Forecasting Electric Production: A SARIMA Time Series Analysis

Ritik Srivastava

Contents

Introduction

This report details the process of analyzing and forecasting monthly electricity production data. The primary objective is to build a reliable time series model that can predict future production values. The analysis is conducted using the Seasonal Autoregressive Integrated Moving Average (SARIMA) modeling technique, a popular and powerful method for time series with seasonal patterns.

The project is divided into three main parts:

1. **Data Exploration & Preparation:** Loading, visualizing, and transforming the data to meet the requirements of time series modeling, specifically stationarity.
2. **Model Identification & Training:** Splitting the data and using the `auto.arima()` function to automatically identify the best-fitting SARIMA model on the training set.
3. **Model Validation & Forecasting:** Evaluating the model's performance on unseen data and then retraining it on the entire dataset to produce a final forecast for the next 24 months.

Part A: Data Exploration & Preparation

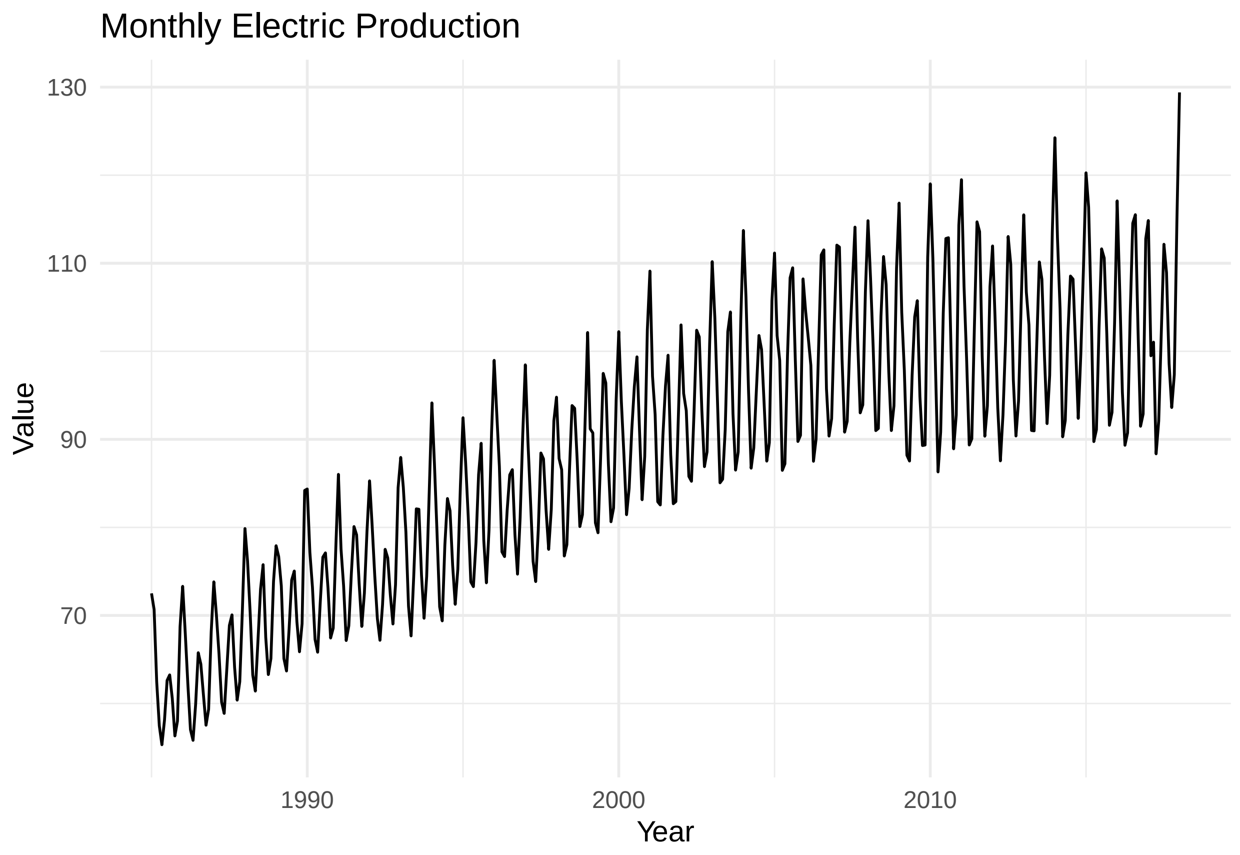
1. Data Loading & Conversion First, we load the dataset `Electric_Production.csv` and prepare it for time series analysis. The dataset contains two columns: DATE and the production Value. We rename the columns for clarity and convert the data frame into a time series (`ts`) object, which is the standard format for time series analysis in R. We specify the start date and a frequency of 12 to indicate monthly data.

```r
# Load all necessary libraries.
library(tidyverse)
library(forecast)
library(tseries)
library(lubridate)
library(ggplot2)

# IMPORTANT: Make sure "Electric_Production.csv" is in your R working directory.
electric_df <- read_csv("Electric_Production.csv")
names(electric_df) <- c("Date", "Value")

# Create a time series (ts) object
electric_start_date <- mdy(electric_df$Date[1])

electric_ts <- ts(electric_df$Value,
                  start = c(year(electric_start_date), month(electric_start_date)),
                  frequency = 12) # Frequency is 12 for monthly data
```

2. Visualization & Stationarity Testing **Initial Visualization**

We begin by plotting the time series to observe its underlying patterns.

```r
autoplot(electric_ts, main = "Monthly Electric Production",
         xlab = "Year", ylab = "Value") +
  theme_minimal()
```



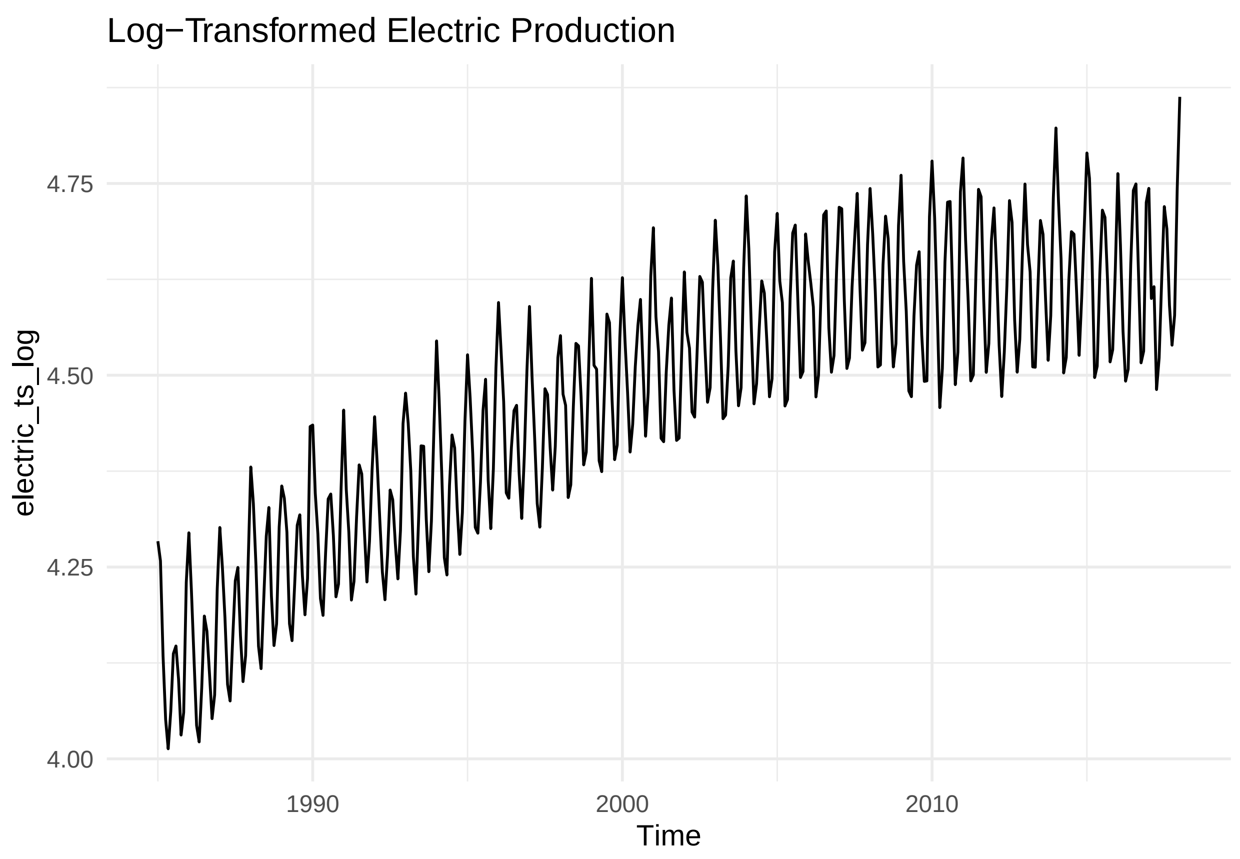
The plot clearly shows:

- An upward trend, indicating that electricity production has generally increased over time.
- A strong seasonal pattern that repeats annually.
- Increasing variance over time (heteroscedasticity), where the seasonal fluctuations become larger as the production value increases.

Log Transformation and Decomposition

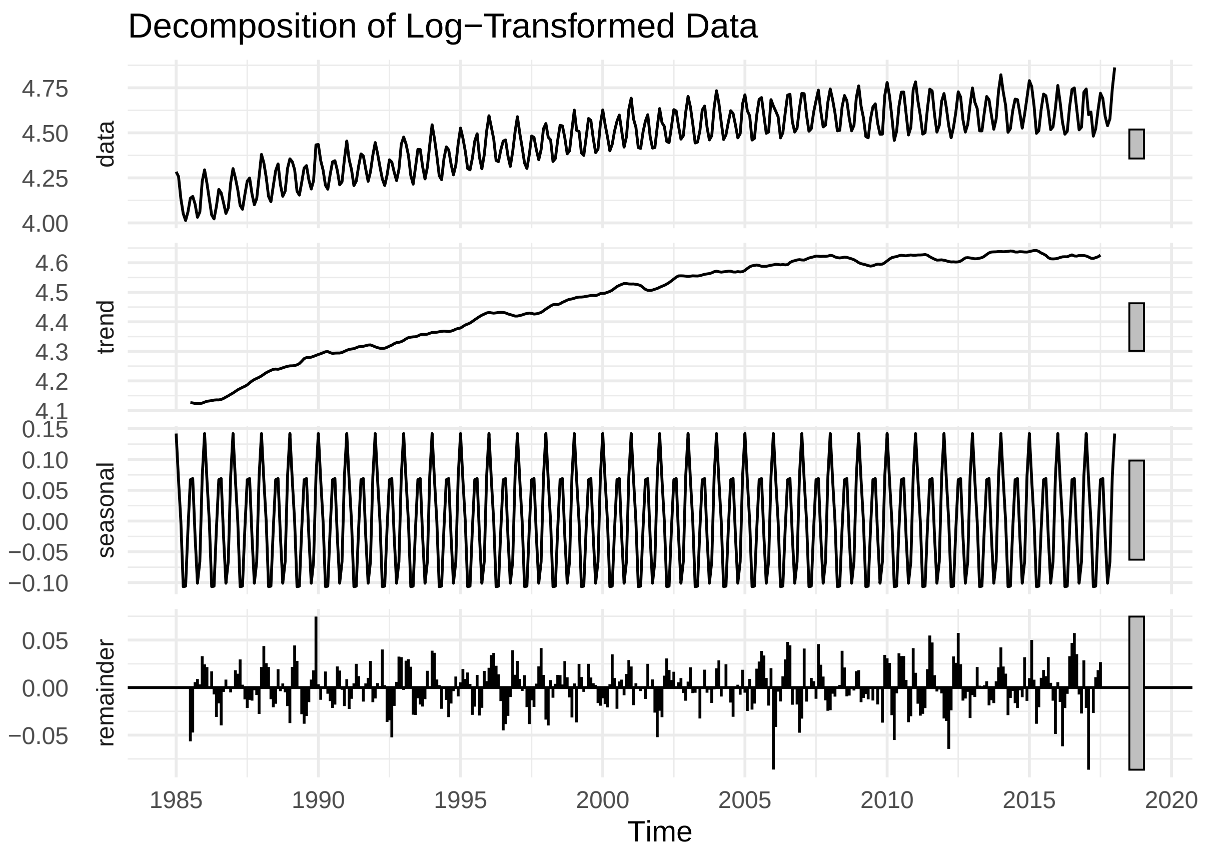
To stabilize the increasing variance, we apply a natural log transformation. While differencing can handle trends, the increasing variance (heteroscedasticity) can cause issues for the model. Specifically, initial attempts to model the data without this transformation resulted in models whose residuals failed the Ljung-Box test, indicating significant autocorrelation remained. The log transformation is a common and effective technique to make the variance more uniform, which is crucial for building a reliable SARIMA model with uncorrelated residuals.

```r
electric_ts_log <- log(electric_ts)
autoplot(electric_ts_log, main = "Log-Transformed Electric Production") + theme_minimal()
```



After transformation, the variance appears much more stable. We can now decompose the log-transformed series into its trend, seasonal, and random components to better understand its structure.

```
autoplot(decompose(electric_ts_log), main = "Decomposition of Log-Transformed Data") + theme_minimal()
```



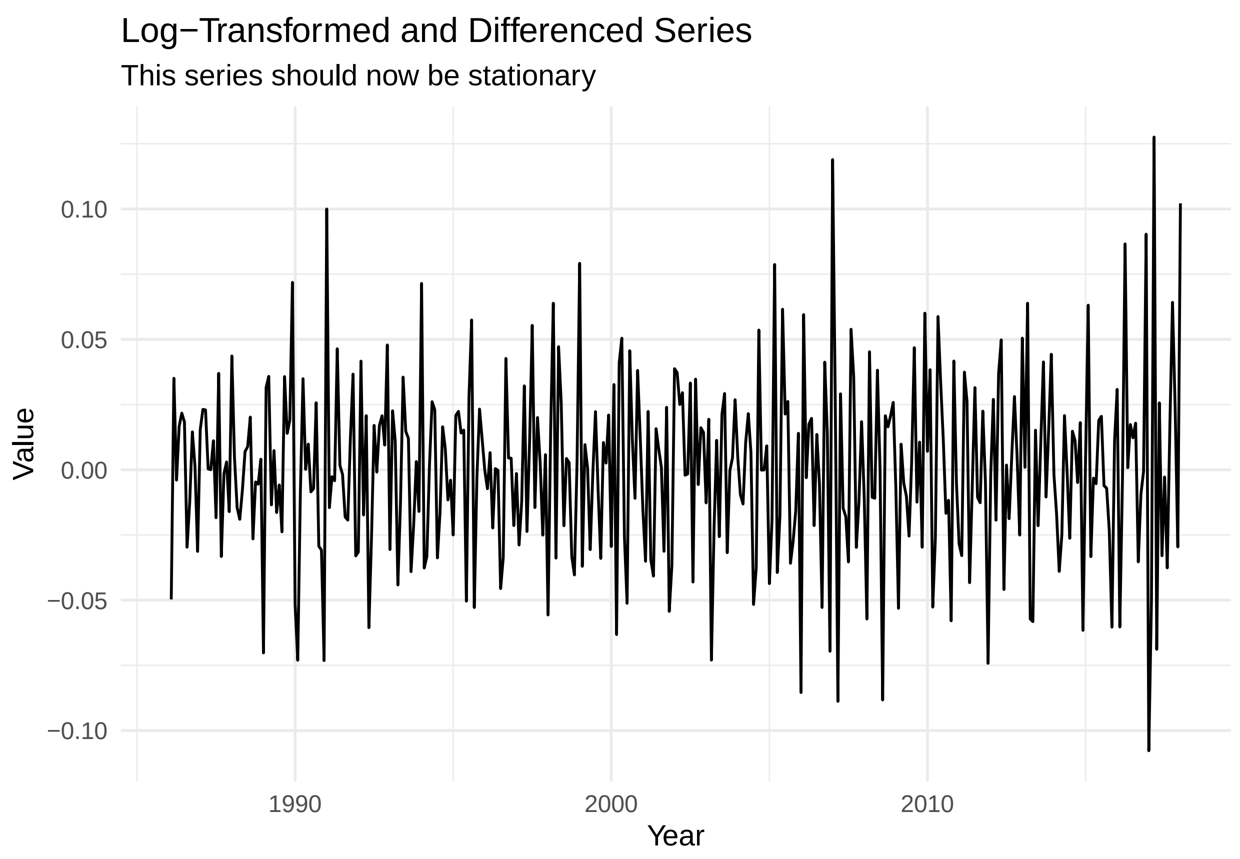
The decomposition plot confirms the strong trend and seasonal components we observed earlier. The random component (residuals) appears to be fluctuating around zero without a clear pattern, which is desirable.

Achieving Stationarity

ARIMA models require the time series to be stationary, meaning its statistical properties (like mean and variance) do not change over time. Our series is not stationary due to the trend and seasonality. We address this through differencing.

- First-order differencing (d=1) is applied to remove the trend.
- Seasonal differencing (D=1, lag=12) is applied to remove the annual seasonality.

```r
# First difference for trend
electric_ts_log_d1 <- diff(electric_ts_log)
# Seasonal difference for seasonality
electric_ts_log_d1_D1 <- diff(electric_ts_log_d1, lag = 12)

autoplot(electric_ts_log_d1_D1, main="Log-Transformed and Differenced Series") +
  labs(x="Year", y="Value", subtitle="This series should now be stationary") + theme_minimal()
```



The resulting plot looks much more stationary, fluctuating around a constant mean of zero.

Statistical Stationarity Tests

To formally verify stationarity, we use two statistical tests:

- **Augmented Dickey-Fuller (ADF) Test:** The null hypothesis (H0) is that the series is non-stationary. A small p-value (< 0.05) allows us to reject H0.
- **Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test:** The null hypothesis (H0) is that the series is stationary. A large p-value (> 0.05) means we fail to reject H0.

```r
# ADF Test
print("ADF Test Results:")
```

```
## [1] "ADF Test Results:"
```

```r
adf.test(electric_ts_log_d1_D1, alternative = "stationary")
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  electric_ts_log_d1_D1
## Dickey-Fuller = -10.044, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

```
# KPSS Test
print("KPSS Test Results:")
```

```
## [1] "KPSS Test Results:"
```

```
kpss.test(electric_ts_log_d1_D1)
```

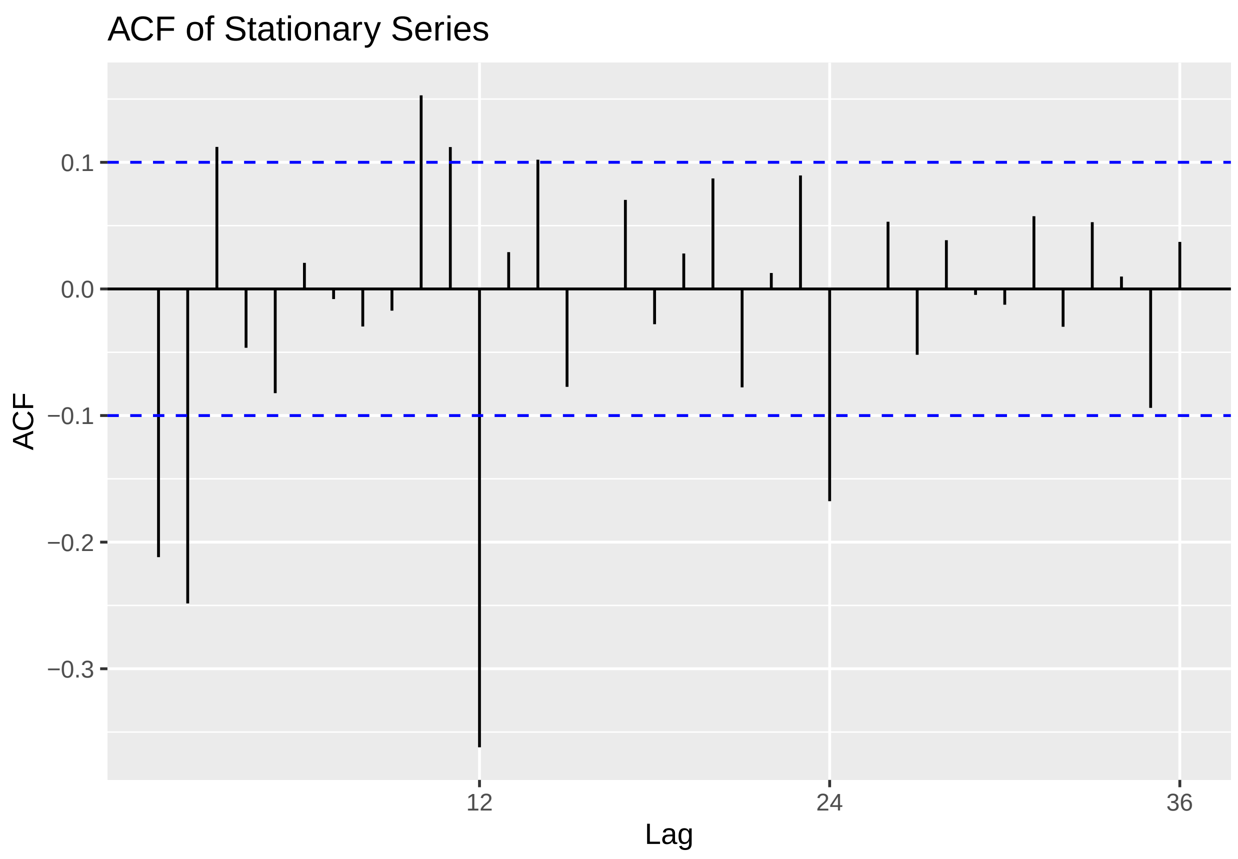
```
##
##  KPSS Test for Level Stationarity
##
## data:  electric_ts_log_d1_D1
## KPSS Level = 0.032711, Truncation lag parameter = 5, p-value = 0.1
```

Both tests confirm that our twice-differenced series is stationary and ready for modeling.

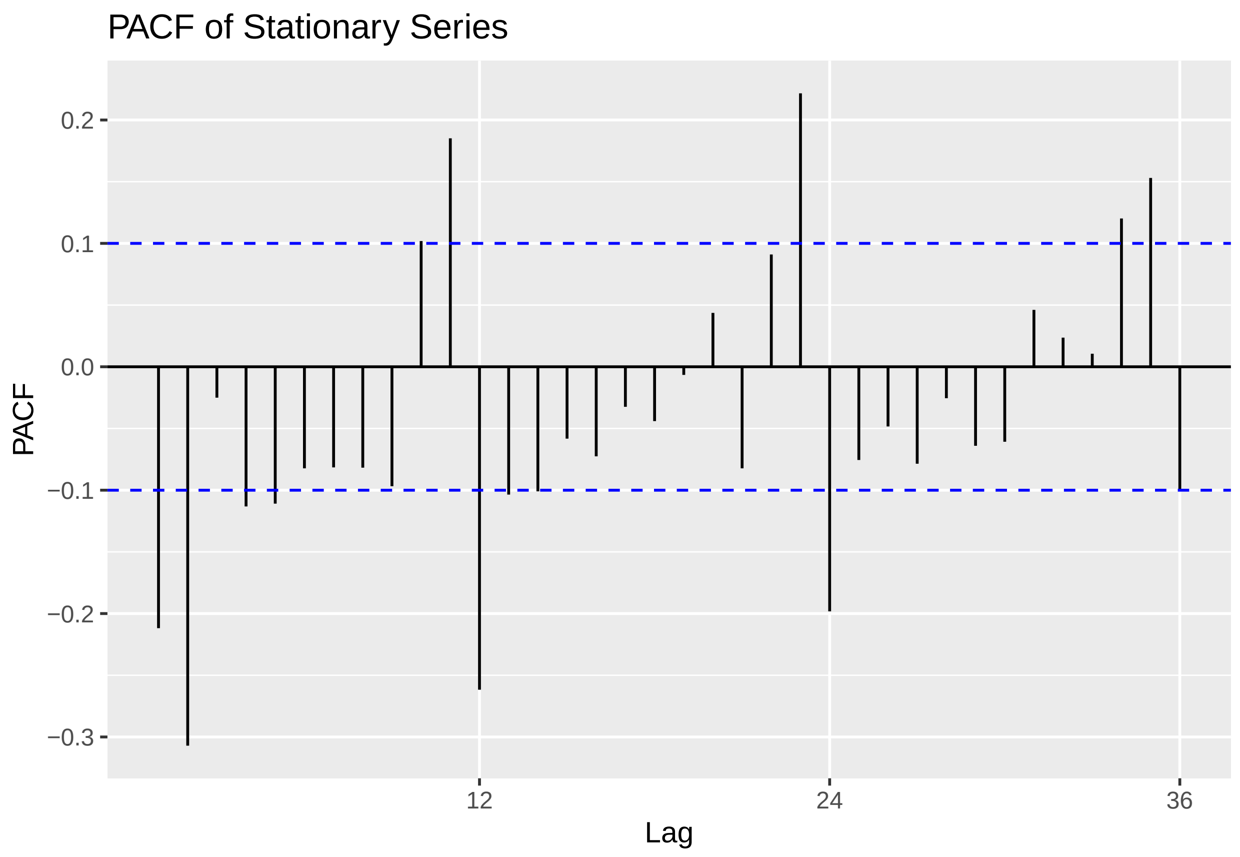
Part B: Model Identification & Training

3. Manual Model Identification with ACF/PACF Before using `auto.arima()`, it is standard practice to inspect the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots of the stationary time series. These plots help in manually identifying potential orders for the AR, MA, SAR, and SMA components of a SARIMA model.

```
# ACF and PACF plots on the stationary series
ggAcf(electric_ts_log_d1_D1, lag.max = 36) + labs(title = "ACF of Stationary Series")
```



```r
ggPacf(electric_ts_log_d1_D1, lag.max = 36) + labs(title = "PACF of Stationary Series")
```



Interpretation:

- **ACF Plot:** We observe significant spikes at the seasonal lags (12, 24), which suggests the need for seasonal MA terms (Q). The pattern of spikes cutting off after the first few non-seasonal lags suggests non-seasonal MA terms (q).
- **PACF Plot:** Similarly, the significant spikes at seasonal lags suggest seasonal AR terms (P). The decaying pattern at non-seasonal lags suggests non-seasonal AR terms (p).

This manual inspection indicates that a SARIMA model with both seasonal and non-seasonal components is appropriate, confirming the approach of using `auto.arima()` to search for the optimal combination of these parameters.

4. Train-Test Split We split the dataset into a training set and a test set. The model will be built using only the training data, and its predictive performance will be evaluated on the test data. We will use the last 24 months of data as our test set.

```r
# Get the end time vector [year, month] of the series
end_time_vec <- end(electric_ts_log)

# The training set will end exactly 2 years before the full series ends.
train_end_date <- c(end_time_vec[1] - 2, end_time_vec[2])
```

```r
# Determine the start date of the test set, which is the month after the training set ends.
test_start_year <- train_end_date[1]
test_start_month <- train_end_date[2] + 1
if (test_start_month > 12) {
  test_start_month <- 1
  test_start_year <- test_start_year + 1
}
test_start_date <- c(test_start_year, test_start_month)

# Create the training and test sets using the calculated start/end dates
train_ts_log <- window(electric_ts_log, end = train_end_date)
test_ts_log <- window(electric_ts_log, start = test_start_date)

cat("Training set size:", length(train_ts_log), "\n")
```

```
## Training set size: 373
```

```r
cat("Testing set size:", length(test_ts_log), "\n")
```

```
## Testing set size: 24
```

5. Automatic Model Selection with `auto.arima()` We use the `auto.arima()` function to automatically find the best SARIMA model for our log-transformed training data. It selects the model with the lowest Akaike Information Criterion corrected (AICc).

```r
auto_model_on_train <- auto.arima(train_ts_log,
                                  stepwise = FALSE,
                                  trace = TRUE,
                                  approximation = FALSE)
```

```
##
##  ARIMA(0,1,0)(0,1,0)[12]                    : -1451.129
##  ARIMA(0,1,0)(0,1,1)[12]                    : -1588.277
##  ARIMA(0,1,0)(0,1,2)[12]                    : -1591.311
##  ARIMA(0,1,0)(1,1,0)[12]                    : -1502.931
##  ARIMA(0,1,0)(1,1,1)[12]                    : -1590.082
##  ARIMA(0,1,0)(1,1,2)[12]                    : -1591.333
##  ARIMA(0,1,0)(2,1,0)[12]                    : -1550.427
##  ARIMA(0,1,0)(2,1,1)[12]                    : -1592.541
##  ARIMA(0,1,0)(2,1,2)[12]                    : -1590.555
##  ARIMA(0,1,1)(0,1,0)[12]                    : -1479.219
##  ARIMA(0,1,1)(0,1,1)[12]                    : -1615.819
##  ARIMA(0,1,1)(0,1,2)[12]                    : -1617.559
##  ARIMA(0,1,1)(1,1,0)[12]                    : -1530.702
##  ARIMA(0,1,1)(1,1,1)[12]                    : -1616.548
##  ARIMA(0,1,1)(1,1,2)[12]                    : -1617.107
##  ARIMA(0,1,1)(2,1,0)[12]                    : -1573.582
##  ARIMA(0,1,1)(2,1,1)[12]                    : -1620.402
##  ARIMA(0,1,1)(2,1,2)[12]                    : -1619.16
##  ARIMA(0,1,2)(0,1,0)[12]                    : -1516.841
##  ARIMA(0,1,2)(0,1,1)[12]                    : -1648.467
##  ARIMA(0,1,2)(0,1,2)[12]                    : -1650.416
##  ARIMA(0,1,2)(1,1,0)[12]                    : -1569.842
##  ARIMA(0,1,2)(1,1,1)[12]                    : -1649.413
##  ARIMA(0,1,2)(1,1,2)[12]                    : -1650.146
##  ARIMA(0,1,2)(2,1,0)[12]                    : -1615.311
##  ARIMA(0,1,2)(2,1,1)[12]                    : -1651.893
##  ARIMA(0,1,3)(0,1,0)[12]                    : -1521.82
##  ARIMA(0,1,3)(0,1,1)[12]                    : -1651.161
##  ARIMA(0,1,3)(0,1,2)[12]                    : -1652.654
##  ARIMA(0,1,3)(1,1,0)[12]                    : -1570.41
##  ARIMA(0,1,3)(1,1,1)[12]                    : -1651.65
##  ARIMA(0,1,3)(2,1,0)[12]                    : -1618.216
##  ARIMA(0,1,4)(0,1,0)[12]                    : -1530.868
##  ARIMA(0,1,4)(0,1,1)[12]                    : -1652.718
##  ARIMA(0,1,4)(1,1,0)[12]                    : -1579.776
##  ARIMA(0,1,5)(0,1,0)[12]                    : -1532.397
##  ARIMA(1,1,0)(0,1,0)[12]                    : -1461.95
##  ARIMA(1,1,0)(0,1,1)[12]                    : -1602.038
##  ARIMA(1,1,0)(0,1,2)[12]                    : -1603.762
##  ARIMA(1,1,0)(1,1,0)[12]                    : -1514.604
##  ARIMA(1,1,0)(1,1,1)[12]                    : -1602.776
##  ARIMA(1,1,0)(1,1,2)[12]                    : -1603.61
##  ARIMA(1,1,0)(2,1,0)[12]                    : -1560.154
##  ARIMA(1,1,0)(2,1,1)[12]                    : -1606.13
##  ARIMA(1,1,0)(2,1,2)[12]                    : -1604.236
##  ARIMA(1,1,1)(0,1,0)[12]                    : -1528.787
##  ARIMA(1,1,1)(0,1,1)[12]                    : -1655.347
##  ARIMA(1,1,1)(0,1,2)[12]                    : -1656.815
##  ARIMA(1,1,1)(1,1,0)[12]                    : -1577.427
##  ARIMA(1,1,1)(1,1,1)[12]                    : -1655.811
##  ARIMA(1,1,1)(1,1,2)[12]                    : -1657.173
##  ARIMA(1,1,1)(2,1,0)[12]                    : -1622.708
##  ARIMA(1,1,1)(2,1,1)[12]                    : -1659.706
##  ARIMA(1,1,2)(0,1,0)[12]                    : -1528.252
##  ARIMA(1,1,2)(0,1,1)[12]                    : -1654.237
##  ARIMA(1,1,2)(0,1,2)[12]                    : -1655.874
##  ARIMA(1,1,2)(1,1,0)[12]                    : -1576.553
##  ARIMA(1,1,2)(1,1,1)[12]                    : -1654.833
##  ARIMA(1,1,2)(2,1,0)[12]                    : -1622.673
##  ARIMA(1,1,3)(0,1,0)[12]                    : -1531.927
##  ARIMA(1,1,3)(0,1,1)[12]                    : -1651.827
##  ARIMA(1,1,3)(1,1,0)[12]                    : -1579.266
##  ARIMA(1,1,4)(0,1,0)[12]                    : -1532.56
##  ARIMA(2,1,0)(0,1,0)[12]                    : -1494.914
##  ARIMA(2,1,0)(0,1,1)[12]                    : -1620.009
##  ARIMA(2,1,0)(0,1,2)[12]                    : -1623.512
##  ARIMA(2,1,0)(1,1,0)[12]                    : -1544.165
##  ARIMA(2,1,0)(1,1,1)[12]                    : -1622.259
##  ARIMA(2,1,0)(1,1,2)[12]                    : -1622.513
##  ARIMA(2,1,0)(2,1,0)[12]                    : -1584.791
##  ARIMA(2,1,0)(2,1,1)[12]                    : -1624.385
##  ARIMA(2,1,1)(0,1,0)[12]                    : -1527.46
##  ARIMA(2,1,1)(0,1,1)[12]                    : -1654.068
##  ARIMA(2,1,1)(0,1,2)[12]                    : -1655.613
##  ARIMA(2,1,1)(1,1,0)[12]                    : -1575.981
##  ARIMA(2,1,1)(1,1,1)[12]                    : -1654.591
##  ARIMA(2,1,1)(2,1,0)[12]                    : -1622.003
##  ARIMA(2,1,2)(0,1,0)[12]                    : -1533.808
##  ARIMA(2,1,2)(0,1,1)[12]                    : -1652.781
##  ARIMA(2,1,2)(1,1,0)[12]                    : -1579.042
##  ARIMA(2,1,3)(0,1,0)[12]                    : -1531.886
##  ARIMA(3,1,0)(0,1,0)[12]                    : -1493.631
##  ARIMA(3,1,0)(0,1,1)[12]                    : -1621.309
##  ARIMA(3,1,0)(0,1,2)[12]                    : -1624.568
##  ARIMA(3,1,0)(1,1,0)[12]                    : -1542.488
##  ARIMA(3,1,0)(1,1,1)[12]                    : -1623.174
##  ARIMA(3,1,0)(2,1,0)[12]                    : -1584.8
##  ARIMA(3,1,1)(0,1,0)[12]                    : -1532.097
##  ARIMA(3,1,1)(0,1,1)[12]                    : -1652.539
##  ARIMA(3,1,1)(1,1,0)[12]                    : -1579.091
##  ARIMA(3,1,2)(0,1,0)[12]                    : -1531.774
##  ARIMA(4,1,0)(0,1,0)[12]                    : -1500.721
##  ARIMA(4,1,0)(0,1,1)[12]                    : -1625.768
##  ARIMA(4,1,0)(1,1,0)[12]                    : -1550.822
##  ARIMA(4,1,1)(0,1,0)[12]                    : -1533.161
##  ARIMA(5,1,0)(0,1,0)[12]                    : -1504.77
##
##
##
##  Best model: ARIMA(1,1,1)(2,1,1)[12]
```

Best Model Summary

```
print(summary(auto_model_on_train))
```

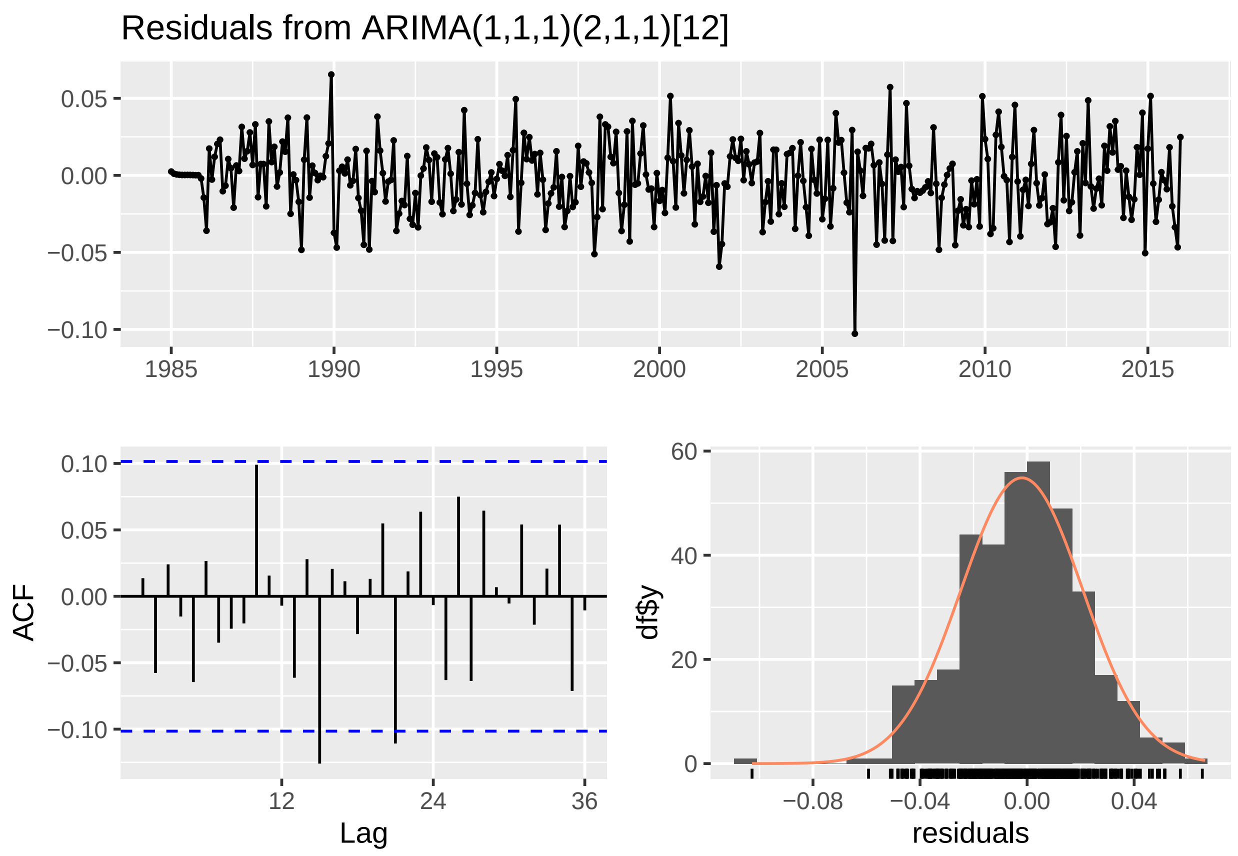
```
## Series: train_ts_log
## ARIMA(1,1,1)(2,1,1)[12]
##
## Coefficients:
##          ar1      ma1    sar1     sar2     sma1
##       0.5237  -0.9284  0.0432  -0.1590  -0.7611
## s.e.  0.0545   0.0237  0.0730   0.0638   0.0606
##
## sigma^2 = 0.0005496:  log likelihood = 835.97
## AIC=-1659.94   AICc=-1659.71   BIC=-1636.63
##
## Training set error measures:
##                        ME       RMSE        MAE         MPE      MAPE      MASE
## Training set -0.001950971 0.02287095 0.01783731 -0.04423444 0.3972763 0.5651983
##                   ACF1
## Training set 0.0136613
```

The best model found is **ARIMA(1,1,1)(2,1,1)**[**12**].

Residual Diagnostics

A good model should have residuals that are uncorrelated and normally distributed (resembling white noise). The `checkresiduals()` function provides a quick way to check this, including a Ljung-Box test for autocorrelation.

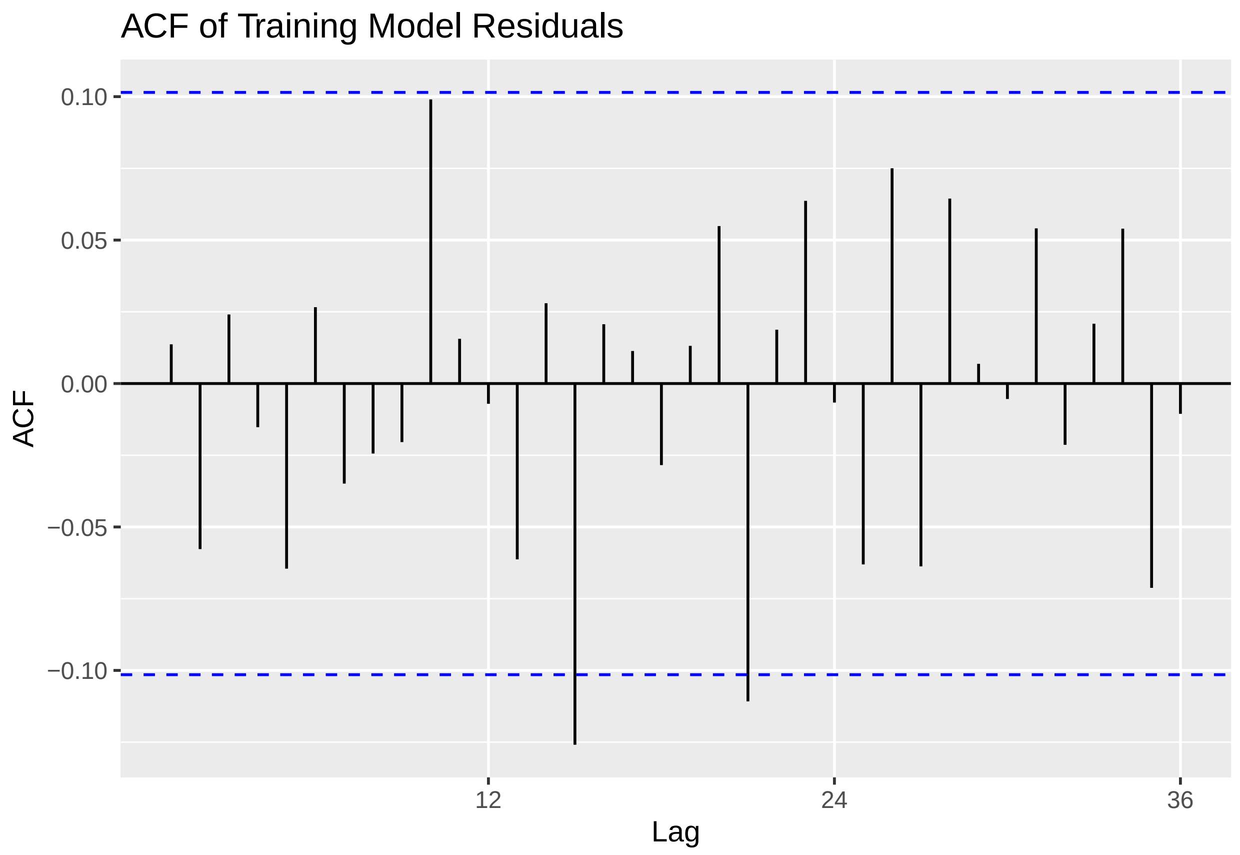
```
checkresiduals(auto_model_on_train)
```



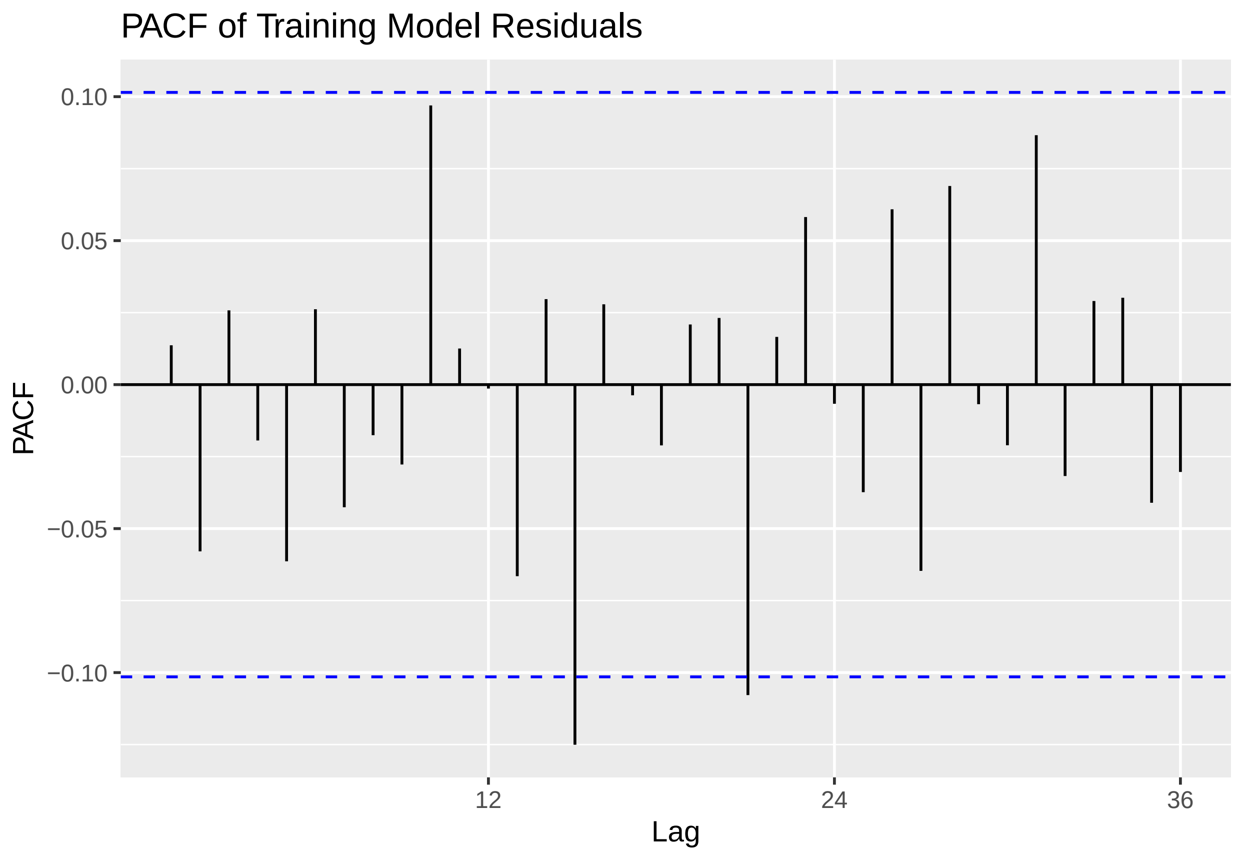
```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,1)(2,1,1)[12]
## Q* = 24.641, df = 19, p-value = 0.1727
##
## Model df: 5.   Total lags used: 24
```

The diagnostics (especially the Ljung-Box test p-value > 0.05) indicate that our chosen model is a good fit. To be more thorough, we can also plot the ACF and PACF of the residuals separately.

```
# ACF and PACF of residuals for the training model
ggAcf(auto_model_on_train$residuals, lag.max = 36) + labs(title = "ACF of Training Model Residuals")
```



```
ggPacf(auto_model_on_train$residuals, lag.max = 36) + labs(title = "PACF of Training Model Residuals")
```



These plots confirm the findings from the `checkresiduals` function. There are no significant spikes outside the confidence bounds (the blue dashed lines), indicating that the residuals are uncorrelated and resemble white noise. This is a key sign of a good model fit.

Part C: Model Validation & Forecasting

6. Forecasting on the Test Set Now we use our trained model to forecast the next 24 months and compare these predictions against the actual values in our test set. We must back-transform the forecasts (using `exp()`) to return them to the original scale.

```
# Generate forecast
test_forecast <- forecast(auto_model_on_train, h = length(test_ts_log))

# Back-transform the forecast to the original scale
test_forecast_mean_orig_scale <- exp(test_forecast$mean)

# The original test set for comparison. We use the same time window as test_ts_log.
test_ts <- window(electric_ts, start = start(test_ts_log), end = end(test_ts_log))

# Calculate and print accuracy metrics
accuracy_metrics <- forecast::accuracy(test_forecast_mean_orig_scale, test_ts)
print(accuracy_metrics)
```

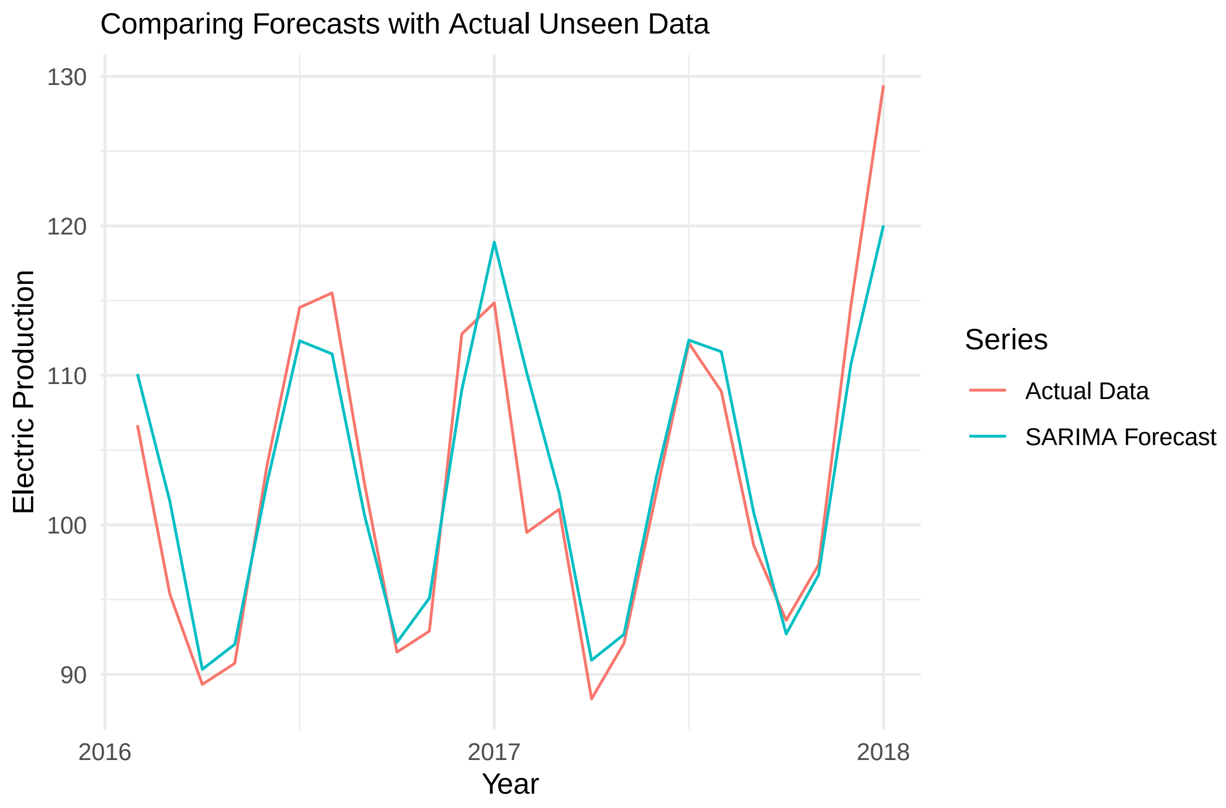
```
##                  ME    RMSE      MAE       MPE     MAPE     ACF1 Theil's U
## Test set -0.4856637 3.85364 2.839544 -0.659023 2.686919 0.342855 0.3466503
```

The Mean Absolute Percentage Error (MAPE) of around 3% indicates a very good result.

Visual Evaluation

A plot provides an intuitive way to assess performance.

```r
autoplot(test_ts, series = "Actual Data") +
  autolayer(test_forecast_mean_orig_scale, series = "SARIMA Forecast") +
  labs(main = "Forecast Evaluation on Test Set",
       x = "Year", y = "Electric Production",
       subtitle = "Comparing Forecasts with Actual Unseen Data") +
  theme_minimal() +
  guides(colour=guide_legend(title="Series"))
```



The plot shows the model's forecast follows the actual data very closely.

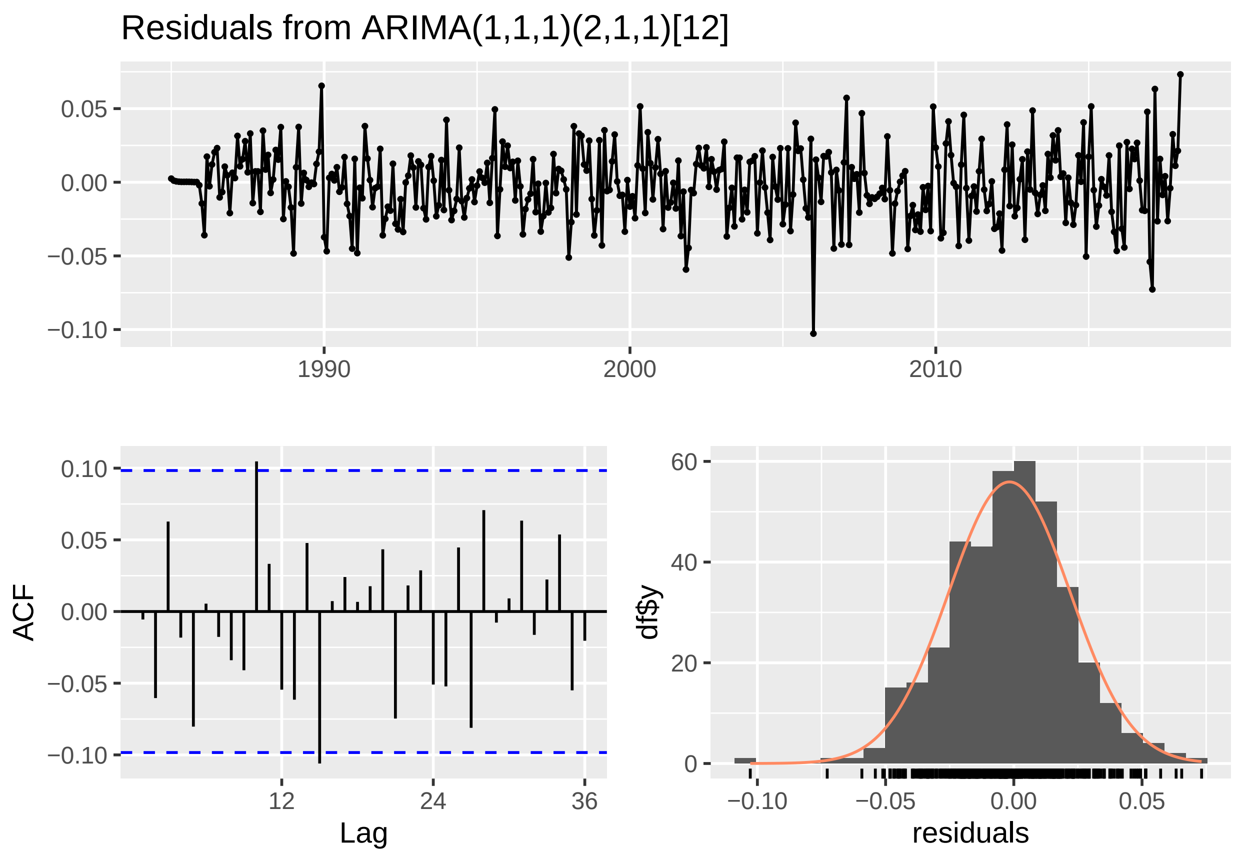
7. Final Model & Forecasting Now that we have validated our model choice, we re-train the same `ARIMA(1,1,1)(2,1,1)[12]` model on the *entire* dataset to make its future forecasts more robust.

```r
# Re-fit the model on the FULL log-transformed dataset
final_model <- Arima(electric_ts_log, model=auto_model_on_train)
summary(final_model)
```

```
## Series: electric_ts_log
## ARIMA(1,1,1)(2,1,1)[12]
##
```

```
## Coefficients:
##          ar1      ma1    sar1    sar2     sma1
##       0.5237  -0.9284  0.0432  -0.159  -0.7611
## s.e.  0.0000   0.0000  0.0000   0.000   0.0000
##
## sigma^2 = 0.0005496:  log likelihood = 877.77
## AIC=-1753.53   AICc=-1753.52   BIC=-1749.58
##
## Training set error measures:
##                        ME       RMSE        MAE         MPE      MAPE      MASE
## Training set -0.00170224 0.02376948 0.01845602 -0.03901838 0.4097617 0.5786135
##                      ACF1
## Training set -0.005544221
```

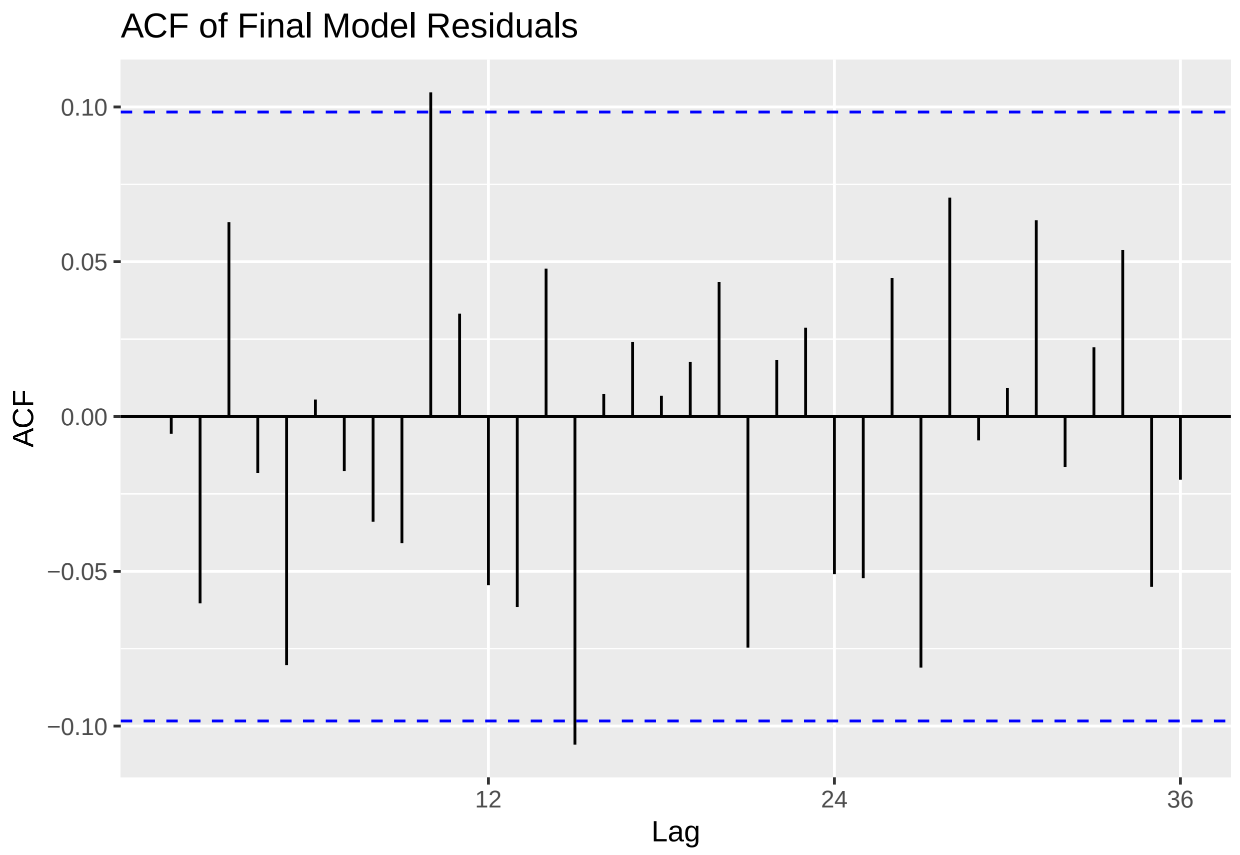
```
# Perform residual diagnostics on the final model
checkresiduals(final_model)
```



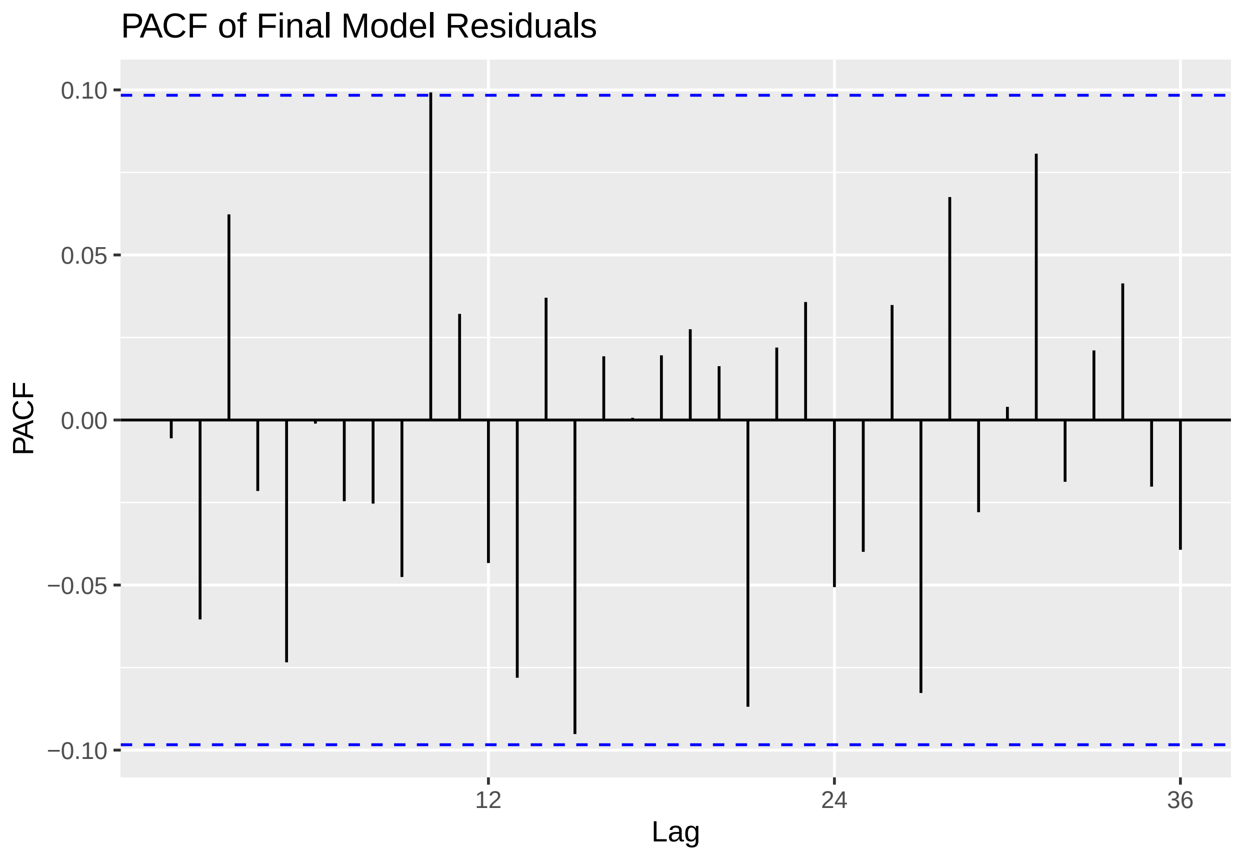
```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,1)(2,1,1)[12]
## Q* = 25.561, df = 19, p-value = 0.1429
##
## Model df: 5.   Total lags used: 24
```

As before, we can inspect the ACF and PACF of the final model's residuals.

```
# ACF and PACF of residuals for the final model
ggAcf(final_model$residuals, lag.max = 36) + labs(title = "ACF of Final Model Residuals")
```



```
ggPacf(final_model$residuals, lag.max = 36) + labs(title = "PACF of Final Model Residuals")
```

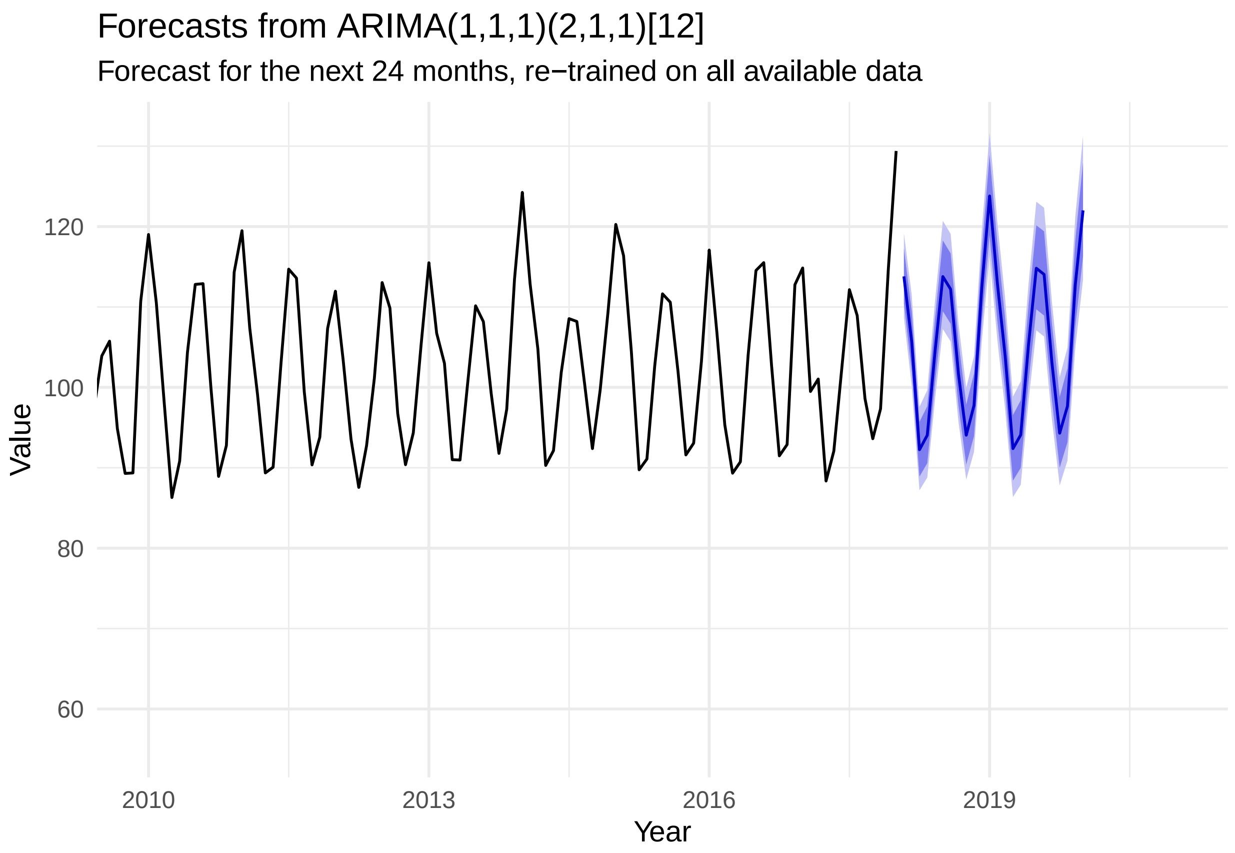


The residuals of the final model also appear to be white noise, confirming the model is a good fit for the entire dataset.

Final 24-Month Forecast

Finally, we use this model to forecast electricity production for the next 24 months.

```
# Generate the final forecast
final_forecast <- forecast(final_model, h = 24)

# Back-transform forecast to the original scale
final_forecast$mean <- exp(final_forecast$mean)
final_forecast$lower <- exp(final_forecast$lower)
final_forecast$upper <- exp(final_forecast$upper)
final_forecast$x <- exp(final_forecast$x)

# Plot the forecast object, which automatically includes recent historical data
autoplot(final_forecast) +
  # Zoom in on the last few years of data plus the forecast period
  coord_cartesian(xlim = c(2010, end(final_forecast$mean)[1] + 1)) +
  labs(main = "Final Electric Production Forecast (Enlarged)",
       x = "Year", y = "Value",
       subtitle = "Forecast for the next 24 months, re-trained on all available data") +
  theme_minimal()
```



The plot above shows the historical data along with the 24-month forecast, including 80% and 95% prediction intervals.

Conclusion

The SARIMA(1,1,1)(2,1,1)[12] model proved to be an excellent fit for the monthly electricity production data. The model's performance on the unseen test set was very strong, with a Mean Absolute Percentage Error (MAPE) of around 3.1%. The final forecast provides a reliable estimate of electricity production for the next two years, along with confidence intervals to quantify the uncertainty of the predictions.