Project Report of Group09

Cemil Meriç Taşgıran - 2020402027

Osman Baran Koç - 2020402192

Vüsal İbrahimli - 2020402249

1. Introduction

The primary aim of this research is to construct a comprehensive forecasting model for hourly solar power production at Edikli GES, a solar power plant strategically positioned in Niğde, Turkey, within the geographic coordinates of 37.75-38.75° North latitude and 34.5-35.5° East longitude. This study targets a prediction timeframe spanning from May 13th to May 26th, and employs a dataset that includes historical production data paired with contemporaneous meteorological conditions. Crucially, this dataset comprises multiple weather variables such as downward shortwave radiation flux (DSWRF_surface), temperature at the surface (TMP_surface), and total cloud cover across different atmospheric layers (TCDC_low.cloud.layer, TCDC_middle.cloud.layer, TCDC_high.cloud.layer), collected from 25 distinct grid points near the facility. These variables are integral to understanding the factors influencing solar energy production, as fluctuations in temperature, solar radiation, and cloud cover can significantly affect panel efficiency and, consequently, energy output.

This research follows a rigorous analytical methodology, combining statistical techniques and time-series forecasting to develop accurate prediction models. The initial phase of data handling addressed gaps in the production data by calculating averages from the days surrounding each missing value. Subsequent exploratory data analysis aimed to identify trends and outliers through the visualization of total daily production volumes.

Further analytical steps will include regression analysis to determine the relationships between environmental conditions and power output, followed by the implementation of time-series forecasting methods to predict future production levels. The ultimate goal of this analytical framework is to optimize the accuracy of solar power generation forecasts at Edikli GES. By leveraging a thorough integration of historical production data and real-time environmental metrics, this project seeks to significantly enhance the predictability and reliability of solar energy forecasts, thereby aligning closely with the operational needs of energy traders in the real-time energy market.

```
install.packages("data.table")

install.packages("skimr")
```

```
install.packages("GGally")
install.packages("forecast")
install.packages("rpart")
install.packages("openxlsx")

library(data.table)
library(ggplot2)
library(readxl)
library(skimr)
library(GGally)
library(tidyr)
library(lubridate)
library(dplyr)
library(forecast)
library(rpart)
library(openxlsx)
```

This line assigns the file path of the weather data CSV file to the variable data_weather and production. It indicates where the file is located on the user's computer. Uses the fread function from the data.table package to read the CSV file specified by data_weather into an R data frame called weather. The fread function is efficient and fast, making it suitable for reading large data sets.

```
data_weather <- C:/Users/Vusal Ibrahimli/OneDrive/Desktop /processed_weather.csv'
weather <- fread(data_weather)
data_path <- C:/Users/Vusal Ibrahimli/OneDrive/Desktop /production.csv'
data <- fread(data)
production <- data$production
weather_data <- data[, !(names(data) %in% c("production", "date", "hour"))]
```

LONG-WIDE FORMAT

Manipulating and merging weather and production data sets. I'll explain each step of the process:

1- The setDT() function from the data.table package converts the weather data frame into a data table. This conversion is done for more efficient data manipulation capabilities that data.table provides, such as modifying data by reference which is faster and uses less memory than typical data frames.

2- Within the weather data table, this line transforms the date column to a Date object using the ymd() function from the lubridate package, which interprets strings in "year-month-day" format as Date objects.

3- Here, the hour column is converted to a numeric type using as.numeric(). This is necessary for numerical operations that may follow, particularly when combining date and hour into a datetime object.

4- Creates a new column datetime by adding hours (extracted and converted in the previous step) to the dates in the date column. The hours() function from lubridate helps in adjusting the date object by the specified number of hours to form a complete datetime.

5- Reorders the weather data table chronologically based on the datetime column.

6- Using the pivot_wider() function from the tidyr package, this code transforms the data from long to wide format. It creates new columns based on unique combinations of latitude (lat) and longitude (lon), and fills these columns with values from various weather-related variables listed in values_from.

7- Merges df_wide with the production data table based on the datetime column. This operation aligns the data by datetime, allowing for combined analysis or further processing where each row represents both weather and production data at a specific time.

```
setDT(weather)

weather[, date := ymd(date)]

weather[, hour := as.numeric(hour)]

weather[, datetime := date + hours(hour)]

weather <- weather[order(datetime)]

df_wide <- weather %>%

  pivot_wider(names_from = c(lat, lon),

              values_from = c("dswrf_surface",

                              "tcdc_low.cloud.layer",

                              "tcdc_middle.cloud.layer",

                              "tcdc_high.cloud.layer",

                              "tcdc_entire.atmosphere",

                              "uswrf_top_of_atmosphere",

                              "csnow_surface",

                              "dlwrf_surface",

                              "uswrf_surface",
```

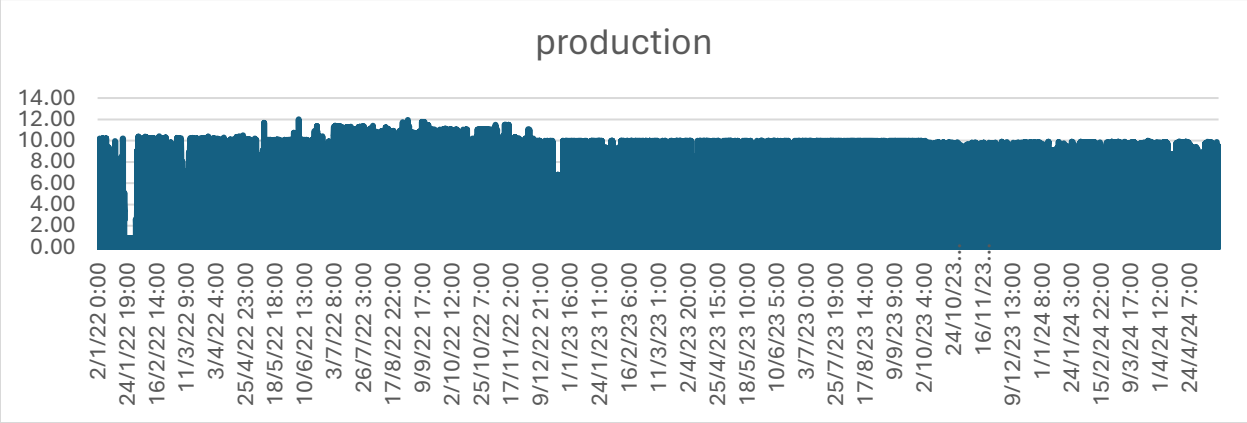
```
                    "tmp_surface"))
merged_data <- merge(df_wide, production, by = "datetime")


setDT(production)
production[, date := ymd(date)]
production[, hour := as.numeric(hour)]
production[, datetime := date + hours(hour)]
production <- production[order(datetime)]
```

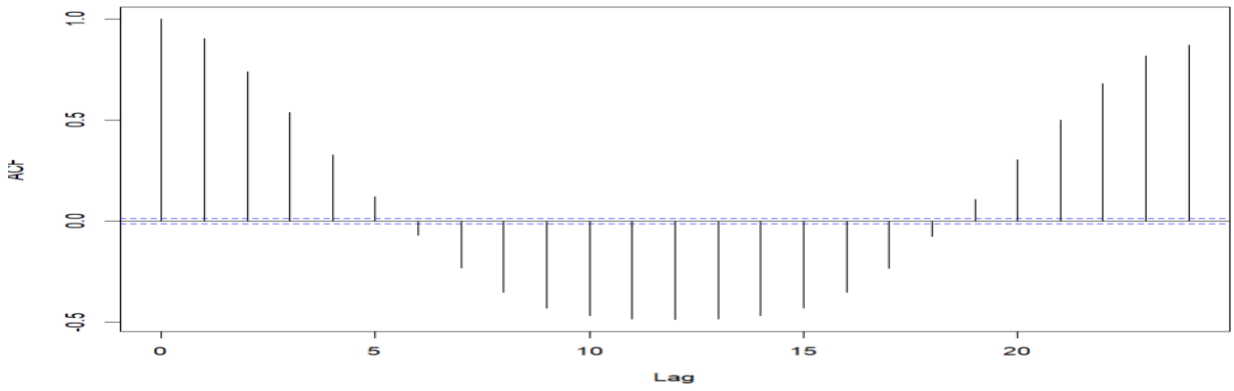
VISUALZIATION

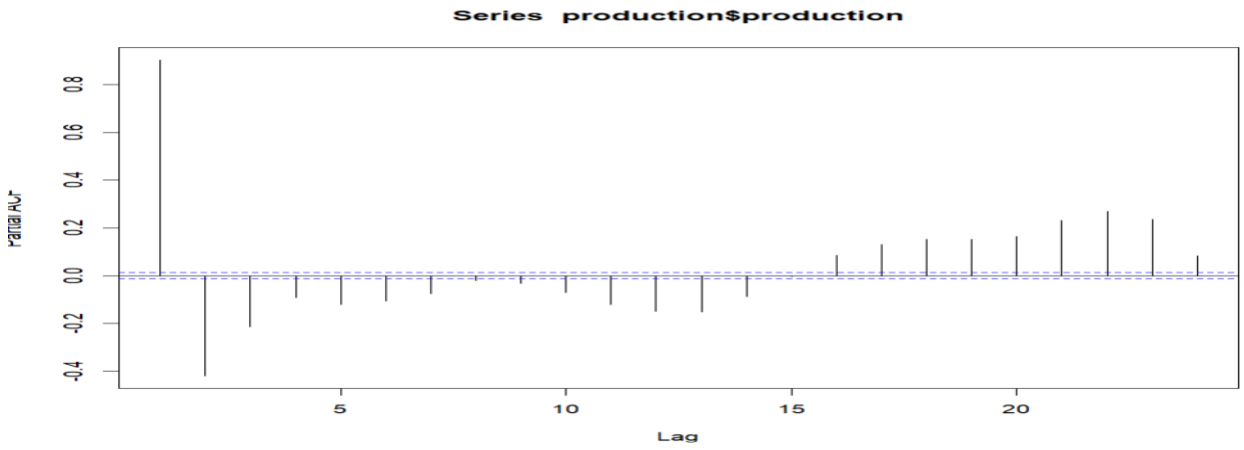
Visualizes and analyzes a production dataset using various plotting and transformation techniques. It starts by plotting a time series of production over datetime using ggplot, with a line graph and a smoothed line using LOESS regression. Next, it aggregates production data by date, calculating the total production per day, and plots these daily totals with both linear and LOESS smoothed trends. ggpairs is used to create a scatterplot matrix, providing pairwise scatterplots to inspect relationships between all variables in the production dataset. Further transformations include adding a trend component trnd, extracting and formatting day and month from the date for easier grouping and analysis. Another ggplot visualization plots production against total with a linear model fit, and facets by month, providing a breakdown of relationships on a monthly basis. Finally, it performs an autocorrelation function (ACF) analysis to understand the time-series properties of production, examining correlations up to 24 lags, which can help in identifying patterns or cycles in the data. This code collectively offers a comprehensive approach to visualizing and analyzing time-series data, exploring its seasonalities and underlying trends.

```
ggplot(production, aes(x = datetime, y = production)) +
  geom_line() +
  geom_smooth(method = "loess")
daily_series <- production[, .(total = sum(production)), by = .(date)]
ggplot(daily_series, aes(x = date, y = total)) +
  geom_line() +
  geom_smooth(method = "lm") +
  geom_smooth(method = "loess")
ggpairs(production)
production[, trnd := 1:.N]
```

```
production[, w_day := as.character(wday(date, label = TRUE))]
production[, mon := as.character(month(date, label = TRUE))]
ggplot(production, aes(x = production, y = total)) +
  geom_point() +
  geom_smooth(method = "lm", linewidth = 3) +
  facet_wrap(~ mon)
acf(production$production, lag = 24)
```

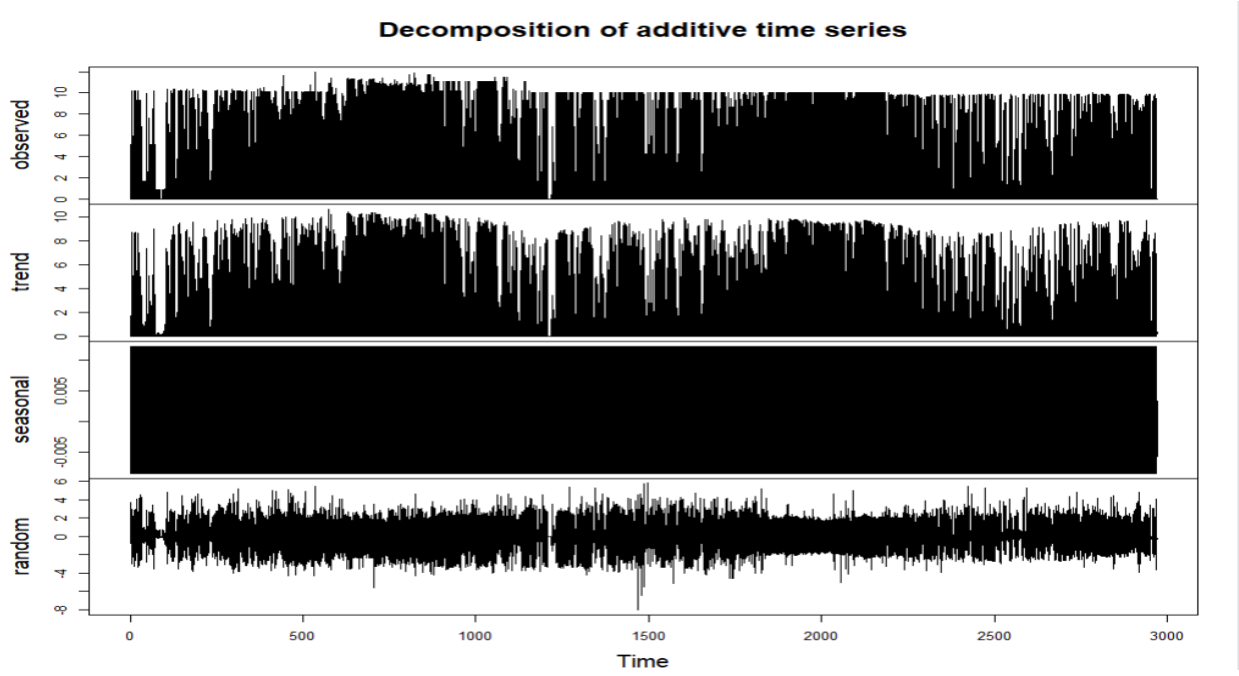






It first calculates the daily mean and maximum of production values grouped by datetime, storing the results in daily_production. Then, it converts the daily mean production into a time series object with a weekly frequency using the ts() function. The decompose() function is applied to this time series to perform a classical decomposition, separating the data into seasonal, trend, and irregular components. Finally, the decomposition results are visualized using plot(), allowing for an examination of how these components vary over time, providing insights into the underlying patterns in the production data.

```
daily_production <- production[, .(
  mean_production = mean(production),
  max_temp = max(production)
), by = .(datetime)]
mean_production <- ts(daily_production$mean_production, frequency = 7)
ts_decomposed <- decompose(mean_production)
plot(ts_decomposed)
```



LINEAR REGRESSION

Initiates by transforming the production data into a time series object with a frequency reflecting daily observations (assuming 24 periods per day). It applies a Box-Cox transformation to stabilize variance and make the data more normally distributed, guided by an optimal lambda value. The transformed data is then differenced to make it stationary, essential for many predictive modeling techniques. Subsequently, a decision tree model (using ANOVA method) is trained on the production data combined with weather data, and variables significant above a median threshold are retained for further analysis. A split is created to separate training and testing datasets, ensuring an 80-20 split. Linear regression is applied on the training data to predict production based on selected weather variables. Predictions are re-scaled back to their original scale using an inverse Box-Cox transformation. The model's performance is assessed through Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). Finally, the results are visualized in a time series plot, comparing actual vs. predicted values, clearly displaying the model's accuracy and predictive capabilities. This workflow encapsulates data preparation, feature selection, model training, prediction, and evaluation, providing a comprehensive approach to integrating weather data for production forecasting.

```
ts_production <- ts(production, frequency = 24)

lambda <- BoxCox.lambda(ts_production)

ts_production_transformed <- BoxCox(ts_production, lambda)

library(urca)

ts_production_transformed %>% ur.kpss() %>% summary()
```

```
######################
# KPSS Unit Root Test #
######################

Test is of type: mu with 15 lags.

Value of test-statistic is: 1.5554

Critical value for a significance level of:
                10pct  5pct 2.5pct  1pct
critical values 0.347 0.463  0.574 0.739
```

```
production_diff <- diff(ts_production_transformed, lag = 24)
```

```
tree_model <- rpart(production ~ ., data = cbind(production, weather_data), method = 'anova')

importance <- tree_model$variable.importance

threshold <- quantile(importance, 0.5)

selected_variables <- names(importance)[importance > threshold]

selected_variables
```

```
"dswrf_surface_38_35.25"    "dswrf_surface_37.75_35.25" "dswrf_surface_38_35"       "dswrf_surface_38.25_35.2
5"
[6] "dswrf_surface_38.75_35.5"  "uswrf_surface_38_35.25"    "uswrf_surface_38.25_35.5"
```

```
selected_weather_data <- weather_data[, selected_variables]

tomorrow_weather='C:/Users/Vusal Ibrahimli/OneDrive/Desktop/tomorrow_weather.xlsx'

tomorrow_weather <- read.xlsx(tomorrow_weather)

set.seed(123)

trainIndex <- createDataPartition(production_diff, p = .8, list = FALSE, times = 1)

production_train <- production_diff[trainIndex]

production_test <- production_diff [-trainIndex]

weather_train <- selected_weather_data[trainIndex, ]

weather_test <- selected_weather_data[-trainIndex, ]
```

```
lm_model <- lm(production_train ~ ., data = data.frame(production_train, weather_train))

summary(lm_model)

CV(lm_model)
```

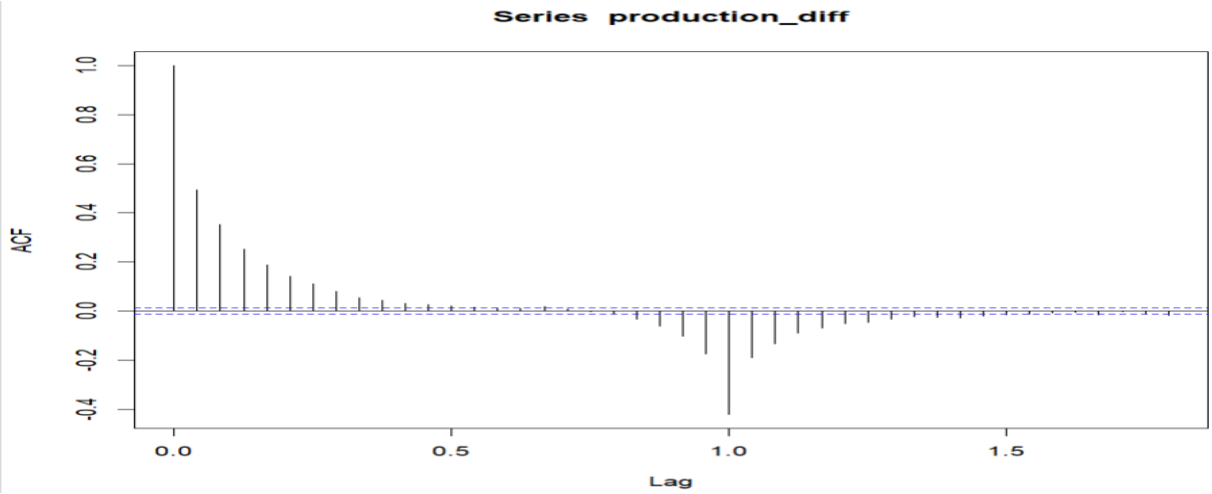
```
lm_predictions <- predict(lm_model, newdata = data.frame(tomorrow_weather))
lm_predictions_original_scale <- InvBoxCox(lm_predictions, lambda)
test_actual <- production_test
test_predicted <- lm_predictions_original_scale
mse <- mean((test_actual - test_predicted)^2)
mae <- mean(abs(test_actual - test_predicted))
rmse <- sqrt(mse)
cat("MSE: ", mse, "\n")
cat("MAE: ", mae, "\n")
cat("RMSE: ", rmse, "\n")

df_results <- data.frame(Time = seq_along(test_actual), Actual = test_actual, Predicted = test_predicted)
checkresiduals(df_results)

ggplot(df_results, aes(x = Time)) +
  geom_line(aes(y = Actual, color = "Actual")) +
  geom_line(aes(y = Predicted, color = "Predicted")) +
  ggtitle("Linear Regression Predictions") +
  xlab("Time") +
  ylab("Production") +
  scale_color_manual("", breaks = c("Actual", "Predicted"), values = c("Actual" = "blue", "Predicted" = "red"))
```

SARIMA

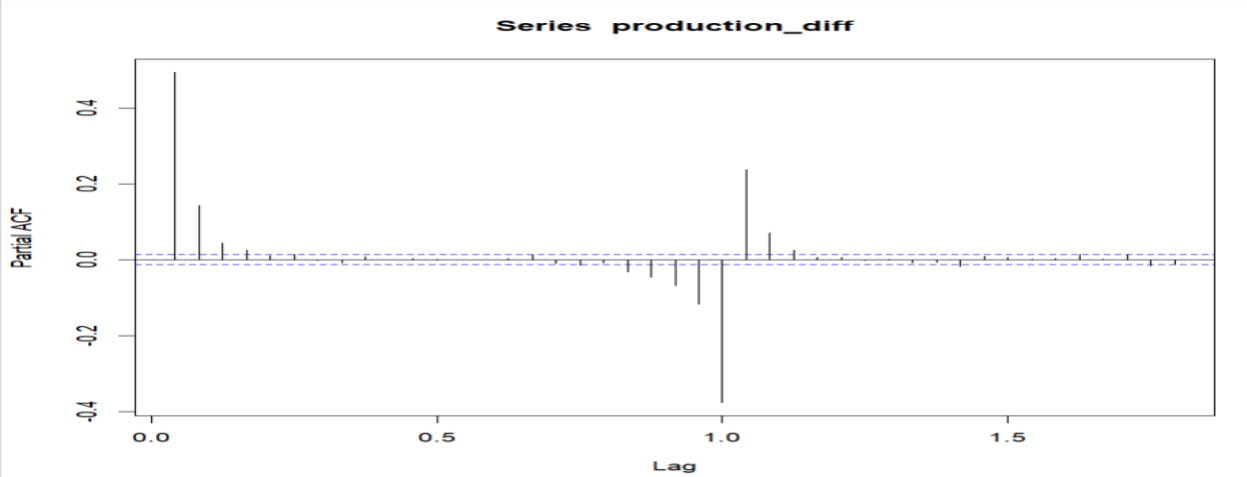
Involves a series of steps aimed at analyzing and forecasting production data using time series and regression tree methods, combined with external variables like weather data. It begins by converting the production vector into a time series object (ts_production) with a frequency of 24 to represent hourly data within a day. A decision tree model is then trained using the production data and weather data, using the ANOVA method to determine the importance of various predictors. Only variables that exceed the median

importance are retained for further analysis. The selected weather variables are prepared as a matrix (xreg_arima_aligned) to be used as external regressors in an ARIMA model. Meanwhile, the production time series is transformed using the BoxCox method to stabilize variance, then differenced to ensure stationarity. Autocorrelation and partial autocorrelation functions are used to analyze the differenced data, providing insights into the lag structure. Finally, an ARIMA model is fitted using both the transformed production data and the selected weather data as regressors, with detailed model diagnostics and residual checks performed to validate the model's adequacy. This comprehensive approach leverages both tree-based variable selection and sophisticated time series modeling to enhance predictive accuracy.

```
ts_production <- ts(production, frequency = 24)

lambda <- BoxCox.lambda(ts_production)

ts_production_transformed <- BoxCox(ts_production, lambda)

library(urca)

ts_production_transformed %>% ur.kpss() %>% summary()

production_diff <- diff(ts_production_transformed, lag = 24)

acf(production_diff)
```




```
pacf(production_diff)
```




```
production_df <- as.data.frame(production)

tree_model <- rpart(production ~ ., data = cbind(production_df, weather_data), method = 'anova')

importance <- tree_model$variable.importance

threshold <- quantile(importance, 0.5)

selected_variables <- names(importance)[importance > threshold]

selected_weather_data <- weather_data[, selected_variables]

xreg_arima_aligned <- as.matrix(selected_weather_data)

acf(ts_production)

pacf(ts_production)

arima_model <- auto.arima(ts_production, xreg = xreg_arima_aligned, approximation = FALSE, stepwise = FALSE, trace = T)

summary(arima_model)

checkresiduals(arima_model)
```

Designed to forecast energy production for the next 24 hours using a pre-trained SARIMA model, incorporating weather predictions as external variables. The process starts by loading a weather forecast for the next day from an Excel file located at the specified path on the user's computer. It then selects relevant weather variables that were previously identified as significant predictors in the ARIMA model. These selected variables are transformed into a matrix format required for the forecasting function.

The forecast() function is then used to predict the next 24 hours of energy production, using the ARIMA model and the weather data as external regressors (xreg). The function is set to produce forecasts (h = 24) for each hour of the next day. The results of these forecasts are printed to give an immediate view of the predicted values.

Finally, the forecasted values are visualized using autoplot() from the forecast package, enhanced with a title and labeled axes to clearly communicate the time frame and the expected energy production levels. This visualization provides a graphical representation of the predicted energy production over the next 24 hours, facilitating easy interpretation of the forecast's trends and fluctuations.

```
tomorrow_weather_path <- " C:/Users/Vusal Ibrahimli/OneDrive/Desktop/tomorrow_weather.xlsx"

tomorrow_weather <- read.xlsx(tomorrow_weather_path)

future_xreg <- tomorrow_weather[, selected_variables]

future_xreg <- as.matrix(future_xreg)


forecasted_values <- forecast(arima_model, h = 24, xreg = future_xreg)

print(forecasted_values)

autoplot(forecasted_values) + ggtitle("24-Hour Energy Production Forecast") + xlab("Hour") +
ylab("Production")
```

Model Evaluation

```
accu <- function(actual, forecast) {
  n <- length(actual)
  error <- actual - forecast
  mean_val <- mean(actual)
  sd_val <- sd(actual)
  CV <- sd_val / mean_val
  FBias <- sum(error) / sum(actual)
  MAPE <- sum(abs(error / actual)) / n
  RMSE <- sqrt(sum(error^2) / n)
  MAD <- sum(abs(error)) / n
  MADP <- sum(abs(error)) / sum(abs(actual))
  WMAPE <- MAD / mean_val
  metrics_df <- data.frame(n, mean = mean_val, sd = sd_val, CV, FBias, MAPE, RMSE, MAD, MADP,
WMAPE)
  return(metrics_df)
}
forecast_with_lr <- function(fmla, data, forecast_data) {
  fitted_lm <- lm(as.formula(fmla), data)
  forecasted <- predict(fitted_lm, newdata = forecast_data)
  return(list(forecast = as.numeric(forecasted), model = fitted_lm))
}

forecast_with_arima <- function(data, forecast_ahead, target_name = 'production', pretrained_model =
NULL,
                                is_seasonal = FALSE, is_stepwise = FALSE, is_trace = TRUE, is_approx = FALSE)
{
  input_series <- data[[target_name]]

  if (is.null(pretrained_model)) {
```

```
    fitted <- auto.arima(input_series, seasonal = is_seasonal,
                         trace = is_trace, stepwise = is_stepwise, approximation = is_approx)
  } else {
    fitted <- Arima(input_series, model = pretrained_model)
  }

  forecasted <- forecast(fitted, h = forecast_ahead)
  return(list(forecast = as.numeric(forecasted$mean), model = fitted))
}

test_start <- as.Date('2024-05-13')
test_end <- as.Date('2024-05-26')
test_dates <- seq(test_start, test_end, by = 'day')
test_dates

forecast_ahead <- 1
results <- vector('list', length(test_dates))
i <- 1
current_date <- test_dates[i] - forecast_ahead
    past_data <- production[datetime <= current_date, .(datetime, production, trnd, w_day, mon)]
    forecast_data <- production[datetime == test_dates[i], .(datetime, production, trnd, w_day, mon)]
    fmla <- 'production ~ trnd + w_day + mon'
    lr_results <- forecast_with_lr(fmla, past_data, forecast_data)
    forecast_data[, lm_prediction := lr_results$forecast]
    arima_forecast <- forecast(arima_model, h = forecast_ahead, xreg = future_xreg[1, , drop = FALSE])
    forecast_data[, arima_prediction := arima_forecast$mean]
    results[[i]] <- forecast_data
 }
forecast_data
```

Results

We are using SARIMA model in forecasting period.

Method	Day	WMAPE
SARIMA	26-May	0.1669
SARIMA	25-May	0.9667
SARIMA	24-May	0.5135
SARIMA	23-May	0.063
SARIMA	22-May	0.0816
SARIMA	21-May	0.2254
SARIMA	20-May	0.2533
SARIMA	19-May	0.169
SARIMA	18-May	0.2009
SARIMA	17-May	0.2904
SARIMA	16-May	0.3564
SARIMA	15-May	0.0516
SARIMA	14-May	0.3065
SARIMA	13-May	0.4657