**Assignment 2 –**

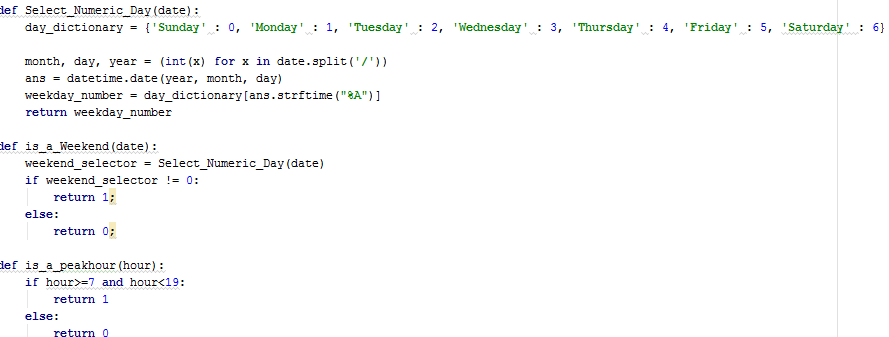
**Case 1 : Energy Forecasting**

1. **Data wrangling and cleansing**

For the problem 1, we have used Python as part of programming to read and write the csv’s. We took help from packages like pandas, numpy, and other general function.

**Creating file from raw\_data1 and raw\_data2**

We read the file raw\_data1 and raw\_data2 as provided and resulted into another csv file which has the desired output as stated in sample sample\_format.csv. We have performed the data cleaning part and removed the unwanted column i.e. Channels which was not playing a role in the desired output csv. Few columns need to be calculated from the data that we have collected from raw\_data files. We need to generate day, year, month, Peakhour, Day of Week, Weekday and total kWh in a day. We are taking the sum of all the hours in a day and keeping it as Kwh. We split the date on ‘/’ so that we can extract the day, date and month from the split data. To calculate the day of week, we have generate a dictionary which stores the day with the corresponding digit which is called by the day number. To calculate weekend column, we check if ‘day of week’ is not 0 or 6, we say that it’s a weekday. To find the peakhour, we took the data from the raw\_data and check if the hours is between 7 and 19, if it is found to be, we keep the value as 1 otherwise value as 0 for peakhour.



The generated file will have Account, Date, kWh, month, day, year, hour, Day of week and Peakhour from the raw\_data.csv file.

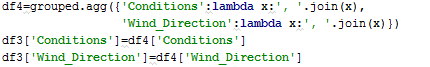
We created another python file, which extracts the data from wunderground api and keeps into the api\_full\_data.csv file.

**Hinderances**

On receiving the data from api, we observed that there are multiple datasets for a particular hour. We thus grouped the data on year,month,day,hour and calculated the mean of data using python’s panda.



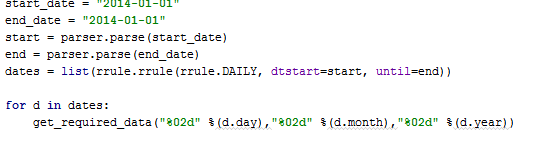
We also observed that panda removes the non-numeric rows when it performs the mean operation. Thus, we aggregated the conditions tab and the Wind\_Direction tab, which have string as values.



And copied them back to the csv file.

**Procedures**

We assigned a global variable which takes the count of hits that the API makes. Since the API provides only 10 hits in a minute, we are proceeding with the max hits and wait for 62 second ; to again get back our minute hit. We also observed that there is cap of 500 API hits in a day, and it will be great loss if we e.g. hit API for 10 months and loose the whole data if any failure occurs. We have handles the case by generating a temporary file of all the collected data till the point any failure happens. This helps us in starting the API calls from the place where we left. Since the API does not have functionality to provide data betweem certain dates, we need to loop between the start and ending date. We can set both the dates and use history function provided by the wunderground API to gather the desired data from API.

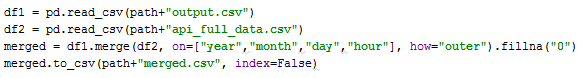


Pandas works on index. It automatically creates index when the panda is used to write the data to csv. We observed that in sample data, no index was provided. We used index = false property to remove the index while writing into csv.



**Handling the “NA” and “0” data in the merged set**

We observed that, the API is not providing all the data and for all the hours. We had some missing data for few hours. To handle this situation, we have used panda’s function fillna, where any parameter can be passed and it fills the “0” as most of columns are numeric in nature.



Panda’s merge function work on dataframes. we merged them keeping index as Year, month, day and hour where the merge criteria is outer.

**Part 2: Multiple-Linear Regression**

After getting the cleansed data that was required, In second part, we have used several model building approaches to select the best parameters for our model’s accuracy. These approaches include exhaustive search, forward selection and backward selection. We selected the parameters for our model based on the intercept value we got, and finally chose backward selection.

We then split the data up into training and test datasets. We chose the ratio as 0.80 : 0.20. Now we applied the linear regression model by selecting the response column as “kWh” and selecting the parameters we got from backward search i.e. hour, Peakhour, month, Humidity, Dew\_Point, and Temperature.

Firstly, we have used the package leaps to evaluate all the best-subset models for exhaustive search.

library(leaps)

regfit.full=regsubsets(kWh~hour+Temperature+Dew\_PointF+Humidity+Sea\_Level\_PressureIn+VisibilityMPH+Wind\_SpeedMPH+WindDirDegrees,data=inputRead)

reg.summary=summary(regfit.full)

names(reg.summary)

reg.summary$rss

reg.summary$adjr2

coef(regfit.full,7)

reg.summary

Secondly, we have used forward selection as follows,

regfit.fwd=regsubsets(kWh~hour+Temperature+Dew\_PointF+Humidity+Sea\_Level\_PressureIn+VisibilityMPH+Wind\_SpeedMPH+WindDirDegrees,data=inputRead,nvmax=8,method="forward")

F=summary(regfit.fwd)

names(F)

F

coef(regfit.full,7)

Then, we used backward selection as our final model building technique,

regfit.bwd=regsubsets(kWh~hour+Peakhour+month+Humidity+Dew\_PointF+Temperature,data=inputRead,nvmax=8,method="backward")

F=summary(regfit.bwd)

names(F)

F

coef(regfit.full,7)

Finally, on the basis of repeated trials, and intercept value, we narrowed down to using using backward selection. After that, we selected 80% data as our train data and rest 20% as test data.

#Set .80 as the percent of rows in our data for training

read\_size <- floor(0.80 \* nrow(inputRead))

#Set the seed to make your partition reproductible

set.seed(80)

train\_data\_ind <- sample(seq\_len(nrow(inputRead)), size = read\_size)

#Split the data into train\_dataing and test\_dataing

train\_data <- inputRead[train\_data\_ind, ]

test\_data <- inputRead[-train\_data\_ind, ]

Finally, we trained this data as follows,

#Modified Linear Model

lm.fit<-lm(kWh~ hour+ Peakhour + month + Humidity +Dew\_PointF + Temperature , data = train\_data)

#Summary of the fit

summary(lm.fit)

#Measures of predictive accuracy

#install.packages("forecast")

library(forecast)

pred = predict(lm.fit, test\_data)

accuracy\_pred=accuracy(pred, test\_data$kWh)

#View(pred)

library(ROCR)

account <- c("Account", unique(inputRead$Account))

write.csv(accuracy\_pred, file = "PerformanceMetrics.csv",row.names=FALSE)

library(devtools)

#getting tidy output

library(broom)

tidy\_lmfit <- tidy(coef(lm.fit))

tidy\_lmfit[,1:2]

account <- c("Account", unique(inputRead$Account))

tidy\_lmfit <- rbind(account,(tidy\_lmfit[,1:2]))

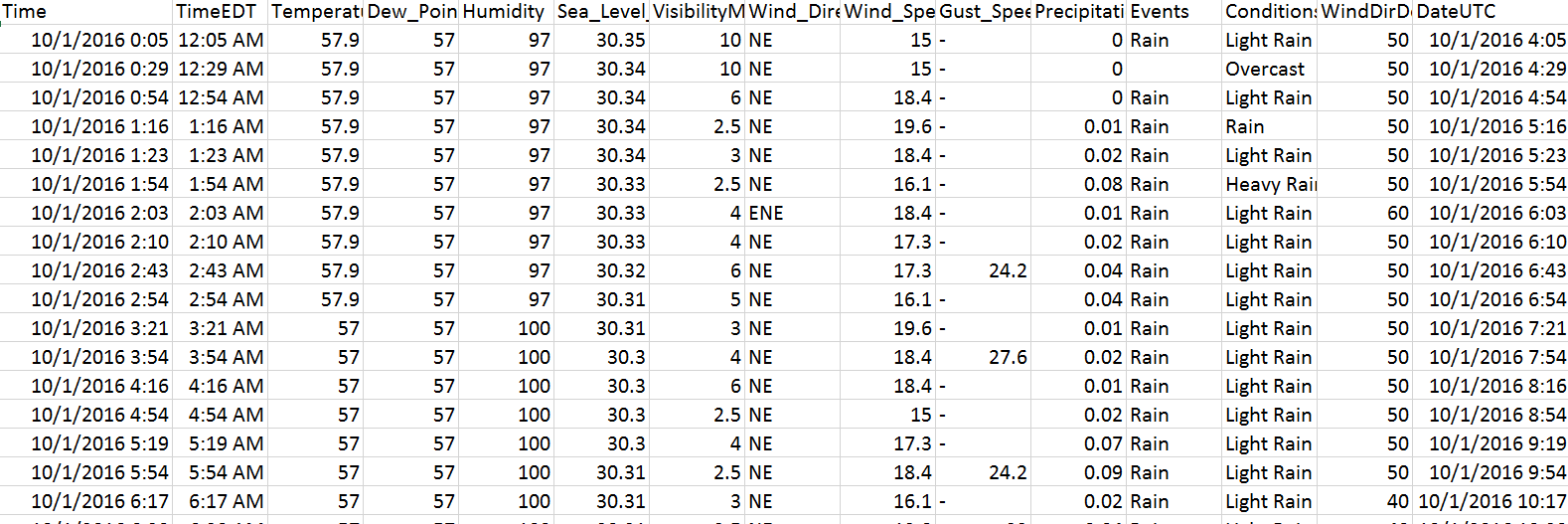
write.csv(tidy\_lmfit[,1:2], file = "RegressionOutputs.csv",row.names=FALSE)

**Part 3: Forecase**

We are using two different scripts to get the final forecast KWH data.

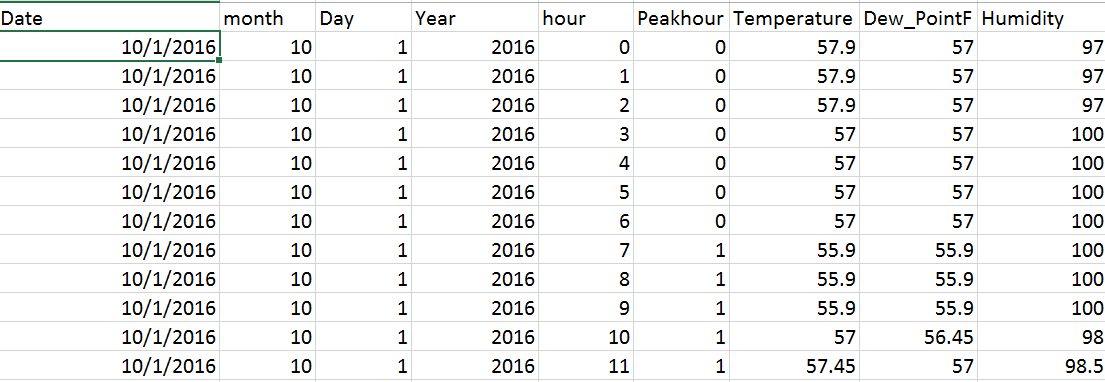
Script 1: AssignPart3\_2.py

The script takes in the forecastData.csv and converts it into forcastInput.csv, which has columns that were identifies as part of variable selection



forecastData.csv

After backward regression variable selection, it was found that we were getting better prediction values for Date, month, Day, Year, hour, Peakhour, Temperature,Dew\_PointF,Humidity. Hence we chose these values and extracted them into forecastInput.csv



forcastInput.csv

Since we have multiple data for each hour we had do a group by for the below columns and take the mean of Temperature, and other columns

grouped=df2.groupby(['Date','month','Day','Year','hour'])

df3=grouped.mean()

Script 2: Assignment2\_Part3.R

The script should be run after running the Python code **assignPart3\_2.py**

Packages to install

install.packages("forecast")

install.packages("devtools")

install.packages("broom")

install.packages("ROCR")

Read the merged.csv into a dataframe, we take the 80% of the data set and train the regression model, we train the model using the variable selected in the variable selection step.

We use below statement to train the data

lm.fit <- lm(kWh ~ hour+Peakhour+month+Humidity+Dew\_PointF+Temperature, data = train\_data)

Code:

#80% of the sample size

inputRead <- read.csv("merged.csv")

#inputRead$Temperature<-na.locf(inputRead$Temperature)

names(inputRead)

read\_size <- floor(0.80 \* nrow(inputRead))

#Set the seed to make your partition reproductible

set.seed(80)

train\_data\_ind <- sample(seq\_len(nrow(inputRead)), size = read\_size)

#Split the data into train\_dataing and test\_dataing

train\_data <- inputRead[train\_data\_ind, ]

lm.fit <- lm(kWh ~ hour+Peakhour+month+Humidity+Dew\_PointF+Temperature, data = train\_data)

After training we predict values for the data using the data read from forcastInput.csv, we use the predict() functions in the forecast library.

install.packages("forecast")

library(forecast)

pred = predict(lm.fit, test\_data)

df<-as.data.frame(pred)

accuracy(pred, test\_data$kWh)

write.csv(pred, "Prediction.csv")

The Final output of the Part 3 is the forecastOutput\_26908650026.csv where 26908650026 is the account number.

