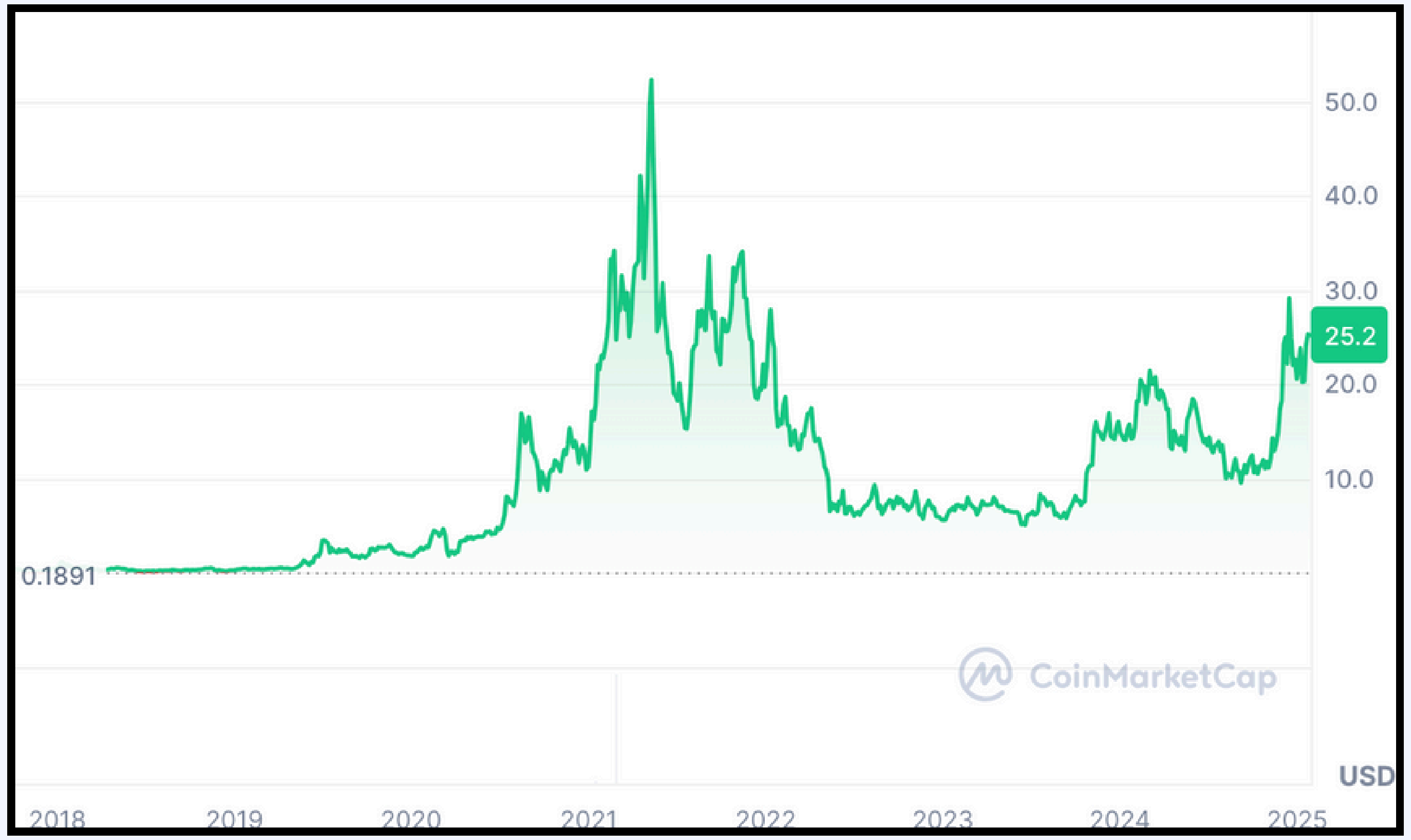
Mission!
Find a model of
Link coin forecasting

Haotian Luo

Our Comparison target

Chainlink



- Chainlink is a decentralized blockchain oracle network
- Chainlink's token is on Ethereum
- Created in 2017 by Sergey Nazarov and Steve Ellis, was formally launched in 2019
- Decentralized oracle network is an open-source technology infrastructure that allows any blockchain to securely connect to off-chain data and computation resources.

Data Fetch & Aggregation

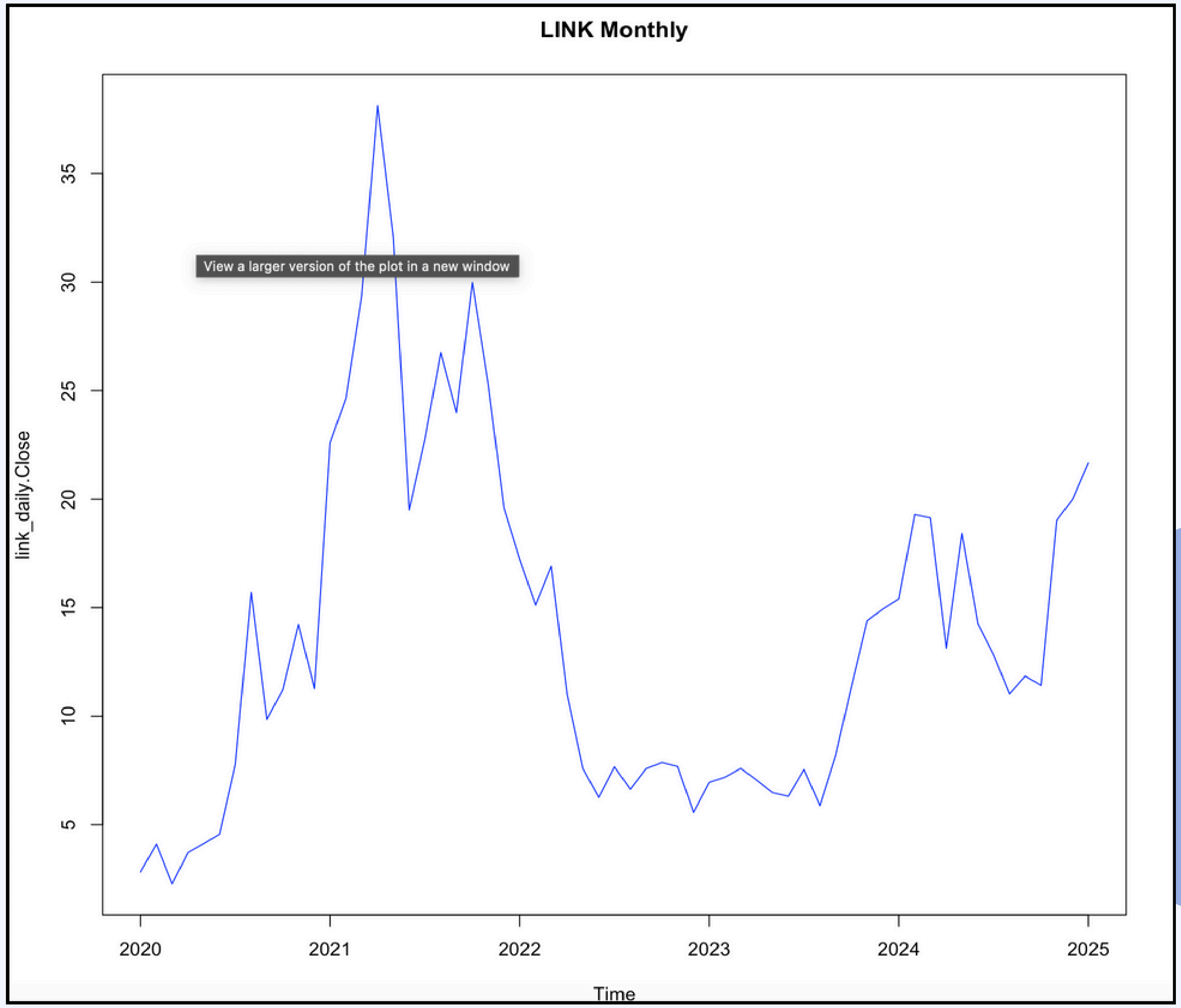
```
start_date <- as.Date("2020-01-01")
end_date   <- as.Date("2025-01-01")
getSymbols("LINK-USD", src="yahoo", from=start_date, to=end_date)
link_daily <- Ad(`LINK-USD`)
link_monthly_xts <- to.monthly(link_daily, indexAt="lastof", drop.time=TRUE)
link_monthly <- Cl(link_monthly_xts)
start_yr <- as.numeric(format(start(link_monthly_xts), "%Y"))
start_mo <- as.numeric(format(start(link_monthly_xts), "%m"))
link_ts <- ts(link_monthly, start=c(start_yr, start_mo), frequency=12)
N_all <- length(link_ts)
```

Explorayory & Outliers

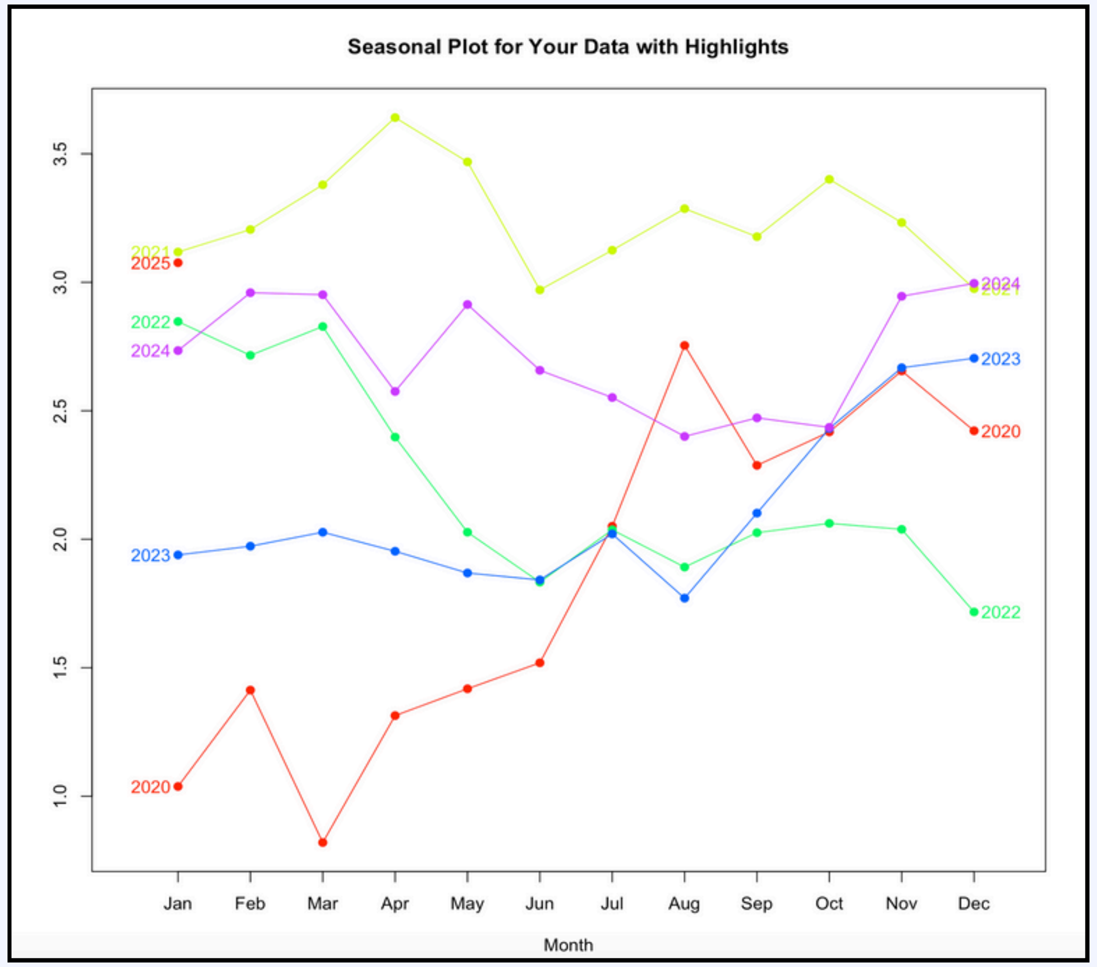
```
plot(link_ts, main="LINK Monthly", col="blue")
summary(link_ts)
link_log <- log(link_ts)
link_diff <- diff(link_log)
box_stats <- boxplot.stats(na.omit(link_diff))
outs <- box_stats$out
num_outs <- length(outs)
num_outs     ##result = 0 which means the data are correct with original one
box_u <- box_stats$stats[5]
box_l <- box_stats$stats[1]
link_diff_cap <- link_diff
link_diff_cap[link_diff_cap>box_u] <- box_u
link_diff_cap[link_diff_cap<box_l] <- box_l
```

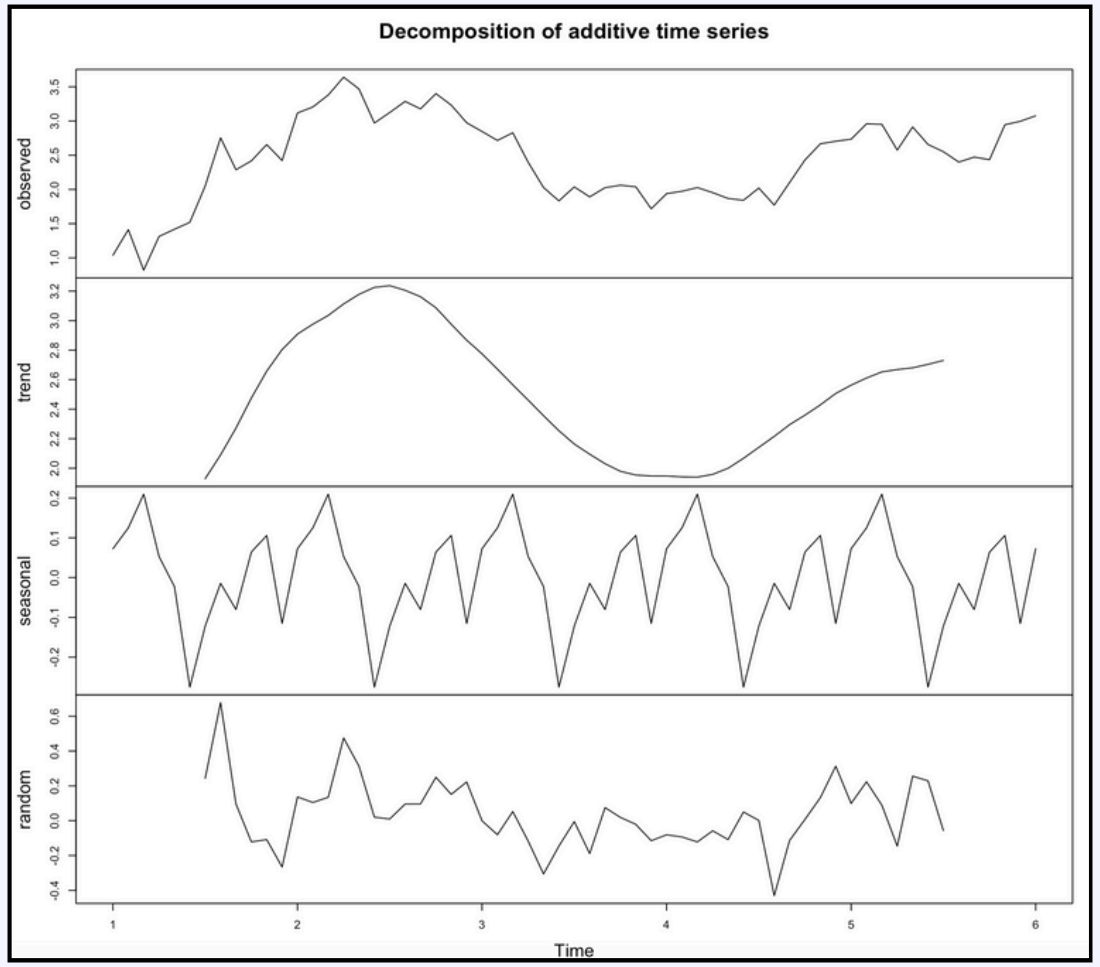
```
link_daily.Close
Min.   : 2.270
1st Qu.: 7.545
Median :11.422
Mean   :13.687
3rd Qu.:19.138
Max.   :38.129
```

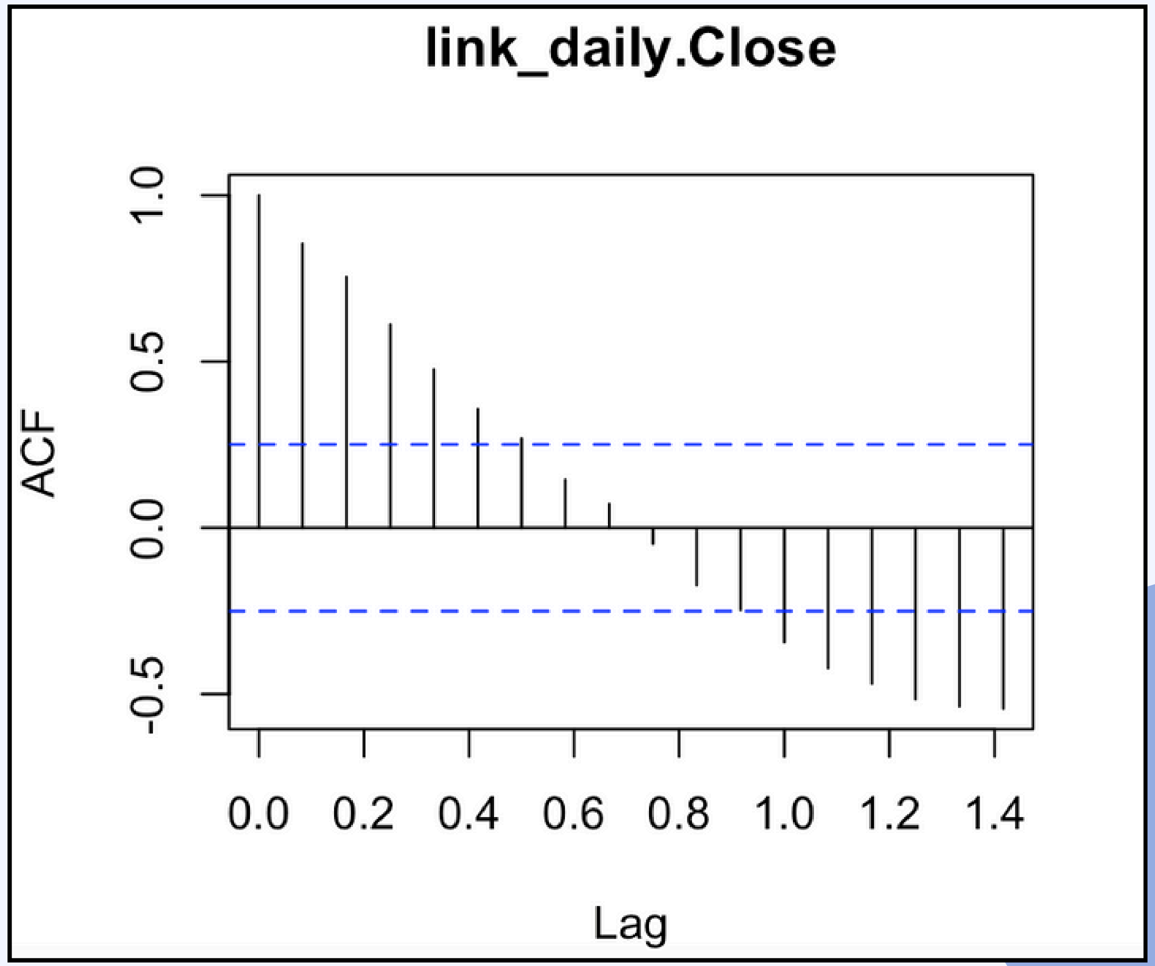
- Possible that it is positive skewness
- Huge difference between min and max



Stationarity & Seasonality?








```r
seasonplot(ts_data,
          year.labels = TRUE,         # Add year labels for better understanding
          year.labels.left = TRUE,    # Place labels on the left side as well
          col = rainbow(5),           # Use rainbow colors (adjust for your number of years)
          main = "Seasonal Plot for Your Data with Highlights",
          pch = 19                    # Use solid points to show exact values
```

```r
decomposed <- decompose(ts(link_log, frequency = 12))
plot(decomposed)
```

```r
acf(ts(link_log, frequency = 12))
```

Stationarity/Seasonality?

```r
d_adf <- ndiffs(link_log,  test="adf")
d_kpss <- ndiffs(link_log, test="kpss")

# Convert link_log to a time series object with frequency = 12 (12 months in a year)
link_log_ts <- ts(link_log, frequency = 12)

# Use the updated object in nsdiffs
D_seas <- nsdiffs(link_log_ts, test = "ocsb")

d_adf ##The value 1 indicates that one non-seasonal difference
d_kpss#The value 0 indicates that no non-seasonal differencing is needed
D_seas#The value 0 indicates that no seasonal differencing
adf_1 <- adf.test(na.omit(link_diff))
adf_1##result p-value is 0.1162 fail to reject H0
kpss_1 <- kpss.test(na.omit(link_diff))
kpss_1##result p-value is 0.1 fail to reject H0
# Convert `link_log` to a time series object with correct frequency
ts_data <- ts(link_log, frequency = 12, start = c(2020, 1))  # Adjust start year if needed
# Add colors and labels to make the plot easier to read
seasonplot(ts_data,
          year.labels = TRUE,        # Add year labels for better understanding
          year.labels.left = TRUE,   # Place labels on the left side as well
          col = rainbow(5),          # Use rainbow colors (adjust for your number of years)
          main = "Seasonal Plot for Your Data with Highlights",
          pch = 19                   # Use solid points to show exact values
)
###########the result shows the seasonality but not every year

##Perform decomposition to separate the trend, seasonality, and residuals for clearer insights.
decomposed <- decompose(ts(link_log, frequency = 12))
plot(decomposed)

acf(ts(link_log, frequency = 12))
```

test="adf" (ADF test)

Check if the data is abnormal (check if there is a unit root)

Result: Calculate how many times it takes to achieve stationarity

test="kpss" (KPSS test)

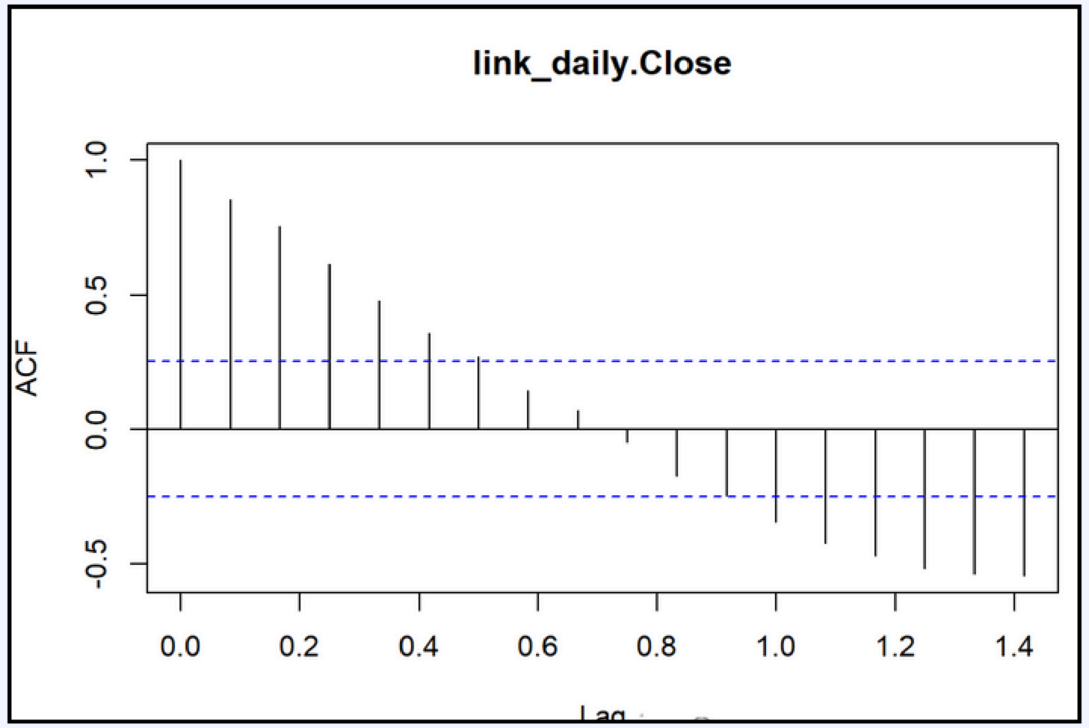
Check if the data is normal.

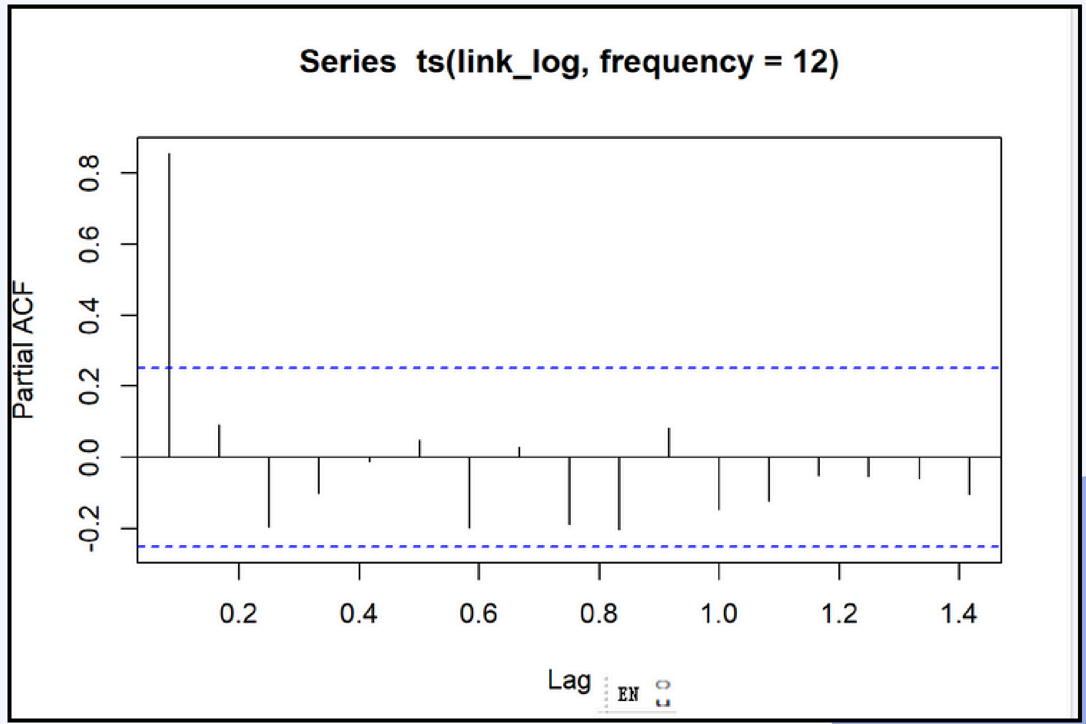
Result: If it is abnormal, calculate how many times it takes to achieve stationarity.

```
        Augmented Dickey-Fuller Test

data:  na.omit(link_diff)
Dickey-Fuller = -3.1346, Lag order = 3, p-value = 0.1162
alternative hypothesis: stationary
```

```
        KPSS Test for Level Stationarity

data:  na.omit(link_diff)
KPSS Level = 0.18043, Truncation lag parameter = 3, p-value = 0.1
```

Try without Seasonality For ARIMA






```
arma_search <- auto.arima(link_ret, stationary=TRUE, seasonal=FALSE, trace=TRUE)
arma_search
```

```
> arma_search
Series: link_ret
ARIMA(1,0,1) with zero mean

Coefficients:
          ar1     ma1
      -0.7155  0.6589
s.e.   0.1219  0.1307

sigma^2 = 0.0033:  log likelihood = 2628.13
AIC=-5250.27   AICc=-5250.25   BIC=-5233.74
>
```

Statistical Test

```
> Box.test(residuals(arma_search))

        Box-Pierce test

data:  residuals(arma_search)
X-squared = 0.10483, df = 1, p-value = 0.7461

> ArchTest(residuals(arma_search))

        ARCH LM-test; Null hypothesis: no ARCH effect

data:  residuals(arma_search)
Chi-squared = 63.412, df = 12, p-value = 5.352e-09
```

1. No leftover linear autocorrelation

2. volatility clustering

A Loop Logic to find multi-garch

```r
n_obs <- length(link_ret)  # Total observations in data (for AIC/BIC scaling)

for (garch_type in garch_models) {
  for (dist in distributions) {  # Loop through distributions
    for (p_garch in p_candidates) {  # Loop through GARCH p-orders
      for (q_garch in q_candidates) {  # Loop through GARCH q-orders

        # Create a unique model label
        model_label <- paste0(garch_type, "_", dist, "_p", p_garch, "_q", q_garch)

        # Define the GARCH specification
        spec_tmp <- ugarchspec(
          variance.model = list(
            model = garch_type,          # GARCH model type (e.g., sGARCH)
            garchOrder = c(p_garch, q_garch)  # (p, q) GARCH orders
          ),
          mean.model = list(
            armaOrder = arma_order,      # Fixed ARMA(1,1) mean model
            include.mean = TRUE          # Include the mean in the model
          ),
          distribution.model = dist      # Dynamic distribution (e.g., std, norm, sstd)
        )

        # Try fitting the model and handle errors gracefully
        fit_tmp <- tryCatch(
          ugarchfit(spec = spec_tmp, data = link_ret),
          error = function(e) NULL
        )

        if (!is.null(fit_tmp)) {
          # Extract Information Criteria and Diagnostics
          ic_vals <- infocriteria(fit_tmp)  # Per-observation AIC/BIC
          aic_raw <- ic_vals["Akaike"] * n_obs  # Convert to raw AIC
          bic_raw <- ic_vals["Bayes"] * n_obs   # Convert to raw BIC
          loglik <- fit_tmp@fit$LLH              # Log-Likelihood
```

```r
          # Residual Diagnostics
          z_resid <- residuals(fit_tmp, standardize = TRUE)
          arch_test <- ArchTest(z_resid, lags = 12)
          lb_test <- Box.test(z_resid^2, lag = 12, type = "Ljung-Box")

          # Append Results to Data Frame
          results_df <- rbind(
            results_df,
            data.frame(
              ModelType = garch_type,
              Distribution = dist,
              p_garch = p_garch,
              q_garch = q_garch,
              Converged = TRUE,
              AIC = aic_raw,
              BIC = bic_raw,
              LogLik = loglik,
              ARCH_LM_p = arch_test$p.value,
              LjungSq_p = lb_test$p.value,
              stringsAsFactors = FALSE
            )
          )

          # Save the fit for later retrieval
          fits_list[[model_label]] <- fit_tmp

          cat(sprintf("[OK] %s => AIC=%.3f, BIC=%.3f, LogLik=%.3f, ARCH.p=%.3f, LjungSq.p=%.3f\n",
                      model_label, aic_raw, bic_raw, loglik, arch_test$p.value, lb_test$p.value))
        } else {
          # If model fails to converge, append NA values
          results_df <- rbind(
            results_df,
            data.frame(
              ModelType = garch_type,
              Distribution = dist,
              p_garch = p_garch,
              q_garch = q_garch,
              Converged = FALSE,
              AIC = NA,
              BIC = NA,
            LogLik = NA,
            ARCH_LM_p = NA,
            LjungSq_p = NA,
            stringsAsFactors = FALSE
          )
        )
        cat(sprintf("[FAIL] %s did not converge.\n", model_label))
      }
    }
  }
}
}
conv_df <- subset(results_df, Converged == TRUE)
```

Result for finding model

```
*---------------------------------*
*          GARCH Model Fit        *
*---------------------------------*

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

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001479    0.000905   1.6347 0.102108
ar1     0.723020    0.276713   2.6129 0.008978
ma1    -0.753913    0.262416  -2.8730 0.004066
omega   0.000115    0.000040   2.8782 0.003999
alpha1  0.102946    0.022575   4.5602 0.000005
beta1   0.864631    0.028930  29.8873 0.000000
shape   5.219507    0.613517   8.5075 0.000000
```

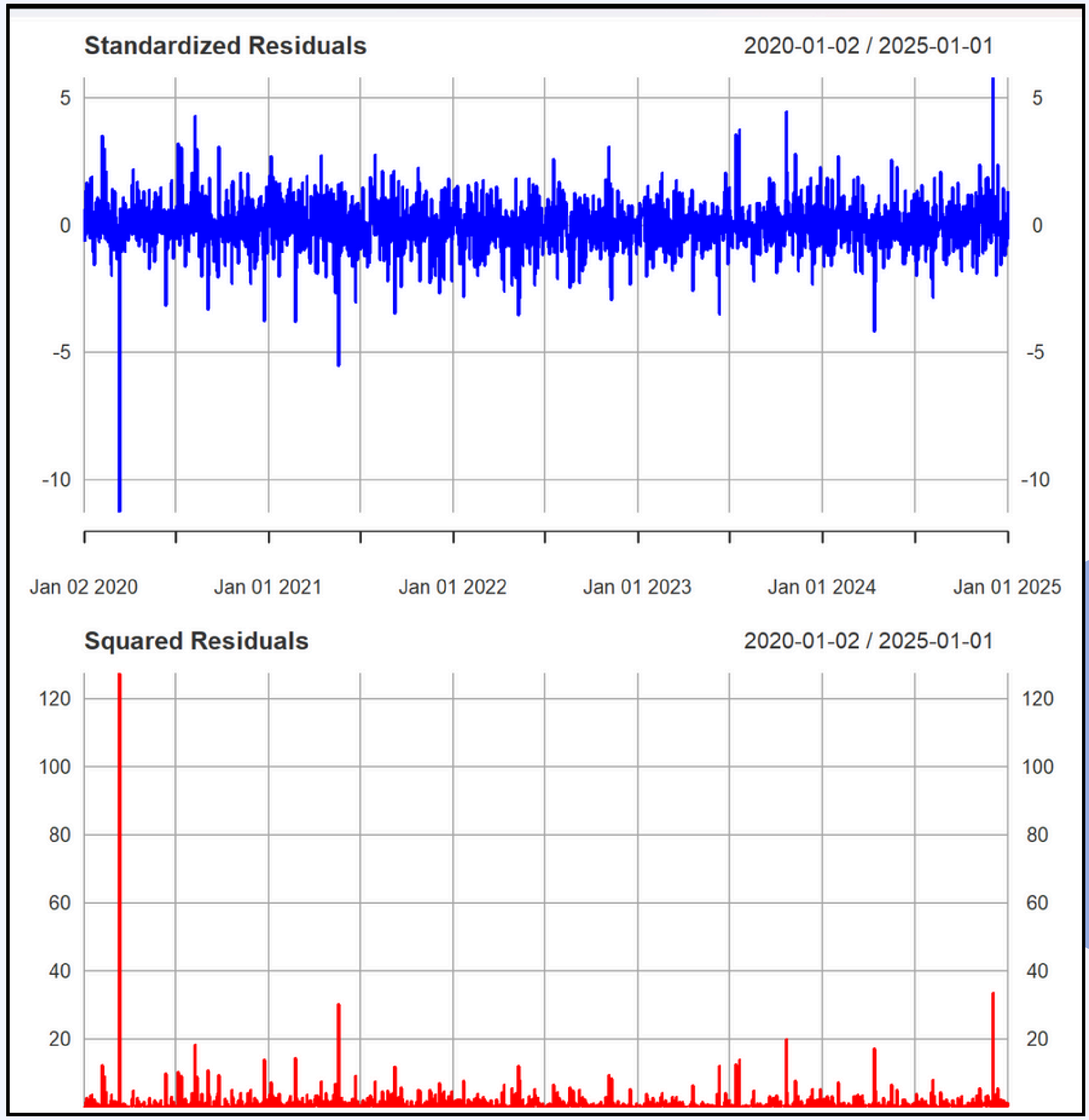
```
Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001479    0.000914   1.6179 0.105688
ar1     0.723020    0.289072   2.5012 0.012378
ma1    -0.753913    0.272932  -2.7623 0.005740
omega   0.000115    0.000051   2.2517 0.024343
alpha1  0.102946    0.027807   3.7022 0.000214
beta1   0.864631    0.038279  22.5875 0.000000
shape   5.219507    0.695308   7.5068 0.000000

LogLikelihood : 2876.413

Information Criteria
------------------------------------

Akaike       -3.1411
Bayes        -3.1200
Shibata      -3.1411
Hannan-Quinn -3.1333
```

```
Weighted Ljung-Box Test on Standardized Residuals
------------------------------------
                        statistic   p-value
Lag[1]                      0.397 5.287e-01
Lag[2*(p+q)+(p+q)-1][5]     6.187 3.616e-05
Lag[4*(p+q)+(p+q)-1][9]     9.089 2.200e-02
d.o.f=2
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.0143  0.9048
Lag[2*(p+q)+(p+q)-1][5]    0.4631  0.9632
Lag[4*(p+q)+(p+q)-1][9]    1.1771  0.9777
d.o.f=2

Weighted ARCH LM Tests
------------------------------------
            Statistic Shape Scale P-Value
ARCH Lag[3]    0.4625 0.500 2.000  0.4965
ARCH Lag[5]    0.7137 1.440 1.667  0.8193
ARCH Lag[7]    1.0950 2.315 1.543  0.8976
```

Graph result

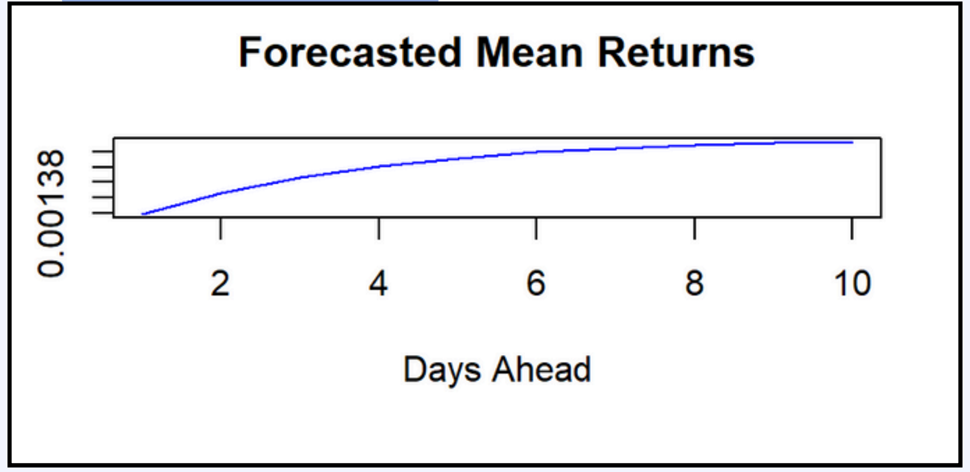
```
        Augmented Dickey-Fuller Test

data:  link_ret
Dickey-Fuller = -12.057, Lag order = 12, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(link_ret) : p-value smaller than printed p-value
> kpss.test(link_ret)

        KPSS Test for Level Stationarity

data:  link_ret
KPSS Level = 0.20485, Truncation lag parameter = 8, p-value = 0.1
```

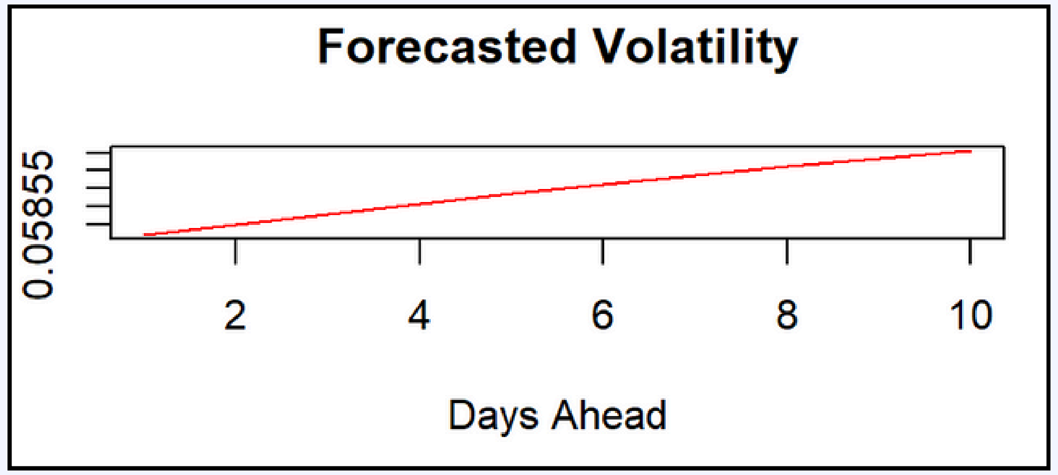
```r
#####Residual Diagnostics
z_resid_best <- residuals(best_fit, standardize = TRUE)
par(mfrow = c(2, 1))
plot(z_resid_best, type = "l", col = "blue", main = "Standardized Residuals")
plot(z_resid_best^2, type = "l", col = "red", main = "Squared Residuals")
par(mfrow = c(1, 1))
```



Forcasting 10 days

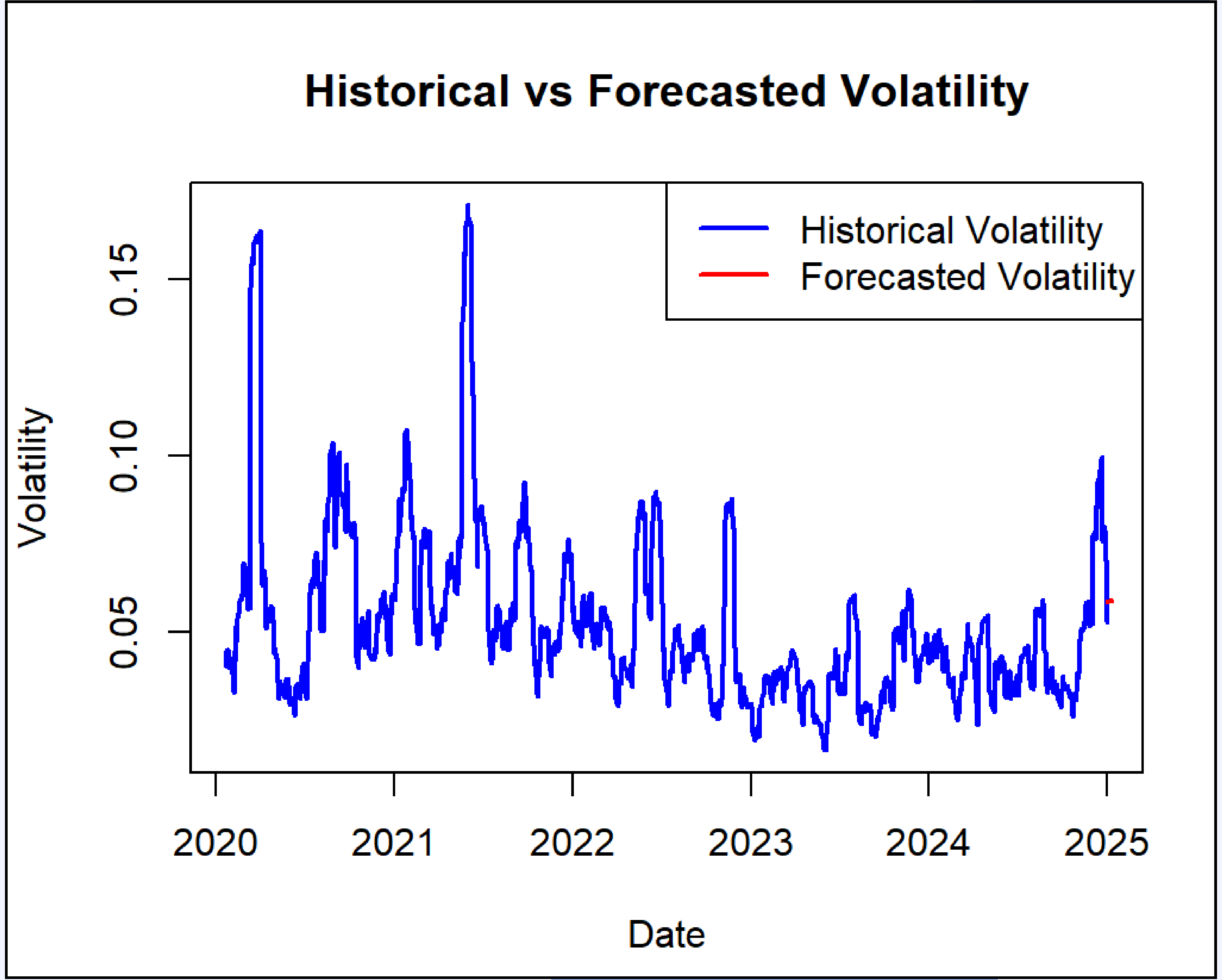
```
Forecasted Mean Returns:
        2025-01-01
T+1   0.001378487
T+2   0.001406348
T+3   0.001426492
T+4   0.001441057
T+5   0.001451587
T+6   0.001459201
T+7   0.001464706
T+8   0.001468686
T+9   0.001471564
T+10  0.001473645

Forecasted Volatility (Standard Deviations):
      2025-01-01
T+1   0.05852022
T+2   0.05855022
T+3   0.05857923
T+4   0.05860729
T+5   0.05863442
T+6   0.05866067
T+7   0.05868605
T+8   0.05871060
T+9   0.05873434
T+10  0.05875730
```






```
# Forecast horizon (e.g., 10 days ahead)
forecast_horizon <- 10

# Ensure best_fit contains the optimal model
if (!is.null(best_fit)) {
  # Generate the forecast
  garch_forecast <- ugarchforecast(best_fit, n.ahead = forecast_horizon)

  # Display the forecast
  cat("\n===== GARCH Forecast =====\n")
  show(garch_forecast)

  # Extract forecasted mean returns
  forecasted_mean <- garch_forecast@forecast$seriesFor
  cat("\nForecasted Mean Returns:\n")
  print(forecasted_mean)

  # Extract forecasted volatility (conditional standard deviations)
  forecasted_volatility <- garch_forecast@forecast$sigmaFor
  cat("\nForecasted Volatility (Standard Deviations):\n")
  print(forecasted_volatility)

  # Visualize the forecast
  par(mfrow = c(2, 1))
  plot(forecasted_mean, type = "l", col = "blue", main = "Forecasted Mean Returns", xlab = "Days Ahead", ylab = "Returns")
  plot(forecasted_volatility, type = "l", col = "red", main = "Forecasted Volatility", xlab = "Days Ahead", ylab = "Volatility")
  par(mfrow = c(1, 1))
} else {
  cat("Best fit model is NULL. Ensure the model has been identified before forecasting.")
}
```

Comparing with history data

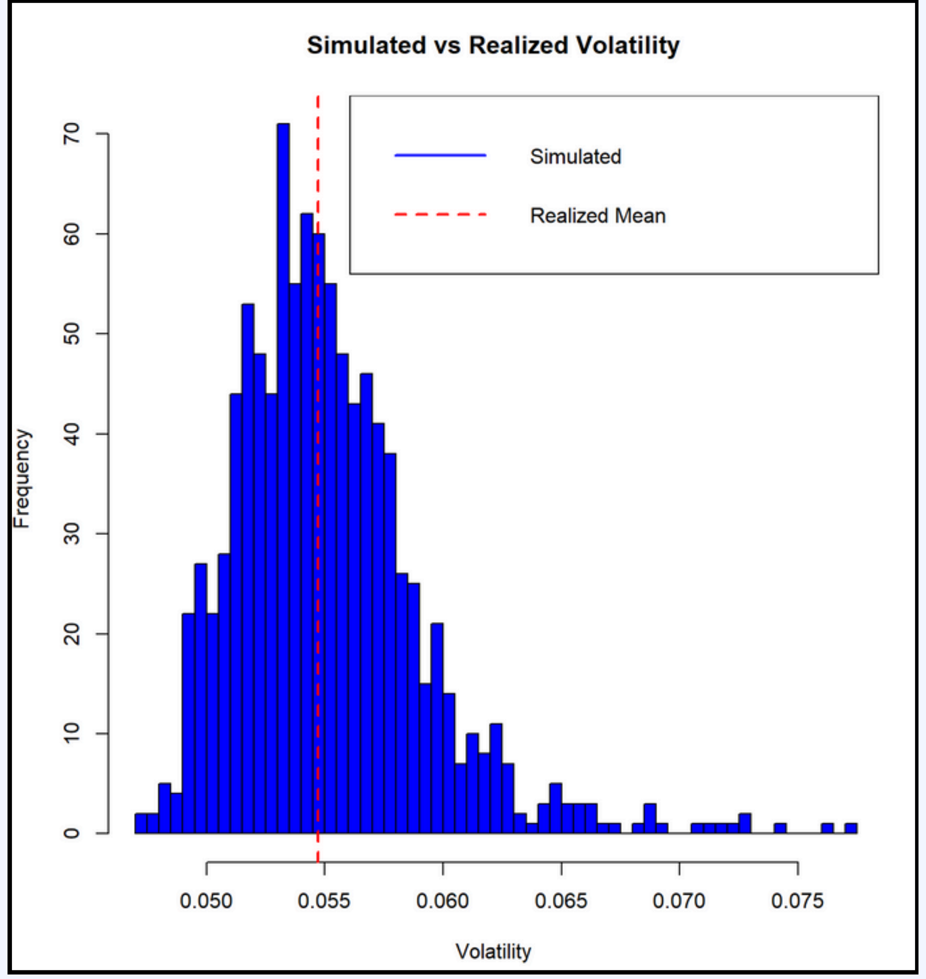


It shows in the range of volatiility

```r
# 确保必要的数据
price_data <- link_daily  # 历史价格数据
forecasted_volatility <- garch_forecast@forecast$sigmaFor  # GARCH 模型预测的波动率

# 确保基准时间从固定预测日期开始，例如 2025-01-01
base_date <- as.Date("2025-01-01")  # 设置预测起点
forecast_dates <- seq.Date(from = base_date, by = "days", length.out = length(forecasted_volatility))

# 计算历史波动率（用对数收益计算）
log_returns <- diff(log(price_data))  # 对数收益
log_returns <- na.omit(log_returns)  # 移除 NA 值

# 滚动计算历史波动率
rolling_window <- 20  # 定义滚动窗口大小，例如 20 天
historical_volatility <- rollapply(
  data = log_returns,
  width = rolling_window,
  FUN = sd,
  align = "right",
  fill = NA
)

# 确保历史波动率与日期对齐
historical_volatility <- na.omit(historical_volatility)  # 去掉滚动计算中的 NA 值
historical_volatility_xts <- xts(historical_volatility, order.by = index(log_returns)[rolling_window:length(log_returns)])

# 创建预测波动率的 xts 对象
forecasted_volatility_xts <- xts(as.numeric(forecasted_volatility), order.by = forecast_dates)

# 可视化比较历史波动率和预测波动率
plot(index(historical_volatility_xts), historical_volatility_xts, type = "l", col = "blue", lwd = 2,
     xlab = "Date", ylab = "Volatility", main = "Historical vs Forecasted Volatility")
lines(index(forecasted_volatility_xts), forecasted_volatility_xts, col = "red", lwd = 2)
legend("topright", legend = c("Historical Volatility", "Forecasted Volatility"), col = c("blue", "red"), lwd = 2)
```

Simulation

```
######monte carlo simulation
# 假设您已经拟合了一个 GARCH 模型
spec <- ugarchspec(mean.model = list(armaOrder = c(1, 1)),
                   variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                   distribution.model = "std")

fit <- ugarchfit(spec = spec, data = link_ret)

# 1. 使用拟合结果进行蒙特卡洛模拟
set.seed(123)
n.sim <- 1000  # 模拟路径数
simulations <- ugarchsim(fit, n.sim = length(link_ret), m.sim = n.sim)

# 2. 提取模拟数据
simulated_returns <- fitted(simulations)  # 模拟的收益率矩阵
simulated_volatility <- sigma(simulations)  # 模拟的波动率矩阵

# 3. 计算模拟波动率的统计特性
# 计算模拟数据每条路径的平均波动率
simulated_mean_vol <- apply(simulated_volatility, 2, mean)
simulated_sd_vol <- apply(simulated_volatility, 2, sd)

# 实际数据的波动率
realized_volatility <- sigma(fit)  # 模型拟合的历史波动率
realized_mean_vol <- mean(realized_volatility)
realized_sd_vol <- sd(realized_volatility)

# 4. 可视化比较
# 绘制模拟的波动率分布与实际波动率
hist(simulated_mean_vol, breaks = 50, col = "blue", main = "Simulated vs Realized Volatility",
     xlab = "Volatility", xlim = range(c(simulated_mean_vol, realized_mean_vol)))
abline(v = realized_mean_vol, col = "red", lwd = 2, lty = 2)  # 实际波动率均值
legend("topright", legend = c("Simulated", "Realized Mean"), col = c("blue", "red"), lty = c(1, 2), lwd = 2)
```



1. The blue bars show how often the simulation predicts different volatilities.
2. The red dashed line shows the real-world average volatility.
3. since the red line matches the peak of the blue bars, the model seems accurate for predicting typical daily volatility.
4. However, there are some extreme cases in the simulation (right tail) that might need attention if you’re worried about rare, high-volatility events.

Conclusion

From Seasonal

We cannot catch any Data that showing it is Seasonal by statistic, plase look at the Link-USDT 1 as an approach to find SARIMA, but it failed

From ARIMA GARCH

ARIMA (1,0,1)
sGARCH(1,1) t-distribution

Data From LINK-USDT 2

After we cut it in seasonality, and comparing 3 GARCH possibility, 81 samples we found the sGarch(1,1) catches the volatility