Stock Forecasting Project

Contents

Libraries

```
library(tidyverse)
library(lubridate)
library(quantmod)
library(tseries)
library(forecast)

library(tsibble)
library(fable)
library(fabletools)
library(feasts)

library(xgboost)
library(caret)

library(rugarch)
```

Data Gathering

User Inputs (Stock and Date)

```
symbol <- "AAPL"
start_date <- as.Date("2020-01-01")
end_date <- Sys.Date()
```

Stock Data Collection

```
getSymbols(symbol,
           src  = "yahoo",
           from = start_date,
           to   = end_date,
           auto.assign = TRUE)
```

```
## [1] "AAPL"
```

```
stock_data <- get(symbol)

# head(stock_data)
```

Data Exploration & Feature Engineering

EDA

```
# Convert time-series data (xts object) to a regular tibble
df_stock <- tibble(
  date     = zoo::index(stock_data),
  open     = as.numeric(stock_data[, paste0(symbol, ".Open")]),
  high     = as.numeric(stock_data[, paste0(symbol, ".High")]),
  low      = as.numeric(stock_data[, paste0(symbol, ".Low")]),
  close    = as.numeric(stock_data[, paste0(symbol, ".Close")]),
  volume   = as.numeric(stock_data[, paste0(symbol, ".Volume")]),
```

```r
  adjusted = as.numeric(stock_data[, paste0(symbol, ".Adjusted")])
)

glimpse(df_stock)
```

```
## Rows: 1,317
## Columns: 7
## $ date     <date> 2020-01-02, 2020-01-03, 2020-01-06, 2020-01-07, 2020-01-08, ~
## $ open     <dbl> 74.0600, 74.2875, 73.4475, 74.9600, 74.2900, 76.8100, 77.6500~
## $ high     <dbl> 75.1500, 75.1450, 74.9900, 75.2250, 76.1100, 77.6075, 78.1675~
## $ low      <dbl> 73.7975, 74.1250, 73.1875, 74.3700, 74.2900, 76.5500, 77.0625~
## $ close    <dbl> 75.0875, 74.3575, 74.9500, 74.5975, 75.7975, 77.4075, 77.5825~
## $ volume   <dbl> 135480400, 146322800, 118387200, 108872000, 132079200, 170108~
## $ adjusted <dbl> 72.71608, 72.00910, 72.58289, 72.24154, 73.40366, 74.96281, 7~
```

```r
summary(df_stock)
```

```
##       date                 open             high              low        
##  Min.   :2020-01-02   Min.   : 57.02   Min.   : 57.12   Min.   : 53.15  
##  1st Qu.:2021-04-23   1st Qu.:130.47   1st Qu.:132.22   1st Qu.:129.04  
##  Median :2022-08-12   Median :155.08   Median :157.33   Median :153.46  
##  Mean   :2022-08-13   Mean   :157.44   Mean   :159.20   Mean   :155.81  
##  3rd Qu.:2023-12-04   3rd Qu.:182.80   3rd Qu.:184.66   3rd Qu.:181.47  
##  Max.   :2025-03-28   Max.   :258.19   Max.   :260.10   Max.   :257.63  
##      close            volume             adjusted     
##  Min.   : 56.09   Min.   : 23234700   Min.   : 54.45  
##  1st Qu.:130.84   1st Qu.: 54126800   1st Qu.:128.13  
##  Median :155.35   Median : 74829200   Median :153.70  
##  Mean   :157.59   Mean   : 88879545   Mean   :155.74  
##  3rd Qu.:182.91   3rd Qu.:105425600   3rd Qu.:181.77  
##  Max.   :259.02   Max.   :426510000   Max.   :258.74
```

```r
# Plot Adjusted Closing Price
ggplot(df_stock, aes(x = date, y = adjusted)) +
  geom_line() +
  labs(title = paste(symbol, "Adjusted Closing Price"),
       x = "Date",
       y = "Adjusted Price") +
  theme_minimal()
```


```r
# Check if there is any NA (rare to have NA)
df_stock %>%
  summarize(across(everything(), ~ sum(is.na(.))))
```

```
## # A tibble: 1 x 7
##    date  open  high   low close volume adjusted
##   <int> <int> <int> <int> <int>  <int>    <int>
## 1     0     0     0     0     0      0        0
```

Feature Engineering

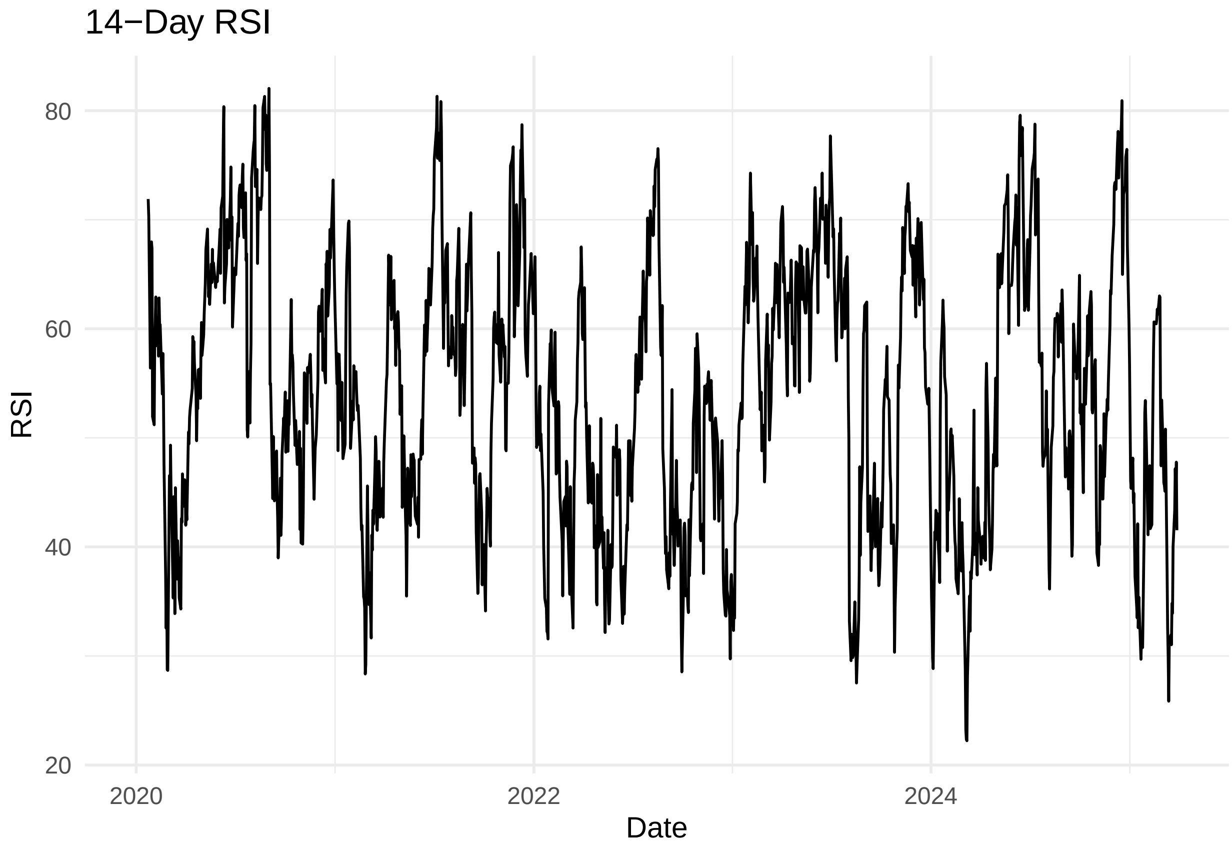
Add new variables

```r
# Bollinger Bands and MACD
bb <- BBands(HLC = df_stock %>% select(high, low, close),
             n = 20, maType = "SMA", sd = 2)
macd_values <- MACD(df_stock$adjusted, nFast = 12, nSlow = 26, nSig = 9)

# Add all new variables
df_stock <- df_stock %>%
  arrange(date) %>%
  mutate(
    daily_return = (adjusted - lag(adjusted)) / lag(adjusted) * 100,
    lag1_close = lag(adjusted, 1),
    lag2_close = lag(adjusted, 2),
    ma20 = rollmean(adjusted, k = 20, fill = NA, align = "right"),
```

```
    ma50 = rollmean(adjusted, k = 50, fill = NA, align = "right"),
    rsi14 = RSI(adjusted, n = 14),
    bb_dn = bb[, "dn"], # lower band
    bb_mavg = bb[, "mavg"],
    bb_up = bb[, "up"],
    bb_pctB = bb[, "pctB"], # 0-1: 0 lower band, 1 upper band
    macd = macd_values[, "macd"],
    macdSig = macd_values[, "signal"],
    rolling_sd_20 = rollapply(daily_return, width = 20,
                              FUN = sd, fill = NA, align = "right"),
    wday = wday(date, label = TRUE),
    sin_wday = sin(2 * pi * wday(date) / 7),
    cos_wday = cos(2 * pi * wday(date) / 7)
  )
```

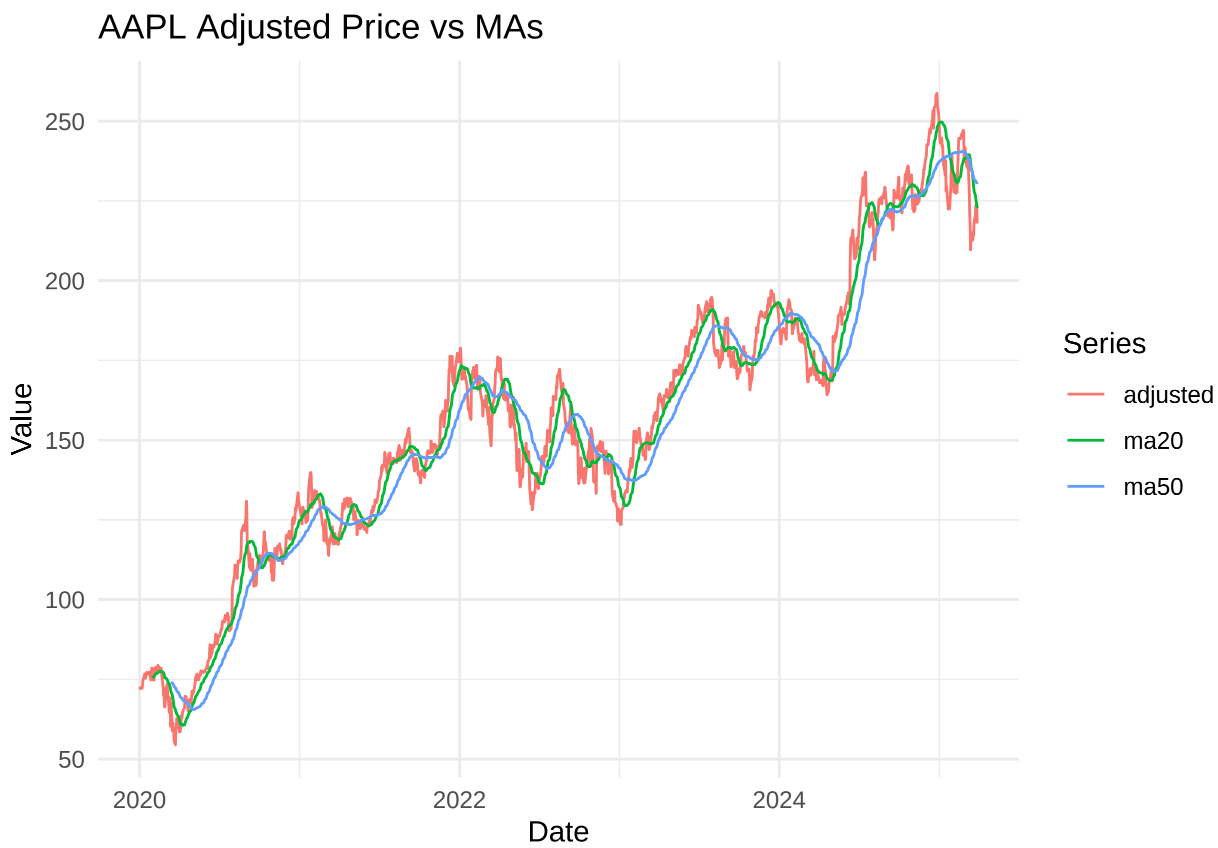
Visualize new variables

```
# 14-Day RSI
ggplot(df_stock, aes(x = date, y = rsi14)) +
  geom_line() +
  labs(title = "14-Day RSI", x = "Date", y = "RSI") +
  theme_minimal()
```

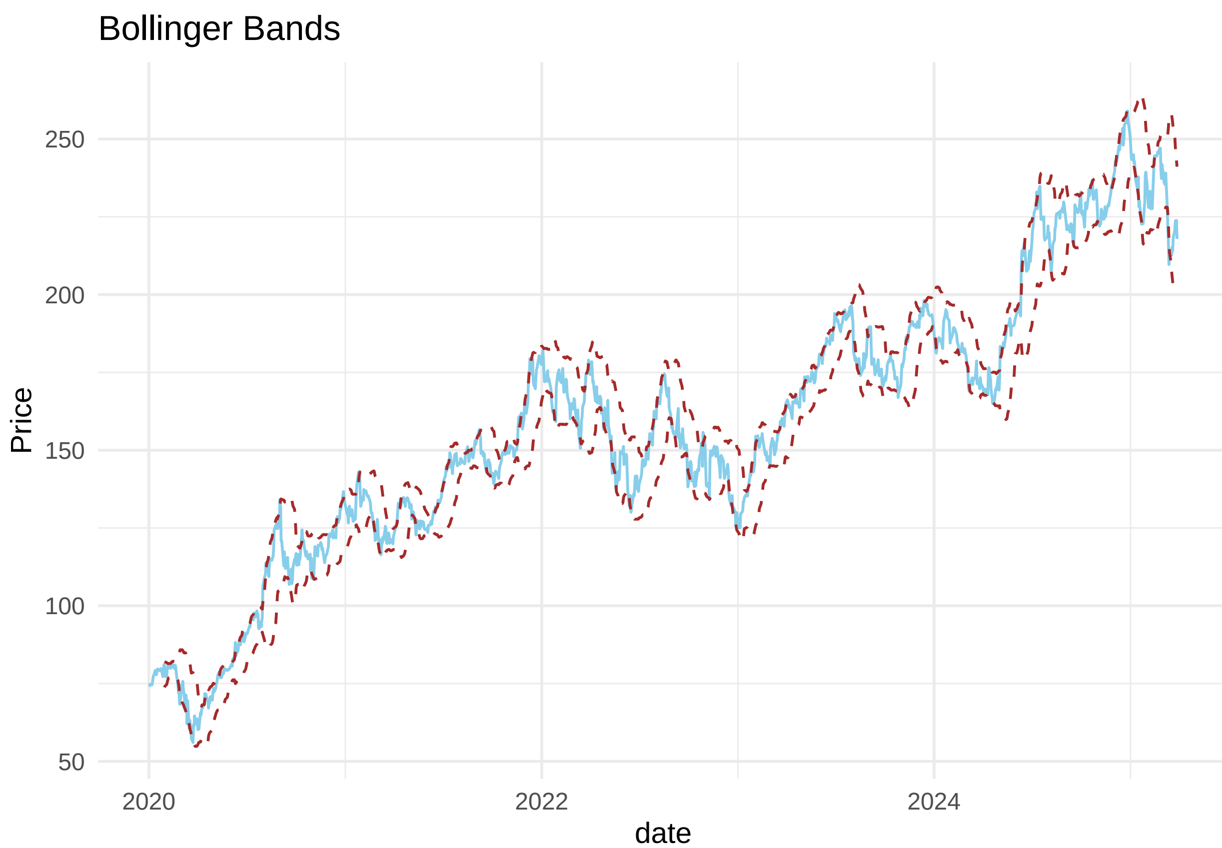



```
# MAs
df_stock_long <- df_stock %>%
  select(date, adjusted, ma20, ma50) %>%
```

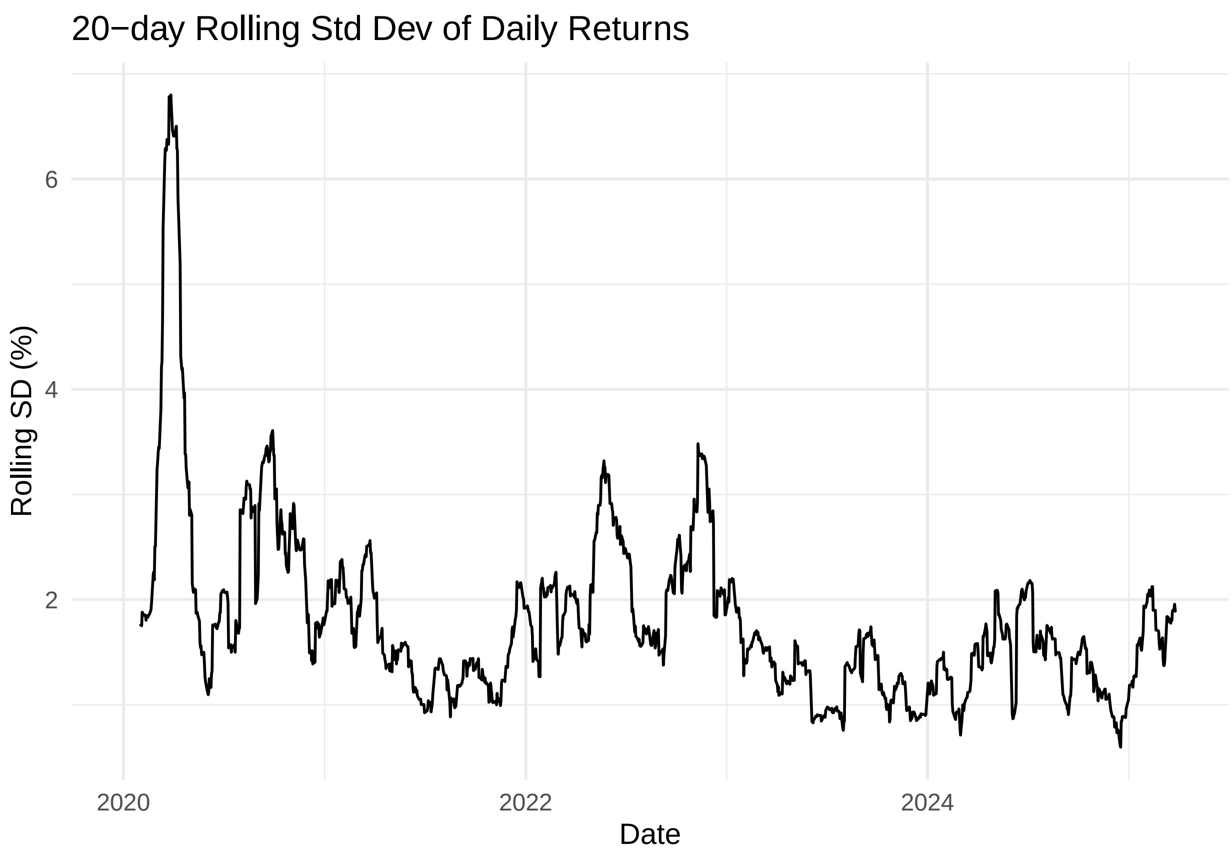
```r
  pivot_longer(cols = c("adjusted", "ma20", "ma50"),
               names_to = "variable", values_to = "value")

ggplot(df_stock_long, aes(x = date, y = value, color = variable)) +
  geom_line() +
  labs(title = paste(symbol, "Adjusted Price vs MAs"),
       x = "Date", y = "Value", color = "Series") +
  theme_minimal()
```




```r
# Bollinger Bands
ggplot(df_stock, aes(x = date)) +
  geom_line(aes(y = close), color = "skyblue") +
  geom_line(aes(y = bb_dn), color = "brown", linetype = "dashed") +
  geom_line(aes(y = bb_up), color = "brown", linetype = "dashed") +
  labs(title = "Bollinger Bands", y = "Price") +
  theme_minimal()
```



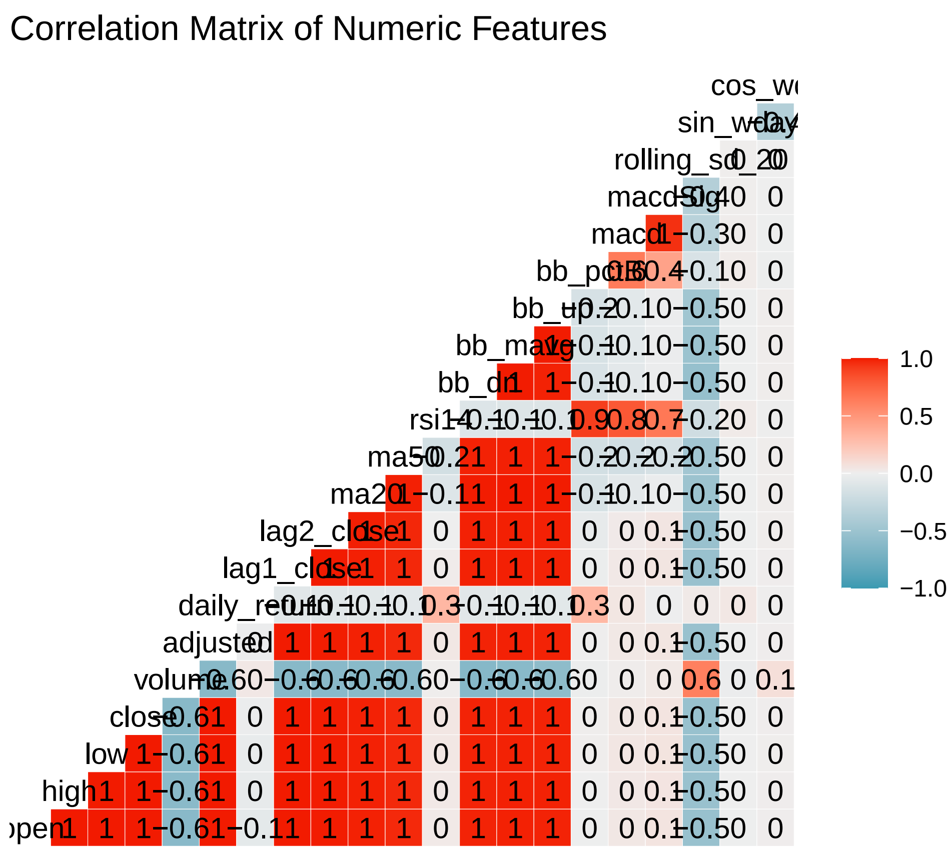

```r
# 20-day Rolling Std Dev of Daily Returns
ggplot(df_stock, aes(x = date, y = rolling_sd_20)) +
  geom_line() +
  labs(title = "20-day Rolling Std Dev of Daily Returns",
       x = "Date", y = "Rolling SD (%)") +
  theme_minimal()
```



Other tests and analysis

```r
# Correlation Matrix
df_numerics <- df_stock %>%
  select(where(is.numeric)) %>%
  drop_na()   # Remove rows with NA for accurate correlation

GGally::ggcorr(df_numerics,
               method = c("pairwise.complete.obs", "pearson"),
               label  = TRUE) +
  ggtitle("Correlation Matrix of Numeric Features")
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```




```r
# Stationarity Tests: Adjusted Price and Daily Returns
adf_result <- adf.test(df_stock$adjusted, alternative = "stationary")
print(adf_result)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df_stock$adjusted
## Dickey-Fuller = -2.9181, Lag order = 10, p-value = 0.1896
## alternative hypothesis: stationary
```

```r
adf_returns <- adf.test(na.omit(df_stock$daily_return), alternative = "stationary")
```

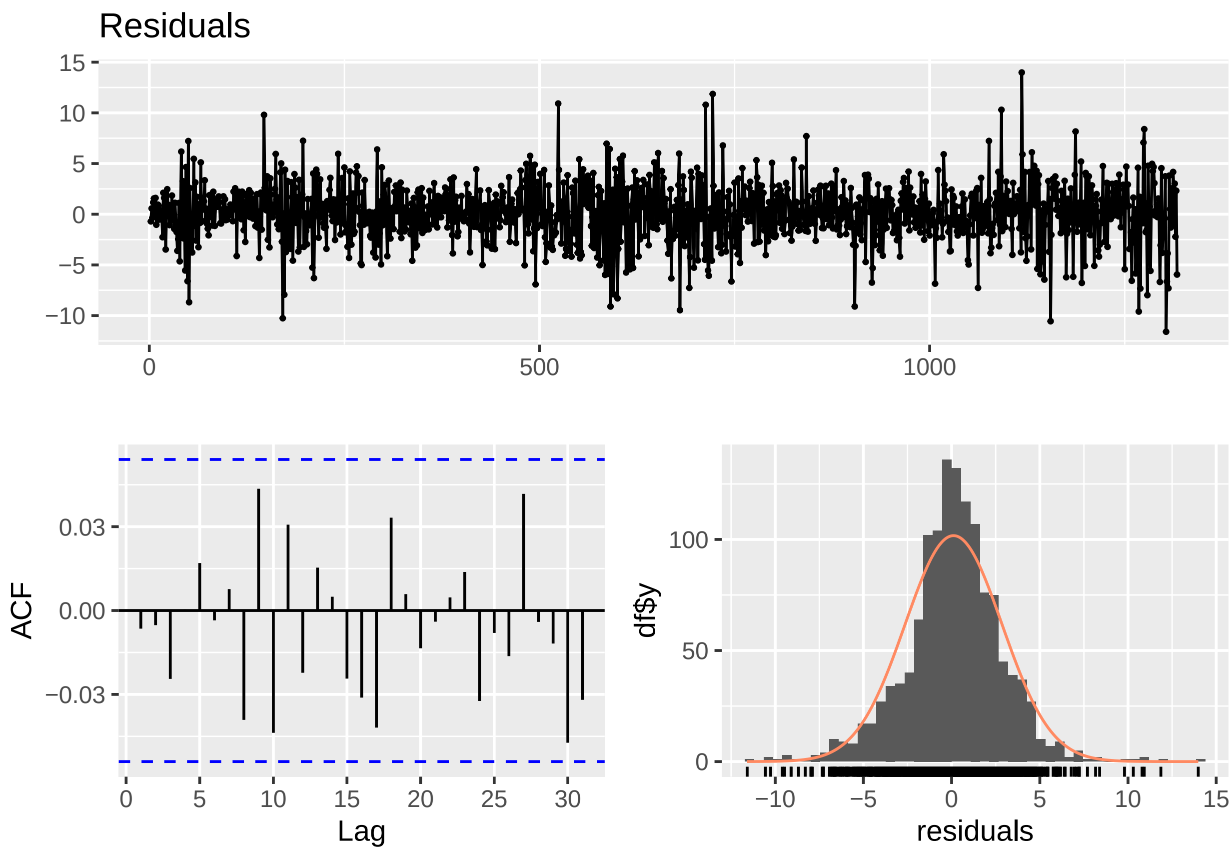
```
## Warning in adf.test(na.omit(df_stock$daily_return), alternative =
## "stationary"): p-value smaller than printed p-value
```

```r
print(adf_returns)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  na.omit(df_stock$daily_return)
## Dickey-Fuller = -10.856, Lag order = 10, p-value = 0.01
## alternative hypothesis: stationary
```

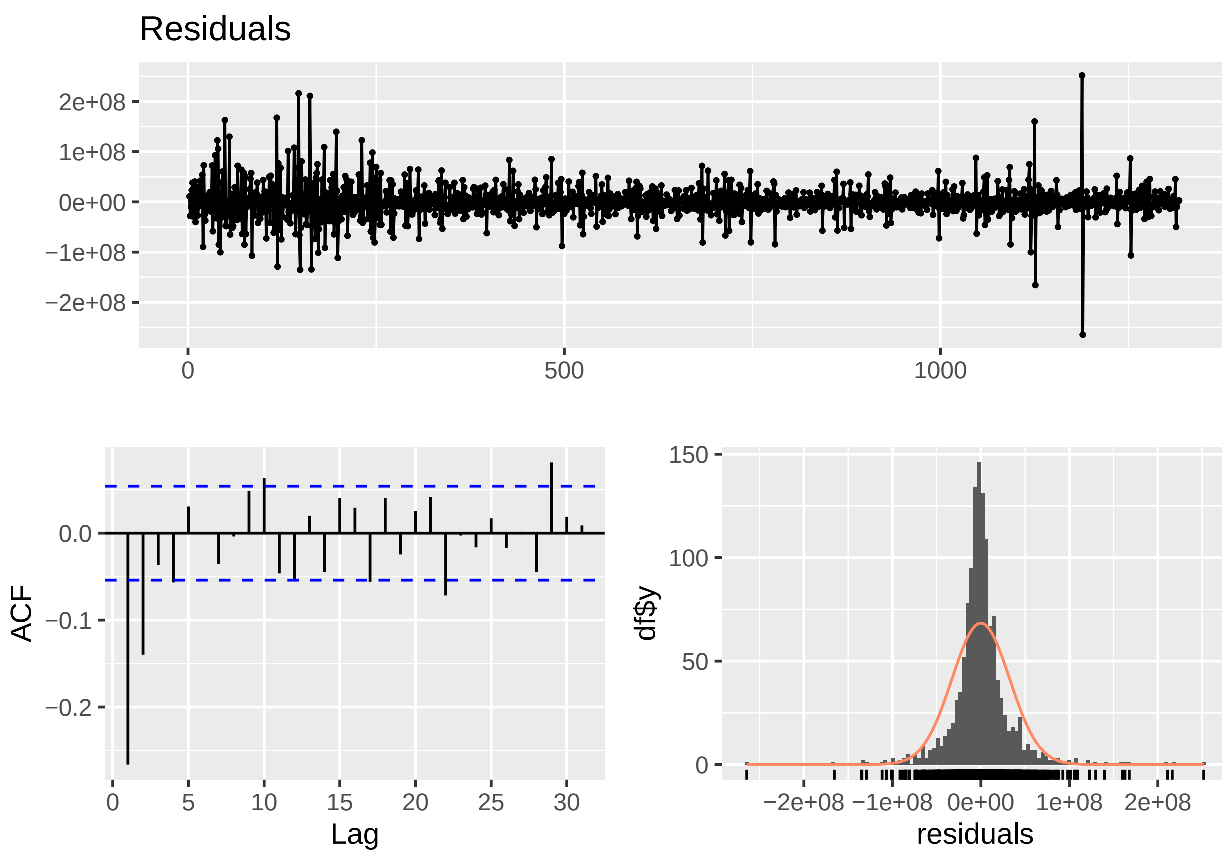
Findings on Stationarity: - The adjusted price is non-stationary, which is expected because stock prices tend to follow a random walk. - The daily returns are stationary, which is typical for financial return series since they fluctuate around a constant mean.

```r
# difference
df_stock <- df_stock %>%
  mutate(
    volume_diff = c(NA, diff(volume)),
    adjusted_diff = c(NA, diff(adjusted)),
    ma20_diff = c(NA, diff(ma20)),
    ma50_diff = c(NA, diff(ma50)),
    rsi14_diff = c(NA, diff(rsi14)),
  )

checkresiduals(df_stock$adjusted_diff)
```



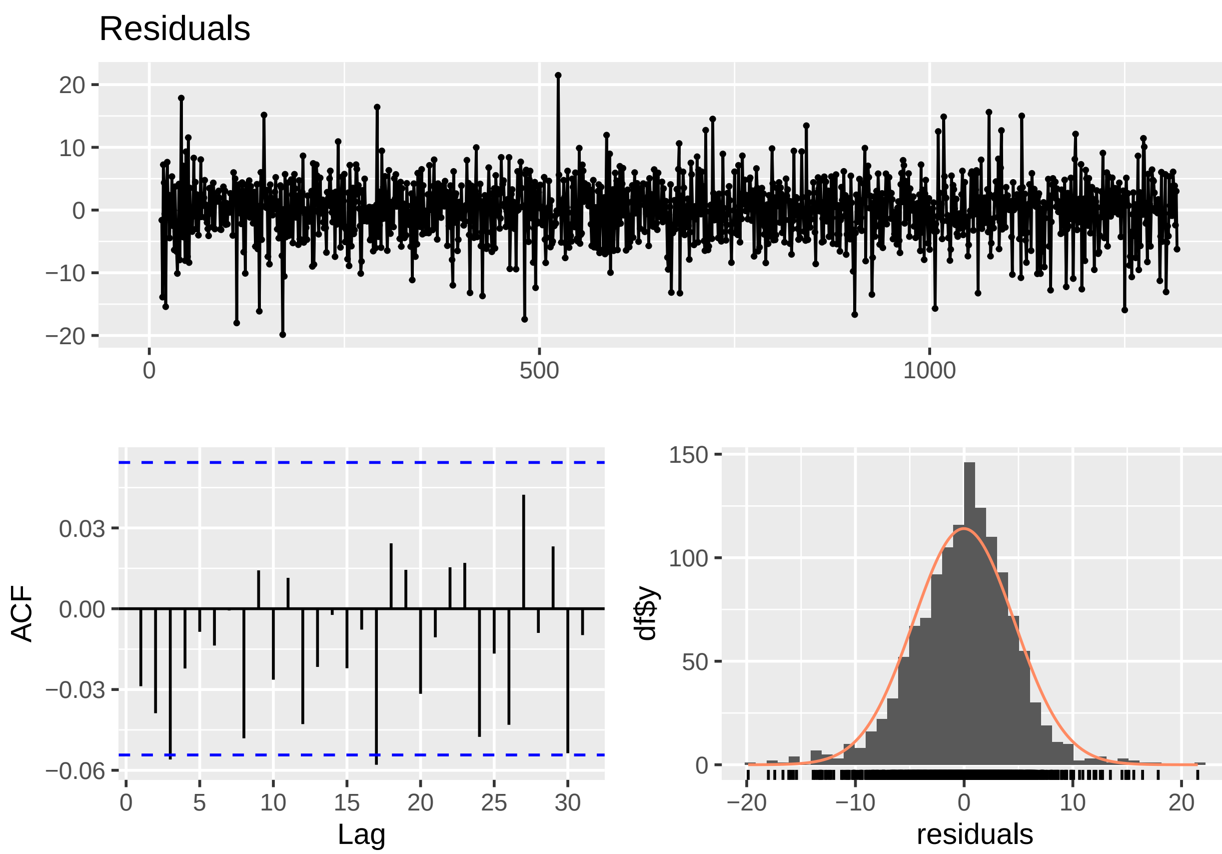

```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 8.445, df = 10, p-value = 0.5855
##
## Model df: 0.   Total lags used: 10
```

```r
checkresiduals(df_stock$volume_diff)
```



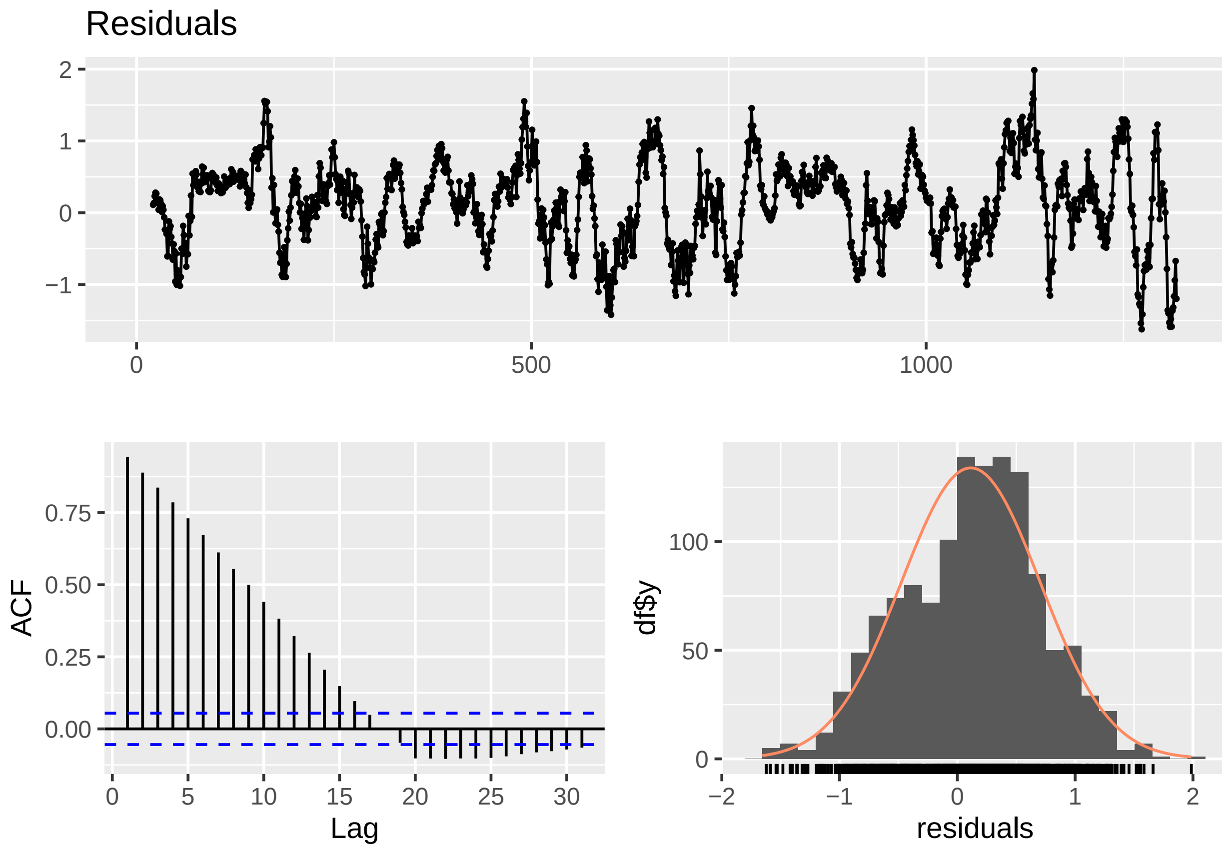

```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 136.56, df = 10, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 10
```

```
checkresiduals(df_stock$rsi14_diff)
```



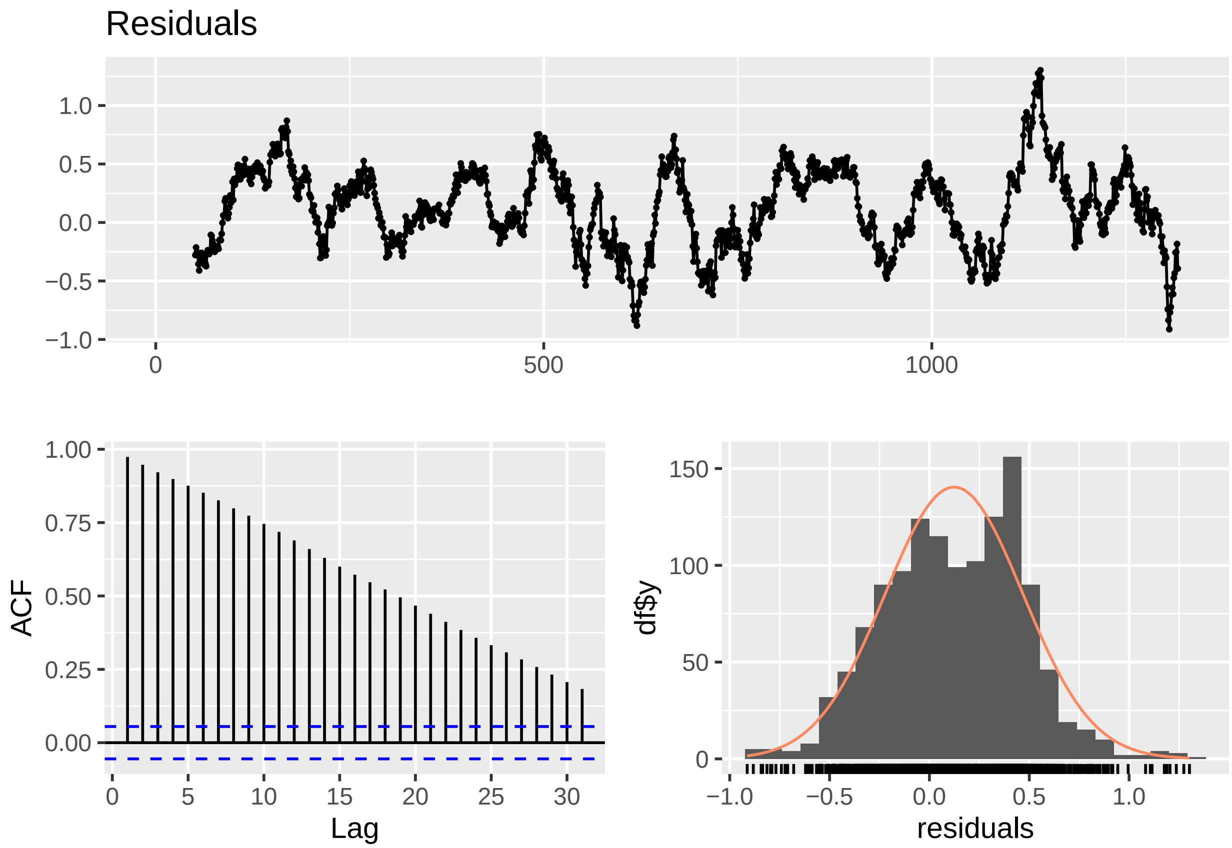

```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 12.349, df = 10, p-value = 0.2624
##
## Model df: 0.   Total lags used: 10
```

```
checkresiduals(df_stock$ma20_diff)
```




```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 6658.7, df = 10, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 10
```

```
checkresiduals(df_stock$ma50_diff)
```




```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 9516.1, df = 10, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 10
```

Based on the Ljung-Box test, `adjusted_diff` and `rsi14_diff` are stationary now. `volume_diff` is still non-stationary, but it is a lot better than the original `volume` variable.

Model 1: ARIMA with Exogenous Regressors (ARIMAX)

Preperation for model 1

Convert to weekly data

```
# Change to weekly data for less computation
df_weekly <- df_stock %>%
  mutate(week = floor_date(date, unit = "week", week_start = 1)) %>%
  group_by(week) %>%
  summarise(
    open = first(open),
    high = max(high, na.rm = TRUE),
    low = min(low, na.rm = TRUE),
    close = last(close),
```

```r
    volume = sum(volume, na.rm = TRUE),
    first_adj = first(adjusted),
    adjusted = last(adjusted),
    weekly_return = (last(adjusted) / first_adj - 1) * 100,
    ma20 = last(ma20),
    ma50 = last(ma50),
    rsi14 = last(rsi14),
    bb_dn = last(bb_dn),
    bb_mavg = last(bb_mavg),
    bb_up = last(bb_up),
    bb_pctB = last(bb_pctB),
    macd = last(macd),
    macdSig = last(macdSig),
    rolling_sd_20 = last(rolling_sd_20),
    wday = last(wday),
    sin_wday = last(sin_wday),
    cos_wday = last(cos_wday),
    volume_diff = mean(volume_diff, na.rm = TRUE),
    adjusted_diff = mean(adjusted_diff, na.rm = TRUE),
    ma20_diff = mean(ma20_diff, na.rm = TRUE),
    ma50_diff = mean(ma50_diff, na.rm = TRUE),
    rsi14_diff = mean(rsi14_diff, na.rm = TRUE),
  ) %>%
  ungroup() %>%
  select(-c(first_adj, wday))

head(df_weekly, 2)
```

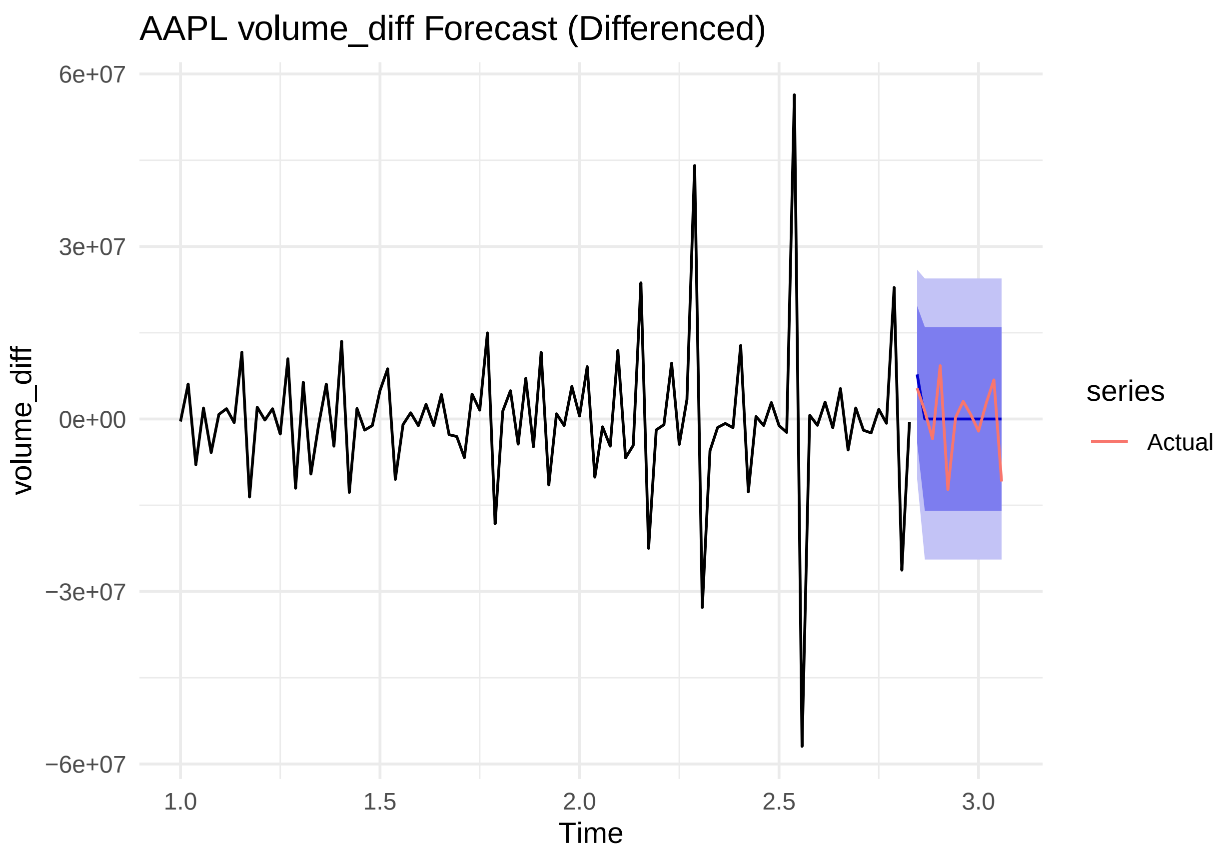
```
## # A tibble: 2 x 25
##   week        open  high   low close   volume adjusted weekly_return  ma20  ma50
##   <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>         <dbl> <dbl> <dbl>
## 1 2019-12-30  74.1  75.2  73.8  74.4   2.82e8     72.0        -0.972    NA    NA
## 2 2020-01-06  73.4  78.2  73.2  77.6   6.70e8     75.1         3.51     NA    NA
## # i 15 more variables: rsi14 <dbl>, bb_dn <dbl>, bb_mavg <dbl>, bb_up <dbl>,
## #   bb_pctB <dbl>, macd <dbl>, macdSig <dbl>, rolling_sd_20 <dbl>,
## #   sin_wday <dbl>, cos_wday <dbl>, volume_diff <dbl>, adjusted_diff <dbl>,
## #   ma20_diff <dbl>, ma50_diff <dbl>, rsi14_diff <dbl>
```

Train test split

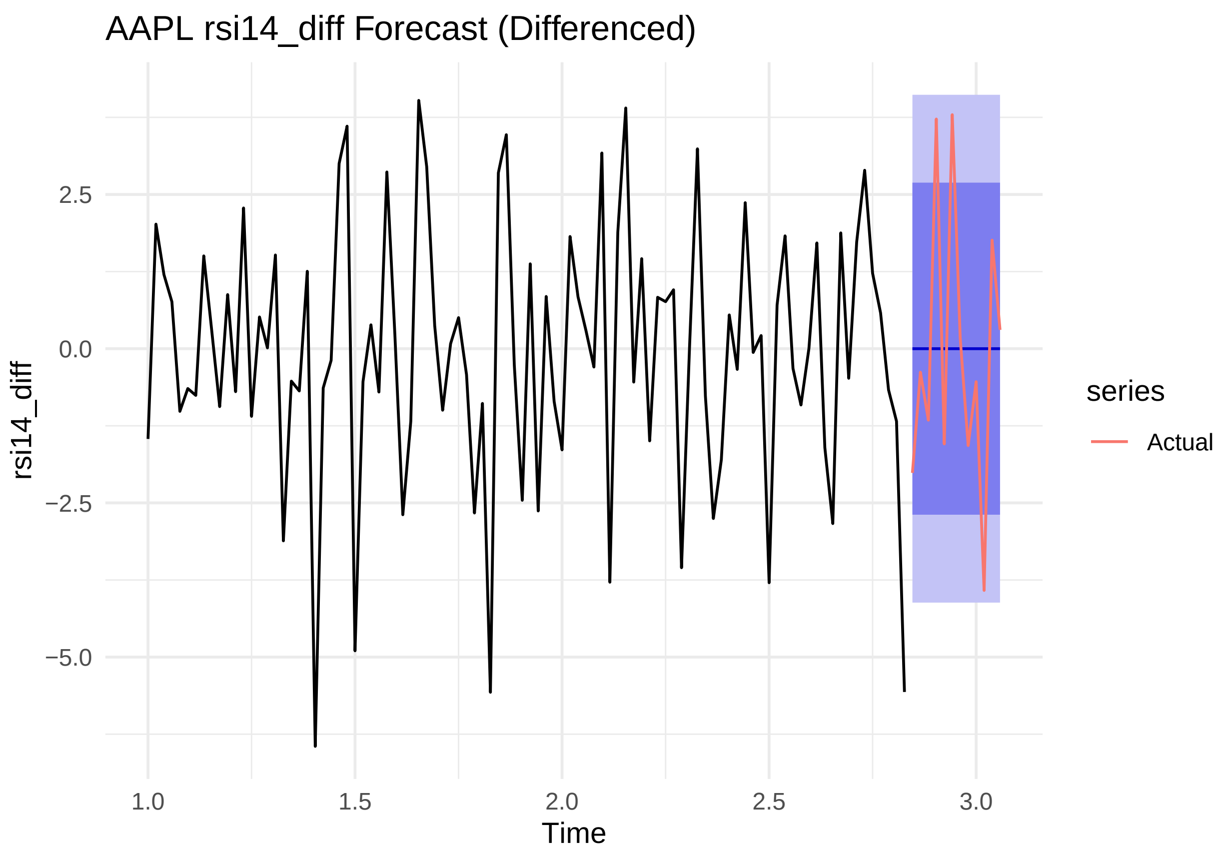
```r
cutoff_date <- as.Date("2024-12-31")
begin_date <- as.Date("2023-03-01")

train_data_week <- df_weekly %>%
  filter(week >= begin_date & week <= cutoff_date) %>%
  drop_na(adjusted_diff, volume_diff, rsi14_diff, ma20_diff, ma50_diff)

test_data_week <- df_weekly %>%
  filter(week > cutoff_date) %>%
  drop_na(adjusted_diff, volume_diff, rsi14_diff, ma20_diff, ma50_diff)
```

Forecast exogenous variables

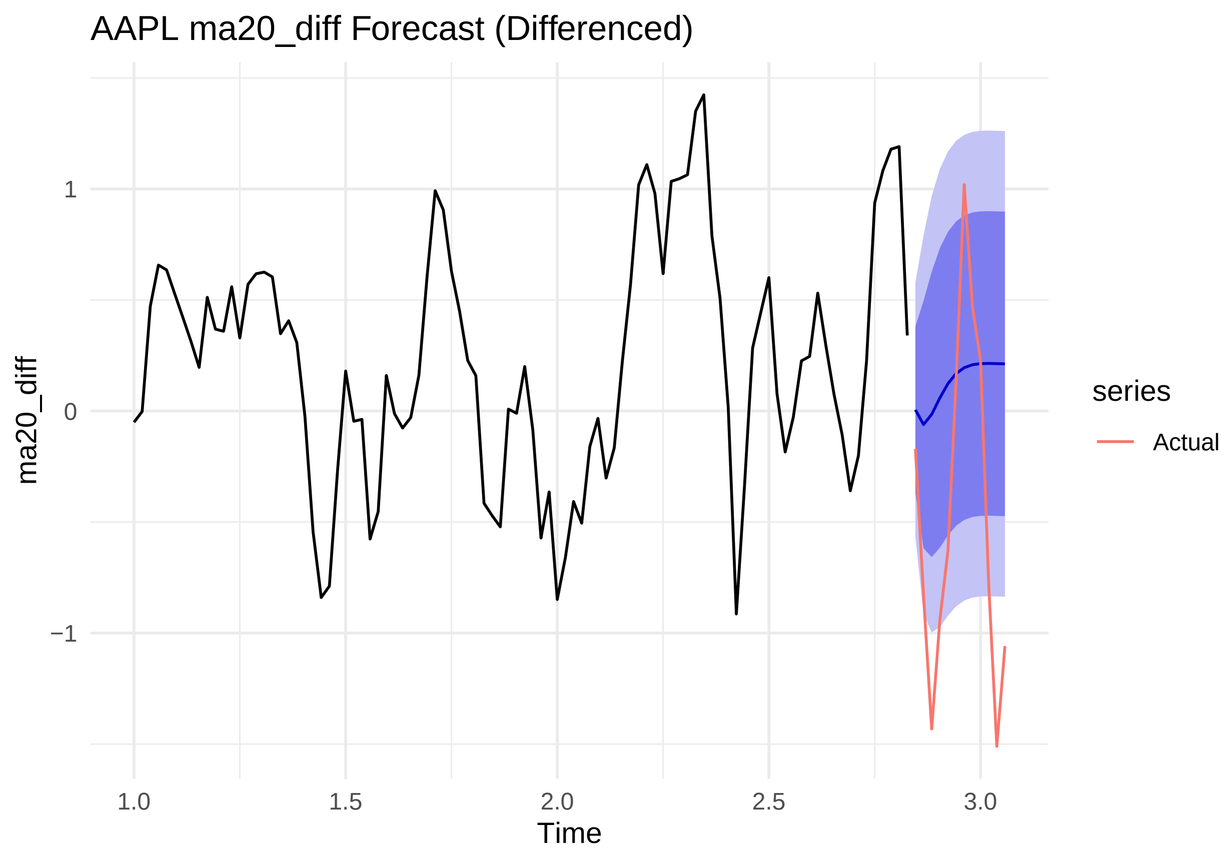
```
# Define a forecast function for a given variable (differenced)
forecast_exog <- function(train_df, test_df, var_name, freq = 52) {
  train_ts <- ts(train_df[[var_name]], frequency = freq)
  fit <- auto.arima(train_ts, stepwise = TRUE, approximation = TRUE)
  h <- nrow(test_df)
  fc <- forecast(fit, h = h)

  list(
    model_fit = fit,
    forecast_obj = fc,
    forecast = as.numeric(fc$mean),
    AIC = AIC(fit),
    actual = ts(test_df[[var_name]], frequency = freq,
                start = c(1, length(train_df[[var_name]]) + 1))
  )
}

# Define the differenced exogenous variables to forecast
exog_vars <- c("volume_diff", "rsi14_diff", "ma20_diff", "ma50_diff")
exog_perf <- tibble(variable = character(),
                    MAPE = numeric(),
                    MSE = numeric(),
                    AIC = numeric())

exog_forecasts <- list()

for (var in exog_vars) {
  fc_result <- forecast_exog(train_data_week, test_data_week, var)
  exog_forecasts[[var]] <- fc_result$forecast

  # Compute performance metrics for this variable
  actual_ts <- fc_result$actual
  mape_exog <- mean(abs(fc_result$forecast - actual_ts) / abs(actual_ts)) * 100
  mse_exog <- mean((fc_result$forecast - actual_ts)^2)

  exog_perf <- exog_perf %>%
    add_row(variable = var, MAPE = mape_exog, MSE = mse_exog, AIC = fc_result$AIC)

  # Plot forecast vs. actual for this variable
  print(
    autoplot(fc_result$forecast_obj) +
      autolayer(actual_ts, series = "Actual") +
      labs(title = paste(symbol, var, "Forecast (Differenced)"),
           x = "Time", y = var) +
      theme_minimal()
  )

  print(fc_result$model_fit)
}
```



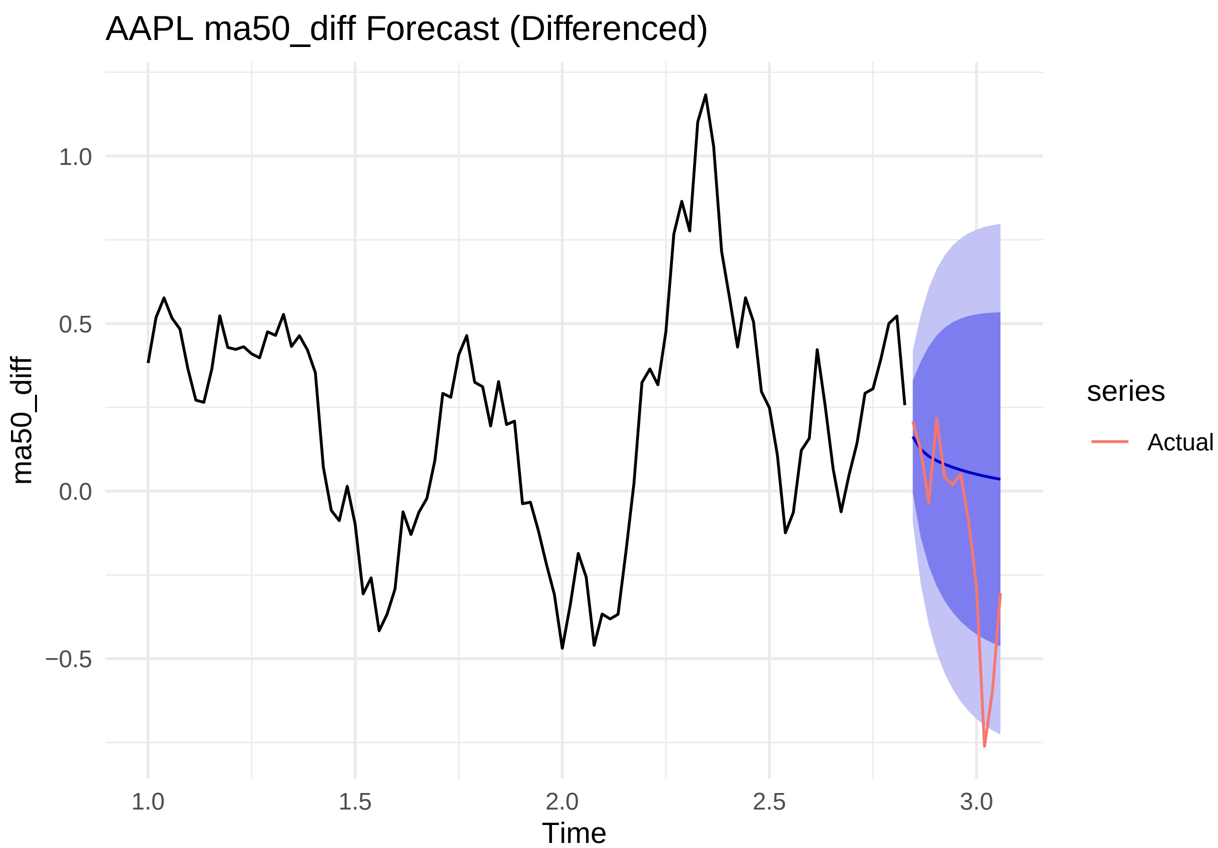

```
## Series: train_ts
## ARIMA(0,0,1) with zero mean
##
## Coefficients:
##           ma1
##       -0.8959
## s.e.   0.0429
##
## sigma^2 = 8.626e+13:  log likelihood = -1676.77
## AIC=3357.54   AICc=3357.67   BIC=3362.67
```




```
## Series: train_ts
## ARIMA(0,0,0) with zero mean
##
## sigma^2 = 4.411:  log likelihood = -207.46
## AIC=416.91   AICc=416.95   BIC=419.48
```




```
## Series: train_ts
## ARIMA(2,0,0) with non-zero mean
##
## Coefficients:
##          ar1      ar2    mean
##       1.0987  -0.3545  0.2098
## s.e.  0.0974   0.0989  0.1132
##
## sigma^2 = 0.0856:  log likelihood = -17.38
## AIC=42.75   AICc=43.19   BIC=53.01
```




```
## Series: train_ts
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##          ar1      ar2
##       1.2058  -0.2806
## s.e.  0.0992   0.1006
##
## sigma^2 = 0.01708:  log likelihood = 58.97
## AIC=-111.94   AICc=-111.67   BIC=-104.24
```

```
exog_perf
```

```
## # A tibble: 4 x 4
##   variable     MAPE      MSE   AIC
##   <chr>       <dbl>    <dbl> <dbl>
## 1 volume_diff  95.4 3.69e+13 3358.
## 2 rsi14_diff  100   4.79e+ 0  417.
## 3 ma20_diff    87.3 8.77e- 1   42.8
## 4 ma50_diff   123.  1.12e- 1 -112.
```

Build model 1

```
train_ts_diff <- ts(train_data_week$adjusted_diff, frequency = 52)
test_ts_diff  <- ts(test_data_week$adjusted_diff, frequency = 52,
                    start = c(1, length(train_data_week$adjusted_diff) + 1))
```

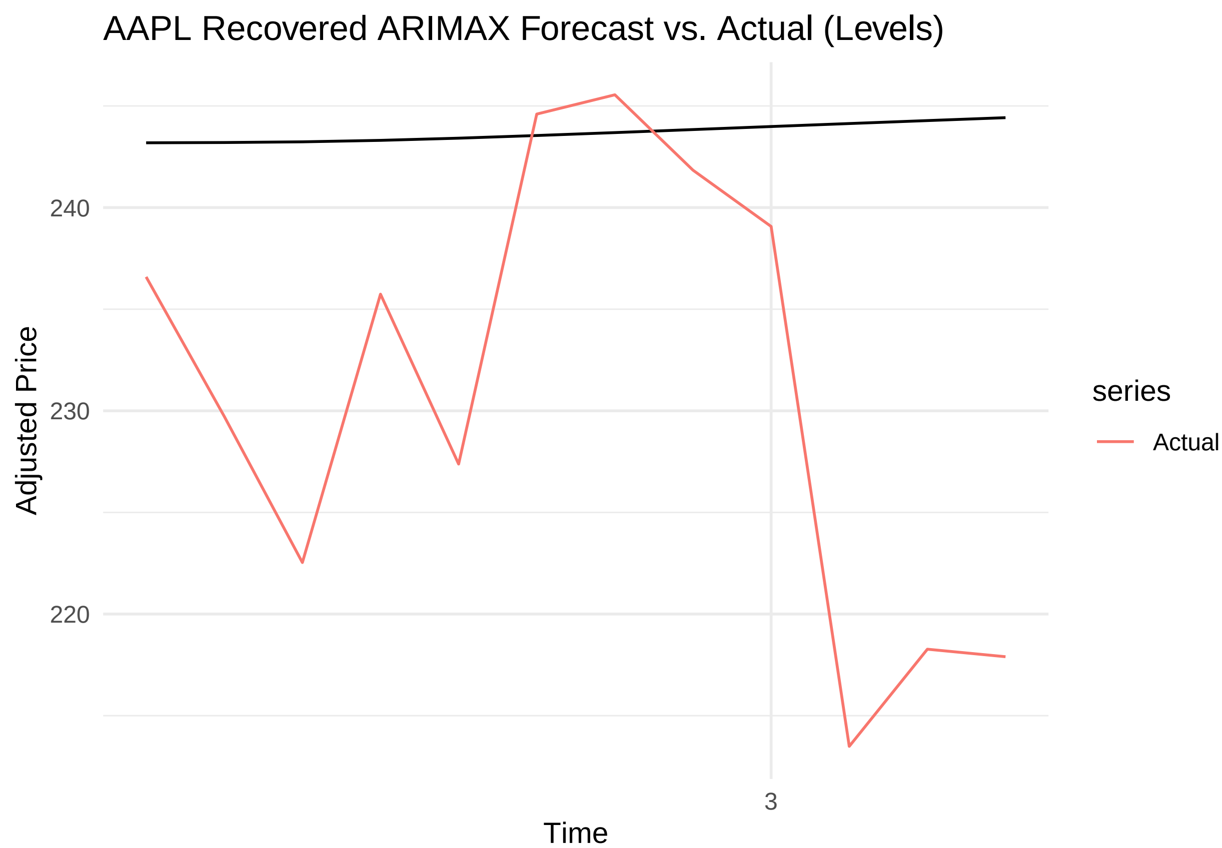
```r
xreg_train_diff <- as.matrix(train_data_week %>%
                               select(volume_diff, rsi14_diff,
                                      ma20_diff, ma50_diff))
xreg_test_diff  <- cbind(
  volume_diff = exog_forecasts[["volume_diff"]],
  rsi14_diff  = exog_forecasts[["rsi14_diff"]],
  ma20_diff   = exog_forecasts[["ma20_diff"]],
  ma50_diff   = exog_forecasts[["ma50_diff"]]
)

# Fit the ARIMAX model
model_arima_diff <- auto.arima(train_ts_diff, xreg = xreg_train_diff)
summary(model_arima_diff)
```

```
## Series: train_ts_diff
## Regression with ARIMA(0,0,0) errors
##
## Coefficients:
##       volume_diff  rsi14_diff  ma20_diff  ma50_diff
##             0e+00      0.5474     0.6037     0.4092
## s.e.        1e-04      0.0161     0.0880     0.1230
##
## sigma^2 = 0.1137:  log likelihood = -29.81
## AIC=69.62   AICc=70.28   BIC=82.44
##
## Training set error measures:
##                        ME      RMSE       MAE     MPE     MAPE      MASE
## Training set -0.003467855 0.3300725 0.2275716 77.3752 217.7349 0.1819467
##                     ACF1
## Training set -0.08182808
```

```r
# Forecast
h <- nrow(test_data_week)
final_forecast_diff <- forecast(model_arima_diff,
                                xreg = xreg_test_diff, h = h)
```

Performance

```r
# convert forecast back to the original scale
last_train_level <- last(df_weekly %>%
                           filter(week <= cutoff_date) %>%
                           pull(adjusted))
recovered_forecast_arima <- last_train_level + cumsum(final_forecast_diff$mean)
test_ts_levels <- ts(test_data_week$adjusted, frequency = 52,
                     start = c(1, length(train_data_week$adjusted) + 1))

# Plot
autoplot(ts(recovered_forecast_arima, frequency = 52,
            start = c(1, length(train_data_week$adjusted) + 1))) +
  autolayer(test_ts_levels, series = "Actual") +
  labs(title = paste(symbol, "Recovered ARIMAX Forecast vs. Actual (Levels)"),
       x = "Time", y = "Adjusted Price") +
  theme_minimal()
```




```r
# Error metrics
mape_final <- mean(abs(recovered_forecast_arima - test_ts_levels) / abs(test_ts_levels)) * 100
mse_final <- mean((recovered_forecast_arima - test_ts_levels)^2)
cat("Final Recovered ARIMAX Forecast -> MAPE:", mape_final, "\nMSE:", mse_final)
```

```
## Final Recovered ARIMAX Forecast -> MAPE: 5.894386
## MSE: 276.6519
```

```r
# Check residuals
checkresiduals(final_forecast_diff)
```


```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(0,0,0) errors
## Q* = 16.642, df = 19, p-value = 0.6141
##
## Model df: 0.   Total lags used: 19
```

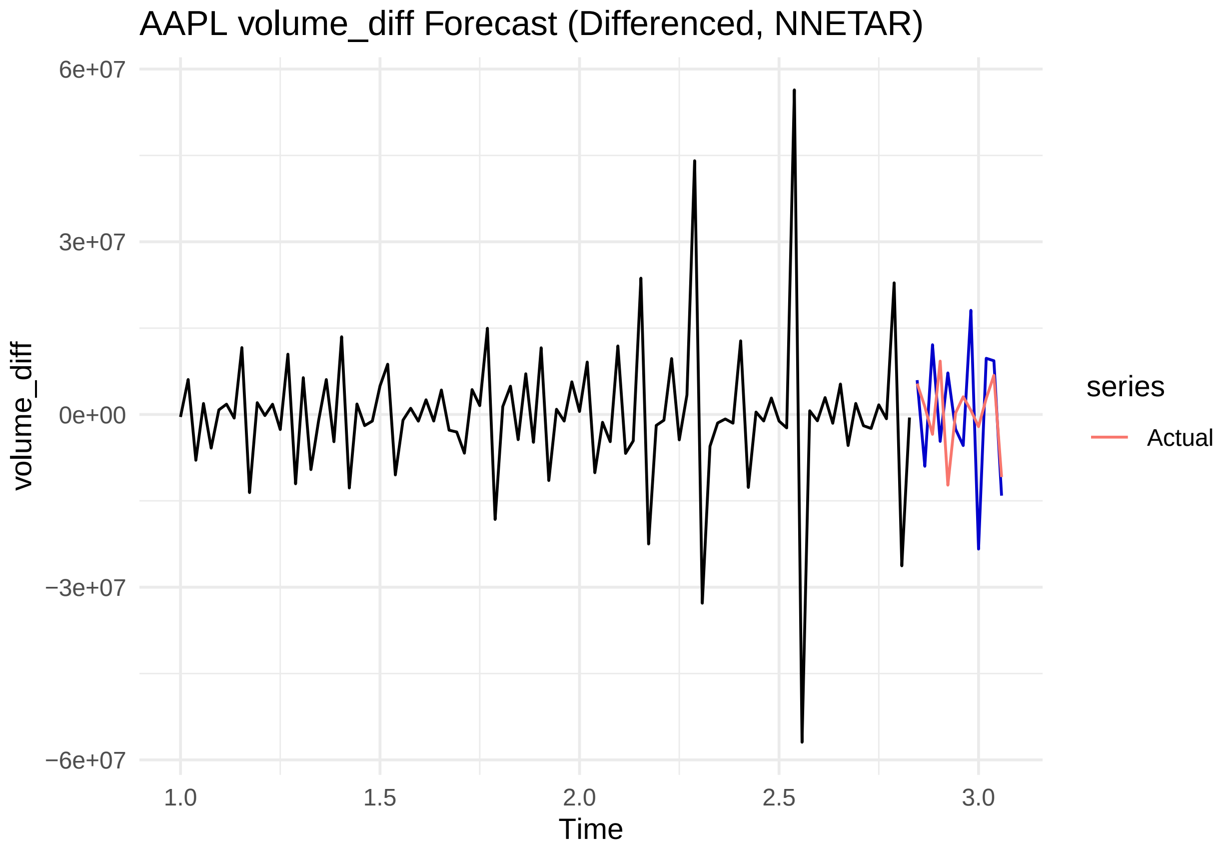
Model 2: NNETAR with Exogenous Regressors

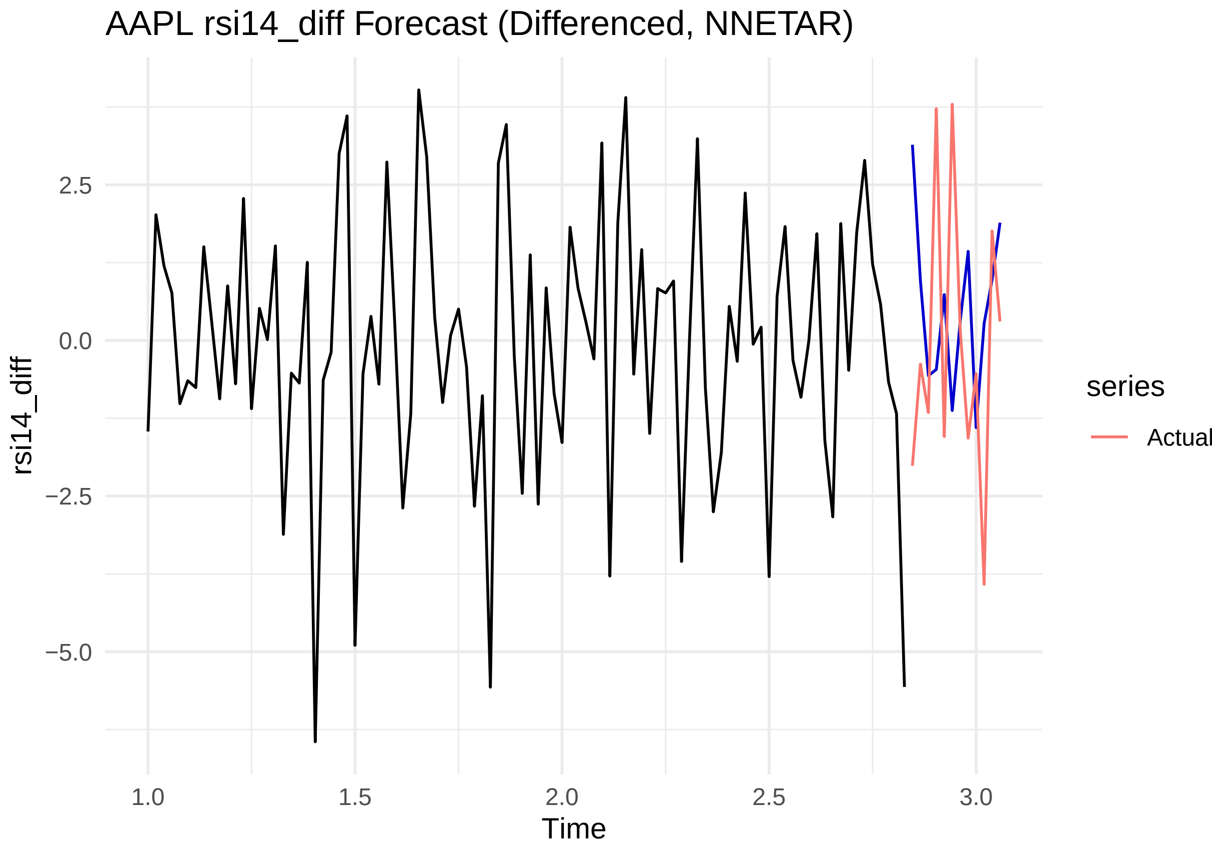
Preparation for model 2

Forecast exogenous variables

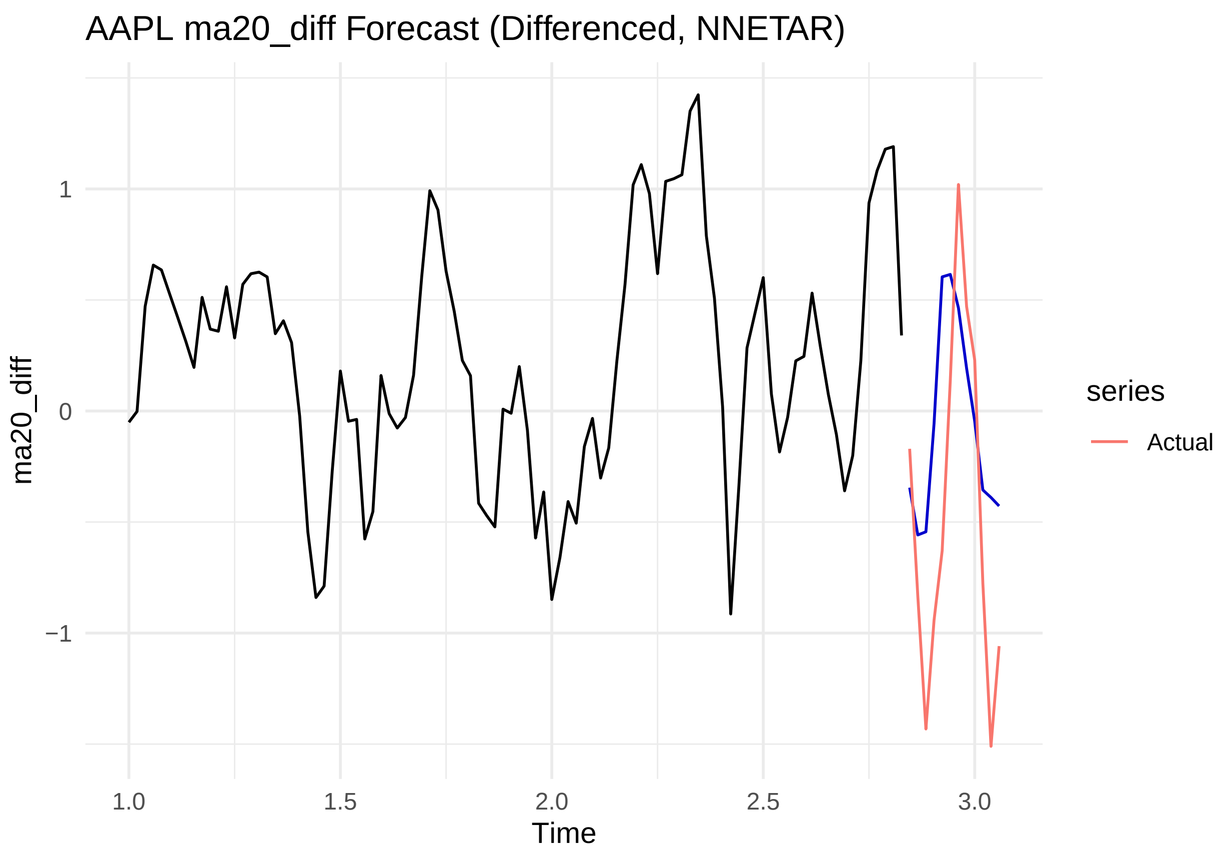
```
forecast_exog <- function(train_df, test_df, var_name, freq = 52) {
  train_ts <- ts(train_df[[var_name]], frequency = freq)
  fit <- nnetar(train_ts)  # Use nnetar for forecasting
  h <- nrow(test_df)
  fc <- forecast(fit, h = h)

  list(
    forecast_obj = fc,
    forecast = as.numeric(fc$mean),
    AIC = NA,  # nnetar() does not provide AIC in the same way
    actual = ts(test_df[[var_name]], frequency = freq,
                start = c(1, length(train_df[[var_name]]) + 1))
```

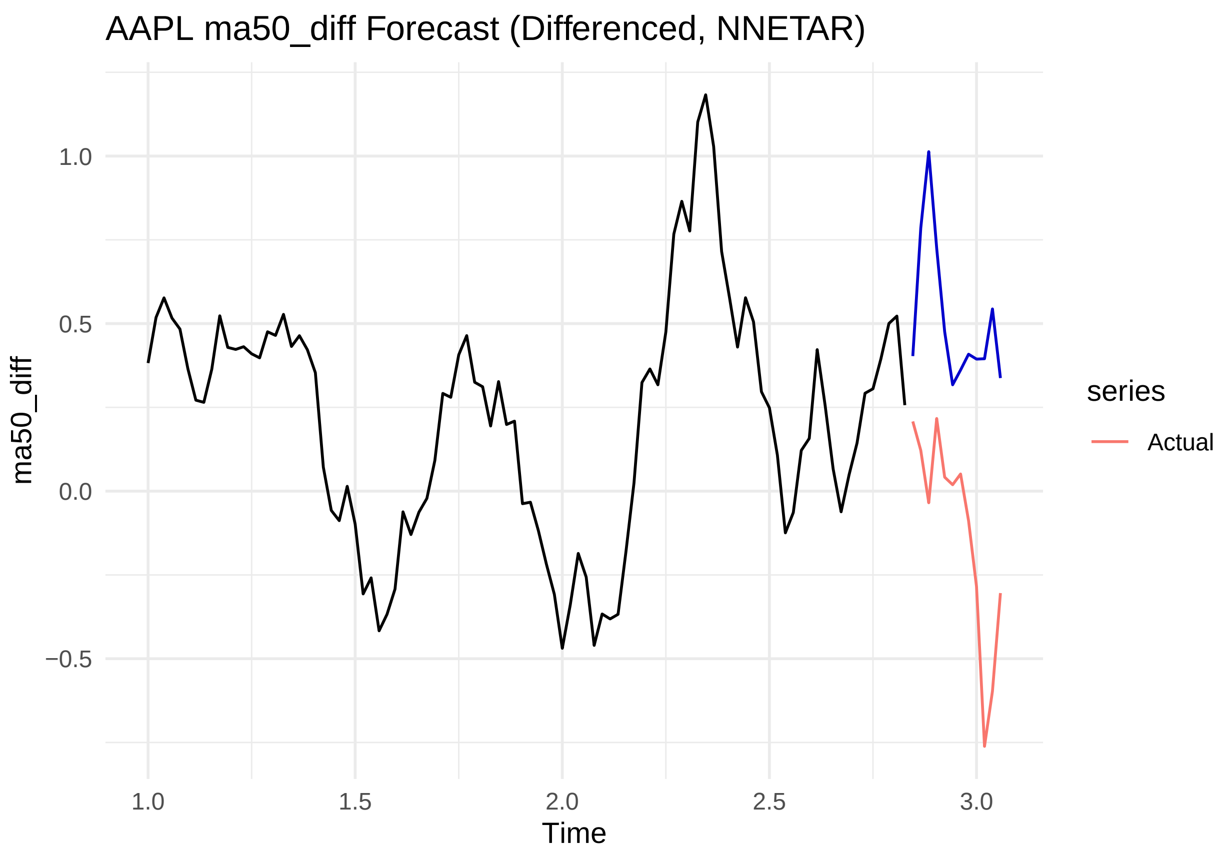
```
  )
}

# Define the differenced exogenous variables to forecast
exog_vars <- c("volume_diff", "rsi14_diff", "ma20_diff", "ma50_diff")
exog_perf <- tibble(variable = character(),
                    MAPE = numeric(),
                    MSE = numeric(),
                    AIC = numeric())
exog_forecasts <- list()
h <- nrow(train_data_week)

for (var in exog_vars) {
  fc_result <- forecast_exog(train_data_week, test_data_week, var)
  exog_forecasts[[var]] <- fc_result$forecast

  # Compute performance metrics for each exogenous variable forecast
  actual_ts <- fc_result$actual
  mape_exog <- mean(abs(fc_result$forecast - actual_ts) / abs(actual_ts)) * 100
  mse_exog <- mean((fc_result$forecast - actual_ts)^2)

  exog_perf <- exog_perf %>%
    add_row(variable = var, MAPE = mape_exog, MSE = mse_exog, AIC = fc_result$AIC)

  # Plot forecast vs. actual for this exogenous variable
  print(
    autoplot(fc_result$forecast_obj) +
      autolayer(actual_ts, series = "Actual") +
      labs(title = paste(symbol, var, "Forecast (Differenced, NNETAR)"),
           x = "Time", y = var) +
      theme_minimal()
  )
}
```


AAPL volume_diff Forecast (Differenced, NNETAR)
6e+07
3e+07
0e+00
−3e+07
−6e+07
volume_diff
1.0
1.5
2.0
2.5
3.0
Time
series
Actual



AAPL rsi14_diff Forecast (Differenced, NNETAR)
rsi14_diff
2.5
0.0
−2.5
−5.0
1.0
1.5
2.0
2.5
3.0
Time
series
Actual


AAPL ma20_diff Forecast (Differenced, NNETAR)
ma20_diff
1
0
−1
1.0
1.5
2.0
2.5
3.0
Time
series
Actual




```
exog_perf
```

```
## # A tibble: 4 x 4
##   variable     MAPE      MSE   AIC
##   <chr>       <dbl>    <dbl> <dbl>
## 1 volume_diff  558. 1.52e+14    NA
## 2 rsi14_diff   179. 8.85e+ 0    NA
## 3 ma20_diff    104. 4.77e- 1    NA
## 4 ma50_diff    704. 4.97e- 1    NA
```

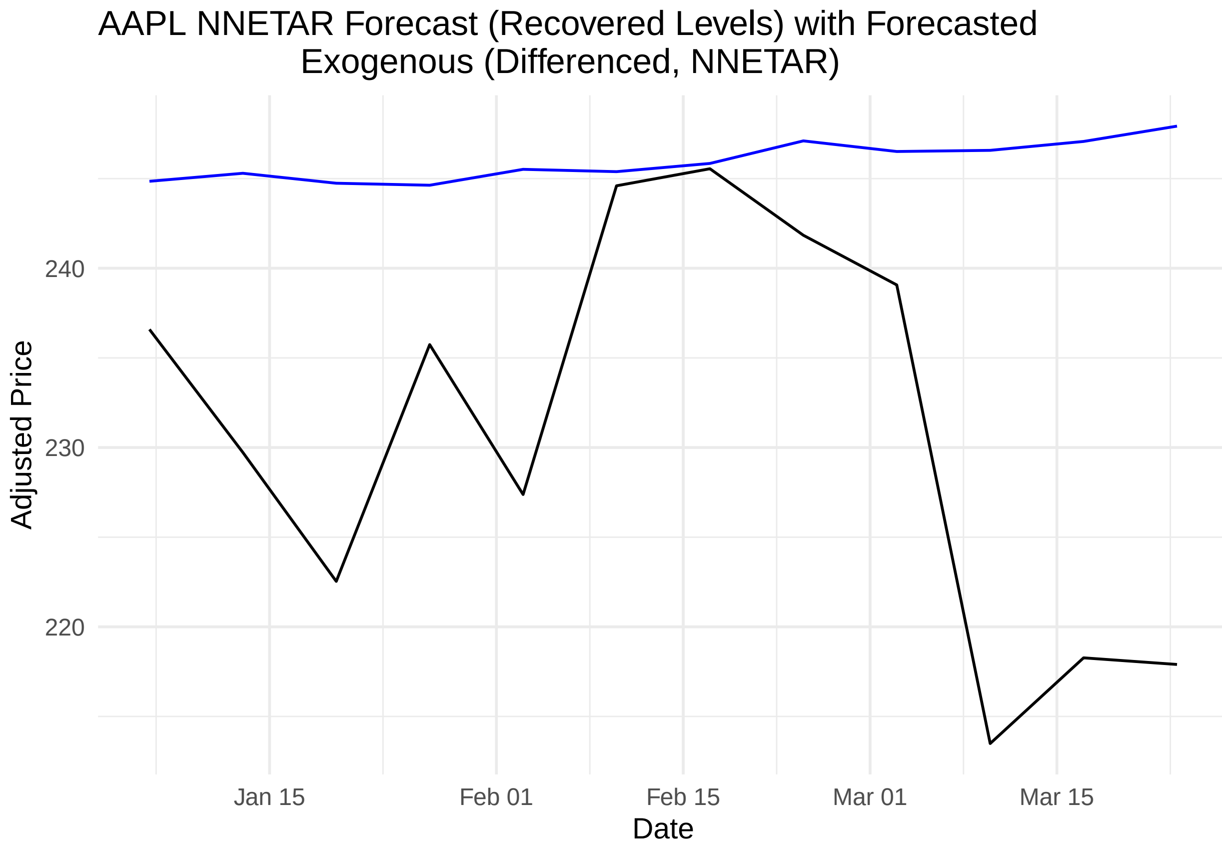
Build model 2

```
train_tsibble_diff <- train_data_week %>% as_tsibble(index = week)

fit_nnet <- train_tsibble_diff %>%
  model(
    nnet = NNETAR(adjusted_diff ~ volume_diff + rsi14_diff + ma20_diff + ma50_diff)
  )

# forecast
test_exog_data <- test_data_week %>%
  select(week) %>%
  mutate(
    volume_diff = exog_forecasts[["volume_diff"]],
    rsi14_diff  = exog_forecasts[["rsi14_diff"]],
    ma20_diff   = exog_forecasts[["ma20_diff"]],
```

```
    ma50_diff    = exog_forecasts[["ma50_diff"]]
  )
test_exog_tsibble <- as_tsibble(test_exog_data, index = week)

fc_nnet_diff <- fit_nnet %>% forecast(new_data = test_exog_tsibble)
```

Performance

```
last_train_level <- last(df_weekly %>% filter(week <= cutoff_date) %>% pull(adjusted))

# Convert the NNETAR differenced forecast to a tibble and order by week
fc_nnet_diff_df <- fc_nnet_diff %>% as_tibble() %>% arrange(week)

# Recover the level forecasts by cumulatively summing the forecasted differences onto last_train_level
recovered_forecast_nnet <- last_train_level + cumsum(fc_nnet_diff_df$.mean)

test_levels <- test_data_week$adjusted

# Build a data frame for plotting and evaluation
df_plot_nnet <- tibble(
  week = fc_nnet_diff_df$week,
  forecast = recovered_forecast_nnet,
  actual = test_levels
)

ggplot(df_plot_nnet, aes(x = week)) +
  geom_line(aes(y = actual), color = "black") +
  geom_line(aes(y = forecast), color = "blue") +
  labs(title = paste(symbol,
                     "NNETAR Forecast (Recovered Levels) with Forecasted
                     Exogenous (Differenced, NNETAR)"),
       x = "Date", y = "Adjusted Price") +
  theme_minimal()
```




```
# Compute error metrics on the original scale
mape_nnet <- mean(abs(df_plot_nnet$forecast - df_plot_nnet$actual) / abs(df_plot_nnet$actual)) * 100
mse_nnet  <- mean((df_plot_nnet$forecast - df_plot_nnet$actual)^2)
cat("NNETAR Recovered Forecast -> MAPE:", mape_nnet, "\nMSE:", mse_nnet)
```

```
## NNETAR Recovered Forecast -> MAPE: 6.683561
## MSE: 343.4616
```

Model 3: Tree-Based Ensemble

Preparation for model 3

Helper function

```
forecast_exog_xgb <- function(train_df, test_df, var_name,
                              lags = 2, freq = 52, nrounds = 50,
                              max_depth = 3, eta = 0.1) {
  require(xgboost)

  # Note: embed returns rows in reverse order; we reverse it back.
  x_train <- embed(train_df[[var_name]], lags + 1)
  x_train <- x_train[nrow(x_train):1, ]
  y_train <- x_train[, 1]
  X_train <- x_train[, -1, drop = FALSE]

  dtrain <- xgb.DMatrix(data = X_train, label = y_train)
```

```
  # Train XGBoost model for one-step forecast
  params <- list(
    objective = "reg:squarederror",
    eta = eta,
    max_depth = max_depth,
    subsample = 0.8,
    colsample_bytree = 0.8
  )

  model <- xgb.train(
    params = params,
    data = dtrain,
    nrounds = nrounds,
    verbose = 0
  )

  # Recursive forecast for h steps
  h_steps <- nrow(test_df)
  preds <- numeric(h_steps)
  # Get the last available 'lags' values from the training series (in chronological order)
  last_values <- tail(train_df[[var_name]], lags)
  for(i in 1:h_steps) {
    # For prediction, the features should be: most recent value as lag1, second most recent as lag2, et
    # Our model was trained with features: [lag1, lag2, ..., lag(lags)] where lag1 is the most recent.
    features <- matrix(rev(last_values), nrow = 1)  # reverse to get most recent first
    dtest <- xgb.DMatrix(data = features)
    pred <- predict(model, dtest)
    preds[i] <- pred
    # Update last_values: drop the oldest and append the new prediction
    last_values <- c(last_values[-1], pred)
  }

  list(
    forecast = preds,
    model_fit = model
  )
}
```

Forecast exogenous variables

```
exog_vars <- c("volume_diff", "rsi14_diff", "ma20_diff", "ma50_diff")
exog_perf <- tibble(variable = character(), MAPE = numeric(), MSE = numeric())
exog_forecasts <- list()

h <- nrow(test_data_week)

for (var in exog_vars) {
  fc_result <- forecast_exog_xgb(train_data_week, test_data_week,
                                 var, lags = 2, freq = 52,
                                 nrounds = 50, max_depth = 3, eta = 0.1)
  exog_forecasts[[var]] <- fc_result$forecast

  # Compute performance metrics on test data
```

```
  actual <- test_data_week[[var]]
  mape_exog <- mean(abs(fc_result$forecast - actual) / abs(actual)) * 100
  mse_exog  <- mean((fc_result$forecast - actual)^2)

  exog_perf <- exog_perf %>%
    add_row(variable = var, MAPE = mape_exog, MSE = mse_exog)

  cat(paste("Exogenous variable:", var, " - MAPE:", round(mape_exog,2), "%, MSE:", round(mse_exog,2), "
}
```

```
## Exogenous variable: volume_diff  - MAPE: 154.45 %, MSE: 42177886983627.3
## Exogenous variable: rsi14_diff  - MAPE: 155.59 %, MSE: 4.43
## Exogenous variable: ma20_diff  - MAPE: 140.47 %, MSE: 1.26
## Exogenous variable: ma50_diff  - MAPE: 218.45 %, MSE: 0.16
```

```
exog_perf
```

```
## # A tibble: 4 x 3
##   variable     MAPE      MSE
##   <chr>       <dbl>    <dbl>
## 1 volume_diff  154. 4.22e+13
## 2 rsi14_diff   156. 4.43e+ 0
## 3 ma20_diff    140. 1.26e+ 0
## 4 ma50_diff    218. 1.58e- 1
```

Build Model 3

```
df_tree <- df_weekly %>%
  arrange(week) %>%
  mutate(
    lag1_adjusted = lag(adjusted, 1),
    lag2_adjusted = lag(adjusted, 2)
  ) %>%
  drop_na()

# Split df_tree using the same cutoff
train_tree <- df_tree %>% filter(week <= cutoff_date)
test_tree  <- df_tree %>% filter(week > cutoff_date)

# For training, we use observed exogenous values
X_train <- as.matrix(train_tree %>% select(lag1_adjusted, lag2_adjusted, volume, rsi14, ma20, ma50))
y_train <- train_tree$adjusted

# For testing, use the observed lag features from test_tree,
# but replace the current exogenous values with forecasted ones.
X_test <- as.matrix(test_tree %>% select(lag1_adjusted, lag2_adjusted))
# Append forecasted exogenous variables (for test period) from our exog_forecasts.
# Ensure the forecasted vectors have length equal to nrow(test_tree)
X_test <- cbind(
  X_test,
  volume = exog_forecasts[["volume_diff"]],   # Note: if you prefer to use levels, you might forecast t
  rsi14  = exog_forecasts[["rsi14_diff"]],
  ma20   = exog_forecasts[["ma20_diff"]],
  ma50   = exog_forecasts[["ma50_diff"]]
```

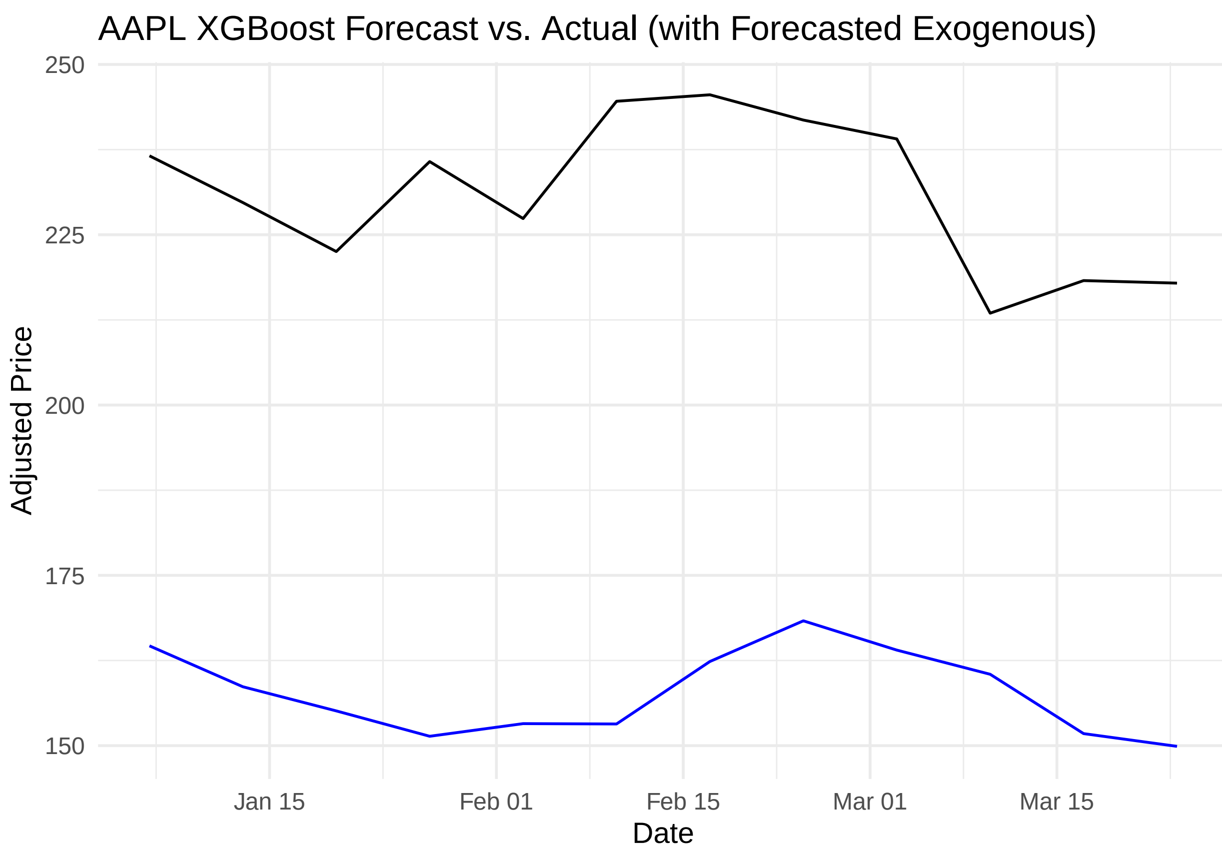
```r
)

# Create DMatrix objects for XGBoost
dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dtest  <- xgb.DMatrix(data = X_test, label = test_tree$adjusted)

# Set parameters for final XGBoost model
params <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8
)

set.seed(123)
model_xgb <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, test = dtest),
  # early_stopping_rounds = 10, # has precision issue
  verbose = 0
)

pred_xgb <- predict(model_xgb, dtest)
```

Performance

```r
mape_tree <- mean(abs(pred_xgb - test_tree$adjusted) / abs(test_tree$adjusted)) * 100
mse_tree  <- mean((pred_xgb - test_tree$adjusted)^2)
cat("Final XGBoost Model -> MAPE:", round(mape_tree,2), "%, MSE:", round(mse_tree,2), "\n")
```

```
## Final XGBoost Model -> MAPE: 31.63 %, MSE: 5463.31
```

```r
# Plot actual vs. forecasted adjusted prices
df_plot_tree <- tibble(
  week = test_tree$week,
  actual = test_tree$adjusted,
  forecast = pred_xgb
)

ggplot(df_plot_tree, aes(x = week)) +
  geom_line(aes(y = actual), color = "black") +
  geom_line(aes(y = forecast), color = "blue") +
  labs(title = paste(symbol, "XGBoost Forecast vs. Actual (with Forecasted Exogenous)"),
       x = "Date", y = "Adjusted Price") +
  theme_minimal()
```



Model 4: GARCH volatility model

Preparation for model 4

Helper function

```
# Winsorize: set outliers to quantile boundaries
winsorize <- function(x, lower = 0.02, upper = 0.98) {
  qs <- quantile(x, probs = c(lower, upper), na.rm = TRUE)
  x[x < qs[1]] <- qs[1]
  x[x > qs[2]] <- qs[2]
  x
}

# Rolling one-step ahead volatility forecast for a series.
# 'series' is the full (training+test) series.
# n_train: number of training observations.
# n_test: number of forecasts (length of test series).
rolling_vol_forecast <- function(series, n_train, n_test, spec) {
  vol_forecasts <- numeric(n_test)
  for(i in 1:n_test) {
    # Expanding window: use training plus the first (i-1) observations from test.
    current_train <- series[1:(n_train + i - 1)]
    fit_i <- ugarchfit(spec = spec, data = current_train, solver = "hybrid", silent = TRUE)
    fc_i <- ugarchforecast(fit_i, n.ahead = 1)
```

```r
    vol_forecasts[i] <- fc_i@forecast$sigmaFor
  }
  vol_forecasts
}

# Rolling forecast for an exogenous variable.
# Returns a list with one-step ahead forecasted mean and volatility.
rolling_exog_forecast <- function(series, n_train, n_test, spec) {
  forecast_means <- numeric(n_test)
  forecast_vols  <- numeric(n_test)
  for(i in 1:n_test) {
    current_train <- series[1:(n_train + i - 1)]
    fit_i <- ugarchfit(spec = spec, data = current_train, solver = "hybrid", silent = TRUE)
    fc_i <- ugarchforecast(fit_i, n.ahead = 1)
    forecast_means[i] <- fc_i@forecast$seriesFor
    forecast_vols[i]  <- fc_i@forecast$sigmaFor
  }
  list(mean = forecast_means, sigma = forecast_vols)
}
```

Sometimes GARCH fails to converge due to outliers, so we use the helper function `winsorize` to handle outliers.

Data Preparation

```r
# Prepare training and test datasets (drop NAs as before)
train_data_day <- df_stock %>%
  filter(date >= begin_date & date <= cutoff_date) %>%
  drop_na(adjusted_diff, volume_diff, rsi14_diff, ma20_diff, ma50_diff)

test_data_day <- df_stock %>%
  filter(date > cutoff_date) %>%
  drop_na(adjusted_diff, volume_diff, rsi14_diff, ma20_diff, ma50_diff)

# Create the target series (differenced returns) for training and test.
train_ts_diff <- ts(train_data_day$adjusted_diff, frequency = 252)
test_ts_diff  <- as.numeric(test_data_day$adjusted_diff)

# Combined series for rolling forecasts:
combined_ts_diff <- c(as.numeric(train_ts_diff), test_ts_diff)
n_train <- length(train_ts_diff)
n_test  <- length(test_ts_diff)
```

Rolling Forecasts for the Main Series and Exogenous Variables

```r
# Define the main GARCH specification.
spec_main <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  # For returns, use a simple AR(1) mean model.
  mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
  distribution.model = "norm"
)
```

```r
# Rolling one-step ahead volatility forecasts for the main series.
rolling_main_vols <- rolling_vol_forecast(combined_ts_diff, n_train, n_test, spec_main)

# For exogenous variables, we do similar rolling forecasts.
exog_vars <- c("volume_diff", "rsi14_diff", "ma20_diff", "ma50_diff")
spec_exog <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  mean.model = list(armaOrder = c(1, 1), include.mean = TRUE),
  distribution.model = "norm"
)

# For each exogenous variable, combine training and test series.
rolling_exog <- lapply(exog_vars, function(var) {
  series_exog <- winsorize(as.numeric(train_data_day[[var]]),
                           lower = 0.1, upper = 0.9)
  combined_exog <- c(series_exog, as.numeric(test_data_day[[var]]))
  rolling_exog_forecast(combined_exog, n_train, n_test, spec_exog)
})
names(rolling_exog) <- exog_vars
```

Build Model 4

Fit the Main ARIMAX-GARCH Model (for Returns)

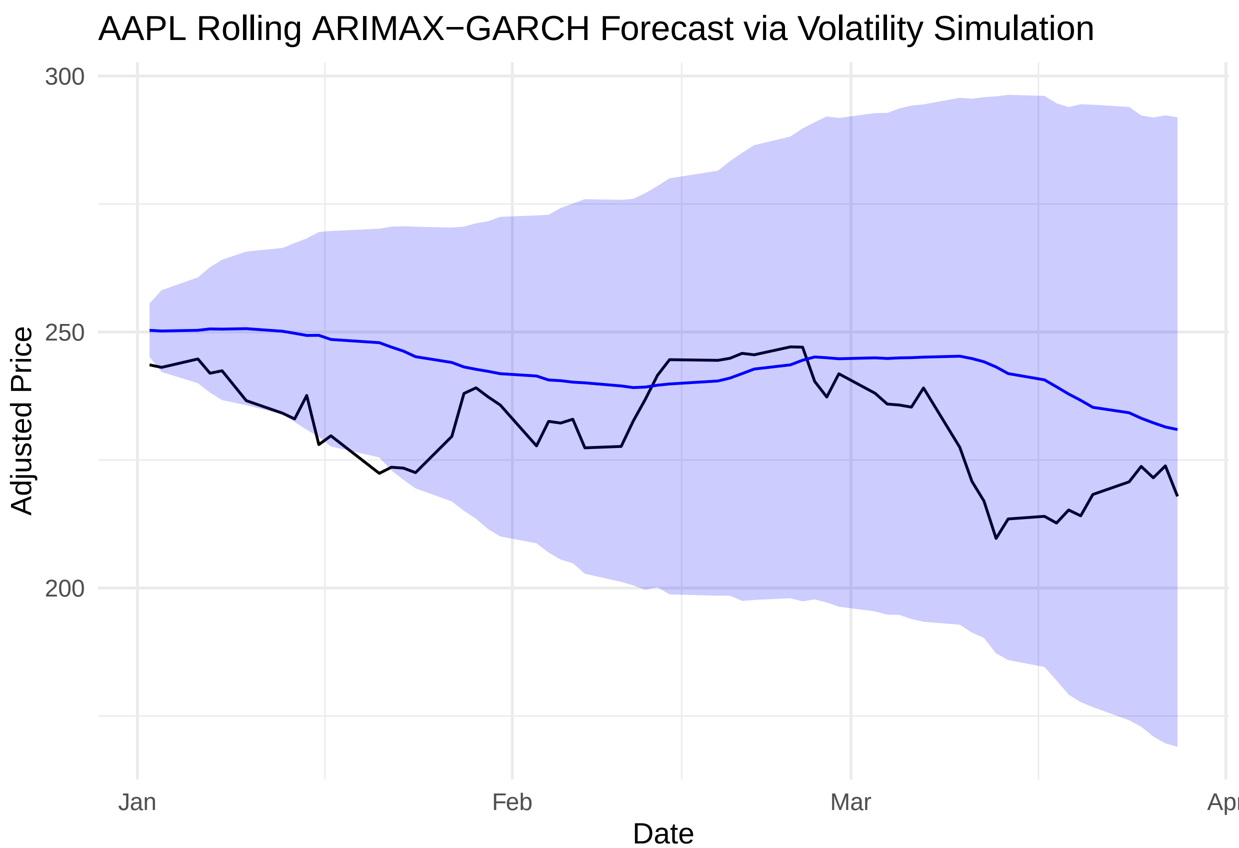
```r
# Prepare external regressors for training.
xreg_train_diff <- as.matrix(train_data_day %>% select(all_of(exog_vars)))

spec_final <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  # The external regressors affect the mean.
  mean.model = list(armaOrder = c(1, 0),
                    external.regressors = xreg_train_diff, include.mean = TRUE),
  distribution.model = "norm"
)

fit_final <- ugarchfit(spec = spec_final, data = train_ts_diff, solver = "hybrid")
coefs <- coef(fit_final)
mu    <- coefs["mu"]
ar1   <- coefs["ar1"]
mxreg <- coefs[grep("mxreg", names(coefs))]

# Last observed return and price level.
y_last <- tail(train_data_day$adjusted_diff, 1)
last_train_level <- tail(train_data_day$adjusted, 1)

# For the exogenous effect in the main model, extract rolling forecasted means.
# Create a matrix (n_test x n_exog) of exogenous forecast means.
exog_forecast_means <- sapply(rolling_exog, function(fc) fc$mean)
# Each column corresponds to one exogenous variable.
```

Nested Monte Carlo Simulation Using Rolling Forecasts

```
n_sims <- 5000

# Pre-generate shocks for the main model.
main_shocks <- matrix(rnorm(n_test * n_sims), nrow = n_test, ncol = n_sims)

# Initialize matrix for simulated returns.
main_returns <- matrix(NA, nrow = n_test, ncol = n_sims)

# Time 1 simulation:
exog_effect_1 <- apply(matrix(exog_forecast_means[1, ], nrow = 1), 1, function(x) sum(mxreg * x))
main_returns[1, ] <- mu + ar1 * y_last + exog_effect_1 +
                     rolling_main_vols[1] * main_shocks[1, ]

# Recursively simulate returns for t = 2,..., n_test.
for (t in 2:n_test) {
  exog_effect_t <- apply(matrix(exog_forecast_means[t, ], nrow = 1), 1, function(x) sum(mxreg * x))
  main_returns[t, ] <- mu + ar1 * main_returns[t - 1, ] +
                       exog_effect_t +
                       rolling_main_vols[t] * main_shocks[t, ]
}

# Recover simulated price paths by cumulatively summing returns and adding last_train_level.
price_sim_paths <- apply(main_returns, 2, cumsum)
price_sim_paths <- last_train_level + price_sim_paths
```

Performance

```
forecast_price <- rowMeans(price_sim_paths)
lower_PI <- apply(price_sim_paths, 1, quantile, probs = 0.025)
upper_PI <- apply(price_sim_paths, 1, quantile, probs = 0.975)

# Prepare plotting data.
test_ts_levels <- as.numeric(ts(test_data_day$adjusted, frequency = 252,
                                start = c(1, nrow(train_data_day) + 1)))
df_plot <- tibble(
  date = test_data_day$date,
  actual = test_ts_levels,
  forecast = forecast_price,
  lower_PI = lower_PI,
  upper_PI = upper_PI
)

ggplot(df_plot, aes(x = date)) +
  geom_line(aes(y = actual), color = "black") +
  geom_line(aes(y = forecast), color = "blue") +
  geom_ribbon(aes(ymin = lower_PI, ymax = upper_PI), fill = "blue", alpha = 0.2) +
  labs(title = paste(symbol, "Rolling ARIMAX-GARCH Forecast via Volatility Simulation"),
       x = "Date", y = "Adjusted Price") +
  theme_minimal()
```



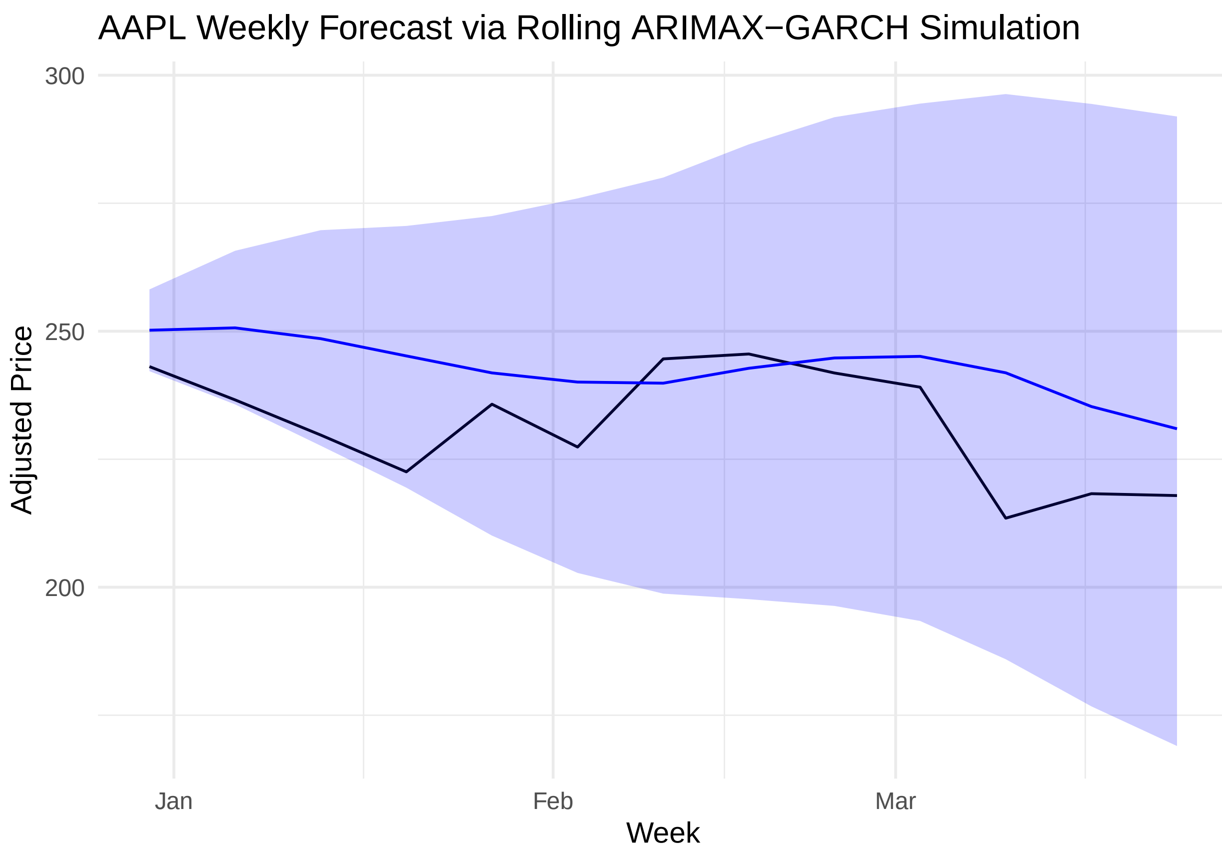

```
mape_final <- mean(abs(forecast_price - test_ts_levels) / abs(test_ts_levels)) * 100
mse_final  <- mean((forecast_price - test_ts_levels)^2)
cat("Final Rolling Forecast (with exogenous simulation) -> MAPE:", round(mape_final, 2),
    "%, MSE:", round(mse_final, 2), "\n")
```

```
## Final Rolling Forecast (with exogenous simulation) -> MAPE: 5.4 %, MSE: 214.45
```

Convert Daily to Weekly

```
df_weekly_forecast <- df_plot %>%
  mutate(week = floor_date(date, unit = "week", week_start = 1)) %>%
  group_by(week) %>%
  summarise(
    actual = last(actual),
    forecast = last(forecast),
    lower_PI = last(lower_PI),
    upper_PI = last(upper_PI)
  ) %>%
  ungroup()

# Plot the weekly forecasts
ggplot(df_weekly_forecast, aes(x = week)) +
  geom_line(aes(y = actual), color = "black") +
  geom_line(aes(y = forecast), color = "blue") +
  geom_ribbon(aes(ymin = lower_PI, ymax = upper_PI), fill = "blue", alpha = 0.2) +
  labs(title = paste(symbol, "Weekly Forecast via Rolling ARIMAX-GARCH Simulation"),
```

```
    x = "Week", y = "Adjusted Price") +
  theme_minimal()
```

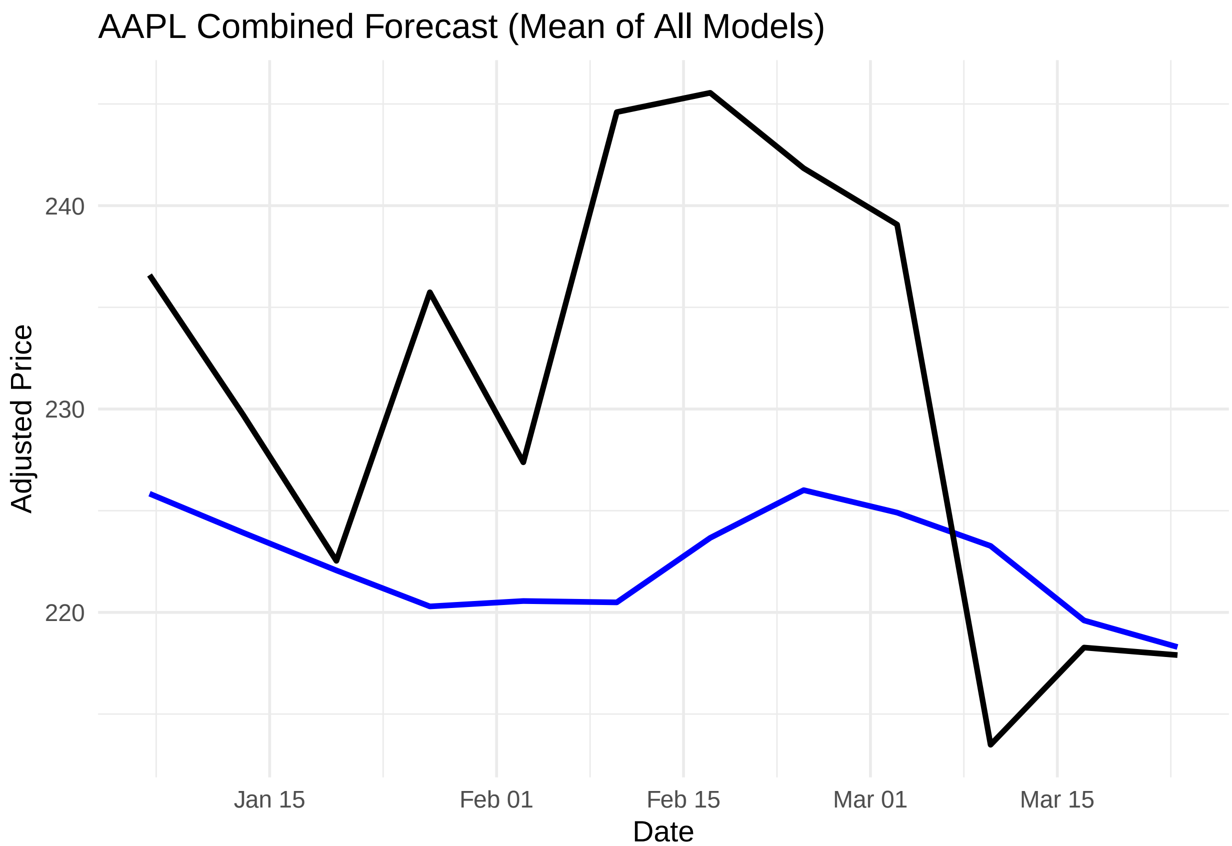


Combined Models

```
# Combine the forecasts into a tibble/data frame.
# Make sure that all forecast vectors are aligned and have the same length as test_data.
combined_forecasts <- tibble(
  week = df_weekly_forecast$week[-1],
  ARIMAX    = recovered_forecast_arima,
  NNETAR    = recovered_forecast_nnet,
  XGBoost   = pred_xgb,
  GARCH     = df_weekly_forecast$forecast[-1]
)

# Calculate the mean forecast across the models.
combined_forecasts <- combined_forecasts %>%
  mutate(Mean_Forecast = (ARIMAX + NNETAR + XGBoost + GARCH) / 4)

# Plot the combined forecast vs. the actual adjusted prices.
ggplot(combined_forecasts, aes(x = week)) +
  geom_line(aes(y = Mean_Forecast), color = "blue", size = 1) +
  geom_line(aes(y = test_data_week$adjusted), color = "black", size = 1) +
  labs(title = paste(symbol, "Combined Forecast (Mean of All Models)"),
```

```r
      x = "Date", y = "Adjusted Price") +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



performance

```r
mape_final <- mean(abs(combined_forecasts$Mean_Forecast - test_tree$adjusted) / abs(test_tree$adjusted)
mse_final  <- mean((combined_forecasts$Mean_Forecast - test_tree$adjusted)^2)
cat("Final Combined Forecast (with exogenous simulation) -> MAPE:", round(mape_final, 2),
    "%, MSE:", round(mse_final, 2), "\n")
```

```
## Final Combined Forecast (with exogenous simulation) -> MAPE: 4.45 %, MSE: 170.3
```