Team 3 - **Sankalp Jadon**

Vyshak Srishylappa

Vivek Kumar

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.

```
def Select_Numeric_Day(date):
    day_dictionary = {'Sunday' : 0, 'Monday' : 1, 'Tuesday' : 2, 'Wednesday' : 3, 'Thursday' : 4, 'Friday' : 5, 'Saturday' : 6}

    month, day, year = (int(x) for x in date.split('/'))
    ans = datetime.date(year, month, day)
    weekday_number = day_dictionary[ans.strftime("%A")]
    return weekday_number

def is_a_Weekend(date):
    weekend_selector = Select_Numeric_Day(date)
    if weekend_selector != 0:
        return 1;
    else:
        return 0;

def is_a_peakhour(hour):
    if hour>=7 and hour<19:
        return 1
    else:
        return 0
```

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.

```
grouped=df.groupby(['year','month','day','hour'],as_index=False)
df3=grouped.mean()
```

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.

```
df4=grouped.agg({'Conditions':lambda x:', '.join(x),
                 'Wind_Direction':lambda x:', '.join(x)})
df3['Conditions']=df4['Conditions']
df3['Wind_Direction']=df4['Wind_Direction']
```

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.

```
start_date = "2014-01-01"
end_date = "2014-01-01"
start = parser.parse(start_date)
end = parser.parse(end_date)
dates = list(rrule.rrule(rrule.DAILY, dtstart=start, until=end))

for d in dates:
    get_required_data("%02d" %(d.day),"%02d" %(d.month),"%02d" %(d.year))
```

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.

```
df3.to_csv(path+'trial.csv', index=False)
```

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.

```
df1 = pd.read_csv(path+"output.csv")
df2 = pd.read_csv(path+"api_full_data.csv")
merged = df1.merge(df2, on=["year","month","day","hour"], how="outer").fillna("0")
merged.to_csv(path+"merged.csv", index=False)
```

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+WindDirDegree
s,data=inputRead)

reg.summary=summary(regfit.full)

names(reg.summary)

reg.summary$rss
```

reg.summary$adjr2

coef(regfit.full,7)

reg.summary

```
  (Intercept)         hour   Temperature    Dew_PointF      Humidity VisibilityMPH
 -720.73138906    0.99244532   15.00538715  -14.30040411    9.49442787   15.23845269
 Wind_SpeedMPH WindDirDegrees
    1.01029837    0.08301372
> reg.summary
Subset selection object
Call: regsubsets.formula(kWh ~ hour + Temperature + Dew_PointF + Humidity +
    Sea_Level_PressureIn + VisibilityMPH + Wind_SpeedMPH + WindDirDegrees,
    data = inputRead)
8 Variables  (and intercept)
                     Forced in Forced out
hour                     FALSE      FALSE
Temperature              FALSE      FALSE
Dew_PointF               FALSE      FALSE
Humidity                 FALSE      FALSE
Sea_Level_PressureIn     FALSE      FALSE
VisibilityMPH            FALSE      FALSE
Wind_SpeedMPH            FALSE      FALSE
WindDirDegrees           FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         hour Temperature Dew_PointF Humidity Sea_Level_PressureIn VisibilityMPH Wind_SpeedMPH
1  ( 1 ) "*"  " "         " "        " "      " "                  " "           " "
2  ( 1 ) "*"  " "         " "        "*"      " "                  " "           " "
3  ( 1 ) "*"  " "         " "        "*"      " "                  "*"           " "
4  ( 1 ) "*"  " "         "*"        " "      " "                  "*"           " "
5  ( 1 ) "*"  "*"         " "        "*"      " "                  "*"           " "
6  ( 1 ) "*"  "*"         "*"        "*"      " "                  "*"           " "
7  ( 1 ) "*"  "*"         "*"        "*"      " "                  "*"           "*"
8  ( 1 ) "*"  "*"         "*"        "*"      "*"                  "*"           "*"
         WindDirDegrees
1  ( 1 ) " "
2  ( 1 ) " "
3  ( 1 ) " "
4  ( 1 ) "*"
5  ( 1 ) "*"
6  ( 1 ) "*"
7  ( 1 ) "*"
8  ( 1 ) "*"
>
```

Secondly, we have used forward selection as follows,

regfit.fwd=regsubsets(kWh~hour+Temperature+Dew_PointF+Humidity
+Sea_Level_PressureIn+VisibilityMPH+Wind_SpeedMPH+WindDirDegre
es,data=inputRead,nvmax=8,method="forward")
F=summary(regfit.fwd)
names(F)
F

coef(regfit.full,7)

```
[1] which    rsq      rss      adjr2    cp       bic      outmat   obj
> F
Subset selection object
Call: regsubsets.formula(kWh ~ hour + Temperature + Dew_PointF + Humidity + 
    Sea_Level_PressureIn + VisibilityMPH + Wind_SpeedMPH + WindDirDegrees, 
    data = inputRead, nvmax = 8, method = "forward")
8 Variables  (and intercept)
                     Forced in Forced out
hour                     FALSE      FALSE
Temperature              FALSE      FALSE
Dew_PointF               FALSE      FALSE
Humidity                 FALSE      FALSE
Sea_Level_PressureIn     FALSE      FALSE
VisibilityMPH            FALSE      FALSE
Wind_SpeedMPH            FALSE      FALSE
WindDirDegrees           FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: forward
         hour Temperature Dew_PointF Humidity Sea_Level_PressureIn VisibilityMPH Wind_SpeedMPH
1  ( 1 ) "*"  " "         " "        " "      " "                  " "           " "          
2  ( 1 ) "*"  " "         " "        "*"      " "                  " "           " "          
3  ( 1 ) "*"  " "         " "        "*"      " "                  "*"           " "          
4  ( 1 ) "*"  " "         " "        "*"      " "                  "*"           " "          
5  ( 1 ) "*"  "*"         " "        "*"      " "                  "*"           " "          
6  ( 1 ) "*"  "*"         "*"        "*"      " "                  "*"           " "          
7  ( 1 ) "*"  "*"         "*"        "*"      " "                  "*"           "*"          
8  ( 1 ) "*"  "*"         "*"        "*"      "*"                  "*"           "*"          
         WindDirDegrees
1  ( 1 ) " "           
2  ( 1 ) " "           
3  ( 1 ) " "           
4  ( 1 ) "*"           
5  ( 1 ) "*"           
6  ( 1 ) "*"           
7  ( 1 ) "*"           
8  ( 1 ) "*"           
> coef(regfit.full,7)
   (Intercept)           hour    Temperature     Dew_PointF       Humidity  VisibilityMPH 
 -720.73138906     0.99244532    15.00538715   -14.30040411     9.49442787    15.23845269 
 Wind_SpeedMPH WindDirDegrees 
    1.01029837     0.08301372 
```

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)

```
[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
> F
Subset selection object
Call: regsubsets.formula(kWh ~ hour + month + Humidity + Dew_PointF +
    Temperature, data = inputRead, nvmax = 8, method = "backward")
5 Variables  (and intercept)
            Forced in Forced out
hour            FALSE      FALSE
month           FALSE      FALSE
Humidity        FALSE      FALSE
Dew_PointF      FALSE      FALSE
Temperature     FALSE      FALSE
1 subsets of each size up to 5
Selection Algorithm: backward
         hour month Humidity Dew_PointF Temperature
1  ( 1 ) "*"  " "   " "      " "        " "
2  ( 1 ) " "  "*"   " "      " "        "*"
3  ( 1 ) "*"  "*"   "*"      " "        " "
4  ( 1 ) "*"  "*"   " "      "*"        "*"
5  ( 1 ) "*"  "*"   "*"      "*"        "*"
> coef(regfit.full,7)
  (Intercept)          hour   Temperature     Dew_PointF      Humidity  VisibilityMPH
-720.73138906    0.99244532   15.00538715   -14.30040411    9.49442787    15.23845269
 Wind_SpeedMPH WindDirDegrees
   1.01029837     0.08301372
>
```

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

Time	TimeEDT	Temperat	Dew_Poin	Humidity	Sea_Level_	VisibilityM	Wind_Dire	Wind_Spe	Gust_Spee	Precipitati	Events	Condition	WindDirD	DateUTC
10/1/2016 0:05	12:05 AM	57.9	57	97	30.35	10	NE	15	-	0	Rain	Light Rain	50	10/1/2016 4:05
10/1/2016 0:29	12:29 AM	57.9	57	97	30.34	10	NE	15	-	0		Overcast	50	10/1/2016 4:29
10/1/2016 0:54	12:54 AM	57.9	57	97	30.34	6	NE	18.4	-	0	Rain	Light Rain	50	10/1/2016 4:54
10/1/2016 1:16	1:16 AM	57.9	57	97	30.34	2.5	NE	19.6	-	0.01	Rain	Rain	50	10/1/2016 5:16
10/1/2016 1:23	1:23 AM	57.9	57	97	30.34	3	NE	18.4	-	0.02	Rain	Light Rain	50	10/1/2016 5:23
10/1/2016 1:54	1:54 AM	57.9	57	97	30.33	2.5	NE	16.1	-	0.08	Rain	Heavy Rai	50	10/1/2016 5:54
10/1/2016 2:03	2:03 AM	57.9	57	97	30.33	4	ENE	18.4	-	0.01	Rain	Light Rain	60	10/1/2016 6:03
10/1/2016 2:10	2:10 AM	57.9	57	97	30.33	4	NE	17.3	-	0.02	Rain	Light Rain	50	10/1/2016 6:10
10/1/2016 2:43	2:43 AM	57.9	57	97	30.32	6	NE	17.3	24.2	0.04	Rain	Light Rain	50	10/1/2016 6:43
10/1/2016 2:54	2:54 AM	57.9	57	97	30.31	5	NE	16.1	-	0.04	Rain	Light Rain	50	10/1/2016 6:54
10/1/2016 3:21	3:21 AM	57	57	100	30.31	3	NE	19.6	-	0.01	Rain	Light Rain	50	10/1/2016 7:21
10/1/2016 3:54	3:54 AM	57	57	100	30.3	4	NE	18.4	27.6	0.02	Rain	Light Rain	50	10/1/2016 7:54
10/1/2016 4:16	4:16 AM	57	57	100	30.3	6	NE	18.4	-	0.01	Rain	Light Rain	50	10/1/2016 8:16
10/1/2016 4:54	4:54 AM	57	57	100	30.3	2.5	NE	15	-	0.02	Rain	Light Rain	50	10/1/2016 8:54
10/1/2016 5:19	5:19 AM	57	57	100	30.3	4	NE	17.3	-	0.07	Rain	Light Rain	50	10/1/2016 9:19
10/1/2016 5:54	5:54 AM	57	57	100	30.31	2.5	NE	18.4	24.2	0.09	Rain	Light Rain	50	10/1/2016 9:54
10/1/2016 6:17	6:17 AM	57	57	100	30.31	3	NE	16.1	-	0.02	Rain	Light Rain	40	10/1/2016 10:17

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

Date	month	Day	Year	hour	Peakhour	Temperature	Dew_PointF	Humidity
10/1/2016	10	1	2016	0	0	57.9	57	97
10/1/2016	10	1	2016	1	0	57.9	57	97
10/1/2016	10	1	2016	2	0	57.9	57	97
10/1/2016	10	1	2016	3	0	57	57	100
10/1/2016	10	1	2016	4	0	57	57	100
10/1/2016	10	1	2016	5	0	57	57	100
10/1/2016	10	1	2016	6	0	57	57	100
10/1/2016	10	1	2016	7	1	55.9	55.9	100
10/1/2016	10	1	2016	8	1	55.9	55.9	100
10/1/2016	10	1	2016	9	1	55.9	55.9	100
10/1/2016	10	1	2016	10	1	57	56.45	98
10/1/2016	10	1	2016	11	1	57.45	57	98.5

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.

Date	hour	Temperatı	KWH
10/1/2016	0	57.9	123.838
10/1/2016	1	57.9	123.4809
10/1/2016	2	57.9	123.1237
10/1/2016	3	57	120.2741
10/1/2016	4	57	119.9169
10/1/2016	5	57	119.5598
10/1/2016	6	57	119.2026
10/1/2016	7	55.9	251.6676
10/1/2016	8	55.9	251.3104
10/1/2016	9	55.9	250.9532
10/1/2016	10	57	253.117
10/1/2016	11	57.45	253.1056
10/1/2016	12	57.9	252.8281
10/1/2016	13	57.9	252.4709
10/1/2016	14	57	250.7879
10/1/2016	15	55.9	248.8102
10/1/2016	16	55	247.1272
10/1/2016	17	55	246.7701
10/1/2016	18	55.9	248.9054
10/1/2016	19	55.9	113.5223