Drexel University | FIN 642 | Business Conditions and Forecasting

ENERGY CONSUMPTION FORECASTING

GROUP 7 : Saniya Khan | Sasidhar Sirivella | Pawan Punera

INTRODUCTION

Machine learning models like Time series produce accurate energy consumption forecasts and they can be used by facilities managers, utility companies and building commissioning projects to implement energy-saving policies. We believe that efforts towards estimating energy consumption and developing tools for researchers to advance their research in energy consumption are necessary for a more scalable and sustainable future.

DATA OVERVIEW

The dataset is obtained from PJM Interconnection which is a regional transmission organization in the United States. PJM is part of the Eastern Interconnection grid operating an electric transmission system serving all or parts of Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, New Jersey, North Carolina, Ohio, Pennsylvania, Tennessee, Virginia, West Virginia and the District of Columbia. The hourly power consumption data are in megawatt. Here, we are just selecting the power consumption for the Duquesne Light Company, which operates primarily in Pittsburgh and surrounding areas for our project.

DATA DESCRIPTION

Data set contains complete power consumption hourly data through 2005 Dec – 2018 Jan. The file has 119069 observations (hourly data) with two variables as shown below:

Date (type time): time frame at which energy was consumed.

Megawatt Energy consumption (type integer): Energy consumption of a particular region.

PROBLEM STATEMENT

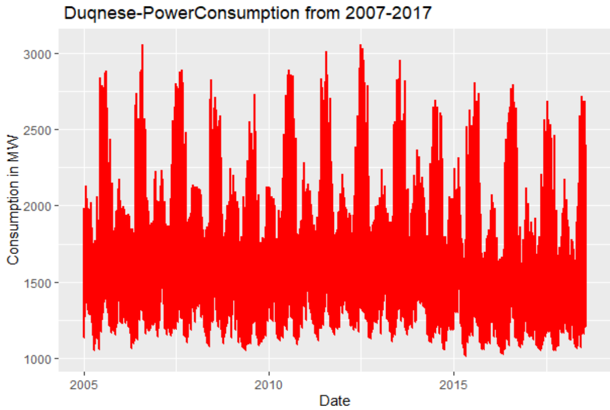
We seek to analyze the electricity consumption data and present the performance of basic forecasting models, STL (Seasonal and Trend decomposition using Loess) with multiple seasonal periods, ETS, ARIMA, TBATS and compare them.

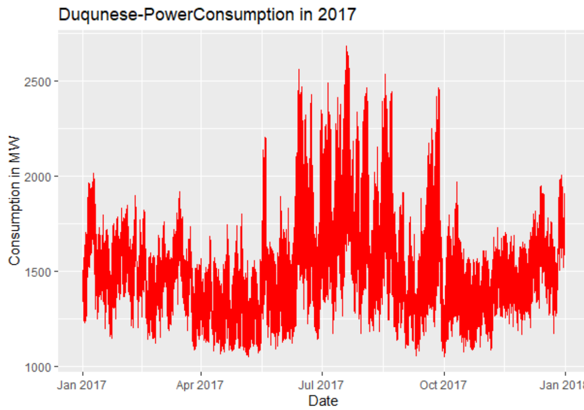
EXPLORATORY ANALYSIS

Our data set has complete **hourly** power consumption data of Duquesne Electric Company from 2005 to 2018. It has 119068 observations of 2 variables (Datetime and DUQ_MW).

However, higher frequency time series often exhibit more complicated seasonal patterns. Hourly data usually has three types of seasonality: a daily pattern, a weekly pattern, and an annual pattern. To deal with such series, we will use the **msts** class which handles multiple seasonality time series which will be discussed later. This allows us to specify all of the frequencies that might be relevant. It is also flexible enough to handle non-integer frequencies.

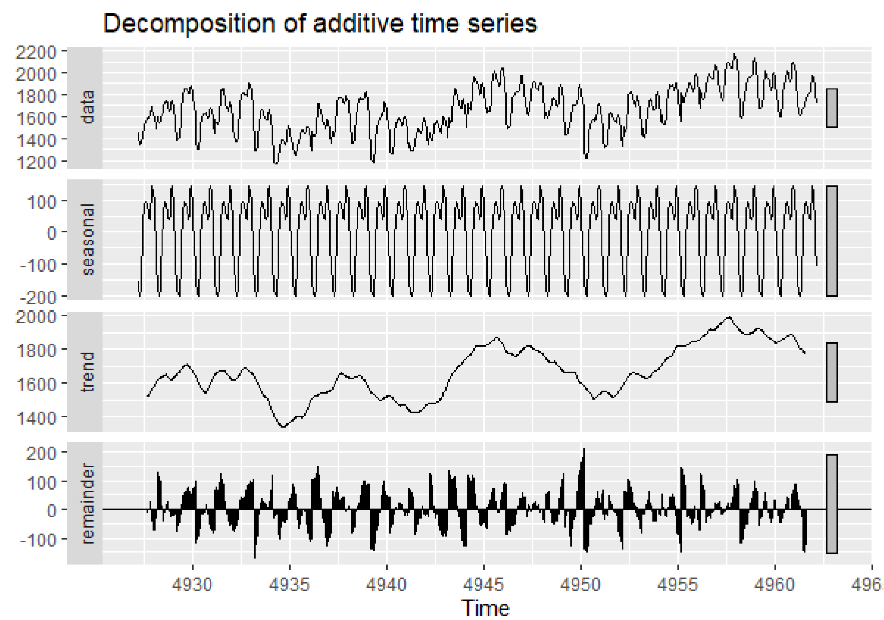
Before we start exploring the data, we are going to read the data, manipulate it and then visualize our data. Overall, there is a clear trend and strong seasonality in the data set and can be seen in below graph. Now let's visualize for one particular year to better understand the data, its trend and seasonality. Let's consider the power consumption in 2017.





Power consumption peaks during summer months from Mid-June to October and reduces from September and again increased slightly from December. This shows that people use electricity more in summers and winters for cooling and heating.

Next, for the ease of visualization and minimizing the processing time, we are restricting our data to 2013-2017. We then, estimated the trend component and seasonal component of our subset data using decompose() function. Trend, seasonal, and irregular components of our data can be estimated using this function.



Decomposition:

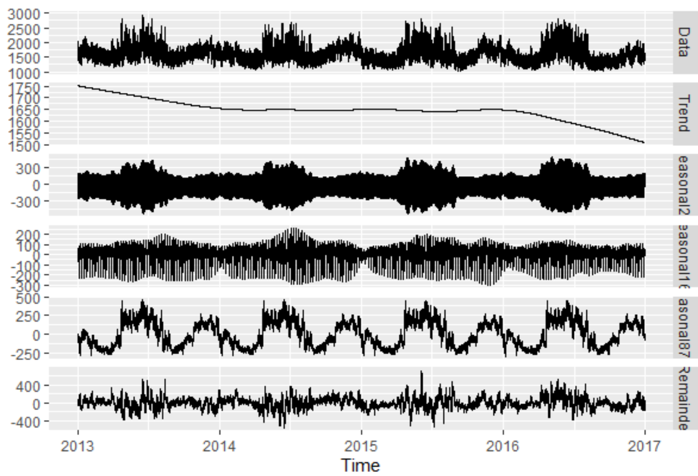
- The first row shows our original time series.
- The second panel shows the seasonal component, with the figure being computed by taking the average for each time unit over all periods and then centering it around the mean.
- The third panel plots the trend component and we see a clear trend pattern. This might be sourced from uncaptured extra seasonality from higher natural period in this case and with our huge data. Hence it can be considered as multi-seasonal data. To deal with such series, we will use the msts class which handles multiple seasonality time series. This allows us to specify all of the

frequencies that might be relevant. Additionally, using msts instead of ts allows us to specify multiple seasons/cycles, for instance hourly as well as daily: c(24, 24*7,24*365.25)
- The last panel shows the remainder component, which is left over data after removing the trend and seasonal components.

Converting to time series:

The next step is to store the data in a time series object, so that we can use many R functions for analyzing our time series data. To store the data in a time series object, we can use the ts() function in R. Sometimes the time series data set that you have, may have been collected at regular intervals that were less than one year, for example, monthly or quarterly. In this case, you can specify the number of times that data was collected per hour by using the frequency parameter in the ts() function. Because each row representing a data within hourly interval, we can set frequency=24, and we will only use Duquesne Electric Company provider.

Now we see a clearer trend in the below graph after decomposing using mstl() and could confirm the daily, weekly and yearly seasonality for our data. Seasonal 168 panel shows the weekly seasonality and seasonal 24 shows the daily seasonality.



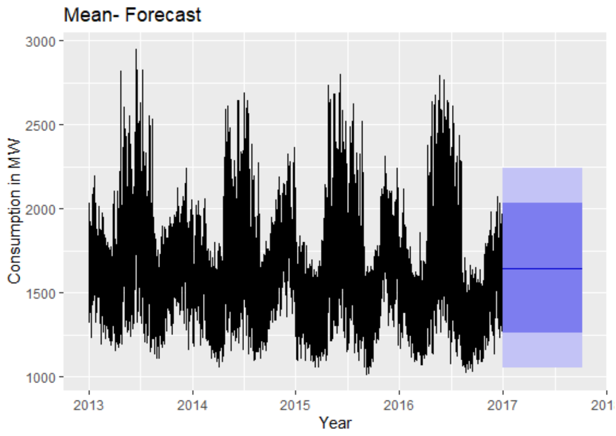
MODELLING – FORECASTING

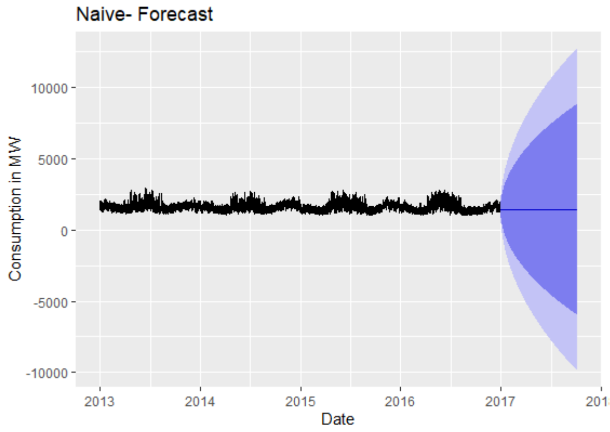
We first split our data into test and train sets. Train data set includes the data from 2013 -2016 and test data contains data from Jan 2017 -Sept 2017 (around **20 %** of data). To deal with multiple seasonality's, we plan to use ARIMA, TBATS, STLM along with few simple basic forecasting models. Before we jump into ARIMA, STLM and TBATS we will first establish baseline forecasting using simple models like Mean, Naïve, and Seasonal Naïve.

Mean & Naïve Forecasts:

The easiest rough estimate for any forecast would be simply the mean or naïve models. So, after training the models with train data, we have established the mean & naïve baseline forecasts for the test data set.

Naïve model has very high prediction intervals which makes it a worse model for our data. Although, mean forecasts has low intervals it failed to consider the high seasonality in the data.

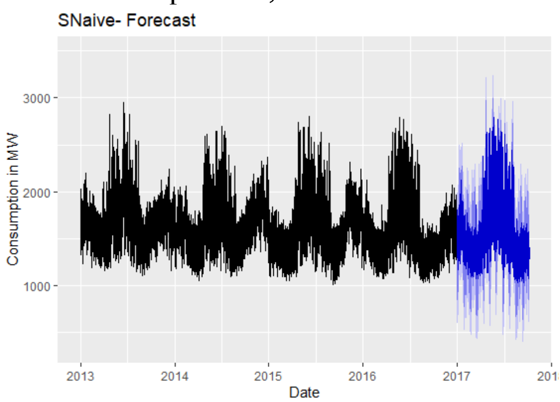


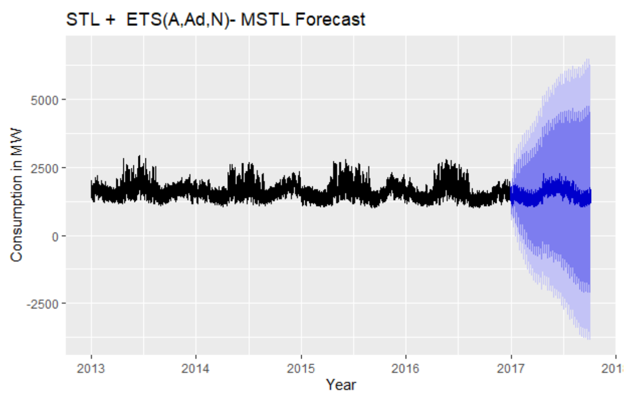


SNaive & MSTL (STL + ETS) Forecasts:

SNaive method is useful for highly seasonal data. In this case, we set each forecast to be equal to the last observed value from the same season of the year.

The mstl() function is a variation on stl() designed to deal with multiple seasonality. It will return multiple seasonal components, as well as a trend and remainder component and here's how it works.



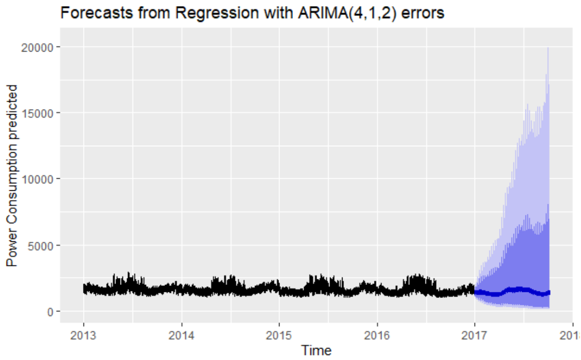


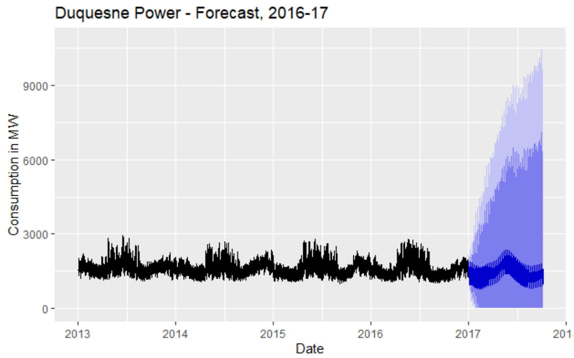
ARIMA & TBATS Forecasts:

With multiple seasonality's, we can use Fourier terms and will fit a dynamic harmonic regression model with **ARIMA.** The only drawback here is that it assumes the frequencies stays constant.

A TBATS model differs from dynamic harmonic regression in that the seasonality is allowed to change slowly over time in a TBATS model, while harmonic regression terms force the seasonal patterns to repeat periodically without changing. One drawback of TBATS models, however, is that they can be slow to estimate, especially with long time series. One advantage of the TBATS model is the seasonality is allowed to change slowly over time.

Here the prediction intervals appear to be much too wide – something that seems to happen quite often with TBATS models unfortunately.





CONCLUSION

Judgement Criteria: We are going to use the RMSE, MAE & size of the prediction intervals as the metrics to compare different models.

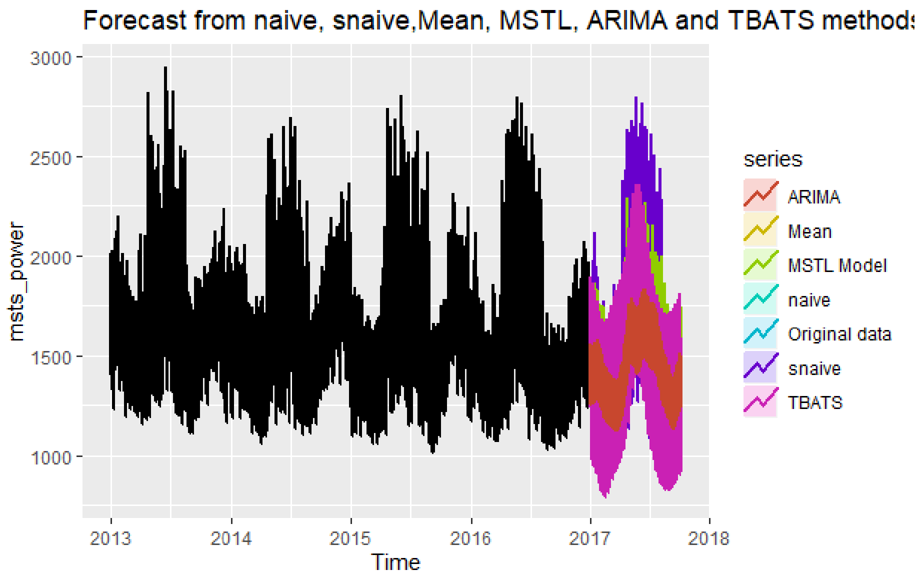
Forecast Metrics Comparison:

```
> kable(Summary_table)
```

ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Split	Method
0.0000000	302.64348	237.40073	-3.2550468	14.6892792	4.6637796	0.9728778	Train	Mean
-84.6340688	303.59034	245.62702	-8.8493055	16.5967126	4.8253865	NA	Test	Mean
-0.0040527	70.47619	50.90308	-0.0952334	3.1421760	1.0000000	0.6255802	Train	Naive
140.8913743	323.81252	238.05178	6.0598402	14.1099969	4.6765697	NA	Test	Naive
0.0000000	70.47619	50.90313	-0.0949793	3.1421745	1.0000010	0.6255802	Train	Drift
154.1195266	328.50970	242.70399	6.9472810	14.3206002	4.7679631	NA	Test	Drift
-32.0274046	354.41441	271.53663	-4.3722712	16.9088447	5.3343855	0.9620306	Train	Snaive
-53.1647005	436.71906	345.04765	-6.2877663	22.4494717	6.7785225	NA	Test	Snaive
-0.0046539	25.73499	15.61940	-0.0116438	0.9665696	0.3068460	0.0328774	Train	STLM
3.3903393	421.15475	336.70676	-3.7400250	21.7411500	6.6146643	NA	Test	STLM
0.4134288	48.33397	32.73674	-0.0468720	2.0272194	0.6431191	0.0337125	Train	ARIMA
140.2493658	426.66738	325.07048	5.0128208	19.8811148	6.3860675	NA	Test	ARIMA
0.0069858	44.05777	27.75356	-0.0416685	1.7313406	0.5452235	0.0129307	Train	TBATS
103.3999728	464.34704	369.02577	2.9842944	23.4373714	7.2495770	NA	Test	TBATS

Looking at forecast from all the models, the forecast from mstl() or stlm is showing a better performance **(lower RMSE & smaller prediction intervals)** and is definitely our winner here.

Comparing all the forecasts with Plots:



REFERENCES

Data Source. (2018). Retrieved from https://www.kaggle.com/robikscube/hourly-energy-consumption?select=DUQ_hourly.csv

Rob J Hyndman and George Athanasopoulos. (2018). Forecasting: Principles and Practice. Retrieved from https://otexts.com/fpp2/complexseasonality.html

Ajeng Prastiwi. (2019, June 23). Multiple Seasonality. Retrieved from https://rpubs.com/AlgoritmaAcademy/multiseasonality

Bhide, A. (2019). Reference. Retrieved from https://www.kaggle.com/apoorvabhide/energy-consumption-time-series-forecasting-in-r/#data

APPENDIX

Step wise results from R:

Loading the data:

```
## Reading the data
duq_pc <- read.csv(file.choose(),stringsAsFactors = F)
```

```
> ## Reading the data
> head(duq_pc)
             Datetime DUQ_MW
1 2005-12-31 01:00:00   1458
2 2005-12-31 02:00:00   1377
3 2005-12-31 03:00:00   1351
4 2005-12-31 04:00:00   1336
5 2005-12-31 05:00:00   1356
6 2005-12-31 06:00:00   1372
> str(duq_pc)
'data.frame':   119068 obs. of  2 variables:
 $ Datetime: POSIXct, format: "2005-12-31 01:00:00" "2005-12-31 02:00:00" ...
 $ DUQ_MW  : num  1458 1377 1351 1336 1356 ...
>
```

Subsetting and converting the data into a time series object

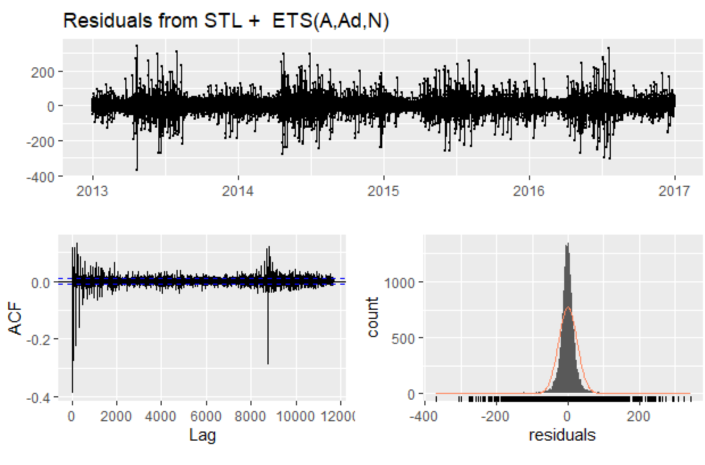
```
##-----Modelling-------

#For ease of visualisation and minimising the processing time, we are restricting our
#data to 2011-2017


duq_new <- duq_pc[duq_pc$Datetime >= '2013-01-01 00:00:00' & duq_pc$Datetime <= '2017-09-30 00:00:00',]
#Dividing our data into train and test

duq_train <- duq_new[duq_new$Datetime <= '2016-12-31',]
duq_test <- duq_new[duq_new$Datetime >= '2017-01-01',]
msts_power <- msts(duq_train$DUQ_MW, seasonal.periods = c(24,24*7,24*365.25), start = decimal_date(as.POSIXct("2013-01-01 00:00:00")))
```

Mean Forecasting:

```
> summary(mean_baseline)

Forecast method: Mean

Model Information:
$mu
[1] 1646.525

$mu.se
[1] 1.61682

$sd
[1] 302.6478

$bootstrap
[1] FALSE

$call
meanf(y = msts_power, h = 24 * 7 * 40)

attr(,"class")
[1] "meanf"

Error measures:
                       ME     RMSE      MAE       MPE     MAPE      MASE      ACF1
Training set 3.508914e-14 302.6435 237.4007 -3.255047 14.68928 0.8742862 0.9728778

Forecasts:
            Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2016.99715       1646.525 1258.654 2034.397 1053.318 2239.733
2016.99726       1646.525 1258.654 2034.397 1053.318 2239.733
2016.99738       1646.525 1258.654 2034.397 1053.318 2239.733
2016.99749       1646.525 1258.654 2034.397 1053.318 2239.733
2016.99760       1646.525 1258.654 2034.397 1053.318 2239.733
2016.99772       1646.525 1258.654 2034.397 1053.318 2239.733
```

STLM Forecasting (STL+ETS):

```
> summary(fcast_mstl)

Forecast method: STL +  ETS(A,Ad,N)

Model Information:
ETS(A,Ad,N)

Call:
 ets(y = na.interp(x), model = etsmodel, allow.multiplicative.trend = allow.multiplicativ
e.trend)

  Smoothing parameters:
    alpha = 0.9999
    beta  = 0.058
    phi   = 0.8

  Initial states:
    l = 1754.1249
    b = 6.0828

  sigma:  25.7368

     AIC     AICc      BIC
594270.6 594270.6 594321.4

Error measures:
                       ME     RMSE     MAE        MPE      MAPE       MASE       ACF1
Training set -0.004653861 25.73499 15.6194 -0.0116438 0.9665696 0.05752227 0.03287738

Forecasts:
           Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
2016.99715       1313.155 1280.1723 1346.138 1262.7121 1363.599
2016.99726       1286.233 1238.4963 1333.970 1213.2258 1359.241
2016.99738       1278.740 1219.1098 1338.370 1187.5435 1369.936
```



Naïve Forecasting :

```
> summary(fcast_naive)

Forecast method: Naive method

Model Information:
Call: naive(y = msts_power, h = 24 * 7 * 40)

Residual sd: 70.4772

Error measures:
                      ME     RMSE      MAE         MPE     MAPE     MASE      ACF1
Training set -0.004052743 70.47619 50.90308 -0.09523341 3.142176 0.187463 0.6255802

Forecasts:
           Point Forecast     Lo 80    Hi 80       Lo 95    Hi 95
2016.99715           1421 1330.6811 1511.319 1282.869205 1559.131
2016.99726           1421 1293.2698 1548.730 1225.653556 1616.346
2016.99738           1421 1264.5631 1577.437 1181.750445 1660.250
2016.99749           1421 1240.3623 1601.638 1144.738410 1697.262
2016.99760           1421 1219.0409 1622.959 1112.130152 1729.870
2016.99772           1421 1199.7648 1642.235 1082.650034 1759.350
2016.99783           1421 1182.0387 1659.961 1055.540268 1786.460
2016.99795           1421 1165.5397 1676.460 1030.307112 1811.693
2016.99806           1421 1150.0434 1691.957 1006.607614 1835.392
2016.99817           1421 1135.3866 1706.613  984.192072 1857.808
2016.99829           1421 1121.4462 1720.554  962.871980 1879.128
2016.99840           1421 1108.1262 1733.874  942.500889 1899.499
2016.99852           1421 1095.3507 1746.649  922.962335 1919.038
2016.99863           1421 1083.0577 1758.942  904.161890 1937.838
2016.99875           1421 1071.1965 1770.803  886.021731 1955.978
2016.99886           1421 1059.7245 1782.275  868.476819 1973.523
2016.99897           1421 1048.6057 1793.394  851.472141 1990.528
2016.99909           1421 1037.8095 1804.191  834.960668 2007.039
```

Seasonal Naïve Forecasting:

```
> summary(fcast_snaive)

Forecast method: Seasonal naive method

Model Information:
Call: snaive(y = msts_power, h = 24 * 7 * 40)

Residual sd: 352.971

Error measures:
                    ME     RMSE      MAE       MPE     MAPE MASE      ACF1
Training set -32.0274 354.4144 271.5366 -4.372271 16.90884    1 0.9620306

Forecasts:
           Point Forecast     Lo 80  Hi 80     Lo 95    Hi 95
2016.99715           1427  972.7997 1881.2  732.3605 2121.639
2016.99726           1412  957.7997 1866.2  717.3605 2106.639
2016.99738           1400  945.7997 1854.2  705.3605 2094.639
2016.99749           1398  943.7997 1852.2  703.3605 2092.639
2016.99760           1416  961.7997 1870.2  721.3605 2110.639
2016.99772           1458 1003.7997 1912.2  763.3605 2152.639
2016.99783           1417  962.7997 1871.2  722.3605 2111.639
2016.99795           1389  934.7997 1843.2  694.3605 2083.639
2016.99806           1373  918.7997 1827.2  678.3605 2067.639
2016.99817           1366  911.7997 1820.2  671.3605 2060.639
2016.99829           1389  934.7997 1843.2  694.3605 2083.639
2016.99840           1445  990.7997 1899.2  750.3605 2139.639
2016.99852           1549 1094.7997 2003.2  854.3605 2243.639
2016.99863           1613 1158.7997 2067.2  918.3605 2307.639
2016.99875           1651 1196.7997 2105.2  956.3605 2345.639
2016.99886           1684 1229.7997 2138.2  989.3605 2378.639
2016.99897           1696 1241.7997 2150.2 1001.3605 2390.639
2016.99909           1716 1261.7997 2170.2 1021.3605 2410.639
2016.99920           1717 1262.7997 2171.2 1022.3605 2411.639
```

ARIMA with Dynamic Harmonic Regression:

```
> f_fourier <-  forecast(fourier_power, xreg=fourier(msts_power, K=
7*40))
> f_fourier
           Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
2016.99715       1366.697 1314.7254 1420.724 1288.0180 1450.183
2016.99726       1318.215 1236.1781 1405.697 1194.8375 1454.333
2016.99738       1284.173 1178.6474 1399.146 1126.3425 1464.119
2016.99749       1261.272 1138.3261 1397.496 1078.1709 1475.468
2016.99760       1250.339 1114.4949 1402.741 1048.6638 1490.800
2016.99772       1259.261 1112.8239 1424.967 1042.3296 1521.340
2016.99783       1285.288 1129.6571 1462.360 1055.0512 1565.768
2016.99795       1318.449 1155.2941 1504.645 1077.2645 1613.631
2016.99806       1349.360 1180.7442 1542.055 1100.1889 1654.964
2016.99817       1389.101 1214.9891 1588.165 1131.8360 1704.843
2016.99829       1439.384 1258.9112 1645.730 1172.7234 1766.680
2016.99840       1479.920 1294.3394 1692.109 1205.7138 1816.486
2016.99852       1509.667 1320.1608 1726.376 1229.6709 1853.418
2016.99863       1536.510 1343.2791 1757.537 1251.0294 1887.136
2016.99875       1558.422 1362.0580 1783.096 1268.3325 1914.861
2016.99886       1559.613 1362.8488 1784.786 1268.9455 1916.862
2016.99897       1537.662 1343.5939 1759.761 1250.9814 1890.039
2016.99909       1513.324 1322.3250 1731.910 1231.1774 1860.129
```

TBATS Forecasting:

```
> tbats_power
TBATS(0.825, {0,0}, -, {<24,5>, <168,5>, <8766,6>})

Call: tbats(y = msts_power)

Parameters
  Lambda: 0.824918
  Alpha: 0.9913571
  Gamma-1 Values: 0.02230484 0.09414321 -0.06695658
  Gamma-2 Values: 0.01192718 0.01054932 0.01004382
```

```
> f_tbats <- forecast(tbats_power, h = 24*7*40)
> f_tbats
           Point Forecast    Lo 80    Hi 80     Lo 95    Hi 95
2016.99715      1346.7512 1292.2077 1401.684 1263.4953 1430.919
2016.99726      1300.1211 1216.7904 1384.398 1173.0749 1429.381
2016.99738      1293.2136 1188.6486 1399.282 1133.9320 1456.013
2016.99749      1318.6596 1198.0658 1441.218 1135.0649 1506.853
2016.99760      1366.7435 1233.1844 1502.630 1163.4788 1575.458
2016.99772      1436.3286 1290.8032 1584.486 1214.8953 1663.924
2016.99783      1522.6235 1365.3888 1682.757 1283.3988 1768.637
2016.99795      1604.6794 1436.7551 1775.742 1349.2102 1867.499
2016.99806      1659.3296 1482.5890 1839.434 1390.4753 1936.061
2016.99817      1686.4501 1502.5248 1873.961 1406.7059 1974.592
2016.99829      1706.5178 1516.1912 1900.641 1417.0768 2004.851
2016.99840      1729.5298 1533.3264 1929.716 1431.1833 2037.205
2016.99852      1739.7024 1538.6997 1944.862 1434.0937 2055.049
2016.99863      1721.4846 1517.0929 1930.223 1410.7785 2042.374
2016.99875      1692.2459 1485.2275 1903.804 1377.6106 2017.518
2016.99886      1691.8412 1481.8637 1906.492 1372.7406 2021.892
2016.99897      1737.4215 1523.8235 1955.727 1412.7974 2073.076
2016.99909      1800.3795 1583.2563 2022.195 1470.3543 2141.397
2016.99920      1834.7804 1615.0908 2059.183 1500.8381 2179.764
2016.99932      1822.2859 1601.1417 2048.240 1486.1631 2169.677
2016.99943      1778.7202 1556.9068 2005.495 1441.6440 2127.420
2016.99954      1721.9474 1500.0882 1948.938 1384.8809 2071.038
2016.99966      1651.5430 1430.3129 1878.099 1315.5307 2000.037
```

Next step, we have compared the accuracies

```
## Comparing the accuracies
mean_results <-accuracy(mean_baseline,duq_test$DUQ_MW)
naive_results <- accuracy(fcast_naive,duq_test$DUQ_MW)
rwf_results <- accuracy(fcast_rwf,duq_test$DUQ_MW)
snaive_results <- accuracy(fcast_snaive,duq_test$DUQ_MW)
stlm_model_results<- accuracy(fcast_mstl,duq_test$DUQ_MW)
arima_results<- accuracy(f_fourier, duq_test$DUQ_MW)
tbats_results<- accuracy(f_tbats,duq_test$DUQ_MW)


Summary_table= data.table(rbind(mean_results,naive_results,rwf_results,snaive_resul
Summary_table[,Split:=c("Train","Test","Train","Test","Train","Test","Train","Test"
Summary_table[,Method:=c(rep("Mean",2),rep("Naive",2),rep("Drift",2),rep("Snaive",2
kable(Summary_table)
```

```
> kable(Summary_table)
```

ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Split	Method
0.0000000	302.64348	237.40073	-3.2550468	14.6892792	4.6637796	0.9728778	Train	Mean
-84.6340688	303.59034	245.62702	-8.8493055	16.5967126	4.8253865	NA	Test	Mean
-0.0040527	70.47619	50.90308	-0.0952334	3.1421760	1.0000000	0.6255802	Train	Naive
140.8913743	323.81252	238.05178	6.0598402	14.1099969	4.6765697	NA	Test	Naive
0.0000000	70.47619	50.90313	-0.0949793	3.1421745	1.0000010	0.6255802	Train	Drift
154.1195266	328.50970	242.70399	6.9472810	14.3206002	4.7679631	NA	Test	Drift
-32.0274046	354.41441	271.53663	-4.3722712	16.9088447	5.3343855	0.9620306	Train	Snaive
-53.1647005	436.71906	345.04765	-6.2877663	22.4494717	6.7785225	NA	Test	Snaive
-0.0046539	25.73499	15.61940	-0.0116438	0.9665696	0.3068460	0.0328774	Train	STLM
3.3903393	421.15475	336.70676	-3.7400250	21.7411500	6.6146643	NA	Test	STLM
0.4134288	48.33397	32.73674	-0.0468720	2.0272194	0.6431191	0.0337125	Train	ARIMA
140.2493658	426.66738	325.07048	5.0128208	19.8811148	6.3860675	NA	Test	ARIMA
0.0069858	44.05777	27.75356	-0.0416685	1.7313406	0.5452235	0.0129307	Train	TBATS
103.3999728	464.34704	369.02577	2.9842944	23.4373714	7.2495770	NA	Test	TBATS

Finally, we plotted forecasts from all the models:

```
## Comparing the models with help of plots

autoplot(msts_power, series = "Original data") +
  geom_line(size = 1) +
  autolayer(fcast_naive, PI = FALSE, size = 1,
            series = "naive") +
  autolayer(fcast_snaive, PI = FALSE, size = 1,
            series = "snaive") +
  autolayer(fcast_mstl, PI = FALSE, size = 1,
            series = "MSTL Model") +
  autolayer(mean_baseline, PI = FALSE, size = 1,
            series = "Mean") +
  autolayer(f_tbats, PI = FALSE, size = 1,
            series = "TBATS") +
  autolayer(f_fourier, PI = FALSE, size = 1,
            series = "ARIMA") +

  ggtitle("Forecast from naive, snaive,Mean, MSTL, ARIMA and TBATS methods")
```