Time Series Modeling: Cocoa Price Forecasting

1. Setup and Package Installation

In this section, we install all the necessary R packages required for time series forecasting, machine learning models, and visualization.

```
# install.packages("imputeTS")
# install.packages("Metrics")
# install.packages("prophet")
# install.packages("xgboost")
# install.packages("rugarch")
# install.packages("ggplot2")
```

2. Load Libraries and Import Data

In this section, we load all the required libraries for data manipulation, visualization, time series modeling, and forecasting. Then, we import the cocoa futures price dataset and climate data for Ghana. This climate data might be used later for exogenous variable modeling (e.g., ARIMAX/XGBoost).

```
# Load necessary libraries
library(tidyverse)
library(lubridate)
library(readr)
library(imputeTS)
library(Metrics)
library(prophet)
library(forecast)
library(tseries)
library(xgboost)
library(dplyr)
library(ggplot2)
library(rugarch)
library(gridExtra)

# Load Cocoa Futures Price Data
cocoa <- read_csv("Daily Prices_ICCO.csv")

# Load Ghana Climate Data
ghana <- read_csv("Ghana_data.csv")

# Preview both datasets
glimpse(cocoa)
```

```
## Rows: 7,812
## Columns: 2
## $ Date                            <chr> "27/02/2025", "26/02/2025", "25/02/2025~
## $ `ICCO daily price (US$/tonne)` <dbl> 9099.66, 9089.95, 8668.57, 8408.72, 910~
```

```
glimpse(ghana)
```

```
## Rows: 53,231
## Columns: 7
## $ STATION <chr> "GHM00065472", "GHM00065472", "GHM00065472", "GHM00065472", "G~
## $ NAME    <chr> "KOTOKA INTERNATIONAL, GH", "KOTOKA INTERNATIONAL, GH", "KOTOK~
## $ DATE    <date> 1990-01-01, 1990-01-02, 1990-01-03, 1990-01-04, 1990-01-05, 1~
## $ PRCP    <dbl> NA, NA, NA, NA, 0.00, NA, NA, NA, NA, 0.00, NA, NA, NA, NA, NA~
## $ TAVG    <dbl> 80, 81, 82, 81, 79, 83, 82, 82, 79, 82, 83, 81, 81, 83, 80, 81~
## $ TMAX    <dbl> 91, 92, NA, NA, 89, 90, NA, NA, NA, NA, NA, NA, NA, NA, 91, NA~
## $ TMIN    <dbl> 76, NA, NA, 75, 75, NA, 78, 75, 75, NA, 76, 75, NA, 76, NA, 78~
```

```
# Display first few rows
head(cocoa)
```

```
## # A tibble: 6 x 2
##   Date       `ICCO daily price (US$/tonne)`
##   <chr>                               <dbl>
## 1 27/02/2025                          9100.
## 2 26/02/2025                          9090.
## 3 25/02/2025                          8669.
## 4 24/02/2025                          8409.
## 5 21/02/2025                          9106.
## 6 20/02/2025                          9962.
```

```
head(ghana)
```

```
## # A tibble: 6 x 7
##   STATION     NAME                     DATE        PRCP  TAVG  TMAX  TMIN
##   <chr>       <chr>                    <date>     <dbl> <dbl> <dbl> <dbl>
## 1 GHM00065472 KOTOKA INTERNATIONAL, GH 1990-01-01    NA    80    91    76
## 2 GHM00065472 KOTOKA INTERNATIONAL, GH 1990-01-02    NA    81    92    NA
## 3 GHM00065472 KOTOKA INTERNATIONAL, GH 1990-01-03    NA    82    NA    NA
## 4 GHM00065472 KOTOKA INTERNATIONAL, GH 1990-01-04    NA    81    NA    75
## 5 GHM00065472 KOTOKA INTERNATIONAL, GH 1990-01-05     0    79    89    75
## 6 GHM00065472 KOTOKA INTERNATIONAL, GH 1990-01-06    NA    83    90    NA
```

3. Cocoa Price Data Cleaning and Preprocessing

In this section, we prepare the cocoa price dataset for modeling. We rename columns for clarity, convert the `Date` column to proper date format, sort the dataset chronologically, and handle any missing values using linear interpolation.

```r
# Data cleaning for cocoa price dataset
# Rename the columns for convenience
cocoa <- cocoa %>%
  rename(
    Date = `Date`,
    Price = `ICCO daily price (US$/tonne)`
  )

# Convert Date from character to Date object
cocoa$Date <- dmy(cocoa$Date)

# Check for NAs and sort
sum(is.na(cocoa$Price))
```

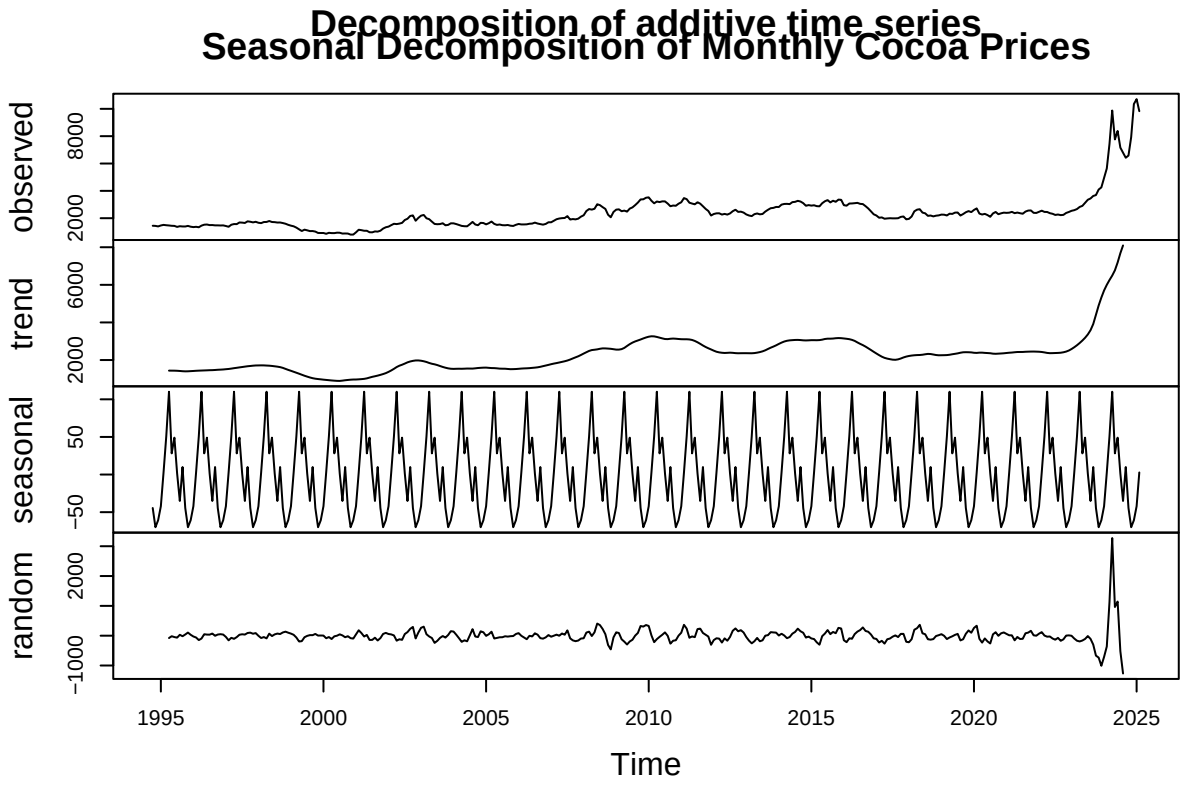
```
## [1] 0
```

```r
cocoa <- cocoa %>% arrange(Date)

# Handing missing values by interpolating to replace the NA with guessed ones
cocoa$Price <- na_interpolation(cocoa$Price)
```

4. Monthly Aggregation and Time Series Decomposition

We now convert the daily cocoa price data into a monthly average time series, which helps in smoothing out daily fluctuations and revealing long-term trends. We then decompose the resulting time series into trend, seasonal, and random (residual) components using classical decomposition.

```r
# Convert daily prices to monthly averages
cocoa_monthly <- cocoa %>%
  filter(!is.na(Date), !is.na(Price)) %>%
  mutate(YearMonth = floor_date(Date, "month")) %>%
  group_by(YearMonth) %>%
  summarise(AvgPrice = mean(Price, na.rm = TRUE)) %>%
  ungroup()

# Convert to time series object
cocoa_ts <- ts(cocoa_monthly$AvgPrice,
               start = c(year(min(cocoa_monthly$YearMonth)),
                         month(min(cocoa_monthly$YearMonth))),
               frequency = 12)

# Decompose time series
cocoa_decomp <- decompose(cocoa_ts)

# Plot decomposition
plot(cocoa_decomp)
title(main = "Seasonal Decomposition of Monthly Cocoa Prices")
```



5. Visual Exploration of Ghana Climate Data

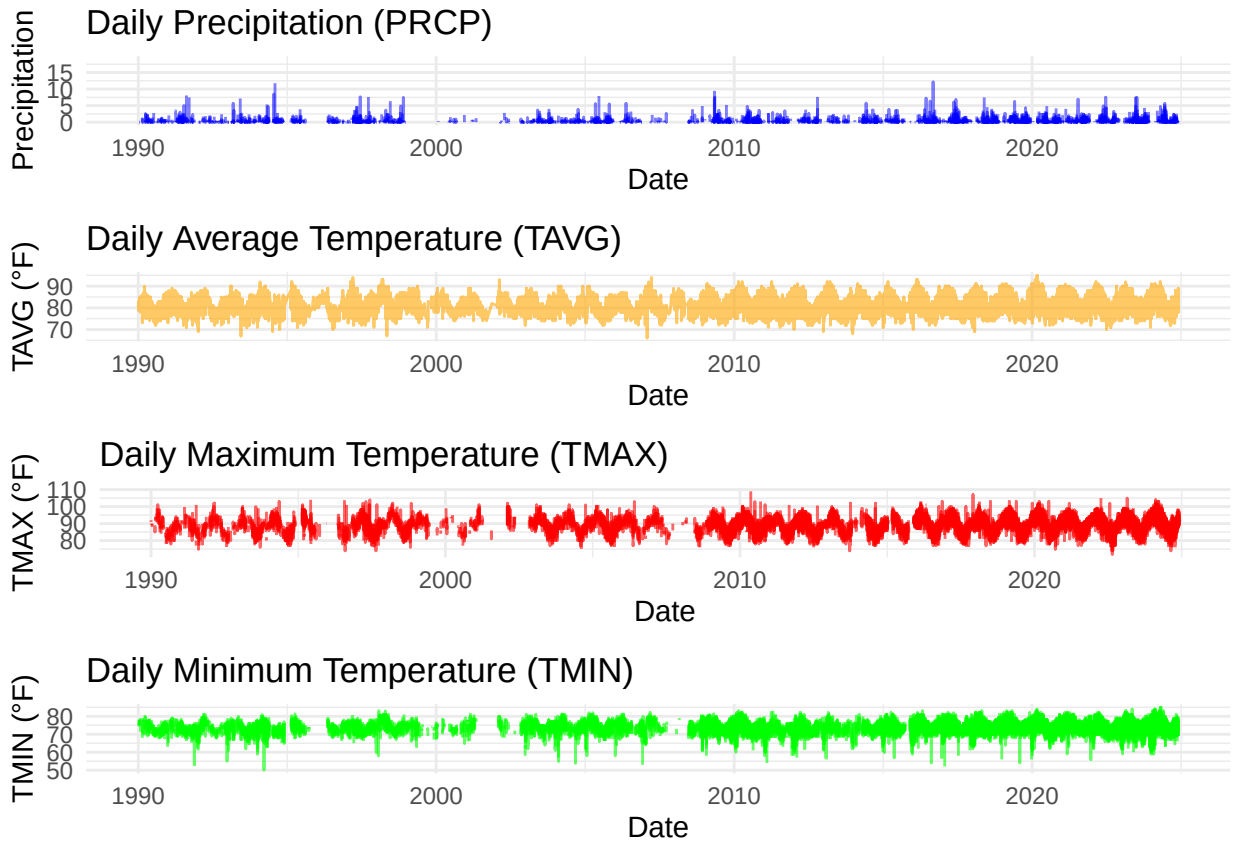
We now visualize various daily climate metrics from the Ghana dataset. This will help us understand seasonal/weather patterns and assess whether variables like precipitation and temperature may have predictive power for cocoa prices.

The following plots show: - Daily precipitation (PRCP) - Daily average temperature (TAVG) - Daily maximum temperature (TMAX) - Daily minimum temperature (TMIN)

```
# Daily Precipitation
p1 <- ggplot(ghana, aes(x = DATE, y = PRCP)) +
  geom_line(color = "blue", alpha = 0.5) +
  labs(title = "Daily Precipitation (PRCP)", x = "Date", y = "Precipitation") +
  theme_minimal()

# Daily Average Temperature
p2 <- ggplot(ghana, aes(x = DATE, y = TAVG)) +
  geom_line(color = "orange", alpha = 0.6) +
  labs(title = "Daily Average Temperature (TAVG)", x = "Date", y = "TAVG (°F)") +
  theme_minimal()

# Daily Maximum Temperature
p3 <- ggplot(ghana, aes(x = DATE, y = TMAX)) +
  geom_line(color = "red", alpha = 0.6) +
  labs(title = "Daily Maximum Temperature (TMAX)", x = "Date", y = "TMAX (°F)") +
```

```
  theme_minimal()

# Daily Minimum Temperature
p4 <- ggplot(ghana, aes(x = DATE, y = TMIN)) +
  geom_line(color = "green", alpha = 0.6) +
  labs(title = "Daily Minimum Temperature (TMIN)", x = "Date", y = "TMIN (°F)") +
  theme_minimal()

# Arrange plots vertically
grid.arrange(p1, p2, p3, p4, ncol = 1)
```

```
## Warning: Removed 7 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

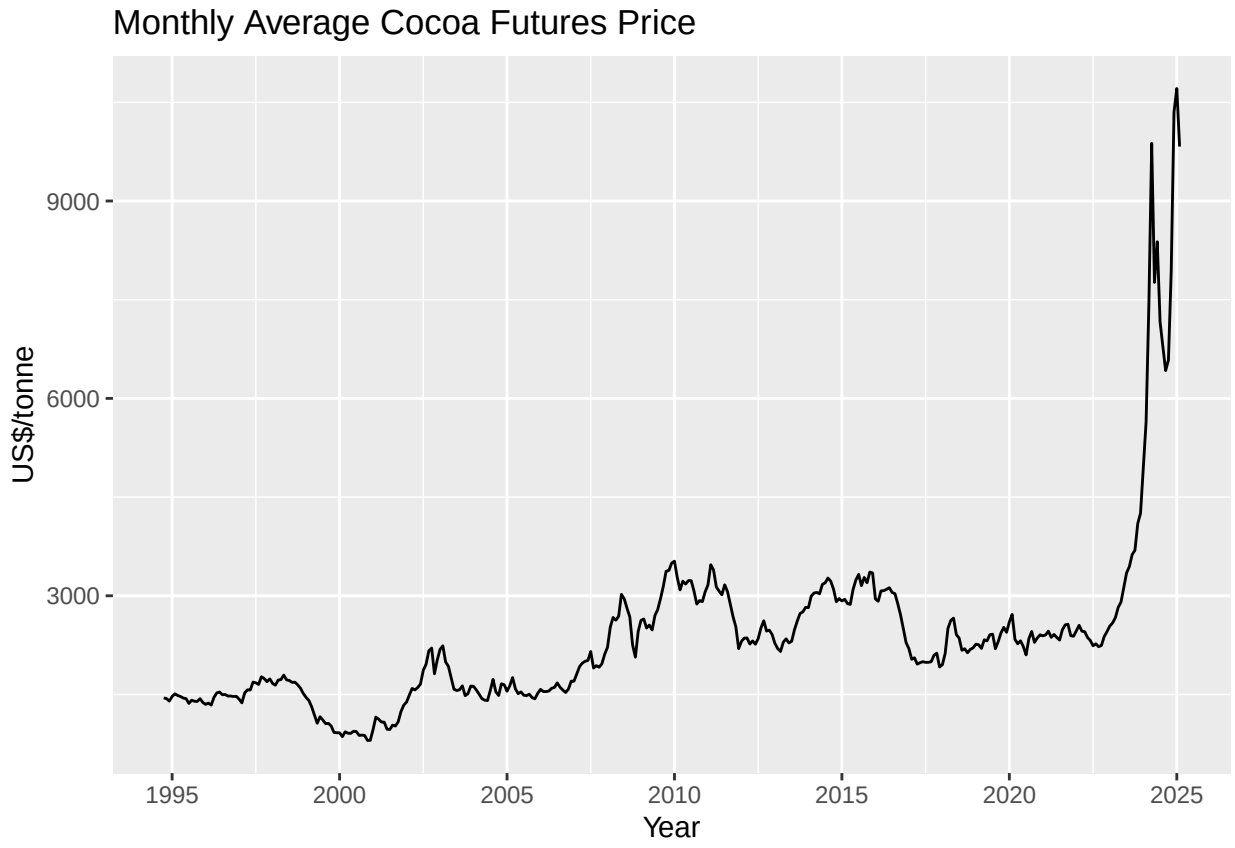


6. Monthly Aggregation and Time Series Plot of Cocoa Prices

Here, we aggregate daily cocoa prices into monthly averages and convert the result into a time series object. This allows for more stable analysis and modeling. Finally, we create a plot of the monthly cocoa price trend to visualize long-term patterns and identify any visible trends or seasonality.

```
# Aggregate daily to monthly average
cocoa_monthly <- cocoa %>%
  mutate(YearMonth = floor_date(Date, "month")) %>%
  group_by(YearMonth) %>%
  summarise(MeanPrice = mean(Price, na.rm = TRUE)) %>%
  ungroup()
```

```r
# Convert aggregated data to a time series object
ts_cocoa <- ts(
  cocoa_monthly$MeanPrice,
  start = c(year(min(cocoa_monthly$YearMonth)),
            month(min(cocoa_monthly$YearMonth))),
  frequency = 12
)

# Plot cocoa price series
autoplot(ts_cocoa) +
  ggtitle("Monthly Average Cocoa Futures Price") +
  xlab("Year") + ylab("US$/tonne")
```



7. Data Cleaning and Monthly Aggregation of Ghana Climate Data

In this section, we clean and prepare the Ghana climate data for modeling. We: - Ensure the `DATE` column is in proper date format - Filter for a specific weather station (`KOTOKA INTERNATIONAL, GH`) - Aggregate the daily data into monthly summaries, computing: - Total precipitation (`PRCP`) - Average temperatures (`TAVG`, `TMAX`, `TMIN`)

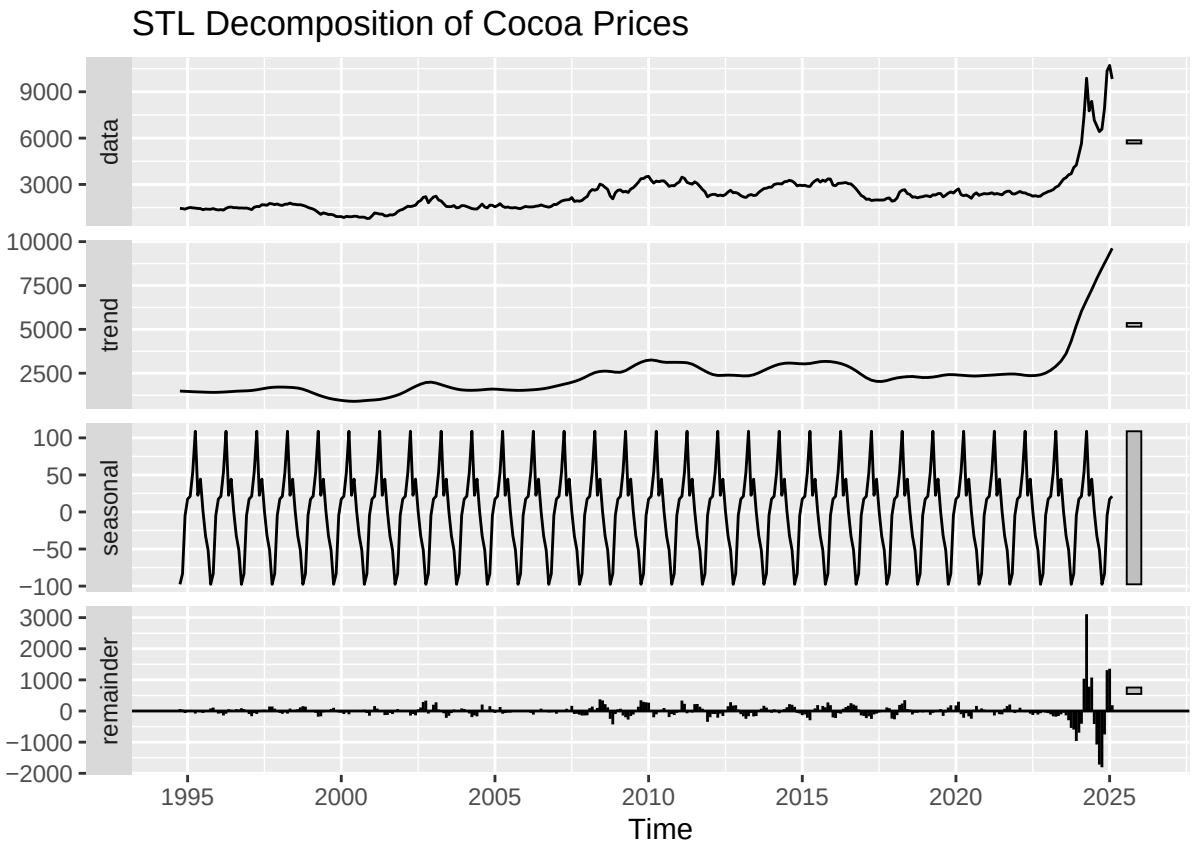
```r
# Convert DATE column
ghana <- ghana %>%
  mutate(DATE = as.Date(DATE))

# Filter for consistent weather station
ghana <- ghana %>% filter(NAME == "KOTOKA INTERNATIONAL, GH")
```

```r
# Taking monthly average of the data by first using floor_date to transform
# every date in the same month into the first date (1st) of that month
# to prepare for data aggregation (herding) to monthly averages
# PRCP, TAVG, TMAX, TMIN is the average monthly value over daily recorded ones
ghana_monthly <- ghana %>%
  mutate(YearMonth = floor_date(DATE, "month")) %>%
  group_by(YearMonth) %>%
  summarise(
    PRCP = sum(PRCP, na.rm = TRUE),
    TAVG = mean(TAVG, na.rm = TRUE),
    TMAX = mean(TMAX, na.rm = TRUE),
    TMIN = mean(TMIN, na.rm = TRUE)
  ) %>%
  ungroup()
```

8. STL Decomposition and Stationarity Testing

Before fitting models like ARIMA, it's important to: - Visualize decomposition to separate trend, seasonality, and residuals - Test for stationarity using the Augmented Dickey-Fuller (ADF) test - Difference the series if it's non-stationary (to stabilize mean/variance)

```r
# STL Decomposition
decomp <- stl(ts_cocoa, s.window = "periodic")
autoplot(decomp) + ggtitle("STL Decomposition of Cocoa Prices")
```

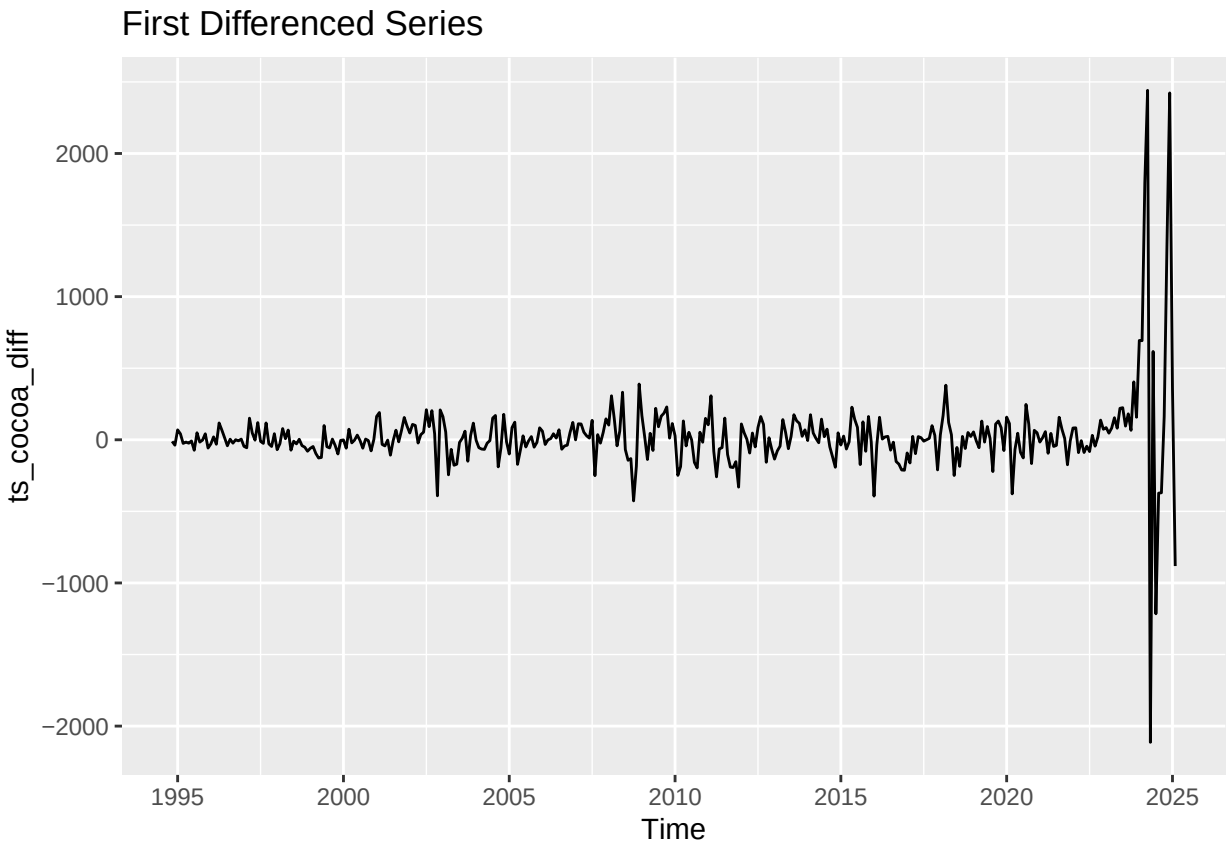



```
# Stationarity Test
adf.test(ts_cocoa)
```

```
## Warning in adf.test(ts_cocoa): p-value greater than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_cocoa
## Dickey-Fuller = 1.7118, Lag order = 7, p-value = 0.99
## alternative hypothesis: stationary
```

```
# If not stationary, difference the series
ts_cocoa_diff <- diff(ts_cocoa)
autoplot(ts_cocoa_diff) + ggtitle("First Differenced Series")
```




```
# Check stationarity again
adf.test(ts_cocoa_diff)
```

```
## Warning in adf.test(ts_cocoa_diff): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_cocoa_diff
## Dickey-Fuller = -5.9476, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

9. ARIMA Modeling and Forecasting

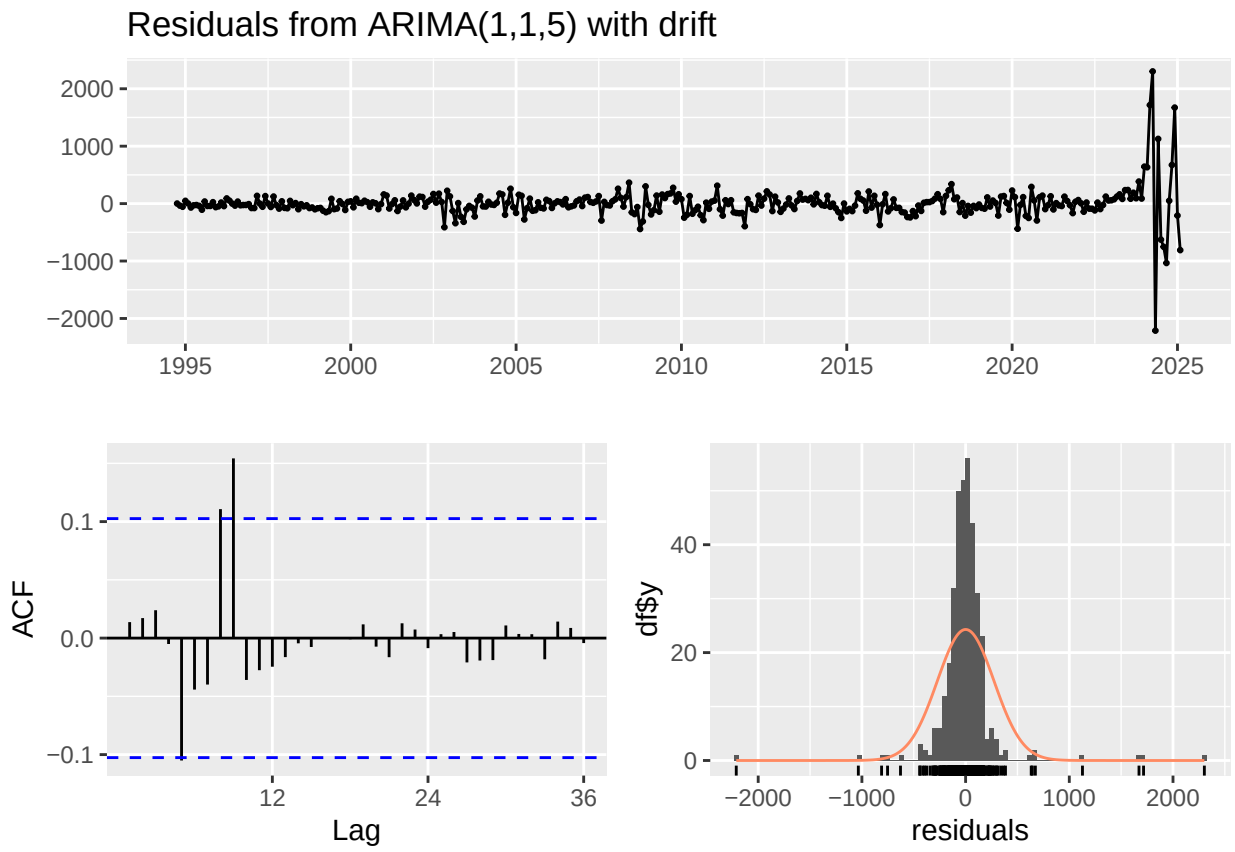
In this section, we build an **ARIMA model** using `auto.arima()` to automatically select the best parameters based on AICc. We then: - Check residuals to validate the model - Forecast cocoa prices for the next 12 months - Visualize the forecast with confidence intervals

```
# Fit ARIMA model using auto.arima
fit_arima <- auto.arima(ts_cocoa)
summary(fit_arima)
```

```
## Series: ts_cocoa
```

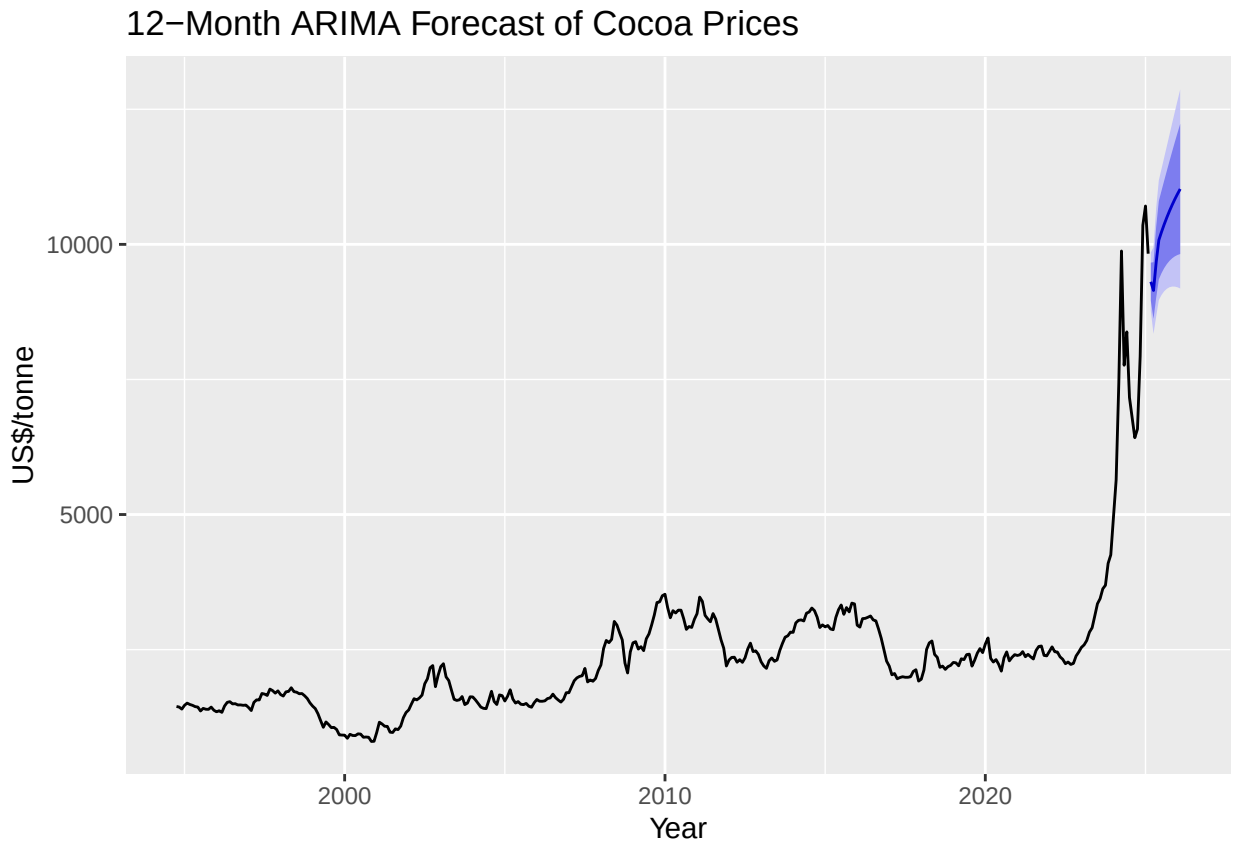
```
## ARIMA(1,1,5) with drift
##
## Coefficients:
##          ar1      ma1      ma2      ma3     ma4     ma5    drift
##       0.9107  -0.8179  -0.0680  -0.2482  0.0086  0.2789  26.4292
## s.e.  0.0563   0.0717   0.0652   0.0679  0.0619  0.0538  24.4563
##
## sigma^2 = 76217:  log likelihood = -2559.29
## AIC=5134.58   AICc=5134.99   BIC=5165.76
##
## Training set error measures:
##                     ME     RMSE      MAE        MPE     MAPE     MASE
## Training set 0.6034782 273.0325 135.9423 -0.5188187 5.150627 0.270636
##                    ACF1
## Training set 0.01372552
```

```
# Check residuals for randomness (white noise)
checkresiduals(fit_arima)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,5) with drift
## Q* = 20.81, df = 18, p-value = 0.2891
##
## Model df: 6.   Total lags used: 24
```

```r
# Forecast cocoa prices for the next 12 months
forecast_horizon <- 12
fc_arima <- forecast(fit_arima, h = forecast_horizon)

# Plot ARIMA forecast
autoplot(fc_arima) +
  ggtitle("12-Month ARIMA Forecast of Cocoa Prices") +
  ylab("US$/tonne") + xlab("Year")
```



10. ARIMA Model Evaluation: Train/Test Split

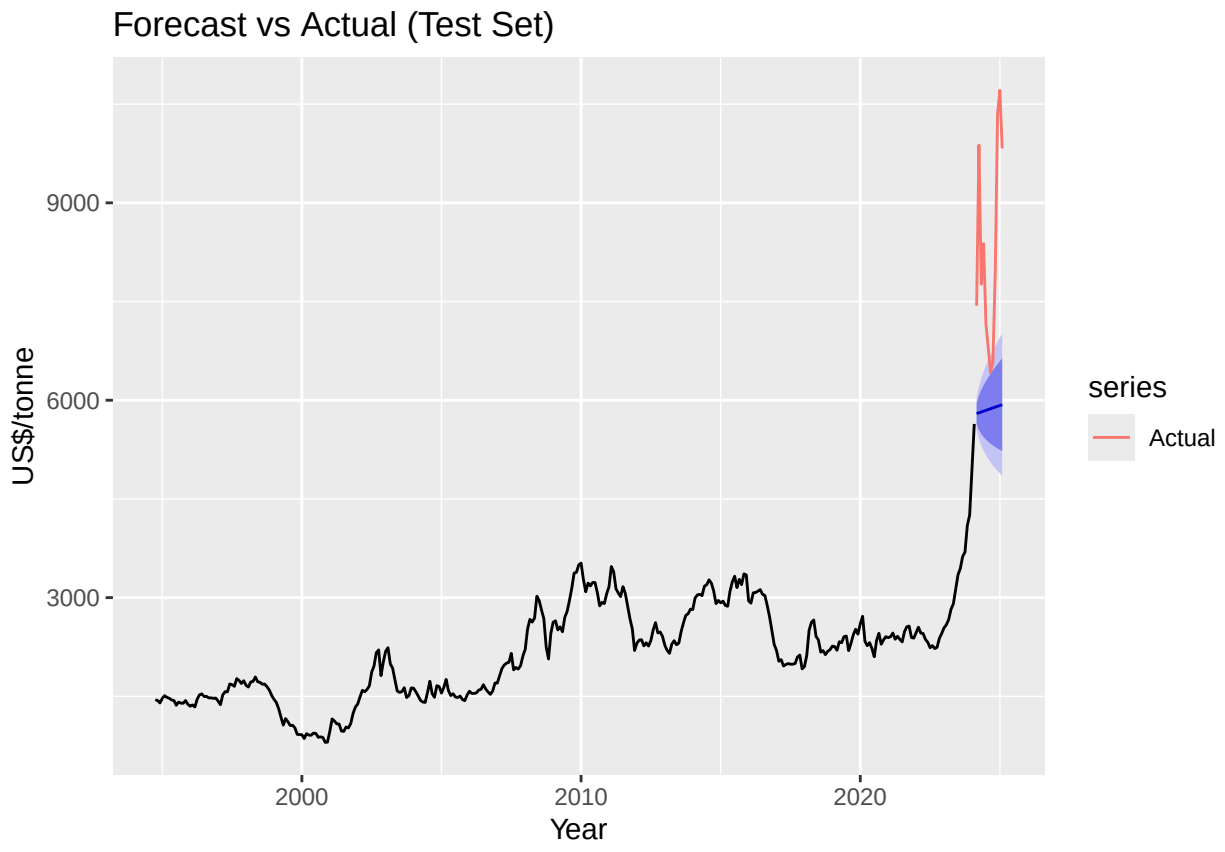
To evaluate the predictive performance of our ARIMA model, we: - Create a train/test split, holding out the last 12 months as test data - Fit an ARIMA model on the training portion - Forecast over the test horizon - Compare the predicted values with the actual ones using RMSE, MAE, and MAPE - Visualize the forecast vs actual values

```r
# Create Train/Test Split
# Suppose we hold out the last 12 months for testing
n <- length(ts_cocoa)

# Use data until Feb 2024 for training, and hold out Mar 2024 onward
train_ts <- window(ts_cocoa, end = c(2024, 2))
test_ts <- window(ts_cocoa, start = c(2024, 3))

# Fit ARIMA on training set
fit_train <- auto.arima(train_ts)
```

```r
# Forecast on test horizon
fc_test <- forecast(fit_train, h = length(test_ts))

# Evaluate Forecast Accuracy on Test Set
rmse_val <- rmse(test_ts, fc_test$mean)
mae_val <- mae(test_ts, fc_test$mean)
mape_val <- mape(test_ts, fc_test$mean)

# Print evaluation metrics
cat("RMSE:", rmse_val, "\nMAE:", mae_val, "\nMAPE:", mape_val, "\n")
```

```
## RMSE: 2811.824
## MAE: 2404.775
## MAPE: 0.2690643
```

```r
# Plot forecast vs actual
autoplot(fc_test) +
  autolayer(test_ts, series = "Actual") +
  ggtitle("Forecast vs Actual (Test Set)") +
  ylab("US$/tonne") + xlab("Year")
```

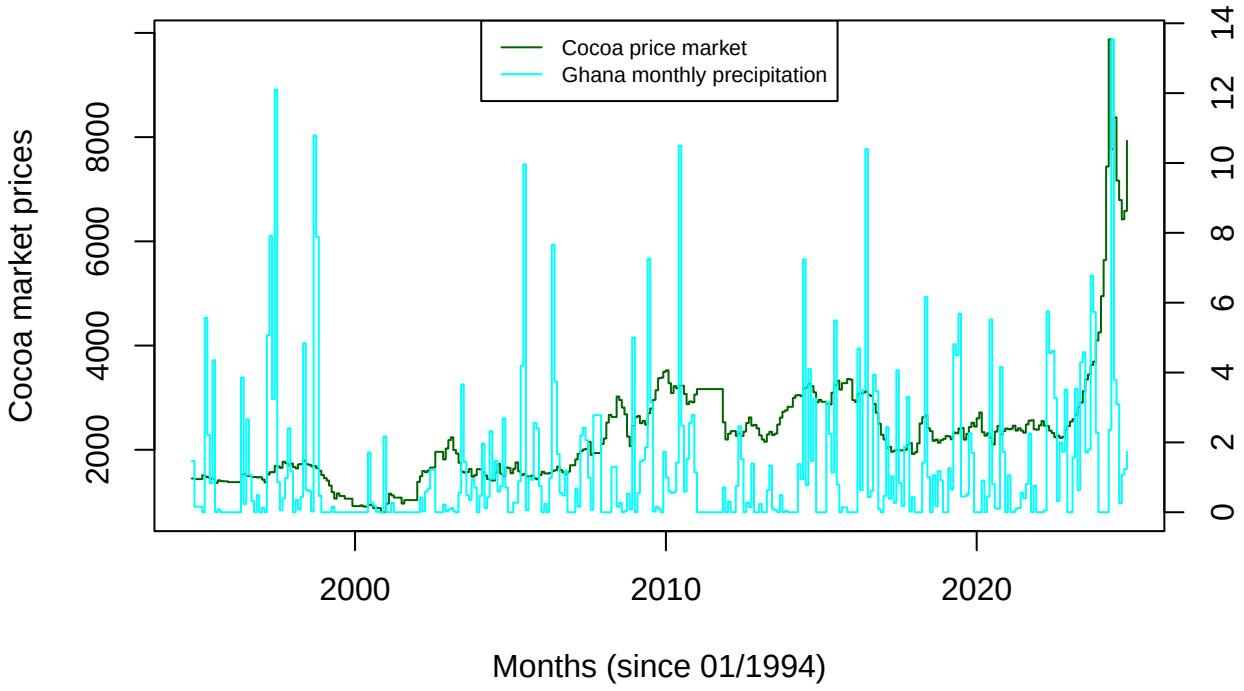


11. Merging Cocoa and Climate Data for Multivariate Modeling

We now prepare the final dataset for modeling by merging monthly cocoa prices with monthly aggregated climate indicators from Ghana. This dataset will allow us to use exogenous variables like precipitation and temperature in models such as Prophet with regressors or ARIMAX/XGBoost.

We also create a rough visualization to explore how cocoa prices and monthly precipitation might relate over time.

```
# Preparing Prophet input data
#Note: Only uncomment this if testing the forecast to actual values
#df_prophet <- cocoa_monthly %>%
#  rename(ds = YearMonth, y = MeanPrice) %>%
#  left_join(ghana_monthly, by = c("ds" = "YearMonth")) %>%
#  na.omit()
#df_prophet_complete = df_prophet

# Used for splitting train data when necessary
#df_prophet = df_prophet[1:323,]

# This left join merges two dataset (cocoa price and Ghana production) into
# one dataset where df stands for dataframe
df_prophet <- cocoa_monthly %>%
  rename(ds = YearMonth, y = MeanPrice) %>%
  left_join(ghana_monthly, by = c("ds" = "YearMonth")) %>%
  na.omit()

# Rough visualisation of the data trend
# Plot cocoa prices (primary axis)
plot(x = df_prophet$ds, y = df_prophet$y,
     xlab = "Months (since 01/1994)",
     ylab = "Cocoa market prices",
     type = "s", col = "darkgreen",
     )

# Overlay precipitation data (secondary axis)
par(new = TRUE)
plot(x = df_prophet$ds, y = df_prophet$PRCP,
     xlab = "", ylab = "",
     type = "s", col = "cyan",
     axes = FALSE,
     )

# Add right-side axis for precipitation
axis(4)
mtext("Sum of monthly precipitation", side = 4, line = 3)

# Add a legend
legend("top",
       legend = c("Cocoa price market", "Ghana monthly precipitation"),
       col = c("darkgreen", "cyan"),
       lwd = 1,
       cex = 0.7)
```



12. Prophet Model with Climate Regressors

In this section, we build a Facebook Prophet model and enhance it by adding four exogenous climate variables as regressors: - Monthly precipitation (`PRCP`) - Average temperature (`TAVG`) - Maximum temperature (`TMAX`) - Minimum temperature (`TMIN`)

This allows Prophet to account for external factors that may influence cocoa prices, beyond seasonal and trend components.

```
# Build and configure the Prophet model
# Initialize a standard Prophet model
prophet_model <- prophet()

# Add climate regressors manually into the model for fitting and predicting
# This is what needs to be done for a typical Prophet's model
prophet_model <- add_regressor(prophet_model, 'PRCP')
prophet_model <- add_regressor(prophet_model, 'TAVG')
prophet_model <- add_regressor(prophet_model, 'TMAX')
prophet_model <- add_regressor(prophet_model, 'TMIN')

# Fit the Prophet model with both trend/seasonality and added regressors
prophet_model <- fit.prophet(prophet_model, df_prophet)
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

13. Forecasting with Prophet: Future Climate Placeholder Strategy

To make future predictions with Prophet (especially when using external regressors), we must provide future values for those regressors. Since we don't have actual future data, we: - Extend the time series by 12 months - Fill in future regressor values using the most recent observed monthly averages (as placeholders)

This allows the Prophet model to forecast cocoa prices with consistent inputs.

```
# Extend the timeframe with the next 12 months in a new dataset (future)
# df_prophet$ds shows at which month the data was recorded (and later
# monthly averaged) for each monthly averaged data
# Note: the maximum in df_prophet$ds is the latest in time.
last_climate <- df_prophet[which.max(df_prophet$ds), c("PRCP", "TAVG", "TMAX",
                                                       "TMIN")]

# Create a new future dataframe and fill in future climate.
# Note: The interval is the overlapping period of two merged datasets
# (cocoa prices and Ghana production) since Ghana production and Cocoa price
# both overlap starting from January 1994.
future <- make_future_dataframe(prophet_model, periods = 12, freq = "month")

# coalesce here is to replace NA values in the first vector, call in X,
# with non-NA values, if any, in Y at the corresponding indexes.
# X <- c(0, 5, NA, 3)
# Y <- c(NA, 1, 2, 4)
# coalesce(X, Y)
# [1] 0 5 2 3
# Notice that is is not [1] 0 1 2 4. It only replaces value in the first
# vector X, if possible.
#
# For instance: last_climate$PRCP (as Y) will use the recorded latest monthly
# average in PRCP (as X) in November 2024, to paste that value over NA ones
# with coelesce over the next 12 months to November 2025.
# Note: The pasted values are just for placeholder, and they are not
# forecasted values
future <- future %>%
  left_join(df_prophet %>% select(ds, PRCP, TAVG, TMAX, TMIN), by = "ds") %>%
  mutate(
    PRCP = coalesce(PRCP, last_climate$PRCP),
    TAVG = coalesce(TAVG, last_climate$TAVG),
    TMAX = coalesce(TMAX, last_climate$TMAX),
    TMIN = coalesce(TMIN, last_climate$TMIN)
  )

# Generate forecasts using the Prophet model
forecast_prophet_future <- predict(prophet_model, future)
```

14. Prophet + ARIMA Hybrid Model: Residual Forecasting

To improve accuracy beyond Prophet's default capabilities, we build a hybrid model: 1. Prophet is used to model long-term trend, seasonality, and regressors. 2. The residuals (actual - forecasted) from Prophet

are extracted. 3. An ARIMA model is then fitted to the residuals to capture autocorrelation or remaining signal. 4. This ARIMA model forecasts the next 12 months of residuals, which can be added back to Prophet forecasts for a hybrid prediction.

```
# Compute residuals from historical fit
# When the right dataframe does not have enough row as the left one
# (in this case, prophet_model has 12 extra than the original df_prophet),
# using left join in SQL with bigger set on the left (prophet_model)
# to merge with small one while retaining the bigger set's row.
# Example as:
#    A                          B
# x  |  y                    y  |  z
# -------                    -------
# 3  |  2                    2  |  4
# 4  |  3
#
# The merged one as C
# x  |  y  |  z
# --------------
# 3  |  2  |  4
# 4  |  3  |  NA
#
# In this case, since prophet_model as X (forecasting) has 12 extra rows than # the original dataset df
# merge both into the bigger dataset residuals_df with the same rows as
# prophet_model
residuals_df <- df_prophet %>%
  left_join(predict(prophet_model, df_prophet) %>% select(ds, yhat), by = "ds") %>%
  mutate(residual = y - yhat)

# Build ARIMA model on the residual of the Prophet's model
# The Prophet's model is a good tool for detrending the time series
# Thus, now the residual is detrended data
# Create time series of residuals
ts_resid <- ts(residuals_df$residual, frequency = 12)

# Fit ARIMA to residuals
arima_resid <- auto.arima(ts_resid)
summary(arima_resid)
```

```
## Series: ts_resid
## ARIMA(2,0,0)(0,0,1)[12] with zero mean
##
## Coefficients:
##          ar1     ar2    sma1
##       0.7689  0.1708  0.2387
## s.e.  0.0555  0.0567  0.0737
##
## sigma^2 = 109393:  log likelihood = -2361.01
## AIC=4730.02   AICc=4730.15   BIC=4745.18
##
## Training set error measures:
##                     ME     RMSE     MAE      MPE     MAPE      MASE        ACF1
## Training set 11.38434 329.2251 218.484 9.168409 169.7952 0.4471038 0.004965371
```

```r
# Forecast ARIMA residuals for next 12 months
fc_resid <- forecast(arima_resid, h = 12)
```

15. Combining Prophet and ARIMA Residual Forecasts

We now generate the final hybrid forecast by: - Extracting the last 12 months of Prophet's future forecast - Adding the ARIMA-predicted residuals to the `yhat` values from Prophet

This gives us a hybrid forecast that incorporates both structured components (via Prophet) and unexplained variance (via ARIMA), improving overall accuracy.

```r
# Final 12-month hybrid forecast dataframe
# It contains forecasted values throughout the entire interval,
# but we are only interested in the last 12 months for hybrid forecast
# model of Prophet's model + fc_resid$mean, which are the predicted values
# from the ARIMA(1, 0, 0) with zero mean itself.
final_forecast <- forecast_prophet_future %>%
  tail(12) %>%
  mutate(hybrid = yhat + fc_resid$mean)
```

16. Hybrid Forecast Evaluation (Prophet + ARIMA)

We now evaluate the hybrid model's forecast accuracy by: - Comparing its predictions with the actual cocoa prices over the last 12 months - Calculating: - RMSE (Root Mean Squared Error) - MAE (Mean Absolute Error) - MAPE (Mean Absolute Percentage Error)

These metrics quantify the prediction error, helping us assess how well the hybrid model performs.

```r
# Extract forecasted dates and actuals
# Get forecasted dates
forecast_dates <- final_forecast$ds

# Get last 12 actual values
# Remember df_prophet is the original merged dataset
# arrange to help bring the last 12 latest into the tail
df_actual <- df_prophet %>%
  arrange(ds) %>%
  tail(12) %>%
  select(ds, y)

# Get last 12 hybrid forecast values
hybrid_forecast <- final_forecast %>%
  arrange(ds) %>%
  tail(12) %>%
  select(ds, hybrid)

# Combine into one frame, containing the last 12-month predicted vs. actual data
df_eval <- data.frame(
  actual = df_actual$y,
  predicted = hybrid_forecast$hybrid
)
```

```r
# Check structure by displaying the first several observations of each column
str(df_eval)
```

```
## 'data.frame':    12 obs. of  2 variables:
##  $ actual   : num  4254 4948 5640 7435 9877 ...
##  $ predicted: num  7294 7368 7063 7555 8015 ...
```

```r
summary(df_eval)
```

```
##      actual       predicted
##  Min.   :4254   Min.   :7063
##  1st Qu.:6226   1st Qu.:7294
##  Median :6978   Median :7462
##  Mean   :6932   Mean   :7617
##  3rd Qu.:7805   3rd Qu.:8028
##  Max.   :9877   Max.   :8257
```

```r
# Forecast performance metrics
# Compute root mean squared error between two numeric vectors: actual, predicted
rmse_hybrid <- rmse(df_eval$actual, df_eval$predicted)
# Compute mean absolute error
mae_hybrid <- mae(df_eval$actual, df_eval$predicted)
# Compute mean absolute percent error
mape_hybrid <- mape(df_eval$actual, df_eval$predicted)

# Print metrics
cat("Hybrid Forecast RMSE:", rmse_hybrid,
    "\n MAE:", mae_hybrid,
    "\n MAPE:", mape_hybrid)
```

```
## Hybrid Forecast RMSE: 1496.657
##  MAE: 1237.336
##  MAPE: 0.2097517
```

17. Visualizing Hybrid Forecast vs Actual Prices

We now prepare the data for visualization by combining actual cocoa prices and the hybrid model's predictions for the last 12 months. This helps visually assess how well the hybrid model tracks real-world price changes.

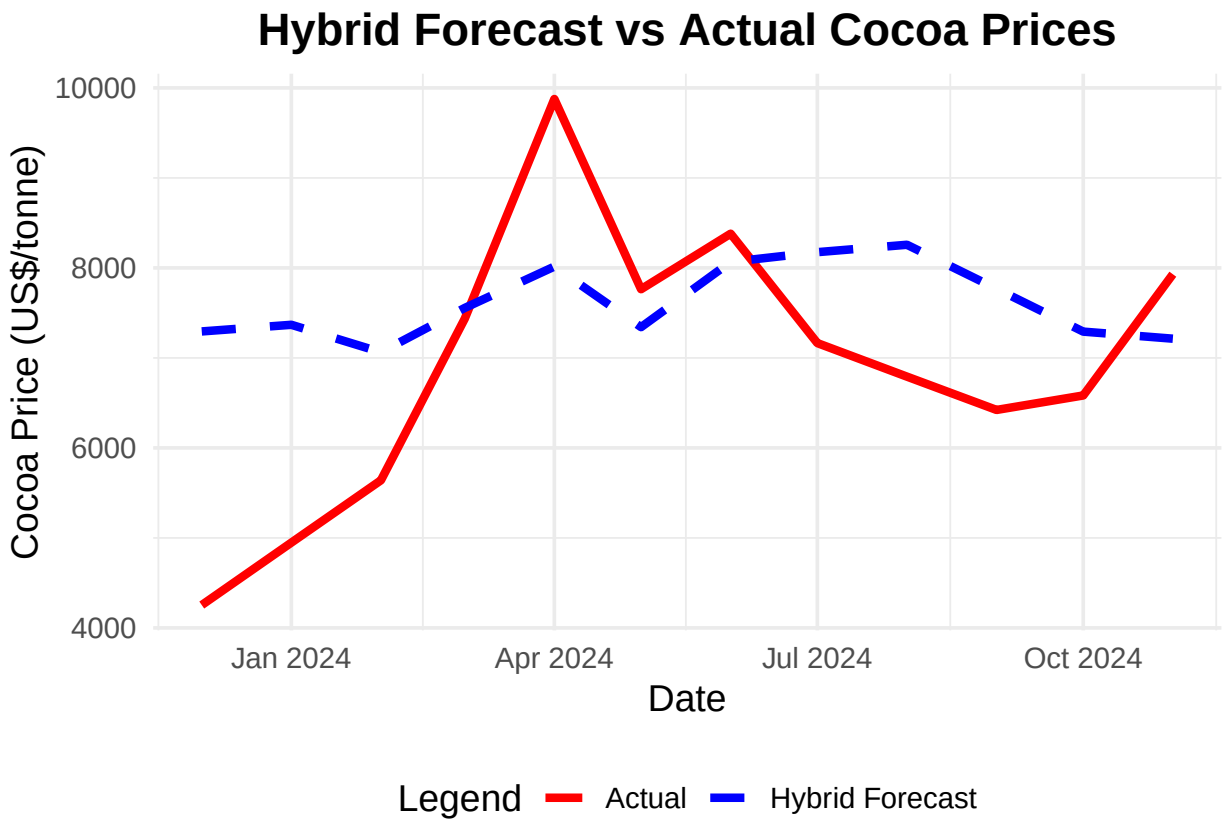
```r
# Combine actual and hybrid forecast values into a data frame, ready for
# plotting
df_plot <- data.frame(
  Date = df_actual$ds,
  Actual = df_actual$y,
  Hybrid = hybrid_forecast$hybrid
)
```

18. Final Plot: Hybrid Forecast vs Actual Prices

Below, we plot the hybrid forecast alongside the actual cocoa prices for the most recent 12 months. This allows us to visually assess the accuracy and consistency of our hybrid model over time.

```
# Plot actual vs. hybrid forecast values-
ggplot(df_plot, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual"), size = 1.5) +
  geom_line(aes(y = Hybrid, color = "Hybrid Forecast"), size = 1.5, linetype = "dashed") +
  labs(
    title = "Hybrid Forecast vs Actual Cocoa Prices",
    x = "Date",
    y = "Cocoa Price (US$/tonne)",
    color = "Legend"
  ) +
  theme_minimal(base_size = 14) +
  scale_color_manual(values = c("Actual" = "red", "Hybrid Forecast" = "blue")) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    legend.position = "bottom"
  )
```

```
## 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.
```



19. Hyperparameter Tuning: Prophet + ARIMA Hybrid Model

To optimize the hybrid model, we perform a grid search over Prophet's key hyperparameters:

- `changepoint.prior.scale (CPS)` controls how sensitive the model is to trend changes.
- `seasonality.prior.scale (SPS)` controls how strongly seasonal patterns are enforced.

For each parameter pair: 1. We fit a Prophet model with climate regressors. 2. Fit an ARIMA model on the residuals. 3. Combine forecasts from both. 4. Evaluate the hybrid forecast using RMSE, MAE, and MAPE on the most recent 12 months.

```
# Set tuning grid and initialize results list
# A special feature of Prophet's model is that it can provide tuning feature
# for more specific application and goal, like in this one.

# To keep the result consistent
set.seed(457)

# Define tuning grid
cps_vals <- c(0.01, 0.05, 0.1, 0.3, 0.5)
sps_vals <- c(5, 10, 20, 40)
grid <- expand.grid(CPS = cps_vals, SPS = sps_vals)

results <- list()

# Repeat the tuning process until CPS (Changepoint Prior Scale -
# Model sensitivity to trend shift)
```

```r
# and SPS (Seasonality Prior Scale - regulate the strength of
# seasonality component) are satisfied
for (i in 1:nrow(grid)) {
  cps <- grid$CPS[i]
  sps <- grid$SPS[i]

  cat("Trying CPS =", cps, "SPS =", sps, "...\n")

  tryCatch({
    # Fit Prophet
    m <- prophet(
      changepoint.prior.scale = cps,
      seasonality.prior.scale = sps,
      yearly.seasonality = TRUE
    )

    m <- add_regressor(m, 'PRCP')
    m <- add_regressor(m, 'TAVG')
    m <- add_regressor(m, 'TMAX')
    m <- add_regressor(m, 'TMIN')

    m <- fit.prophet(m, df_prophet)

    # Predict future
    last_climate <- df_prophet[which.max(df_prophet$ds), c("PRCP", "TAVG", "TMAX", "TMIN")]

    future <- make_future_dataframe(m, periods = 12, freq = "month")
    future <- future %>%
      left_join(df_prophet %>% select(ds, PRCP, TAVG, TMAX, TMIN), by = "ds") %>%
      mutate(
        PRCP = coalesce(PRCP, last_climate$PRCP),
        TAVG = coalesce(TAVG, last_climate$TAVG),
        TMAX = coalesce(TMAX, last_climate$TMAX),
        TMIN = coalesce(TMIN, last_climate$TMIN)
      )

    forecast_prophet <- predict(m, future)

    # Residuals from in-sample
    fitted_df <- df_prophet %>%
      left_join(predict(m, df_prophet) %>% select(ds, yhat), by = "ds") %>%
      mutate(residual = y - yhat)

    ts_resid <- ts(fitted_df$residual, frequency = 12)
    arima_resid <- auto.arima(ts_resid)
    fc_resid <- forecast(arima_resid, h = 12)

    final_forecast <- forecast_prophet %>%
      tail(12) %>%
      mutate(hybrid = yhat + fc_resid$mean)

    df_actual <- df_prophet %>%
      arrange(ds) %>%
```

```
      tail(12) %>%
      select(ds, y)

    hybrid_forecast <- final_forecast %>%
      arrange(ds) %>%
      tail(12) %>%
      select(ds, hybrid)

    df_eval <- data.frame(
      actual = df_actual$y,
      predicted = hybrid_forecast$hybrid
    )

    # Evaluate
    rmse_val <- rmse(df_eval$actual, df_eval$predicted)
    mae_val <- mae(df_eval$actual, df_eval$predicted)
    mape_val <- mape(df_eval$actual, df_eval$predicted)

    cat("CPS:", cps, "| SPS:", sps,
        "| RMSE:", round(rmse_val, 2),
        "| MAPE:", round(mape_val * 100, 2), "%\n")

    results[[i]] <- data.frame(CPS = cps, SPS = sps,
                               RMSE = rmse_val, MAE = mae_val, MAPE = mape_val)

  }, error = function(e) {
    cat("Failed for CPS =", cps, "SPS =", sps, "\n")
    results[[i]] <- data.frame(CPS = cps, SPS = sps,
                               RMSE = NA, MAE = NA, MAPE = NA)
  })
}
```

```
## Trying CPS = 0.01 SPS = 5 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.01 | SPS: 5 | RMSE: 1611.87 | MAPE: 22.64 %
## Trying CPS = 0.05 SPS = 5 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.05 | SPS: 5 | RMSE: 1497.43 | MAPE: 21 %
## Trying CPS = 0.1 SPS = 5 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.1 | SPS: 5 | RMSE: 1528.99 | MAPE: 21.25 %
## Trying CPS = 0.3 SPS = 5 ...
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.3 | SPS: 5 | RMSE: 1551.25 | MAPE: 21.54 %
## Trying CPS = 0.5 SPS = 5 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.5 | SPS: 5 | RMSE: 1557.08 | MAPE: 21.61 %
## Trying CPS = 0.01 SPS = 10 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.01 | SPS: 10 | RMSE: 1611.91 | MAPE: 22.67 %
## Trying CPS = 0.05 SPS = 10 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.05 | SPS: 10 | RMSE: 1496.66 | MAPE: 20.98 %
## Trying CPS = 0.1 SPS = 10 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.1 | SPS: 10 | RMSE: 1523.91 | MAPE: 21.17 %
## Trying CPS = 0.3 SPS = 10 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.3 | SPS: 10 | RMSE: 1555.84 | MAPE: 21.59 %
## Trying CPS = 0.5 SPS = 10 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.5 | SPS: 10 | RMSE: 1556.55 | MAPE: 21.6 %
## Trying CPS = 0.01 SPS = 20 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.01 | SPS: 20 | RMSE: 1614.07 | MAPE: 22.71 %
## Trying CPS = 0.05 SPS = 20 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```
## CPS: 0.05 | SPS: 20 | RMSE: 1496.73 | MAPE: 20.99 %
## Trying CPS = 0.1 SPS = 20 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.1 | SPS: 20 | RMSE: 1523.73 | MAPE: 21.15 %
## Trying CPS = 0.3 SPS = 20 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.3 | SPS: 20 | RMSE: 1557.86 | MAPE: 21.62 %
## Trying CPS = 0.5 SPS = 20 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.5 | SPS: 20 | RMSE: 1557.02 | MAPE: 21.61 %
## Trying CPS = 0.01 SPS = 40 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.01 | SPS: 40 | RMSE: 1612.72 | MAPE: 22.68 %
## Trying CPS = 0.05 SPS = 40 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.05 | SPS: 40 | RMSE: 1498.7 | MAPE: 20.98 %
## Trying CPS = 0.1 SPS = 40 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.1 | SPS: 40 | RMSE: 1525.18 | MAPE: 21.18 %
## Trying CPS = 0.3 SPS = 40 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.3 | SPS: 40 | RMSE: 1553.74 | MAPE: 21.57 %
## Trying CPS = 0.5 SPS = 40 ...

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## CPS: 0.5 | SPS: 40 | RMSE: 1556.06 | MAPE: 21.59 %
```

```r
# Combine all results
tuning_results <- bind_rows(results)

# Sort by best RMSE
tuning_results %>% arrange(RMSE)
```

```
##     CPS SPS     RMSE      MAE      MAPE
## 1  0.05  10 1496.657 1237.336 0.2097517
## 2  0.05  20 1496.727 1238.090 0.2098633
## 3  0.05   5 1497.429 1238.021 0.2099830
## 4  0.05  40 1498.695 1239.374 0.2098472
## 5  0.10  20 1523.729 1250.960 0.2115275
## 6  0.10  10 1523.909 1254.094 0.2116734
## 7  0.10  40 1525.178 1255.209 0.2118492
## 8  0.10   5 1528.989 1259.970 0.2124943
## 9  0.30   5 1551.254 1269.562 0.2153772
## 10 0.30  40 1553.742 1271.293 0.2156814
## 11 0.30  10 1555.841 1272.994 0.2158876
## 12 0.50  40 1556.063 1270.391 0.2159350
## 13 0.50  10 1556.552 1270.209 0.2160281
## 14 0.50  20 1557.015 1270.761 0.2160993
## 15 0.50   5 1557.076 1270.870 0.2161152
## 16 0.30  20 1557.857 1274.432 0.2162080
## 17 0.01   5 1611.871 1352.123 0.2264303
## 18 0.01  10 1611.911 1356.695 0.2266892
## 19 0.01  40 1612.716 1357.918 0.2268197
## 20 0.01  20 1614.069 1359.966 0.2270545
```

```r
# ------------------------------------------------------------------------------
# Find the changepoint prior scale (CPS) and seasonality prior scale (SPS)
# ------------------------------------------------------------------------------
# The (CPS, SPS) choice for the best GARCH predictive in term of RMSE
best.RMSE <- tuning_results[which.min(tuning_results$RMSE),][1:2]
# The (CPS, SPS) choice for the best GARCH predictive in term of MAE
best.MAE <- tuning_results[which.min(tuning_results$MAE),][1:2]
# The (CPS, SPS) choice for the best GARCH predictive in term of MAPE
best.MAPE <- tuning_results[which.min(tuning_results$MAPE),][1:2]

best.RMSE.cps <- as.numeric(best.RMSE[1])
best.MAE.cps <- as.numeric(best.MAE[1])
best.MAPE.cps <- as.numeric(best.MAPE[1])

best.RMSE.sps <- as.numeric(best.RMSE[2])
best.MAE.sps <- as.numeric(best.MAE[2])
best.MAPE.sps <- as.numeric(best.MAPE[2])

if (all.equal(best.RMSE.cps, best.MAE.cps, best.MAPE.cps) &&
    all.equal(best.RMSE.sps, best.MAE.sps, best.MAPE.sps)) {
  best.cps <- best.RMSE.cps
  best.sps <- best.MAPE.sps
} else {
  # In case of conflict that RMSE, MAE, and MAPE do not agree on one candidate
  # Since RMSE puts more penalty on larger errors it is used.
```

```
  best.cps <- best.RMSE.cps
  best.sps <- best.RMSE.sps
}
```

20. Selecting Optimal Hyperparameters and Fitting Final Prophet Model

Based on the tuning results, we now: - Identify the best values of `changepoint.prior.scale` (CPS) and `seasonality.prior.scale` (SPS) using RMSE, MAE, and MAPE - Prioritize RMSE in case of disagreement between metrics - Refit the final Prophet model on the entire dataset using the selected CPS and SPS

```
# Now it should have a controlled sensitivity
# to trend shift and good regulation of strength in seasonality component in
# the model
# Refit the final Prophet's model
# Refit Prophet on the full dataset
prophet_final <- prophet(
  changepoint.prior.scale = best.cps,
  seasonality.prior.scale = best.sps
)

# Add climate regressors
prophet_final <- add_regressor(prophet_final, 'PRCP')
prophet_final <- add_regressor(prophet_final, 'TAVG')
prophet_final <- add_regressor(prophet_final, 'TMAX')
prophet_final <- add_regressor(prophet_final, 'TMIN')

# Fit with full data
prophet_final <- fit.prophet(prophet_final, df_prophet)
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

21. Final 12-Month Forecast Using Tuned Prophet Model

Using the best-tuned `changepoint.prior.scale` and `seasonality.prior.scale` values, we now: - Recreate the future dataframe for the next 12 months - Fill missing regressor values with the most recent observed ones - Generate the final forecast using the optimized Prophet model

```
# Prepare future dataframe with climate placeholders
# Extract latest observed climate values
last_climate <- df_prophet[which.max(df_prophet$ds), c("PRCP", "TAVG", "TMAX", "TMIN")]

# Create future periods for the next 12 months
future_final <- make_future_dataframe(prophet_final, periods = 12, freq = "month")

# Add and fill regressors
future_final <- future_final %>%
```

```
  left_join(df_prophet %>% select(ds, PRCP, TAVG, TMAX, TMIN), by = "ds") %>%
  mutate(
    PRCP = coalesce(PRCP, last_climate$PRCP),
    TAVG = coalesce(TAVG, last_climate$TAVG),
    TMAX = coalesce(TMAX, last_climate$TMAX),
    TMIN = coalesce(TMIN, last_climate$TMIN)
  )

# Generate final forecast
forecast_final <- predict(prophet_final, future_final)
```

22. Final ARIMA Residual Modeling (Hybrid Step)

To further improve accuracy, we fit an ARIMA model to the residuals of the final, tuned Prophet model. This allows us to correct for any autocorrelated errors not captured by Prophet.

```
# Compute residuals from tuned Prophet model
# Residuals from historical fit
residuals_df_final <- df_prophet %>%
  left_join(predict(prophet_final, df_prophet) %>% select(ds, yhat), by = "ds") %>%
  mutate(residual = y - yhat)

# Convert residuals to a time series
ts_resid_final <- ts(residuals_df_final$residual, frequency = 12)

# Fit ARIMA to Prophet residuals
# Fit ARIMA on residuals
arima_final <- auto.arima(ts_resid_final)

# Forecast residuals for next 12 months
fc_resid_final <- forecast(arima_final, h = 12)
```

23. Final Hybrid Forecast (Prophet + ARIMA Residuals)

We now construct the final hybrid forecast by combining: - `yhat`: Forecasted cocoa prices from the tuned Prophet model - `fc_resid_final$mean`: Forecasted residuals from the ARIMA model

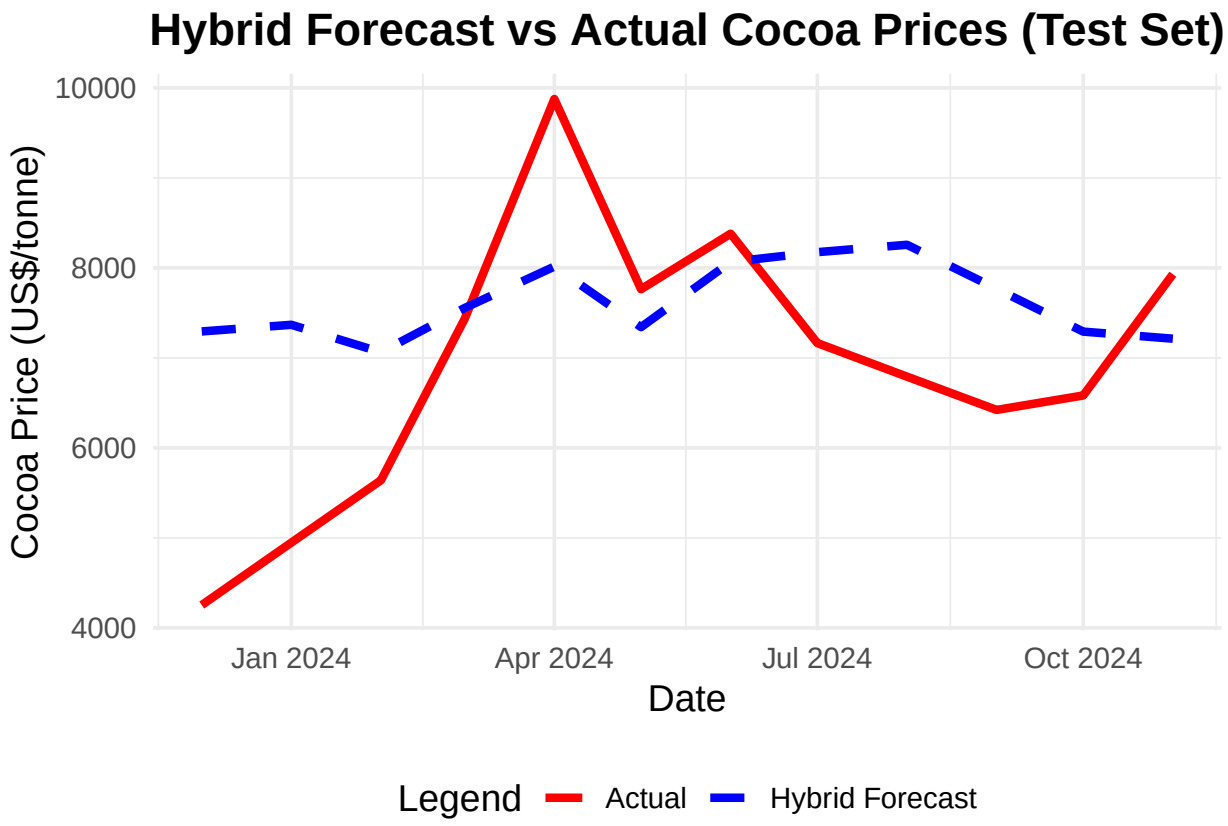
This gives us the most accurate and informed prediction available in this analysis.

```
# Construct final hybrid forecast for next 12 months
hybrid_forecast_final <- forecast_final %>%
  tail(12) %>%
  mutate(hybrid = yhat + fc_resid_final$mean)
```

24. Final Visualization: Hybrid Forecast vs Actual

This final plot compares the actual cocoa prices over the last 12 months with the predictions from our tuned hybrid model (Prophet + ARIMA residuals). The visualization helps assess how well the model captures real price movements.

```r
# Prepare data for final plot
df_actual_final <- df_prophet %>%
  arrange(ds) %>%
  tail(12) %>%
  select(ds, y)

df_plot_final <- data.frame(
  Date = df_actual_final$ds,
  Actual = df_actual_final$y,
  Hybrid = hybrid_forecast_final$hybrid
)

# Plot: Actual vs Hybrid Forecast
ggplot(df_plot_final, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual"), size = 1.5) +
  geom_line(aes(y = Hybrid, color = "Hybrid Forecast"), size = 1.5, linetype = "dashed") +
  labs(
    title = "Hybrid Forecast vs Actual Cocoa Prices (Test Set)",
    x = "Date",
    y = "Cocoa Price (US$/tonne)",
    color = "Legend"
  ) +
  theme_minimal(base_size = 14) +
  scale_color_manual(values = c("Actual" = "red", "Hybrid Forecast" = "blue")) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    legend.position = "bottom"
  )
```



25. Full Series Visualization: Final Hybrid Forecast vs Actual Prices

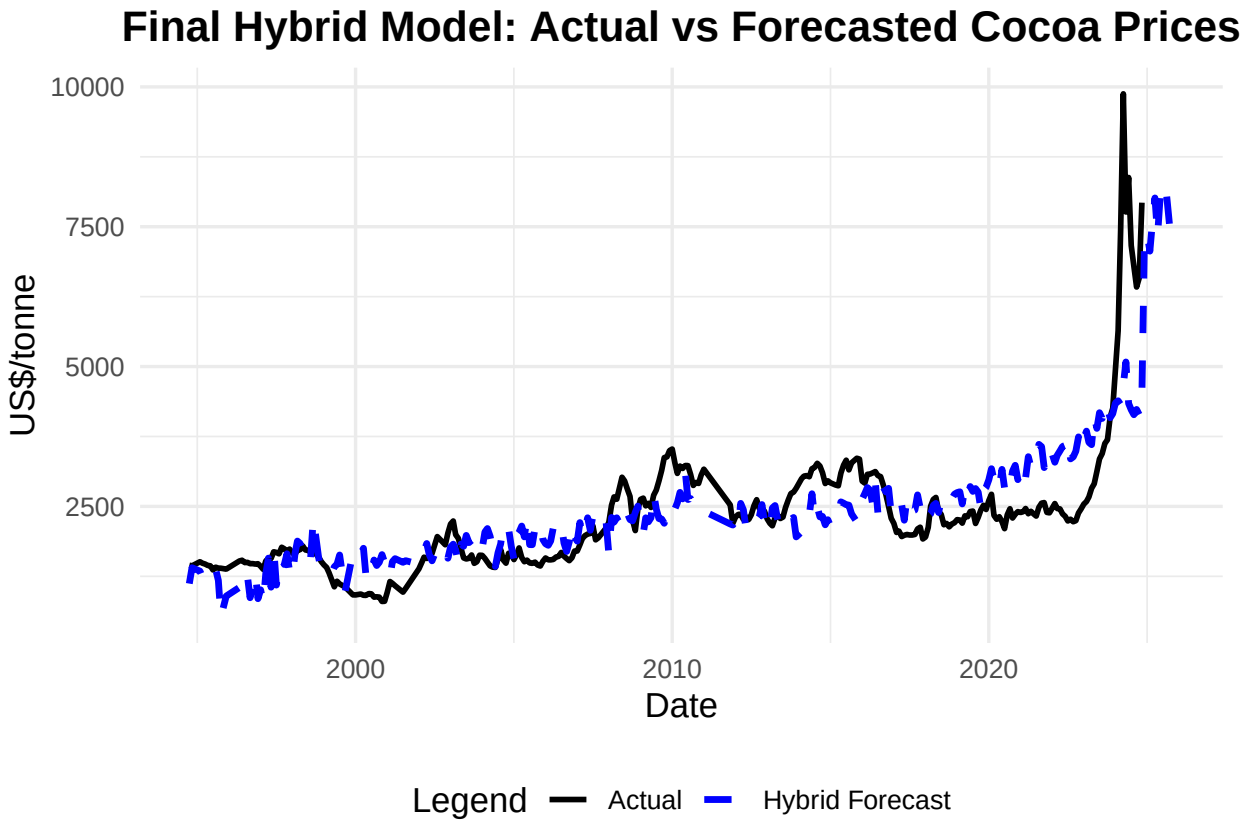
This plot shows the full timeline of cocoa prices, including: - Actual prices (black solid line) - Hybrid model predictions: - Historical fit = tuned Prophet forecast - Future forecast = Prophet + ARIMA residual correction (hybrid)

This final visualization provides a complete view of model performance both in-sample and out-of-sample.

```
# Prepare full timeline for final hybrid forecast plot
df_full <- df_prophet %>%
  select(ds, y) %>%
  right_join(forecast_final %>% select(ds, yhat), by = "ds") %>%
  mutate(hybrid = yhat)  # Placeholder before adding residuals

# Inject ARIMA residual forecast into the last 12 months
df_full$hybrid[(nrow(df_full) - 11):nrow(df_full)] <-
  df_full$yhat[(nrow(df_full) - 11):nrow(df_full)] + fc_resid_final$mean

# Plot full series: actual vs hybrid forecast
ggplot(df_full, aes(x = ds)) +
  geom_line(aes(y = y, color = "Actual"), size = 1) +
  geom_line(aes(y = hybrid, color = "Hybrid Forecast"), size = 1.2, linetype = "dashed") +
  labs(
    title = "Final Hybrid Model: Actual vs Forecasted Cocoa Prices",
    x = "Date",
    y = "US$/tonne",
    color = "Legend"
  ) +
```

```
  theme_minimal(base_size = 13) +
  scale_color_manual(values = c("Actual" = "black", "Hybrid Forecast" = "blue")) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    legend.position = "bottom"
  )
```

```
## Warning: Removed 12 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



26. GARCH Residual Modeling: Setup

We now explore using a GARCH model to capture potential volatility clustering in the residuals of the hybrid model. This is useful when residuals display non-constant variance, which violates ARIMA assumptions.

First, we: - Recompute residuals from the tuned Prophet model - Convert them into a time series object for GARCH modeling

```
# ---------------------------------
# Residual analysis
# ---------------------------------
# Since GARCH model works on the residual plot to make it white noise.
# It is important to extract the residual component and make a time series
# of it.
#
# Recompute residuals from the hybrid model after updating the Prophet's model
```

```r
residuals_df <- df_prophet %>%
  left_join(predict(prophet_model, df_prophet) %>% select(ds, yhat), by = "ds") %>%
  mutate(residual = y - yhat)

# Convert residuals to time series for GARCH
ts_resid <- ts(residuals_df$residual, frequency = 12)
```

27. GARCH Model Selection for Residual Volatility

To model volatility clustering in the residuals of our time series, we perform a grid search over multiple GARCH(p, q) models, where: - `p` = number of lagged conditional variances (ARCH terms) - `q` = number of lagged forecast errors (GARCH terms)

We evaluate all models with $0 \leq p, q \leq 2$, excluding the invalid (0,0) case. Each model is scored based on AIC and BIC, and the best candidate is selected based on minimum AIC.

```r
# -------------------------------------
# Constructing candidate GARCH models
# -------------------------------------
# Try multiple GARCH(p,q) combinations up to p = 2, and q = 2
# This code snippet helps create two columns of p and q
# with combinations (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), ..., (2, 2).
orders <- expand.grid(p = 0:2, q = 0:2)
orders <- orders[!(orders$p == 0 & orders$q == 0), ]  # remove (0,0)

# Create a list object called results
results <- list()

# To keep the finding consistent
set.seed(457)

# This is basically looping over all possible combinations of ARIMA(p, 0, q),
# simplified it as ARMA(p, q) as (0, 1), ..., (2, 2) like the ones in orders
#
# ugarchspec(...) is to create a univariate GARCH specification object prior to
# fitting

for (i in 1:nrow(orders)) {
  p <- orders$p[i]
  q <- orders$q[i]

  # garchOrder is for the current ARMA(p, q)
  # It is for mean model with ARMA(0, 0) or simply white noise?
  spec_try <- ugarchspec(
    variance.model = list(model = "sGARCH", garchOrder = c(p, q)),
    mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
    distribution.model = "norm"
  )

  tryCatch({
    # spec_try is from right above, ts_resid is the prophet's (?) model
    fit_try <- ugarchfit(spec = spec_try, data = ts_resid, solver = "hybrid")
```

```
    results[[i]] <- data.frame(
      p = p,
      q = q,
      AIC = infocriteria(fit_try)[1],
      BIC = infocriteria(fit_try)[2]
    )
    cat("GARCH(", p, ",", q, ") - AIC:", round(infocriteria(fit_try)[1], 2), "\n")
  }, error = function(e) {
    # Some cases, GARCH will fail for some invalid combination of (p, q)
    # So this is to simply handle when that error arises
    cat("Failed for GARCH(", p, ",", q, ")\n")
    results[[i]] <- data.frame(p = p, q = q, AIC = NA, BIC = NA)
  })
}
```

```
## GARCH( 1 , 0 ) - AIC: 15.29
## GARCH( 2 , 0 ) - AIC: 15.29
## GARCH( 0 , 1 ) - AIC: 16.09
## GARCH( 1 , 1 ) - AIC: 15.29
## GARCH( 2 , 1 ) - AIC: 15.3
## GARCH( 0 , 2 ) - AIC: 16.09
## GARCH( 1 , 2 ) - AIC: 15.3
## GARCH( 2 , 2 ) - AIC: 15.31
```

```
# Combine and rank results
garch_results <- do.call(rbind, results)
# To push the best (p, q) on top of the output
garch_results %>% arrange(AIC)
```

```
##   p q      AIC      BIC
## 1 1 1 15.28519 15.31996
## 2 1 0 15.28859 15.31177
## 3 2 0 15.29089 15.32566
## 4 2 1 15.29938 15.34574
## 5 1 2 15.30001 15.34637
## 6 2 2 15.30508 15.36303
## 7 0 1 16.08995 16.11313
## 8 0 2 16.09220 16.12697
```

28. Selecting the Optimal GARCH(p, q) Model

We now select the best GARCH model based on: - AIC (Akaike Information Criterion) — prefers better fit, less strict - BIC (Bayesian Information Criterion) — prefers simpler models, penalizes complexity more

If both criteria agree, we accept that model. Otherwise, we prioritize BIC due to its stronger theoretical justification for forecasting tasks involving larger datasets.

```
# ----------------------------------------
# Find the best (p, q) model
# ----------------------------------------
# The (p, q) choice for the best GARCH predictive in term of AIC
```

```r
best.AIC <- garch_results[which.min(garch_results$AIC),][1:2]
# The (p, q) choice for the best GARCH predictive in term of BIC
best.BIC <- garch_results[which.min(garch_results$BIC),][1:2]

if (best.AIC[1] == best.BIC[1] && best.AIC[2] == best.BIC [2]) {
  best.p <- best.AIC[1]
  best.q <- best.BIC[2]
} else {
  # Since this is a big model, and BIC places a heavy penalty on
  # the squared errors in predictive performance, it is generally
  # preferred over AIC
  best.p <- best.BIC[1]
  best.q <- best.BIC[2]
}
best.p <- as.numeric(best.p)
best.q <- as.numeric(best.q)
```

29. Fitting the Optimal GARCH Model on Residuals

After identifying the best (p, q) order via model selection, we now: - Define a GARCH specification using `ugarchspec()` - Fit the GARCH model to the residuals of the hybrid Prophet model - Summarize the model to check parameter significance and diagnostics

```r
# ---------------------------------------
# Apply the best candidate GARCH model
# ---------------------------------------
# From inspection above, the lowest AIC (15.47862) and lowest BIC (15.51925)
# happens at (p = 1, q = 0) as garchOrder
# Apply the best GARCH spec
best_spec <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(best.p, best.q)),
  mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
  distribution.model = "norm"
)

# Fit the model on residuals
garch_fit <- ugarchfit(spec = best_spec, data = ts_resid)

# View the model summary
show(garch_fit)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,0)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
```

```
##
## Optimal Parameters
## ------------------------------------
##          Estimate  Std. Error  t value Pr(>|t|)
## omega  7.2133e+04  1.1273e+04   6.3989        0
## alpha1 9.8907e-01  1.4294e-01   6.9197        0
##
## Robust Standard Errors:
##          Estimate  Std. Error  t value Pr(>|t|)
## omega  7.2133e+04  1.0746e+04   6.7127        0
## alpha1 9.8907e-01  1.0984e-01   9.0045        0
##
## LogLikelihood : -2497.685
##
## Information Criteria
## ------------------------------------
##
## Akaike       15.289
## Bayes        15.312
## Shibata      15.289
## Hannan-Quinn 15.298
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                      160.9       0
## Lag[2*(p+q)+(p+q)-1][2]     234.7       0
## Lag[4*(p+q)+(p+q)-1][5]     411.5       0
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.7292  0.3931
## Lag[2*(p+q)+(p+q)-1][2]    1.6153  0.3354
## Lag[4*(p+q)+(p+q)-1][5]    2.7706  0.4505
## d.o.f=1
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[2]     1.751 0.500 2.000  0.1858
## ARCH Lag[4]     2.516 1.397 1.611  0.3382
## ARCH Lag[6]     2.803 2.222 1.500  0.5091
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  6.506
## Individual Statistics:
## omega  0.07272
## alpha1 5.29791
##
## Asymptotic Critical Values (10% 5% 1%)
```

```
## Joint Statistic:          0.61 0.749 1.07
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## -----------------------------------
##                    t-value   prob sig
## Sign Bias           0.4248 0.6713
## Negative Sign Bias  1.1291 0.2597
## Positive Sign Bias  0.3814 0.7032
## Joint Effect        1.9111 0.5911
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     79.18    2.577e-09
## 2    30    102.27    4.231e-10
## 3    40     99.54    3.388e-07
## 4    50    115.66    2.592e-07
##
##
## Elapsed time : 0.1171939
```

30. GARCH-Based Correction for Final Forecast Enhancement

In this final modeling step, we leverage the volatility estimates (t) from the fitted GARCH model to explain and correct residuals from our hybrid Prophet + ARIMA model.

Workflow:

1. Extract conditional standard deviations from the GARCH model
2. Fit a linear regression: residual ~ volatility (GARCH t)
3. Add the fitted correction to the existing hybrid forecast
4. Evaluate the corrected forecast using RMSE, MAE, and MAPE

```r
# Extract residual standard deviation from GARCH fitted model
vol_series <- sigma(garch_fit)
```

```r
# Fit correction model over the full series
correction_model <- lm(residuals_df$residual ~ vol_series)
```

```r
# ------------------------------------------------------------
# Construct a prediction model (Prophet's + ARIMA + GARCH)
# ------------------------------------------------------------
# Ensure vol_series is a numeric vector
vol_series <- as.numeric(vol_series)

# Refit the correction model just to be safe (clean version)
correction_model <- lm(residuals_df$residual ~ vol_series)

# Get full hybrid forecast
```

```r
full_forecast <- predict(prophet_model, df_prophet) %>%
  select(ds, yhat) %>%
  mutate(hybrid = yhat + ts_resid)  # old hybrid = prophet + ARIMA residual

# Incorporating the updated model + GARCH into the existing hybrid one
# When predicting, make sure the input is a data.frame with same column name
full_forecast$correction <- predict(
  correction_model,
  newdata = data.frame(vol_series = vol_series)
)

# Final corrected hybrid forecast
full_forecast$hybrid_corrected <- full_forecast$hybrid + full_forecast$correction
```

```r
# ------------------------------------------------------------
# Forecast and assess the final forecast model performance
# ------------------------------------------------------------
# This is the original dataset (actual data)
df_actual_all <- df_prophet %>% select(ds, y)

# This is to combine the corrected forecasted data (with GARCH) with actual
# data
df_eval_full <- full_forecast %>%
  left_join(df_actual_all, by = "ds") %>%
  select(ds, actual = y, hybrid_corrected)

# Metrics for predictive power again
rmse_corr_all <- rmse(df_eval_full$actual, df_eval_full$hybrid_corrected)
mae_corr_all <- mae(df_eval_full$actual, df_eval_full$hybrid_corrected)
mape_corr_all <- mape(df_eval_full$actual, df_eval_full$hybrid_corrected)

cat("Full Hybrid Corrected RMSE:", rmse_corr_all,
    "\nMAE:", mae_corr_all,
    "\nMAPE:", mape_corr_all)
```

```
## Full Hybrid Corrected RMSE: 353.0653
## MAE: 199.8329
## MAPE: 0.0833957
```

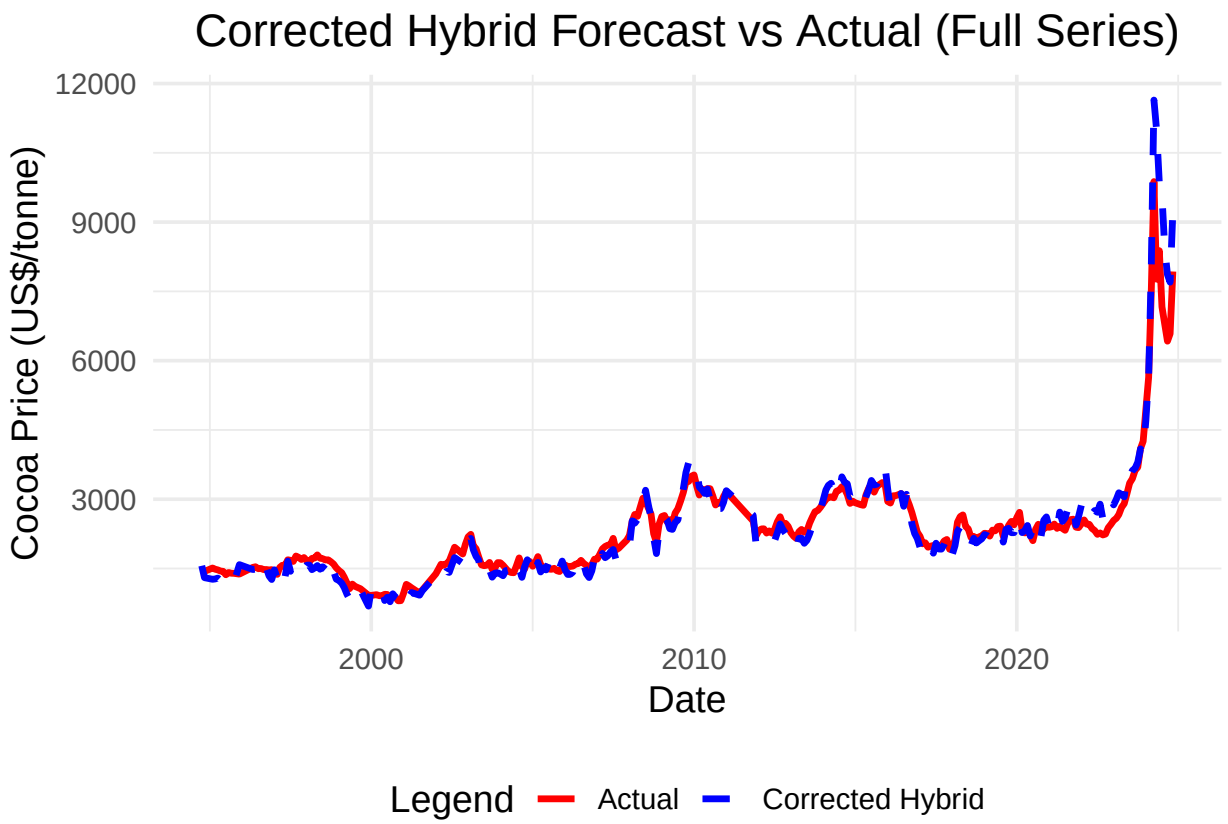
31. Final Visualization: Corrected Hybrid Forecast vs Actual Prices

This plot compares the actual cocoa prices against the fully corrected hybrid forecast, which includes: - Prophet for long-term trend and seasonality - ARIMA for residual autocorrelation - GARCH for volatility-driven correction

The final result is a robust and flexible model with enhanced predictive performance.

```r
# ------------------------------------------------------------
# Contrast the final forecast model with the actual data
# ------------------------------------------------------------
```

```r
ggplot(df_eval_full, aes(x = ds)) +
  geom_line(aes(y = actual, color = "Actual"), size = 1.2) +
  geom_line(aes(y = hybrid_corrected, color = "Corrected Hybrid"), size = 1.2, linetype = "dashed") +
  scale_color_manual(values = c("Actual" = "red", "Corrected Hybrid" = "blue")) +
  labs(
    title = "Corrected Hybrid Forecast vs Actual (Full Series)",
    x = "Date",
    y = "Cocoa Price (US$/tonne)",
    color = "Legend"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5))
```



32. Forecasting Cocoa Prices: 2-Year Outlook

We now encapsulate our entire forecasting pipeline — Prophet + ARIMA + GARCH correction — into a reusable function. This function allows forecasting cocoa prices for any number of months (`n_ahead`), using the latest available data and volatility-corrected residual modeling.

Function Logic:

- Uses latest known climate data to populate regressors
- Generates forecasts using:
  - Prophet (trend/seasonality + regressors)
  - ARIMA (residual structure)

  - GARCH (volatility correction via linear adjustment)
- Returns both:
  - `hybrid` = Prophet + ARIMA
  - `hybrid_corrected` = Prophet + ARIMA + GARCH-corrected

```r
# Final 2-Year Forecast Function
forecast_cocoa_prices <- function(df_prophet, prophet_model, arima_resid, garch_fit, correction_model, n
  library(dplyr)
  library(prophet)
  library(forecast)
  library(rugarch)

  # Get last known climate values with index at which.max(df_prophet$ds)
  last_climate_vals <- df_prophet[which.max(df_prophet$ds), c("PRCP", "TAVG", "TMAX", "TMIN")]

  # Build a rough dataframe for forecasting
  # Create future dataframe, by default, if nothing passes in as n_ahead,
  # will be assumed as 1 year ahead (n = 12 by default)
  future_df <- make_future_dataframe(prophet_model, periods = n_ahead, freq = "month")
  future_df <- future_df %>%
    left_join(df_prophet %>% select(ds, PRCP, TAVG, TMAX, TMIN), by = "ds") %>%
    mutate(
      PRCP = coalesce(PRCP, last_climate_vals$PRCP),
      TAVG = coalesce(TAVG, last_climate_vals$TAVG),
      TMAX = coalesce(TMAX, last_climate_vals$TMAX),
      TMIN = coalesce(TMIN, last_climate_vals$TMIN)
    )

  # Forecast with Prophet
  forecast_prophet <- predict(prophet_model, future_df)

  # Forecast ARIMA residuals
  fc_resid <- forecast(arima_resid, h = n_ahead)

  # Forecast GARCH volatility
  garch_forecast <- ugarchforecast(garch_fit, n.ahead = n_ahead)
  vol_forecast <- sigma(garch_forecast)

  # Combine all components
  forecast_result <- forecast_prophet %>%
    tail(n_ahead) %>%
    mutate(
      hybrid = yhat + fc_resid$mean,
      correction = predict(correction_model, newdata = data.frame(vol_series = as.numeric(vol_forecast))
      hybrid_corrected = hybrid + correction
    )

  # Return the final model as a dataframe with 3 columns, defined as:
  # timestamp (ds),
  # ARIMA + Prophet forecast (hybrid),
  # ARIMA + Prophet forecast + GARCH correction (hybrid_corrected)
  return(forecast_result %>% select(ds, hybrid, hybrid_corrected))
}
```

```r
# Generate 2-Year Forecast (24 months ahead)
forecast_output <- forecast_cocoa_prices(
  df_prophet = df_prophet,
  prophet_model = prophet_model,
  arima_resid = arima_resid,
  garch_fit = garch_fit,
  correction_model = correction_model,
  n_ahead = 24
)

head(forecast_output)
```

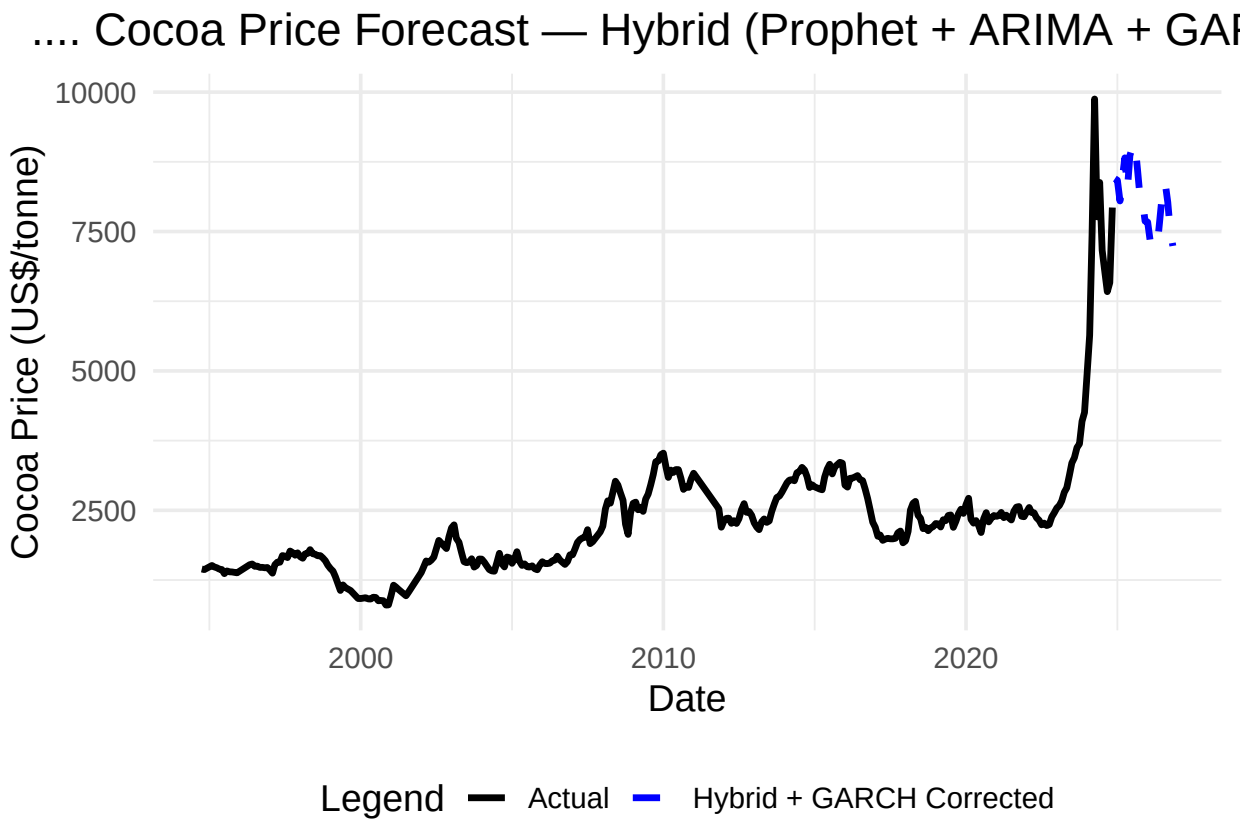
```
##             ds   hybrid hybrid_corrected
## 328 2024-12-01 6455.301         8448.470
## 329 2025-01-01 6431.254         8418.578
## 330 2025-02-01 6066.593         8048.123
## 331 2025-03-01 6480.761         8456.546
## 332 2025-04-01 6851.504         8821.592
## 333 2025-05-01 6286.656         8251.098
```

```r
# ---------------------------------------
# Output the full model forecast into plots
# ---------------------------------------
# Combine forecast with historical data
df_plot_extended <- df_prophet %>%
  select(ds, Actual = y) %>%
  bind_rows(
    forecast_output %>%
      select(ds, Hybrid_Corrected = hybrid_corrected) %>%
      mutate(Actual = NA)  # placeholder for actual future values
  )

# Plot
ggplot(df_plot_extended, aes(x = ds)) +
  geom_line(aes(y = Actual, color = "Actual"), size = 1.2) +
  geom_line(aes(y = Hybrid_Corrected, color = "Hybrid + GARCH Corrected"), size = 1.2, linetype = "dash
  labs(
    title = "  Cocoa Price Forecast - Hybrid (Prophet + ARIMA + GARCH)",
    x = "Date",
    y = "Cocoa Price (US$/tonne)",
    color = "Legend"
  ) +
  scale_color_manual(values = c("Actual" = "black", "Hybrid + GARCH Corrected" = "blue")) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5))
```

```
## Warning: Removed 24 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

```
## Warning: Removed 327 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <f0>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <9f>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <88>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <f0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <9f>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <93>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  Cocoa Price Forecast - Hybrid (Prophet + ARIMA +
## GARCH)' in 'mbcsToSbcs': dot substituted for <88>
```



33. Forecasting Cocoa Prices with Uncertainty Bands

We now enhance our forecasting function to include 95% confidence intervals by: - Extracting upper/lower bounds from Prophet and ARIMA - Converting them to approximate standard errors - Combining with GARCH volatility to model overall uncertainty - Returning forecast mean and confidence bands for interpretation and plotting

The result is a risk-adjusted, full-confidence forecast.

Updated Forecast Function (with Error Bands)

```
forecast_cocoa_prices <- function(df_prophet, prophet_model, arima_resid, garch_fit, correction_model, r
  library(dplyr)
  library(prophet)
  library(forecast)
  library(rugarch)

  # Get last known climate values with index at which.max(df_prophet$ds)
  last_climate_vals <- df_prophet[which.max(df_prophet$ds), c("PRCP", "TAVG", "TMAX", "TMIN")]

  # Build a rough dataframe for forecasting
  # Create future dataframe, by default, if nothing passes in as n_ahead,
  # will be assumed as 1 year ahead (n = 12 by default)
```

```
  future_df <- make_future_dataframe(prophet_model, periods = n_ahead, freq = "month")
  future_df <- future_df %>%
    left_join(df_prophet %>% select(ds, PRCP, TAVG, TMAX, TMIN), by = "ds") %>%
    mutate(
      PRCP = coalesce(PRCP, last_climate_vals$PRCP),
      TAVG = coalesce(TAVG, last_climate_vals$TAVG),
      TMAX = coalesce(TMAX, last_climate_vals$TMAX),
      TMIN = coalesce(TMIN, last_climate_vals$TMIN)
    )

  # Forecast with Prophet
  forecast_prophet <- predict(prophet_model, future_df)
  fc_proph_upper = tail(forecast_prophet$yhat_upper, n_ahead)
  fc_proph_lower = tail(forecast_prophet$yhat_lower, n_ahead)
  se_prophet = (fc_proph_upper - fc_proph_lower) / 2 * qnorm(0.975)

  # Forecast ARIMA residuals
  fc_resid <- forecast(arima_resid, h = n_ahead, level = 95)
  se_resid = (fc_resid$upper[, "95%"] - fc_resid$lower[, "95%"]) / 2 * qnorm(0.975)

  # Forecast GARCH volatility
  garch_forecast <- ugarchforecast(garch_fit, n.ahead = n_ahead)
  vol_forecast <- sigma(garch_forecast)

  # Get full model forecast standard error
  se_mean_comp = sqrt(se_resid^2 + se_prophet^2)
  se_forecast = sqrt(se_mean_comp^2 + vol_forecast)

  # Combine all components
  forecast_result <- forecast_prophet %>%
    tail(n_ahead) %>%
    mutate(
      hybrid = yhat + fc_resid$mean,
      correction = predict(correction_model, newdata = data.frame(vol_series = as.numeric(vol_forecast))
      hybrid_corrected = hybrid + correction,
      se_forecast = as.numeric(se_forecast),
      upper95 = hybrid_corrected + qnorm(0.975) * se_forecast,
      lower95 = hybrid_corrected - qnorm(0.975) * se_forecast,
    )

  # Return the final model as a dataframe with 3 columns, defined as:
  # timestamp (ds),
  # ARIMA + Prophet forecast (hybrid),
  # ARIMA + Prophet forecast + GARCH correction (hybrid_corrected)
  return(forecast_result %>% select(ds, hybrid, hybrid_corrected, se_forecast, upper95, lower95))
}
```

```
### Forecasting Cocoa Prices for the Next 2 Years (24 Months)
forecast_output <- forecast_cocoa_prices(
  df_prophet = df_prophet,
  prophet_model = prophet_model,
  arima_resid = arima_resid,
  garch_fit = garch_fit,
```

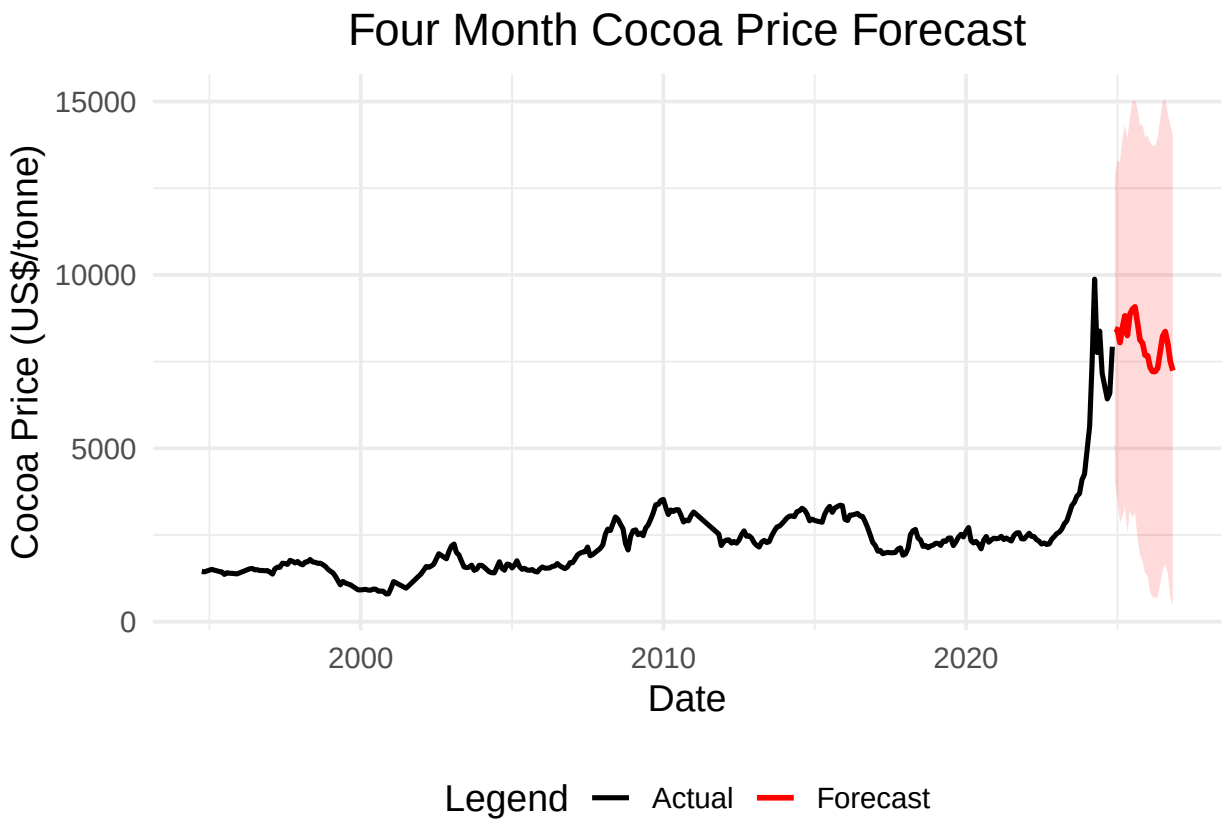
```
  correction_model = correction_model,
  n_ahead = 24
)

head(forecast_output)
```

```
##             ds   hybrid hybrid_corrected se_forecast  upper95  lower95
## 328 2024-12-01 6455.301         8448.470    2243.353 12845.36 4051.578
## 329 2025-01-01 6431.254         8418.578    2495.710 13310.08 3527.077
## 330 2025-02-01 6066.593         8048.123    2659.807 13261.25 2834.997
## 331 2025-03-01 6480.761         8456.546    2772.850 13891.23 3021.860
## 332 2025-04-01 6851.504         8821.592    2809.844 14328.79 3314.399
## 333 2025-05-01 6286.656         8251.098    2898.676 13932.40 2569.797
```

```
# ---------------------------------------
# Output the forecast into plots
# ---------------------------------------
# Combine forecast with historical data
df_plot_extended <- df_prophet %>%
  select(ds, Actual = y) %>%
  bind_rows(
    forecast_output %>%
      select(ds,
             Hybrid_Corrected = hybrid_corrected,
             upper95 = upper95,
             lower95 = lower95) %>%
      mutate(Actual = NA)  # placeholder for actual future values
  )

# Plot
ggplot(df_plot_extended, aes(x = ds)) +
  geom_line(aes(y = Actual, color = "Actual"), size = 0.9) +
  geom_line(aes(y = Hybrid_Corrected, color = "Forecast"), size = 1, linetype = "solid") +
  geom_ribbon(aes(ymin = lower95, ymax = upper95), alpha = 0.15, fill = "red") +
  labs(
    title = "Four Month Cocoa Price Forecast",
    x = "Date",
    y = "Cocoa Price (US$/tonne)",
    color = "Legend"
  ) +
  scale_color_manual(values = c("Actual" = "black", "Forecast" = "red")) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5))
```

```
## Warning: Removed 24 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

```
## Warning: Removed 327 rows containing missing values or values outside the scale range
## (`geom_line()`).
```




```
# ----------------------------------------
# Four month forecast of data against actual values
# ----------------------------------------
# Note that this requires refitting the data with a train set, and testing against the test set. That i

#forecast_comparison = df_prophet_complete$y[324:327]
# Test set against actual values (close up)
#ggplot(forecast_output, aes(x = ds)) +
#  geom_line(aes(y = hybrid_corrected, color = "Forecast"), size = 1, linetype = "solid") +
#  geom_ribbon(aes(ymin = lower95, ymax = upper95), alpha = 0.15, fill = "red") +
#  geom_line(aes(y = forecast_comparison, color = "Actual"), size = 0.9, linetype = "solid") +
#  labs(
#    title = "Four Month Cocoa Price Forecast",
#   x = "Date",
#    y = "Cocoa Price (US$/tonne)",
#    color = "Legend"
#  ) +
#  scale_color_manual(values = c("Actual" = "black", "Forecast" = "red")) +
#  theme_minimal(base_size = 14) +
#  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5))

# Present results
#four_month_rmse = rmse(forecast_comparison, forecast_output$hybrid_corrected)
#four_month_mae = mae(forecast_comparison, forecast_output$hybrid_corrected)
#four_month_mape = mape(forecast_comparison, forecast_output$hybrid_corrected)
```

```
#four_month_assessment = data.frame(
#  RMSE = four_month_rmse,
#  MAE = four_month_mae,
#  MAPE = four_month_mape
#)

#knitr::kable(four_month_assessment)
```

```
# ---------------------------------------
# Four month forecast summary table
# ---------------------------------------
#forecasted_points = data.frame(
#  Date = as.Date(df_plot_extended$ds[328:331]),
#  Forecast = df_plot_extended$Hybrid_Corrected[328:331],
#  Upper = df_plot_extended$upper95[328:331],
#  Lower = df_plot_extended$lower95[328:331]
#)

#knitr::kable(forecasted_points)
```