Industrial Engineering Department

IE360 – Statistical Forecasting And Time Series

Spring 2024

Instructor: Mustafa Gökçe Baydoğan

Term Project Report

<u>GROUP 20:</u>

Kayra Şener 2020402009

Selahattin Eyüp Gülçimen 2020402066

Table Of Contents:

Introduction: Predicting Solar Power Production for Edikli GES

Solar energy is rapidly becoming a key player in the renewable energy sector, offering a cleaner and more sustainable alternative to traditional power sources. However, its intermittent nature presents significant challenges for integrating it into the power grid. To address this, accurate forecasting of solar power production is essential for energy traders and power plant operators. This project aims to develop a statistical model to predict the hourly solar power output of the Edikli GES power plant in Niğde, Turkey, for the following day (day d+1).

Our project simulates real-world conditions faced by energy traders, providing daily forecasts through Google Forms and tracking performance with Google Sheets. By analyzing historical production data up to the day before (d-1) and weather data from 25 grid points around the power plant, we aim to improve the accuracy of our predictions.

The weather data includes several key variables that influence solar power production:

1. **Shortwave Radiation Flux (DSWRF_surface)**: This measures the intensity of sunlight hitting the ground, which is crucial for solar power generation. The more sunlight reaching the panels, the more electricity they can produce.
2. **Other Solar Radiation Variables (USWRF, DLWRF)**:
    - **USWRF (Upward Shortwave Radiation Flux)** measures the amount of solar radiation reflected back into space from the top of the Earth's atmosphere.
    - **DLWRF (Downward Longwave Radiation Flux)** measures the amount of energy radiating from the atmosphere to the Earth’s surface.
3. **Cloud Cover Data (TCDC)**: This represents the percentage of the sky covered by clouds, which can range from completely clear (0%) to fully overcast (100%). Different types of clouds impact sunlight differently:
    - **High Clouds (e.g., Cirrus)**: Usually composed of ice crystals and found at high altitudes, these clouds allow most sunlight to pass through.
    - **Middle Clouds (e.g., Altocumulus)**: Found at mid-altitudes, these clouds can partially block sunlight depending on their thickness.

    - **Low Clouds (e.g., Stratus)**: Typically dense and low-lying, these clouds can significantly reduce the amount of sunlight reaching the ground.
4. **Snow Cover Data (CSNOW_surface)**: Indicates whether snow is present on the ground. A categorical variable. Snow can reflect sunlight, reducing the amount reaching solar panels.
5. **Surface Temperature (TMP_surface)**: This influences solar panel efficiency and can affect power production.
6. **The latitude and longitude pairs for weather variables:** 25 available grids and their data can be aggregated for the model.

Our data-driven approach will explore various modeling techniques, such as SARIMA (Seasonal Autoregressive Integrated Moving Average) models, to identify the relationships between these weather variables and solar power production. By doing so, we aim to develop a reliable forecasting model that can help optimize energy management and improve grid integration for Edikli GES.

This project not only aims to provide accurate forecasts but also to offer valuable insights into how different weather conditions affect solar power production, contributing to the broader effort of integrating renewable energy into our power systems effectively.

2. Problem Solving Approach

Integrating multiple independent variables, such as various weather parameters, into a time series regression model is a complex and demanding task. It requires a thorough understanding of how each factor influences solar power production. Also, since the weather data is assessed to different longitudes and latitudes, aggregation regarding the coordinates was necessary, which could harm the predictions. Due to this complexity, there is a high risk of inaccuracies if not handled correctly. Therefore, we concluded that constructing such a model might be too risky and could compromise the accuracy of our predictions.

To mitigate this risk, we chose to use a SARIMA (Seasonal Autoregressive Integrated Moving Average) model. SARIMA models predict future values based solely on past data by differencing the series (the target variable becomes the difference of consecutive original target variables), adding past values of the target variable (AR components) and past errors

(MA components), without the need to integrate multiple independent variables (exogenous variables). This makes the modeling process simpler and more reliable. By focusing on the historical production data, we can leverage established patterns and trends to make more accurate forecasts. This approach allows us to deliver reliable predictions while avoiding the pitfalls of integrating and interpreting numerous complex variables.

We concluded that every hour of the day can be modeled independently to avoid hour-of-the-day seasonality. This approach also helped us to reduce the complexity of the big problem and divide it into smaller ones while also helping to detect anomalies and outliers more easily. So we divided the data according to the hours via EXCEL by counting the number of observations, dividing it into 24 to find the class size, and dividing the data accordingly by using functions like:

H1,Day1=INDEX($C:$C;((COLUMN(B$1)-1)*COUNT($B:$B)/24+ROW($A2)))

H2, Day1==INDEX($C:$C;((COLUMN(C$1)-1)*COUNT($B:$B)/24+ROW($A3)))

We also figured that the H0, H1, H2, H3, H20, H21, H22, and H23 columns have a constant value of 0, historically. So we decided not to make forecasts for these periods and predict all 0's to decrease the running time.

3. Code Structure

We used R for coding purposes due to its extensive number of statistical packages and ease of use.

Loading Required Libraries:

- **`forecast`**: This library is used for time series forecasting, including ARIMA models.
- **`data.table`**: This library provides an efficient way to handle and manipulate data tables.
- **`lubridate`**: This library simplifies working with dates and times in R.

```
# Loading the required libraries
require(forecast)
require(data.table)
require(lubridate)
```

Defining Data Path And Loading Production Data:

- **`data_path`**: Specifies the directory where the data file is located.
- **`file_production`**: Constructs the full path to the data file.
- **`fread`**: Reads the data from the CSV file into a data table.

- **`as.Date`**: Converts the `date` column to a date format.
- tail function is used to check if the raw data is updated correctly day by day

```
# Assessing data path
data_path <- 'C:/Users/kayra/Desktop/Okul/3-2/ie360_project/'
file_production <- paste0(data_path, 'production_by_hours.csv')

# Reading the production data and format the date
production_data <- fread(file_production)
production_data$date <- as.Date(production_data$date, format = "%d.%m.%y")

# Checking if the table is updated correctly
tail(production_data$date)
```

Defining Hourly Columns:

`hourly_columns`: Extracts the names of the columns corresponding to the 24 hourly production values. H4 is located in column "I" and H19 in "X", hence the index is "[9:24]".

```
# Defining the hours
hourly_columns <- names(production_data)[9:24]
```

Enhanced Forecast with SARIMA Models:

`forecast_with_detailed_arima`: Function to forecast using ARIMA models.

- **`data`**: The dataset.
- **`forecast_ahead`**: The number of periods to forecast.
- **`target_name`**: The column name (hour) to forecast.
- **`seasonal_freqs`**: Frequencies to consider for seasonality.
- **`eval(parse(text = command_string))`**: Dynamically creates the input series from the target column.
- **`scale`**: Standardizes the input series.
- **`auto.arima`**: Fits ARIMA models, attempting different seasonal frequencies.
- **`checkresiduals`**: Diagnoses the residuals of the fitted model.
- **`forecast`**: Produces forecasts from the best ARIMA model.
- **`forecasted_original`**: Converts the standardized forecast back to the original scale.

The `forecast_with_detailed_arima` function takes the dataset, the number of periods to forecast, the target column, and potential seasonal frequencies as inputs. First, it extracts and checks the target time series for missing values. It then standardizes the data to ensure consistent scaling, which prevents the model from producing erratic forecasts. Standardizing the data step became required after the submission day 3, in which our model gave a ridiculous prediction for H8 (the predicted value was 118 even though the highest ever observation of H8 was 10.35) and we found out that standardizing could help,

which it really did. The core of the function involves fitting SARIMA models with different seasonal frequencies (weekly, monthly, and yearly). It initializes the best model with the highest possible AIC (Akaike Information Criterion) and iterates through the provided seasonal frequencies. For each frequency, it fits a SARIMA model and selects the one with the lowest AIC, indicating the best fit to the data. If no suitable model is found, it stops and informs the user. The selected model's residuals are checked for issues, and the function then forecasts future values. The standardized forecast is converted back to the original scale for interpretability. Finally, the function returns the forecasted values and the best-fitting model.

```r
# Forecasting with SARIMA models with different seasonality values and detailed diagnostics
forecast_with_detailed_arima <- function(data, forecast_ahead, target_name, seasonal_freqs = c(7, 30, 365)) {
  command_string <- sprintf('input_series=data$%s', target_name)
  eval(parse(text = command_string))

  # Checking for actual missing values in the input series
  if (any(is.na(input_series))) {
    stop(sprintf("The input series %s contains missing values.", target_name))
  }

  # Standardizing the input series: this became required because the model started to gave
  # forecast in orders of 100 for H8 a few times, so we decided to normalize and de-normalize
  # the forecasts, which solved the issue.
  mean_val <- mean(input_series, na.rm = TRUE)
  sd_val <- sd(input_series, na.rm = TRUE)
  standardized_series <- scale(input_series)

  # Initializing performance measures
  best_aic <- Inf
  best_model <- NULL

  # Checking for weekly, monthly, and yearly seasonality periods and pick the best model
  for (freq in seasonal_freqs) {
    ts_input_series <- ts(standardized_series, frequency = freq)

    # Trying to fit a SARIMA model, catch potential errors
    fitted <- tryCatch(auto.arima(ts_input_series, seasonal = TRUE), error = function(e) NULL)

    if (!is.null(fitted) && fitted$aic < best_aic) {
      best_aic <- fitted$aic
      best_model <- fitted
    }
  }
  # Notifying if there are any problems
  if (is.null(best_model)) {
    stop(sprintf("Failed to fit a model for %s", target_name))
  }

  # Checking the residuals of the selected model
  checkresiduals(best_model)

  # Forecasting
  forecasted <- tryCatch(forecast(best_model, h = forecast_ahead), error = function(e) NULL)

  # Notifying if there are any problems
  if (is.null(forecasted)) {
    stop(sprintf("Failed to forecast for %s", target_name))
  }

  # Converting forecast back to original scale
  forecasted_original <- (forecasted$mean * sd_val) + mean_val

  return(list(forecast = as.numeric(forecasted_original), model = best_model))
}
```

Forecasting Each Hourly Column:

- **forecast_ahead**: Specifies that we are forecasting 1 period ahead.
- **results**: List to store forecasts for each hourly column.
- **for (h in hourly_columns)**: Loop over each hourly column to make forecasts.
- **cat(sprintf("Processing %s...\n", h))**: Prints the current column being processed.
- **tryCatch**: Handles any errors that occur during the forecasting process.

```r
# Defining the forecast period
forecast_ahead <- 1

# Initializing a list to store forecasts for each hourly column
results <- vector('list', length(hourly_columns))

# Iterating over each hourly column
for (h in hourly_columns) {
  cat(sprintf("Processing %s...\n", h))  # Debugging output
  # Making the forecast for the current hourly column
  forecast_result <- tryCatch(forecast_with_detailed_arima(production_data, forecast_ahead, h, seasonal_freqs = c(7, 30, 365)),
                              error = function(e) {
                                cat(sprintf("Error processing %s: %s\n", h, e$message))
                                return(NULL)
                              })

  if (!is.null(forecast_result)) {
    # Storing the forecast result in the list
    results[[h]] <- forecast_result$forecast
  } else {
    results[[h]] <- NA  # Handling failed forecasts
  }
}
```

Post Forecasting Processes:

- **unlist**: Converts the list of forecasts into a numeric vector.
- **hourly_forecasts_numeric[hourly_forecasts_numeric < 0] <- 0**: Ensures no forecasted value is negative, which wouldn't make sense in an energy-generating context.
- **hourly_forecasts_numeric[hourly_forecasts_numeric > 10] <- 10**: Caps the forecasted values at 10. Although historically, the production sometimes exceeds 10 by small amounts but it happens very rarely due to some government regulations, so we decided not to exceed the practical regulated capacity of 10.
- **c(rep(0, 4), hourly_forecasts_numeric, rep(0, 4))**: Adds padding of zeros at the beginning and end of the forecast vector.
- **as.character**: Converts the numeric forecasts to character strings.
- **writeClipboard**: Copies the forecasted values to the clipboard.

```r
# Transforming results to character and apply value limits
hourly_forecasts_numeric <- as.numeric(unlist(results))
hourly_forecasts_numeric

# Negative values are impossible and >10 values are exceptional, so we fixed them
hourly_forecasts_numeric[hourly_forecasts_numeric < 0] <- 0
max(hourly_forecasts_numeric)
hourly_forecasts_numeric[hourly_forecasts_numeric > 10] <- 10

# Adding the first 4 and last 4 hours predictions and copying them to clipboard
hourly_forecasts_numeric <- c(rep(0, 4), hourly_forecasts_numeric, rep(0, 4))
hourly_forecasts_character <- as.character(hourly_forecasts_numeric)
writeClipboard(hourly_forecasts_character)
```

To forecast hourly solar power production for the Edikli GES power plant, we started by loading the necessary libraries for time series data and date manipulation. We then read and prepared the production data, converting date formats and identifying columns for hourly production values.

We defined a function, forecast_with_detailed_arima, to fit SARIMA models and pick the best one, perform diagnostics, and make forecasts using historical data. This method avoids integrating complex external variables, reducing potential inaccuracies.

Using this function, we forecasted the next day's production for each hourly column, handling any errors that occurred. Finally, we adjusted the forecasts to ensure they were within reasonable bounds and converted them to a usable format. With this code, we are forecasting 1 season ahead.

We considered several potential seasonal frequencies to capture different patterns in the data. Specifically, we are using:

1. **Weekly Seasonality (7 days)**: This frequency is used to capture any weekly patterns in the solar power production data. For example, differences in production between weekdays and weekends possibly due to regulative or anthropogenic reasons.
2. **Monthly Seasonality (30 days)**: This frequency helps in identifying monthly trends or cycles that might affect solar power production, such as changes in weather patterns or seasonal variations in sunlight.
3. **Yearly Seasonality (365 days)**: This frequency is usually selected by the hours that are mostly affected by the lack of sunlight in the winter (H7, H15, H16, H17). Such a seasonality incorporates day-of-the-year data to account for seasons. For example, days between 151:243 would be in summer, which could be correlated via climate. We

could also use season-of-the-year seasonality but it would result in jumps in the season-changing days like 31th of May and 1th of June.

The model with the best AIC (Akaike Information Criterion) value is selected as the final model for forecasting which works by balancing the trade-off between the complexity of a model and its goodness of fit to the data. It penalizes models for being too complex, thus discouraging overfitting, while also rewarding models that fit the data well.

Why Exclude 8 Hourly Columns:

Hours H0, H1, H, H3, H20, H21, H22, and H23 were not included since our observation of historical production data showed that without an exception, these hours summed up to 0 production in any of the months. We think that the reasons can be regulatory or related to power grid complexities.


```
# Defining the hours
hourly_columns <- names(production_data)[9:24]
```

```
# Adding the first 4 and last 4 hours predictions and copying them to clipboard
hourly_forecasts_numeric <- c(rep(0, 4), hourly_forecasts_numeric, rep(0, 4))
```

4. Results

Ljung-Box Test and p-values

The Ljung-Box test checks whether there are significant autocorrelations in the residuals of a time series model. Here's a more precise interpretation:

critical p-value is (0.05)

🎬 **p-value (> 0.05)**: Indicates that the residuals are likely to be white noise, meaning the model has accounted for the autocorrelations in the data adequately. This is a **good** indication that the model is fitting well.

🎬 **p-value (≤ 0.05)**: Indicates that there is significant autocorrelation remaining in the residuals, meaning the model has not captured all the patterns in the data. This is a **bad** indication, suggesting the model is inadequate.

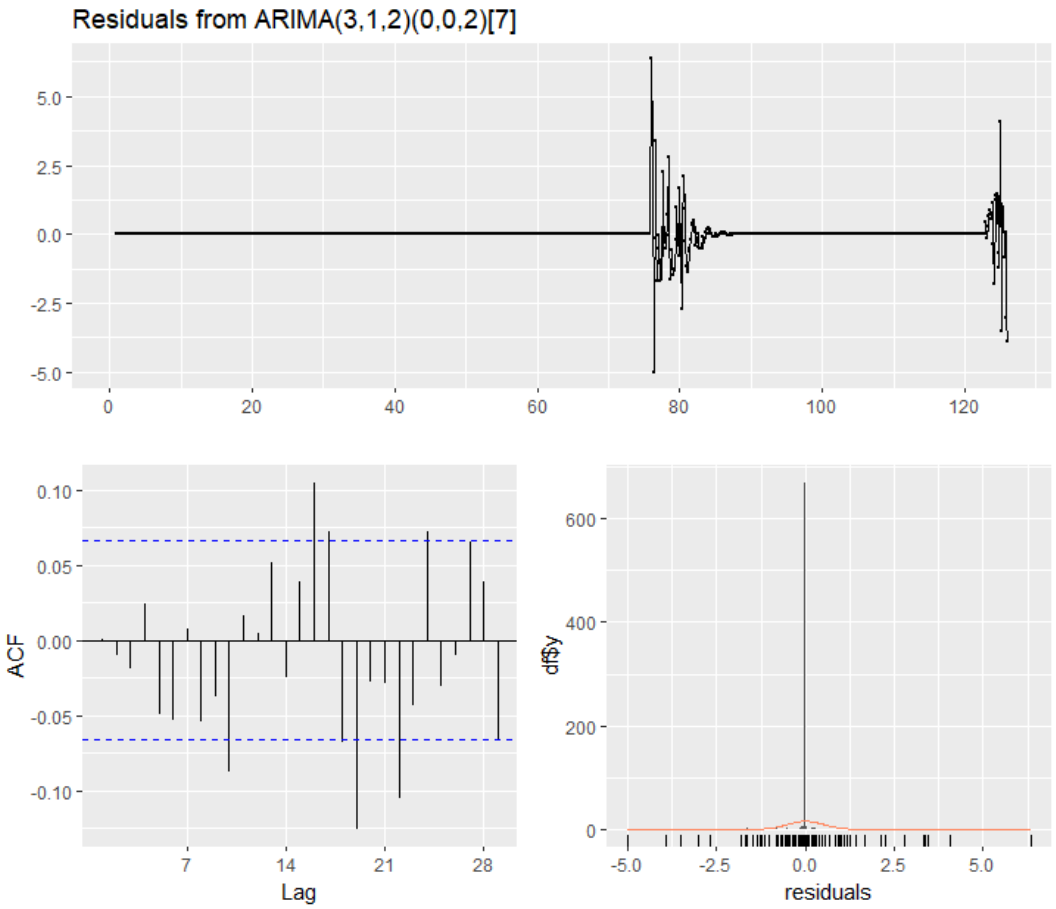
Hourly Prediction Results:

```
Processing H4...
```

H4:

- **Model**: ARIMA(3,1,2)(0,0,2)[7]
- **Ljung-Box Test**: Q∗=19.33, df=7, p−value=0.007214
- **Interpretation**: The low p-value (< 0.05) suggests that the residuals are not white noise, indicating that the model might be missing some patterns in the data. However, we didn't mind the autocorrelation since although all observations were not 0's, there are wide chains of 0's that explain the autocorrelation. Also, they averaged out in <0.001 so they wouldn't contribute significantly to the overall WMAPE.

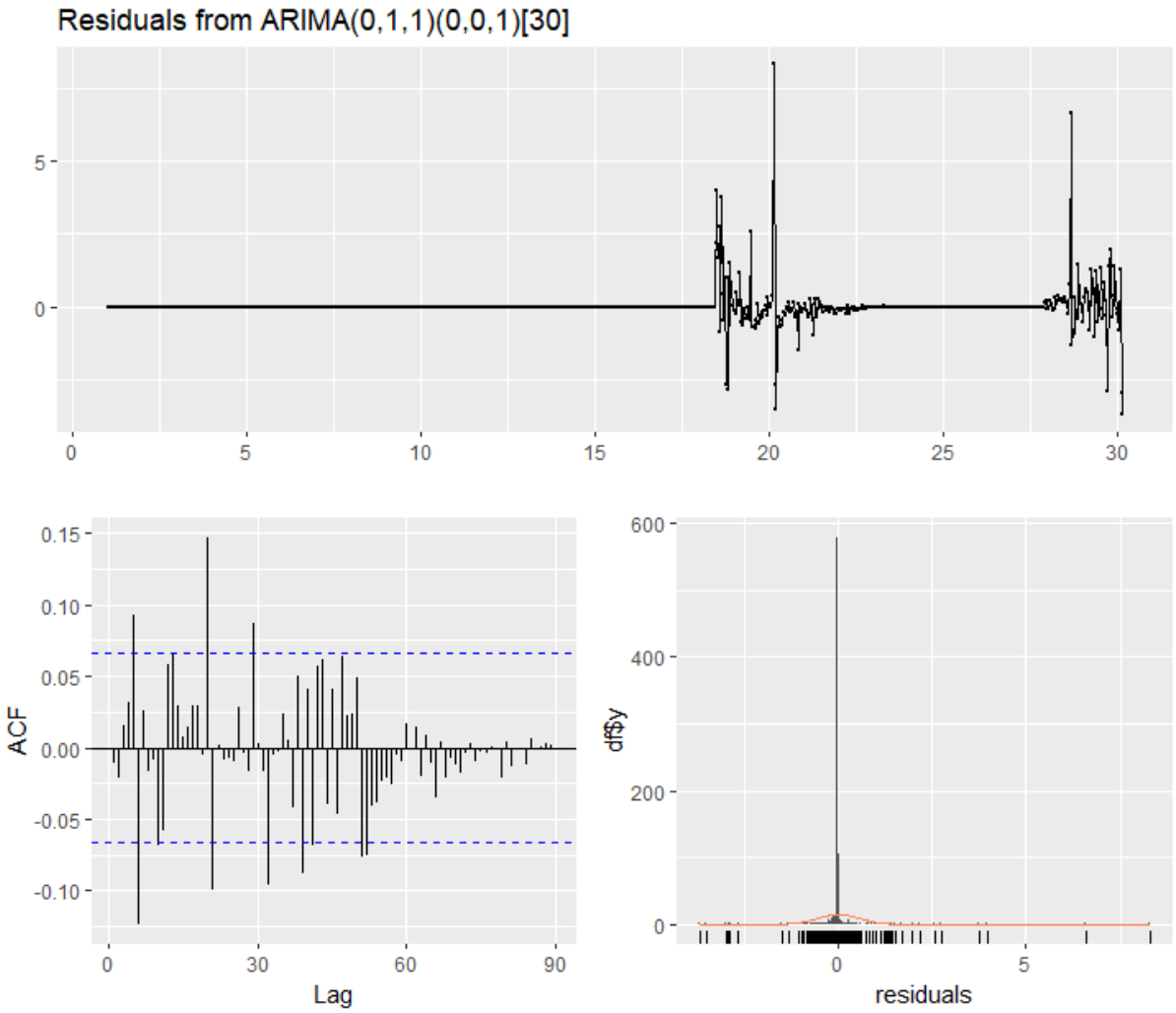
```
Average: 0.005216895    Count: 877    Sum: 4.57
```



H5:

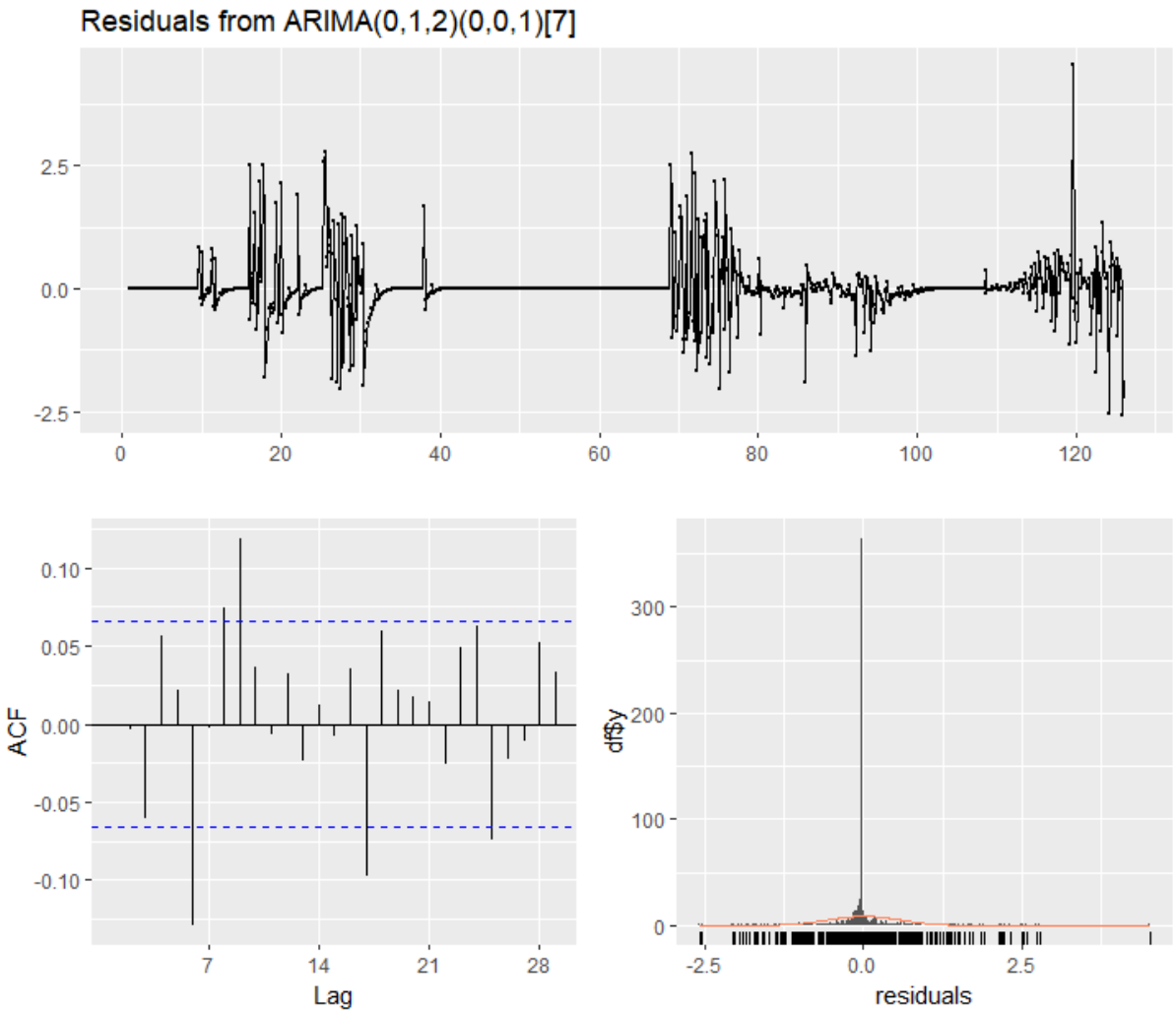
- **Model**: ARIMA(0,1,1)(0,0,1)[30]
- **Ljung-Box Test**: Q∗=136.94, df=58, p−value=2.481e−08
- **Interpretation**: The very low p-value indicates significant autocorrelation in the residuals, suggesting that the model does not adequately capture the data's structure. However, the same logic as H4 applied to H5 too since their wide chains of 0's and small average, so we didn't mind the p-value.

Average: 0.079965753	Count: 877	Sum: 70.05



H6:

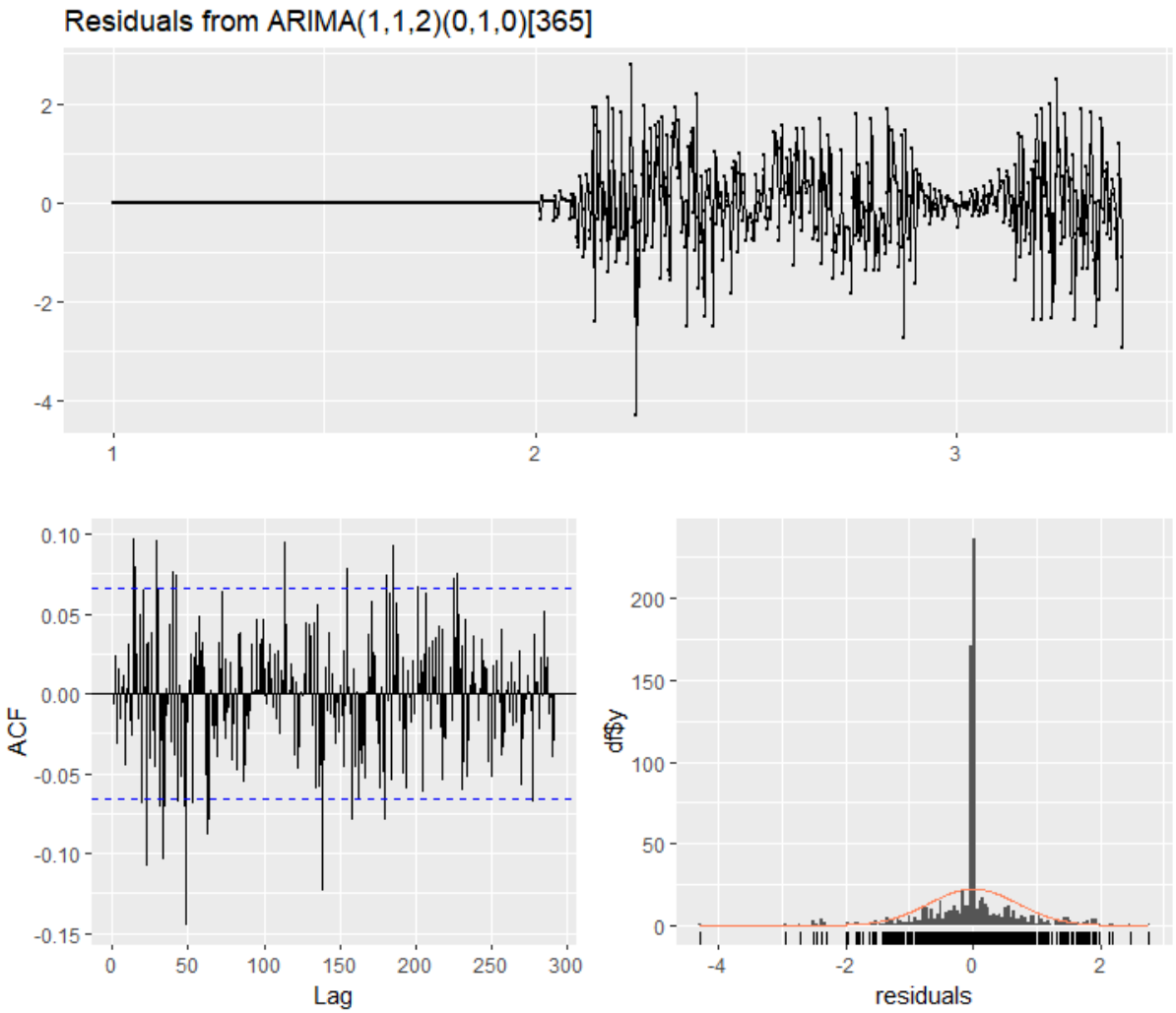
- **Model**: ARIMA(0,1,2)(0,0,1)[7]
- **Ljung-Box Test**: Q∗=41.49, df=11, p−value=1.984e−05
- **Interpretation**: The low p-value indicates that the residuals are autocorrelated, suggesting the model is insufficient. Although their average were not as small as H4 and H5, wide chains of 0's can still explain the p-value.



H7:

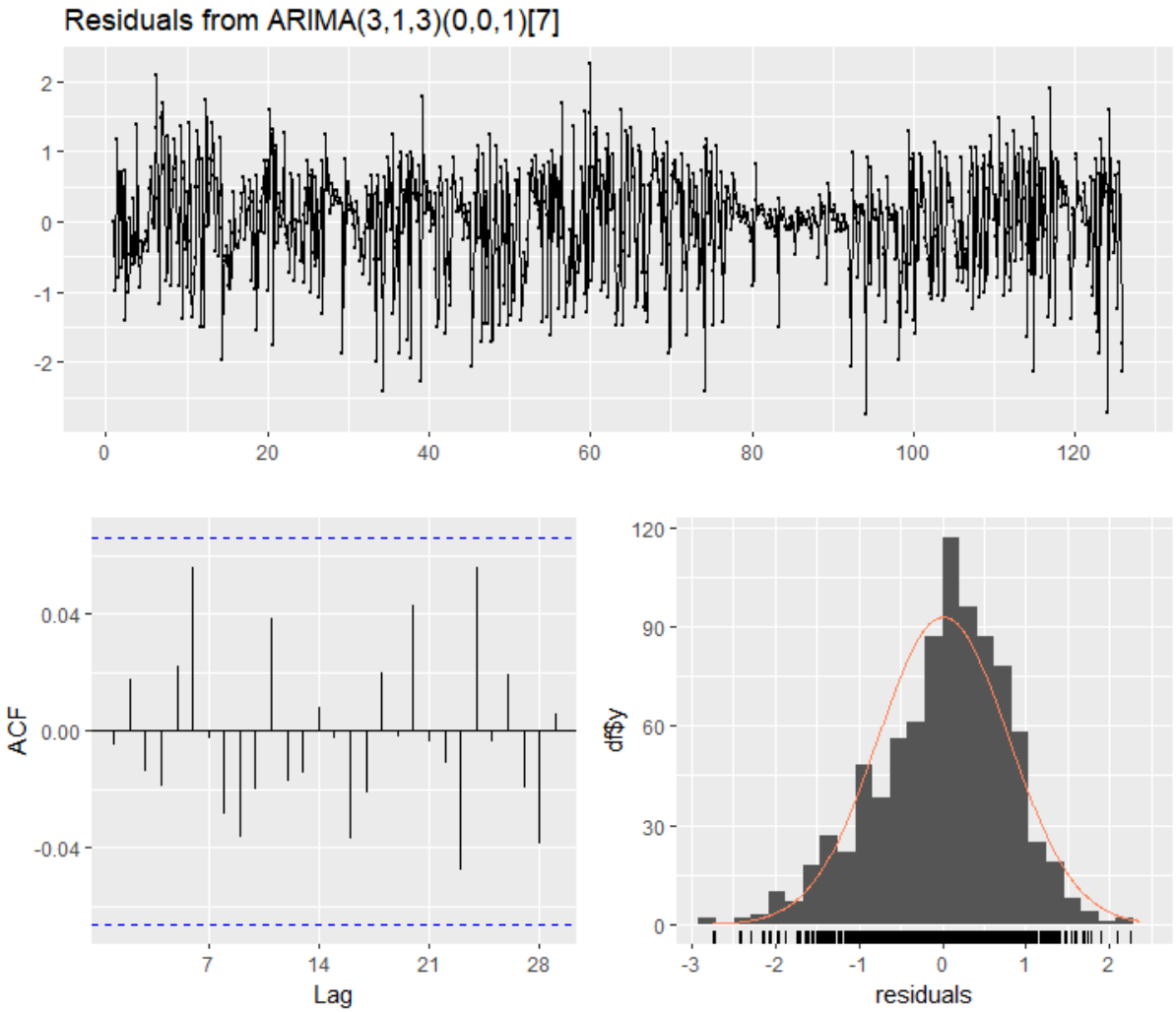
- **Model**: ARIMA(1,1,2)(0,1,0)[365]
- **Ljung-Box Test**: Q∗=295.13, df=172, p−value=1.553e-08
- **Interpretation**: The low p-value indicates that the residuals are autocorrelated, suggesting the model is insufficient. Although the average was significant, wide chains of 0's in the winter months can still explain the p-value.

Average: 2.852009132 Count: 877 Sum: 2498.36



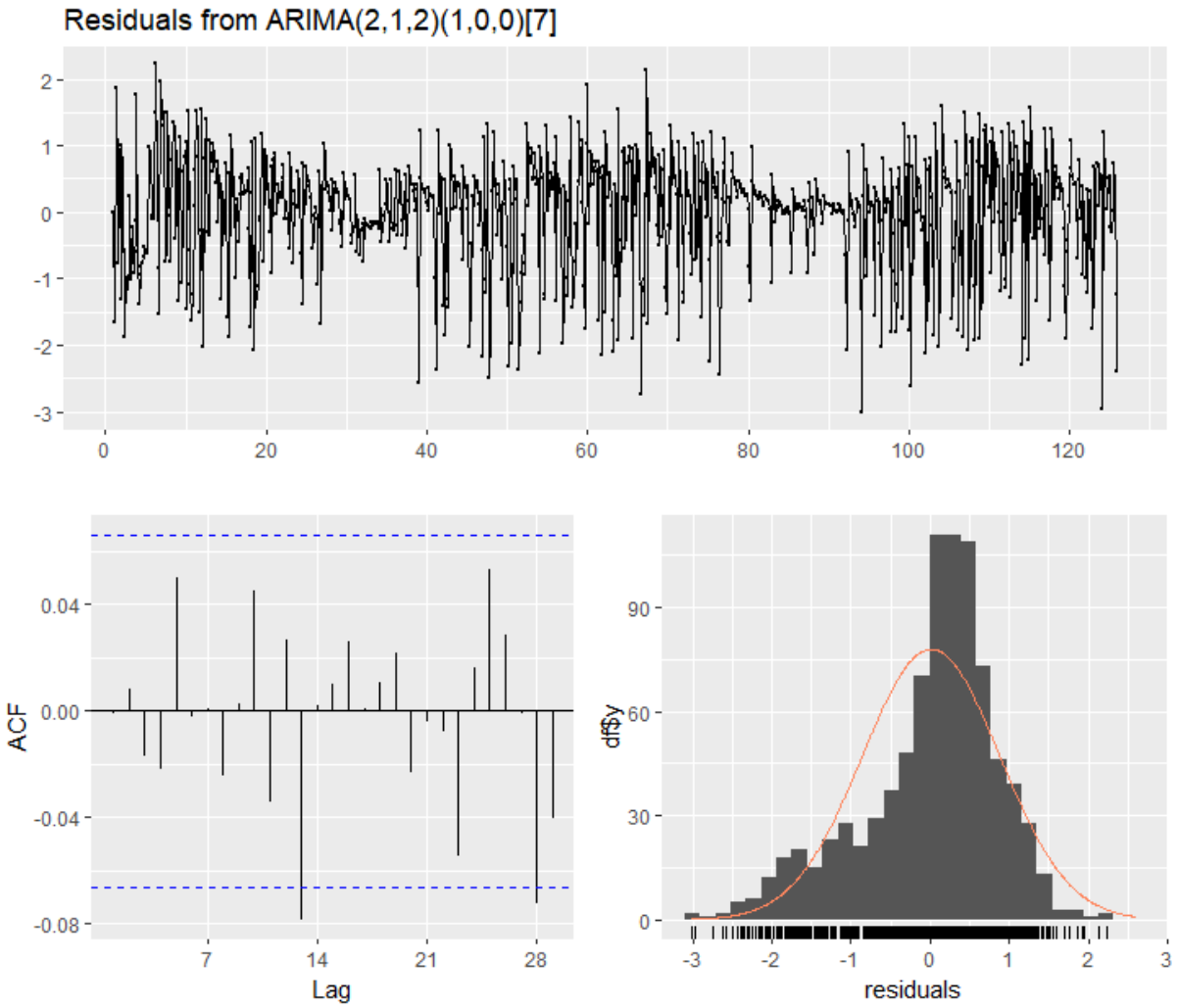
H8:

- **Model**: ARIMA(3,1,3)(0,0,1)[7]
- **Ljung-Box Test**: Q∗=8.1117, df=7 , p−value=0.3228
- **Interpretation**: The higher p-value indicates that the residuals do not show significant autocorrelation, suggesting a reasonable model fit.



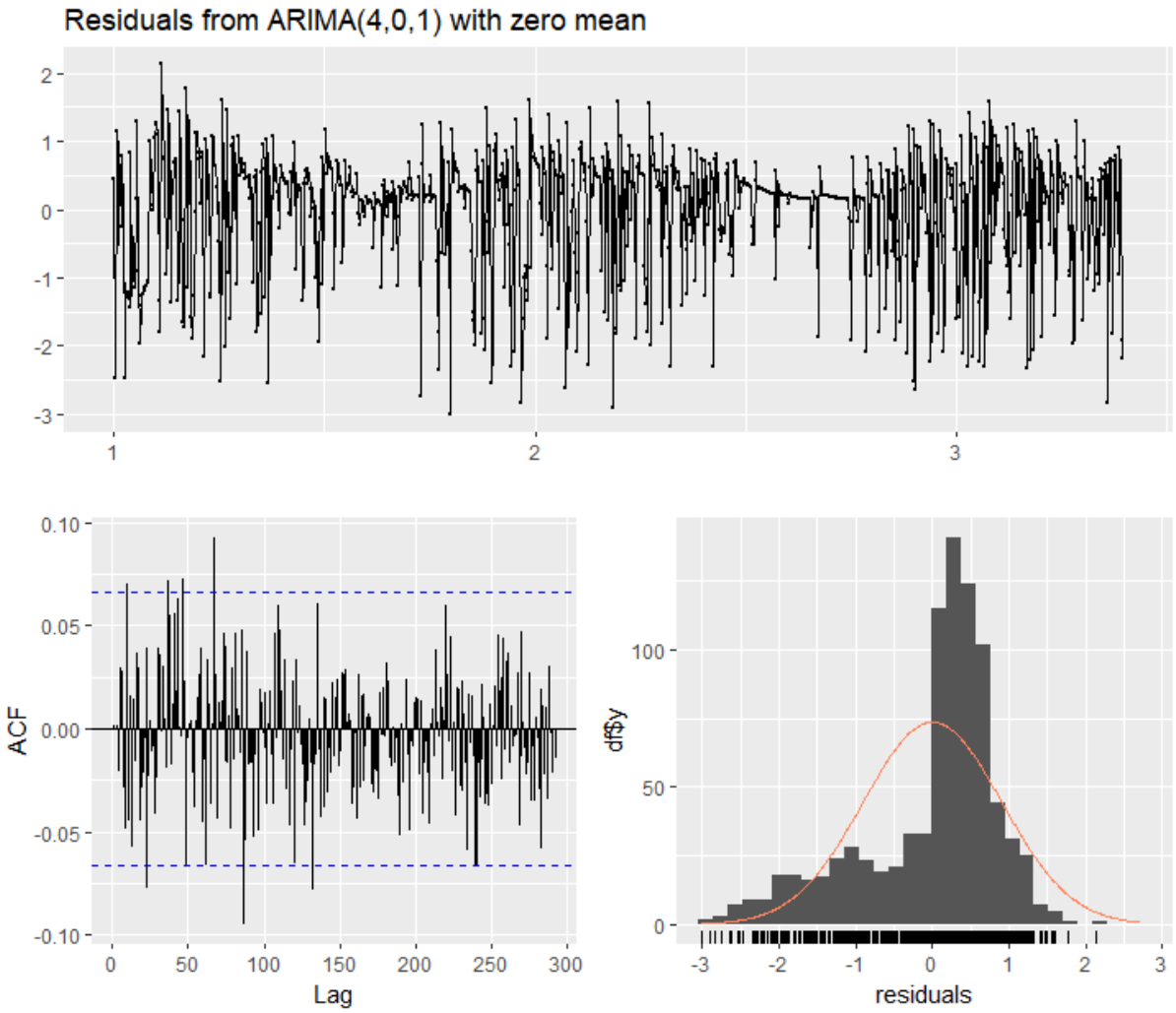
H9:

- **Model**: ARIMA(2,1,2)(1,0,0)[7]
- **Ljung-Box Test**: Q∗=12.5, df=9, p−value=0.1866
- **Interpretation**: The p-value indicates that the residuals are close to white noise, suggesting a good model fit.



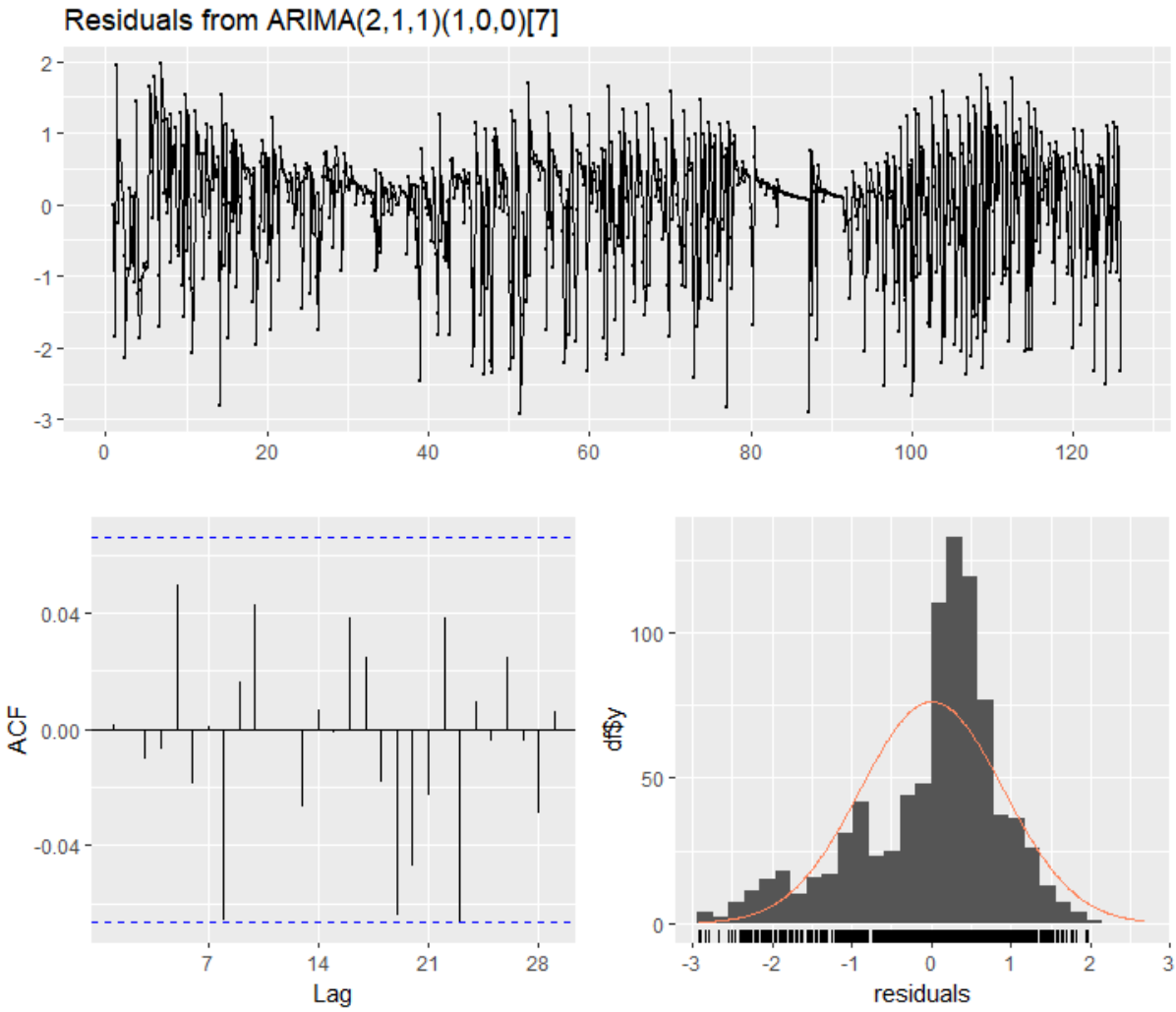
H10:

- **Model**: ARIMA(4,0,1) with zero mean
- **Ljung-Box Test**: Q∗=178.94 df=170, p−value=0.304
- **Interpretation**: The p-value suggests that the residuals are likely white noise, indicating a reasonable model fit.



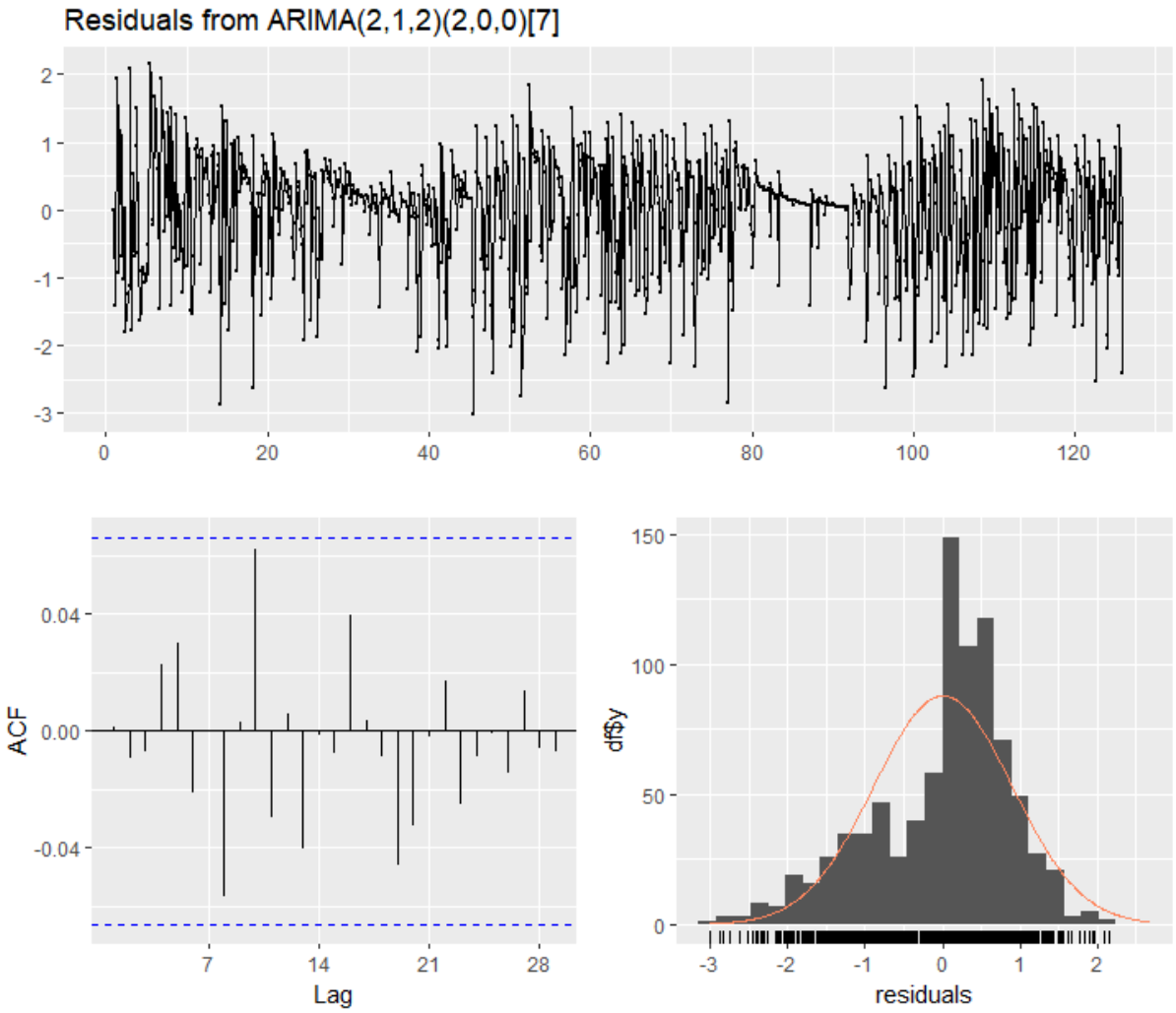
H11:

- **Model**: ARIMA(2,1,1)(1,0,0)[7]
- **Ljung-Box Test**: Q∗=8.9348, df=10, p−value=0.5383
- **Interpretation**: The high p-value indicates that the residuals are white noise, suggesting a good model fit.



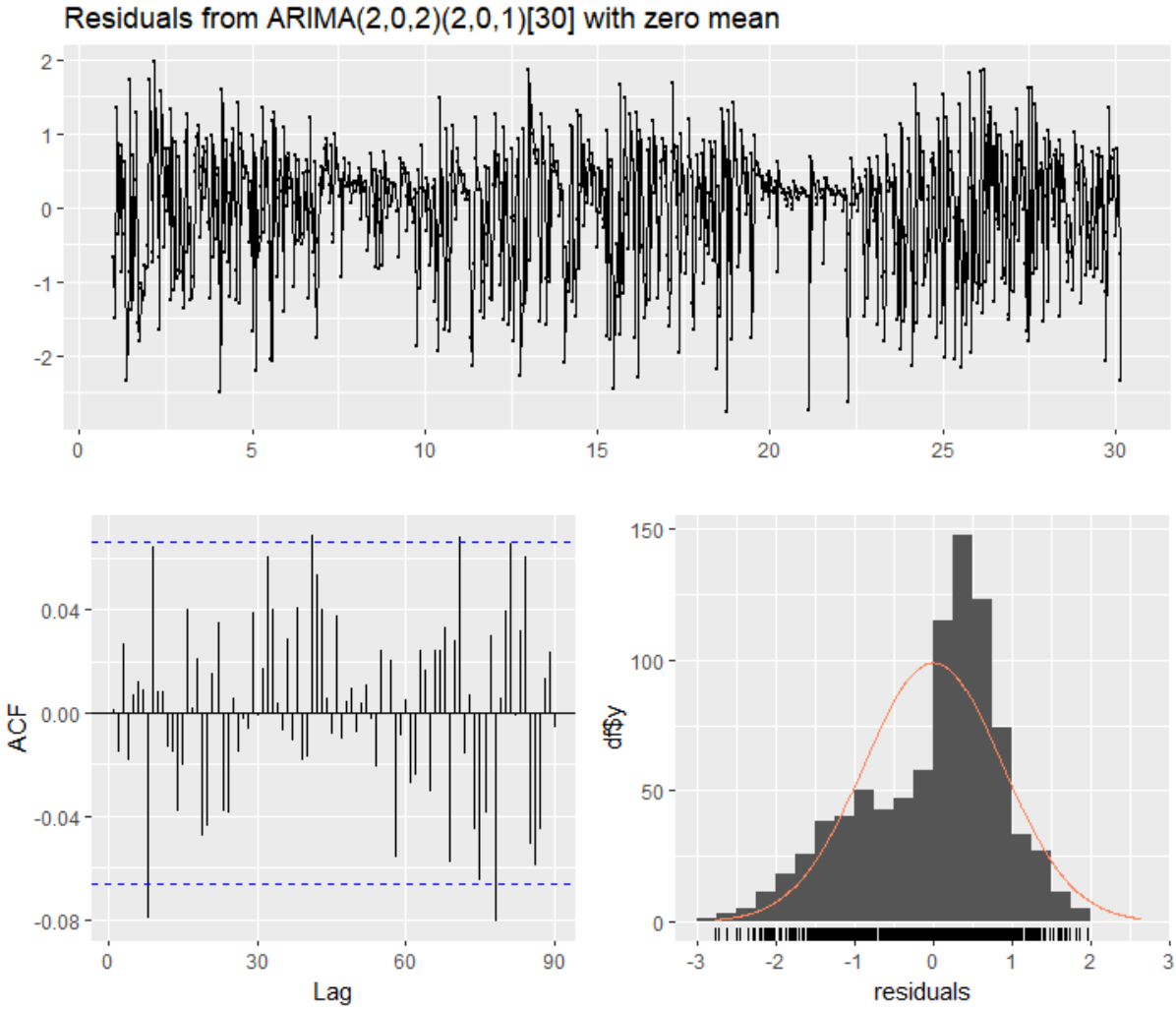
H12:

- **Model**: ARIMA(2,1,2)(2,0,0)[7]
- **Ljung-Box Test**: Q∗=10.338 df=8, p−value=0.2421
- **Interpretation**: The p-value suggests that the residuals are not significantly autocorrelated, indicating a reasonable model fit.



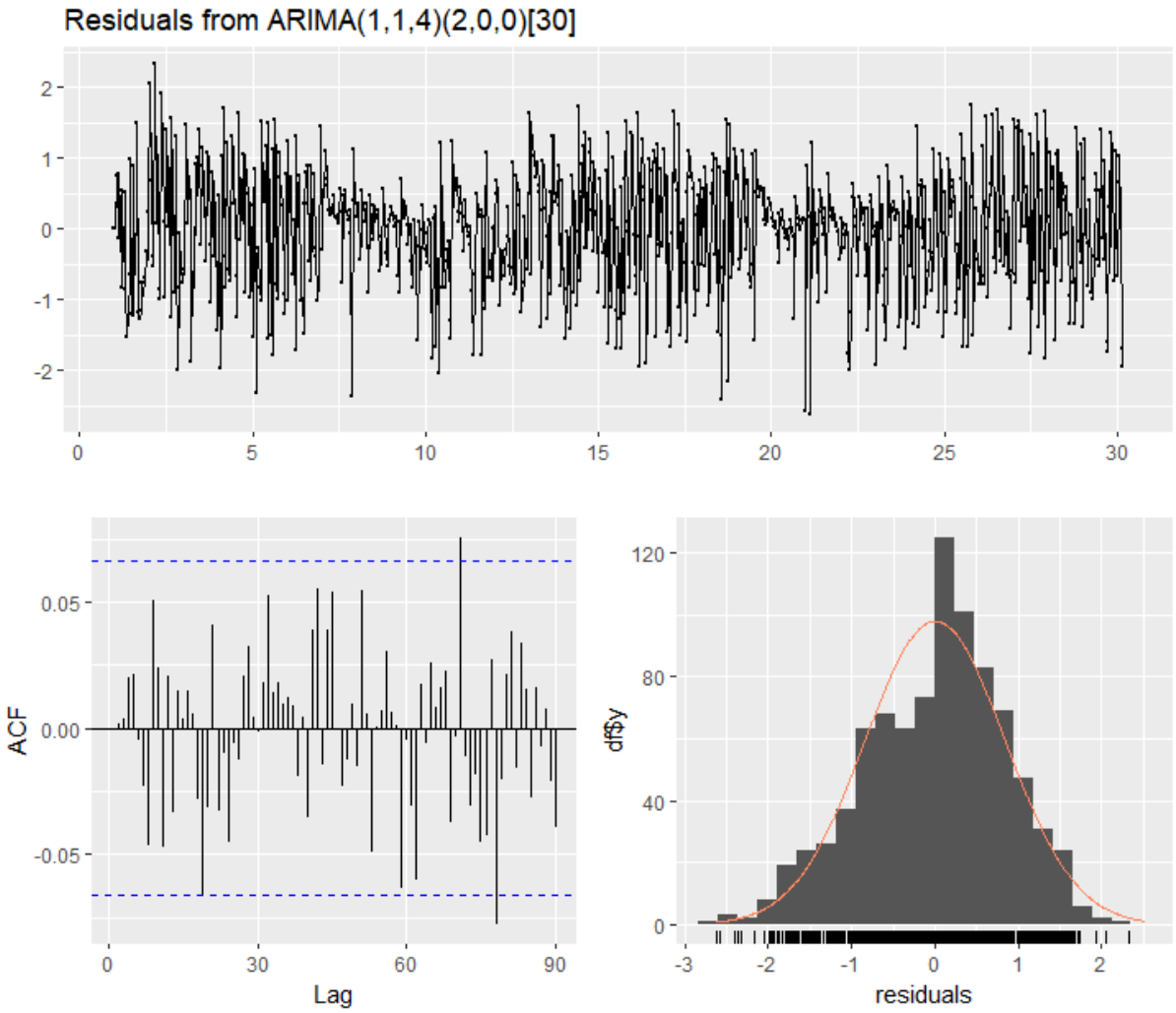
H13:

- **Model**: ARIMA(2,0,2)(2,0,1)[30] with zero mean
- **Ljung-Box Test**: Q∗=46.909 df=53, p−value=0.709
- **Interpretation**: The high p-value indicates that the residuals are white noise, suggesting a good model fit.



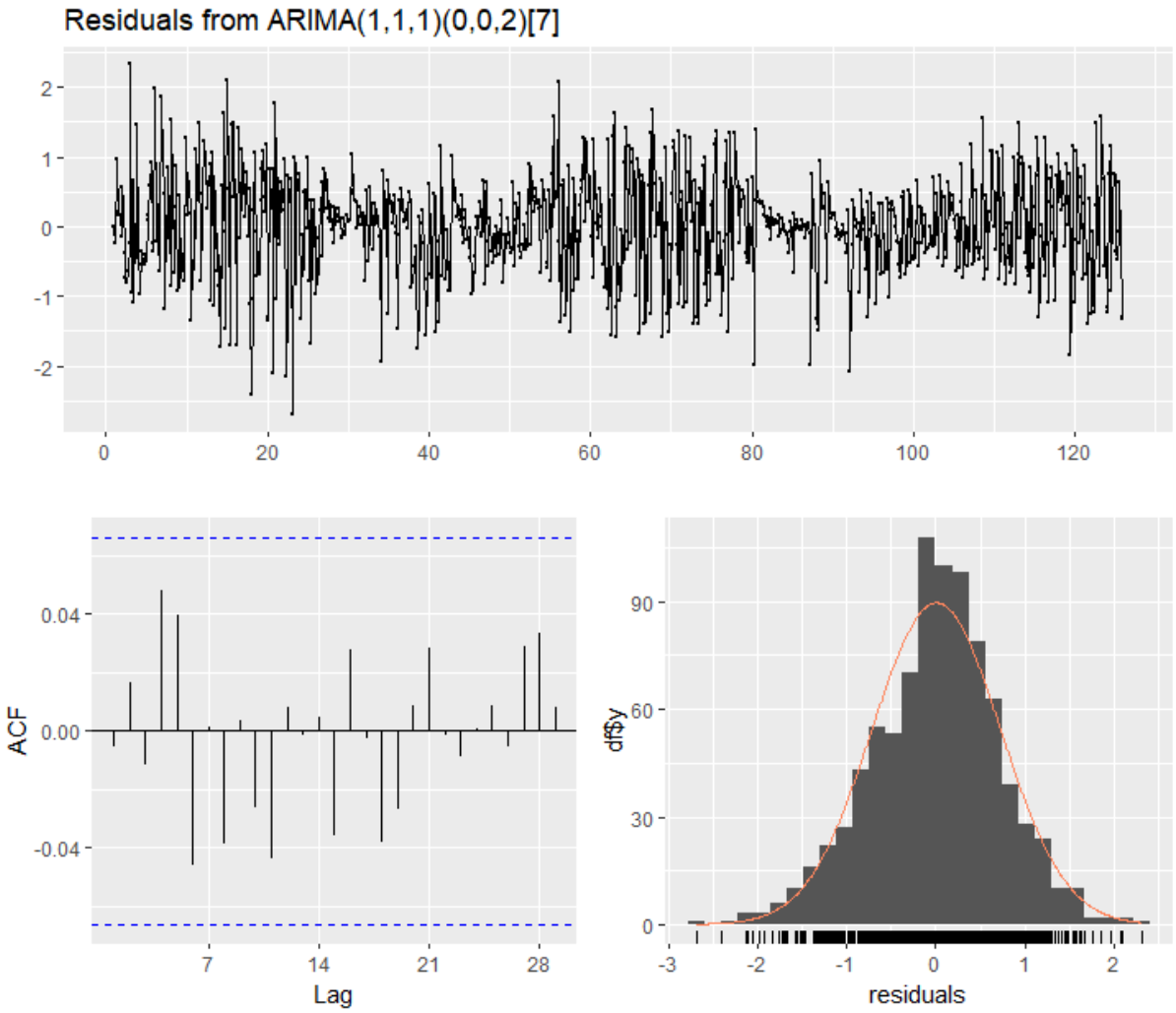
H14:

- **Model**: ARIMA(1,1,4)(2,0,0)[30]
- **Ljung-Box Test**: Q∗=45.615, df=53, p−value=0.7543
- **Interpretation**: The high p-value suggests that the residuals are white noise, indicating a good model fit.



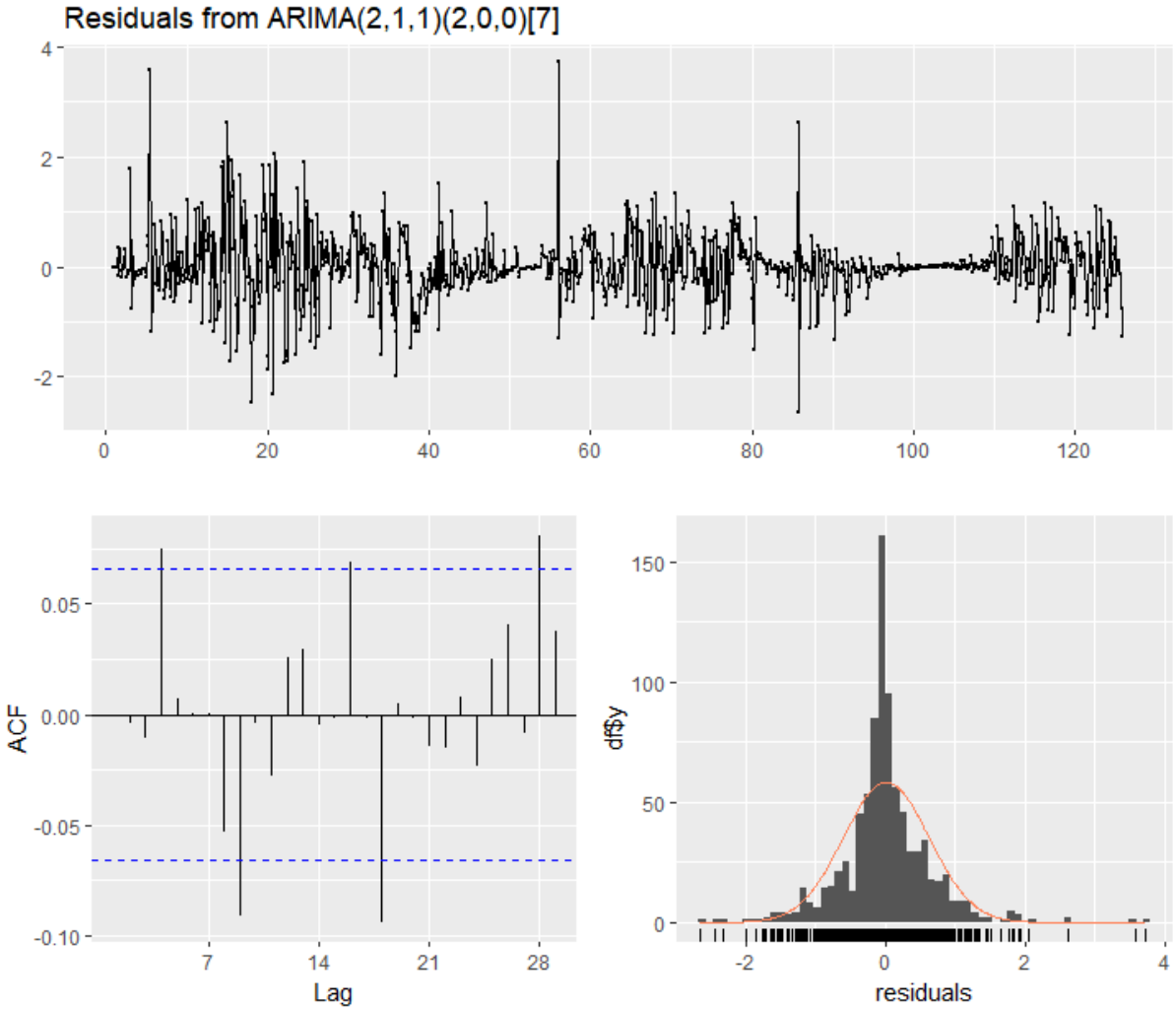
H15:

- **Model**: ARIMA(1,1,1)(0,0,2)[7]
- **Ljung-Box Test**: Q∗=9.432, df=10, p−value=0.4917
- **Interpretation**: The p-value suggests that the residuals are likely white noise, indicating a reasonable model fit.



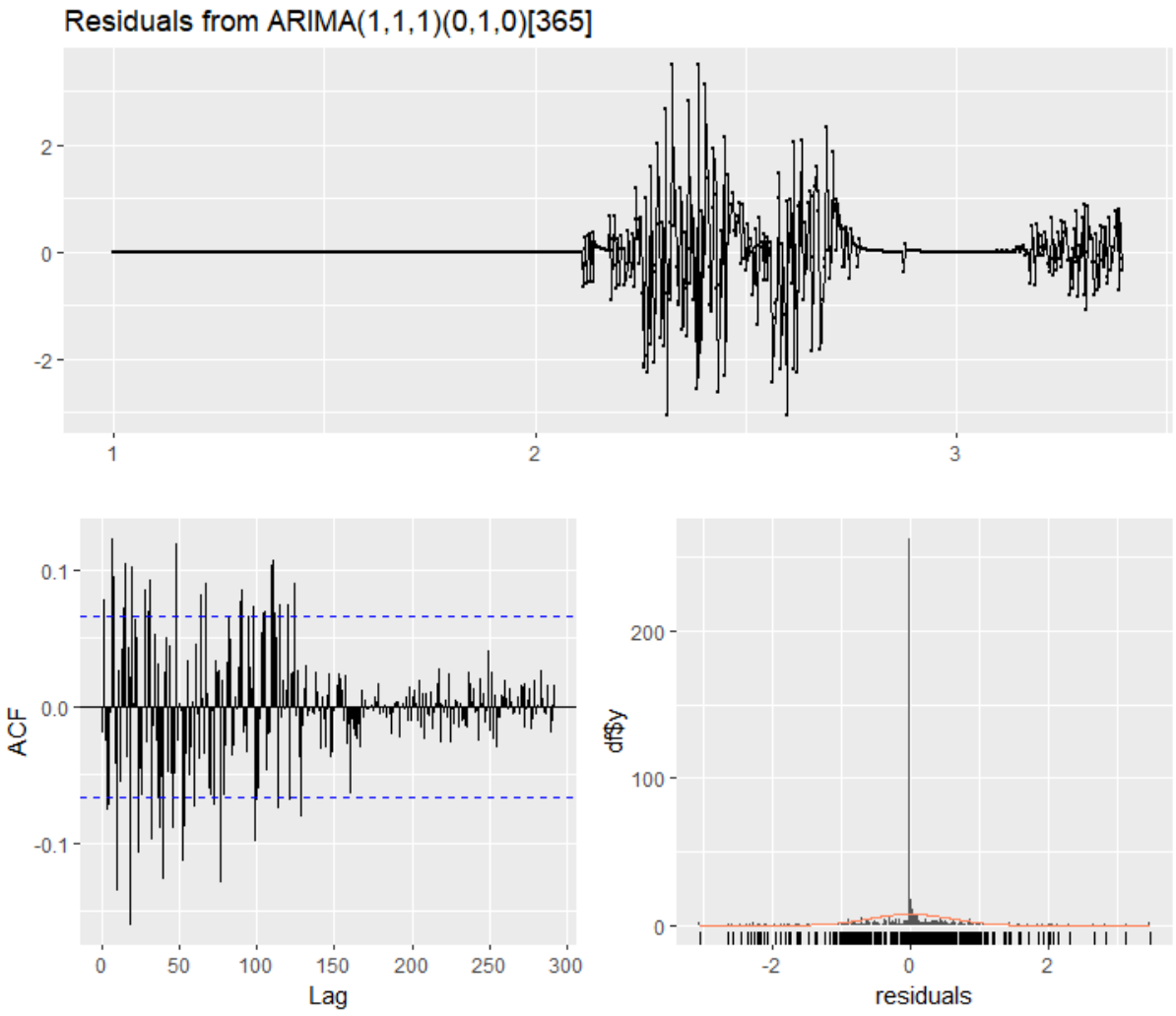
H16:

- **Model**: ARIMA(2,1,1)(2,0,0)[7]
- **Ljung-Box Test**: Q∗=17.046, df=9 , p−value=0.048
- **Interpretation**: The p-value is marginally low, suggesting some autocorrelation in the residuals, indicating the model might be slightly inadequate. Still, the autocorrelation can be explained by the chains of 0's in the winter months.



H17:

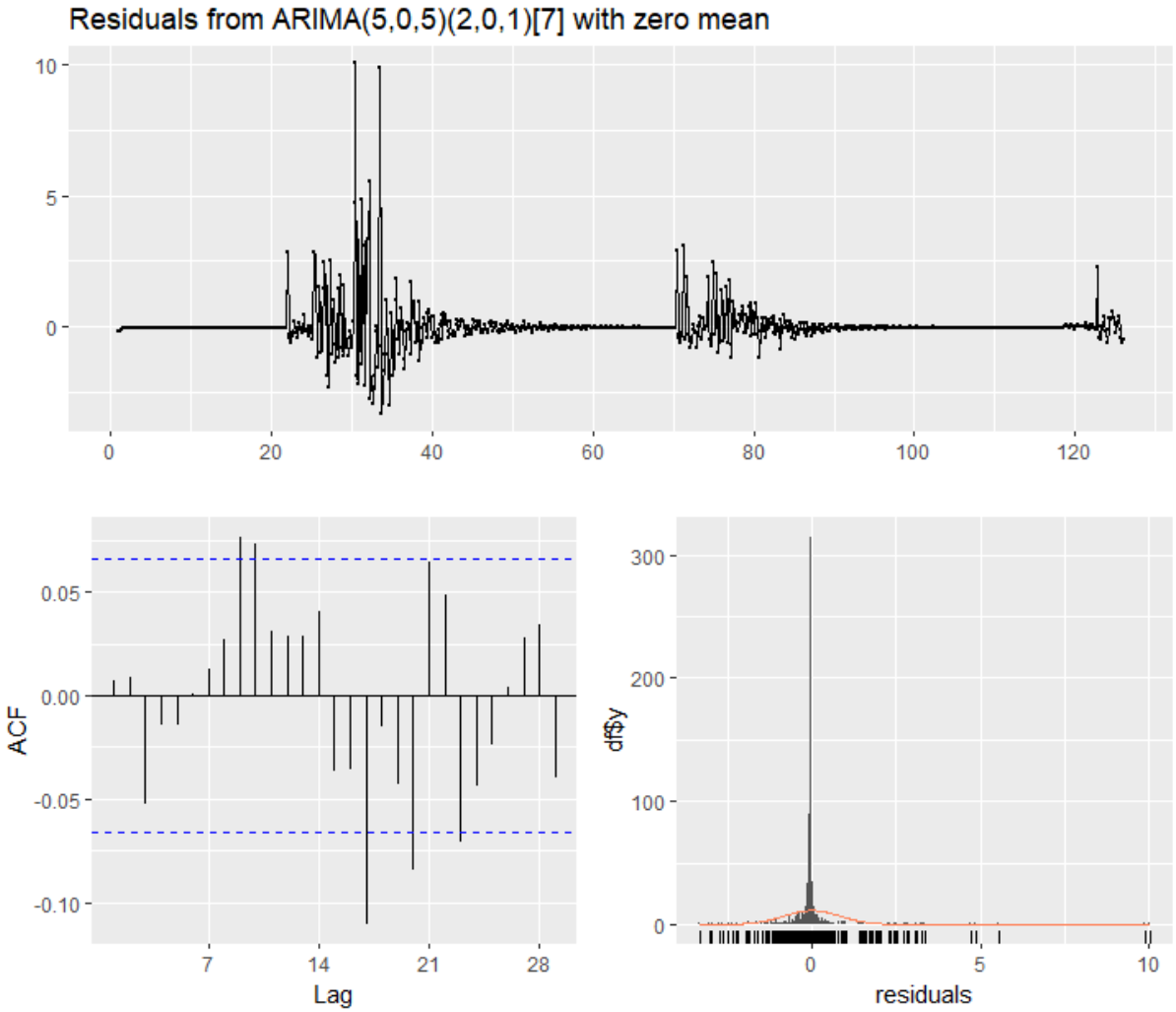
- **Model**: ARIMA(1,1,1)(0,0,2)[7]
- **Ljung-Box Test**: Q∗=471.17 df=173, p−value=2.2e-16
- **Interpretation**: The low p-value indicates that the residuals are autocorrelated, suggesting the model is insufficient. However, wide chains of 0's can still explain the p-value. In addition, the average makes the possible errors negligible.



H18:

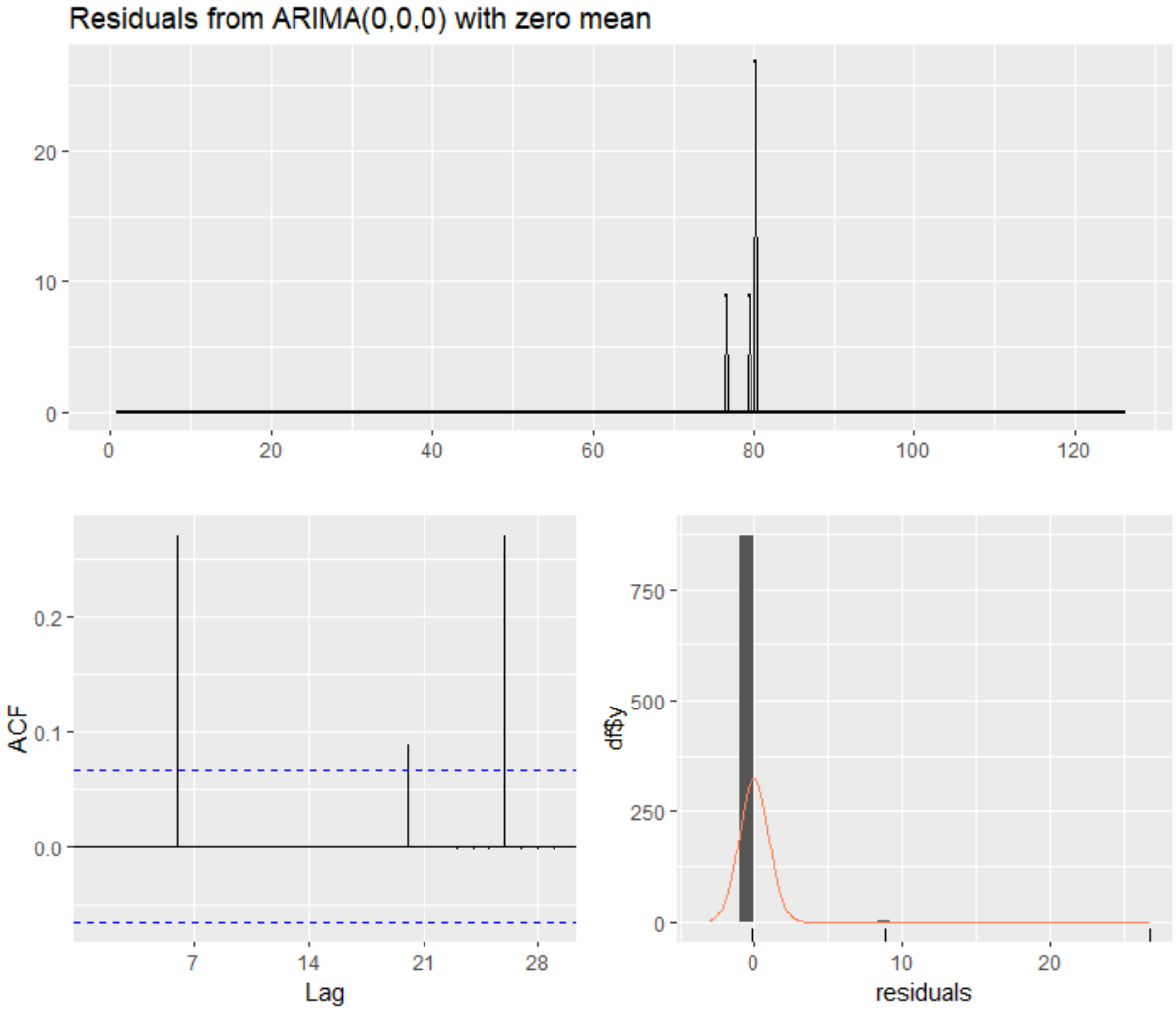
- **Model**: ARIMA(5,0,5)(2,0,1)[7] with zero mean
- **Ljung-Box Test**: Q∗=19.797, df=3 , p−value=0.000187
- **Interpretation**: The very low p-value indicates significant autocorrelation in the residuals, suggesting that the model does not adequately capture the data's structure. Still, wide chains of 0's explain the autocorrelation, and the low average makes it not a problem.

Average: 0.070388128 Count: 877 Sum: 61.66



H19:

- **Model**: ARIMA(0,0,0) with zero mean
- **Ljung-Box Test**: Q∗=64.918, df=14, p−value=1.582e−08
- **Interpretation**: The very low p-value suggests that the residuals are highly autocorrelated, indicating that this model is inadequate for the data. H19 insists on almost solely 0's, so there's no unexplained autocorrelation nor a problem.



ARIMA VS SARIMA:

In this section, we compared some hourly predictions with ARIMA and SARIMA models. SARIMA models are more flexible since they also have seasonal components but also require more computational time. Also, if there's little to no seasonal effect in the data, SARIMA models have a risk of overfitting. On the other hand, ARIMA models are simpler and less time-consuming, but they tend to fail more while addressing seasonal effects if there are any.

Arima H12 Result:

```
data:  Residuals from ARIMA(2,1,4) with drift
Q* = 8.7357, df = 4, p-value = 0.06805

Model df: 6.   Total lags used: 10
```

Sarima H12 Result:

```
data:  Residuals from ARIMA(2,1,2)(2,0,0)[7]
Q* = 10.338, df = 8, p-value = 0.2421

Model df: 6.   Total lags used: 14
```

Arima H19 Result:

```
data:  Residuals from ARIMA(0,0,0) with zero mean
Q* = 64.893, df = 10, p-value = 4.249e-10

Model df: 0.   Total lags used: 10
```

Sarima H19 Result:

```
data:  Residuals from ARIMA(0,0,0) with zero mean
Q* = 64.918, df = 14, p-value = 1.582e-08

Model df: 0.   Total lags used: 14
```

Ranking:

Group	26-May	25-May	24-May	23-May	22-May	21-May	20-May	19-May	18-May	17-May	16-May	15-May	14-May	13-May	Mean Daily Rank
Group-20	14	20	21	20	18	3	17	9	13	3	3	5	2	1	10.64

We averaged a mean daily rank of 10.64 during the competition phase with our predictions with the SARIMA model that's discussed above. We were the best total-ranking group until 20-21 May but eventually, we were passed by other groups. Although we kept a relatively consistent WMAPE throughout the competition period, our competitors managed to lower their WMAPEs in the later days to great levels.

Weighted Mean Average Percentage Errors:

	26-May	25-May	24-May	23-May	22-May	21-May	20-May	19-May	18-May	17-May	16-May	15-May	14-May	13-May	# Days w/ submissions
Group-20	0.1856	0.8234	0.9261	0.2409	0.2358	0.2336	0.3015	0.2105	0.2323	0.2102	0.2796	0.2091	0.2606	0.1857	14

25th and 24th Of May WMAPE values are incredibly high and out of order. However, WMAPE values of these days are high for all competing groups suggesting an unusual behavior that couldn't be predicted well with previous data. This could be explained such that the total production on 24 May is 34.37 whereas the weekly average before that day was 76.16, and 25 May consists of only 0's. Still, our error deviation is pretty high even in comparison. Besides these two days, we kept a consistent WMAPE between 18% and 30%.

5. Ways To Improve The Model

We can utilize the weather data to make a SARIMAX (Seasonal Autoregressive Integrated Moving Average + eXogenous variables) model to incorporate the effects of rain, snow, cloud, or any other relevant parameter. These variables are mostly independent and can give crucial information about sudden drops in production, which our SARIMA models failed to address on some days during the submission days. Also, not incorporating them may have deteriorated our seasonal components since our model most likely tried to explain the drops in production that are caused by weather conditions with complex and not very realistic seasonalities. Creating a SARIMAX model can help us foresee some of the deviations of observations from SARIMA predictions and give more realistic model parameters (p, d, q, P, D, Q, s).

6. Conclusion

In this project, we aimed to develop a statistical model to predict the hourly solar power output of the Edikli GES power plant in Niğde, Turkey, for the following day using SARIMA (Seasonal Autoregressive Integrated Moving Average) models. The provided code demonstrates how we constructed the SARIMA model and obtained the hourly forecasts for each day ahead from the 13th to the 26th of May. Instead of integrating dependent variables to the time series regression we used autoregressive components, moving average components, differencing, and seasonality integration to the other components, all of which are based on the historical data of energy production.

Through rigorous testing and validation, we evaluated the performance of our model using metrics such as the Ljung-Box test and the Weighted Mean Absolute Percentage Error (WMAPE). While our model demonstrated reasonable accuracy in predicting solar power production, there is still room for improvement regarding the incorporation of exogenous variables (SARIMAX modeling) such as rainfall, cloud cover, and other relevant weather parameters. By integrating these variables into our model, we can better capture the impact of weather conditions on solar power production and improve the accuracy of our forecasts.