National College of Ireland

Forecasting Residential Electricity Load Demand using Machine Learning Configuration Manual

MSc Research Project
Data Analytics

Arun Kumar Panigrahi
Student ID: x18186840

School of Computing
National College of Ireland

Supervisor: Dr. Manaz Kaleel

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Arun Kumar Panigrahi
Student ID:	x18186840
Programme:	M.Sc. Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Manaz Kaleel
Submission Due Date:	28/08/2020
Project Title:	Forecasting Residential Electricity Load Demand using Machine Learning Configuration Manual
Word Count:	XXX
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	27th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Forecasting Residential Electricity Load Demand using Machine Learning Configuration Manual

Arun Kumar Panigrahi
x18186840

1 Introduction

The configuration manual includes a comprehensive overview of the device specifications and various prerequisites along with detailed description of the programming language utilized and the number of libraries, bundles and packages considered for the research study:

"Forecasting Residential Electricity Load Demand using Machine Learning."

This manual also provides a detailed description of the procedures that had been followed to obtain the required data, clean and transform the collected data, and use the data for the implementation of proposed machine learning models.

2 System Configuration

2.1 Hardware Specification

The entire project has been implemented in a PC with the system configuration as presented in the Table 1.

Operating System:	WINDOWS 10 (2020 Microsoft Corporation)
Processor:	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90GHz
RAM:	8GB
System Type:	64-bit Operating System, x64-based Processor
Hard Disk:	1TB

Table 1: System Configuration

2.2 Software Specification

The below mentioned programming tools with different packages was used for the complete execution of the project. RStudio was used as framework to execute the R programming code and Jupyter Notebook from the Anaconda Navigator was used for execution of the Python Programming code.

2.2.1 R[1]

RStudio[2] - Version 1.3.959 is used as an Integrated Development Environment (IDE) for R Programming Language and installed packages are as mentioned below,

1. tseries[3]
2. Plotly [4]
3. jsonlite[5]
4. Prophet[6]
5. tidyverse[7]
6. Lubridate[8]
7. Forecast[9]
8. MLmetrics[10]

2.2.2 Python[11] Version 3.7.4

- Jupyter Notebook - Version 6.0.3 from Anaconda Navigator[12] - Version 1.9.12 is used for python programming

Libraries Utilised

1. Pandas[13]
2. Numpy[14]
3. Keras[15]
4. Tensorflow[16]
5. Matplotlib[17]
6. Scikit-Learn[18]

[1]https://www.r-project.org/
[2]https://rstudio.com/products/rstudio/download/
[3]https://rdrr.io/cran/tseries/
[4]https://www.rdocumentation.org/packages/plotly/versions/4.9.2.1
[5]https://rdrr.io/cran/jsonlite/
[6]http://facebook.github.io/prophet/docs/installation.html
[7]https://www.rdocumentation.org/packages/tidyverse/versions/1.3.0
[8]https://www.rdocumentation.org/packages/lubridate/versions/1.7.9
[9]https://www.rdocumentation.org/packages/forecast/versions/8.12
[10]https://rdrr.io/cran/MLmetrics/
[11]https://www.python.org/
[12]https://www.anaconda.com/
[13]https://anaconda.org/anaconda/pandas
[14]https://anaconda.org/anaconda/numpy
[15]https://anaconda.org/conda-forge/keras
[16]https://anaconda.org/conda-forge/tensorflow
[17]https://anaconda.org/anaconda/matplotlib
[18]https://anaconda.org/anaconda/scikit-learn

3 Data Source

3.1 Electricity Load Consumption Data[19]

Below Mentioned Figure 1 and Figure 2 is the snippets of the electricity and the weather dataset that has been considered for the analysis

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energy_std	energy_sum	energy_min
1	MAC000131	2011-12-15	0.4850	0.4320455	0.868	22	0.23914580	9.505	0.072
2	MAC000131	2011-12-16	0.1415	0.2961667	1.116	48	0.28147132	14.216	0.031
3	MAC000131	2011-12-17	0.1015	0.1898125	0.685	48	0.18840469	9.111	0.064
4	MAC000131	2011-12-18	0.1140	0.2189792	0.676	48	0.20291928	10.511	0.065
5	MAC000131	2011-12-19	0.1910	0.3259792	0.788	48	0.25920496	15.647	0.066
6	MAC000131	2011-12-20	0.2180	0.3575000	1.077	48	0.28759657	17.160	0.066
7	MAC000131	2011-12-21	0.1305	0.2350833	0.705	48	0.22206965	11.284	0.066
8	MAC000131	2011-12-22	0.0890	0.2213542	1.094	48	0.26723888	10.625	0.062
9	MAC000131	2011-12-23	0.1605	0.2911250	0.749	48	0.24907605	13.974	0.065
10	MAC000131	2011-12-24	0.1070	0.1690000	0.613	47	0.15068467	7.943	0.065
11	MAC000131	2011-12-25	0.2175	0.3391875	0.866	48	0.26310120	16.281	0.069
12	MAC000131	2011-12-26	0.1495	0.2617083	0.838	48	0.24479274	12.562	0.066
13	MAC000131	2011-12-27	0.1430	0.2740000	0.778	48	0.25212746	13.152	0.068
14	MAC000131	2011-12-28	0.1455	0.3005208	1.207	48	0.29868029	14.425	0.066
15	MAC000131	2011-12-29	0.1520	0.3070417	0.888	48	0.26445463	14.738	0.066

Showing 1 to 18 of 3,510,433 entries, 9 total columns

Figure 1: Electricity Load Consumption Data

3.2 Daily Weather Data[20]

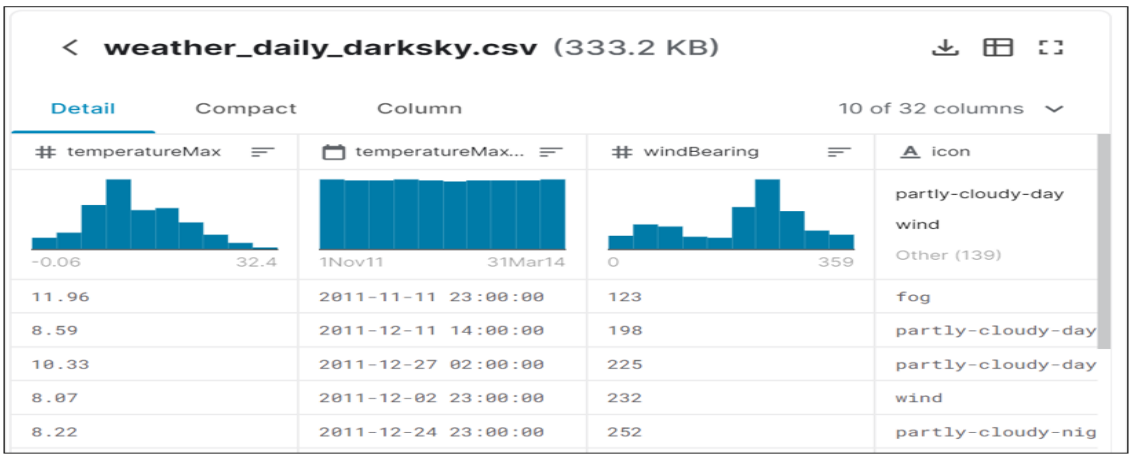

Figure 2: Electricity Load Consumption Data

[19]https://www.kaggle.com/jeanmidev/smart-meters-in-london?select=daily_dataset.csv.gz
[20]https://www.kaggle.com/jeanmidev/smart-meters-in-london?select=weather_daily_darksky.csv

4 Project Implementation

After the selection of the appropriate dataset, it has imported into RStudio for Data Pre-Processing, Data Transformation and Model implementation as described in the report.

4.1 Selecting the Working Directory

Run the below mentioned code snippets in Figure 3 to select the working directory and import the dataset into the Rstudio.

```
wroking_dirc <- choose.dir(getwd(), "Choose a suitable folder")
setwd(wroking_dirc)
getwd()

wth_daily <- read_csv("weather_daily_darksky.csv") #Reading daily weather report dataset
all_block <- read_csv("daily_dataset.csv") #Reading load consumtion dataset
```

Figure 3: Selecting the Working Directory

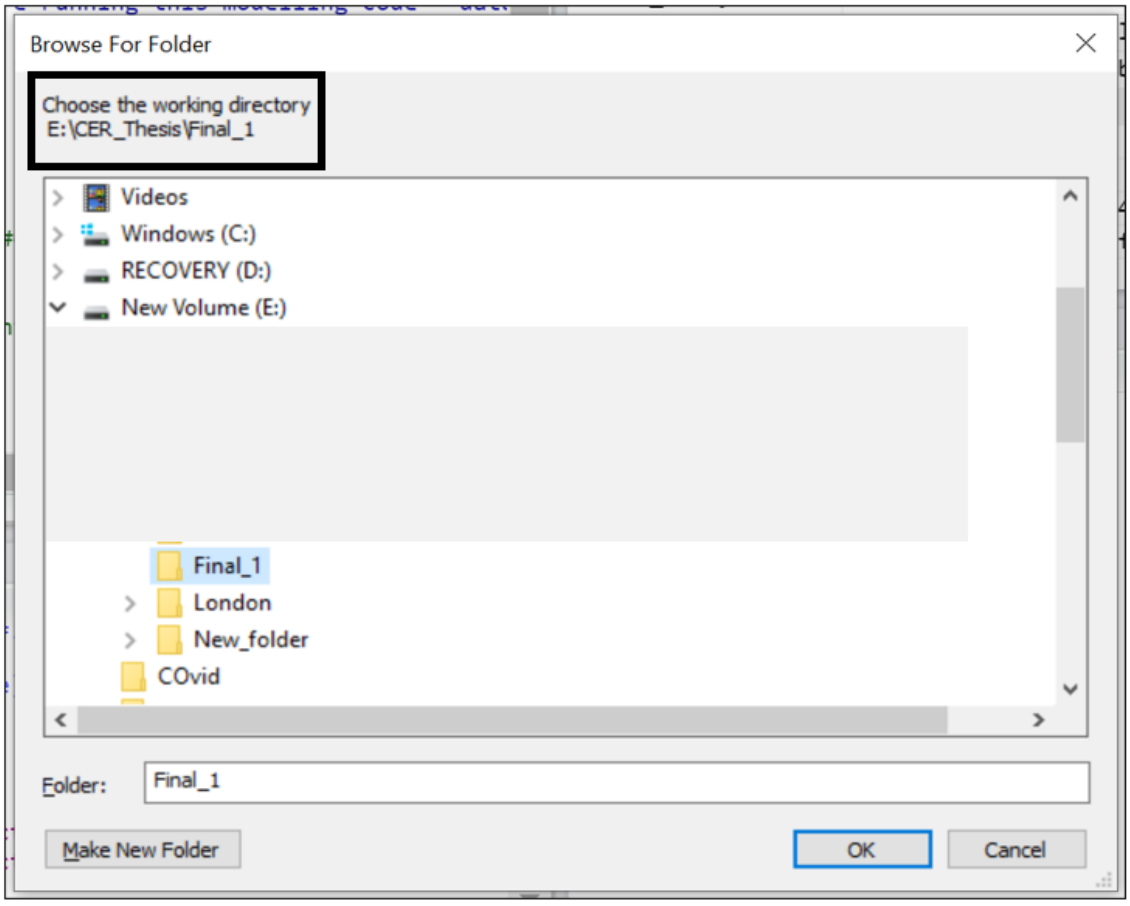

Figure 4: Selecting the Working Directory

4.2 Data Pre-Processing

Computation of Missing Value

```
########################Checking For Null Values#####################################
apply(all_block, 2, function(x) any(is.na(x)))
apply(wth_daily, 2, function(x) any(is.na(x)))

##########Missing value and outlier adjustment###############
new_energy_sum <- ts(all_block[, c('energy_sum')])
#TS = The ts() function will convert a numeric vector into an R time series object.
all_block$energy_sum <- na.interpolation(new_energy_sum, option = "linear")
#na.interpolation() identifies and imputes the missing values
```

Figure 5: Missing Value Computation

4.3 Data Transformation

In this phase, at first, the daily average daily temperature was computed.

```
#####Calculation of average Temperature of the day#####################
wth_daily$avg_temperature <- (wth_daily$temperatureHigh + wth_daily$temperatureMin)/2
```

Figure 6: Average Daily Temperature Computation

As mentioned in the research report, Figure 7 presents the code snippets of the further data transformation processes, such as grouping of target attributes of both the dataset as per date followed by the concatenation of both the dataset.

```
################Creating a Final DataFrame#####################
#Grouping of Electricity consumption records in accordance to date
all_block.year_month_grp_eng <- all_block %>%
  group_by(year_month_day) %>%
  summarise(LCLid_uniq_count = length(unique(LCLid)),#accumulating the MeterID
            energy_sum_mean = mean(energy_sum), #calculating the mean of daily electricity consumption
            energy_sum_total = sum(energy_sum), #calculation the total electricity consumption in a day
            n=n())


#Grouping of average daily temp in accordance to date
wth_daily.year_month_grp_wth <- wth_daily %>%
  group_by(year_month_day) %>%
  summarise(avg_temp_mean = mean(avg_temperature),
            n=n())

#joining temperature and electricity consumtion dataframe
all_year_month_eng_wth <- inner_join(all_block.year_month_grp_eng,
                                     wth_daily.year_month_grp_wth,
                                     by ="year_month_day")
```

Figure 7: Dataset Concatenation

After the transformed data has been gathered, the data is analysed and features like those of seasonal variation and trend is observed by creating various plot as mentioned in Figure 8 and Figure 9.

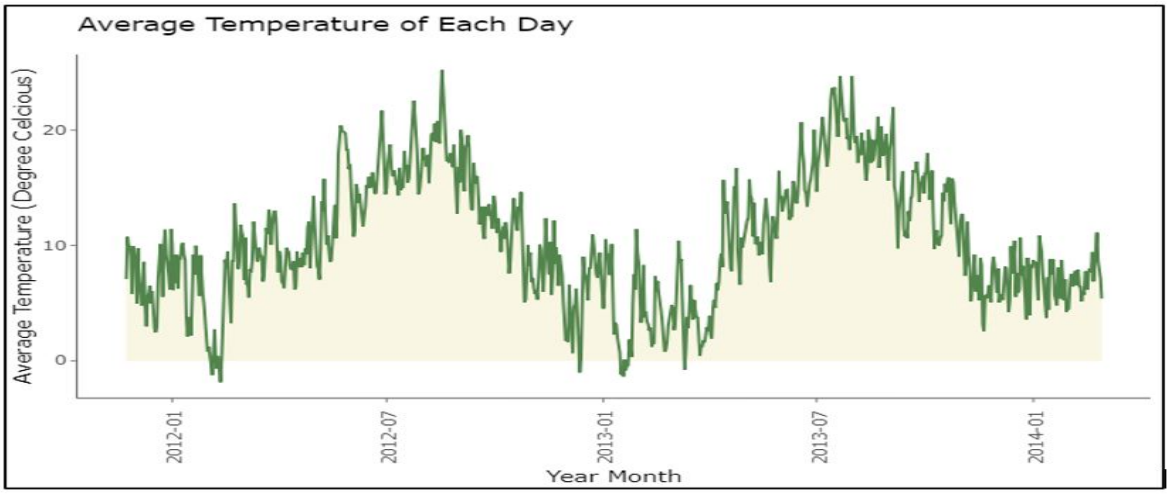

Figure 8: Average Temperature Per day (°C)

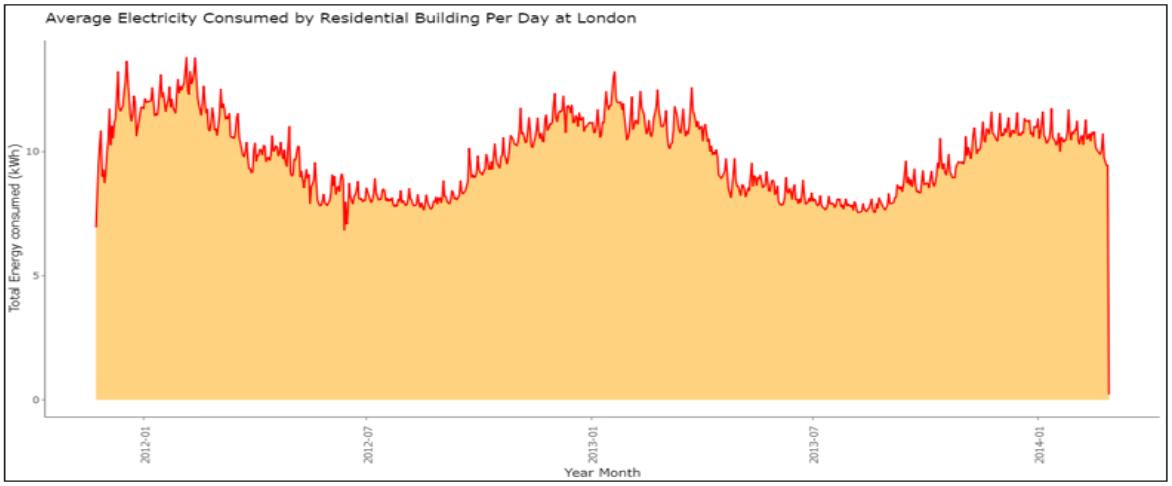

Figure 9: Average Electricity Consumption Per Day (KWH)

4.4 Modeling

4.4.1 R Programming

Figure 10 presents the snippet of calling and cunning of Complete Data Pre-Processing codes at one go, instead of executing all the code one-by-one.

```
#########Prerequisite##############

print("Please Run the Data PreProcessing Code File Before running this modelling code - data

wroking_dirc <- choose.dir(getwd(), "Choose the working directory")
setwd(wroking_dirc)
getwd()
source("Data_Pre_Processing.r")
options(warn=-1)
```

Figure 10: Execution of initial Pre-Processing codes

Pre-Processing for ARIMAX and SARIMAX

A number of tests were conducted prior to implementation of the ARIMAX Model to check whether the data comprises of appropriate knowledge or random noise at all. As, stated in the code snippet present in Figure 11, Initially the decomposition of the time series data into seasonal, trent and irregular components using Loess (STL) has been done. Next, using the seasadj() function seasonal adjustment was performed. Finally, as described in the research report the Dickey Fuller test (adf.test) , Autocorrelation Function (ACF) and Partial Autocorrelation (PACF) plots has been carried out.

```
#STL = Decompose a time series into seasonal, trend and irregular components using loess, acronym STL.
decomp = stl(count_er,s.window="periodic")
plot(decomp)

decomp_wether = stl(count_wethe,s.window="periodic")
plot(decomp_wether)


#seasadj = Returns seasonally adjusted data constructed by removing the seasonal component.
deseasonal_cnt <- seasadj(decomp)
deseasonal_cnt_wether <- seasadj(decomp_wether)


############# ADF test test for Electricity Data##############
adf.test(deseasonal_cnt, alternative = "stationary")
par(mfrow = c(1,2))
Acf(deseasonal_cnt, main='', lag.max = 100)
Pacf(deseasonal_cnt, main='', lag.max = 10)

############# ADF test test for temerature Data##############
adf.test(deseasonal_cnt_wether, alternative = "stationary")
par(mfrow = c(1,2))
Acf(deseasonal_cnt_wether, main='', lag.max = 20)
Pacf(deseasonal_cnt_wether, main='', lag.max = 20)
```

Figure 11: Data Testing

The Graphs in Figure 12 represents the seasonal, trend and irregular components of the daily average residential Electricity consumption and daily average temperature.

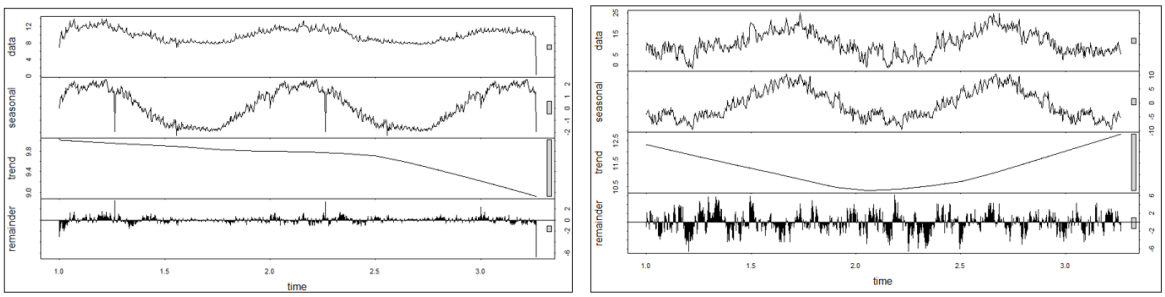

Figure 12: STL Graph for Electricty and Weather data

ARIMAX Modelling - Figure 13 presents the code snippet for the implementation of the ARIMAX model and Figure 14 represents the evaluation of the ARIMAX model outcomes.

```
################ Arimax Modeling#############################################################
training_electricity <- subset(deseasonal_cnt, end=566)
test_electricity <-subset(deseasonal_cnt, start =567)

training_weather <- subset(deseasonal_cnt_wether, end = 566)
test_weather <-subset(deseasonal_cnt_wether, start =567)

multivari_arima <- arima(training_electricity, xreg = training_weather, order = c(9,0,7))
summary(multivari_arima)

preds.temporal <- predict(multivari_arima, newxreg = test_weather, n.ahead = 261)

result  <- cbind(test_electricity,preds.temporal$pred)
#View(result)

plot(result, plot.type="single", col = 1:ncol(result))
legend("bottomleft", colnames(result), col=1:ncol(result), lty=1, cex=.65)

#############################################################################################
```

Figure 13: ARIMAX Modelling

```
####################Evaluation of Arimax Model#######################################
library(MLmetrics)
MAPE(preds.temporal$pred, test_electricity)
MSE(preds.temporal$pred, test_electricity)
RMSE(preds.temporal$pred, test_electricity)
MAE(preds.temporal$pred, test_electricity)
```

Figure 14: ARIMAX Modelling

SARIMAX Modelling -

The graph in Figure 15 represents the ACF and PACF test of the daily average temperature attribute.

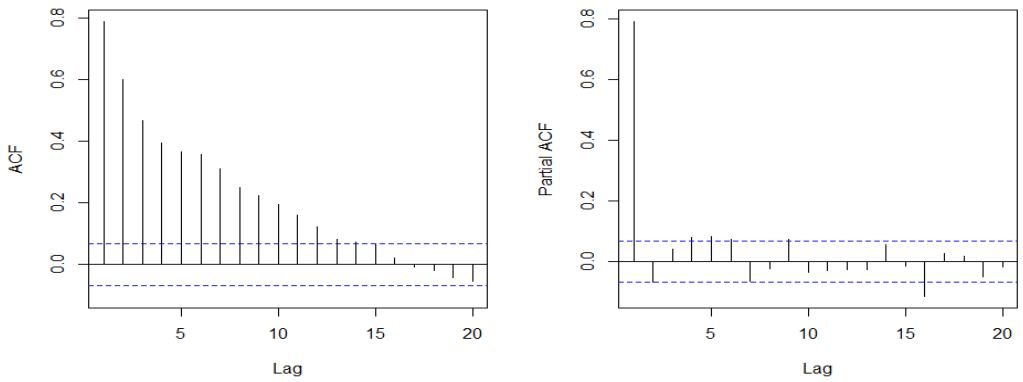

Figure 15: ACF and PACF Plot of Weather Data

Figure 16, presents the SARIMAX model specification and accuracy testing procedures. As mentioned in the report it is being treated identical as of the ARIMAX model.

```
#######################SARIMAX#########################################################

fit1 = arima(training_electricity, xreg=training_weather, order=c(9,0,7), seasonal=list(order=c(9,0,1), period=3))
summary(fit1)
preds.temporal123 <- predict(fit1, newxreg = test_weather, n.ahead = 261)


result_sarimax  <- cbind(test_electricity,preds.temporal123$pred)

plot(result_sarimax, plot.type="single", col = 1:ncol(result))
legend("bottomleft", colnames(result), col=1:ncol(result), lty=1, cex=.65)


library(MLmetrics)
MAPE(preds.temporal123$pred, test_electricity)
MSE(preds.temporal123$pred, test_electricity)
RMSE(preds.temporal123$pred, test_electricity)
MAE(preds.temporal123$pred, test_electricity)

#########################################################################
```

Figure 16: SARIMAX Modelling

PROPHET Implementation - Figure 17, indicates the pre-processing steps along with the specification and accuracy testing procedures for PROPHET model.

```
####################PROPHET Model#####################################################################
dataset_prof <- dataset123
names(dataset_prof)[names(dataset_prof) == "year_month_day"] <- "ds"
names(dataset_prof)[names(dataset_prof) == "energy_sum_mean"] <- "y"


m <- prophet(dataset_prof, daily.seasonality=FALSE)
future <- make_future_dataframe(m, periods = 365)
predt <- predict(m, future)

View(predt)

plot(m, predt)
prophet_plot_components(m, predt)
dyplot.prophet(m, predt)

#################Evaluation of Prophet Model#################################
df.cv <- cross_validation(m, initial = 400, period = 30, horizon = 365, units = 'days')
df.cv

MAPE(df.cv$y, df.cv$yhat) #Y = Real Value and Yhat = Predicted value
MSE(df.cv$y, df.cv$yhat)
RMSE(df.cv$y, df.cv$yhat)
MAE(df.cv$y, df.cv$yhat)
```

Figure 17: PROPHET Modelling

4.5 Python Programming

Pre-Processing for LSTM Model Implementation

As presented in the Figure 18 and Figure 19, before the implementation of the LSTM model, the dataset is import into the Python programming environment - Jupyter notebook. Initially the pre-proceeded file was exported from rstudio and stored as csv file, this csv file was imported into the jupyter notebook for LSTM model implementation

```
In [3]: weather_energy = pd.read_csv("LSTM.csv")
        weather_energy
```

Out[3]:

	year_month_day	LCLid_uniq_count	energy_sum_mean	energy_sum_total	avg_temp_mean
0	2011-11-23	13	6.952692	90.385000	7.085
1	2011-11-24	25	8.536480	213.412000	10.745
2	2011-11-25	32	9.499781	303.993000	9.865
3	2011-11-26	41	10.267707	420.976000	9.985
4	2011-11-27	41	10.850805	444.883001	9.005
...	...	...	...	...	...
822	2014-02-24	4975	9.802926	48769.557646	11.110
823	2014-02-25	4975	9.644231	47980.048978	8.550
824	2014-02-26	4972	9.490112	47184.835606	7.730
825	2014-02-27	4969	9.488395	47147.836575	7.120
826	2014-02-28	4925	0.211628	1042.266000	5.390

827 rows × 5 columns

```
In [4]: weather_energy = weather_energy[['year_month_day','energy_sum_mean']].copy()
        weather_energy.set_index('year_month_day', inplace=True)
        weather_energy.head()
```

Out[4]:

year_month_day	energy_sum_mean
2011-11-23	6.952692
2011-11-24	8.536480
2011-11-25	9.499781
2011-11-26	10.267707
2011-11-27	10.850805

Figure 18: Data Importing

```
In [7]:
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(weather_energy)

In [8]:
train_size = int(len(scaled) * 0.70)
test_size = len(scaled - train_size)
train, test = scaled[0:train_size, :], scaled[train_size: len(scaled), :]
print('train: {}\ntest: {}'.format(len(train), len(test)))

train: 578
test: 249

In [15]:
def create_dataset(dataset, look_back=1):
    print(len(dataset), look_back)
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        print(i)
        print('X {} to {}'.format(i, i+look_back))
        print(a)
        print('Y {}'.format(i + look_back))
        print(dataset[i + look_back, 0])
        dataset[i + look_back, 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

In [16]:
look_back = 1
X_train, y_train = create_dataset(train, look_back)
X_test, y_test = create_dataset(test, look_back)
```

Figure 19: Data Preprocessing

LSTM Modelling - The code snippet present in Figure 20, describes the LSTM model implementation Procedure. The model summary has also be displayed.

Model Training

```
In [124]:
model = Sequential()
validation_split = 0.3
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
# fit network
history = model.fit(X_train, y_train, epochs=50, batch_size=15, validation_split=validation_split, verbose=1, shuffle=False)
model.summary()
403/403 [==============================] - 0s 200us/step - loss: 0.0026 - val_loss: 0.0018
Epoch 48/50
403/403 [==============================] - 0s 208us/step - loss: 0.0026 - val_loss: 0.0017
Epoch 49/50
403/403 [==============================] - 0s 208us/step - loss: 0.0025 - val_loss: 0.0017
Epoch 50/50
403/403 [==============================] - 0s 181us/step - loss: 0.0025 - val_loss: 0.0017
Model: "sequential_35"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_35 (LSTM)               (None, 50)                10400
_________________________________________________________________
dense_35 (Dense)             (None, 1)                 51
=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: LSTM Modelling

After model fit, the Loss and Validation Loss of the Model during the training phase has been plotted.

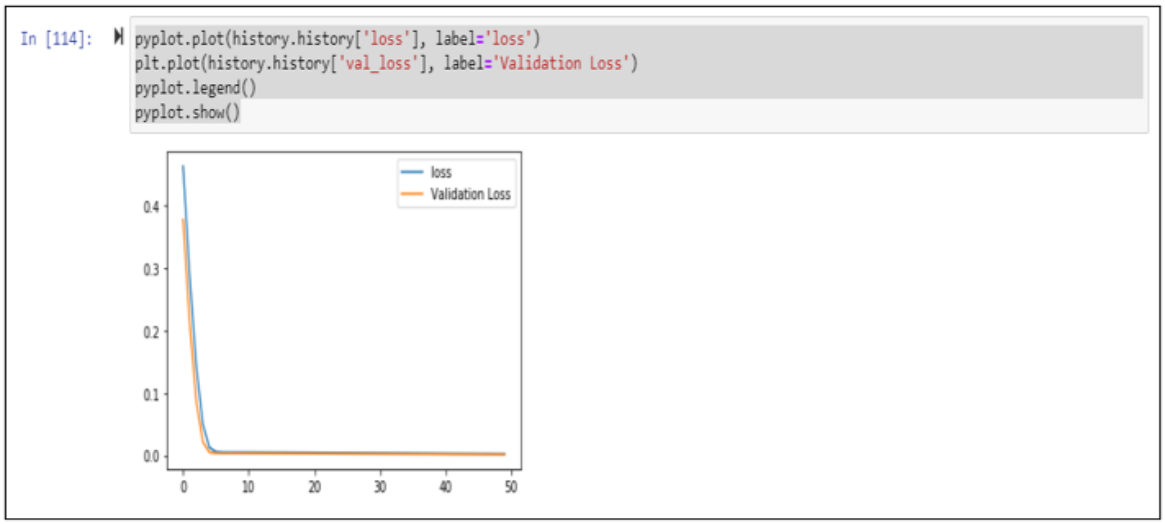

Figure 21: Plotting Loss and Validation Loss of the LSTM Model

The forecasting outcome are store into a data-frame in order to compare it with the real values. Finally the accuracy of the forecasting outcome is checked.

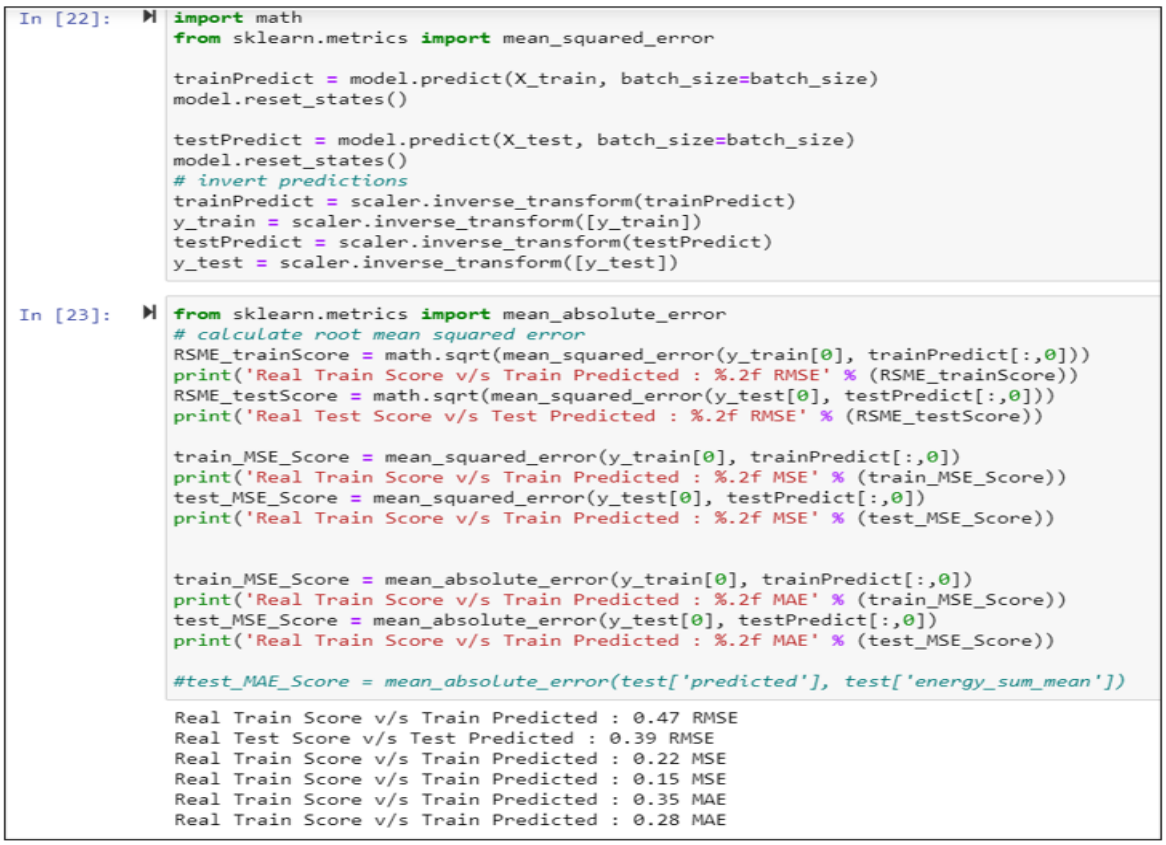

Figure 22: LSTM Model Accuracy Check