Implementation of regARIMA Procedure Using R

Rohit Jangid

2024-11-01

Contents

Introduction

This document presents an implementation of the `regARIMA` procedure from the X13-ARIMA-SEATS software using R. The `regARIMA` procedure is utilized for seasonal adjustment and modeling of time series data. The methodology involves several steps, including data preparation, model fitting, outlier detection, and model selection based on information criteria.

Loading Required Libraries

We begin by loading the necessary R libraries for data manipulation, visualization, and time series analysis.

```
library(ggplot2)
library(gridExtra)
library(forecast)
library(tseries)
library(seasonal)
```

Helper Functions

The implementation relies on several helper functions to create regressors, evaluate model selection criteria, plot model outputs, and detect outliers. Below, we define these functions with detailed explanations.

Creating Regressor for Trading Days

The `td1nolpyear_regressor` function generates a regressor based on the number of trading days and weekends for each month in the time series.

```
td1nolpyear_regressor <- function(start = c(2023, 1), n) {
    # Initialize an empty vector to store the regressor values
    regressor <- numeric(n)

    # Loop through each month from the start date for n months
    for (i in 0:(n - 1)) {
        # Calculate the year and month for the current iteration
        year <- start[1] + (start[2] + i - 1) %/% 12
        month <- (start[2] + i - 1) %% 12 + 1

        # Start and end dates for the current month
        start_date <- as.Date(paste(year, sprintf("%02d", month), "01", sep = "-"))
        end_date <- seq(start_date, by = "month", length.out = 2)[2] - 1

        # Get all dates in the current month
        dates <- seq(start_date, end_date, by = "day")

        # Count weekdays and weekends (Saturday and Sunday)
        weekdays_count <- sum(!weekdays(dates) %in% c("Saturday", "Sunday"))
        weekends_count <- sum(weekdays(dates) %in% c("Saturday", "Sunday"))

        # Calculate the regressor for the month
        regressor[i + 1] <- weekdays_count - (5 / 2) * weekends_count
    }

    return(regressor)
}
```

Selection Criteria for Model Evaluation

The `selection_criteria` function calculates various information criteria (AIC, BIC, AICc, HQ) to evaluate and compare different ARIMA models.

```
selection_criteria <- function(model, adjust_for_log_transform = FALSE) {
    # Extract log-likelihood from the model
```

```r
    loglik <- as.numeric(logLik(model))

    # Extract original series and frequency from model
    original_series <- model$x
    freq <- frequency(model$x)

    # Extract non-seasonal and seasonal orders from model$call
    arima_order <- eval(model$call$order)
    seasonal_order <- eval(model$call$seasonal)

    # Determine the number of regressors
    number_of_regressors <- dim(eval(model$call$xreg))[2]
    if (is.null(number_of_regressors)) {
        number_of_regressors <- 1
    }

    # Non-seasonal and seasonal differencing
    order_diff <- arima_order[2]
    seasonal_diff <- seasonal_order[2]

    # Calculate the effective number of observations
    effective_n <- length(original_series) - order_diff - freq * seasonal_diff

    if (adjust_for_log_transform) {
        loglik <- loglik - sum(original_series[(length(original_series) -
                                          effective_n + 1):length(original_series)])
    }

    # Calculate the number of parameters (including the variance term)
    n_p <- number_of_regressors + arima_order[1] + arima_order[3] +
        seasonal_order[1] + seasonal_order[3] + 1

    # Calculate AIC
    aic <- -2 * loglik + 2 * n_p

    # Calculate BIC
    bic <- -2 * loglik + n_p * log(effective_n)

    # Calculate AICc (corrected AIC)
    aicc <- aic + (2 * n_p * (n_p + 1)) / (effective_n - n_p - 1)

    # Calculate Hannan-Quinn Criterion
    hq <- -2 * loglik + 2 * n_p * log(log(effective_n))

    # Return the results as a list
    return(list(AIC = aic, BIC = bic, AICc = aicc, HQ = hq, npar = n_p))
}
```

Plotting Model Output

The `plot_model_output` function visualizes the original time series, fitted values, residuals, and provides a summary of the model fit, including information criteria and Ljung-Box test results.

```r
plot_model_output <- function(model, series_name = "Series",
                              adj_inf_criteria_for_log = FALSE) {
    # Extract the original series, fitted values, and residuals from the model
    original_series <- model$x
    fitted_values <- fitted(model)
    residuals <- residuals(model)

    # Perform Ljung-Box test for residuals independence
    ljung_box_test <- Box.test(residuals, lag = 20, type = "Ljung-Box")
    lb_pvalue <- round(ljung_box_test$p.value, 4)

    # Extract model coefficients and standard errors
    coef_vals <- round(coef(model), 4)
    coef_errors <- round(sqrt(diag(vcov(model))), 4)

    # Create the model equation as a string
    model_eq <- paste0("ARIMA", substr(paste(model$call)[3], 2, 10),
                       " x ",
                       substr(paste(model$call)[4], 2, 10),
                       " Method = ", paste(model$call$method))

    # Use the selection_criteria function to get AIC, BIC, AICc, and HQ
    criteria <- selection_criteria(model,
                                   adjust_for_log_transform = adj_inf_criteria_for_log)

    n_p <- criteria$npar

    # Create a dataframe to store the original series, fitted values, and residuals
    df <- data.frame(
        Time = as.numeric(time(original_series)),
        Original = as.numeric(original_series),
        Fitted = as.numeric(fitted_values),
        Residuals = as.numeric(residuals)
    )

    # Plot the original series and fitted values
    p1 <- ggplot(df, aes(x = Time)) +
        geom_line(aes(y = Original), color = "blue", linewidth = 1,
                  show.legend = TRUE) +
        geom_line(aes(y = Fitted), color = "red",
                  linetype = "dashed", linewidth = 1, show.legend = TRUE) +
        ggtitle(paste(series_name, ": Original vs Fitted")) +
        xlab("Time") + ylab("Values") +
        theme_minimal() +
        labs(caption = paste("Model Summary: AIC =", round(criteria$AIC, 2),
                             ", npar= ", n_p,
                             ", BIC =", round(criteria$BIC, 2),
                             ", AICc =", round(criteria$AICc, 2),
                             ",HQ =", round(criteria$HQ, 2),
                             ", Log-Likelihood =", round(logLik(model), 2),
                             "\nLjung-Box Test p-value:", lb_pvalue,
                             "\nModel Equation:", model_eq,
                             "\nCoefficients (Estimate ± Std. Error):\n",
```

```
                              paste(names(coef_vals), "=", coef_vals, "±", coef_errors,
                                    collapse = ", ")))

    # Plot the residuals
    p2 <- ggplot(df, aes(x = Time, y = Residuals)) +
        geom_line(color = "purple", linewidth = 1) +
        geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
        ggtitle(paste(series_name, ": Residuals")) +
        xlab("Time") + ylab("Residuals") +
        theme_minimal()

    # Print the model summary and Ljung-Box test result on the console
    print(summary(model))
    print(ljung_box_test)

    # Combine the plots
    grid.arrange(p1, p2, nrow = 2)
}
```

Generating Constant Term

The `const_term` function creates a constant term for the model based on differencing orders.

```
const_term <-  function(y, d, D){
    return(diffinv(diffinv(rep(1, length(y)), lag = 1, differences =d), lag = 12,
                   differences = D)[-(1:(d+D*frequency(y)))])
}
```

Outlier Detection Functions

Functions `AO` (Additive Outlier) and `LS` (Level Shift) are defined to detect different types of outliers in the time series.

```
AO <- function(y, t) {
    time_point = start(y)
    time_point[2] = time_point[2] + (t-1)
    time_point[1]  = time_point[1] + ((time_point[2]-1) %/% 12)
    time_point[2] = ifelse( time_point[2] %% 12 == 0 , 12, time_point[2]%%12)
    new_var_name = paste("AO", time_point[1], ".", sprintf("%02d", time_point[2]), sep  = "")

    x = as.integer(seq_along(y) == t)
    x = matrix(x, ncol = 1)
    colnames(x)[1] <- new_var_name
    return(x)
}

LS <- function(y, t) {
    time_point = start(y)
    time_point[2] = time_point[2] + (t-1)
    time_point[1]  = time_point[1] + ((time_point[2]-1) %/% 12)
    time_point[2] = ifelse( time_point[2] %% 12 == 0 , 12, time_point[2]%%12)
    new_var_name = paste("LS", time_point[1], ".", sprintf("%02d", time_point[2]), sep  = "")

    x = as.integer(seq_along(y) >= t) - 1
```

```
    x = matrix(x, ncol = 1)
    colnames(x)[1] <- new_var_name
    return(x)
}
```

Calculating Outlier T-Value

The `outlier_t_value` function computes the t-value for potential outliers to assess their significance in the model.

```
outlier_t_value <- function(model, t, xreg = NULL, type = c("AO", "LS")){
    y = model$x
    X = xreg
    nreg = ifelse(is.null(xreg),  0, dim(xreg)[2])
    ncoeff = sum(c(eval(model$call$order)[-2], eval(model$call$seasonal)[-2]))
    model_order = eval(model$call$order)
    model_seasonal = eval(model$call$seasonal)
    new_variable = NULL
    if(type == 'AO'){
        new_variable = AO(y, t)
    }
    else if(type == "LS"){
        new_variable = LS(y, t)
    }
    X = cbind(X, new_variable)
    if(abs(det(t(X) %*% X)) < 1e-10){
        return(list(tvalue = 0, time_point = colnames(X)[dim(X)[2]] ))
    }
    fit = Arima(y, order = model_order,
                seasonal = model_seasonal,
                xreg = X,
                include.mean = FALSE, method = 'ML',
                fixed = c(coef(model)[1:ncoeff], rep(NA, nreg + 1)))
    ind = length(fit$coef)
    ind2 = dim(fit$var.coef)[1]
    return(list(tvalue = abs((fit$coef)[ind]) / sqrt((fit$var.coef)[ind2, ind2]),
                time_point = colnames(X)[dim(X)[2]]))
}
```

Forward and Backward Pass for Outlier Detection

The `forward_pass` and `backward_pass` functions iteratively identify and remove outliers based on t-values exceeding a critical threshold.

```
forward_pass <- function(model, xreg=NULL, types = c("AO", "LS"), tcritical=3.88){
    y = model$x
    n = length(y)

    max_tvalue = 0
    ind_max_tvalue = NULL
    type_max_tvalue = NULL
    for(outliertype in types){
        for(i in 1:n){
            foo = outlier_t_value(model, i, xreg, outliertype)
```

```
            tvalue = foo$tvalue
            time_point = foo$time_point
            if(tvalue > tcritical){
                print(paste(time_point, tvalue))
                if(tvalue > max_tvalue){
                    max_tvalue = tvalue
                    ind_max_tvalue = i
                    type_max_tvalue = outliertype
                }
            }
        }
    }
    if(is.null(type_max_tvalue)){
        return(NULL)
    }
    if(type_max_tvalue == "AO"){
        return(AO(y, ind_max_tvalue))
    }
    if(type_max_tvalue == "LS"){
        return(LS(y, ind_max_tvalue))
    }
    return(NULL)
}

backward_pass <-  function(model, xreg=NULL, tcritical=3.88){
    y = model$x
    X = xreg
    nreg = ifelse(is.null(xreg),  0, dim(xreg)[2])
    ncoeff = sum(c(eval(model$call$order)[-2], eval(model$call$seasonal)[-2]))
    model_order = eval(model$call$order)
    model_seasonal = eval(model$call$seasonal)

    fit = Arima(y, order = model_order,
                seasonal = model_seasonal,
                xreg = X,
                include.mean = FALSE, method = 'ML',
                fixed = c(coef(model)[1:ncoeff], rep(NA, nreg)))
    betas = abs(fit$coef[-c(1:ncoeff)])
    se_betas = sqrt(diag(fit$var.coef))
    tvalues  = betas / se_betas

    min_tvalue = 1e8
    min_ind = NULL
    for(i in 1:length(tvalues)){
        if(grepl("^(AO|LS)(1[6-9][0-9]|20[0-9][0-9])\\.(0[1-9]|1[0-2])$",
                 colnames(X)[i])){
            if(tvalues[i] < tcritical ){
                print(paste(colnames(X)[i], tvalues[i]))
                if(tvalues[i] < min_tvalue ){
                    min_tvalue = tvalues[i]
                    min_ind = i
                }
            }
```

```r
        }
    }

    return(min_ind)
}
```

Data Preparation

Reading Diwali Dates

We start by reading the Diwali dates from a CSV file. This file was generated using the data vendor : Calendarific The `ddates` dataframe is expected to contain the columns: `year`, `month`, and `day` for each Diwali date.

```r
setwd("~/Documents/Projects/X-13-ARIMA-SEATS-Model-for-Seasonal-Adjustment")
ddates <- read.csv('data/diwali.csv')
```

Calculating Diwali Regressor

The `diwali_w` function calculates the proportion of days falling in September, October, and November within a window of `w` days before Diwali.

```r
diwali_w <- function(year, w) {
    # Find Diwali month and day for the given year
    m <- ddates$month[which(ddates$year == year)]
    d <- ddates$day[which(ddates$year == year)]

    # Create a Date object for Diwali
    diwali_date <- as.Date(paste(year, m, d, sep = "-"))

    # Calculate the date 'w' days before Diwali
    start_date <- diwali_date - w

    # Initialize counts for each month
    sep_count <- 0
    oct_count <- 0
    nov_count <- 0

    # Iterate over each day in the range
    for (i in 1:w) {
        current_date <- diwali_date - i + 1
        current_month <- format(current_date, "%m")

        # Count days in each month
        if (current_month == "09") {
            sep_count <- sep_count + 1
        } else if (current_month == "10") {
            oct_count <- oct_count + 1
        } else if (current_month == "11") {
            nov_count <- nov_count + 1
        }
    }

    total_days <- w
```

```r
    return(c("sep" = sep_count / total_days,
             "oct" = oct_count / total_days, "nov" = nov_count / total_days))
}
```

We then compute the average proportions over a 200-year period.

```r
P_diwali <- matrix(0, ncol = 3, nrow = 200)
for(i in 1:200){
    P_diwali[i,] <- diwali_w(1899 + i, 20)
}
colMeans(P_diwali)
```

```
## [1] 0.01075 0.79800 0.19125
```

Creating Trading Day Regressor

We generate the trading day regressor using the `td1nolpyear_regressor` function.

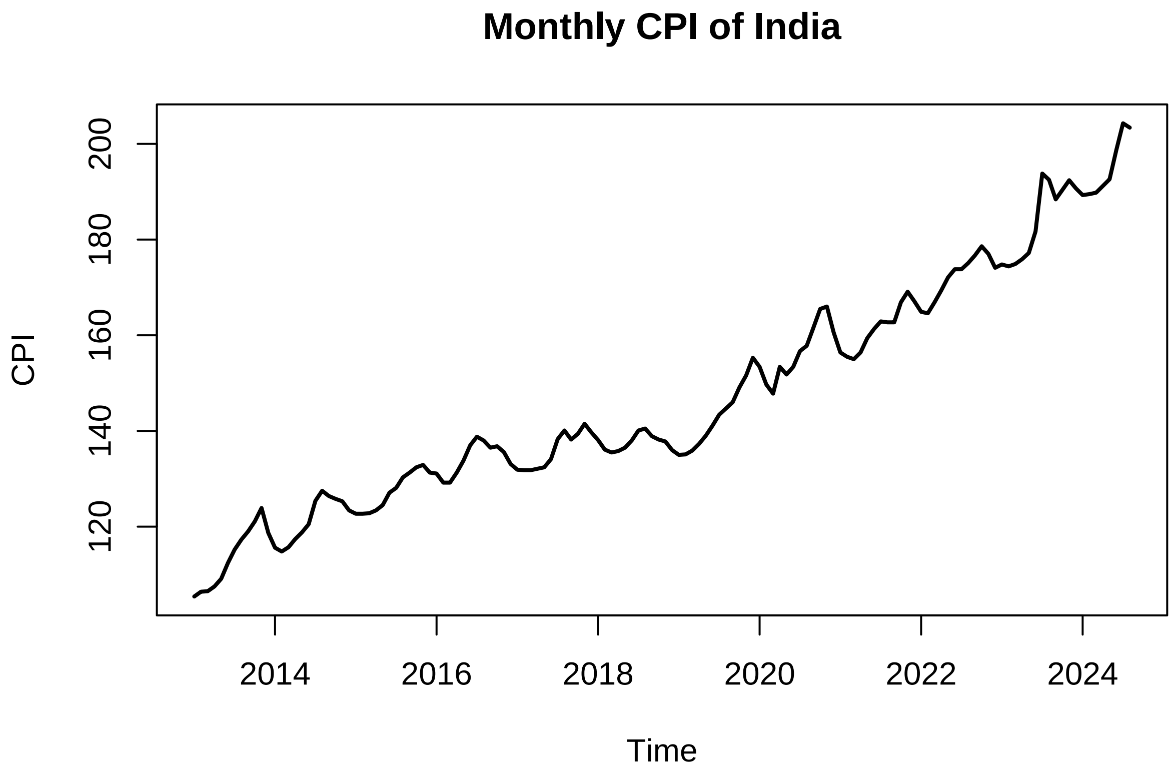
```r
# Assuming 'data' length will be defined later
# td1nolpyear_regressor function defined earlier
```

Preparing Time Series Objects

We create time series objects for the main data, Diwali indicator, and trading day regressor.

```r
# Load helper functions if any
# source('code/helper_functions.r') # Uncomment if helper functions are in a separate file

# Loading data
data <- c(105.4, 106.4, 106.5, 107.5, 109.1, 112.4, 115.2, 117.3, 119.0, 121.1, 123.9, 118.7,
          115.6, 114.8, 115.7, 117.4, 118.8, 120.5, 125.4, 127.5, 126.4, 125.8, 125.3, 123.4,
          122.7, 122.7, 122.8, 123.4, 124.5, 127.1, 128.1, 130.3, 131.3, 132.4, 132.9, 131.3,
          131.1, 129.2, 129.2, 131.3, 133.8, 137.0, 138.8, 138.0, 136.5, 136.8, 135.6, 133.1,
          131.9, 131.8, 131.8, 132.1, 132.4, 134.1, 138.3, 140.1, 138.2, 139.4, 141.5, 139.7,
          138.1, 136.1, 135.5, 135.8, 136.5, 138.0, 140.1, 140.5, 138.9, 138.2, 137.8, 136.0,
          135.0, 135.1, 135.9, 137.3, 139.0, 141.1, 143.4, 144.7, 146.0, 149.1, 151.6, 155.3,
          153.4, 149.7, 147.8, 153.4, 151.8, 153.4, 156.7, 157.8, 161.6, 165.5, 166.0, 160.6,
          156.4, 155.5, 155.0, 156.4, 159.4, 161.3, 162.9, 162.7, 162.7, 166.9, 169.1, 167.1,
          164.9, 164.6, 166.9, 169.4, 172.1, 173.8, 173.8, 175.1, 176.7, 178.6, 177.0, 174.1,
          174.8, 174.4, 174.9, 175.9, 177.2, 181.7, 193.8, 192.5, 188.4, 190.4, 192.4, 190.7,
          189.3, 189.5, 189.8, 191.2, 192.6, 198.7, 204.3, 203.4)

diwali_ind <- c(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0405, -0.0405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.6595, 0.6595, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.3595, 0.3595, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.6595, 0.6595, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0595, 0.0595, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.6595, 0.6595, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3405, -0.3405, 0.0)
```

```r
# Create trading day regressor
td1nolpyear <- td1nolpyear_regressor(start = c(2013, 1), n = length(data))

# It matches exactly with the regressor being used in the program
# Here is the image of the regressor of the program:
# ![Regressor Image](path_to_image.png) # Replace with actual image path

# Create time series objects
y <- ts(data, start = c(2013, 1), frequency = 12)
plot(y, main = "Monthly CPI of India", lwd = 2, ylab = "CPI")
```




```r
diwali <- ts(diwali_ind[1:length(y)], start = c(2013, 1), frequency = 12)
td1nolpyear <- ts(td1nolpyear, start = c(2013, 1), frequency = 12)

print(td1nolpyear)
```

```
##       Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
## 2013  3.0  0.0 -4.0  2.0  3.0 -5.0  3.0 -0.5 -1.5  3.0 -1.5 -0.5
## 2014  3.0  0.0 -4.0  2.0 -0.5 -1.5  3.0 -4.0  2.0  3.0 -5.0  3.0
## 2015 -0.5  0.0 -0.5  2.0 -4.0  2.0  3.0 -4.0  2.0 -0.5 -1.5  3.0
## 2016 -4.0  1.0  3.0 -1.5 -0.5  2.0 -4.0  3.0  2.0 -4.0  2.0 -0.5
## 2017 -0.5  0.0  3.0 -5.0  3.0  2.0 -4.0  3.0 -1.5 -0.5  2.0 -4.0
## 2018  3.0  0.0 -0.5 -1.5  3.0 -1.5 -0.5  3.0 -5.0  3.0  2.0 -4.0
## 2019  3.0  0.0 -4.0  2.0  3.0 -5.0  3.0 -0.5 -1.5  3.0 -1.5 -0.5
## 2020  3.0 -2.5 -0.5  2.0 -4.0  2.0  3.0 -4.0  2.0 -0.5 -1.5  3.0
## 2021 -4.0  0.0  3.0  2.0 -4.0  2.0 -0.5 -0.5  2.0 -4.0  2.0  3.0
## 2022 -4.0  0.0  3.0 -1.5 -0.5  2.0 -4.0  3.0  2.0 -4.0  2.0 -0.5
## 2023 -0.5  0.0  3.0 -5.0  3.0  2.0 -4.0  3.0 -1.5 -0.5  2.0 -4.0
## 2024  3.0  1.0 -4.0  2.0  3.0 -5.0  3.0 -0.5
```

```r
print(y)
```

```
##        Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
## 2013 105.4 106.4 106.5 107.5 109.1 112.4 115.2 117.3 119.0 121.1 123.9 118.7
## 2014 115.6 114.8 115.7 117.4 118.8 120.5 125.4 127.5 126.4 125.8 125.3 123.4
## 2015 122.7 122.7 122.8 123.4 124.5 127.1 128.1 130.3 131.3 132.4 132.9 131.3
## 2016 131.1 129.2 129.2 131.3 133.8 137.0 138.8 138.0 136.5 136.8 135.6 133.1
## 2017 131.9 131.8 131.8 132.1 132.4 134.1 138.3 140.1 138.2 139.4 141.5 139.7
## 2018 138.1 136.1 135.5 135.8 136.5 138.0 140.1 140.5 138.9 138.2 137.8 136.0
## 2019 135.0 135.1 135.9 137.3 139.0 141.1 143.4 144.7 146.0 149.1 151.6 155.3
## 2020 153.4 149.7 147.8 153.4 151.8 153.4 156.7 157.8 161.6 165.5 166.0 160.6
## 2021 156.4 155.5 155.0 156.4 159.4 161.3 162.9 162.7 162.7 166.9 169.1 167.1
## 2022 164.9 164.6 166.9 169.4 172.1 173.8 173.8 175.1 176.7 178.6 177.0 174.1
## 2023 174.8 174.4 174.9 175.9 177.2 181.7 193.8 192.5 188.4 190.4 192.4 190.7
## 2024 189.3 189.5 189.8 191.2 192.6 198.7 204.3 203.4
```

Model Fitting and Transformation

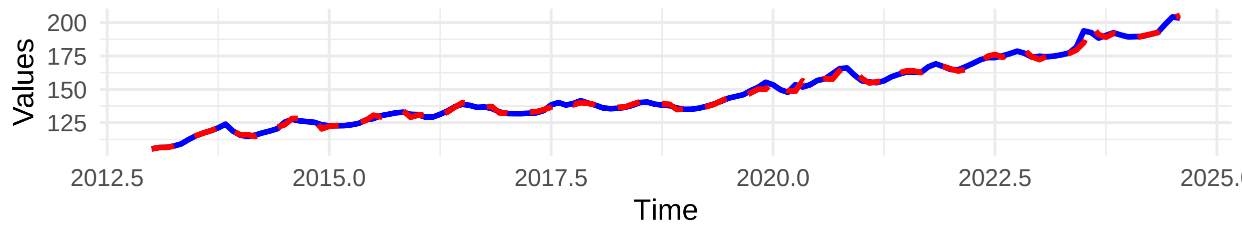
We fit ARIMA models to the time series data `y` and its logarithm `log(y)`, then compare them using the Akaike Information Criterion corrected (AICc) to decide on the appropriate transformation.

```r
# Regression matrix
Xreg <- cbind(td1nolpyear, diwali)

# Fit ARIMA models without transformation
no_transformation_test_fit <- Arima(y, xreg = Xreg,
                                    order = c(0,1,1), seasonal = c(0,1,1),
                                    include.mean = FALSE, method = 'ML', lambda = NULL)

# Fit ARIMA models with log transformation
log_transformation_test_fit <- Arima(log(y), xreg = Xreg,
                                     order = c(0,1,1), seasonal = c(0,1,1),
                                     include.mean = FALSE, method = 'ML', lambda = NULL)

# Plot model outputs
plot_model_output(no_transformation_test_fit, series_name = "No Transformation")
```

```
## Series: y
## Regression with ARIMA(0,1,1)(0,1,1)[12] errors
##
## Coefficients:
##          ma1     sma1  td1nolpyear  diwali
##       0.3482  -0.9124      -0.0074  0.2917
## s.e.  0.0952   0.1868       0.0253  0.3759
##
## sigma^2 = 3.167:  log likelihood = -261.37
## AIC=532.74   AICc=533.23   BIC=546.96
##
## Training set error measures:
##                      ME     RMSE      MAE         MPE      MAPE      MASE
## Training set 0.03178592 1.668093 1.097779 0.001223092 0.7192208 0.1414778
##                     ACF1
## Training set -0.06194926
##
```

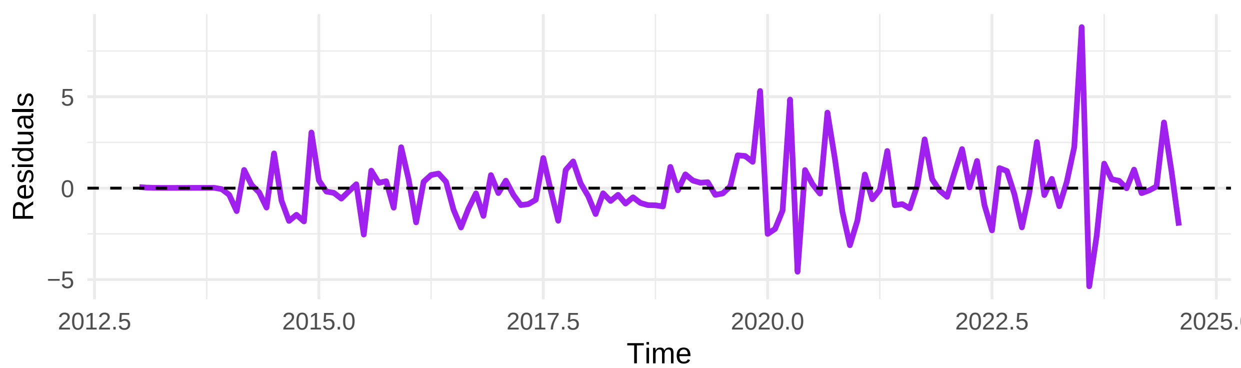
```
##  Box-Ljung test
##
## data:  residuals
## X-squared = 14.742, df = 20, p-value = 0.791
```

No Transformation : Original vs Fitted



Model Summary: AIC = 532.74 , npar= 5 , BIC = 546.96 , AICc = 533.23 ,HQ = 538.52 , Log−Likelihood = −261.37
Ljung−Box Test p−value: 0.791
Model Equation: ARIMA(0, 1, 1) x (0, 1, 1) Method = ML
Coefficients (Estimate ± Std. Error):
ma1 = 0.3482 ± 0.0952, sma1 = −0.9124 ± 0.1868, td1nolpyear = −0.0074 ± 0.0253, diwali = 0.2917 ± 0.3759

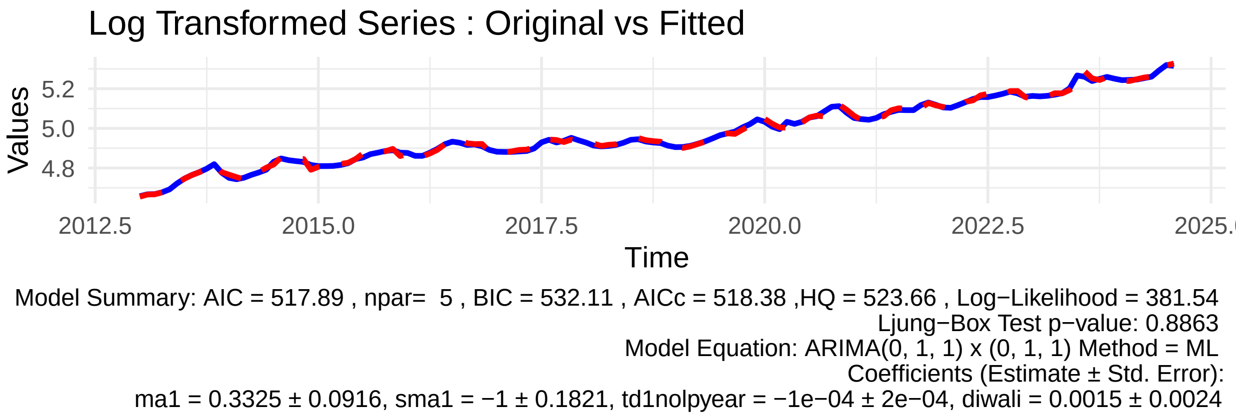
No Transformation : Residuals

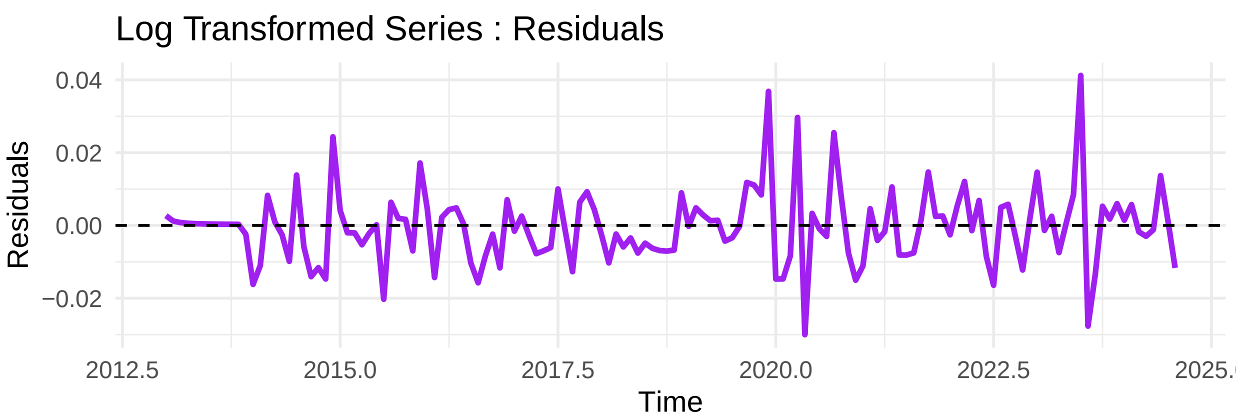



```
plot_model_output(log_transformation_test_fit,
                  series_name = "Log Transformed Series", adj_inf_criteria_for_log = TRUE)
```

```
## Series: log(y)
## Regression with ARIMA(0,1,1)(0,1,1)[12] errors
##
## Coefficients:
##          ma1     sma1  td1nolpyear  diwali
##       0.3325  -1.0000       -1e-04  0.0015
## s.e.  0.0916   0.1821        2e-04  0.0024
##
## sigma^2 = 0.00012:  log likelihood = 381.54
## AIC=-753.09   AICc=-752.59   BIC=-738.87
##
## Training set error measures:
##                         ME       RMSE         MAE         MPE      MAPE
## Training set -0.0005356371 0.01026985 0.007298123 -0.01142566 0.1458139
##                   MASE        ACF1
## Training set 0.1402815 -0.03762701
##
##  Box-Ljung test
##
## data:  residuals
```

```
## X-squared = 12.787, df = 20, p-value = 0.8863
```





Interpretation of AICc

The AICc values from both models are compared to determine whether a log transformation is preferable. According to the comments, if `AICC_nolog - AICC_log < -2`, we prefer no log transform. In this case:

`AICC_nolog - AICC_log = 529.65 - 514.66 = 14.99`

Since 14.99 is not less than -2, we prefer the log-transformed model.

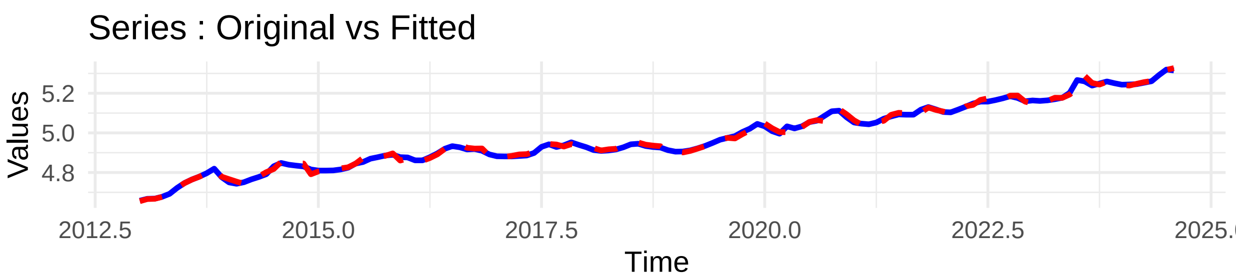
```r
Z <- log(y)
```

Default Model Estimation

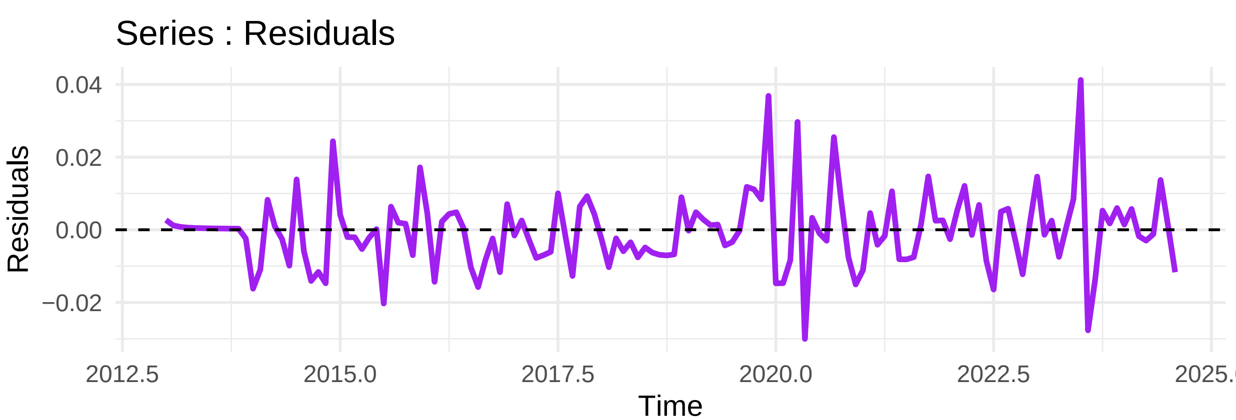
We proceed with fitting the default ARIMA model to the log-transformed series `Z`. This includes initial outlier identification and tests for trading day effects.

```r
# Create constant term
const <- const_term(y, 1, 1)

# Fit default ARIMA model without constant term
default_model_fit1 <- Arima(Z, xreg = Xreg, order = c(0,1,1),
                            seasonal = c(0,1,1), include.mean = FALSE,
                            method = 'ML', lambda = NULL)

# Plot model output
plot_model_output(default_model_fit1, adj_inf_criteria_for_log = TRUE)
```

```
## Series: Z
## Regression with ARIMA(0,1,1)(0,1,1)[12] errors
##
## Coefficients:
##          ma1     sma1  td1nolpyear  diwali
##       0.3325  -1.0000       -1e-04  0.0015
## s.e.  0.0916   0.1821        2e-04  0.0024
##
## sigma^2 = 0.00012:  log likelihood = 381.54
## AIC=-753.09   AICc=-752.59   BIC=-738.87
##
## Training set error measures:
##                         ME       RMSE         MAE         MPE      MAPE
## Training set -0.0005356371 0.01026985 0.007298123 -0.01142566 0.1458139
##                   MASE        ACF1
## Training set 0.1402815 -0.03762701
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 12.787, df = 20, p-value = 0.8863
```



Model Summary: AIC = 517.89 , npar= 5 , BIC = 532.11 , AICc = 518.38 ,HQ = 523.66 , Log−Likelihood = 381.54
Ljung−Box Test p−value: 0.8863
Model Equation: ARIMA(0, 1, 1) x (0, 1, 1) Method = ML
Coefficients (Estimate ± Std. Error):
ma1 = 0.3325 ± 0.0916, sma1 = −1 ± 0.1821, td1nolpyear = −1e−04 ± 2e−04, diwali = 0.0015 ± 0.0024




```
# Perform t-test on residuals to test the presence of a constant term
t.test(residuals(default_model_fit1))
```

```
##
##  One Sample t-test
##
```

```
## data:  residuals(default_model_fit1)
## t = -0.61575, df = 139, p-value = 0.5391
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.002255568  0.001184294
## sample estimates:
##     mean of x
## -0.0005356371
```

Interpretation:

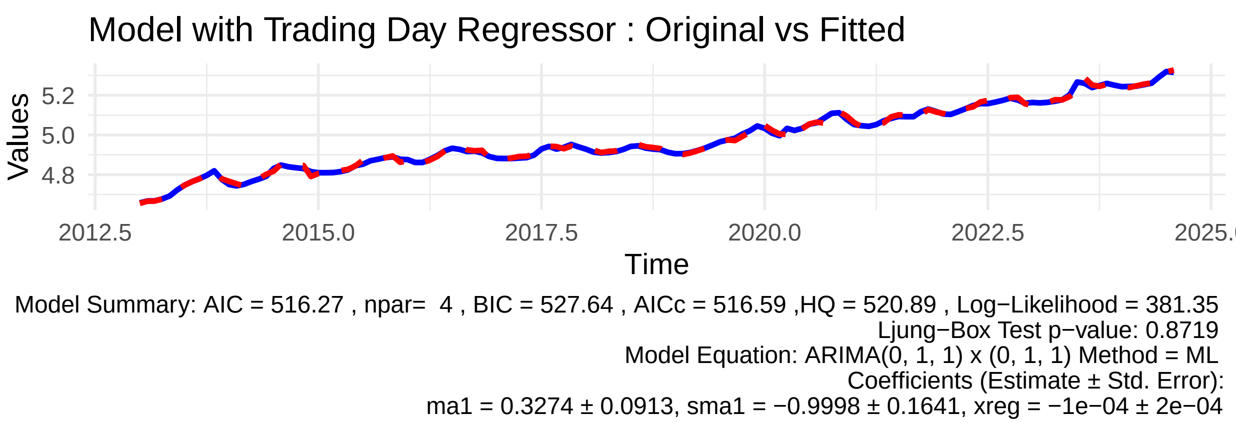
The p-value from the t-test (`0.5391`) indicates that the constant term is not significant. Therefore, we will not include the constant term in the model.

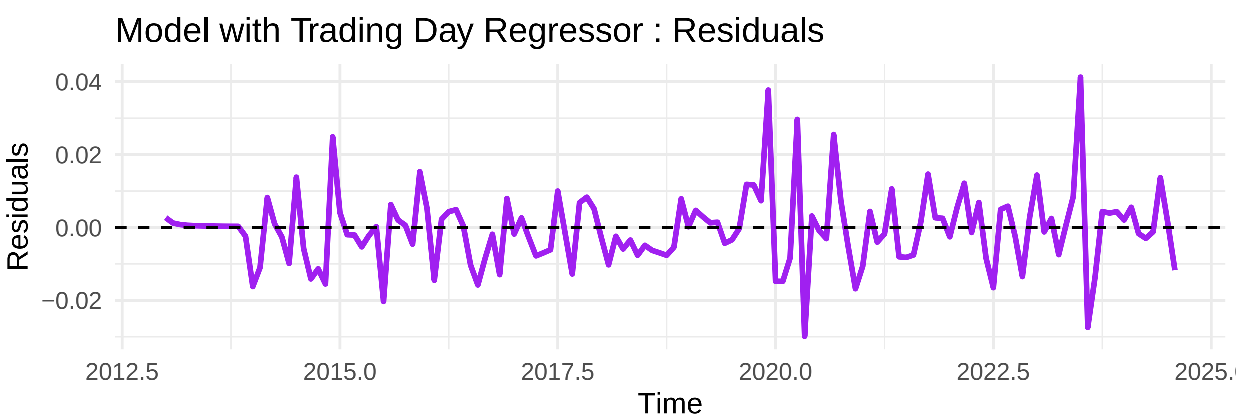
Trading Day Regressor Selection

We evaluate whether to include the trading day regressor (`td1nolpyear`) and the Diwali regressor (`diwali`) based on AICc differences.

```r
# AIC test for td1nolpyear and diwali

# Model with trading day regressor
model_with_td <- Arima(Z, xreg = td1nolpyear,
                       order = c(0,1,1), seasonal = c(0,1,1),
                       include.mean = FALSE, method = 'ML', lambda = NULL)
plot_model_output(model_with_td,
                  series_name = "Model with Trading Day Regressor",
                  adj_inf_criteria_for_log = TRUE)
```

```
## Series: Z
## Regression with ARIMA(0,1,1)(0,1,1)[12] errors
##
## Coefficients:
##          ma1     sma1    xreg
##       0.3274  -0.9998  -1e-04
## s.e.  0.0913   0.1641   2e-04
##
## sigma^2 = 0.0001195:  log likelihood = 381.35
## AIC=-754.71   AICc=-754.38   BIC=-743.33
##
## Training set error measures:
##                          ME       RMSE         MAE         MPE      MAPE
## Training set -0.0005362574 0.01028626 0.007273905 -0.01144279 0.1453235
##                    MASE        ACF1
## Training set 0.139816 -0.03619314
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 13.127, df = 20, p-value = 0.8719
```

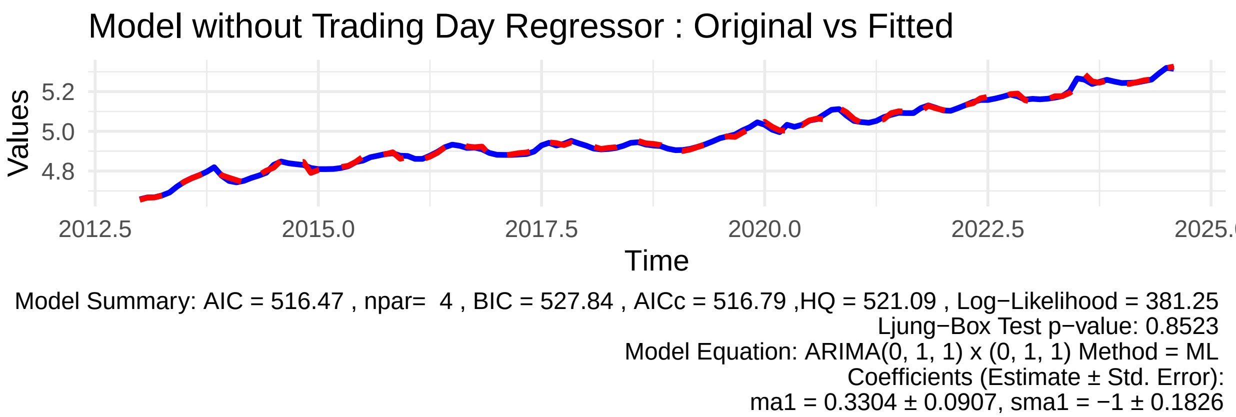


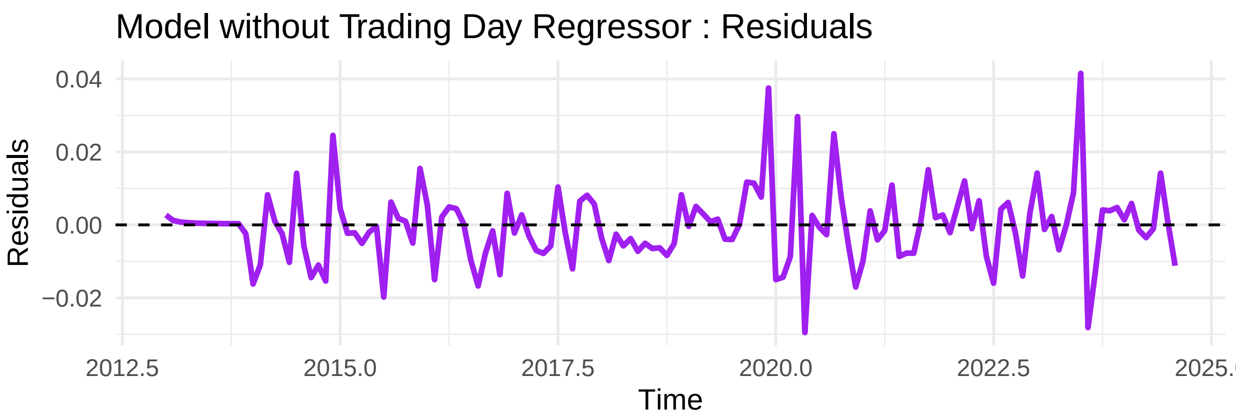



```r
# Model without trading day regressor
model_without_td <- Arima(Z, order = c(0,1,1),
                          seasonal = c(0,1,1), include.mean = FALSE,
                          method = 'ML', lambda = NULL)
plot_model_output(model_without_td,
                  series_name = "Model without Trading Day Regressor",
                  adj_inf_criteria_for_log = TRUE)
```

```
## Series: Z
## ARIMA(0,1,1)(0,1,1)[12]
##
## Coefficients:
##          ma1     sma1
##       0.3304  -1.0000
## s.e.  0.0907   0.1826
##
## sigma^2 = 0.0001187:  log likelihood = 381.25
## AIC=-756.51   AICc=-756.31   BIC=-747.97
##
## Training set error measures:
##                         ME       RMSE         MAE         MPE      MAPE
## Training set -0.0005364757 0.01029314 0.007268806 -0.01144638 0.1452537
##                  MASE        ACF1
## Training set 0.139718 -0.03733072
##
##  Box-Ljung test
##
```

```
## data:  residuals
## X-squared = 13.555, df = 20, p-value = 0.8523
```



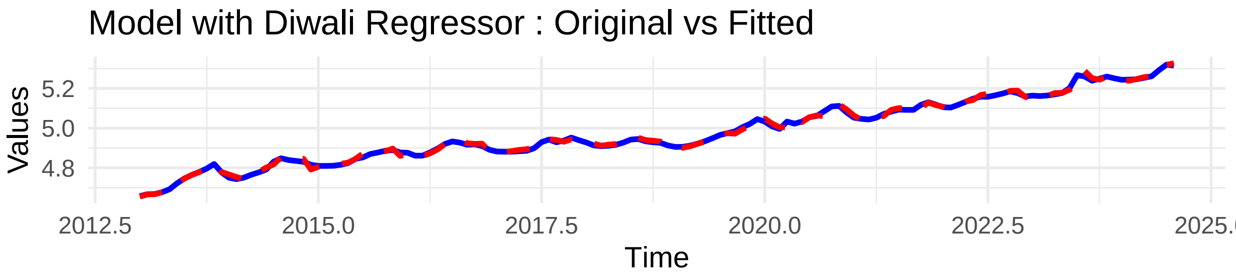



```r
# AICC comparison
# AICC_with - AICC_without = -754.38 + 754.18 = 0.2 > 0
# Trading day regressor will not be included in the model
```

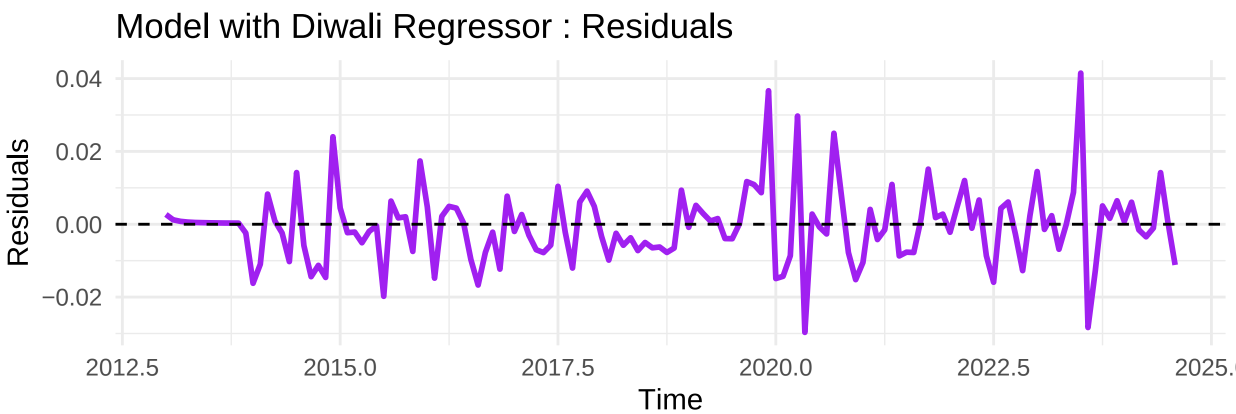
```r
# Model with Diwali regressor
model_with_diwali <- Arima(Z, xreg = diwali, order = c(0,1,1),
                           seasonal = c(0,1,1), include.mean = FALSE,
                           method = 'ML', lambda = NULL)
plot_model_output(model_with_diwali,
                  series_name = "Model with Diwali Regressor",
                  adj_inf_criteria_for_log = TRUE)
```

```
## Series: Z
## Regression with ARIMA(0,1,1)(0,1,1)[12] errors
##
## Coefficients:
##          ma1     sma1    xreg
##       0.3356  -1.0000  0.0015
## s.e.  0.0910   0.2093  0.0024
##
## sigma^2 = 0.0001192:  log likelihood = 381.45
## AIC=-754.9   AICc=-754.57   BIC=-743.53
##
## Training set error measures:
##                          ME       RMSE         MAE         MPE      MAPE      MASE
```

```
## Training set -0.000535844 0.01027714 0.007297058 -0.01142898 0.1458251 0.140261
##                      ACF1
## Training set -0.03880191
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 13.168, df = 20, p-value = 0.8701
```



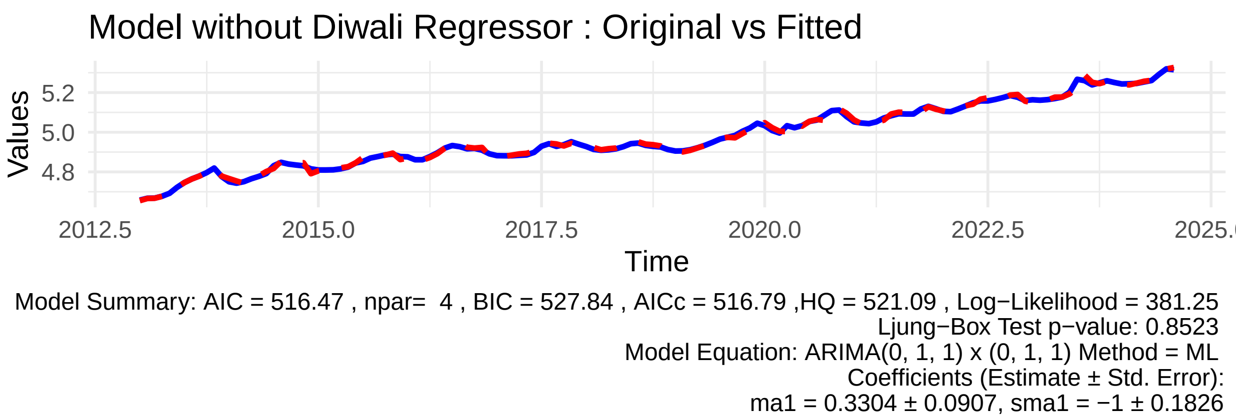
Model Summary: AIC = 516.07 , npar= 4 , BIC = 527.45 , AICc = 516.4 ,HQ = 520.69 , Log−Likelihood = 381.45
Ljung−Box Test p−value: 0.8701
Model Equation: ARIMA(0, 1, 1) x (0, 1, 1) Method = ML
Coefficients (Estimate ± Std. Error):
ma1 = 0.3356 ± 0.091, sma1 = −1 ± 0.2093, xreg = 0.0015 ± 0.0024

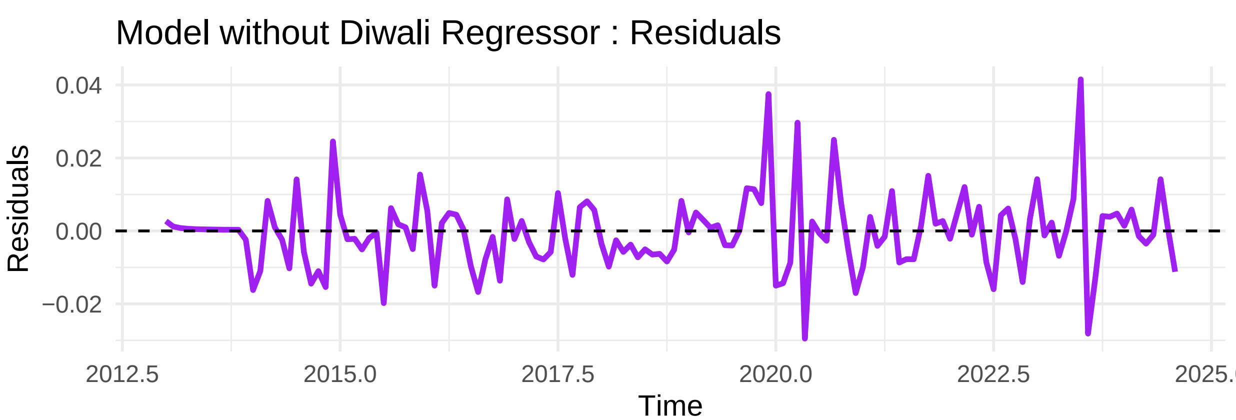



```r
# Model without Diwali regressor
model_without_diwali <- Arima(Z, order = c(0,1,1),
                              seasonal = c(0,1,1), include.mean = FALSE,
                              method = 'ML', lambda = NULL)
plot_model_output(model_without_diwali,
                  series_name = "Model without Diwali Regressor",
                  adj_inf_criteria_for_log = TRUE)
```

```
## Series: Z
## ARIMA(0,1,1)(0,1,1)[12]
##
## Coefficients:
##          ma1     sma1
##       0.3304  -1.0000
## s.e.  0.0907   0.1826
##
## sigma^2 = 0.0001187:  log likelihood = 381.25
## AIC=-756.51   AICc=-756.31   BIC=-747.97
##
```

```
## Training set error measures:
##                         ME       RMSE         MAE         MPE      MAPE
## Training set -0.0005364757 0.01029314 0.007268806 -0.01144638 0.1452537
##                     MASE        ACF1
## Training set 0.139718 -0.03733072
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 13.555, df = 20, p-value = 0.8523
```






```
# AICC comparison
# AICC_with - AICC_without = -754.57 + 754.18 = 0.39 > 0
# Diwali regressor will not be included in the model
```

Outlier Detection in Default Model

We fit the default ARIMA model and perform forward and backward passes to identify significant outliers.

```
# Fit default ARIMA model without regressors
Xreg <- NULL
default_model <- Arima(Z, order = c(0,1,1), seasonal = c(0,1,1), xreg = Xreg,
                       include.mean = FALSE, method = 'ML', lambda = NULL)

default_model
```

```
## Series: Z
## ARIMA(0,1,1)(0,1,1)[12]
```

```
##
## Coefficients:
##          ma1     sma1
##       0.3304  -1.0000
## s.e.  0.0907   0.1826
##
## sigma^2 = 0.0001187:  log likelihood = 381.25
## AIC=-756.51   AICc=-756.31   BIC=-747.97
```

```r
# Note: The coefficients are negative compared to X13-ARIMA-SEATS
# due to different parameterizations.

# Forward Pass for Outlier Detection
print("Forward pass 1")
```

```
## [1] "Forward pass 1"
```

```r
curr_outlier <- forward_pass(default_model, xreg = Xreg,
                             types = c("AO", "LS"), tcritical = 3.88)
```

```
## [1] "AO2020.04 4.17002561699784"
## [1] "AO2023.07 4.08495930589087"
## [1] "LS2019.12 3.88044849190372"
## [1] "LS2023.07 4.64218785557147"
```

```r
# default value taken from the manual See X-13ARIMA-SEATS Manual
k <- 2
while(!is.null(curr_outlier)){
    Xreg <- cbind(Xreg, curr_outlier)
    print(paste("Forward pass", k))
    k <- k +1
    curr_outlier <- forward_pass(default_model,
                                 xreg = Xreg,
                                 types = c("AO", "LS"), tcritical = 3.88)
}
```

```
## [1] "Forward pass 2"
## [1] "AO2020.03 3.92397472587285"
## [1] "AO2020.04 4.51624029586687"
## [1] "LS2019.12 4.23721133059193"
## [1] "LS2020.04 4.12090040971873"
## [1] "Forward pass 3"
## [1] "AO2013.11 4.17134927338926"
## [1] "LS2019.12 4.56295072308321"
## [1] "Forward pass 4"
## [1] "AO2013.11 4.10983186897022"
## [1] "Forward pass 5"
```

```r
# Backward Pass for Outlier Detection
k <- 1
while(TRUE){
    print(paste("Backward_pass", k))
    k <- k +1
    ind <- backward_pass(default_model, Xreg, tcritical = 3.88)
    if(is.null(ind)){
        print("Outlier detection is done")
        break
```

```r
    }
    else{
        Xreg <- Xreg[,-ind]
    }
}
```

```
## [1] "Backward_pass 1"
## [1] "Outlier detection is done"
```

```r
# Outliers identified from the default model
print(colnames(Xreg))
```

```
## [1] "LS2023.07" "AO2020.04" "LS2019.12" "AO2013.11"
```

Interpretation:

The identified outliers (`colnames(Xreg)`) match those obtained from the X13-ARIMA-SEATS program, confirming the accuracy of our implementation.

Next Steps

Our next goal is to understand and implement the Iterative Generalized Least Squares (IGLS) algorithm to align the parameter estimates with those from the X13-ARIMA-SEATS program.

Conclusion

This document detailed the implementation of the `regARIMA` procedure using X13-ARIMA-SEATS in R. We covered data preparation, model fitting, transformation decisions based on AICc, AIC tests for regressors and outlier detection.

References

- X-13ARIMA-SEATS Manual
- R Documentation for `Arima`