Energy Price Final Forecasting

J Alexandra

2025-07-17

Libraries

```
suppressPackageStartupMessages({
library(dplyr)
library(tidyr)
library(fGarch)
library(gridExtra)
library(reshape2)
library(ggplot2)
library(ggpmisc)
library(tseries)
library(nortest)
library(zoo)
library(car)
library(lubridate)
library(purrr)
library(caret)
library(FinTS)
library(xts)
library(rugarch)
library(tibble)
library(forecast)})
```

Ensuring reproducibility of results:

```
set.seed(48)
```

Loading models and datasets

```
ARIMA_model1 <- readRDS("Models/Initial models from training/arima_model.rds")
SARIMAX_model2 <- readRDS("Models/Initial models from training/arima_with_xreg.rds")
SARIMAX_model1 <- readRDS("Models/Initial models from training/SARIMAX_model.rds")
GARCH_model <- readRDS("Models/Initial models from training/garch_model.rds")
SARIMA_model1 <- readRDS("Models/Initial models from training/sarima_model.rds")
STL_ARIMA_model1 <- readRDS("Models/Initial models from training/stl_arima_model.rds")
```

```
time_series_data <- read.csv("Saved_Datasets/timeseries.csv")
Training_set <- read.csv("Saved_Datasets/Training_set.csv")

Val_time_series <- read.csv("Saved_Datasets/Val_time_series.csv")
Val_set <- read.csv("Saved_Datasets/Val_set.csv")

xreg <- read.csv("Saved_Datasets/xreg.csv")
xreg_future <- read.csv("Saved_Datasets/xreg_future.csv")

xreg <- as.matrix(xreg)
xreg_future <- as.matrix(xreg_future)
```

```
time_series_data <- ts(time_series_data$Value, start = c(2017, 1), frequency = 48)
Val_time_series <- ts(Val_time_series$Value, start = c(2018, 1), frequency = 48)
First_order_diff <- diff(time_series_data)
```

```
csv_files_2017 <- list.files(path = "2017", pattern = "*.csv", full.names = TRUE) %>% lapply(read.csv)
csv_files_2018 <- list.files(path = "2018", pattern = "*.csv", full.names = TRUE) %>% lapply(read.csv)

df_2017_2018 <- rbind(csv_files_2017, csv_files_2018)
df_2017_2018 <- df_2017_2018[df_2017_2018$PointOfConnection == "ABY0111", ]
df_2017_2018 <- df_2017_2018[order(df_2017_2018$TradingDate, df_2017_2018$TradingPeriod), ]
```

Combining the training and test sets

I will be training the following models on both the 2017 and 2018 energy prices (my training set and validation set), then I will forecast the 2019 energy prices, and evaluating the models performance based on the actual 2019 energy prices (test set).

```
# Combining train and validation sets
train_val <- bind_rows(as_tibble(time_series_data), as_tibble(Val_time_series))
train_val <- ts(train_val, start = c(2017, 1), frequency = 48)

# box-cox transformation
lambda <- BoxCox.lambda(train_val)
ts_bc <- BoxCox(train_val, lambda)

# First order differencing:
train_val_diff <- diff(train_val)
```

Refitting models on combined dataset and forecasting:

ARIMA

```
ARIMA_model_Final_411 <- Arima(train_val, order = c(4,1,1))
```

I'm going to forecast 48 trading periods for 12 months

```r
forecast_ARIMA_411 <- forecast(ARIMA_model_Final_411, h = 17520)
forecast_ARIMA_411 <- ts(forecast_ARIMA_411$mean, start = c(2019, 1), frequency = 48)
```

```r
ARIMA_model_Final_312 <- Arima(train_val, order = c(3,1,2))
```

I'm going to forecast 48 trading periods for 12 months

```r
forecast_ARIMA_312 <- forecast(ARIMA_model_Final_312, h = 17520)
forecast_ARIMA_312 <- ts(forecast_ARIMA_312$mean, start = c(2019, 1), frequency = 48)
```

SARIMA

```r
SARIMA_model_Final <- Arima(train_val, order = c(3,1,1),
                            seasonal = list(order = c(0, 0, 1), period = 48))
```

I'm going to forecast 48 trading periods for 12 months

```r
forecast_SARIMA_1 <- forecast(SARIMA_model_Final, h = 17520)
forecast_SARIMA <- ts(forecast_SARIMA_1$mean, start = c(2019, 1), frequency = 48)
```

STL-ARIMA

```r
STL_ARIMA_FINAL <- stlm(train_val, s.window = "periodic", robust = TRUE,
                        modelfunction = function(x) Arima(x, order = c(5, 1, 2)))
```

I'm going to forecast 48 trading periods for 12 months

```r
forecast_STL_ARIMA_1 <- forecast(STL_ARIMA_FINAL, h = 17520)
forecast_STL_ARIMA <- ts(forecast_STL_ARIMA_1$mean, start = c(2019, 1), frequency = 48)
```

SARIMAX

Making the xreg for the 2017 and 2018 data:

```r
csv_files <- list.files(path = "Energy Generation", pattern = "*.csv", full.names = TRUE)
csv_files2018 <- list.files(path = "Energy Generation - 2018", pattern = "*.csv", full.names = TRUE)

Generation_2017 <- csv_files %>% lapply(read.csv) %>% bind_rows()
Generation_2018 <- csv_files2018 %>% lapply(read.csv) %>% bind_rows()
Generation <- rbind(Generation_2017, Generation_2018)
head(Generation)
```

```
##   Site_Code POC_Code Nwk_Code  Gen_Code Fuel_Code Tech_Code Trading_date   TP1
## 1       ARA  ARA2201     MRPL aratiatia     Hydro     Hydro   2017-01-01 11510
## 2       ARA  ARA2201     MRPL aratiatia     Hydro     Hydro   2017-01-02 11570
## 3       ARA  ARA2201     MRPL aratiatia     Hydro     Hydro   2017-01-03 11090
```

```
## 4       ARA  ARA2201     MRPL aratiatia     Hydro     Hydro   2017-01-04 13970
## 5       ARA  ARA2201     MRPL aratiatia     Hydro     Hydro   2017-01-05 19680
## 6       ARA  ARA2201     MRPL aratiatia     Hydro     Hydro   2017-01-06 27190
##     TP2   TP3   TP4   TP5   TP6   TP7   TP8   TP9  TP10  TP11  TP12  TP13  TP14
## 1 11480 11450 11500 11520 11530 11520 11530 11600 11620 11460 11500 11580 11600
## 2 11050 11060 11020 11000 11040 11060 11100 11070 11100 11140 11180 11170 11120
## 3 11100 11110 11120 11100 11150 11860 12770 13280 14420 14400 14420 14420 15530
## 4 13240 11070 11200 11140 11230 10900 10950 11180 11220 11230 11100 12800 25070
## 5 14640 14250 14250 14280 14300 14290 14300 14330 14340 14330 13960 21180 25510
## 6 27170 27130 17870 14730 14820 14850 14850 14840 14830 14840 14820 14830 14830
##    TP15  TP16  TP17  TP18  TP19  TP20  TP21  TP22  TP23  TP24  TP25  TP26  TP27
## 1 11600 11620 11620 11680 11580 11520 11420 11420 11340 11380 11300 11260 11350
## 2 11030 11180 10910 12770 14680 14490 13550 13470 13700 13820 13270 13260 13280
## 3 20380 23480 25260 25490 25960 26100 25760 25850 25870 25980 25690 25780 25870
## 4 20940 20510 20380 20430 27170 27040 26650 26640 26720 26790 26520 21370 21110
## 5 26640 27640 27590 27520 27460 27420 27110 27160 27200 27180 26840 26910 26920
## 6 19900 27630 27510 27400 27220 27090 26740 26730 25760 22330 22090 22280 22360
##    TP28  TP29  TP30  TP31  TP32  TP33  TP34  TP35  TP36  TP37  TP38  TP39  TP40
## 1 11310 11230 11220 11340 11310 11160 11200 11230 11280 11190 11330 11270 11350
## 2 14370 14540 14540 14590 14600 14440 14490 14550 14590 14610 14620 14630 14670
## 3 25860 25660 25720 25810 25830 25610 25860 26090 22840 21090 21220 20790 20620
## 4 21240 20890 21020 20700 21120 20780 20810 20810 20730 20590 18540 14270 14410
## 5 26910 26660 22180 22200 22420 22190 22290 22410 22490 22490 22660 22710 25130
## 6 22250 14260 14310 14310 14340 21140 21350 21280 21060 23430 24320 23330 23400
##    TP41  TP42  TP43  TP44  TP45  TP46  TP47  TP48 TP49 TP50
## 1 11340 11580 11580 11530 11420 11580 11570 11600   NA   NA
## 2 14670 14670 14670 13060 11220 10990 11090 11040   NA   NA
## 3 20540 20560 16150 13960 13970 13980 13940 13870   NA   NA
## 4 14670 17560 20560 20700 20920 20830 20910 20650   NA   NA
## 5 27270 27260 27260 27230 27260 27210 27230 27210   NA   NA
## 6 23540 23860 24250 23910 23430 23810 24130 20620   NA   NA
```

```r
Generation_Overall_Output <- Generation[, -c(56, 57)] %>%
  pivot_longer(cols = matches("^TP\\d+$"), names_to = "TradingPeriod",
               names_prefix = "TP", values_to = "Energy_output") %>%
  mutate(TradingPeriod = as.integer(TradingPeriod), Trading_date = as.Date(Trading_date)) %>%
  group_by(Trading_date, TradingPeriod) %>%
  summarise(Energy_output = sum(Energy_output, na.rm = TRUE), .groups = "drop")
```

```r
any(Generation_Overall_Output$Energy_output == 0)
```

```
## [1] TRUE
```

```r
which(Generation_Overall_Output$Energy_output == 0)
```

```
## [1] 12815 12816 30623 30624
```

There are four rows where there is no energy output.

This lack of energy output is due to daylight savings. There is one less hour in the day and as a result no energy output is recorded for the lost hour.

```r
Generation_Overall_Output[c(12815, 12816, 30623, 30624), ]
```

```
## # A tibble: 4 x 3
##   Trading_date TradingPeriod Energy_output
##   <date>               <int>         <dbl>
## 1 2017-09-24              47             0
## 2 2017-09-24              48             0
## 3 2018-09-30              47             0
## 4 2018-09-30              48             0
```

I'm going to use linear interpolation to fill the gap of energy output for these trading periods

```r
lin_interp <- function(Generation_Overall_Output, index_1, index_2){
i_prev <- which(Generation_Overall_Output$Trading_date == as.Date("2017-09-24") &
                Generation_Overall_Output$TradingPeriod == 46)

price_prev <- Generation_Overall_Output$Energy_output[i_prev]
price_next <- Generation_Overall_Output$Energy_output[i_prev + 1]

interp <- approx(x = c(46, 49), y = c(price_prev, price_next), xout = c(47, 48), method = "linear")

Generation_Overall_Output$Energy_output[index_1] <- interp$y[1]
Generation_Overall_Output$Energy_output[index_2] <- interp$y[2]

# Generation_Overall_Output[c(index_1, index_2), ]
return(Generation_Overall_Output)
}

Generation_Overall_Output <- lin_interp(Generation_Overall_Output, 12815, 12816)
Generation_Overall_Output <- lin_interp(Generation_Overall_Output, 30623, 30624)
```

Feature engineering:

```r
# Energy generation output
Energy_generation <- Generation_Overall_Output$Energy_output / 1e6

# Season:
# I'm using meteorological season start dates (which are the same every year)
# for simplicity and consistency
get_season <- function(date) {
  month_day <- format(as.Date(date), "%m-%d")
  if (month_day >= "12-01" || month_day < "03-01") {
    return("Summer")
  } else if (month_day >= "03-01" && month_day < "06-01") {
    return("Autumn")
  } else if (month_day >= "06-01" && month_day < "09-01") {
    return("Winter")
  } else {
    return("Spring")
  }
}
```

```r
seasons_df <- sapply(df_2017_2018$TradingDate, get_season)

Season <- data.frame(
  TradingPeriod = df_2017_2018$TradingDate,
  Season = seasons_df
)

# Converting seasons to dummy variables (with one-hot encoding)
Season_dummies <- model.matrix(~ Season, data = Season)[, -1]
# dropped 1 season (autumn) to avoid the dummy variable trap

# Weekend / Weekday variable: (1 is a weekend, 0 is a weekday)


Is_Weekend <- ifelse(weekdays(as.Date(df_2017_2018$TradingDate)) %in% c("Saturday", "Sunday"), 1, 0)

xreg <- cbind(Energy_generation, Season_dummies, Is_Weekend)
```

Note that energy generation was scaled down because if the regressors in xreg have very large values, it can cause numerical instability. These engineered variables make up my xreg matrix for SARIMAX.

```r
SARIMAX_FINAL <- Arima(train_val, order = c(2,1,2), seasonal = list(order = c(0, 0, 1), period = 48),
                      xreg = xreg)
```

```r
summary(SARIMAX_FINAL)
```

```
## Series: train_val
## Regression with ARIMA(2,1,2)(0,0,1)[48] errors
##
## Coefficients:
```

```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

```
##          ar1     ar2      ma1     ma2    sma1  Energy_generation  SeasonSpring
##       0.0536  0.4852  -0.2997  -0.655  0.1107            46.0325       -3.2375
## s.e.     NaN     NaN      NaN     NaN  0.0052             1.5695       15.4610
##       SeasonSummer  SeasonWinter  Is_Weekend
##            -4.8844        0.1069     -0.9759
## s.e.       13.3978       13.3922      1.5664
##
## sigma^2 = 1005:  log likelihood = -170820.6
## AIC=341663.2   AICc=341663.2   BIC=341756.3
##
## Training set error measures:
##                     ME     RMSE      MAE      MPE     MAPE    MASE        ACF1
## Training set 0.0139367 31.69562 12.11189 -137.471 155.5981 0.43767 -0.02340478
```

Forecasting the xreg for the 2019 data:

Estimating xreg_future Estimating the Energy generation values.

Model for forecasting Energy_generation values, note that a Box-Cox transformation has been applied to ensure all outputs are non-negative. This is done because a power plant cannot generate less than zero energy:

```
ts_Energy <- ts(Energy_generation, start = c(2017, 1), frequency = 48)
lambda <- BoxCox.lambda(ts_Energy)
Energy_generation_bc <- BoxCox(ts_Energy, lambda)

Energy_generation_model_2 <- Arima(Energy_generation_bc, order = c(3, 0, 1),
                                   seasonal = list(order = c(1, 1, 0), period = 48),
                                   include.drift = TRUE)

fcast_bc <- forecast(Energy_generation_model_2, h = 17520)
Energy_generation <- InvBoxCox(fcast_bc$mean, lambda)
```

```
# Encoding season and is_weekend for 2019 dates:

# Creating a full sequence of dates for 2019
dates <- seq.Date(as.Date("2019-01-01"), as.Date("2019-12-31"), by = "day")
# Repeating each date for the 48 trading periods
repeated_dates <- rep(dates, each = 48)

seasons <- sapply(repeated_dates, get_season)

Season <- data.frame(
  Date = repeated_dates,
  Season = seasons
)

# Converting seasons to dummy variables (with one-hot encoding)
Season_dummies <- model.matrix(~ Season, data = Season)[, -1]
# dropped 1 season (autumn) to avoid the dummy variable trap

Is_Weekend <- ifelse(weekdays(repeated_dates) %in% c("Saturday", "Sunday"), 1, 0)

# turn into matrix
xreg_future <- cbind(Energy_generation, Season_dummies, Is_Weekend)
colnames(xreg_future)[2] <- "SeasonSpring"
colnames(xreg_future)[3] <- "SeasonSummer"
colnames(xreg_future)[4] <- "SeasonWinter"
```

I'm going to forecast 48 trading periods for 12 months

```
forecast_SARIMAX_1 <- forecast(SARIMAX_FINAL, h = 17520, xreg = xreg_future)
forecast_SARIMAX <- ts(forecast_SARIMAX_1$mean, start = c(2019, 1), frequency = 48)
```

TBATS

Training a TBATS model on cleaned data (removing outliers)

```r
outliers  <- tsoutliers(train_val)
cleaned_ts <- replace(train_val, outliers$index, outliers$replacements)
TBATS_model <- tbats(cleaned_ts)
```

```r
show(TBATS_model)
```

```
## TBATS(0.524, {0,0}, 0.819, {<48,10>})
##
## Call: tbats(y = cleaned_ts)
##
## Parameters
##   Lambda: 0.523721
##   Alpha: 0.8301009
##   Beta: -0.1495079
##   Damping Parameter: 0.819016
##   Gamma-1 Values: 0.003325572
##   Gamma-2 Values: 0.0003835308
##
## Seed States:
##               [,1]
##  [1,] 11.661093964
##  [2,] -0.476167218
##  [3,] -0.709933698
##  [4,] -0.141080634
##  [5,]  0.223864009
##  [6,]  0.102968107
##  [7,] -0.082140941
##  [8,] -0.034971003
##  [9,] -0.010519532
## [10,]  0.049307880
## [11,]  0.039109115
## [12,] -0.080136298
## [13,] -0.377215729
## [14,] -0.567732719
## [15,]  0.071954876
## [16,]  0.106791236
## [17,]  0.009823883
## [18,]  0.079797183
## [19,]  0.037852045
## [20,] -0.016862304
## [21,] -0.041178497
## [22,]  0.003860543
## attr(,"lambda")
## [1] 0.5237214
##
## Sigma: 1.497597
## AIC: 535971.8
```

```r
forecast_TBATS_1 <- forecast(TBATS_model, h = 17520)
forecast_TBATS <- ts(forecast_TBATS_1$mean, start = c(2019, 1), frequency = 48)
```

Loading test data:

```r
csv_folder <- "2019"
csv_files <- list.files(path = csv_folder, pattern = "*.csv", full.names = TRUE)

test_data <- csv_files %>% lapply(read.csv) %>% bind_rows()
test_data <- test_data[test_data$PointOfConnection == "ABY0111", ]
test_data <- test_data[order(test_data$TradingDate, test_data$TradingPeriod), ]
test_data$TradingDate <- as.Date(test_data$TradingDate)
```

Because I removed daylight savings effected days from my training set then I am also going to remove these days from my test set. Daylight savings for 2019 starts on 2019-04-07

```r
ErrorIndices2 <- which(test_data$TradingPeriod > 48)
test_data <- test_data[-ErrorIndices2, ]
```

```r
test_data %>% mutate(test_data = as.Date(TradingDate)) %>% count(TradingDate) %>% filter(n < 48)
```

```
##   TradingDate  n
## 1  2019-09-29 46
```

I'm going to use linear interpolation to fill in the 2 trading period gap for 2019-09-29

```r
# finding the row just before period 47 on 2019-09-29
i_prev <- which(test_data$TradingDate == as.Date("2019-09-29") &
                test_data$TradingPeriod == 46)

# Computing the two interpolated values:
# Extracting the two known prices
price_prev <- test_data$DollarsPerMegawattHour[i_prev]
price_next <- test_data$DollarsPerMegawattHour[i_prev + 1]

# runing approx() over the X = {46,49} → Y = {prev,next}, get Y at Xout = {47,48}
interp <- approx(
  x    = c(46, 49),
  y    = c(price_prev, price_next),
  xout = c(47, 48),
  method = "linear"
)

# interp$x == c(47,48); interp$y == interpolated prices
interp
```

```
## $x
## [1] 47 48
##
## $y
## [1] 112.9367 104.0133
```

```r
new_rows <- tibble(
  TradingDate             = as.Date("2019-09-29"),
  TradingPeriod           = interp$x,
  PointOfConnection       = "ABY0111",
  DollarsPerMegawattHour  = interp$y
)

# bind back and re-sort
test_data <- bind_rows(test_data, new_rows) %>%
  arrange(TradingDate, TradingPeriod)

# view the gap now filled
filter(test_data,
       TradingDate == as.Date("2019-09-29"),
       TradingPeriod %in% 46:49)
```

```
##   TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 1  2019-09-29            46           ABY0111               121.8600
## 2  2019-09-29            47           ABY0111               112.9367
## 3  2019-09-29            48           ABY0111               104.0133
```

Making it a time series object:

```r
price_vector2 <- as.numeric(test_data$DollarsPerMegawattHour)
Test_time_series <- ts(price_vector2, start = c(2019, 1), frequency = 48)
```

Double checking the start date, frequency, and length of my time series object:

```r
start(Test_time_series)
```

```
## [1] 2019    1
```

```r
frequency(Test_time_series)
```

```
## [1] 48
```

```r
length(Test_time_series)
```

```
## [1] 17520
```

Overall Model Comparison

```r
ARIMA_312_acc <- accuracy(forecast_ARIMA_312, Test_time_series)
ARIMA_411_acc <- accuracy(forecast_ARIMA_411, Test_time_series)
SARIMA_acc <- accuracy(forecast_SARIMA, Test_time_series)
STL_ARIMA_acc <- accuracy(forecast_STL_ARIMA, Test_time_series)
SARIMAX_acc <- accuracy(forecast_SARIMAX, Test_time_series)
TBATS_acc <- accuracy(forecast_TBATS, Test_time_series)
```

```
acc_list <- list(ARIMA312 = ARIMA_312_acc, ARIMA411 = ARIMA_411_acc, SARIMA = SARIMA_acc,
                  STL_ARIMA = STL_ARIMA_acc, SARIMAX = SARIMAX_acc, TBATS = TBATS_acc)

acc_df <- map_df(acc_list, ~ as.data.frame(.x)["Test set", ], .id = "Model")
```

Manually calculating the MASE values:

```
MASE_cal <- function(forecast_1) {
# MAE of forecast errors
mae_model <- mean(abs(Test_time_series - forecast_1))

# Naive forecast (random walk with no drift)
naive_forecast <- naive(train_val, h = length(Test_time_series))
naive_forecast <- ts(naive_forecast$mean, start = c(2019, 1), frequency = 48)
mae_naive <- mean(abs(Test_time_series - naive_forecast))

# MASE = MAE(model) / MAE(naive)
MASE <- mae_model / mae_naive
return(MASE)
}
```

```
mase_SARIMA <- MASE_cal(forecast_SARIMA)
mase_ARIMA3 <- MASE_cal(forecast_ARIMA_312)
mase_ARIMA4 <- MASE_cal(forecast_ARIMA_411)
mase_STL <- MASE_cal(forecast_STL_ARIMA)
mase_SARIMAX <- MASE_cal(forecast_SARIMAX)
mase_TBATS <- MASE_cal(forecast_TBATS)
```

Manually Calculating the Theil's U

```
theils_u <- function(forecast_1){
# Calculating RMSE
rmse <- sqrt(mean( (forecast_1 - Test_time_series)^2 ))
# Calculating root mean squared actuals
rmsa <- sqrt(mean( Test_time_series^2 ))
# Calculating Theil's U
theilsU <- rmse / rmsa
return(theilsU)
}
```

```
Theil_SARIMA <- theils_u(forecast_SARIMA)
Theil_ARIMA3 <- theils_u(forecast_ARIMA_312)
Theil_ARIMA4 <- theils_u(forecast_ARIMA_411)
Theil_STL <- theils_u(forecast_STL_ARIMA)
Theil_SARIMAX <- theils_u(forecast_SARIMAX)
Theil_TBATS <- theils_u(forecast_TBATS)
```

```
acc_df <- acc_df %>% mutate(MASE = c(mase_ARIMA3, mase_ARIMA4, mase_SARIMA, mase_STL, mase_SARIMAX, mase
acc_df <- acc_df %>% mutate(`Theil's U` = c(Theil_ARIMA3, Theil_ARIMA4, Theil_SARIMA,
                                            Theil_STL, Theil_SARIMAX, Theil_TBATS))

acc_df %>% arrange(RMSE) %>% mutate(across(where(is.numeric), ~ round(.x, digits = 4)))
```

```
##                  Model      ME    RMSE     MAE  MPE MAPE   ACF1 Theil's U
## Test set...1   SARIMAX 11.2403 59.2407 40.4201 -Inf  Inf 0.8811    0.4783
## Test set...2     TBATS 11.8560 59.4417 40.0670 -Inf  Inf 0.8798    0.4799
## Test set...3 STL_ARIMA 13.7859 60.1209 40.4767 -Inf  Inf 0.8831    0.4854
## Test set...4    SARIMA 12.3399 60.5459 40.7679 -Inf  Inf 0.8848    0.4888
## Test set...5  ARIMA312 12.6308 60.6052 40.8170 -Inf  Inf 0.8848    0.4893
## Test set...6  ARIMA411 13.1930 60.7250 40.9186 -Inf  Inf 0.8848    0.4903
##                MASE
## Test set...1 0.7914
## Test set...2 0.7845
## Test set...3 0.7926
## Test set...4 0.7983
## Test set...5 0.7992
## Test set...6 0.8012
```

The SARIMAX model had the best ME, RMSE, and Theil's U.

The TBATS model had the best MAE, ACF1 and MASE.

The TBATS model had the lowest ACF1 which means that, compared to the other models, it captured the most autocorrelation present in the data. However since the TBATS models ACF1 is still much higher than 0, then there is still uncaptured autocorrelation in the data.

The MASE for all of the models is less than 1, which means that all of the models outperform a naive forecast.

The MASE for SARIMAX(2,1,2)(0,0,1)[48] is 0.79 which means that its errors are 21% smaller than a naive forecasts. With a Theil's U of 0.4783, the model's forecast error is less than half that of a naive model, indicating strong predictive power.

The SARIMAX models mean error was 11.24 which means on average its forecasts overestimate actual prices by about 11.24 dollars per megawatt hour.

The SARIMAX models root mean square error was 59.24 which indicates large forecast errors, since squared errors penalize larger errors more heavily this high RMSE value is probably due to how volatile the data is.

The SARIMAX models mean absolute error was 40.42 which means that forecasts were off by 40.42 units on average. This is 37.16% of the annual mean energy price, which was 108.749 in 2019. The mean absolute error is relatively high, even when considering how volatile the data is. There is a lot of room for improvement in the models accuracy.

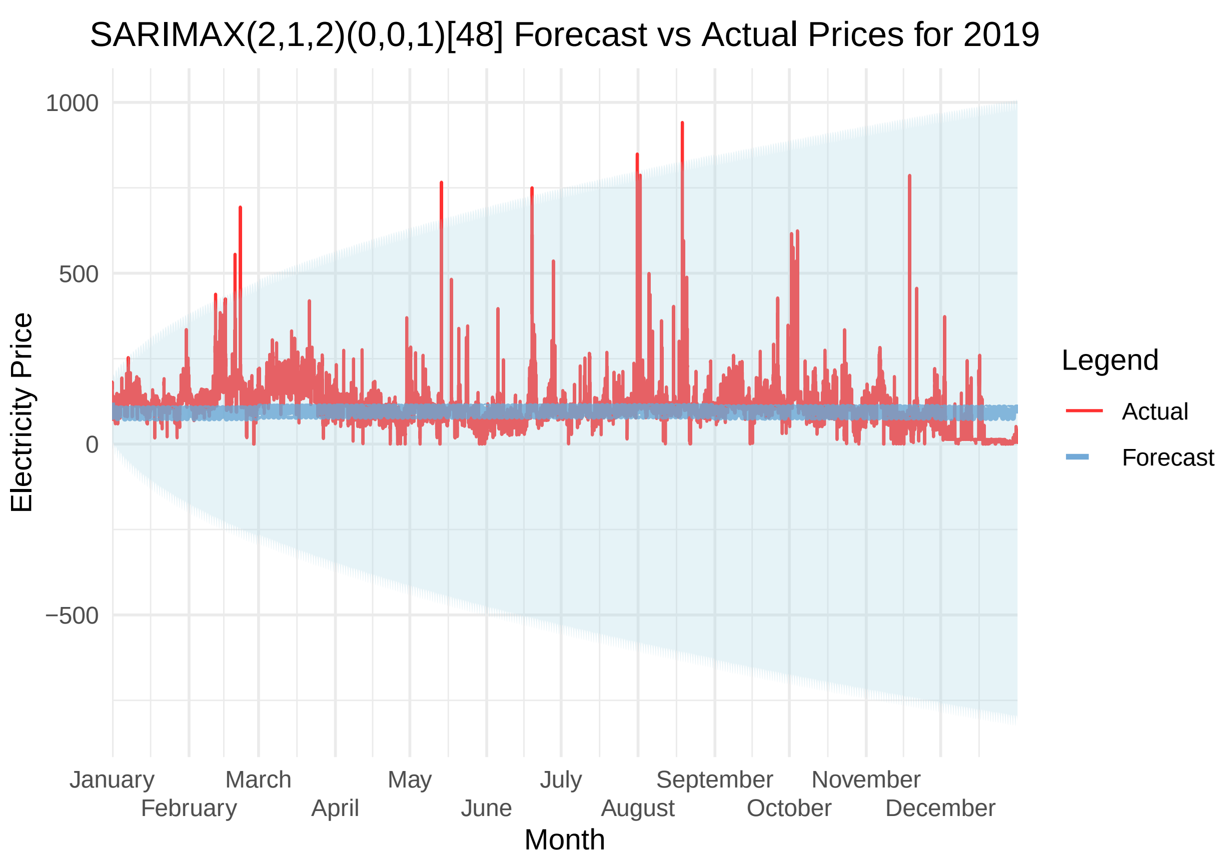
Forecast Plots

```r
plot_actual_forecast <- function(start_time, actual_vals, start_year, forecasts, model_name) {
start_time     <- as.POSIXct(start_time, tz = "UTC")
times_forecast <- seq(start_time, by = "30 min", length.out = 17520)

Test_time_series2 <- ts(actual_vals$DollarsPerMegawattHour, frequency = 48, start = c(start_year, 1))
df_actual <- data.frame(Date = times_forecast, Value = as.numeric(Test_time_series2), Type = "Actual")

df_forecast <- data.frame(
  Date  = times_forecast,
  Mean  = as.numeric(forecasts$mean),
  Lower = as.numeric(forecasts$lower[,2]),  # 95% lower bound
  Upper = as.numeric(forecasts$upper[,2]),  # 95% upper bound
```

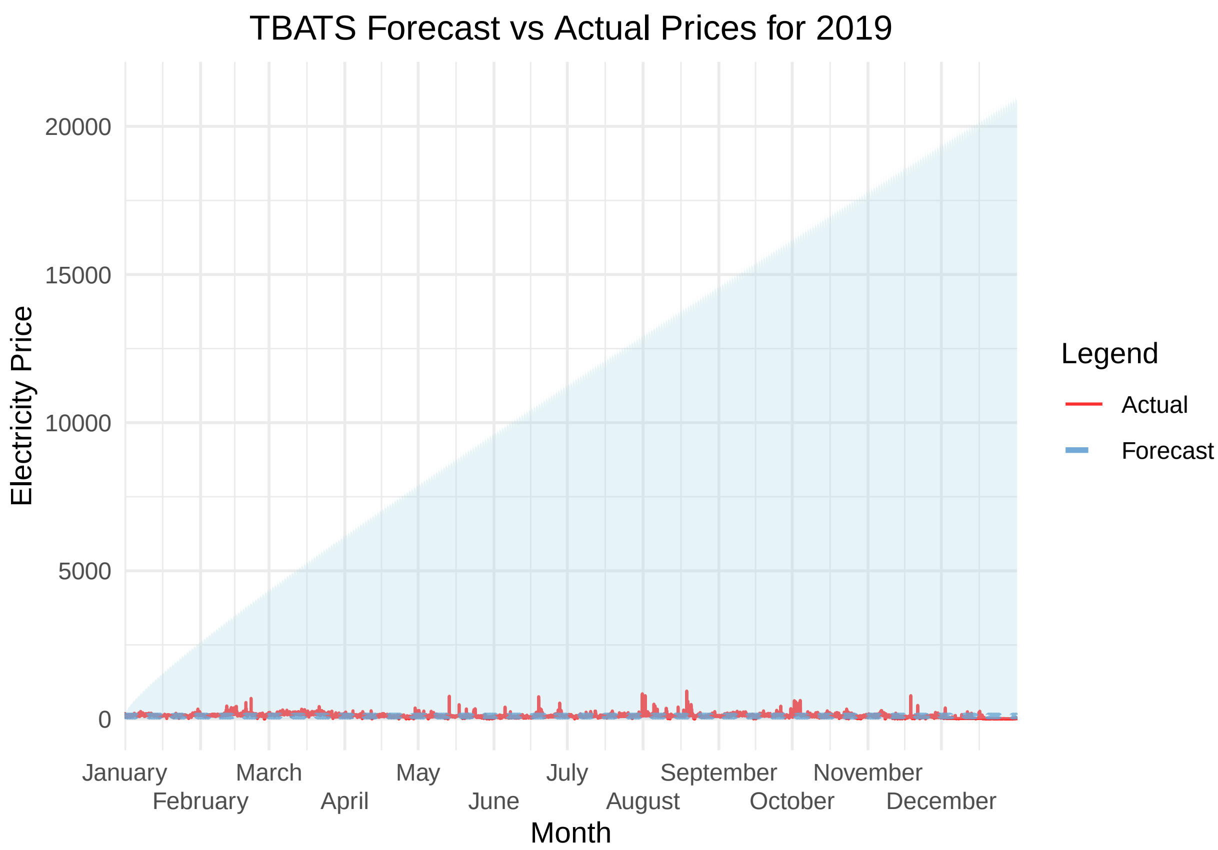
```
  Type  = "Forecast"
)

ggplot() +
  geom_line(data = df_actual, aes(x = Date, y = Value, color = "Actual"), size = 0.55) +
  geom_line(data = df_forecast, aes(x = Date, y = Mean, color = "Forecast"),
            size = 1, alpha = 0.8, linetype = "dashed") +
  geom_ribbon(data = df_forecast, aes(x = Date, ymin = Lower, ymax = Upper),
              fill = "lightblue", alpha = 0.3) +
  scale_color_manual(values = c("Actual" = "firebrick1", "Forecast" = "steelblue3")) +
  scale_x_datetime(date_breaks = "1 month",  date_labels = "%B",  expand = c(0, 0),
                   guide = guide_axis(n.dodge = 2)) +
  labs(title = paste(model_name, " Forecast vs Actual Prices for ", start_year, sep = ""),
       x = "Month", y = "Electricity Price", color = "Legend") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
}
```

```
plot_actual_forecast("2019-01-01 00:00:00", test_data, 2019, forecast_SARIMAX_1, "SARIMAX(2,1,2)(0,0,1)
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



```r
plot_actual_forecast("2019-01-01 00:00:00", test_data, 2019, forecast_TBATS_1, "TBATS")
```



Saving models

```r
saveRDS(ARIMA_model_Final_312, file = "Models/Final models/arima_312_model.rds")
saveRDS(ARIMA_model_Final_411, file = "Models/Final models/arima_412_model.rds")
saveRDS(SARIMA_model_Final, file = "Models/Final models/sarima_model.rds")
saveRDS(STL_ARIMA_FINAL, file = "Models/Final models/stl_arima_model.rds")
saveRDS(SARIMAX_FINAL, file = "Models/Final models/SARIMAX_model.rds")
saveRDS(TBATS_model, file = "Models/Final models/TBATS_model.rds")

# Saving SARIMAX with xreg (done because future xreg contains forecasted variable)
saveRDS(list(model = SARIMAX_FINAL, future_xreg = xreg_future), "Models/Final models/arima_with_xreg.rds
saveRDS(Energy_generation_model_2, file = "Models/Final models/Energy_generation_model.rds")
```

References and Citations

Electricity Authority. (n.d.). Final energy prices by month [Dataset]. EMI – Electricity Market Information. Retrieved between July 11 and July 15, 2025, from https://www.emi.ea.govt.nz/Wholesale/Datasets/DispatchAndPricing/FinalEnergyPrices/ByMonth

Electricity Authority. (n.d.). Generation output by plant [Dataset]. EMI – Electricity Market Information. Retrieved between July 18 and July 20, 2025, from https://www.emi.ea.govt.nz/Wholesale/Datasets/Generation/Generation_MD