M11: Advanced Forecasting Methods

Luana Lima

04/01/2021

Learning Goals for M11

1. Discuss challenges in forecasting higher frequency time series.
2. Discuss complex seasonalities.
3. Go over some functions in R that can handle multiple seasonal pattern.
4. Develop a code for forecasting daily electricity demand.

Setting R code chunk options

First R code chunk is used for setting the options for all R code chunks. The choice echo=TRUE means both code and output will appear on report, include = FALSE neither code nor output is printed.

Loading packages and initializing

Second R code chunk is for loading packages. By setting message = FALSE and warning = FALSE, the code will appear but it will node include messages and warnings.

```
library(lubridate)
library(ggplot2)
library(forecast)
library(Kendall)
library(tseries)
library(outliers)
library(tidyverse)
library(smooth)
library(zoo)
library(kableExtra)
```

Introduction to Complex Seasonality

So far we considered only monthly seasonal patterns. But higher frequency data is becoming more and more common. They usually exhibit more complicated seasonal pattern.

- Ex: daily data for electricity usage will have a weekly pattern as well as an annual pattern.
- Ex: hourly data for electricity usage will have three types of seasonality: daily pattern, weekly pattern and annual pattern

The `ts()` function we have been using to create time series objects can only handle one type of seasonality and the `frequency` argument only takes integer values. When working with higher frequency data with multiple seasonalities you should use the `msts` class from package `forecast`. `msts` stands for **m**multiple **s**easonal **t**ime **s**eries. The `msts()` function takes the following main arguments.

- **data** numeric vector, matrix or data frame

- **seasonal.periods** vector with the seasonal periods
- **ts.frequency** seasonal period that should be used as frequency of the underlying ts object. The default value is max(seasonal.periods)
- **start** starting date/time of the time series. Now instead of only using start=`c(year, month)` we will use for example `start=c(year,month,day)`

Recall from `ts()` function that frequency is the number of observation per season.

Table 1: Frequency used with ts() function

Data	Frequency
Annual	1
Quartely	2
Monthly	12
Weekly	52

The weekly is already a challenge for the ts() function because typically the seasonal period is 365.25/7 ~ 52.179 on average and not exactly 52. The 365.25 should be used instead of 365 to account for leap years. One way to work around it and keep frequency integer is to eliminate one day from the leap years.

For the `msts()` we will use different frequency specifications. Here are some examples. The `msts` allow for non-integer frequency specification as you can see for daily data.

Table 2: Frequency used with msts() function

Data	Day	Week	Year
Daily		7	365.25
Hourly	24	24*7=168	24*365.25=8766
Half-hourly	2*24=48	2*24*7=336	2*24*365.25=17532

Keep in mind that sometimes it's not necessary to include all of these frequencies — just the ones that are likely to be present in the data. For example solar radiance, wind speed, water inflows are unlikely to have a weekly period because there is no difference in behavior for weekdays and weekends. Another example, if you have one year of daily data, you don't need to model the yearly pattern. Because you only have one observation for each day of the year.

Other examples of frequency specification can be found here.

Modeling and Forecasting with complex seasonalities

We learned some approaches to model series with a single seasonal pattern.

- Moving Average, Naive Methos
- Simple Exponential smoothing
- Seasonal ARIMA
- State Space models
- ETS - Exponential smoothing state space model

Today we will learn a few examples of models for multiple seasonal patterns.

- ETS
- ARIMA with dynamic harmonic fourier components
- TBATS

Importing data with multiple seasonal patterns

For this module we will work with household electricity consumption. This file contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months). The data is part of Machine Learning Repository of the University of California, Irvine, School of Information and Computer Sciences. You may download the data here.

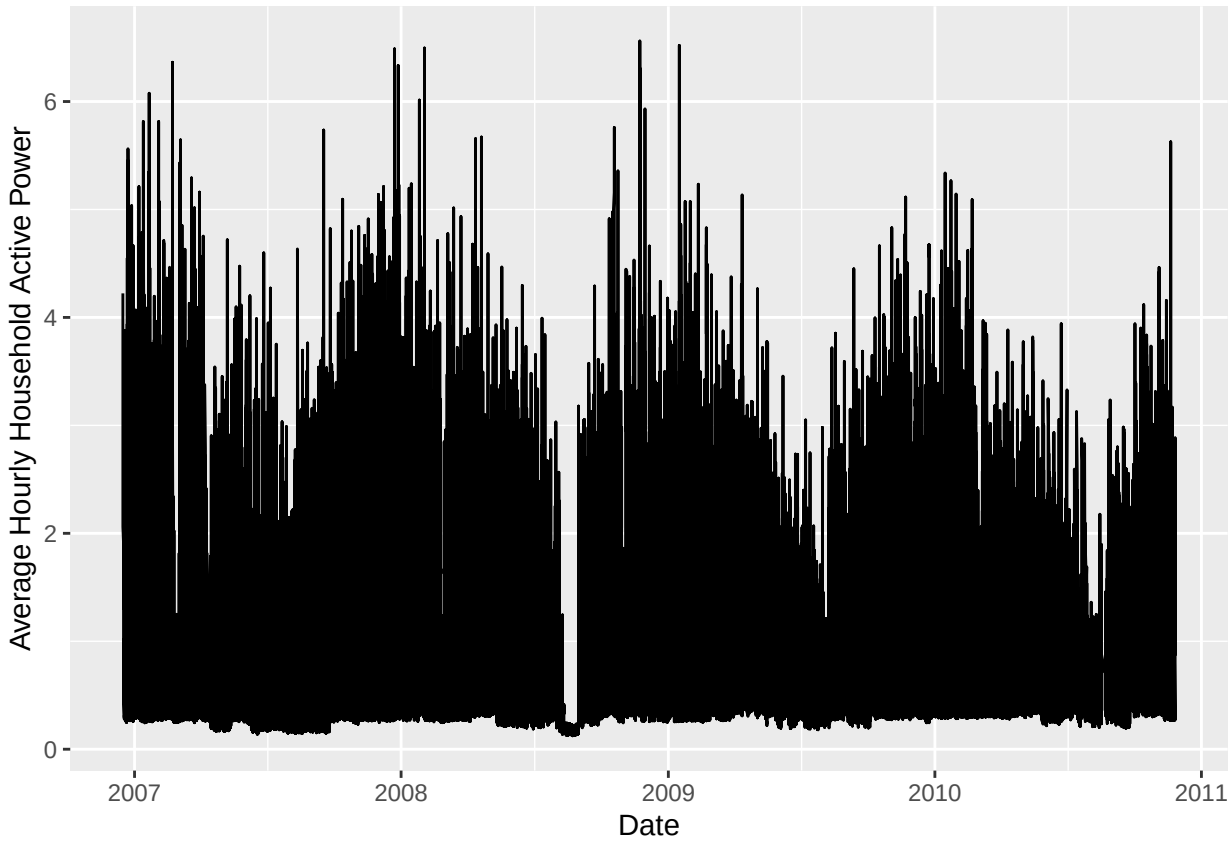
The data will need some tidying up. Here we are using the `tidyverse` package and pipes to do the data wrangling.

```
#Importing time series data from text file#
household_all <- read.table(
  file="./Data/household_power_consumption.txt",
  header=TRUE, sep=";")

#Fixing date columns and selecting active power only column
act_power_all <- household_all %>%
  mutate( Global_active_power = as.numeric(Global_active_power)) %>%
  mutate( Date = dmy(Date)) %>%
  mutate( Time = hms(Time)) %>%
  mutate( Year = year(Date),
          Month = month(Date),
          Day = day(Date),
          Hour = hour(Time),
          Min = minute(Time)) %>%
  select( Date, Year, Month, Day, Hour, Min, Global_active_power)

summary(act_power_all$Global_active_power)
```

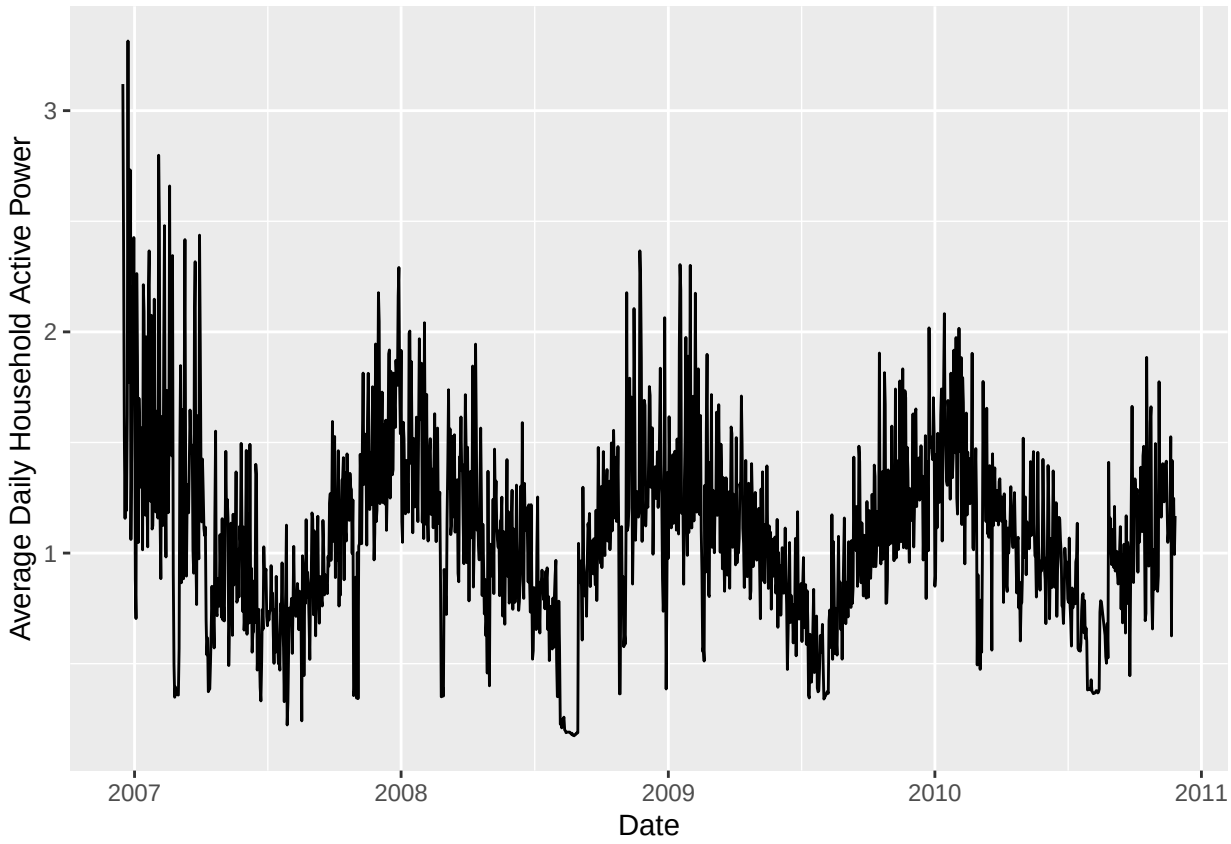
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.076   0.308   0.602   1.092   1.528  11.122   25979
```

Note that we have observations every minute. For electricity consumption we are usually interested in forecasting hourly or daily demand. So let's aggregate observations by hour and then by day.

```
#Creating a data frame with hourly observations
act_power_hourly <- act_power_all %>%
  filter( !is.na(Global_active_power)) %>%
  group_by(Date,Year,Month,Day,Hour) %>%
  summarise( hourly_mean_active_power = mean(Global_active_power))

ggplot(act_power_hourly, aes(x=Date,y=hourly_mean_active_power)) +
  geom_line() +
  ylab("Average Hourly Household Active Power")
```




```
#Creating a data frame with daily observations
act_power_daily <- act_power_hourly %>%
  filter( !is.na(hourly_mean_active_power)) %>%
  group_by(Date,Year,Month,Day) %>%
  summarise( daily_mean_active_power = mean(hourly_mean_active_power))

ggplot(act_power_daily, aes(x=Date,y=daily_mean_active_power)) +
  geom_line() +
  ylab("Average Daily Household Active Power")
```




```
summary(act_power_daily$daily_mean_active_power)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1738  0.8168  1.0807  1.0922  1.3239  3.3149
```

```
#no NAs
```

Transforming both series into time series objects

Let's use `msts` on our two aggregated data sets.

```
ts_act_power_hourly <- msts(act_power_hourly$hourly_mean_active_power,
                            seasonal.periods =c(24,168,8766),
                            start=c(2006,12,16))

ts_act_power_daily <- msts(act_power_daily$daily_mean_active_power,
                           seasonal.periods =c(7,365.25),
                           start=c(2006,12,16))
```

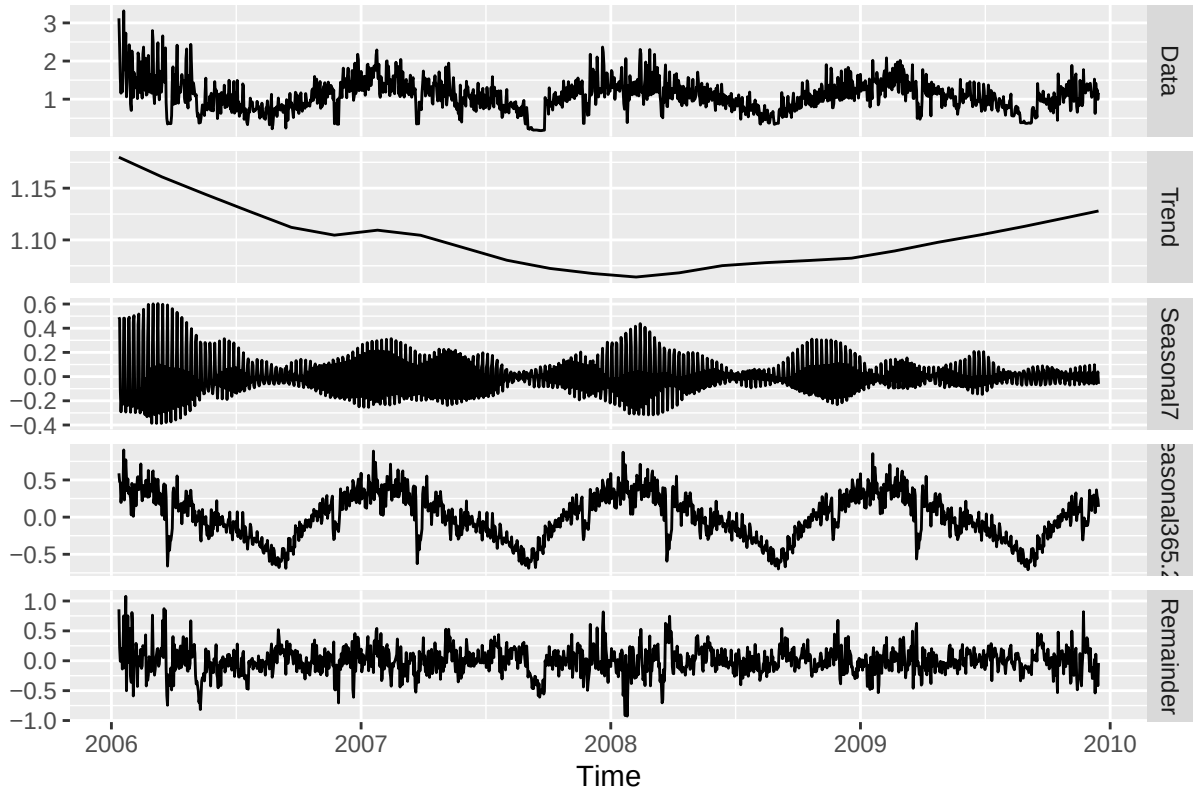
Decomposing both time series objects

So far we used `decompose()` and `stl()` to decompose time series in seasonal, trend and random. For higher frequency data we will use a variation of `stl()`, the `mstl()` that can handle `msts` objects.

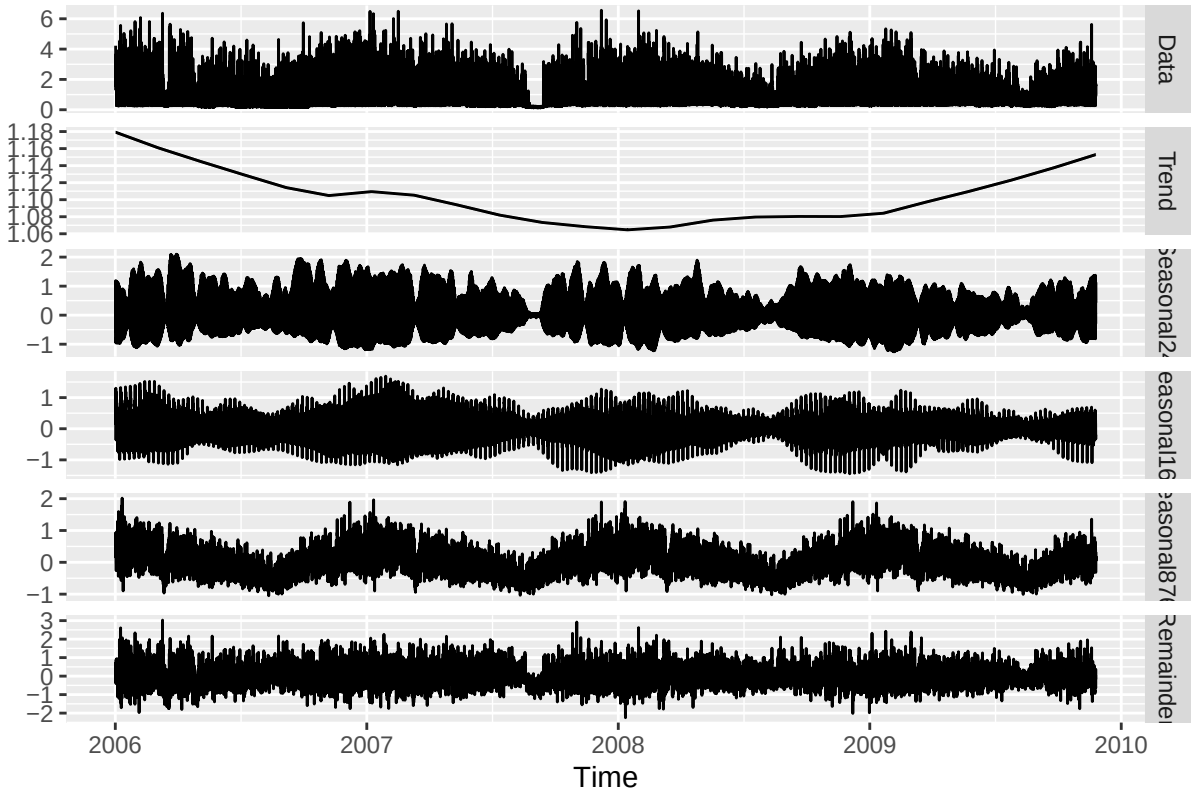
Since we are only analysing the components and don't need to store in our environment, let's use pipes to generate the decomposition plot.

```
## ts_act_power_daily %>% mstl() = mstl(ts_act_power_daily)
ts_act_power_daily %>% mstl() %>%
```

```
  autoplot()
```




```
ts_act_power_hourly %>% mstl() %>%
  autoplot()
```



For the hourly data note three seasonal components based on the frequency we specified in the `msts` object. To interpret this graph look at the vertical scales. The trend has relatively narrow range compared to the other components, because there is little trend seen in the data. As for the seasonal components still looking at vertical scales, yearly and daily pattern are stronger than weekly seasonality.

For the daily data both weekly and yearly pattern have strong scales and trend components. once again has a narrow range compared to seasonal components.

Forecasting Daily Active Power

Now that we have the aggregated and tidy data we can start fitting models and forecasting. For the purpose of this lesson we will work with the daily series. *You can work on your own with the hourly series.*

Start by creating a subset of the time series to fit the models and leave observations out to check performance of the model. We will set aside a year of data out-of-sample analysis.

```
#create a subset for training purpose
n_for = 365
ts_act_power_daily_train <- subset(ts_act_power_daily,
                                   end = length(ts_act_power_daily)-n_for)
```

Model 1: STL + ETS

Forecasts of `stl` objects can be obtained using function `stlf()`. Forecast will obtained by applying a non-seasonal exponential smoothing model to the seasonally adjusted data and re-seasonalizing using the last year of the seasonal component (for all seasonal components).

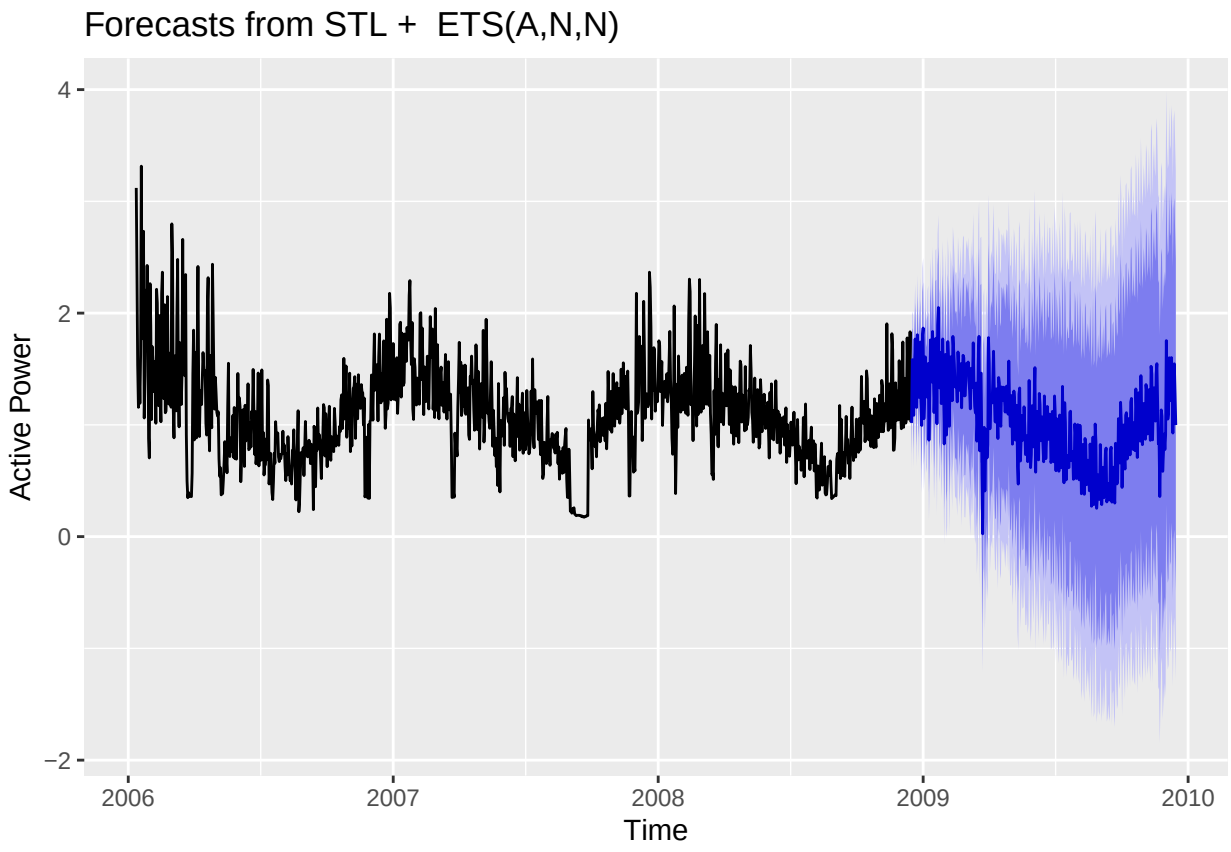
The model used with `stfl` is a ETS(.,.,.) model. ETS stands for **E**rror, **T**rend, **S**easonal and can also be be thought of as **E**xponen**T**ial **S**moothing.

Possibilities for each component: Error={A,M}, Trend={N,A,M} and Seasonal = {N,A,M}. Where N =

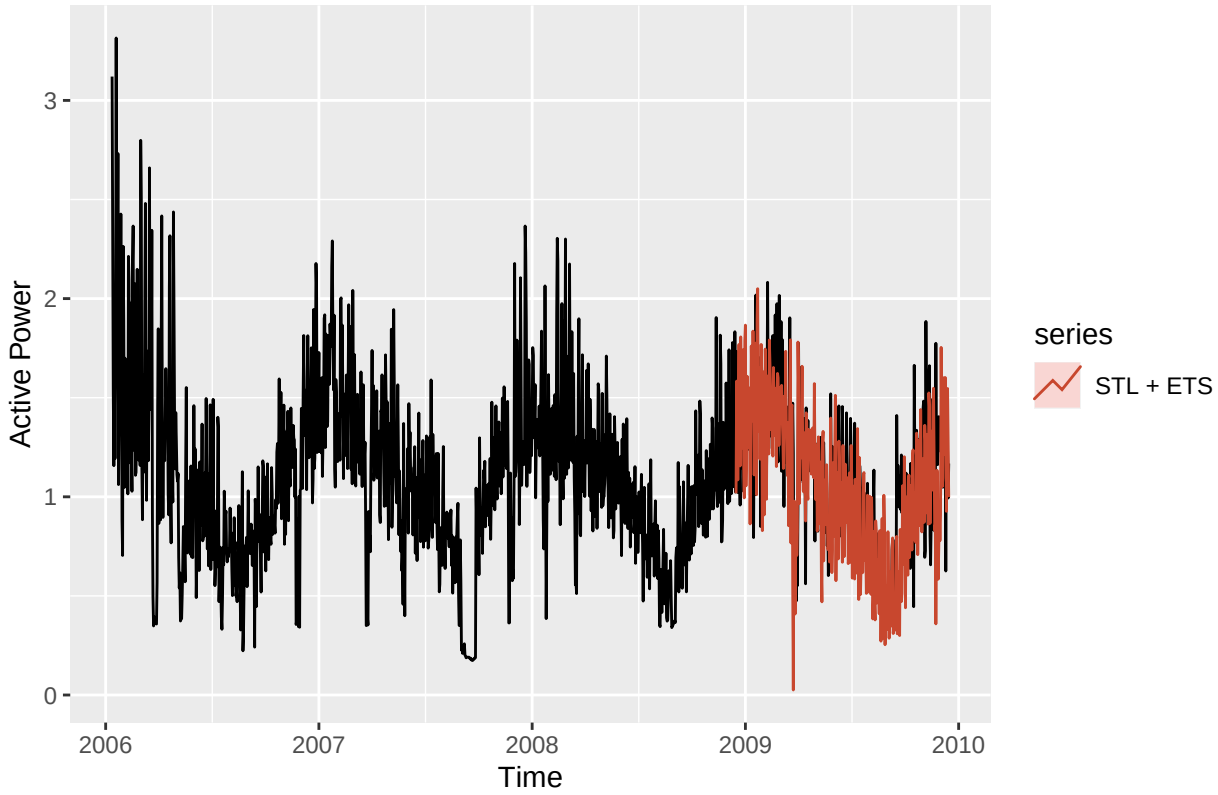
none, A = additive and M = multiplicative. R will **automatically** find the best fit for each component. For more info and ETS equations refer to this link.

Functions: `es()`, `ets()`, `hw()`

```
#Fit and forecast STL + ETS model to data
ETS_fit <-  stlf(ts_act_power_daily_train,h=365)

#Plot foresting results
autoplot(ETS_fit) + ylab("Active Power")
```




```
#Plot model + observed data
autoplot(ts_act_power_daily) +
  autolayer(ETS_fit, series="STL + ETS",PI=FALSE) +
  ylab("Active Power")
```




```
#alternative coding if you don't need to save the objects
#ts_act_power_daily_train %>% stlf(h=365) %>% autoplot()
#very easy to read/run/implement
#but you don't have anything stored on your environment
```

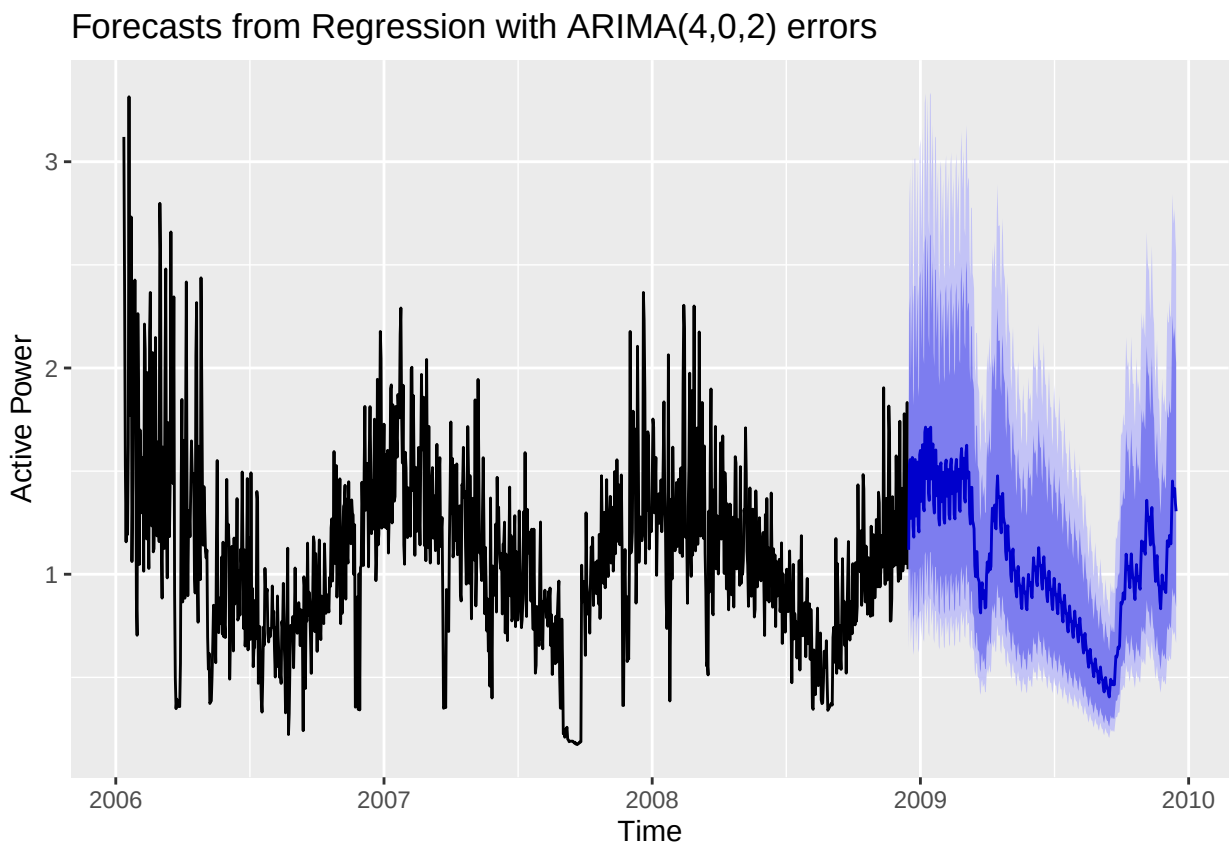
Model 2: ARIMA + FOURIER terms

Since we have multiple seasonalities, the SARIMA model will not work. But we can work with an ARIMA model with Fourier terms for each seasonal period. This models is known as a dynamic harmonic regression model with an ARMA error structure. We will use a log transformation (`lambda=0`) in the `auto.arima()` to ensure the forecasts and prediction intervals remain positive. And for the fourier terms we will use function `fourier()` from package `forecast`. The number of fourier terms needs are specified in argument `K=`. K should be a vector of integers specifying the number of sine and cosine terms for each of the seasonal periods. Ideally K is chosen to minimize the AICc, but we will not go over it.

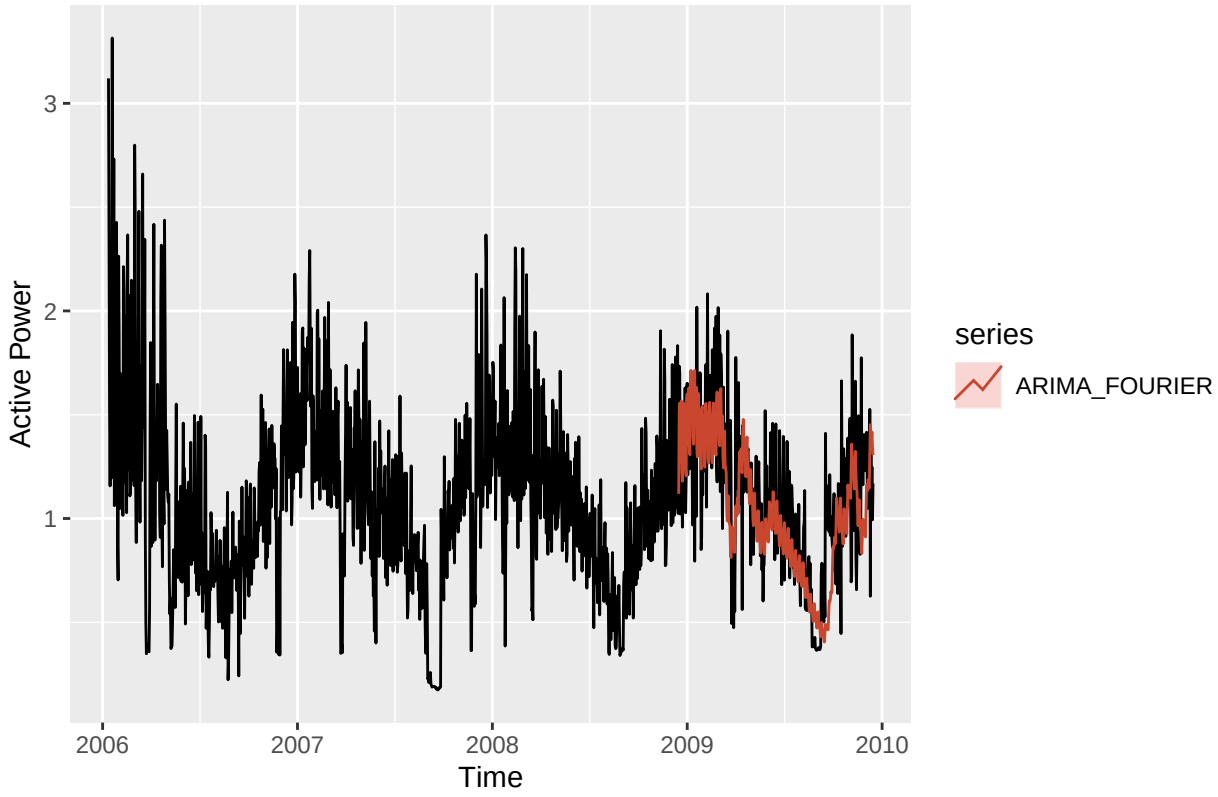
Fourier terms are used to model the seasonal components. It resembles the seasonal dummies we used in previous modules, but they are based on trigonometrics functions. More info on fourier terms here.

```
#Fit arima model with fourier terms as exogenous regressors
# seasonal = FALSE is the same as P=D=Q=0
ARIMA_Four_fit <- auto.arima(ts_act_power_daily_train,
                             seasonal=FALSE,
                             lambda=0,
                             xreg=fourier(ts_act_power_daily_train,
                                          K=c(2,12))
                             )

#Forecast with ARIMA fit
#also need to specify h for fourier terms
ARIMA_Four_for <- forecast(ARIMA_Four_fit,
```

```
                          xreg=fourier(ts_act_power_daily_train,
                                       K=c(2,12),
                                       h=365),
                          h=365
                          )

#Plot foresting results
autoplot(ARIMA_Four_for) + ylab("Active Power")
```




```
#Plot model + observed data
autoplot(ts_act_power_daily) +
  autolayer(ARIMA_Four_for, series="ARIMA_FOURIER",PI=FALSE) +
  ylab("Active Power")
```

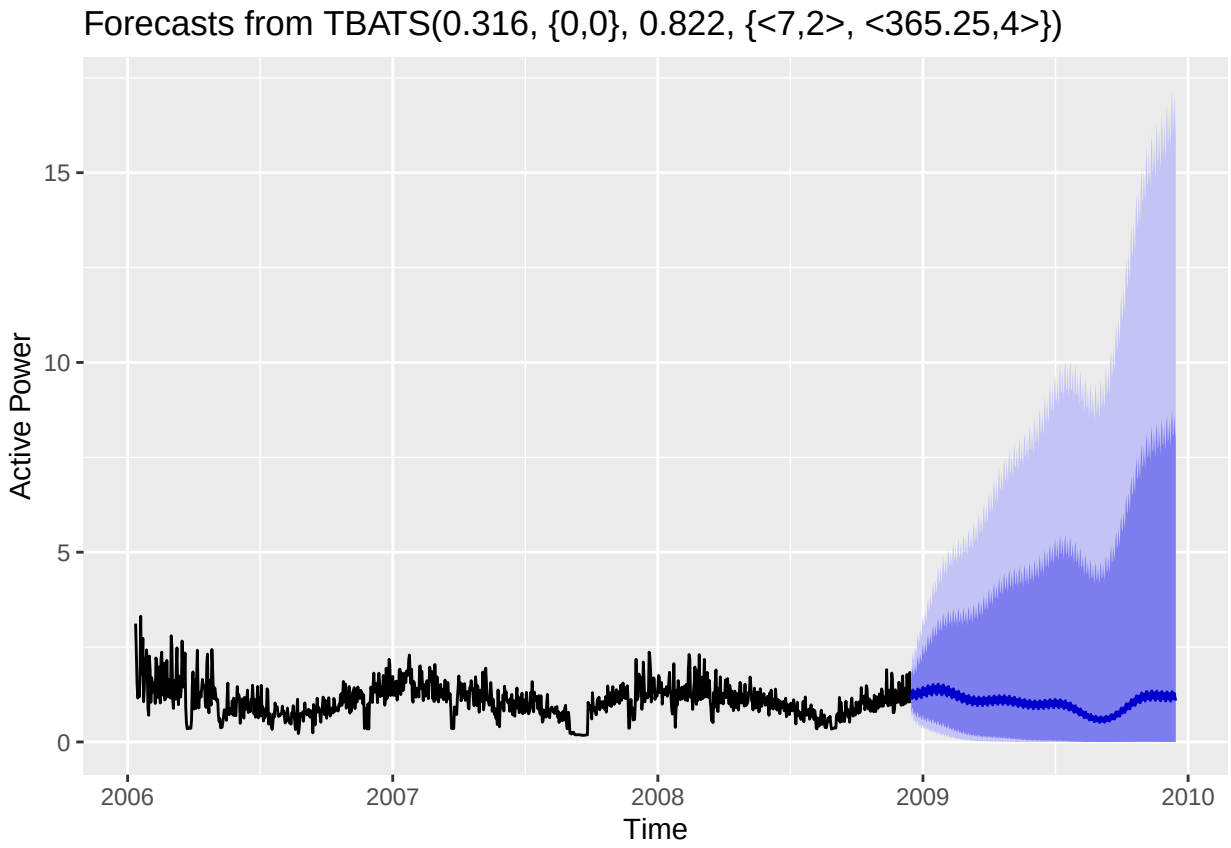


Model 3: TBATS

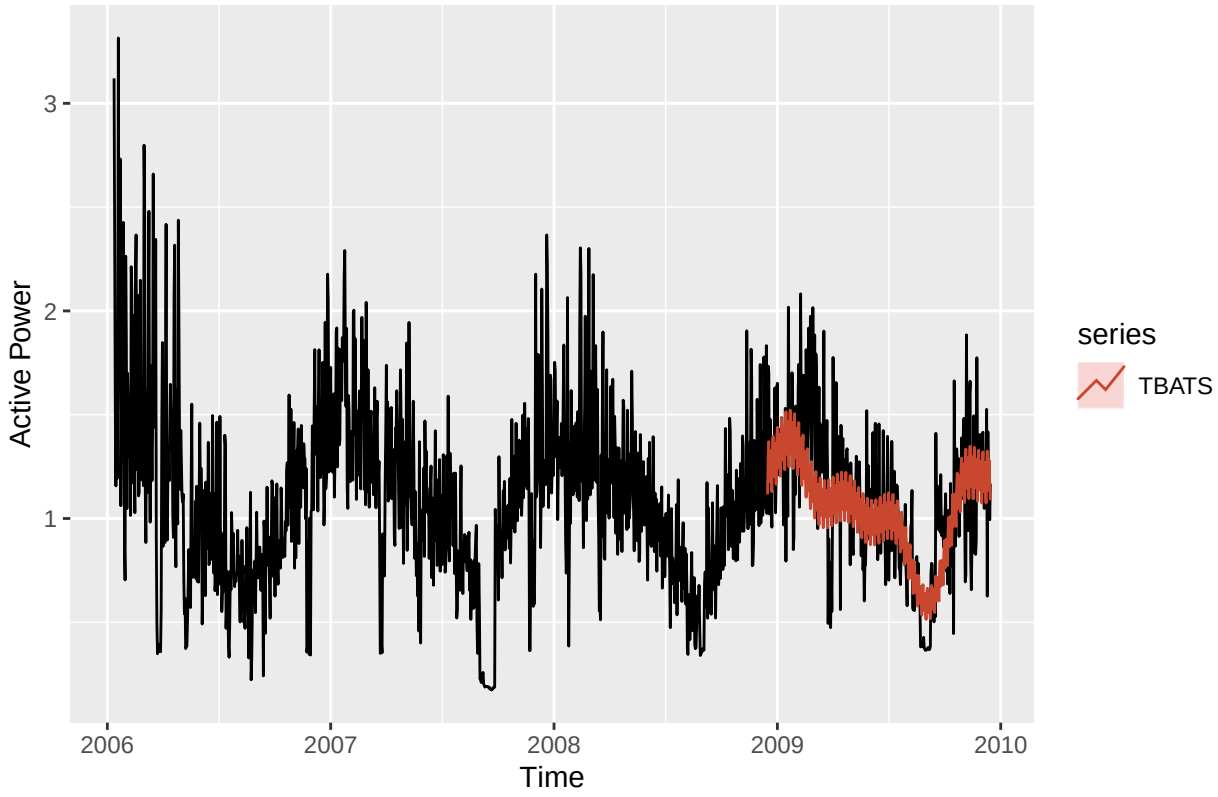
BATS is Exponential smoothing state space model with **B**ox-Cox transformation, **A**RMA errors, **T**rend and **S**easonal components. TBATS is a trigonometric seasonal variation of BATS. The Box-Cox Transformation is used to deal with non-linear data.

More info on BATS and TBATS with corresponding equations here.

```
# TBATS can take time to fit
TBATS_fit <- tbats(ts_act_power_daily_train)

TBATS_for <- forecast(TBATS_fit, h=365)

#Plot foresting results
autoplot(TBATS_for) +
  ylab("Active Power")
```




```
#Plot model + observed data
autoplot(ts_act_power_daily) +
  autolayer(TBATS_for, series="TBATS",PI=FALSE)+
  ylab("Active Power")
```



Checking accuracy of the three models

```
nobs <- length(ts_act_power_daily)
last_obs <- ts_act_power_daily[(nobs-n_for+1):nobs]

#Model 1: STL + ETS
ETS_scores <- accuracy(ETS_fit$mean,last_obs)

#Model 2: ARIMA + Fourier
ARIMA_scores <- accuracy(ARIMA_Four_for$mean,last_obs)

# Model 3:  SARIMA
TBATS_scores <- accuracy(TBATS_for$mean,last_obs)
```

Compare performance metrics

Now we will create a data frame that combines performance metrics for all the three models. You can choose one metric to help you choose among models.

```
#create data frame
scores <- as.data.frame(
  rbind(ETS_scores, ARIMA_scores, TBATS_scores)
  )
row.names(scores) <- c("STL+ETS", "ARIMA+Fourier","TBATS")

#choose model with lowest RMSE
best_model_index <- which.min(scores[,"RMSE"])
cat("The best model by RMSE is:", row.names(scores[best_model_index,]))
```

```
## The best model by RMSE is: TBATS
```

If you want generate a table to compare model accuracy and help visualize the results here is a suggestion on how to include a table on your Rmd report. You can use the `kable_styling(latex_options="striped")` to highlight the model that leads to minimum RMSE.

Table 3: Forecast Accuracy for Daily Active Power

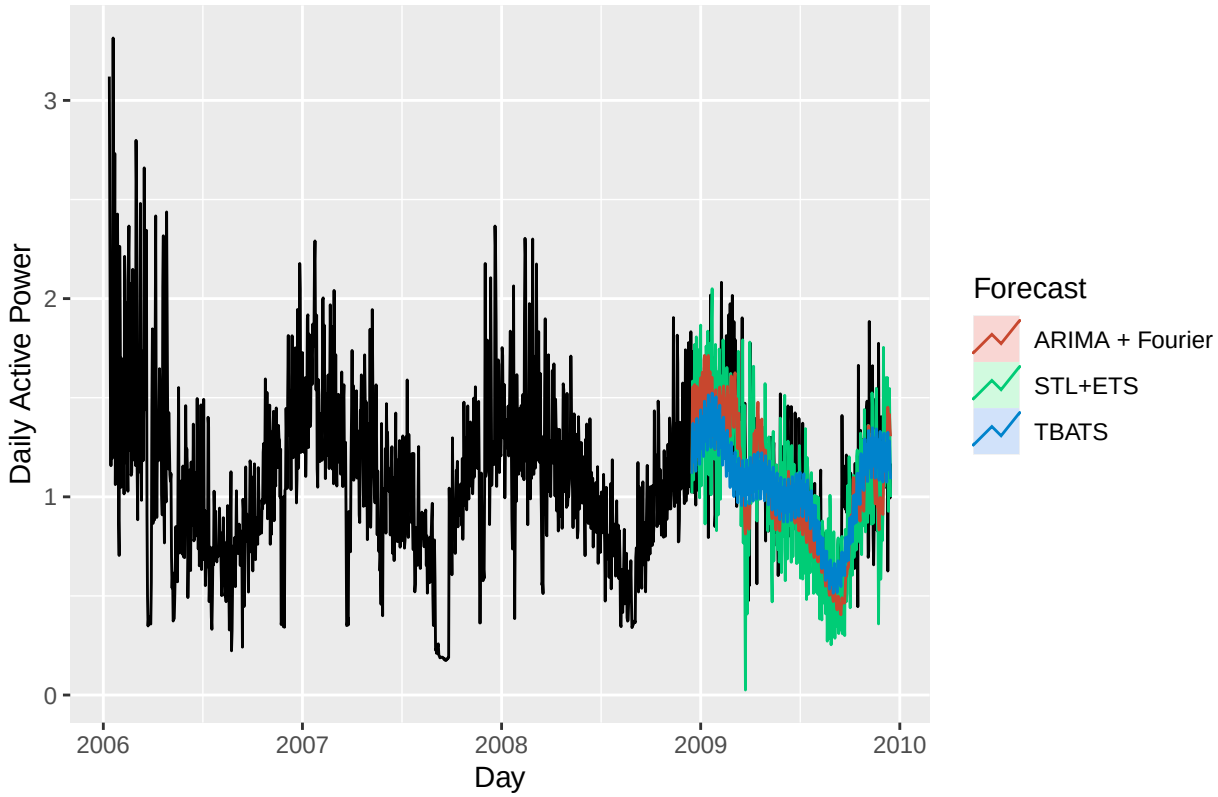
	ME	RMSE	MAE	MPE	MAPE
STL+ETS	0.05139	0.34345	0.26739	0.66625	26.67199
ARIMA+Fourier	0.04463	0.28064	0.21317	-0.55505	21.11937
TBATS	0.05222	0.27994	0.21273	-2.20272	21.58328

Plotting everything together

Here we will use autoplot() and autolayer() from package `ggplot2` to draw a particular plot for time series. The function autolayer() takes a few main arguments.

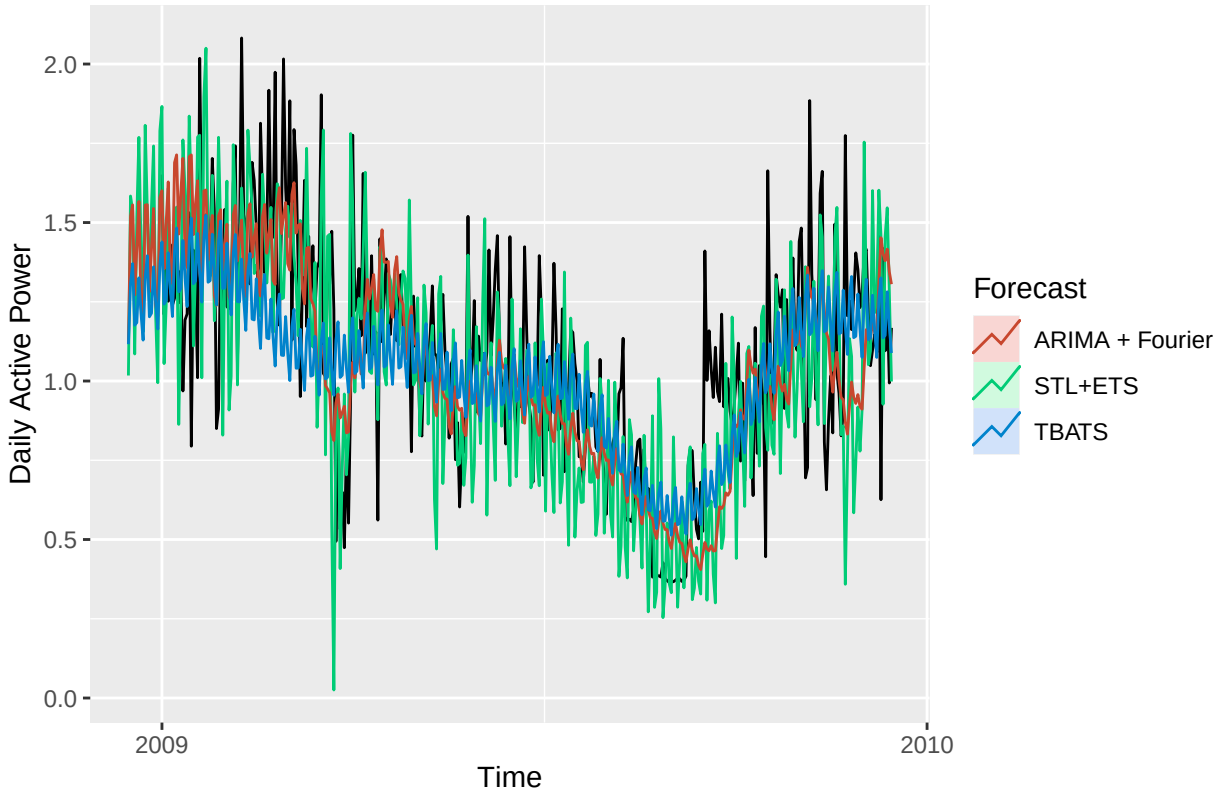
- **x** Forecast object produced by forecast() function.If forecasts were generated with another function you may need to point to the object either mean or forecast to get the values.
- **include** number of values from time series to include in plot.Default is all values.
- **PI** Logical flag indicating whether to plot prediction intervals.
- **series** Matches an unidentified forecast layer with a colored object on the plot.

```
autoplot(ts_act_power_daily) +
  autolayer(ETS_fit, PI=FALSE, series="STL+ETS") +
  autolayer(ARIMA_Four_for, PI=FALSE, series="ARIMA + Fourier") +
  autolayer(TBATS_for,PI=FALSE, series="TBATS") +
  xlab("Day") + ylab("Daily Active Power") +
  guides(colour=guide_legend(title="Forecast"))
```



If you want a closer look on last year just change the xlab range.

```
autoplot(window(ts_act_power_daily,start=2009)) +
  autolayer(ETS_fit, PI=FALSE, series="STL+ETS") +
  autolayer(ARIMA_Four_for, PI=FALSE, series="ARIMA + Fourier") +
  autolayer(TBATS_for,PI=FALSE, series="TBATS") +
  ylab("Daily Active Power") +
  guides(colour=guide_legend(title="Forecast"))
```


Daily Active Power
2.0
1.5
1.0
0.5
0.0
2009
2010
Time
Forecast
ARIMA + Fourier
STL+ETS
TBATS