Prediction of Changes in Electrical Power Consumption in future with the help of ARIMA model, with other Machine and Deep Learning Model-Configuration Manual

MSc Research Project
Data Analytics

Syed Mohammad Shahrukh
Student ID: x20131674

School of Computing
National College of Ireland

Supervisor: Bharathi Chakravarthi

1. Introduction

Here, the configuration manual represents a brief overview of the device specification which has been used with a detailed explanation of the programming language which has been implemented to build the idea. Also, it explains libraries and the packages that have been implemented in the development of our topic :

Prediction of Changes in Electrical Power Consumption in future with the help of the ARIMA model.

This manual procedure will be showing how the data has been loaded, cleaned, and pre-processed, and then how it is implemented on the suitable models.

2. System Configuration

In this section the system configuration which has required for the implementation of the model.

2.1 Hardware Specification

For the implementation of the whole idea of the project, system configuration which is required in the respective processes is given in the Figure 1:

System	**RAM 8G**
Processor	**Intel I5**
Speed	**2.5 GHz**
Software	**Jupyter Notebook**
Programming Language	**Python 3**
Python libraries	**Python libraries**

Fig1

2.2 Software Specification

There are some programming tools used for the implementation of the idea with their different packages. For coding, the entire framework python language algorithm has been used, and the platform used to execute the idea is google colab.

2.3 python

The current version which has been used in the idea is 3.6.9 for the development of algorithmic structure.

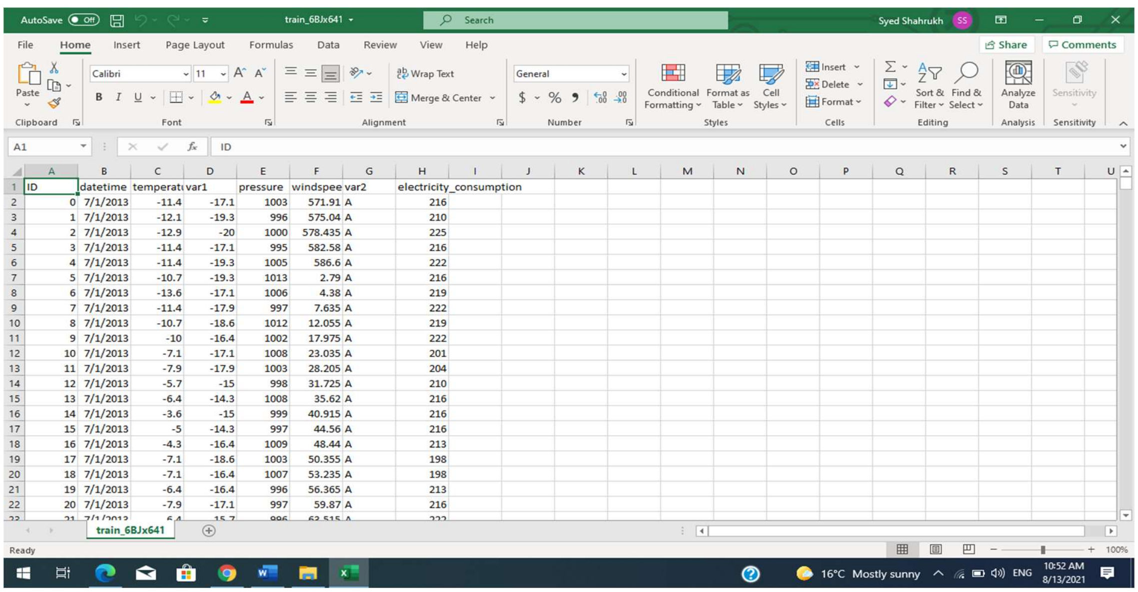
Libraries

1. Pandas — For handling structured data.
2. Numpy — For linear algebra and mathematics.
3. Keras - for development and evaluating deep learning models.
4. Tensorflow – is used for fast numerical computing.
5. Scikit Learn — For machine learning.
6. Pmdarima- For ARIMA model.
7. Seaborn — For data visualization.
8. Matplotlib- For data plotting and visualization.
9. Compare Models- For metrics plotting followed by their comparison.

3. Data Sources

3.1 Data set

We have selected our data set from Kaggle.com. The data set have 7 features that are: electrical consumption, date-time, pressure, windspeed, var1, var2 and temperature.



ID	datetime	temperat	var1	pressure	windspee	var2	electricity_consumption
0	7/1/2013	-11.4	-17.1	1003	571.91	A	216
1	7/1/2013	-12.1	-19.3	996	575.04	A	210
2	7/1/2013	-12.9	-20	1000	578.435	A	225
3	7/1/2013	-11.4	-17.1	995	582.58	A	216
4	7/1/2013	-11.4	-19.3	1005	586.6	A	222
5	7/1/2013	-10.7	-19.3	1013	2.79	A	216
6	7/1/2013	-13.6	-17.1	1006	4.38	A	219
7	7/1/2013	-11.4	-17.9	997	7.635	A	222
8	7/1/2013	-10.7	-18.6	1012	12.055	A	219
9	7/1/2013	-10	-16.4	1002	17.975	A	222
10	7/1/2013	-7.1	-17.1	1008	23.035	A	201
11	7/1/2013	-7.9	-17.9	1003	28.205	A	204
12	7/1/2013	-5.7	-15	998	31.725	A	210
13	7/1/2013	-6.4	-14.3	1008	35.62	A	216
14	7/1/2013	-3.6	-15	999	40.915	A	216
15	7/1/2013	-5	-14.3	997	44.56	A	216
16	7/1/2013	-4.3	-16.4	1009	48.44	A	213
17	7/1/2013	-7.1	-18.6	1003	50.355	A	198
18	7/1/2013	-7.1	-16.4	1007	53.235	A	198
19	7/1/2013	-6.4	-16.4	996	56.365	A	213
20	7/1/2013	-7.9	-17.1	997	59.87	A	216

Fig 2

https://pandas.pydata.org/
https://numpy.org/
https://keras.io/
https://www.tensorflow.org/
https://scikit-learn.org/stable/
https://pypi.org/project/pmdarima/
https://seaborn.pydata.org/
https://matplotlib.org/
https://pycaret.org/compare-models/

3.2 Consumption of energy according to changes in the weather

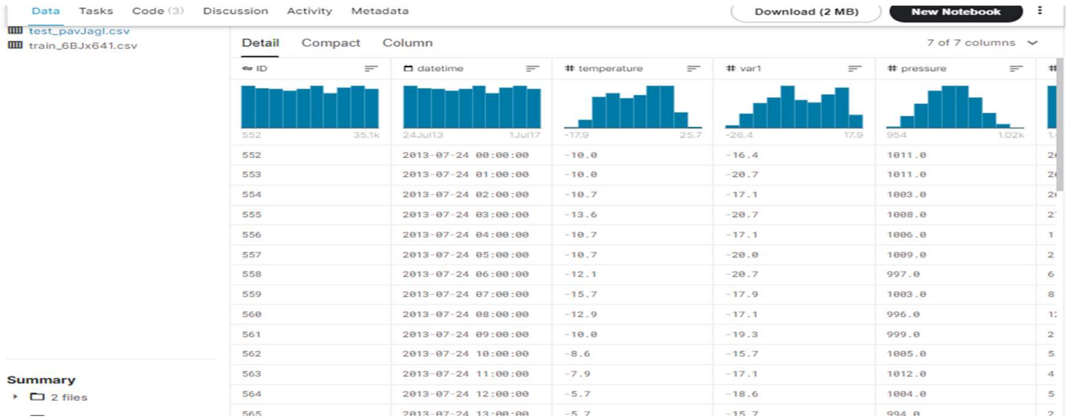
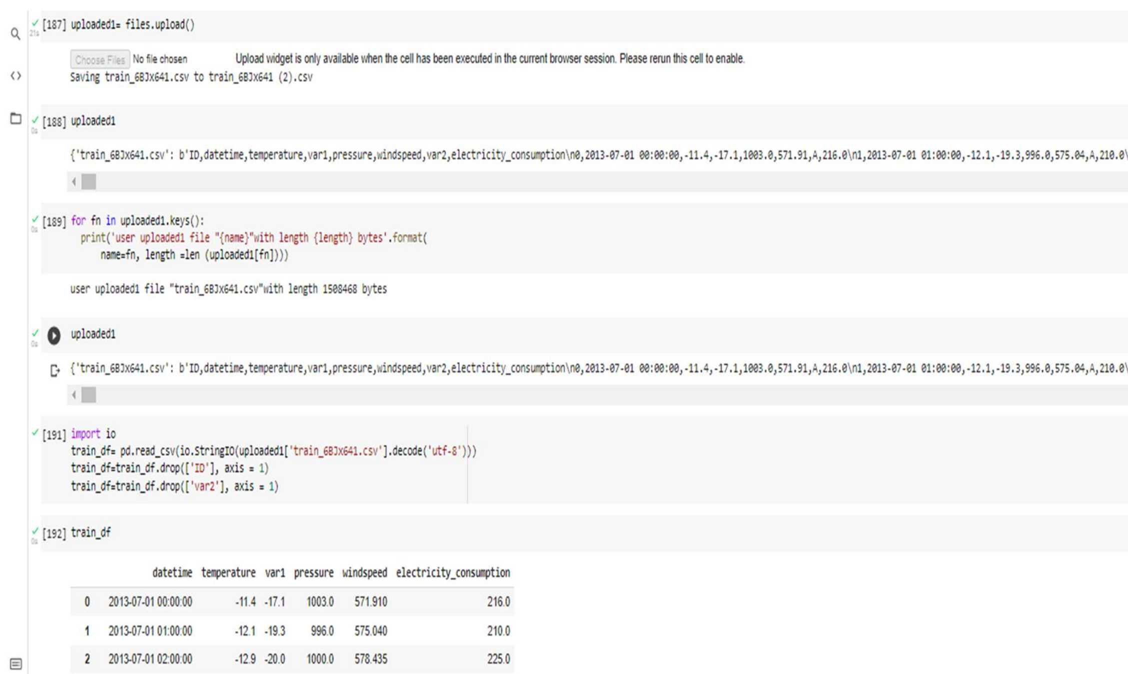


Fig 3

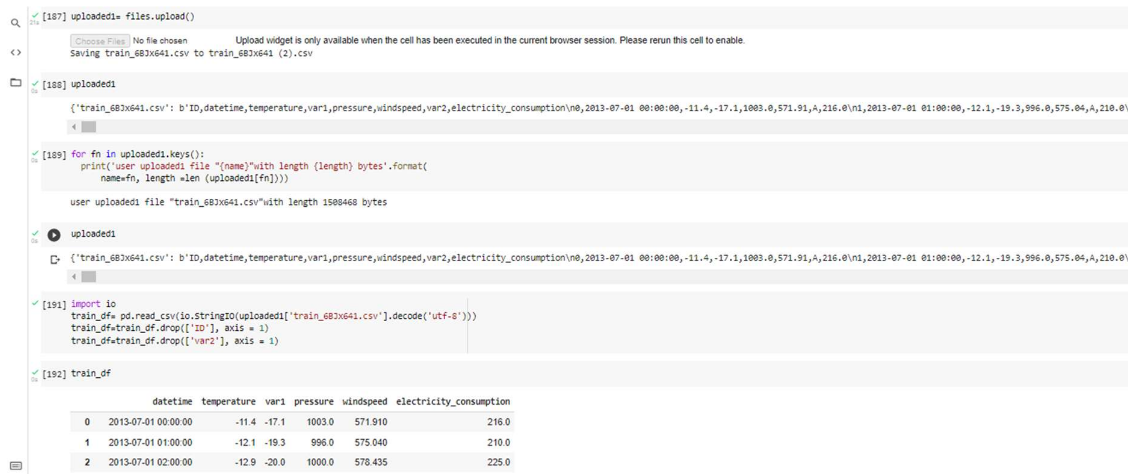
4.Project implementation.

After the selection of data set, the data set has been imported into the python environment on the google colab platform.



https://www.kaggle.com/ashoksrinivas/electrical-consumption

4.1 Selection of Working data set



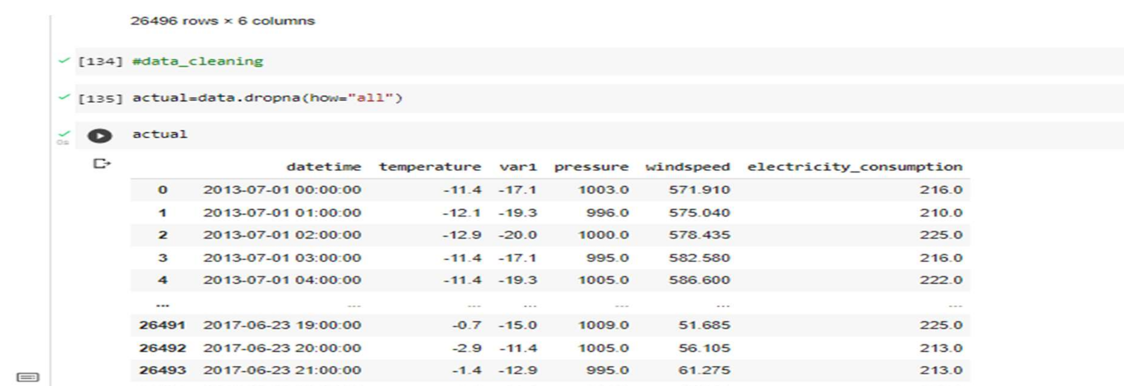
```
[187] uploaded1= files.upload()

Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving train_6BJx641.csv to train_6BJx641 (2).csv

[188] uploaded1

{'train_6BJx641.csv': b'ID,datetime,temperature,var1,pressure,windspeed,var2,electricity_consumption\n0,2013-07-01 00:00:00,-11.4,-17.1,1003.0,571.91,A,216.0\n1,2013-07-01 01:00:00,-12.1,-19.3,996.0,575.04,A,210.0\n

[189] for fn in uploaded1.keys():
        print('user uploaded1 file "{name}"with length {length} bytes'.format(
            name=fn, length =len (uploaded1[fn])))

user uploaded1 file "train_6BJx641.csv"with length 1508468 bytes

uploaded1

{'train_6BJx641.csv': b'ID,datetime,temperature,var1,pressure,windspeed,var2,electricity_consumption\n0,2013-07-01 00:00:00,-11.4,-17.1,1003.0,571.91,A,216.0\n1,2013-07-01 01:00:00,-12.1,-19.3,996.0,575.04,A,210.0\n

[191] import io
      train_df= pd.read_csv(io.StringIO(uploaded1['train_6BJx641.csv'].decode('utf-8')))
      train_df=train_df.drop(['ID'], axis = 1)
      train_df=train_df.drop(['var2'], axis = 1)

[192] train_df
```

	datetime	temperature	var1	pressure	windspeed	electricity_consumption
0	2013-07-01 00:00:00	-11.4	-17.1	1003.0	571.910	216.0
1	2013-07-01 01:00:00	-12.1	-19.3	996.0	575.040	210.0
2	2013-07-01 02:00:00	-12.9	-20.0	1000.0	578.435	225.0

Fig 4

Now, in the Figure 4 we can see that the data set has been imported on the platform and then it has been placed into a proper data frame.

4.2 Data Pre-Processing



```
26496 rows × 6 columns

[134] #data_cleaning

[135] actual=data.dropna(how="all")

actual
```

	datetime	temperature	var1	pressure	windspeed	electricity_consumption
0	2013-07-01 00:00:00	-11.4	-17.1	1003.0	571.910	216.0
1	2013-07-01 01:00:00	-12.1	-19.3	996.0	575.040	210.0
2	2013-07-01 02:00:00	-12.9	-20.0	1000.0	578.435	225.0
3	2013-07-01 03:00:00	-11.4	-17.1	995.0	582.580	216.0
4	2013-07-01 04:00:00	-11.4	-19.3	1005.0	586.600	222.0
...	...	...	...	...	...	...
26491	2017-06-23 19:00:00	-0.7	-15.0	1009.0	51.685	225.0
26492	2017-06-23 20:00:00	-2.9	-11.4	1005.0	56.105	213.0
26493	2017-06-23 21:00:00	-1.4	-12.9	995.0	61.275	213.0

Fig 5

Now, in the figure 5 we can see that we have dropped all NA values. Similarly, for ARIMA model implementation we have combined the test and train data set. Then we have found that there were large number of NA and NAN values present, that's why we have applied the data cleaning step for the implementation of the processes.

4.3 Data Transformation

```
[139] actual['datetime'] = pd.to_datetime(actual['datetime'])
```

```
[140] actual.set_index('datetime',inplace= True)
```

```
actual
```

datetime	temperature	var1	pressure	windspeed	electricity_consumption
2013-07-01 00:00:00	-11.4	-17.1	1003.0	571.910	216.0
2013-07-01 01:00:00	-12.1	-19.3	996.0	575.040	210.0
2013-07-01 02:00:00	-12.9	-20.0	1000.0	578.435	225.0
2013-07-01 03:00:00	-11.4	-17.1	995.0	582.580	216.0
2013-07-01 04:00:00	-11.4	-19.3	1005.0	586.600	222.0
...	...	...	...	...	...
2017-06-23 19:00:00	-0.7	-15.0	1009.0	51.685	225.0
2017-06-23 20:00:00	-2.9	-11.4	1005.0	56.105	213.0
2017-06-23 21:00:00	-1.4	-12.9	995.0	61.275	213.0
2017-06-23 22:00:00	-2.9	-11.4	996.0	67.210	210.0
2017-06-23 23:00:00	-2.1	-11.4	1009.0	71.880	210.0

26496 rows × 5 columns

Fig 6

Now, in this step we have transformed our data set, because in ARIMA model presence of time frame is necessary. So, we have converted normal date-time record into date-time format.

Now, for the implementation of deep learning model and other machine learning models, we have implemented and then we have performed the previous data transformation processes and then the following steps has been taken:

1) The normal data record has been converted into a list form (fig7).

```
datatime=[]
temperature=[]
var1=[]
pressure=[]
windspeed=[]
var2=[]
electricity_consumption=[]
total=[]
```

Fig 7

2) Then we have appended all the features list (fig 8).

```
import collections
print("[INFO] Data is getting processed----------------")
for i in tqdm(range((10000000))):
  pass
with open("train_6BJx641.csv", "r") as csv_file:
    csv_reader = csv.DictReader(csv_file, delimiter=',')
    for lines in tqdm(csv_reader):
      #print(lines['States'])
      datatime.append(lines['datetime'])
      temperature.append(lines['temperature'])
      var1.append(lines['var1'])
      pressure.append(lines['pressure'])
      windspeed.append(lines['windspeed'])
      var2.append(lines['var2'])
      electricity_consumption.append(lines['electricity_consumption'])
```

Fig 8

Then, for checking the stationarity of the data set, we have applied some test and EDA processes also. The result of EDA graphs are (fig 9):

```
#Electrical_Consumption data look
actual['electricity_consumption'].plot(figsize=(12,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1021b4f490>
```

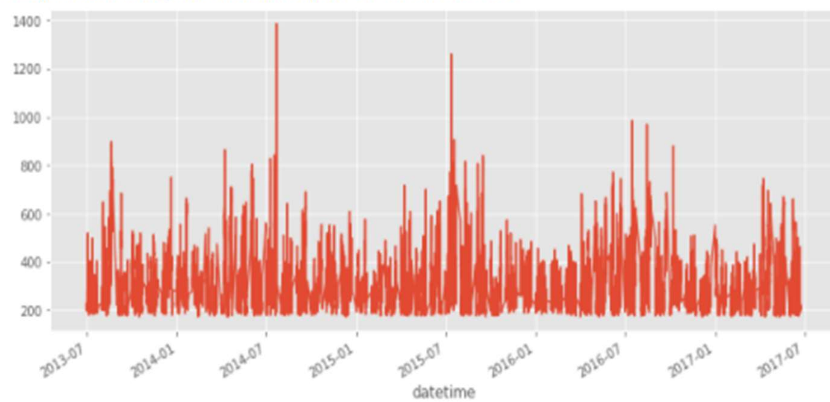


Fig 9(i) Electrical Consumption data record

```
#pressure data look
actual['pressure'].plot(figsize=(12,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0f9eb53710>
```

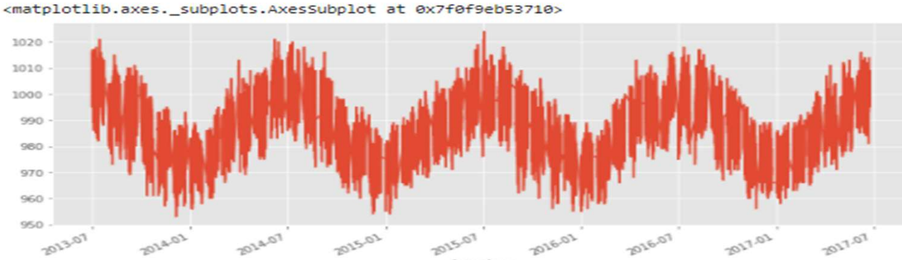


Fig 9(ii) Atmospheric pressure data record

4. Modelling

In this step we have applied certain number of models which can be proven helpful for predicting and forecasting the future values.

4.1 ARIMA and SARIMA model

```
#For pressure
stepwise_fit=auto_arima(actual['pressure'],trace=True,
                        suppress_warninns=True)

stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=174795.464, Time=11.08 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=188885.250, Time=0.83 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=181498.304, Time=1.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=174790.209, Time=5.77 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=188883.250, Time=0.34 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=174792.207, Time=9.01 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=174792.207, Time=8.52 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=174793.628, Time=7.07 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=174788.209, Time=1.77 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=174790.207, Time=2.84 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=174790.207, Time=2.53 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=181496.304, Time=0.42 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=174791.628, Time=2.88 sec

Best model:  ARIMA(0,1,1)(0,0,0)[0]
Total fit time: 54.137 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	26496
Model:	SARIMAX(0, 1, 1)	**Log Likelihood**	-87392.105
Date:	Sun, 11 Jul 2021	**AIC**	174788.209
Time:	10:35:30	**BIC**	174804.579
Sample:	0	**HQIC**	174793.493
	- 26496		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.8384	0.003	-255.910	0.000	-0.845	-0.832
sigma2	42.9040	0.489	87.755	0.000	41.946	43.862

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	774.29
Prob(Q):	0.98	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.96	**Skew:**	0.01
Prob(H) (two-sided):	0.09	**Kurtosis:**	2.16

Fig 10. Selecting and implementation of ARIMA and SARIMA model

Then, after the implementation of the model, we have got the forecasted values of electrical consumption.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0f9fab1fd0>
```

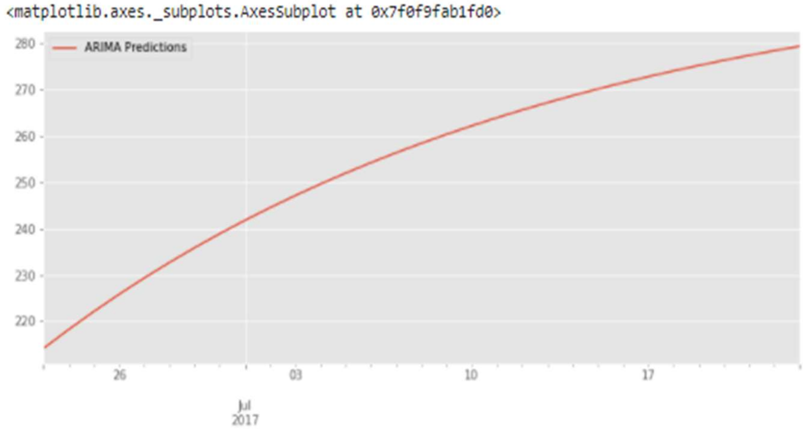


Fig 11. Electrical Consumption

So, after the implementation of the python algorithm we can see the forecasting of 2 months of electrical consumption. Fig(11)

4.2 LSTM Model

```
##########################################################################################LSTM model##########################################################################################
model = Sequential()
model.add(LSTM(512,input_shape=(X_train2.shape[1],X_train2.shape[2]),return_sequences=True))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add((LSTM(256,return_sequences=True)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add((LSTM(128,return_sequences=True)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add((LSTM(64)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(1)) #, activation='softmax'
print(model.summary())
from keras.optimizers import SGD,Adam
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True,clipvalue=0.5)
adam=Adam(clipvalue=0.5)

model.compile(loss='mean_squared_error', optimizer=sgd)

checkpoint = ModelCheckpoint("check.h5", monitor='loss', verbose=1, save_best_only=True, mode='min')


model.fit(X_train2, y_train,batch_size=64, epochs=1, verbose=1,validation_data=(X_test2,y_test))

y_pred=model.predict(X_test2)
```

Fig 12. Algorithm of LSTM model.

In fig(12) we can see the algorithm which has been implemented for LSTM model

4.3 BiLSTM Model.

```
##########################################################################################BiLSTM model##########################################################################################

model = Sequential()
model.add(LSTM(512,input_shape=(X_train2.shape[1],X_train2.shape[2]),return_sequences=True))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add(Bidirectional(LSTM(256,return_sequences=True)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(128,return_sequences=True)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(64)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(1)) #, activation='softmax'
print(model.summary())
from keras.optimizers import SGD,Adam
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True,clipvalue=0.5)
adam=Adam(clipvalue=0.5)

model.compile(loss='mean_squared_error', optimizer=sgd)

checkpoint = ModelCheckpoint("check.h5", monitor='loss', verbose=1, save_best_only=True, mode='min')
```

Fig 13. Algorithm of BiLSTM model

Now fig(13) we can see the implementation of algorithm for BiLSTM model

4.4 Linear Regression Model.

If we see the figure (14) , so it shows the algorithm of linear regression model and it also shows the evaluation parameters.

```
##########################################################################################linear regression model####################################################################################
from sklearn.linear_model import LinearRegression
lr = LinearRegression(n_jobs=-1)
lr.fit(X_train, y_train)
# Predicting the yield
y_pred = lr.predict(X_test)
print("y_pred",y_pred)


print("[INFO] Metrics calculation for LR(linear regression)  Starts--------------------------------------------------------------------------------------------")
print('Mean Absolute Error- LR:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:- - LR', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error: - LR', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("[INFO] Metrics calculation for LR(linear regression)  Ends---------------------------------------------------------------------------------------------")
sleep(4)
# Metrics
CompareModels.R2AndRMSE(y_test=y_test, y_pred=y_pred)
plot = CompareModels()
plot.add(model_name='Linear Regression', y_test=y_test, y_pred=y_pred)
plot.show(figsize=(10, 5))

print(" ")
print(" ")
print(" ")
print(" ")
```

Fig 14. Algorithm of Linear Regression Model.

4.5 Lasso Regression Model.

```
##########################################################################################Lasso model##########################################################################################

# Fitting training set to lasso regression model
from sklearn.linear_model import Lasso
ls = Lasso()
ls.fit(X_train, y_train)

# Predicting the yield
y_pred = lr.predict(X_test)
print("[INFO] Metrics calculation for Lasso Starts--------------------------------------------------------------------------------------------")
print('Mean Absolute Error- Lasso:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:- - Lasso', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error: - Lasso', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("[INFO] Metrics calculation for Lasso  Ends--------------------------------------------------------------------------------------------")
sleep(4)
CompareModels.R2AndRMSE(y_test=y_test, y_pred=y_pred)
plot.add('Lasso', y_test, y_pred)
plot.show(figsize=(10, 5))
print(" ")
print(" ")
print(" ")
print(" ")
```

Fig 15. Algorithm of Lasso Regression model

Similarly the fig (15) is representing the Lasso regression model and their evaluation parameter.

4.6 KNN Model

```
##########################################################################################KNN model##########################################################################################

# Fitting KNN model to the dataset
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(metric='minkowski', n_neighbors=5, n_jobs=-1)
knr.fit(X_train, y_train)

# Predicting the yield
y_pred = knr.predict(X_test)
print("[INFO] Metrics calculation for KNN Starts--------------------------------------------------------------------------------------------")
print('Mean Absolute Error- KNN:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:- - KNN', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error: - KNN', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("[INFO] Metrics calculation for KNN  Ends--------------------------------------------------------------------------------------------")
sleep(4)
CompareModels.R2AndRMSE(y_test=y_test, y_pred=y_pred)
plot.add('KNN', y_test, y_pred)
plot.show(figsize=(10, 5))
print(" ")
print(" ")
print(" ")
print(" ")
```

Fig16. KNN Model

Where in figure(16) the KNN model is representing and the evaluation parameters which shows the performance of the model.

4.7 Random Forest Model

```
###########################################################################################Random Forest model###########################################################################################
# Fitting Random Forest model to the dataset
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators=10, random_state=10, n_jobs=-1)
rfr.fit(X_train, y_train)
# Predicting the yield
y_pred = rfr.predict(X_test)
#print("y_pred",y_pred)



print("[INFO] Metrics calculation for Random Forest  Starts-----------------------------------------------------------------------------------------------------------")
print('Mean Absolute Error- Random Forest:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:- - Random Forest', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error: - Random Forest', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("[INFO] Metrics calculation for Random Forest  Ends-------------------------------------------------------------------------------------------------------------")
print(" ")
print(" ")
print(" ")
print(" ")
print("START TESTING OF THE MODEL ON TEST DATA__________________________________________________________________________________________________________"
sleep(4)
```

Fig 17 Algorithm of Random Forest Model

In the fig (17) we can see the implementation of another machine learning model (Random Forest) and their evaluation parameters.

4.8 Performance comparison of machine learning models.

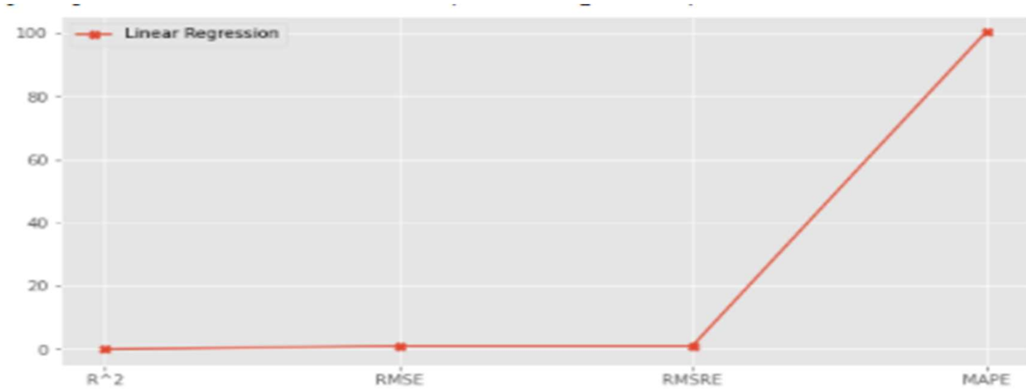


Fig 18

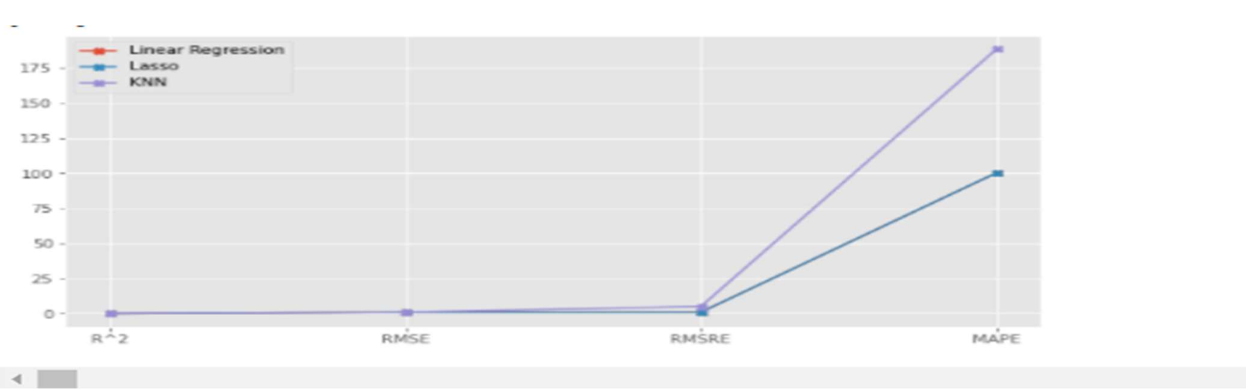


Fig 19

Now, in the fig (18) and (19) we can compare and see that which machine learning model has better performance in terms of predicting the targeted value of electrical consumption after the testing of the model.