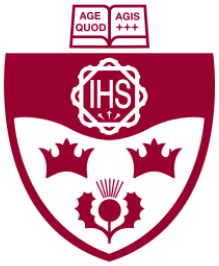

SAINT MARY'S UNIVERSITY SINCE 1802

One University. One World. Yours.

MSc in Computing and Data Analytics

MCDA 5580 – Data and Text Mining

Assignment – 4

Submitted to:

Trishla Shah

Prepared by:

Allen Mathew - A00432526

Gyaneshwar Rao - A00433014

Meghashyam - A00432392

Table of Contents

Executive Summary

The current consumption of electricity in the business world has a requirement to predict the consumption, which in turn is used to calculate, store and manage production of electricity for future purposes and maintain a continuous supply to economic zones, factories etc. GLM, SVM, Neural networks, Random forest and Time Series (Arima) are used to create different models to predict the consumption and calculate the MAPE. The lowest MAPE values for a model are chosen to be the best model for the prediction of consumption in 3 different intervals (15 minutes, hourly and daily).

Objective

We have used electricity consumption dataset which has records from 2011 to 2015 and the dataset specifies the consumption of electricity for every 15-minute interval. We have strictly considered the dataset of the year 2014. Calculated MAPE for all the prediction models such as GLM, SVM, Neural Networks, Random Forest and Time Series (Arima). Choosing the model with the least error is chosen to be the best model, which will help to predict the accurate electricity data consumption points according to the historic data points.

Data Summary

For our analysis, we will be using the "MT123electricity" dataset. It consists of 140,244 records which are from January 2011 to January 2015, of the amount of electricity consumed for every 15-minute interval. The following are the attributes of the data set:

Attributes	Description
RecordDateTime	It is a timestamp indicating the date (yyyy.mm.dd) and time (hh:mm:ss).
Value	It indicates the amount of electric that has been consumed.

Table 1: Attributes of the "MT123electricity" Dataset

Observations:

The following table briefly describes the observation made on each of the attributes in the "MT123electricity" data set:

Attributes	Description
RecordDateTime	• Each record in the attribute is a continuous series of timestamps with a 15-minute interval between January 2011 to January 2015.

Value	• Each record has an Integer value (up to 15-decimal points) that indicates the amount of electricity that is consumed during a 15-minute interval, the specific interval is indicated by the ***RecordDateTime*** Attribute.
	• The values in this attribute ranges between:
	  ○ Min: 9.569377990430620
	  ○ Max: 765.550239234450000

Table 2: Observations made on the Attributes of the "MT123electricity" Dataset

The Dataset follows Daylight Saving Time (DST)

Each record in the dataset consists of a continuous series of electricity consumption during a 15-minute interval along with a timestamp from January 2011 to January 2015.

To calculate the daily and hourly consumption of electricity, we will need to sum up all the records in the dataset for a day (i.e. 96 records) and for every hour of a day (i.e. 4 records).

But the dataset is missing a total of twelve records of the 15-minute intervals, there are four records each that are missing from the following day:

- 2011-03-27
- 2012-03-25
- 2014-03-30

It can be observed that the above days occur during the month of March for almost every consecutive year. We believe that the "MT123electricity" dataset had a Daylight-saving factor. DST is a measure where the clocks are normally set ahead by one hour of the standard time. It is used during the summer when there is seasonal time change. Since the dataset follows the DST measure, then all hours of the day are accounted for and there are no missing records in the dataset.

Descriptive Statistics of the Dataset:

The below table shows the total number of records for the dataset(2011-2015), indicating the electricity consumption for specific intervals .

Length:

Interval	# of data points
Daily	1461
Hourly	35061
15 Minute	140244

Table 3: Total Number of records for specific intervals

The below table, it is evident that the data points for the dataset(2011-2015) are hovering around mean and the data is likely to be stationary.

Mean and Median:

Interval	Mean	Median
Daily	26953	26549
Hourly	1123	1210
15 Minute	280	301

Table 4: Mean & Median for specific intervals

Outcome:

From the “MT123electricity” dataset the latest records **ranging from 2014 to 2015** will be considered for the analysis. Three separate dataset needs to be created from the “MT123electricity” dataset to calculate daily/hourly/15-minute consumption of electricity. SQL commands are used to create the datasets, the filtered records are then exported into their respective CSV files (i.e. 15min.csv, hourly.csv & daily.csv). The SQL command used to create and filter the dataset is give in the Appendix.

The respective CSV files contains data that is represented in 2 columns, but this data format will not be accepted as input for the models that will be used for the analysis. The models requires the data to be in a matrix format, hence the data needs to be transformed.

To transform the data a python script is used (i.e. toMatrix.py). It reads the records in the respective CSV files and from the data it makes a matrix with eight columns. If the last row of the matrix consists of any zeros then that row is truncated, this is done to prevent the dataset form having any bias. The matric is then stored in the respective text files (i.e. 15minmatrix.txt, hourlymatrix.txt, dailymatrix.txt) and will be used as input for the models that will be used for the analysis. The Python Code used to create the matric from the dataset is give in the Appendix.

Design/Method/Approach

The following steps were performed for our analysis:

1. **Dataset Selection:**
   - We begin by selecting the "MT123electricity" dataset that contains records pertaining to the electricity consumption during January 2011 to January 2015.
   - Then we review the source data and perform descriptive statistics to gain a better understanding of the stationarity as well as the structure of the dataset.
   - Then we check if the data is correctly formatted and consistent. To do so we perform various analytical checks on the data, such as maximum/minimum, sum and count.
2. **Sample Selection:**
   - From the above dataset we create multiple sample datasets, which records the 15-minute, hourly and daily consumption of electricity from 2014 to 2015.
3. **Transform Data:**
   - The two-column sample datasets are transformed into a matrix (8xn) using a Python script. The matrix is then stored into a text file.
4. **Regression Model:**
   - The analysis will use the following regression models:
     - Generalized Linear Model (GLM)
     - Support Vector Machine (SVM)
     - Neural Networks (NN)
     - Random Forest (RF)
   - The mean absolute percentage error (MAPE) is calculated for each regression model. The model with the least value of MAPE is the most optimal model for that sample dataset.
5. **Time Series:**
   - Plot the time series graphs to visually test for stationarity.
   - Statistical analysis to determine the stationarity.
   - Plot ACF and PACF to determine the AR, MA values in the ARIMA.
   - Use Auto.Arima and compare the results against the previous step.
   - Calculate MAPE for all the intervals.
6. **Compare MAPE between various models:**
   - The MAPE of each model is compared between each other.
   - The model with the least value of MAPE is the most optimal model for that sample dataset.

Libraries/Packages Used:

Caret:

Complex classification and regression problems can be streamlined using model training process which are provided by functions in the Classification and Regression Training (i.e. caret) package

- ***Train*:** Train method is a function in Caret package, which helps us to use different mathematical models with configuration parameters. The experiment is performed using glm, svmRadial, avNNet, rf. The most use parameters are mentioned below.

Forecast:

It is a method for analyzing and displaying univariate time series forecasts. Automatic ARIMA modelling and state space models are used for exponential smoothing. (Rdocumentation.org, 2019)

Tseries:

It is used for time series analysis and computational finance. The method provides a general regression equation which incorporates a constant and a linear trend, the t-statistic for a first order autoregressive coefficient equals one is computed. Missing values are not permitted. A warning message is generated if the computed statistic is outside the table of critical values. (Trapletti, 2019)

Regression Models:

The relationship between a dependent (target) and independent variable (s) (predictor) are explored using regression analysis which is a predictive modelling technique. This technique is utilized to find the effective relationship between the variables, for time series modelling and for forecasting. Hence it is an adequate tool for modelling and analyzing data (Ray, 2019). For this assignment we are trying to find the relationship between the electricity consumption for different time intervals and indicates the strength of impact of multiple independent variables on a dependent variable.

Regression analysis is where data points are fit to a line or curve, such that the differences between the distances of the line or curve from data points is minimized.

For building predictive models the regression analysis permits the elimination and evaluation of the best set of variables to be used. It also enables comparison of variables that are measured on different scales

Various regression models are used for the analysis. Many of the models utilize the same variables, a short description of the various models used are as follows (Rdocumentation.org, 2019):

Variables:

Arguments	Description
Data	The main source on which the algorithms are supposed to process on.
Method	The algorithm which is used to create the model. Repeatedcv - A resampling method for repeated training/test splits
preProc	Preprocessing on the data before performing the algorithm's operations on the data. preProc has 2 variable, center and scale. Center = difference between the mean and the actual value. Scale = divides the value with standard deviation.
tuneLength	A parameter which specifies the granularity of optimizing different tuning parameters generated in tuneGrid. Tells the algorithm to try different default values (i.e. 10) for the main parameter
trControl	Train Control – it provides a list of values that define how this function acts.
linout	Linear output units since it is numeric, the default value is logistic output units.
trace	Switch for tracing optimization.
MaxNWts	The maximum allowable number of weights. There is no intrinsic limit in the code, but increasing MaxNWts will probably allow fits that are very slow and time-consuming
Maxit	Maximum number of iterations
ntree	Used for random forest to specify the number of decision trees.

Table 5: Various variables used for different models

K-Fold

"TrainControl" is one of the utility functions available in "caret" which helps to define the rule set on how many folds the algorithm should try partitioning test and train datasets which helps to choose the best model.

Formula

```
myCvControl <- trainControl(method = "repeatedcv",  number = 10, repeats = 5)
```

Variable

Arguments	Value	Description
Method	repeatedcv	A resampling method for repeated training/test splits
Number	10	Number of resampling iterations (i.e. k)
Repeats	5	For repeated k-fold cross-validation, it is the number of complete sets of folds to compute. Note: Neural Network and SVM are posing issues with repeat with value 5 , hence we have considered Repeats = 0

Table 6: Variables used by trainControl function

GLM

Generalized linear model is a classification technique used to fit a linear predictor and generates a description of error distribution.

Formula

```
glmFitTime <- train(V8 ~ ., data = xy, method = "glm", preProc = c("center",
"scale"), tuneLength = 10, trControl = myCvControl)
```

Variable

Arguments	Value
Method	glm
preProc	c(“center” , “scale”)
tuneLength	10

Table 7: Variables used for creating GLM model

SVM

Support Vector Machine are the classification technique which generates hyper planes or simple lines to separate and classify data into different regions.

Formula

```
svmFitTime <- train(V8 ~ ., data = xy, method = "svmRadial", preProc =
c("center", "scale"), tuneLength = 10, trControl = myCvControl)
```

Variable

Arguments	Value
Method	svmRadial
preProc	c(“center” , “scale”)
tuneLength	10

Table 8: Variables used for creating SVM model

Neural Networks

Model Averaged Neural Network (NN) is an imitation of human brain neurons and dendrites where perceptron layers are constructed to create a prediction model on a particular dataset.

Formula

```
nnFitTime <- train(V8 ~ ., data = xy, method = "avNNet", preProc = c("center",
"scale"), trControl = myCvControl, tuneLength = 10,  linout = T,  trace = F,
MaxNWts = 10 * (ncol(xy) + 1) + 10 + 1, maxit = 500)
```

Variable

Arguments	Value
Method	avNNet
preProc	c(“center” , “scale”)
tuneLength	10
linout	T
trace	F
MaxNWts	10 * (ncol(xy) + 1) + 10 + 1
Maxit	500

Table 9: Variables used for creating NN model

Random Forest

Random Forest (RF) is a classification method which is a cluster of multiple decision trees.

Formula

```
rfFitTime <- train(V8 ~ ., data = xy, method = "rf",  ntree = 500, preProc =
c("center", "scale"), trControl = myCvControl)
```

Variable

Arguments	Value
Method	rf
ntree	500
preProc	c(“center” , “scale”)
tuneLength	10

Table 10: Variables used for creating RF model

For each of the above models, the MAPE value is calculated and compared in more details in coming sections.

Insights/Observations Drawn:

1. It take a lot of time to build the SVM, NN, RF regression models.
2. K-fold implementation with repeats consumes a lot of time and is hungry for process time.
3. tuneLength optimizes every parameter in tuneGrid. As the value of tuneLength increases, process time increases.

Time Series:

To analyze the 15-minute interval, hourly and daily electricity consumption to observe/identify trend, seasonality and any other interesting insights so that we could forecast the consumption for the next 7 data points.

ARIMA model is used to forecast the datapoints in the future. The parameters used in the ARIMA function are (AR, I, MA) also known as (p, d, q). ACF and PACF graphs are used to understand and configure the values for AR, MA.

AR (p) - This is a linear model that relies on the previous data points, with coefficients for each level. As name suggests, this is regressive model where the prediction happens using the previous points.

MA (q) - This is a moving average model which calculates the dependency between the current point and the error from the moving average applied on the lagged data points (previous data points).

Converting the vectors to timeseries and choosing the frequency:

The given data set has the cycles in years and as the intervals being tested are daily, hourly and 15 minutes, frequency in the timeseries function is chosen as per the below table.

	Frequencies				
Data	**Minute**	**Hour**	**Day**	**Week**	**Year**
Daily				7	365.25
Hourly			24	168	8766
Half-hourly			48	336	17532
Minutes		60	1440	10080	525960
Seconds	60	3600	86400	604800	31557600

Table 11: Choosing the right frequency for the time-series

Plotting the time series:

The below graphs are the plots of the consumption of power from 2011 to 2015 for different data set like daily, hourly and 15-minutes. The visual representation of the plots helps us to identify the trend of consumption of power.

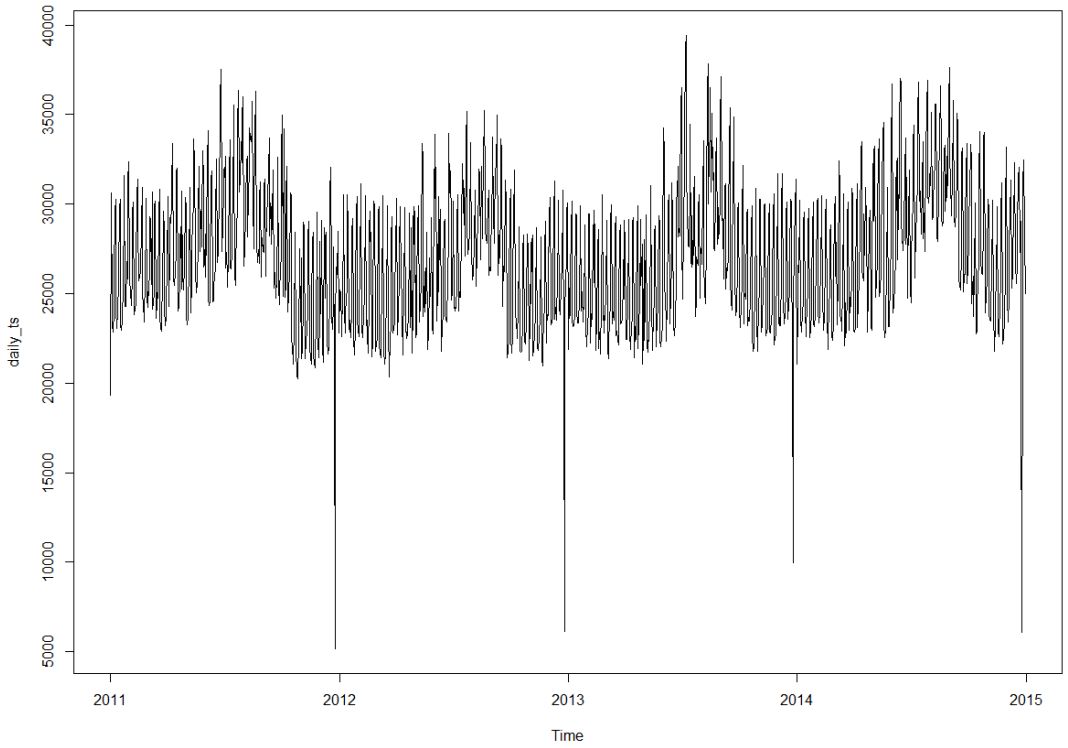

Figure 1: Daily power consumption

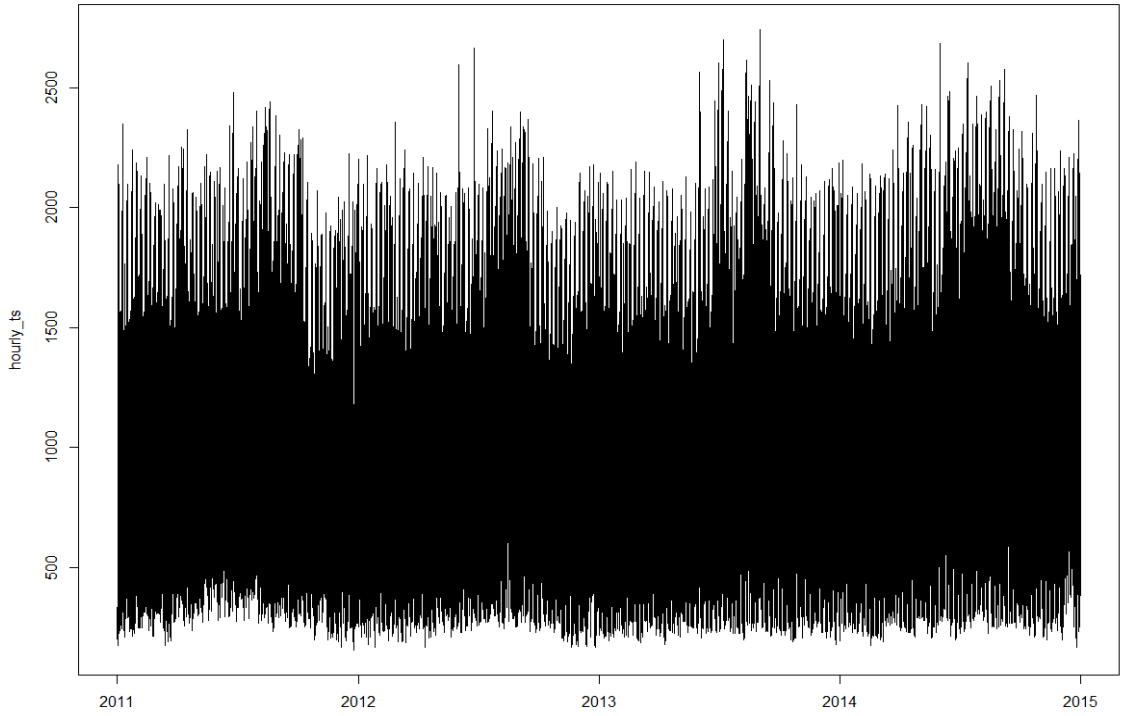

Figure 2: Hourly power consumption

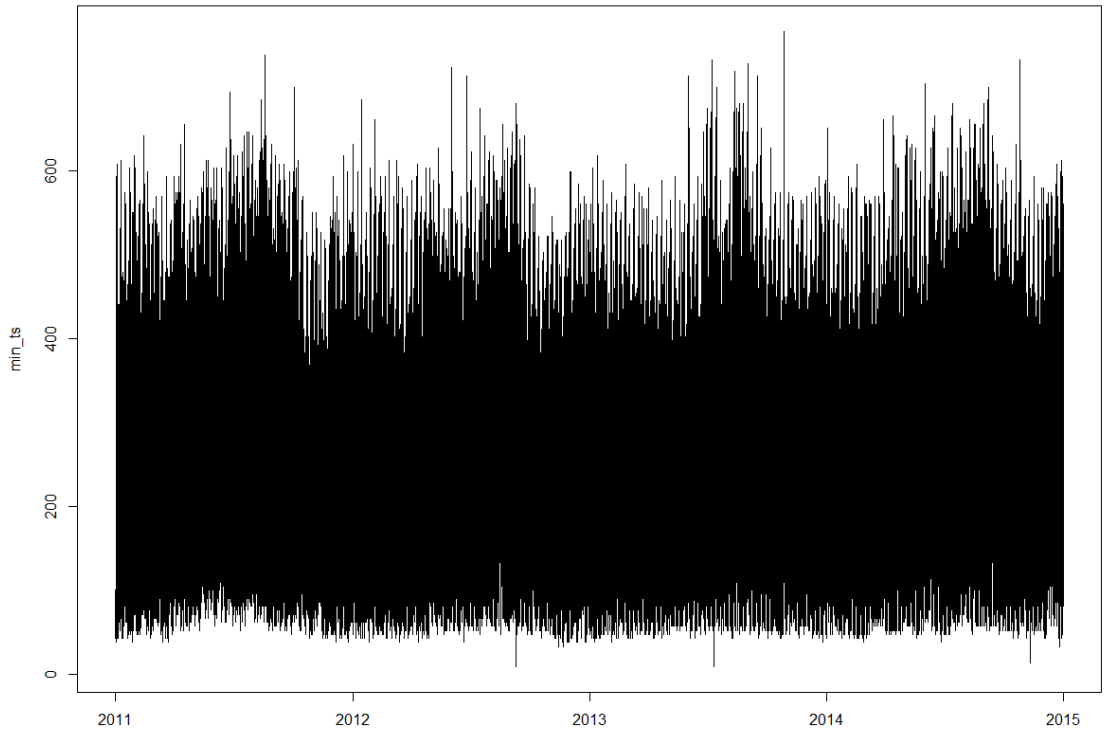

Figure 3: 15-minute interval power consumption

Test for Stationarity:

We have used Dickey-Fuller test to test the stationarity. The null hypothesis is used to test whether the present data set is stationary or not stationary and the test states that the time series is stationary.

Results:

```
        Augmented Dickey-Fuller Test

data:  daily_ts
Dickey-Fuller = -5.3831, Lag order = 11, p-value = 0.01
alternative hypothesis: stationary
```

Figure 4: Daily result after hypothesis test.

```
        Augmented Dickey-Fuller Test

data:  hourly_ts
Dickey-Fuller = -18.518, Lag order = 32, p-value = 0.01
alternative hypothesis: stationary
```

Figure 5: Hourly result after hypothesis test.

```
        Augmented Dickey-Fuller Test

data:  min_ts
Dickey-Fuller = -72.89, Lag order = 51, p-value = 0.01
alternative hypothesis: stationary
```

Figure 6: 15-minute result after hypothesis test.

From the results above, we have significant results for all the intervals

Plotting ACF and PACF to select the p, q values in ARIMA model:

ACF can be used to estimate the MA(q) and PACF can be used to estimate the AR(p).

Auto Correlation:

This graph is used to determine the value of 'p' in ARIMA (**p** ,d, q). The graph depicts an interval of spikes. The number of continuous spikes is used to determine the value of p. Below example demonstrate the p value for daily, hourly, 15-minute datasets.

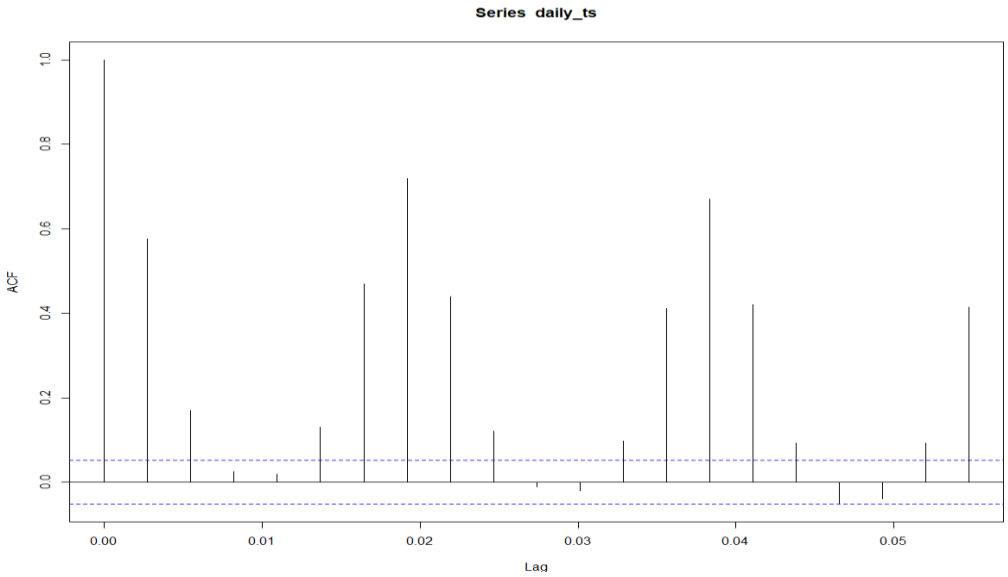

Figure 7: ACF for daily energy consumption

Note: We could observe 5 significant values until the effect dips down below the significant results and hence we will choose a value close to 5 for MA i.e. (q).

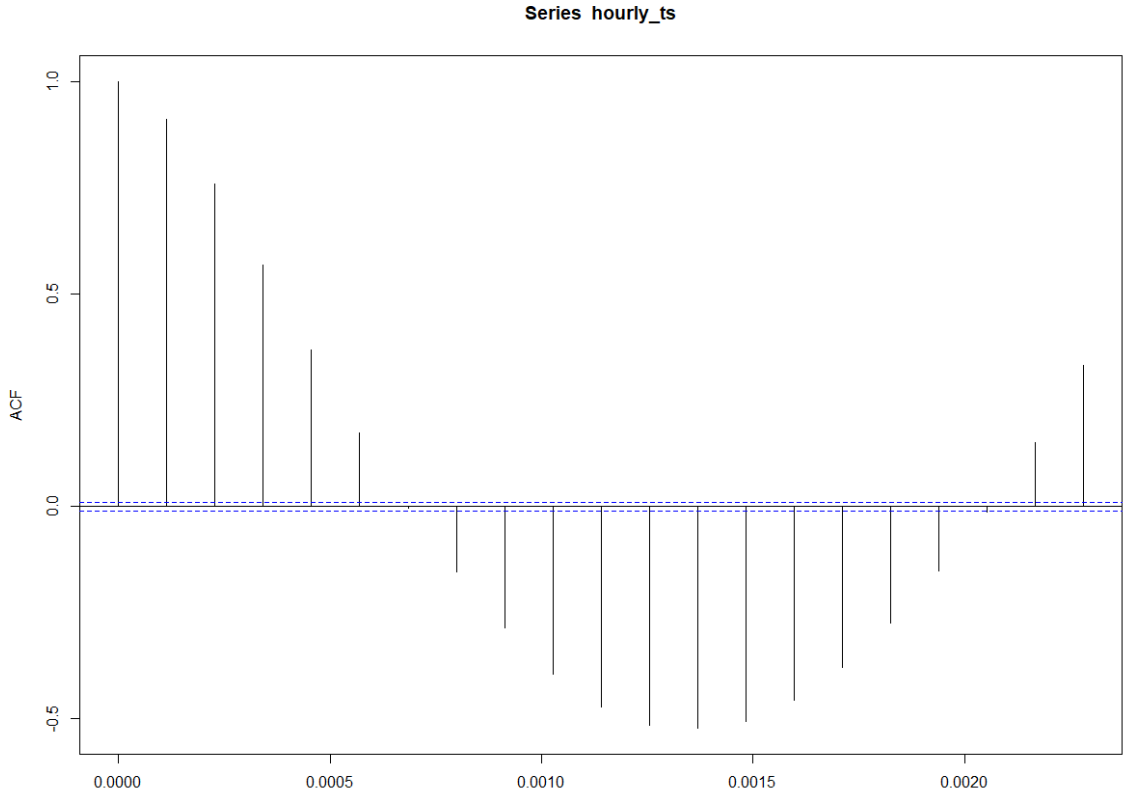

Figure 8: ACF for hourly power consumption

Note: We clearly observe 5-7 significant values for the ACF plot for the hourly data.

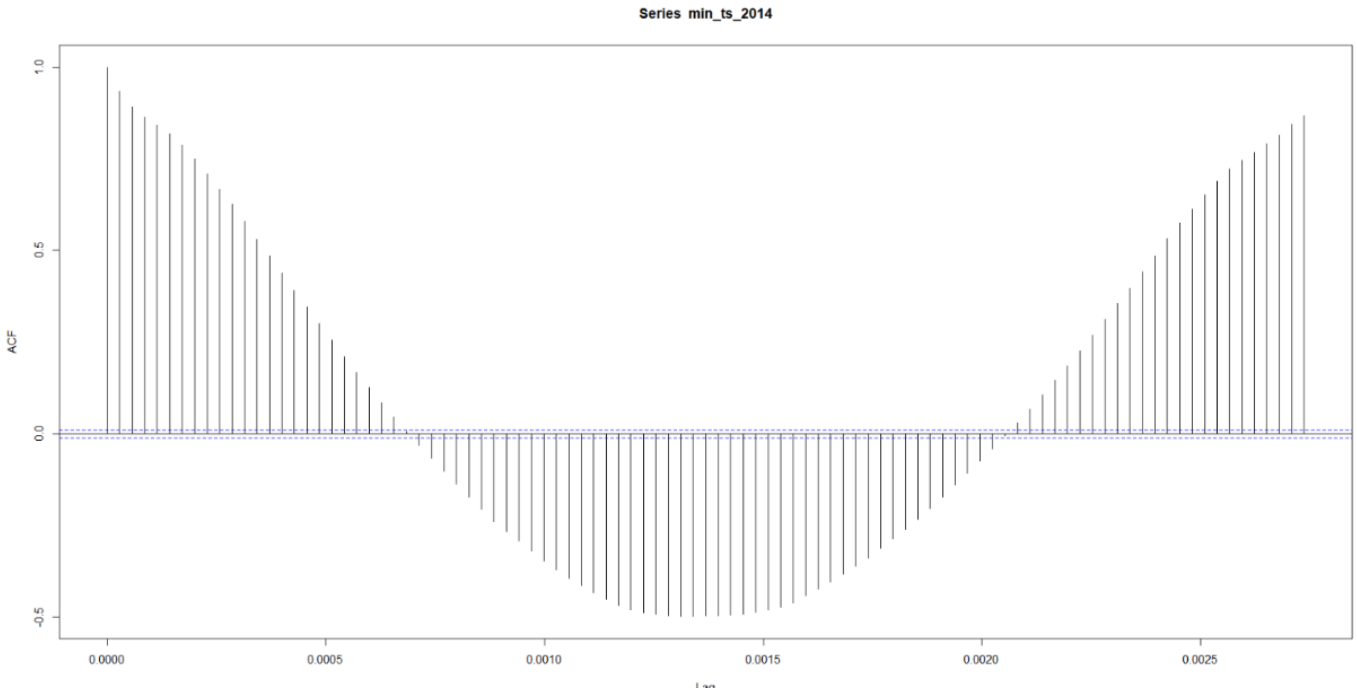

Figure 9: ACF for 15 minute interval power consumption

Note: We clearly observe 25 – 40 significant values for the ACF plot for the hourly data.

Partial Auto Correlation :

This graph is used to determine the value of 'q' in ARIMA (p**,** d, **q**). The graph depicts an interval of spikes. The number of continuous spikes is used to determine the value of q. Below example demonstrate the q value for daily, hourly, 15-minute datasets. This is a moving average model which calculates the dependency between the current point and the error from the moving average applied on the lagged data points (previous data points).

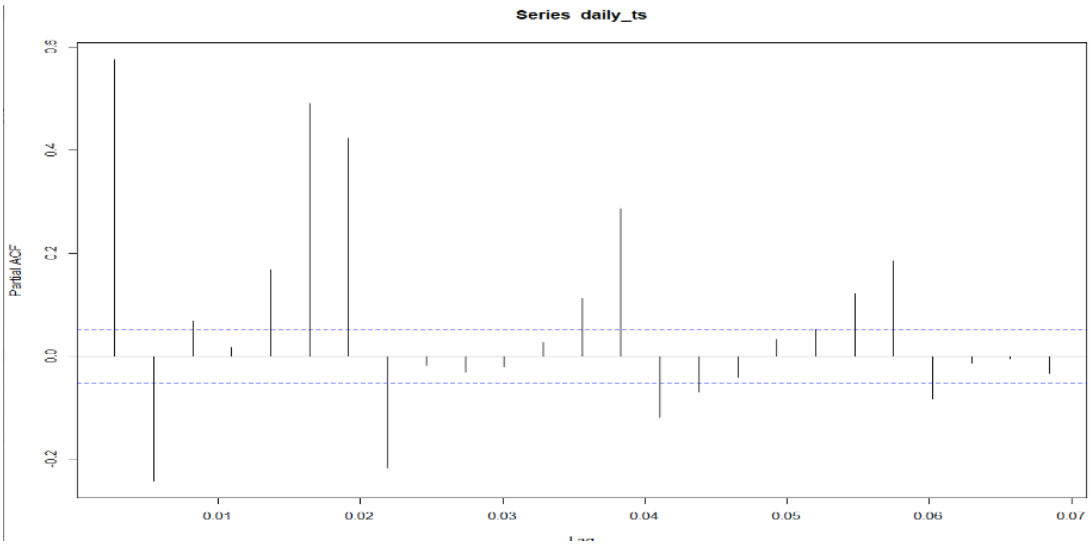

Figure 10: PACF plot for daily power consumption

Note: We observe 2-3 significant results for the PACF for Daily intervals.

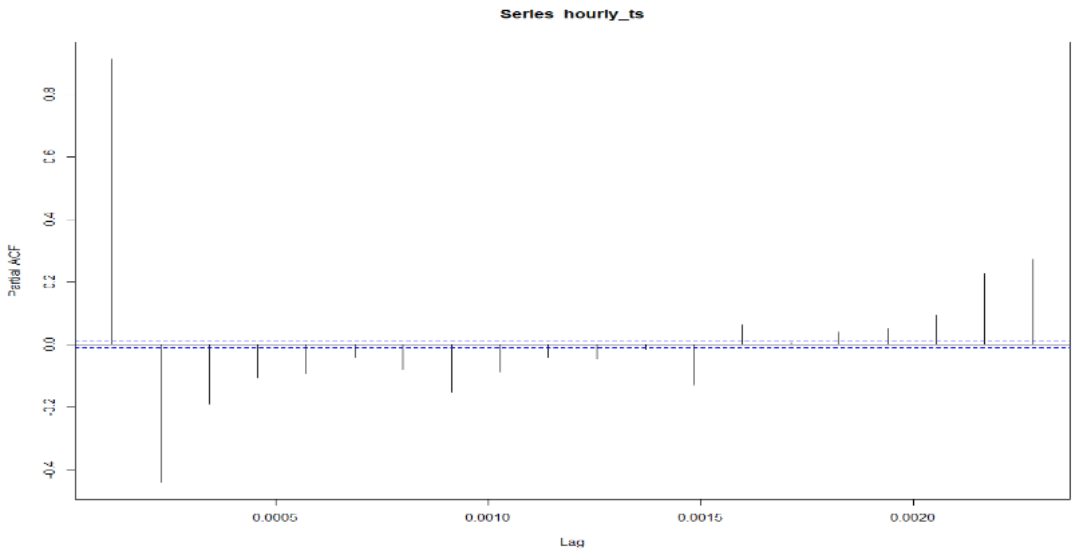

Figure 11: PACF for hourly energy consumption

Note: We observe 10 significant results for the PACF for Hourly intervals.

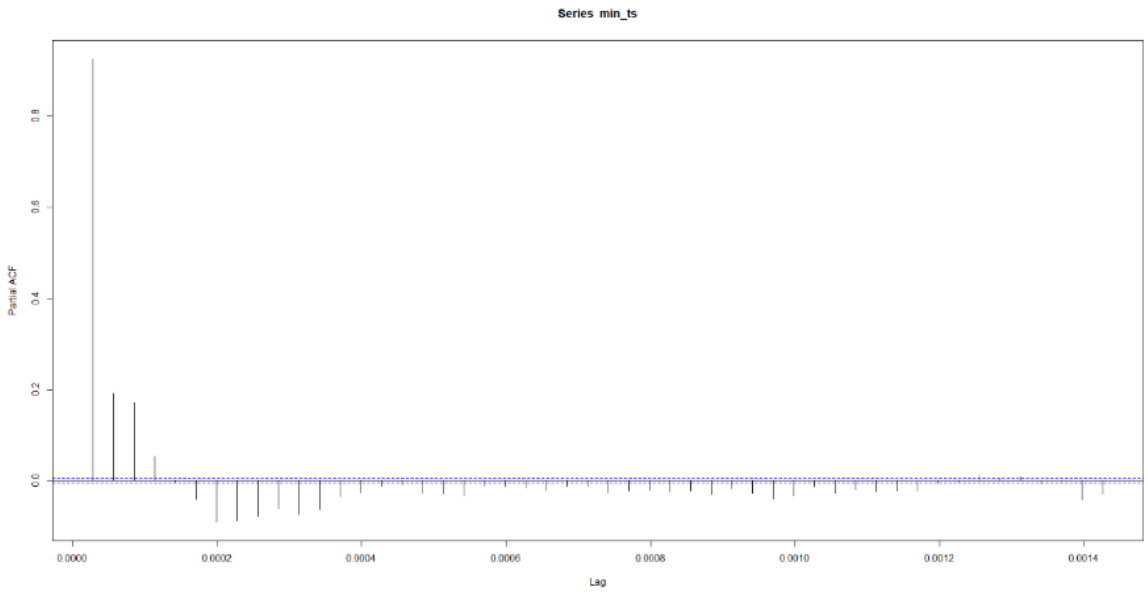

Figure 12: PACF for 15 minute interval power consumption

Note: We observe ~20 significant results for the PACF before the points sunk below the significant mark for 15-minute intervals.

Decomposition of time-series:

The ‘stl’ decompose function is used to divide the time series into data, seasonal, trend and remainder(error).

Seasonality: This is defined as the pattern of occurrences that repeats over a period within the cycles of data that we take.

Trend: Trend defines the increase, decrease or the stagnant level of the data over a period.

Remainder: This is the error that could not be explained.

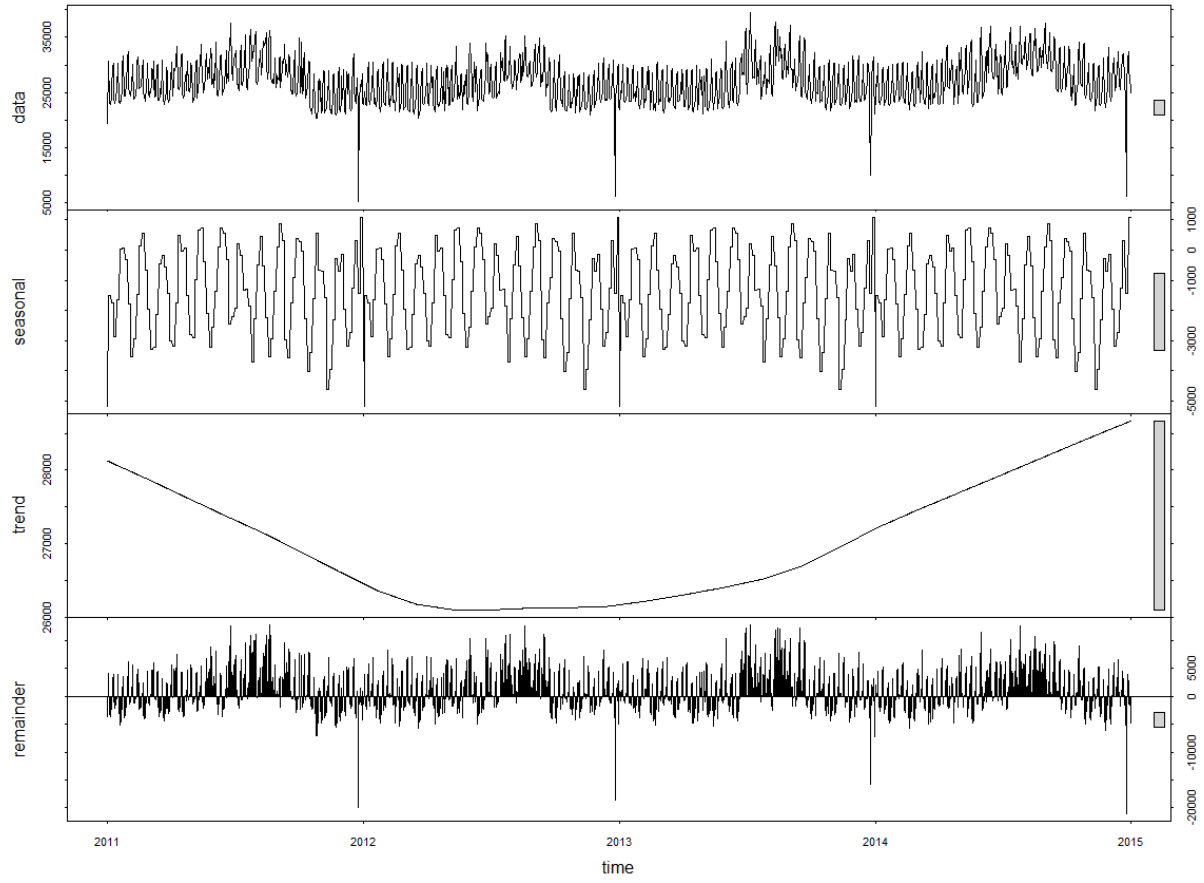

Figure 13: Decomposing the daily energy consumption

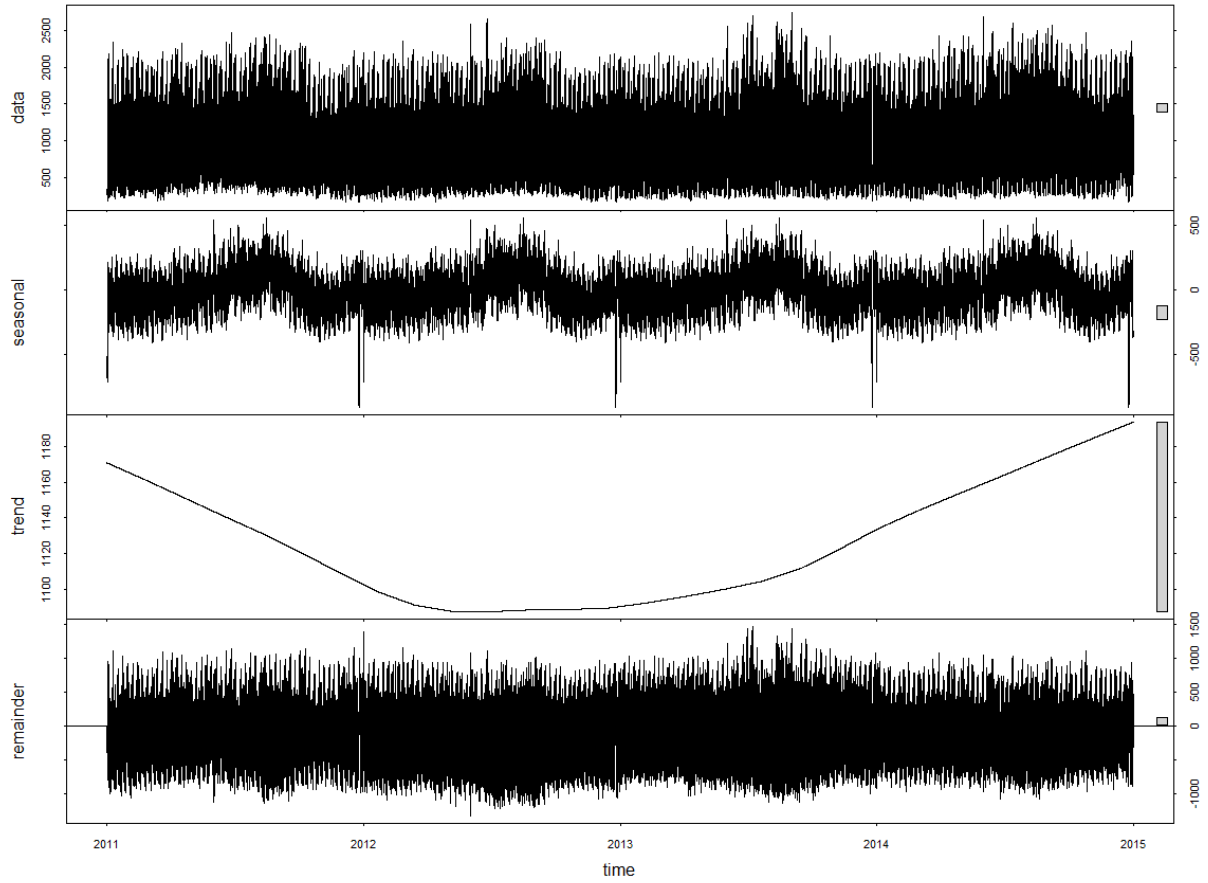

Figure 14: Decomposing the hourly power consumption

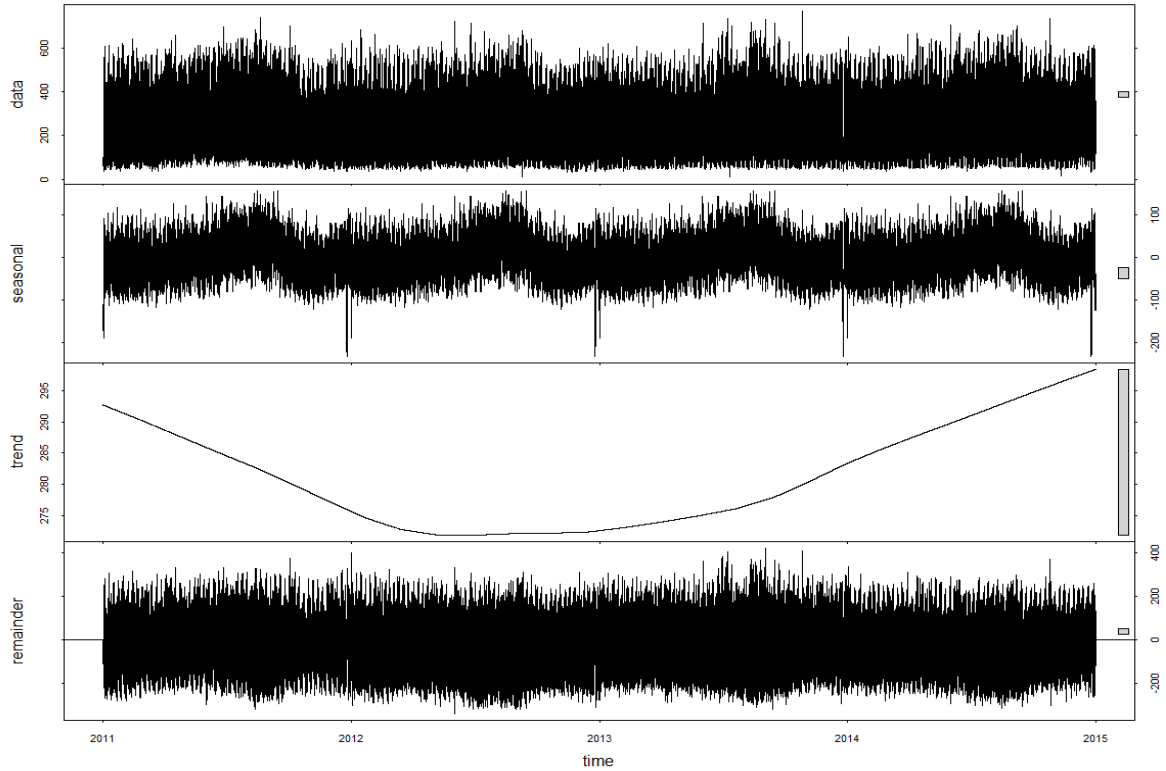

Figure 15: Decomposing the 15-minute interval power consumption

Insights Drawn:

1. The given data is stationary in all the intervals (daily, hourly, 15-minute).
2. Mean and Median are close.
3. There is not much variance present in the data throughout the period.
4. The trend is hovering around the mean.
5. Seasonality is observed.

MAPE:

The following are the MAPE values attained for the specific intervals for Auto Arima and with the optimal values for Arima:

Interval	**Auto.Arima(5,1,0)**	**Arima (7,1,7)**
Daily	7.64	5.13

Interval	**Auto.Arima(2,1,2)**	**Arima (10,1,11)**
Hourly	13.9	12.7

Interval	**Auto.Arima(3,1,2)**	**Arima (3,1,10)**
15 Minute	12.10	12.23

Comparison of MAPE

The Mean Absolute Percent Error (MAPE) calculates the size of the error in a percentage form.

Formula:

$$\text{MAPE} = \frac{\sum \frac{|A-F|}{A} \times 100}{N}$$

From the formula:

- A= Actual,
- F= Forecast,
- N= Number of observations,
- The vertical bars stand for absolute values.

15 Minutes Data Summary:

```
Call:
summary.resamples(object = resamps)

Models: lm, svm, nn, rf
Number of resamples: 10

MAE
        Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
lm  29.31101 30.53869 31.04940 31.53766 33.04436 33.65261    0
svm 26.25715 28.16281 28.69736 28.91605 29.75718 31.25275    0
nn  29.01760 29.31327 29.49511 29.63542 29.82464 30.88552    0
rf  27.63461 28.11291 28.65402 28.53761 28.98464 29.27957    0

RMSE
        Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
lm  39.56884 39.78019 40.48582 41.26160 42.75393 44.91197    0
svm 35.87297 37.46084 38.22594 38.43904 39.53533 41.71858    0
nn  37.66160 38.28089 38.83616 38.97561 39.29813 41.49810    0
rf  36.78316 37.16323 37.56929 38.01369 38.75914 39.79182    0

Rsquared
         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
lm  0.8809403 0.8944746 0.9051385 0.9010927 0.9100133 0.9119399    0
svm 0.9001791 0.9090643 0.9159507 0.9144298 0.9194532 0.9273490    0
nn  0.8997384 0.9064820 0.9152532 0.9119750 0.9176018 0.9185352    0
rf  0.9065714 0.9120411 0.9160971 0.9162221 0.9211883 0.9252161    0
```

Hourly Data Summary:

```
Call:
summary.resamples(object = resamps)

Models: lm, svm, nn, rf
Number of resamples: 10

MAE
        Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
lm  70.48811 79.99161 82.19831 82.25946 85.47407 90.15975    0
svm 66.43066 67.95866 71.78365 72.37924 75.32983 81.29567    0
nn  72.45003 74.47647 76.65886 75.93793 76.95908 78.86867    0
rf  65.53894 68.36086 71.64161 70.10207 71.97204 72.59501    0

RMSE
        Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
lm  90.32056 103.20509 106.14043 105.33985 108.23704 114.2618    0
svm 81.40964  91.50284  95.62893  95.69603 100.77450 110.4319    0
nn  85.06658  94.56180  96.04737  96.22881  97.39752 105.5243    0
rf  87.55603  88.86972  89.75676  91.54810  91.67456 105.5635    0

Rsquared
         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
lm  0.9357795 0.9420868 0.9456420 0.9464152 0.9494286 0.9607255    0
svm 0.9441896 0.9519524 0.9563393 0.9559581 0.9596649 0.9686766    0
nn  0.9469504 0.9533457 0.9549283 0.9550599 0.9562767 0.9659498    0
rf  0.9422472 0.9583614 0.9608334 0.9590520 0.9626930 0.9637123    0
```

Daily Data Summary:

```
Call:
summary.resamples(object = resamps)

Models: lm, svn, nn, rf
Number of resamples: 50

MAE
        Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
lm  502.2537 1264.070 1653.077 1704.546 2015.695 3546.679    0
svn 189.1134 1009.083 1317.297 1433.512 1843.559 3049.431    0
nn  645.9683 1270.194 1698.868 1844.724 2282.328 3787.481    0
rf  500.6255 1101.012 1521.493 1569.952 1975.657 3147.781    0

RMSE
        Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
lm  864.0008 1515.497 2035.942 2080.517 2591.075 3953.703    0
svn 269.8275 1215.767 1668.381 1788.520 2093.700 3685.390    0
nn  917.0675 1492.823 2126.400 2212.137 2870.801 4208.718    0
rf  568.1371 1206.577 1894.464 1927.712 2710.589 3714.566    0

Rsquared
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
lm  0.27657771 0.6112917 0.7974140 0.7360869 0.8716476 0.9933970    0
svn 0.49450547 0.7559384 0.8156542 0.8127872 0.9233037 0.9989533    0
nn  0.14809486 0.6227261 0.7825911 0.7308069 0.8664148 0.9695734    0
rf  0.05423925 0.6786136 0.8159728 0.7656468 0.9033542 0.9846665    0
```

	MAPE		
Model	**15-minutes**	**Hourly**	**Daily**
GLM	13.2005	10.42207	5.047716
SVM	10.63101	7.215507	3.229672
NN	11.8904	9.050787	1.760021
RF	5.304776	3.650955	2.363314
ARIMA	12.23	12.7	5.13
AUTO ARIMA	12.10	13.9	7.64

Table 12: Comparing MAPE for different Models for 15-min, Hourly and Daily Datasets

From the above table it is evident that the Random Forest Model (RF) is the most optimal model for the 15-minutes and Hourly dataset, since its equivalent MAPE value is the least compared the other models. Similarly, the Neural Network Model (NN) is the most optimal model for the Daily dataset, since its equivalent MAPE value is the least compared the other models.

Conclusion:

From the analysis on the different datasets i.e., 15 minutes , hourly, daily , using different algorithms like RF, SVM, Neural networks, linear regression, it is evident that 'rf' fits the curve well for timeseries , where the MAPE value is minimal when compared to the rest of the models. The MAE, RMSE, R-squared has given the summary which can be studied and concluded that 'rf' has least error rate when compared to rest of the models. The time series model has been useful to understand the consumption of the power along the timeline as is able to predict the consumption. ARIMA implementation has the error rate and is very much comparable with the rest of the models. The fitting of the curve for ARIMA depends on p,d,q parameters which after analyzing auto correlation and partial auto correlation were able to identify and tried on with different values, which in turn helped to reduce the error rate. This experimentation demonstrates the implementation of different classification regression models and time series to model the prediction of the event of consumption of electricity.

References:

Trapletti, A. (2019). *Package 'tseries'*. [online] Cran.r-project.org. Available at: https://cran.rproject.org/web/packages/tseries/tseries.pdf [Accessed 26 Jul. 2019].

Ray, S. (2019). *7 Regression Types and Techniques in Data Science*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/ [Accessed 26 Jul. 2019].

Rdocumentation.org. (2019). *R Documentation and manuals | R Documentation*. [online] Available at: https://www.rdocumentation.org [Accessed 26 Jul. 2019].

Appendix:

Appendix A: Database Queries:

```
use dataset04;

SELECT `Value` FROM `MT123electricity` WHERE `RecordDateTime` BETWEEN "2014-01-01" AND
"2015-01-01";

SELECT SUM(`Value`) FROM `MT123electricity` WHERE `RecordDateTime` BETWEEN "2014-01-01"
AND "2015-01-01" GROUP BY CAST(`RecordDateTime` as date), EXTRACT(HOUR FROM
`RecordDateTime`);

SELECT SUM(`Value`) FROM `MT123electricity` WHERE `RecordDateTime` BETWEEN "2014-01-01"
AND "2015-01-01" GROUP BY CAST(`RecordDateTime` as date), EXTRACT(DAY FROM
`RecordDateTime`);
```

Appendix B: Python Code:

```
#convert column to matrix
import re
def columnarToMatrix(input,output):
      f = open(input,"r")
      w = open(output,"w")
      s = ""
      columns = 8
      count = 0
      for x in f:
            s = s + x.rstrip()
            count = count + 1
            if(count >= columns):
                  #append the content into new line
                  count = 0
                  w.write("\n")
                  s = ""
            else:
                  s = s + " "
      w.close()
      f.close()

columnarToMatrix("15min.csv","15minutesmatrix.txt")
columnarToMatrix("day.csv","daymatrix.txt")
columnarToMatrix("hourly.csv","hourlymatrix.txt")
```

Appendix C: R Code:

```
library(caret)
setwd("F:/SMU2/Data mining/MLAssignment4/")
getwd()
# xy needs to be tried for 3 data sets.
# 1. 15minutesmatrix.txt
# 2. daymatrix.txt
# 3. hourlymatrix.txt
xy=read.table("15minutesmatrix.txt",sep=' ',header=F)
y=xy[,8]
head(y)
x=xy[,1:7]

# Using pre-sliced data
# method = the resampling method is repeatedcv.
# K-fold , dividing the training set into 10 parts.
# number = is number of folds or number of resampling iterations.
# repeats= for repeated k-fold cross-validation only: the number of complete sets of
folds to compute.

myCvControl <- trainControl(method = "repeatedcv", number = 10)
# myCvControl <- trainControl(method = "repeatedcv", number = 10,repeats = 5)

# Linear regression
# glm (Generalized linear model)
# Cross-Validated (10 fold)
```

```
# RMSE = Root mean square deviation or Root mean square error
# Math.sqrt(Sigma[0 - n](values predicted - values observed)^2)
# MAE = computes the average absolute difference between two numeric vectors

glmFitTime <- train(V8 ~ .,
                    data = xy,
                    method = "glm",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
print(glmFitTime)
summary(glmFitTime)
y_hat = predict(glmFitTime, newdata = x)
summary(y_hat)
mean(100*abs(y_hat-y)/y)
# Your error with linear regression

# Support Vector Machine Radial
svmFitTime <- train(V8 ~ .,
                    data = xy,
                    method = "svmRadial",
                    preProc = c("center", "scale"),
                    #tuneLength = 10,
                    trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = x)
mean(100*abs(y_hat-y)/y)
# Your error with support vector regression

# Neural Network
nnFitTime <- train(V8 ~ .,
                   data = xy,
                   method = "avNNet",
                   preProc = c("center", "scale"),
                   trControl = myCvControl,
                   #tuneLength = 10,
                   linout = T,
                   trace = F,
                   MaxNWts = 10 * (ncol(xy) + 1) + 10 + 1,
                   maxit = 500)
nnFitTime
summary(nnFitTime)
y_hat = predict(nnFitTime, newdata = x)
mean(100*abs(y_hat-y)/y)
# Your error with neural networks

# random forest
randomforest <- train(V8 ~ ., data = xy,method = "rf",ntree = 500,trControl =
myCvControl
)
randomforest
```

```
summary(randomforest)
y_hat = predict(randomforest, newdata = x)
mean(100*abs(y_hat-y)/y)

# You can experiment with other methods, here is where you can find the methods caret
supports:
# https://topepo.github.io/caret/available-models.html

# Compare models
resamps <- resamples(list(lm = glmFitTime,
                          svn = svmFitTime,
                          nn = nnFitTime,
                          rf = randomforest))
summary(resamps)

# Now working with the time-series modeling
# t needs to be tried for 3 data sets.
# 1. 15minutes.csv
# 2. day.csv
# 3. hourly.csv
#install.packages("forecast")
#install.packages("ggplot2")
library(forecast)
library(ggplot2)
t= read.csv("15minutes.csv",header=FALSE)
head(t)
tSeries = ts(t,freq=4)
head(tSeries)
plot.ts(tSeries)
time(tSeries)
quantile(tSeries)
plot(decompose(tSeries))
plot(diff(tSeries))
ggseasonplot(tSeries)
adf.test(tSeries)
acf(tSeries, lag.max = 20)

#install.packages("forecast")
library(forecast)

ar <- Arima(tSeries,order=c(1,0,2))
mean(100*abs(fitted(ar) - tSeries)/tSeries)#13.50665
# Your Arima error

res = auto.arima(tSeries, stepwise = F, approximation = F)
res
plot(forecast(res, h= 3))
```