Forecasting: Exam Assignment 2022

Fernando Delgado

4/30/2022

Contents

Introduction

Nowadays, we live in a world with an ever-changing market, and businesses must continuously adapt to meet the market needs. Thus, it is always beneficial to know what's going to happen in the future. Forecasting is the practice of foretelling future events based on past or present data.

We call a time-series a series of data points ordered through time. Whether we want to predict the prices in the stock market, or cinema box office quarterly demand, time is always an important factor to be considered. However, when performing time-series analyses, there are many aspects to consider such as, seasonality, stationarity, autocorrelation along other characteristics.

Through this report, we will describe two different time series analyses using R programming. For each analysis we use a different dataset.

Part I: Belgium Air Passenger Transport

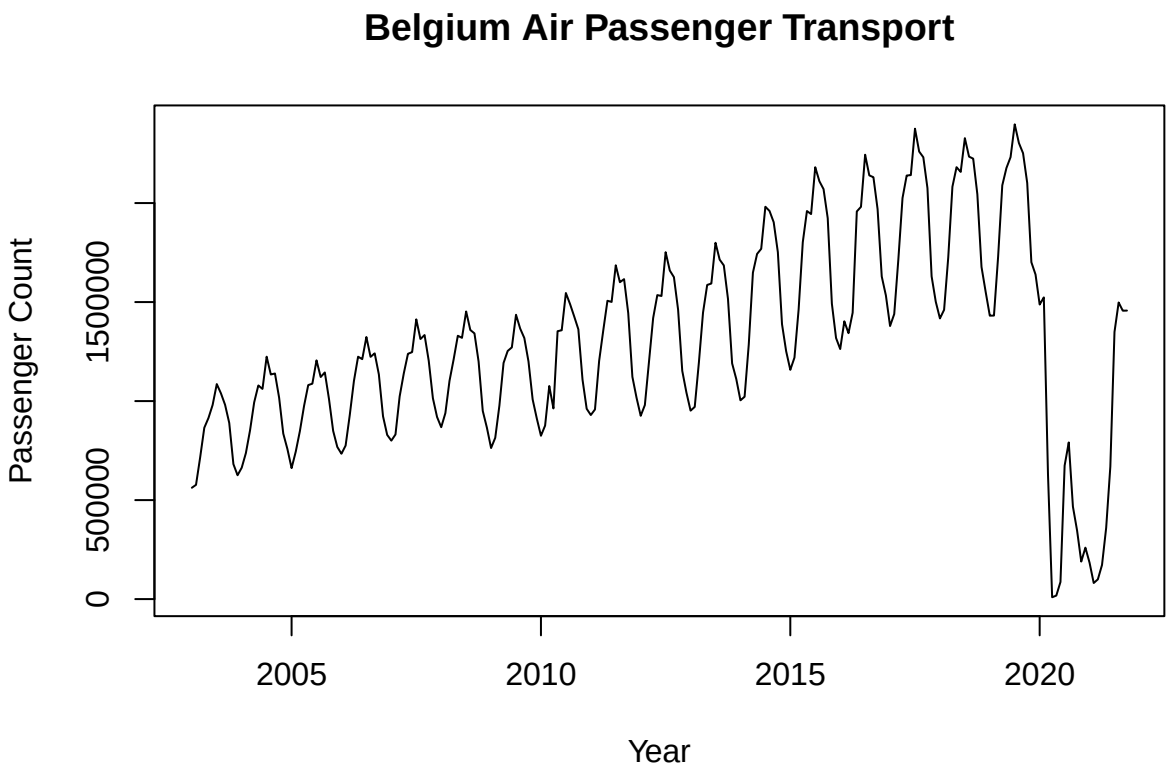
For the first section of our report, we will use the Airpass_BE dataset (https://ec.europa.eu/eurostat/databrowser/view/AVIA_PAINCC___custom_1737837/default/line?lang=en), which contains monthly international intra-EU air passenger transport by Belgium and EU partner countries, from January 2003 to October 2021.*

First we load our libraries, set a random seed, and load our data:

```
# Data manipulation
for (i in c('dplyr','tidyverse','data.table', 'readxl')){
  if (!require(i, character.only=TRUE)) +
    install.packages(i, repos = "http://cran.us.r-project.org")
  require(i, character.only=TRUE)
}

# Visualization
for (i in c('ggplot2', 'gcookbook')){
  if (!require(i, character.only=TRUE)) +
    install.packages(i, repos = "http://cran.us.r-project.org")
  require(i, character.only=TRUE)
}

# Forecasting
for (i in c('fpp2')){
  if (!require(i, character.only=TRUE)) +
    install.packages(i, repos = "http://cran.us.r-project.org")
  require(i, character.only=TRUE)
}

# Load data
data <- read_excel("./data/raw/DataSets2022.xlsx", sheet="Airpass_BE")
abe <- ts(data[,2], frequency = 12, start = 2003)

# Set Seed
set.seed(1)
```

If we take a first look at our data, we can observe that there's a strange behavior during 2020 compared to the previous years. This is most likely due to COVID-19 restrictions:

```
# Timeseries Plot
plot(abe, main="Belgium Air Passenger Transport",
     ylab="Passenger Count", xlab="Year")
```



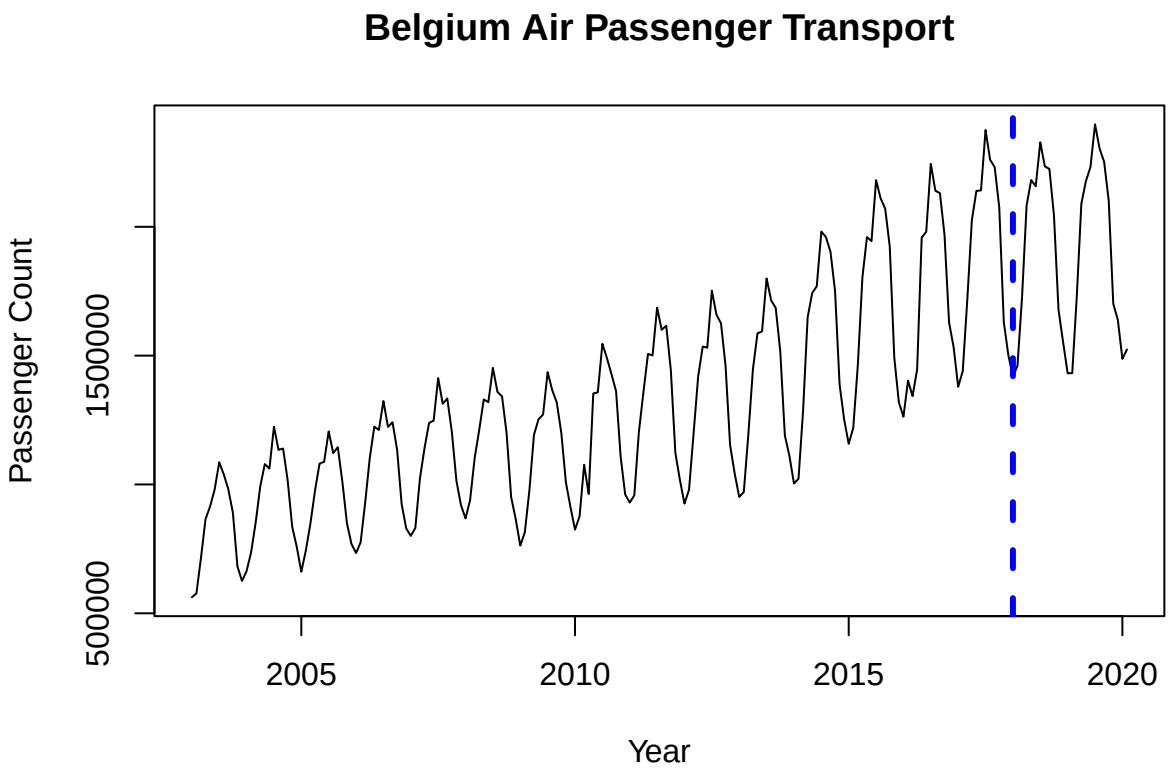
Thus, we will split the time series in a training set from January 2003 up to December 2017 and a test set from January 2018 up to February 2020. We will be using the training set for estimation of the models, and the test set for assessing their forecast accuracy. The remaining observations (dating from March 2020 to October 2021) are not included in our split, but we will keep the for future reference.

```
# Train Test Split
train <- window(abe, start = 2003, end = c(2017,12))
test <- window(abe, start = 2018, end = c(2020, 2))
full_data <- window(abe, start = 2003, end = c(2020, 2))
```

Exploratory Data Analysis

First, we take a look at the full time series we are working with (the blue line divides our train and test split):

```
# Timeseries Plot
plot(full_data, main="Belgium Air Passenger Transport",
     ylab="Passenger Count", xlab="Year") +
  abline(v = 2018, col = "blue", lty = 2, lwd = 3)
```

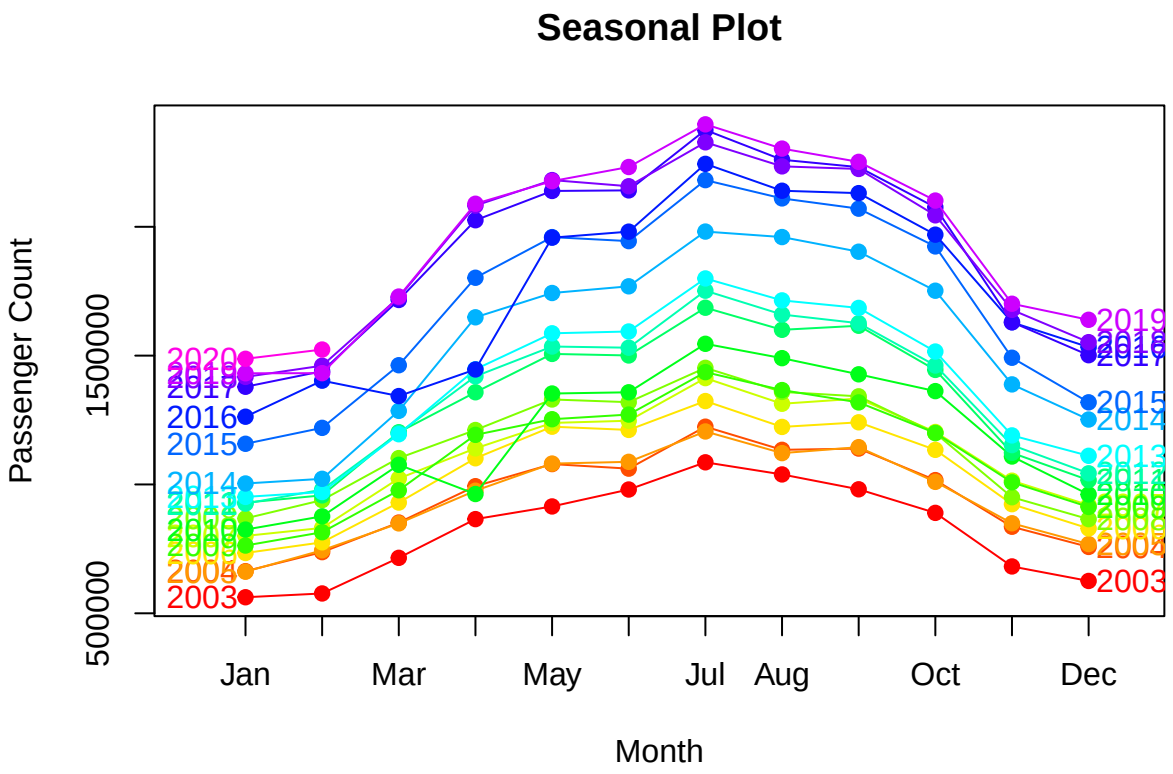



```
## integer(0)
```

From a first glance, we could say that there is a clear trend and seasonality in our time series. Seasonality refers to periodic fluctuations, such as people increasingly traveling by plane during summer for vacations and decreasing for winter.

To better observe seasonality, we can plot a seasonality graph:

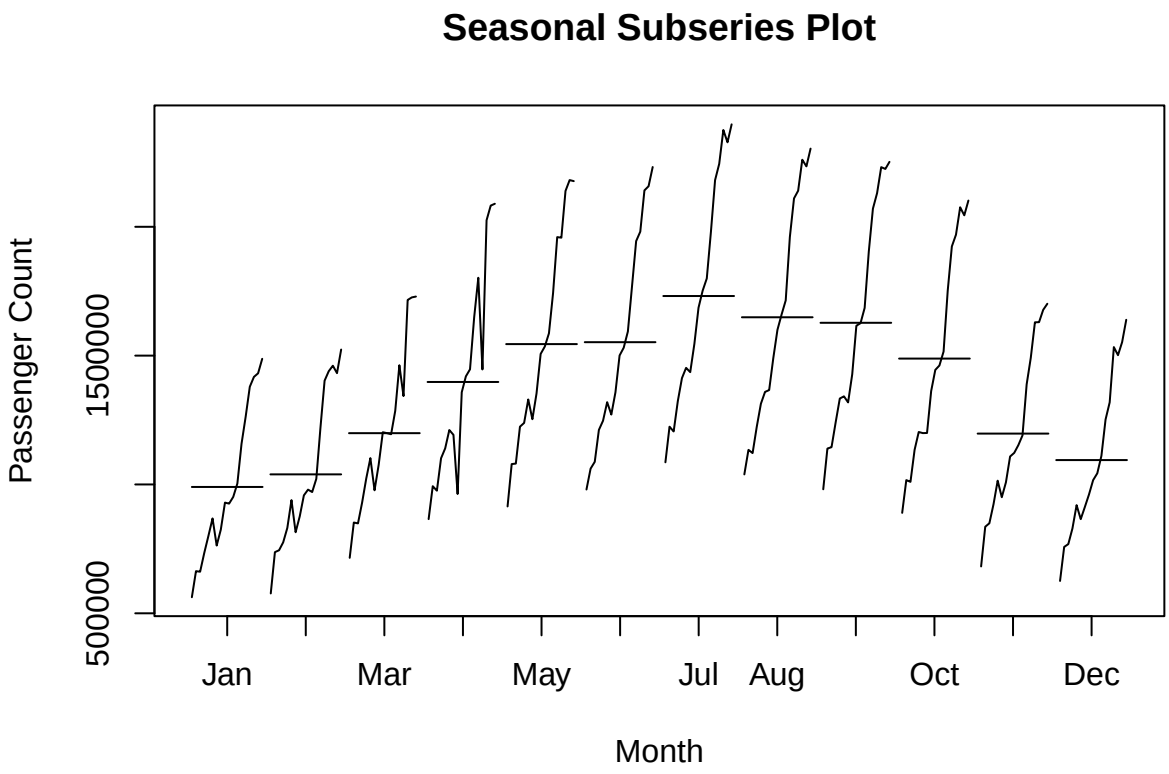
```
# Seasonal Plot
seasonplot(full_data, year.labels=TRUE, year.labels.left=TRUE,
           main="Seasonal Plot",
           ylab="Passenger Count", xlab="Month", col=rainbow(20), pch=19)
```



The seasonality plot simply overlaps each year on top of each other in order to better visualize monthly behavior. From the plot above we can observe that there is clearly an increase in passenger count during the months of June, July, and August.

Furthermore, we can also visualize our time series with a seasonal subseries plot, for a different perspective of seasonality. It emphasizes seasonal patterns for each season collected in separate smaller time plots:

```
# Seasonal subseries
monthplot(full_data, ylab="Passenger Count", xlab="Month", xaxt="n",
          main="Seasonal Subseries Plot",type="l") +
  axis(1, at=1:12, labels=month.abb, cex=0.8)
```

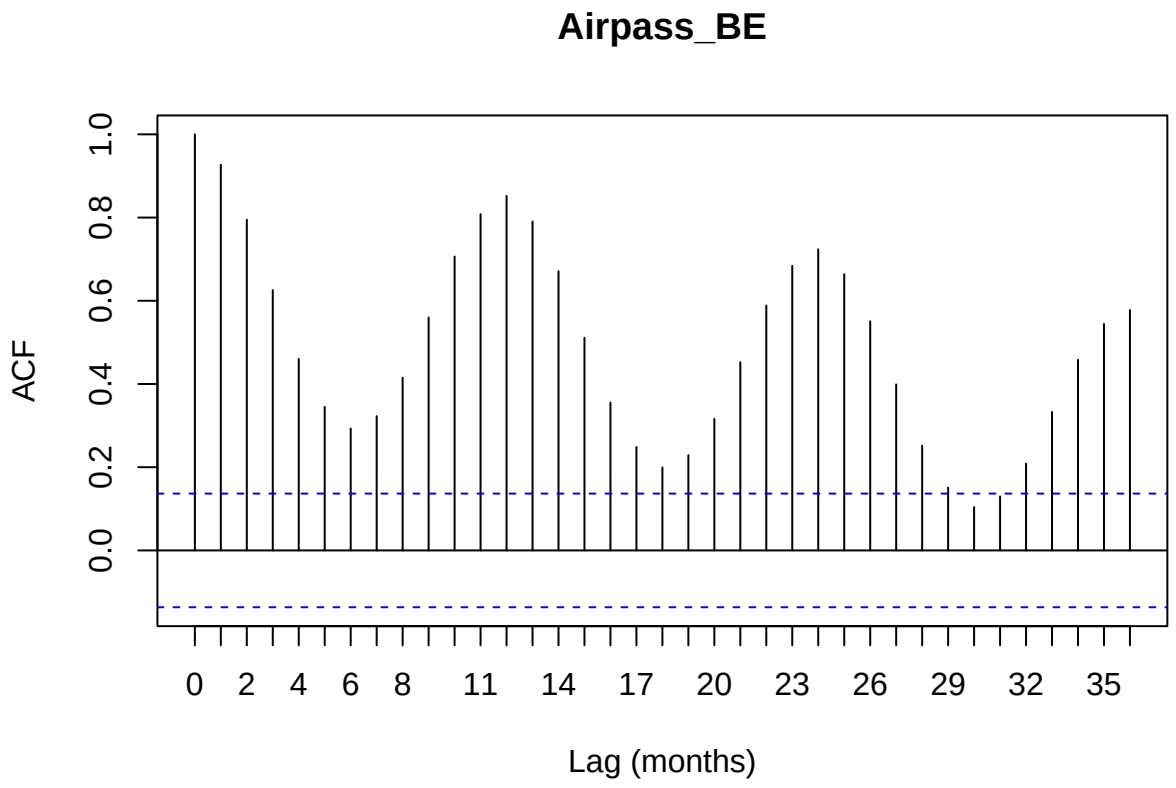



```
## numeric(0)
```

Moreover, we wish to understand the autocorrelation in our data, or in other words, the linear relationship between lagged values of a time series. An auto correlation function (ACF), also called correlogram, is a visual way to show serial correlation in data that changes over time.

We plot the ACF over a time-period of 36 months (36 lags):

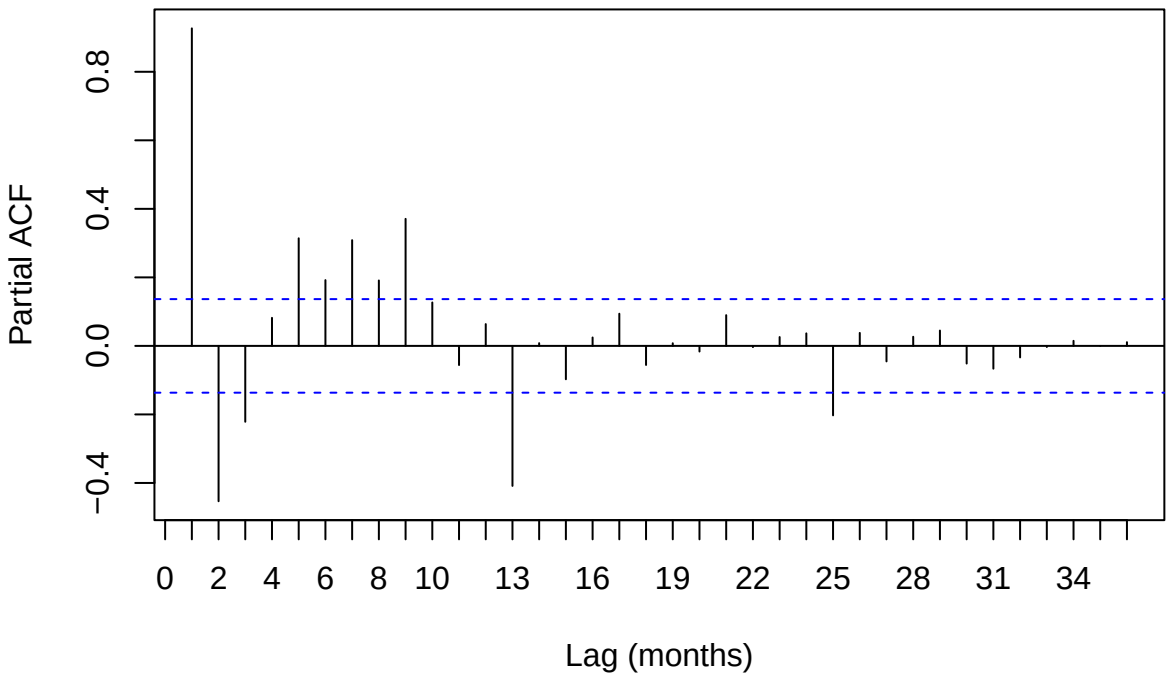
```
# ACF
mx=36
acf(full_data, lag.max=mx, xaxt="n", xlab="Lag (months)")
axis(1, at=0:mx/12, labels=0:mx)
```



We can observe high peaks at lag 0,12,24, and 36, indicating a seasonality of length 12, which makes sense since we are dealing with monthly data.

Additionally, we perform a partial autocorrelation function (PACF), which is a measure of the correlation between observations of a time series that are separated by K time units, after adjusting the presence of all the other terms of shorter lag.

```
# Partial ACF
pacf(full_data, mx, xaxt="n", xlab="Lag (months)")
axis(1, at=0:mx/12, labels=0:mx)
```



To interpret the PACF, we examine the spikes at each lag to determine whether they are significant or not, by observing if they extend beyond the significant limits (blue lines). We can clearly observe a significant correlation on the 1st, 13th, and 25th lag (representing the first month of the year), which was expected from our previous exploratory plots. However, we can also observe other lags, such as 5, 7, and 9 having a significant correlation.

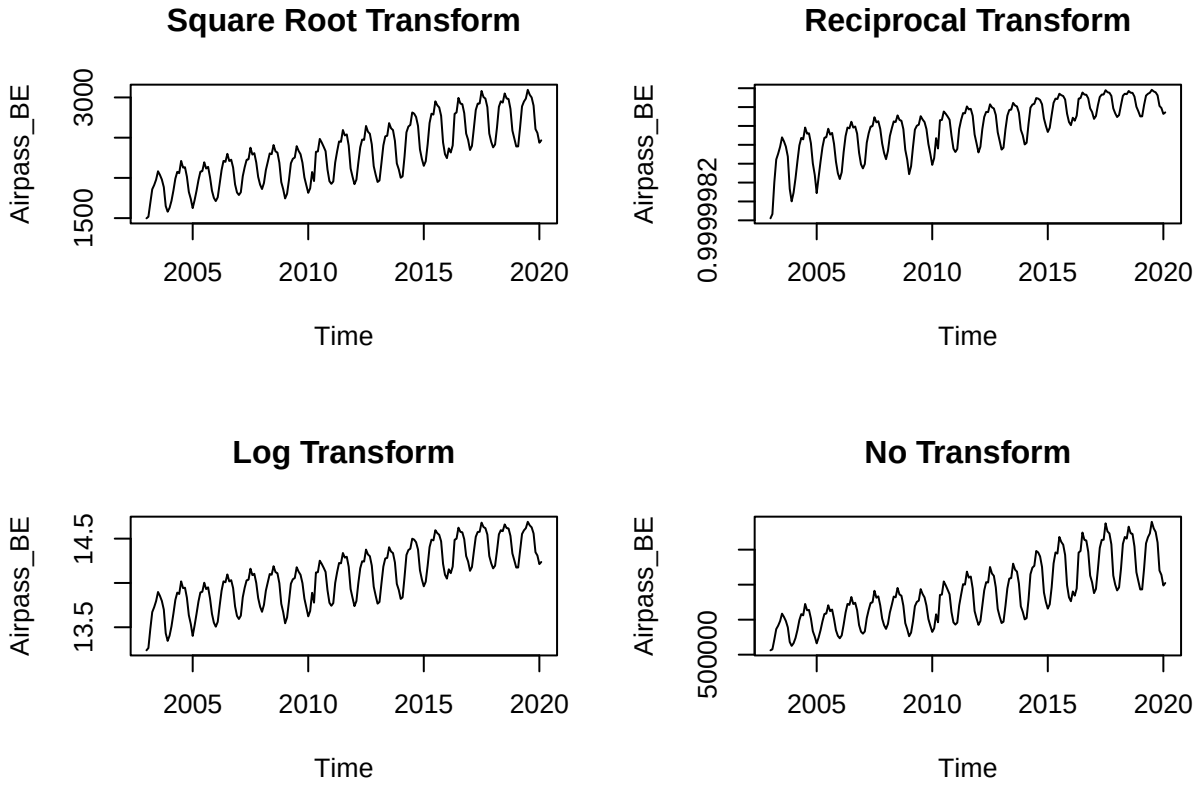
Transformations

We can remove noise and improve time series forecasting by transforming our data. Since our data is non-stationary and has a seasonal component, it could be difficult to model with forecasting methods like ARIMA without transforming.

The Box-Cox transform is a data transform method that supports square root and log transformations. It can be configured to evaluate a suite of transforms automatically and select a best fit.

In R, we can simply change the value of lambda to obtain different transformations. For example, a Lambda of -1 will give us a reciprocal transform:

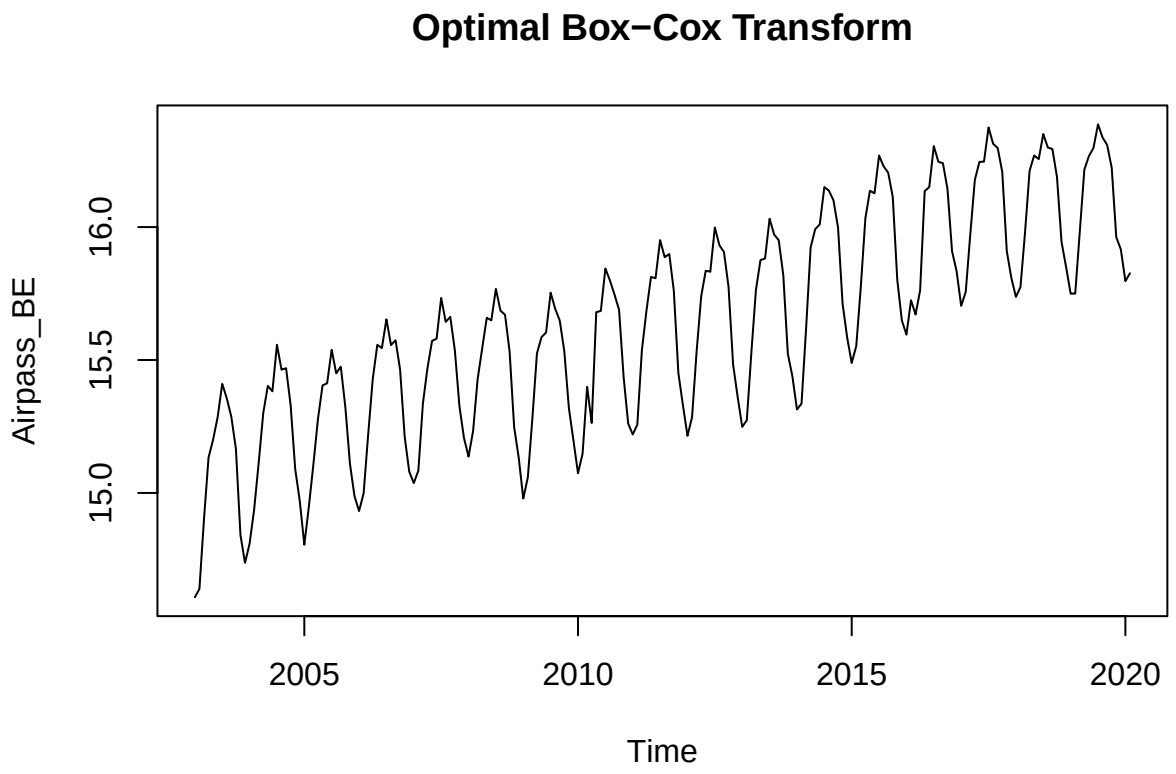
```
# Box-Cox Examples
par(mfrow=c(2,2)) # Show 4 plots together
plot(BoxCox(full_data,lambda=0.5), main = "Square Root Transform")
plot(BoxCox(full_data,lambda= -1), main = "Reciprocal Transform")
plot(BoxCox(full_data,lambda=0), main = "Log Transform")
plot(BoxCox(full_data,lambda=1), main = "No Transform")
```



Thankfully for us, we can find the most optimal value for lambda with R by using:

```
# Get lambda
l <- BoxCox.lambda(full_data)
```

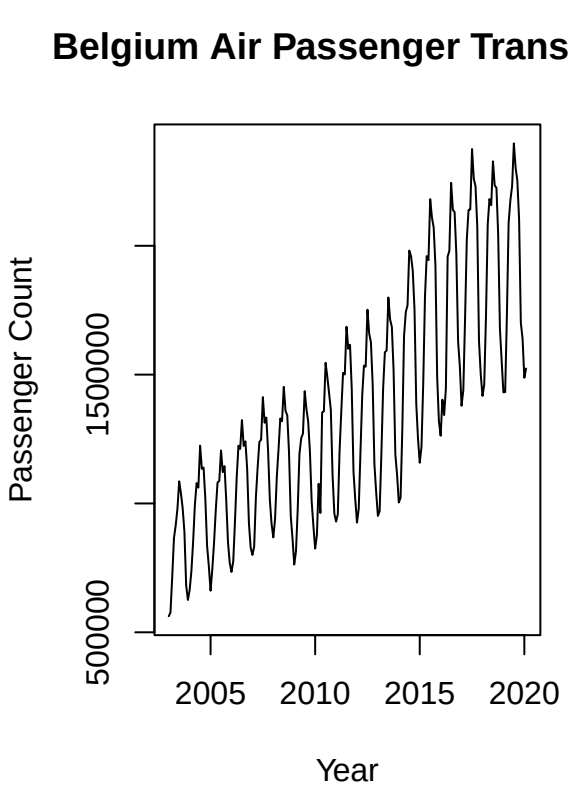
Our optimal value of lambda is: 0.0146. Since our value is so close to 0, it would result in a log transform:

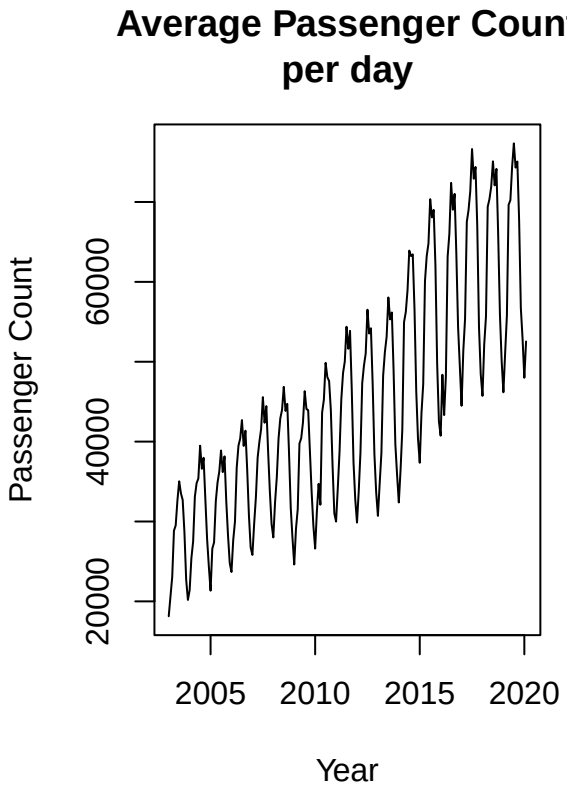


However, if we transform our data, it means we should back-transform our data at some point in order to make forecast on the regular scale. Thus, we can simply take our optimal lambda for our forecasting models and make automated Box-Cox transformations with it.

Adjustments

Sometimes, time series variations are due to calendar effects such as the month length. It is important to remove these variations because it may show as a seasonal effect. With R's monthdays function we can observe the average Passenger Count per day:





There is really no visual difference compared to our original time series plot, meaning an adjustment would not make much of a difference.

Seasonal Naive Forecasting

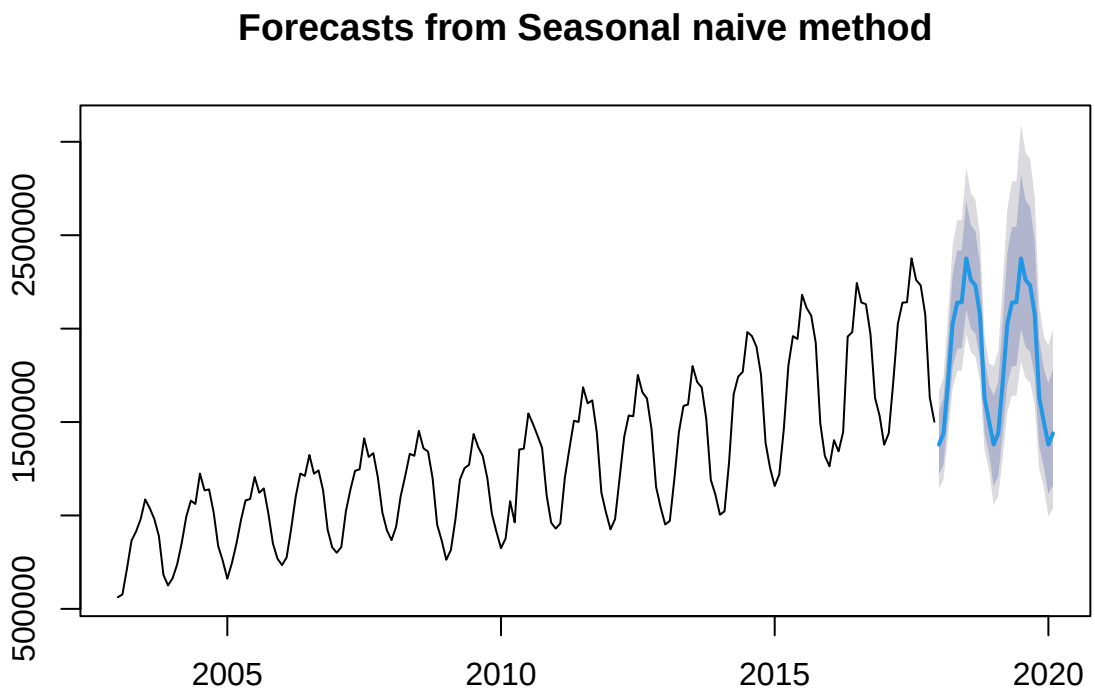
Now that we have a much better understanding of our data, we can begin with our forecasting methods. We apply a seasonal Naïve method. We can use R's function `snaive()`.

First, we save variable `h` with the length of the test set. The reasoning behind this is that `h` specifies the number of periods we want to forecast, and it is ideal to compare it to the test set to measure the accuracy and performance of our model.

```
# Length of test for forecasts
h = length(test)
```

Then, we fit our model on the training set and we can observe our forecasted plot:

```
# Seasonal Naive
abe_m1 <- snaive(train, h = h, lambda =1)

#Plot
par(mfrow=c(1,1))
plot(abe_m1)
```



Notice that we are using our previously saved lambda value as `l`.

Furthermore, we can analyze the performance of our model with a Ljung-Box test:

```
# Ljung-Box Test
res <- residuals(abe_m1)
Box.test(res, lag=h, fitdf=0, type="Lj")
```

```
##
##  Box-Ljung test
##
## data:  res
## X-squared = 231.09, df = 26, p-value < 2.2e-16
```

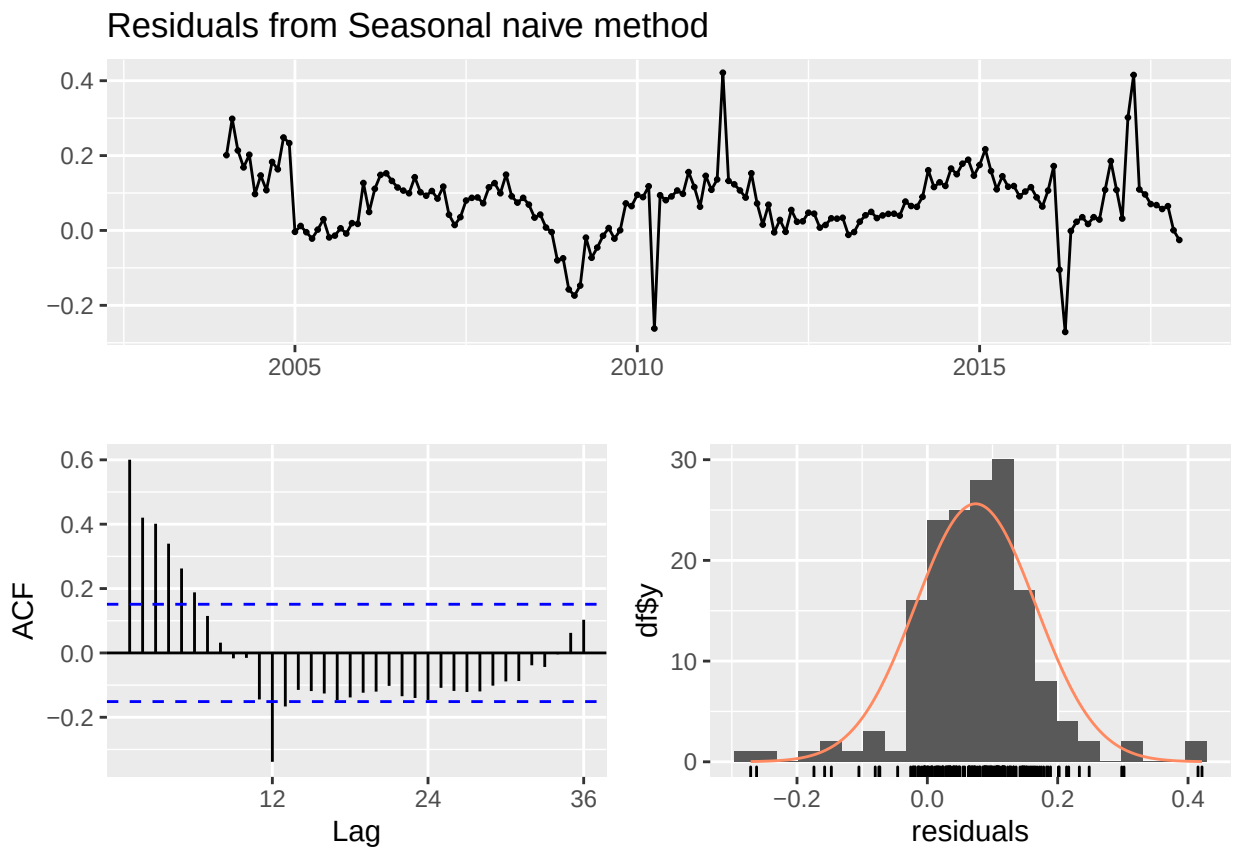
According to statology.org, the Ljung Box-Test uses the following hypothesis:

- H0: The residuals are independently distributed
- HA: The residuals are not independently distributed; they exhibit serial correlation.

Ideally, we would like to accept the null hypothesis (fail to reject). In other words, we want the p-value to be greater than 0.05 which would mean that our residuals are independent.

In this case, we can observe our p-value is very small, meaning that the residuals are not independently distributed.

Another way of checking our residuals is by using `checkresiduals()`.




```
##
##  Ljung-Box test
##
## data:  Residuals from Seasonal naive method
## Q* = 225.93, df = 24, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 24
```

Just by taking a quick glance at the left ACF plot, we can observe that we have some peaks going beyond the blue lines, meaning that there is a correlation for some of the residuals. Again, we validate that the residuals are not independently correlated.

Finally, we can observe the accuracy of the model by using our test set:

```
accuracy(abe_m1, test)
```

```
##                     ME      RMSE      MAE      MPE     MAPE      MASE      ACF1
## Training set 77362.97 124232.13 95011.47 5.592279 7.225186 1.0000000 0.6178910
## Test set     35966.85  55167.45 45156.54 2.088867 2.518184 0.4752746 0.5473326
##              Theil's U
## Training set        NA
## Test set      0.294355
```

According to Hyndman and Koehler, the Mean Absolute Scaled Error (MASE) is a "generally applicable measurement of forecast accuracy without problems seen in other measurements." MASE is the mean

absolute error of the forecasts values, divided by the mean absolute error of the in-sample one-step naïve forecast. We have a MASE of 0.48 which is not so bad since the closer it is to 0, the better. However, there is still some improvements we could do with our model.

Some of the general characteristics of the MASE are:

1. MASE gives an indication of effectiveness of forecasting algorithm with respect to a naïve forecast.
2. If the value is greater than 1, it indicates the algorithm is performing poorly compared to the naïve forecast.
3. MASE is immune to the problem faced by Mean Absolute Percentage Error (MAPE) when actual time series output at any time step is zero. In this situation MAPE gives an infinite output, which is not meaningful.
4. MASE is independent of the scale of the forecast since it is defined using ratio of errors in the forecast.

STL Decomposition

STL stands for Seasonal and Trend decomposition using Loess. STL is a statistical method of decomposing a Time Series data into 3 components containing seasonality, trend, and residual. According to Cleveland, R. B., STL is a versatile and robust method for decomposing time series, while Loess is a method for estimating nonlinear relationships.

We perform STL forecast on our time series using various underlying forecasting methods. By using `stlf()` function in R, we can shortcut our way to decompose out timeseries with 4 different approaches:

```
#Forecasting by decomposition
abe_m2 <- stlf(train, method="naive", h=h, lambda = l, biasadj = TRUE)
abe_m3 <- stlf(train, method="rwdrift", h=h, lambda = l, biasadj = TRUE)
abe_m4 <- stlf(train, method="ets", h=h, lambda = l, biasadj = TRUE)
abe_m5 <- stlf(train, method="arima", h=h, lambda = l, biasadj = TRUE)
```

Then we can loop the performance and residuals and observe them in a table:

```
# Source:
# Van de Bossche, F. (2022). Forecasting. [Course].
# Lille: IESEG Management School. MSc in Big Data Analytics.

models = c("STL naive lambda", "STL rwdrift lambda", "STL ets lambda", "STL arima lambda")

n <- length(models); n  #number of models

#naming of models for the given data set
m <- "abe_m"

#prepare the accuracy tables
a_train <- matrix(nrow = 0, ncol = 5)
a_test  <- matrix(nrow = 0, ncol = 5)

#prepare the residual diagnostic table
res_matrix <- matrix(nrow = n, ncol = 4)
rownames(res_matrix) <- models
colnames(res_matrix) <- c("nr", "Q*", "df", "p-value")

for(i in 1:n) {
```

```
  #accuracy measures
  assign(paste0("a_m", i), accuracy(get(paste0(m, i)), test)[,c(2,3,5,6)])
  a_train <- rbind(a_train, c(i, get(paste0("a_m", i))[1,]))
  a_test <- rbind(a_test, c(i, get(paste0("a_m", i))[2,]))

  #residual diagnostics
  assign(paste0("res_m", i), checkresiduals(get(paste0(m, i)), plot = FALSE))
  res_matrix[i,] <- c(i, get(paste0("res_m", i))$statistic,
                      get(paste0("res_m", i))$parameter,
                      get(paste0("res_m", i))$p.value)
}

rownames(a_train) <- models
colnames(a_train)[1] <- "nr"
rownames(a_test) <- models
colnames(a_test)[1] <- "nr"
```

And we observe the performance of the test set:

```
# Accuracy
print(a_test[2:5,])
```

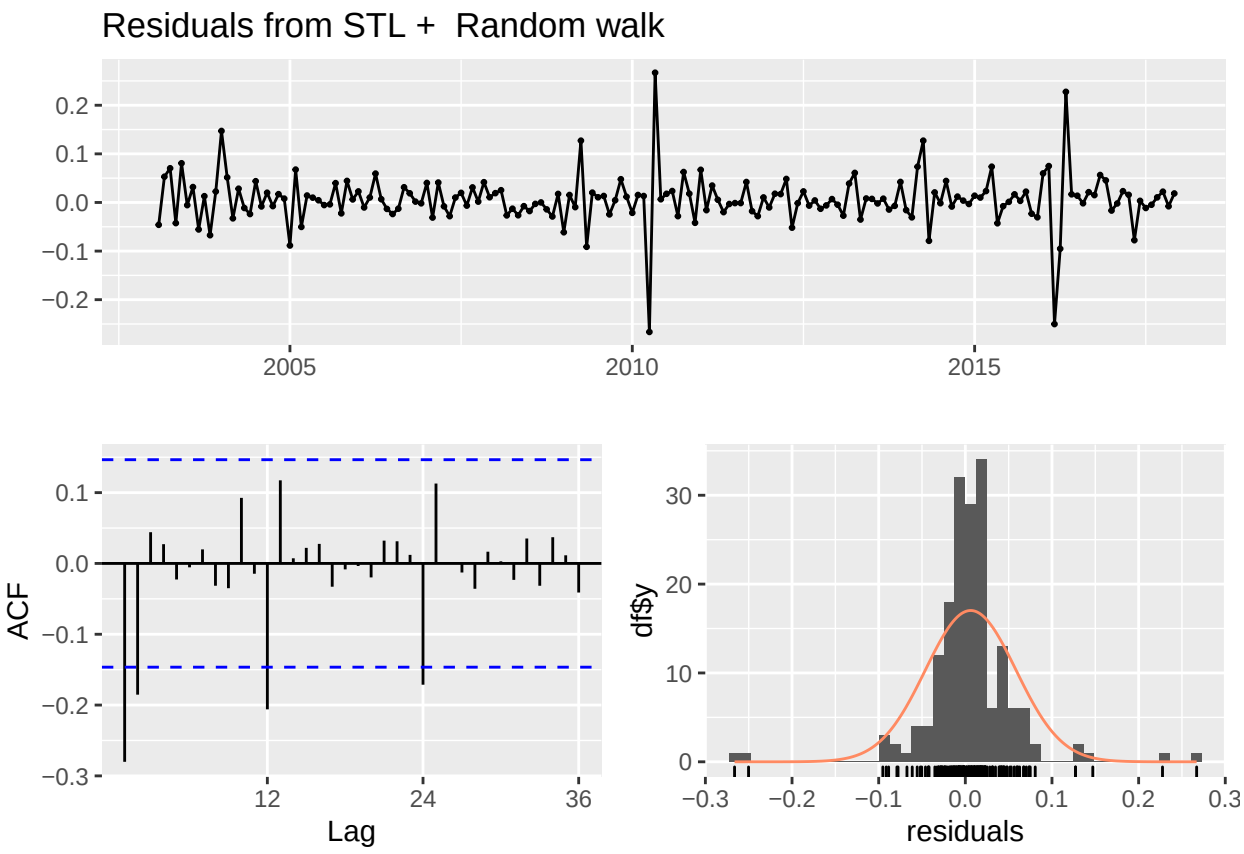
```
##                     nr      RMSE       MAE     MAPE      MASE
## STL naive lambda     2  72108.91  61211.46 3.168164 0.6442533
## STL rwdrift lambda   3 180895.43 151634.82 7.557488 1.5959633
## STL ets lambda       4 158865.08 131761.40 6.528619 1.3867947
## STL arima lambda     5 155561.35 128532.09 6.350925 1.3528060
```

```
# Residuals
print(round(res_matrix[2:5,], digits = 4))
```

```
##                     nr      Q* df p-value
## STL naive lambda     2 41.4131 24  0.0150
## STL rwdrift lambda   3 41.4131 23  0.0106
## STL ets lambda       4 23.1571 20  0.2811
## STL arima lambda     5 18.1386 21  0.6402
```

If we first look at our accuracy scores, basing ourselves on the MASE, we can observe that the STL naive lambda has the lowest MASE. However, the p-value is still below 0.05, meaning that there is correlation between our residuals. We can visualize this again by checking our `checkresiduals()`:

```
## Warning in checkresiduals(abe_m2): The fitted degrees of freedom is based on the
## model used for the seasonally adjusted data.
```




```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  Random walk
## Q* = 41.413, df = 24, p-value = 0.015
##
## Model df: 0.   Total lags used: 24
```

Exponential Smoothing Methods and ETS

Exponential Smoothing methods use weighted averages of past observations to make forecasts and they give more importance to recent values of the time series. These methods combine Error, Trend and Seasonal components during the calculation (ETS). Each one of these terms can be mixed in an additive or multiplicative way, as well as being left out of the model (AMN: Additive, Multiplicative or None).

For example, we could consider a simple exponential smoothing with multiplicative errors as ETS(M,N,N), or a Holt's linear method with additive errors as ETS(A,A,N).Basically, they are different combinations for seasonal components.

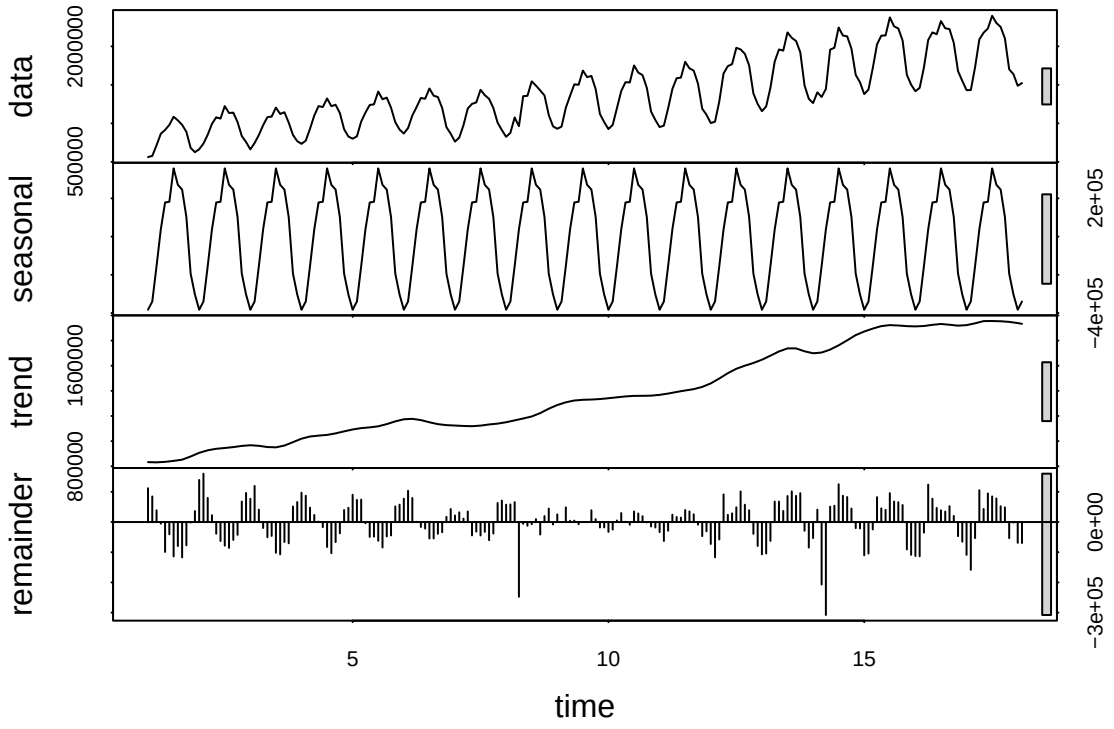
Moreover, each term can be used in the following:

- ETS(A,N,N): Simple exponential smoothing with additive errors
- ETS(A,A,N): Holt's linear method with additive errors
- ETS(A,A,A): Additive Holt-Winters' method with additive errors
- ETS(M,A,M): Multiplicative Holt-Winters' method with multiplicative errors

- ETS(A,Ad,N): Damped trend method with additive errors

In order to choose the best model, we can observe the STL decompose to observe the patterns of our data:

```
#stl decompose
plot(stl(ts(full_data[1:206], frequency=12), s.window = "periodic"))
```



1. If the trend component is increasing or decreasing linearly (or close to linear), then it has an additive pattern.
2. If the magnitude of the seasonal gaps is constant, it has an additive behavior (otherwise its multiplicative).
3. If the Remainder is not constant, then its multiplicative.

From our data we can conclude that we have a:

- Additive Trend behavior since the trend is increasing. However, the behavior is not so linear, so we could try with multiplicative as well.
- Additive season since it remains constant.
- Multiplicative Error since it's not constant.

Thus, we can conclude that our optimal models are ETS(MAA) and ETS(MAM). However, we should also consider that if we want to apply our box-cox transformation done previously, we can't use multiplicative methods. Thus, we will only perform ETS(AAA) models.

We fit our models into R, additionally adding the auto ETS, which will find the best model according to the AIC.

```r
# Auto ETS
m6 <- ets(train, lambda = l)
abe_m6 <- forecast(m6, h=h)

# AAA
m7 <- ets(train, model = "AAA", damped = FALSE, lambda = l, biasadj = TRUE)
abe_m7 <- forecast(m7, h=h)

m8 <- ets(train, model = "AAA", damped = TRUE, lambda = l, biasadj = TRUE)
abe_m8 <- forecast(m8, h=h)
```

```
##                  nr      RMSE       MAE     MAPE      MASE
## Auto ETS lambda  6  63807.65  51281.27 2.692921 0.5397377
## AAA lambda       7 140475.49 115174.60 5.736623 1.2122178
## AAdA lambda      8  64270.91  51535.06 2.663379 0.5424088
```

We can observe that the Auto ETS lambda is already performing better. Additionally, our AAdA with box-cox lambda is performing almost as good as the Auto ETS lambda.

If we check the Auto ETS lambda model chosen, we can observe the parameters that it chose:

```
## ETS(A,Ad,A)
##
## Call:
##  ets(y = train, lambda = l)
##
##   Box-Cox transformation: lambda= 0.0146
##
##   Smoothing parameters:
##     alpha = 0.5609
##     beta  = 8e-04
##     gamma = 1e-04
##     phi   = 0.9783
##
##   Initial states:
##     l = 14.9707
##     b = 0.0147
##     s = -0.2989 -0.1731 0.1023 0.2255 0.2469 0.3201
##            0.1809 0.1801 0.0479 -0.1186 -0.3238 -0.3892
##
##   sigma:  0.0541
##
##       AIC      AICc       BIC
## -97.39913 -93.15069 -39.92591
```

ETS(AAdA) is one model we did not attempt manually. Thankfully for us, Auto ETS is very useful to find the optimal model without having to try all the possible combinations.

Finally, if we analyze the residuals, we can observe how Auto ETS lambda and AAdA lambda have a very good p-value approaching the 0.05 value.

```
##                  nr      Q* df p-value
## Auto ETS lambda  6 14.2760  7  0.0465
## AAA lambda       7 27.3932  8  0.0006
## AAdA lambda      8 14.2760  7  0.0465
```

ARIMA Procedures

ARIMA stands for "Auto Regressive Integrated Moving Average" and it is a class of models that attempts to "explain" a given time series based on it's own past values (it's own lags) so that it makes a forecast. The difference between ARIMA and ETS is that it works by removing all the non-stationary aspects of a time-series. An ARIMA model is composed of 3 elements:

1. P: Auto Regressive (AR) term
2. Q: Moving Average (MA) term
3. D: The differencing required to make the time series stationary.

In essence, AR represents the number of lags that will be used as predictors and MA shows the number of lagged forecast errors to be used by the model.

Here, we use R's `auto.arima()` function. `auto.arima()` works by selecting the number of D differences via unit root tests. Also, it will select optimal P and Q by minimizing the AICc.

```
# ARIMA models
m9 <- auto.arima(train, stepwise = FALSE, approximation = FALSE, lambda = l, biasadj = TRUE)
abe_m9 <- forecast(m9, h=h)

m10 <- auto.arima(train, stepwise = FALSE, approximation = FALSE, lambda = l, biasadj = TRUE, d=1, D=1)
abe_m10 <- forecast(m10, h=h)
```

```
##                  nr     RMSE      MAE     MAPE     MASE
## ARIMA lambda      9 162334.6 143075.6 7.311609 1.505877
## ARIMA dD lambda  10 146048.8 128704.1 6.567899 1.354616
```

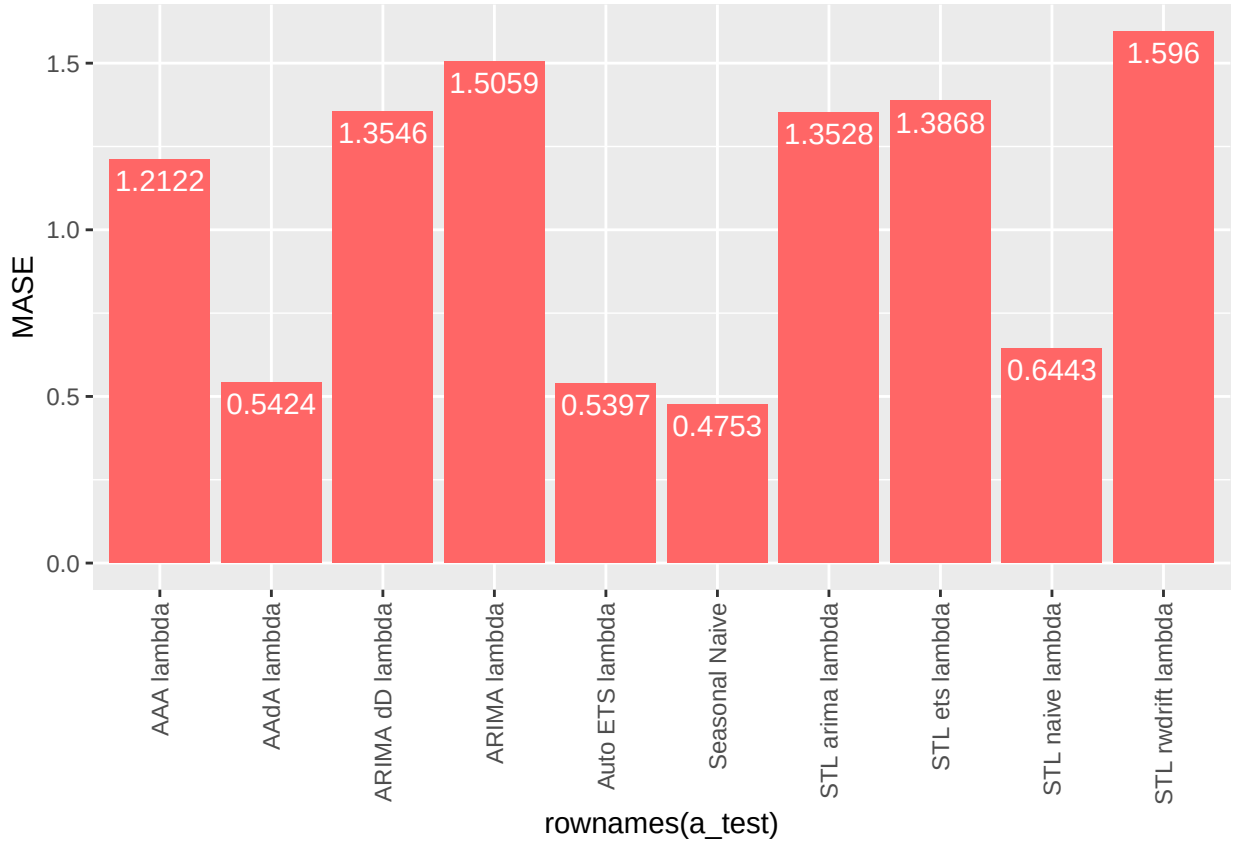
```
##                  nr     Q* df p-value
## ARIMA lambda      9 6.1312 18  0.9956
## ARIMA dD lambda  10 8.6890 20  0.9862
```

We can observe the performance of both of our ARIMA models and conclude that just by looking at the MASE performance, it is not a very good fit.

Overview

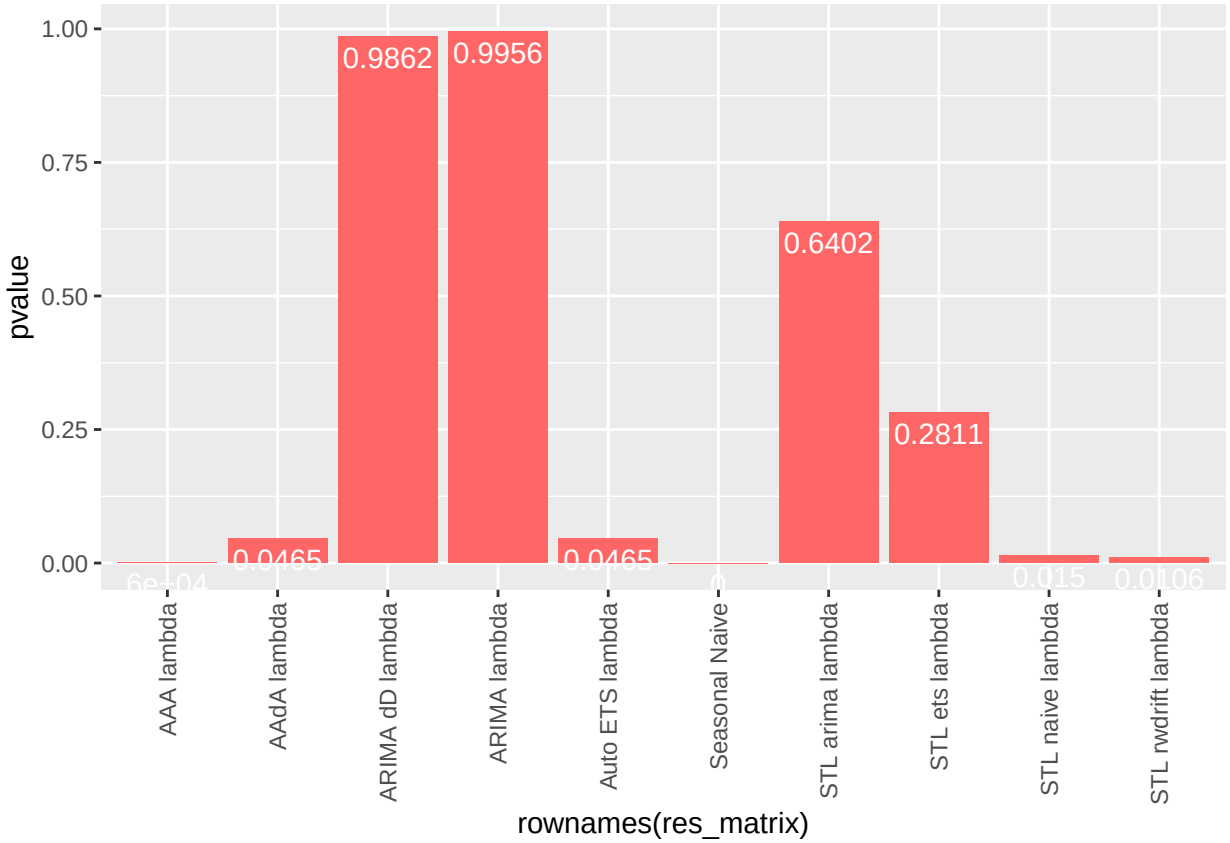
We observe all of our results in the following table:

```
##                     nr      RMSE       MAE     MAPE      MASE
## Seasonal Naive       1  55167.45  45156.54 2.518184 0.4752746
## STL naive lambda     2  72108.91  61211.46 3.168164 0.6442533
## STL rwdrift lambda   3 180895.43 151634.82 7.557488 1.5959633
## STL ets lambda       4 158865.08 131761.40 6.528619 1.3867947
## STL arima lambda     5 155561.35 128532.09 6.350925 1.3528060
## Auto ETS lambda      6  63807.65  51281.27 2.692921 0.5397377
## AAA lambda           7 140475.49 115174.60 5.736623 1.2122178
## AAdA lambda          8  64270.91  51535.06 2.663379 0.5424088
## ARIMA lambda         9 162334.63 143075.63 7.311609 1.5058775
## ARIMA dD lambda     10 146048.82 128704.08 6.567899 1.3546162
```



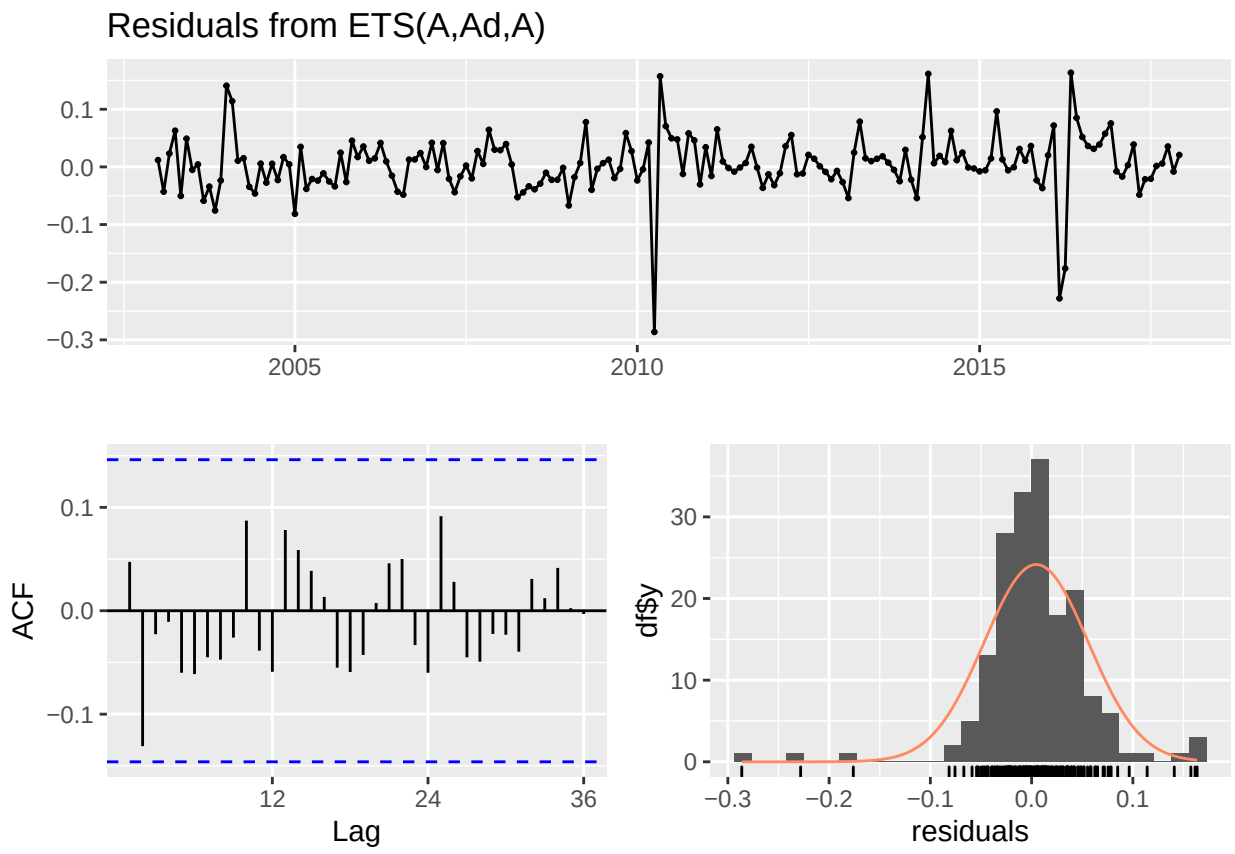
Then we observe the residuals:

```
##                     nr       Q* df pvalue
## Seasonal Naive       1 225.9332 24 0.0000
## AAA lambda           7  27.3932  8 0.0006
## STL rwdrift lambda   3  41.4131 23 0.0106
## STL naive lambda     2  41.4131 24 0.0150
## Auto ETS lambda      6  14.2760  7 0.0465
## AAdA lambda          8  14.2760  7 0.0465
## STL ets lambda       4  23.1571 20 0.2811
## STL arima lambda     5  18.1386 21 0.6402
## ARIMA dD lambda     10   8.6890 20 0.9862
## ARIMA lambda         9   6.1312 18 0.9956
```



From MASE alone, we could conclude that Seasonal Naïve is the best method because it has the lowest error. However, the p-value for this method is not good at all, thus we take a look at our second option which is Auto ETS lambda. Auto ETS Lambda has a very good MASE performance whilst being very close to our p-value threshold of 0.05. Thus, we choose this model to perform our forecasts.

We perform one last check on the best model:




```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,Ad,A)
## Q* = 14.276, df = 7, p-value = 0.04648
##
## Model df: 17.   Total lags used: 24
```

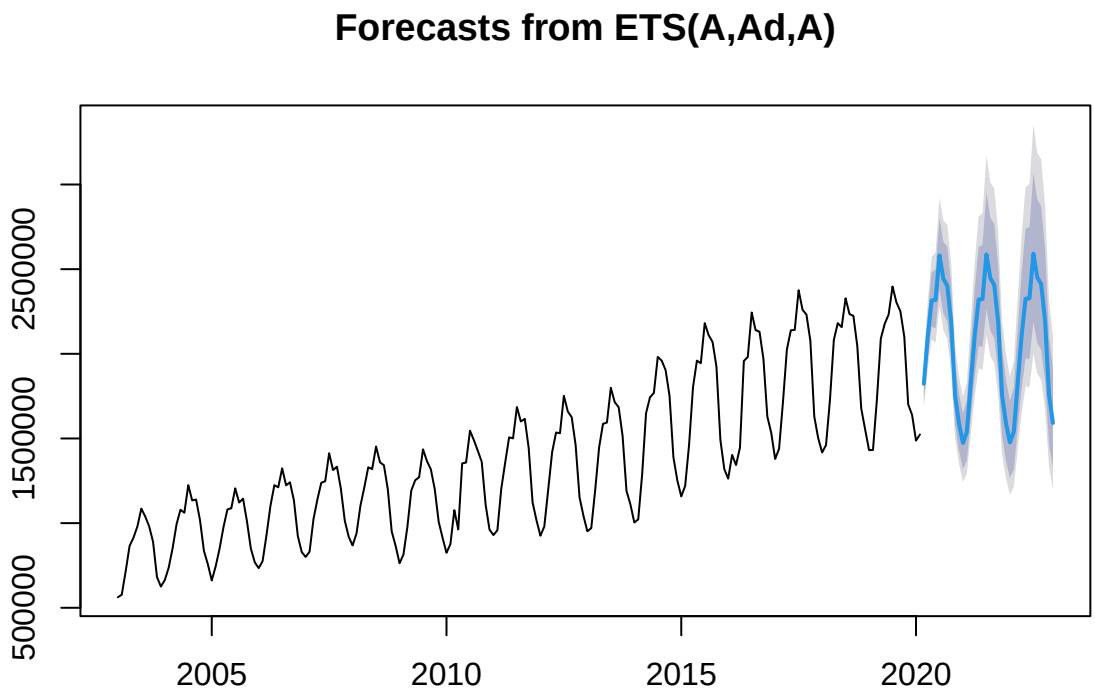
Forecasting

We re-fit our best model into the full dataset (train + test). For this we have to specify manually the parameters since our best model is the auto ETS.

```
# Re fit model
new_fit <- ets(full_data, model="AAA", damped = TRUE, lambda=l, biasadj=FALSE)
```

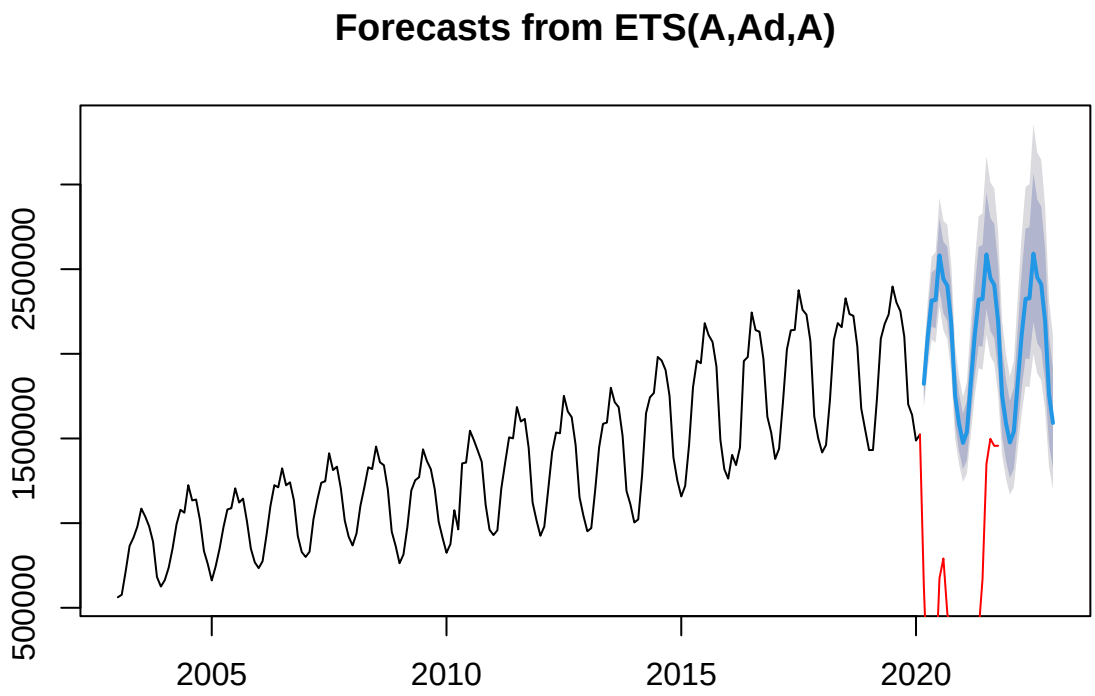
Then, we make a forecast of 34 months up to December 2022.

```
abe_new_fit <- forecast(new_fit, h=34) #34 months up to dec 2022
plot(abe_new_fit)
```



Part I Conclusion

To conclude we observe our forecast with the best model as well as the real data for 2020. Despite our model having great accuracy and residual performance, its impossible to predict all the external factors that could affect our timeseries, such as COVID. However, if we observe the end of the curve for the real data, we can see how it is already approximating the level it used to have before COVID, which could be an indicator that during the last 2 months of 2021 and the full duration of 2022, the normal behavior of our data could be reached once again, meaning that our forecast is not in vain.



Part II: South Korea's Renewable Energy Consumption

Energy is one of the biggest issues worldwide, which is directly linked to global warming. A shift from coal and oil energy to renewable sources is necessary for economies to become sustainable. In order to make change in traditional energy structure, analysis of energy data is essential.

The following dataset contains monthly energy usage in South Korea from 1997 to Nov,2021. It is measured in units of 1000toe (Which measures amounts energy). This dataset is taken from Kaggle in the following link:

https://www.kaggle.com/datasets/bell2psy/korea-energy-usage analysis?select=EnergyUsage_byType.csv

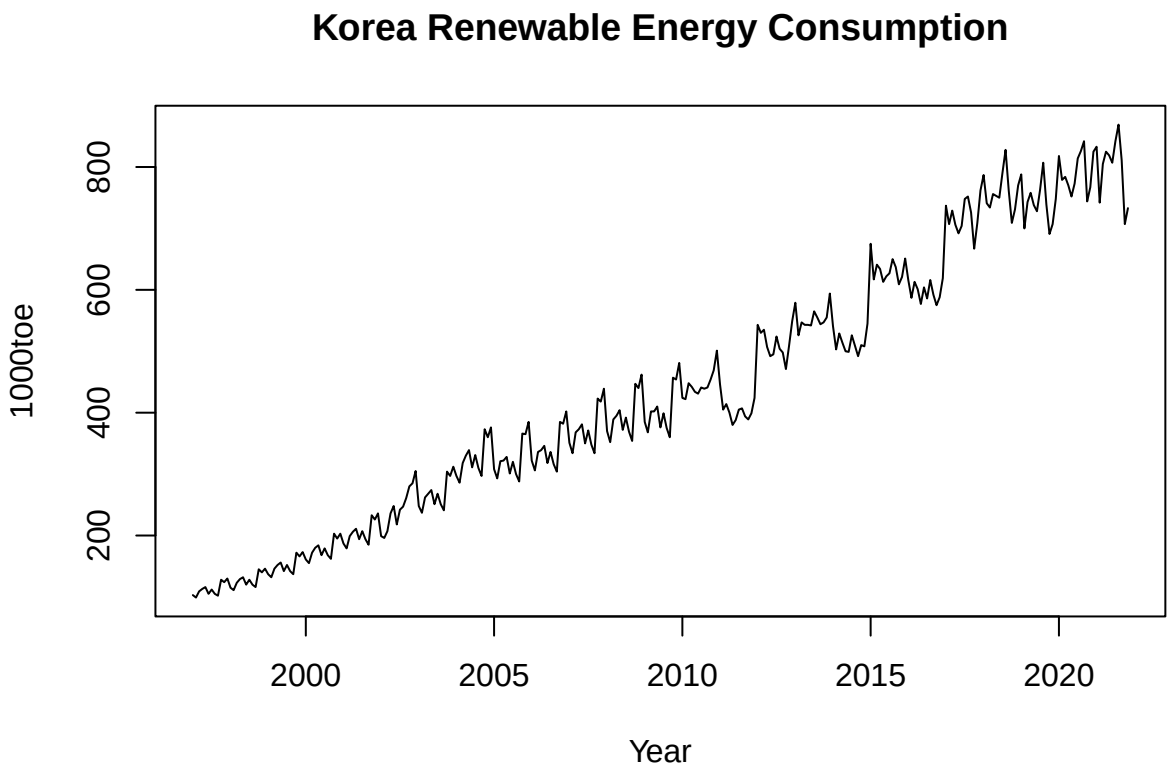
The dataset has many different energy measures by usage and by industry, but for this particular report we will focus only on renewable energy.

Exploratory Data Analysis

First, we load our data and plot our time series. From a first glance its hard to observe any clear seasonality components, but there are surely some patterns repeated, as well as as a trend:

```
# Load data
data <- fread("./data/raw/EnergyUsage_byType.csv")
renewable <- ts(data$RenewableETC, frequency = 12, start = 1997)

# Set Seed
```

```r
set.seed(1)

# Timeseries Plot
plot(renewable, main="Korea Renewable Energy Consumption",
     ylab="1000toe", xlab="Year")
```

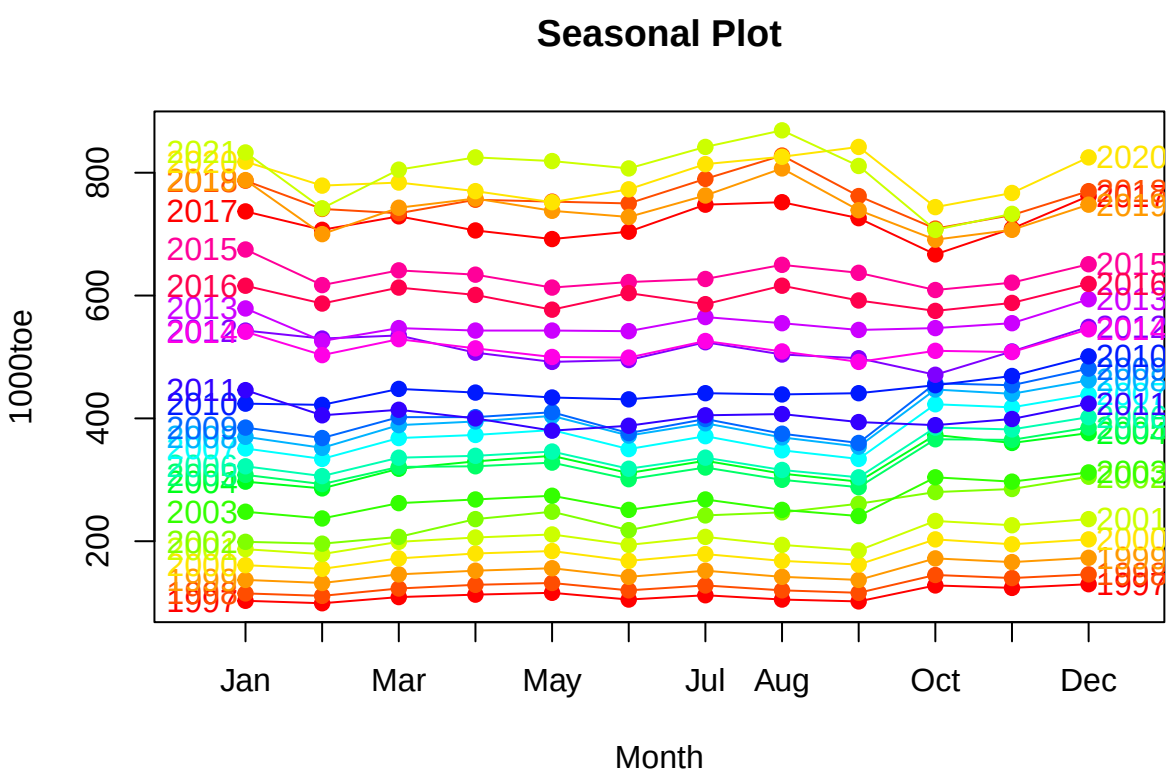


Then we split into Train and Test:

```r
# Train Test Split
train <- window(renewable, start = 1997, end = c(2017,12))
test <- window(renewable, start = 2018, end = c(2021, 11))

#Length of test
h = length(test)
```

We observe our seasonal plot:

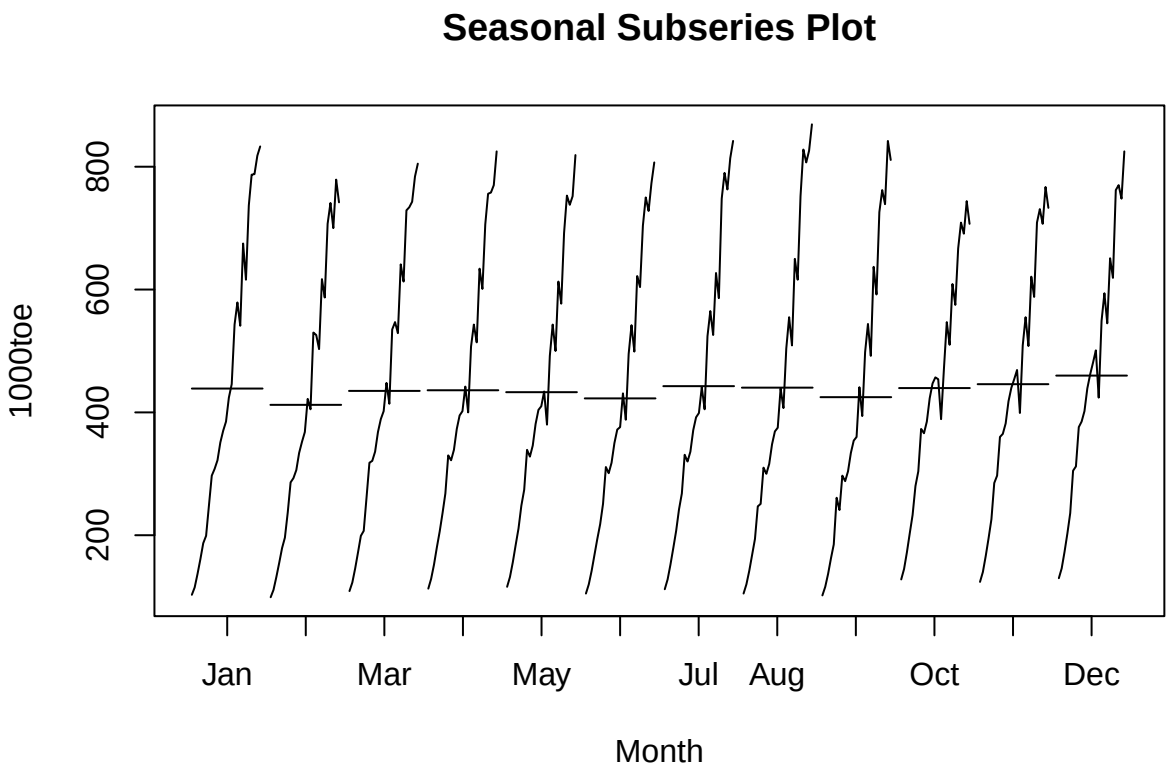
```r
# Seasonal Plot
seasonplot(renewable, year.labels=TRUE, year.labels.left=TRUE,
           main="Seasonal Plot",
           ylab="1000toe", xlab="Month", col=rainbow(20), pch=19)
```



It's hard to observe but over the decades there is a different seasonality. For instance, from 1997 to 2001 there are similar consumption patterns, while 2017 to 2020 has a completely different pattern. This may be due to the increasing importance given to renewable energy during the last years.

hen we observe the seasonal subseries plot:

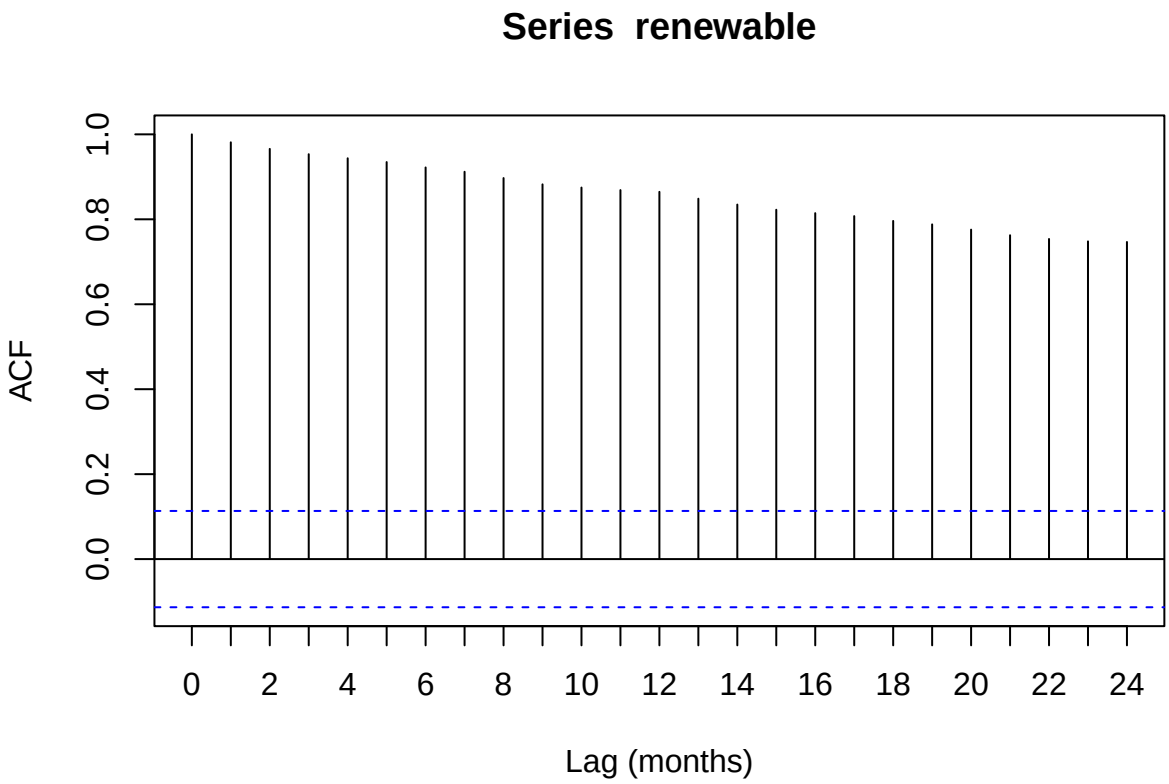
```
# Seasonal subseries
monthplot(renewable, ylab="1000toe", xlab="Month", xaxt="n",
          main="Seasonal Subseries Plot",type="l") +
  axis(1, at=1:12, labels=month.abb, cex=0.8)
```



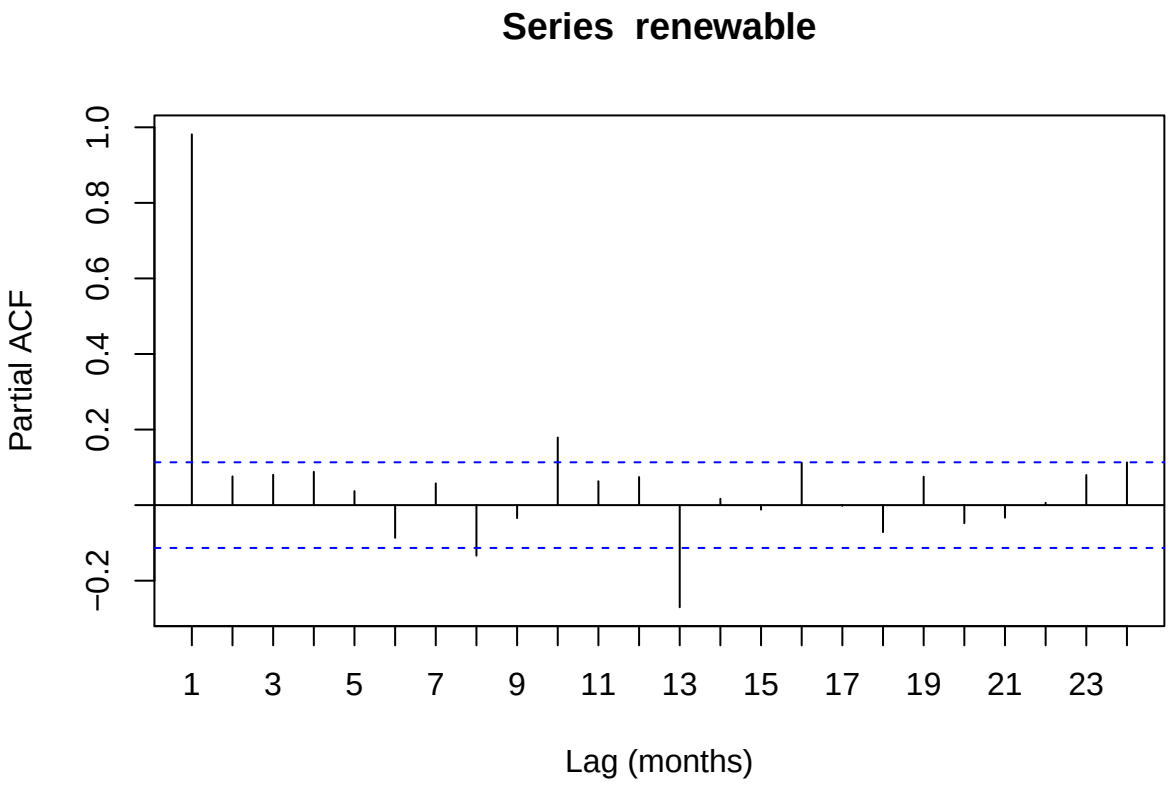

```
## numeric(0)
```

We can observe that there is a higher mean of renewable energy consumption for the months of July, August, and December. On the contrary, June, September and February have the lowest mean energy consumption.

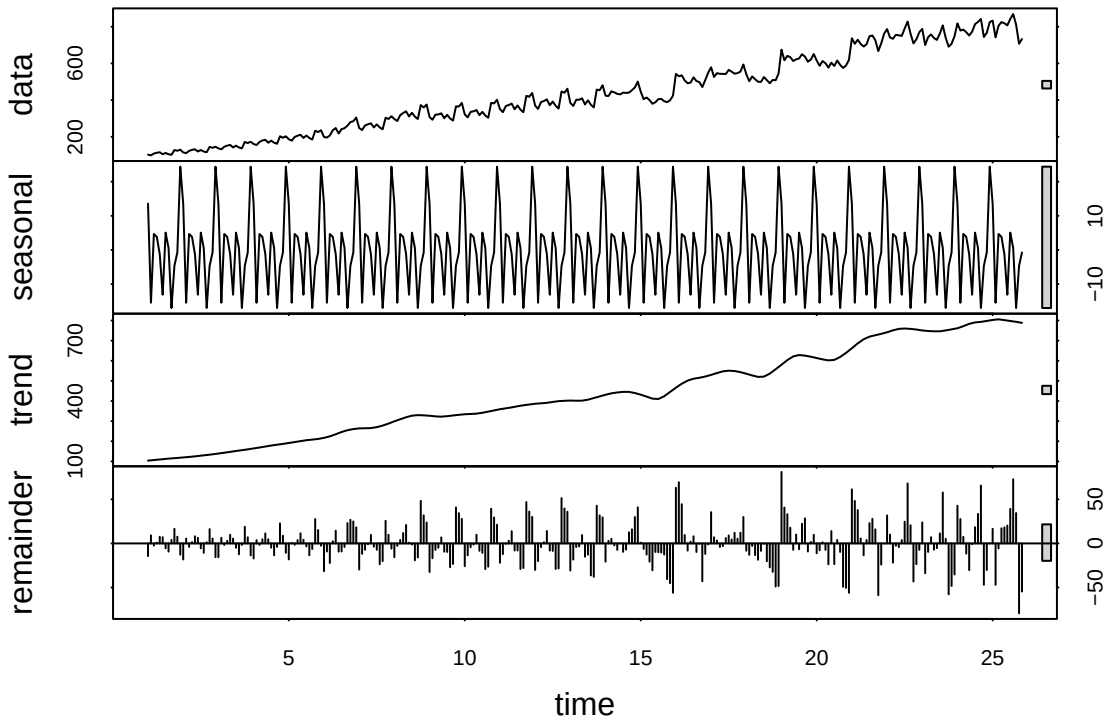
Furthermore, from observing our ACF we can't observe the seasonality correlation so easily as in the previous exercise:



From the PACF we can see how the first month of the year hold a clear correlation (lag 1 and 13):



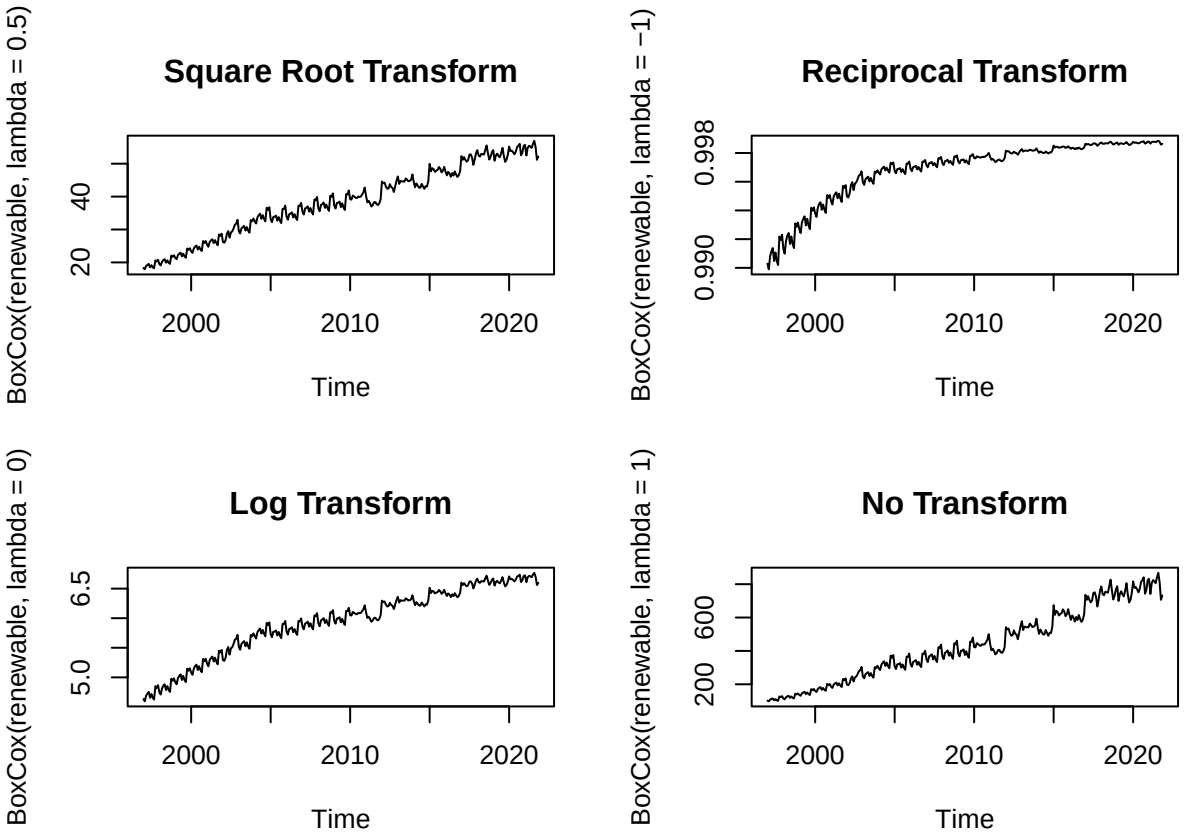
Additionally, before modeling we also explore the decomposition of the data for a better understanding of the trend, seasonality, and error:



We can observe a clear seasonal patter that is constant. The trend seems to be increasing and somewhat liner, and the error is not constant at all.

Transform

We can observe our different lambda transformations and observe how our time series changes a lot in shape:

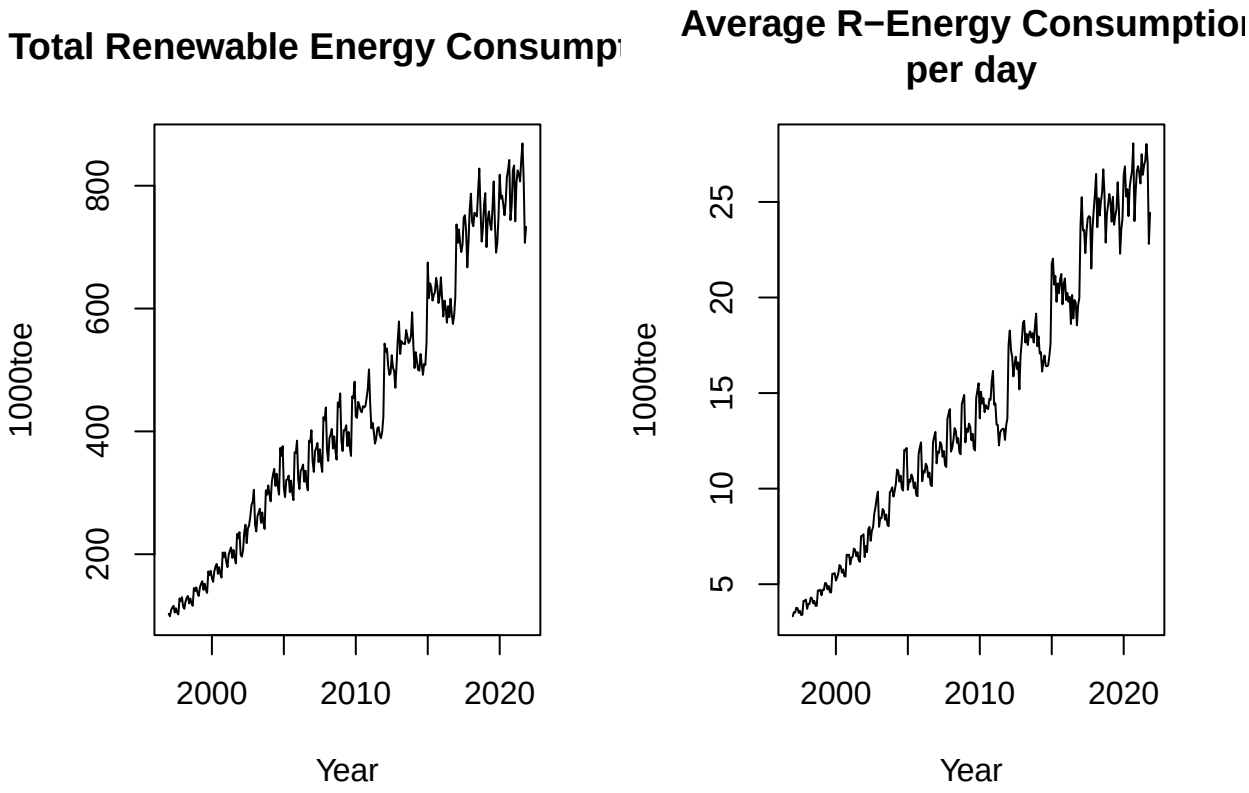



```
l <- BoxCox.lambda(renewable)
l
```

```
## [1] 0.4860756
```

The value approaches 0.5, meaning that its close to a square root transformation. Checking the calendar adjustments, the plots look the same so there is no need to make calendar adjustments.

```
# Adjustment month length
par(mfrow=c(1,2))
plot(renewable, main="Total Renewable Energy Consumption",
     ylab="1000toe", xlab="Year")

plot(renewable/monthdays(renewable), main="Average R-Energy Consumption
per day", ylab="1000toe", xlab="Year")
```



We will proceed by using the box-cox lambda transform in all the models just as in the previous exercise.

Modeling

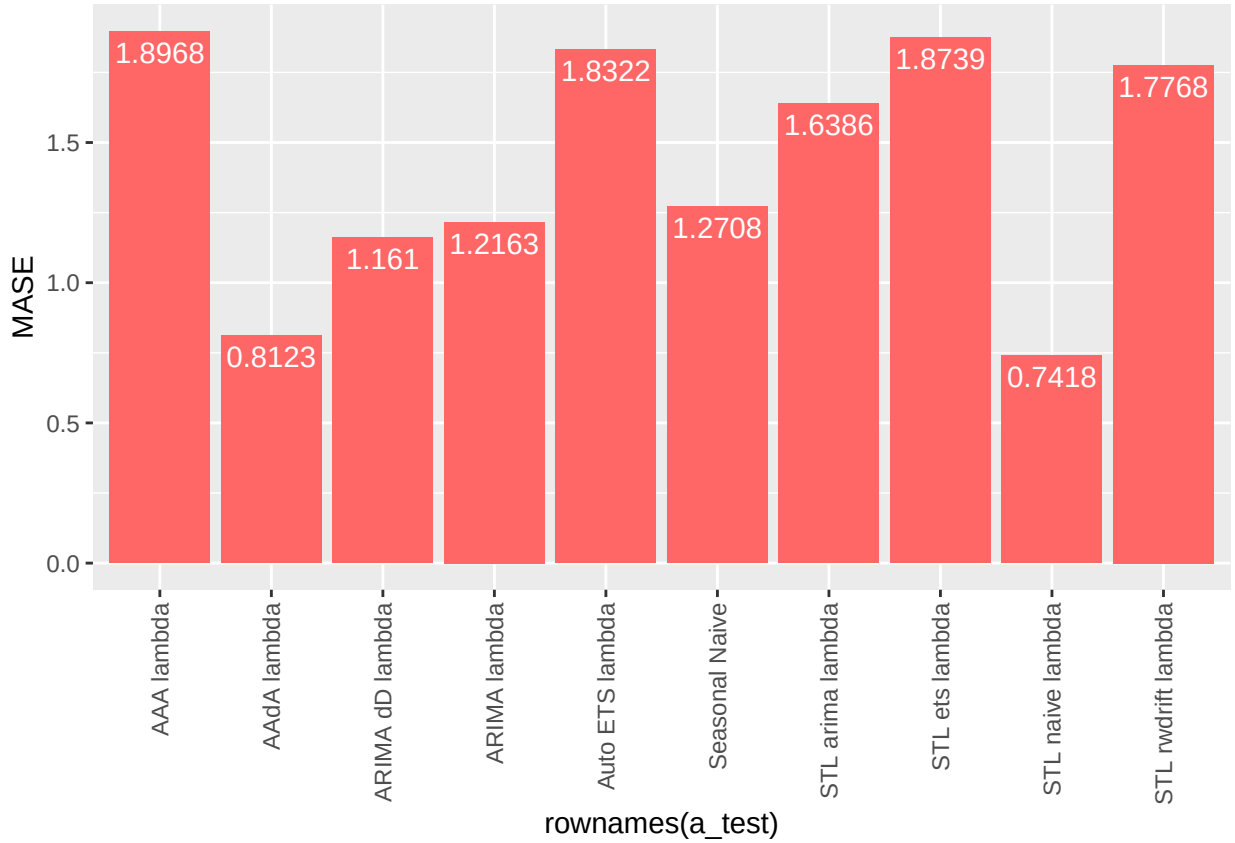
We will perform the models explored on the previous exercise for a better understanding and comparison of the same methodologies with different datasets.

```
# Seasonal Naive
abe_m1 <- snaive(train, h = h, lambda =l)

# Forecasting by decomposition
abe_m2 <- stlf(train, method="naive", h=h, lambda = l, biasadj = TRUE)
abe_m3 <- stlf(train, method="rwdrift", h=h, lambda = l, biasadj = TRUE)
abe_m4 <- stlf(train, method="ets", h=h, lambda = l, biasadj = TRUE)
abe_m5 <- stlf(train, method="arima", h=h, lambda = l, biasadj = TRUE)

# Auto ETS
m6 <- ets(train, lambda = l)
abe_m6 <- forecast(m6, h=h)

# AAA
m7 <- ets(train, model = "AAA", damped = FALSE, lambda = l, biasadj = TRUE)
abe_m7 <- forecast(m7, h=h)

m8 <- ets(train, model = "AAA", damped = TRUE, lambda = l, biasadj = TRUE)
```

```r
abe_m8 <- forecast(m8, h=h)

# ARIMA models
m9 <- auto.arima(train, stepwise = FALSE, approximation = FALSE, lambda = 1, biasadj = TRUE)
abe_m9 <- forecast(m9, h=h)

m10 <- auto.arima(train, stepwise = FALSE, approximation = FALSE, lambda = 1, biasadj = TRUE, d=1, D=1)
abe_m10 <- forecast(m10, h=h)
```

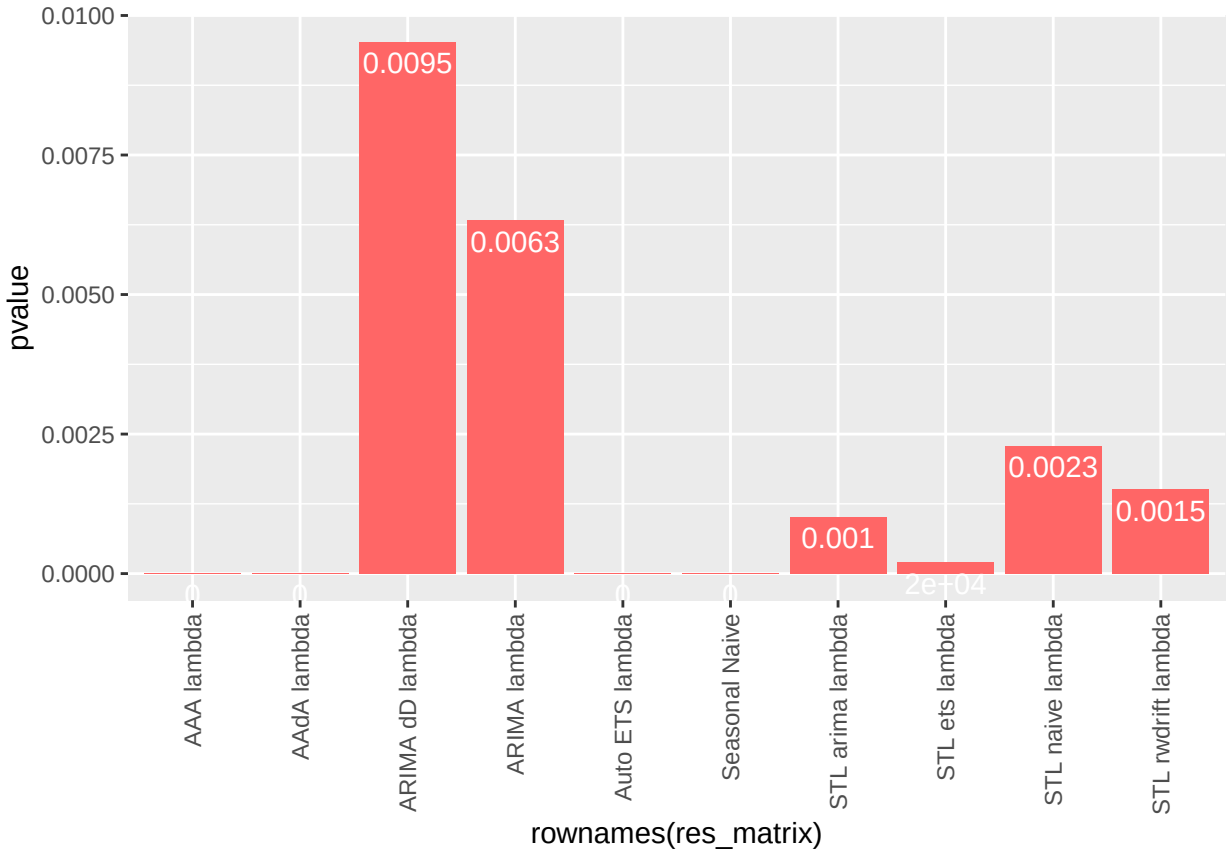
Then we measure the performance in a similar fashion as in the previous exercise and display our results:

```
##                     nr     RMSE      MAE      MAPE      MASE
## Seasonal Naive       1 63.31650 54.46809  6.888571 1.2707631
## STL naive lambda     2 42.70853 31.79461  3.994488 0.7417815
## STL rwdrift lambda   3 87.98781 76.15616  9.944006 1.7767549
## STL ets lambda       4 92.92131 80.32155 10.474847 1.8739354
## STL arima lambda     5 81.88682 70.23310  9.177461 1.6385676
## Auto ETS lambda      6 90.89949 78.53176 10.220222 1.8321786
## AAA lambda           7 93.84066 81.30320 10.576130 1.8968375
## AAdA lambda          8 44.32275 34.81716  4.432011 0.8122990
## ARIMA lambda         9 62.84121 52.13214  6.763173 1.2162646
## ARIMA dD lambda     10 58.84483 49.76442  6.491853 1.1610248
```



Then we observe our residuals:

```
##                     nr       Q* df p-value
## Seasonal Naive       1 773.1037 24  0.0000
## STL naive lambda     2  48.3420 24  0.0023
## STL rwdrift lambda   3  48.3420 23  0.0015
## STL ets lambda       4  50.1980 20  0.0002
## STL arima lambda     5  46.7424 21  0.0010
## Auto ETS lambda      6  88.8785  8  0.0000
## AAA lambda           7  88.8785  8  0.0000
## AAdA lambda          8  84.7848  7  0.0000
## ARIMA lambda         9  37.7805 19  0.0063
## ARIMA dD lambda     10  37.7386 20  0.0095
```

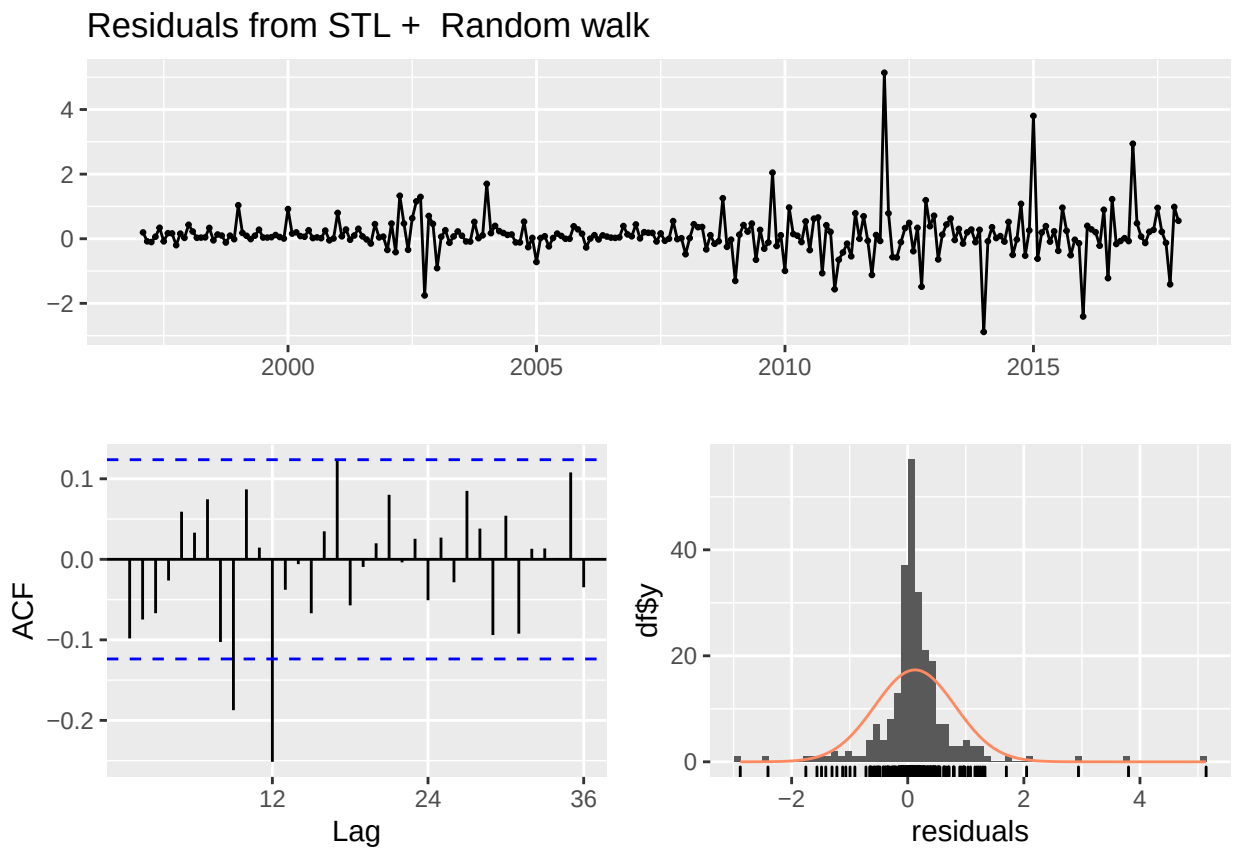


By observing our MASE and p-values we can observe that this data is clearly more unstable than the Airplane data on the previous exercise. However we can still work with it. Based on the MASE we choose the STL naive lambda as our best model with a MASE of 0.7418.

The p-value of 0.0023 is not optimal, but it is still our best p-value among all our models.

Finally, we check the residuals of our model:

```
## Warning in checkresiduals(abe_m2): The fitted degrees of freedom is based on the
## model used for the seasonally adjusted data.
```

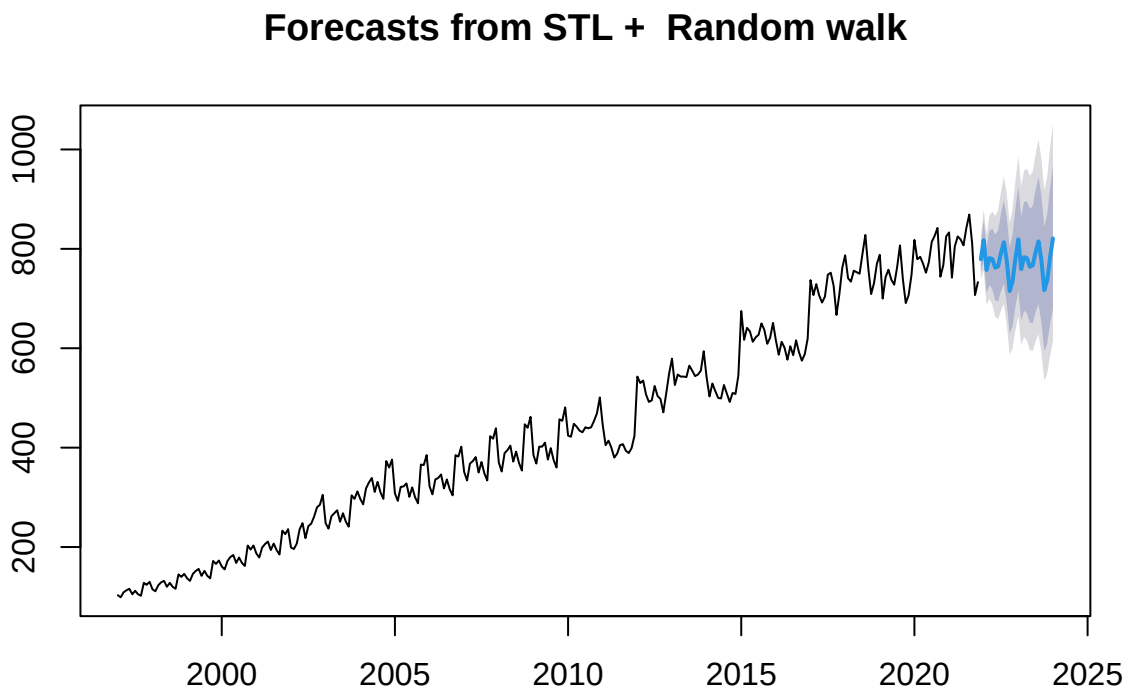



```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  Random walk
## Q* = 48.342, df = 24, p-value = 0.002289
##
## Model df: 0.   Total lags used: 24
```

As expected, the ACF still has information from the residuals that is not being considered, which is not optimal.

Forecasting

Finally, we re-fit our model with the full data and proceed to forecast:



Part II Conclusion and Further Steps

It is very likely that the increasing trend of renewable energies will keep its course in South Korea, however, it is very hard to achieve an accurate forecast since the behavior is very different every 5 or 10 years.

Perhaps by running a model with a shorter time series window would return better results and a more precise forecast.

References

6 reasons why forecasting is important for your business. (2020, November 12). ForceBrands Newsroom. https://forcebrands.com/blog/business-forecasting/

Daitan. (2019, August 28). Exponential smoothing methods for time series forecasting. Medium. https://betterprogramming.pub/exponential-smoothing-methods-for-time-series-forecasting-d571005cdf80

(n.d.). European Commission | Choose your language | Choisir une langue | Wählen Sie eine Sprache. https://ec.europa.eu/eurostat/databrowser/view/AVIA_PAINCC__custom_1737837/default/line?lang=en

Exponential smoothing (ETS) algorithm. (n.d.). https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-ets.html

How to use power transforms for time series forecast data with Python. (2019, August 28). Machine Learning Mastery. https://machinelearningmastery.com/power-transform-time-series-forecast-data-python/

Ljung-box test: Definition + example. (2020, October 15). Statology. https://www.statology.org/ljung-box-test/

Manani, K. (2022, April 28). Multi-seasonal time series decomposition using MSTL in Python. Medium. https://towardsdatascience.com/multi-seasonal-time-series-decomposition-using-mstl-in-python-136630e67530

Peixeiro, M. (2021, June 22). The complete guide to time series analysis and forecasting. Medium. https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

Van de Bossche, F. (2022). Forecasting. [Course].Lille: IESEG Management School. MSc in Big Data Analytics.