Energy Price Forecasting

Juliet Alexandra

2025-07-11

Libraries

```
library(dplyr)
```

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

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

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

```
library(ggplot2)
library(ggpmisc)
```

```
## Loading required package: ggpp

## Registered S3 methods overwritten by 'ggpp':
##   method                  from
##   heightDetails.titleGrob ggplot2
##   widthDetails.titleGrob  ggplot2

##
## Attaching package: 'ggpp'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(tseries)
```

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

```
library(nortest)
library(zoo)
```

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

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

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

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

```
library(lubridate)
```

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

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

```
library(purrr)
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:car':
##
##     some
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(FinTS)
library(rugarch)
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'rugarch'
```

```
## The following object is masked from 'package:purrr':
##
##     reduce
```

```r
library(tibble)
library(forecast)
```

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:FinTS':
##
##     Acf
```

Loading Data

```r
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:

```r
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:

```
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

```
# 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
```

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

```
)

# bind back and re-sort
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
```

```
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:

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

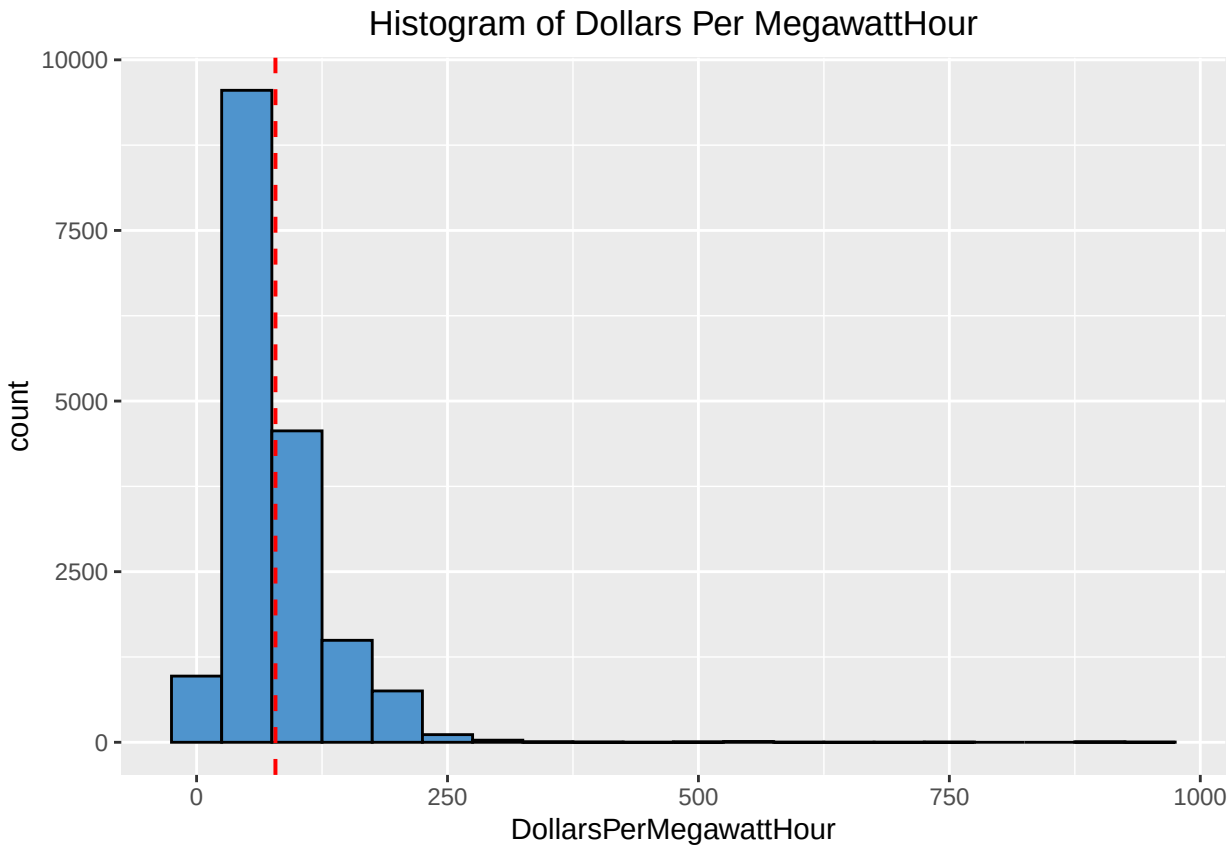
```
## [1] 365
```

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

```
ggplot(Training_set, aes(x = DollarsPerMegawattHour)) +
  geom_histogram(binwidth = 50, color = 'black', fill = 'steelblue3') +
  geom_vline(aes(xintercept = mean(DollarsPerMegawattHour)), color = "red", linetype = "dashed", linewi
  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.

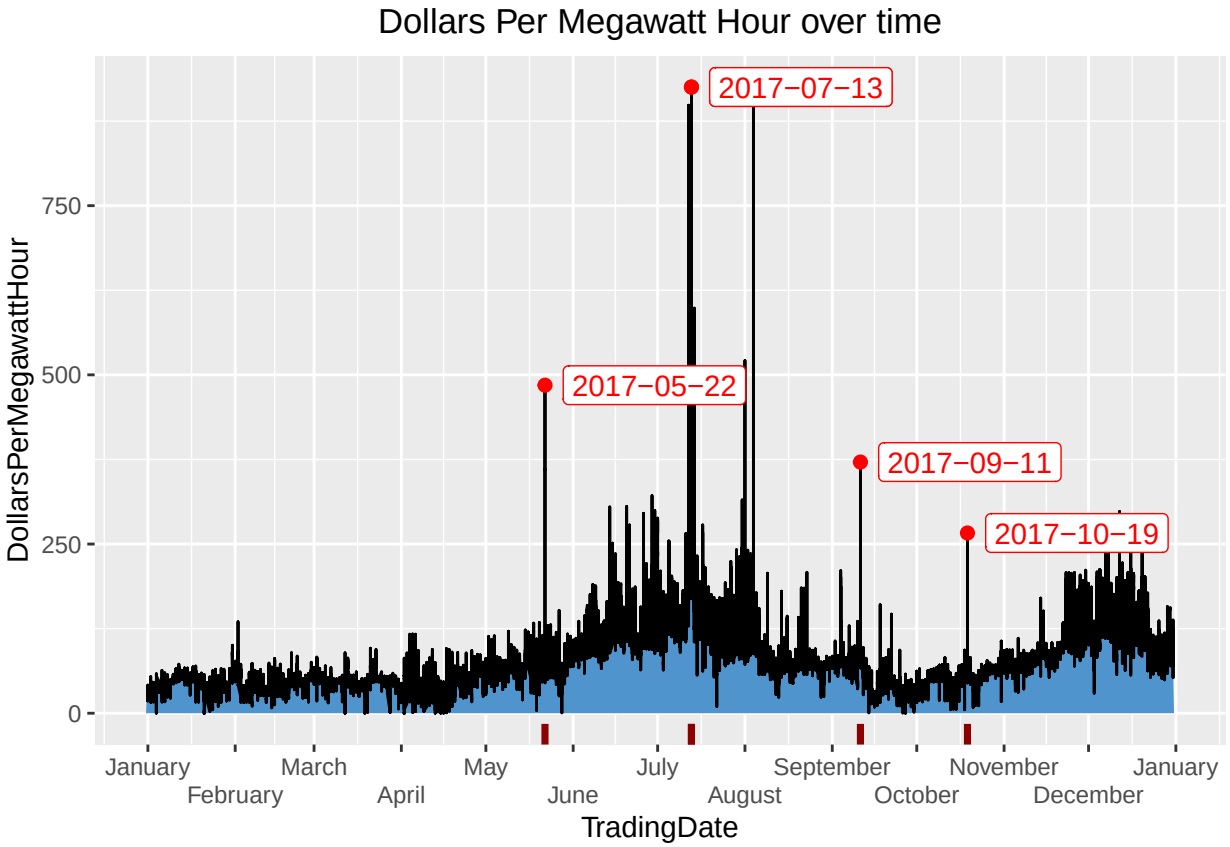
```
mean(Training_set$DollarsPerMegawattHour)
```

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

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

Time series plot:

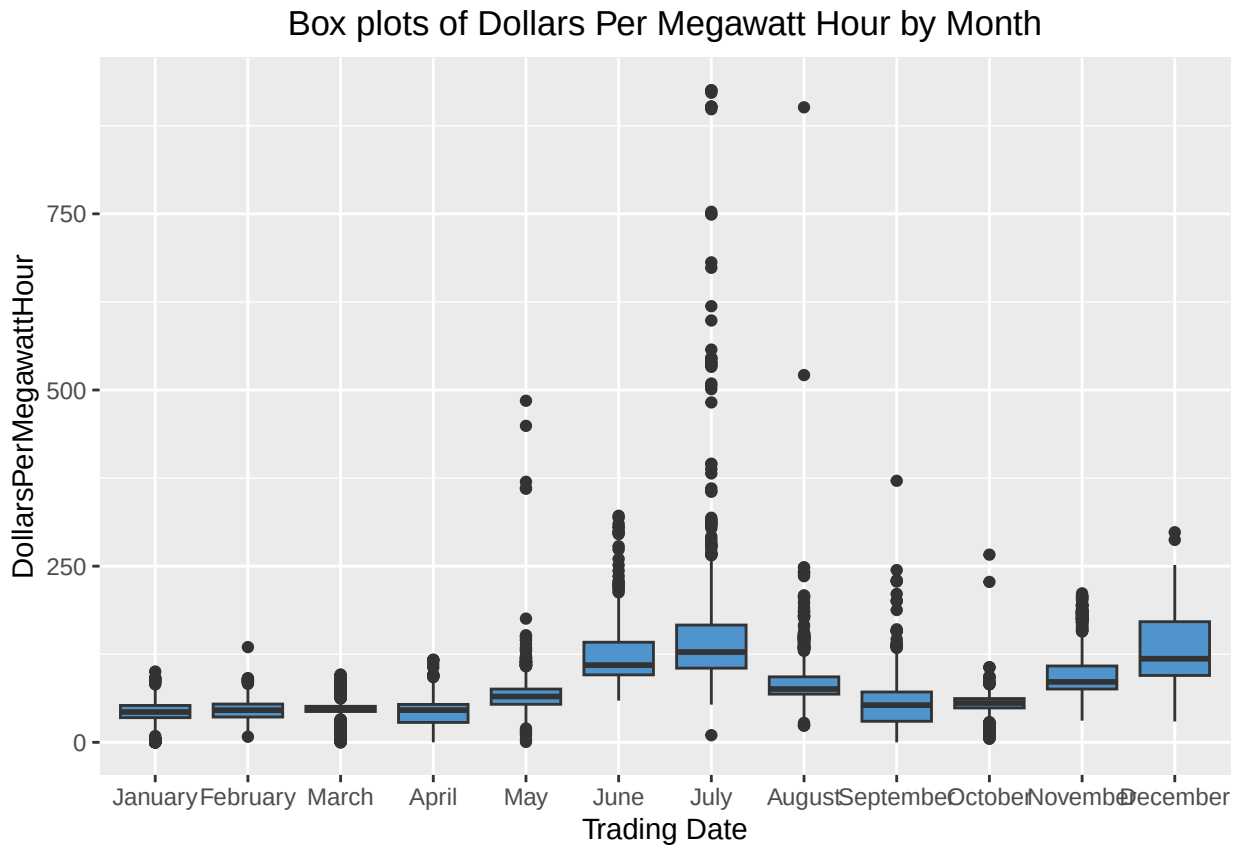
```
ggplot(Training_set, aes(x = TradingDate, y = DollarsPerMegawattHour)) +
  geom_area(fill = "steelblue3")+
  geom_line() +
  scale_x_date(date_breaks = "1 month", date_labels = "%B", guide = guide_axis(n.dodge = 2)) +
  stat_peaks(geom = "point", span = 3225, color = "red", size = 2) +
  stat_peaks(geom = "label", span = 3225, color = "red", angle = 0, hjust = -0.1, x.label.fmt = "%Y-%m-
  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))
```



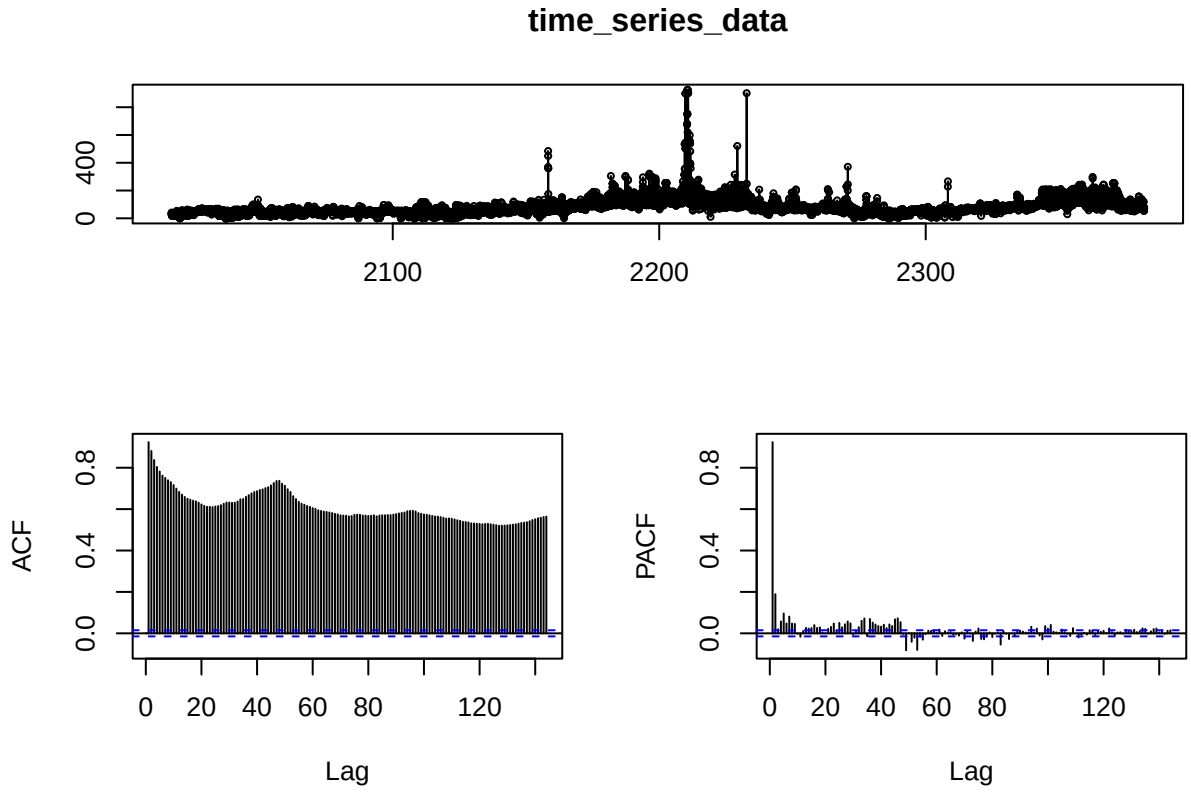
The price of electricity is much higher for June and July. The lower quartile of June and July doesn't even overlap with the upper quartiles of the previous months.

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.

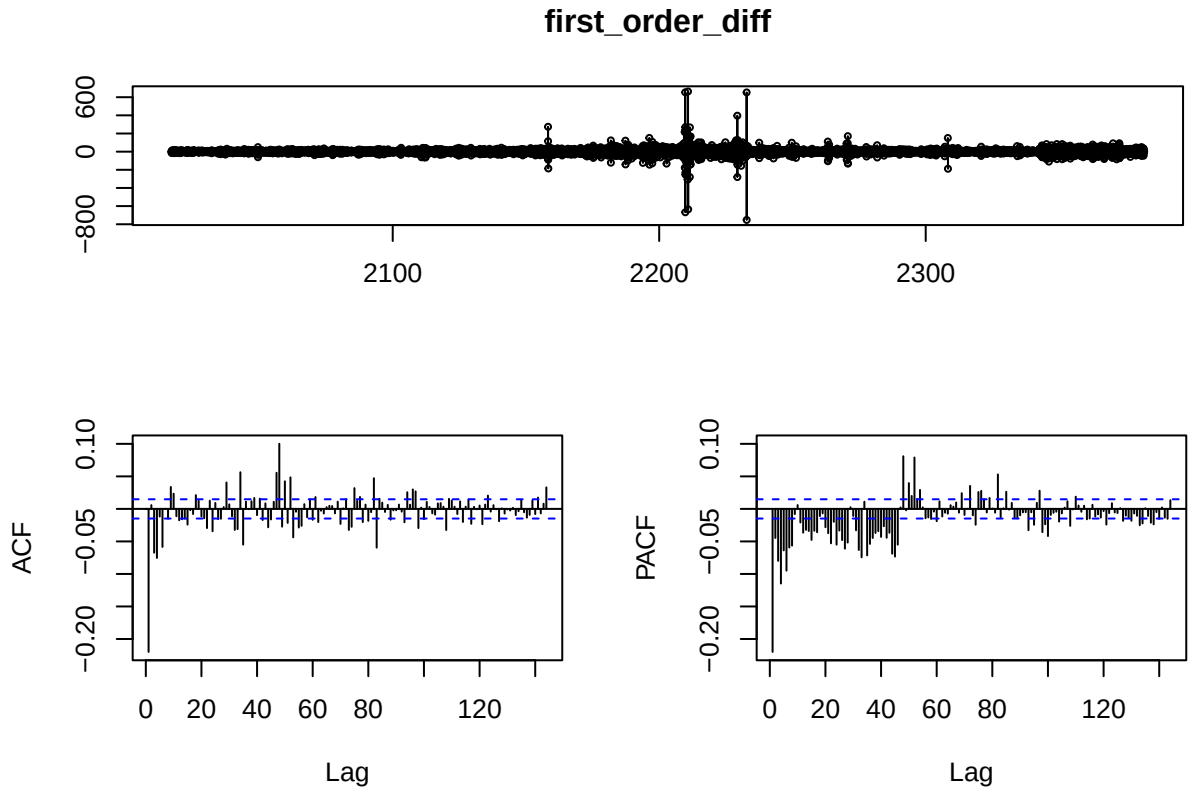
Conclusion:

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: is trend stationary

Alternative hypothesis: non-stationary (has 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)

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

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

```r
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 starting ARIMA model that I can manually adjust later based on its residuals.

```r
ARIMA_model1 <- auto.arima(time_series_data, stepwise = TRUE, approximation = TRUE, d = 1, max.p = 3, ma
```

```r
summary(ARIMA_model1)
```

```
## Series: time_series_data
## ARIMA(1,1,3)(0,0,2)[48] with drift
##
## Coefficients:
##          ar1      ma1     ma2      ma3    sma1    sma2   drift
##       0.7882  -1.0962  0.1825  -0.0633  0.1287  0.0274  0.0021
## s.e.  0.0103   0.0127  0.0112   0.0099  0.0076  0.0074  0.0186
##
## sigma^2 = 380.6:  log likelihood = -76902.96
## AIC=153821.9   AICc=153821.9   BIC=153884.1
##
## Training set error measures:
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.006298128 19.50496 8.508423 -80.26014 88.85882 0.4363118
##                      ACF1
## Training set -0.001060086
```

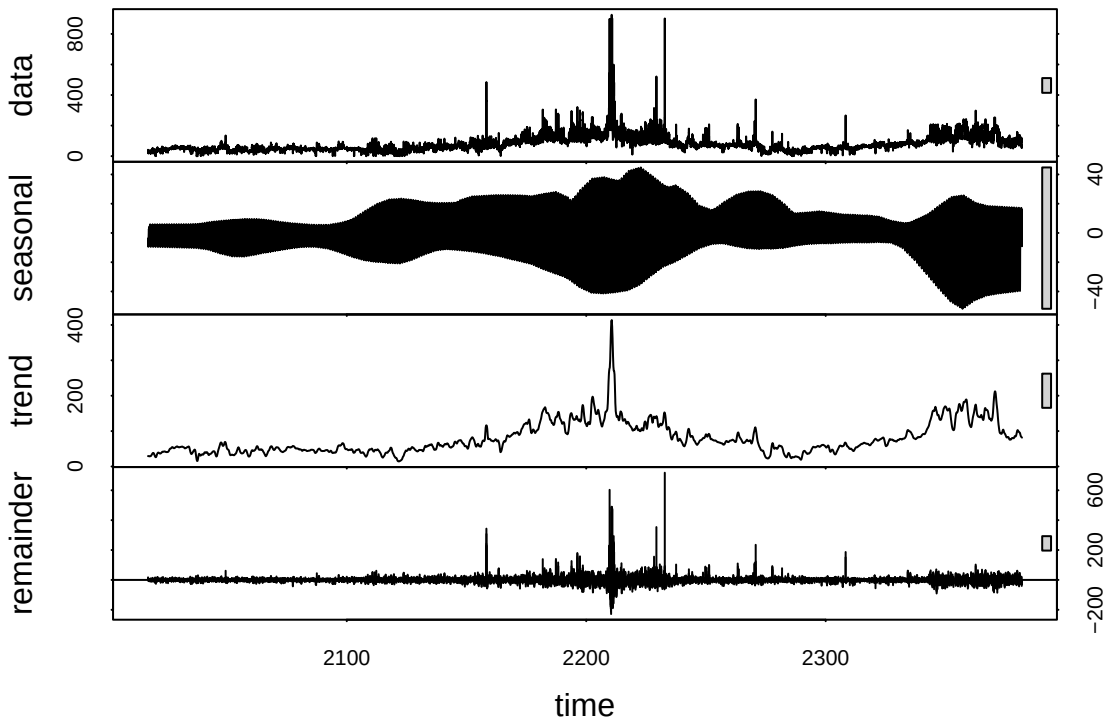
My current model is ARIMA(3,1,1)(0,0,2)[48]

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

STL-ARIMA

Decomposing the series and visualizing the decomposition:

```
decomp <- stl(time_series_data, s.window = 48)
plot(decomp)
```



The second panel shows a repeating pattern cycling between 40 and -40, meaning there's definitely a seasonal component. It consistently oscillates around zero, meaning that the default seasonal detection works well.

The third panel shows a smooth and mild upward movement suggests a weak long-term drift. The overall trend shows short-term fluctuations.

The fourth panel shows that the remainder is spiky and uneven, it contains outliers and volatility that is not explained by seasonality or trend.

Fitting an STL-ARIMA to seasonally adjusted series:

```
STL_ARIMA_model1 <- stlm(time_series_data, method = "arima")
```

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

ARIMAX

Feature engineering:

```r
# Day of Week
Day_of_Week <- as.numeric(format(Training_set$TradingDate, "%w"))
# Weekend / Weekday flag
Is_Weekend <- as.numeric(format(Training_set$TradingDate, "%w") %in% c("0", "6"))
# Month
Month <- as.numeric(format(Training_set$TradingDate, "%m"))
# Hour of day
Hour_of_Day <- floor((Training_set$TradingPeriod - 1) / 2)
# Peak period flag
Is_peak <- ifelse(Training_set$TradingPeriod %in% c(15:40), 1, 0)

xreg <- cbind(Day_of_Week, Is_Weekend, Month, Hour_of_Day, Is_peak)
```

These engineered variables are my xreg matrix for ARIMAX

fitting model:

```r
ARIMAX_model1 <- auto.arima(time_series_data, xreg = xreg, stepwise = TRUE, approximation = TRUE, d = 1
```

Since the series is being differenced (d = 1) I've decided that drift is not necessary.

TBATS

There is some residual autocorrelation in the series as seen from the acf and pacf plots. So I have set use.arma.errors = TRUE to include ARMA errors to model this residual autocorrelation.

Fitting model:

```r
TBATS_model1 <- tbats(time_series_data, use.arma.errors = TRUE, seasonal.periods = c(48, 336)) # daily
```

Forecasting

Loading Test Data

Loading test data:

```r
csv_folder <- "2018"
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 2018 starts on 2018-09-30

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

```
test_data %>% mutate(test_data = 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

```
# finding the row just before period 47 on 2017-09-24
i_prev <- which(test_data$TradingDate == as.Date("2018-09-30") &
                test_data$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] 73.85 84.49
```

```
new_rows <- tibble(
  TradingDate             = as.Date("2018-09-30"),
  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("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:

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

ARIMA

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

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

STL-ARIMA

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

```
forecast12months <- forecast(STL_ARIMA_model1, h = 17520)
```

```
forecast_raw_STL <- ts(forecast12months, start = c(2018, 1), frequency = 48)
```

ARIMAX forecasting

```
# Day of Week
Day_of_Week <- as.numeric(format(test_data$TradingDate, "%w"))
# Weekend / Weekday flag
Is_Weekend <- as.numeric(format(test_data$TradingDate, "%w") %in% c("0", "6"))
# Month
Month <- as.numeric(format(test_data$TradingDate, "%m"))
# Hour of day
Hour_of_Day <- floor((test_data$TradingPeriod - 1) / 2)
# Peak period flag
Is_peak <- ifelse(test_data$TradingPeriod %in% c(15:40), 1, 0)

xreg_future <- cbind(Day_of_Week, Is_Weekend, Month, Hour_of_Day, Is_peak)
```

forecasting:

```
forecast12months2 <- forecast(ARIMAX_model1, h = 17520, xreg = xreg_future)
```

```
forecast_raw_ARIMAX <- ts(forecast12months2, start = c(2018, 1), frequency = 48)
```

TBATS

Forecasting:

```
forecast_tbats <- forecast(TBATS_model1, h = 17520)
forecast_raw_TBATS <- ts(forecast_tbats, start = c(2018, 1), frequency = 48)
```

Accuracy metrics

Model accuracy:

```
ARIMA_acc <- accuracy(forecast_raw_ARIMA, Test_time_series)
STL_ARIMA_acc <- accuracy(forecast_raw_STL, Test_time_series)
ARIMAX_acc <- accuracy(forecast_raw_ARIMAX, Test_time_series)
TBATS_acc <- accuracy(forecast_raw_TBATS, Test_time_series)
```

```
acc_list <- list(
  ARIMA     = ARIMA_acc,
  STL_ARIMA = STL_ARIMA_acc,
  ARIMAX    = ARIMAX_acc,
  TBATS     = TBATS_acc)

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

acc_df
```

```
##                   Model        ME     RMSE      MAE       MPE     MAPE      MASE
## Test set...1     ARIMA  6.658897 19.20814 13.71418  3.405619 15.63555 0.7032626
## Test set...2 STL_ARIMA 12.108617 21.68740 17.48504 10.049182 19.23103 0.8966325
## Test set...3    ARIMAX 53.054223 56.16298 53.06168 58.348721 58.36713 2.7210020
## Test set...4     TBATS 13.977189 24.05575 18.33851 11.831874 20.04822 0.9403986
##                    ACF1 Theil's U
## Test set...1 0.6495645  1.393746
## Test set...2 0.6537431  1.503628
## Test set...3 0.6531938  3.889135
## Test set...4 0.6833433  1.667525
```

The ARIMA model has the best RMSE, MAE, MPE, MAPE and MASE. ARIMA was the most accurate forecasting model.

The worst model was ARIMAX with the worst ME, RMSE, MAE, MPE, MAPE and MASE.

TBATS had the best ME but had higher MAE, MAPE, and MASE than STL-ARIMA

All models have similar ACF1 values which suggests that the residuals for all models still have short range correlation.

The MASE for ARIMA, STL-ARIMA and TBATS iss less than 1 which means they beat a naive "yesterday's price" benchmark on average absolute error.

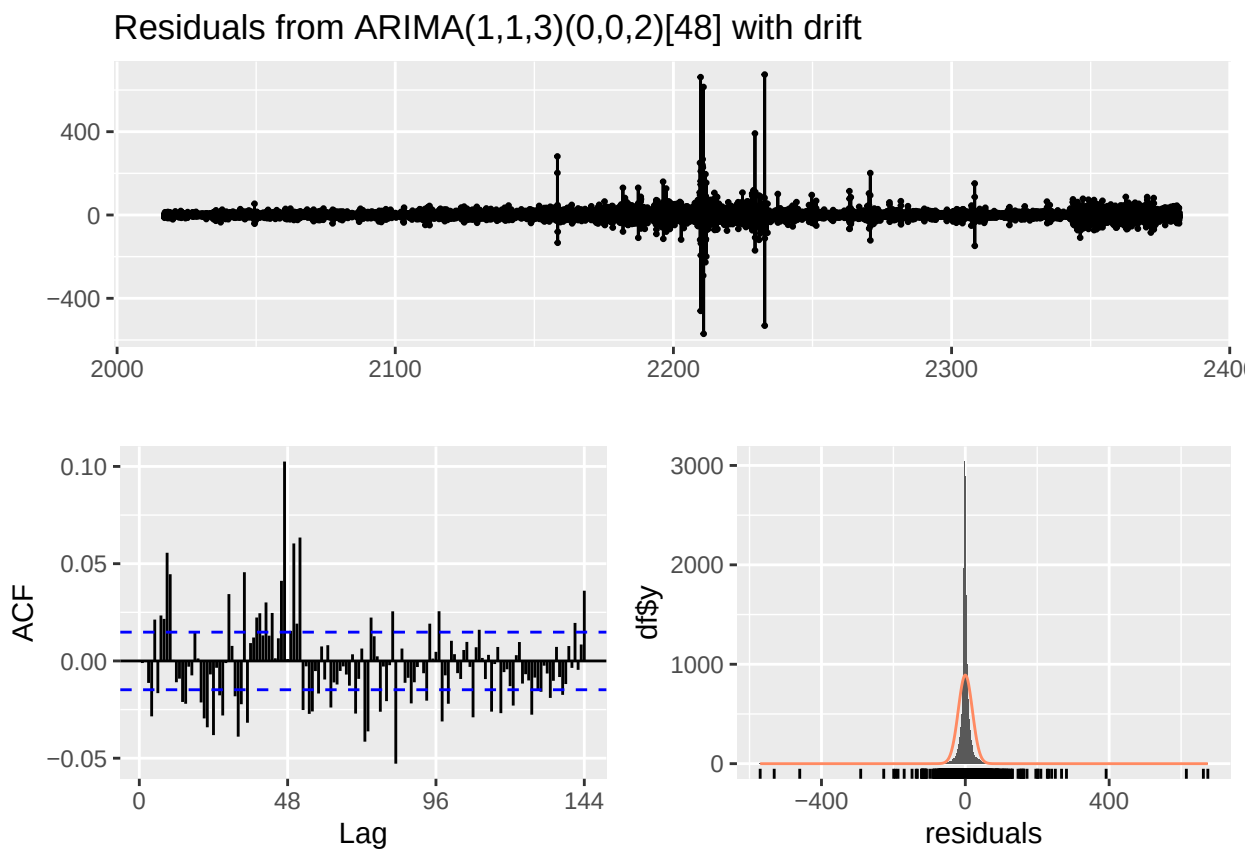
I'm going to check the residuals of the ARIMA and STL-ARIMA models and do some model refinement. I will not be continuing with the worst performing models ARIMAX and TBATS.

Structural checks and Statistical diagnostics

ARIMA

Autocorrelation check with Ljung Box test, acf and pacf plots:

```
checkresiduals(ARIMA_model1)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,3)(0,0,2)[48] with drift
## Q* = 1040.8, df = 90, p-value < 2.2e-16
##
## Model df: 6.   Total lags used: 96
```

The p-value is extremely small I reject the null hypothesis that there is no autocorrelation in the residuals and conclude that there is autocorrelation in the residuals. This means that the TBATS model hasn't fully captured the time dependent structure in the series.

Anderson-Darling normality test:

```
ad.test(residuals(ARIMA_model1))
```

```
##
##  Anderson-Darling normality test
##
## data:  residuals(ARIMA_model1)
## A = 1604.7, p-value < 2.2e-16
```

The p-value is very small, I reject the null hypothesis that the residuals are normally distributed and conclude that the residuals are non-normal.

ARCH test on residuals:

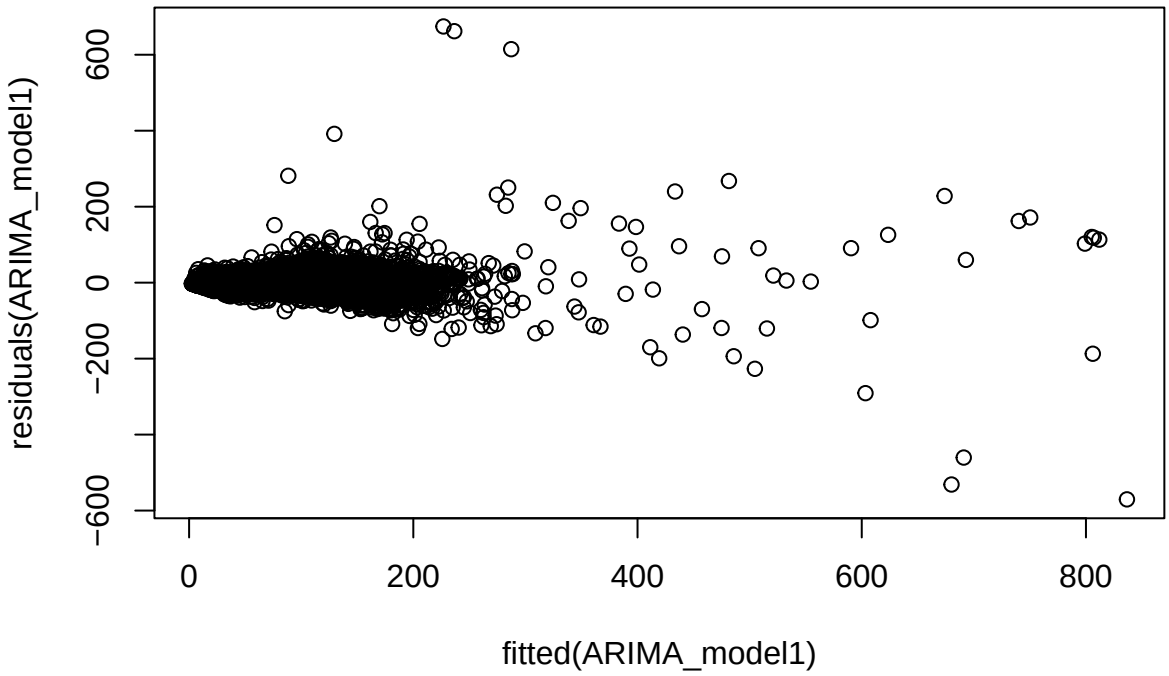
```
ArchTest(residuals(ARIMA_model1), lags = 48)
```

```
##
##  ARCH LM-test; Null hypothesis: no ARCH effects
##
## data:  residuals(ARIMA_model1)
## Chi-squared = 3182.4, df = 48, p-value < 2.2e-16
```

The p-value is much smaller than 0.05 which means I reject H0 (that there is no ARCH effect (residuals are homoskedastic)) and conclude that there is evidence of heteroskedasticity

Checking for constant variance in plot:

```
plot(residuals(ARIMA_model1) ~ fitted(ARIMA_model1))
```



As the fitted values increase, the residuals fan out, showing increasing variance.

There's a tight cluster around zero residuals for lower fitted values.

Due to these results I'm going to try fitting an ARIMA-GARCH model and see how its forecasting accuracy compares to my ARIMA and STL-ARIMA models. I'm choosing a GARCH model because its useful for capturing volatility clustering in time series data, particularly when the residuals show heteroskedasticity. By modelling the conditional variance it accounts for the changes in volatility over time.

Checking mean is reasonably close to zero with a one sample t-test:

```
t.test(residuals(ARIMA_model1), mu = 0)
```

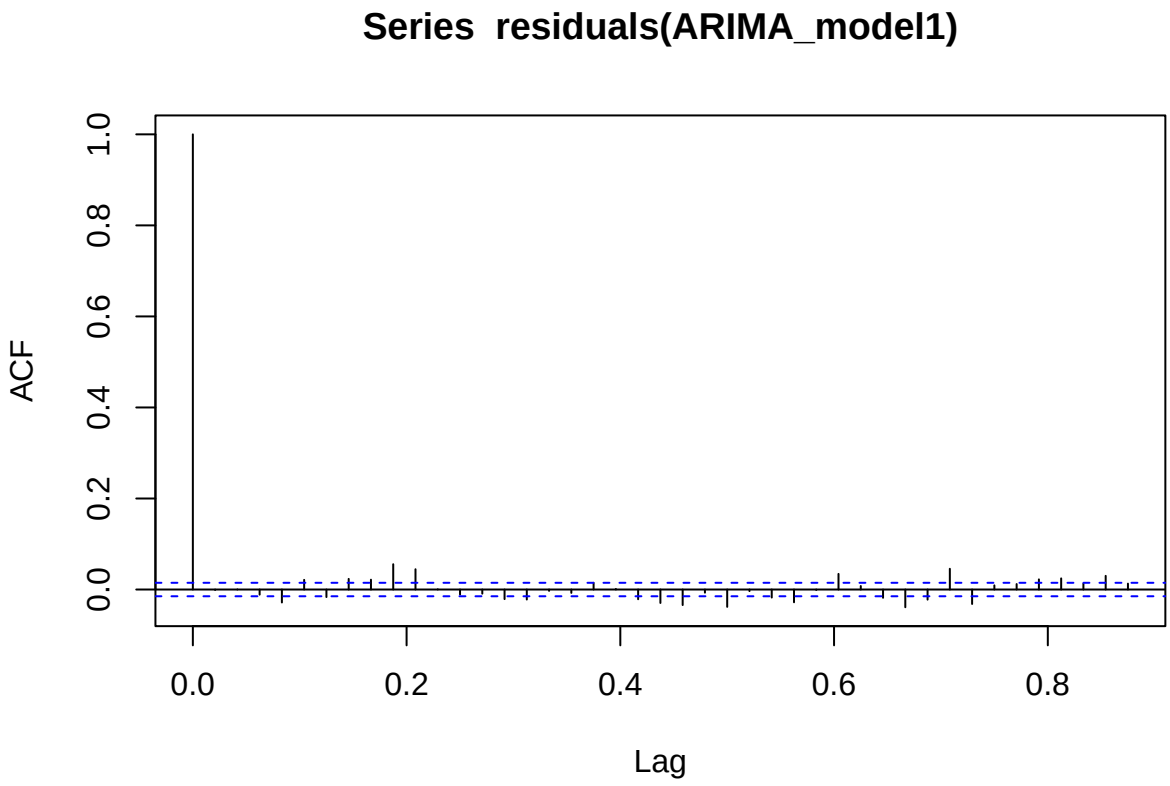
```
##
##  One Sample t-test
##
## data:  residuals(ARIMA_model1)
## t = 0.042739, df = 17519, p-value = 0.9659
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.2825493  0.2951456
## sample estimates:
##   mean of x
## 0.006298128
```

The mean of the residuals is 0.023

The p-value of the one sample t-test is 0.8714, there is not enough evidence to reject the null hypothesis, I conclude that the mean of the residuals is equal to zero.

This tells me the model is unbiased, with residuals (errors) centered around 0. It also tells me that the models forecasts aren't systematically over or under predicting.

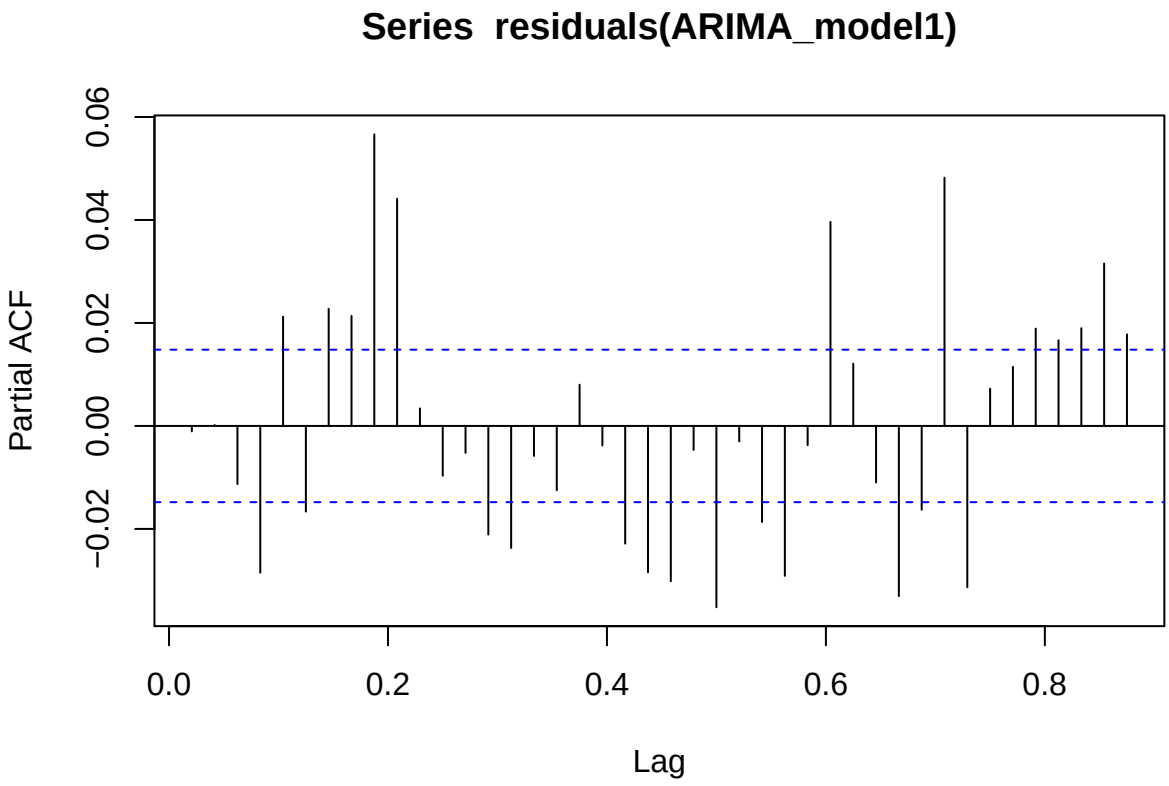
```
acf(residuals(ARIMA_model1))
```



Most autocorrelation bars fall within the dashed blue confidence bands, suggesting they're not statistically significant.

There's no strong pattern of lingering autocorrelation. The residuals are behaving like white noise.

```
pacf(residuals(ARIMA_model1))
```



Most lags fall within the blue dashed confidence bands, meaning their partial autocorrelations aren't statistically significant.

There are a few spikes outside of the confidence bounds, particularly at lag 1 suggesting short-term autocorrelation. This implies that the AR term (p) is slightly under-specified. I am going to try increasing the AR component to try and capture the remaining autocorrelation.

Checking that the time series is invertible (eg. if its errors can be represented as a weighted sum of past observations)

```
ARIMA_model1$coef
```

```
##         ar1         ma1         ma2         ma3        sma1        sma2
##  0.78824593 -1.09615215  0.18254135 -0.06326365  0.12865843  0.02741802
##       drift
##  0.00212466
```

The MA term ma1 and seasonal MA terms sma1 and sma2 all have absolute values less than 1, so the invertibility condition is satisfied.

However the MA term ma1 is -0.97 which is quite close to 1. This isn't too concerning since the residuals don't appear to have any autocorrelation or patterns, and there was no warning from R when fitting the model (R can give a warning "possible non-invertible MA part")

I'm going to check the roots of the MA polynomial (All roots should be outside the unit circle (modulus > 1) for the model to be invertible if the model violates invertibility).

```
ma_coefs <- c(1, -ARIMA_model1$coef["ma1"])  # Inverting sign for root checking
Mod(polyroot(ma_coefs))  # Gives modulus of each root (polyroot gives the complex roots)
```

```
## [1] 0.9122821
```

I'm checking the seasonal MA separately from the non seasonal MA because the invertibility condition must hold for both polynomials (non seasonal and seasonal) independently.

```
sma_coefs <- c(1, -ARIMA_model1$coef["sma1"], -ARIMA_model1$coef["sma2"])
Mod(polyroot(sma_coefs))
```

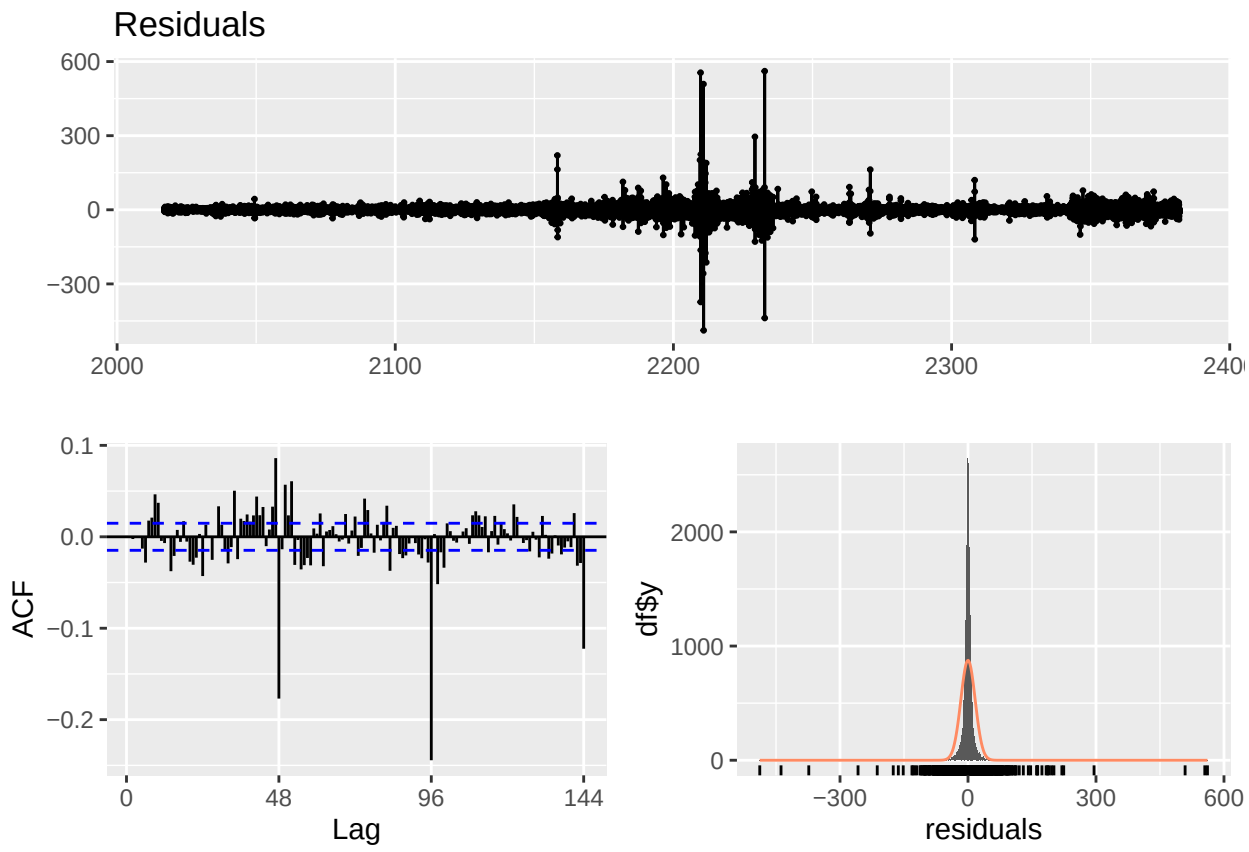
```
## [1] 4.132743 8.825220
```

For invertibility, the modulus (absolute value) of all roots must be larger than 1.

All of these are well outside the unit circle (i.e. modulus > 1), which means the ARIMA model satisfies the invertibility condition for both the non-seasonal and seasonal MA components.

STL-ARIMA

```
checkresiduals(STL_ARIMA_model1)
```



```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 2618.3, df = 96, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 96
```

The p-value is extremely small I reject the null hypothesis that there is no autocorrelation in the residuals and conclude that there is autocorrelation in the residuals. This means that the TBATS model hasn't fully captured the time-dependent structure in the series.

Anderson-Darling normality test:

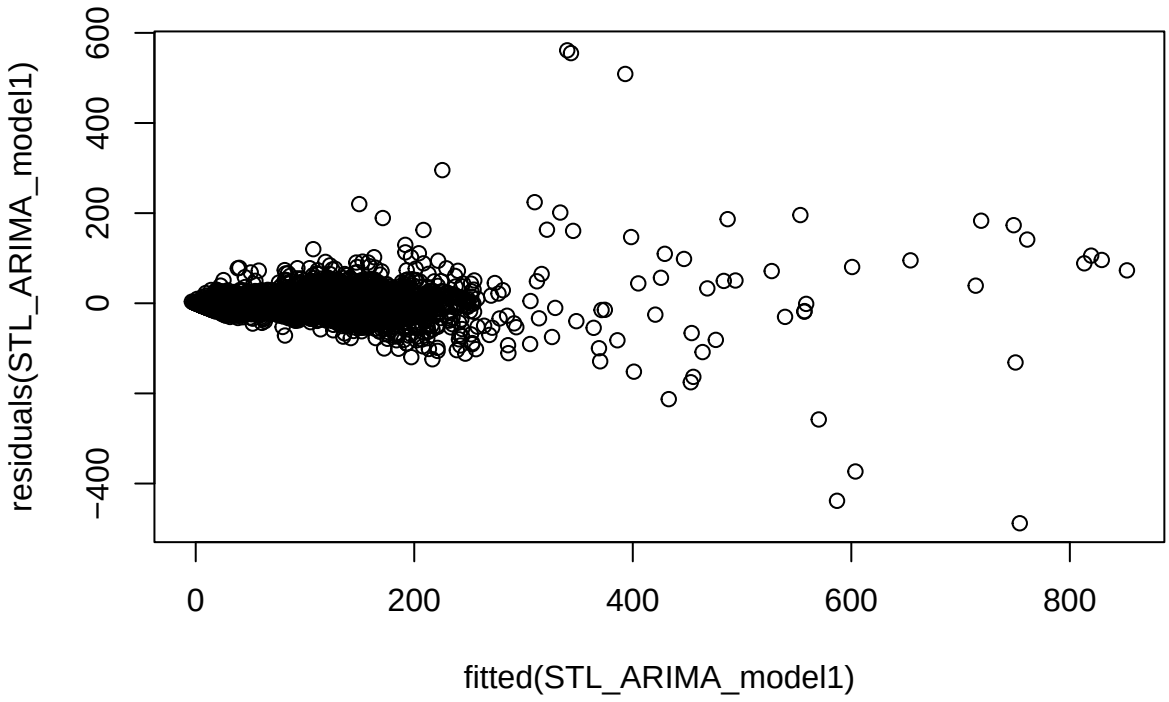
```
ad.test(residuals(STL_ARIMA_model1))
```

```
##
##  Anderson-Darling normality test
##
## data:  residuals(STL_ARIMA_model1)
## A = 1479.2, p-value < 2.2e-16
```

The p-value is very small, I reject the null hypothesis that the residuals are normally distributed and conclude that the residuals are non-normal.

Checking for constant variance and mean zero in plot:

```
plot(residuals(STL_ARIMA_model1) ~ fitted(STL_ARIMA_model1))
```



As the fitted values increase, the residuals fan out, showing increasing variance.

There's a tight cluster around zero residuals for lower fitted values.

Checking mean is reasonably close to zero with a one sample t-test:

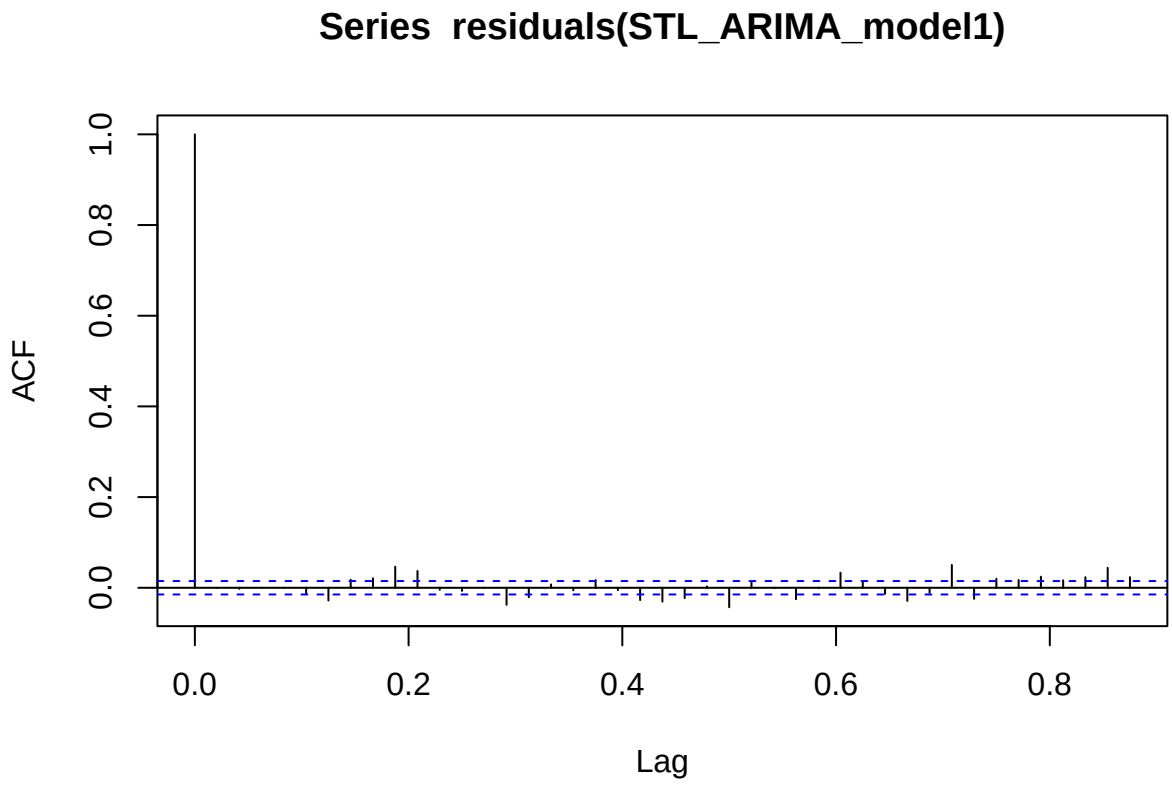
```
t.test(residuals(STL_ARIMA_model1), mu = 0)
```

```
##
##  One Sample t-test
##
## data:  residuals(STL_ARIMA_model1)
## t = 0.12333, df = 17519, p-value = 0.9018
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.2324401  0.2636542
## sample estimates:
##  mean of x
## 0.01560702
```

The mean of the residuals is 0.015

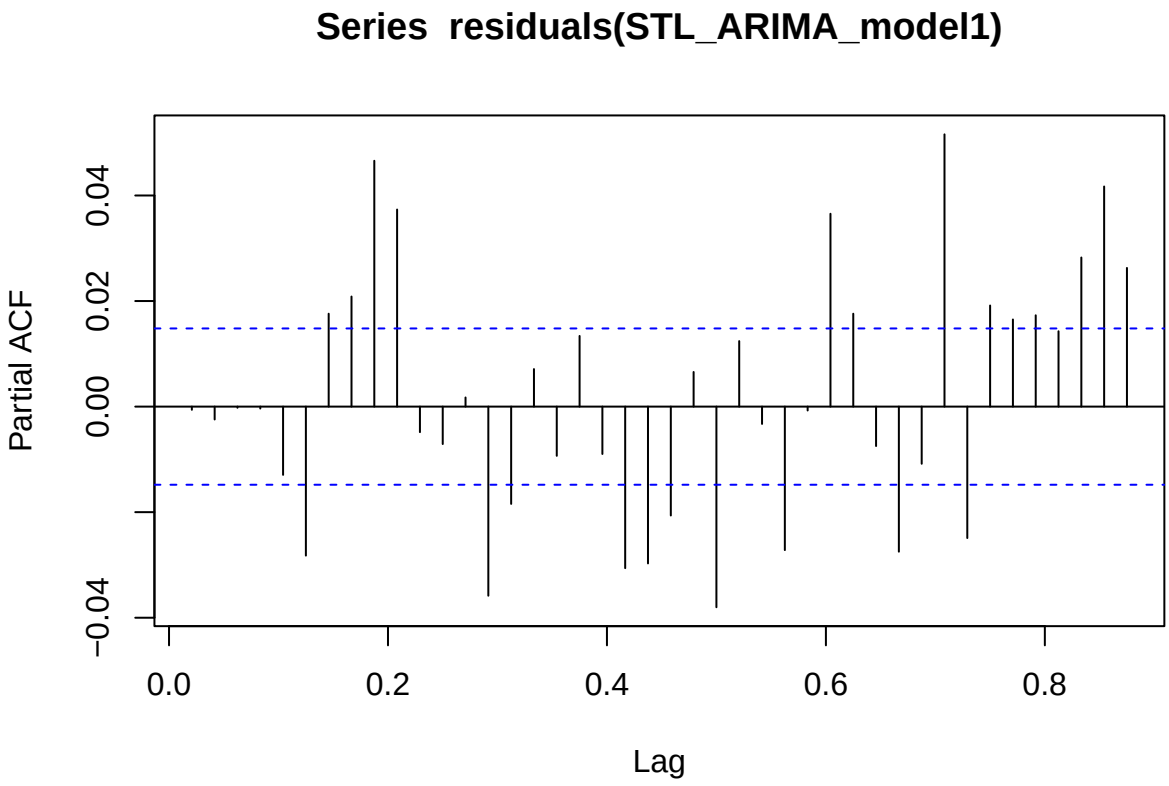
The p-value of the one sample t-test is 0.9019, there is not enough evidence to reject the null hypothesis, I conclude that the mean of the residuals is equal to zero.

```
acf(residuals(STL_ARIMA_model1))
```



The models residuals are like white noise, most of the bars are within the confidence bounds.

```
pacf(residuals(STL_ARIMA_model1))
```



There are significant spikes outside the confidence bands, this suggests that there are lags that aren't captured by my current AR structure.

I am going to increase the AR component for the model.

Checking that the time series is invertible (eg. if its errors can be represented as a weighted sum of past observations)

```
arima_part <- STL_ARIMA_model1$model
arima_part$coef
```

```
##         ar1         ar2         ar3         ar4         ar5         ma1
##  0.62597389  0.14893563 -0.03100311 -0.03780019  0.05064457 -0.96131232
```

The MA term ma1 has an absolute value of less than 1, so the invertibility condition is satisfied.

However the MA term ma1 is -0.96 which is quite close to 1. This isn't too concerning since the residuals don't appear to have any autocorrelation or patterns, and there was no warning from R when fitting the model.

I'm going to check the roots of the MA polynomial (All roots should be outside the unit circle (modulus > 1) for the model to be invertible if the model violates invertibility).

```
ma_coefs <- c(1, -arima_part$coef["ma1"])  # Inverting sign for root checking
Mod(polyroot(ma_coefs))  # Gives modulus of each root (polyroot gives the complex roots)
```

```
## [1] 1.040245
```

For invertibility, the modulus (absolute value) of all roots must be larger than 1.

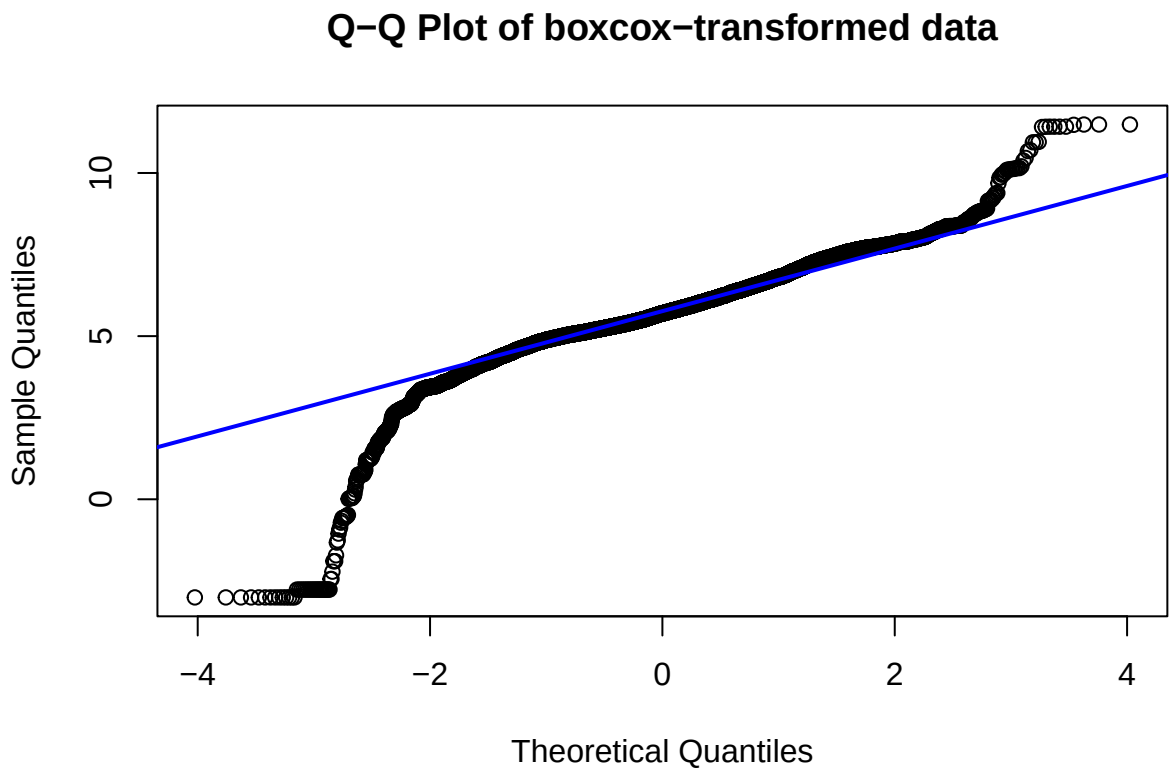
Since the absolute value of the root is outside the unit circle (i.e. modulus > 1), this means the ARIMA part of the STL-ARIMA model satisfies the invertibility condition for both the MA components.

Model refinement

Applying a box cox transformation to the data to deal with non-normality:

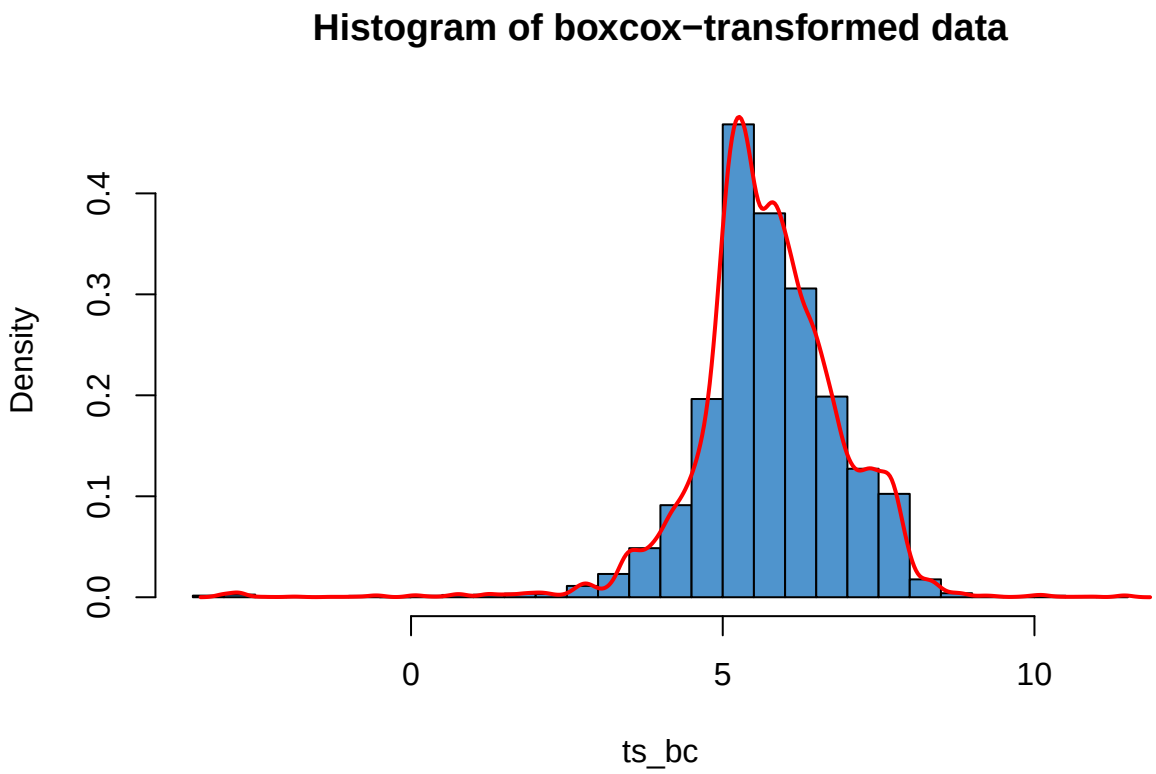
```
lambda <- BoxCox.lambda(time_series_data)
ts_bc <- BoxCox(time_series_data, lambda)
```

```
qqnorm(ts_bc, main = "Q-Q Plot of boxcox-transformed data")
qqline(ts_bc, col = "blue", lwd = 2)
```



There is a curved pattern in the data which indicates that the data is skewed or is not normally distributed.

```
hist(ts_bc, breaks = 50, probability = TRUE,
     main = "Histogram of boxcox-transformed data", col = "steelblue3")
lines(density(ts_bc), col = "red", lwd = 2)
```



The data is not centered around zero, so its not a standard normal distribution.

The data appears to be right skewed.

The data is not normally distributed, however the box-cox transformed data is closer to a normal distribution than the untransformed data, so I will be using it for further modelling.

Anderson-Darling normality test:

```
ad.test(ts_bc)
```

```
##
##  Anderson-Darling normality test
##
## data:  ts_bc
## A = 145.74, p-value < 2.2e-16
```

The small p-value indicates that the data is still not normal even after a BoxCox transformation.

Checking for outliers with Tukey's Fences:

```
y <- as.numeric(time_series_data)

# Computing the basic boxplot stats
stats <- boxplot.stats(y)$stats  # Q1, median, Q3, ...
iqr   <- stats[3] - stats[1]     # Q3 - Q1
```

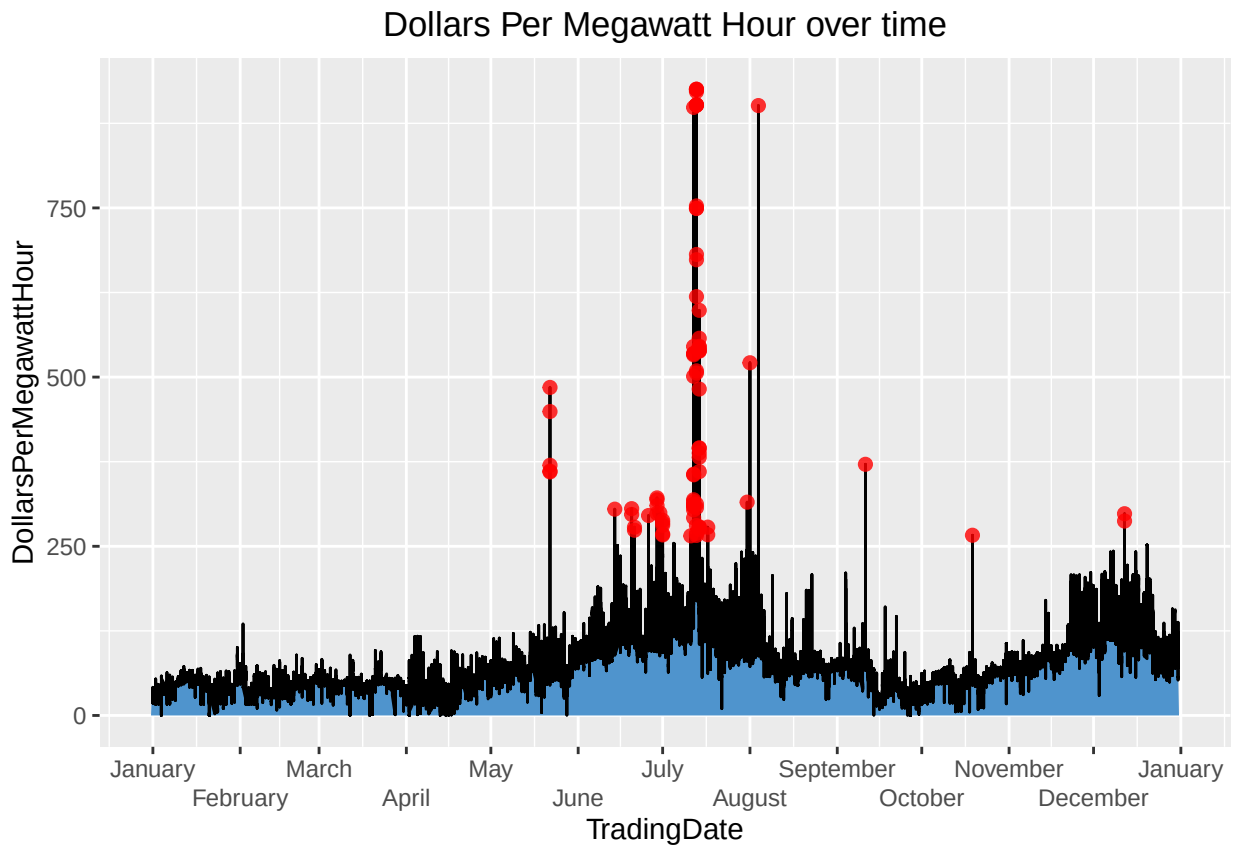
```r
# Defining the outer fences
lower <- stats[1] - 3*iqr
upper <- stats[3] + 3*iqr

# Finding the outliers
outliers_global <- which(y < lower | y > upper)

length(outliers_global)
```

```
## [1] 84
```

There are 84 outliers according to Tukey's Fences.

Visualizing the outliers on a time series plot:

```r
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)) +
  ggtitle("Dollars Per Megawatt Hour over time") +
  # Adding red points for outliers
  geom_point(data = Training_set[outliers_global, ], aes(x = TradingDate, y = DollarsPerMegawattHour),
             color = "red", size = 2, alpha = 0.8) +
  theme(plot.title = element_text(hjust = 0.5))
```



There are outliers in the series but since energy prices are so volatile, I don't want to remove them, I am choosing not to discard the outliers in the series.

Refitting models:

For both models I have increased AR component by 1, and applied a box-cox transformation to the time series data.

Fitting ARIMA:

```
ARIMA_model2 <- Arima(ts_bc, order = c(4,1,1), # Non-seasonal ARIMA(p,d,q)
             seasonal = list(order = c(0, 0, 2), period = 48),  # Seasonal ARIMA(P,D,Q)[S]
             include.constant = FALSE)  # Excluding constant since first order differencing is used
```

Fitting an STL - ARIMA to seasonally adjusted series:

```
STL_ARIMA_model2 <- stlm(ts_bc, s.window = 48, robust = TRUE,
                         modelfunction = function(x) Arima(x, order = c(6, 1, 1)))
```

For both of these models I have only increased the AR component by 1, and I have not applied a box-cox transformation to the time series data.

Fitting ARIMA:

```
ARIMA_model3 <- Arima(time_series_data, order = c(4,1,1), # Non-seasonal ARIMA(p,d,q)
             seasonal = list(order = c(0, 0, 2), period = 48),  # Seasonal ARIMA(P,D,Q)[S]
             include.constant = FALSE)  # Excluding constant since first order differencing is used
```

Fitting an STL - ARIMA to seasonally adjusted series:

```
STL_ARIMA_model3 <- stlm(time_series_data, s.window = 48, robust = TRUE,
                         modelfunction = function(x) Arima(x, order = c(6, 1, 1)))
```

Fitting an ARIMA-GARCH model:

```
spec <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  mean.model     = list(armaOrder = c(4, 1), include.mean = TRUE), # AR(4) and MA(1)
  distribution.model = "norm"
)
```

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).

If the GARCH model doesn't have high forecasting accuracy I'll try setting distribution.model to "std" for Student-t (for heavy tails) and adjusting the GARCH order

```
GARCH_model <- ugarchfit(spec = spec, data = first_order_diff)
```

```
show(GARCH_model)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(4,0,1)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error     t value Pr(>|t|)
## mu     -0.012814    0.001324 -9.6789e+00 0.000000
## ar1     0.767472    0.009059  8.4720e+01 0.000000
## ar2     0.074173    0.011496  6.4519e+00 0.000000
## ar3    -0.032934    0.011315 -2.9107e+00 0.003606
## ar4     0.001917    0.008982  2.1338e-01 0.831031
## ma1    -0.976206    0.000019 -5.2341e+04 0.000000
## omega   1.196619    0.122106  9.7999e+00 0.000000
## alpha1  0.090520    0.003670  2.4663e+01 0.000000
## beta1   0.908480    0.004036  2.2508e+02 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error     t value Pr(>|t|)
## mu     -0.012814    0.001415 -9.0567e+00 0.000000
## ar1     0.767472    0.020887  3.6744e+01 0.000000
## ar2     0.074173    0.024971  2.9704e+00 0.002974
## ar3    -0.032934    0.018123 -1.8172e+00 0.069181
## ar4     0.001917    0.017661  1.0852e-01 0.913586
## ma1    -0.976206    0.000032 -3.0482e+04 0.000000
## omega   1.196619    1.106954  1.0810e+00 0.279697
## alpha1  0.090520    0.032790  2.7606e+00 0.005769
## beta1   0.908480    0.040534  2.2413e+01 0.000000
##
## LogLikelihood : -66579.4
##
## Information Criteria
## ------------------------------------
##
## Akaike       7.6018
## Bayes        7.6058
## Shibata      7.6018
## Hannan-Quinn 7.6032
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic   p-value
## Lag[1]                      17.83 2.422e-05
## Lag[2*(p+q)+(p+q)-1][14]    29.20 0.000e+00
## Lag[4*(p+q)+(p+q)-1][24]    34.78 2.071e-09
## d.o.f=5
## H0 : No serial correlation
```

```
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic  p-value
## Lag[1]                      7.002 0.008141
## Lag[2*(p+q)+(p+q)-1][5]     7.880 0.031564
## Lag[4*(p+q)+(p+q)-1][9]     8.532 0.101158
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3] 0.0009509 0.500 2.000  0.9754
## ARCH Lag[5] 0.0701050 1.440 1.667  0.9920
## ARCH Lag[7] 0.4833668 2.315 1.543  0.9799
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  12.437
## Individual Statistics:
## mu     0.30502
## ar1    1.38657
## ar2    1.30012
## ar3    0.96504
## ar4    0.24139
## ma1    0.08613
## omega  1.41219
## alpha1 1.24376
## beta1  0.78486
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          2.1 2.32 2.82
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value      prob sig
## Sign Bias           2.9968 2.732e-03 ***
## Negative Sign Bias  0.1041 9.171e-01
## Positive Sign Bias  3.5326 4.126e-04 ***
## Joint Effect       32.0167 5.191e-07 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20      3393            0
## 2    30      3511            0
## 3    40      3618            0
## 4    50      3710            0
##
##
## Elapsed time : 2.249202
```

I have: sGARCH(1,1) and ARFIMA(4,1,1) (note d = 1 because I fit first order differencing to data prior to fitting model)

Comparing the simple ARIMA-GARCH with ARIMA-GARCH models with higher GARCH and ARCH terms.

```
spec2 <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 2)),
  mean.model     = list(armaOrder = c(4, 1), include.mean = TRUE), # AR(4) and MA(1)
  distribution.model = "norm"
)
GARCH_model2 <- ugarchfit(spec = spec2, data = first_order_diff)
```

```
spec3 <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(2, 1)),
  mean.model     = list(armaOrder = c(4, 1), include.mean = TRUE), # AR(4) and MA(1)
  distribution.model = "norm"
)
GARCH_model3 <- ugarchfit(spec = spec3, data = first_order_diff)
```

```
spec4 <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(2, 2)),
  mean.model     = list(armaOrder = c(4, 1), include.mean = TRUE), # AR(4) and MA(1)
  distribution.model = "norm"
)
GARCH_model4 <- ugarchfit(spec = spec4, data = first_order_diff)
```

Picking best GARCH model to move forward with:

```
ic1 <- infocriteria(GARCH_model)
ic2 <- infocriteria(GARCH_model2)
ic3 <- infocriteria(GARCH_model3)
ic4 <- infocriteria(GARCH_model4)
data.frame(
  Model = c("GARCH(1,1)", "GARCH(1,2)", "GARCH(2,1)", "GARCH(2,2)"),
  AIC = c(ic1[1], ic2[1], ic3[1], ic4[1]),
  BIC = c(ic1[2], ic2[2], ic3[2], ic4[2])
)
```

```
##        Model      AIC      BIC
## 1 GARCH(1,1) 7.601850 7.605842
## 2 GARCH(1,2) 7.589963 7.594399
## 3 GARCH(2,1) 7.601964 7.606400
## 4 GARCH(2,2) 7.590078 7.594957
```

The ARIMA-GARCH model with GARCH(1,2) had the lowest AIC and BIC, meaning it is the best ARIMA-GARCH model, I will move forward with that model for forecasting.

Final Forecasting and reporting

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

ARIMA 2nd model:

```
forecast12months_A <- forecast(ARIMA_model2, h = 17520)
```

Returning to the original scale (boxcox scale was used for training the ARIMA model) after forecasting to obtain interpretable electricity price predictions.

```
forecast_raw_A <- InvBoxCox(forecast12months_A$mean, lambda) - 1
```

```
forecast_raw_ARIMA2 <- ts(forecast_raw_A, start = c(2018, 1), frequency = 48)
```

STL-ARIMA 2nd model:

```
forecast12months_S <- forecast(STL_ARIMA_model2, h = 17520)
```

Returning to the original scale (boxcox scale was used for training the STL-ARIMA model) after forecasting to obtain interpretable electricity price predictions.

```
forecast_raw_S <- InvBoxCox(forecast12months_S$mean, lambda) - 1
```

```
forecast_raw_STL2 <- ts(forecast_raw_S, start = c(2018, 1), frequency = 48)
```

STL-ARIMA 3rd model:

```
forecast12months_S3 <- forecast(STL_ARIMA_model3, h = 17520)
```

```
forecast_raw_STL3 <- ts(forecast12months_S3, start = c(2018, 1), frequency = 48)
```

ARIMA 3rd model:

```
forecast12months_A3 <- forecast(ARIMA_model3, h = 17520)
```

```
forecast_raw_ARIMA3 <- ts(forecast12months_A3, start = c(2018, 1), frequency = 48)
```

Forecasting ARIMA-GARCH model:

```
forecast_Garch <- ugarchforecast(GARCH_model2, n.ahead = 17520)
mean_fc_vector <- as.numeric(fitted(forecast_Garch)[ , 1])

# I first order differenced the time series before fitting the model so I'm converting back
recovered_forecast <- as.numeric(cumsum(mean_fc_vector))

# Adding the Last Observed Value of the original series:
last_value <- as.numeric(tail(time_series_data, 1))
final_forecast <- last_value + recovered_forecast
forecast_Garch2 <- ts(final_forecast, start = c(2018, 1), frequency = 48)
```

Overall Model Comparison

```
ARIMA_acc2 <- accuracy(forecast_raw_ARIMA2, Test_time_series)
STL_ARIMA_acc2 <- accuracy(forecast_raw_STL2, Test_time_series)
ARIMA_acc3 <- accuracy(forecast_raw_ARIMA3, Test_time_series)
STL_ARIMA_acc3 <- accuracy(forecast_raw_STL3, Test_time_series)
ARIMA_GARCH_acc <- accuracy(forecast_Garch2, Test_time_series)
```

```
acc_list2 <- list(ARIMA = ARIMA_acc, ARIMA_2 = ARIMA_acc2, ARIMA_3 = ARIMA_acc3,
                  STL_ARIMA = STL_ARIMA_acc, STL_ARIMA_2 = STL_ARIMA_acc2, STL_ARIMA_3 = STL_ARIMA_acc3
                  ARIMA_GARCH = ARIMA_GARCH_acc)

acc_df2 <- map_df(acc_list2, ~ as.data.frame(.x)["Test set", ], .id = "Model")
```

Manually calculating the MASE values for box-cox ARIMA and STL-ARIMA models and the ARIMA-GARCH model. Also calculating the ACF1 for the ARIMA-GARCH model.

```
# 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(Test_time_series - as.numeric(forecast_Garch2)))

# MASE for ARIMA GARCH
mase_garch <- mae_garch / mae_naive

# ARIMA
naive_errors <- diff(time_series_data)
mae_naive <- mean(abs(naive_errors))
mase_ARIMA <- mean(abs(Test_time_series - as.numeric(forecast_raw_ARIMA3$mean)))
mase_ARIMA <- mase_ARIMA / mae_naive

# STL-ARIMA
naive_errors <- diff(time_series_data)
mae_naive <- mean(abs(naive_errors))
mase_STL <- mean(abs(Test_time_series - as.numeric(forecast_raw_STL3$mean)))
mase_STL <- mase_STL / mae_naive

# ARIMA-GARCH ACF1
residuals_garch <- Test_time_series - as.numeric(forecast_Garch2)
acf_result <- acf(residuals_garch, lag.max = 1, plot = FALSE)
acf1_value <- acf_result$acf[2]  # Index 2 = lag-1 autocorrelation
```

```
acc_df2[2, 7] <- mase_ARIMA
acc_df2[5, 7] <- mase_STL
acc_df2[7, 7] <- mase_garch
acc_df2[7, 8] <- acf1_value
```

```
acc_df2
```

```
##                   Model         ME       RMSE        MAE          MPE       MAPE
```

```
## Test set...1       ARIMA   6.658897  19.20814  13.71418     3.405619   15.63555
## Test set...2     ARIMA_2  18.094073 102.20348  52.86636 -2086.335834 2114.56889
## Test set...3     ARIMA_3   6.564999  19.17508  13.70700     3.293977   15.65053
## Test set...4   STL_ARIMA  12.108617  21.68740  17.48504    10.049182   19.23103
## Test set...5 STL_ARIMA_2  18.577036 101.73848  52.66305 -1995.217663 2024.31285
## Test set...6 STL_ARIMA_3   5.782979  19.99405  15.63605     2.514401   18.38582
## Test set...7 ARIMA_GARCH 161.822847 219.34022 163.38778  3555.285399 3763.46644
##                     MASE      ACF1 Theil's U
## Test set...1  0.7032626 0.6495645  1.393746
## Test set...2  6.5194641 0.9063878 12.284069
## Test set...3  0.7028946 0.6498515  1.394256
## Test set...4  0.8966325 0.6537431  1.503628
## Test set...5  6.5409505 0.9053458 11.674291
## Test set...6  0.8018164 0.6899861  1.514280
## Test set...7 20.1497073 0.9567283 23.474371
```

The MASE had to be manually calculated for my two of the models (ARIMA 2 and STL-ARIMA 2) presumably due to the box-cox transformation I applied to the time series for those models. I also had to manually calculate the MASE and ACF1 for my ARIMA-GARCH model.

The original ARIMA model and ARIMA model 3 had identical results which means that increasing the AR component from AR(3) to AR(4) did not significantly improve the model. The original ARIMA model and ARIMA model 3 had the best results in regards to their RMSE, MAE, MAPE, MASE and ACF1. These models seem to have strong accuracy and are reliably forecasting final energy prices for 2018.

The 2nd ARIMA model and the second STL-ARIMA model had the worst performance with extremely high RMSE and MAE. The models had the highest ACF1. Slightly higher MAE, RMSE, and MASE than ARIMA, but solid MAPE and relatively low ACF1

The ARIMA_GARCH model performed better than the box-cox ARIMA and STL-ARIMA models, but worse than my original ARIMA model in regards to the RMSE, MAE, MPE, MAPE, MASE and ACF1 values. It had a better ME value though.

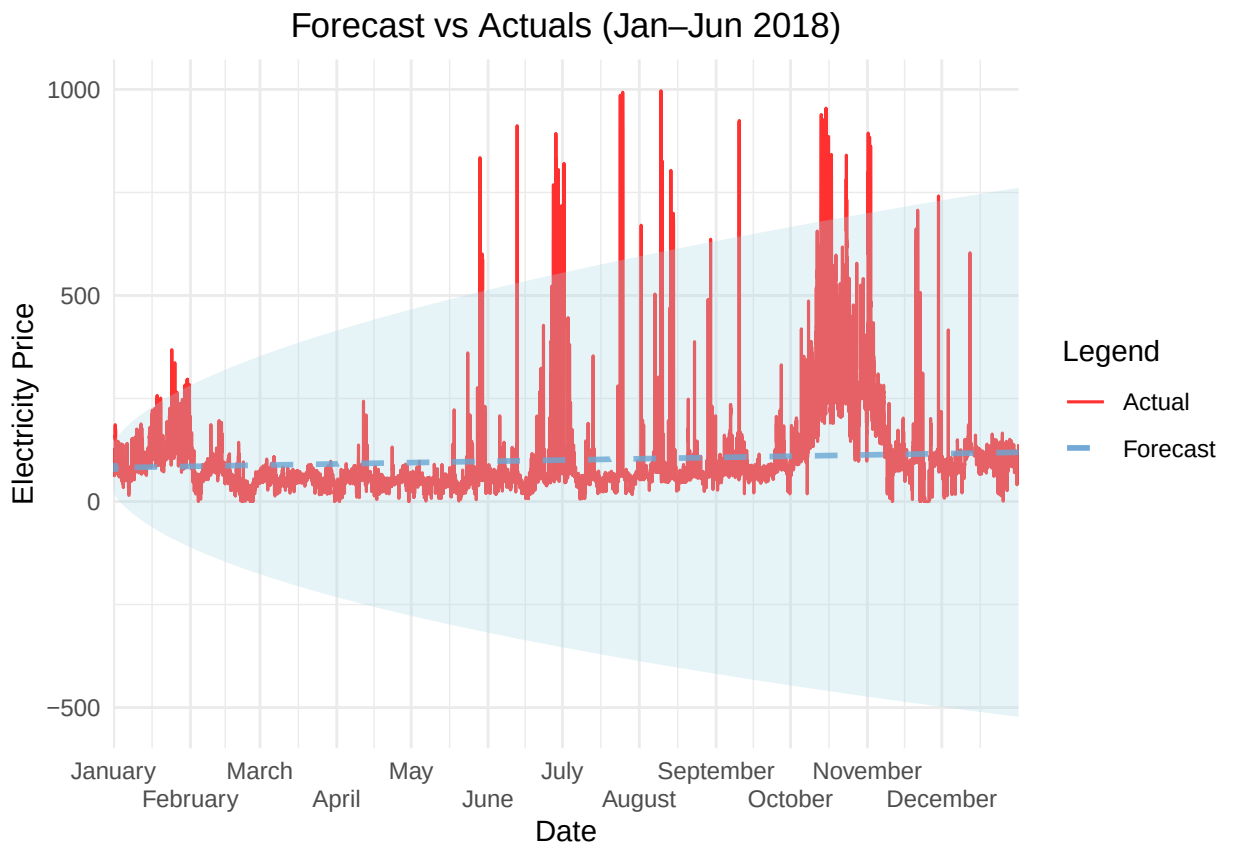
The model that performed best was my original ARIMA model.

The prior adjustments to the original ARIMA and STL-ARIMA models did not improve the ARIMA model's performance but it did improve the STL-ARIMA model's performance. The box-cox transformation of the series made the performance of the models much worse. The best model was ARIMA(3,1,1)(0,0,2)[48] and the worst models were the box-cox ARIMA and STL-ARIMA models.

```r
start_time     <- as.POSIXct("2018-01-01 00:00:00", tz = "UTC")
times_forecast <- seq(start_time, by = "30 min", length.out = 17520)

Test_time_series2 <- ts(test_data$DollarsPerMegawattHour, frequency = 48, start = c(2018, 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(forecast_raw_ARIMA$mean),
  Lower = as.numeric(forecast_raw_ARIMA$lower[,2]),  # 95% lower bound
  Upper = as.numeric(forecast_raw_ARIMA$upper[,2]),  # 95% upper bound
  Type  = "Forecast"
)
```

```r
ggplot() +
  geom_line(data = df_actual, aes(x = Date, y = Value, color = "Actual"), size = 0.55) +
```

```r
  geom_line(data = df_forecast, aes(x = Date, y = Mean, color = "Forecast"), size = 1, alpha = 0.8, lin
  geom_ribbon(data = df_forecast, aes(x = Date, ymin = Lower, ymax = Upper), fill = "lightblue", alpha =
  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(
  labs(title = "Forecast vs Actuals (Jan-Jun 2018)",
       x = "Date", y = "Electricity Price", color = "Legend") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



This plot shows my ARIMA 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 is underestimating the real volatility.

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

Due to the models issues with volatility if I were to continue forecasting the 2018 energy price data with another model I would consider a GARCH model as it is good for modelling volatility clustering.

Conclusion

The best model was the ARIMA model: ARIMA(3,1,1)(0,0,2)[48].

Its non-seasonal component had AR(3) (which captured short term autocorrelation up to lag 3), I(1) (a first order differencing to make the data stationary), MA(1) (which corrected for noise and shocks using lag 1 residuals). Its seasonal component with a periodicity of 48 (the seasonality repeats every full day as there are 48 trading periods per day) and it did not include a seasonal AR or I (no seasonal autocorrelation or differencing) and had a seasonal MA(2) (uses forecast errors from 1 and 2 days ago (lags at 48 and 96 intervals) to adjust the current days forecast).

The models mean error was 6.56 which means on average its forecasts overestimate actual prices by about 6.56 dollars per megawatt hour.

The models root mean square error was 19.17 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 13.71 which means that forecasts were off by 13.71 units on average. This is 13.6% 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 3.29% which means that the model tends to forecast higher electricity prices than reality.

The mean absolute percentage error was 15.65% which means that the models forecasts were off by 15.65% on average.

The models MASE was 0.70 which is a scaled comparison to a naive model which means the ARIMA model outperforms a naive forecast.

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 TBATS models which just 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.

References and Citations

Electricity Authority. (n.d.). Final energy prices by month [Dataset]. EMI – Electricity Market Information. Retrieved between May 2 and July 13, 2025, from https://www.emi.ea.govt.nz/Wholesale/Datasets/DispatchAndPricing/FinalEnergyPrices/ByMonth