1-Data Preprocessing:

```r
# Load the necessary libraries
library(readxl)
library(forecast)
library(forecast)
library(ggplot2)
library(imputeTS)
library(trend)
library(writexl)
library(dplyr)
```

```r
#Load data
# Read the Excel file '2023-11-Elec-train.xlsx'
data <- read_excel('C:/Users/eyami/2023-11-Elec-train.xlsx')
print(data)
```

Timestamp <chr>	Power (kW) <dbl>	Temp (C°) <dbl>
40179.052083333336	165.1	10.555556
1/1/2010 1:30	151.6	10.555556
1/1/2010 1:45	146.9	10.555556
1/1/2010 2:00	153.7	10.555556
1/1/2010 2:15	153.8	10.555556
1/1/2010 2:30	159.0	10.555556
1/1/2010 2:45	157.7	10.555556
1/1/2010 3:00	163.2	10.555556
1/1/2010 3:15	151.7	10.000000
1/1/2010 3:30	148.7	10.000000

1-10 of 4,987 rows

Previous 1 2 3 4 5 6 ... 100 Next

The data set provided contains three columns: Time stamp, Power (kW) and Temperature (C°) with a total of 4,987 lines. Each row represents an observation of power consumption and temperature at a specific point in time. The dataset is intended to cover 52 days, assuming a time step (interval) of 15 minutes between observations. This interval should ideally provide 96 observations per day (since there are 96 15-minute intervals in a 24-hour day). But looking at the dataset we have only 4,987, which suggests that the dataset is missing 5 observations Given the importance of continuous and complete data in time series forecasting, it will be essential to identify the missing timestamps handle them appropriately before proceeding with modeling.

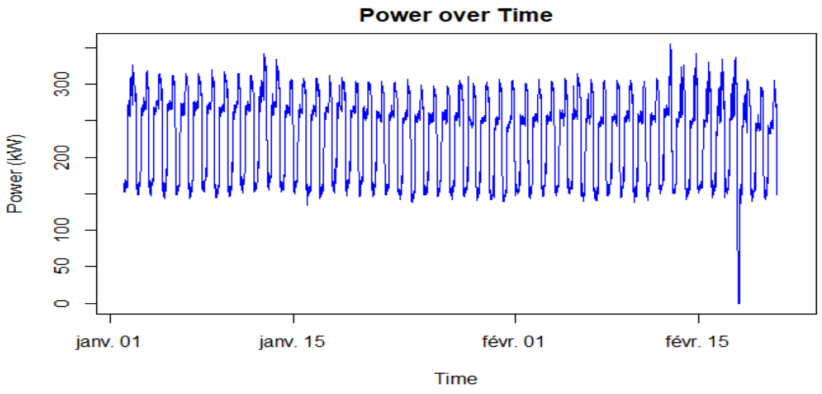
1.1 Removing Incomplete Data.

After verification, it was found that January 1, 2010, has only 91 observations. Therefore, in the next step, I removed the first 91 rows to balance the number of observations in the dataset.

```r
# Remove the first 92 rows from the data
data <- data[-(1:91), ]
```

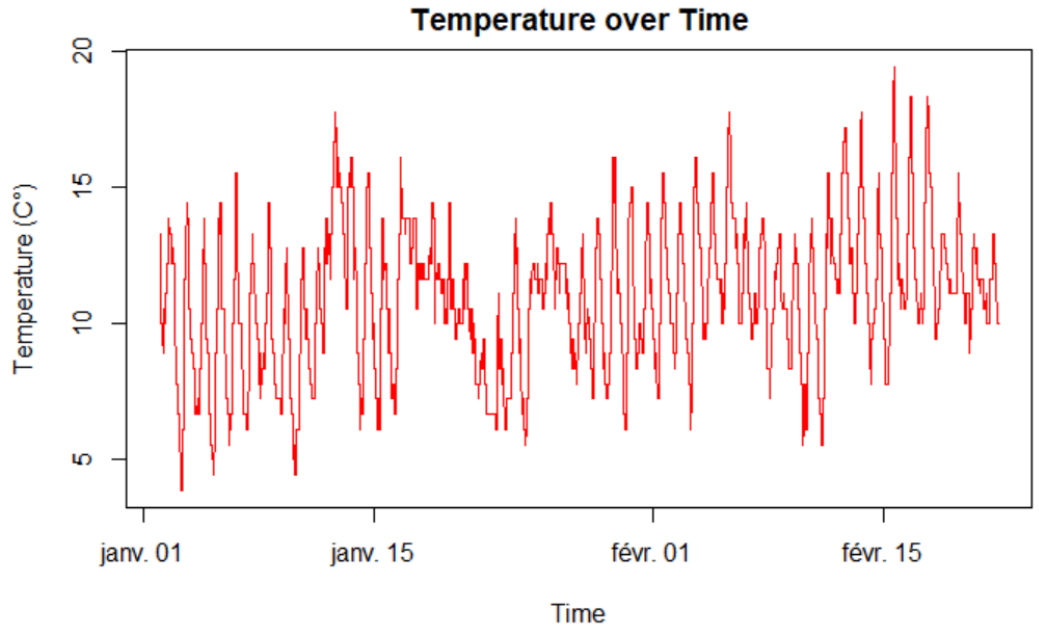
1.2 Plot Data

```r
# Convert the 'Timestamp' column to POSIXct format for time series analysis
data$Timestamp <- as.POSIXct(data$Timestamp, format="%m/%d/%Y %H:%M",tz="UTC")

# Plot power consumption over time
plot(data$Timestamp, data$`Power (kW)`, type="l", col="blue",
     xlab="Time", ylab="Power (kW)", main="Power over Time")

# Plot temperature over time
plot(data$Timestamp, data$`Temp (C°)`, type="l", col="red",
     xlab="Time", ylab="Temperature (C°)", main="Temperature over Time")
```



The time series plot of power consumption over the observed period reveals a consistent daily pattern with regular fluctuations. However, there is a notable and significant anomaly where the power consumption drops sharply to near zero**.**

This issue will be systematically addressed in the following stages of the analysi**s**



1.3 Split Data

The dataset is split into three parts: training set, testing set, and unseen data for forecasting.
Training Set: Consists of 80% of the data before the cutoff date (2010-02-21 00:00:00). This set is used to train the time series model, allowing it to learn patterns and seasonal trends.
Testing Set: Comprises the remaining 20% of the data before the cutoff date. This set is used to validate the model's performance, ensuring it can generalize to unseen data that it hasn't been trained on.
Unseen Data: Contains data from and after the cutoff date. This dataset represents future time points for which the actual values of the target variable (Power (kW)) are unknown. The model will generate forecasts for this unseen period

1.4 Data Imputation

```r
# Display rows in the training set where Power (kW) is 0
null_rows_train <- train_data[train_data$`Power (kW)` == 0, ]
print(null_rows_train)
# Display rows in the test set where Power (kW) is 0
null_rows_test <- test_data[test_data$`Power (kW)` == 0, ]
print(null_rows_test)
```

tbl_df 0 x 4

tbl_df 11 x 4

Timestamp <S3: POSIXct>	Power (kW) <dbl>	Temp (C°) <dbl>	Date <date>
2010-02-18 00:00:00	0	12.77778	2010-02-18
2010-02-18 00:15:00	0	12.22222	2010-02-18
2010-02-18 00:30:00	0	12.22222	2010-02-18
2010-02-18 00:45:00	0	12.22222	2010-02-18
2010-02-18 01:00:00	0	12.22222	2010-02-18
2010-02-18 01:15:00	0	11.66667	2010-02-18
2010-02-18 01:30:00	0	11.66667	2010-02-18
2010-02-18 01:45:00	0	11.66667	2010-02-18
2010-02-18 02:00:00	0	11.66667	2010-02-18
2010-02-18 02:15:00	0	11.11111	2010-02-18

1-10 of 11 rows Previous 1 2 Next

I identified and replaced zero values in the power consumption data with NaN in the test_data, then performed imputation to fill in these gaps. This process was important to ensure the accuracy of the time series analysis by preventing misleading patterns caused by erroneous zeros

```r
# Replace null values (0) with NaN in the Power (kW) column
test_data$`Power (kW)`[test_data$`Power (kW)` == 0] <- NaN

# Replace NaN values using moving average imputation
test_data$`Power (kW)` <- na_ma(test_data$`Power (kW)`)
```

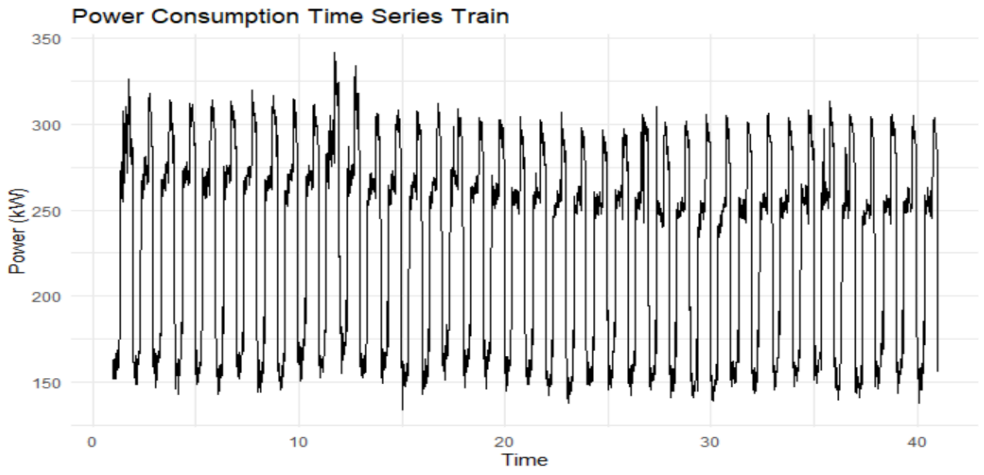
To address the anomaly of values dropping to 0 in the Power (kW) data, I implemented the following methodology:

I systematically checked both the training and test datasets for any occurrences of zero values in the Power (kW) column. This step was crucial to ensure that the training data was free from such anomalies, and to quantify the extent of the issue in the test data.

In the test dataset, where zero values were detected, I replaced these zeros with NaN to flag them as missing data. This approach prevented these erroneous values from skewing the model's training process.
I then applied a Moving Average Imputation technique to these NaN values. This method effectively smoothed the data by filling in the missing values with the average of neighboring data points, thereby maintaining the integrity of the data's underlying trends and seasonal patterns.

1.4 Plot of Power Consumption: Training and Test Sets Over Time

```r
# Set the frequency of the time series (96 periods per day, representing 15-minute intervals)
frequency <- 96

# Define the start of the time series as the 2nd day of the year, which corresponds to January 2, 2010
start <- c(1, 1)

# Create the time series for power consumption
power_ts_train <- ts(train_data$`Power (kW)`, frequency = frequency, start = start)

# Create the time series for temperature
temp_ts_train <- ts(train_data$`Temp (C°)`, frequency = frequency, start = start)

# Visualize the power consumption time series with a clean and minimalistic plot
autoplot(power_ts_train) +
  ggtitle("Power Consumption Time Series Train ") +
  xlab("Time") +
  ylab("Power (kW)") +
  theme_minimal()

# Visualize the temperature time series with a clean and minimalistic plot
autoplot(temp_ts_train) +
  ggtitle("Temperature Time Series Train") +
  xlab("Time") +
  ylab("Temperature (C°)") +
  theme_minimal()
```




```r
# Set the frequency of the time series (96 periods per day, representing 15-minute intervals)
frequency <- 96

# Define the start of the time series based on the first date in the test dataset
start_test <- c(40, 1)  # February 18, 2010

# Create the time series for power consumption (test data)
power_ts_test <- ts(test_data$`Power (kW)`, frequency = frequency, start = start_test)

# Create the time series for temperature (test data)
temp_ts_test <- ts(test_data$`Temp (C°)`, frequency = frequency, start = start_test)

# Visualize the power consumption time series for test data
autoplot(power_ts_test) +
  ggtitle("Power Consumption Time Series (Test Data)") +
  xlab("Time") +
  ylab("Power (kW)") +
  theme_minimal()

# Visualize the temperature time series for test data
autoplot(temp_ts_test) +
  ggtitle("Temperature Time Series (Test Data)") +
  xlab("Time") +
  ylab("Temperature (C°)") +
  theme_minimal()
```

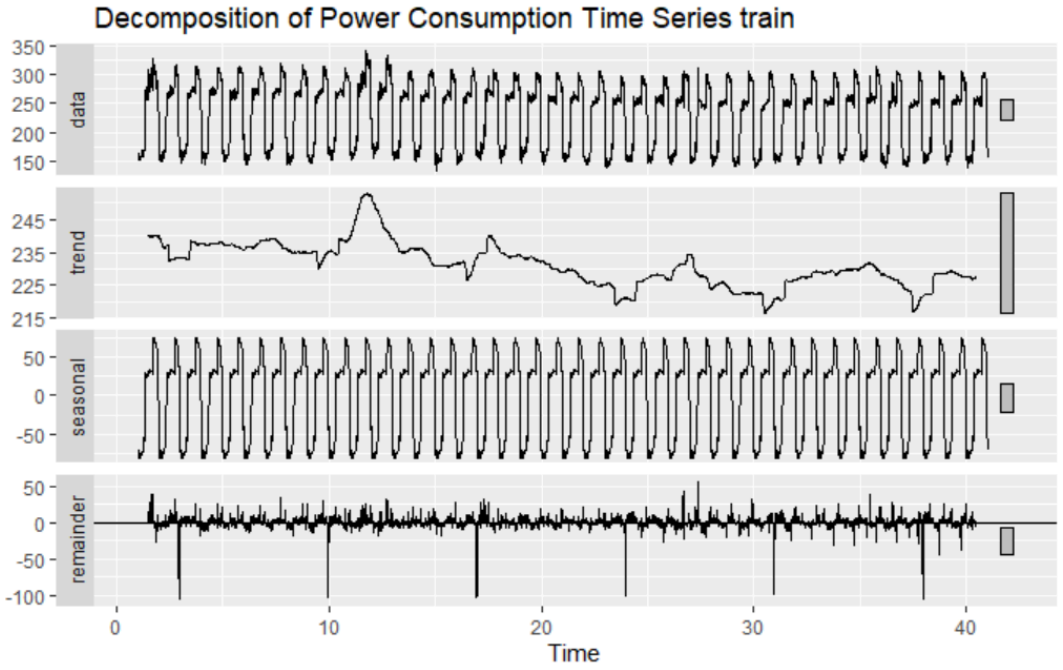
1.5 Analysis of Box-Pierce test

```{r}
Box.test(power_ts_train, lag = 10, type = "Box-Pierce")
Box.test(power_ts_test, lag = 10, type = "Box-Pierce")
```

```
        Box-Pierce test

data:  power_ts_train
X-squared = 24267, df = 10, p-value < 2.2e-16


        Box-Pierce test

data:  power_ts_test
X-squared = 6015, df = 10, p-value < 2.2e-16
```

The time series exhibits significant autocorrelations over the first 10 lags. This indicates that the series has a non-random temporal structure and that there are likely significant dependencies between observations over time

1.6 Decomposition of Power Consumption Time Series



The decomposition of the power consumption time series reveals key insights into the underlying structure of the data. The trend component shows a relatively stable pattern with minor fluctuations, indicating that overall power usage remains within a consistent range over time. The seasonal

component highlights a strong and regular cyclical pattern corresponding to daily usage behaviors. Lastly, the residuals, while mostly stable, suggest the presence of anomalies or events that are not explained by the trend and seasonality, which may require further investigation.

2-Forecast without temperature

2.1 Model 1 : Additive seasonal Holt-Winters

```r
# Initialize the number of folds and the list to store RMSE values
folds <- 4
segments <- list()
rmse_values <- numeric(folds)

# Total number of observations per fold
observations_per_fold <- 960

# Segment the time series into 4 equal parts and convert them into `ts` objects
for (i in 1:folds) {
    start_idx <- ((i - 1) * observations_per_fold) + 1
    end_idx <- i * observations_per_fold
    segments[[i]] <- ts(power_ts_train[start_idx:end_idx], frequency = frequency, start = c(1, 1))
}

# Loop through each segment for cross-validation
for (i in 1:folds) {
    # Use the other segments for training, exclude the current segment i
    train_segments <- segments[-i]
    val_segment <- segments[[i]]

    # Combine the training segments into one time series
    train_data_c <- do.call(c, train_segments)

    # Convert the combined data into a time series object
    train_data_ts <- ts(train_data_c, frequency = frequency)

    # Fit the Holt-Winters model on the training data
    model <- HoltWinters(train_data_ts)

    # Predict on the validation set (h = 960 because each segment has 960 observations)
    predictions <- forecast(model, h = observations_per_fold)$mean

    # Convert the validation set and predictions to numeric vectors if necessary
    val_set <- as.numeric(val_segment)
    predictions <- as.numeric(predictions)

    # Calculate the RMSE between the predictions and the actual values of the validation set
    rmse_values[i] <- sqrt(mean((val_set - predictions)^2))
}

# Calculate the average RMSE across all folds
mean_rmse <- mean(rmse_values)
print(mean_rmse)
```

```
[1] 38.87823
```

In this step, I performed cross-validation using the Holt-Winters model on the time series data to evaluate its performance. The cross-validation was done by dividing the dataset into four segments (folds) and iteratively training the model on three segments while validating it on the remaining one. The RMSE (Root Mean Square Error) was calculated for each fold by comparing the model's predictions against the actual observed values within that fold.

Cross-validation is essential for assessing the generalization ability of the model, as it provides a more reliable measure of its performance on unseen data. By averaging the RMSE across all folds, I obtained a robust estimate of the model's accuracy.

This cross-validation was conducted without including the temperature as an explanatory variable, providing a baseline RMSE. In the next step, I will include the temperature variable to determine whether its inclusion improves the model's predictive accuracy. Comparing the RMSE from this baseline model with the RMSE from the model that includes temperature will help to evaluate the added value of incorporating this additional feature.

```r
# Fit the Holt-Winters model on the entire training data
final_model <- HoltWinters(power_ts_train)

# Predict on the test dataset
test_predictions <- forecast(final_model, h = length(power_ts_test))$mean

# Convert the test set and predictions to numeric vectors if necessary
test_set <- as.numeric(power_ts_test)
test_predictions <- as.numeric(test_predictions)

# Calculate the RMSE between the predictions and the actual values of the test set
test_rmse <- sqrt(mean((test_set - test_predictions)^2))

# Print the RMSE for the test data
print(test_rmse)
```

```
[1] 27.30942
```

The model has been tested on the test dataset, and the RMSE (Root Mean Square Error) value obtained is approximately 27.31

2.2 Model 2 :Multiplicative seasonal Holt-Winters

```r
# Segment the time series into 4 equal parts and convert them into `ts` objects
for (i in 1:folds) {
    start_idx <- ((i - 1) * observations_per_fold) + 1
    end_idx <- i * observations_per_fold
    segments[[i]] <- ts(power_ts_train[start_idx:end_idx], frequency = frequency, start = c(1, 1))
}

# Loop through each segment for cross-validation
for (i in 1:folds) {
    # Use the other segments for training, exclude the current segment i
    train_segments <- segments[-i]
    val_segment <- segments[[i]]

    # Combine the training segments into one time series
    train_data_c <- do.call(c, train_segments)

    # Convert the combined data into a time series object
    train_data_ts <- ts(train_data_c, frequency = frequency)

    # Fit the Holt-Winters model on the training data
    model <- HoltWinters(train_data_ts,alpha=NULL,beta=NULL,gamma=NULL,seasonal = "multi")

    # Predict on the validation set (h = 960 because each segment has 960 observations)
    predictions <- forecast(model, h = observations_per_fold)$mean

    # Convert the validation set and predictions to numeric vectors if necessary
    val_set <- as.numeric(val_segment)
    predictions <- as.numeric(predictions)

    # Calculate the RMSE between the predictions and the actual values of the validation set
    rmse_values[i] <- sqrt(mean((val_set - predictions)^2))
}

# Calculate the average RMSE across all folds
mean_rmse <- mean(rmse_values)
print(mean_rmse)
```

```
[1] 37.74431
```

```r
# Fit the Holt-Winters model on the entire training data
final_model <- HoltWinters(power_ts_train,alpha=NULL,beta=NULL,gamma=NULL,seasonal = "multi")

# Predict on the test dataset
test_predictions <- forecast(final_model, h = length(power_ts_test))$mean

# Convert the test set and predictions to numeric vectors if necessary
test_set <- as.numeric(power_ts_test)
test_predictions <- as.numeric(test_predictions)

# Calculate the RMSE between the predictions and the actual values of the test set
test_rmse <- sqrt(mean((test_set - test_predictions)^2))

# Print the RMSE for the test data
print(test_rmse)
```

```
[1] 26.52527
```

The model has been tested on the test dataset, and the RMSE (Root Mean Square Error) value obtained is approximately 26.52527

2.3 Model 3 : Auto ARIMA

```r
# Segment the time series into 4 equal parts and convert them into `ts` objects
for (i in 1:folds) {
    start_idx <- ((i - 1) * observations_per_fold) + 1
    end_idx <- i * observations_per_fold
    segments[[i]] <- ts(power_ts_train[start_idx:end_idx], frequency = frequency, start = c(1, 1))
}

# Loop through each segment for cross-validation
for (i in 1:folds) {
    # Use the other segments for training, exclude the current segment i
    train_segments <- segments[-i]
    val_segment <- segments[[i]]

    # Combine the training segments into one time series
    train_data_c <- do.call(c, train_segments)

    # Convert the combined data into a time series object
    train_data_ts <- ts(train_data_c, frequency = frequency)

    # Fit the ARIMA model on the training data
    model <- auto.arima(train_data_ts)

    # Predict on the validation set (h = 960 because each segment has 960 observations)
    predictions <- forecast(model, h = observations_per_fold)$mean

    # Convert the validation set and predictions to numeric vectors if necessary
    val_set <- as.numeric(val_segment)
    predictions <- as.numeric(predictions)

    # Calculate the RMSE between the predictions and the actual values of the validation set
    rmse_values[i] <- sqrt(mean((val_set - predictions)^2))
}

# Calculate the average RMSE across all folds
mean_rmse <- mean(rmse_values)
print(mean_rmse)
```

```
[1] 18.97768
```

```r
# Fit the ARIMA model on the entire training data
final_model_arima <- auto.arima(power_ts_train)

# Predict on the test dataset
test_predictions_arima <- forecast(final_model_arima, h = length(power_ts_test))$mean

# Convert the test set and predictions to numeric vectors if necessary
test_set <- as.numeric(power_ts_test)
test_predictions_arima <- as.numeric(test_predictions_arima)

# Calculate the RMSE between the predictions and the actual values of the test set
test_rmse_arima <- sqrt(mean((test_set - test_predictions_arima)^2))

# Print the RMSE for the test data
print(test_rmse_arima)
```

```
[1] 15.86546
```

The model has been tested on the test dataset, and the RMSE (Root Mean Square Error) value obtained is approximately 15.86546

2.4 Conclusion 1 :

various time series forecasting models were evaluated for predicting power consumption,
Each model's performance was assessed by calculating the RMSE on the test dataset, providing a comparative understanding of their predictive accuracy.
Among the models tested, the one that achieved the lowest RMSE was Auto ARIMA, indicating its superior ability to capture the underlying patterns in the data.

3 Forecast with temperature

```
# Fit the ARIMA model on the entire training data
final_model_arima <- auto.arima(power_ts_train,xreg = temp_ts_train)

# Predict on the test dataset
test_predictions_arima <- forecast(final_model_arima,xreg = temp_ts_test, h = length(power_ts_test))$mean

# Convert the test set and predictions to numeric vectors if necessary
test_set <- as.numeric(power_ts_test)
test_predictions_arima <- as.numeric(test_predictions_arima)

# Calculate the RMSE between the predictions and the actual values of the test set
test_rmse_arima <- sqrt(mean((test_set - test_predictions_arima)^2))

# Print the RMSE for the test data
print(test_rmse_arima)
```

```
[1] 15.7194
```

The Auto ARIMA model, when incorporating the temperature variable as an explanatory factor, produced the most accurate predictions with a resulting RMSE of 15.7194.

4 Forecasting Unseen Power Consumption Values Using the Selected ARIMA Model

In this section, we use the best-performing model, the Auto ARIMA model with temperature as an explanatory variable, to forecast the power consumption values for unseen data. The unseen data contains timestamps and temperature values, but the power consumption values are missing. By converting the temperature data into a time series format, we use the trained ARIMA model to predict the missing power consumption values. These predictions are then added back to the dataset. Additionally, we ensure that the timestamp format in the unseen data is consistent with the other datasets for seamless integration.

```
# Convert the temperature of the unseen data into a time series
unseen_temp_ts <- ts(unseen_data$`Temp (C°)`, frequency = frequency, start = c(1, 1))

# Predict the "Power (kW)" values for the unseen data using the trained ARIMA model
unseen_power_predictions <- forecast(final_model_arima, xreg = unseen_temp_ts, h = length(unseen_temp_ts))$mean

# Add the predictions to the "Power (kW)" column of the unseen data
unseen_data$`Power (kW)` <- unseen_power_predictions

# Convert the Timestamps in 'unseen_data' to character strings
# This ensures that the format is consistent with the other datasets
unseen_data$Timestamp <- format(unseen_data$Timestamp, format="%Y-%m-%d %H:%M:%S", tz="UTC")

print(unseen_data)
```

5 Saving Forecasted Power Consumption Values to Excel

```
```{r}

# Standardize the format of the Timestamps
# Convert the 'Timestamp' column in both 'train_data' and 'test_data' to POSIXct format
# and then back to a standardized string format ("%Y-%m-%d %H:%M:%S")
train_data$Timestamp <- format(as.POSIXct(train_data$Timestamp, format="%Y-%m-%d %H:%M:%S"), format="%Y-%m-%d %H:%M:%S")
test_data$Timestamp <- format(as.POSIXct(test_data$Timestamp, format="%Y-%m-%d %H:%M:%S"), format="%Y-%m-%d %H:%M:%S")

# Convert the Timestamps of 'train_data' and 'test_data' to character strings
# to ensure they match the format of 'unseen_data'
train_data$Timestamp <- as.character(train_data$Timestamp)
test_data$Timestamp <- as.character(test_data$Timestamp)

# Remove the 'Date' column
train_data <- train_data %>% select(-Date)
test_data <- test_data %>% select(-Date)

# Concatenate the three datasets vertically (train_data, test_data, and unseen_data)
final_data <- rbind(train_data, test_data, unseen_data)
```
```

```
```{r}
# Concatenate the three datasets (train_data, test_data, unseen_data) vertically into a single dataset
final_data <- rbind(train_data, test_data, unseen_data)
print(final_data)

# Write the combined dataset to an Excel file named "forecast.xlsx"
write_xlsx(final_data, path = "C:/Users/eyami/forecast.xlsx")

# Print a confirmation message to notify the user that the file has been successfully saved
cat("The Excel file has been updated and saved as 'forecast.xlsx'.\n")
```
```