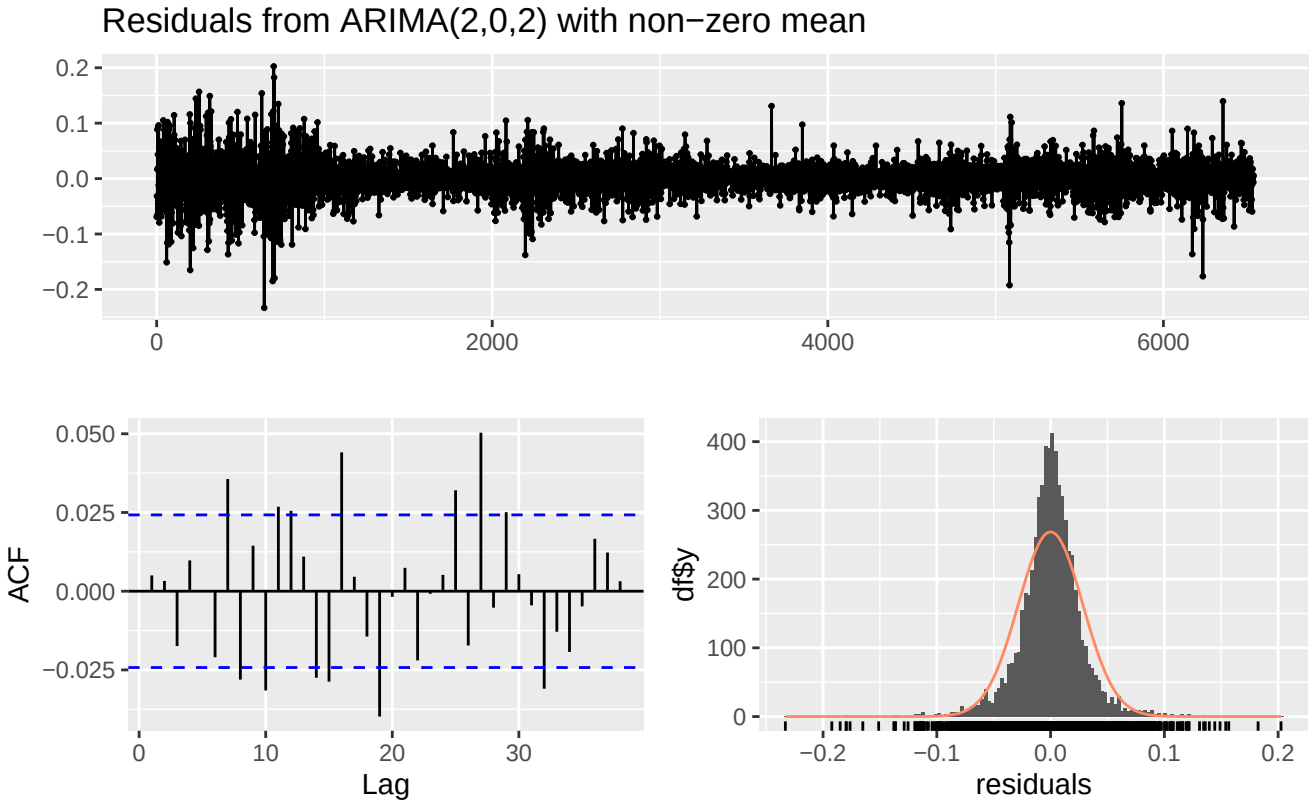
```
z test of coefficients:

             Estimate  Std. Error z value  Pr(>|z|)
ar1        0.75216134  0.10514098  7.1538 8.439e-13 ***
```

```
ar2         0.00505353  0.01520891  0.3323   0.73968
ma1        -0.79218013  0.10452116 -7.5791 3.479e-14 ***
intercept   0.00052281  0.00030083  1.7379   0.08223 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.8 ARMA(2,2)

```
arma_22 <- Arima(log_returns$LogReturns, order=c(2,0,2))
checkresiduals(arma_22)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(2,0,2) with non-zero mean
Q* = 27.067, df = 6, p-value = 0.0001407

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

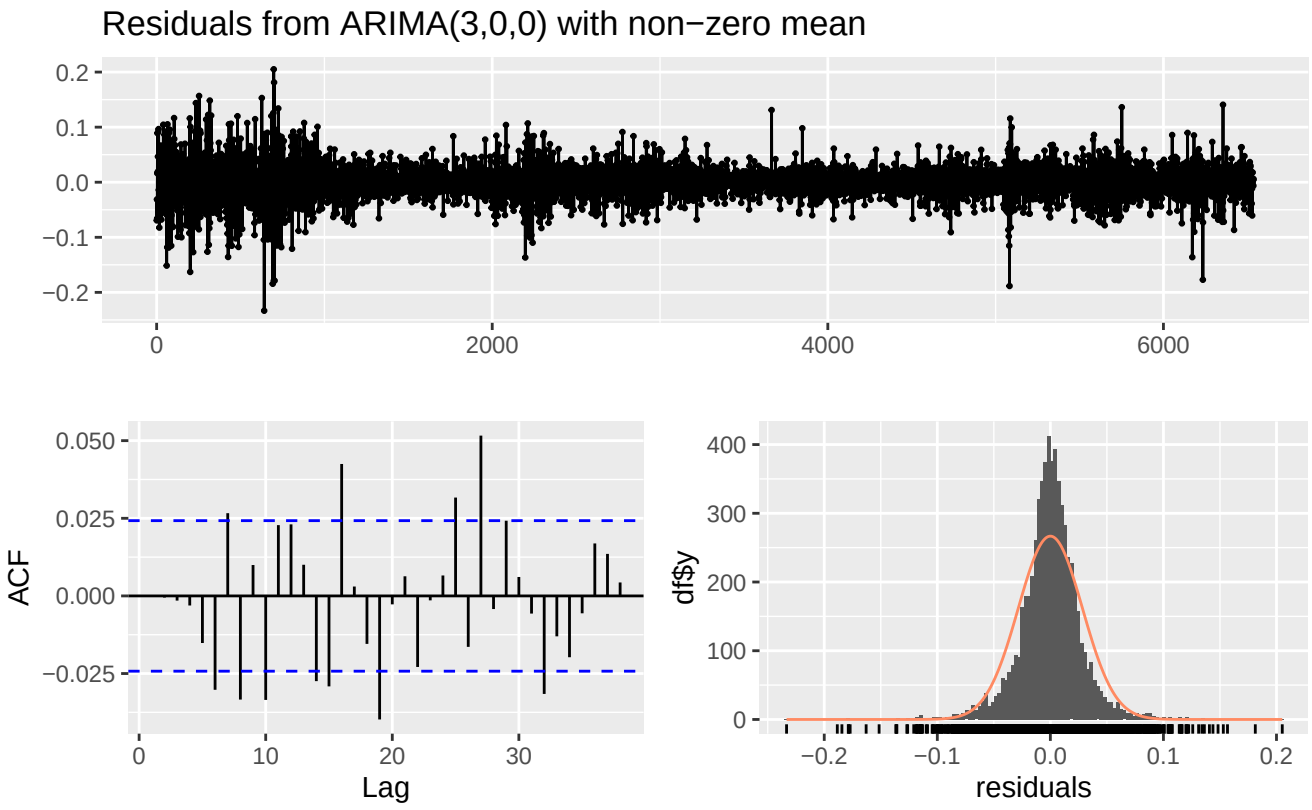
```
coeftest(arma_22)
```

```
z test of coefficients:

              Estimate  Std. Error z value  Pr(>|z|)
ar1         0.13889774  0.20808368  0.6675 0.5044470
ar2         0.49269101  0.13447436  3.6638 0.0002485 ***
ma1        -0.18349571  0.20966818 -0.8752 0.3814804
ma2        -0.50222839  0.14273213 -3.5187 0.0004337 ***
intercept   0.00052958  0.00029978  1.7666 0.0772978 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.9 AR(3)

```
arma_30 <- Arima(log_returns$LogReturns, order=c(3,0,0))
checkresiduals(arma_30)
```



```
    Ljung-Box test

data:  Residuals from ARIMA(3,0,0) with non-zero mean
Q* = 27.506, df = 7, p-value = 0.0002702

Model df: 3.   Total lags used: 10
```

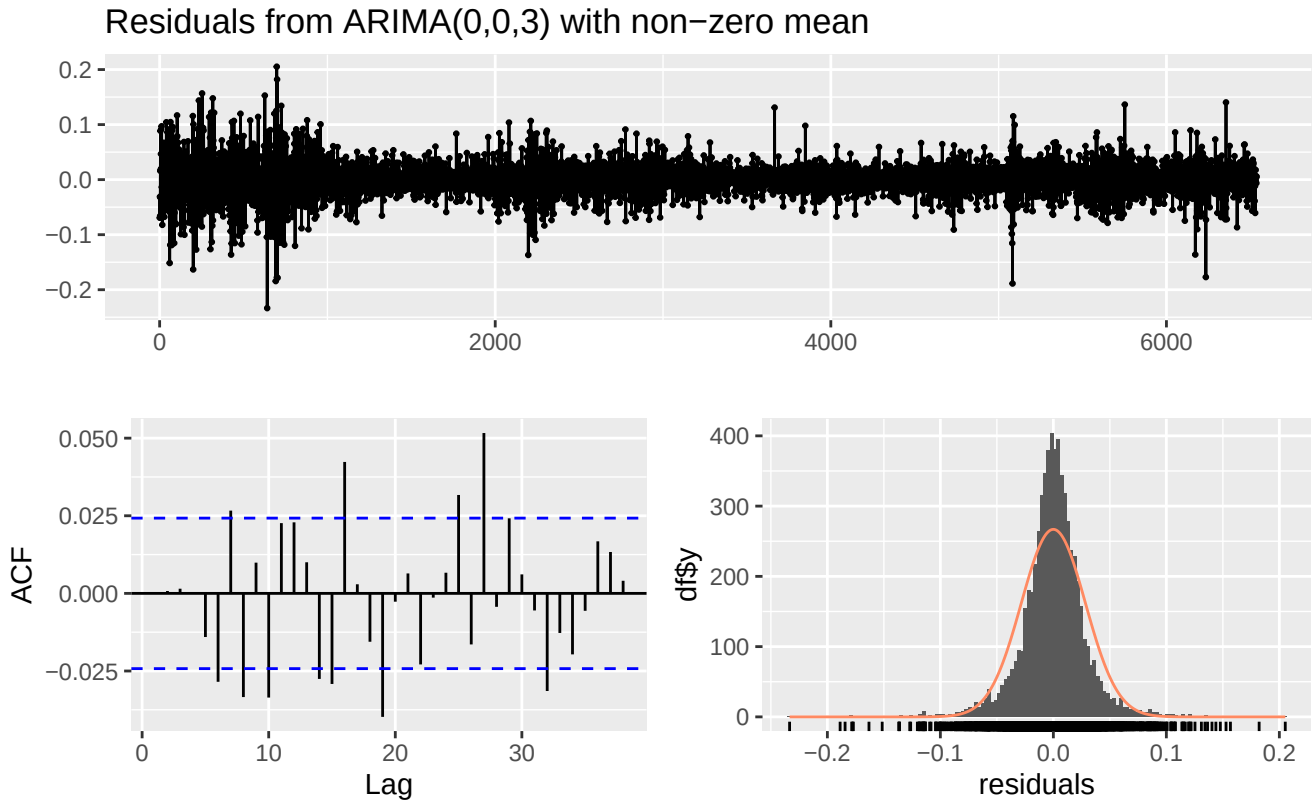
```
coeftest(arma_30)
```

```
z test of coefficients:

             Estimate  Std. Error z value  Pr(>|z|)
ar1       -0.03863782  0.01236319 -3.1252 0.0017767 **
ar2       -0.01340434  0.01237126 -1.0835 0.2785840
ar3       -0.04157144  0.01236735 -3.3614 0.0007755 ***
intercept  0.00051836  0.00032127  1.6135 0.1066419
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.10 MA(3)

```
arma_03 <- Arima(log_returns$LogReturns, order=c(0,0,3))
checkresiduals(arma_03)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(0,0,3) with non-zero mean
Q* = 26.596, df = 7, p-value = 0.000394

Model df: 3.   Total lags used: 10
```

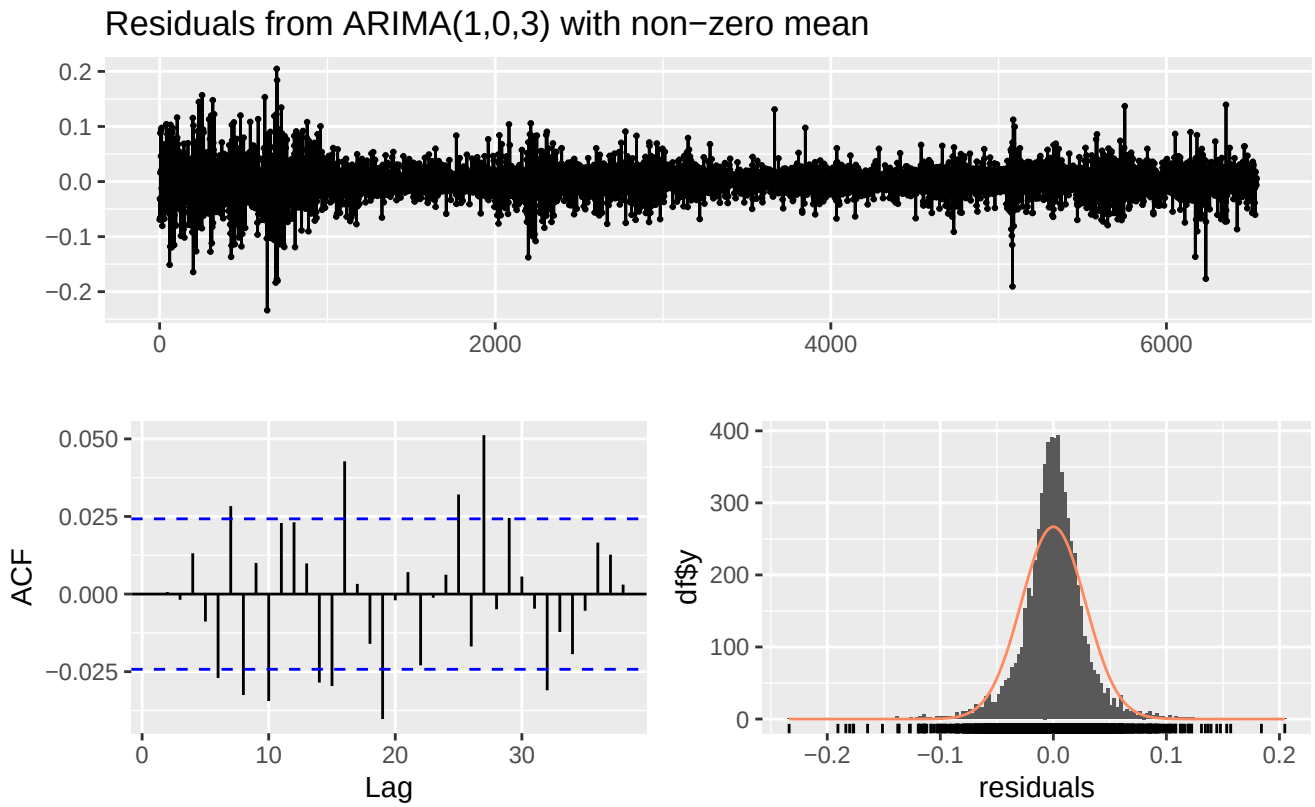
```
coeftest(arma_03)
```

```
z test of coefficients:

              Estimate  Std. Error z value  Pr(>|z|)
ma1       -0.03884123  0.01237078 -3.1398 0.0016909 **
ma2       -0.01337127  0.01235618 -1.0822 0.2791849
ma3       -0.04358374  0.01269756 -3.4325 0.0005982 ***
intercept  0.00051686  0.00031767  1.6270 0.1037265
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.11 ARMA(1,3)

```
arma_13 <- Arima(log_returns$LogReturns, order=c(1,0,3))
checkresiduals(arma_13)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(1,0,3) with non-zero mean
Q* = 27.07, df = 6, p-value = 0.0001405

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

```
coeftest(arma_13)
```

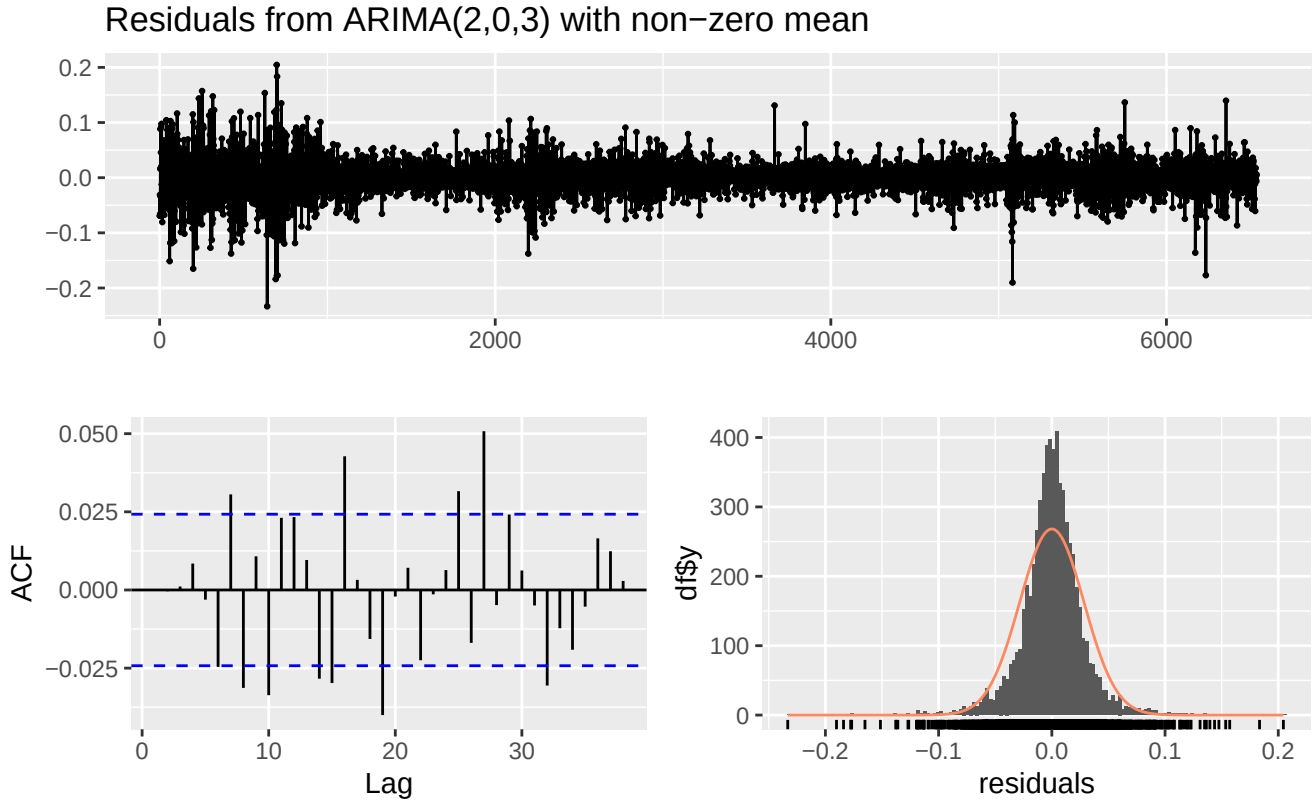
```
Warning in sqrt(diag(se)): NaNs produced


z test of coefficients:
```

```
              Estimate  Std. Error z value Pr(>|z|)
ar1         0.33895106         NaN     NaN      NaN
ma1        -0.37787376         NaN     NaN      NaN
ma2        -0.00033557         NaN     NaN      NaN
ma3        -0.03535024         NaN     NaN      NaN
intercept   0.00051530  0.00031168  1.6533  0.09828 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.12 ARMA(2,3)

```
arma_23 <- Arima(log_returns$LogReturns, order=c(2,0,3))
checkresiduals(arma_23)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(2,0,3) with non-zero mean
Q* = 25.221, df = 5, p-value = 0.0001263
```

```
Model df: 5.   Total lags used: 10
```

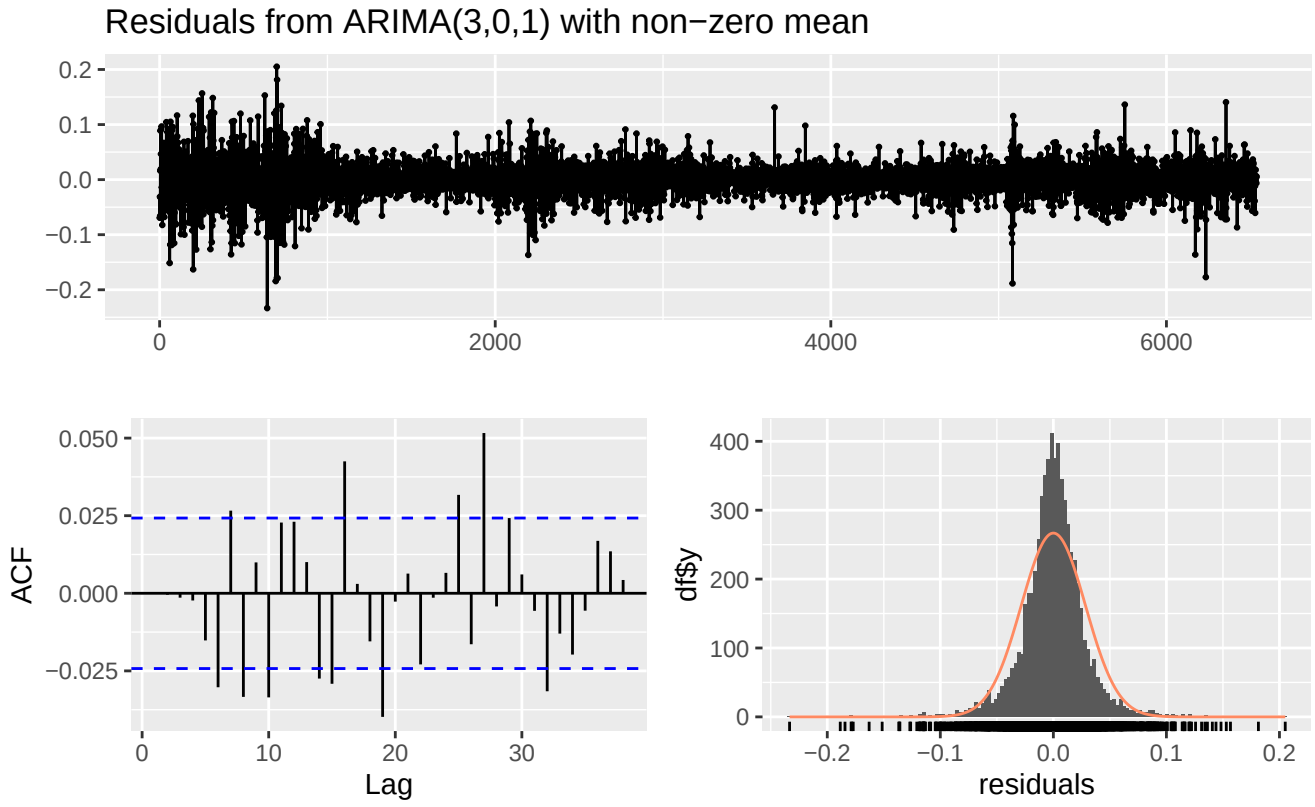
```
coeftest(arma_23)
```

```
z test of coefficients:

             Estimate  Std. Error z value Pr(>|z|)
ar1        0.12928859  0.26476852  0.4883  0.62533
ar2        0.22589080  0.19486455  1.1592  0.24637
ma1       -0.16826815  0.26487623 -0.6353  0.52525
ma2       -0.23309281  0.19068257 -1.2224  0.22155
ma3       -0.03297002  0.01859016 -1.7735  0.07614 .
intercept  0.00050770  0.00030819  1.6474  0.09948 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.13 ARMA(3,1)

```
arma_31 <- Arima(log_returns$LogReturns, order=c(3,0,1))
checkresiduals(arma_31)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(3,0,1) with non-zero mean
Q* = 27.5, df = 6, p-value = 0.0001167

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

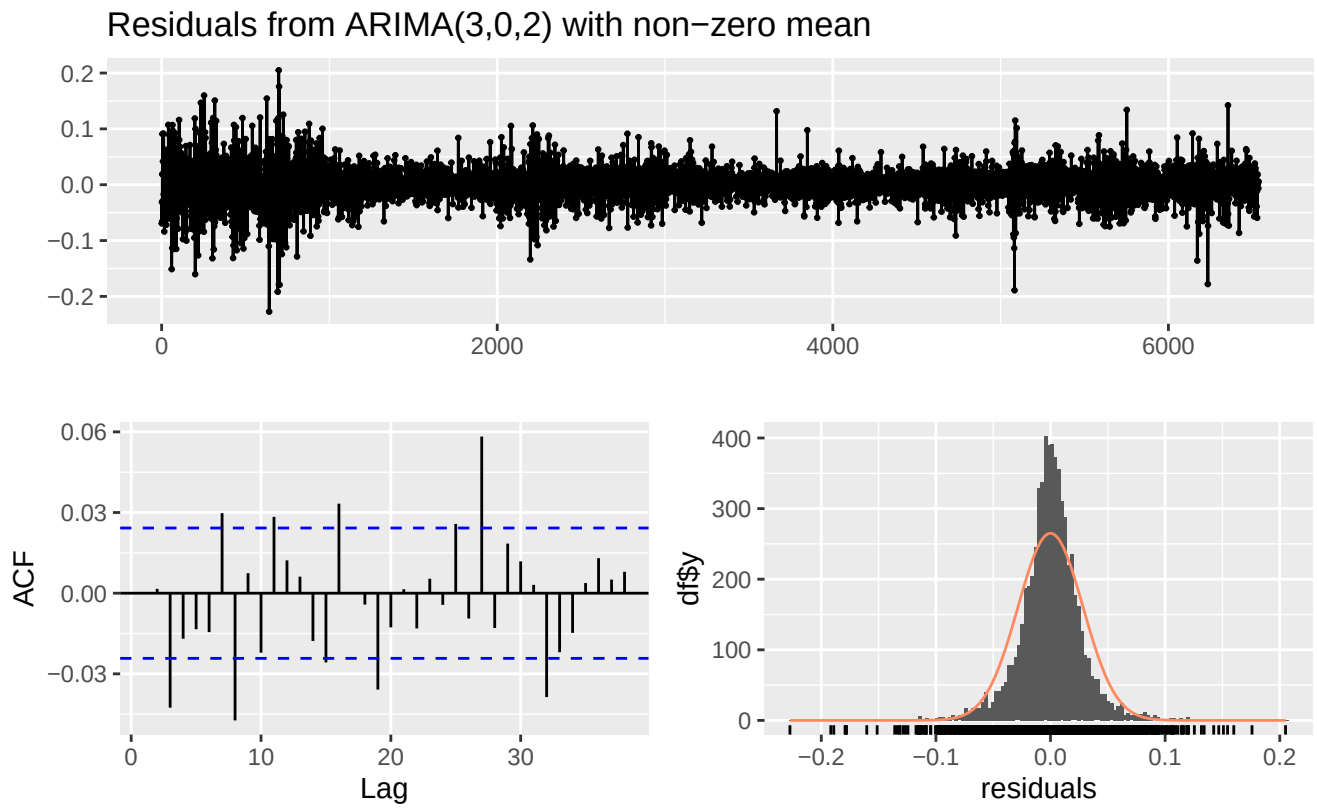
```
coeftest(arma_31)
```

```
z test of coefficients:

              Estimate  Std. Error z value Pr(>|z|)
ar1       -0.01946408  0.55210686 -0.0353 0.971877
ar2       -0.01274576  0.02435614 -0.5233 0.600760
ar3       -0.04139634  0.01342376 -3.0838 0.002044 **
ma1       -0.01920247  0.55288877 -0.0347 0.972294
intercept  0.00051900  0.00032097  1.6170 0.105887
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.14 ARMA(3,2)

```
arma_32 <- Arima(log_returns$LogReturns, order=c(3,0,2))
checkresiduals(arma_32)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(3,0,2) with non-zero mean
Q* = 40.334, df = 5, p-value = 1.279e-07

Model df: 5.   Total lags used: 10
```

```
coeftest(arma_32)
```

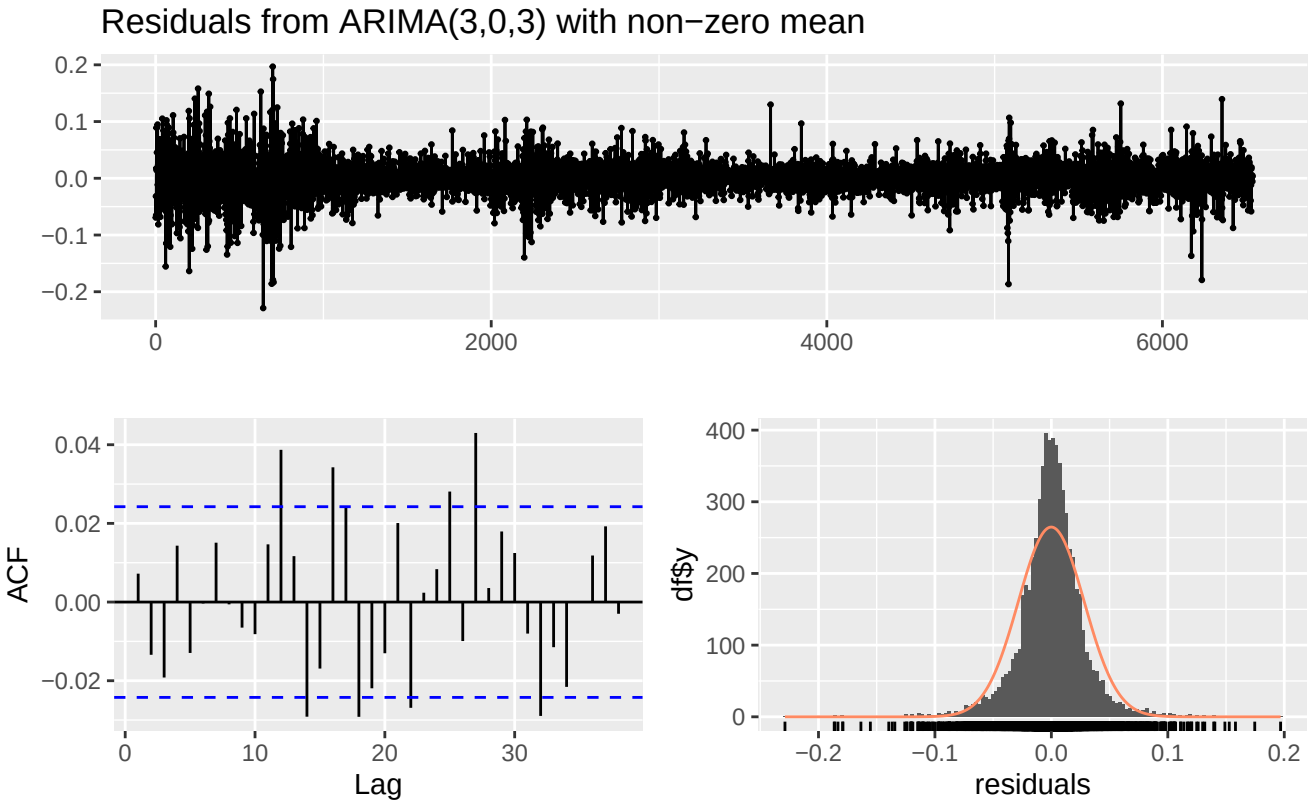
```
z test of coefficients:

             Estimate  Std. Error  z value  Pr(>|z|)
ar1        0.03147753  0.02132298   1.4762  0.139883
```

```
ar2        -0.97055711  0.01697418 -57.1784 < 2.2e-16 ***
ar3        -0.03525799  0.01280948  -2.7525  0.005914 **
ma1        -0.06976751  0.01731179  -4.0301 5.576e-05 ***
ma2         0.96109526  0.02058307  46.6935 < 2.2e-16 ***
intercept   0.00053285  0.00033637   1.5841  0.113169
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.15 ARMA(3,3)

```
arma_33 <- Arima(log_returns$LogReturns, order=c(3,0,3))
checkresiduals(arma_33)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(3,0,3) with non-zero mean
Q* = 8.5727, df = 4, p-value = 0.07271

Model df: 6.   Total lags used: 10
```

```r
coeftest(arma_33)
```

```
Warning in sqrt(diag(se)): NaNs produced
```

```
z test of coefficients:

             Estimate  Std. Error  z value Pr(>|z|)
ar1       -0.93303265         NaN      NaN      NaN
ar2        0.68686580  0.06141265  11.1844  < 2e-16 ***
ar3        0.81111732  0.07456554  10.8779  < 2e-16 ***
ma1        0.89063594         NaN      NaN      NaN
ma2       -0.72881204  0.05908310 -12.3354  < 2e-16 ***
ma3       -0.80508569  0.07150341 -11.2594  < 2e-16 ***
intercept  0.00060076  0.00028783   2.0872  0.03687 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As observed from the outputs, the simpler models (such as AR(1) and MA(1)) yield very low p-values for the Ljung-Box test (< 0.05). This indicates that significant autocorrelation remains within the residuals, implying that these models have failed to capture all the relevant information.

Conversely, as we increase complexity towards an **ARMA(3,3)**, the p-value rises above the critical 5% threshold. Consequently, we fail to reject the null hypothesis: the residuals of this model behave as **White Noise**. This suggests that the ARMA(3,3) is mathematically effective in capturing the patterns of the time series, although its high complexity warrants caution regarding the risk of **overfitting**.

Additionally, the **ARMA(1,3)** model exhibits instability in parameter estimation (incalculable Std. Error), pointing to potential **over-parameterization**.

2.3 AIC vs BIC

We visualize the trends of the information criteria for all estimated models.

```r
models_list <- list(
  "AR(1)" = arma_10, "MA(1)" = arma_01, "ARMA(1,1)" = arma_11,
  "AR(2)" = arma_20, "MA(2)" = arma_02, "ARMA(1,2)" = arma_12,
  "ARMA(2,1)" = arma_21, "ARMA(2,2)" = arma_22,
  "AR(3)" = arma_30, "MA(3)" = arma_03, "ARMA(1,3)" = arma_13,
  "ARMA(2,3)" = arma_23, "ARMA(3,1)" = arma_31, "ARMA(3,2)" = arma_32,
  "ARMA(3,3)" = arma_33
)
```

```r
metrics_df <- data.frame(
  Model = names(models_list),
  AIC = sapply(models_list, AIC),
  BIC = sapply(models_list, BIC)
)

metrics_df$Model <- factor(metrics_df$Model, levels = metrics_df$Model)

min_aic_val <- min(metrics_df$AIC)
min_bic_val <- min(metrics_df$BIC)

plot_aic <- plot_ly(metrics_df, x = ~Model, y = ~AIC, name = 'AIC',
  type = 'scatter', mode = 'lines+markers',
  line = list(color = '#1f77b4', width = 2),
  marker = list(size = 8, color = '#1f77b4')) %>%
  layout(title = "AIC Trend", yaxis = list(title = "AIC"))

best_aic <- metrics_df[metrics_df$AIC == min_aic_val, ]
plot_aic <- plot_aic %>%
  add_trace(data = best_aic, x = ~Model, y = ~AIC,
    type = 'scatter', mode = 'markers', name = 'Best AIC',
    marker = list(color = 'red', size = 12, symbol = 'star'),
    showlegend = FALSE
  )

plot_bic <- plot_ly(metrics_df, x = ~Model, y = ~BIC, name = 'BIC',
  type = 'scatter', mode = 'lines+markers',
  line = list(color = '#ff7f0e', width = 2),
  marker = list(size = 8, color = '#ff7f0e')) %>%
  layout(title = "BIC Trend", yaxis = list(title = "BIC"))

best_bic <- metrics_df[metrics_df$BIC == min_bic_val, ]
plot_bic <- plot_bic %>%
  add_trace(
    data = best_bic, x = ~Model, y = ~BIC,
    type = 'scatter', mode = 'markers', name = 'Best BIC',
    marker = list(color = 'red', size = 12, symbol = 'star'),
    showlegend = FALSE
  )

final_plot <- subplot(plot_aic, plot_bic, nrows = 2, shareX = TRUE, titleY = TRUE) %>%
  layout(
    title = "Model Selection Criteria: AIC & BIC Trends",
    hovermode = "x unified",
    margin = list(t = 50)
```

```
  )

final_plot
```

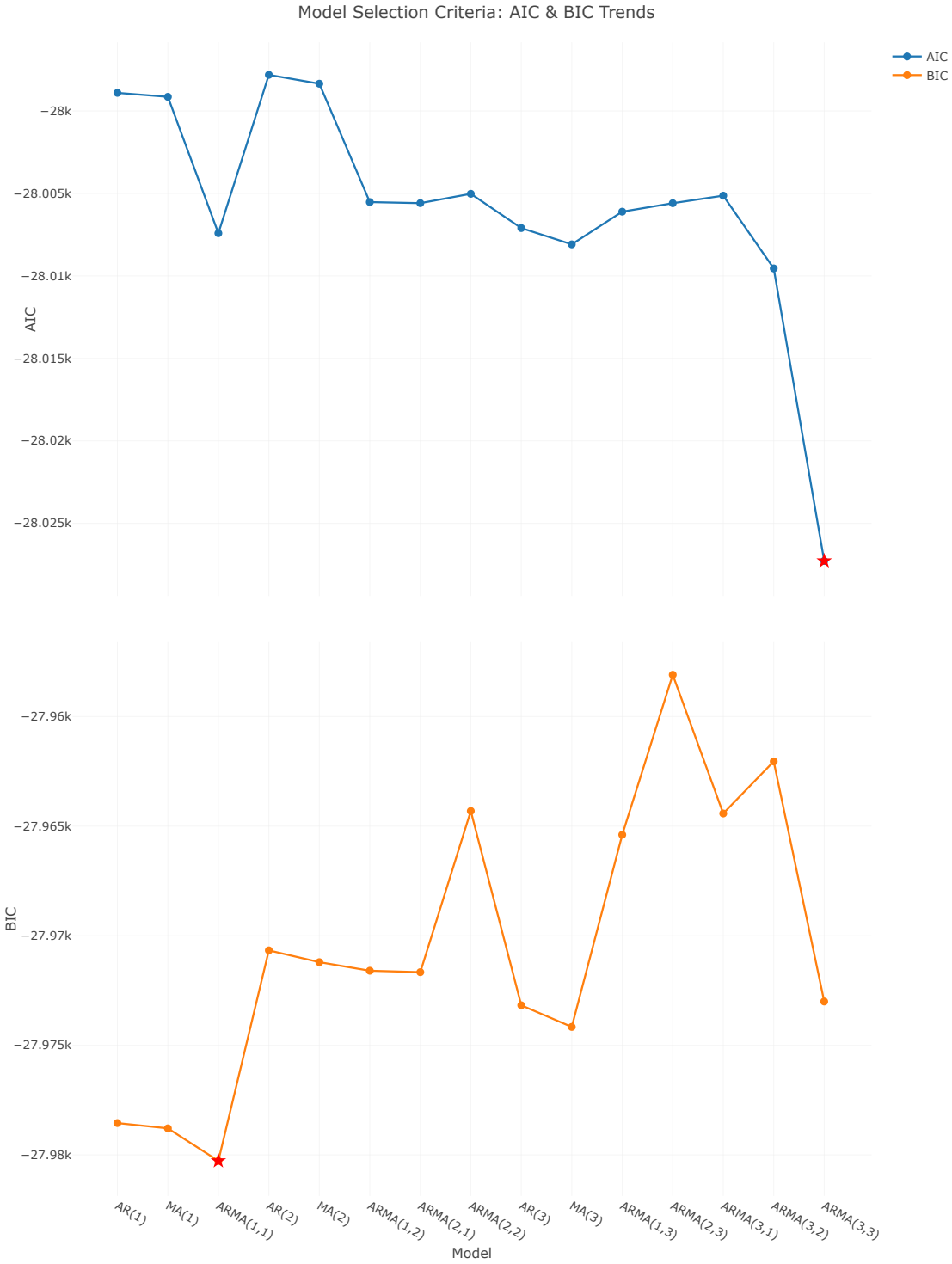


Figure 2.2: Comparison between AIC and BIC

The graphical analysis of the information criteria highlights a typical divergence found in financial time series modeling.

As evidenced by the plot, the **AIC** criterion, which prioritizes **goodness of fit**, identifies the **ARMA(3,3)** as the optimal model (minimum value). Conversely, the **BIC** criterion, which penalizes model complexity more severely to prevent overfitting, suggests a much more **parsimonious** model: the **ARMA(1,1)**.

Faced with these conflicting indications (a complex model vs a simple model), it is not possible to determine a priori which one is superior. To resolve this ambiguity and assess which model generalizes better on unobserved data, we will proceed with an **Out-of-Sample validation**, computing forecasts and comparing error metrics against actual data.

2.4 Forecast

Since the information criteria (AIC and BIC) suggest different models, we proceed with an Out-of-Sample validation to test the actual predictive capacity on data not used for estimation.

We split the time series into two subsets:

- **Training Set** (80%): Used to estimate the model parameters.
- **Test Set** (20%): Used to compare forecasts against actual data (h-step-ahead forecast).

```
n_total <- nrow(log_returns)
n_train <- floor(0.80 * n_total)

train_data <- log_returns$LogReturns[1:n_train]
test_data <- log_returns$LogReturns[(n_train + 1):n_total]

set.seed(123)

mod_winner_bic <- Arima(train_data, order=c(1,0,1))
mod_winner_aic <- Arima(train_data, order=c(3,0,3))
mod_naive <- Arima(train_data, order=c(0,0,0))

h_steps <- length(test_data)

fc_bic <- forecast(mod_winner_bic, h=h_steps)
fc_aic <- forecast(mod_winner_aic, h=h_steps)
fc_naive <- forecast(mod_naive, h=h_steps)
```

The following plot displays the time series trend and the forecasts generated by the three models over the test period.

> **Note**
>
> Given the stationary nature and high volatility of log-returns, it is expected that medium-to-long-term forecasts will tend to converge rapidly towards the unconditional mean of the process. For this reason we apply method as **one-step-ahead** and **k-step-ahead**.

2.4.1 One-Steap-Ahead

```
fit_bic_rolling <- Arima(log_returns$LogReturns, model = mod_winner_bic)
one_step_bic <- fitted(fit_bic_rolling)[(n_train + 1):n_total]

fit_aic_rolling <- Arima(log_returns$LogReturns, model = mod_winner_aic)
one_step_aic <- fitted(fit_aic_rolling)[(n_train + 1):n_total]

dates_test  <- log_returns$Date[(n_train + 1):n_total]

fig_roll <- plot_ly(
  x = dates_test,
  y = test_data,
  type = 'scatter',
  mode = 'lines',
  name = 'Actual Test Data',
  line = list(color = 'black', width = 1, dash = 'dot')
)

fig_roll <- add_trace(
  fig_roll,
  x = dates_test,
  y = as.numeric(one_step_bic),
  name = 'Rolling ARMA(1,1)',
  line = list(color = 'blue', width = 1)
)

fig_roll <- add_trace(
  fig_roll,
  x = dates_test,
  y = as.numeric(one_step_aic),
  name = 'Rolling ARMA(3,3)',
  line = list(color = 'red', width = 1)
)

fig_roll <- layout(
  fig_roll,
  title = "One-Step-Ahead Rolling Forecast (Test Set: 20%)",
```

```
  xaxis = list(title = "Date"),
  yaxis = list(title = "Log Return"),
  legend = list(orientation = "h", x = 0.1, y = -0.2)
)

fig_roll
```

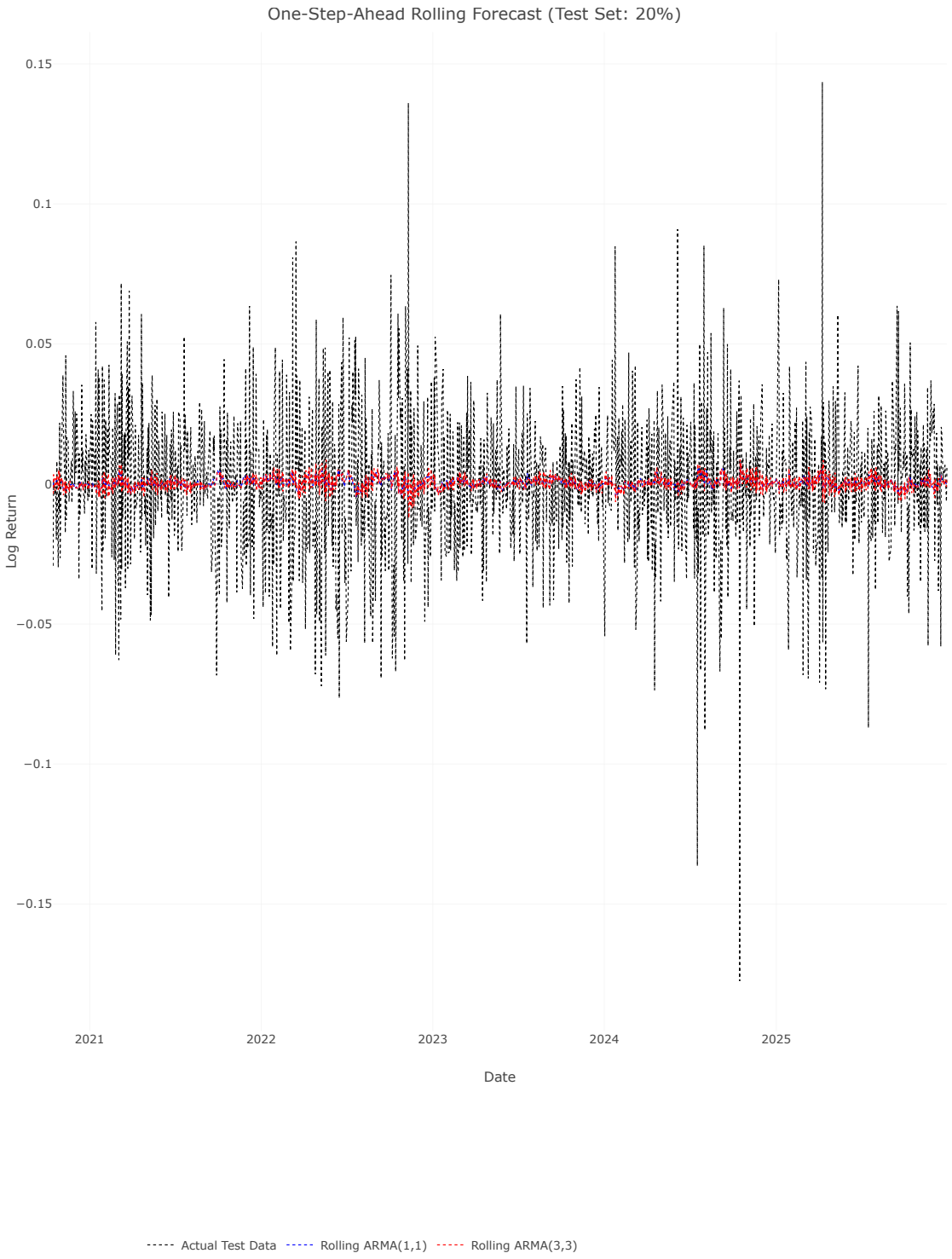


Figure 2.3: Comparison One-Step-Ahead Rolling Forecast

2.4.2 Five-Steap-Ahead

```
calc_k_step_rolling <- function(model_obj, series, n_train, k=5) {
  n_total <- length(series)
  test_indices <- (n_train + 1):n_total
  forecasts <- numeric(length(test_indices))

  for (i in seq_along(test_indices)) {
    target_idx <- test_indices[i]
    origin_idx <- target_idx - k

    if (origin_idx > 0) {
      subset_data <- series[1:origin_idx]
      fit_temp <- Arima(subset_data, model = model_obj)

      fc_temp <- forecast(fit_temp, h = k)

      forecasts[i] <- as.numeric(fc_temp$mean[k])
    } else {
      forecasts[i] <- NA
    }
  }
  return(forecasts)
}

k_horizon <- 5
vec_5step_bic <- calc_k_step_rolling(mod_winner_bic, log_returns$LogReturns, n_train, k = k_horiz
vec_5step_aic <- calc_k_step_rolling(mod_winner_aic, log_returns$LogReturns, n_train, k = k_horiz

dates_test <- log_returns$Date[(n_train + 1):n_total]

fig_roll_5 <- plot_ly(
  x = dates_test,
  y = test_data,
  type = 'scatter',
  mode = 'lines',
  name = 'Actual Test Data',
  line = list(color = 'black', width = 1, dash = 'dot')
)

fig_roll_5 <- add_trace(
  fig_roll_5,
  x = dates_test,
  y = vec_5step_bic,
  name = "Rolling ARMA(1,1)",
```

```
  line = list(color = 'blue', width = 1.5)
)

fig_roll_5 <- add_trace(
  fig_roll_5,
  x = dates_test,
  y = vec_5step_aic,
  name = "Rolling ARMA(3,3)",
  line = list(color = 'red', width = 1.5)
)

fig_roll_5 <- layout(
  fig_roll_5,
  title = "Five-Step-Ahead Rolling Forecast (Test Set: 20%)",
  xaxis = list(title = "Date"),
  yaxis = list(title = "Log Return"),
  legend = list(orientation = "h", x = 0.1, y = -0.2)
)

fig_roll_5
```

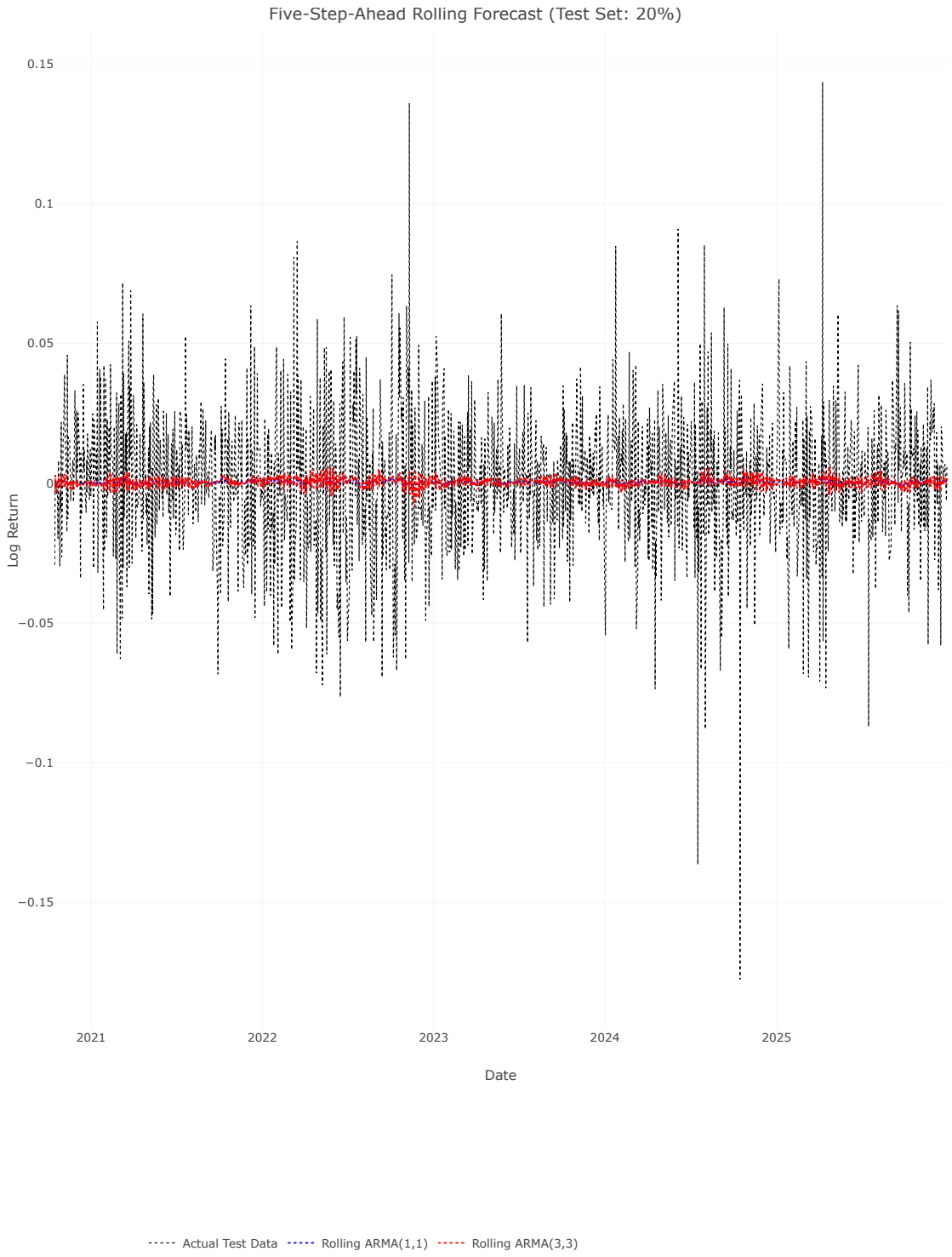


Figure 2.4: Comparison Five-Step-Ahead Rolling Forecast

We evaluate the predictive performance using:

- **MAE (Mean absolute Error)**
- **RMSE (Root Mean Squared Error)**
- **Theil's U2 index:** The ratio between the model's RMSE and the Naive model's RMSE.
  - $U < 1$: The model outperforms the benchmark.
  - $U \approx 1$: The model performs equivalently to the benchmark.
  - $U > 1$: The model performs worse than the benchmark.

```
get_metrics_robust <- function(forecast_obj, actuals) {
  vec_pred <- as.numeric(forecast_obj$mean)
  vec_obs <- as.numeric(actuals)

  acc <- accuracy(vec_pred, vec_obs)

  return(c(RMSE = acc[1, "RMSE"], MAE = acc[1, "MAE"]))
}

res_bic <- get_metrics_robust(fc_bic, test_data)
res_aic <- get_metrics_robust(fc_aic, test_data)
res_naive <- get_metrics_robust(fc_naive, test_data)

u_bic <- res_bic["RMSE"] / res_naive["RMSE"]
u_aic <- res_aic["RMSE"] / res_naive["RMSE"]
u_naive <- 1.0

validation_table <- data.frame(
  Model = c("ARMA(1,1) [BIC Winner]", "ARMA(3,3) [AIC Winner]", "Naive (Benchmark)"),
  RMSE = c(res_bic["RMSE"], res_aic["RMSE"], res_naive["RMSE"]),
  MAE = c(res_bic["MAE"], res_aic["MAE"], res_naive["MAE"]),
  Theil_U2 = c(u_bic, u_aic, u_naive)
)

datatable(validation_table, options = list(dom = 't'), rownames = FALSE)
```

Model	RMSE	MAE	Theil_U2
ARMA(1,1) [BIC Winner]	0.02595553770781629	0.01907402960434227	0.9999467426618117
ARMA(3,3) [AIC Winner]	0.02595187999168085	0.01907078148251036	0.9998058277874452
Naive (Benchmark)	0.02595692010428861	0.01907580888693115	1

Figure 2.5: Comparison of out-of-sample error metrics

As highlighted in the table, the **Theil's U2 index** for both ARMA models is extremely close to 1. This indicates that the statistical models, despite their complexity, fail to significantly outperform the simple forecast based on the historical mean (Naive Benchmark).

> **!** Important
>
> This result is consistent with the **Efficient Market Hypothesis (EMH)**, according to which past returns contain little to no useful information for predicting future short-term returns, making the series resemble a Random Walk.

2.5 Conclusion

The analysis conducted in this chapter has highlighted the limitations of stationary linear models in forecasting the direction of ASML returns.

Although information criteria (AIC) identified the ARMA(3,3) as a mathematical structure capable of "whitening" the in-sample residuals, the *Out-of-Sample* validation delivered an unequivocal verdict: **a Theil's U index close to 1 confirms that no ARMA model systematically outperforms the Naive benchmark.** This empirical result corroborates the hypothesis that, regarding the **conditional mean**, returns follow a process closely resembling a Random Walk, rendering trading strategies based solely on past autocorrelation futile.

However, the unpredictability of the mean does not imply a total absence of structure within the series. Visual inspection of the log-returns plot reveals evident **volatility clustering**: periods of high volatility tend to be followed by high volatility, and periods of calm by calm (a phenomenon known as *conditional heteroskedasticity*).

By assuming constant variance over time (homoskedasticity), ARMA models fail to capture this fundamental characteristic of financial markets. Therefore, in the next chapter, we will shift our focus from predicting the *sign* of returns to modeling their **magnitude** (risk) by introducing the **GARCH** (Generalized AutoRegressive Conditional Heteroskedasticity) family of models.

3 Volatility Analysis

In the previous chapter, we established that ASML returns exhibit a conditional mean behavior closely resembling a Random Walk, making directional forecasting ineffective. However, financial time series are characterized by a "stylized fact" known as **volatility clustering**: large changes tend to be followed by large changes, and small changes by small changes. This implies that while the *sign* of the return is unpredictable, its *magnitude* (risk) follows a recognizable pattern.

The objective of this chapter is to model and forecast the conditional variance to quantify the risk associated with the asset. To achieve this, we adopt a structured modeling framework:

1. **Detection of ARCH Effects:** We verify the presence of conditional heteroskedasticity through the analysis of squared returns' autocorrelation (**ACF**) and formal statistical tests such as the **ARCH-LM Test**.
2. **Asymmetry Testing:** We investigate whether the volatility reacts differently to positive versus negative shocks (leverage effect) using the **Sign Bias Test**, **Negative Size Bias Test**, **Positive Size Bias Test** and **Joint Effect**.
3. **Model Estimation & Selection:** We estimate and compare three classes of models to capture different volatility dynamics: **GARCH (Standard)** and **GJR-GARCH & E-GARCH**.
4. **Forecasting & Evaluation:** We assess the predictive power of the models using error metrics suitable for volatility (e.g., **RMSE** and **QLIKE**) and a visual comparison of the estimated conditional variance over time.
5. **Risk Management Application:** Finally, we translate the volatility forecasts into a practical risk metric by computing and plotting the **Value at Risk (VaR)**, a standard tool for quantifying potential losses in extreme market scenarios.

3.1 Exploratory Volatility Analysis

Let's conduct a visual inspection of the squared log-returns (r_t^2) and absolute log-returns ($|r_t|$), while raw returns typically show no correlation (as established in the previous chapter), their squared or absolute versions often exhibit strong persistence.

3.1.1 Squared Returns

```
fig_sq <- plot_ly(data = log_returns, x = ~Date, y = ~Squared, type = 'scatter',
  mode = 'lines', name = 'Squared Returns', line = list(color = 'darkred', width = 1))

fig_sq <- layout(
```

```
  fig_sq,
  title = "ASML Squared Log-Returns",
  xaxis = list(title = "Date"),
  yaxis = list(title = "Squared Log Return"),
  shapes = list(
    list(
      type = "line",
      x0 = min(log_returns$Date),
      x1 = max(log_returns$Date),
      y0 = 0,
      y1 = 0,
      line = list(color = "black", width = 1)
    )
  )
)

fig_sq
```

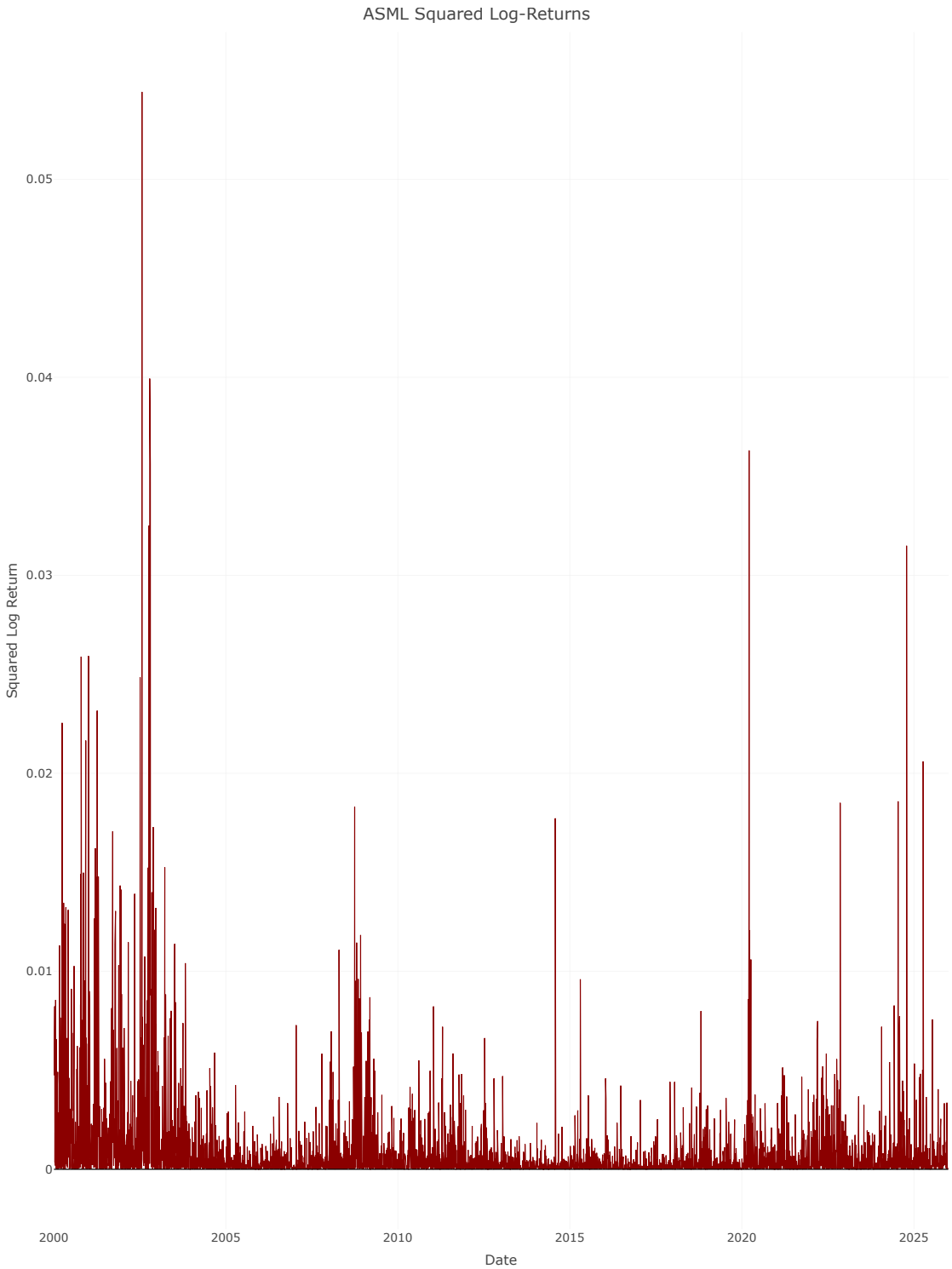


Figure 3.1: ASML Squared Daily Log-Returns

```r
acf_sq <- acf(log_returns$Squared, plot = FALSE, lag.max = 30)
pacf_sq <- pacf(log_returns$Squared, plot = FALSE, lag.max = 30)

ci <- qnorm((1 + 0.95)/2)/sqrt(length(log_returns$Squared))

df_acf_sq <- data.frame(
  lag = as.numeric(acf_sq$lag)[-1],
  acf = as.numeric(acf_sq$acf)[-1])

df_pacf_sq <- data.frame(
  lag = as.numeric(pacf_sq$lag),
  pacf = as.numeric(pacf_sq$acf))

fig_acf_sq <- plot_ly(df_acf_sq, x = ~lag, y = ~acf, type = 'bar', name = 'ACF',
  marker = list(color = '#1f77b4')) %>%

  add_segments(x = min(df_acf_sq$lag), xend = max(df_acf_sq$lag), y = ci, yend = ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  add_segments(x = min(df_acf_sq$lag), xend = max(df_acf_sq$lag), y = -ci, yend = -ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  layout(yaxis = list(title = "ACF"))

fig_pacf_sq <- plot_ly(df_pacf_sq, x = ~lag, y = ~pacf, type = 'bar', name = 'PACF',
  marker = list(color = '#ff7f0e')) %>%

  add_segments(x = min(df_pacf_sq$lag), xend = max(df_pacf_sq$lag), y = ci, yend = ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  add_segments(x = min(df_pacf_sq$lag), xend = max(df_pacf_sq$lag), y = -ci, yend = -ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  layout(yaxis = list(title = "PACF"))

subplot(fig_acf_sq, fig_pacf_sq, nrows = 1, margin = 0.05, titleY = TRUE) %>%
  layout(title = "Squared Returns: Autocorrelation & Partial Autocorrelation",
  margin = list(t = 50))
```

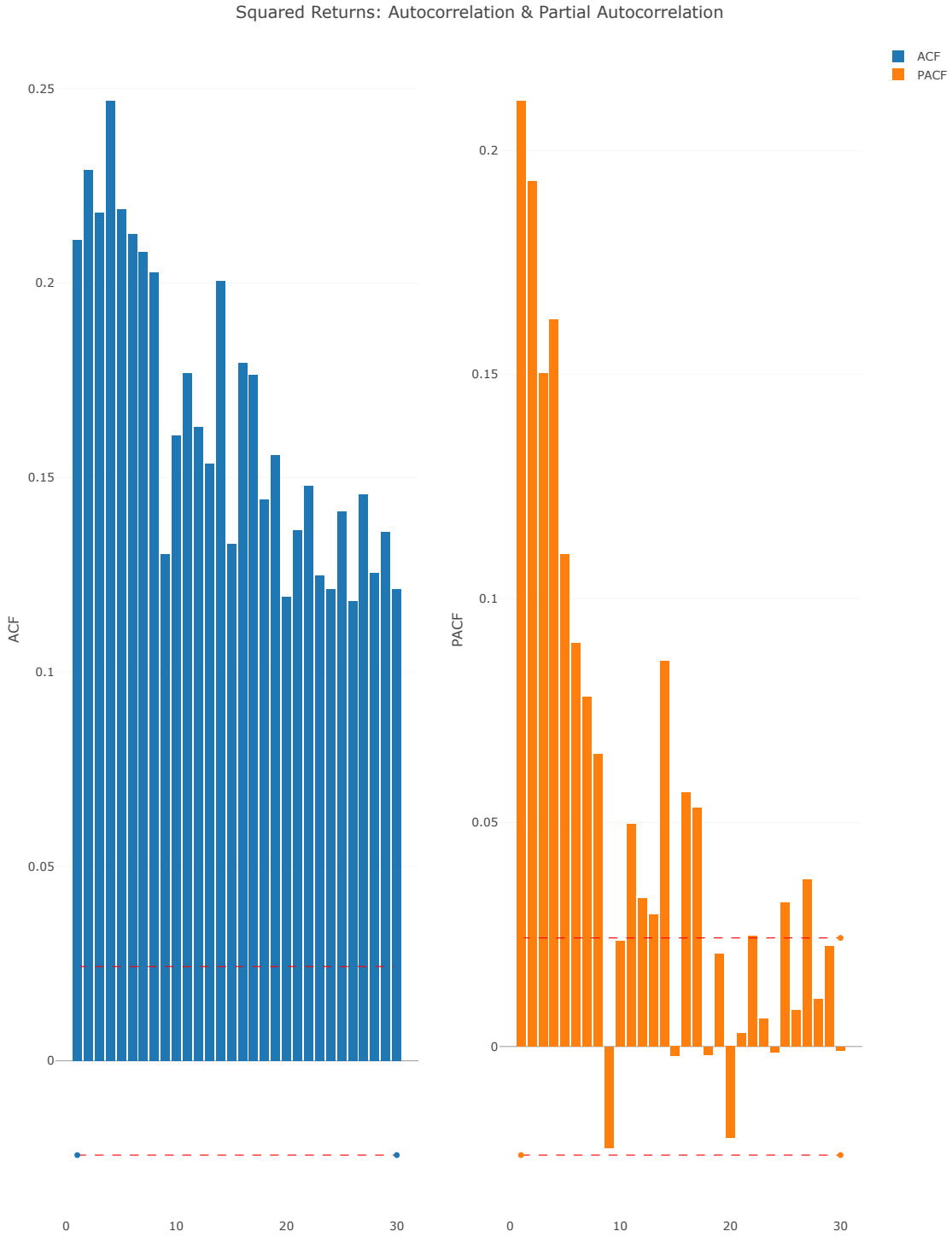


Figure 3.2: Squared Returns ACF and PACF

3.1.2 Absolute Returns

```
fig_sq <- plot_ly(data = log_returns, x = ~Date, y = ~Abs, type = 'scatter',
  mode = 'lines', name = 'Absolute Returns', line = list(color = 'darkred', width = 1))

fig_sq <- layout(
  fig_sq,
  title = "ASML Absolute Log-Returns",
  xaxis = list(title = "Date"),
  yaxis = list(title = "Absolute value Log Return"),
  shapes = list(
    list(
      type = "line",
      x0 = min(log_returns$Date),
      x1 = max(log_returns$Date),
      y0 = 0,
      y1 = 0,
      line = list(color = "black", width = 1)
    )
  )
)

fig_sq
```

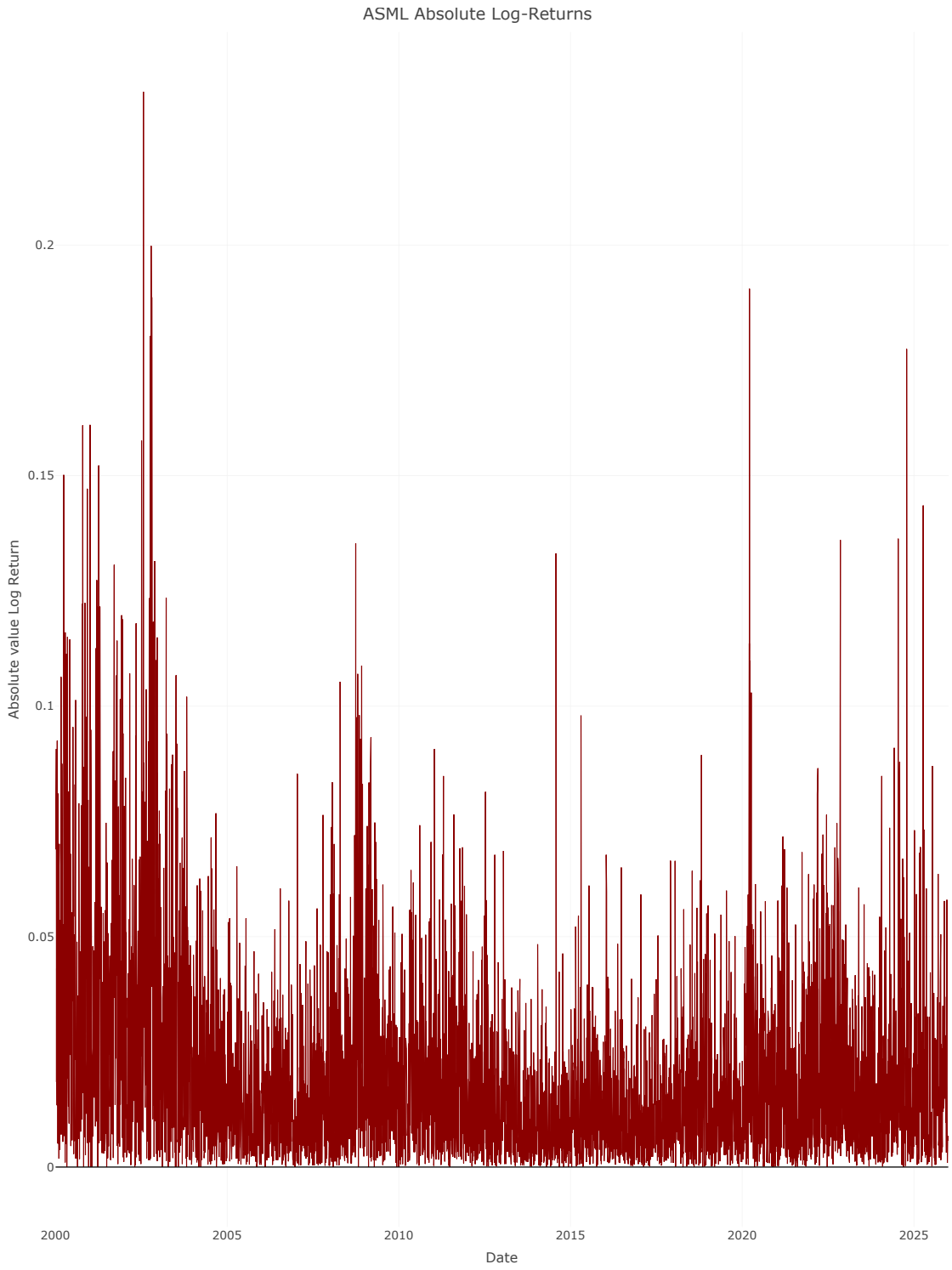


Figure 3.3: ASML Absolute Daily Log-Returns

```r
acf_abs <- acf(log_returns$Abs, plot = FALSE, lag.max = 30)
pacf_abs <- pacf(log_returns$Abs, plot = FALSE, lag.max = 30)

ci <- qnorm((1 + 0.95)/2)/sqrt(length(log_returns$Abs))

df_acf_abs <- data.frame(
  lag = as.numeric(acf_abs$lag)[-1],
  acf = as.numeric(acf_abs$acf)[-1])

df_pacf_abs <- data.frame(
  lag = as.numeric(pacf_abs$lag),
  pacf = as.numeric(pacf_abs$acf))

fig_acf_abs <- plot_ly(df_acf_abs, x = ~lag, y = ~acf, type = 'bar', name = 'ACF',
  marker = list(color = '#1f77b4')) %>%

  add_segments(x = min(df_acf_abs$lag), xend = max(df_acf_abs$lag), y = ci, yend = ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  add_segments(x = min(df_acf_abs$lag), xend = max(df_acf_abs$lag), y = -ci, yend = -ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  layout(yaxis = list(title = "ACF"))

fig_pacf_abs <- plot_ly(df_pacf_abs, x = ~lag, y = ~pacf, type = 'bar', name = 'PACF',
  marker = list(color = '#ff7f0e')) %>%

  add_segments(x = min(df_pacf_abs$lag), xend = max(df_pacf_abs$lag), y = ci, yend = ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  add_segments(x = min(df_pacf_abs$lag), xend = max(df_pacf_abs$lag), y = -ci, yend = -ci,
  line = list(color = 'red', dash = 'dash', width = 1), showlegend = FALSE) %>%

  layout(yaxis = list(title = "PACF"))

subplot(fig_acf_abs, fig_pacf_abs, nrows = 1, margin = 0.05, titleY = TRUE) %>%
  layout(title = "Absolute Returns: Autocorrelation & Partial Autocorrelation",
  margin = list(t = 50))
```

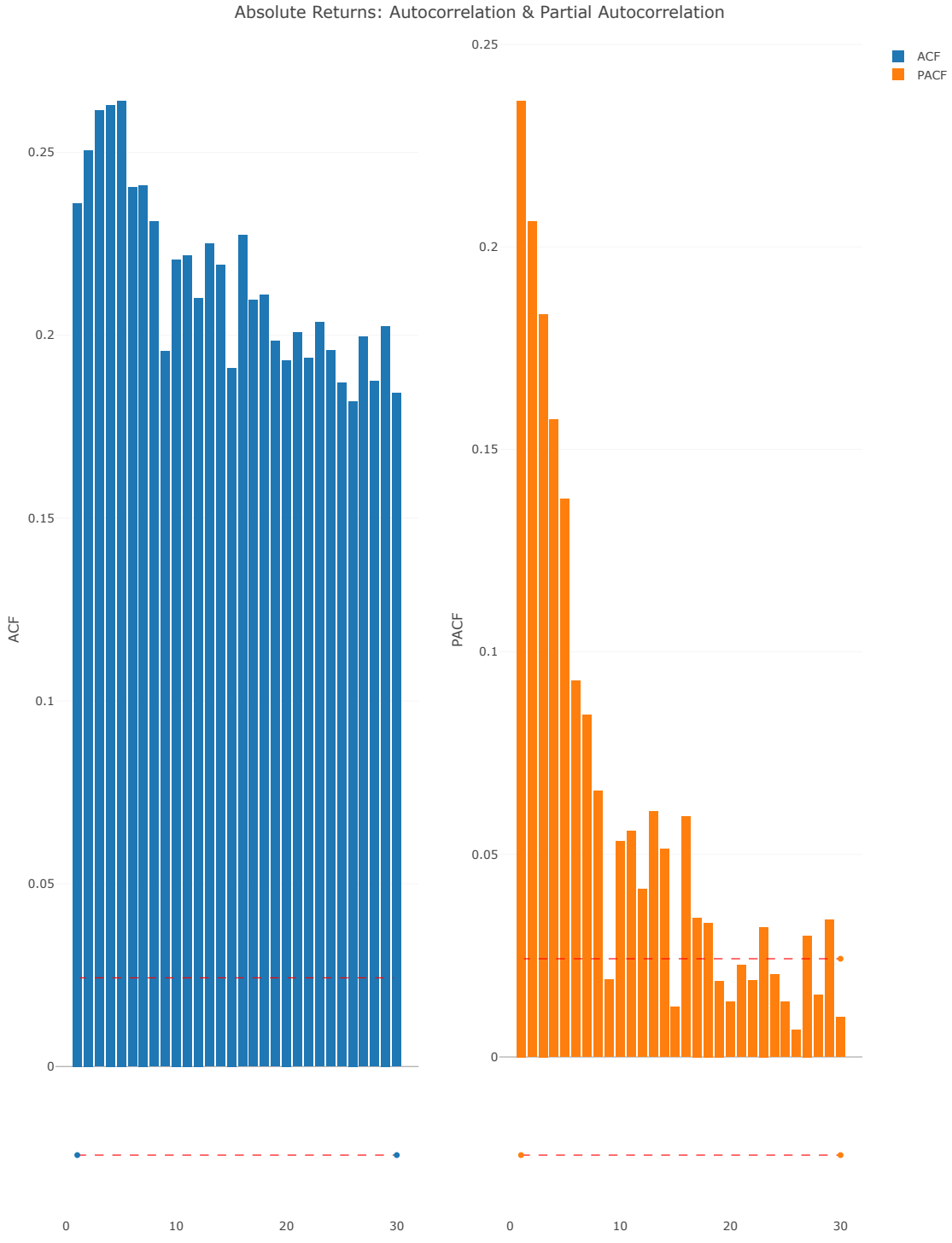


Figure 3.4: Absolute Returns ACF and PACF

1. **Volatility Clustering:** The time series plots of both Squared Log-Returns and Absolute Log-Returns clearly illustrate the phenomenon of volatility clustering. We observe distinct periods of high volatility (large spikes) followed by periods of relative calm. This intermittent bursting behavior violates the assumption of constant variance (homoskedasticity) required by standard linear models.
2. **Persistence (Long Memory):** The analysis of the correlograms provides further evidence. Unlike the ACF of raw returns, the ACF of Squared Returns displays significant positive autocorrelation that decays very slowly. A similar, perhaps even clearer, pattern of slow decay is visible in the ACF of Absolute Returns. This "long memory" indicates that the magnitude of today's shock has a strong predictive power for future volatility, justifying the use of GARCH-type models.

3.2 Formal Testing for ARCH Effects and Asymmetry

While the graphical analysis provides strong intuitive evidence of conditional heteroskedasticity, we must validate these observations using rigorous statistical procedures. Furthermore, visual inspection alone cannot quantify whether the volatility reacts symmetrically to news or if there is a "leverage effect" (where negative returns increase volatility more than positive ones).

To address this, we perform the following battery of diagnostic tests:

1. **ARCH-LM Test:** To formally test the null hypothesis of no ARCH effects in the residuals up to a specified lag.
2. **Asymmetry Tests:** useful for checking the presence of asymmetric volatility (leverage effects)

3.2.1 ARCH Effects

```
arma_fit <- Arima(log_returns$LogReturns, order = c(1,0,1))
residuals_arma <- residuals(arma_fit)

lags_to_test <- c(5, 10, 20)

run_arch_lm <- function(lag_val) {
  lm_test <- ArchTest(residuals_arma, lags = lag_val)

  return(data.frame(
    Lag = lag_val,
    Statistic = lm_test$statistic,
    P_Value = lm_test$p.value
  ))
}

arch_results <- do.call(rbind, lapply(lags_to_test, run_arch_lm))

arch_display <- arch_results %>%
```

```
  mutate(
    Conclusion = ifelse(P_Value < 0.05, "Reject H0", "Fail to Reject"),
    P_Value = format.pval(P_Value, digits = 3, eps = 0.001)
  )

knitr::kable(arch_display,
  caption = "Engle's ARCH-LM Test for Volatility Clustering",
  align = "c", row.names = FALSE)
```

Table 3.1: Engle's ARCH-LM Test for Volatility Clustering

Lag	Statistic	P_Value	Conclusion
5	870.6739	<0.001	Reject H0
10	983.5282	<0.001	Reject H0
20	1086.0644	<0.001	Reject H0

3.2.2 Leverage Effects

```
spec_std <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  mean.model = list(armaOrder = c(1, 1), include.mean = TRUE),
  distribution.model = "std"
)

fit_std <- ugarchfit(spec = spec_std, data = log_returns$LogReturns)
engle_test <- signbias(fit_std)

engle_df <- data.frame(
  Test = rownames(engle_test),
  t_value = engle_test[, 1],
  Prob = engle_test[, 2],
  Result = ifelse(engle_test[, 2] < 0.05, "Asymmetry", "Symmetry")
)

engle_df$Prob <- format.pval(engle_df$Prob, digits = 3, eps = 0.001)

knitr::kable(engle_df,
  digits = 4,
  caption = "Engle-Ng Tests for Asymmetry (Leverage Effect)",
  row.names = FALSE,
  align = "c")
```

Table 3.2: Engle-Ng Tests for Asymmetry (Leverage Effect)

Test	t_value	Prob	Result
Sign Bias	0.6572	0.511	Symmetry
Negative Sign Bias	0.3227	0.747	Symmetry
Positive Sign Bias	0.1283	0.898	Symmetry
Joint Effect	1.5848	0.663	Symmetry

The preliminary diagnostic phase has already established the presence of significant conditional heteroskedasticity, as the **ARCH-LM test strongly rejected the null hypothesis** of constant variance. However, regarding the nature of this volatility, the results diverge from the standard market behavior. Contrary to the stylized facts of broad equity indices—where the Leverage Effect typically induces higher volatility during market downturns—the **Engle-Ng tests for ASML indicate a symmetric volatility response** (H_0 is not rejected). This suggests that for ASML, large positive shocks (rallies) generate volatility levels comparable to negative shocks. This behavior is often observed in high-growth technology stocks, where upside volatility is driven by speculative trading and aggressive repricing during expansion cycles.Despite the statistical lack of significant asymmetry, we will proceed with the estimation of asymmetric models (**GJR-GARCH** and **E-GARCH**) alongside the standard specification. This ensures a comprehensive model comparison based on Information Criteria (AIC/BIC), verifying whether these models might still offer a superior fit due to their flexibility in handling the error distribution.

3.3 Models

Given the evidence of volatility clustering and the potential for non-normal error distributions, we estimate three distinct volatility specifications to identify the model that best describes the data generating process of ASML returns.

We compare the following models:

1. **Standard GARCH (sGARCH):** Assumes symmetric response to shocks.
2. **GJR-GARCH:** Allows for asymmetric response (leverage effect) targeting the variance directly.
3. **Exponential GARCH (E-GARCH):** Models the logarithm of variance, allowing for asymmetry without imposing positivity constraints on coefficients.

To ensure robustness and account for the "fat tails" observed in the preliminary analysis (Jarque-Bera test), all models are estimated using a **Student-t distribution** for the innovation process, coupled with the **ARMA(1,1)** mean equation identified in the previous chapter.

We rely on **Information Criteria (AIC and BIC)** for model selection, where lower values indicate a better trade-off between goodness-of-fit and model parsimony.

```
common_mean <- list(armaOrder = c(1, 1), include.mean = TRUE)
common_dist <- "std"

spec_std <- ugarchspec(
```

```
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  mean.model = common_mean,
  distribution.model = common_dist
)

spec_gjr <- ugarchspec(
  variance.model = list(model = "gjrGARCH", garchOrder = c(1, 1)),
  mean.model = common_mean,
  distribution.model = common_dist
)

spec_egarch <- ugarchspec(
  variance.model = list(model = "eGARCH", garchOrder = c(1, 1)),
  mean.model = common_mean,
  distribution.model = common_dist
)

fit_std    <- ugarchfit(spec_std, data = log_returns$LogReturns, solver = "hybrid")
fit_gjr    <- ugarchfit(spec_gjr, data = log_returns$LogReturns, solver = "hybrid")
fit_egarch <- ugarchfit(spec_egarch, data = log_returns$LogReturns, solver = "hybrid")

criteria_df <- data.frame(
  Model = c("sGARCH(1,1)", "GJR-GARCH(1,1)", "E-GARCH(1,1)"),
  AIC = c(infocriteria(fit_std)[1], infocriteria(fit_gjr)[1], infocriteria(fit_egarch)[1]),
  BIC = c(infocriteria(fit_std)[2], infocriteria(fit_gjr)[2], infocriteria(fit_egarch)[2])
)

criteria_df <- criteria_df[order(criteria_df$BIC), ]

knitr::kable(criteria_df,
  digits = 4,
  caption = "Model Selection: AIC and BIC Comparison",
  row.names = FALSE,
  align = "c")
```

Table 3.3: Model Selection: AIC and BIC Comparison

Model	AIC	BIC
E-GARCH(1,1)	-4.6872	-4.6789
GJR-GARCH(1,1)	-4.6820	-4.6737
sGARCH(1,1)	-4.6718	-4.6645

Comparison between Symmetric and Asymmetric GARCH

3.4 Forecast

Since the information criteria (AIC and BIC) provide an in-sample assessment, we proceed with an Out-of-Sample validation to test the actual predictive capacity on data not used for estimation.

We split the time series into two subsets:

- **Training Set** (80%): Used to estimate the GARCH parameters.
- **Test Set** (20%): Used to compare volatility forecasts against the proxy of realized volatility.

```
n_total <- nrow(log_returns)
n_train <- floor(0.80 * n_total)
n_test  <- n_total - n_train

common_mean <- list(armaOrder = c(1, 1), include.mean = TRUE)
common_dist <- "std"

spec_std <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  mean.model = common_mean, distribution.model = common_dist)

spec_gjr <- ugarchspec(
  variance.model = list(model = "gjrGARCH", garchOrder = c(1, 1)),
  mean.model = common_mean, distribution.model = common_dist)

spec_egarch <- ugarchspec(
  variance.model = list(model = "eGARCH", garchOrder = c(1, 1)),
  mean.model = common_mean, distribution.model = common_dist)

roll_std <- ugarchroll(
  spec_std, data = log_returns$LogReturns, n.ahead = 1,
  n.start = n_train, refit.every = 50, refit.window = "moving", solver = "hybrid")

roll_gjr <- ugarchroll(
  spec_gjr, data = log_returns$LogReturns, n.ahead = 1,
  n.start = n_train, refit.every = 50, refit.window = "moving", solver = "hybrid")

roll_egarch <- ugarchroll(
  spec_egarch, data = log_returns$LogReturns, n.ahead = 1,
  n.start = n_train, refit.every = 50, refit.window = "moving", solver = "hybrid")
```

The following plot displays the estimated conditional volatility ($\hat{\sigma}_t$) generated by the sGARCH, GJR-GARCH and E-GARCH model:

> **Note**
>
> Unlike price forecasting, volatility is **latent** (not directly observable). Therefore, in the following chart, we visually compare the forecasted standard deviation $\hat{\sigma}_t$ (colored lines) against the **Absolute Returns** $|r_t|$ (grey bars). While noisy, absolute returns serve as a standard unbiased proxy for actual market turbulence.

```
df_plot <- as.data.frame(roll_egarch)

df_plot$Date <- log_returns$Date[(n_train + 1):n_total]
df_plot$AbsReturn <- abs(df_plot$Realized)
df_plot$Sigma_EGARCH <- df_plot$Sigma
df_plot$Sigma_GJR <- as.data.frame(roll_gjr)$Sigma
df_plot$Sigma_STD <- as.data.frame(roll_std)$Sigma

fig_unified <- plot_ly(data = df_plot, x = ~Date) %>%

  add_bars(y = ~AbsReturn, name = 'Abs Returns (|r|)',
  marker = list(color = 'lightgrey'), opacity = 0.5) %>%

  add_lines(y = ~Sigma_STD, name = 'sGARCH',
  line = list(color = 'green', width = 1.5)) %>%

  add_lines(y = ~Sigma_GJR, name = 'GJR-GARCH',
  line = list(color = 'steelblue', width = 1.5)) %>%

  add_lines(y = ~Sigma_EGARCH, name = 'E-GARCH',
  line = list(color = 'darkorange', width = 1.5)) %>%

  layout(
    title = "Volatility Forecast Comparison (Refit Window: 50 Days)",
    yaxis = list(title = "Volatility (Sigma)"),
    xaxis = list(title = "Date"),
    legend = list(orientation = "h", x = 0.1, y = -0.2),
    barmode = "overlay",
    hovermode = "x unified"
  )

fig_unified
```

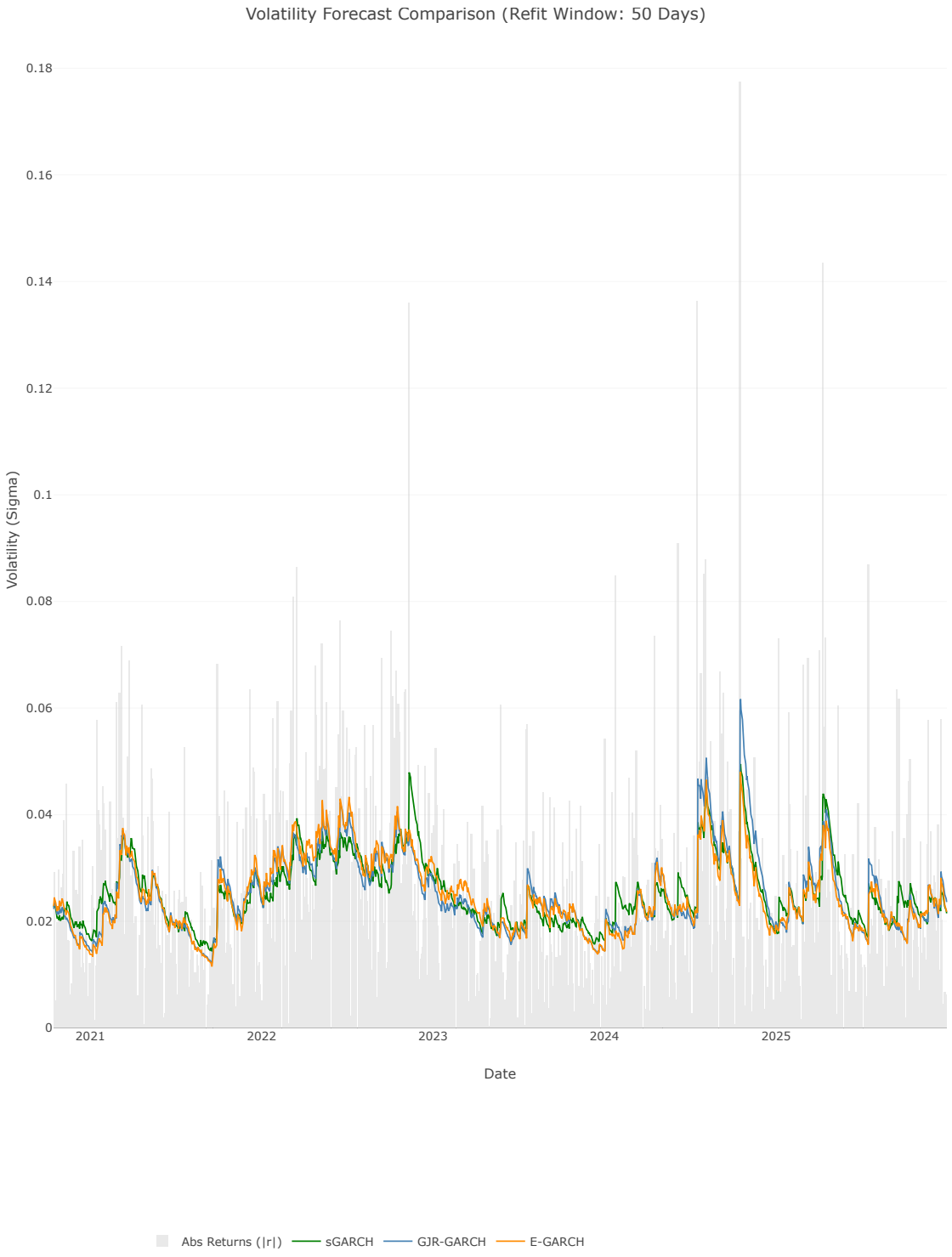


Figure 3.5: Volatility Forecast Comparison: sGARCH vs GJR-GARCH vs E-GARCH

We evaluate the predictive performance using two loss functions, utilizing squared returns (r_t^2) as the proxy for latent variance:

- **RMSE (Root Mean Squared Error):** Standard symmetric metric.
- **QLIKE (Quasi-Likelihood):** Asymmetric loss function that penalizes under-prediction of risk more heavily. **This is the preferred metric for volatility model selection.**

```
calc_vol_metrics <- function(roll_obj, name) {
  df <- as.data.frame(roll_obj)
  actual_var <- df$Realized^2
  pred_var   <- df$Sigma^2

  rmse <- sqrt(mean((actual_var - pred_var)^2))
  qlike <- mean(log(pred_var) + (actual_var / pred_var))

  return(c(Model = name, RMSE = rmse, QLIKE = qlike))
}

m_std <- calc_vol_metrics(roll_std, "sGARCH(1,1)")
m_gjr <- calc_vol_metrics(roll_gjr, "GJR-GARCH(1,1)")
m_eg <- calc_vol_metrics(roll_egarch, "E-GARCH(1,1)")

vol_val_table <- data.frame(rbind(m_std, m_gjr, m_eg))
vol_val_table$RMSE <- as.numeric(vol_val_table$RMSE)
vol_val_table$QLIKE <- as.numeric(vol_val_table$QLIKE)
vol_val_table <- vol_val_table[order(vol_val_table$QLIKE), ]

datatable(vol_val_table, options = list(dom = 't'), rownames = FALSE) %>%
  formatRound(columns = c('RMSE', 'QLIKE'), digits = 5)
```

Model	RMSE	QLIKE
sGARCH(1,1)	0.00161	-6.32360
E-GARCH(1,1)	0.00160	-6.31954
GJR-GARCH(1,1)	0.00162	-6.31323

3.5 Conclusion

The comprehensive analysis conducted in this chapter—spanning diagnostic tests, in-sample information criteria, and out-of-sample forecasting—provides a clear and robust picture of ASML's risk profile.

The results confirm the standard stylized facts of financial time series regarding the presence of **volatility clustering**: the rejection of the null hypothesis in the **ARCH-LM test** proves that ASML's volatility is not constant but evolves dynamically over time.

However, a distinct and insightful characteristic emerged regarding the *nature* of this volatility. Contrary to broad equity indices—where the "Leverage Effect" typically induces higher volatility during market downturns—our analysis highlights an **unusually low leverage effect**. This is supported by multiple converging evidences:

1. The **Engle-Ng diagnostic tests**, which failed to reject the hypothesis of symmetry.
2. The **Quantitative Performance Metrics**, where both in-sample Information Criteria (AIC/BIC) and Out-of-Sample forecast error measures (RMSE/QLIKE) **do not exhibit significant distinctions** across specifications. The complex asymmetric models (GJR and E-GARCH) failed to provide a substantial improvement over the standard benchmark, implying that the asymmetry component adds little explanatory power.

This symmetric behavior, while atypical for the general market, is characteristic of **high-growth technology stocks**. For ASML, large positive shocks (rallies) generate volatility levels comparable to negative shocks (crashes), likely driven by speculative trading and aggressive repricing during expansion cycles.

Final Model Selection: Given the negligible difference in performance metrics, we invoke the **Principle of Parsimony** and select the **sGARCH(1,1) with Student-t distribution** as the optimal model. It offers the simplest and most robust representation of the data generating process, correctly capturing the fat tails and volatility clustering without overfitting a leverage effect that is not structurally significant.

We conclude this analysis by visually translating the statistical estimates of the selected model into a practical Risk Management metric. The following chart displays the **Dynamic Value at Risk (VaR)**, showing the estimated thresholds for the maximum expected loss at 95% and 99% confidence levels.

```
sigma_t <- sigma(fit_std)
mu_t    <- fitted(fit_std)
shape_param <- coef(fit_std)["shape"]

q_05 <- qdist(distribution = "std", p = 0.05, shape = shape_param)
q_01 <- qdist(distribution = "std", p = 0.01, shape = shape_param)

VaR_95 <- mu_t + sigma_t * q_05
VaR_99 <- mu_t + sigma_t * q_01

df_risk <- data.frame(
  Date = log_returns$Date,
  Returns = log_returns$LogReturns,
  VaR_95 = as.numeric(VaR_95),
```

```
  VaR_99 = as.numeric(VaR_99)
)

fig_var <- plot_ly(df_risk, x = ~Date)

fig_var <- add_trace(
  fig_var, y = ~Returns, type = 'scatter', mode = 'lines',
  name = 'Log Returns', line = list(color = 'grey', width = 0.5, opacity = 0.5)
)

fig_var <- add_trace(
  fig_var, y = ~VaR_95, type = 'scatter', mode = 'lines',
  name = 'VaR 95% (sGARCH)', line = list(color = 'orange', width = 1.5)
)

fig_var <- add_trace(
  fig_var, y = ~VaR_99, type = 'scatter', mode = 'lines',
  name = 'VaR 99% (sGARCH)', line = list(color = 'darkred', width = 1.5)
)

fig_var <- layout(
  fig_var,
  title = "Risk Management: ASML Returns vs Dynamic VaR (sGARCH)",
  yaxis = list(title = "Log Return"),
  xaxis = list(title = "Date"),
  legend = list(orientation = "h", x = 0.1, y = -0.2)
)

fig_var
```

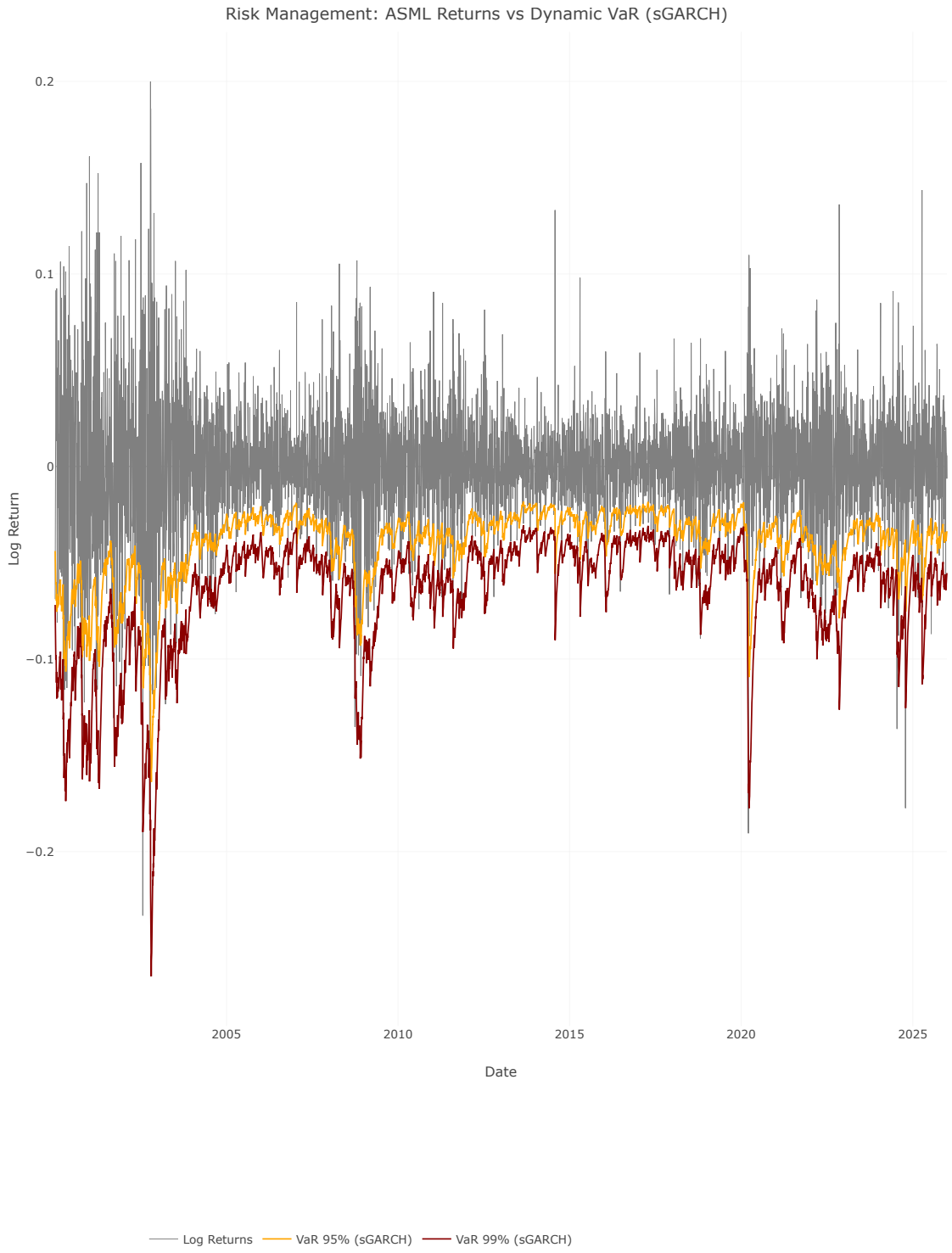


Figure 3.6: Value at Risk (VaR) In-Sample with sGARCH(1,1)