btc-stats-models

```r
btc_df <- read_csv("data/BTC-Daily.csv")
```

```
## Rows: 2651 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): symbol
## dbl  (7): unix, open, high, low, close, Volume BTC, Volume USD
## dttm (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

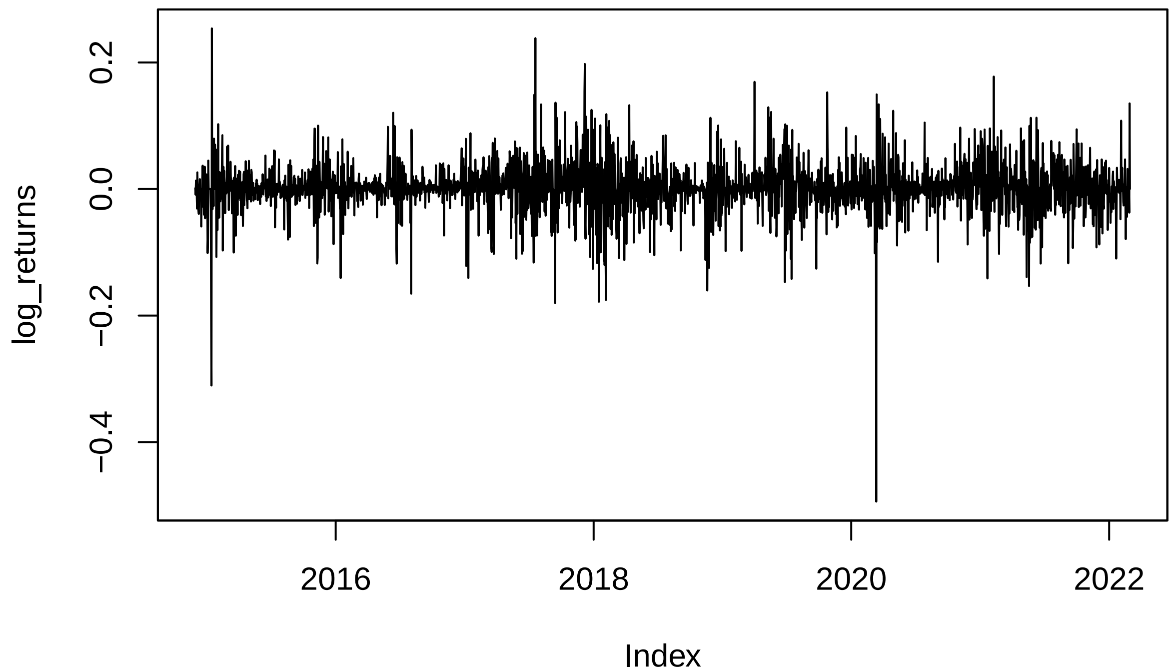
```r
x = zoo(btc_df$close, as.Date(btc_df$date))

log_returns = diff(log(x))

head(log_returns)
```

```
##   2014-11-29   2014-11-30   2014-12-01   2014-12-02   2014-12-03   2014-12-04
##  0.001168659 -0.009012673  0.013435877  0.002270208 -0.006826146 -0.030924324
```

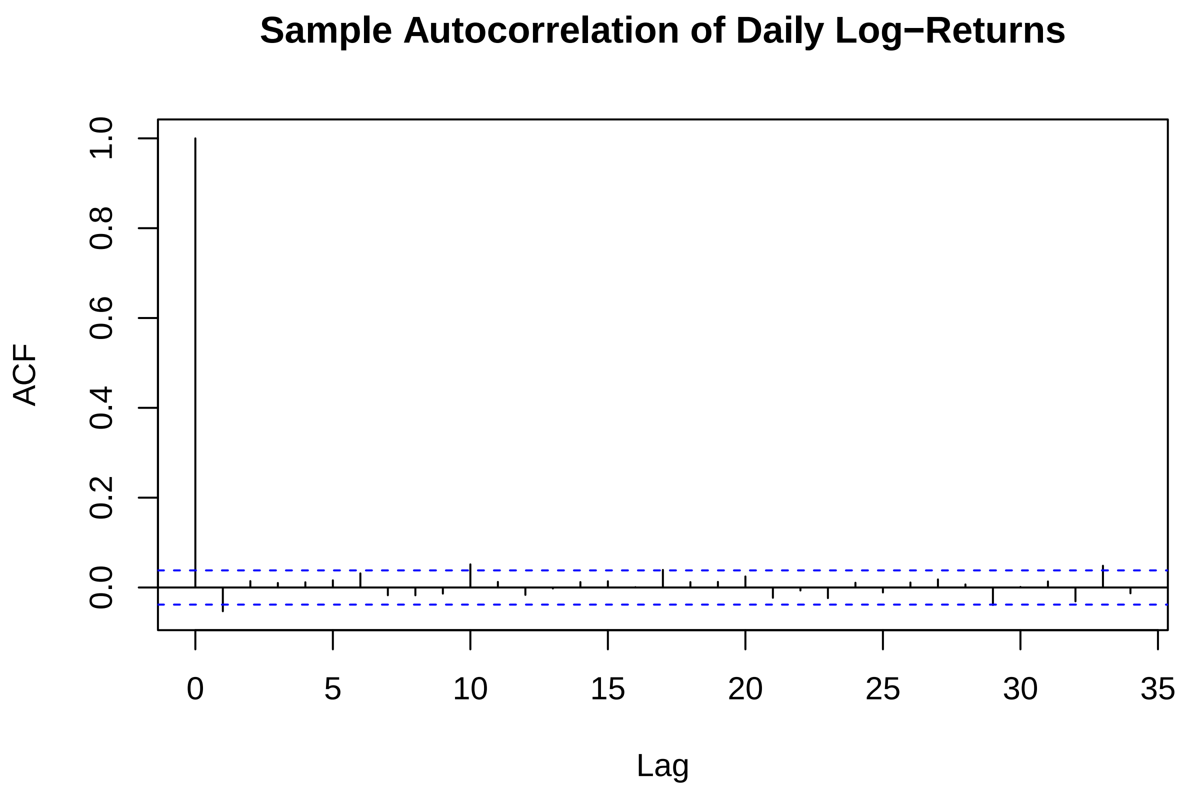
```r
tail(log_returns)
```

```
##    2022-02-24    2022-02-25    2022-02-26    2022-02-27    2022-02-28
##  0.0291543332  0.0220283692 -0.0021684582 -0.0373187285  0.1353574253
##    2022-03-01
##  0.0001505249
```

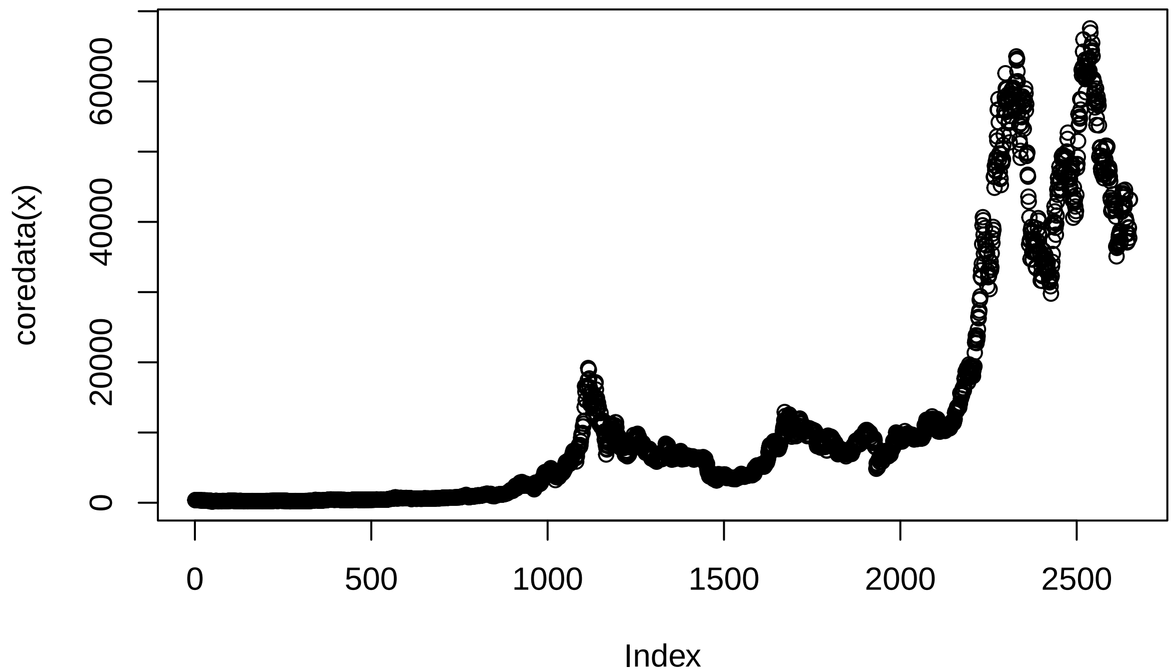
```r
plot(log_returns)
```



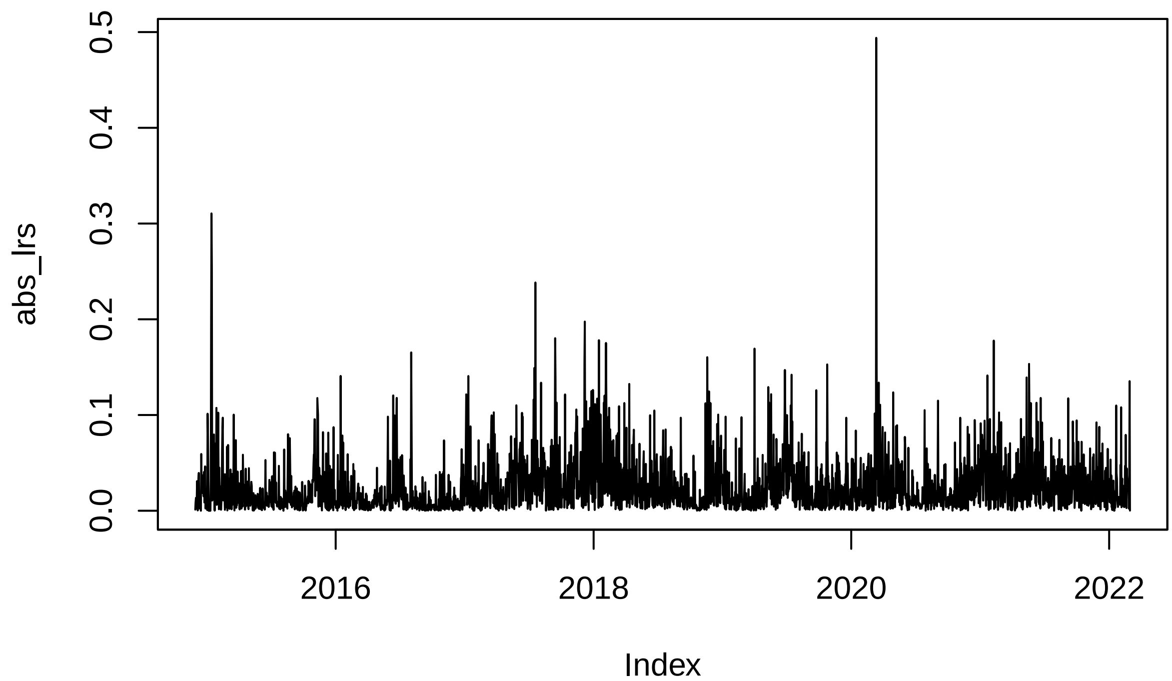

```r
acf(coredata(log_returns), main="Sample Autocorrelation of Daily Log-Returns")
```



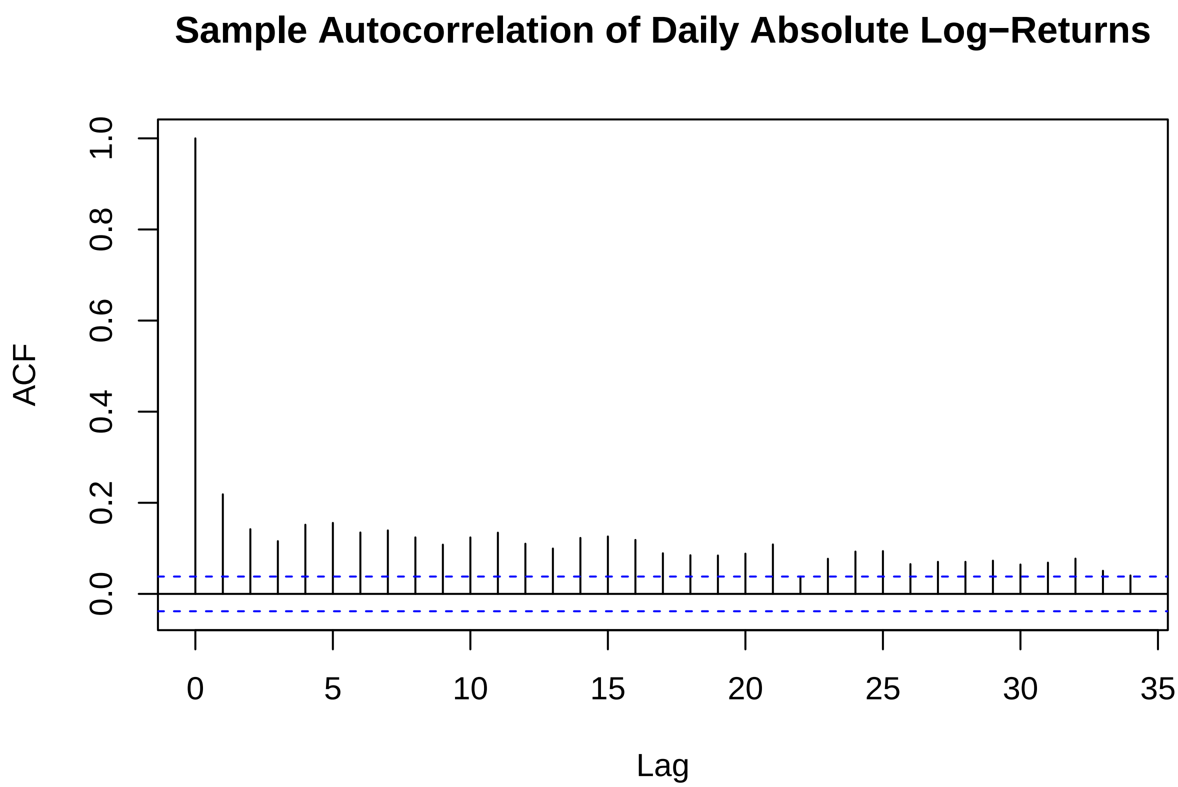

```r
plot(coredata(x))
```



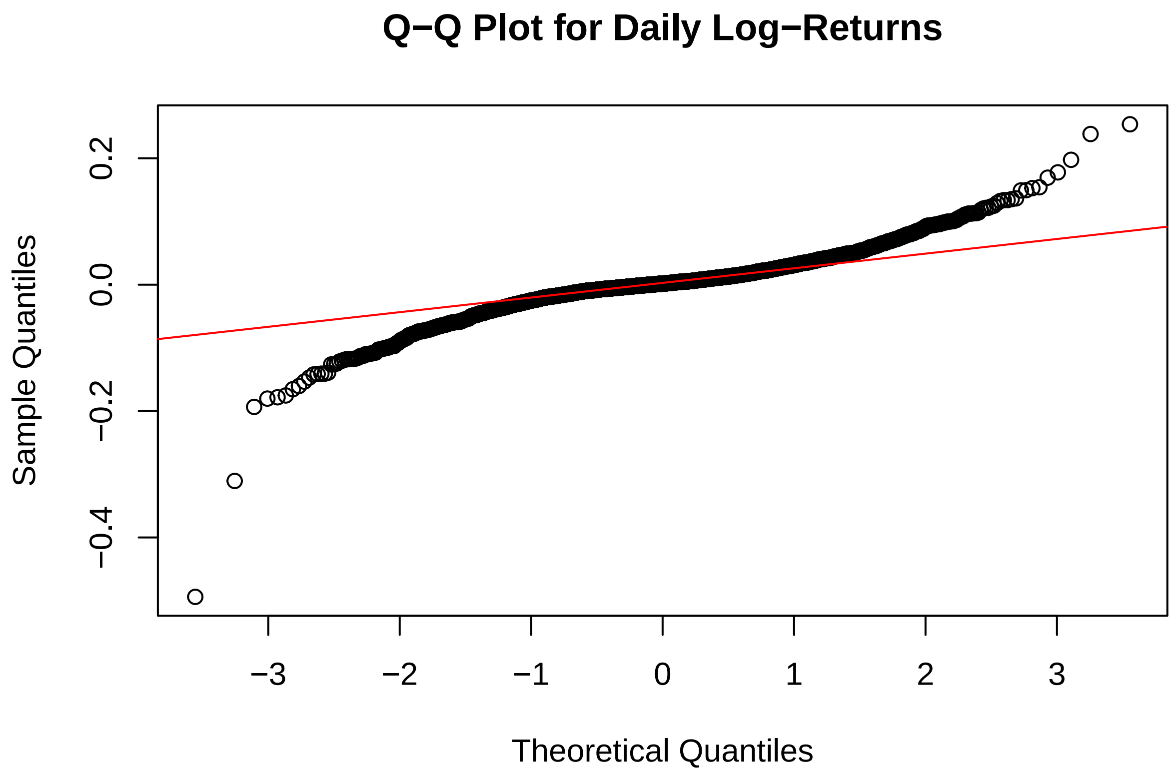

```
abs_lrs <- abs(log_returns)
plot(abs_lrs)
```



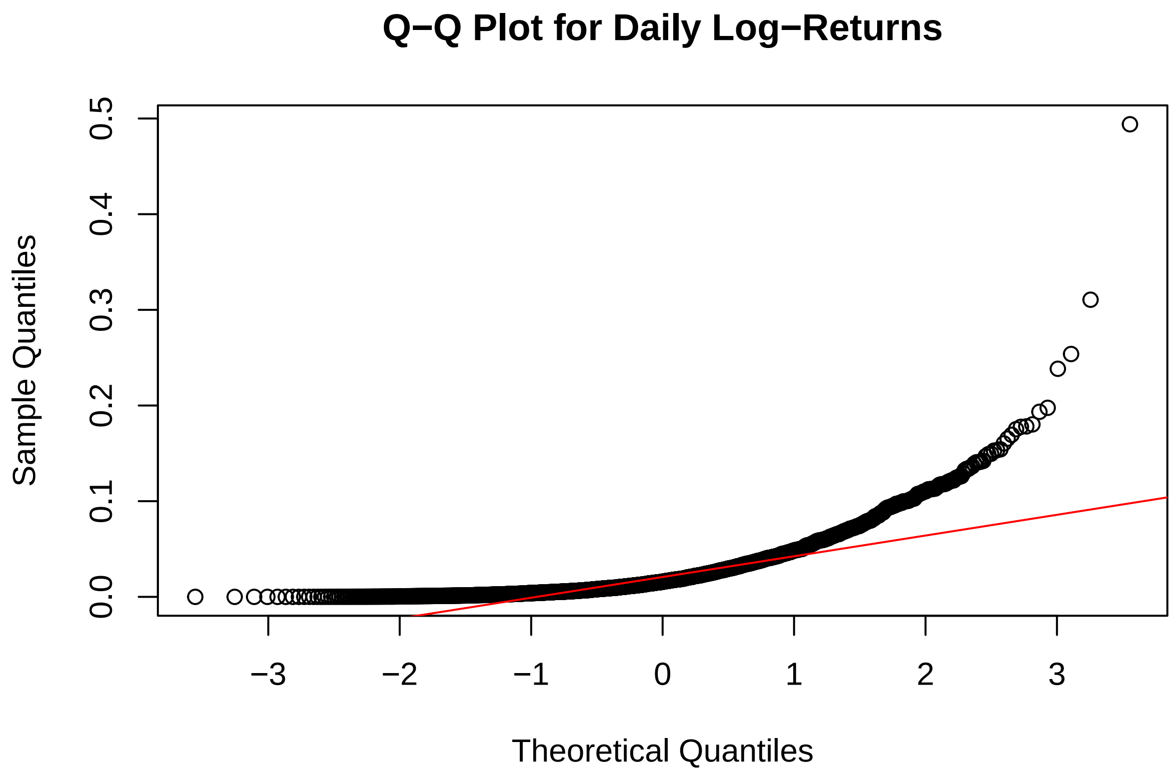

```r
acf(coredata(abs_lrs), main="Sample Autocorrelation of Daily Absolute Log-Returns")
```




```r
qqnorm(log_returns, main="Q-Q Plot for Daily Log-Returns")
qqline(log_returns, col="red")
```




```
qqnorm(abs_lrs, main="Q-Q Plot for Daily Log-Returns")
qqline(abs_lrs, col="red")
```



Split into train, test

```
n <- length(log_returns)

# Split index
test_size <- 30
train <- log_returns[1:(n - test_size)]
test  <- log_returns[(n - test_size + 1):n]
```

Arima model

```
fit_auto <- auto.arima(train)
summary(fit_auto)
```

```
## Series: train
## ARIMA(0,0,1) with non-zero mean
##
## Coefficients:
##           ma1    mean
##       -0.0513  0.0018
## s.e.   0.0192  0.0007
##
## sigma^2 = 0.001612:  log likelihood = 4707.26
```

```
## AIC=-9408.52   AICc=-9408.51   BIC=-9390.91
##
## Training set error measures:
##                        ME      RMSE        MAE MPE MAPE     MASE           ACF1
## Training set 1.38876e-07 0.0401311 0.02591193 NaN  Inf 0.658159 -0.0009093721
```

```
forecast_auto <- forecast(fit_auto, h = 30)
```

ARIMA(0,0,1) with non-zero mean

Coefficients: ma1 mean -0.0513 0.0018 s.e. 0.0192 0.0007

```
library(forecast)

fit <- auto.arima(train)
fc  <- forecast(fit, h = test_size)

accuracy(fc, test)
```

```
##                         ME       RMSE        MAE      MPE     MAPE      MASE
## Training set 1.388760e-07 0.04013110 0.02591193      NaN      Inf 0.6581590
## Test set     2.540786e-03 0.04155324 0.02821374 62.67331 133.9955 0.7166247
##                       ACF1
## Training set -0.0009093721
## Test set                NA
```

metrics in ordinary format (not log)

```
# Get the last known price before forecast starts
last_price <- coredata(x)[length(x) - test_size]

# Convert forecasted log-returns to prices
predicted_prices <- numeric(test_size)
predicted_prices[1] <- last_price * exp(fc$mean[1])

for (i in 2:test_size) {
  predicted_prices[i] <- predicted_prices[i - 1] * exp(fc$mean[i])
}

# Get true prices to compare against
true_prices <- coredata(x)[(length(x) - test_size + 1):(length(x))]

# Evaluate metrics manually
mae  <- mean(abs(true_prices - predicted_prices))
rmse <- sqrt(mean((true_prices - predicted_prices)^2))

cat("MAE: ", round(mae, 4), "\n")
```

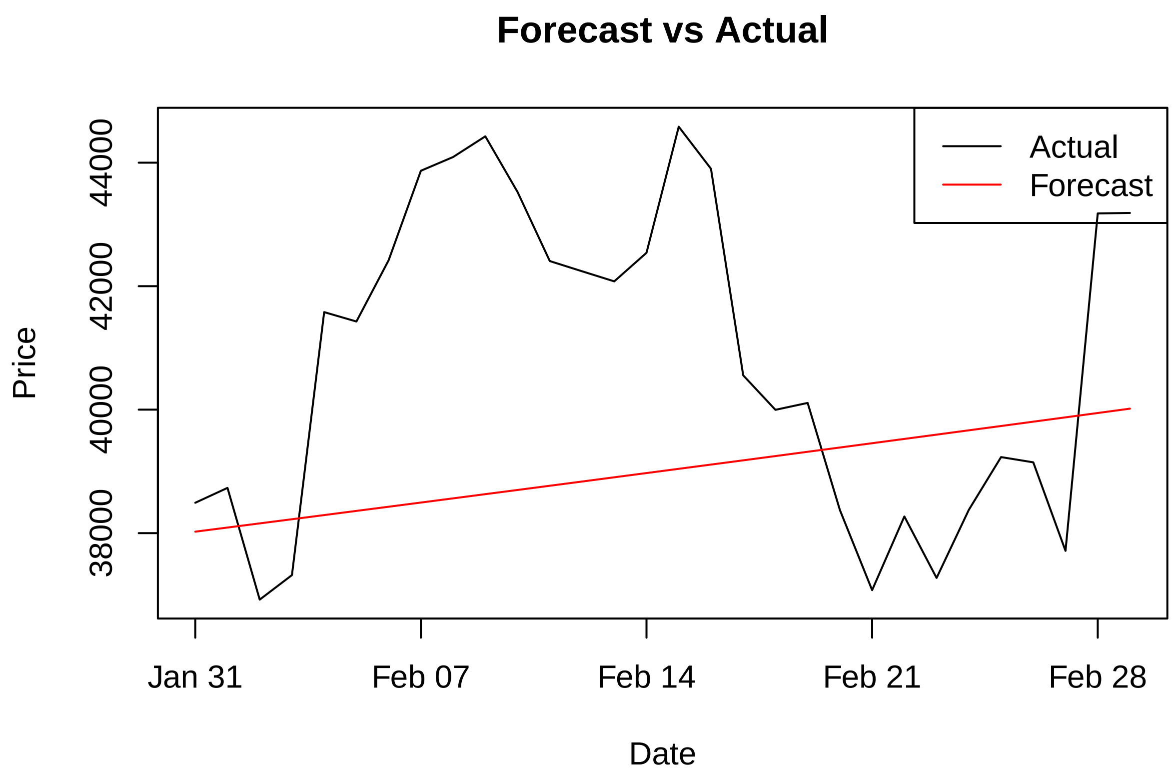
```
## MAE:  2671.279
```

```
cat("RMSE:", round(rmse, 4), "\n")
```

```
## RMSE: 3167.454
```

```r
arima_pred_prices = predicted_prices
```

```r
plot(index(x)[(length(x) - test_size + 1):length(x)], true_prices,
     type = "l", col = "black", ylab = "Price", xlab = "Date", main = "Forecast vs Actual")
lines(index(x)[(length(x) - test_size + 1):length(x)], predicted_prices, col = "red")
legend("topright", legend = c("Actual", "Forecast"), col = c("black", "red"), lty = 1)
```



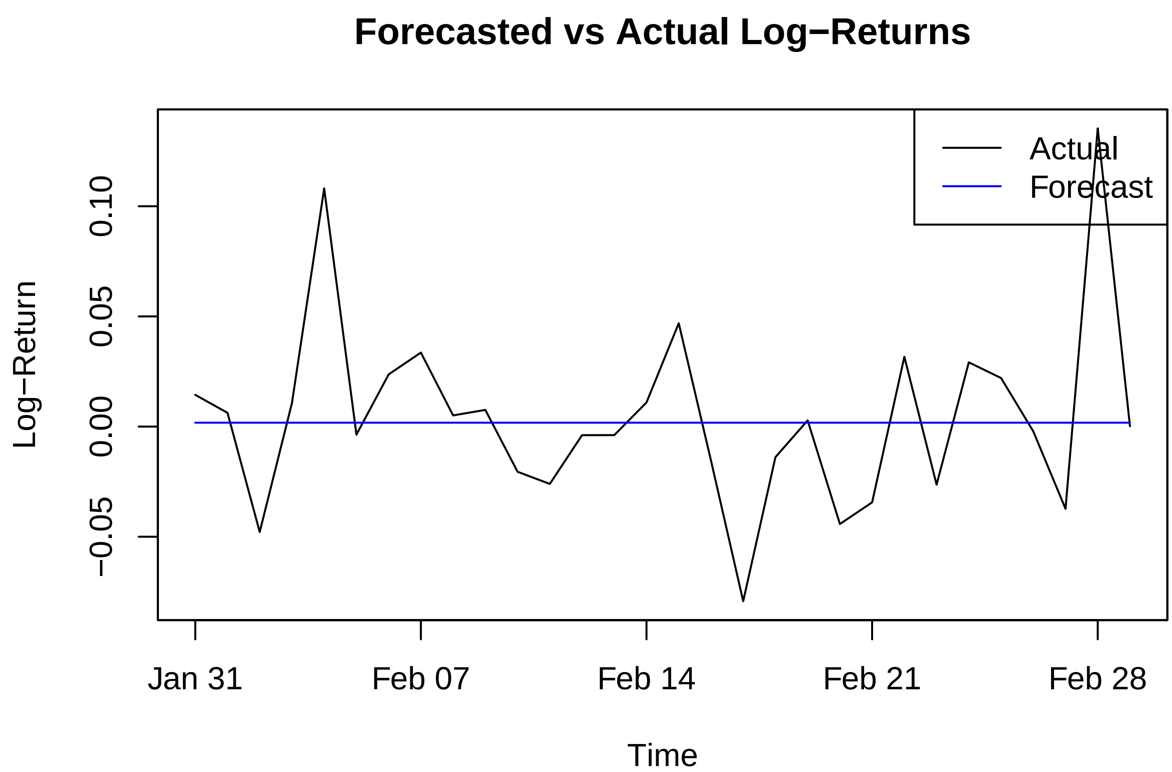
Exponential smoothing

```r
library(forecast)
fit_ets <- ets(train) # train is already in log scale.
fc_ets <- forecast(fit_ets, h = test_size)

accuracy(fc_ets)
```

```
##                        ME       RMSE        MAE  MPE MAPE      MASE        ACF1
## Training set 3.584627e-06 0.04018761 0.02601443 -Inf  Inf 0.6607625 -0.05288728
```

```r
actual_log_returns <- test
forecasted_log_returns <- fc_ets$mean
```

```r
# Plot
plot(actual_log_returns, type = "l", col = "black",
     ylab = "Log-Return", xlab = "Time", main = "Forecasted vs Actual Log-Returns")
lines(forecasted_log_returns, col = "blue")
legend("topright", legend = c("Actual", "Forecast"), col = c("black", "blue"), lty = 1)
```




```r
last_price <- coredata(x)[length(x) - test_size]  # last actual price before forecast

# Reconstruct price forecast
predicted_prices_ets <- numeric(test_size)
predicted_prices_ets[1] <- last_price * exp(fc_ets$mean[1])

for (i in 2:test_size) {
  predicted_prices_ets[i] <- predicted_prices_ets[i - 1] * exp(fc_ets$mean[i])
}
```

```r
true_prices <- coredata(x)[(length(x) - test_size + 1):(length(x))]

# Metrics
mae_ets  <- mean(abs(true_prices - predicted_prices_ets))
rmse_ets <- sqrt(mean((true_prices - predicted_prices_ets)^2))

cat("ETS Forecast\n")
```

```
## ETS Forecast
```

```r
cat("MAE: ", round(mae_ets, 4), "\n")
```

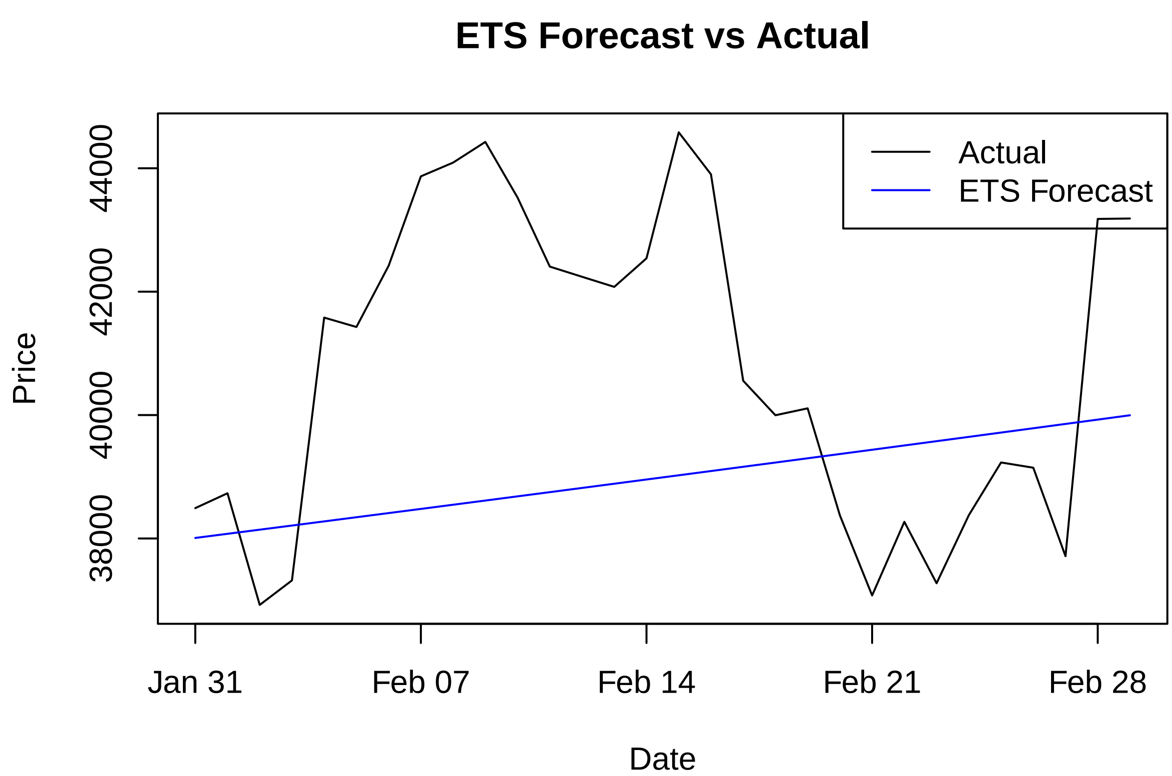
```
## MAE:  2676.604
```

```r
cat("RMSE:", round(rmse_ets, 4), "\n")
```

```
## RMSE: 3176.747
```

plot

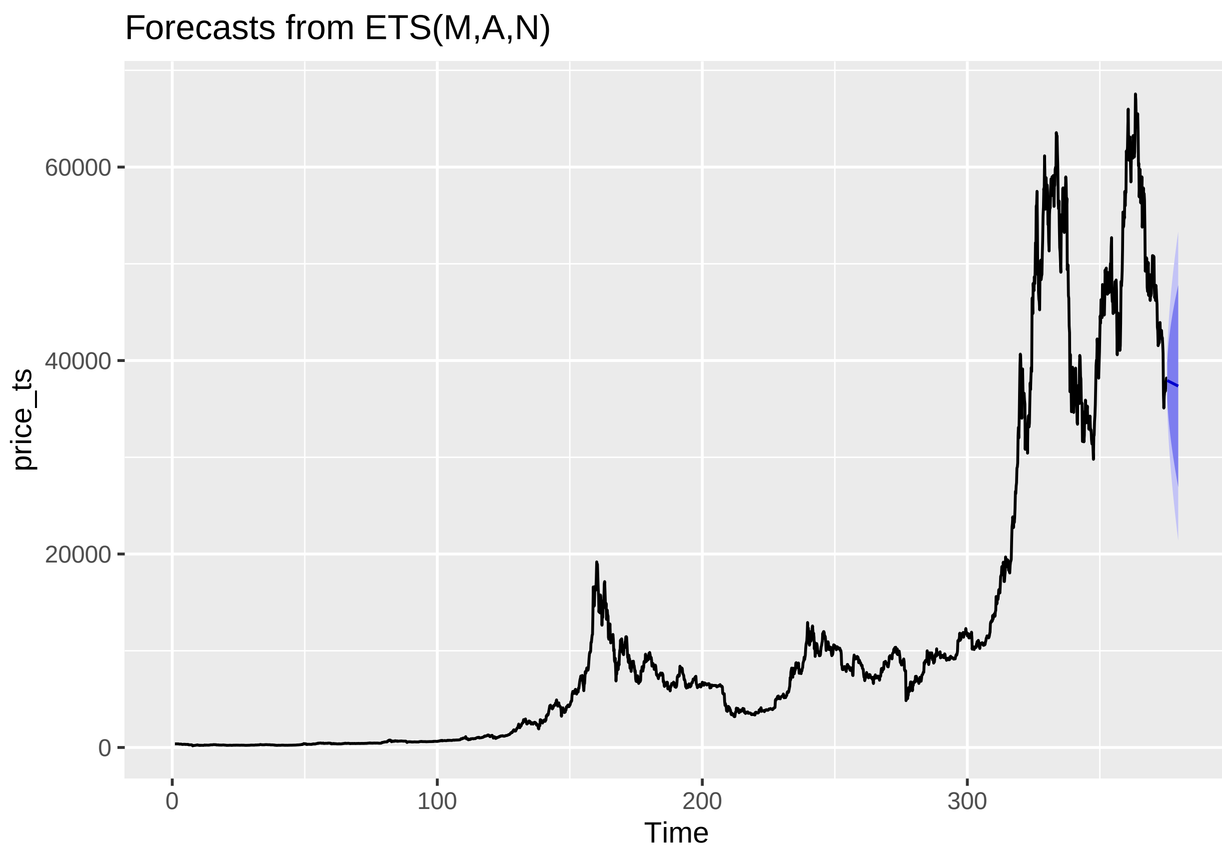
```r
plot(index(x)[(length(x) - test_size + 1):length(x)], true_prices,
     type = "l", col = "black", ylab = "Price", xlab = "Date", main = "ETS Forecast vs Actual")
lines(index(x)[(length(x) - test_size + 1):length(x)], predicted_prices_ets, col = "blue")
legend("topright", legend = c("Actual", "ETS Forecast"), col = c("black", "blue"), lty = 1)
```



ETS on just price

```r
price_ts <- ts(coredata(x)[1:(length(x) - test_size)], frequency = 7)  # weekly freq (adjust as needed)

fit_ets_price <- ets(price_ts)
fc_price <- forecast(fit_ets_price, h = test_size)

predicted_prices <- fc_price$mean
true_prices <- coredata(x)[(length(x) - test_size + 1):length(x)]
```

```r
ets_raw_pred_price = predicted_prices

autoplot(fc_price)
```




```r
mae  <- mean(abs(true_prices - predicted_prices))
rmse <- sqrt(mean((true_prices - predicted_prices)^2))

mae
```

```
## [1] 3265.099
```

```r
rmse
```

```
## [1] 3982.001
```

```r
accuracy(fc_price, true_prices)
```

```
##                       ME      RMSE       MAE          MPE     MAPE     MASE
## Training set   -1.856009  783.7749  333.4589 -0.006612916 2.601616 0.996717
## Test set     3119.879498 3982.0007 3265.0990  7.308372829 7.699924 9.759462
##                    ACF1
## Training set 0.04045423
## Test set             NA
```

GARCH

```r
library(rugarch)

best_aic <- Inf
best_model <- NULL
best_spec <-

train_vec <- as.numeric(train)

for (p in 0:2) {
  for (q in 0:2) {
    for (r in 1:2) {
      for (s in 1:2) {
        cat(sprintf("Trying ARMA(%d,%d)-GARCH(%d,%d)\n", p, q, r, s))

        spec <- ugarchspec(
          mean.model = list(armaOrder = c(p, q), include.mean = TRUE),
          variance.model = list(garchOrder = c(r, s)),
          distribution.model = "norm"  # Or "std" for Student-t
        )

        tryCatch({
          fit <- ugarchfit(spec, data = train_vec, solver = "hybrid")
          aic <- infocriteria(fit)[1]  # AIC

          if (aic < best_aic) {
            best_aic <- aic
            best_model <- fit
            best_spec <- spec
          }
        }, error = function(e) {
          cat("Model failed: ", e$message, "\n")
        })
      }
    }
  }
}
```

```
## Trying ARMA(0,0)-GARCH(1,1)
## Trying ARMA(0,0)-GARCH(1,2)
## Trying ARMA(0,0)-GARCH(2,1)
## Trying ARMA(0,0)-GARCH(2,2)
## Trying ARMA(0,1)-GARCH(1,1)
## Trying ARMA(0,1)-GARCH(1,2)
## Trying ARMA(0,1)-GARCH(2,1)
## Trying ARMA(0,1)-GARCH(2,2)
## Trying ARMA(0,2)-GARCH(1,1)
## Trying ARMA(0,2)-GARCH(1,2)
## Trying ARMA(0,2)-GARCH(2,1)
## Trying ARMA(0,2)-GARCH(2,2)
## Trying ARMA(1,0)-GARCH(1,1)
```

```
## Trying ARMA(1,0)-GARCH(1,2)
## Trying ARMA(1,0)-GARCH(2,1)
## Trying ARMA(1,0)-GARCH(2,2)
## Trying ARMA(1,1)-GARCH(1,1)
## Trying ARMA(1,1)-GARCH(1,2)
## Trying ARMA(1,1)-GARCH(2,1)
## Trying ARMA(1,1)-GARCH(2,2)
## Trying ARMA(1,2)-GARCH(1,1)
## Trying ARMA(1,2)-GARCH(1,2)
## Trying ARMA(1,2)-GARCH(2,1)
## Trying ARMA(1,2)-GARCH(2,2)
## Trying ARMA(2,0)-GARCH(1,1)
## Trying ARMA(2,0)-GARCH(1,2)
## Trying ARMA(2,0)-GARCH(2,1)
## Trying ARMA(2,0)-GARCH(2,2)
## Trying ARMA(2,1)-GARCH(1,1)
## Trying ARMA(2,1)-GARCH(1,2)
## Trying ARMA(2,1)-GARCH(2,1)
## Trying ARMA(2,1)-GARCH(2,2)
## Trying ARMA(2,2)-GARCH(1,1)
## Trying ARMA(2,2)-GARCH(1,2)
## Trying ARMA(2,2)-GARCH(2,1)
## Trying ARMA(2,2)-GARCH(2,2)
```

```r
cat("\nBest model AIC:", best_aic, "\n")
```

```
##
## Best model AIC: -3.773393
```

```r
show(best_model)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(2,0,2)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001994    0.000661   3.0165 0.002557
## ar1     1.575740    0.021763  72.4039 0.000000
## ar2    -0.849327    0.020577 -41.2758 0.000000
## ma1    -1.590459    0.024182 -65.7713 0.000000
## ma2     0.879828    0.015622  56.3214 0.000000
## omega   0.000062    0.000010   6.3333 0.000000
## alpha1  0.139252    0.015665   8.8894 0.000000
## beta1   0.838630    0.015208  55.1447 0.000000
```

```
##
## Robust Standard Errors:
##         Estimate  Std. Error   t value Pr(>|t|)
## mu      0.001994    0.000689    2.8921 0.003827
## ar1     1.575740    0.024820   63.4874 0.000000
## ar2    -0.849327    0.033927  -25.0338 0.000000
## ma1    -1.590459    0.015873 -100.1975 0.000000
## ma2     0.879828    0.021770   40.4146 0.000000
## omega   0.000062    0.000025    2.5406 0.011066
## alpha1  0.139252    0.037162    3.7471 0.000179
## beta1   0.838630    0.027188   30.8451 0.000000
##
## LogLikelihood : 4951.145
##
## Information Criteria
## ------------------------------------
##
## Akaike       -3.7734
## Bayes        -3.7555
## Shibata      -3.7734
## Hannan-Quinn -3.7669
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                          statistic   p-value
## Lag[1]                       3.822 0.0505840
## Lag[2*(p+q)+(p+q)-1][11]     8.266 0.0003371
## Lag[4*(p+q)+(p+q)-1][19]    15.062 0.0274064
## d.o.f=4
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                      0.234  0.6286
## Lag[2*(p+q)+(p+q)-1][5]     2.095  0.5961
## Lag[4*(p+q)+(p+q)-1][9]     3.111  0.7404
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]     1.725 0.500 2.000  0.1890
## ARCH Lag[5]     2.865 1.440 1.667  0.3100
## ARCH Lag[7]     3.249 2.315 1.543  0.4678
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1.9205
## Individual Statistics:
## mu     0.12078
## ar1    0.03063
## ar2    0.06082
## ma1    0.06063
```

```
## ma2    0.05446
## omega  0.70892
## alpha1 0.08441
## beta1  0.41949
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.89 2.11 2.59
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias            1.324 0.1856
## Negative Sign Bias   1.063 0.2880
## Positive Sign Bias   1.417 0.1565
## Joint Effect         3.191 0.3631
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     426.9    1.120e-78
## 2    30     445.5    4.131e-76
## 3    40     465.6    1.882e-74
## 4    50     464.5    4.870e-69
##
##
## Elapsed time : 0.3868918
```

```
best_garch = best_model
```

The model is Conditional Variance Dynamics, GARCH Model : sGARCH(1,1), Mean Model :ARFIMA(2,0,2), Distribution : norm

```
# garch_spec <- ugarchspec(
#   mean.model     = list(armaOrder = c(1, 1), include.mean = TRUE),
#   variance.model = list(garchOrder = c(1, 1)),
#   distribution.model = "norm"  # you can change to "std" for Student-t
# )
#
#
# # train is your zoo object of log-returns, from earlier
# train_vec <- as.numeric(train)  # convert to numeric vector
#
# garch_fit <- ugarchfit(spec = garch_spec, data = train_vec)
# show(garch_fit)
```

```
# use best one
garch_fc <- ugarchforecast(best_garch, n.ahead = test_size)

# Extract forecasted log-returns
fc_log_returns_garch <- fitted(garch_fc)
```

```r
last_price <- coredata(x)[length(x) - test_size]

# Reconstruct forecasted prices from log-returns
predicted_prices_garch <- numeric(test_size)
predicted_prices_garch[1] <- last_price * exp(fc_log_returns_garch[1])

for (i in 2:test_size) {
  predicted_prices_garch[i] <- predicted_prices_garch[i - 1] * exp(fc_log_returns_garch[i])
}


true_prices <- coredata(x)[(length(x) - test_size + 1):length(x)]

mae_garch  <- mean(abs(true_prices - predicted_prices_garch))
rmse_garch <- sqrt(mean((true_prices - predicted_prices_garch)^2))

cat("GARCH Forecast\n")
```

```
## GARCH Forecast
```

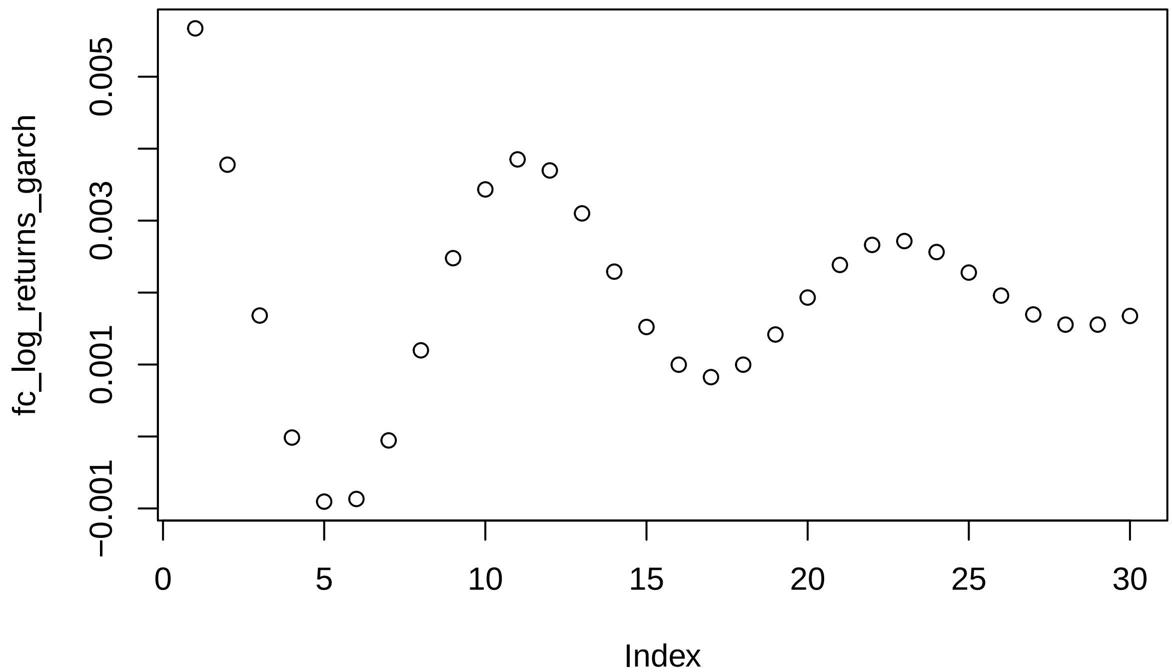
```r
cat("MAE : ", round(mae_garch, 4), "\n")
```

```
## MAE :  2688.547
```

```r
cat("RMSE: ", round(rmse_garch, 4), "\n")
```

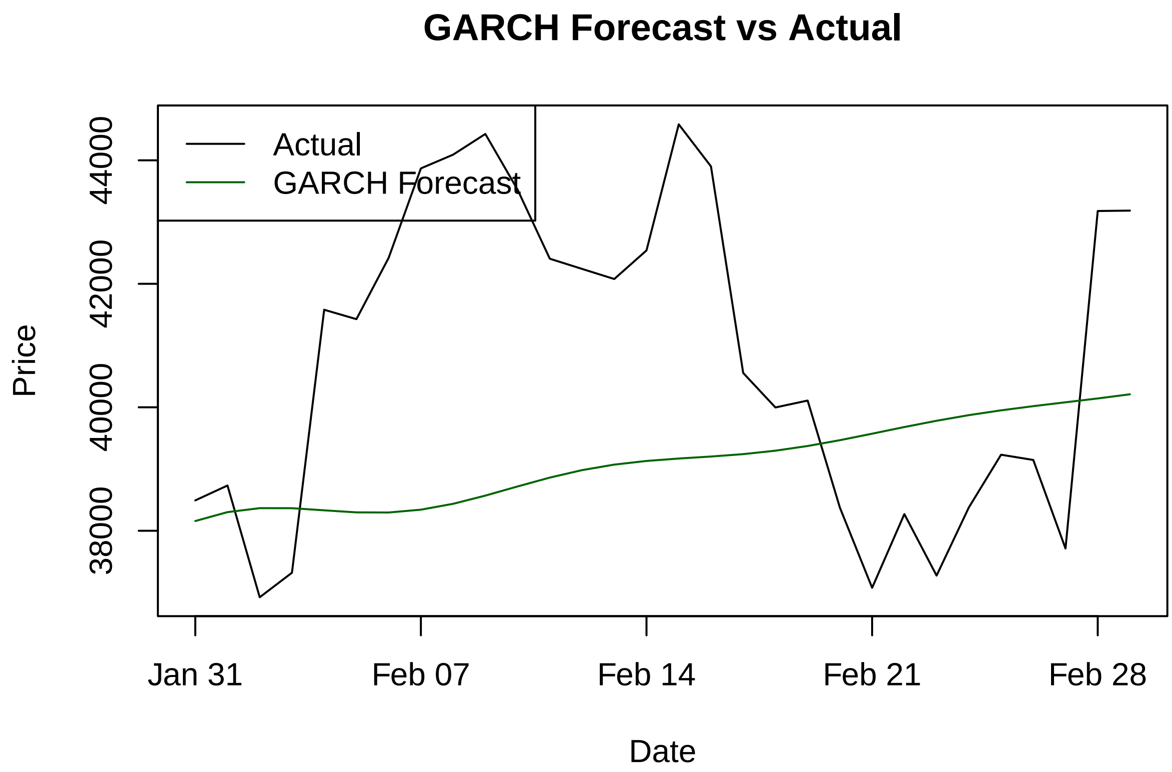
```
## RMSE:  3170.944
```

```r
plot(fc_log_returns_garch)
```



plot

```
plot(index(x)[(length(x) - test_size + 1):length(x)], true_prices,
     type = "l", col = "black", ylab = "Price", xlab = "Date", main = "GARCH Forecast vs Actual")
lines(index(x)[(length(x) - test_size + 1):length(x)], predicted_prices_garch, col = "darkgreen")
legend("topleft", legend = c("Actual", "GARCH Forecast"), col = c("black", "darkgreen"), lty = 1)
```



Summary

Regression (Forecasting)

```
evaluate_model <- function(model_type = c("ARIMA", "ETS", "GARCH"),
                           train, test, full_price_series, test_size = 30, best_garch) {

  model_type <- match.arg(model_type)

  # Reconstruct price from forecasted log-returns
  reconstruct_prices <- function(log_returns, last_price) {
    pred <- numeric(length(log_returns))
    pred[1] <- last_price * exp(log_returns[1])
    for (i in 2:length(log_returns)) {
      pred[i] <- pred[i - 1] * exp(log_returns[i])
    }
    pred
  }

  last_price <- coredata(full_price_series)[length(full_price_series) - test_size]
  true_prices <- coredata(full_price_series)[(length(full_price_series) - test_size + 1):length(full_pr

  if (model_type == "ARIMA") {
```

```r
    fit <- auto.arima(train)
    train_pred <- fitted(fit)
    test_fc <- forecast(fit, h = test_size)$mean

  } else if (model_type == "ETS") {
    fit <- ets(train)
    train_pred <- fitted(fit)
    test_fc <- forecast(fit, h = test_size)$mean

  } else if (model_type == "GARCH") {


    # garch_spec <- ugarchspec(
    #   mean.model = list(armaOrder = c(p, q), include.mean = TRUE),
    #   variance.model = list(garchOrder = c(r, s)),
    #   distribution.model = dist
    # )
    # fit <- ugarchfit(best_garch, as.numeric(train))
    # train_pred <- fitted(fit)
    train_pred <- fitted(best_garch)

    garch_fc <- ugarchforecast(best_garch, n.ahead = test_size)
    test_fc <- fitted(garch_fc)
    # test_fc <- fitted(ugarchforecast(fit, n.ahead = test_size))
  }

  # Price predictions
  predicted_train_prices <- reconstruct_prices(train_pred, coredata(full_price_series)[length(full_pric
  predicted_test_prices  <- reconstruct_prices(test_fc, last_price)

  # Actual prices for training range (approximate)
  actual_train_prices <- coredata(full_price_series)[(length(full_price_series) - test_size - length(tra

  # Compute metrics
  train_mae  <- mean(abs(actual_train_prices - predicted_train_prices))
  train_rmse <- sqrt(mean((actual_train_prices - predicted_train_prices)^2))
  test_mae   <- mean(abs(true_prices - predicted_test_prices))
  test_rmse  <- sqrt(mean((true_prices - predicted_test_prices)^2))

  # Return named result
  data.frame(
    Model = model_type,
    Train_MAE = round(train_mae, 2),
    Test_MAE = round(test_mae, 2),
    Train_RMSE = round(train_rmse, 2),
    Test_RMSE = round(test_rmse, 2)
  )
}
```

```r
evaluate_model(model = "GARCH", train = train, test = test, full_price_series = x, best_garch = best_ga
```

```
##   Model Train_MAE Test_MAE Train_RMSE Test_RMSE
## 1 GARCH   4307.19  2688.55     6806.5   3170.94
```

```r
results <- rbind(
  evaluate_model("ARIMA", train, test, x, test_size),
  evaluate_model("ETS", train, test, x, test_size),
  evaluate_model("GARCH", train, test, x, test_size, best_garch = best_garch)
)
results
```

```
##   Model Train_MAE Test_MAE Train_RMSE Test_RMSE
## 1 ARIMA   4880.19  2671.28    9363.21   3167.45
## 2   ETS   4725.47  2676.60    9185.44   3176.75
## 3 GARCH   4307.19  2688.55    6806.50   3170.94
```

Classification