TSF Project

Logan Eades, Emmanuel Epau, and Thomas Zwiller

2025-02-09

1. Loading in the London Weather Data

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

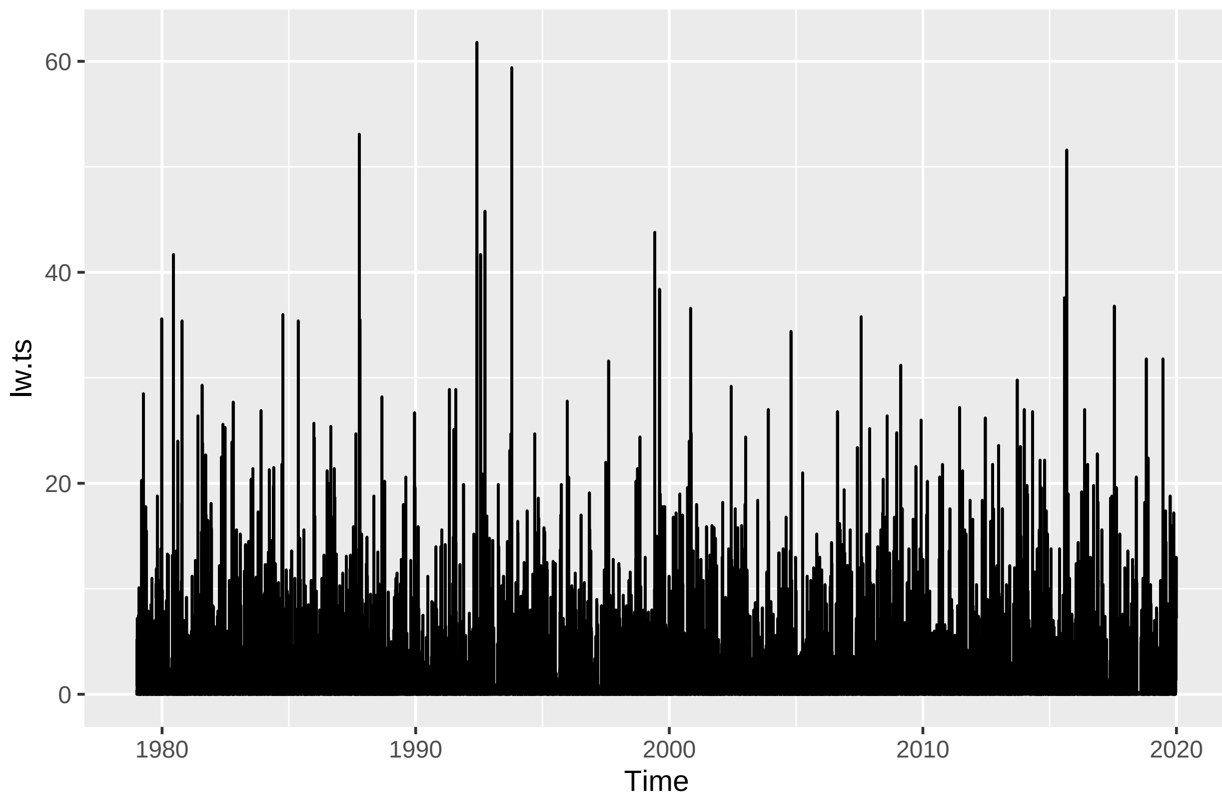
```
library(readxl)
library(ggplot2)
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.4.1
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
# Read the London weather CSV file into a dataframe
lw.df <- read.csv("/Users/TomTheIntern/Downloads/london_weather.csv")
lw.df$date <- as.Date(as.character(lw.df$date), format = "%Y%m%d")

# Make precipitation into a daily time series object starting in 1979 and ending in 2019
lw.ts <- ts(lw.df$precipitation, start = c(1979, 1), end = c(2019, 365), freq = 365)

# Plot the time series
autoplot(lw.ts)
```



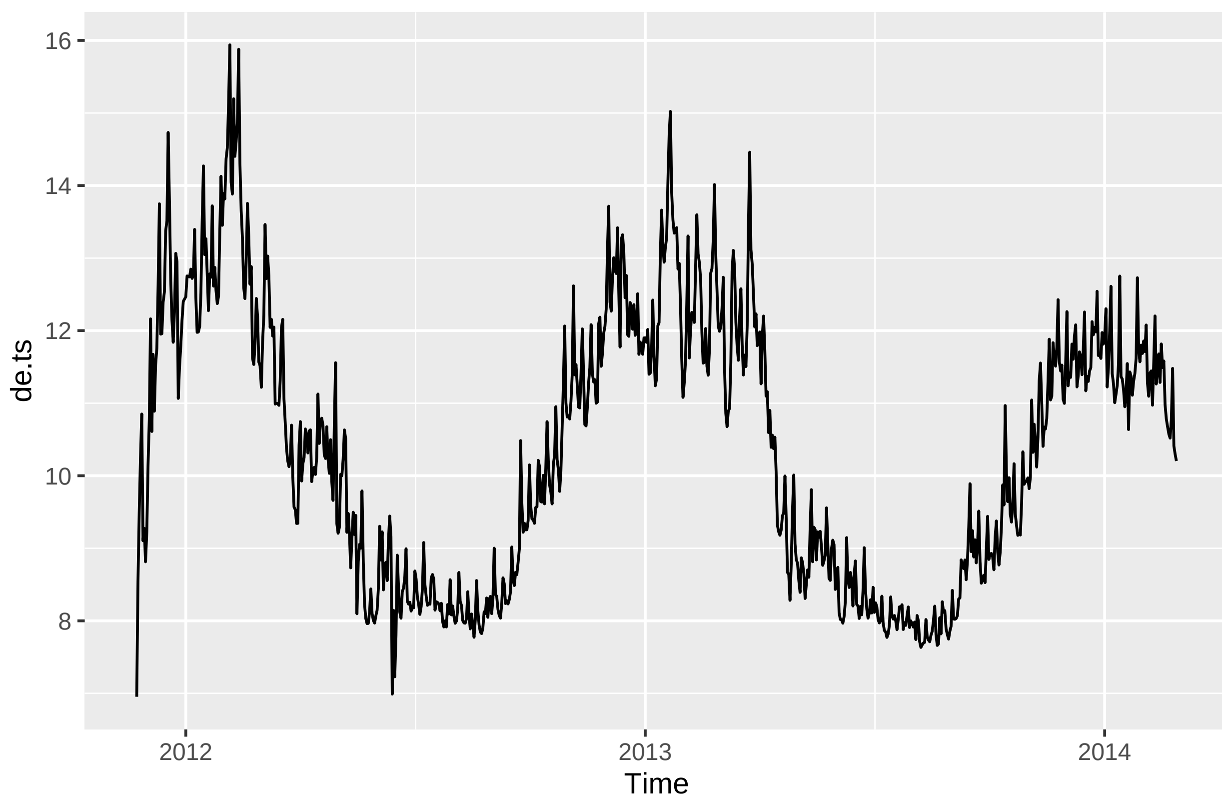
Reading in the London Energy data

```r
library(forecast)
library(readxl)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(readr)

# Read the London energy data CSV file into a dataframe
le.df <- read.csv("/Users/TomTheIntern/Downloads/london_energy.csv")

# Convert date column to date format
le.df$Date <- as.Date(le.df$Date, format="%Y-%m-%d")
```

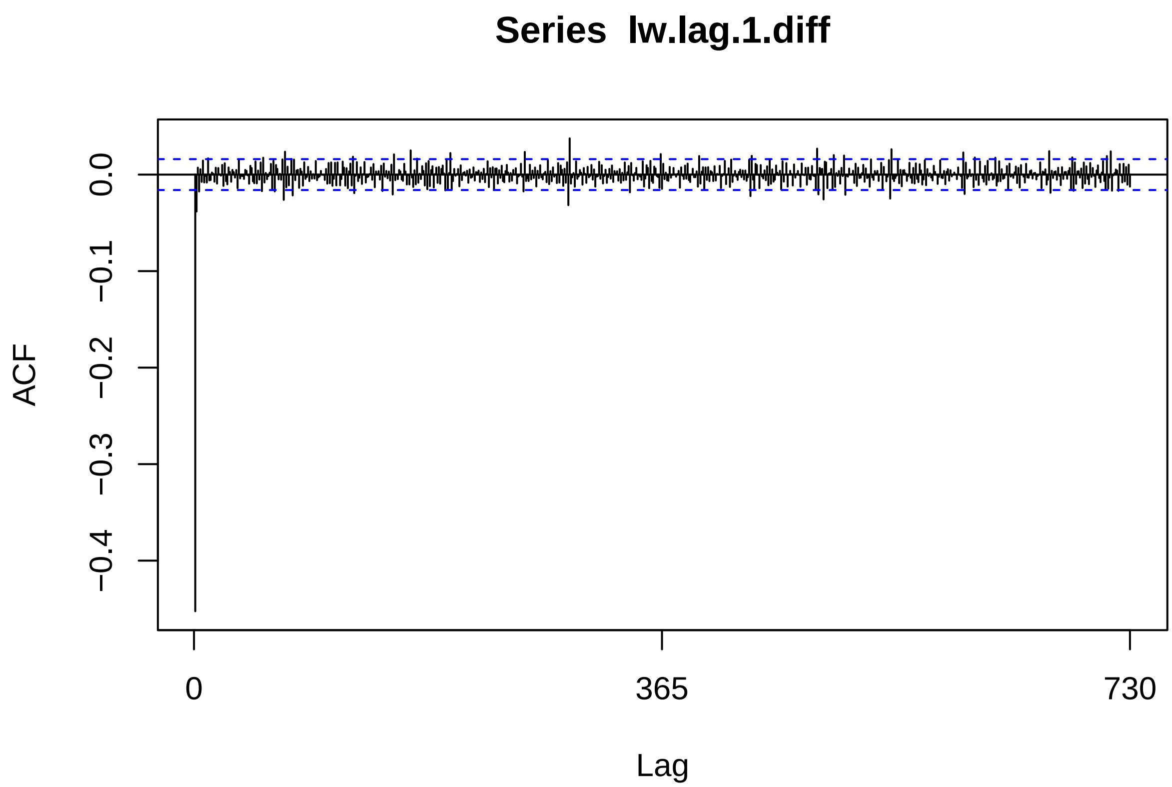
```r
# Group the energy data by date and get the average kWh consumption for each day
daily_energy <- le.df %>%
  group_by(Date) %>%
  summarise(Avg_kWh = mean(KWH, na.rm = TRUE))

# Convert daily energy into a time series object from 2011 to 2014
de.ts <- ts(daily_energy$Avg_kWh, start = c(2011, 327), end = c(2014, 58), freq = 365)

# Plot the daily energy usage time series
autoplot(de.ts)
```



2. Test lw data to see if it is a Random Walk

Note: If you see an error with with Arima model, use `method  = "ML"`, which fits maximum likelihood estimation which is a better model than the default CSS(conditional sum-of-squares).

```r
# Option 1: Differencing and Acf plot
lw.lag.1.diff <- diff(lw.ts, lag = 1)

# Plot the ACF of the differenced series to check for randomness
Acf(lw.lag.1.diff)
```




```r
# Option 2: Building an AR(1) model and testing based on coefficient.
# Fit an Arima model of order 1 (AR(1)) to the time series
lw.ar.1 <- Arima(lw.ts, order = c(1, 0, 0))
summary(lw.ar.1)
```

```
## Series: lw.ts
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##          ar1    mean
##       0.1689  1.6659
## s.e.  0.0081  0.0362
##
## sigma^2 = 13.57:  log likelihood = -40746.12
## AIC=81498.24   AICc=81498.24   BIC=81521.08
##
## Training set error measures:
##                        ME     RMSE      MAE  MPE MAPE      MASE        ACF1
## Training set 6.228843e-05 3.683346 2.170001 -Inf  Inf 0.7997338 -0.01063643
```

We can conclude that our coefficient is 0.1689 is signifcantly different from 0.

Null Hypothesis: beta = 1 (i.e., random walk)

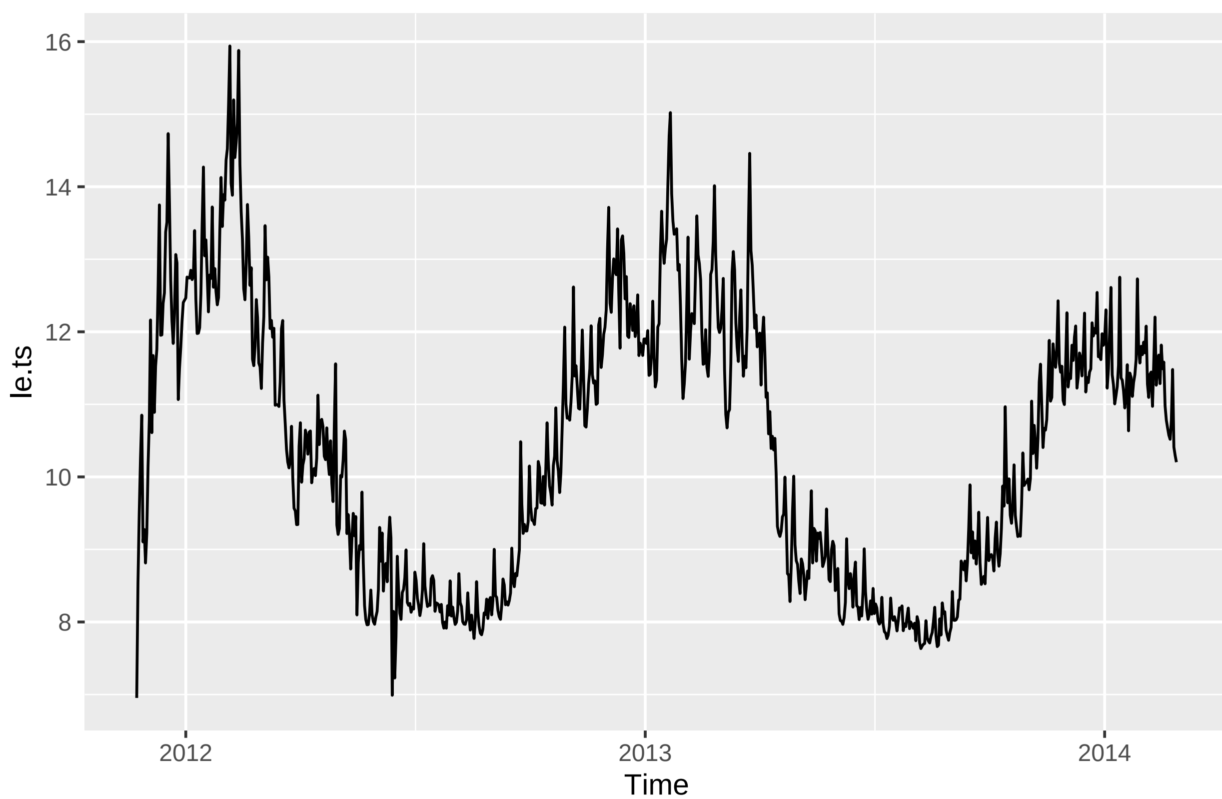
Alternative Hypothesis: beta not equal to 1 (i.e., not random walk)

To be specific, our t-stat = (ceofficient - 1)/s.e = (0.1689-1)/(0.0081) = -102.6049.

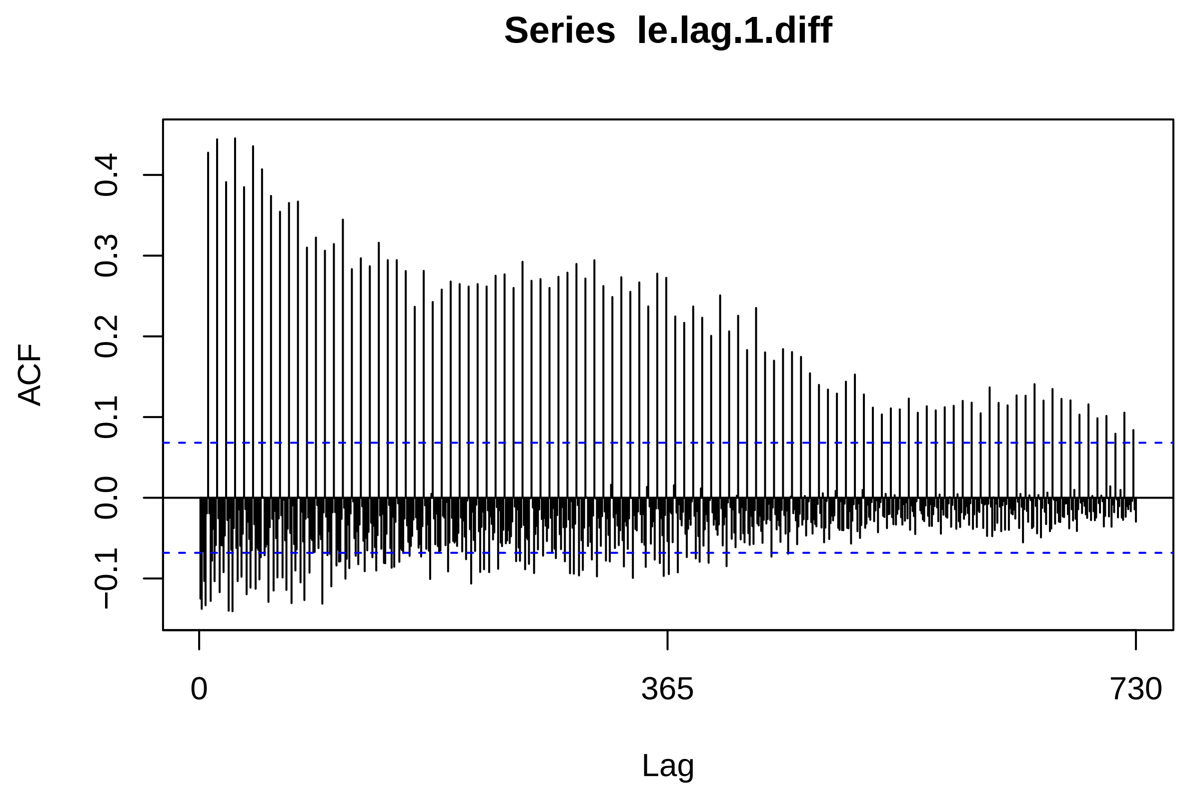
If we consider alpha = 0.05, then our critical values are -2. Since -102.6049. < -2, we can reject the null hypothesis and say **beta is not equal to 1**. Therefore ridership data is not is **not a random walk**.

Checking to make sure the London Energy data is not a random walk.

```
library(forecast)
library(readxl)
library(ggplot2)

# Read the London weather CSV file into a dataframe
le.df <- read.csv("/Users/TomTheIntern/Downloads/london_energy.csv")

# Make precipitation into a daily time series object starting in 1979 and ending in 2019
le.ts <- ts(daily_energy$Avg_kWh, start = c(2011, 327), end = c(2014, 58), freq = 365)

# Plot the time series
autoplot(le.ts)
```




```
# Option 1: Differencing and Acf plot
le.lag.1.diff <- diff(le.ts, lag = 1)

# Plot the ACF of the differenced series to check for randomness
Acf(le.lag.1.diff)
```




```r
# Option 2: Building an AR(1) model and testing based on coefficient.
# Fit an Arima model of order 1 (AR(1)) to the time series
le.ar.1 <- Arima(le.ts, order = c(1, 0, 0))
summary(le.ar.1)
```

```
## Series: le.ts
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##          ar1     mean
##       0.9622  10.2790
## s.e.  0.0094   0.4565
##
## sigma^2 = 0.2613:  log likelihood = -618.88
## AIC=1243.75   AICc=1243.78   BIC=1257.9
##
## Training set error measures:
##                       ME      RMSE       MAE        MPE     MAPE      MASE
## Training set 0.006522431 0.5105978 0.3782152 -0.1651644 3.578932 0.4140908
##                    ACF1
## Training set -0.1115376
```

We can conclude that our coefficient is 0.9622 is signifcantly different from 0.

Null Hypothesis: beta = 1 (i.e., random walk)

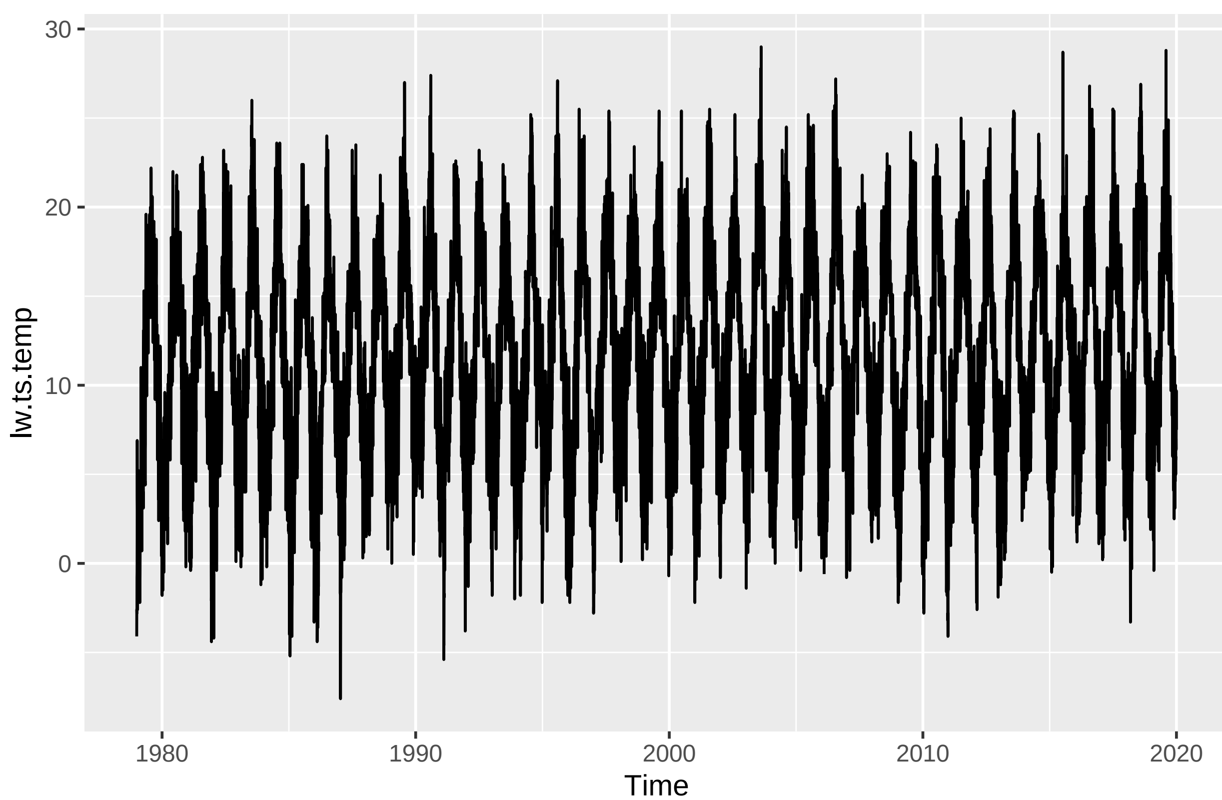
Alternative Hypothesis: beta not equal to 1 (i.e., not random walk)

To be specific, our t-stat = (ceofficient - 1)/s.e = (0.9622-1)/(0.0094) = -4.021277.

If we consider alpha = 0.05, then our critical values are -2. Since -4.021277 < -2, we can reject the null hypothesis and say **beta is not equal to 1**. Therefore ridership data is not is **not a random walk**.

Plotting the London Weather mean temperature

```
# Load libraries for plotting and forecasting
library(ggplot2)
library(forecast)

# Convert mean_temp to a daily time series
lw.ts.temp <- ts(lw.df$mean_temp,  start = c(1979, 1), end = c(2019, 365), freq = 365)

# Plot time series
autoplot(lw.ts.temp)
```



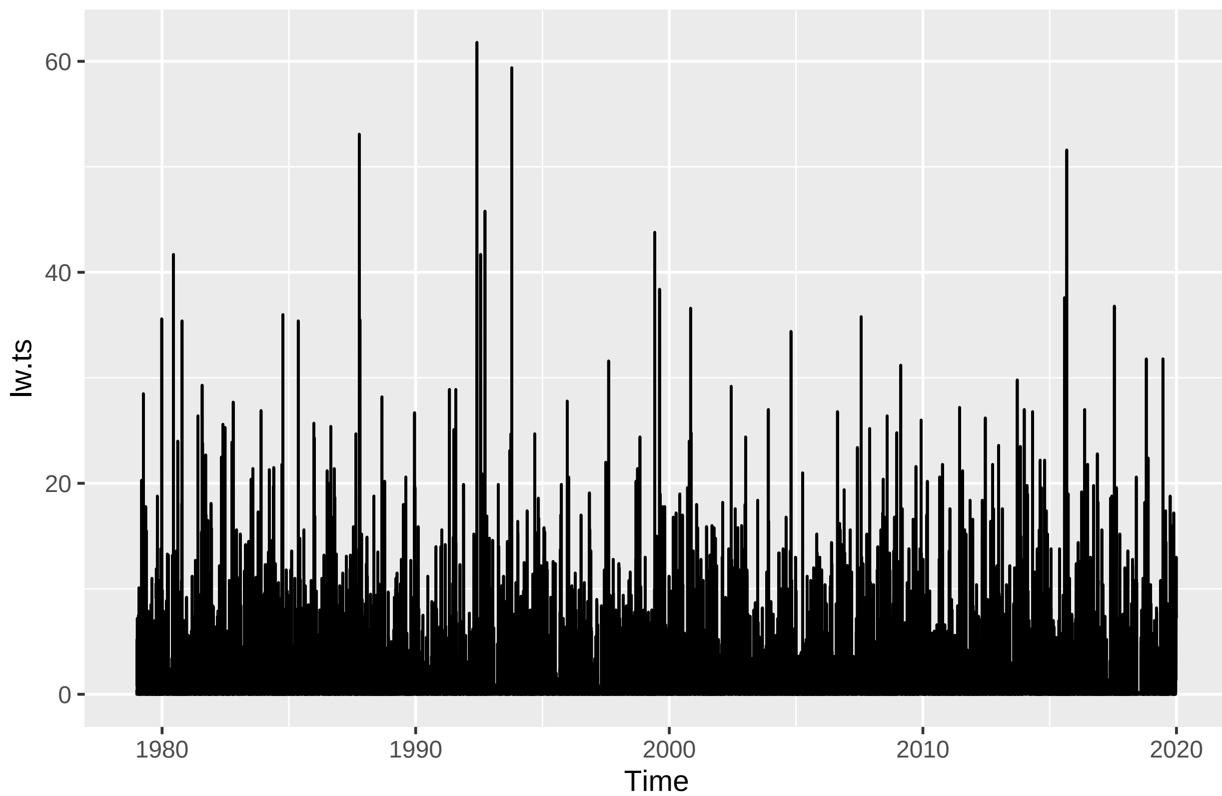

```
library(dplyr)
library(lubridate)

# Convert the date column in the weather data to date format
lw.df$Date <- ymd(lw.df$date)

# Daily energy dates messed up so reconvert to be in proper date format
daily_energy$Date <- ymd(daily_energy$Date)
```

```r
# Merge the weather and energy datasets by the Date column using an inner join
merged_data <- merge(lw.df, daily_energy, by = "Date", all.y = TRUE)

# Remove the extra date column from the merged dataset
merged_data <- merged_data[, !names(merged_data) %in% "date"]

# View first few rows
head(merged_data)
```

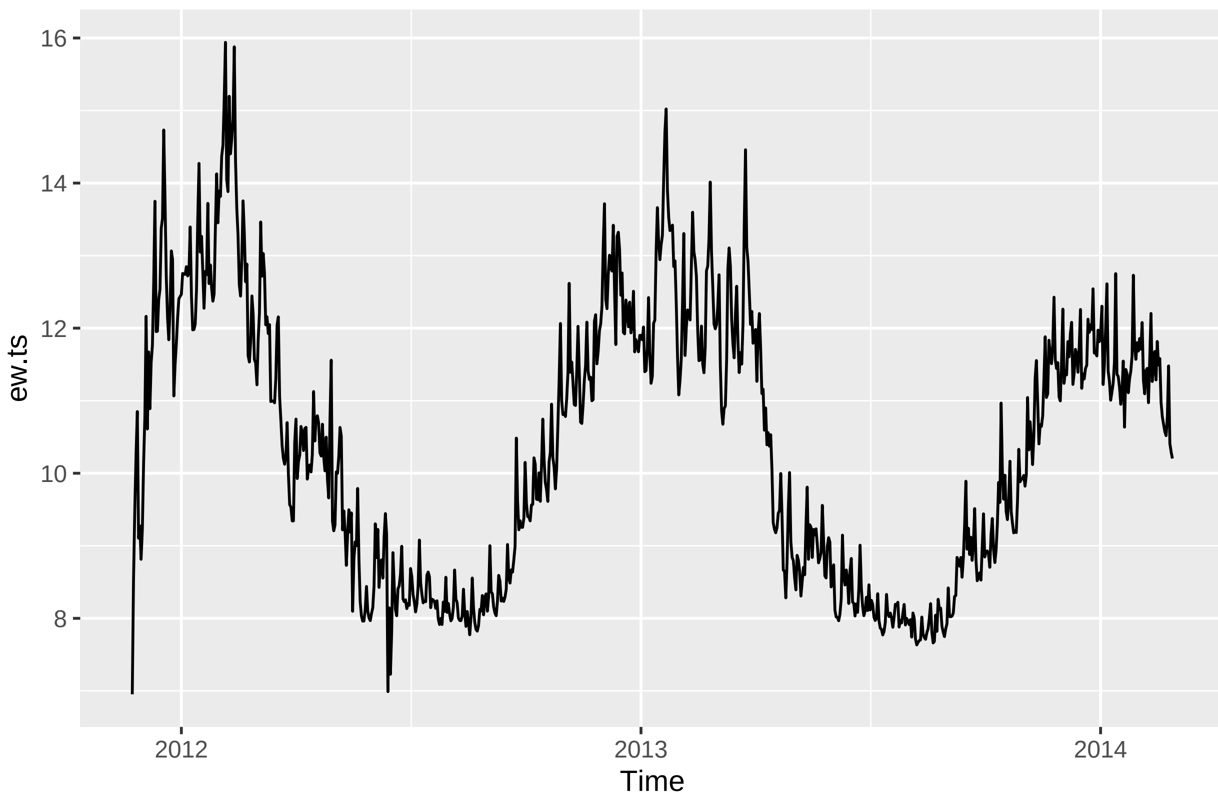
```
##         Date cloud_cover sunshine global_radiation max_temp mean_temp min_temp
## 1 2011-11-23           7      2.0               35     13.5       6.8      2.6
## 2 2011-11-24           3      2.0               35     12.5       8.6      3.7
## 3 2011-11-25           3      5.0               52     14.0      11.0      9.5
## 4 2011-11-26           4      0.7               24     13.9      10.2      6.3
## 5 2011-11-27           3      5.9               55     13.2      11.8      9.7
## 6 2011-11-28           5      0.0               15     13.9       6.7      0.2
##   precipitation pressure snow_depth   Avg_kWh
## 1           0.2   102720          0  6.952692
## 2           0.2   102710          0  8.536480
## 3           0.0   102450          0  9.499781
## 4           0.0   102580          0 10.267707
## 5           0.0   102130          0 10.850805
## 6           0.0   102270          0  9.103382
```

Plotting the London Energy data

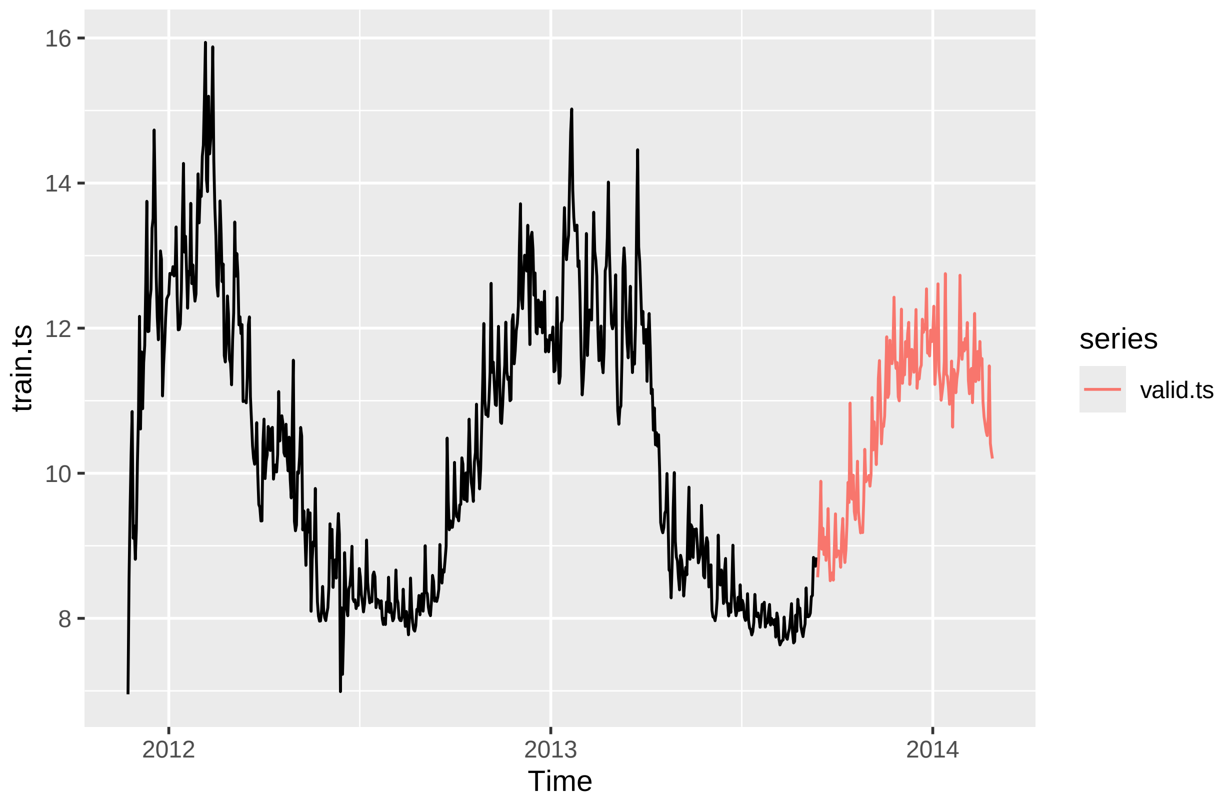
```r
## Carryover from above for ease of viewing
lw.ts <- ts(lw.df$precipitation, start = c(1979, 1), end = c(2019, 365), freq = 365)

autoplot(lw.ts)
```




```r
# Create a time series for energy usage from the merged dataset from 2011 to 2014
ew.ts <- ts(merged_data$Avg_kWh, start = c(2011, 327), end = c(2014, 58), freq = 365)

autoplot(ew.ts)
```

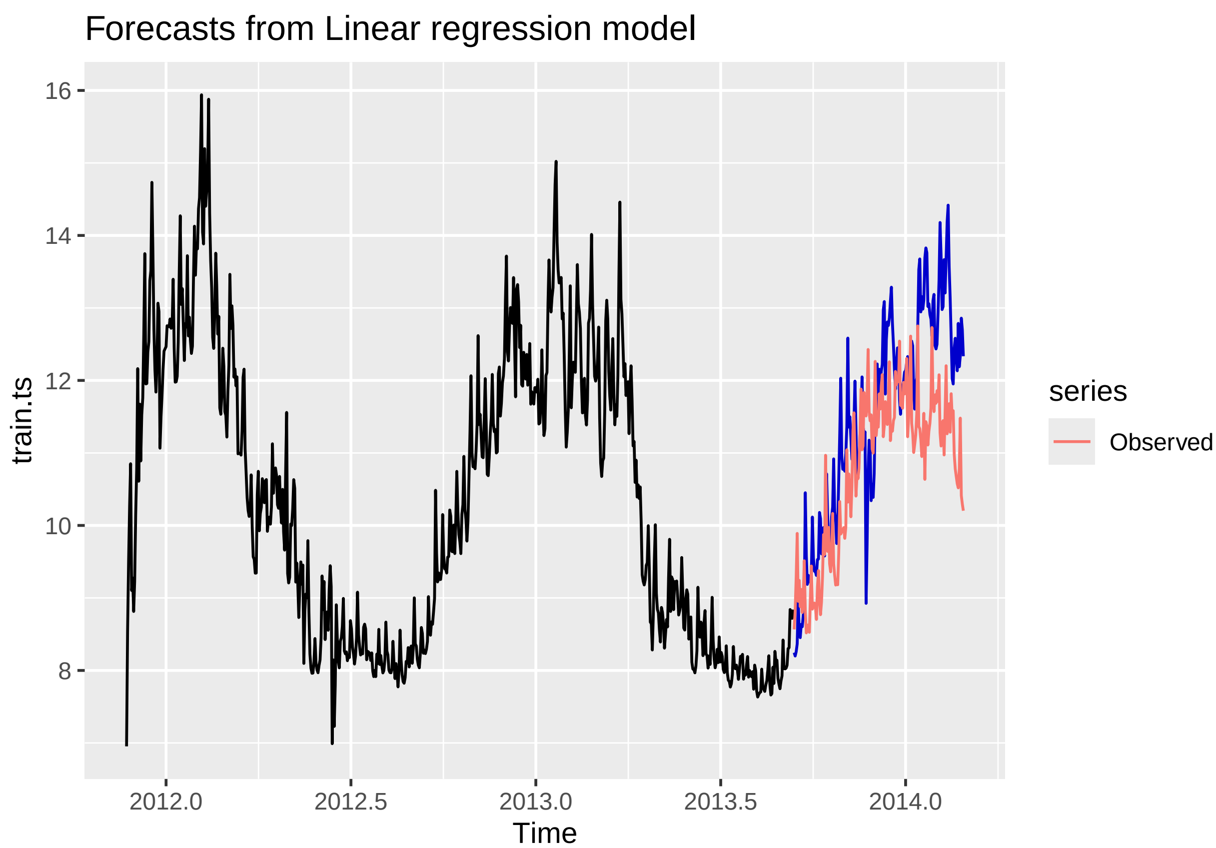



```r
# Define the number of observations in the validation set
nValid <- 168 # 168 observations

# Determine the number of observations for the training set
nTrain <- length(ew.ts) - nValid

# Create the training time series as a window ending on day 255 of 2013
train.ts <- window(ew.ts, end = c(2013, 255))

# Create the validation time series starting after the training set and ending on day 58 of 2014
valid.ts <- window(ew.ts, start=c(2013, 256), end = c(2014, 58))

# Plot both training and validation time series for visual comparison
autoplot(train.ts) +
  autolayer(valid.ts)
```



London Energy Lingear Regression Model

```
# Build a linear regression model on the training data including trend and seasonal components
# Dataset is very reliant on having seasonality
ew.lm <- tslm(train.ts ~ trend + season)

# Forecast future values using  linear model for the validation period
ew.lm.forecast <- forecast(ew.lm, h = nValid, level = 0)

# Plot the forecasted values and overlay the actual observed validation data
autoplot(ew.lm.forecast) +
  autolayer(valid.ts, series = "Observed")
```




```r
accuracy(ew.lm.forecast, valid.ts)
```

```
##                         ME     RMSE       MAE       MPE     MAPE     MASE
## Training set  1.248369e-16 0.595830 0.4117446 -0.289938 3.832608 0.444454
## Test set     -7.071820e-01 1.222471 0.9675930 -6.605798 8.996423 1.044460
##                   ACF1 Theil's U
## Training set 0.7943164        NA
## Test set     0.7030000  2.497657
```

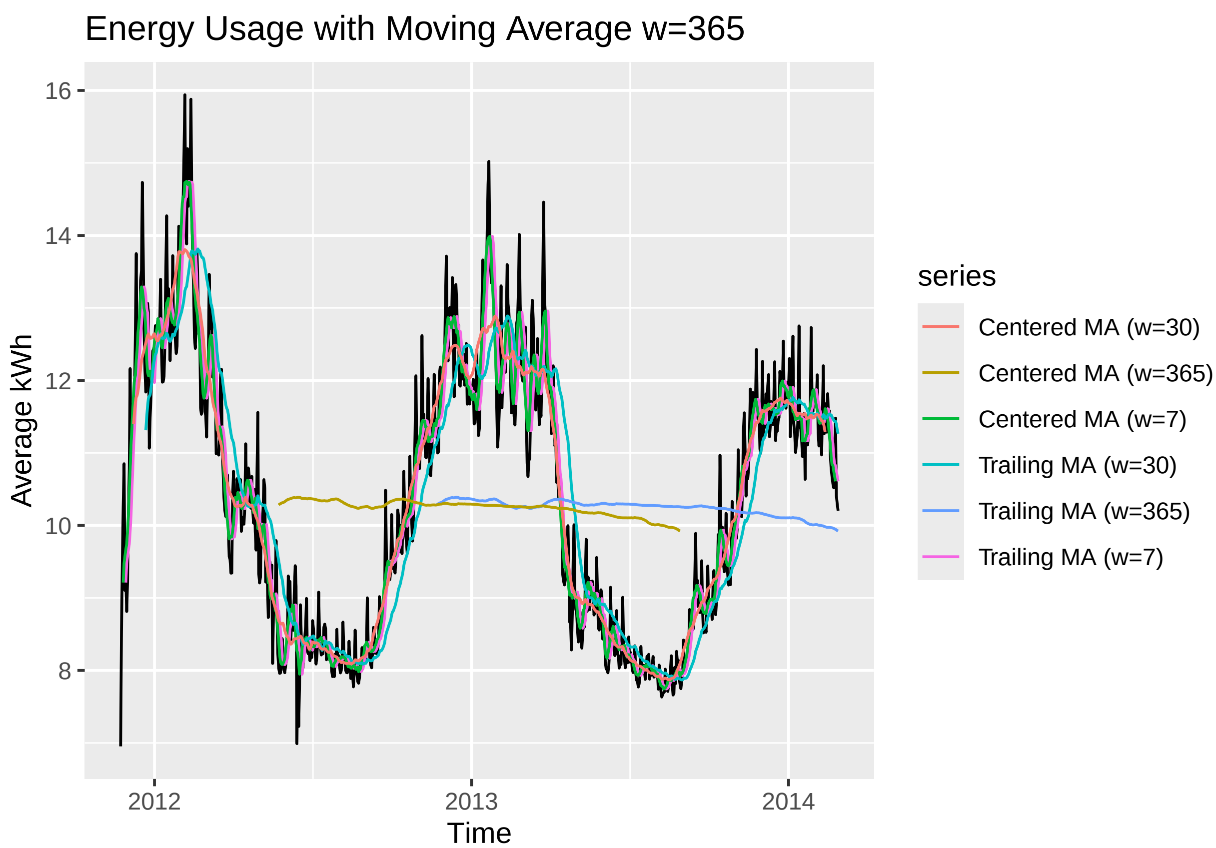
```r
library(zoo)
```

Data Visualization with Moving Averages

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
# Calculate trailing (right-aligned) and centered moving averages for a 7-day window
ma7.trailing <- rollmean(ew.ts, k = 7, align = "right")
ma7.centered <- ma(ew.ts, order = 7)

# Calculate trailing and centered moving averages for a 30-day window
ma30.trailing <- rollmean(ew.ts, k = 30, align = "right")
ma30.centered <- ma(ew.ts, order = 30)

# Calculate trailing and centered moving averages for a 365-day window (annual)
ma365.trailing <- rollmean(ew.ts, k = 365, align = "right")
ma365.centered <- ma(ew.ts, order = 365)

# Plot the original energy usage time series with all moving averages overlaid for comparison
autoplot(ew.ts) +
  autolayer(ma7.trailing, series = "Trailing MA (w=7)") +
  autolayer(ma7.centered, series = "Centered MA (w=7)") +
  autolayer(ma30.trailing, series = "Trailing MA (w=30)") +
  autolayer(ma30.centered, series = "Centered MA (w=30)") +
  autolayer(ma365.trailing, series = "Trailing MA (w=365)") +
  autolayer(ma365.centered, series = "Centered MA (w=365)") +
  xlab("Time") + ylab("Average kWh") +
  ggtitle("Energy Usage with Moving Average w=365")
```

```
## Warning: Removed 6 rows containing missing values or values outside the scale range
## (‘geom_line()‘).

## Warning: Removed 30 rows containing missing values or values outside the scale range
## (‘geom_line()‘).

## Warning: Removed 364 rows containing missing values or values outside the scale range
## (‘geom_line()‘).
```



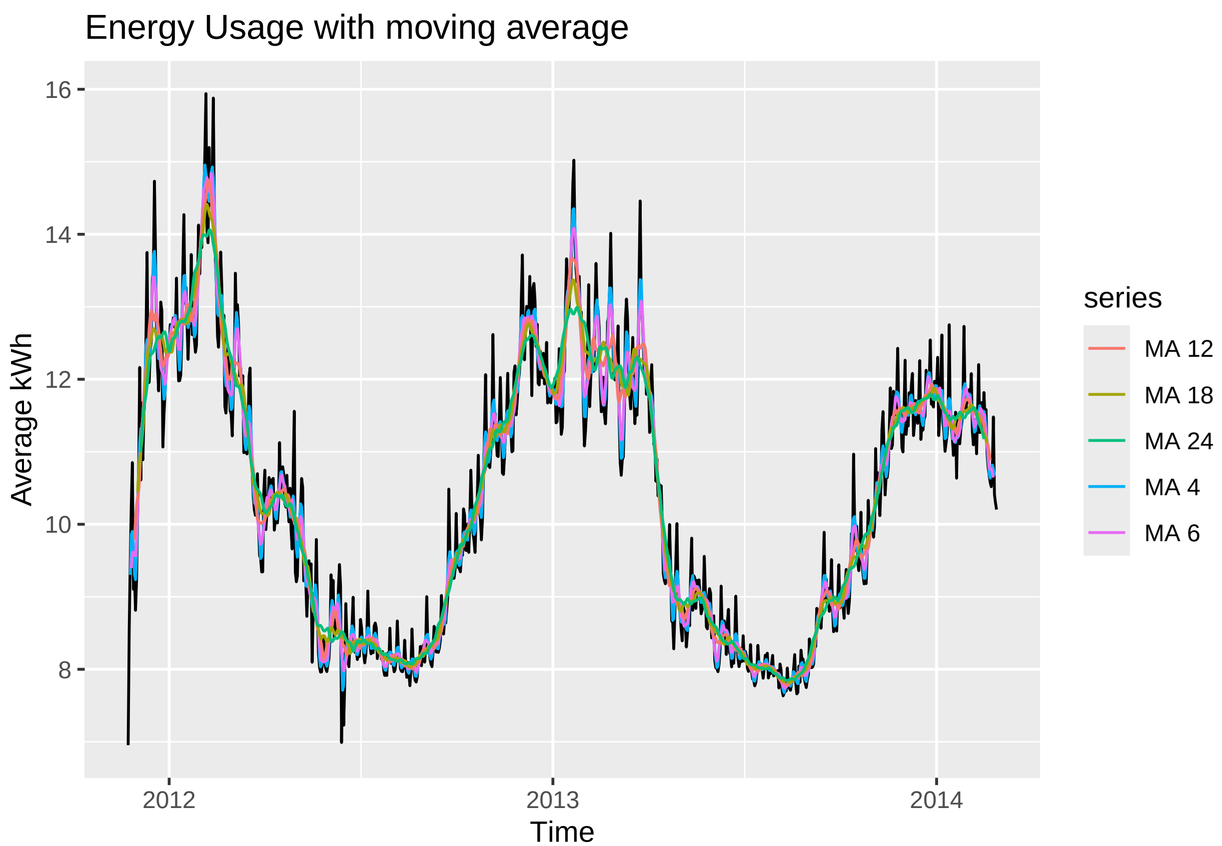
Moving average with various window sizes

1. Set window sizes of various sizes w=4, w=6, w=12, w = 18, w=24 to visualize CENTERED moving averages (MA) for the local and global patterns of the ridership data

If we want to remove seasonality pick window size that is number of seasons or a multiple of number of seasons

```r
# Compute centered moving averages for different window sizes
ma.4 <- ma(ew.ts, order = 4)
ma.6 <- ma(ew.ts, order = 6)
ma.12 <- ma(ew.ts, order = 12)
ma.18 <- ma(ew.ts, order = 18)
ma.24 <- ma(ew.ts, order = 24)

# Plot the original energy usage series with the different moving averages overlaid
autoplot(ew.ts) +
  autolayer(ma.4, series = "MA 4") +
  autolayer(ma.6, series = "MA 6") +
  autolayer(ma.12, series = "MA 12") +
  autolayer(ma.18, series = "MA 18") +
  autolayer(ma.24, series = "MA 24") +
  xlab("Time") + ylab("Average kWh") +
  ggtitle("Energy Usage with moving average")
```

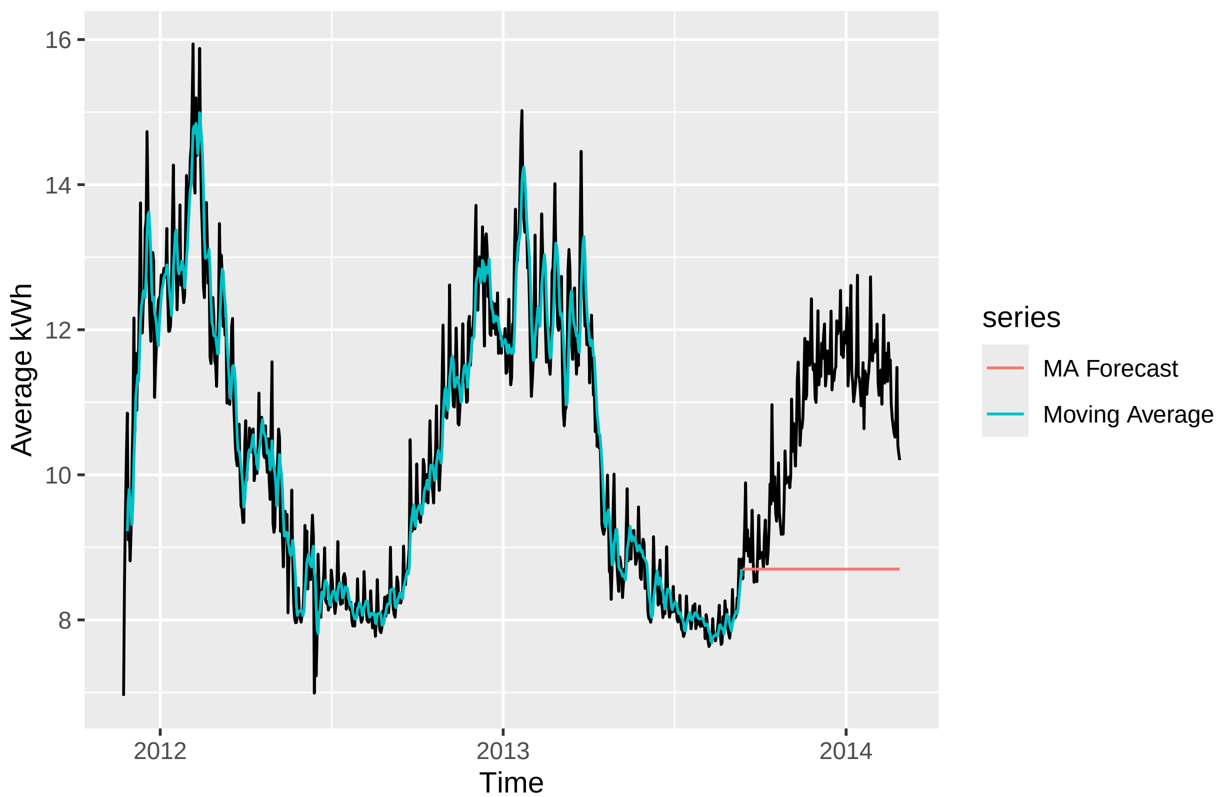
```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (‘geom_line()‘).

## Warning: Removed 6 rows containing missing values or values outside the scale range
## (‘geom_line()‘).

## Warning: Removed 12 rows containing missing values or values outside the scale range
## (‘geom_line()‘).

## Warning: Removed 18 rows containing missing values or values outside the scale range
## (‘geom_line()‘).

## Warning: Removed 24 rows containing missing values or values outside the scale range
## (‘geom_line()‘).
```



Forecasting with moving averages We can compute the moving average based on training data and forecast the last updated average for the rest of the validation period

```
# Calculate the trailing moving average for the training set with a 5-day window
ma.trailing <- rollmean(train.ts, k = 5, align = "right")

last.ma <- tail(ma.trailing, 1)

# The portion that belongs to the validation period
```

```r
ma.trailing.pred <- ts(rep(last.ma, nValid), start = c(2013, 256), end = c(2014, 58),
                       freq = 365)

#Ploting the original series, the training and moving average
autoplot(ew.ts) +
  autolayer(ma.trailing, series = "Moving Average") +
  autolayer(ma.trailing.pred, series = "MA Forecast") +
  xlab("Time") + ylab("Average kWh")
```




```r
accuracy(ma.trailing.pred, ew.ts)
```

```
##                ME     RMSE      MAE      MPE     MAPE      ACF1 Theil's U
## Test set 2.067287 2.343517 2.075669 18.28291 18.38097 0.8893074  4.635568
```

Differencing: Removes Trend and Seasonality

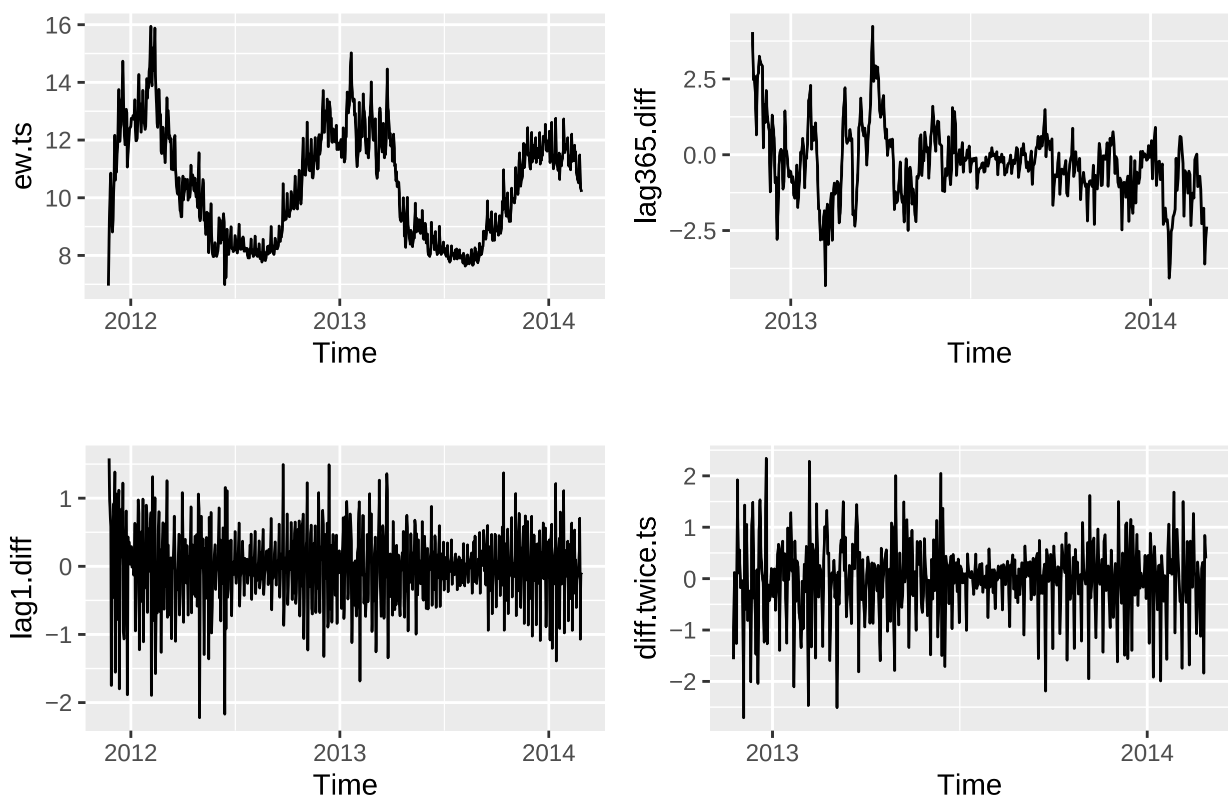
1-lag difference removes the trend and m-lag difference removes seasonality with m seasons

```r
# Load gridExtra for arranging multiple plots in a grid layout
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
# Compute the first difference (lag 1) to remove trend
lag1.diff <- diff(ew.ts, lag = 1)

# Compute the seasonal difference (lag 365) to remove annual seasonality
lag365.diff <- diff(ew.ts, lag = 365)

# Apply differencing twice: first remove seasonality (lag 365) then remove trend (lag 1)
diff.twice.ts <- diff(diff(ew.ts, lag = 365), lag = 1)

# Set up a 2x2 plot layout
par(mfrow=c(2, 2))

# Generate plots for the original series and each differenced series
rider.plot <- autoplot(ew.ts)
lag365.plot <- autoplot(lag365.diff)
lag1.plot <- autoplot(lag1.diff)
diff.twice.plot <- autoplot(diff.twice.ts)

# Arrange all four plots in a 2x2 grid for comparison
grid.arrange(rider.plot, lag365.plot, lag1.plot, diff.twice.plot, ncol = 2, nrow = 2)
```



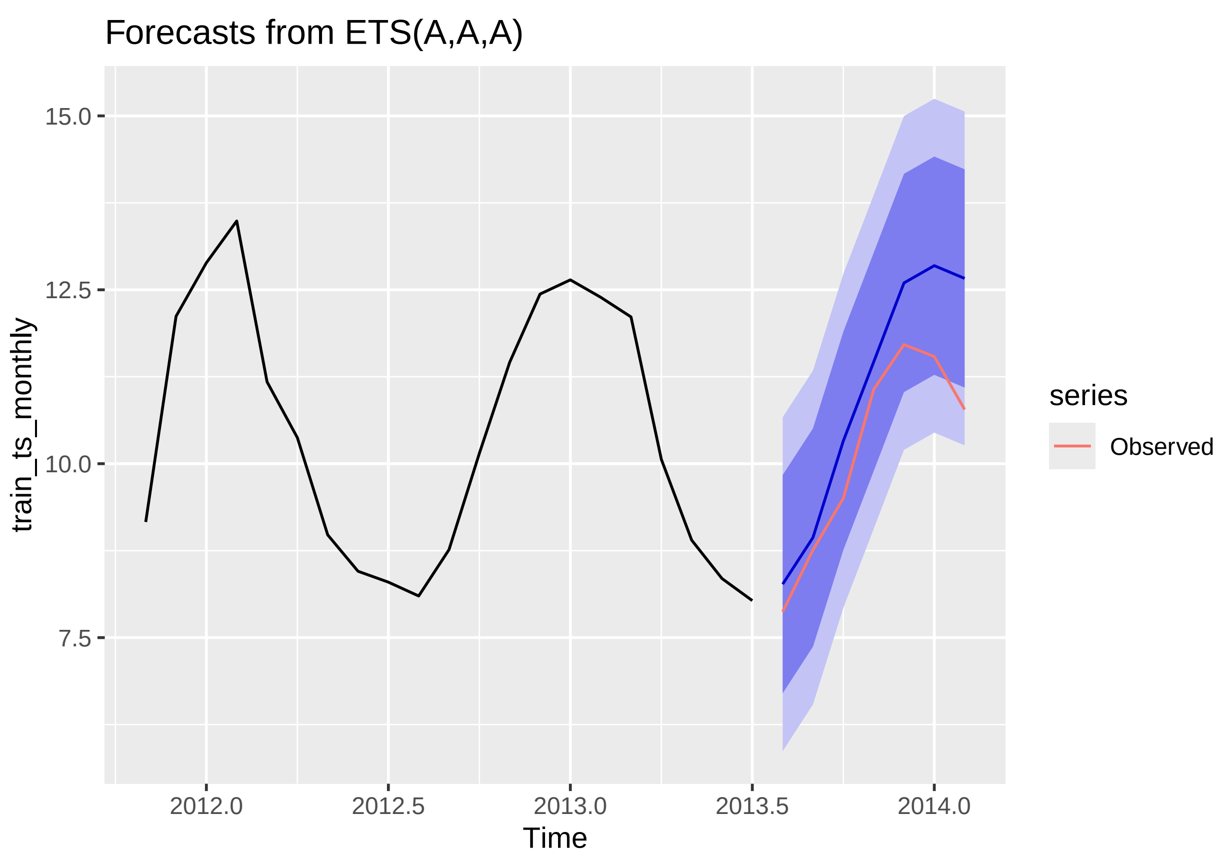
Holt Winter Model

```r
library(dplyr)
library(lubridate)

# Aggregate the merged dataset to a monthly frequency
# Create a 'Month' column by flooring the Date to the first day of the month
# Group by Month and compute the average precipitation, mean temperature, and energy usage
monthly_merged <- merged_data %>%
  mutate(Month = floor_date(Date, unit = "month")) %>%
  group_by(Month) %>%
  summarise(
    Avg_precipitation = mean(precipitation, na.rm = TRUE),
    Avg_mean_temp = mean(mean_temp, na.rm = TRUE),
    Avg_kWh = mean(Avg_kWh, na.rm = TRUE)
  ) %>%
  ungroup()

# Create a time series for the monthly average energy usage
monthly_ts <- ts(monthly_merged$Avg_kWh,
                 start = c(year(min(monthly_merged$Month)), month(min(monthly_merged$Month))),
                 frequency = 12)

# Split the data into training (75%) and validation (25%) sets like in the original split
n_months <- length(monthly_ts)
n_valid <- ceiling(0.25 * n_months)
n_train <- n_months - n_valid

# Create the training and validation monthly time series
train_ts_monthly <- window(monthly_ts, end = time(monthly_ts)[n_train])
valid_ts_monthly <- window(monthly_ts, start = time(monthly_ts)[n_train + 1])

# Load the forecast package and fit an Exponential Smoothing (ETS) model with additive error, trend, an
library(forecast)
ew.hwin <- ets(train_ts_monthly, model = "AAA")
summary(ew.hwin)
```

```
## ETS(A,A,A)
##
## Call:
## ets(y = train_ts_monthly, model = "AAA")
##
##   Smoothing parameters:
##     alpha = 2e-04
##     beta  = 2e-04
##     gamma = 0.9209
##
##   Initial states:
##     l = 10.0973
##     b = 0.0142
##     s = -0.039 -1.4515 -2.1701 -1.9692 -1.8193 -1.2418
##            0.2553 1.1992 3.3277 2.7725 2.0732 -0.9371
##
##   sigma:  1.2242
##
```

```
##        AIC      AICc       BIC
##   76.29326 280.29326  94.05015
##
## Training set error measures:
##                        ME      RMSE       MAE        MPE     MAPE      MASE
## Training set -0.02100844 0.5973341 0.3232039 -0.3144498 2.954716 0.5153589
##                      ACF1
## Training set -0.05114027
```

```r
# Forecast the validation period using the fitted ETS model
ew.hwin.pred <- forecast(ew.hwin, h = n_valid)

# Plot the ETS forecast along with the observed monthly validation data
autoplot(ew.hwin.pred) + autolayer(valid_ts_monthly, series = "Observed")
```

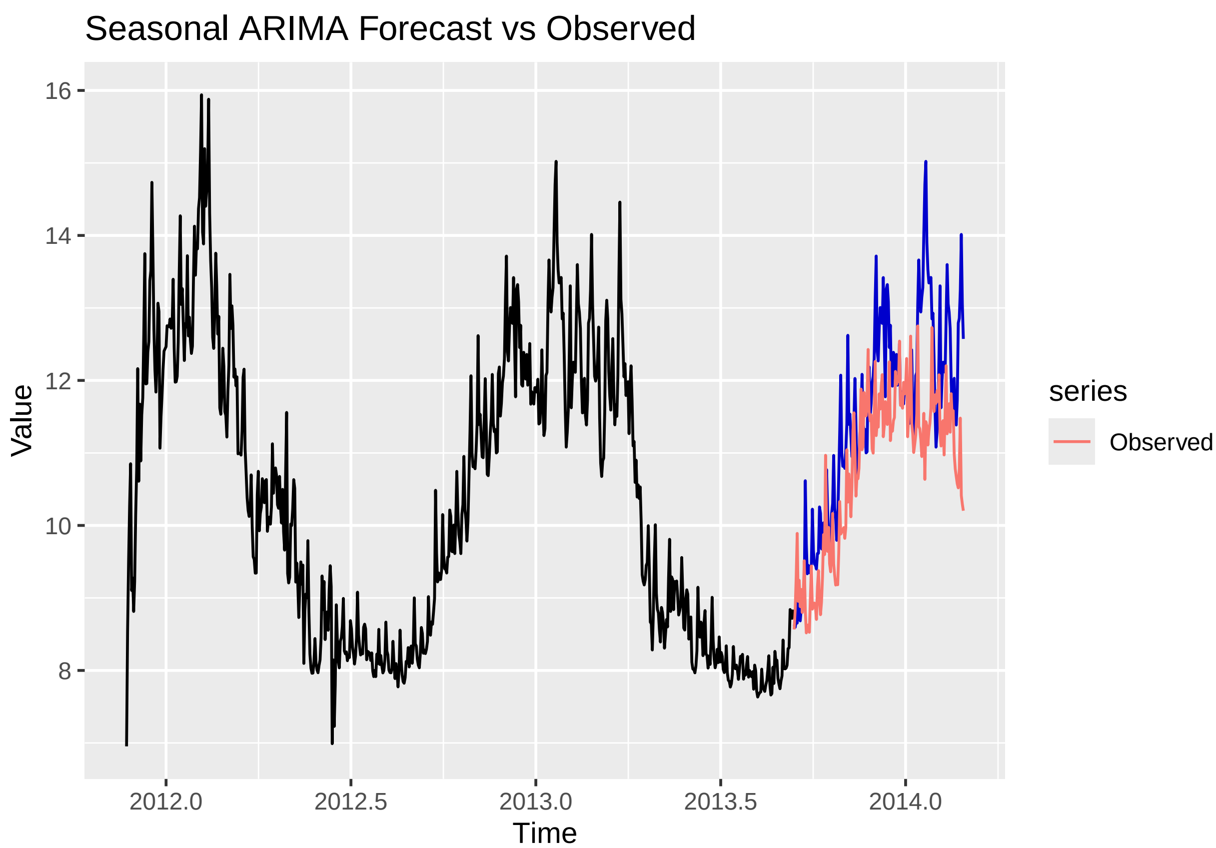



```r
# Calculate forecast accuracy metrics (e.g., MAPE) for the ETS model on the validation set
accuracy(ew.hwin.pred, valid_ts_monthly)
```

```
##                        ME      RMSE       MAE        MPE     MAPE      MASE
## Training set -0.02100844 0.5973341 0.3232039 -0.3144498 2.954716 0.5153589
## Test set     -0.83876698 1.0047280 0.8387670 -7.9555977 7.955598 1.3374407
##                      ACF1 Theil's U
## Training set -0.05114027        NA
## Test set      0.37246955    1.0145
```

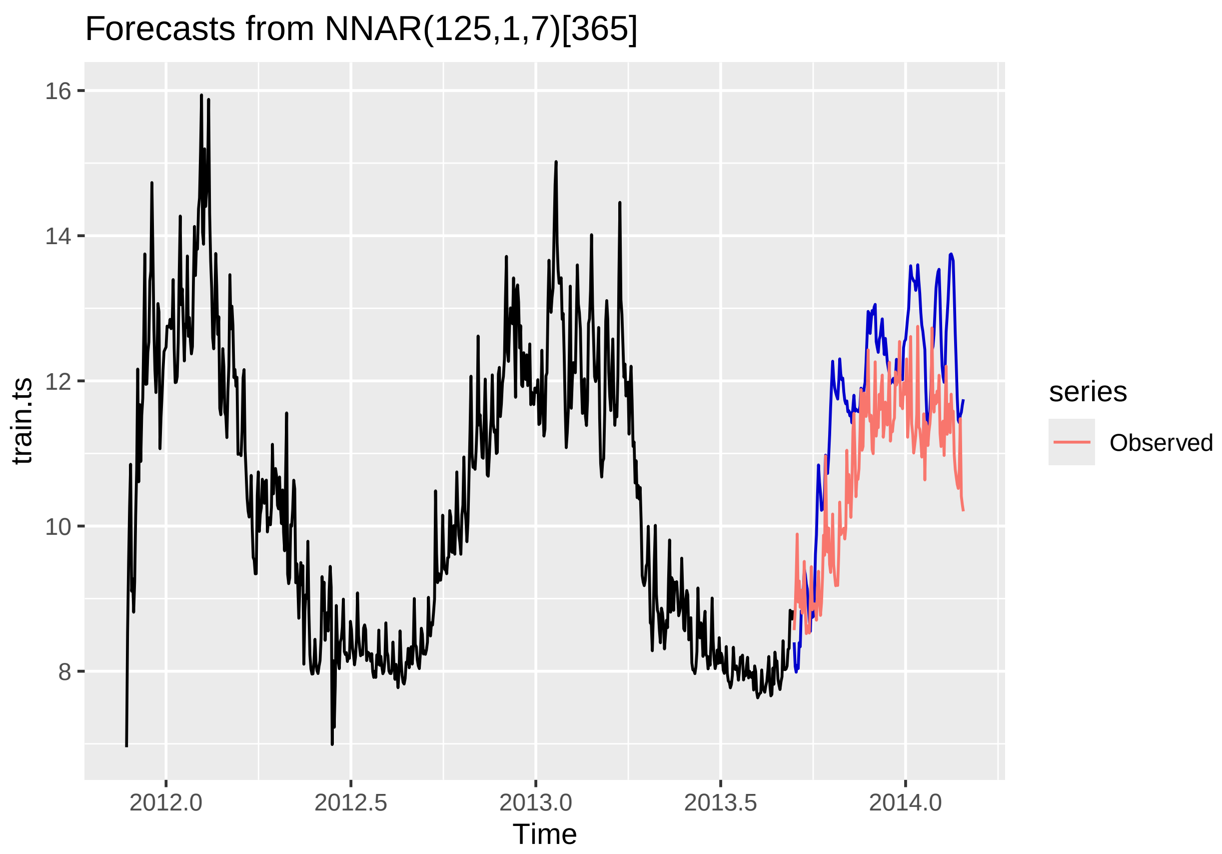
Step 3 and 4: Build an arima() model - Model 3 Our SARIMA model took a long time to run so we ran it using the commented code below and then saved it, importing it from memory each time we needed it instead of re-training and re-predicting each time.

```
library(forecast)

# Fit seasonal ARIMA with frequency 365
# ew.arima <- auto.arima(train.ts, seasonal = TRUE, D = 1, max.P = 1, max.Q = 1, stepwise = FALSE, appr

# Takes forever to run so I saved it
# saveRDS(ew.arima, file = "ew_arima_model.rds")

# Forecast
# ew.arima.forecast <- forecast(ew.arima, h = nValid, level = 0)

# Save the forecast to be safe
# saveRDS(ew.arima.forecast, file = "ew_arima_forecast.rds")

# Code to load the arima and forecast
# Load the fitted model
ew.arima <- readRDS('/Users/TomTheIntern/Desktop/Mendoza/Mod 3/tsf/ew_arima_model (2).rds')

# Load the forecast
ew.arima.forecast <- readRDS('/Users/TomTheIntern/Desktop/Mendoza/Mod 3/tsf/ew_arima_forecast (2).rds')

# Plot
autoplot(ew.arima.forecast) +
  autolayer(valid.ts, series = "Observed") +
  ggtitle("Seasonal ARIMA Forecast vs Observed") +
  xlab("Time") + ylab("Value")
```

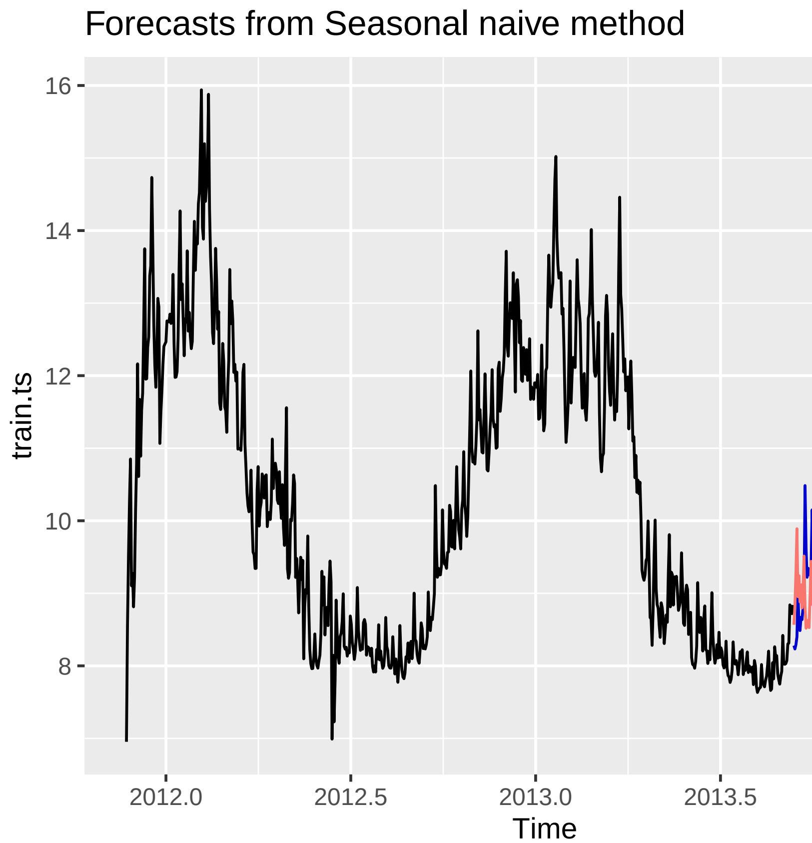


Step 3 and 4: Build a NN model - Model 4 We will build a NN model with a few parameters

```
# Set parameters for the Neural Network Time Series model:
p <- 125  # Number of previous time steps used for forecast
P <- 1   # Number of previous seasonal values to use
size <- 7  # Number of hidden nodes
repeats <- 20 # Number of iterations or epochs to train the neural network

# Fit the neural network time series model (NNETAR) on the training data with the specified parameters
ew.nnetar <- nnetar(train.ts, repeats = repeats, p = p, P = P, size = size)

# Generate forecasts from the NN model for the validation period
ew.nnetar.forecast <- forecast(ew.nnetar, h = nValid)

# Plot the NN forecast along with the actual observed validation data
autoplot(ew.nnetar.forecast) +
  autolayer(valid.ts, series = "Observed")
```



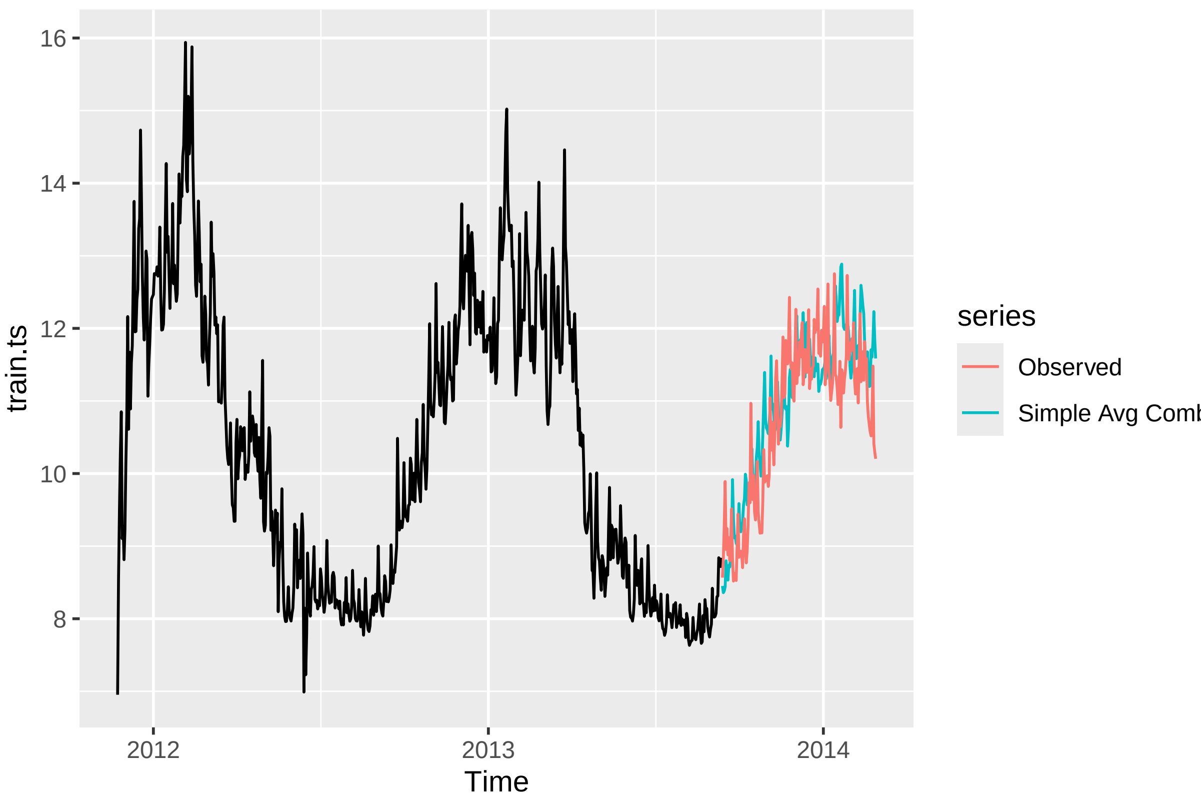

```r
# Fit a seasonal naive model on the training data, which uses the last observed value from the same sea
ew.snaive <- snaive(train.ts, h = nValid, level = 0)

# Forecast using the seasonal naive model for the validation period
ew.snaive.forecast <- forecast(ew.snaive, h = nValid)

# Plot the seasonal naive forecast and overlay the observed validation data
autoplot(ew.snaive.forecast) +
  autolayer(valid.ts, series = "Observed")
```



Step 3 and 4: Build a seasonal naive - Model 5

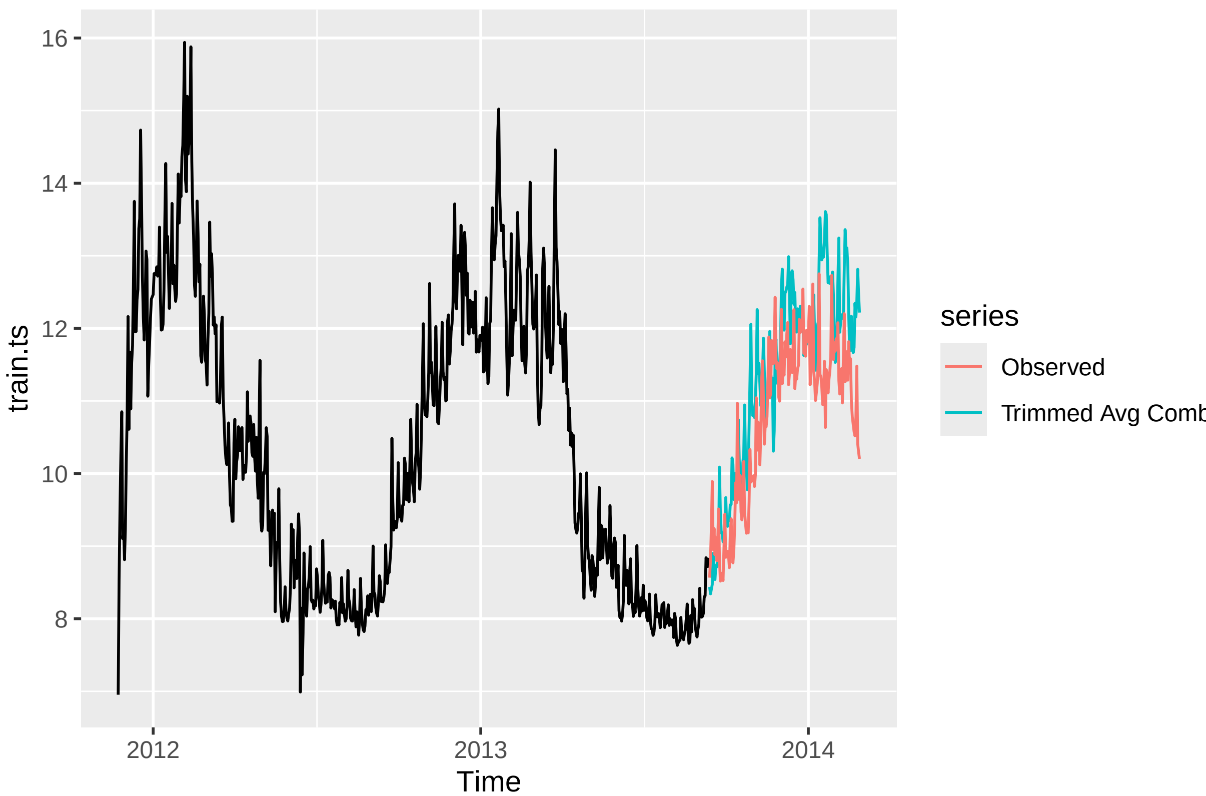
Step 5: Aggregate multiple forecasts

```
# Combine forecasts from five different models by computing their simple average.
# Models: Linear regression, ARIMA, NN, Seasonal Naive, and Moving Average.

num.models <- 5
ew.comb.simple.avg <- (ew.lm.forecast$mean +
                       ew.arima.forecast$mean +
                       ew.nnetar.forecast$mean +
                       ew.snaive$mean +
                       ma.trailing.pred) / num.models

# Plot the training series, the combined forecast (simple average), and the observed validation data
autoplot(train.ts) +
  autolayer(ew.comb.simple.avg, series = "Simple Avg Comb") +
  autolayer(valid.ts, series = "Observed")
```



Simple Average

```
# Collect forecasts into a dataframe
forecast.vectors.df <- data.frame(cbind(
  ew.lm.forecast$mean,
  ew.arima.forecast$mean,
  ew.nnetar.forecast$mean,
  ew.snaive$mean,
  ma.trailing.pred))

# Apply 20% trimming (removes highest and lowest model forecasts)
# Calculate a trimmed mean by removing the highest and lowest 20% of forecasts for each time point
forecast.vectors.df$comb.trimmed.avg <- apply(forecast.vectors.df, 1, function(x) mean(x, trim = 0.2))

# Convert into time series object
ew.comb.trimmed.avg <- ts(forecast.vectors.df$comb.trimmed.avg, start = c(2013, 256), end = c(2014, 58)

# Plot the training series, the trimmed average forecast, and the observed validation series
autoplot(train.ts) +
  autolayer(ew.comb.trimmed.avg, series = "Trimmed Avg Comb") +
  autolayer(valid.ts, series = "Observed")
```



Trimmed mean

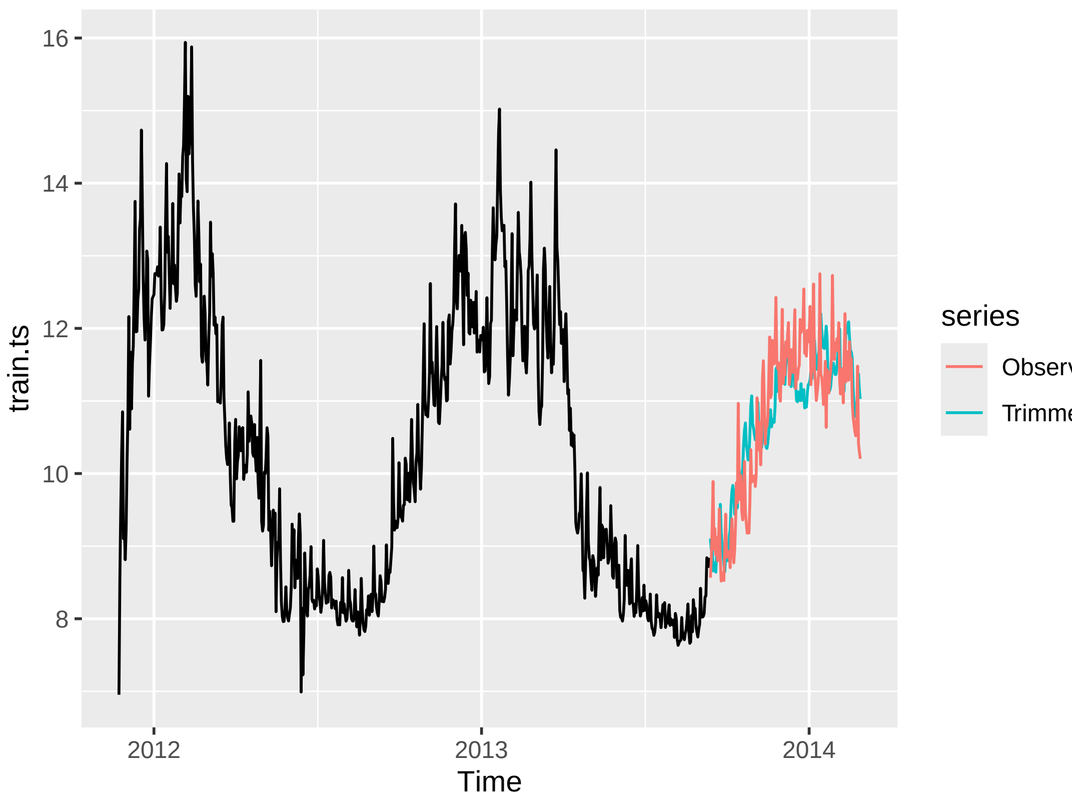
```r
# Collect forecasts into a dataframe
forecast.vectors.df <- data.frame(cbind(
  ew.lm.forecast$mean,
  ew.arima.forecast$mean,
  ew.nnetar.forecast$mean,
  ew.snaive$mean,
  ma.trailing.pred))

# Add the validation set as another column for model fitting
forecast.vectors.df$valid <- valid.ts

# Fit a linear regression model where the validation data is regressed on the forecasts
# This finds optimal weights to combine the forecasts
forecasts.lm <- lm(valid.ts ~ ew.lm.forecast$mean + ew.arima.forecast$mean + ew.nnetar.forecast$mean + e

# Display the summary of the regression model to assess forecast combination
summary(forecasts.lm)
```

Running a regression that best fits the validation data

```
##
## Call:
## lm(formula = valid.ts ~ ew.lm.forecast$mean + ew.arima.forecast$mean +
```

```
##     ew.nnetar.forecast$mean + ew.snaive$mean + ma.trailing.pred,
##     data = forecast.vectors.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.39576 -0.43800  0.03149  0.36319  1.38669
##
## Coefficients: (1 not defined because of singularities)
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.10991    0.70623   1.572   0.1180
## ew.lm.forecast$mean       0.03918    0.07794   0.503   0.6159
## ew.arima.forecast$mean    3.17794    1.19298   2.664   0.0085 **
## ew.nnetar.forecast$mean   0.49724    0.06239   7.970 2.61e-13 ***
## ew.snaive$mean           -2.89204    1.18260  -2.445   0.0155 *
## ma.trailing.pred               NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5985 on 163 degrees of freedom
## Multiple R-squared:  0.7148, Adjusted R-squared:  0.7078
## F-statistic: 102.1 on 4 and 163 DF,  p-value: < 2.2e-16
```

```r
# Convert the fitted values from the regression model into a time series object
# Aligns the forecast with the correct time indices
ew.comb.regression <- ts(forecasts.lm$fitted.values, start = c(2013, 256), end = c(2014, 58), freq = 365

# Plot the training series, the regression combined forecast, and the observed validation data
autoplot(train.ts) +
  autolayer(ew.comb.regression, series = "Trimmed Avg Comb") +
  autolayer(valid.ts, series = "Observed")
```



Plotting the regression fit

```
# Compute and compare the Mean Absolute Percentage Error (MAPE) for various forecasting methods:
# LM: Linear Regression, ARIMA: Seasonal ARIMA, NNAR: Neural Network, SNAIVE: Seasonal Naive, MA: Movin
c(
  LM = accuracy(ew.lm.forecast, valid.ts)["Test set", "MAPE"],
  ARIMA = accuracy(ew.arima.forecast, valid.ts)["Test set", "MAPE"],
  NNAR = accuracy(ew.nnetar.forecast, valid.ts)["Test set", "MAPE"],
  SNAIVE = accuracy(ew.snaive, valid.ts)["Test set", "MAPE"],
  MA = accuracy(ma.trailing.pred, valid.ts)["Test set", "MAPE"],
  comb.simple.avg = accuracy(ew.comb.simple.avg, valid.ts)["Test set", "MAPE"],
  comb.trimmed.avg = accuracy(ew.comb.trimmed.avg, valid.ts)["Test set", "MAPE"],
  comb.reg = accuracy(forecasts.lm$fitted.values, valid.ts)["Test set", "MAPE"]
)
```

Finally, compare the accuracy of all the models - MAPE

```
##               LM            ARIMA             NNAR           SNAIVE
##         8.996423         8.312332        10.280355         8.347393
##               MA  comb.simple.avg comb.trimmed.avg         comb.reg
##        18.380967         5.358319         7.260593         4.541132
```

Based on the MAPE, the regression-based combination model (comb.reg) is the best performer with a MAPE of 4.90. Lower MAPE means that the forecast errors are smaller in relation to the actual values.

To summarize the models:

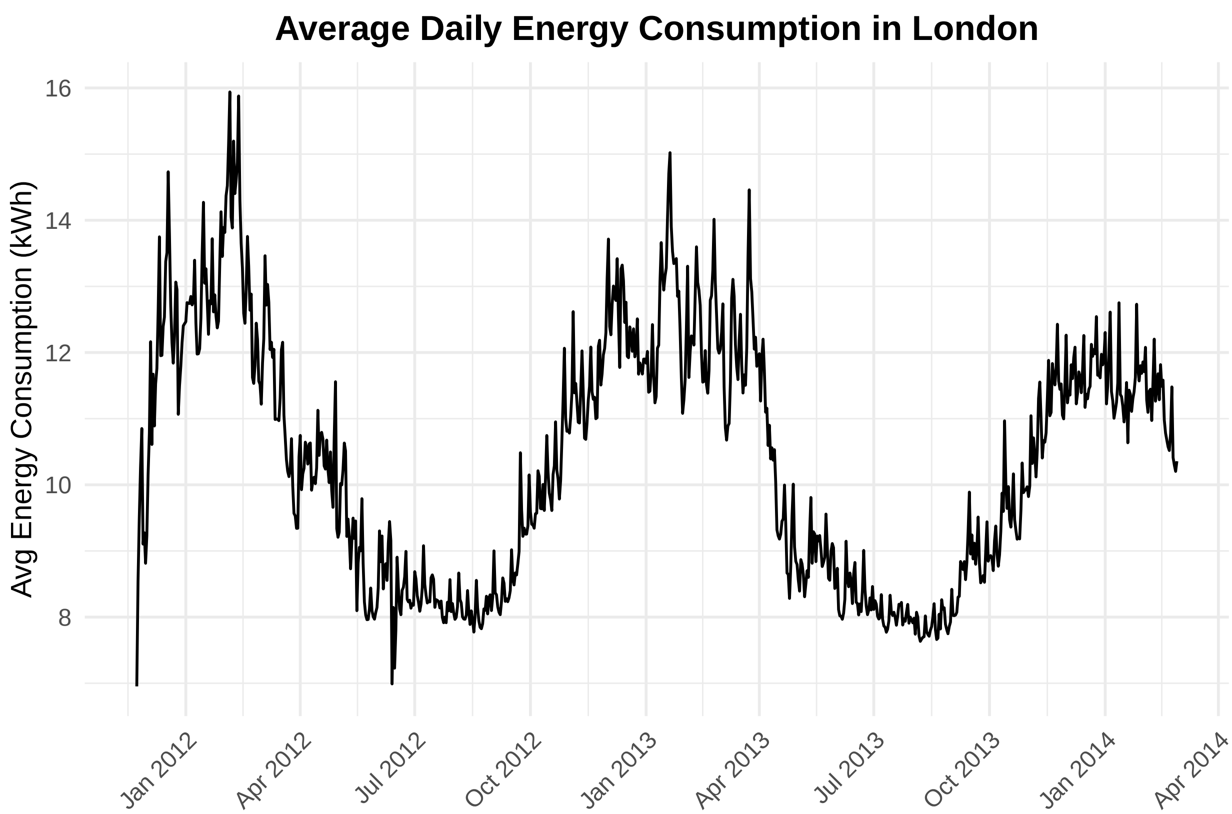
Individual models like LM, ARIMA, NNAR, and SNAIVE have MAPE values ranging from about 8.3 to 10.65. The moving average (MA) model has a very high MAPE of 19.67. This is likely due to it not handling seasonal data and differencing attempts not working.

The combined forecasts via simple averaging and trimmed averaging improve the performance to 5.54 and 7.63 respectively, but the regression-based combination outperforms them all at 4.90.

The regression-based combination (comb.reg) is the most accurate among the models tested.

This is where we developed the lagged models, so some of the code is repeated as the responsiblties were split between group members.

```r
library(forecast)
library(readxl)
library(ggplot2)
library(dplyr)
library(readr)
library(lubridate)

#making a new dataframe for reference
daily_energy <- le.df %>%
  group_by(Date) %>%
  summarise(Avg_kWh = mean(KWH, na.rm = TRUE))

#making a time series
de.ts <- ts(daily_energy$Avg_kWh, start = c(2011, 327), end = c(2014, 58), freq = 365)

#changing the format of the London energy dataframe
le.df$Date <- as.Date(le.df$Date, format="%Y-%m-%d")

#making a graph of energy plot by removing the last row
graph_energy <- daily_energy[ -nrow(daily_energy), ]

#changing the column name of graph_energy
colnames(graph_energy)[2] <- "Avg_kWh"

#formatting the dates of graph energy
graph_energy$Date <- ymd(graph_energy$Date)

#making a plot of graph enegry
ggplot(graph_energy, aes(x = Date, y = Avg_kWh)) +
  geom_line(color = "black") +
  labs(title = "Average Daily Energy Consumption in London",
       y = "Avg Energy Consumption (kWh)",
       x = "" )  +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_x_date(date_breaks = "3 months", date_labels = "%b %Y")
```



Unioning dataframes

```r
library(dplyr)
#Joining daily energy with weather on the date
merged_data <- merge(lw.df, daily_energy, by.x = "date", by.y = "Date" , all.y = TRUE)

#View first few rows
head(merged_data)
```

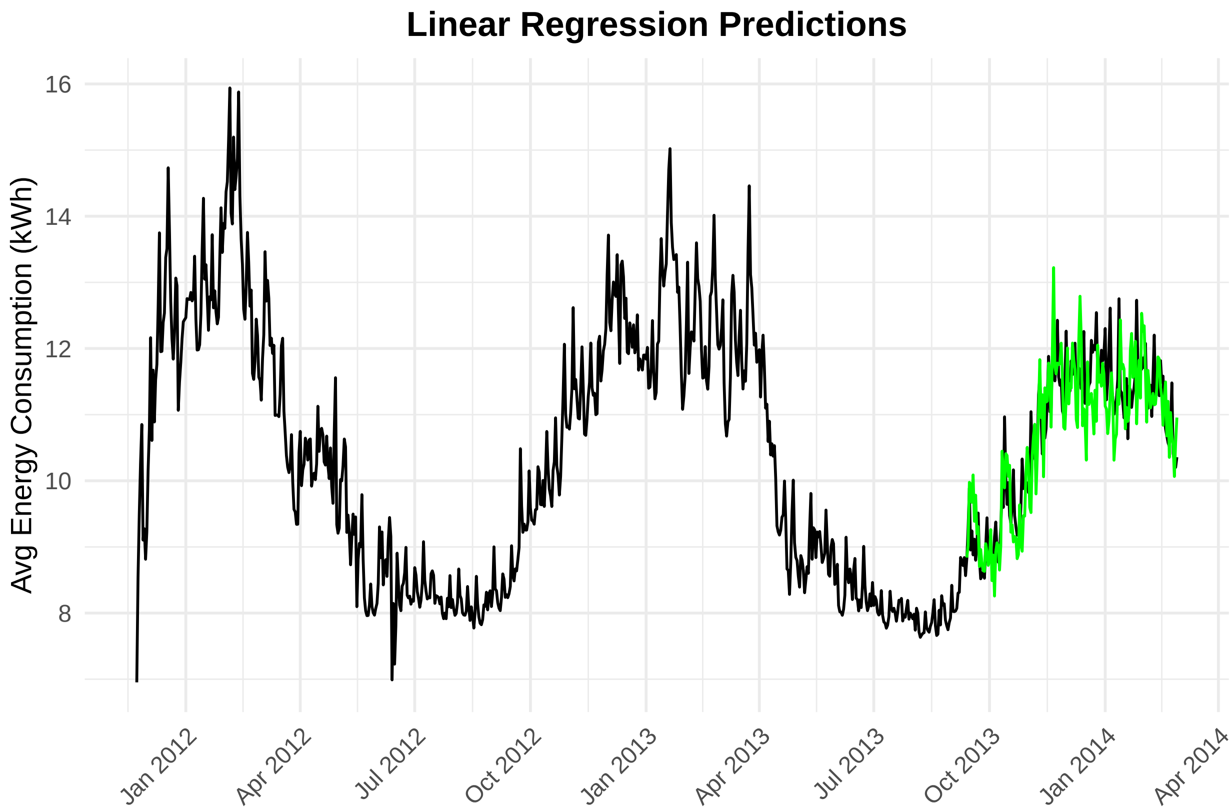
```
##         date cloud_cover sunshine global_radiation max_temp mean_temp min_temp
## 1 2011-11-23           7      2.0               35     13.5       6.8      2.6
## 2 2011-11-24           3      2.0               35     12.5       8.6      3.7
## 3 2011-11-25           3      5.0               52     14.0      11.0      9.5
## 4 2011-11-26           4      0.7               24     13.9      10.2      6.3
## 5 2011-11-27           3      5.9               55     13.2      11.8      9.7
## 6 2011-11-28           5      0.0               15     13.9       6.7      0.2
##   precipitation pressure snow_depth       Date   Avg_kWh
## 1           0.2   102720          0 2011-11-23  6.952692
## 2           0.2   102710          0 2011-11-24  8.536480
## 3           0.0   102450          0 2011-11-25  9.499781
## 4           0.0   102580          0 2011-11-26 10.267707
## 5           0.0   102130          0 2011-11-27 10.850805
## 6           0.0   102270          0 2011-11-28  9.103382
```

```
#making a timeseries of the merged data
merged.ts <- ts(merged_data$Avg_kWh, start = c(2011, 327), end = c(2014, 58), freq = 365)
```

Creating lagged variables and making a lagged linear regression

```
#getting the number of rows
nPeriods <- nrow(merged_data)

#creating lagged variables
merged_data$Lag_Mean_Temp <- dplyr::lag(merged_data$mean_temp, n=1)
merged_data$Lag_Snow <- dplyr::lag(merged_data$snow_depth, n=1)
merged_data$Lag_Precip <- dplyr::lag(merged_data$precipitation, n=1)
merged_data$Lag_Sun <- dplyr::lag(merged_data$sunshine, n=1)

#inputting a time variable
merged_data$time <- seq(1, nPeriods, 1)

#making the seasonal cosine and sine
merged_data$Seasonal_sine <- sin(2*pi*merged_data$t/365.25)
merged_data$Seasonal_cosine <- cos(2*pi*merged_data$t/365.25)

#making a train sample
train_merged <- merged_data[merged_data$date <= as.Date("2013-09-12"), ]
train_merged <- train_merged[2:nrow(train_merged), ]

#making a test set
test_merged <- merged_data[merged_data$date > as.Date("2013-09-12"), ]
test_merged <- test_merged[1:nrow(test_merged) - 1, ]

#initial energy regression model
energy.lr <- glm(Avg_kWh ~ Lag_Mean_Temp + Lag_Snow + Lag_Precip + Lag_Sun + Seasonal_cosine + Seasonal_
                 data = train_merged,
                 family = gaussian())

#making a simplified regression model
energy.lr <- glm(Avg_kWh ~ Lag_Mean_Temp + Lag_Precip + Lag_Sun,
                 data = train_merged,
                 family = Gamma())

#making predictions of the regression
energy.lr.pred <- predict(energy.lr, test_merged, type = 'response')

lr_pred_df <- data.frame(Date = test_merged$date,
                         Avg_kWh = energy.lr.pred)

#plotting the linear regression predictions
ggplot(graph_energy, aes(x = Date, y = Avg_kWh)) +
  geom_line(color = "black") +
  geom_line(data = lr_pred_df, color = "green") +
  labs(title = "Linear Regression Predictions",
       y = "Avg Energy Consumption (kWh)",
       x = "" )  +
  theme_minimal() +
```

```r
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "bottom") +
  scale_x_date(date_breaks = "3 months", date_labels = "%b %Y")
```



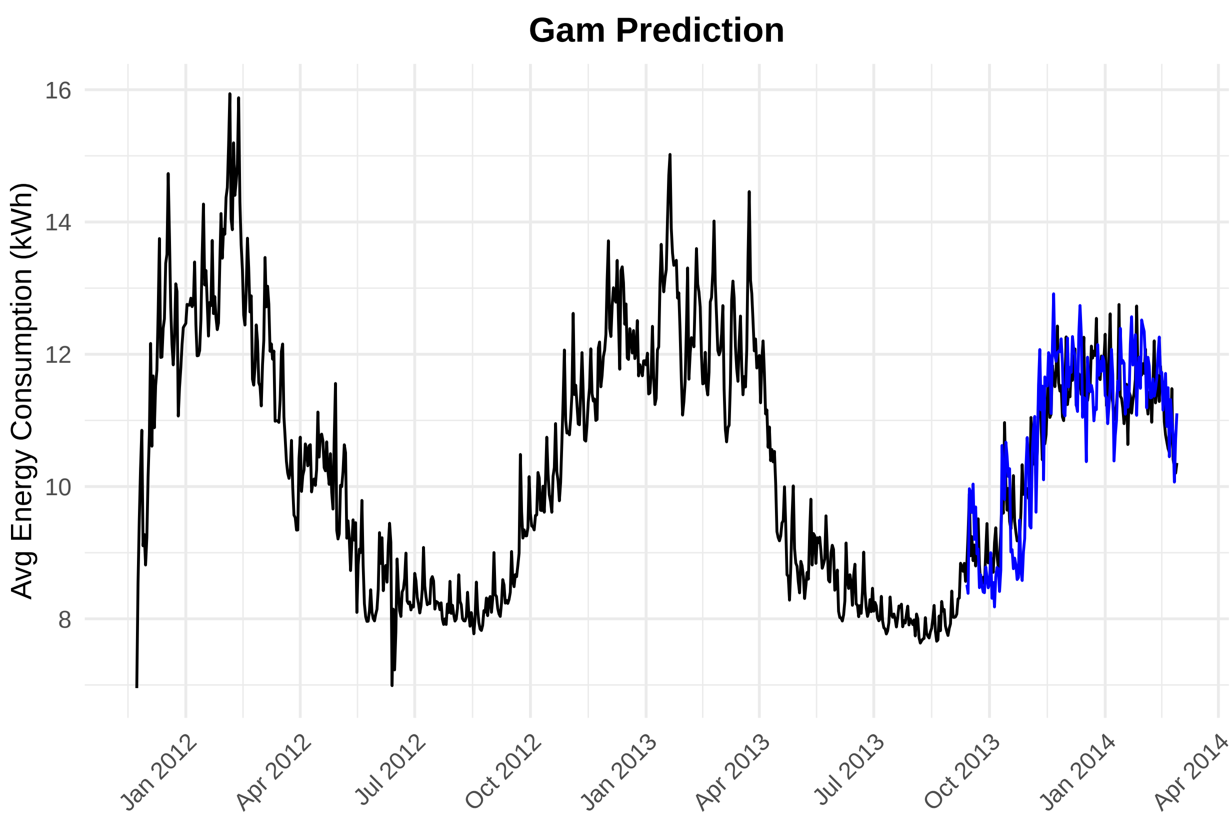
Making a lagged GAM model

```r
library(gam)
```

```
## Warning: package 'gam' was built under R version 4.4.1

## Loading required package: splines

## Loading required package: foreach

## Loaded gam 1.22-5
```

```r
#making a gam model with splines
energy.gam <- gam(Avg_kWh ~ s(Lag_Mean_Temp, df = 4) + s(Lag_Snow, df = 4) +
                    s(Lag_Precip, df = 4) + s(Lag_Sun, df = 4),
    data = train_merged, family = Gamma())

#making predictions with the gam model
energy.gam.pred <- predict(energy.gam, test_merged, type = 'response')
```

```r
gam_pred_df <- data.frame(Date = test_merged$date,
                          Avg_kWh = energy.gam.pred)

#plotting the gam predictions
ggplot(graph_energy, aes(x = Date, y = Avg_kWh)) +
  geom_line(color = "black") +
  geom_line(data = gam_pred_df, color = "blue") +
  labs(title = "Gam Prediction",
       y = "Avg Energy Consumption (kWh)",
       x = "" )  +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "bottom") +
  scale_x_date(date_breaks = "3 months", date_labels = "%b %Y")
```



Making a lagged ARIMAX model

```r
library(forecast)

#making a time series from the train data
ts_data <- ts(train_merged$Avg_kWh, frequency = 365, start = c(2011, 327))  # adjust start as needed

#Making lagged variables
xreg_train <- as.matrix(train_merged[, c("Lag_Mean_Temp", "Lag_Snow",
```

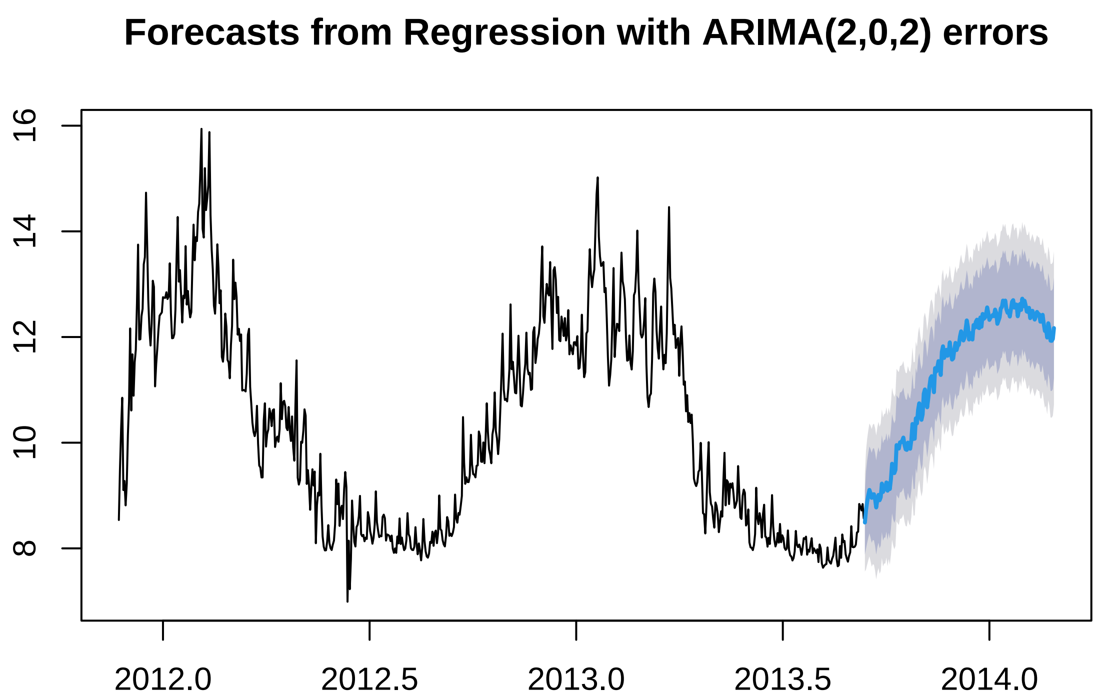
```r
                                             "Lag_Precip", "Lag_Sun",
                                             "Seasonal_cosine", "Seasonal_sine")])
#training the arimax model with lagged variables
arimax_model <- auto.arima(ts_data, xreg = xreg_train)
summary(arimax_model)
```

```
## Series: ts_data
## Regression with ARIMA(2,0,2) errors
##
## Coefficients:
##          ar1      ar2      ma1      ma2  intercept  Lag_Mean_Temp  Lag_Snow
##       1.2453  -0.2858  -0.5600  -0.1456    10.8846        -0.0662   -0.0477
## s.e.  0.1200   0.1097   0.1187   0.0619     0.1812         0.0112    0.0460
##       Lag_Precip  Lag_Sun  Seasonal_cosine  Seasonal_sine
##           0.0077   0.0163           1.0555         1.7500
## s.e.      0.0053   0.0054           0.1873         0.1905
##
## sigma^2 = 0.2236:  log likelihood = -436.51
## AIC=897.02   AICc=897.5   BIC=950.91
##
## Training set error measures:
##                      ME      RMSE       MAE         MPE     MAPE      MASE
## Training set 0.01032617 0.4689192 0.3463915 -0.07391186 3.298813 0.3791426
##                     ACF1
## Training set -0.01798961
```

```r
#making a test series
xreg_test <- as.matrix(test_merged[, c("Lag_Mean_Temp", "Lag_Snow",
                                          "Lag_Precip", "Lag_Sun",
                                          "Seasonal_cosine", "Seasonal_sine")])

#forecasting using the ARIMAX
arimax_forecast <- forecast(arimax_model, xreg = xreg_test, h = nrow(test_merged$Avg_kWh))

#testing the accuracy and plotting the model
accuracy(arimax_forecast, test_merged$Avg_kWh)
```

```
##                       ME      RMSE       MAE         MPE     MAPE      MASE
## Training set  0.01032617 0.4689192 0.3463915 -0.07391186 3.298813 0.9266127
## Test set     -0.54179043 0.7904620 0.6510393 -5.01045461 6.028512 1.7415588
##                     ACF1
## Training set -0.01798961
## Test set              NA
```

```r
plot(arimax_forecast)
```



Making a lagged neural net model

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute
```

```
#making a model matrix
x_train_nn <- model.matrix( ~., data = train_merged, na.rm = TRUE)

#getting the mean for each column
x_mean <- apply(x_train_nn, MARGIN = 2, FUN = mean)
#getting the sd for each column
x_sd <- apply(x_train_nn, MARGIN = 2, FUN = sd)
#scaling the train data
x_train_nn <- scale(x_train_nn, center = x_mean, scale = x_sd)

#dropping intercept
x_train_nn <- x_train_nn[ , -1]

#dropping date
```

```
x_train_nn <- x_train_nn[ , -1]

x_train_nn <- cbind.data.frame(train_merged$Avg_kWh[-1], x_train_nn)

#renaming the dependent
colnames(x_train_nn)[1] <- 'Avg_kWh'

#passing the test data to a matrix
x_test_nn <- model.matrix( ~ ., data = test_merged, na.rm = TRUE)

#scaling the test data using the train mean and sd
x_test_nn <- scale(x_test_nn, center = x_mean, scale = x_sd)

#dropping the intercept
x_test_nn <- x_test_nn[ , -1]

#dropping the date
x_test_nn <- x_test_nn[ , -1]

#adding the dependent
x_test_nn <- cbind.data.frame(test_merged$Avg_kWh, x_test_nn)

#renaming the dependent
colnames(x_test_nn)[1] <- 'Avg_kWh'

#setting the random seed
set.seed(7)

#making the neural net model
nn1 <- neuralnet(Avg_kWh ~ Lag_Mean_Temp + Lag_Snow + Lag_Precip + Lag_Sun + Seasonal_cosine + Seasonal_
                 hidden = c(6, 6), #6 hidden units in 2 layers
                  data = x_train_nn, #using train data
                   linear.output = TRUE,
                 stepmax = 1e6)

#plotting the NN
plot(nn1, type = "best")

#making predictions
nn1_pred <- predict(nn1, newdata = x_test_nn, type = 'response')

#passing those predictions to an accuracy function
nn1_pred_numeric <- as.vector(nn1_pred)
accuracy(nn1_pred_numeric, test_merged$Avg_kWh)
```

```
##                  ME      RMSE       MAE       MPE     MAPE
## Test set -0.4806818 0.8851741 0.7340787 -4.512057 6.820228
```

```
nn_pred_df <- data.frame(Date = test_merged$date,
                         Avg_kWh = nn1_pred)

#plotting the neural network predictions
ggplot(graph_energy, aes(x = Date, y = Avg_kWh)) +
```

```r
  geom_line(color = "black") +
  geom_line(data = nn_pred_df, color = "red") +
  labs(title = "Neural Network Regression Prediction",
       y = "Avg Energy Consumption (kWh)",
       x = "" )  +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "bottom") +
  scale_x_date(date_breaks = "3 months", date_labels = "%b %Y")

#getting accuracy functions for the three models
accuracy(energy.gam.pred, test_merged$Avg_kWh)
```

```
##                     ME      RMSE       MAE       MPE     MAPE
## Test set -0.03629603 0.6448493 0.5231751 -0.318847 4.874108
```

```r
accuracy(energy.lr.pred, test_merged$Avg_kWh)
```

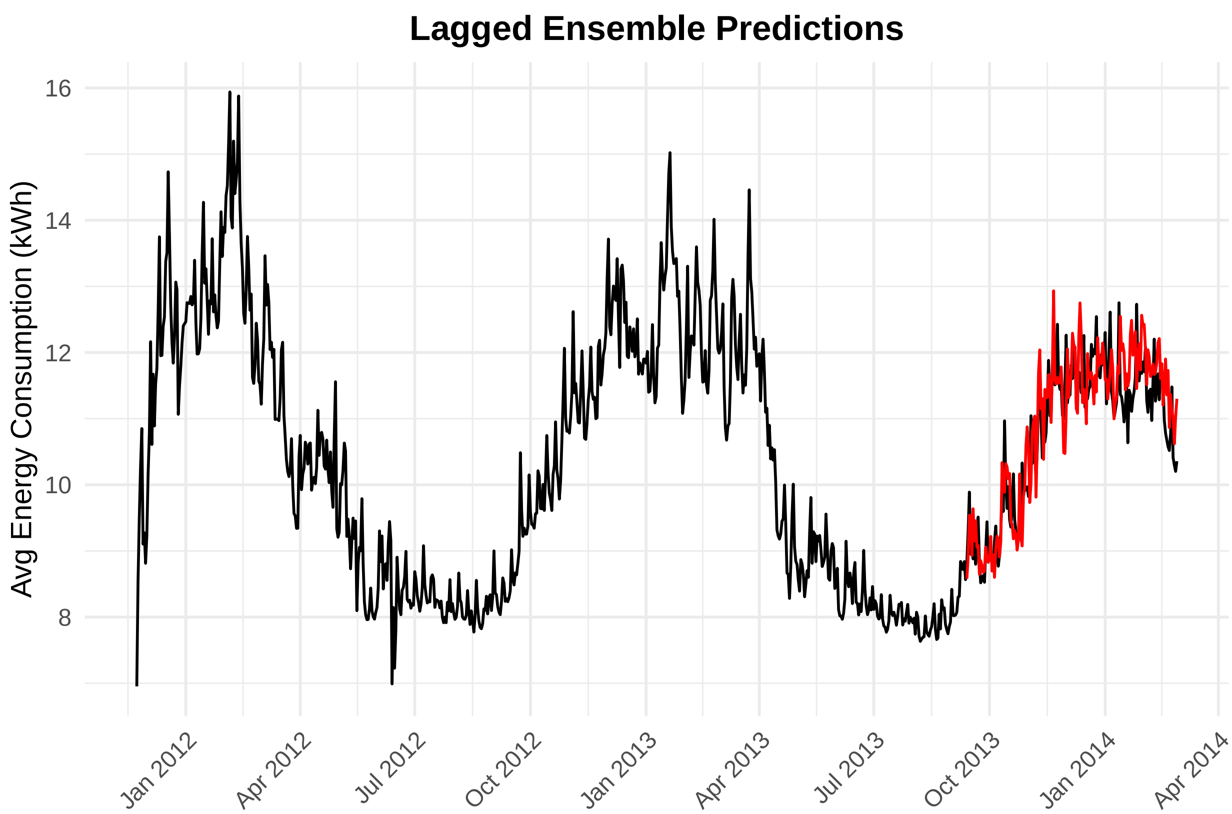
```
##                   ME      RMSE       MAE       MPE     MAPE
## Test set 0.0673378 0.6119364 0.4821305 0.4714703 4.415303
```

```r
accuracy(nn1_pred_numeric, test_merged$Avg_kWh)
```

```
##                    ME      RMSE       MAE       MPE     MAPE
## Test set -0.4806818 0.8851741 0.7340787 -4.512057 6.820228
```

Making the lagged ensemble model

```r
#making the regression average into a data frame
regression_ave_df <- data.frame(
  Avg_kWh = (nn_pred_df$Avg_kWh + gam_pred_df$Avg_kWh + lr_pred_df$Avg_kWh) / 3,
  Date = lr_pred_df$Date)

#plotting the regression average predictions
ggplot(graph_energy, aes(x = Date, y = Avg_kWh)) +
  geom_line(color = "black") +
  geom_line(data = regression_ave_df, color = "red") +
  labs(title = "Lagged Ensemble Predictions",
       y = "Avg Energy Consumption (kWh)",
       x = "" )  +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "bottom") +
  scale_x_date(date_breaks = "3 months", date_labels = "%b %Y")
```




```r
#getting accuracy of the regression model compared
accuracy(regression_ave_df$Avg_kWh, test_merged$Avg_kWh)
```

```
##                 ME      RMSE       MAE       MPE     MAPE
## Test set -0.14988 0.5743233 0.4622861 -1.453145 4.254878
```