```
# Load necessary libraries
library(readxl)
library(dplyr)
library(lubridate)
library(ggplot2)
library(forecast)
library(nnet)

# Load the data
file_path <- "C:/Users/ou_ba/Downloads/2023-11-Elec-train.xlsx"
# replace with your file path
data <- read_excel(file_path)

# Rename the columns
colnames(data) <- c("Timestamp", "Power", "Temp")

# Convert Timestamp to the correct format
data$Timestamp <- as.POSIXct(data$Timestamp, format="%m/%d/%Y %H:%M")

# Remove the first line of data
data <- data[-1, ]

# Convert the relevant Power rows to NA
data$Power[4603:4613] <- NA

# Prepare the surrounding data for interpolation (40 rows before and after)
before_data <- data[(4603-40):(4603-1), ]
after_data <- data[(4613+1):(4613+40), ]

# Combine before and after data
surrounding_data <- rbind(before_data, after_data)

# Fit a polynomial model (degree = 3 as an example)
model <- lm(Power ~ poly(1:nrow(surrounding_data), 3), data = surrounding_data)

# Predict the missing values
predicted_values <- predict(model, newdata = data.frame('1:nrow(surrounding_data)' = 4603:46

# Fill the missing values in the original data
data$Power[4603:4613] <- predicted_values

# Step 3: Data Visualization
# Plot Power over time
ggplot(data, aes(x = Timestamp, y = Power)) +
  geom_line() +
  labs(title = "Power Consumption Over Time", x = "Time", y = "Power (kW)")
```

```
# Plot Temp over time
ggplot(data, aes(x = Timestamp, y = Temp)) +
  geom_line(color = 'red') +
  labs(title = "Temperature Over Time", x = "Time", y = "Temperature (°C)")

# Step 4: Data Splitting
train_size <- floor(0.8 * nrow(data))
train_data <- data[1:train_size, ]
test_data <- data[(train_size + 1):nrow(data), ]

# Prepare the training and test sets (just the Power column initially)
train_power <- ts(train_data$Power, frequency = 96) # 96 because data is collected every 15
test_power <- ts(test_data$Power, start = c(floor(train_size/96)+1, 1), frequency = 96)
```

Models implementation

Without using outdoor temperature

```
# 1. Exponential Smoothing (ETS)
ets_model <- ets(train_power)
ets_forecast <- forecast(ets_model, h = length(test_power))
ets_rmse <- sqrt(mean((test_power - ets_forecast$mean)^2, na.rm = TRUE))

# 4. Additive Seasonal Holt-Winters with Trend
hw_add_trend_model <- HoltWinters(train_power, seasonal = "additive")
hw_add_trend_forecast <- forecast(hw_add_trend_model, h = length(test_power))
hw_add_trend_rmse <- sqrt(mean((test_power - hw_add_trend_forecast$mean)^2, na.rm = TRUE))

# 5. Multiplicative Seasonal Holt-Winters with Trend
hw_mul_trend_model <- HoltWinters(train_power, seasonal = "multiplicative")
hw_mul_trend_forecast <- forecast(hw_mul_trend_model, h = length(test_power))
hw_mul_trend_rmse <- sqrt(mean((test_power - hw_mul_trend_forecast$mean)^2, na.rm = TRUE))

# 6. Auto ARIMA
arima_model <- auto.arima(train_power)
arima_forecast <- forecast(arima_model, h = length(test_power))
arima_rmse <- sqrt(mean((test_power - arima_forecast$mean)^2, na.rm = TRUE))

# 7. Neural Networks (NNET)
nnet_model <- nnetar(train_power)
nnet_forecast <- forecast(nnet_model, h = length(test_power))
nnet_rmse <- sqrt(mean((test_power - nnet_forecast$mean)^2, na.rm = TRUE))

# Print RMSE results for comparison
rmse_results <- data.frame(
  Model = c("Exponential Smoothing", "Additive HW with Trend",
            "Multiplicative HW with Trend", "Auto ARIMA",
            "Neural Network" ),
  RMSE = c(ets_rmse, hw_add_trend_rmse,
           hw_mul_trend_rmse, arima_rmse, nnet_rmse )
```

```
)

print(rmse_results)
```

###Check Model Diagnostics Before finalizing the model, it's also useful to check the residuals of the model to ensure that the model is fitting the data well:

```
# Plot residuals
checkresiduals(ets_model)
```

Ljung-Box test

data: Residuals from ETS(A,Ad,N) Q* = 5759, df = 192, p-value < 2.2e-16

Model df: 0. Total lags used: 192

```
# Summary of the model
summary(ets_model)
```

ETS(A,Ad,N)

Call: ets(y = train_power)

Smoothing parameters: alpha = 0.9071 beta = 0.0867 phi = 0.8

Initial states: l = 152.1136 b = 0.3679

sigma: 14.8158

```
 AIC     AICc      BIC
```

54572.52 54572.54 54610.27

Training set error measures: ME RMSE MAE MPE MAPE MASE ACF1 Training set 0.01978679 14.80647 6.926094 -0.1479204 3.188766 0.8754026 -0.002555648

```
# Bar plots of RMSE values by each model:
ggplot(rmse_results, aes(x = Model, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity", color = "black") +
  labs(title = "RMSE Values by Model",
       x = "Model",
       y = "RMSE") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate x-axis labels for bette

# Print forecasted values
print(ets_forecast$mean)

# Print confidence intervals
print(ets_forecast$lower)
print(ets_forecast$upper)
```

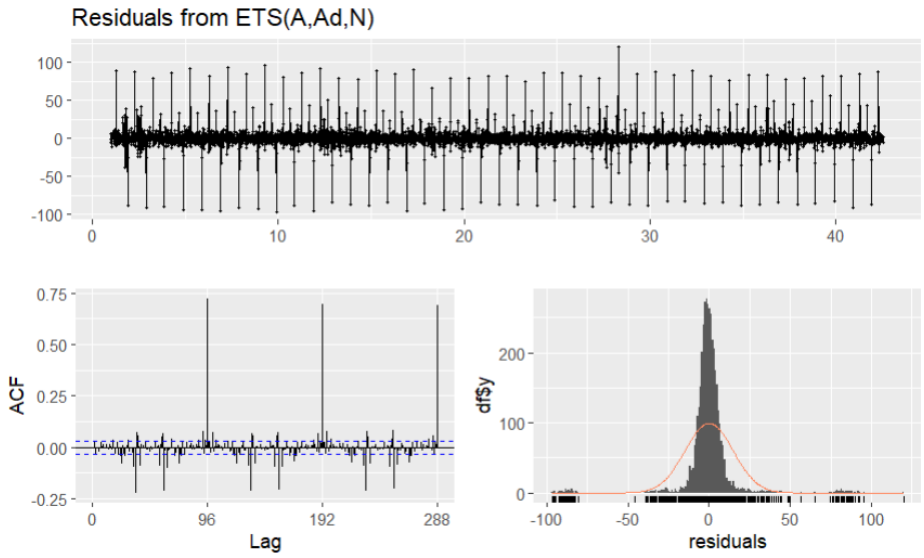

Figure 1: image

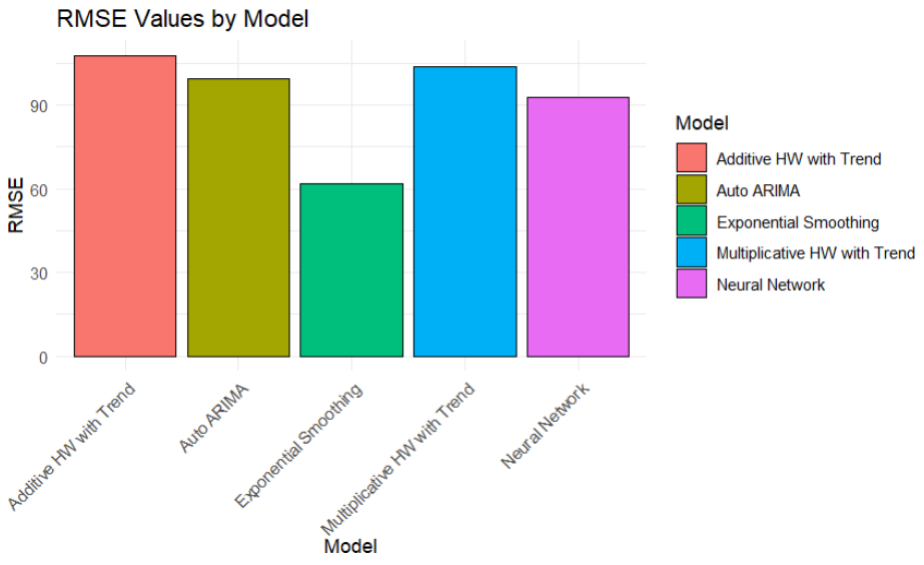

Figure 2: image

```
# Plot the forecast
plot(ets_forecast)
```

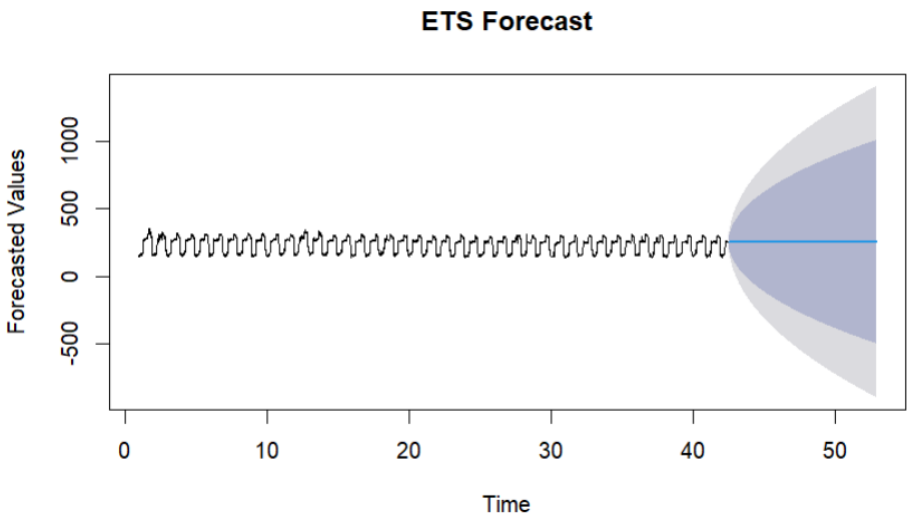

Figure 3: image

Using outdoor temperature

```
# Prepare the training and test sets
train_power <- ts(train_data$Power, frequency = 96)
test_power <- ts(test_data$Power, start = c(floor(train_size/96)+1, 1), frequency = 96)
train_temp <- ts(train_data$Temp, frequency = 96)
test_temp <- ts(test_data$Temp, start = c(floor(train_size/96)+1, 1), frequency = 96)

# 1. Auto ARIMA with External Regressor
arima_model <- auto.arima(train_power, xreg = train_temp)
arima_forecast <- forecast(arima_model, xreg = test_temp)
arima_rmse <- sqrt(mean((test_power - arima_forecast$mean)^2, na.rm = TRUE))

# 2. Neural Networks (NNET)
nnet_model <- nnetar(train_power)
nnet_forecast <- forecast(nnet_model, h = length(test_power))
nnet_rmse <- sqrt(mean((test_power - nnet_forecast$mean)^2, na.rm = TRUE))

# Print RMSE results for comparison
rmse_results <- data.frame(
  Model = c("Auto ARIMA", "Neural Network"),
  RMSE = c(arima_rmse, nnet_rmse)
)
```

```
print(rmse_results)
```

```
# Prepare RMSE results for comparison
rmse_results <- data.frame(
  Model = c("Auto ARIMA", "Neural Network"),
  RMSE = c(arima_rmse, nnet_rmse)
)

# Bar plot of RMSE values by each model
ggplot(rmse_results, aes(x = Model, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity", color = "black") +
  labs(title = "RMSE Values by Model",
       x = "Model",
       y = "RMSE") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate x-axis labels for bette
```

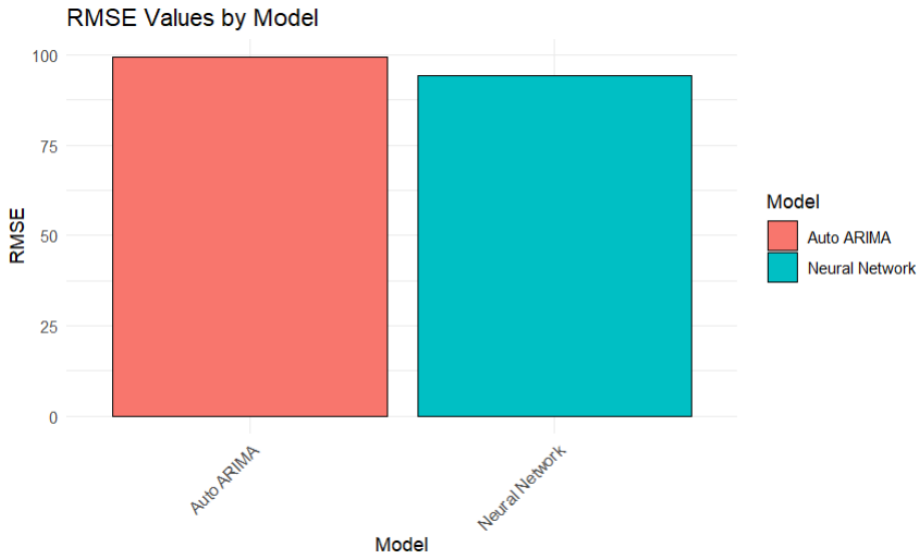

Figure 4: image

```
# Convert forecast results to data frames for ggplot2
# Auto ARIMA Data Frame
arima_forecast_df <- data.frame(
  Date = time(arima_forecast$mean),
  Forecast = arima_forecast$mean,
  Lower80 = ifelse(!is.null(arima_forecast$lower), arima_forecast$lower[, "80%"], NA),
  Upper80 = ifelse(!is.null(arima_forecast$upper), arima_forecast$upper[, "80%"], NA)
)
```

```
# Neural Network Data Frame (without confidence intervals)
nnet_forecast_df <- data.frame(
  Date = time(nnet_forecast$mean),
  Forecast = nnet_forecast$mean
)

# Plot both forecasts using ggplot2
ggplot() +
  geom_line(data = arima_forecast_df, aes(x = Date, y = Forecast, color = "Auto ARIMA"), siz
  geom_ribbon(data = arima_forecast_df, aes(x = Date, ymin = Lower80, ymax = Upper80), alpha
  geom_line(data = nnet_forecast_df, aes(x = Date, y = Forecast, color = "Neural Network"),
  labs(
    title = "Forecast Comparison: Auto ARIMA vs Neural Network",
    x = "Time",
    y = "Forecasted Power Values"
  ) +
  scale_color_manual(name = "Model", values = c("Auto ARIMA" = "blue", "Neural Network" = "r
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate x-axis labels for bette
```

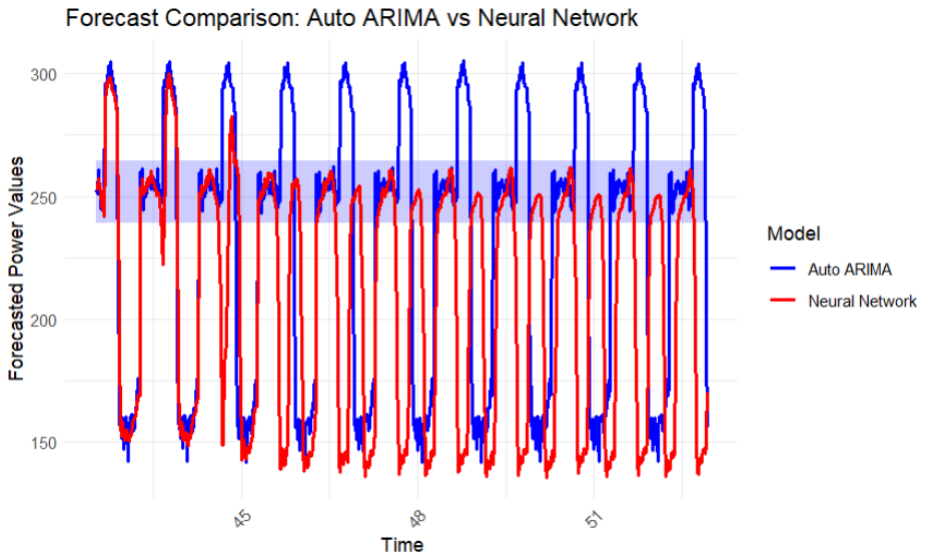

Figure 5: image

Summary : I have decided to finally go with ETS model for Power only variable and Arima for Power + Temp variables. Both models have shown best RMSE values during testing and best forecast graphs. Neural networks can be the better option if there was a more complicated model with more variables.