Forecasting Electricity Demand in Northern Maine

Team Colborne

April 2, 2017

Executive summary

- Team Colborne was approached by Algonquin Power and Utilities Company to modify their load forecasting technique at the request of the Northern Maine Independent System Administrator.
- The Team built a fully functional time series model in the R programming language that incorporates weekly and daily variations in load that reflect the nature of their industrial clients in the Maine Public Service Region.
- Additionally, the model considers regionally observed holidays as well as local temperature conditions.
- The model automatically reads in demand data from the template provided by the settlements team and converts it to useable form. Next, it accounts for missing observations in the temperature data from using a multiple imputation. Lastly, using established time-series relationships in the underlying data sets, it creates an hourly demand profile for the next seven days based on upcoming holidays and temperature data.
- The next step is to develop a testing regime for the model so that it can be further developed and tested against the current load forecasting technique.

1. Problem Statement

Algonquin Power and Utilities Corporation (APUC) owns and operates a 34 MW run-of-the-river hydroelectric facility in the Northern Maine Independent System Administrator (NMISA) power market. The dam was built in 1923 and is located just east of the US-Canada border in Tinker, New Brunswick. The company uses the power generated from the power plant to serve electric load for industrial clients in the Maine Public Service (MPS) region. The generation from the plant fluctuates with the flow in the Aroostook River, and, coupled with the variable nature of electricity demand, creates challenges for the company. When river flows are high and the available generation exceeds the electric load, the company must export the additional generation to external markets. Similarly, when river flows are low, the company must import power to serve their load obligations. Historically, load forecasting at the company has been accomplished using a simple averaging technique for similar days, but with increasing stress on the grid caused by intermittent energy sources, the System Operator has mandated that the forecasting technique capture hourly fluctuations in supply and demand. Therefore, the purpose of this project is to develop and propose an updated load forecasting methodology for the industrial clients in the Maine Public Service region that meets the requirements of the System Administrator.

The report is separated into three sections: Data Sources, Analyisis, and Results. Throughout these sections, the team has highlighted five distinct tasks that were completed, namely:

1. Cleaning and Merging the Data
2. Dealing with Missing Data
3. Automated Model Development
4. Dealing with Error Terms
5. Forecasting Time Series Data

The findings for each of these tasks are presented from both managerial and technical perspectives so to provide the reader with a wholesome understanding of our results. While the managerial perspective speaks generally about the resulting outcomes of each task, the report emphasizes the technical perspective due to the consideration that the resulting model is likely to transition to a testing environment in the coming weeks. The intent is that the executive summary will be used primarily for presentation to the APUC management team, while the report will focus on the technical and practical considerations of deploying the model in a test environment.

2. Data Sources

To develop this model, the team sourced three data sources:

1. Hourly Electric Demand
2. Hourly Regional Temperature
3. Observed Regional Holidays

Hourly Electric Demand Data

Algonquin Power supplied the team with hourly electric demand data for each month of the year dating back to 2012. Subsequent discussions with the company determined that a significant number of new industrial clients were onboarded throughout 2015, and as a result, requested that the team use only the data for the 2016 period. Each monthly data file reports the hourly electric demand in MWh in the following format:

NORTH ACTUALS - MPS

HI/LO Actual

Forecast

HE / Date	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11	1/12	1/13	1/14	1/15	1/16	1/17	1/18	1/19	1/20	1/21	1/22	1/23	1/24	1/25	1/26	1/27	1/28	1/29	1/30	1/31
1	9.4	9.1	9.2	9.2	11.0	11.0	10.7	10.5	10.3	10.8	9.6	11.6	11.6	12.7	11.0	10.1	10.1	9.9	11.5	11.0	11.7	11.2	11.1	10.8	10.9	11.4	9.7	11.2	10.6	9.2	9.7
2	9.6	9.2	9.4	9.5	11.2	11.8	11.7	11.4	11.3	10.1	9.8	12.3	12.2	13.5	12.5	10.2	10.2	10.2	12.6	12.1	12.4	12.0	11.4	10.8	11.1	12.4	9.8	12.2	11.3	9.6	9.9
3	9.8	9.1	9.3	9.8	8.1	11.8	11.9	11.7	11.6	10.5	9.9	12.5	12.6	13.8	12.4	9.7	10.0	10.5	12.7	12.5	12.9	12.3	11.6	11.4	11.4	12.2	10.2	12.4	11.4	10.3	10.2
4	9.5	9.1	9.6	9.8	8.2	11.9	11.6	11.3	11.5	10.8	9.9	12.7	12.5	13.9	12.4	9.9	9.8	10.8	12.7	12.6	12.4	12.3	11.7	11.8	11.7	12.3	10.3	12.2	11.2	10.5	10.3
5	9.4	9.3	9.9	10.5	8.8	12.5	11.8	12.4	11.7	11.2	10.5	13.4	13.3	14.5	13.3	10.2	10.1	11.6	13.5	13.3	13.5	13.1	12.2	12.0	12.4	13.3	11.4	12.7	11.9	10.9	10.5
6	10.2	9.7	10.3	12.4	9.6	13.5	13.0	12.8	11.2	10.9	13.4	13.7	13.8	13.7	14.3	11.8	11.2	13.8	14.4	14.2	14.4	13.9	12.4	12.8	15.2	14.1	13.4	13.2	13.2	11.4	11.2
7	11.2	10.4	10.8	15.7	12.4	16.0	16.3	14.9	11.7	10.9	16.4	17.1	16.5	16.3	15.6	12.4	11.5	16.9	17.8	17.4	17.8	14.8	13.2	13.1	17.8	17.1	16.6	14.6	15.1	12.0	11.7
8	11.5	10.3	11.1	16.0	13.0	16.1	16.4	15.0	12.0	11.1	17.5	16.7	17.0	16.9	15.9	12.7	11.1	17.5	17.8	18.0	17.7	15.5	13.4	13.5	18.4	17.4	17.5	15.0	15.4	12.3	11.8
9	11.7	11.2	11.1	17.0	13.5	16.4	16.8	15.7	12.4	11.6	18.1	17.3	17.4	17.7	16.6	13.0	11.3	17.7	18.4	18.7	17.6	15.6	13.3	13.7	20.1	18.1	17.8	15.4	16.1	12.7	12.1
10	12.5	11.3	11.3	16.8	13.4	15.8	16.5	15.8	12.3	11.7	18.3	17.0	17.1	16.9	16.8	13.2	11.7	18.2	18.5	18.3	17.4	15.9	13.7	13.7	20.2	17.7	17.8	14.6	16.0	12.7	12.0
11	11.7	11.7	11.3	16.0	12.6	16.0	16.0	14.8	12.7	11.6	17.4	16.8	16.7	13.8	15.4	12.9	11.9	17.8	17.9	18.5	16.8	15.0	13.6	13.9	19.2	17.3	17.3	14.1	15.0	12.4	12.0
12	11.3	11.0	10.5	15.3	11.3	14.5	14.6	13.9	11.8	11.1	16.0	15.4	15.3	12.6	14.5	12.0	11.2	16.0	16.3	16.6	15.3	13.6	12.8	13.0	17.8	15.8	15.8	13.0	13.7	11.7	11.5
13	11.3	10.4	10.0	15.6	11.8	15.3	15.3	13.6	11.1	10.3	16.6	16.5	16.1	12.7	14.2	11.4	11.0	16.9	16.8	16.7	16.1	12.9	11.9	12.0	18.7	16.3	15.6	13.2	13.9	11.0	10.9
14	11.5	10.4	10.3	16.1	12.2	16.5	15.5	14.3	11.3	10.8	17.5	17.2	16.8	14.0	14.5	11.6	11.2	17.6	17.2	17.2	16.8	14.8	12.3	12.4	19.7	16.9	16.4	13.3	13.9	11.1	11.0
15	11.6	10.7	10.3	15.6	12.5	16.7	15.2	14.1	11.3	10.8	17.0	16.9	16.1	13.6	14.0	12.0	11.3	17.5	17.1	17.5	16.4	15.3	12.4	12.2	19.8	16.2	16.4	13.9	14.0	11.4	11.2
16	11.8	11.1	10.4	15.9	12.6	16.2	15.6	13.8	11.4	11.5	16.8	16.8	16.4	14.1	14.3	12.3	11.4	17.0	17.2	17.1	16.4	14.9	12.6	12.8	17.6	16.7	16.2	14.2	13.9	11.7	11.5
17	12.2	11.5	11.3	16.3	12.5	16.4	15.5	14.6	12.2	11.8	16.5	16.9	17.4	14.7	14.6	12.7	12.0	17.1	17.1	16.4	16.1	14.6	13.1	13.2	16.8	16.2	15.9	14.4	14.0	12.1	11.5
18	11.9	11.1	10.9	13.6	9.8	13.5	12.9	12.6	11.3	11.4	14.6	14.3	15.6	13.3	14.3	12.2	12.1	14.2	15.1	14.1	14.6	14.3	12.7	12.9	14.3	14.4	14.5	13.6	13.9	11.9	11.7
19	11.5	10.3	10.4	14.3	10.5	14.0	13.3	12.8	10.3	10.7	15.3	15.0	15.6	14.1	13.6	11.4	11.4	15.6	15.8	15.3	15.4	14.5	12.2	12.3	15.3	14.4	15.2	14.3	13.4	10.9	11.0
20	11.2	10.3	10.5	14.0	10.0	13.7	13.6	13.3	10.3	10.6	14.4	14.7	15.7	13.7	13.2	11.2	11.4	15.0	15.5	15.1	15.0	14.0	12.1	12.3	15.1	13.4	14.9	14.2	12.7	11.0	10.8
21	10.3	10.1	10.2	12.6	9.0	12.9	12.5	12.3	10.0	10.0	13.5	13.5	15.1	13.1	12.4	11.0	10.9	14.0	14.5	14.3	13.8	13.1	11.8	12.2	14.0	12.6	13.9	13.1	12.0	10.6	10.8
22	9.5	9.5	9.6	11.8	7.6	11.4	11.5	11.5	9.2	9.6	12.4	12.5	13.6	11.8	11.2	10.2	10.0	12.7	12.9	13.1	12.6	12.2	11.1	11.2	12.8	11.5	12.8	12.1	11.0	10.0	10.0
23	9.5	9.3	9.3	11.9	7.4	11.5	11.4	11.5	9.1	9.0	12.2	12.4	13.7	11.6	10.7	10.0	9.9	12.5	12.8	12.6	12.6	11.9	11.0	10.9	12.8	10.5	12.5	11.5	10.6	9.9	9.8
24	9.2	8.9	9.1	11.8	7.4	11.6	11.2	10.8	9.8	9.1	12.3	12.4	13.7	11.8	10.2	10.0	9.9	12.5	12.6	12.8	12.4	11.6	10.8	10.5	12.4	10.1	12.3	11.0	10.0	9.6	9.4
	259.3	245.0	246.1	327.6	256.5	337.1	331.1	315.4	267.9	258.0	345.9	355.8	362.0	334.9	327.9	274.2	262.6	353.5	368.6	365.4	358.2	329.2	294.4	295.1	375.4	348.4	343.9	319.4	314.1	266.9	262.4

Figure 1: Electric Demand for industrial clients in Maine Public Service region in the NMISA power market.

Task #1: Cleaning and Merging the Data

Managerial Perspective:

Before this data could be used for analysis, the team needed to load each Excel file, read the required data, arrange the data into observations and variables, and collapse the data into a single dataframe. Furthermore, discussions with the settlements team found that these data files are manually prepared every month by copying and pasting the daily raw text files obtained from the System Administrator FTP site. It is only arranged in this format so to easily allow them to calculate the average electric demand by hour for load forecasting, and subsequent discussions with the management team discovered that this file is not used in other business processes.

This report therefore suggests that the settlements team automate the retrieval of the FTP text files and source a database solution such as Access or MySQL to store the data in an easily accessible manner. Automated reports can easily be generated in this way, and avoids the possibility of data corruption.

Technical Perspective:

The R script below provides a detailed progression of the steps taken to complete this task. Comments have been added for the sake of clarity, and where external package were used, a discussion of the functions involved is presented.

```
#First, the team set the working directory to the folder containing the electric load
#files.
setwd("C:/Users/Chris Gervais/Desktop/Term Project/Load")

#Next, a character vector called filenames was created containing a list of the
#file names that match the .xls file type in the working directory.
filenames <- list.files(full.names = TRUE, pattern = "*.xls")
```

As expected, there are 12 files in the working directory, with the following names:

```
##  [1] "./201601.xls" "./201602.xls" "./201603.xls" "./201604.xls"
##  [5] "./201605.xls" "./201606.xls" "./201607.xls" "./201608.xls"
##  [9] "./201609.xls" "./201610.xls" "./201611.xls" "./201612.xls"
```

```
#The xlsx package allows for programatic control of Excel files using R. This package
#was loaded using hte library command.
library(xlsx)

#Since the xlsx package uses the default working directory of RStudio, we need to once
```

```
#again specify the working directory of the load files.
setwd("C:/Users/Chris Gervais/Desktop/Term Project/Load")

#The lapply function takes as the first argument a list variable, and as the second
#argument a function you would like to apply across the entire list. It functions
#similar to a for loop but is faster, uses less memory, and is immune to infinite
#looping. Pass the list of filenames to the read.xlsx function within the xlsx package.
all_load_data <- lapply(filenames, function(i){read.xlsx(i,

    #The data files contain numerous sheets, we need to specify the first sheet.
    sheetIndex=1,

    #Note that the data range has been statically declared. Although the number of hours
    #in a day is static, the number of days in a month is not. This simplification will
    #have to be corrected in subsequent steps.
    rowIndex=c(5:28),
    colIndex=c(2:32),

    #Explicitly direct the read.xlsx function to return a data frame object without
    #headers.
    as.data.frame=TRUE,
    header=FALSE)})

#For demonstration purposes, the variable data_preview shows the first six hours of
#the first ten days.
data_preview <- all_load_data[[1]]
data_preview[1:6, 1:10]
```

```
##       X2    X3     X4     X5     X6     X7     X8     X9    X10    X11
## 1  9.427 9.051  9.173  9.248 10.978 11.044 10.693 10.493 10.319 10.753
## 2  9.601 9.189  9.361  9.456 11.212 11.765 11.749 11.390 11.345 10.110
## 3  9.761 9.146  9.333  9.774  8.108 11.846 11.936 11.679 11.593 10.496
## 4  9.545 9.118  9.637  9.831  8.156 11.935 11.610 11.278 11.478 10.754
## 5  9.353 9.303  9.875 10.486  8.842 12.474 11.825 12.408 11.709 11.181
## 6 10.217 9.743 10.299 12.383  9.620 13.487 13.009 12.753 11.249 10.925
```

```
#Load the extensible time series (xts) package which extends the base R capabilities
#of time series (ts) objects. This package will be used to create an xts object.
library(xts)

#Next, the team created a custom function to change the list of underlying dataframes
#in all_load_data to a list of single variable vectors stored in the variable called
#change_to_single_vector. We again make use of the lapply() function but this time
#pass it an integer vector from 1 to length(all_load_data) using seq_along()
seq_along(all_load_data)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
change_to_single_vector <- lapply(seq_along(all_load_data), function(i){

    #The function starts by initializing the data frame df_0 with the ith monthly data.
    df_0 <- all_load_data[[i]]

    #Next, we create a new data frame called df_1 that contains data for the first day.
    df_1 <- df_0[,1]
```

```
    #Using a for loop for the remaining 30 days, the script adds the remaining days by
    #concatenating df_1 on itself.
    for(j in c(2:31)){
        df_1 <- c(df_1, df_0[,j])
    }

#Lastly, the function explicitly returns the df_1 to the first position of the list and
#the process repeates for the remaining monthly data frames.
return(df_1)})

#Next the script takes the list of vectors, and colappses it down into a single list of
#vectors using unlist, and converts it to a data frame using the as.data.frame()
#function.
hourly_df <- as.data.frame(unlist(change_to_single_vector))

#We can now remove the 0's that were introduced by the 31 days / month assumption using
#base R subsetting functionality.
hourly_df <- hourly_df[hourly_df$`unlist(change_to_single_vector)`!=0,]

#Create the time index of known start and end times, given by the data files.
time_index <- seq(from = as.POSIXct("2016-01-01 01:00"),
                  to = as.POSIXct("2017-01-01 00:00"), by = "hour")

#Create the final hourly xts object using the xts() function.
hourly_xts <- xts(hourly_df, order.by = time_index, period = 168)

#Name the first variable / column, so that we can call it later on.
names(hourly_xts) = c("Demand")

#Preview the first six hours of the final xts object.
head(hourly_xts, 6)
```

```
##                     Demand
## 2016-01-01 01:00:00  9.427
## 2016-01-01 02:00:00  9.601
## 2016-01-01 03:00:00  9.761
## 2016-01-01 04:00:00  9.545
## 2016-01-01 05:00:00  9.353
## 2016-01-01 06:00:00 10.217
```

Once the electric demand data was available in the xts object, a rigourous quality control check was conducted to ensure that the newly created data object matched the original.

Task #2: Dealing with Missing Data

Managerial Perspective:

It is widely known in the power industry that temperature is a major contributor to the demand for electricity for residential customers because heating and cooling represents a significant portion of their energy consumption. For industrial clients, however, the effect of temperature on the demand for electricity largely depends on the nature of the facility's operations. For instance, in commercial applications where the main driver of electrical demand is from large pumps or motors, outside ambient temperature may not play a critical role in energy consumption. At the time of writing this report, the makeup of the commercial clients was not well-understood, and the team therefore decided to include temperature in the modeling process.

Given that the power plant is located within close proximity to the commercial facilities that it serves, a logical data source would be to use ambient temperature sensors from the power plant. Surprisingly, the plant does not collect ambient temperature data and the team was therefore forced to source this information externally from the Government of Canada historical climate records. Ambient temperature sensors could be installed and integrated into the on-site database at minimal cost, and thus, this report recommends a follow up cost-benefit analysis be conducted to explore this possibility further.

The closest weather station to the Maine Public Service region data on an hourly frequency is located in Saint Leonard, NB. This station, although located within 60 km of the majority of the industrial clients, is only updated periodically and suffers from missing data observations. To correct for the missing data, temperatures from Edmunston and Fredericton, NB were included in the dataset. More specifically, data from the *Saint Leonard CS*, *Fredericton CDA CS*, and *Edmunston* weather stations was collected. The three stations were used to determine the linear relationship between Saint Leonard, Fredericton, and Edmunston, and these relationships were used to compute the missing observations for each station. This process was then repeated ten times using twenty-five iterations, creating ten different data sets, each containing complete cases. Finally, linear regression models were created for each data set, the results of which were pooled together to obtain a final regression model and a complete dataset for Saint Leonard.

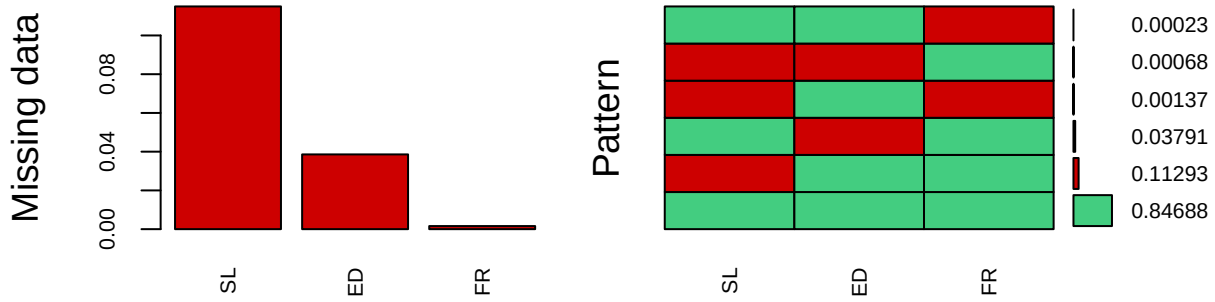
Technical Perspective:

To deal with the missing data in the Saint Leonard data set, the team used two R packages called the *Visualization and Imputatin of Missing Values (VIM)* package and the *Multivariate Imputation by Chained Equations (MICE)* package. A detailed progression of the steps taken and scripts used is presented below, and comments have been added for completeness.

```
#To start, the team loaded the VIM package.
library(VIM)

#The weather data obtained from the Government of Canada historical climate records
#was read into a variable called weather.
weather <- read.csv("C:/Users/Chris Gervais/Desktop/Term Project/Weather2.csv")

#Saint Leonard, Edmunston, and Fredericton are denoted using SL, ED, and FR in the
#weather data frame.
names(weather)
```

```
## [1] "SL" "FR" "ED"
```

```
#An aggregated plot function aggr() was used to display the missing data. Red indicates
#missing data while blue indicates complete data.
VIM_plot <- aggr(weather, col=c('seagreen3','red3'),
                 numbers=TRUE, sortVars=TRUE,
                 labels=names(weather), cex.axis=.7,
                 gap=3, ylab=c("Missing data","Pattern"),
                 cex.numbers = 0.7)
```




```
##
##  Variables sorted by number of missings:
##  Variable       Count
##        SL 0.114981785
##        ED 0.038592896
##        FR 0.001593807
```

This left hand plot shows that the data sets for Saint Leonard, Edmunston, and Fredericton are missing approximately 11.5%, 3.9%, and 0.2%, respectively. Furthermore, the right hand plot indicates the pattern of the missing cases, and shows that there are no hourly observations where all three variables are missing. Next, the nature of the missing observations was explored further using the *ggplot2* and *gridExtra* packages in R.

```
#Load the ggplot2 and gridExtra packages.
library(ggplot2)
library(gridExtra)

#Create default axis labels that can be applied to each plot.
ylabel <- "Temperature (Degrees C)"
xlabel <- "Time"

#Create the plot for Saint Leonard using the ggplot() function and the time_index object
#we created in the first task.
SL_plot <- ggplot(weather, aes(x = time_index, y = weather$SL)) +
  ylab(ylabel) +
  xlab(xlabel) +
  ggtitle("Saint Leonard") +
  geom_point()

#Create the plot for Fredericton using the ggplot() function and the time_index object
#we created in the first task.
FR_plot <- ggplot(weather, aes(x = time_index, y = weather$FR)) +
  ylab(ylabel) +
  xlab(xlabel) +
  ggtitle("Fredericton") +
  geom_point()

#Create the plot for Edmunston using the ggplot() function and the time_index object
#we created in the first task.
ED_plot <- ggplot(weather, aes(x = time_index, y = weather$ED)) +
```

```
  ylab(ylabel) +
  xlab(xlabel) +
  ggtitle("Edmunston") +
  geom_point()

#Use the grid.arrange() function from the gridExtra package to line up the plots for
#visual effect.
grid.arrange(SL_plot, FR_plot, ED_plot, nrow = 3, ncol = 1,
  bottom = "Figure 2. Hourly temperature data for Saint Leonard, Fredericton and Edmunston.")
```

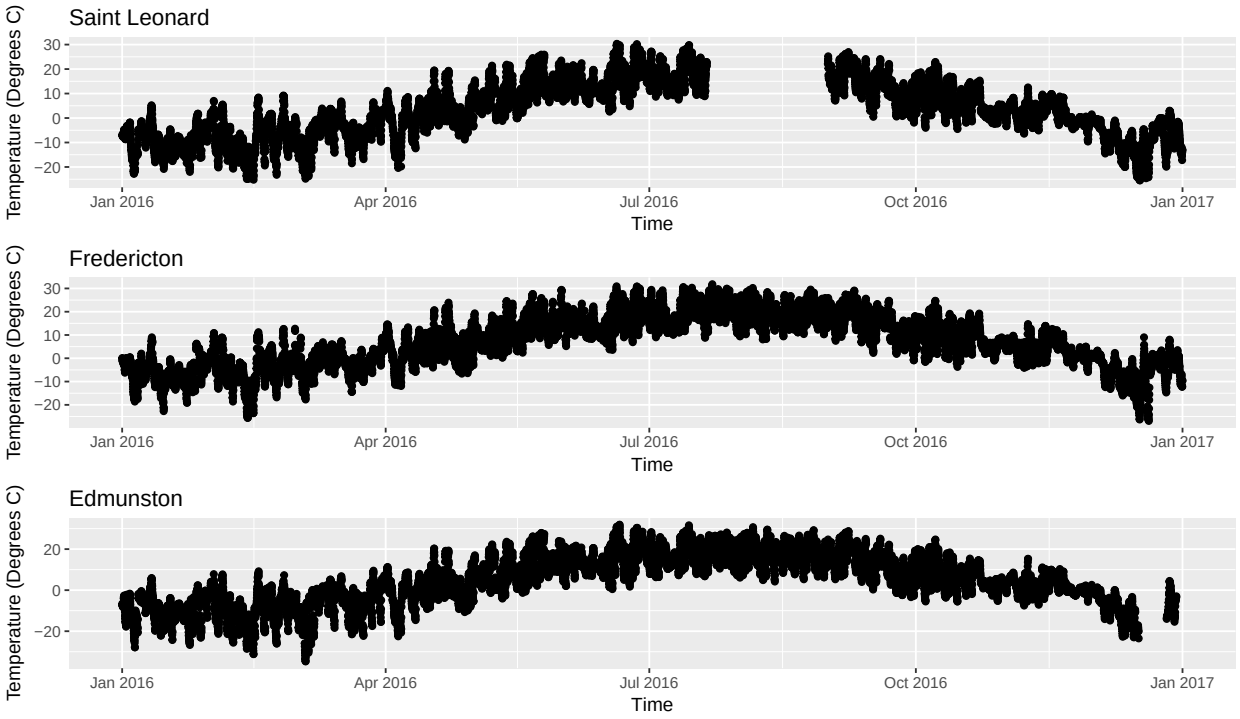


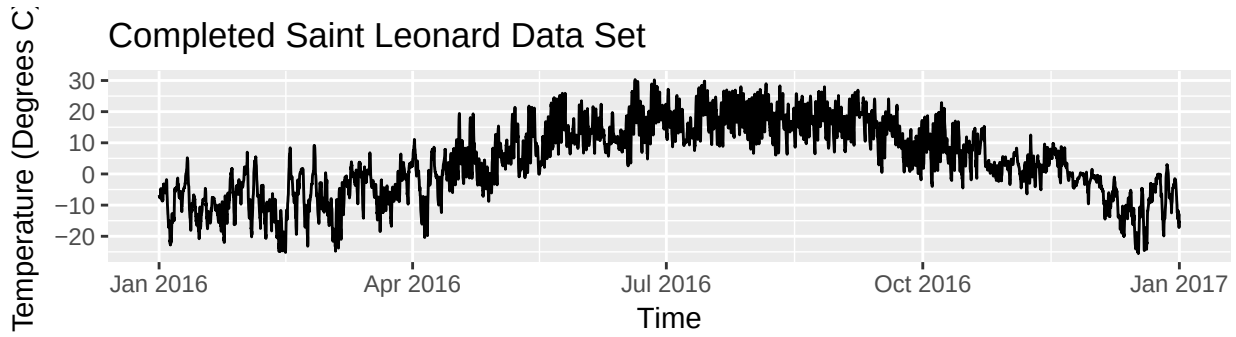
Figure 2. Hourly temperature data for Saint Leonard, Fredericton and Edmunston.

The resulting plots suggest that a large, consecutive portion of the Saint Leonard data is missing in the month of August. This result could be explained by a communication failure, an instrumentation failure, or even a combination of the two. Fortunately, the data for Fredericton and Edmunston over the same time horizon does not exhibit the same pattern. To fix the missing data, the team used multiple imputation using the *mice* package. Ten imputations were completed using 25 iterations each. The resulting regression equation and the completed data set are presented below.

```
#Create a final completed data set for Saint Leonard called SL_New, and summarize the
#final regression fit for future use.
SL_New <- complete(imputed_Data,10)
summary(final_fit)
```

```
##                   est          se         t       df Pr(>|t|)      lo 95
## (Intercept) -0.6467643 0.027169422 -23.80486 8778.037        0 -0.7000227
## FR           0.3614710 0.006141616  58.85600 8778.037        0  0.3494320
## ED           0.6348775 0.005488490 115.67436 8778.037        0  0.6241188
##                  hi 95 nmis          fmi       lambda
## (Intercept) -0.5935058   NA 0.0002277635 6.042853e-16
## FR           0.3735100   14 0.0002277635 8.127773e-16
## ED           0.6456362  339 0.0002277635 1.366025e-20
```

```
#Plot the final completed data set for Saint Leonard.
ggplot(weather, aes(x = time_index, y = SL_New$SL))+
    ylab(ylabel) +
    xlab(xlabel) +
    ggtitle("Completed Saint Leonard Data Set") +
    geom_line()
```



Note that the p-values for the parameter estimates for both Fredericton and Edmunston are significant, and the resulting completed Saint Leonard data set appears to fit well with the overall trend in the regional temperatures.

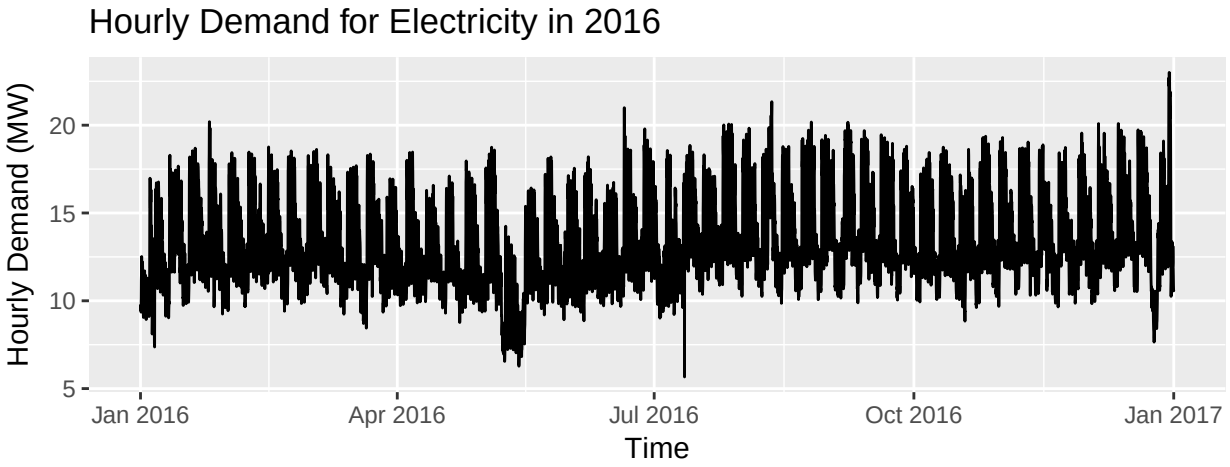
The final step in the data source section is to read in the observed regional holiday data set, provided by the HR department at APUC, and merge all three data sources into one.

```
#Create a binary variable called holidays and read the file from the working directory.
holidays <- read.csv("C:/Users/Chris Gervais/Desktop/Term Project/holiday.csv")

#Select only the holiday column from the underlying data set.
holidays <- holidays[,2]

#Merge the temperature and holiday data sets into a final variable called xreg which
#will be used as the regression terms in the time series analysis.
xreg <- cbind(holidays, SL_New$SL)
names(xreg) <- c("Demand", "Holiday", "Temperature")

#Display the first six items in xreg, representing the first six hours in January 2016.
head(xreg)
```

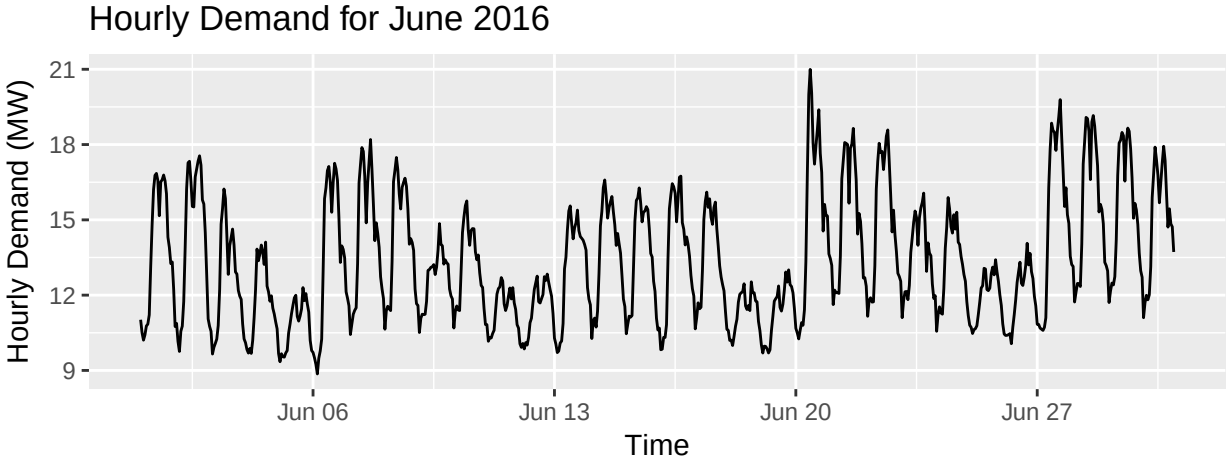
```
##      holidays
## [1,]        1 -6.8
## [2,]        1 -7.1
## [3,]        1 -7.3
## [4,]        1 -7.3
## [5,]        1 -7.4
## [6,]        1 -7.4
```

3. Analysis

The following plot shows the hourly electric demand from APUC's industrial customers in 2016.



Over this longer term horizon, it is not immediately obvious that there is an overall trend in the data set. A closer look at the individual month of June, pictured below, shows that there is clearly both a weekly and hourly trend in the data. Weekdays typically have a higher demand profile than weekends, and daytime hours typically have a higher demand profile than overnight hours. Modeling this type of multiseasonal behaviour in time series data is not a trivial task, so the team decided to start with an automated model from an established R package.



Task #3: Automated Model Development:

The team decided to use the R package *forecast* as the starting point for building the time series model. This package has automated the process of creating both exponential smoothing and autoregressive integrated moving average (ARIMA) models, and makes forecasting simple time series data accessible through the *ets()* and *auto.arima()* functions. The following section summarizes the results of these two functions.

Managerial Perspective:

The most important outcome of this modeling excercise is that APUC be able to use the forecasting technique to balance the hourly supply of the available hydroelectric output with the hourly demand of electricity from the commercial and industrial customers. If they can better forecast their requirements to serve the electric

demand in the region, they can better optimize the profitability of the asset through timely imports and exports. In short, the hourly profile of the load forecast is critically important.

Upon review of the *ets()* and *auto.arima()* models, it became apparent that more sophisticated modeling tools would be required in order for the model to be useful to the company. For example, the twenty-four hour forecasts from both the *ets()* and *auto.arima()* forecasts, presented below, show that the best prediction for the next day hourly demand is approximately 10 MW for each hour. While both models provide a reasonable estimate for the hourly demand, they are not useful from a business context.

```
#The first 24 hour forecast of the automated exponential smoothing model:
auto_ets_day1
```

```
##  [1] 10.52024 10.44365 10.38239 10.33337 10.29416 10.26279 10.23769
##  [8] 10.21761 10.20155 10.18870 10.17842 10.17020 10.16362 10.15836
## [15] 10.15415 10.15078 10.14808 10.14593 10.14420 10.14282 10.14172
## [22] 10.14084 10.14013 10.13957
```

```
#The first 24 hour forecast of the automated ARIMA model:
auto_arima_day1
```

```
##  [1] 10.63829 10.66187 10.66712 10.66871 10.66829 10.66800 10.66783
##  [8] 10.66778 10.66777 10.66778 10.66778 10.66778 10.66778 10.66778
## [15] 10.66778 10.66778 10.66778 10.66778 10.66778 10.66778 10.66778
## [22] 10.66778 10.66778 10.66778
```

Technical Perspective:

The following scripts were used to create the automated models with the *forecast* package, and subsequently forecast the first week of January 2017 hourly demand. Finally, the summary statistics for both models are presented.

```
#Load the forecast package.
library(forecast)

#Use the ets() function with the hourly demand to determine the best fit for an
#exponential smothing model.
auto_ets_model <- ets(hourly_xts)

#Use the automated ets() model to forecast the next week of hourly demand.
auto_ets_forecast <- forecast(auto_ets_model, h=168)

#Provide summary statistics on the automated ets() model.
summary(auto_ets_model)
```

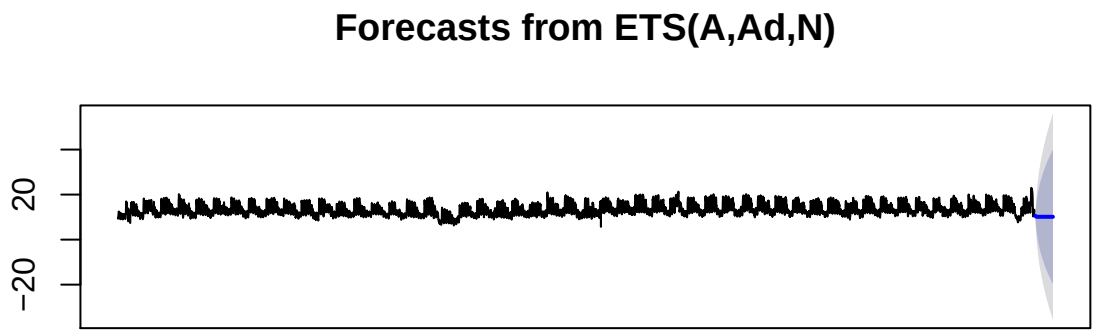
```
## ETS(A,Ad,N)
##
## Call:
##  ets(y = hourly_xts)
##
##   Smoothing parameters:
##     alpha = 0.9999
##     beta  = 0.2784
##     phi   = 0.8
##
##   Initial states:
##     l = 9.1322
##     b = 0.3884
##
```

```
##   sigma:  0.863
##
##      AIC     AICc      BIC
## 77187.70 77187.71 77230.19
##
## Training set error measures:
##                         ME      RMSE       MAE         MPE    MAPE
## Training set -2.953286e-05 0.8629719 0.6180809 -0.02587074 4.48151
##                    MASE       ACF1
## Training set 0.04565128 0.08316965
```

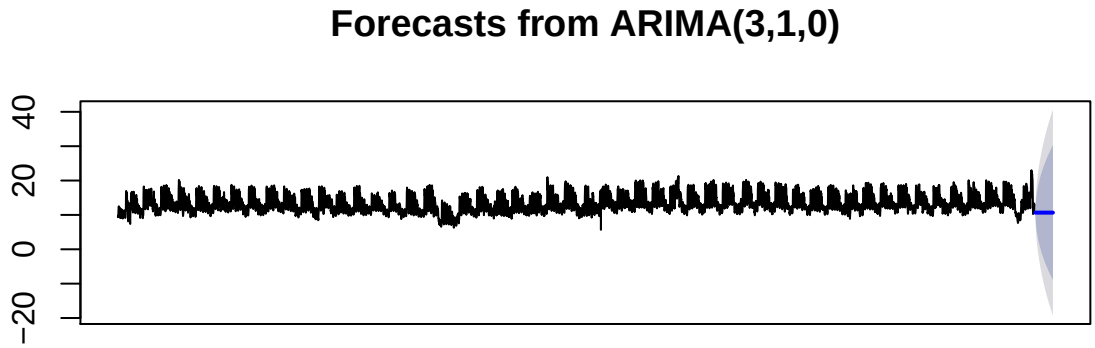
```
#Use the auto.arima() function with the hourly demand to determine the best fit for
#a non-seasonal ARIMA model.
auto_arima_model <- auto.arima(hourly_xts)

#Use the automated auto.arima() model to forecast the next week of hourly demand.
auto_arima_forecast <- forecast(auto_arima_model, h=168)

#Provide summary statistics on the automated auto.arima() model.
summary(auto_arima_model)
```

```
## Series: hourly_xts
## ARIMA(3,1,0)
##
## Coefficients:
##          ar1     ar2      ar3
##       0.2830  0.0493  -0.0476
## s.e.  0.0107  0.0111   0.0107
##
## sigma^2 estimated as 0.7163:  log likelihood=-10996.11
## AIC=22000.22   AICc=22000.22   BIC=22028.54
##
## Training set error measures:
##                         ME      RMSE       MAE        MPE     MAPE
## Training set 0.0001013619 0.8461788 0.6023556 -0.1155496 4.387218
##                    MASE         ACF1
## Training set 0.04448981 0.0008495544
```

```
#Plot both forecasts using the plot function in base R.
auto_ets_plot <- plot(auto_ets_forecast, xaxt = "n")
```




```
auto_arima_plot <- plot(auto_arima_forecast, xaxt ="n")
```



Although both the Akaike Information Criterion (AIC) and Bayesian Information Criterion show that the automated ARIMA model outperforms the exponential smoothing model, it is clear that neither are useful to the business. Future iterations of these models may consider using dummy variables for both the day of the week and the hour the day, but a more appropriate technique may be to explicitly build a seasonal ARIMA model mixed with explanatory variables.

Task #4: Dealing with Error Terms

While the *forecast* package is helpful in automating the model development process, there are other packages available that provide more control to the user. One such package is called *astsa*, which is capable of building seasonal ARIMA models with any lag and any number of explanatory variables. The following section uses the tools available in this package to develop a more appropriate model.

Managerial Perspective:

Rather than allowing the automated model selection process to determine the most statistically relevant

model, an alternative model using a forced time lag was created on the basis of business accumen. The main drawback of the previous models were that they were unable to account for the seasonality resulting from both the day of the week fluctuations as well as the hourly fluctuations each day. Rather than attempt to disentangle two seasonal components, the team decided instead to consider a lag of 168 hours, effectively accounting for both at the same time. Although this approach may not produce the most accurate results, it most certainly guarantees that the results will be usable. To accomplish this task, the team followed the steps outlined in the *astsa* package, the details of which are presented in the technical perspective below.

Technical Perspective:

To begin, the team first determined if the data is stationary using the Augmented Dickey Fuller (ADF) test to test for unit roots. The ADF test uses the null hypothesis that the data is stationary, and an alternative that the data is not stationary. If the data is not stationary, we need to perform a transformation such as taking a first difference before continuing. The ADF test in the *astsa* package allows the user to select a lag to test if the seasonality is stationary as well.

```
#Load the tseries and astsa packages.
library(tseries)
library(astsa)

#Run the ADF test using the default lag of 1 and the seasonal lag of 168. Note that we
#use 168 to indicate the weekly seasonality.
adf.test(hourly_xts)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  hourly_xts
## Dickey-Fuller = -6.7236, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(hourly_xts, k = 168)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  hourly_xts
## Dickey-Fuller = -2.8622, Lag order = 168, p-value = 0.2131
## alternative hypothesis: stationary
```

While the seasonal ADF test did not reject the null hypothesis, the non-seasonal ADF test suggests that we should reject the hypothesis. In this scenario, the *astsa* package suggests to use the results of the component you would most ideally like to model, or in our case, the seasonally lagged ADF test.

Next, the package calls for a review of the both the seasonal and non-seasonal Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) output graphs to determine the order of the ARIMA models. If the PACF output cuts off dramatically at lag k, while the ACF output tails off slowly, it suggests the model can be best defined as an autoregressive model of order k. If, on the other hand, the ACF cuts off dramatically at lag k while the PACF tails off slowly, it suggests the model is best defined as a moving average model at lag k. This exercise is best demonstrated using our data set as an example.

```
#Extract the core data from the hourly_xts object.
core <- coredata(hourly_xts)

#Select only the electric demand data.
core <- core[,1]

#Create an hourly difference data set for the PACF and ACF functions.
```

```r
Hourly_Difference <- diff(core, 1)

#Create a lagged difference data set for the PACF and ACF functions.
Lagged_Difference <- diff(core, 168)

#Plot both sets of graphs using the acf2() function in the astsa package.
acf2(Hourly_Difference)
acf2(Lagged_Difference)
```

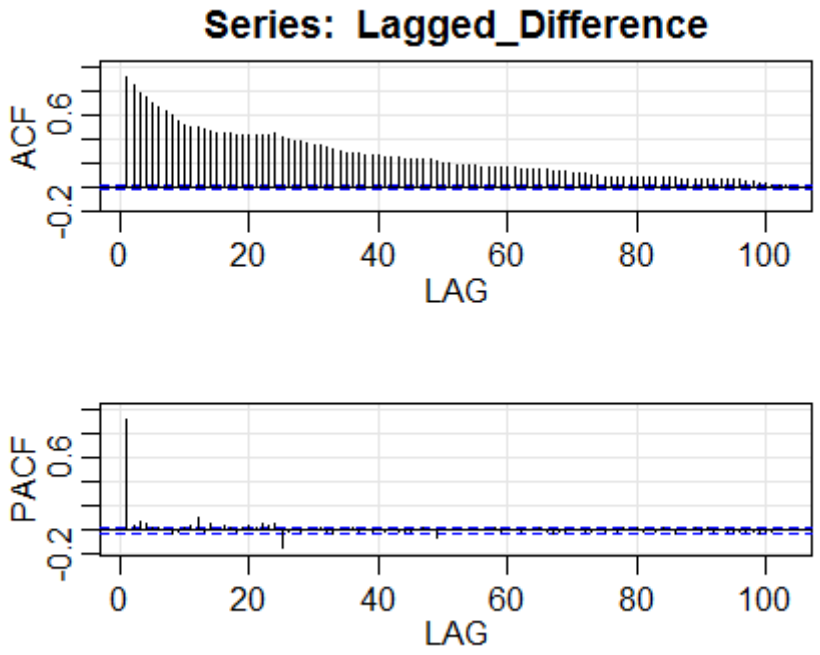


Figure 2: The ACF and PACF plots for the seasonally differenced data set.

For the hourly demand data set, the seasonal component is quite pronounced. The PACF plot clearly cuts off at the first lag while the ACF plot slowly tails off. Therefore, the *astsa* package recommends we use a seasonal ARIMA model of order (1,1,0) where the seasonal lag is = 168.

For the non-seasonal component, the choice is less clear as the non-seasonal component contains a mix of both autoregressive and The PACF plot appears to cut off at the first lag, while the ACF plot tails off after the second lag. Therefore, we will use a non-seasonal ARIMA model of order (1,1,2).

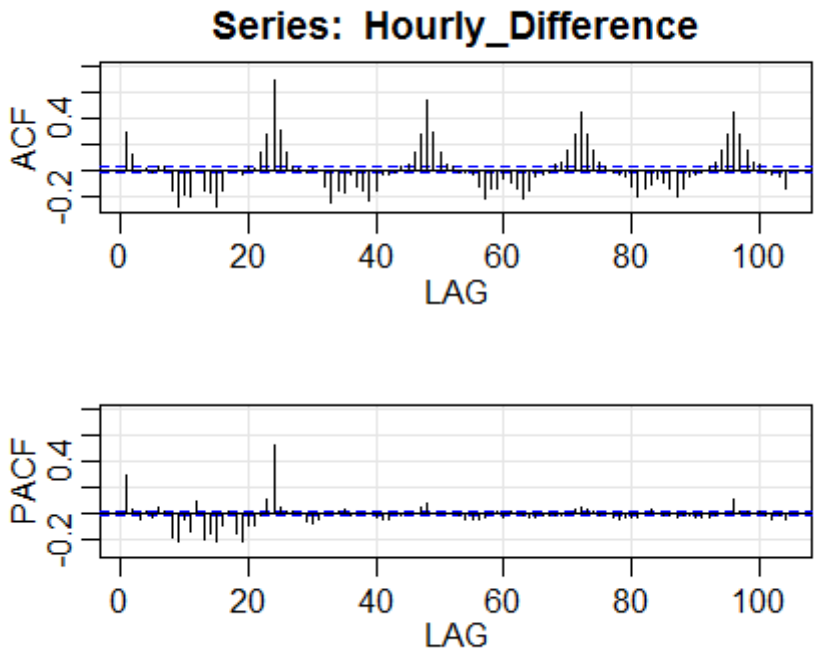


Figure 3: The ACF and PACF plots for the hourly differenced data set.

Task #5: Forecasting Time Series Data

Managerial Perspective:

The new model takes as inputs the hourly demand data, the order determined for the model in the previous section, the seasonal lag S, historical regressive terms for holiday and temperature, and the data for the upcoming period. In this case, data for the first week of January 2017 was used. It is important to note that, while the model appears to reflect the hourly and daily variation in the data, the model has not been thoroughly tested against actual data. The next steps in the development of this model is test it against real conditions to determine the performance.

Technical Perspective: Now that the order of the seasonal ARIMA model has been chosen, we can add the regressive data frame to the model and pass it new data. The *astsa* package makes this surprisingly simple, and is accomplished using the R commands below.

```
#Load the new xreg term file.
xreg_New <- read.csv("C:/Users/Chris Gervais/Desktop/Term Project/new_xregs.csv")

#Use the sarima.for() function in the astsa package to forecast the next 168 hours.
sarima.for(hourly_xts, 1,1,2,1,1,0, S= 168,
           xreg = xreg, xreg.for = xreg_New,
           n.ahead=168)
```

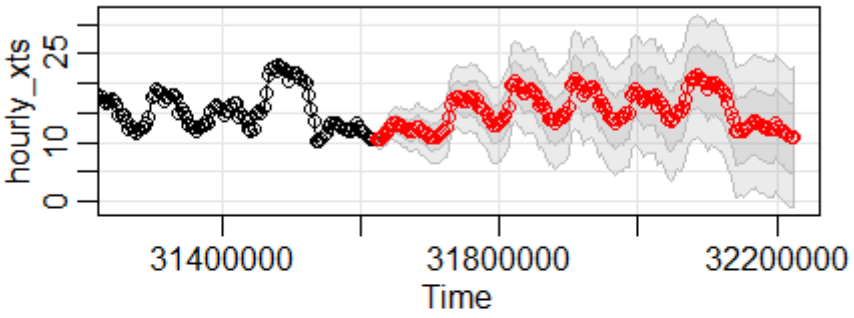


Figure 4: The ACF and PACF plots for the seasonally differenced data set.