Energy Price Forecasting

JAlexandra

2025-07-11

Purpose

Research question: Which time series model most accurately predicts 2019 energy prices, using historical energy price data from 2017 and 2018?

My hypothesis is that an ARIMA model will outperform the more complex models in predicting 2019 energy prices using 2017 and 2018 data.

Model performance is assessed using ME, RMSE, MAE, MPE, MAPE and MASE.

My training set is energy price data from 2017, my validation set is 2018 data and my test set is 2019 data.

After refining the models parameters using the 2018 validation set, the final models will be retrained on the combined the 2017 and 2018 datasets. The final models performance will be evaluated using 2019 data (the unseen test set).

Note that this project is only using one point of connection ABY0111 for simplicity, it was selected because it is the first point of connection alphabetically.

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 Data

```
csv_folder <- "2017"
csv_files <- list.files(path = csv_folder, pattern = "*.csv", full.names = TRUE)

training_set <- csv_files %>% lapply(read.csv) %>% bind_rows()
head(training_set)
```

```
##   TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 1  2017-01-01             1           ABY0111                  35.95
## 2  2017-01-01             1           ALB0331                  38.11
## 3  2017-01-01             1           ALB1101                  38.04
## 4  2017-01-01             1           APS0111                  35.37
## 5  2017-01-01             1           ARA2201                  34.82
## 6  2017-01-01             1           ARG1101                  37.44
```

This forecasting is going to be performed on one Point of Connection (ABY0111) for simplicity:

```
Training_set <- training_set[training_set$PointOfConnection == "ABY0111", ]
Training_set <- Training_set[order(Training_set$TradingDate, Training_set$TradingPeriod), ]
head(Training_set)
```

```
##      TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 1     2017-01-01             1           ABY0111                  35.95
## 244   2017-01-01             2           ABY0111                  41.65
## 487   2017-01-01             3           ABY0111                  26.58
## 730   2017-01-01             4           ABY0111                  26.78
## 973   2017-01-01             5           ABY0111                  26.66
## 1216  2017-01-01             6           ABY0111                  19.67
```

Data Cleaning and Wrangling

```
summary(Training_set)
```

```
##  TradingDate        TradingPeriod   PointOfConnection  DollarsPerMegawattHour
##  Length:17520       Min.   : 1.00   Length:17520       Min.   :  0.02
##  Class :character   1st Qu.:12.75   Class :character   1st Qu.: 47.17
##  Mode  :character   Median :24.50   Mode  :character   Median : 65.47
##                     Mean   :24.50                      Mean   : 78.56
##                     3rd Qu.:36.25                      3rd Qu.: 96.36
##                     Max.   :50.00                      Max.   :925.29
```

TradingDate has being incorrectly classed as a categorical variable instead of a date.

```
Training_set$TradingDate <- as.Date(Training_set$TradingDate)
```

```
summary(Training_set)
```

```
##   TradingDate         TradingPeriod   PointOfConnection  DollarsPerMegawattHour
##  Min.   :2017-01-01   Min.   : 1.00   Length:17520       Min.   :  0.02
##  1st Qu.:2017-04-02   1st Qu.:12.75   Class :character   1st Qu.: 47.17
##  Median :2017-07-02   Median :24.50   Mode  :character   Median : 65.47
##  Mean   :2017-07-01   Mean   :24.50                      Mean   : 78.56
##  3rd Qu.:2017-10-01   3rd Qu.:36.25                      3rd Qu.: 96.36
##  Max.   :2017-12-31   Max.   :50.00                      Max.   :925.29
```

There are four variables TradingDate, TradingPeriod, PointOfConnection, DollarsPerMegawattHour.

There are 17520 observations (rows).

The TradingDate variable is a date object ranging from 2017-01-01 to 2017-12-31.

The TradingPeriod variable gives the 30 minute interval where electricity was bought and sold. It is numerical.

The PointOfConnection variable is an ordinal categorical variable, it gives the grid location where electricty is entering (or exiting) the network. I'm only working with the ABY0111 point of connection.

The DollarsPerMegawattHour variable is numerical, it gives the wholesale price of electricity. It's the price at which electricity is bought and sold in the wholesale market at a specific date and time, and point of connection.

I can see that TradingPeriod has a max value of 50 which shouldn't be possible as there are only 48 trading periods in New Zealand's electricity market.

```
Training_set[Training_set$TradingPeriod > 48, ]
```

```
##         TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 1073089  2017-04-02            49           ABY0111                   5.81
## 1073332  2017-04-02            50           ABY0111                   2.05
```

There are 2 observations where the trading period is larger than 48, since NZ's electricity market has 48 trading periods a day (each representing a 30 minute interval) these observations are unusual. Both are on 2017-04-02 which is when daylight savings ended and clocks were turned back one hour, meaning there was an extra hour for that day, resulting in two more 30 minute trading periods

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

```
##   TradingDate  n
## 1  2017-09-24 46
```

I can see that on 2017-09-24 there were two less trading periods. This is also when daylight savings time starts for 2017.

I am going to remove the two extra trading periods for 2017-04-02 from the dataset as I want to maintain a consistent daily structure (fixed periodicity) for ARIMA:

```r
ErrorIndices <- which(Training_set$TradingPeriod > 48)
Training_set <- Training_set[-ErrorIndices, ]
```

I'm going to use linear interpolation to fill in the 2 trading period gap for 2017-09-24

```r
# finding the row just before period 47 on 2017-09-24
i_prev <- which(Training_set$TradingDate == as.Date("2017-09-24") &
                Training_set$TradingPeriod == 46)

# Computing the two interpolated values:
# Extracting the two known prices
price_prev <- Training_set$DollarsPerMegawattHour[i_prev]
price_next <- Training_set$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] 13.40667 15.90333
```

```r
new_rows <- tibble(
  TradingDate             = as.Date("2017-09-24"),
  TradingPeriod           = interp$x,
  PointOfConnection       = "ABY0111",
  DollarsPerMegawattHour  = interp$y
)

# bind back and resort
Training_set <- bind_rows(Training_set, new_rows) %>%
  arrange(TradingDate, TradingPeriod)

# view the gap now filled
filter(Training_set,
       TradingDate == as.Date("2017-09-24"),
       TradingPeriod %in% 46:49)
```

```
##   TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 1  2017-09-24            46           ABY0111               10.91000
## 2  2017-09-24            47           ABY0111               13.40667
## 3  2017-09-24            48           ABY0111               15.90333
```

```r
sapply(Training_set, anyNA)
```

```
##             TradingDate           TradingPeriod      PointOfConnection
##                   FALSE                   FALSE                   FALSE
## DollarsPerMegawattHour
##                   FALSE
```

There are no NA values in the data set for any of the variables.

Checking that the data contains all 365 days of the year:

```r
NROW(unique(Training_set$TradingDate))
```

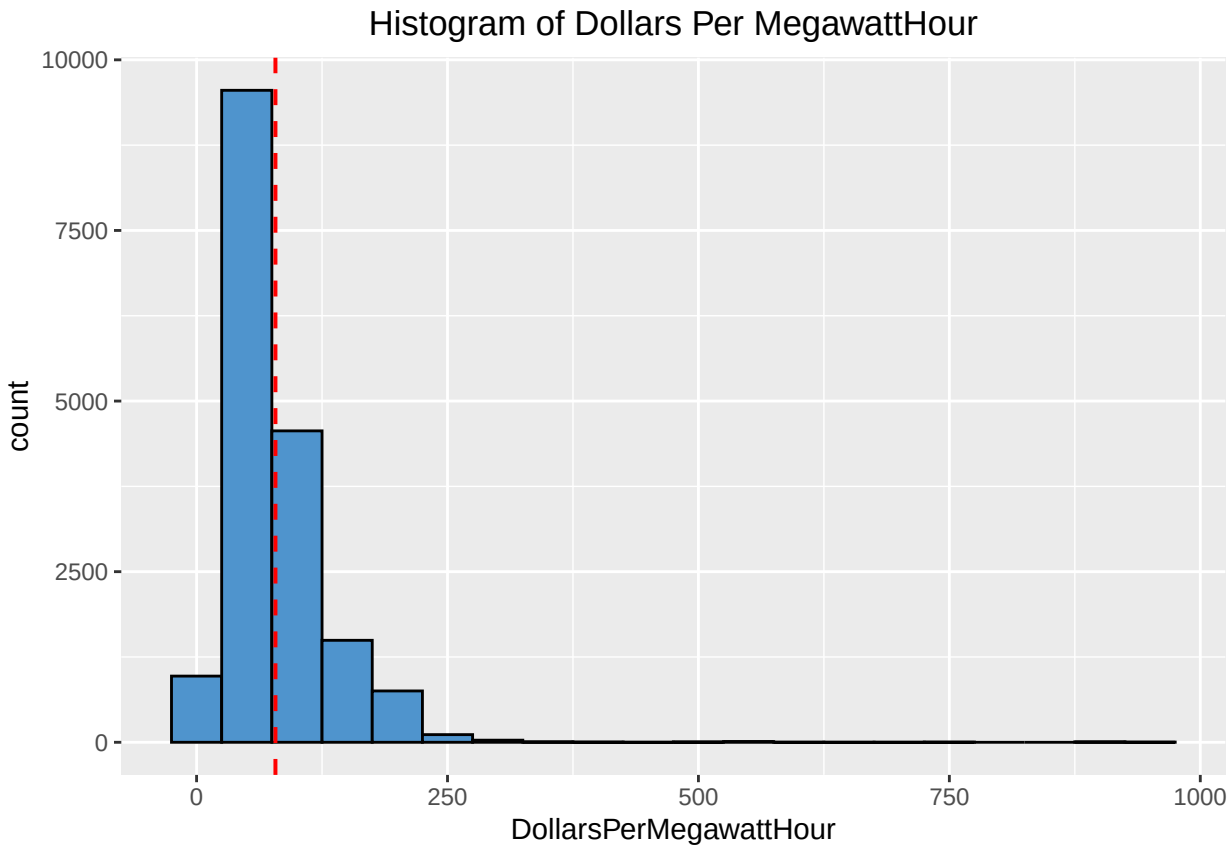
```
## [1] 365
```

There are 365 days like I expected

If there hadn't been (eg due to leap years), I would have dropped the extra day as ARIMA model which needs uniform seasonality

EDA

```r
ggplot(Training_set, aes(x = DollarsPerMegawattHour)) +
  geom_histogram(binwidth = 50, color = 'black', fill = 'steelblue3') +
  geom_vline(aes(xintercept = mean(DollarsPerMegawattHour)), color = "red",
             linetype = "dashed", linewidth = 0.7) +
  ggtitle("Histogram of Dollars Per MegawattHour") +
  theme(plot.title = element_text(hjust = 0.5))
```



There are very few values above 400 for DollarsPerMegawattHour.

The mode for DollarsPerMegawattHour is around 70 dollars.

There are no negative values for DollarsPerMegawattHour.

```
Training_set[Training_set$DollarsPerMegawattHour > 400,]
```

```
##       TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 6784   2017-05-22            16           ABY0111                 484.73
## 6785   2017-05-22            17           ABY0111                 449.12
## 9244   2017-07-12            28           ABY0111                 534.98
## 9245   2017-07-12            29           ABY0111                 532.88
## 9254   2017-07-12            38           ABY0111                 898.49
## 9256   2017-07-12            40           ABY0111                 501.21
## 9257   2017-07-12            41           ABY0111                 545.22
## 9259   2017-07-12            43           ABY0111                 534.55
## 9286   2017-07-13            22           ABY0111                 506.06
## 9287   2017-07-13            23           ABY0111                 673.43
## 9288   2017-07-13            24           ABY0111                 681.21
## 9289   2017-07-13            25           ABY0111                 509.21
## 9290   2017-07-13            26           ABY0111                 749.12
## 9291   2017-07-13            27           ABY0111                 749.40
## 9292   2017-07-13            28           ABY0111                 752.90
## 9293   2017-07-13            29           ABY0111                 902.01
## 9294   2017-07-13            30           ABY0111                 921.85
## 9295   2017-07-13            31           ABY0111                 925.29
```

```
## 9296   2017-07-13           32          ABY0111                925.29
## 9297   2017-07-13           33          ABY0111                925.21
## 9298   2017-07-13           34          ABY0111                618.93
## 9304   2017-07-13           40          ABY0111                901.95
## 9305   2017-07-13           41          ABY0111                902.35
## 9306   2017-07-13           42          ABY0111                901.84
## 9334   2017-07-14           22          ABY0111                538.99
## 9335   2017-07-14           23          ABY0111                598.66
## 9336   2017-07-14           24          ABY0111                557.35
## 9338   2017-07-14           26          ABY0111                545.40
## 9339   2017-07-14           27          ABY0111                544.52
## 9340   2017-07-14           28          ABY0111                539.85
## 9341   2017-07-14           29          ABY0111                538.32
## 9344   2017-07-14           32          ABY0111                482.39
## 10192  2017-08-01           16          ABY0111                521.17
## 10360  2017-08-04           40          ABY0111                901.24
```

There are 34 observations where the DollarsPerMegawattHour value was larger than the 400, which in a data set of 17520 observations is very few.

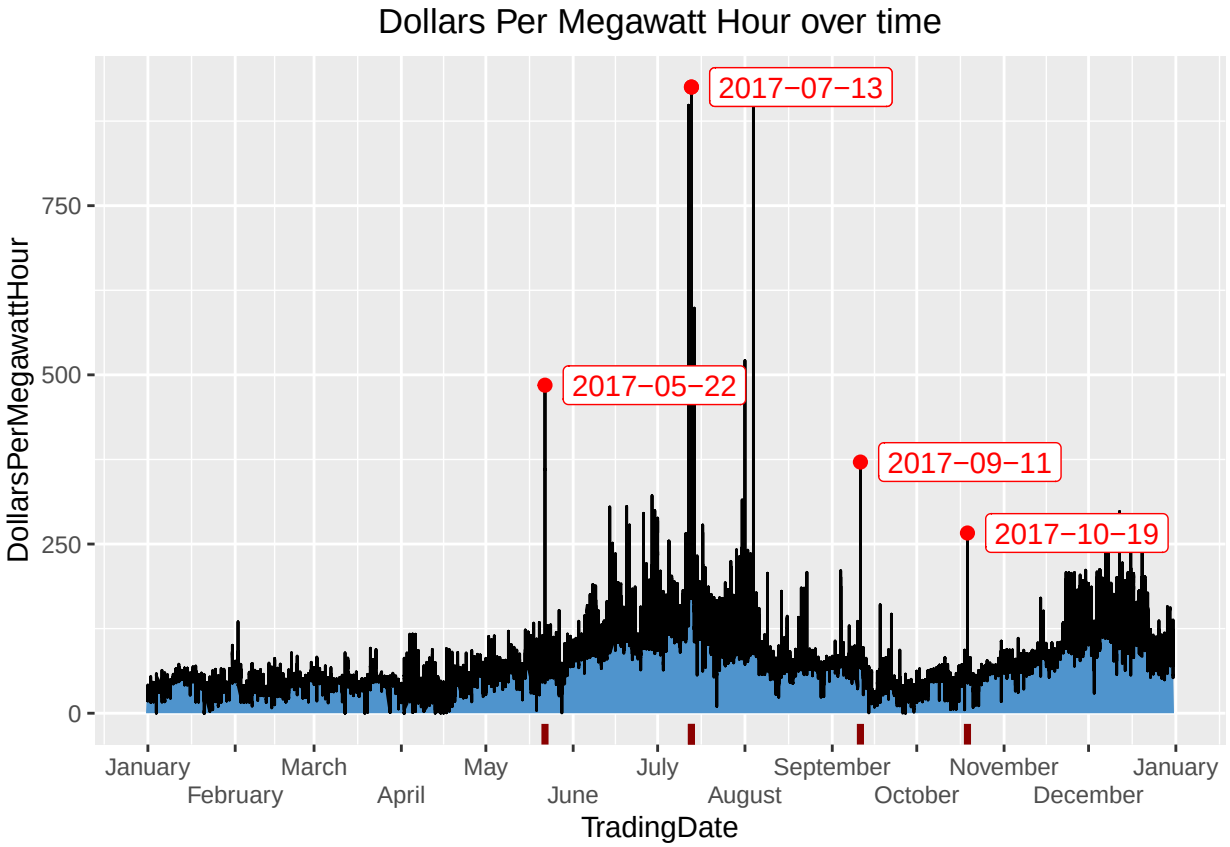
```
mean(Training_set$DollarsPerMegawattHour)
```

```
## [1] 78.56288
```

The mean DollarsPerMegawattHour for year 2017 at point connection ABY0111 is 78.71 (round to 2.dp)

Time series plot:

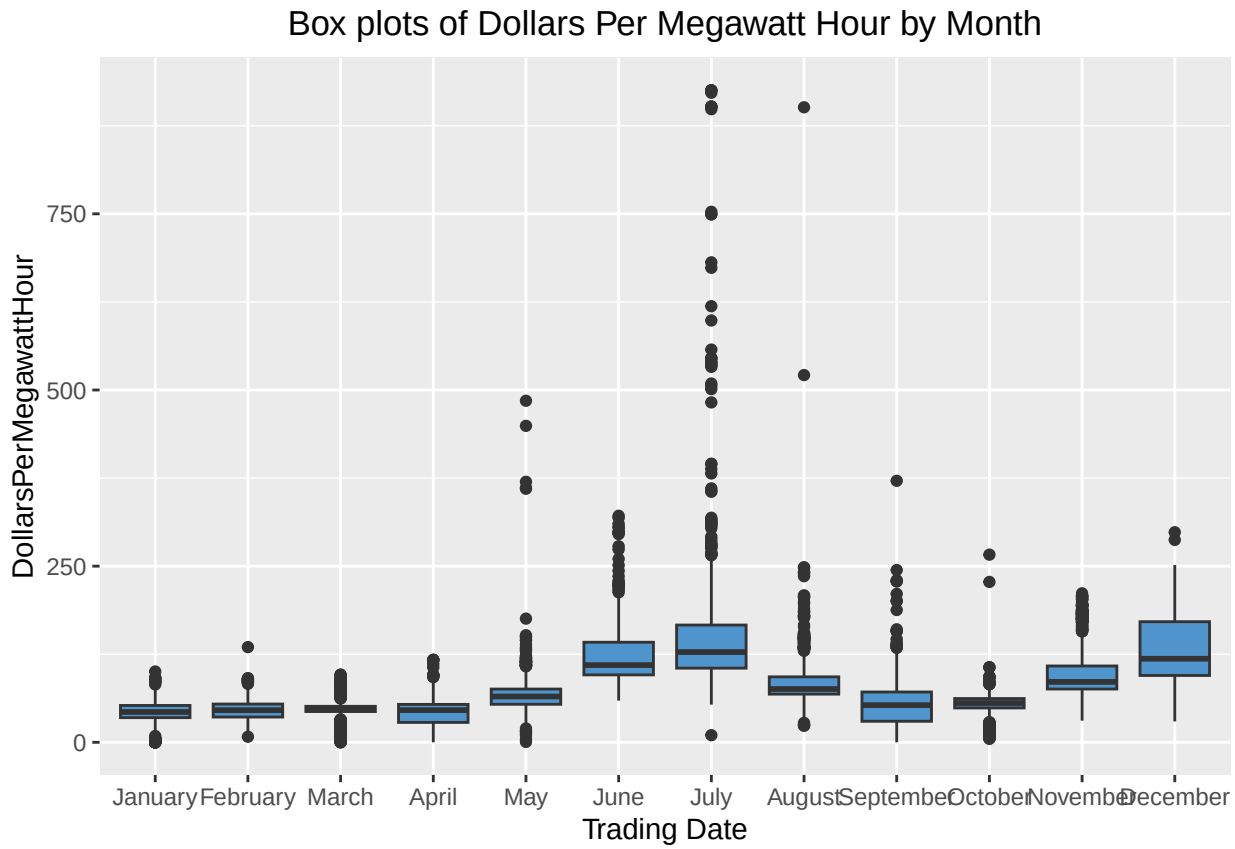
```
ggplot(Training_set, aes(x = TradingDate, y = DollarsPerMegawattHour)) +
  geom_area(fill = "steelblue3")+
  geom_line() +
  scale_x_date(date_breaks = "1 month", date_labels = "%B", guide = guide_axis(n.dodge = 2)) +
  stat_peaks(geom = "point", span = 3225, color = "red", size = 2) +
  stat_peaks(geom = "label", span = 3225, color = "red", angle = 0,
             hjust = -0.1, x.label.fmt = "%Y-%m-%d") +
  stat_peaks(geom = "rug", span = 3225, color = "darkred", sides = "b", size = 1.25) +
  ggtitle("Dollars Per Megawatt Hour over time") +
  theme(plot.title = element_text(hjust = 0.5))
```



I can see that there are specific dates where the whole sale price of electricity spiked to unusually high amounts.

There were spikes in electricity price on 2027-05-22, 2017-07-13, 2017-09-11, and 2017-10-19.

```
ggplot(Training_set, aes(x = format(TradingDate, "%m"), y = DollarsPerMegawattHour)) +
  geom_boxplot(fill = "steelblue3") +
  xlab("Trading Date") + scale_x_discrete(labels = month.name) +
  ggtitle("Box plots of Dollars Per Megawatt Hour by Month") +
  theme(plot.title = element_text(hjust = 0.5))
```



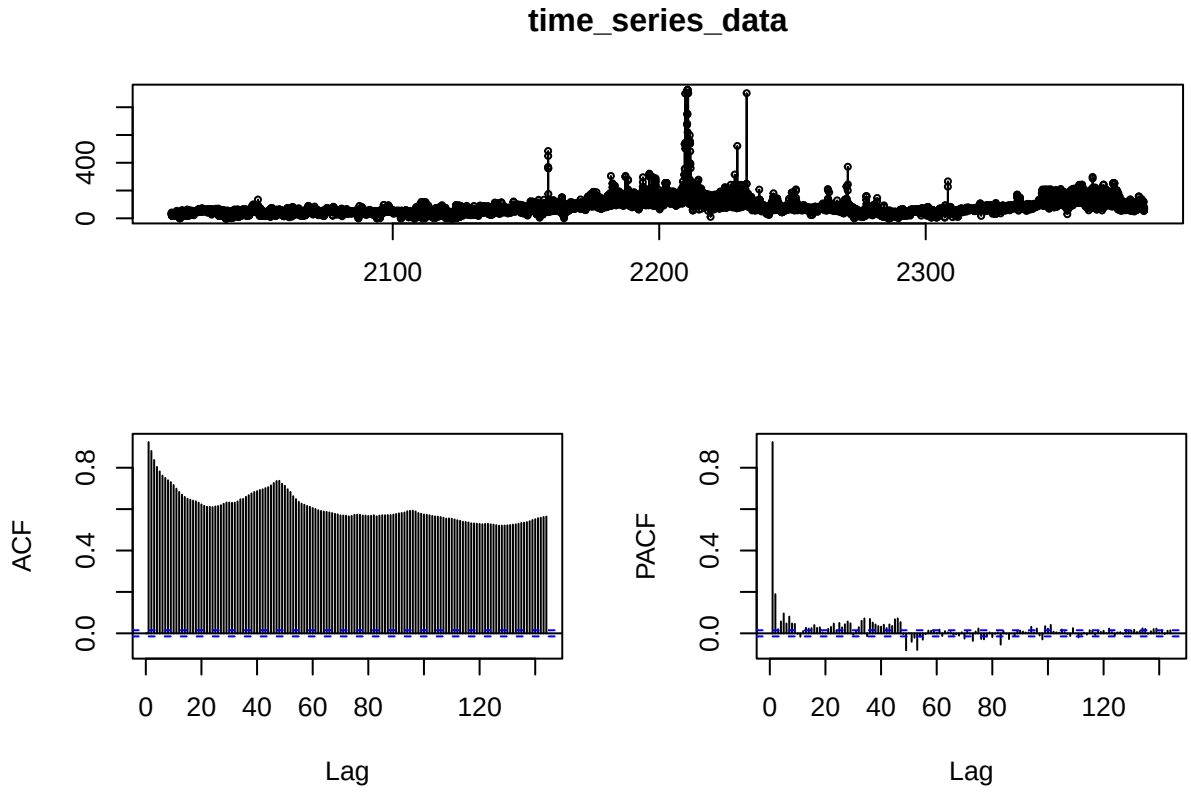
The price of electricity is much higher for June and July. The lower quartile of June and July doesn't even overlap with the upper quartiles of the previous months.

The prices dip again for August, September and October but then start increasing again in November and December.

Autocorrelation & Partial Autocorrelation Analysis

```
price_vector <- as.numeric(Training_set$DollarsPerMegawattHour)
# frequency is 48 because I have half hourly data
time_series_data <- ts(price_vector, start = c(2017, 1), frequency = 48)
```

```
tsdisplay(time_series_data)
```



ACF:

I can see that all of the spikes in the ACF plot are outside of the blue dashed significance bound this means that the time series data is highly autocorrelated and non-stationary. The autocorrelation at all of the lags is statistically meaningful and not just white noise. Past values have a strong influence on future ones across many lags.

There is particularly high autocorrelation at the early lags. This suggests that recent values strongly influence near future behavior, implying short term memory or an autoregressive structure.

There is a spike at around every 48 lags which suggests a seasonal pattern.

The bars gradually decay suggesting a persistent trend or that it's non-stationary. This suggests the mean and variance may be changing over time, and the structure could be due to an autoregressive process or an underlying seasonal component.

PACF:

There is a large spike at lag 1 in the PACF plot suggesting that the current price is heavily influenced by the immediate past period. This spike at lag 1 suggests an AR(1) structure. This supports the idea that the series has short-term autoregressive structure.

Past lag 1 there is gradual tapering, rather than a sharp cutoff like a pure AR(p) process.

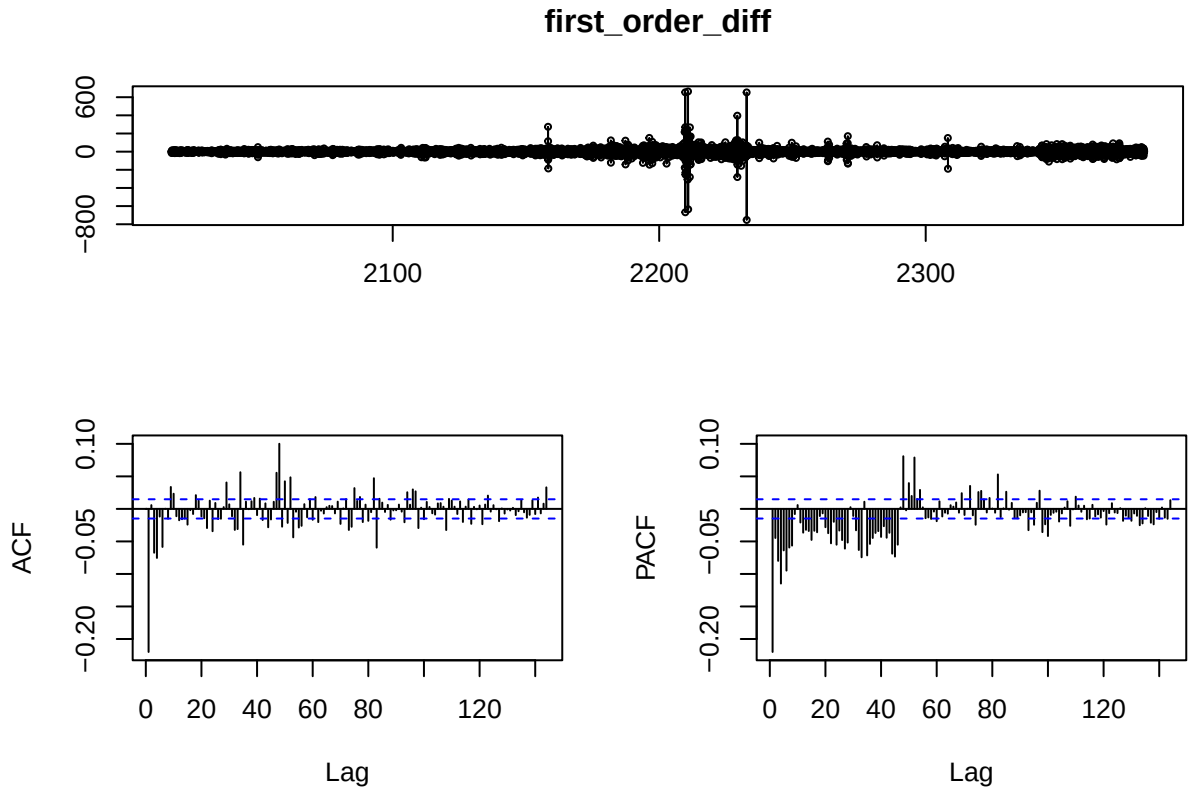
Conclusion:

All of the above information from the ACF and PACF plots suggests that the data is non-stationary and that there could be a seasonal component to the data.

I am going to apply first order differencing to the data to make the data stationary, as a stationary time series is a requirement of ARIMA.

```r
first_order_diff <- diff(time_series_data)
```

```r
tsdisplay(first_order_diff)
```



ACF:

After applying first order differencing Most lags in the ACF plot are within the confidence bounds, meaning there is no strong autocorrelation.

There is no sharp cutoff or pattern in the plot. There are no clear periodic spikes (e.g. at lag 48), which indicates that there is no seasonality.

The ACF plot no longer shows a slow decay, instead there are isolated significant spikes in the plot. This pattern indicates that the trend has been removed and that the series is now stationary, with stable mean and variance over time.

PACF:

There is a moderate spike at lag 1, outside the confidence bounds. Most subsequent lags are within the confidence bounds. This suggests a short term autoregressive pattern. This means each electricity price is influenced by its immediate predecessor.

The decay after the first lag also supports that the first order differencing has stabilized the series, leaving behind no long range autocorrelation.

Since this resembles an AR(1) structure still, when I fit the ARIMA model I will include an AR(1) component to model the autoregressive behavior.

Stationarity Diagnostics: ADF, KPSS & Phillips–Perron

A stationary time series is an assumption of ARIMA models, I'm going to check that the time series is stationary (eg. the time series mean, variance and autocorrelation structure are constant over time).

Augmented Dickey-Fuller test:

Null hypothesis: has a unit root (non stationary)

Alternative hypothesis: doesn't have a unit root (stationary)

```
adf_result <- adf.test(first_order_diff)
```

```
## Warning in adf.test(first_order_diff): p-value smaller than printed p-value
```

```
print(adf_result)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  first_order_diff
## Dickey-Fuller = -39.22, Lag order = 25, p-value = 0.01
## alternative hypothesis: stationary
```

The p-value is less than the significance level of 0.5, I reject the null hypothesis and conclude that the differenced series is stationary.

KPSS test:

Null hypothesis: the time series is trend stationary

Alternative hypothesis: the time series is not trend stationary (contains a unit root)

```
kpss_result <- kpss.test(first_order_diff, null = "Trend")
```

```
## Warning in kpss.test(first_order_diff, null = "Trend"): p-value greater than
## printed p-value
```

```
print(kpss_result)
```

```
##
##  KPSS Test for Trend Stationarity
##
## data:  first_order_diff
## KPSS Trend = 0.0010826, Truncation lag parameter = 14, p-value = 0.1
```

The p-value is larger than the significance level of 0.5 which means I fail to reject the null hypothesis and conclude that the differenced series is stationary.

Phillips–Perron test:

Null hypothesis: has a unit root (non stationary)

Alternative hypothesis: doesn't have a unit root (stationary)

```
Phillips <- pp.test(first_order_diff)
```

```
## Warning in pp.test(first_order_diff): p-value smaller than printed p-value
```

```
print(Phillips)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  first_order_diff
## Dickey-Fuller Z(alpha) = -16746, Truncation lag parameter = 14, p-value
## = 0.01
## alternative hypothesis: stationary
```

The p-value is less than the significance level of 0.5, I reject the null hypothesis and conclude that the differenced series is stationary.

The ADF, KPSS and Phillips–Perron tests all concluded that the first order differenced series is stationary.

Fitting models

ARIMA

I'm using auto.arima with d = 1 (first order differencing) to find a suitable ARIMA model that I can manually adjust later based on its residuals.

```
ARIMA_model1 <- auto.arima(time_series_data, stepwise = FALSE, approximation = TRUE,
                           d = 1, max.p = 3, max.q = 3, max.P = 2, max.Q = 2, seasonal = FALSE)
```

```
summary(ARIMA_model1)
```

```
## Series: time_series_data
## ARIMA(3,1,1)
##
## Coefficients:
##          ar1     ar2      ar3      ma1
##       0.6831  0.1472  -0.0395  -0.9765
## s.e.  0.0079  0.0091   0.0077   0.0023
##
## sigma^2 = 387.1:  log likelihood = -77052.94
## AIC=154115.9   AICc=154115.9   BIC=154154.7
##
## Training set error measures:
##                       ME    RMSE      MAE       MPE    MAPE      MASE
## Training set 0.02744566 19.6731 8.577677 -79.28617 87.9714 0.4398631
##                       ACF1
## Training set -0.0001270427
```

My current model is ARIMA(3,1,1)

I have an autoregressive component of order 3, first order differencing applied to the time series, and a moving average component of 1.

SARIMA

I'm using auto.arima with d = 1 (first order differencing) to find a suitable SARIMA model that I can manually adjust later based on its residuals.

```
SARIMA_model1 <- auto.arima(time_series_data, stepwise = FALSE, approximation = TRUE,
                            d = 1, max.p = 3, max.q = 3, max.P = 2, max.Q = 2, seasonal = TRUE)
```

```
summary(SARIMA_model1)
```

```
## Series: time_series_data
## ARIMA(3,1,1)(0,0,1)[48]
##
## Coefficients:
##          ar1     ar2      ar3      ma1    sma1
##       0.6684  0.1445  -0.0312  -0.9769  0.1270
## s.e.  0.0080  0.0091   0.0078   0.0025  0.0074
##
## sigma^2 = 380.9:  log likelihood = -76909.32
## AIC=153830.6   AICc=153830.6   BIC=153877.3
##
## Training set error measures:
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.02477343 19.51207 8.525955 -80.45479 89.09627 0.4372108
##                       ACF1
## Training set -0.0001153311
```

My current model is ARIMA(3,1,1)(0,0,1)[48]

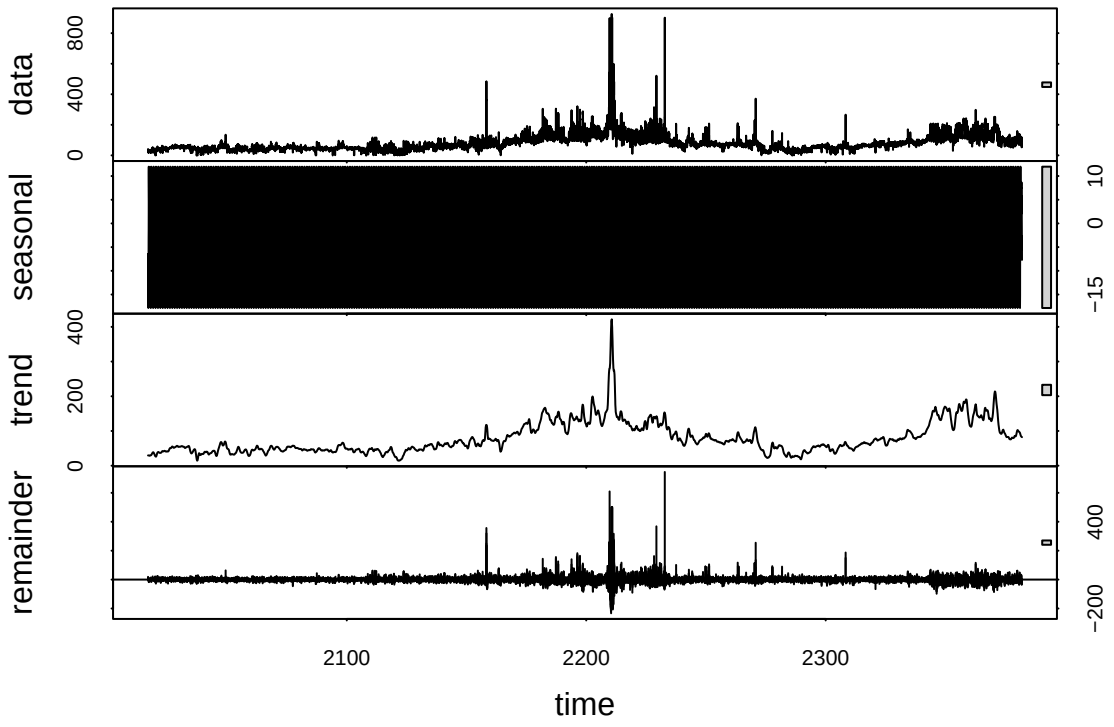
I have a non seasonal AR(3), first order differencing, MA(1). There is no seasonal AR or differencing. There is a seasonal MA(1) and a seasonal period of 48

STL-ARIMA

I've chosen to fit an STL-ARIMA model because it could potentially handle the seasonality present in the time series better than the ARIMA model.

Decomposing the series and visualizing the decomposition:

```
decomp <- stl(time_series_data, s.window = "periodic")
plot(decomp)
```



The first panel is a time series plot of the 2017 energy price data.

The second panel (seasonal component) shows consistent patterns that repeat over fixed intervals. There is seasonal component to the data.

The third panel (Trend component) shows the overall direction of the data, the middle part of the plot shows a large increase in energy prices, this dips and then rises again at the end of the plot.

The fourth panel (Remainder) shows whats leftover after removing the seasonal and trend effects, it contains outliers and volatility that is not explained by other components.

I've enabled robust fitting for the model (robust = TRUE) to make it more resistant to outliers. The weighting function will reduce the influence of outliers, causing most outliers to be captured in the remainder component. As a result, the trend and seasonal patterns will be preserved and will more accurately reflect the underlying structure of the time series.

Fitting an STL-ARIMA to seasonally adjusted series:

```
STL_ARIMA_model1 <- stlm(time_series_data, method = "arima", robust = TRUE, s.window = "periodic")
```

```
summary(STL_ARIMA_model1$model)
```

```
## Series: x
## ARIMA(5,1,1)
##
## Coefficients:
##          ar1     ar2      ar3      ar4     ar5      ma1
##       0.6755  0.1464  -0.0428  -0.0325  0.0497  -0.9772
```

```
## s.e.  0.0080  0.0091   0.0092   0.0091  0.0078   0.0025
##
## sigma^2 = 379.8:  log likelihood = -76883.49
## AIC=153781   AICc=153781   BIC=153835.4
##
## Training set error measures:
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.02607221 19.48375 8.445554 -2.423972 13.98295 0.4330878
##                       ACF1
## Training set -0.0004486496
```

My STL-ARIMA model is: STL-ARIMA(5,1,1)

I have an autoregressive component of order 5, first order differencing has been applied to the time series and there is a moving average component of order 1.

ARIMAX

I've chosen to fit an ARIMAX model to the energy price data because I want to see if incorporating external information will improve the models forecasting accuracy.

Feature engineering

On the EMI site, I found energy generation information. My understanding is that if the supply of energy (energy generation output) decreases, then the price of energy will increase, so energy generation information could be useful for predicting energy prices.

This generation output information is from the EMI website:

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

Generation <- csv_files %>% lapply(read.csv) %>% bind_rows()
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
```

I am going to use the total energy output for each date and trading period as an exogenous variable.

```
any(Generation[8:57] == 0)
```

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

There are some trading periods for each date and plant where the energy output is 0, this could be due to maintenance being performed on the plant at those specific times, or due to daylight savings.

```
which(sapply(Generation, anyNA), TRUE)[]
```

```
## TP47 TP48 TP49 TP50
##   54   55   56   57
```

There are some NA values for the 47th, 48th, 49th and 50th Trading periods.

These NA values will be due to daylight savings.

There are 48 trading periods in New Zealand's electricity market, but 49th and 50th trading periods have been recorded in this dataset because of daylight savings. Since not every day is affected by daylight savings (resulting in there being 1 hour more or less in the day) then not every day has values for the 49th and 50th trading periods.

I'm going to drop the 49th and 50th trading periods from the data.

```
Generation_Overall_Output <- Generation[, -c(56, 57)] %>%
  pivot_longer(
    cols = matches("^TP\\d+$"), # gets all the trading period columns (TP1 to TP50)
    names_to = "TradingPeriod", # makes a new column containing the old period column names
    names_prefix = "TP", # removes the "TP" prefix from all rows in the TradingPeriod column
    values_to = "Energy_output" # makes a new column for site energy output
  ) %>%
  mutate(
    TradingPeriod = as.integer(TradingPeriod),  # converting the TradingPeriod values to integers
    Trading_date = as.Date(Trading_date)  # converting Trading_date into a date object
  ) %>%
```

```
  group_by(Trading_date, TradingPeriod) %>%
  # Treats each unique combination of trading date and period as its own group
  summarise(
    Energy_output = sum(Energy_output, na.rm = TRUE),
  # calculates the total energy output for (that trading date and period)
  # each group by summing the energy output values across all sites
    .groups = "drop"
  )

head(Generation_Overall_Output)
```

```
## # A tibble: 6 x 3
##   Trading_date TradingPeriod Energy_output
##   <date>               <int>         <dbl>
## 1 2017-01-01               1      1788059.
## 2 2017-01-01               2      1738803.
## 3 2017-01-01               3      1689687.
## 4 2017-01-01               4      1641789.
## 5 2017-01-01               5      1604032.
## 6 2017-01-01               6      1568819.
```

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

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

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

```
## [1] 12815 12816
```

There are two rows where there is no energy output.

```
Generation_Overall_Output[c(12815, 12816), ]
```

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

This lack of energy output is due to daylight savings. There is one less hour in the day on 2017-09-24 and as a result no energy output is recorded for the lost hour.

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

```
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]
```

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

Generation_Overall_Output$Energy_output[12815] <- interp$y[1]
Generation_Overall_Output$Energy_output[12816] <- interp$y[2]

Generation_Overall_Output[c(12815, 12816), ]
```

```
## # A tibble: 2 x 3
##   Trading_date TradingPeriod Energy_output
##   <date>               <int>         <dbl>
## 1 2017-09-24              47      1222843.
## 2 2017-09-24              48       611421.
```

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
Season <- data.frame(Date = Training_set$TradingDate)
Season$Season <- rep("", nrow(Training_set))
Season$Season[
  which(Season$Date >= as.Date("2017-03-01") & Season$Date <= as.Date("2017-05-31"))
] <- "Autumn"

Season$Season[
  which(Season$Date >= as.Date("2017-06-01") & Season$Date <= as.Date("2017-08-31"))
] <- "Winter"

Season$Season[
  which(Season$Date >= as.Date("2017-09-01") & Season$Date <= as.Date("2017-11-30"))
] <- "Spring"

Season$Season[
  which(Season$Date >= as.Date("2017-12-01") & Season$Date <= as.Date("2017-12-31"))
] <- "Summer"
Season$Season[
  which(Season$Date >= as.Date("2017-01-01") & Season$Date <= as.Date("2017-02-28"))
] <- "Summer"

# 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(Training_set$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.

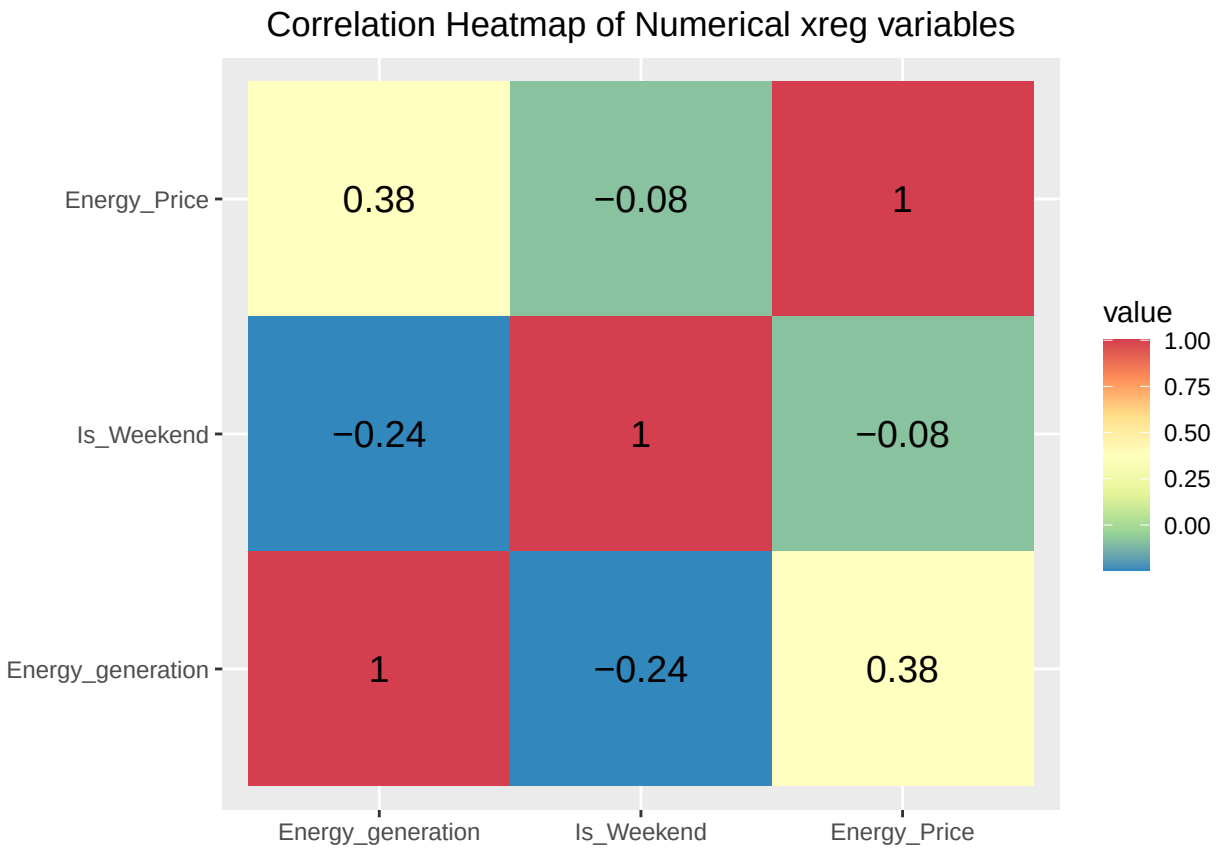
These engineered variables make up my xreg matrix for ARIMAX. I'm going to see if including these eXogenous variables in the model results in better forecasting accuracy.

Feature correlation with response

Checking correlation with response:

```
numeric_vars <- data.frame(Energy_generation, Is_Weekend)
numeric_vars["Energy_Price"] <- Training_set$DollarsPerMegawattHour
cor_matrix1 <- cor(numeric_vars, use = "complete.obs")
cor_matrix <- melt(cor_matrix1)

ggplot(data = cor_matrix, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  geom_text(aes(label = round(value, 2)), color = "black", size = 5) +
  scale_fill_distiller(palette = "Spectral") +
  labs(title = "Correlation Heatmap of Numerical xreg variables", x = NULL, y = NULL) +
  theme(plot.title = element_text(hjust = 0.5))
```

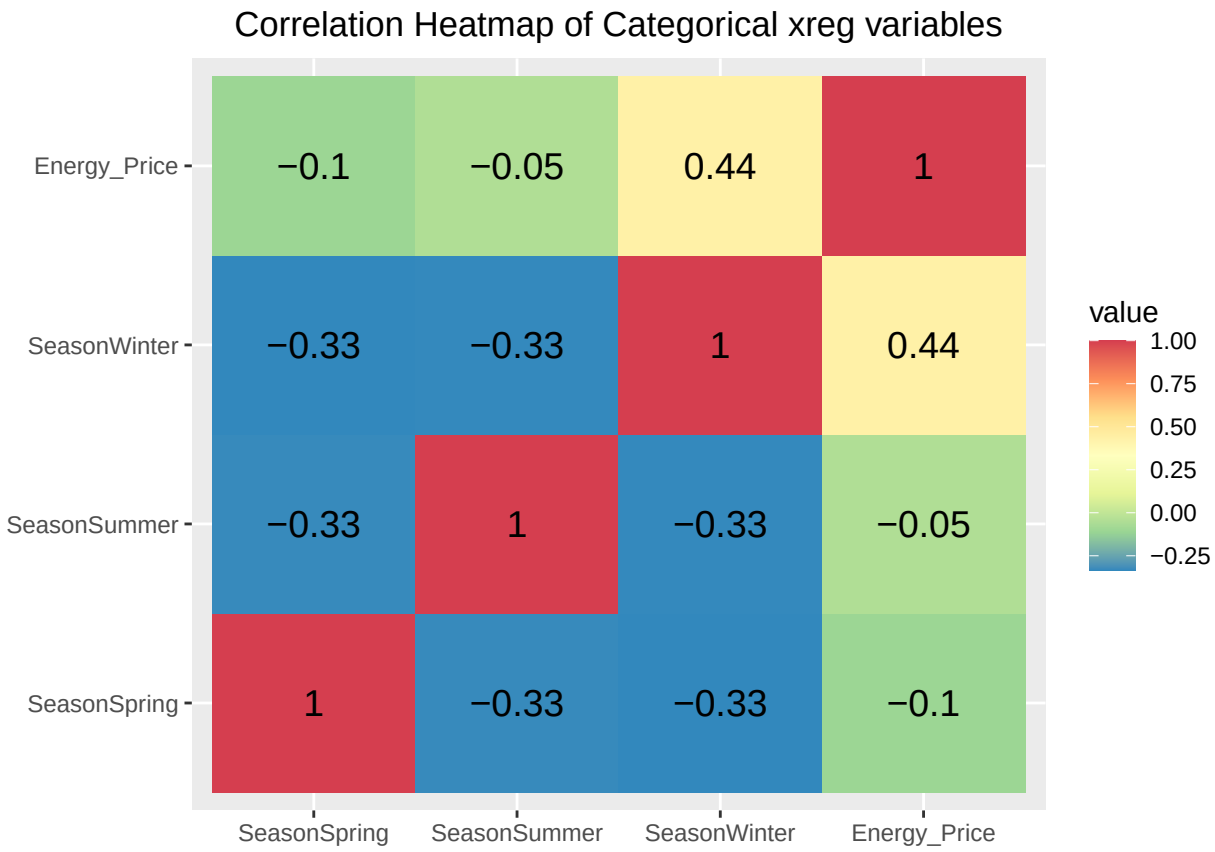


None of the regressor variables are strongly correlated with each other. The response has almost no correlation (-0.07) with the variable Is_Weekend and has a weak correlation (0.38) with Energy_generation.

Because Season is categorical I'm checking this variable's correlation with the response separately:

```
categorical_vars <- data.frame(Season_dummies)
categorical_vars["Energy_Price"] <- Training_set$DollarsPerMegawattHour
cor_matrix2 <- cor(categorical_vars, use = "complete.obs")
cor_matrix_categorical <- melt(cor_matrix2)

ggplot(data = cor_matrix_categorical, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  geom_text(aes(label = round(value, 2)), color = "black", size = 5) +
  scale_fill_distiller(palette = "Spectral") +
  labs(title = "Correlation Heatmap of Categorical xreg variables", x = NULL, y = NULL) +
  theme(plot.title = element_text(hjust = 0.5))
```



The response variable has a weak negative correlation of -0.05 with Season Summer and a weak negative correlation of -0.1 with Season Spring. The response variable has a moderate positive correlation of 0.44 with Season Winter.

Fitting model

Fitting model:

```
ARIMAX_model1 <- auto.arima(time_series_data, xreg = xreg, stepwise = TRUE, approximation = TRUE,
                            d = 1, max.p = 2, max.q = 2, max.P = 1, max.Q = 1, allowdrift = TRUE)
```

```r
summary(ARIMAX_model1)
```

```
## Series: time_series_data
## Regression with ARIMA(2,1,2)(0,0,1)[48] errors
##
## Coefficients:
##          ar1     ar2      ma1      ma2    sma1  Energy_generation  SeasonSpring
##       0.4547  0.2635  -0.7851  -0.1860  0.1152            32.5625       17.1364
## s.e.  0.0470  0.0335   0.0481   0.0463  0.0075             1.4108       13.8609
##       SeasonSummer  SeasonWinter  Is_Weekend
##             5.0377        8.9367      0.9156
## s.e.       12.0068       12.0052      1.4094
##
## sigma^2 = 369.9:  log likelihood = -76651.97
## AIC=153326   AICc=153326   BIC=153411.4
##
## Training set error measures:
##                       ME    RMSE      MAE       MPE    MAPE      MASE
## Training set 0.02214019 19.2276 8.470801 -70.76959 80.0227 0.4343825
##                      ACF1
## Training set 0.0003994259
```

My ARIMAX model is: ARIMA(2,1,2)(0,0,1)[48] errors

The time series is being modeled using the external predictors (xreg) but the errors are following an ARIMA process.

The ARIMA structure of the errors is: non seasonal AR(2), first order differencing, non seasonal MA(2). There is no seasonal AR or differencing. There is a seasonal MA(1) and a seasonal period of 48

ARIMA-GARCH models

I've decided to fit an ARIMA-GARCH model because it can handle seasonality, volatility clustering, conditional heteroskedasticity and capture tail behavior. This combination could make it better at capturing long term patterns and dealing with sudden market shocks than an ARIMA model.

Fitting an ARIMA-GARCH model:

Since GARCH models don't handle differencing internally I'm applied a first order differencing to my time series prior to fitting the model.

I'm starting with a simple ARIMA-GARCH model with 1 GARCH term (lagged conditional variances) and 1 ARCH term (lagged squared residuals). Each model has an autocorrelation component of order 1, and a moving average component of order 1.

I'm starting by testing different distributions with the same model to see which distribution is best for modelling the differenced time series. The distributions I'm checking are the normal distribution, the standard normal distribution, the generalized error distribution, the skew normal distribution, and the standardized skew Students t distribution.

Testing different distributions:

```r
fit_ARIMA_GARCH <- function(distribution, garchorder_1, garchorder_2, armaOrder1, armaOrder2){
spec <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(garchorder_1, garchorder_2)),
```

```
  mean.model     = list(armaOrder = c(armaOrder1, armaOrder2), include.mean = TRUE),
  distribution.model = distribution
)
ugarchfit(spec = spec, data = first_order_diff, solver = "hybrid")
}
```

```
fit_norm <- fit_ARIMA_GARCH("norm", 1, 1, 1, 1)
fit_std  <- fit_ARIMA_GARCH("std", 1, 1, 1, 1)
fit_ged <- fit_ARIMA_GARCH("ged", 1, 1, 1, 1)
fit_snorm <- fit_ARIMA_GARCH("snorm", 1, 1, 1, 1)
fit_sstd  <- fit_ARIMA_GARCH("sstd", 1, 1, 1, 1)
```

```
models <- list(norm = fit_norm, std = fit_std, ged = fit_ged, snorm = fit_snorm, sstd = fit_sstd)

info_table <- sapply(models, infocriteria)

info_df <- as.data.frame(t(info_table))
colnames(info_df) <- c("AIC", "BIC", "Shibata", "Hannan-Quinn")

print(info_df)
```

```
##            AIC      BIC  Shibata Hannan-Quinn
## norm  7.603597 7.606258 7.603596     7.604473
## std   7.148654 7.151759 7.148653     7.149676
## ged   6.995758 6.998863 6.995757     6.996780
## snorm 7.573111 7.576216 7.573111     7.574133
## sstd  7.146194 7.149742 7.146193     7.147362
```

The generalized error distribution distribution (GED) has the best model fit according to AIC, BIC, Shibata, and Hannan-Quinn. I'm going to double check that its the best distribution for the data by looking at the residuals.

kernel density plots

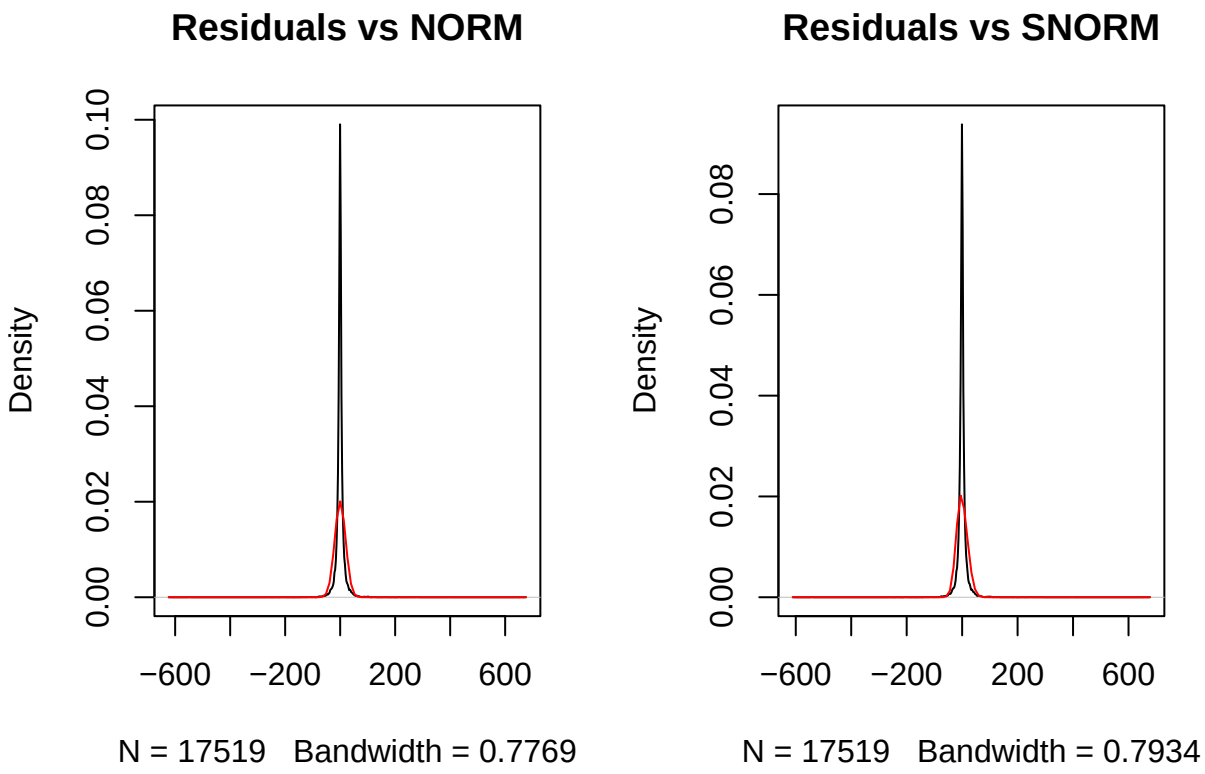
To visually check which distribution best fits the model, I am going to plot the theoretical distribution of each model over the actual distribution of its residuals (errors). This will allow me to see how closely each models theoretical distribution aligns with its observed residuals.

I will be using kernel density plots of residuals for this comparison, where the red line represents the theoretical distribution and the black line represents the actual distribution of the model's residuals.

```
par(mfrow = c(1, 2))

x <- residuals(fit_norm)
plot(density(residuals(fit_norm)), main = "Residuals vs NORM")
curve(dnorm(x, mean = 0, sd = sd(residuals(fit_norm))), add = TRUE, col = "red")

x <- residuals(fit_snorm)
plot(density(residuals(fit_snorm)), main = "Residuals vs SNORM")
curve(dsnorm(x, mean = 0, sd = sd(residuals(fit_snorm)),
             xi = coef(fit_snorm)["skew"]), add = TRUE, col = "red")
```



For both plots, the red line (representing the theoretical distribution) doesn't closely follow the black line (representing the actual distribution of the model's residuals).

The red curve is wider than the black curve meaning that the theoretical distribution has more variability (spread) than the actual distribution of the residuals. The actual distribution of the residuals shown by the black line has a narrower curve indicating tighter clustering. I can see that this clustering is centered around 0 and that the distribution is symmetric. For both plots, the gap between the curves suggests that the model is not capturing the error structure accurately.

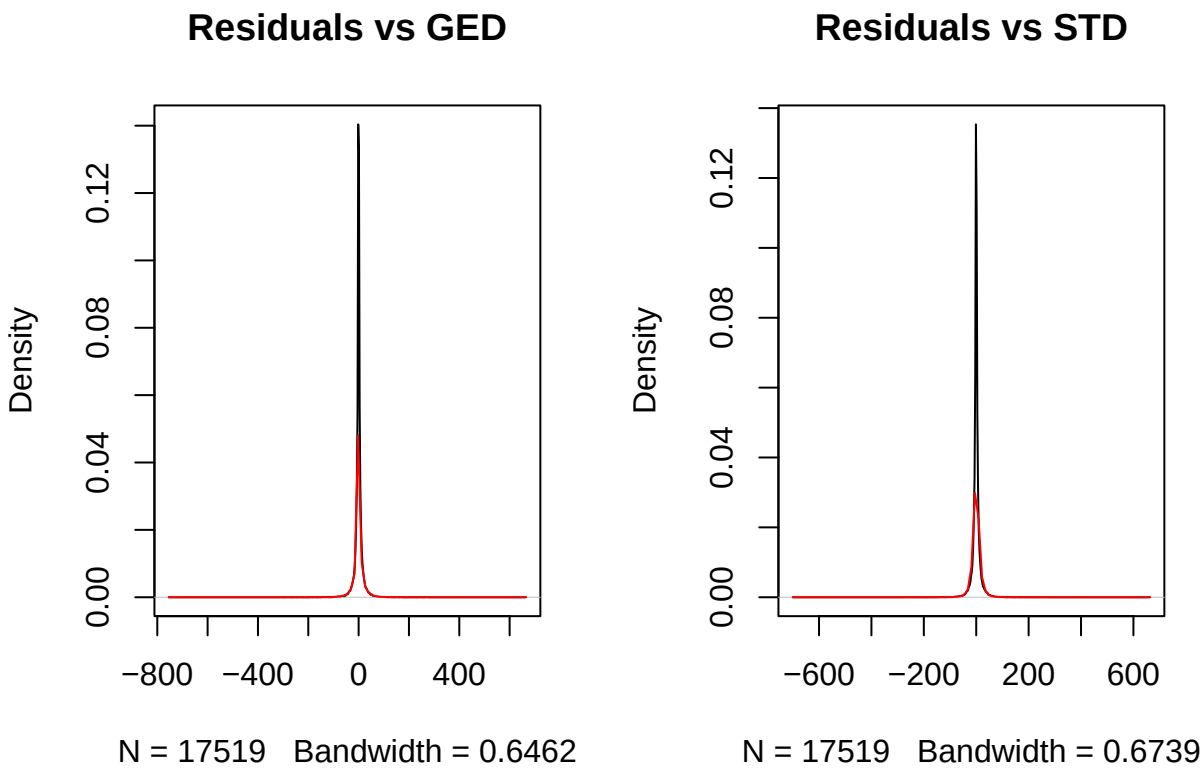
For both plots, the peak of the actual residuals (shown by black line) is much higher than the theoretical distribution's peak (shown by red line).

This suggests that the normal distribution and skewed normal distribution are poor fits for the ARIMA-GARCH models.

```
par(mfrow = c(1, 2))

x <- residuals(fit_ged)
plot(density(residuals(fit_ged)), main = "Residuals vs GED")
curve(dged(x, mean = 0, sd = sd(residuals(fit_ged)),
           nu = coef(fit_ged)["shape"]), add = TRUE, col = "red")

x <- residuals(fit_std)
plot(density(residuals(fit_std)), main = "Residuals vs STD")
curve(dstd(x, mean = 0, sd = sd(residuals(fit_std)),
           nu = coef(fit_std)["shape"]), add = TRUE, col = "red")
```



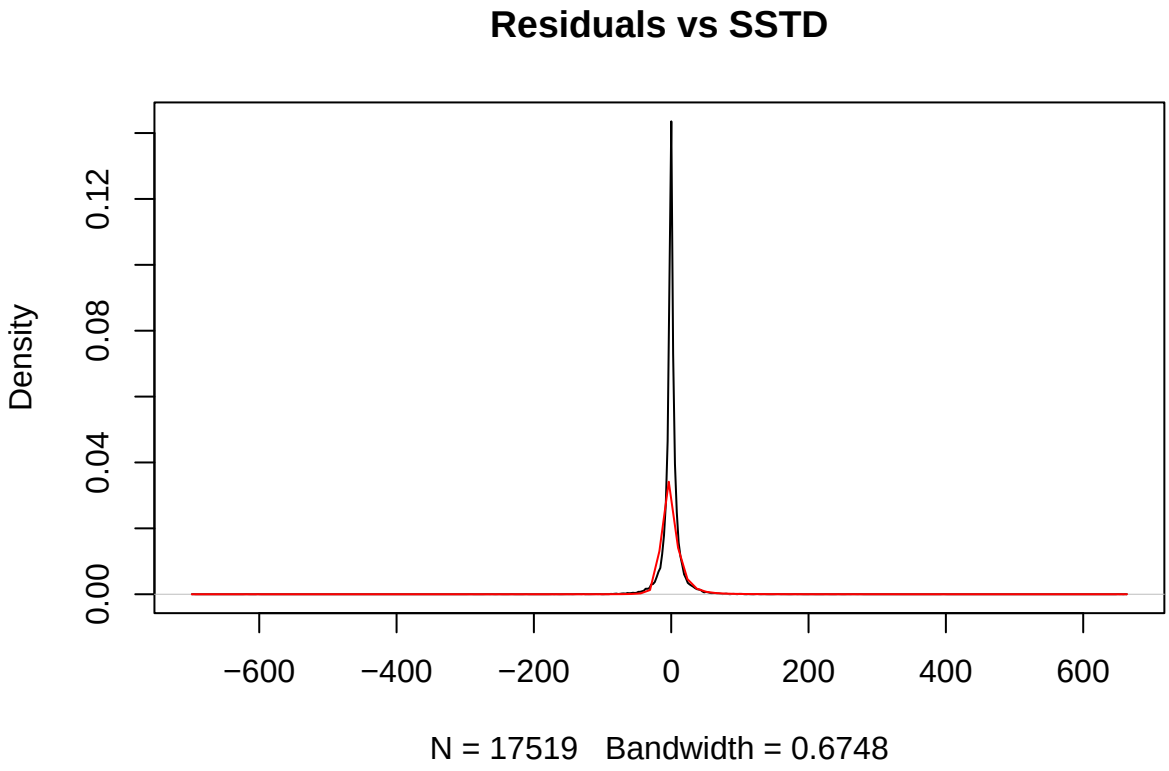
For both plots, the theoretical distribution overlaps the actual distribution of the residuals almost perfectly, expect for the peak.

This means that the spread (variability) of the distributions, theoretical and actual, is around the same.

I can see that the distributions are symmetric and centered around 0.

For both plots, the peak of the actual residuals (shown by black line) is much higher than the theoretical distribution's peak (shown by red line). The GED theoretical distribution appears to more accurately capture the structure of the errors than the std distribution.

```
x <- residuals(fit_sstd)
plot(density(residuals(fit_sstd)), main = "Residuals vs SSTD")
curve(dsstd(x, mean = 0, sd = sd(residuals(fit_sstd)),
            nu = coef(fit_sstd)["shape"]), add = TRUE, col = "red")
```



The red line (representing the theoretical distribution) doesn't closely follow the black line (representing the actual distribution of the model's residuals). The peak of the actual residuals (shown by black line) is much higher than the theoretical distribution's peak (shown by red line).

This shows that the theoretical distribution has much more variability (spread) than the actual distribution of the residuals. The gap between the curves suggests that the model with the sstd distribution does not accurately capture the error structure.

For every single model the peak of the observed residuals, shown by the black line, is much higher than the theoretical distribution curve, which is shown by the red line.

The model with the GED distribution most accurately captures the structure of the errors, but it can't capture the peak (mode of the data) where majority of the data is concentrated.

```
library(moments)
skewness(residuals(fit_ged))
```

```
##          X
## -0.6718785
```

```
kurtosis(residuals(fit_ged))
```

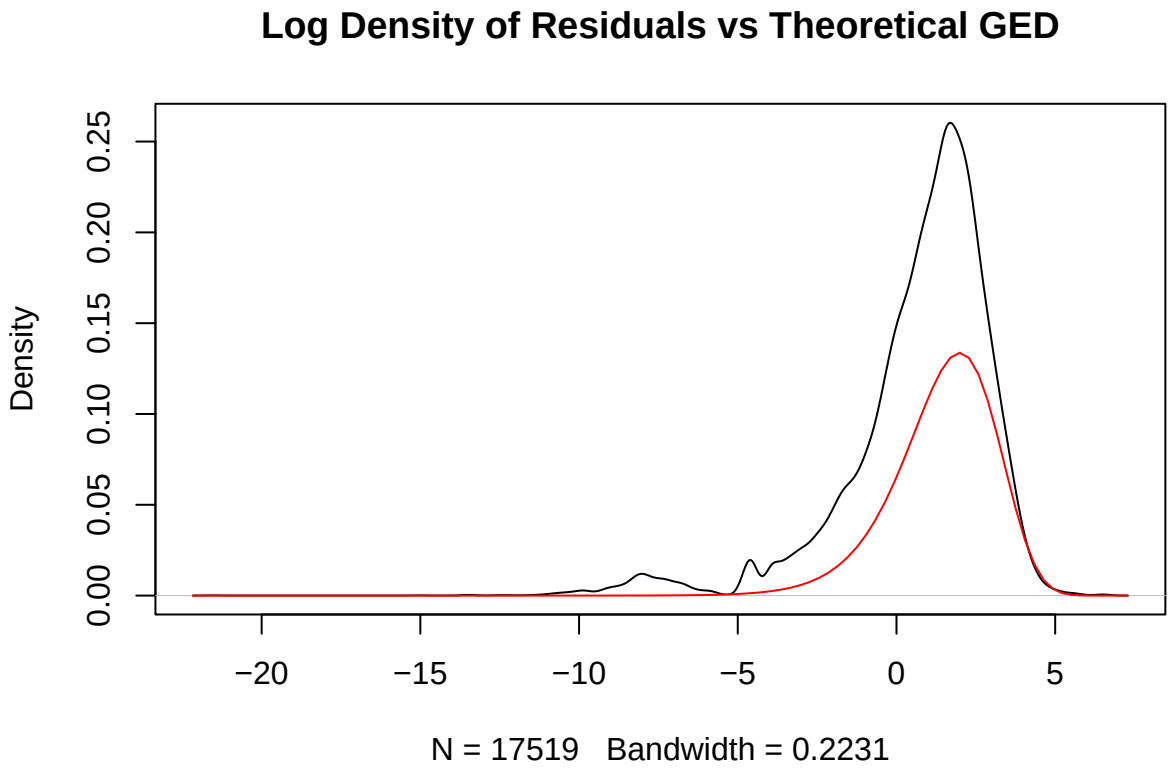
```
##        X
## 409.5173
```

The skew is quite close to 0, meaning the residuals are symmetric, so skewed distributions like snorm and sstd are not necessary.

The kurtosis is much larger than 10, which mean that the residuals are extremely leptokurtic.

```
# log-transformed GED density
dged_log <- function(x, mean, sd, nu) {
  y <- exp(x) # inverse of log
  dged(y, mean = mean, sd = sd, nu = nu) * y # multiplying by the derivative of inverse (Jacobian)
}

plot(density(log(abs(residuals(fit_ged)))), main = "Log Density of Residuals vs Theoretical GED")
curve(dged_log(x, mean = 0, sd = sd(residuals(fit_ged)),
               nu = coef(fit_ged)["shape"]), add = TRUE, col = "red")
```



This plot compares the log density of residuals (black curve) against the log of the theoretical GED distribution (red curve).

Although both curves peak around 2, the black curve has a sharper and higher peak than the red, this suggests that the observed residuals are more concentrated (have less spread) near their mode than the theoretical GED distribution predicts.

The log residual distribution shows multiple peaks, the highest peak is at around 2, and the two smaller peaks are at around -5 and -7. This shows multi-modal behavior and indicates latent structure.

The black curve extends further into the left tail and with higher density than the theoretical GED would predict. This indicates tail risk, which is the increased likelihood of extreme negative errors that could lead to substantial model errors.

The heavy left tail of the actual residuals indicate that the model underestimates the probability of large negative errors. When forecasting energy prices, this issue could result in poor estimates during unexpected price shocks, particularly sudden drops.

The extreme kurtosis in the residuals suggests that parameter tuning is necessary to better capture tail behavior and improve model fit.

ARIMA-Garch grid search

```
fit_ARIMA_GARCH_model <- function(garchorder_1, garchorder_2, armaOrder1, armaOrder2, spec_model){
spec <- ugarchspec(
  variance.model = list(model = spec_model, garchOrder = c(garchorder_1, garchorder_2)),
  mean.model     = list(armaOrder = c(armaOrder1, armaOrder2)),
  distribution.model = "ged"
)
ugarchfit(spec = spec, data = first_order_diff, solver = "hybrid")
}
```

ARIMA-GARCH Grid search function for best model:

```
Grid_search <- function(grid, daily_returns = "default", realized_vol_xts = "default") {
results <- list()

# Looping through grid of parameters
for (i in 1:nrow(grid)) {
  g1 <- grid$garchorder_1[i]
  g2 <- grid$garchorder_2[i]
  p  <- grid$armaOrder1[i]
  q  <- grid$armaOrder2[i]
  spec_model <- as.character(grid$spec_model_list[i])


# Labeling each model tested and printing the label to keep track
  model_name <- paste0("ARMA(", p, ",", q, ") GARCH(", g1, ",", g2, ")", " ", spec_model)
  cat("Fitting:", model_name, "\n")

# Fits the model using custom function fit_ARIMA_GARCH
# Captures errors and warnings to avoid crashing the loop
# Returns NULL if fitting the model fails
  fit <- tryCatch(
    fit_ARIMA_GARCH_model(g1, g2, p, q, spec_model),
    error = function(e) {
      cat("Error:", e$message, "\n")
      return(NULL)
    },
    warning = function(w) {
      cat("Warning:", w$message, "\n")
      return(NULL)
    }
  )

# If the model isn't NULL then stores info criteria in a list
# Otherwise stores NA's for that model
  if (!is.null(fit)) {
    ic <- infocriteria(fit)
```

```
    results[[model_name]] <- ic
  } else {
    results[[model_name]] <- rep(NA, 4)
  }
}
return(results)
}
```

For each of the following grid search's, I've set the maximum order of the autoregressive component to be 3, this is because the prior ARIMA and SARIMA models found using auto.arima had autoregressive components of order 3.

```
# Defining parameter grid for ARIMA-GARCH:
grid <- expand.grid( garchorder_1 = 1:2,  garchorder_2 = 1:2,
                     armaOrder1   = 0:3,  armaOrder2   = 0:2, spec_model_list = "sGARCH")
results <- Grid_search(grid)
```

```
## Fitting: ARMA(0,0) GARCH(1,1) sGARCH
## Fitting: ARMA(0,0) GARCH(2,1) sGARCH
## Fitting: ARMA(0,0) GARCH(1,2) sGARCH
## Fitting: ARMA(0,0) GARCH(2,2) sGARCH
## Fitting: ARMA(1,0) GARCH(1,1) sGARCH
## Fitting: ARMA(1,0) GARCH(2,1) sGARCH
## Fitting: ARMA(1,0) GARCH(1,2) sGARCH
## Fitting: ARMA(1,0) GARCH(2,2) sGARCH
## Fitting: ARMA(2,0) GARCH(1,1) sGARCH
## Fitting: ARMA(2,0) GARCH(2,1) sGARCH
## Fitting: ARMA(2,0) GARCH(1,2) sGARCH
## Fitting: ARMA(2,0) GARCH(2,2) sGARCH
## Fitting: ARMA(3,0) GARCH(1,1) sGARCH
## Fitting: ARMA(3,0) GARCH(2,1) sGARCH
## Fitting: ARMA(3,0) GARCH(1,2) sGARCH
## Fitting: ARMA(3,0) GARCH(2,2) sGARCH
## Fitting: ARMA(0,1) GARCH(1,1) sGARCH
## Fitting: ARMA(0,1) GARCH(2,1) sGARCH
## Fitting: ARMA(0,1) GARCH(1,2) sGARCH
## Fitting: ARMA(0,1) GARCH(2,2) sGARCH
## Fitting: ARMA(1,1) GARCH(1,1) sGARCH
## Fitting: ARMA(1,1) GARCH(2,1) sGARCH
## Fitting: ARMA(1,1) GARCH(1,2) sGARCH
## Fitting: ARMA(1,1) GARCH(2,2) sGARCH
## Fitting: ARMA(2,1) GARCH(1,1) sGARCH
## Fitting: ARMA(2,1) GARCH(2,1) sGARCH
## Fitting: ARMA(2,1) GARCH(1,2) sGARCH
## Fitting: ARMA(2,1) GARCH(2,2) sGARCH
## Fitting: ARMA(3,1) GARCH(1,1) sGARCH
## Fitting: ARMA(3,1) GARCH(2,1) sGARCH
## Fitting: ARMA(3,1) GARCH(1,2) sGARCH
## Fitting: ARMA(3,1) GARCH(2,2) sGARCH
## Fitting: ARMA(0,2) GARCH(1,1) sGARCH
## Fitting: ARMA(0,2) GARCH(2,1) sGARCH
## Fitting: ARMA(0,2) GARCH(1,2) sGARCH
## Fitting: ARMA(0,2) GARCH(2,2) sGARCH
```

```
## Fitting: ARMA(1,2) GARCH(1,1) sGARCH
## Fitting: ARMA(1,2) GARCH(2,1) sGARCH
## Fitting: ARMA(1,2) GARCH(1,2) sGARCH
## Fitting: ARMA(1,2) GARCH(2,2) sGARCH
## Fitting: ARMA(2,2) GARCH(1,1) sGARCH
## Fitting: ARMA(2,2) GARCH(2,1) sGARCH
## Fitting: ARMA(2,2) GARCH(1,2) sGARCH
## Fitting: ARMA(2,2) GARCH(2,2) sGARCH
## Warning:
## rugarch-->warning: failed to invert hessian
##
## Fitting: ARMA(3,2) GARCH(1,1) sGARCH
## Fitting: ARMA(3,2) GARCH(2,1) sGARCH
## Fitting: ARMA(3,2) GARCH(1,2) sGARCH
## Fitting: ARMA(3,2) GARCH(2,2) sGARCH
```

```
grid_fGARCH <- expand.grid(garchorder_1 = 1,    garchorder_2 = 1,
                           armaOrder1   = 0:3,  armaOrder2   = 0:2, spec_model_list = "fiGARCH")
results_f <- Grid_search(grid_fGARCH)
```

```
## Fitting: ARMA(0,0) GARCH(1,1) fiGARCH
## Fitting: ARMA(1,0) GARCH(1,1) fiGARCH
## Fitting: ARMA(2,0) GARCH(1,1) fiGARCH
## Fitting: ARMA(3,0) GARCH(1,1) fiGARCH
## Fitting: ARMA(0,1) GARCH(1,1) fiGARCH
## Fitting: ARMA(1,1) GARCH(1,1) fiGARCH
## Fitting: ARMA(2,1) GARCH(1,1) fiGARCH
## Fitting: ARMA(3,1) GARCH(1,1) fiGARCH
## Fitting: ARMA(0,2) GARCH(1,1) fiGARCH
## Fitting: ARMA(1,2) GARCH(1,1) fiGARCH
## Fitting: ARMA(2,2) GARCH(1,1) fiGARCH
## Fitting: ARMA(3,2) GARCH(1,1) fiGARCH
```

```
grid_iGARCH <- expand.grid(garchorder_1 = 1:2,    garchorder_2 = 1:2,
                           armaOrder1   = 0:3,  armaOrder2   = 0:2, spec_model_list = "iGARCH")
results_i <- Grid_search(grid_iGARCH)
```

```
## Fitting: ARMA(0,0) GARCH(1,1) iGARCH
## Fitting: ARMA(0,0) GARCH(2,1) iGARCH
## Fitting: ARMA(0,0) GARCH(1,2) iGARCH
## Fitting: ARMA(0,0) GARCH(2,2) iGARCH
## Fitting: ARMA(1,0) GARCH(1,1) iGARCH
## Fitting: ARMA(1,0) GARCH(2,1) iGARCH
## Fitting: ARMA(1,0) GARCH(1,2) iGARCH
## Fitting: ARMA(1,0) GARCH(2,2) iGARCH
## Fitting: ARMA(2,0) GARCH(1,1) iGARCH
## Fitting: ARMA(2,0) GARCH(2,1) iGARCH
## Fitting: ARMA(2,0) GARCH(1,2) iGARCH
## Fitting: ARMA(2,0) GARCH(2,2) iGARCH
## Fitting: ARMA(3,0) GARCH(1,1) iGARCH
## Fitting: ARMA(3,0) GARCH(2,1) iGARCH
## Fitting: ARMA(3,0) GARCH(1,2) iGARCH
## Fitting: ARMA(3,0) GARCH(2,2) iGARCH
```

```
## Fitting: ARMA(0,1) GARCH(1,1) iGARCH
## Fitting: ARMA(0,1) GARCH(2,1) iGARCH
## Fitting: ARMA(0,1) GARCH(1,2) iGARCH
## Fitting: ARMA(0,1) GARCH(2,2) iGARCH
## Fitting: ARMA(1,1) GARCH(1,1) iGARCH
## Fitting: ARMA(1,1) GARCH(2,1) iGARCH
## Fitting: ARMA(1,1) GARCH(1,2) iGARCH
## Fitting: ARMA(1,1) GARCH(2,2) iGARCH
## Fitting: ARMA(2,1) GARCH(1,1) iGARCH
## Fitting: ARMA(2,1) GARCH(2,1) iGARCH
## Fitting: ARMA(2,1) GARCH(1,2) iGARCH
## Fitting: ARMA(2,1) GARCH(2,2) iGARCH
## Fitting: ARMA(3,1) GARCH(1,1) iGARCH
## Fitting: ARMA(3,1) GARCH(2,1) iGARCH
## Fitting: ARMA(3,1) GARCH(1,2) iGARCH
## Fitting: ARMA(3,1) GARCH(2,2) iGARCH
## Fitting: ARMA(0,2) GARCH(1,1) iGARCH
## Fitting: ARMA(0,2) GARCH(2,1) iGARCH
## Fitting: ARMA(0,2) GARCH(1,2) iGARCH
## Fitting: ARMA(0,2) GARCH(2,2) iGARCH
## Fitting: ARMA(1,2) GARCH(1,1) iGARCH
## Fitting: ARMA(1,2) GARCH(2,1) iGARCH
## Fitting: ARMA(1,2) GARCH(1,2) iGARCH
## Fitting: ARMA(1,2) GARCH(2,2) iGARCH
## Fitting: ARMA(2,2) GARCH(1,1) iGARCH
## Fitting: ARMA(2,2) GARCH(2,1) iGARCH
## Fitting: ARMA(2,2) GARCH(1,2) iGARCH
## Fitting: ARMA(2,2) GARCH(2,2) iGARCH
## Fitting: ARMA(3,2) GARCH(1,1) iGARCH
## Fitting: ARMA(3,2) GARCH(2,1) iGARCH
## Fitting: ARMA(3,2) GARCH(1,2) iGARCH
## Fitting: ARMA(3,2) GARCH(2,2) iGARCH
```

```
metrics2 <- c("Akaike", "Bayes", "Shibata", "Hannan-Quinn")
# Combining the lists into one big list
all_results <- c(results, results_f, results_i)

# Assigning names to each vector
for(i in seq_along(all_results)) {
  names(all_results[[i]]) <- metrics2
}

# Turning the list into a tibble with two columns: Model and ICs (the numeric vector)
df_all <- enframe(all_results, name = "Model", value = "ICs") %>%
  # Unpacking the named vector in each row into its own columns
  unnest_wider(ICs)

df_all %>% arrange(Akaike)
```

```
## # A tibble: 108 x 5
##    Model                     Akaike[,1] Bayes[,1] Shibata[,1] `Hannan-Quinn`[,1]
##    <chr>                          <dbl>     <dbl>       <dbl>              <dbl>
##  1 ARMA(1,0) GARCH(1,1) fiG~       6.97      6.98        6.97               6.97
##  2 ARMA(0,1) GARCH(1,1) fiG~       6.97      6.98        6.97               6.97
```

```
##  3 ARMA(1,1) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
##  4 ARMA(2,0) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
##  5 ARMA(0,2) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
##  6 ARMA(2,1) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
##  7 ARMA(1,2) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
##  8 ARMA(2,2) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
##  9 ARMA(3,1) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
## 10 ARMA(3,0) GARCH(1,1) fiG~       6.97      6.98        6.97                6.97
## # i 98 more rows
```

Out of the 108 models considered the ARMA(1,0) fiGARCH(1,1) model had the lowest AIC, BIC, Shibata and Hannan-Quinn.

Of the 108 models that I attempted to fit in the grid search, 1 of those models (the ARMA(2,2) GARCH(2,2) sGARCH model), failed to invert its hessian matrix.

I will be moving forward with the ARMA(1,0) fiGARCH(1,1) model for forecasting due to it having the best performance in terms of AIC, BIC, Shibata and Hannan-Quinn.

```
spec <- ugarchspec(
  variance.model = list(model = "fiGARCH", garchOrder = c(1, 1)),
  mean.model     = list(armaOrder = c(1, 0)),
  distribution.model = "ged")
```

```
GARCH_model <- ugarchfit(spec = spec, data = first_order_diff, solver = "hybrid")
show(GARCH_model)
```

Fitting ARIMAX-fiGARCH

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : fiGARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : ged
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error   t value Pr(>|t|)
## mu       0.00000    0.000006 -0.000394 0.999686
## ar1      0.00000    0.000005  0.012207 0.990261
## omega   19.95721    2.763668  7.221274 0.000000
## alpha1   0.14562    0.070090  2.077609 0.037745
## beta1    0.34702    0.074165  4.678975 0.000003
## delta    0.59054    0.031626 18.672700 0.000000
## shape    0.49791    0.007045 70.679744 0.000000
```

```
##
## Robust Standard Errors:
##         Estimate  Std. Error   t value Pr(>|t|)
## mu       0.00000    0.000012 -0.000198 0.999842
## ar1      0.00000    0.000003  0.022044 0.982413
## omega   19.95721    3.630345  5.497331 0.000000
## alpha1   0.14562    0.055238  2.636185 0.008384
## beta1    0.34702    0.057933  5.989967 0.000000
## delta    0.59054    0.035412 16.676163 0.000000
## shape    0.49791    0.016362 30.430402 0.000000
##
## LogLikelihood : -61066.72
##
## Information Criteria
## ------------------------------------
##
## Akaike       6.9723
## Bayes        6.9754
## Shibata      6.9723
## Hannan-Quinn 6.9733
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic  p-value
## Lag[1]                      69.55 1.11e-16
## Lag[2*(p+q)+(p+q)-1][2]     73.88 0.00e+00
## Lag[4*(p+q)+(p+q)-1][5]    114.80 0.00e+00
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                      0.105  0.7459
## Lag[2*(p+q)+(p+q)-1][5]     0.569  0.9465
## Lag[4*(p+q)+(p+q)-1][9]     1.099  0.9818
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]  0.001927 0.500 2.000  0.9650
## ARCH Lag[5]  0.938161 1.440 1.667  0.7514
## ARCH Lag[7]  1.181746 2.315 1.543  0.8823
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  12.3753
## Individual Statistics:
## mu     0.09441
## ar1    0.46468
## omega  7.80626
## alpha1 0.68165
## beta1  3.31331
```

```
## delta  0.97648
## shape  1.10442
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.69 1.9 2.35
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                     t-value   prob sig
## Sign Bias           1.1029 0.2701
## Negative Sign Bias  1.1498 0.2502
## Positive Sign Bias  0.3515 0.7252
## Joint Effect        4.1331 0.2474
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20      1012   1.294e-202
## 2    30      1353   9.503e-267
## 3    40      1598   2.056e-310
## 4    50      1787    0.000e+00
##
##
## Elapsed time : 13.39576
```

Since I applied first order differencing to the time series prior to fitting the model, then the model is:

ARMA(1,1,0) fiGARCH(1,1) model

Where the time series has been differenced one (d = 1) and includes an autoregressive component of order 1 in the mean.

The models variance dynamics include 1 ARCH term and 2 GARCH terms.

Forecasting

Loading Validation set:

Loading validation data:

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

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

Because I removed daylight savings effected days from my training set then I am also going to remove these days from my validation set. Daylight savings for 2018 starts on 2018-09-30

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

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

```
##   TradingDate  n
## 1  2018-09-30 46
```

I'm going to use linear interpolation to fill in the 2 trading period gap for 2018-09-30

```r
# finding the row just before period 47 on 2018-09-30
i_prev <- which(val_set$TradingDate == as.Date("2018-09-30") &
                val_set$TradingPeriod == 46)

# Extracting the two known prices
price_prev <- val_set$DollarsPerMegawattHour[i_prev]
price_next <- val_set$DollarsPerMegawattHour[i_prev + 1]

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

```r
new_rows <- tibble(
  TradingDate             = as.Date("2018-09-30"),
  TradingPeriod           = interp$x,
  PointOfConnection       = "ABY0111",
  DollarsPerMegawattHour  = interp$y
)

# bind back and resort
val_set <- bind_rows(val_set, new_rows) %>%
  arrange(TradingDate, TradingPeriod)

# viewing the gap now filled
filter(val_set,
       TradingDate == as.Date("2018-09-30"),
       TradingPeriod %in% 46:49)
```

```
##   TradingDate TradingPeriod PointOfConnection DollarsPerMegawattHour
## 1  2018-09-30            46           ABY0111                  63.21
## 2  2018-09-30            47           ABY0111                  73.85
## 3  2018-09-30            48           ABY0111                  84.49
```

Making it a time series object:

```r
price_vector2 <- as.numeric(val_set$DollarsPerMegawattHour)
Val_time_series <- ts(price_vector2, start = c(2018, 1), frequency = 48)
```

ARIMA

```
forecast_raw_ARIMA_1 <- forecast(ARIMA_model1, h = 17520)
```

```
forecast_ARIMA <- ts(forecast_raw_ARIMA_1, start = c(2018, 1), frequency = 48)
```

SARIMA

```
forecast_raw_SARIMA_1 <- forecast(SARIMA_model1, h = 17520)
```

```
forecast_SARIMA <- ts(forecast_raw_SARIMA_1, start = c(2018, 1), frequency = 48)
```

STL-ARIMA

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

```
forecast_raw_STL_1 <- forecast(STL_ARIMA_model1, h = 17520)
```

```
forecast_STL <- ts(forecast_raw_STL_1, start = c(2018, 1), frequency = 48)
```

ARIMAX

Of my three ARIMAX variables, only two are known in advance every year (Is_Weekend and Season), and one isn't known in advance every year (Energy_generation). Because I want to simulate real world conditions during validation I am going to compare the results of two ARIMAX models, one with the actual Energy_generation values and one with forecasted Energy_generation values. This is to understand how sensitive the model is to uncertainty in external inputs.

ARIMAX; actual Energy_generation

Forecasting with actual Energy_generation values:

```
csv_files2 <- list.files(path = "Energy Generation - 2018", pattern = "*.csv", full.names = TRUE)

Generation2 <- csv_files2 %>% lapply(read.csv) %>% bind_rows()
```

```
Generation_Overall_Output2 <- Generation2[, -c(56, 57)] %>%
  pivot_longer(
    cols = matches("^TP\\d+$"), # gets all the trading period columns (TP1 to TP50)
    names_to = "TradingPeriod", # makes a new column containing the old period column names
    names_prefix = "TP", # removes the "TP" prefix from all rows in the TradingPeriod column
    values_to = "Energy_output" # makes a new column for site energy output
  ) %>%
  mutate(
    TradingPeriod = as.integer(TradingPeriod),  # converting the TradingPeriod values to integers
    Trading_date = as.Date(Trading_date)  # converting Trading_date into a date object
```

```
  ) %>%
  group_by(Trading_date, TradingPeriod) %>%
  # Treats each unique combination of trading date and period as its own group
  summarise(
    Energy_output = sum(Energy_output, na.rm = TRUE),
  # calculates the total energy output for (that trading date and period)
  # each group by summing the energy output values across all sites
    .groups = "drop"
  )
```

```
Generation_Overall_Output2[which(Generation_Overall_Output2$Energy_output == 0), ]
```

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

```
which(Generation_Overall_Output2$Energy_output == 0)
```

```
## [1] 13103 13104
```

```
i_prev <- which(Generation_Overall_Output2$Trading_date == as.Date("2018-09-30") &
                Generation_Overall_Output2$TradingPeriod == 46)

price_prev <- Generation_Overall_Output2$Energy_output[i_prev]
price_next <- Generation_Overall_Output2$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_Output2$Energy_output[13103] <- interp$y[1]
Generation_Overall_Output2$Energy_output[13104] <- interp$y[2]

Generation_Overall_Output2[c(13103, 13104), ]
```

```
## # A tibble: 2 x 3
##   Trading_date TradingPeriod Energy_output
##   <date>               <int>         <dbl>
## 1 2018-09-30              47      1323838.
## 2 2018-09-30              48       661919.
```

```
# Energy generation output
Energy_generation <- Generation_Overall_Output2$Energy_output / 1e6

# Season:
# I'm using meteorological season start dates (which are the same every year)
```

```r
# for simplicity and consistency
Season2 <- data.frame(Date = val_set$TradingDate)
Season2$Season <- rep("", nrow(val_set))
Season2$Season[
  which(Season2$Date >= as.Date("2018-03-01") & Season2$Date <= as.Date("2018-05-31"))
] <- "Autumn"

Season2$Season[
  which(Season2$Date >= as.Date("2018-06-01") & Season2$Date <= as.Date("2018-08-31"))
] <- "Winter"

Season2$Season[
  which(Season2$Date >= as.Date("2018-09-01") & Season2$Date <= as.Date("2018-11-30"))
] <- "Spring"

Season2$Season[
  which(Season2$Date >= as.Date("2018-12-01") & Season2$Date <= as.Date("2018-12-31"))
] <- "Summer"
Season2$Season[
  which(Season2$Date >= as.Date("2018-01-01") & Season2$Date <= as.Date("2018-02-28"))
] <- "Summer"

# Converting seasons to dummy variables (with one-hot encoding)
Season_dummies <- model.matrix(~ Season, data = Season2)[, -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(val_set$TradingDate) %in% c("Saturday", "Sunday"), 1, 0)

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

Forecasting:

```r
forecast_raw_ARIMAX_1 <- forecast(ARIMAX_model1, h = 17520, xreg = xreg_future)
```

```r
forecast_ARIMAX <- ts(forecast_raw_ARIMAX_1, start = c(2018, 1), frequency = 48)
```

ARIMAX; forecasted Energy_generation

Forecasting with estimated (forecasted) 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:

```r
Energy_generation_2017 <- Generation_Overall_Output$Energy_output / 1e6
ts_Energy <- ts(Energy_generation_2017, start = c(2017, 1), frequency = 48)

lambda <- BoxCox.lambda(ts_Energy)
Energy_generation_bc <- BoxCox(ts_Energy, lambda)
fit_Energy <- auto.arima(Energy_generation_bc)
fcast_bc <- forecast(fit_Energy, h = 17520)
Energy_generation <- InvBoxCox(fcast_bc$mean, lambda)
```

```r
summary(fit_Energy)
```

```
## Series: Energy_generation_bc
## ARIMA(3,0,1)(1,1,0)[48] with drift
##
## Coefficients:
##          ar1    ar2     ar3     ma1     sar1  drift
##       0.5427  0.274  0.0884  0.2342  -0.4284      0
## s.e.  0.0564  0.046  0.0098  0.0571   0.0069      0
##
## sigma^2 = 0.0001821:  log likelihood = 50431.09
## AIC=-100848.2   AICc=-100848.2   BIC=-100793.8
##
## Training set error measures:
##                        ME       RMSE         MAE          MPE      MAPE
## Training set 2.813245e-06 0.01347348 0.004342664 -0.004720894 0.8149397
##                    MASE         ACF1
## Training set 0.2353266 0.0005773134
```

Model for forecasting energy generation is ARIMA(3,0,1)(1,1,0)[48] with drift.

```r
# Season:
# I'm using meteorological season start dates (which are the same every year)
# for simplicity and consistency
Season2 <- data.frame(Date = val_set$TradingDate)
Season2$Season <- rep("", nrow(val_set))
Season2$Season[
  which(Season2$Date >= as.Date("2018-03-01") & Season2$Date <= as.Date("2018-05-31"))
] <- "Autumn"

Season2$Season[
  which(Season2$Date >= as.Date("2018-06-01") & Season2$Date <= as.Date("2018-08-31"))
] <- "Winter"

Season2$Season[
  which(Season2$Date >= as.Date("2018-09-01") & Season2$Date <= as.Date("2018-11-30"))
] <- "Spring"

Season2$Season[
  which(Season2$Date >= as.Date("2018-12-01") & Season2$Date <= as.Date("2018-12-31"))
] <- "Summer"
Season2$Season[
  which(Season2$Date >= as.Date("2018-01-01") & Season2$Date <= as.Date("2018-02-28"))
] <- "Summer"

# Converting seasons to dummy variables (with one-hot encoding)
Season_dummies <- model.matrix(~ Season, data = Season2)[, -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(val_set$TradingDate) %in% c("Saturday", "Sunday"), 1, 0)

xreg_future2 <- cbind(Energy_generation, Season_dummies, Is_Weekend)
```

```r
# Correcting incorrect column names:
colnames(xreg_future2)[2] <- "SeasonSpring"
colnames(xreg_future2)[3] <- "SeasonSummer"
colnames(xreg_future2)[4] <- "SeasonWinter"
```

Forecasting:

```r
forecast_raw_ARIMAX_2 <- forecast(ARIMAX_model1, h = 17520, xreg = xreg_future2)
```

```r
forecast_ARIMAX2 <- ts(forecast_raw_ARIMAX_2, start = c(2018, 1), frequency = 48)
```

ARIMA-GARCH

Forecasting ARIMA-GARCH model:

```r
forecast_raw_Garch <- ugarchforecast(GARCH_model, n.ahead = 17520)
mean_fc_vector <- as.numeric(fitted(forecast_raw_Garch)[ , 1])
```

I first order differenced the time series before fitting the model so I'm converting back:

```r
recovered_forecast <- as.numeric(cumsum(mean_fc_vector))
```

Adding the Last Observed Value of the original series:

```r
last_value <- as.numeric(tail(time_series_data, 1))
final_forecast <- last_value + recovered_forecast
forecast_GARCH <- ts(final_forecast, start = c(2019, 1), frequency = 48)
```

Accuracy metrics for forecasting 2018 energy prices

Model accuracy:

```r
ARIMA_acc <- accuracy(forecast_ARIMA, Val_time_series)
SARIMA_acc <- accuracy(forecast_SARIMA, Val_time_series)
STL_ARIMA_acc <- accuracy(forecast_STL, Val_time_series)
ARIMAX_acc <- accuracy(forecast_ARIMAX, Val_time_series)
ARIMAX2_acc <- accuracy(forecast_ARIMAX2, Val_time_series)
GARCH_acc <- accuracy(forecast_GARCH, Val_time_series)
```

```r
acc_list <- list(
  ARIMA      = ARIMA_acc,
  SARIMA     = SARIMA_acc,
  STL_ARIMA  = STL_ARIMA_acc,
  ARIMAX     = ARIMAX_acc,
  ARIMAX_model2 = ARIMAX2_acc,
  ARIMA_fiGARCH = GARCH_acc)
```

Manually calculating MASE for ARIMA-fiGARCH:

```r
# Naive forecast errors from training set
naive_errors <- diff(time_series_data)  # Assuming naive one-step forecasts

# MAE from naive model
mae_naive <- mean(abs(naive_errors))

# MAE from the ARIMA-GARCH forecast
mae_garch <- mean(abs(Val_time_series - as.numeric(forecast_GARCH)))

# MASE for ARIMA GARCH
mase_garch <- mae_garch / mae_naive
```

```r
acc_df <- map_df(acc_list, ~ as.data.frame(.x)["Test set", ], .id = "Model")
acc_df[6, 7] <- mase_garch

acc_df %>% arrange(MASE) %>% mutate(across(where(is.numeric), ~ round(.x, digits = 2)))
```

```
##                      Model    ME   RMSE   MAE      MPE    MAPE MASE ACF1
## Test set...1         ARIMA  4.25  18.39 13.12     0.61   15.29 0.67 0.65
## Test set...2        SARIMA  6.67  19.18 13.62     3.44   15.51 0.70 0.65
## Test set...3     STL_ARIMA  4.65  19.77 14.90     0.82   17.62 0.76 0.69
## Test set...4 ARIMAX_model2  7.96  20.78 15.85     4.93   18.09 0.81 0.68
## Test set...5        ARIMAX  7.80  20.67 15.88     4.79   18.08 0.81 0.67
## Test set...6 ARIMA_fiGARCH 26.97 104.26 51.42 -1855.52 1889.67 6.34 0.91
##              Theil's U
## Test set...1      1.38
## Test set...2      1.38
## Test set...3      1.53
## Test set...4      1.50
## Test set...5      1.47
## Test set...6     10.95
```

The ARIMA model has the best ME, RMSE, MAE, MPE, MAPE, MASE and Theil's U. It had the second best ACF1. ARIMA was the most accurate forecasting model.

The worst model was ARIMA-GARCH with the worst performance on every metric.

SARIMA and STL-ARIMA had similar performance with SARIMA having a slightly lower value for most metrics. STL-ARIMA had the lower values for ME and MPE though.

The ARIMAX model with the (ARIMA(3,0,1)(1,1,0)[48] with drift) forecasted energy generation variable in its xreg (labeled ARIMAX_model2), performed similarly to ARIMAX with the actual values for energy generation in its xreg. There is an average difference of 0.05 between the performances. This tells me that the use of an estimated Energy_generation variable does not harm model performance significantly.

All models have ACF1 values of 0.65 or larger indicating a strong positive autocorrelation at lag 1, this suggests that every model fitted has not yet captured all autocorrelation in the time series. Ideally, residuals should resemble white noise (where ACF1 approximately equals 0). During model refinement I will try adding more AR and MA terms to try and capture this autocorrelation. I will also reassess the differencing order to see if the series needs further transformation.

The MASE for all models, except for ARIMA-GARCH, is less than 1 which means they beat a naive "yesterday's price" benchmark on average absolute error. On average, these models are producing more precise forecasts than simply carrying forward the last known value.

The best model so far is the ARIMA(3,1,1) model.

ARIMA interpretation

The Mean Absolute Scaled Error (MASE) for the ARIMA model is 0.67, which means that, on average, the ARIMA model's forecast errors are 33% smaller than those of a naive (random walk) model. The ARIMA model outperforms a naive forecasting model, where each forecasted energy price is just the last observed price carried forward.

Note that when I say "smaller errors", I mean that when the model's forecast for an energy price was incorrect, the size (magnitude) of the error was smaller. So the distance between the forecasted and actual price was smaller for the ARIMA model than that of a naive model.

The models mean error was 4.25 which means on average its forecasts overestimate actual prices by about 4.25 dollars per megawatt hour.

The models root mean square error was 18.39 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 models mean absolute error was 13.12 which means that forecasts were off by 13.12 units on average. This is 13.04% of the annual mean, the mean energy price for 2018 was 100.62, so the mean absolute error isn't that high considering how volatile the data is.

The mean percentage data was 0.61%, which means the models forecasts are slightly overestimate actual electricity prices overall.

The mean absolute percentage error was 15.29% which means that the models forecasts were off by 15.29% on average.

The models ACF1 was 0.65 which indicates that the residuals are moderately autocorrelated so there is some structure or pattern in the data that the model did not capture.

The ARIMA model did better than the STL-ARIMA, ARIMAX and ARIMA-GARCH models which so far shows that a more complex model does not equal a better performance, those other models were overfitting on the training data and not generalizing well enough to forecast the following year.

Theil's U is 1.38 which means that the ARIMA models forecasts are slightly worse than the naive forecasts, but this could be due to RMSE sensitivity (Theil's U is sensitive to large errors).

I'm going to look at the residuals of the models and do some model refinement. I will then forecast the 2019 energy prices with the refined models and consider the models accuracy.

Plot: actual vs forecast for best model

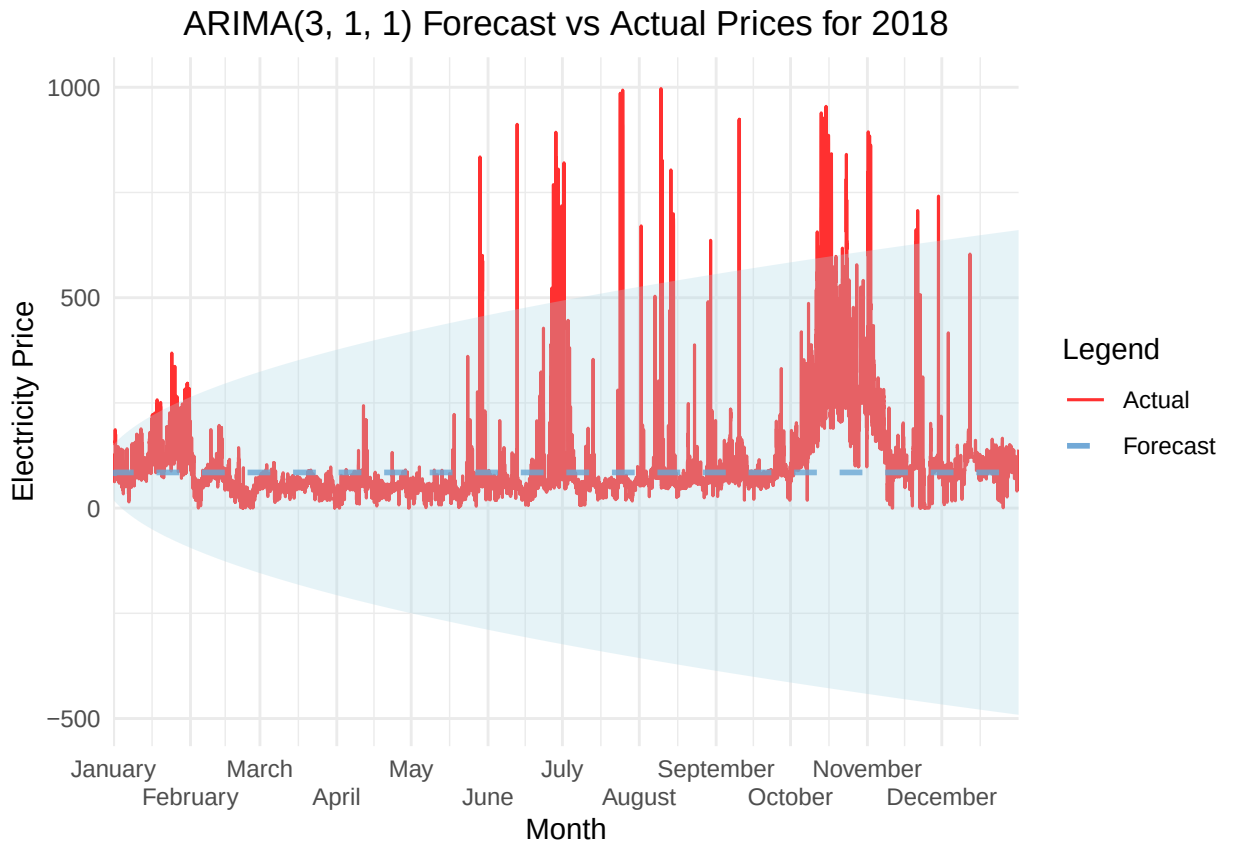
```
plot_actual_forecast <- function(start_time, actual_vals, start_year, forecasts, model_name) {
start_time     <- as.POSIXct(start_time, tz = "UTC")
times_forecast <- seq(start_time, by = "30 min", length.out = 17520)

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("2018-01-01 00:00:00", val_set, 2018, forecast_ARIMA, "ARIMA(3, 1, 1)")
```



This plot shows the ARIMA(311) model forecast for the final half hourly electricity prices for the year 2018 compared to the actual values for that year.

There are large fluctuations in the actual electricity prices, particularly in July and November.

The forecast as shown by the blue dashed lines appears to follow the overall trend of electricity prices. Its 95% confidence interval (the light blue ribbon) covers the majority of the electricity prices for 2018 but misses many large spikes between June and December which means the model was underestimating the real volatility.

However it does model the initial five months January to May quite well.

The ARIMA model has significant issues modelling the volatility of the energy price data.

Saving models for future use

```
saveRDS(ARIMA_model1, file = "Models/arima_model.rds")
saveRDS(SARIMA_model1, file = "Models/sarima_model.rds")
saveRDS(STL_ARIMA_model1, file = "Models/stl_arima_model.rds")
saveRDS(ARIMAX_model1, file = "Models/arimax_model.rds")
saveRDS(GARCH_model, file = "Models/garch_model.rds")

# Saving ARIMAX with xreg (done because future xreg contains forecasted variable)
saveRDS(list(model = ARIMAX_model1, future_xreg = xreg_future2), "Models/arima_with_xreg.rds")
saveRDS(fit_Energy, file = "Models/Energy_generation_model.rds")
```

Saving Training set and time series as csv's:

```
df <- data.frame(Value = as.numeric(time_series_data))

write.csv(df, "Saved_Datasets/timeseries.csv", row.names = FALSE)
write.csv(Training_set, "Saved_Datasets/Training_set.csv", row.names = FALSE)

df_V <- data.frame(Value = as.numeric(Val_time_series))
write.csv(df_V, "Saved_Datasets/Val_time_series.csv", row.names = FALSE)
write.csv(val_set, "Saved_Datasets/Val_set.csv", row.names = FALSE)
```

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