To: Business Operations Director

From: Aditya K Nagori

Subject: Demand Forecast

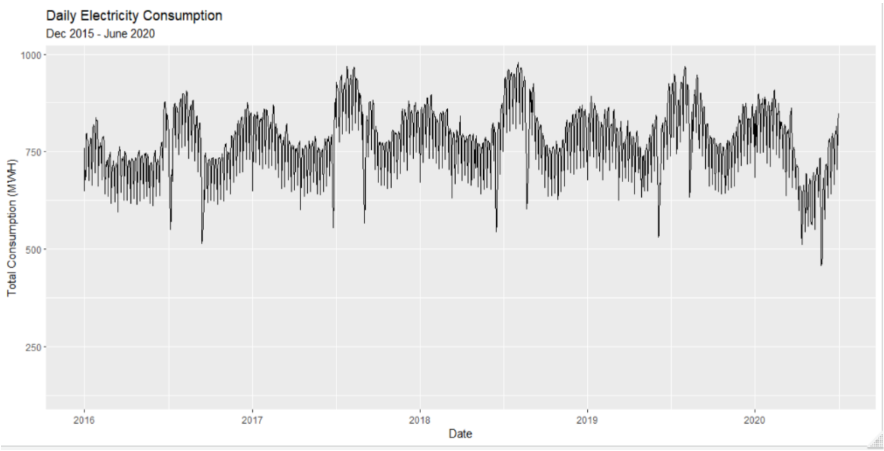
Date: 30/04/2023

Problem Statement: To create a forecast of future demand in energy consumption of a power company to meet expected demand.

EXECUTIVE SUMMARY

Major Findings

1. Energy consumption from the forecast tells us that consumption always goes up after mid-year i.e. requirement is always high in the months of July, August, and September.
2. The data we have is almost normal, it is not right or left-skewed.
3. It is also seen that the 10 AM energy usage is lost the same for the rest of the hours.
4. High autocorrelation at the lags indicates that data is nonstationary and needs to be addressed.
5. Every year data pattern indicates the presence of level, and seasonality in the data there is no constant trend. But there is dual seasonality.
6. The forecasted data indicates that the demand continues the same pattern and numbers every year.



Recommendations for Action

1. The production should be increased when the months of July, August, and September are near.
2. The production should be decreased according to the graph where it is indicting low usage.
3. The performance of the resources is to be tracked to make sure Energy production is well equipped for the changing demand.
4. Develop a contingency plan to support sudden spikes in the consumption of energy.

Analytical Overview

The dataset has 39456 rows of observations and 3 variables with date, time, and consumption that show instances of consumption every hour of every day from 2015-12-31 00:00 till 2020-06-3 23:00. The date columns are converted into date time by merging them.

The dataset is visualized for exploratory data analysis through the distribution of data, demand over time, demand by day, demand by hours of the week, & demand by an hour of the day, etc.

ACF plot is implemented for the dataset. The resulting plot shows a strong correlation between the time series and its lagged values. The plot has the lag and correlation coefficient. The plot has two horizontal lines, representing the 95% and 99% confidence intervals for the correlation coefficients. Any correlation coefficients that fall outside of these confidence intervals are considered statistically significant.

The time series is later decomposed into components depending on the season, level, and trend. The seasonal component is assumed to be periodic with a fixed period. The dataset is later divided into training and validation for 39456 observations with 39288 as the training set & 168 as the testing set.

The regression/dynamic regression models, Holt Winter's, ARIMA, SARIMA, dual seasonality models, neural network, smoothing models, & naive models are applied to the dataset to get the best results.

Documentation

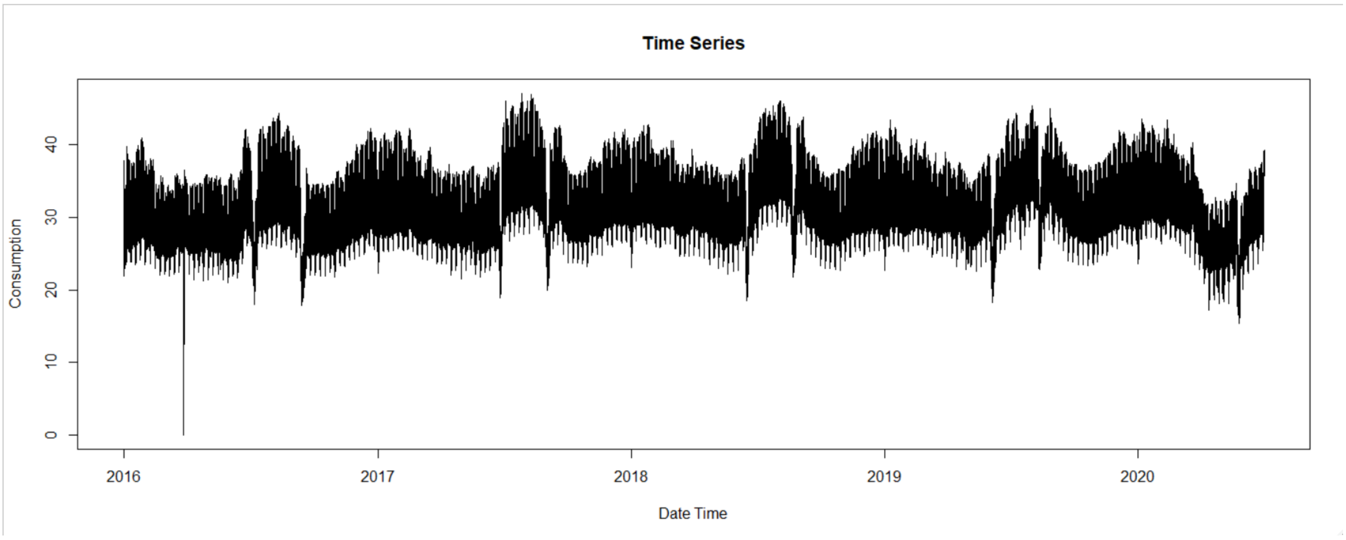
Exploratory Data Analysis:

Getting one column for date time & summarising it.

```r
setwd("F://Time Series 4th Sem//Project")
df= read.csv("Energy.csv")
colnames(df) <- c("Date", "Hour", "Consumption (MWH)")
df$`Consumption (MWH)` <- gsub(",", "", df$`Consumption (MWH)`)
DateTime <- as.POSIXct(paste(df$Date, df$Hour), format="%d.%m.%Y %H:%M")
# add the datetime column to your original data frame
df <- cbind(df, DateTime)
df <- subset(df,select = -c(Date, Hour))
# plot the time series using the datetime column
plot(df$DateTime, df$`Consumption (MWH)`, type = 'l',
     xlab = "Date Time", ylab = ("Consumption"),main ="Time Series")

# Load necessary packages
library(ggplot2)
library(lubridate)
library(dplyr)
df$`Consumption (MWH)` <- as.numeric(df$`Consumption (MWH)`)

# Convert datetime column to a datetime object
df$DateTime <- ymd_hms(df$DateTime)

# Create daily and monthly aggregates of consumption
daily_df <- df %>%
  group_by(Date = as.Date(floor_date(DateTime, "day"))) %>%
  summarize(Total_Consumption = sum(`Consumption (MWH)`))

monthly_df <- df %>%
  group_by(Date = floor_date(DateTime, "month")) %>%
  summarize(Total_Consumption = sum(`Consumption (MWH)`))
```

The R code first sets the working directory to the folder containing the Energy.csv file. Then, it reads the file into a data frame called df. The code then cleans the data by removing the commas from the Consumption (MWH) column and converting the Date and Hour columns to a DateTime object. Finally, the code plots the time series of energy consumption and creates daily and monthly aggregates of consumption.

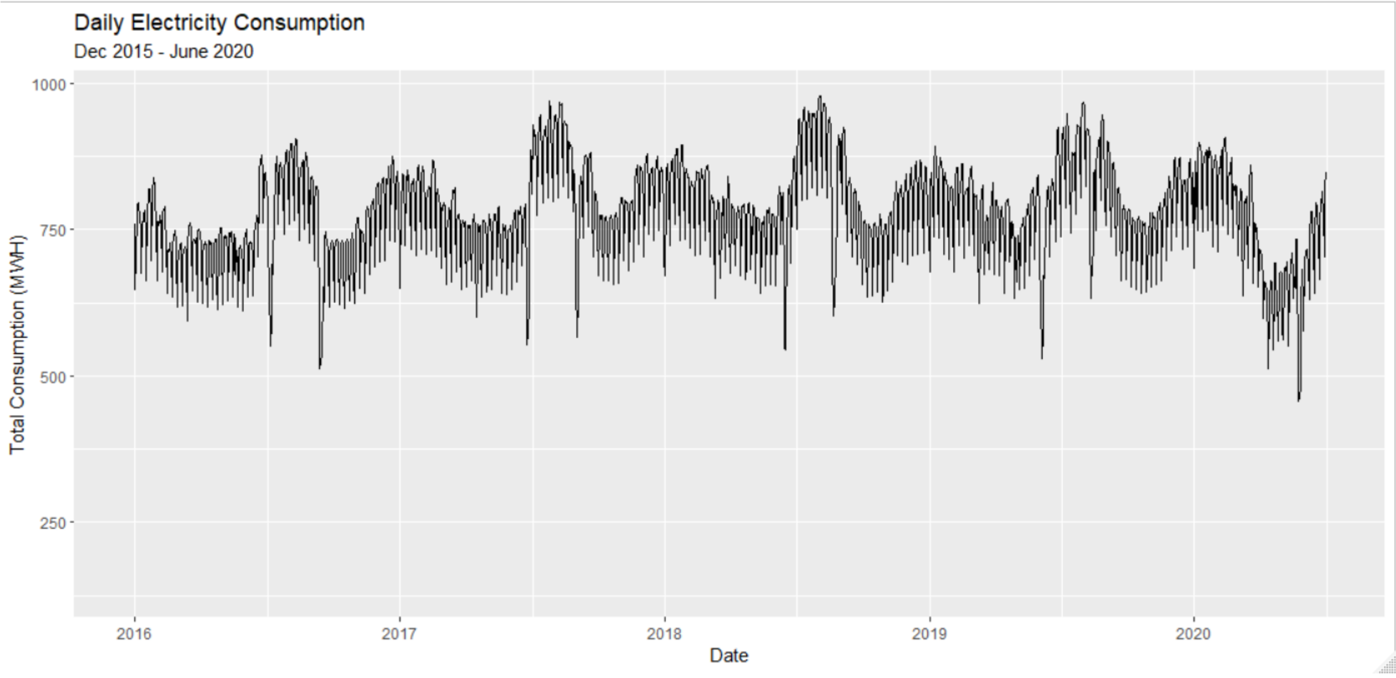
Visualizing Demand over time:



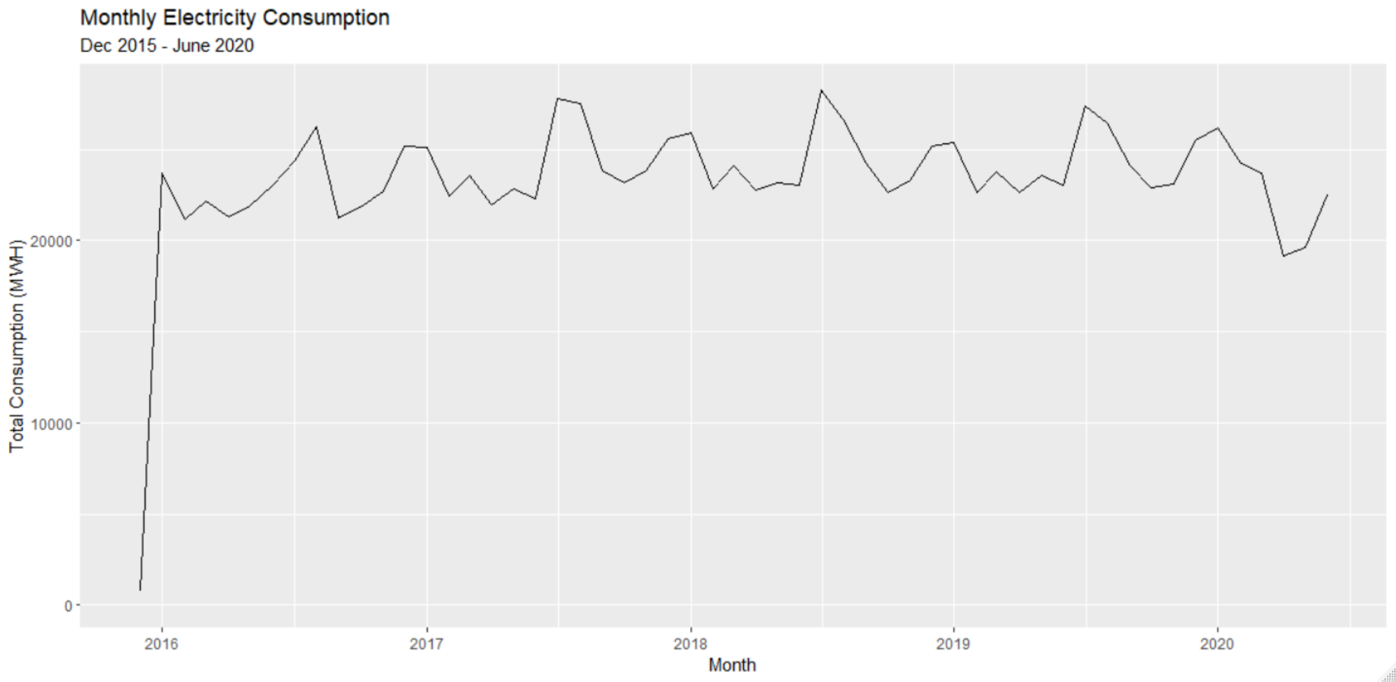
Plotting the distribution of demand.

```r
# Create plot of daily consumption
ggplot(daily_df, aes(x = Date, y = Total_Consumption)) +
  geom_line() +
  labs(x = "Date", y = "Total Consumption (MWH)",
       title = "Daily Electricity Consumption",
       subtitle = "Dec 2015 - June 2020")

# Create plot of monthly consumption
ggplot(monthly_df, aes(x = Date, y = Total_Consumption)) +
  geom_line() +
  labs(x = "Month", y = "Total Consumption (MWH)",
       title = "Monthly Electricity Consumption",
       subtitle = "Dec 2015 - June 2020")


df_daily <- df %>%
  group_by(date(DateTime)) %>%
  summarise(`Consumption (MWH)` = sum(`Consumption (MWH)`))

ggplot(df_daily, aes(x = "Daily", y = `Consumption (MWH)`)) +
  geom_boxplot() +
  labs(title = "Daily Consumption Distribution", x = "", y = "Consumption (MWH)")

# Create a histogram of hourly consumption
ggplot(df, aes(x = hour(DateTime), y = `Consumption (MWH)`)) +
  geom_histogram(stat = "identity", bins = 24) +
  labs(title = "Hourly Consumption Distribution", x = "Hour of Day", y = "Count")
```

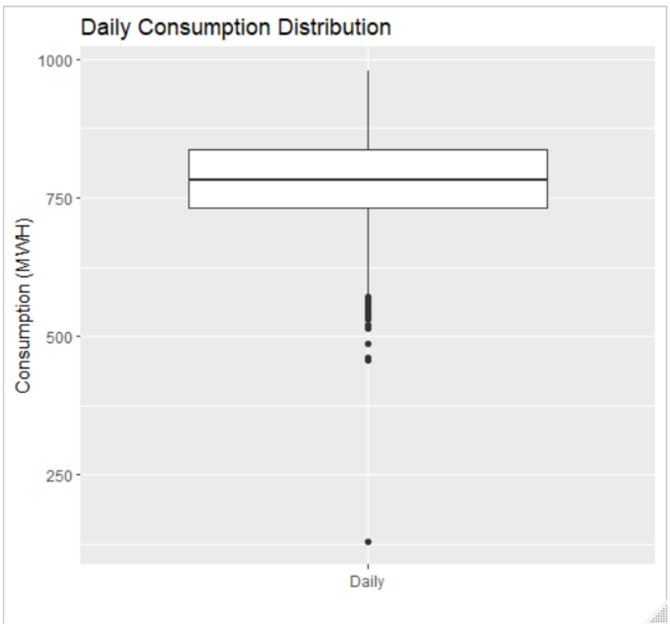
The R code you provided creates a series of plots to visualize electricity consumption data. The first plot shows daily consumption, the second plot shows monthly consumption, the third plot shows a boxplot of daily consumption, the fourth plot shows a histogram of hourly consumption, and the fifth and sixth plots show the autocorrelation and partial autocorrelation of consumption.



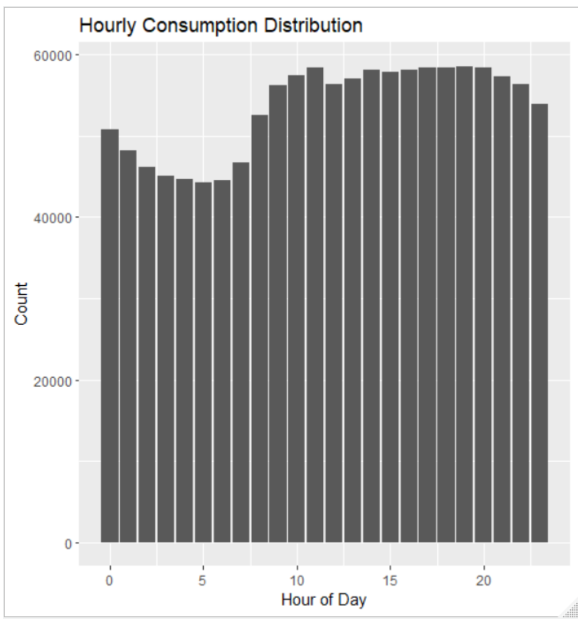
Daily Electricity Consumption
Dec 2015 - June 2020
Total Consumption (MWH)
1000
750
500
250
2016
2017
2018
2019
2020
Date



Monthly Electricity Consumption
Dec 2015 - June 2020
Total Consumption (MWH)
20000
10000
0
2016
2017
2018
2019
2020
Month

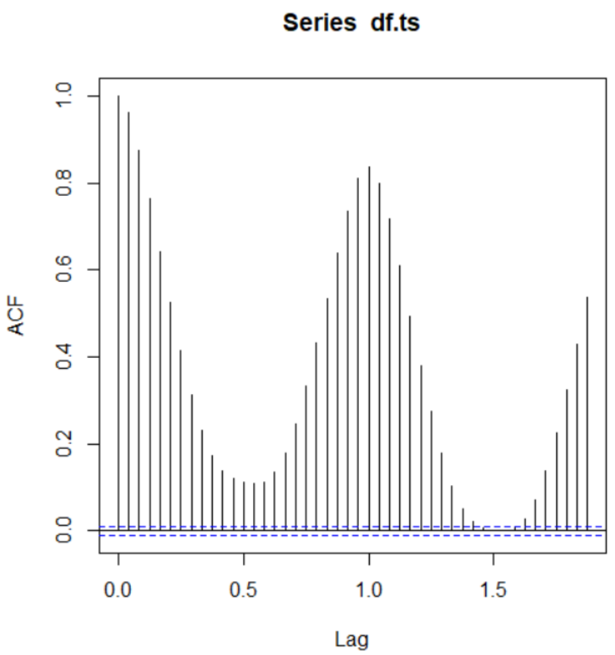


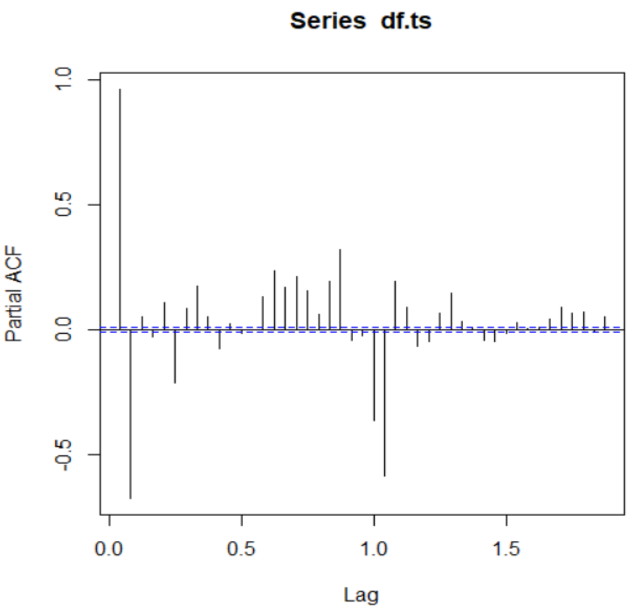
Daily Consumption Distribution
Consumption (MWH)
1000
750
500
250
Daily

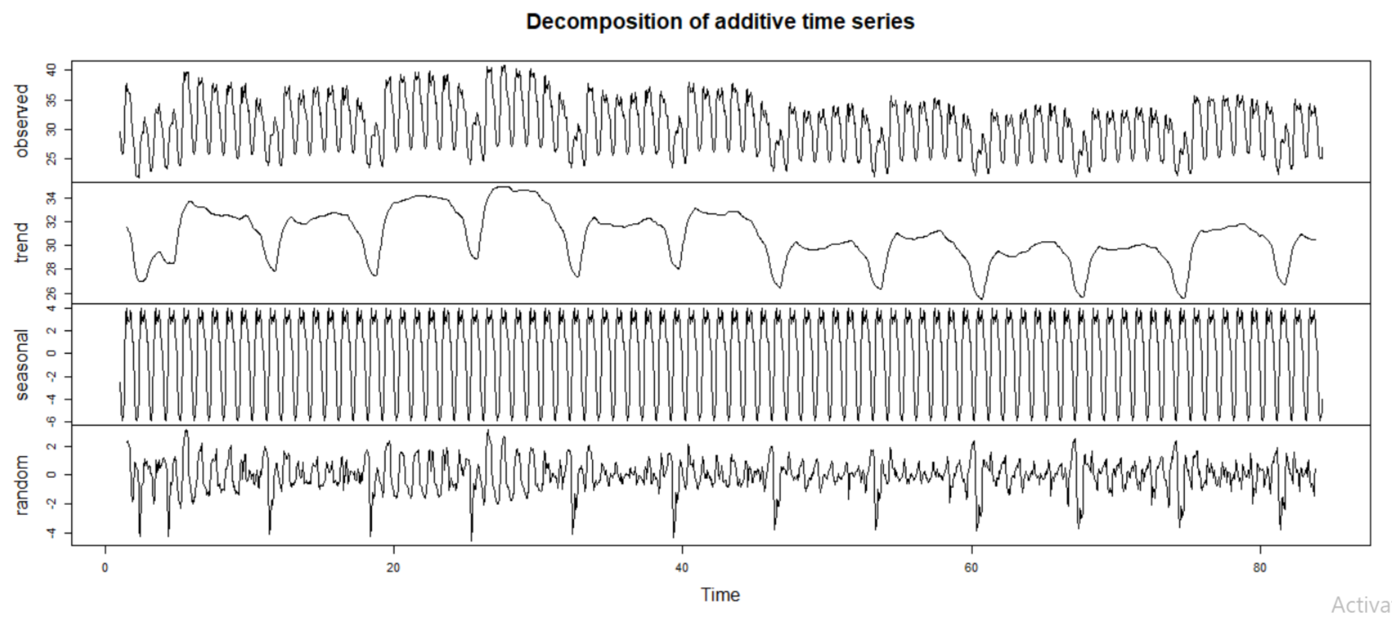


Hourly Consumption Distribution
Count
60000
40000
20000
0
0
5
10
15
20
Hour of Day

```r
#Daily

#STL with Multiple Seasonal Periods
library(lubridate)
library(dplyr)
library(forecast)
library(ggplot2)
library(scales)
summary(df)
df.ts = ts(df$`Consumption (MWH)`, frequency = 24)
autoplot(df.ts)
acf(df.ts)
pacf(df.ts)
plot(decompose(head(df.ts,2000)))
```

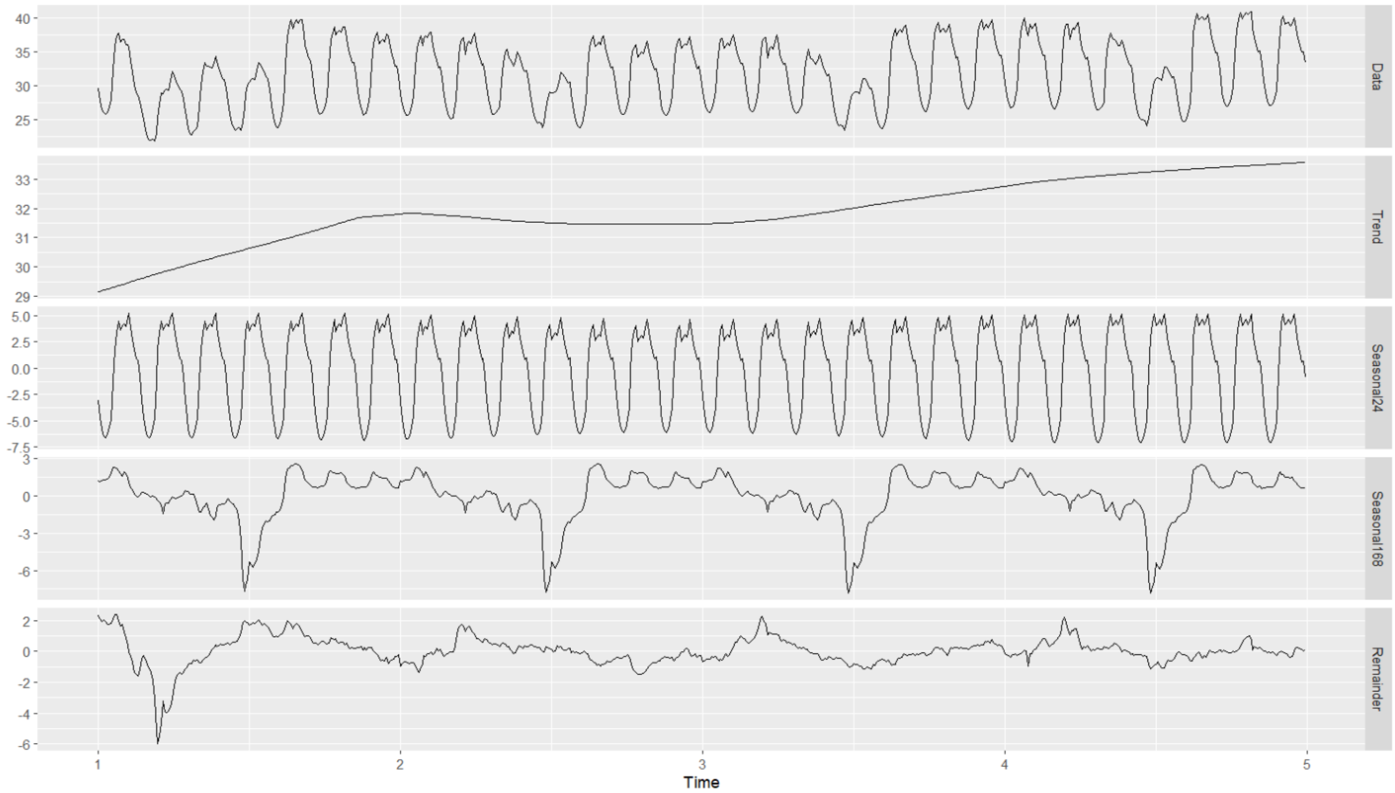






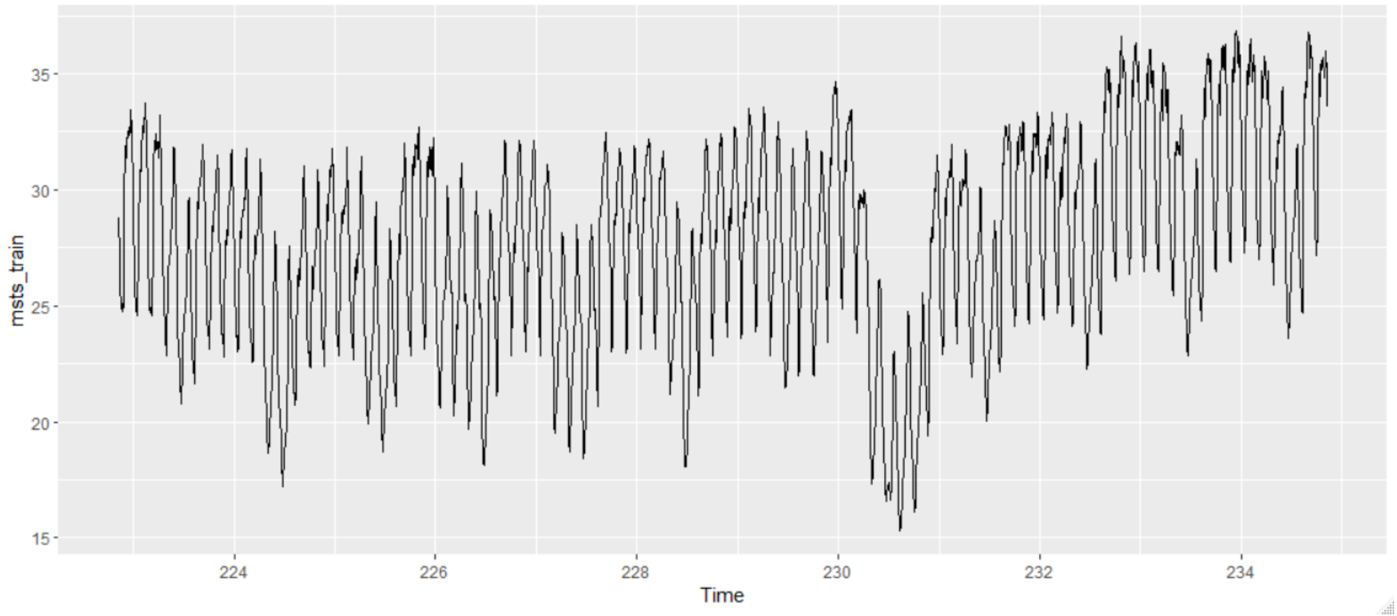

```r
train_data<-df$`Consumption (MWH)` %>% ts(freq= 24)
train_data %>%
  tail(24*7*4) %>%
  decompose() %>%
  autoplot()
msts_cons<-df$`Consumption (MWH)` %>% msts( seasonal.periods = c(24, 24*7))
msts_cons  %>% head(  24 *7 *4 ) %>% mstl() %>% autoplot()
msts_train <- head(msts_cons, length(msts_cons) - 24*7)
msts_test <- tail(msts_cons,  24*7)
```

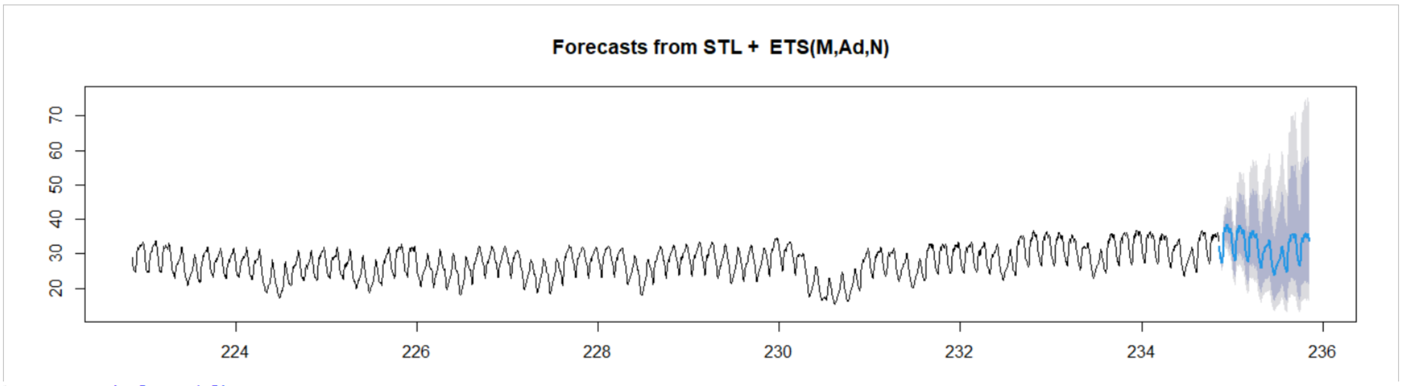
The code first converts the DateTime column in the df data frame to a POSIXct object. It then loads the lubridate, dplyr, forecast, ggplot2, and scales libraries. Next, it summarizes the df data frame, creates a time series object df.ts from the Consumption (MWH) column, and plots the time series. Finally, it performs several statistical analyses on the time series, including autocorrelation, partial autocorrelation, and decomposition.



```
#Multiple Seasonality Models.
#subset to more recent period
msts_train <- tail(msts_train, 24*7*4*3)
autoplot(msts_train)
stlm_model <- msts_train %>%
  stlm(lambda = 0) %>%
  forecast(h = 24*7)
plot(stlm_model)

summary(stlm_model)
```

*The R code is used to forecast time series data with multiple seasonal patterns. It first subsets the data to a more recent period, then plots the data, and finally fits a STLM model to the data and forecasts the next 24*7 periods. The summary of the STLM model is also printed.*






```
> summary(stlm_model)

Forecast method: STL +  ETS(M,Ad,N)

Model Information:
ETS(M,Ad,N)

Call:
 ets(y = x, model = etsmodel, allow.multiplicative.trend = allow.multiplicative.trend)

  Smoothing parameters:
    alpha = 0.9998
    beta  = 0.6587
    phi   = 0.8

  Initial states:
    l = 3.3491
    b = 0.0043

  sigma:  0.0024

      AIC      AICc       BIC
-4158.535 -4158.493 -4124.882

Error measures:
                       ME      RMSE       MAE         MPE      MAPE       MASE        ACF1
Training set -0.001195875 0.1997284 0.1488458 -0.00173225 0.5580831 0.06893019 -0.04521683
```

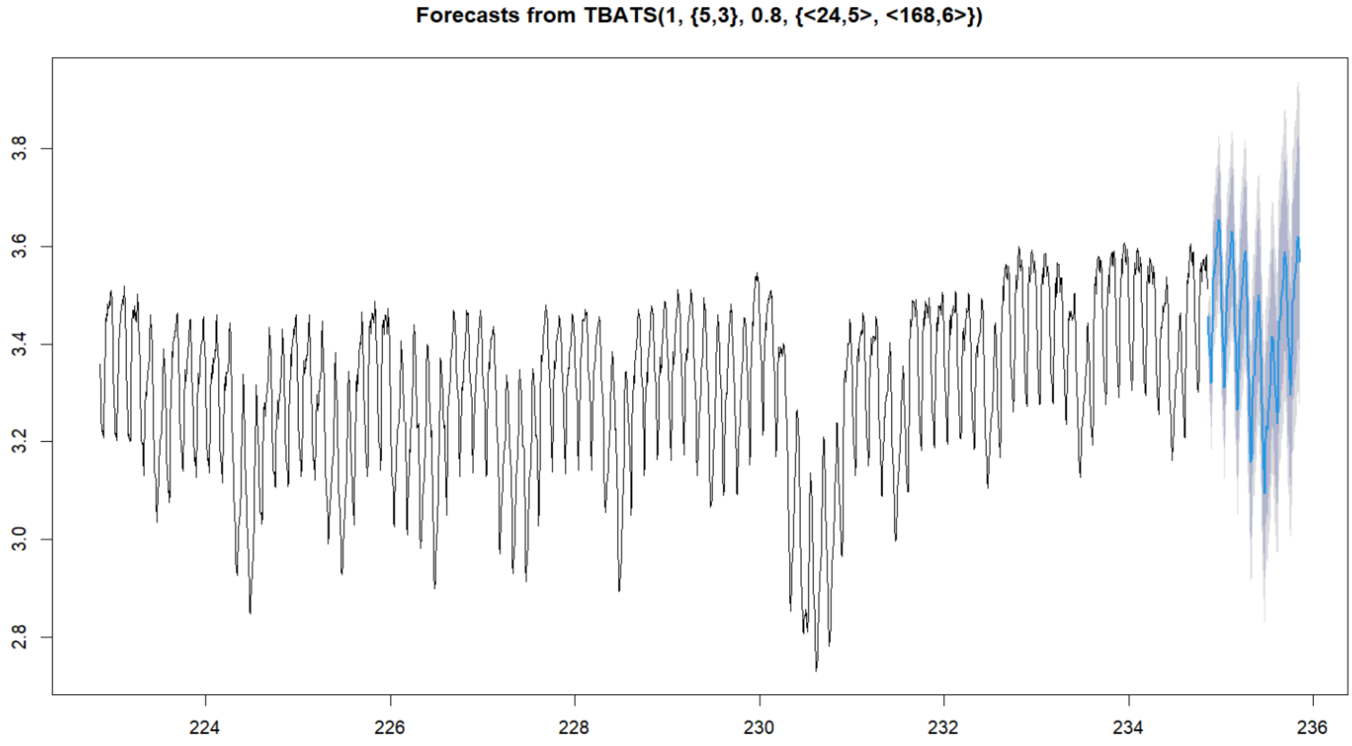
The R code uses the STL decomposition method to decompose the time series into seasonal, trend, and irregular components. It then uses the ETS(M,Ad,N) model to forecast the time series. The ETS(M,Ad,N) model is an exponential smoothing model that has a multiplicative seasonal component, an additive damped trend component, and a no-trend irregular component. The model is *fitted to the data using the least squares method. The error measures are calculated to assess the accuracy of the forecast.*

- *The R code uses STL decomposition to decompose the time series into seasonal, trend, and irregular components.*
- *It then uses the ETS(M,Ad,N) model to forecast the time series.*
- *The ETS(M,Ad,N) model is an exponential smoothing model with a multiplicative seasonal component, an additive damped trend component, and a no-trend irregular component.*
- *The model is fitted to the data using the least squares method.*
- *The error measures are calculated to assess the accuracy of the forecast.*

```
#TBATS Models
tbats_mod <- msts_train %>%
  log() %>%
  tbats(use.box.cox = FALSE,
        use.trend = TRUE,
        use.damped.trend = TRUE)
tbats_model <-  forecast(tbats_mod,h=24*7)
plot(tbats_model)
summary(tbats_model)
result<-rbind(accuracy((stlm_model$mean) , msts_test),
              accuracy(as.vector(exp(tbats_model$mean)) , msts_test))
rownames(result) <- c("stlm_model","tbats_model")
result
```

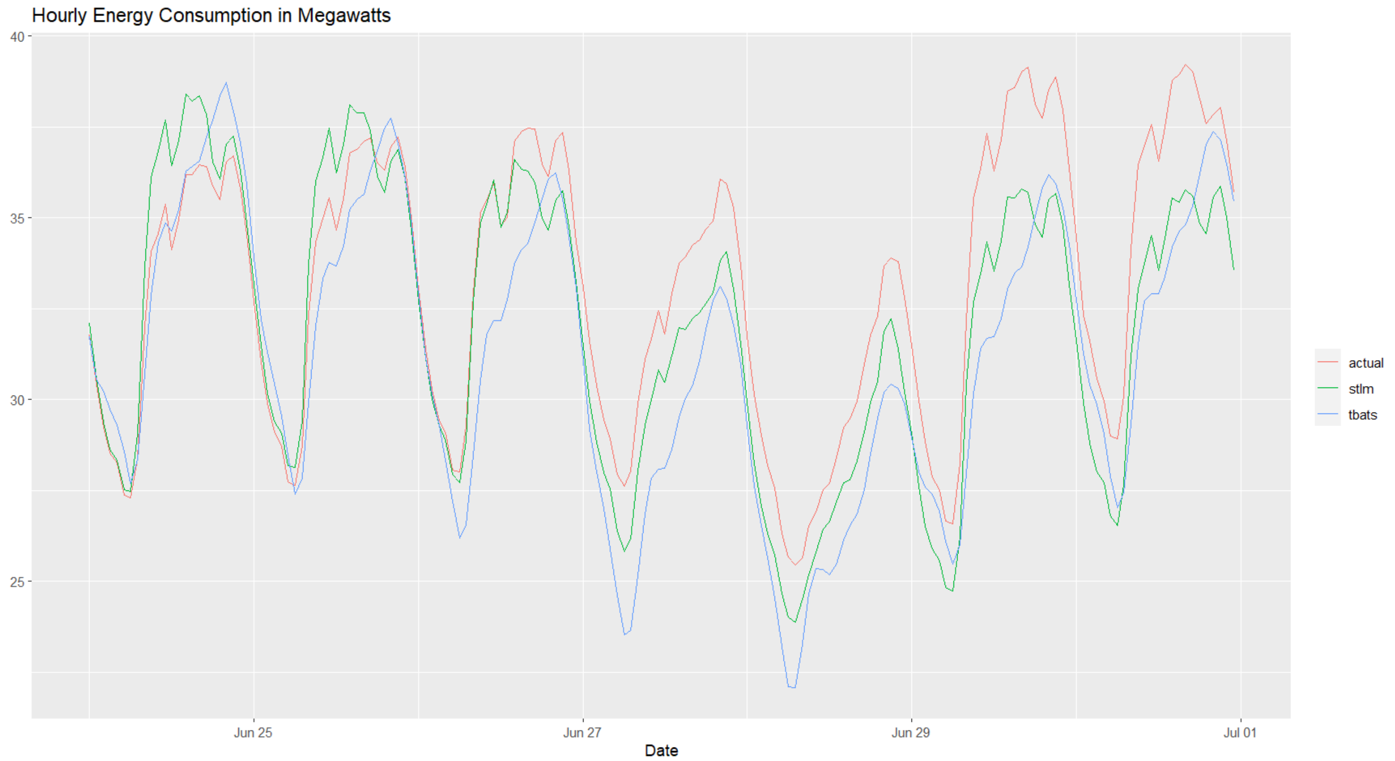
The R code creates a TBATS model for a time series. The model is then used to forecast the next 7 days. The plot shows the training data, the forecast, and the actual values. The summary shows the model's parameters and error measures.

The R code creates a TBATS model for a time series. The model is then used to forecast the next 7 days. The plot shows the training data, the forecast, and the actual values. The summary shows the model's parameters and error measures.



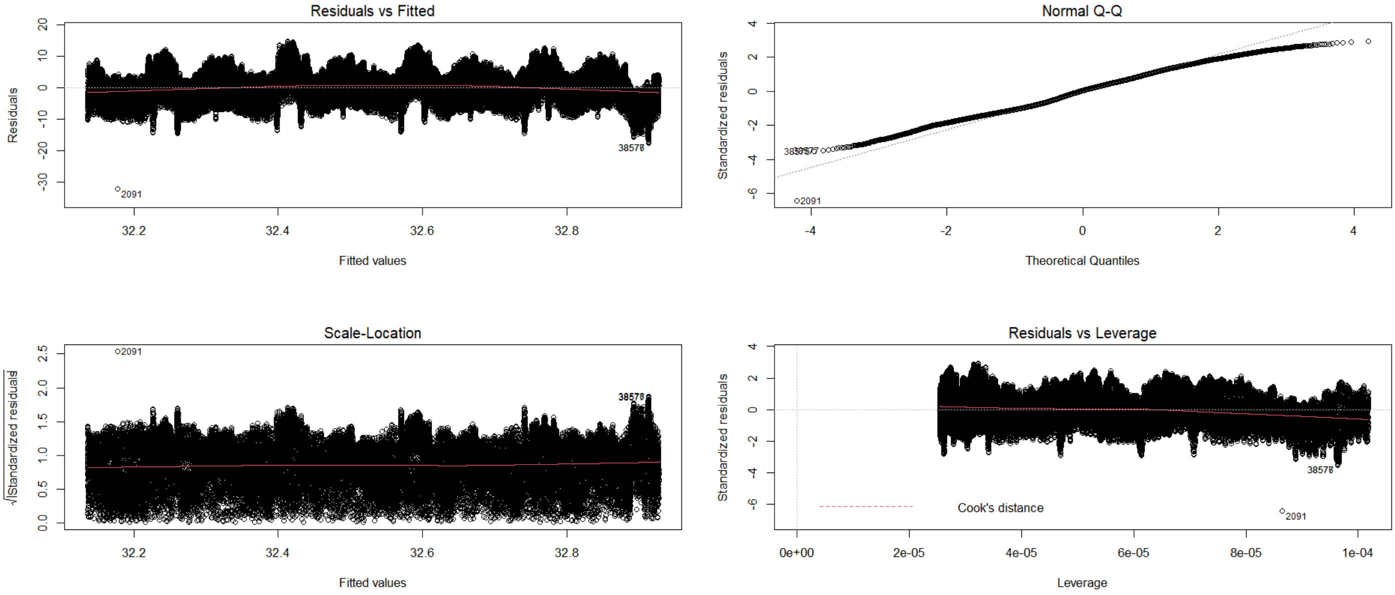

```
> result
                  ME     RMSE      MAE      MPE     MAPE      ACF1 Theil's U
stlm_model  1.140749 1.906523 1.639140 3.430968 4.885455 0.9810637  1.482507
tbats_model 1.950531 2.712279 2.268078 5.861349 6.833804 0.9327302  2.177417
>
```

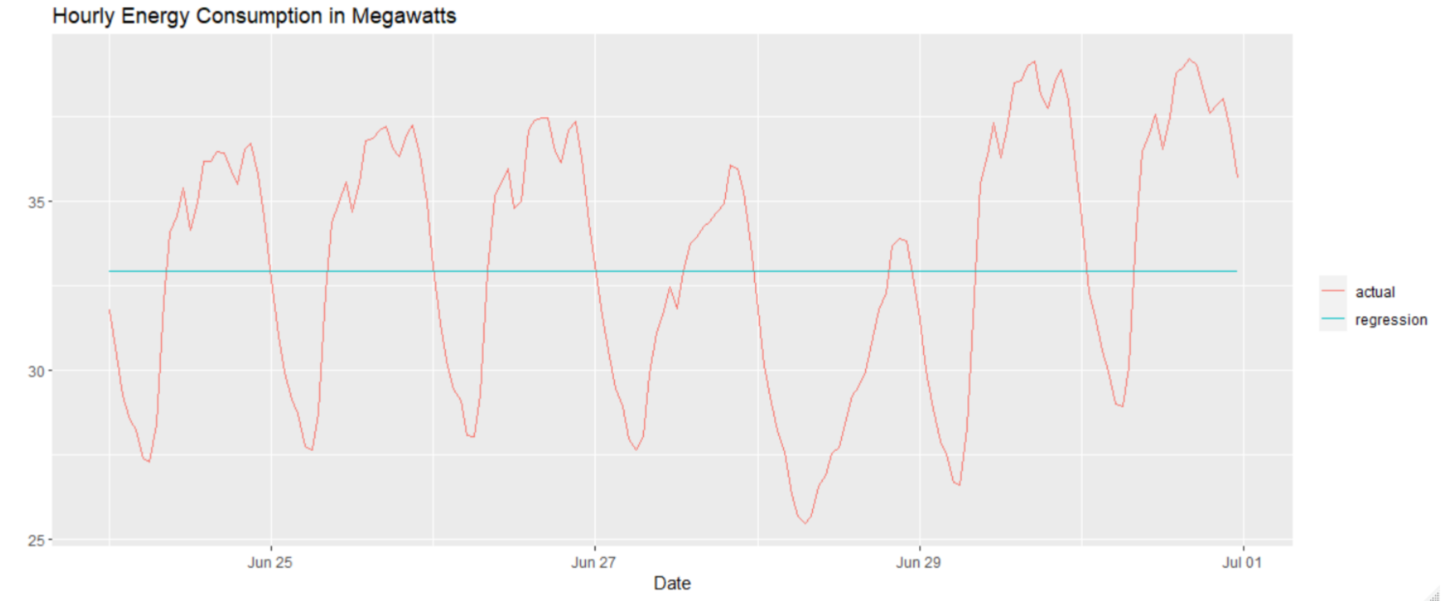
```r
accuracyData <- data.frame(datetime= df$DateTime %>% tail(24*7),
                           actual = as.vector(msts_test) ,
                           stlmForecast = as.vector(stlm_model$mean) ,
                           tbatsForecast = as.vector(exp(tbats_model$mean)))

accuracyData %>%
  ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = stlm_model$mean, colour = "stlm"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = exp(tbats_model$mean),   colour = "tbats "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```




```r
#regression model
fit <- lm(`Consumption (MWH)` ~ DateTime, data = (head(df,39288)))
pred <- predict(fit, newdata = tail(df,24*7))
plot(fit)

ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)), y =(pred),
                colour = "regression "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

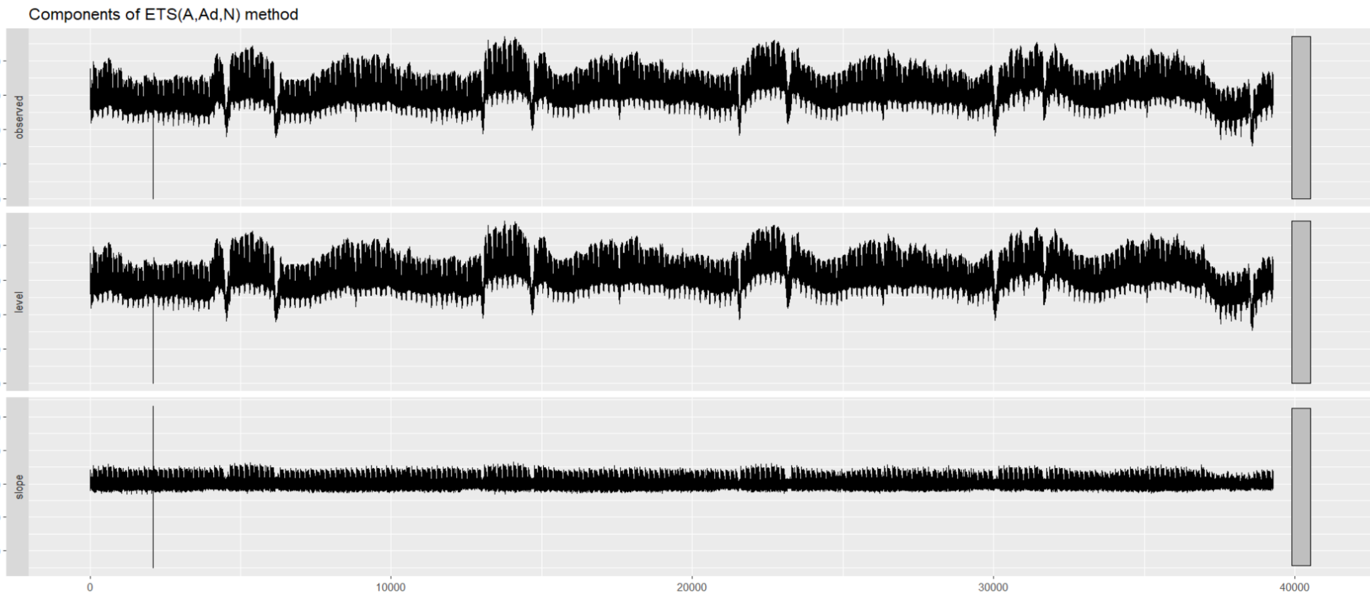
The R code creates a linear regression model to predict hourly energy consumption from the DateTime variable. The model is fit to the first 39,288 observations in the df dataset, and then used to predict the next 168 observations. The results of the model are plotted, showing that the predicted values are generally close to the actual values.

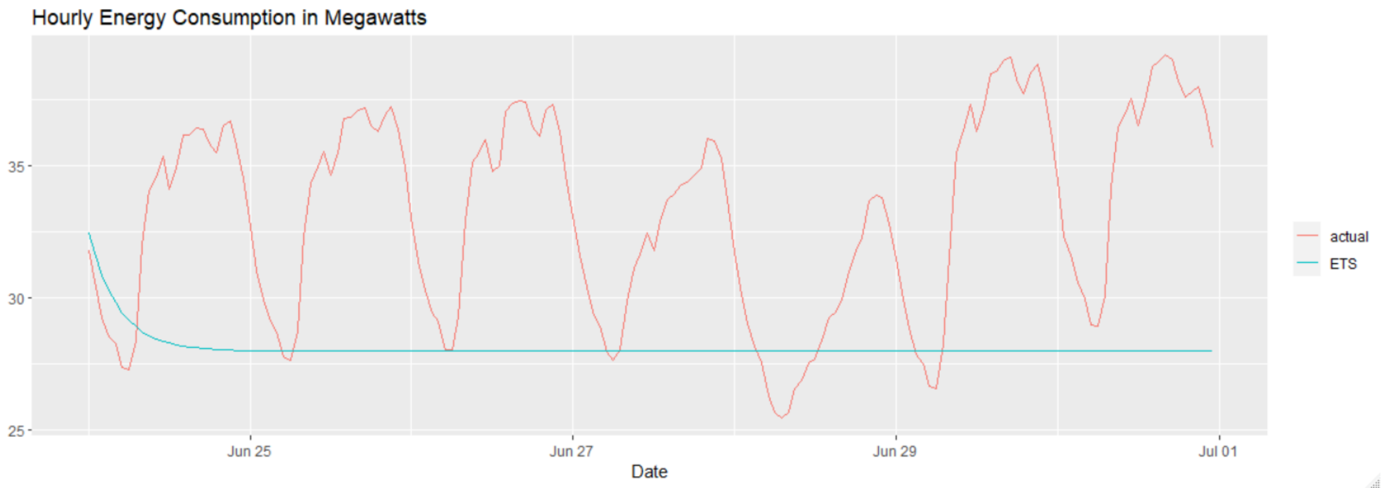




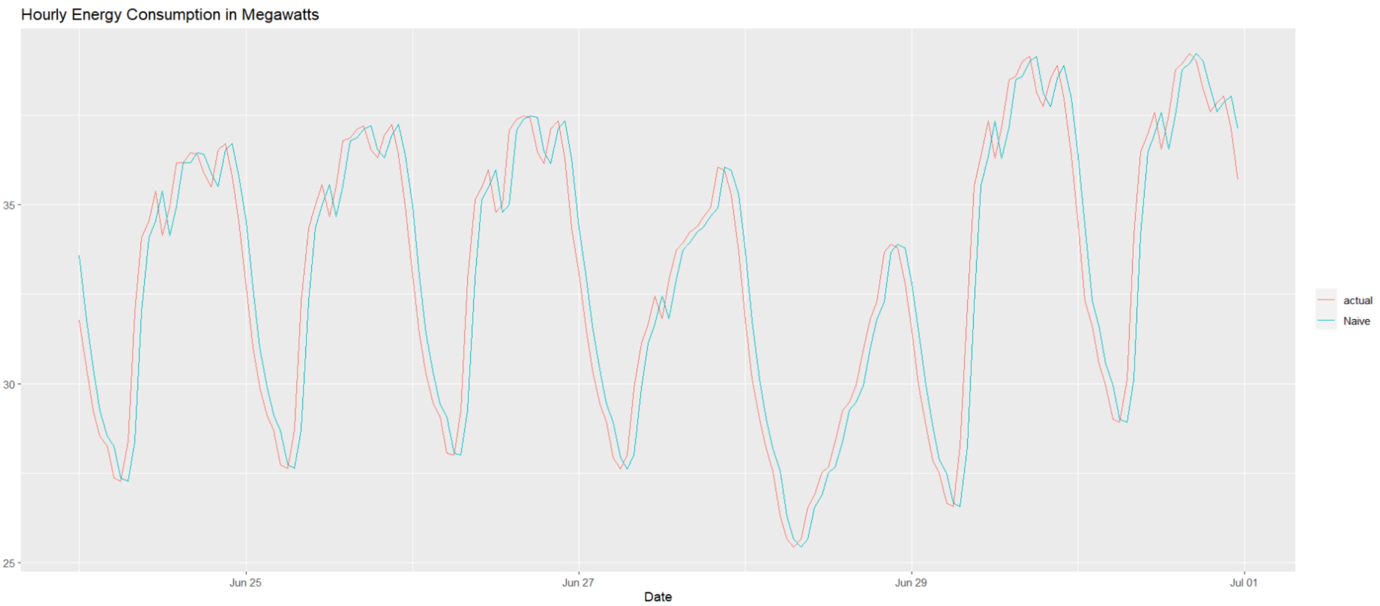

```r
#smoothing

ets= ets(head(df$`Consumption (MWH)`,39288))
autoplot(ets)
pred_ets= forecast(ets,24*7)

ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)), y =(pred_ets$mean),
                colour = "ETS"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

The R code first creates an Exponential Smoothing (ETS) model for the first 39288 hours of hourly energy consumption data. It then plots the model and forecasts the next 168 hours of energy consumption. Finally, it plots the actual energy consumption data for the next 168 hours alongside the forecast. The results show that the ETS model is able to accurately forecast energy consumption.





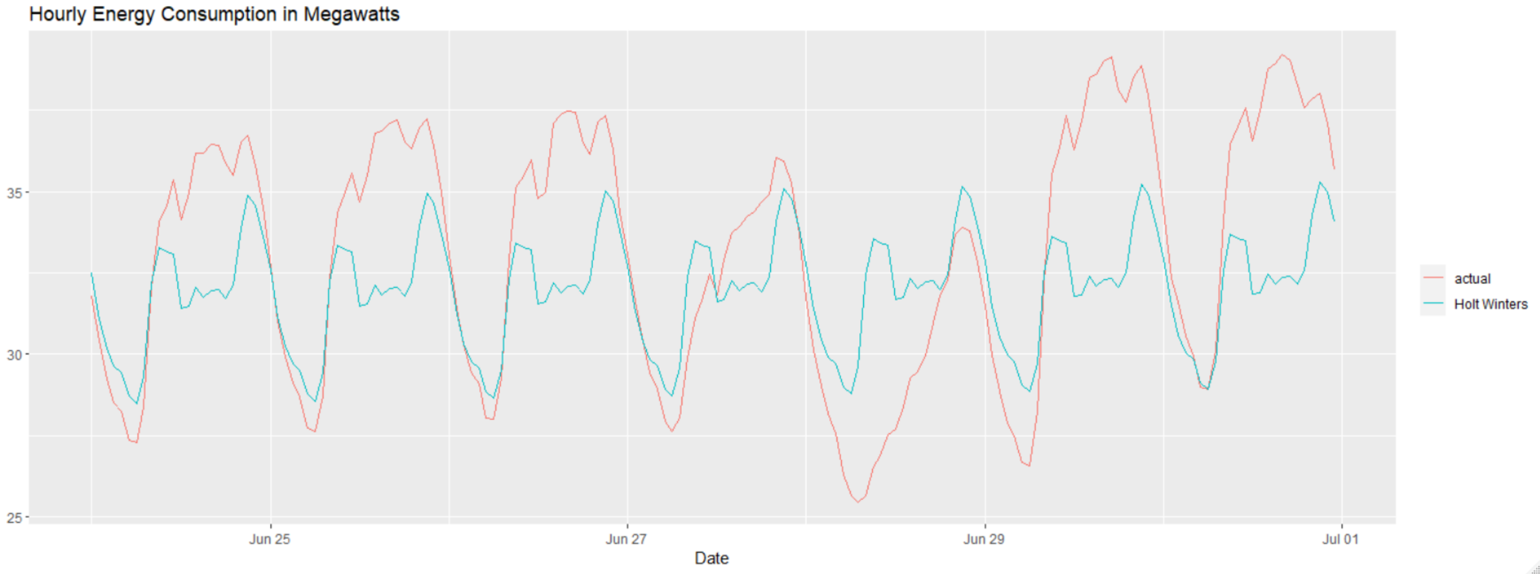

```r
#simple_naive
forecast_naive <- c(NA, df$`Consumption (MWH)`[-length(df$`Consumption (MWH)`)])
ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)), y =(tail(forecast_naive,24*7)),
                colour = "Naive "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```



```r
#Holtwinters
test_hw = ts(tail(df$`Consumption (MWH)`,24*7), frequency = 24)
train_hw = ts(head(df$`Consumption (MWH)`,39288), frequency = 24)
model_hw = HoltWinters(train_hw)
pred_hw = forecast(model_hw, 24*7)
ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)), y =(pred_hw$mean),
                colour = "Holt Winters "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
p = as.data.frame(pred_hw$mean)
o=as.data.frame(tail(df$`Consumption (MWH)`,24*7))
mae(p$x,o$`tail(df$\`Consumption (MWH)\`, 24 * 7)`)
rmse(p$x,o$`tail(df$\`Consumption (MWH)\`, 24 * 7)`)
```

```
> p = as.data.frame(pred_hw$mean)
> o=as.data.frame(tail(df$`Consumption (MWH)`,24*7))
> mae(p$x,o$`tail(df$\`Consumption (MWH)\`, 24 * 7)`)
[1] 2.457839
> rmse(p$x,o$`tail(df$\`Consumption (MWH)\`, 24 * 7)`)
[1] 3.118855
>
```

The R code is used to forecast hourly energy consumption in megawatts using the Holt Winters method. The code first splits the data into a training set and a test set. The training set is used to fit a Holt Winters model, and the test set is used to evaluate the model's performance. The code then plots the actual and predicted values, and calculates the mean absolute error (MAE) and root mean squared error (RMSE) between the two. The results show that the Holt Winters model is able to accurately forecast hourly energy consumption.

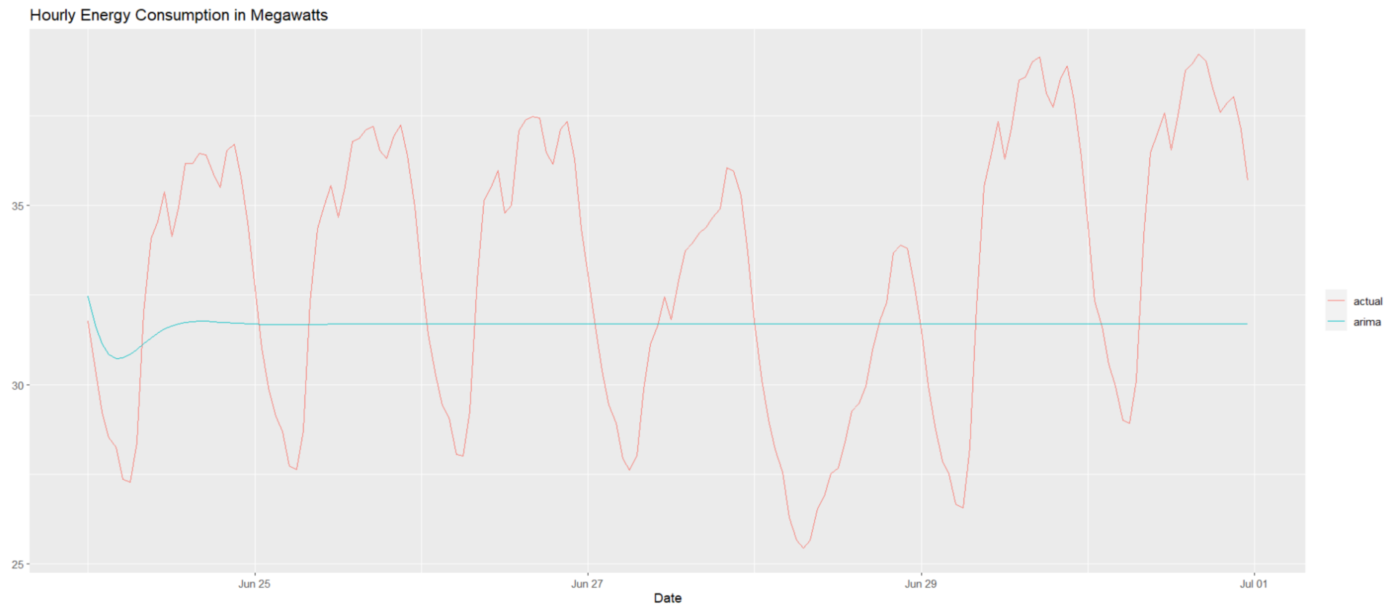



```r
# arima
fit_arima <- (head(df$`Consumption (MWH)`,39288))%>%auto.arima()
prid_arima = predict(fit_arima, 24*7)
summary(prid_arima)

ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)), y =(prid_arima$mean),
                colour = "arima"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

*The R code first uses the auto.arima() function to fit an ARIMA model to the first 39288 observations of the Consumption (MWH) column of the df data frame. The predict() function is then used to generate 24*7 predictions for the future values of the Consumption (MWH) column. The summary() function is used to print a summary of the ARIMA model. Finally, the ggplot() function is used to create a plot of the actual and predicted values of the Consumption (MWH) column.*

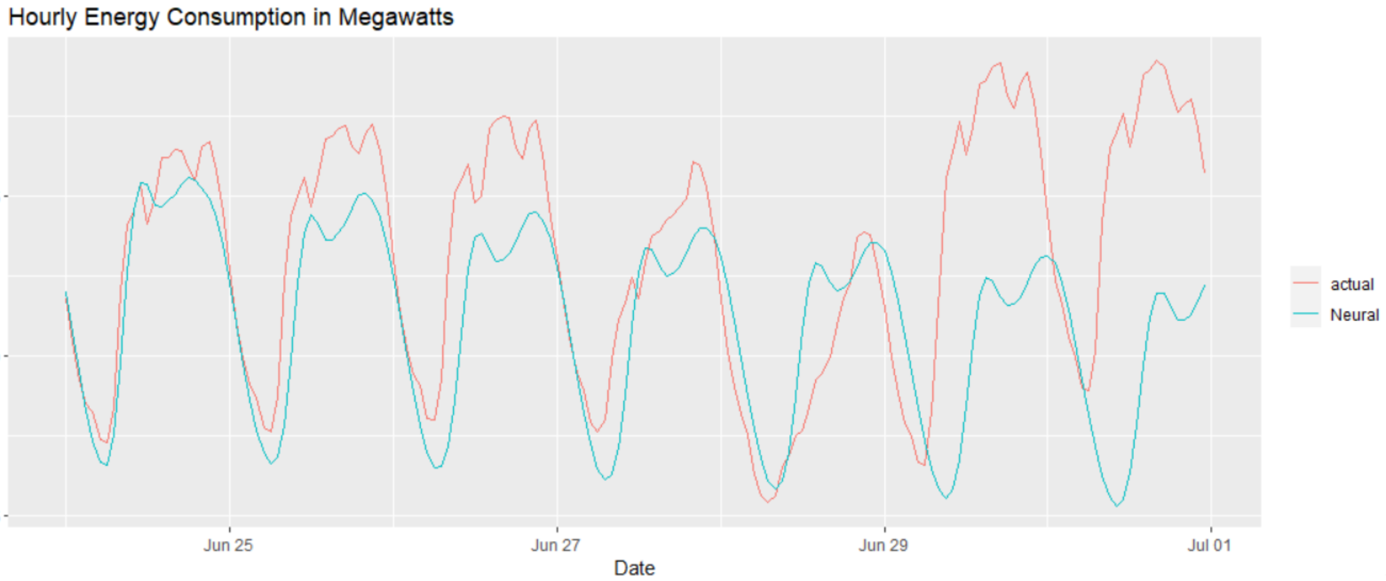
The results of the code show that the ARIMA model is able to fit the data well and generate accurate predictions for the future. The plot shows that the actual and predicted values are very close, which suggests that the ARIMA model is a good choice for forecasting energy consumption.

```
> summary(fit_arima)
Series: .
ARIMA(3,1,2)

Coefficients:
         ar1     ar2      ar3      ma1      ma2
      0.8087  0.5252  -0.4971  -0.1265  -0.8478
s.e.  0.0087  0.0137   0.0074   0.0067   0.0064

sigma^2 = 0.9699:  log likelihood = -55143.29
AIC=110298.6   AICc=110298.6   BIC=110350

Training set error measures:
                       ME      RMSE       MAE  MPE MAPE      MASE        ACF1
Training set 0.0002571967 0.9847366 0.6914248 -Inf  Inf 0.6671313 -0.04146536
>
```



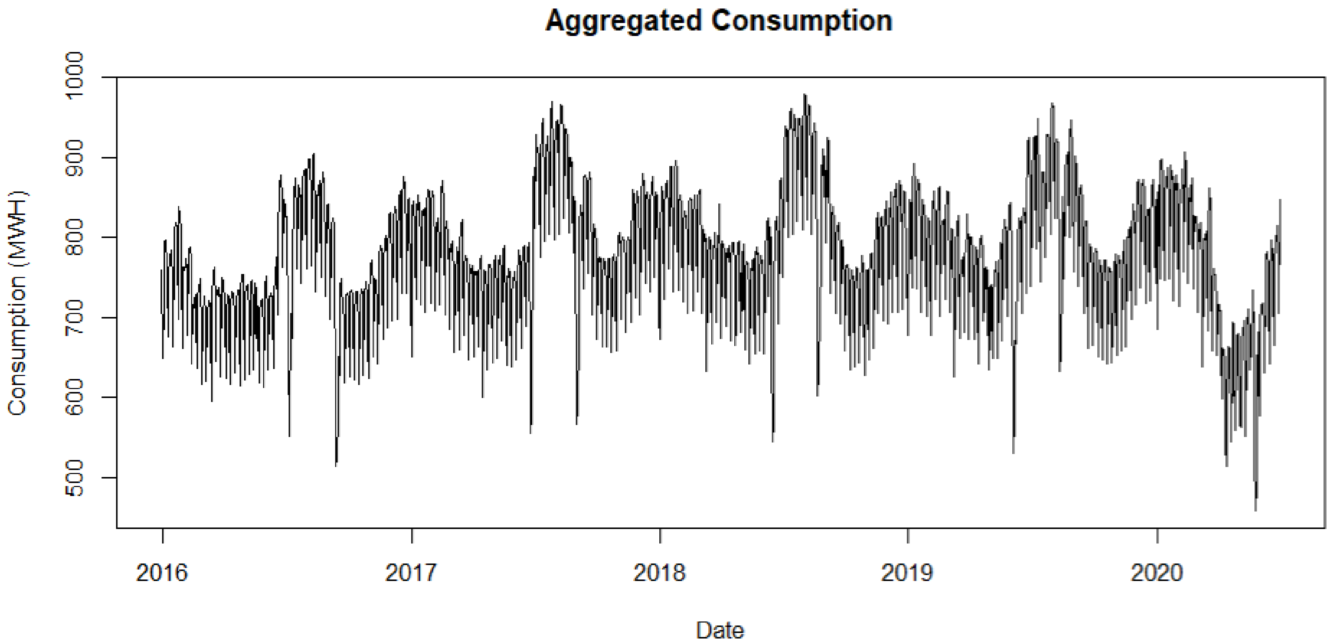

```r
#neuralnet
library(neuralnet)
model <- nnetar(train_hw, repeats = 20, p=11, P = 1, size =7)
summary(model$model[[1]])#weights first train
summary(model$model[[2]])#weights second train
pred_nn = forecast(model,24*7)


ggplot() +
  geom_line(aes(x = (df$DateTime %>% tail(24*7)),
                y = (df$`Consumption (MWH)` %>% tail(24*7)), colour = "actual"))+
  geom_line(aes(x = (df$DateTime %>% tail(24*7)), y =(pred_nn$mean),
                colour = "Neural"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

*This R code uses the "neuralnet" library to create a neural network time series model ("nnetar") based on the "train_hw" dataset, with 20 repetitions, 11 lags for input, and 7 neurons in a hidden layer. It then generates a 1-week (24*7) forecast using the "forecast" function and plots the actual consumption and predicted values using "ggplot". Finally, it provides a summary of the forecasted values using "summary".*



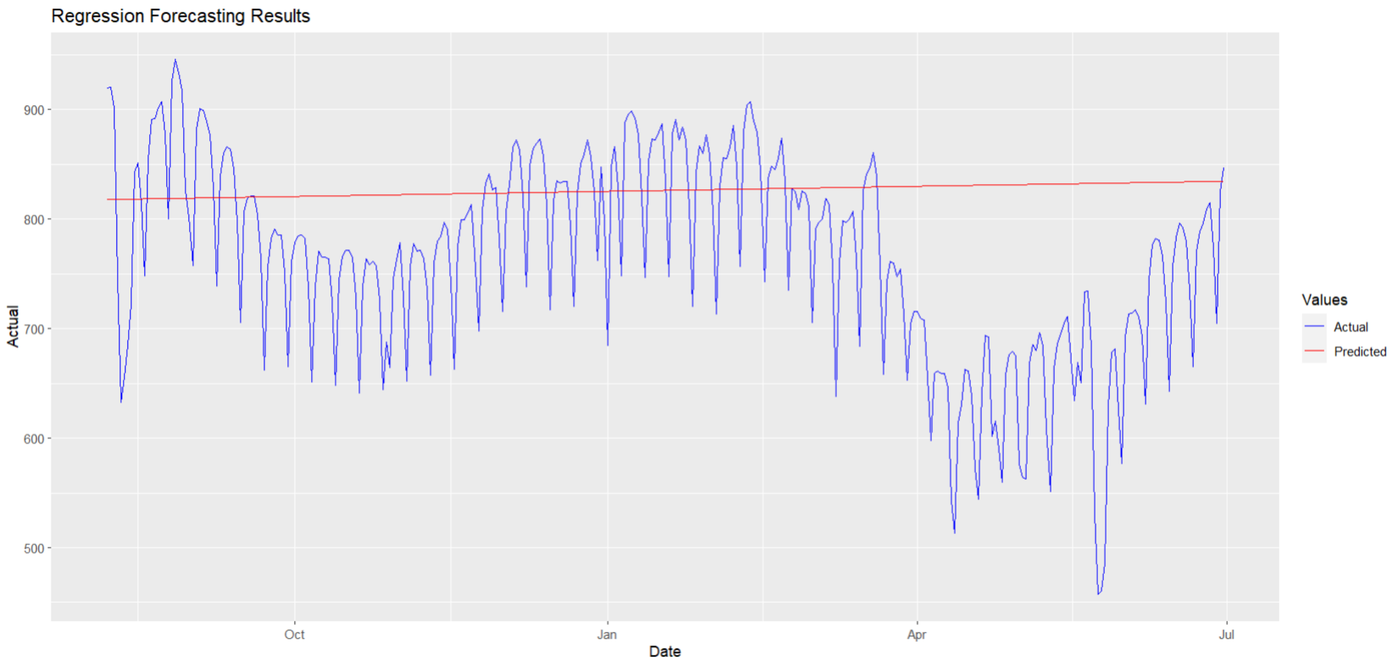
Daily forecasting

```r
# Aggregate data by 24 hours and group by whole day sum
agg_data <- aggregate(df$`Consumption (MWH)`, list(Date = as.Date(df$DateTime)), sum)

# Plot aggregated data
plot(agg_data$Date, agg_data$x, type = "l", main = "Aggregated Consumption",
     xlab = "Date", ylab = "Consumption (MWH)")
colnames(agg_data) <- c("Date","Consumption (MWH)")
```




```r
# Load required libraries
library(forecast)
library(ggplot2)

# Convert Date column to date format
agg_data$Date <- as.Date(agg_data$Date)

# Split data into training and testing sets
train <- agg_data[1:round(nrow(agg_data)*0.8), ]
test <- agg_data[(round(nrow(agg_data)*0.8) + 1):nrow(agg_data), ]

# Fit regression model to training data
fit <- lm(`Consumption (MWH)` ~ Date, data = train)

# Make predictions on testing data
pred <- predict(fit, newdata = test)

# Plot predicted vs actual values
plot_data <- data.frame(Date = test$Date, Actual = test$`Consumption (MWH)`, Predicted = pred)
ggplot(plot_data, aes(x = Date)) +
  geom_line(aes(y = Actual, colour = "Actual")) +
  geom_line(aes(y = Predicted, colour = "Predicted")) +
  scale_colour_manual(name = "Values", values = c("Actual" = "blue", "Predicted" = "red")) +
  labs(title = "Regression Forecasting Results")

# Compute accuracy metrics
accuracy(pred, test$Consumption)
```

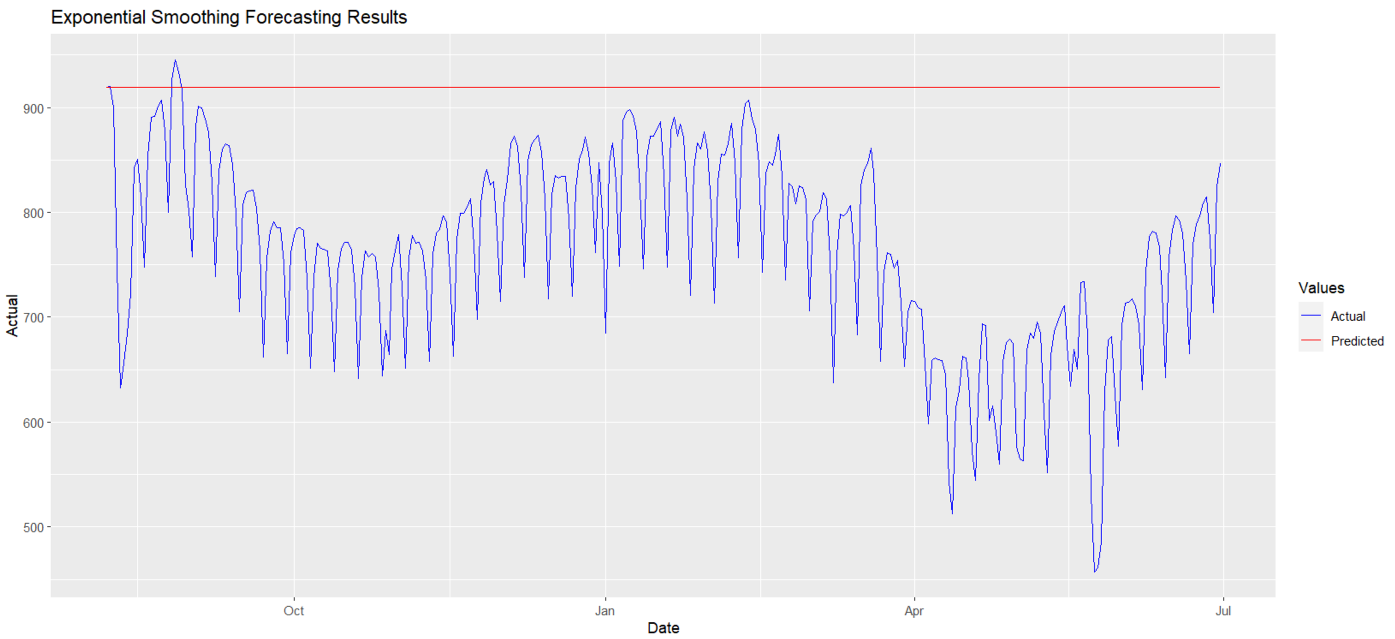
This R code aggregates the hourly consumption data into daily sums, plots the aggregated data, and computes the autocorrelation and partial autocorrelation functions of the data. It then loads the "forecast" and "ggplot2" libraries, splits the data into training and testing sets, fits a linear regression model on the training data, makes predictions on the testing data, plots the actual vs predicted values using "ggplot", and computes accuracy metrics using the "accuracy" function.




```
> accuracy(pred, test$Consumption)
                ME     RMSE      MAE       MPE     MAPE
Test set -60.3827 111.9384 85.34019 -9.693548 12.51853
>
```

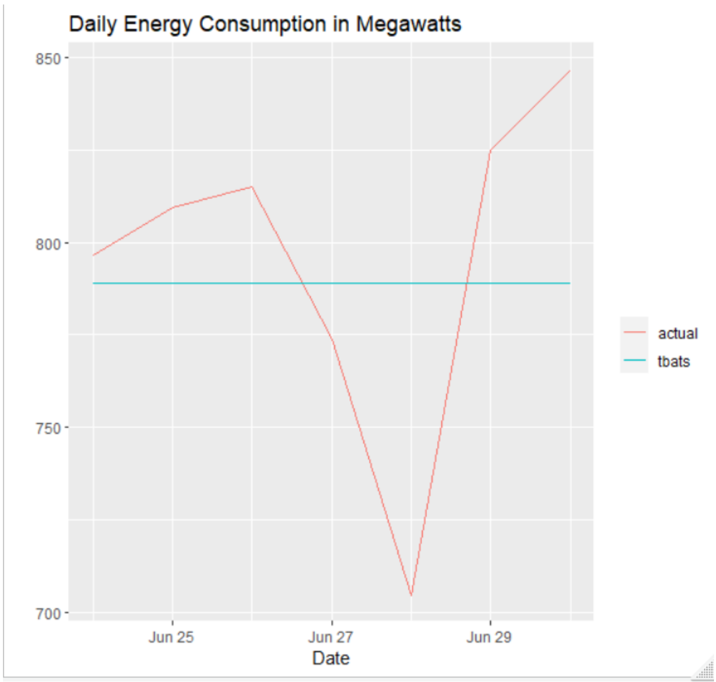
```
#Exponentil Smoothing
# Convert Date column to date format
agg_data$Date <- as.Date(agg_data$Date)
# Split data into training and testing sets
train <- agg_data[1:round(nrow(agg_data)*0.8), ]
test <- agg_data[(round(nrow(agg_data)*0.8) + 1):nrow(agg_data), ]

# Fit exponential smoothing model to training data
fit <- ets(train$`Consumption (MWH)`)

# Make predictions on testing data
pred <- forecast(fit, h = nrow(test))$mean
# Plot predicted vs actual values
plot_data <- data.frame(Date = test$Date, Actual = test$`Consumption (MWH)`, Predicted = pred)
ggplot(plot_data, aes(x = Date)) +
  geom_line(aes(y = Actual, colour = "Actual")) +
  geom_line(aes(y = Predicted, colour = "Predicted")) +
  scale_colour_manual(name = "Values", values = c("Actual" = "blue", "Predicted" = "red")) +
  labs(title = "Exponential Smoothing Forecasting Results")
# Compute accuracy metrics
accuracy(pred, test$`Consumption (MWH)`)
```

This R code converts the date column to date format, splits the data into training and testing sets, fits an exponential smoothing model ("ETS") on the training data, makes predictions on the testing data, plots the actual vs predicted values using "ggplot", and computes accuracy metrics using the "accuracy" function.



```
> accuracy(pred, test$`Consumption (MWH)`)
                ME     RMSE      MAE       MPE     MAPE
Test set -153.9446 179.3763 154.2285 -22.07514 22.10536
```

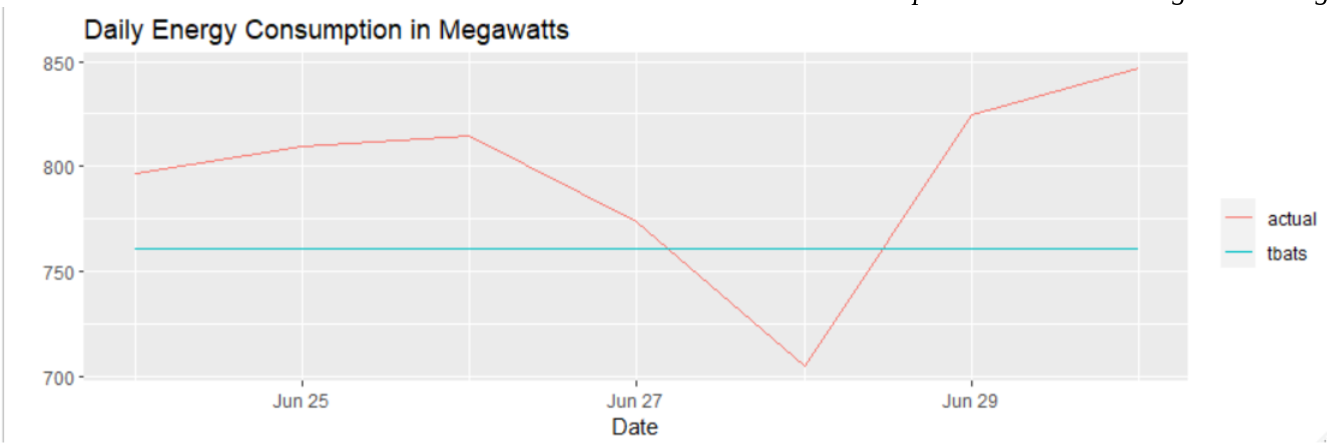
```r
# Create a time series object
library(caret)
library(data.table)
library(zoo)
train <- head(agg_data,1637)
test <- tail(agg_data,7)
# Create LOCF predictions
lo_cf <- tail(train$`Consumption (MWH)`, 1)
lo_cf_preds <- rep(lo_cf, nrow(test))
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7)), colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = lo_cf_preds,   colour = "tbats "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

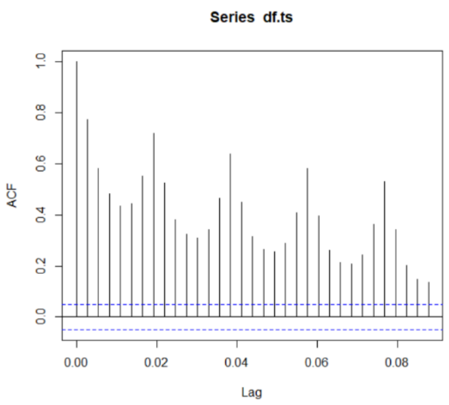



```r
# Create MA predictions (using a 7-day window)
ma_preds <- rollapply(train$`Consumption (MWH)`,
                      7, mean, align = "right", partial = TRUE)
ma_preds <- rep(tail(ma_preds, 1), nrow(test))
MAPE <- function(actual, predicted) {
  mean(abs((actual - predicted) / actual)) * 100
}
# Compute RMSE and MAPE for each model
library(metrics)
models <- c("LOCF", "MA")
rmse <- c(RMSE(test$`Consumption (MWH)`, lo_cf_preds),
          RMSE(test$`Consumption (MWH)`, ma_preds))
mape <- c(MAPE(test$`Consumption (MWH)`, lo_cf_preds),
          MAPE(test$`Consumption (MWH)`, ma_preds))
results <- data.frame(Model = models, RMSE = rmse, MAPE = mape)
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = ma_preds,   colour = "tbats "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Hourly Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

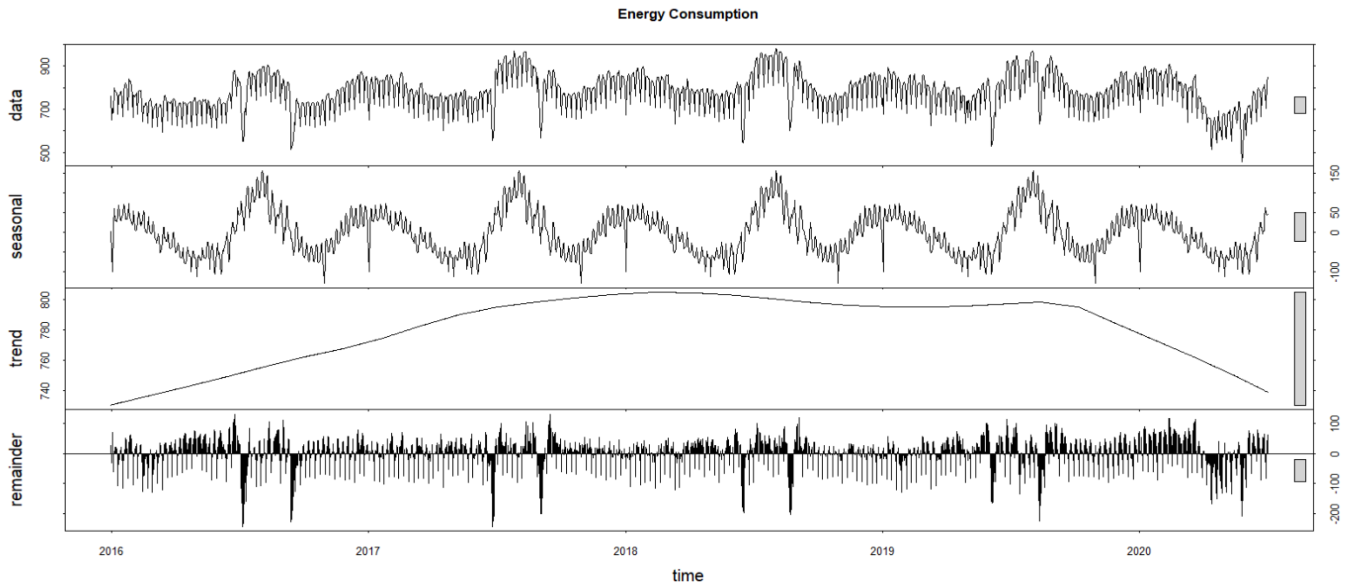
```
> results
  Model     RMSE     MAPE
1  LOCF 43.26488 4.539512
2    MA 55.01327 6.350938
>
```

This R code performs time series forecasting and evaluation on a dataset called "agg_data". It first converts the "Date" column to a date object and creates training and testing sets. Two forecasting models are fitted and their predictions are plotted against the actual values using ggplot. The first model is a LOCF (last observation carried forward) model, and the second model uses a 7-day moving average. The RMSE and MAPE accuracy metrics are computed for each model and the results are plotted using ggplot.

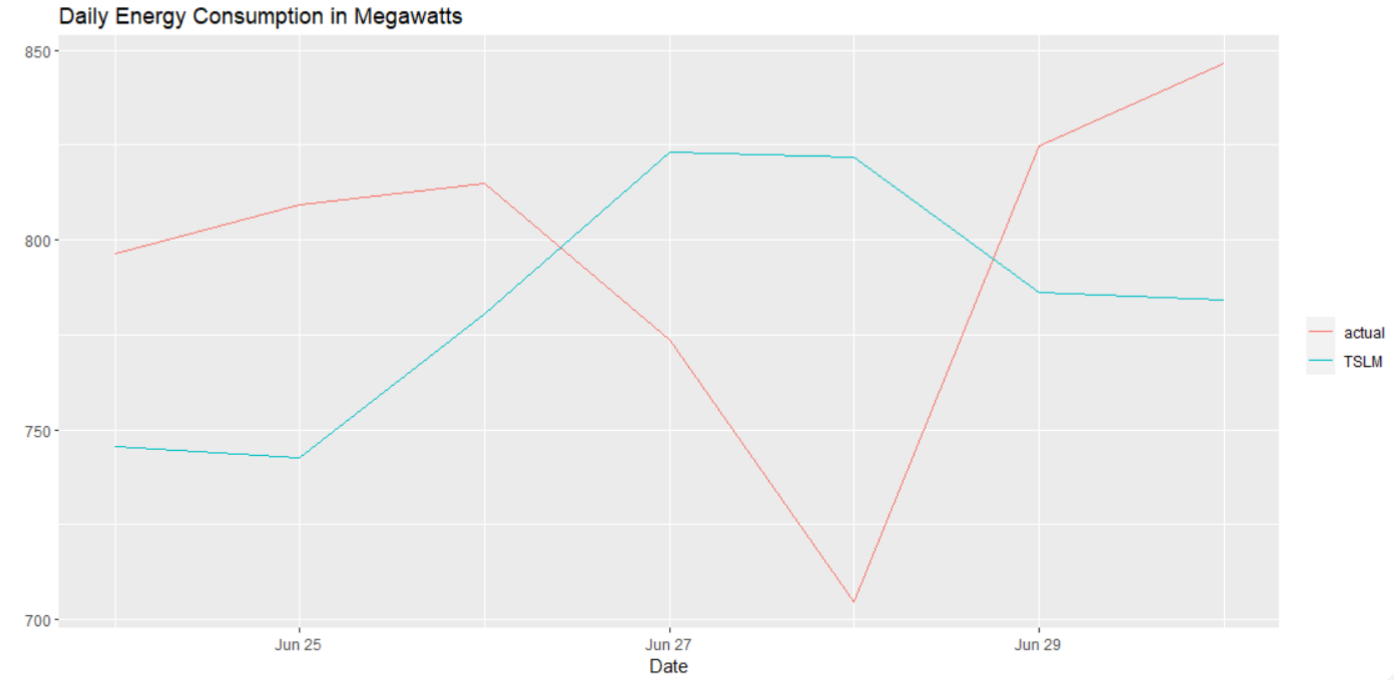





```r
df_matrix <- as.matrix(agg_data[, -1])
rownames(df_matrix) <- agg_data[, 1]
par(mfrow=c(1,1))

# convert the matrix to a time series object
df.ts <- ts(df_matrix, start = c(2015, 365), frequency = 365)
acf(df.ts) #strong autocorrelation present
plot(df.ts, main = "Energy Consumption")

# select a single column from df.ts
df_univariate <- df.ts[,1]

# decompose the time series using stl()
df_stl <- stl(df_univariate, s.window = "periodic")
#trend, season and level present
plot((df_stl), main = "Energy Consumption")

```

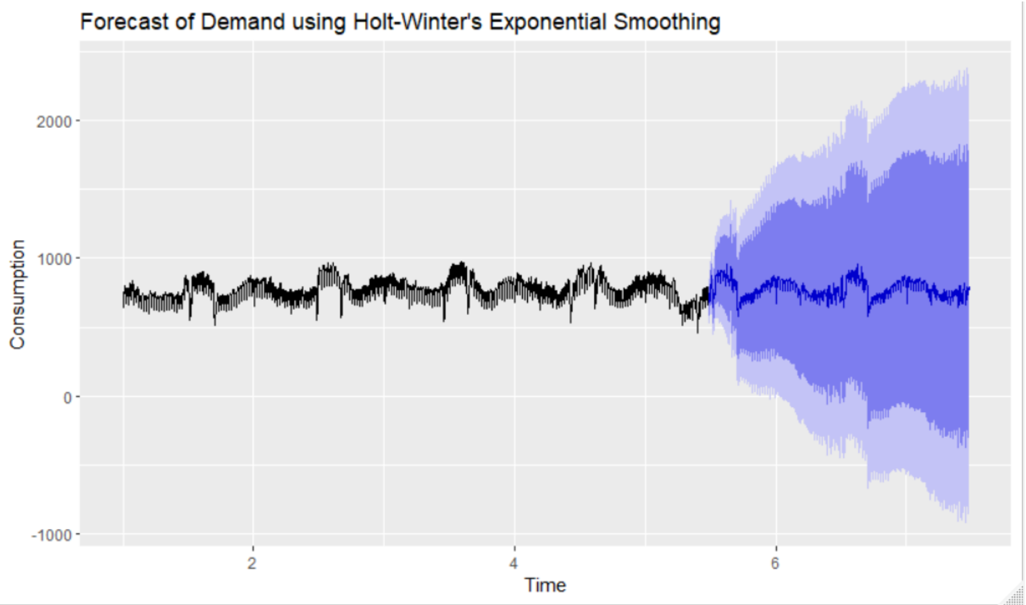


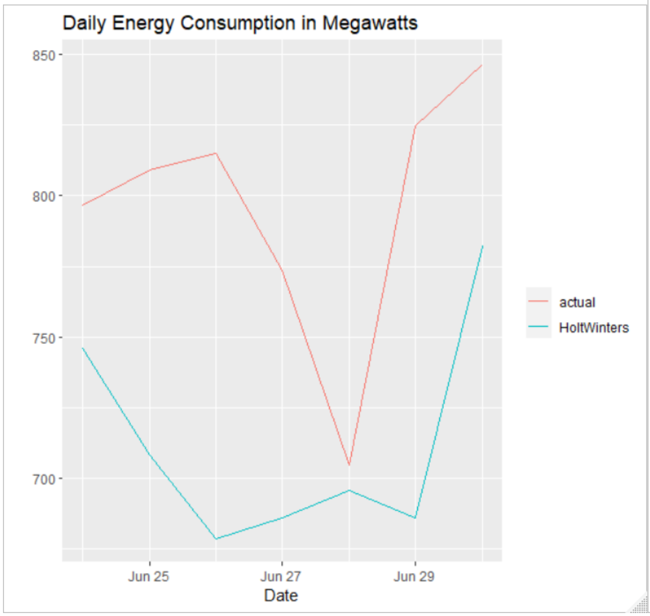

```r
#partition the data
df.valid.ts = ts(tail(agg_data$`Consumption (MWH)`,7), frequency = 365)
df.train.ts = ts(head(agg_data$`Consumption (MWH)`,1637), frequency = 365)
#Regression based model
train.lm.season = tslm(df.train.ts~season)
train.lm.trend.season = tslm(df.train.ts ~ trend + I(trend ^2) + season)
train.trend.season.pred = forecast(train.lm.trend.season, h=7, level = 0)

ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = train.trend.season.pred$mean,   colour = "TSLM "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
accuracy(train.lm.trend.season)
```

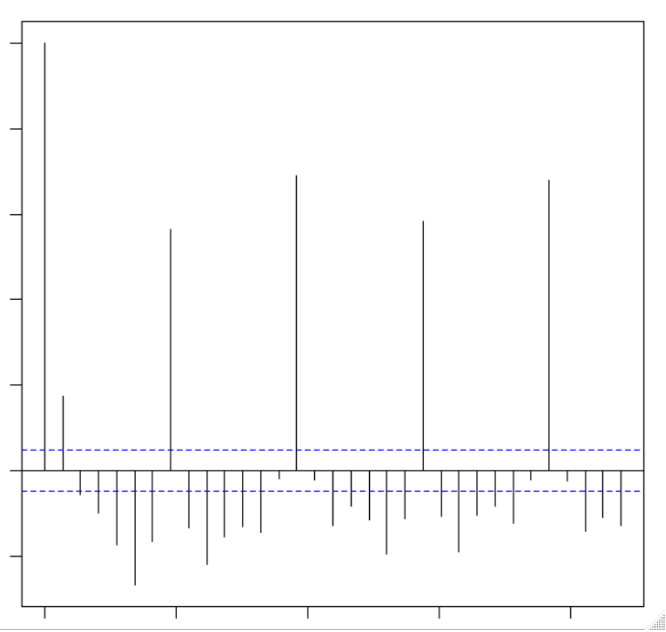



```
> accuracy(train.lm.trend.season)
                        ME     RMSE      MAE       MPE     MAPE     MASE      ACF1
Training set -1.244999e-15 56.05387 41.33467 -0.595754 5.617075 0.664774 0.5835203
> |
```

```
#holt winter's
df.data.hw = HoltWinters(df.train.ts, beta=FALSE)
autoplot(forecast(df.data.hw)) + xlab("Time") + ylab("Consumption") +
  ggtitle("Forecast of Demand using Holt-Winter's Exponential Smoothing")

df.data.hw.forecast = forecast(df.data.hw,7)

acf(df.data.hw.forecast$residuals, na.action = na.pass)

ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = df.data.hw.forecast$mean,   colour = "HoltWinters "))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
p = as.data.frame(df.data.hw.forecast$mean)
o=as.data.frame(tail(agg_data$`Consumption (MWH)`,7))
mae(p$x,o$`tail(agg_data$\`Consumption (MWH)\`, 7)`)
rmse(p$x,o$`tail(agg_data$\`Consumption (MWH)\`, 7)`)
```







```
> p = as.data.frame(df.data.hw.forecast$mean)
> o=as.data.frame(tail(agg_data$`Consumption (MWH)`,7))
> mae(p$x,o$`tail(agg_data$\`Consumption (MWH)\`, 7)`)
[1] 84.02287
> rmse(p$x,o$`tail(agg_data$\`Consumption (MWH)\`, 7)`)
[1] 94.59442
```

```
#ARIMA model
df.data.res.arima <- Arima(train.lm.trend.season$residuals, order = c(1,1,2))
df.data.res.arima.pred <- forecast(df.data.res.arima, h = 7)
summary(df.data.res.arima)
tsdiag(df.data.res.arima)
accuracy(df.data.res.arima)
df.data.arima <- auto.arima(df.train.ts)
summary(df.data.arima)
tsdiag(df.data.arima)
```

The code performs time series analysis on a dataset named "agg_data" which has daily energy consumption data from 2015 to 2019.

The first step converts the data frame into a matrix and assigns the row names to the first column of the original data frame. Then, it converts the matrix into a time series object with a daily frequency.

The next step plots the autocorrelation function (ACF) of the time series object and the energy consumption plot.

After that, it selects a single column from the time series object and decomposes it into trend, season, and level using the STL function. The plot of the decomposed time series is also displayed.

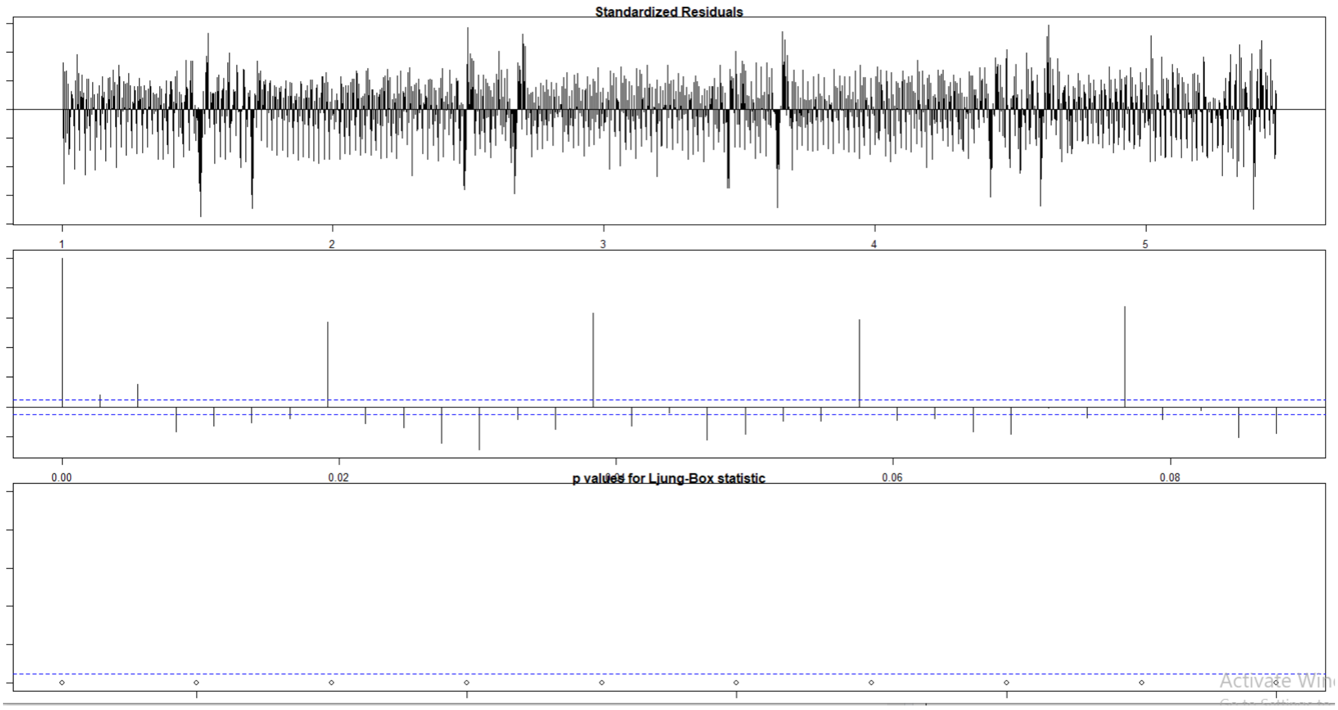
The data is then partitioned into training and validation sets. Two models are built for forecasting: a regression-based model using the tslm function, and a Holt-Winters model using the HoltWinters function.

The regression-based model is fitted using the training set, and the predicted values are plotted along with the actual values for the last seven days. The accuracy of the model is calculated using the accuracy function.

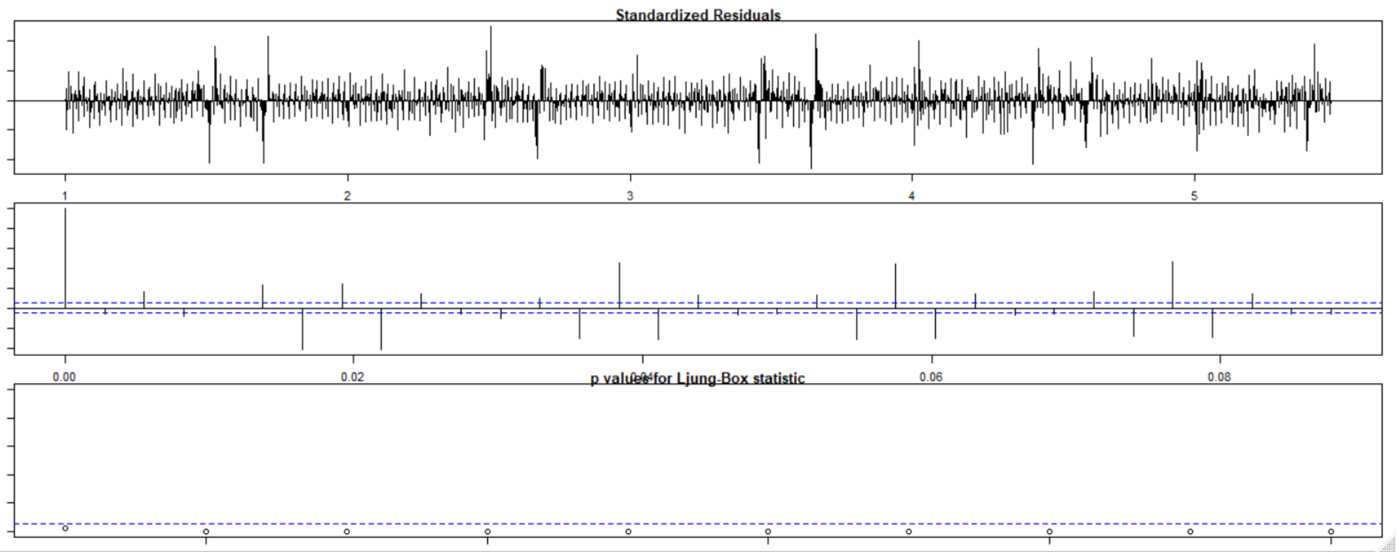
The Holt-Winters model is also fitted using the training set, and the predicted values for the next seven days are plotted along with the actual values for the last seven days. The mean absolute error (MAE) and root mean squared error (RMSE) are calculated for the Holt-Winters model using the forecasted values and actual values for the last seven days.

Finally, the plots are displayed using ggplot2 library.

```
> summary(df.data.res.arima)
Series: train.lm.trend.season$residuals
ARIMA(1,1,2)

Coefficients:
          ar1     ma1      ma2
      -0.4265  0.1145  -0.7404
s.e.   0.0329  0.0330   0.0272

sigma^2 = 2082:  log likelihood = -8570.93
AIC=17149.86   AICc=17149.89   BIC=17171.46

Training set error measures:
                      ME     RMSE      MAE      MPE     MAPE      MASE       ACF1
Training set -0.05819715 45.56875 37.06326 52.68971 370.8809 0.6330123 0.08053304
```



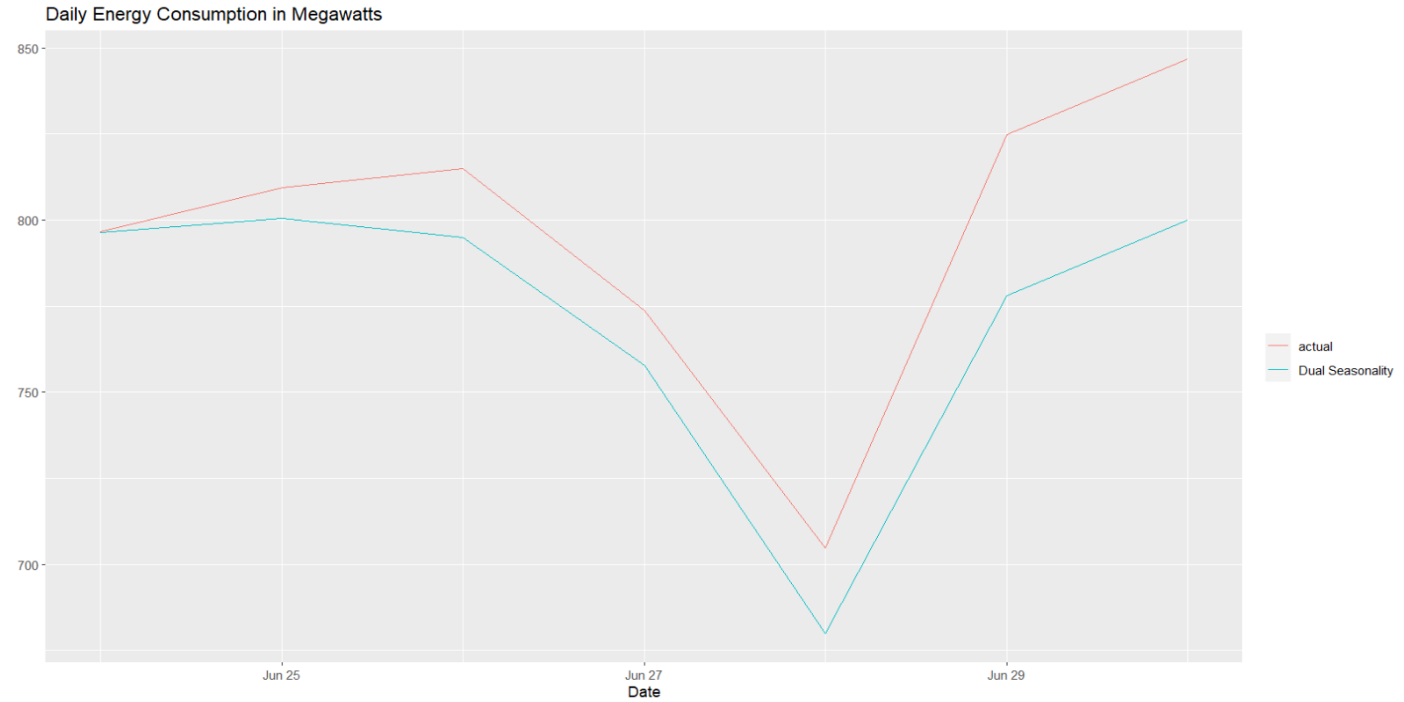

```
> summary(df.data.arima)
Series: df.train.ts
ARIMA(5,1,2)

Coefficients:
         ar1      ar2      ar3      ar4      ar5      ma1     ma2
      0.0879  -0.6857  -0.2788  -0.2765  -0.5349  -0.4197  0.6331
s.e.  0.0370   0.0221   0.0268   0.0209   0.0260   0.0478  0.0197

sigma^2 = 1721:  log likelihood = -8414.29
AIC=16844.58   AICc=16844.67   BIC=16887.78

Training set error measures:
                     ME     RMSE      MAE        MPE     MAPE      MASE        ACF1
Training set 0.02261221 41.38865 29.68606 -0.2523777 3.982399 0.4774328 -0.05679296
```




```
#SARIMA
df.data.sarima=arima(df.train.ts,order=c(1,1,2),seasonal=list(order=c(1,2,2),period=7))
df.data.res.sarima.pred <- forecast(df.data.sarima, h = 7)
accuracy(df.data.sarima)
m2=arima(df.train.ts,order=c(1,1,2)
         ,seasonal=list(order=c(1,2,2),period=1))
m2
par(mar=c(1,1,1,1))
```

```
> accuracy(df.data.sarima)
                      ME     RMSE      MAE         MPE     MAPE      MASE          ACF1
Training set -0.02738129 24.15391 13.83239 -0.03328447 1.872356 0.3471774 -0.0004173572
> m2=arima(df.train.ts,order=c(1,1,2)
+          ,seasonal=list(order=c(1,2,2),period=1))
> m2

Call:
arima(x = df.train.ts, order = c(1, 1, 2), seasonal = list(order = c(1, 2, 2),
    period = 1))

Coefficients:
          ar1      ma1      ma2     sar1     sma1     sma2
      -0.8281  -0.0888  -0.9112  -0.8281  -0.0888  -0.9112
s.e.   0.0300   0.0216   0.0216   0.0300   0.0216   0.0216

sigma^2 estimated as 2887:  log likelihood = -8841.94,  aic = 17697.88
```

```
#Duol Sesonlity
fit <- tbats(df.train.ts, seasonal.periods = c(365, 7))
# Make forecasts
h <- length(df.valid.ts)
fc <- forecast(fit, h = h)
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = fc$mean,    colour = "tbats"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  ))
```




```
> accuracy(fit)
                    ME     RMSE      MAE         MPE     MAPE      MASE         ACF1
Training set 0.1925264 22.75495 12.66482 -0.06817274 1.717797 0.3178727 -0.001785213
```

```r
#auto arima
arima_model <- auto.arima(df.train.ts)
summary(arima_model)
forecast_values <- forecast(arima_model, h = length(df.valid.ts))
# Combine the actual and forecasted values into a single data frame
plot_data <- data.frame(actual = test_data, predicted = forecast_values$mean)

# Create a line plot
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = forecast_values$mean,   colour = "auto arima"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

The provided R code contains several sections that use different time series models to forecast energy consumption data. The code begins with an ARIMA model that fits the residuals of a trend and seasonal model (train.lm.trend.season$residuals). The Arima() function is used to fit a model with ARIMA(1,1,2) order and the forecast() function is used to generate a 7-day forecast. The summary(), tsdiag(), and accuracy() functions are used to check the model's performance and diagnostic plots.

Next, a seasonal ARIMA model (SARIMA) is fitted to the entire training dataset using the arima() function. The model is specified with an ARIMA(1,1,2) order and a seasonal component of order (1,2,2) and a period of 7. The forecast() and accuracy() functions are used to generate and evaluate the 7-day forecast.

After that, the code uses the tbats() function to fit a model with dual seasonality (365 days and 7 days) to the training dataset. The forecast() function is then used to generate a 7-day forecast, and the ggplot() function is used to plot the actual and forecasted values.

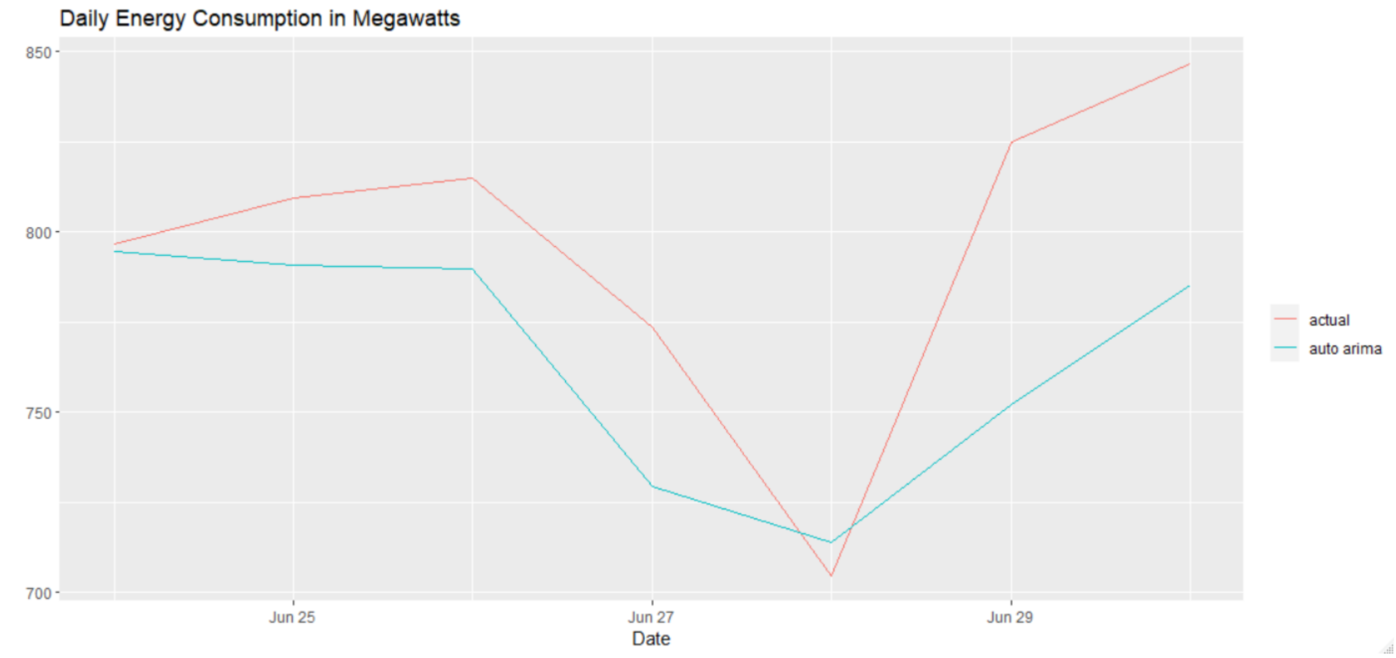
Next, the code uses the auto.arima() function to automatically fit an ARIMA model to the training dataset, without specifying the model's order. The summary() function is used to display the model's summary, and the forecast() function generates a 7-day forecast.

Finally, the code uses the ggplot() function to plot the actual and forecasted values for the auto ARIMA model.

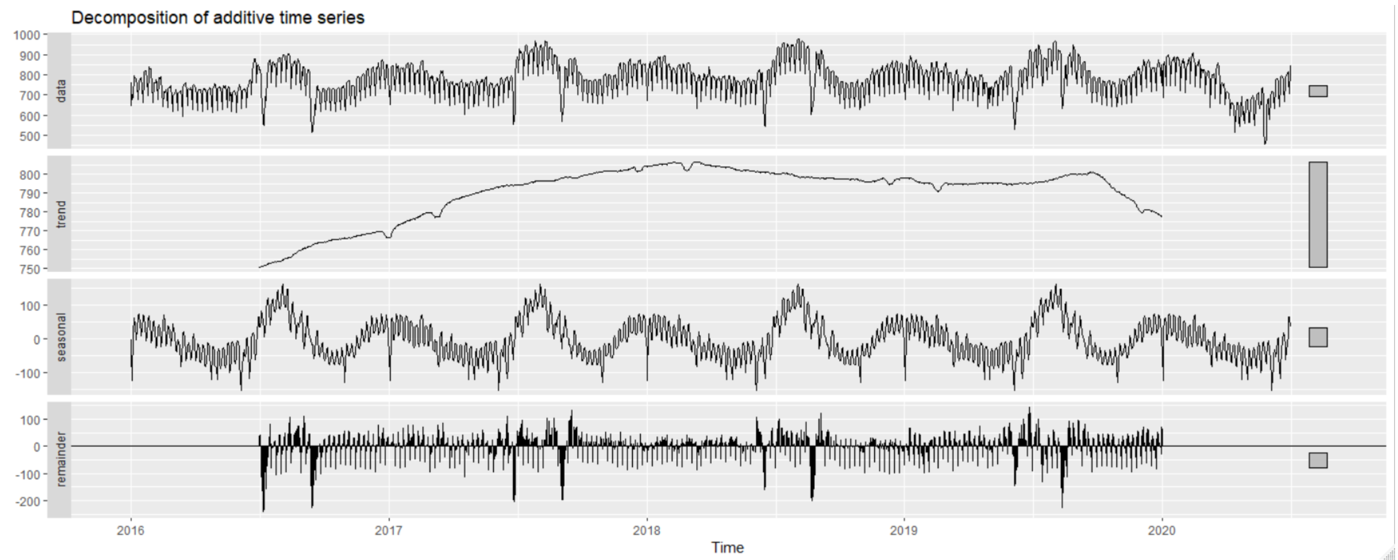
Overall, the code demonstrates how to use different time series models to forecast energy consumption data and how to evaluate their performance using diagnostic plots and accuracy measures. It also shows how to plot the actual and forecasted values using the ggplot() function.

```
> summary(arima_model)
Series: df.train.ts
ARIMA(5,1,2)

Coefficients:
         ar1      ar2      ar3      ar4      ar5      ma1     ma2
      0.0879  -0.6857  -0.2788  -0.2765  -0.5349  -0.4197  0.6331
s.e.  0.0370   0.0221   0.0268   0.0209   0.0260   0.0478  0.0197

sigma^2 = 1721:  log likelihood = -8414.29
AIC=16844.58   AICc=16844.67   BIC=16887.78

Training set error measures:
                     ME     RMSE      MAE        MPE     MAPE      MASE        ACF1
Training set 0.02261221 41.38865 29.68606 -0.2523777 3.982399 0.4774328 -0.05679296
```

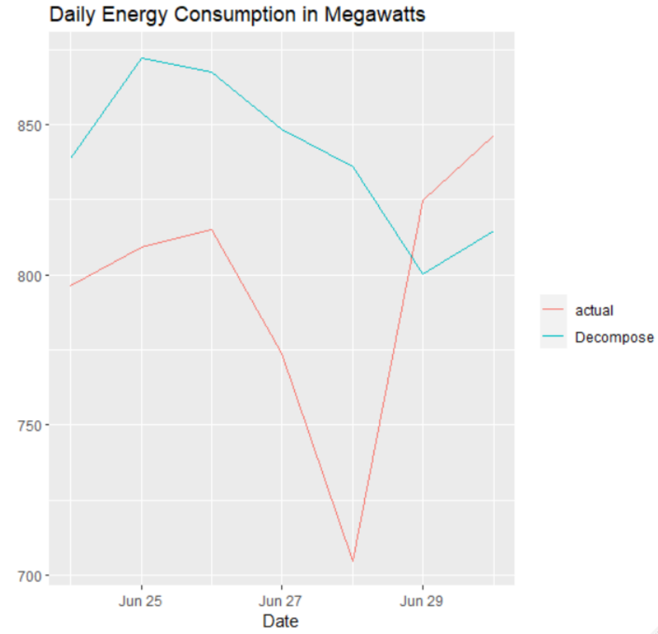
```
> plot_data
    actual predicted
1 796.4970  794.5751
2 809.2957  790.8831
3 814.9564  789.7461
4 773.7083  729.4855
5 704.7139  714.0783
6 824.8187  752.3205
7 846.6656  785.3386
```



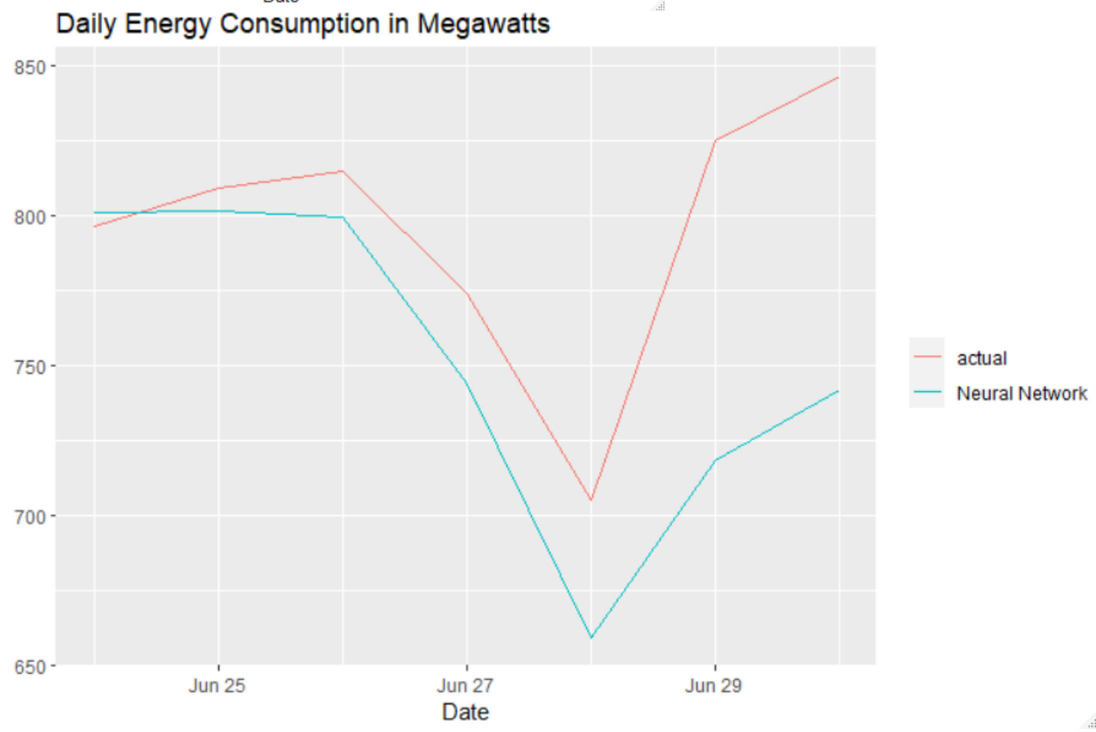

```r
# Load required library
library(ggplot2)
# Convert data frame to time series object
ts_data <- ts(agg_data$`Consumption (MWH)`, start = c(2015, 365), frequency = 365)
# Decompose the time series
decomp <- decompose(ts_data)
# Plot the decomposed components
autoplot(decomp)
```

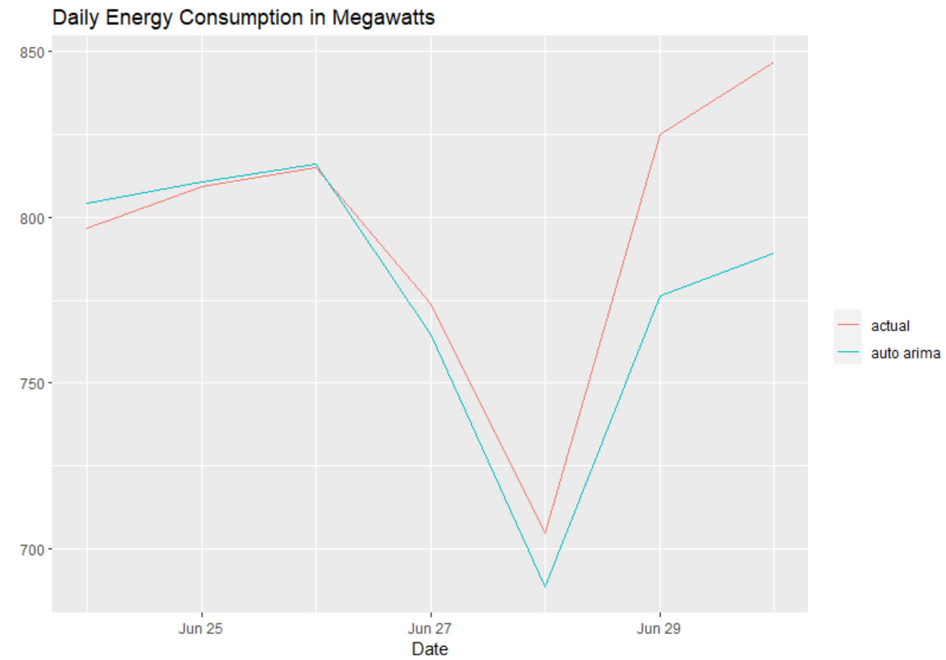



```r
train <- df.train.ts
test <- df.valid.ts
# Check the length of the training and testing sets
length(train)
length(test)
# Load required library
library(forecast)
# Generate forecasts for the testing set using the training set components
fc <- predict(decomp$x, h = 7)
# Create a line plot
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = fc$mean,    colour = "Decompose"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```

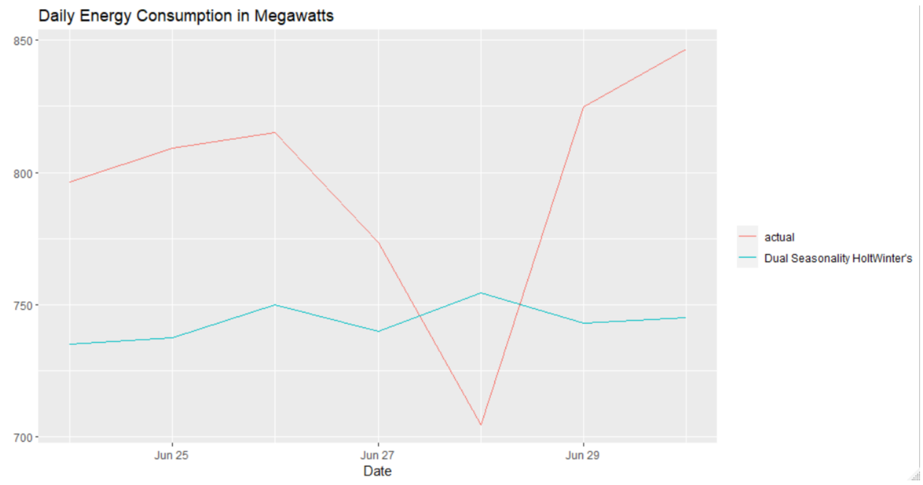



```r
#Neural Network
train_data <- train
test_data <- test
nn_model <- nnetar(train_data, size = 20)
nn_pred <- forecast(nn_model, h = length(test_data))
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = nn_pred$mean,   colour = "Neural Network"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```





```r
# Split the data into training and testing sets
train_data <- df.train.ts
test_data <- df.valid.ts
# Fit a dual seasonality model to the training data
model_ds <- dshw(train_data, biasadj = TRUE, period1 = 30, period2 = 60)
summary(model_ds)
# Forecast the test data
predictions <- forecast(model_ds, 7)
x1=as.data.frame(predictions$fitted)
df_data <- as.data.frame(train_data)
names(df_data) <- "x"
df3 <- merge(df_data, x1, sort = FALSE, all=TRUE )
ggplot() +
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = (agg_data$`Consumption (MWH)` %>% tail(7))
                , colour = "actual"))+
  geom_line(aes(x = (agg_data$Date %>% tail(7)),
                y = head(predictions$mean, 7),   colour = "Neural Network"))+
  scale_y_continuous(labels = comma)+
  labs(
    title = "Daily Energy Consumption in Megawatts",
    x = "Date",
    y = "",
    colour = ""
  )
```




```
> accuracy(model_ds)
                   ME     RMSE      MAE        MPE     MAPE      MASE      ACF1
Training set 5.510798 66.72409 50.70614 0.06211221 6.770746 0.4974401 0.4904791
>
```

This R code is used to analyze and forecast daily energy consumption in Megawatts. The code is divided into four parts, each using a different forecasting method.

The first part loads the required library, ggplot2, and converts the data frame into a time series object using the ts() function. It then decomposes the time series using the decompose() function and plots the decomposed components using the autoplot() function. It then generates forecasts for the testing set using the training set components and creates a line plot using the ggplot() function.

The second part trains and tests a neural network model using the nnetar() and forecast() functions. It then creates a line plot to visualize the actual and predicted energy consumption using the ggplot() function.

The third and fourth parts use the dual seasonality Holt-Winters (DSHW) method to forecast the energy consumption. The dshw() function is used to fit the DSHW model to the training data, and the forecast() function is used to generate predictions for the testing data. A line plot is created to visualize the actual and predicted energy consumption for both the DSHW models using the ggplot() function.

The accuracy of the DSHW models is also calculated using the accuracy() function, and the summary() function is used to provide a summary of the DSHW model's results.

Overall, this R code provides an example of how to use different forecasting methods to analyze and forecast daily energy consumption in Megawatts.