Automobile Charging Station Load Analysis and Prediction

Professor: Tariq Zeenat

Group 5
Vishnu Sudireddy
Sreekar Thanda
Nagateja Ravilla
Sai Vishwak Jadala

Contents

Chapter 1

1.1 Motivation:

Regularly balancing demand and supply of power is a major task for the utilities sector, and as the number of electric Automobiles sales rises, managing the daily load is becoming more difficult. Making precise decisions about things like manual regulation of voltage and load switching is challenging and leads to loss of electricity while switching and transmitting. The solution could be proposed using Data analysis and Data analytics techniques and algorithms which will assist in predicting the load for automobiles charging a day in advance, and decisions on power generation and load balance may be made based on forecasted data. Hence, this project is designed to forecast, analyze and predict the load required for automobile charge stations that predicts the required amount of load/energy on daily basis.

1.2 Objectives:

In this model we are building a prediction model using time series analysis to extract meaningful statistical data that will help to identify the Automobiles/Electric vehicle load demand in advance for the coming day and week ahead. This prediction data will help to make decisions and manage the operations efficiently with low operational costs and help in reducing electric transmission power losses.

1.3 Significance:

As the world is transforming towards Electric vehicles, the load of energy required to charge the Electric vehicles at charging stations is also high in demand all over the world. Our project is based on estimating the demand of load at the charging stations that is required for day-to-day and weekly basis. This idea and implementation could play a significant role in adjusting the need of electric energy at charging stations. The number of automobiles on the road may rise to 77 million by December 2025 and 229 million by December 2030, based on Bloomberg New Energy Finance Economic Transition Scenario. So, there is a need to estimate the future need for energy which is required to cope with demand. Hence, this project is important to real world estimation of need of energy load with respect to rise in Electric Mobilities. Another purpose of this project is to help the energy suppliers supply only the required amount of power efficiently, so that the unnecessary loss of energy could be reduced, and energy could be conserved, customers could get satisfied, and business could run smooth.

Chapter 2

2.1 Introduction:

Understanding how much electricity automobile charging stations use a crucial area of research is. This is known as load analysis and prediction for charging stations. The need for charging stations is increasing as more automobiles are put on the road. This in turn is increasing the burden on the electrical system, necessitating a thorough examination of the charging station load. The aim of load analysis and prediction for charging stations is to create models that can anticipate the energy consumption patterns of the charging stations with accuracy. The grid operators can better control the electricity supply by using these models to estimate the power demand at charging stations. In order to lower the cost of charging and lessen the demand on the power grid, the analysis also tries to optimize the energy usage of the charging station. The shift to a more sustainable and effective transportation system depends on accurate load analysis and forecasting at automotive charging stations. In order to understand charging station behavior, it makes use of cutting-edge machine learning algorithms and data analysis methodologies, which will ultimately result in better charging station infrastructure and more effective use of energy resources.

2.2 Background:

Due to high CO2 emissions from existing use high use of petroleum, the world has been transforming into use of green energy which cloud decrease the pollution. For that reason, everyone is switching to use to electric energy generated from different forms. As a result, the use of Electric power machines especially automobiles are high. Hence, there is high increase in requirement of Charging stations to these machines. Though demand for EV charging stations has grown considerably, predicting their load has become crucial for utilities and ensure its stability and reliability.

Charging stations require sufficient power to operate and so they need to maintain load analysis. There are some issues if the load analysis is not properly maintained. Some of them were, Overloading, Inefficient Charging, Safety Hazards, Equipment Damage.

The project typically involves collecting data from sources of Colorado charging station location data, applying advanced data analytics techniques to analyze and predict the future charging station load. The primary objective of this project is to predict the charging station load accurately, based on historical data. The analysis can help identify trends, patterns, and anomalies in the charging station load, such as peak usage hours and seasonality based on daily wise. It can also help in identifying the location and capacity of new charging stations and improving existing charging infrastructure.

2.3 Related Work:

The study and prediction of the load at automotive charging stations has been the subject of extensive research and development. Models that may precisely predict the energy consumption patterns of the charging station have been developed through studies using machine learning algorithms and data analysis approaches. These models could optimize the energy consumption of the charging station, lower the cost of charging, and lessen the demand on the power grid. The Charge4Cobi project, which seeks to improve the management of automobile charging stations, and the EPRI study on automobile charging load profiles and their effects on the power grid are two noteworthy initiatives in this area.

Reference links:

[1] the webpage for the "Charge4Cobi - Charge for COBility intelligence" initiative.

Website address: https://charge4cobi.eu

[2] A. Zaidi and colleagues published "A Machine Learning Approach to Forecasting Automobile Charging Demand" in science.

Refer to this URL: https://www.sciencedirect.com/science/article/pii/S1876610219301574

Chapter 3

3.1 Dataset:

The dataset used for this project is taken from Colorado state government's website (https://open-data.bouldercolorado.gov/). Data is download in CSV format. This data is related to all the charging stations that are present in the Boulder city in Colorado state. Our Data set is the latest data generated from different charging station locations across Colorado State from 1/20/2018 to 8/1/2022. The Dataset has columns:

Station_Name: Name of the charging station. This column intakes only String Data type.

Address: Address of the station. This column intakes only String Data type.

City: City of the charging station. This column intakes only String Data type.

State_Province: Name of the state the charging station is present. This column intakes only String Data type.

Start_Time_Zone: The time zone of the charging station. This column intakes only String Data type.

Start_Date_Time: The time at which user has started charging the vehicle. These are of Timestamp value.

End_Date_Time: The time at which user has started charging the vehicle. These are of Timestamp value.

End_Time_Zone: The time zone of the charging station. This column intakes only String Data type.

Day: The day of the charging date. This column intakes only String Data type.

Total_Duration: The total duration of vehicle in the charging station. This column intakes only String Data type.

Charging_Time: The duration for which the vehicle is connected to the charging port. This column intakes only String Data type.

Energy_kWh: The amount of energy consumed by the vehicle in charging the battery. This column intakes only String Data type.

GHG_Savings_kg: The amount of fuel saved by using electricity instead of fuel in kilograms. This column intakes only String Data type.

Gasoline_Savings_gallons: The amount of fuel saved by using electricity instead of fuel in gallons. This column intakes only String Data type.

Port_Type: This has the information about the type of the charging port. This column intakes only String Data type.

ObjectId: This is the unique id given to the data. This column intakes only String Data type.

Our data set consists of 43659 rows which makes our data set a complex data to understand. This data is stored as a csv file to make use in all applications. Though the dataset has these many columns, we have used the columns Energy_kWh and End_Date_Time in training and testing the model. These columns have string values, which is not the correct datatype of those columns. Energy_kWh should be a float type and the End_Date_Time should be in DateTime datatype. We have changed the datatype of those columns using inbuilt methods in python.

3.2 Features:

Our project is designed such that it analyses the load for a changing station that went through since last year (2018 to 2022). The use of Python Libraries NumPy and pandas used for Data analysis gives out efficient data processing, which can help for better understanding and analyzing of data and can improve business operations. Other features such as modelling techniques for predictions can help to predict the next coming day wise requirement of electricity load.

Some of the important features of our project are:

Data Gathering: Gather data from a variety source including grid systems, charging stations, Weather forecasts.

Data Cleaning: The gathered data is then cleaned by removing the null values and blanks. The datatypes of the columns are corrected. After cleaning the data to get rid of any missing values, combine data from different sources into a single data set.

Analysis of Data: Do exploratory data analysis using Python modules like Pandas and NumPy to draw conclusions from the data such that Day wise consumption of energy of every charging station can be viewed.

Model selection: Choose the one that best fits the data, then train it. ARIMA, LSTM, XG Boost and Random Forest are a few common models.

Cross-validation techniques could be used to validate the model. This will help to ensure that the model is accurate. The results of the models are examined by plotting a actual data and the predicted data on a single graph.

3.3 STEPS in Workflow:

The steps that will be performed as per the project are:

1. Data Gathering
2. Data Aggregation
3. Outlier Remover
4. Finding Missing Data Using Linear Interpolation
5. Trend Graphs and Analysis
6. Machine Learning Model fitting
7. Evaluation of model
8. Results

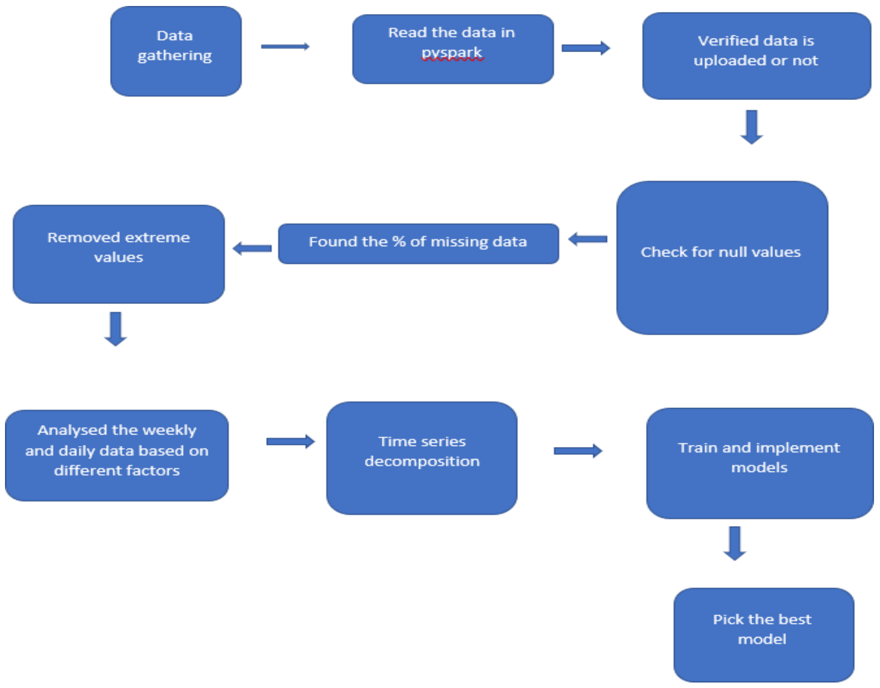

Data gathering
Read the data in pyspark
Verified data is uploaded or not
Check for null values
Found the % of missing data
Removed extreme values
Analysed the weekly and daily data based on different factors
Time series decomposition
Train and implement models
Pick the best model

Figure: Basic workflow steps

Data gathering is collecting all the information from the previous data which contains load consumptions at charging stations and the station name, the total duration taken for a automobile to charge, Load/Energy, starting time and ending time respectively.

Data Cleaning is removal of unwanted data i.e, null values, negative values, missing values since it must not affect the prediction.

Seasonal decomposition occurs by checking the following trends

- Seasonal
- Resid
- Trend

Analyzing the models by using Arima, LSTM, XG Boost Regressor.

3.4 Hardware Specifications:

- CPU: For speedier processing, a multicore processor such the Intel Core i5 or i7 or AMD Ryzen 5 or 7 is advised.
- Memory: It's advised to have at least 8GB of RAM when dealing with huge datasets.
- Storage: Enough room in which to keep the software tools and the dataset.
- GPU: A GPU equipped with CUDA cores helps speed up the training of deep learning models.

3.5 Software Specifications:

3.5.1 HIVE:

- SQL-like Interface: Hive is a data warehouse system built on top of Hadoop. It provides a SQL-like interface to query the data stored in Hadoop.
- HiveQL: Hive uses HiveQL, a SQL-like language, to query and process data.
- Compatibility: Hive is compatible with Hadoop and can be used with other Hadoop components.
- Data Formats: Hive supports various data formats, including textfile, CSV and RCFile etc.
- Partitioning: Hive supports data partitioning allowing data to be stored in a partitioned format for faster querying.
- User-Defined Functions: Hive supports user-defined functions (UDFs) that can be used to extend HiveQL and perform custom data processing.

3.5.2 Python

- To run the libraries NumPy and Pandas, a minimum python version of 3.7 is required. NumPy will need a minimum version of python 3.6.
- NumPy supports multi-dimensional arrays and it has a methods to compute mathematical expressions that are necessary to build and run machine learning and deep learning models. NumPy is a library that is built using both C and Python languages. So, models built using NumPy can be executed from either command line interface or a python shell.
- Pandas is a library that is built on top of NumPy. This library is used for data analysis and data manipulation. This will help in easy creation of data frames and manipulation of data frames and RDDs.
- To run NumPy and pandas the needs a C compiler.

3.5.3 PySpark:

- Pyspark is a Python API Hence, Python Programming language is used.
- Compatibility: PySpark is compatible with Python 2.7 and Python 3.
- Distributed Processing: PySpark is built on top of Apache Spark, which is a distributed processing framework, allowing it to perform parallel processing of data.

- Libraries: PySpark consists of multiple inbuilt libraries which support data processing and machine learning tasks, including NumPy, pandas, Scikit-learn, and TensorFlow.
- APIs: PySpark provides APIs in order to work with Spark's distributed data processing, including RDD (Resilient Distributed Datasets), DataFrame, and Dataset APIs.

Chapter 4

4.1 Model:

The dataset has the charging information of a vehicle at charging station. The data has date, charging duration and energy consumed by vehicle for charging the batteries. Form this data, a timeseries machine learning or deep learning models needs to be selected to a predict the energy required by the charging station. We have identified three such models that me predict the energy consumption with less loss and more accuracy. They are Arima model, LSTM model and XgBoost.

4.2 ARIMA Model:

ARIMA refers to Auto-Regressive Integrated Moving Average. This model is widely used in many industries for predicting the future values by using the timeseries data. ARIMA models needs stationary data and so it uses differencing technique to convert no stationary data to stationary data. The residual errors in the data are handled in this model by calculating the moving averages and finding the auto correlations. The main reason for choosing Arima as one of the models for predicting the future load in that it can handle nonstationary data and the model is widely used for time series forecasting. This model uses a range of previous data to predict the current time's result.

Formula to predict current value: $\mathbf{m_t = {}_0 + {}_1m_{t-1} + {}_2m_{t-2} + {}_3m_{t-3} + \ldots + {}_pm_{t-p}}$

4.2.1 Architecture and Workflow:

For Arima model the data needs to be a stationary data. Before providing the data to the model, we need to check if the data is stationary or not. That can be identified either by using the graph plot or numerical computations. The first block in the flow chart provided below is 'Plot the time Series'. Then the data is tested if the provided data is stationary or not. This is identified by performing ad-fuller test in python. If the result of ad-fuller test say that the data is not stationary, then the data needs to be transformed to stationary by calculating the moving average of the non-stationary data and then subtracting the data with the moving average will transform the non-stationary data to stationary data.

The transformed data is again checked for stationary and of the data is a stationary data then we are good to build Arima model. On providing the data to the model, it will run various iterations and allocate itself with the best model for the provided input. After fitting the model with the data, now we are good to predict the future values by providing the test data to the model. Then we can compare the predictions with the actual data and can draw conclusions if the model is performing well or not. The performance of the model can be drawn by calculating the various performance metrics like accuracy, absolute percentage error and root mean square error.

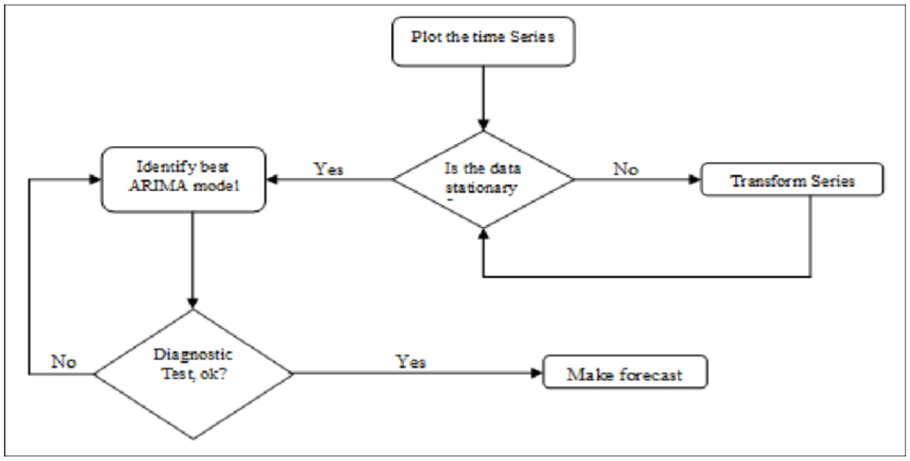


Figure: Arima model workflow

4.3 LSTM Model

LSTM refers to Long Short-Term Memory Networks. This is a deep learning model that comes under Recurrent Neural Networks (RNN). The help in predicting the future data using the past data. LSTM can be used not only in the field of time series forecasting but it can be also used in speech recognition and natural language processing.

4.3.1 Architecture of LSTM

Just like Recurrent Neural Networks, LSTM also have a set of memory cells that store the data for a short time. But LSTM overcomes the problem of recurrent neural networks that they lose the stored data if the storage space is full. The deleted data is replaced by new data. Whereas LSTM

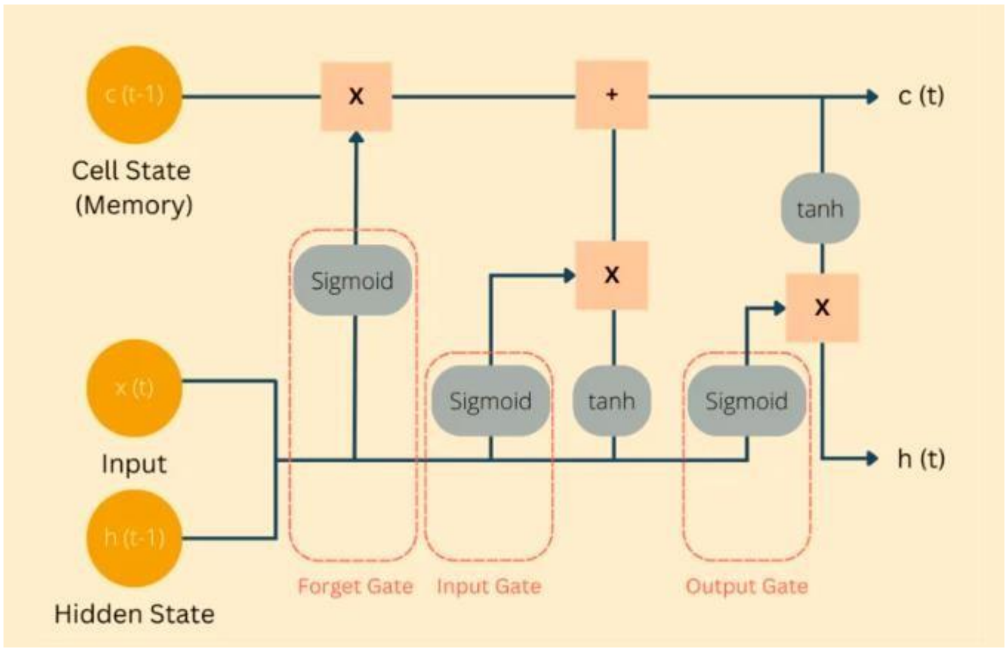


Figure: Architecture diagram of LSTM Model

will eliminate this problem by storing the information that may help in future for predicting the data in long term memory. This long-term memory is stored in cell state.

The forget gate in the architecture will receive the information from current input and the precious saved state. This gate decides if the data is still needs to be stored in the long-term memory. The output of this gate will be a zero or one. Zero means that the data is no longer need and the previous data can be forgotten as there may be a possibility of new data.

In the input gate, the information that seems important is multiplied by the hidden state value and the weighted matrix. The result is then added to the memory, cell state. The new cell state will replace the old cell state and the old cell state will no longer be available. The new cell state will be used wherever required.

In the output gate, the output of the model is calculated in the hidden state. The data that can be used for the calculation is decided by the sigmoid block in the architecture diagram. The output of the output gate is then multiplied by the cell state data after the output is activated by tanh function.

4.3.2 Workflow of LSTM

The data in the dataset is modified into the required format. The data types of the necessary columns are altered in the feature extraction block. The dataset then split in the desired ration for testing and training purposes. The training data is fed to the model and the model is constructed

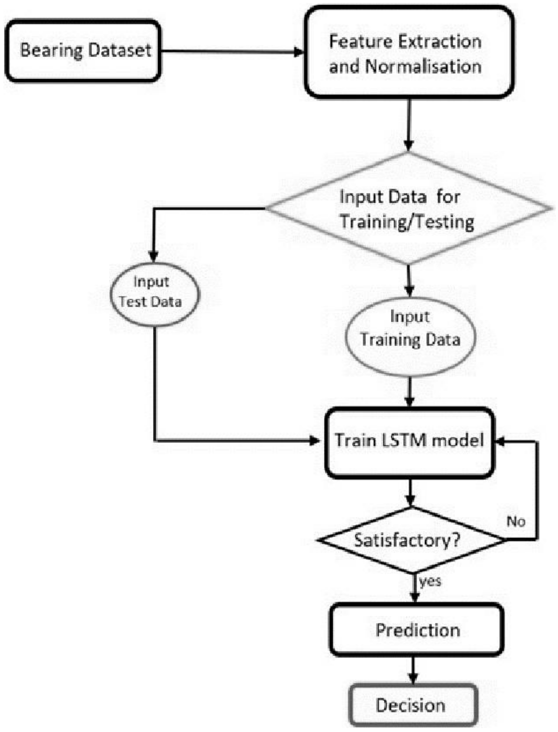


Figure: Workflow diagram of LSTM model

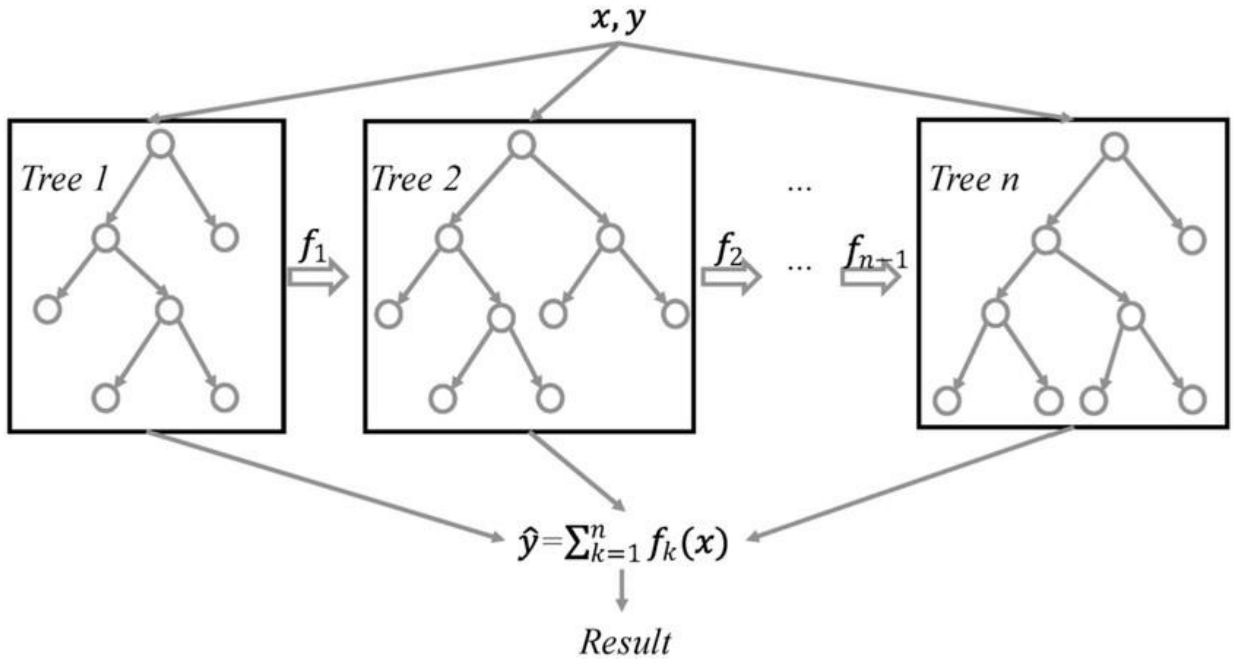
with the training data. The constructed model is then evaluated with the training data. If the result seems satisfactory then the predictions can be drawn. If the result seems unsatisfactory, the model is trained again by changing the optimizer or the epoch of the model. This process is repeated till the desired outputs are obtained. After the predictions of the model, the decisions can be made based on the predicted outputs.

4.4 XgBoost

XGBoost stands for Extreme Gradient Boosting. It is the leading model for regression, classification and ranking problems. It is widely used in various industries. XgBoost is built on top of decision trees, gradient boosting and ensemble learning. This model works by identifying patterns the dataset. The main reason for choosing this model is that this has been widely used for solving the problems in Kaggle competition as this model has the capability of producing highly accurate results. XG Boost has gained huge popularity in recent times in very small span.

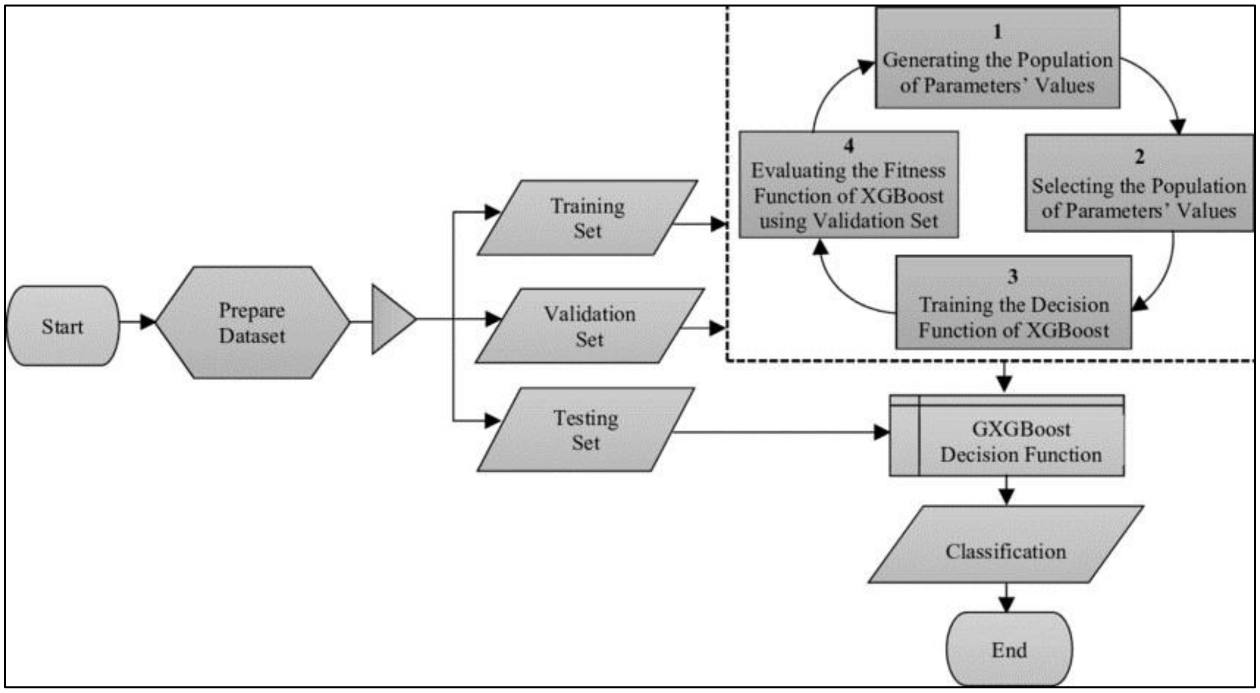
4.4.1 Architecture:

As discussed earlier that the XgBoost model is built on top of the decision tress. The data provided to the model is split into tiny subsets and various algorithms are applied on each subset. The results are the analyzed by find the difference between the results. Each model applied on the subsets will use different techniques to build the full decision tree. The result of model is the average of the results that are obtained from each subset. Here in the below figure x represents the variable that need to be predicted and y is the label of that value. Label helps to identify the value uniquely.



4.4.2 Workflow:

The data is first prepared by creating an index to identify each row uniquely. Date dimension table is created for the dataset and is appended to the dataset. The date dimension table will have the columns date, hour, day of week, quarter, month, year, week of year. Then the data set is split into training set, validation set, and testing set. The training set and the validation set are provided to the model and the testing set is kept aside for testing the model after it is fitted with the training data. The training data is converted into small subsets and the decision trees, random forest and various tree algorithms are applied on the subsets. The result of all the subsets is averaged and a result is obtained. The result is given to the decision function along with the testing data. The predicted data is compared with the actual data the conclusions are drawn for the model weather the model is the best model or not.

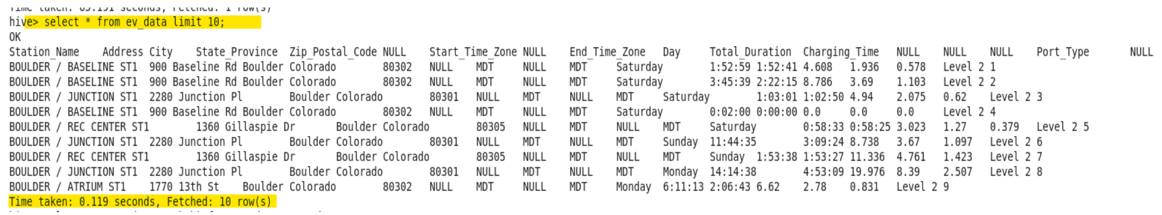

Start
Prepare Dataset
Training Set
Validation Set
Testing Set
1 Generating the Population of Parameters' Values
2 Selecting the Population of Parameters' Values
3 Training the Decision Function of XGBoost
4 Evaluating the Fitness Function of XGBoost using Validation Set
GXGBoost Decision Function
Classification
End

Chapter 5

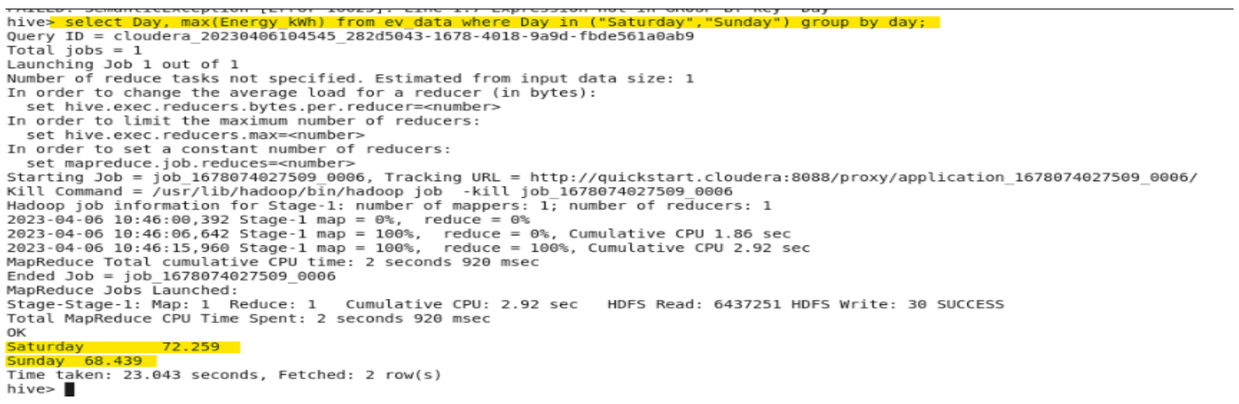
5.1 Analysis:

5.1.1. Data Gathering:

Data Gathering is a key process in any project. Unless the data is properly gathered, further steps cannot be processed properly. The data set that is considered is a CSV file and as it has 43659 rows, it would be hard and not enough space to display all the rows, Hence, Initially the data is loaded into Hive and displayed top 10 rows in the csv file. Here the fields could be seen, and the columns could be extracted. Here the time taken to display all the top 10 columns is 0.119 seconds. As the data is loaded into Hive Platform, found the top energy consumption occurred on Saturday and Sunday. As per result, On Saturday, the highest energy consumed is 72.259kWh and on Sunday it is 68.439kWh. But to process this, it took around 23+ seconds which is very difficult and huge time in real time scenario.



```
hive> select * from ev_data limit 10;
OK
Station_Name    Address City    State_Province  Zip_Postal_Code NULL    Start_Time_Zone NULL    End_Time_Zone   Day     Total_Duration  Charging_Time   NULL    NULL    NULL    Port_Type       NULL
BOULDER / BASELINE ST1  900 Baseline Rd Boulder Colorado        80302   NULL    MDT     NULL    MDT     Saturday        1:52:59 1:52:41 4.608   1.936   0.578   Level 2 1
BOULDER / BASELINE ST1  900 Baseline Rd Boulder Colorado        80302   NULL    MDT     NULL    MDT     Saturday        3:45:39 2:22:15 8.786   3.69    1.103   Level 2 2
BOULDER / JUNCTION ST1  2280 Junction Pl        Boulder Colorado        80301   NULL    MDT     NULL    MDT     Saturday        1:03:01 1:02:50 4.94    2.075   0.62    Level 2 3
BOULDER / BASELINE ST1  900 Baseline Rd Boulder Colorado        80302   NULL    MDT     NULL    MDT     Saturday        0:02:00 0:00:00 0.0     0.0     0.0     Level 2 4
BOULDER / REC CENTER ST1        1360 Gillaspie Dr       Boulder Colorado        80305   NULL    MDT     NULL    MDT     Saturday        0:58:33 0:58:25 3.023   1.27    0.379   Level 2 5
BOULDER / JUNCTION ST1  2280 Junction Pl        Boulder Colorado        80301   NULL    MDT     NULL    MDT     Sunday  11:44:35        3:09:24 8.738   3.67    1.097   Level 2 6
BOULDER / REC CENTER ST1        1360 Gillaspie Dr       Boulder Colorado        80305   NULL    MDT     NULL    MDT     Sunday  1:53:38 1:53:27 11.336  4.761   1.423   Level 2 7
BOULDER / JUNCTION ST1  2280 Junction Pl        Boulder Colorado        80301   NULL    MDT     NULL    MDT     Monday  14:14:38        4:53:09 19.976  8.39    2.507   Level 2 8
BOULDER / ATRIUM ST1    1770 13th St    Boulder Colorado        80302   NULL    MDT     NULL    MDT     Monday  6:11:13 2:06:43 6.62    2.78    0.831   Level 2 9
Time taken: 0.119 seconds, Fetched: 10 row(s)
```

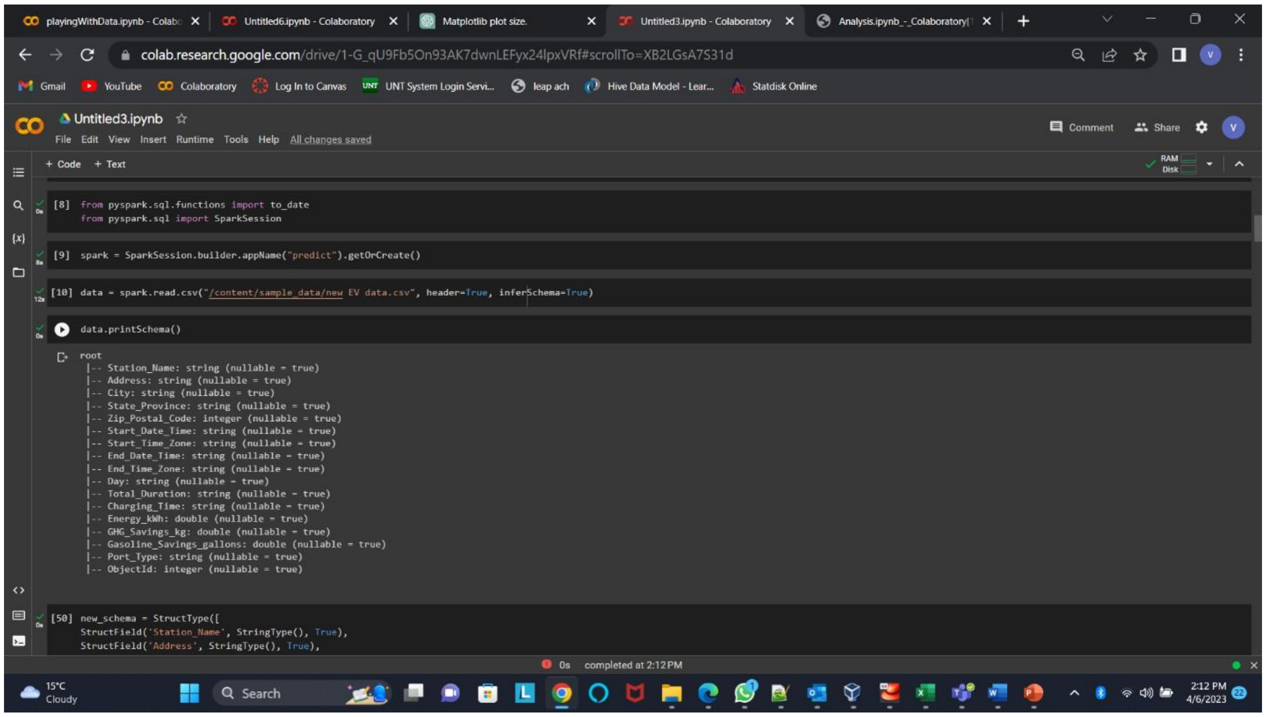


```
hive> select Day, max(Energy_kWh) from ev_data where Day in ("Saturday","Sunday") group by day;
Query ID = cloudera_20230406104545_282d5043-1678-4018-9a9d-fbde561a0ab9
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1678074027509_0006, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1678074027509_0006/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1678074027509_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-04-06 10:46:00,392 Stage-1 map = 0%,  reduce = 0%
2023-04-06 10:46:06,642 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.86 sec
2023-04-06 10:46:15,960 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.92 sec
MapReduce Total cumulative CPU time: 2 seconds 920 msec
Ended Job = job_1678074027509_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.92 sec   HDFS Read: 6437251 HDFS Write: 30 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 920 msec
OK
Saturday        72.259
Sunday  68.439
Time taken: 23.043 seconds, Fetched: 2 row(s)
hive> █
```

As Hive is taking longer duration for processing and hive is costly, the project is built using python. Python has many libraries that are necessary for performing computing complex calculations. Python also has large community support they help the developer in find the right solution for right situation and in fixing the potential problems that me arise due to deprecated dependencies.

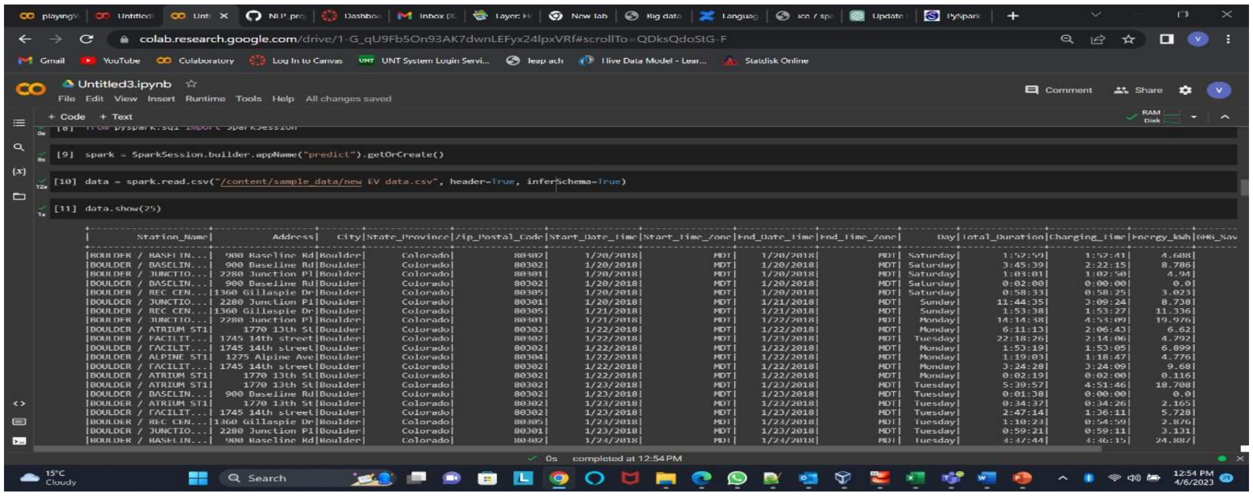
The basis analysis part is done using PySpark. Pyspark is fast processing tool compared to hive and other tools and is easily compatible with multiple systems.

The data frame and RDDs are created. By default, pyspark will consider the "string data type" But, that could give wrong results and might take much time for processing, Hence, the data type of energy is converted to double since it has numerical values.



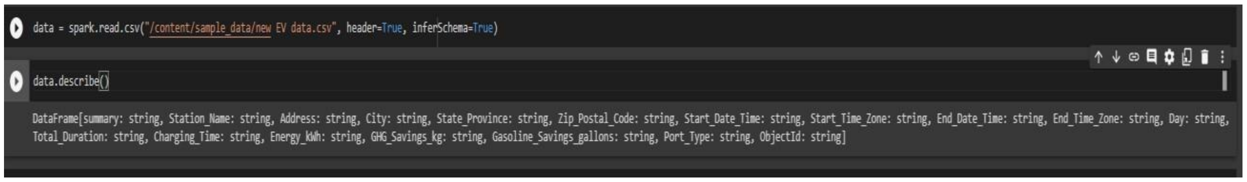
```
[8] from pyspark.sql.functions import to_date
    from pyspark.sql import SparkSession

[9] spark = SparkSession.builder.appName("predict").getOrCreate()

[10] data = spark.read.csv("/content/sample_data/new EV data.csv", header=True, inferSchema=True)

data.printSchema()

root
 |-- Station_Name: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State_Province: string (nullable = true)
 |-- Zip_Postal_Code: integer (nullable = true)
 |-- Start_Date_Time: string (nullable = true)
 |-- Start_Time_Zone: string (nullable = true)
 |-- End_Date_Time: string (nullable = true)
 |-- End_Time_Zone: string (nullable = true)
 |-- Day: string (nullable = true)
 |-- Total_Duration: string (nullable = true)
 |-- Charging_Time: string (nullable = true)
 |-- Energy_kWh: double (nullable = true)
 |-- GHG_Savings_kg: double (nullable = true)
 |-- Gasoline_Savings_gallons: double (nullable = true)
 |-- Port_Type: string (nullable = true)
 |-- ObjectId: integer (nullable = true)

[50] new_schema = StructType([
     StructField('Station_Name', StringType(), True),
     StructField('Address', StringType(), True),
```

The dataset is loaded and displayed which just took 12 to 15 seconds for execution. The snippet, data.show(25) will fetch 25 rows from the dataset and is shown to the user.



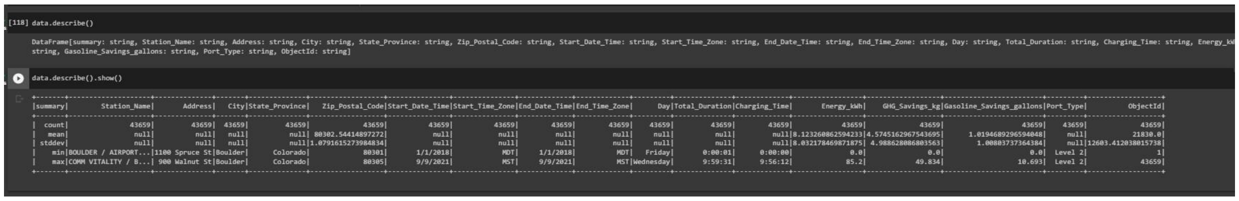
```
[9] spark = SparkSession.builder.appName("predict").getOrCreate()

[10] data = spark.read.csv("/content/sample_data/new EV data.csv", header=True, inferSchema=True)

[11] data.show(25)

+--------------------+----------------+-------+--------------+---------------+---------------+---------------+-------------+-------------+---------+--------------+-------------+----------+-------
|        Station_Name|         Address|   City|State_Province|Zip_Postal_Code|Start_Date_Time|Start_Time_Zone|End_Date_Time|End_Time_Zone|      Day|Total_Duration|Charging_Time|Energy_kWh|GHG_Sav
+--------------------+----------------+-------+--------------+---------------+---------------+---------------+-------------+-------------+---------+--------------+-------------+----------+-------
|BOULDER / BASELIN...|  900 Baseline Rd|Boulder|      Colorado|          80302|      1/20/2018|            MDT|    1/20/2018|          MDT| Saturday|       1:52:59|      1:52:41|     4.608|
|BOULDER / BASELIN...|  900 Baseline Rd|Boulder|      Colorado|          80302|      1/20/2018|            MDT|    1/20/2018|          MDT| Saturday|       3:45:39|      2:22:15|     8.786|
|BOULDER / JUNCTIO...|  2280 Junction Pl|Boulder|      Colorado|          80301|      1/20/2018|            MDT|    1/20/2018|          MDT| Saturday|       1:03:01|      1:02:50|      4.94|
|BOULDER / BASELIN...|  900 Baseline Rd|Boulder|      Colorado|          80302|      1/20/2018|            MDT|    1/20/2018|          MDT| Saturday|       0:02:00|      0:00:00|       0.0|
|BOULDER / REC CEN...|1360 Gillaspie Dr|Boulder|      Colorado|          80305|      1/20/2018|            MDT|    1/20/2018|          MDT| Saturday|       0:58:33|      0:58:25|     3.023|
|BOULDER / JUNCTIO...|  2280 Junction Pl|Boulder|      Colorado|          80301|      1/20/2018|            MDT|    1/21/2018|          MDT|   Sunday|      11:44:35|      3:09:24|     8.738|
|BOULDER / REC CEN...|1360 Gillaspie Dr|Boulder|      Colorado|          80305|      1/21/2018|            MDT|    1/21/2018|          MDT|   Sunday|       1:53:38|      1:53:27|    11.336|
|BOULDER / JUNCTIO...|  2280 Junction Pl|Boulder|      Colorado|          80301|      1/21/2018|            MDT|    1/22/2018|          MDT|   Monday|      14:14:38|      4:53:09|    19.976|
|BOULDER / ATRIUM ST1|     1770 13th St|Boulder|      Colorado|          80302|      1/22/2018|            MDT|    1/22/2018|          MDT|   Monday|       6:11:13|      2:06:43|      6.62|
|BOULDER / FACILIT...| 1745 14th street|Boulder|      Colorado|          80302|      1/22/2018|            MDT|    1/23/2018|          MDT|  Tuesday|      22:18:26|      2:14:06|     4.792|
|BOULDER / FACILIT...| 1745 14th street|Boulder|      Colorado|          80302|      1/22/2018|            MDT|    1/22/2018|          MDT|   Monday|       1:53:19|      1:53:05|     6.899|
|BOULDER / ALPINE ST1|  1275 Alpine Ave|Boulder|      Colorado|          80304|      1/22/2018|            MDT|    1/22/2018|          MDT|   Monday|       1:19:03|      1:18:47|     4.776|
|BOULDER / FACILIT...| 1745 14th street|Boulder|      Colorado|          80302|      1/22/2018|            MDT|    1/22/2018|          MDT|   Monday|       3:24:28|      3:24:09|      9.68|
|BOULDER / ATRIUM ST1|     1770 13th St|Boulder|      Colorado|          80302|      1/22/2018|            MDT|    1/22/2018|          MDT|   Monday|       0:02:19|      0:02:00|     0.116|
|BOULDER / ATRIUM ST1|     1770 13th St|Boulder|      Colorado|          80302|      1/23/2018|            MDT|    1/23/2018|          MDT|  Tuesday|       5:39:57|      4:51:46|    18.708|
|BOULDER / BASELIN...|  900 Baseline Rd|Boulder|      Colorado|          80302|      1/23/2018|            MDT|    1/23/2018|          MDT|  Tuesday|       0:01:38|      0:00:00|       0.0|
|BOULDER / ATRIUM ST1|     1770 13th St|Boulder|      Colorado|          80302|      1/23/2018|            MDT|    1/23/2018|          MDT|  Tuesday|       0:34:37|      0:34:26|     2.165|
|BOULDER / FACILIT...| 1745 14th street|Boulder|      Colorado|          80302|      1/23/2018|            MDT|    1/23/2018|          MDT|  Tuesday|       2:47:14|      1:36:11|     5.728|
|BOULDER / REC CEN...|1360 Gillaspie Dr|Boulder|      Colorado|          80305|      1/23/2018|            MDT|    1/23/2018|          MDT|  Tuesday|       1:10:23|      0:54:59|     2.876|
|BOULDER / JUNCTIO...|  2280 Junction Pl|Boulder|      Colorado|          80301|      1/23/2018|            MDT|    1/23/2018|          MDT|  Tuesday|       0:59:21|      0:59:11|     3.131|
|BOULDER / BASELIN...|  900 Baseline Rd|Boulder|      Colorado|          80302|      1/24/2018|            MDT|    1/24/2018|          MDT|  Tuesday|       3:37:44|      3:36:15|    24.887|
```

As describe() function is used such that it displays all the summary of the given dataset. As seen in the below screenshot it shows entire columns and its data type.



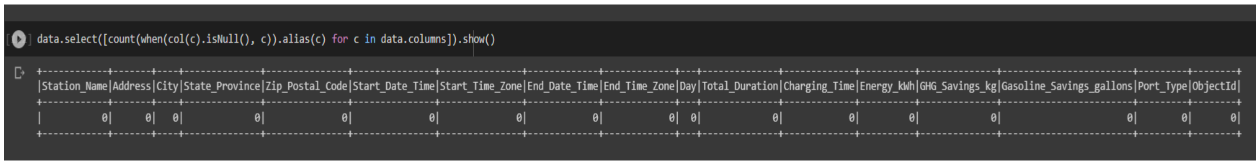
```
data = spark.read.csv("/content/sample_data/new EV data.csv", header=True, inferSchema=True)

data.describe()

DataFrame[summary: string, Station_Name: string, Address: string, City: string, State_Province: string, Zip_Postal_Code: string, Start_Date_Time: string, Start_Time_Zone: string, End_Date_Time: string, End_Time_Zone: string, Day: string,
Total_Duration: string, Charging_Time: string, Energy_kWh: string, GHG_Savings_kg: string, Gasoline_Savings_gallons: string, Port_Type: string, ObjectId: string]
```

5.1.2. Data Cleaning:
Dataframe.describe().show() describes the entire dataset summary such as it gives out the count of rows, Mean, minimum and maximum of each column.



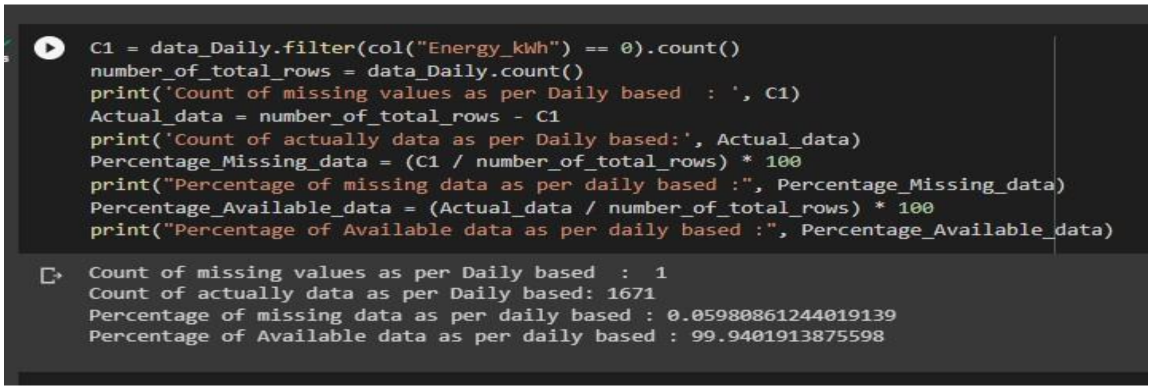
```
[118] data.describe()

DataFrame[summary: string, Station_Name: string, Address: string, City: string, State_Province: string, Zip_Postal_Code: string, Start_Date_Time: string, Start_Time_Zone: string, End_Date_Time: string, End_Time_Zone: string, Day: string, Total_Duration: string, Charging_Time: string, Energy_kW
string, Gasoline_Savings_gallons: string, Port_Type: string, ObjectId: string]

data.describe().show()

+-------+--------------------+---------------+-------+--------------+------------------+---------------+---------------+-------------+-------------+---------+--------------+-------------+-----------------+-----------------+------------------------+---------+------------------+
|summary|        Station_Name|        Address|   City|State_Province|   Zip_Postal_Code|Start_Date_Time|Start_Time_Zone|End_Date_Time|End_Time_Zone|      Day|Total_Duration|Charging_Time|       Energy_kWh|    GHG_Savings_kg|Gasoline_Savings_gallons|Port_Type|          ObjectId|
+-------+--------------------+---------------+-------+--------------+------------------+---------------+---------------+-------------+-------------+---------+--------------+-------------+-----------------+-----------------+------------------------+---------+------------------+
|  count|               43659|          43659|  43659|         43659|             43659|          43659|          43659|        43659|        43659|    43659|         43659|        43659|            43659|            43659|                   43659|    43659|             43659|
|   mean|                null|           null|   null|          null| 80302.54414897272|           null|           null|         null|         null|     null|          null|         null|8.123260862594233|4.5745162967543695|      1.0194689296594048|     null|           21830.0|
| stddev|                null|           null|   null|          null|1.0791615273984834|           null|           null|         null|         null|     null|          null|         null|8.032178469871875| 4.988628086803563|        1.00803737364384|     null|12603.412038015738|
|    min|BOULDER / AIRPORT...|1100 Spruce St|Boulder|      Colorado|             80301|       1/1/2018|            MDT|     1/1/2018|          MDT|   Friday|       0:00:01|      0:00:00|              0.0|               0.0|                     0.0|  Level 2|                 1|
|    max|COMM VITALITY / B...| 900 Walnut St|Boulder|      Colorado|             80305|       9/9/2021|            MST|     9/9/2021|          MST|Wednesday|       9:59:31|      9:56:12|             85.2|            49.834|                  10.693|  Level 2|             43659|
+-------+--------------------+---------------+-------+--------------+------------------+---------------+---------------+-------------+-------------+---------+--------------+-------------+-----------------+-----------------+------------------------+---------+------------------+
```

Checked for null values in each column. Found that there are no null values in the dataset.



```
data.select([count(when(col(c).isNull(), c)).alias(c) for c in data.columns]).show()

+------------+-------+----+--------------+---------------+---------------+---------------+-------------+-------------+---+--------------+-------------+----------+--------------+------------------------+---------+--------+
|Station_Name|Address|City|State_Province|Zip_Postal_Code|Start_Date_Time|Start_Time_Zone|End_Date_Time|End_Time_Zone|Day|Total_Duration|Charging_Time|Energy_kWh|GHG_Savings_kg|Gasoline_Savings_gallons|Port_Type|ObjectId|
+------------+-------+----+--------------+---------------+---------------+---------------+-------------+-------------+---+--------------+-------------+----------+--------------+------------------------+---------+--------+
|           0|      0|   0|             0|              0|              0|              0|            0|            0|  0|             0|            0|         0|             0|                       0|        0|       0|
+------------+-------+----+--------------+---------------+---------------+---------------+-------------+-------------+---+--------------+-------------+----------+--------------+------------------------+---------+--------+
```

The missing values were checked and displayed the number of missing values based on day in the energy column, found that there is only 1 missing value in the data and the availability of data is 99.94%.



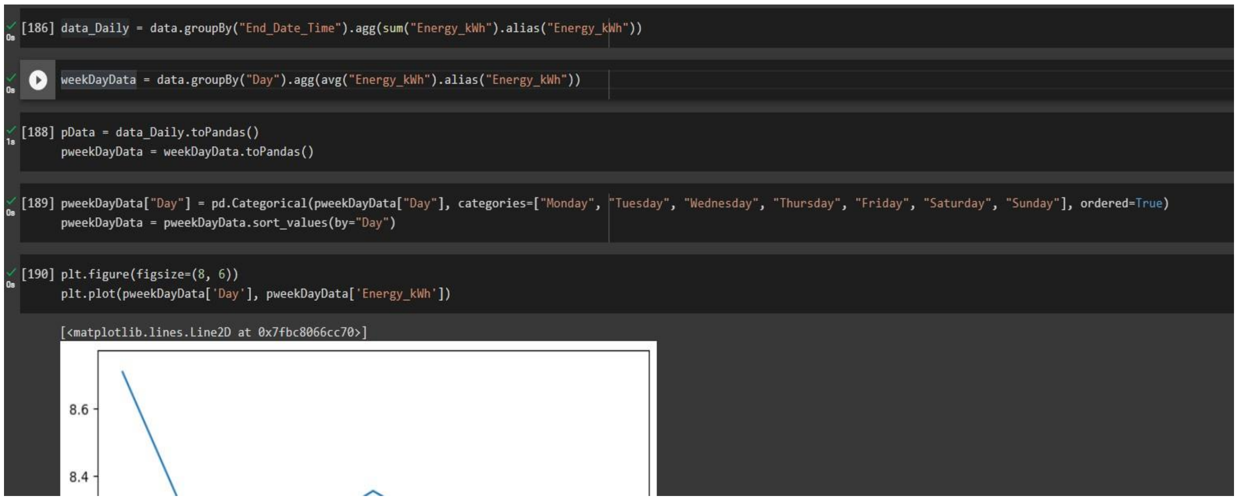
```
C1 = data_Daily.filter(col("Energy_kWh") == 0).count()
number_of_total_rows = data_Daily.count()
print('Count of missing values as per Daily based  : ', C1)
Actual_data = number_of_total_rows - C1
print('Count of actually data as per Daily based:', Actual_data)
Percentage_Missing_data = (C1 / number_of_total_rows) * 100
print("Percentage of missing data as per daily based :", Percentage_Missing_data)
Percentage_Available_data = (Actual_data / number_of_total_rows) * 100
print("Percentage of Available data as per daily based :", Percentage_Available_data)

Count of missing values as per Daily based  :  1
Count of actually data as per Daily based: 1671
Percentage of missing data as per daily based : 0.05980861244019139
Percentage of Available data as per daily based : 99.9401913875598
```

5.2 Implementation & Results:
Vehicle Charge Station Load Analysis and Prediction examines how electric car charging stations are used and forecasts future demand. Statistical analysis and machine learning methods can be used for this.

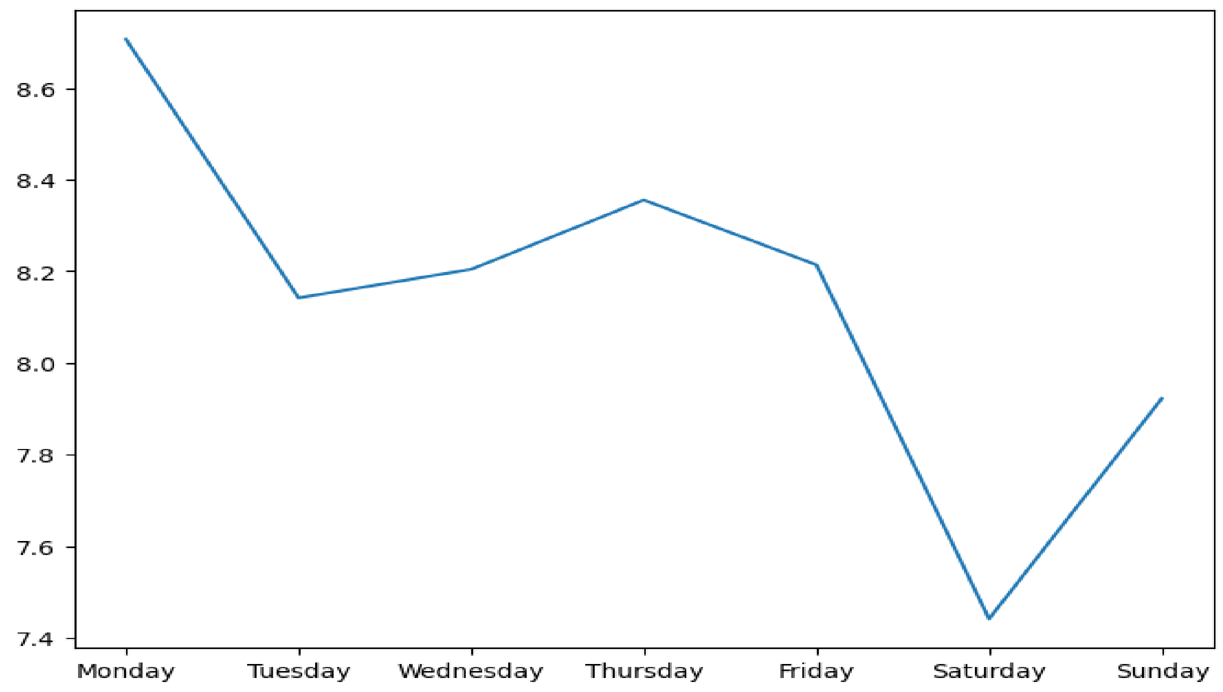
1. Data gathering: Gather information on how the charging stations are used, including the time of day, day of the week, amount of time spent charging, kind of vehicle, etc. Users can be surveyed or sensors put in the charging stations can be used to obtain this data.
2. Cleaning and converting the data into a format that is appropriate for analysis constitute preprocessing. Outliers may be eliminated, missing values filled in, and categorical variables encoded as a part of this process.
3. Exploratory Data Analysis (EDA): Use EDA to learn more about how the charging stations are used. To see trends and patterns, the data may be shown using plots and charts.
4. Choose the elements that have the greatest impact on the demand for charging stations. Techniques like feature significance ranking and correlation analysis can be used for this.

5. Model Selection: To create a predictive model, pick an appropriate machine learning algorithm, such as regression, time-series analysis, or clustering. The kind of problem and the qualities of the data will determine which method is used.
6. Model Training: Using the preprocessed data, train the chosen model. In order to do this, the data must be divided into training and validation sets, and the model parameters must be optimized using methods like grid search or cross-validation.
7. Using appropriate metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Square Error, evaluate the performance of the model (RMSE). By highlighting the model's advantages and disadvantages, this will aid in its improvement.
8. The trained model should be deployed in an appropriate context, such as a web application or an API. This will make it possible to make forecasts and suggestions in real-time depending on the demand for charging stations at the moment.
9. The performance of the deployed model should be continuously monitored, and it should be updated on occasion in light of fresh information and user feedback.

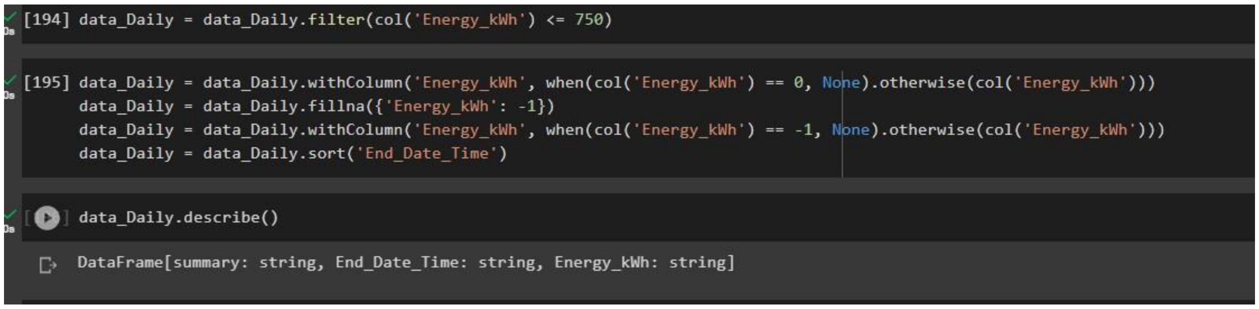
In general, data collection, preprocessing, exploratory data analysis, feature selection, model selection, training, assessment, deployment, and maintenance are all involved in the implementation of Car Charging Station Load Analysis and Prediction. High-quality data must be available, appropriate techniques must be used, and the models must be continuously monitored and improved if this application is to be successful.
Here,is the analysis of data according to the day with respective to Energy in KWh and the plot is attached below.



```
[186] data_Daily = data.groupBy("End_Date_Time").agg(sum("Energy_kWh").alias("Energy_kWh"))

      weekDayData = data.groupBy("Day").agg(avg("Energy_kWh").alias("Energy_kWh"))

[188] pData = data_Daily.toPandas()
      pweekDayData = weekDayData.toPandas()

[189] pweekDayData["Day"] = pd.Categorical(pweekDayData["Day"], categories=["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"], ordered=True)
      pweekDayData = pweekDayData.sort_values(by="Day")

[190] plt.figure(figsize=(8, 6))
      plt.plot(pweekDayData['Day'], pweekDayData['Energy_kWh'])

      [<matplotlib.lines.Line2D at 0x7fbc8066cc70>]
```



Later we filter the data of Energy to less than 750 because we observed the data whose values are less than 750. If the data is more than 750 then we may not get the accurate data, so we filter all the values to less than 750.

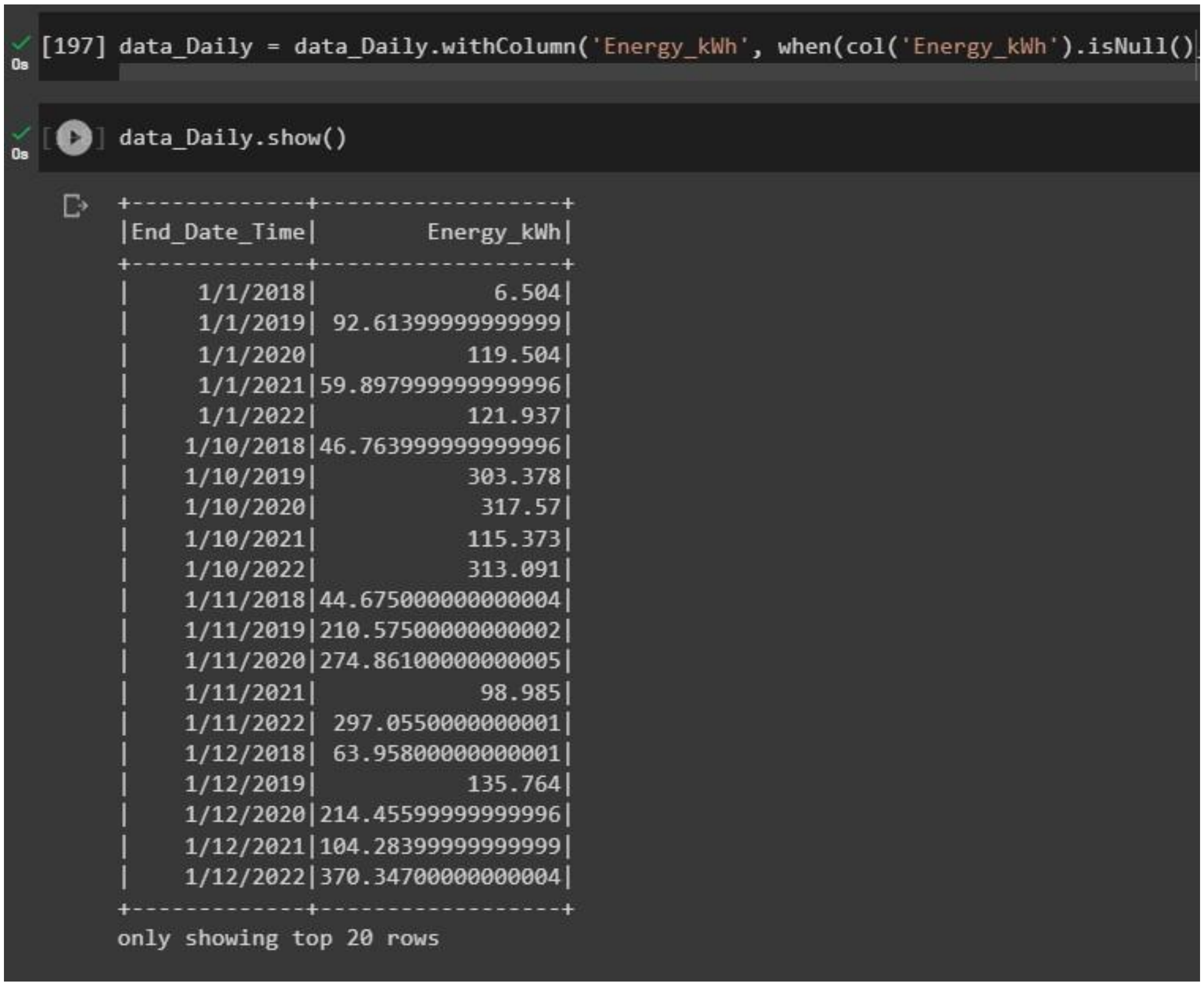


```
[194] data_Daily = data_Daily.filter(col('Energy_kWh') <= 750)

[195] data_Daily = data_Daily.withColumn('Energy_kWh', when(col('Energy_kWh') == 0, None).otherwise(col('Energy_kWh')))
      data_Daily = data_Daily.fillna({'Energy_kWh': -1})
      data_Daily = data_Daily.withColumn('Energy_kWh', when(col('Energy_kWh') == -1, None).otherwise(col('Energy_kWh')))
      data_Daily = data_Daily.sort('End_Date_Time')

      data_Daily.describe()

      DataFrame[summary: string, End_Date_Time: string, Energy_kWh: string]
```

After that we just display the data to check whether the values of Energy are below 750 and we check for any null values present or not .

Here is the display of the data of Energy with respective End date time .

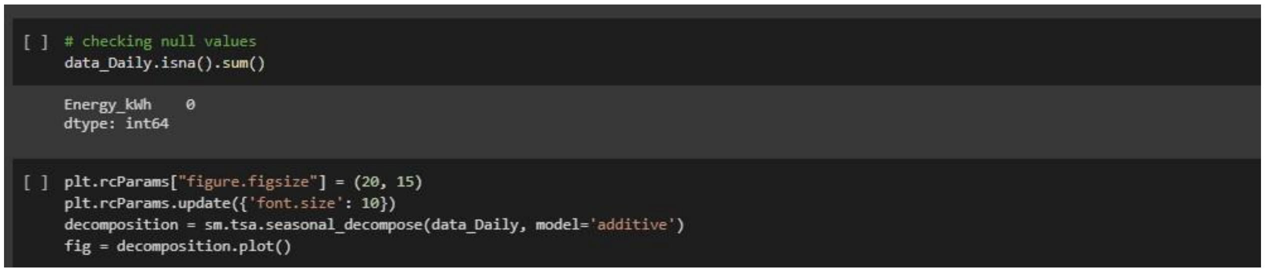
Note : Here , the data displayed is not in the sorted order .



```
[197] data_Daily = data_Daily.withColumn('Energy_kWh', when(col('Energy_kWh').isNull()
```

```
[ ] data_Daily.show()
```

```
+-------------+------------------+
|End_Date_Time|        Energy_kWh|
+-------------+------------------+
|     1/1/2018|             6.504|
|     1/1/2019| 92.61399999999999|
|     1/1/2020|           119.504|
|     1/1/2021|59.897999999999996|
|     1/1/2022|           121.937|
|    1/10/2018|46.763999999999996|
|    1/10/2019|           303.378|
|    1/10/2020|            317.57|
|    1/10/2021|           115.373|
|    1/10/2022|           313.091|
|    1/11/2018|44.675000000000004|
|    1/11/2019|210.57500000000002|
|    1/11/2020|274.86100000000005|
|    1/11/2021|            98.985|
|    1/11/2022| 297.0550000000001|
|    1/12/2018| 63.95800000000001|
|    1/12/2019|           135.764|
|    1/12/2020|214.45599999999996|
|    1/12/2021|104.28399999999999|
|    1/12/2022|370.34700000000004|
+-------------+------------------+
only showing top 20 rows
```



```
[ ] # checking null values
    data_Daily.isna().sum()

    Energy_kWh    0
    dtype: int64
```

```
[ ] plt.rcParams["figure.figsize"] = (20, 15)
    plt.rcParams.update({'font.size': 10})
    decomposition = sm.tsa.seasonal_decompose(data_Daily, model='additive')
    fig = decomposition.plot()
```

Decomposing a series into its trend, seasonal, and residual components is a frequent practice in time series analysis. Making forecasts and comprehending the underlying structures and patterns in the data can both benefit from this.

The trend, seasonal, and residual components of the data may be determined using a time series decomposition approach for the study and forecast of the demand at automotive charging stations. Here's how to accomplish it:

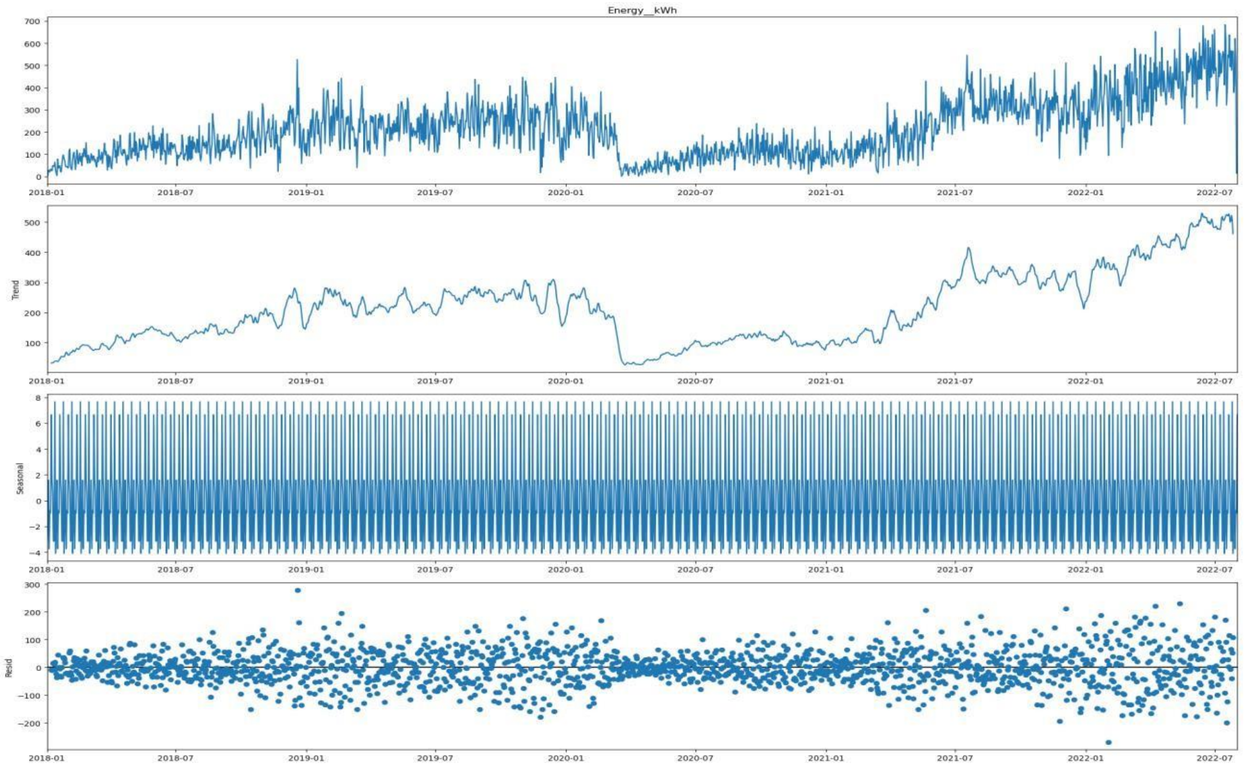
1. Trend Component : The trend component, which might be growing or decreasing over time, shows the series' long-term orientation. The trend component may reflect the progressive rise in demand for electric vehicles and, accordingly, charging stations in the context of the load at automotive charging stations. Using a smoothing method like moving

averages, exponential smoothing, or regression analysis, we may estimate the trend component.

2. Seasonal Component : The periodic changes in the series that can happen every day, every week, every month, or every year are represented by the seasonal component. When discussing the load at charging stations for cars, the seasonal component could be able to account for daily and weekly fluctuations in demand, such as higher demand during the workweek and lower demand on the weekends. We may calculate the seasonal component utilizing seasonal decomposition methods as seasonal naïve, seasonal moving average, or seasonal decomposition of time series (STL) approaches.
3. Residual Component : The random fluctuations or noise in the series that cannot be accounted for by the trend or seasonal components are represented by the residual component. When discussing the load on automotive charging stations, the residual component may account for unforeseen or unplanned shifts in demand, such as sharp increases or decreases in usage. The gap between the observed data and the total of the trend and seasonal components allows us to determine the residual component.

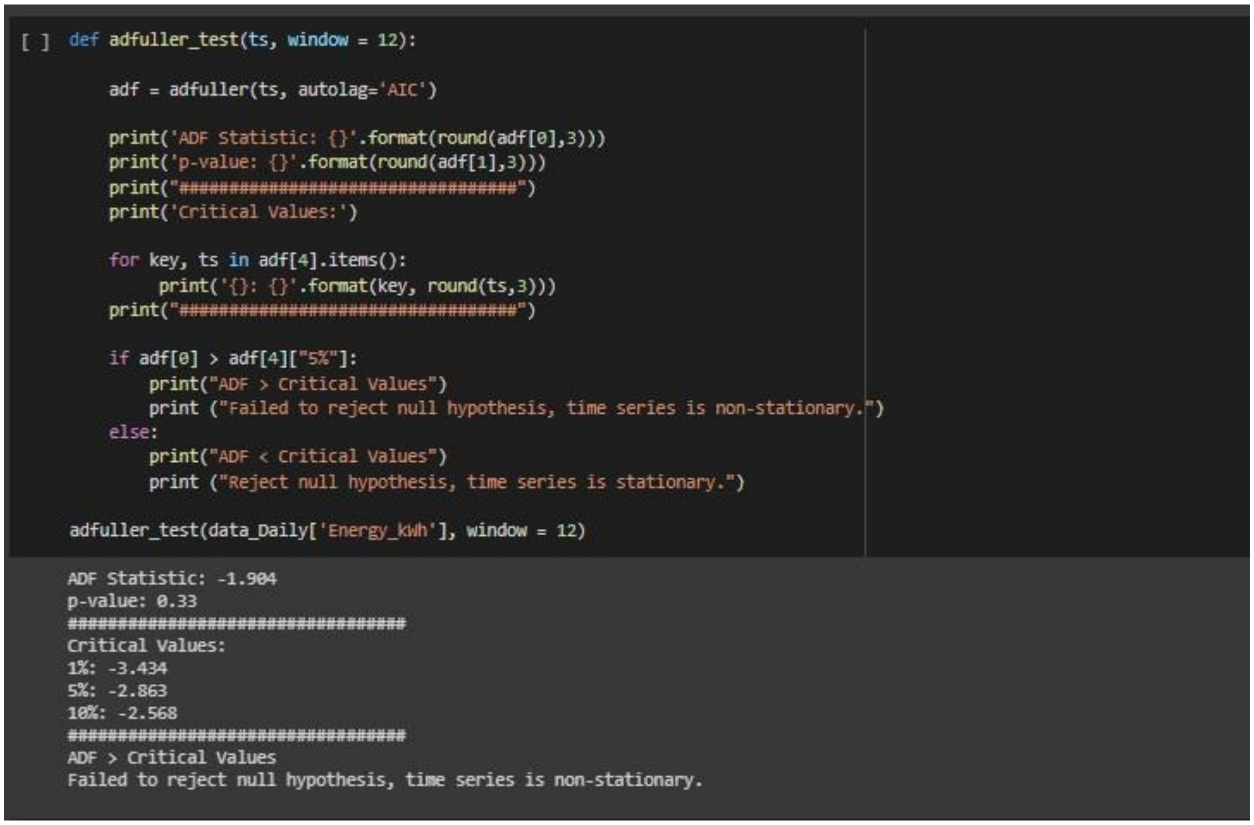
We may utilize the trend, seasonal, and residual components of the data on the load at automotive charging stations after we have approximated them. For instance, we can anticipate the long-term need for charging stations using the trend component, the daily and weekly variations in demand using the seasonal component, and the unexpected changes in usage using the residual component.

Overall, breaking down time series data into its constituent parts can offer useful insights into the underlying structures and patterns of the data, which can aid in developing reliable forecasts and well-informed choices.


Energy__kWh
Trend
Seasonal
Resid
2018-01
2018-07
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07

5.2.1 Implementation of ARIMA model

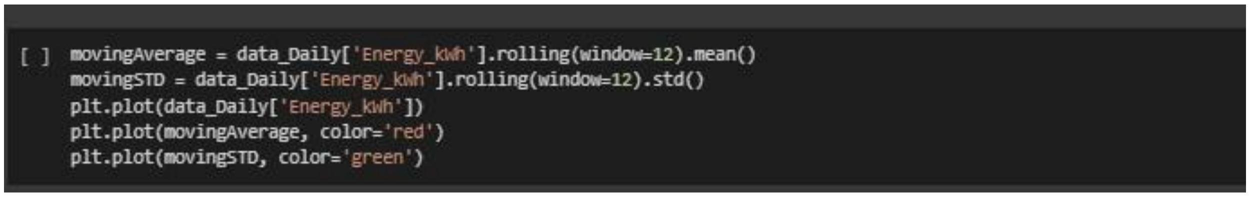
Using adfuller test, we are checking if the data is stationary or not. Arima model needs to stationary data to train the model and predict the future values. This test is carried out on the energy_kwh column using the adfuller method available in python.



```
[ ] def adfuller_test(ts, window = 12):

        adf = adfuller(ts, autolag='AIC')

        print('ADF Statistic: {}'.format(round(adf[0],3)))
        print('p-value: {}'.format(round(adf[1],3)))
        print("##################################")
        print('Critical Values:')

        for key, ts in adf[4].items():
             print('{}: {}'.format(key, round(ts,3)))
        print("##################################")

        if adf[0] > adf[4]["5%"]:
            print("ADF > Critical Values")
            print ("Failed to reject null hypothesis, time series is non-stationary.")
        else:
            print("ADF < Critical Values")
            print ("Reject null hypothesis, time series is stationary.")

    adfuller_test(data_Daily['Energy_kWh'], window = 12)
```

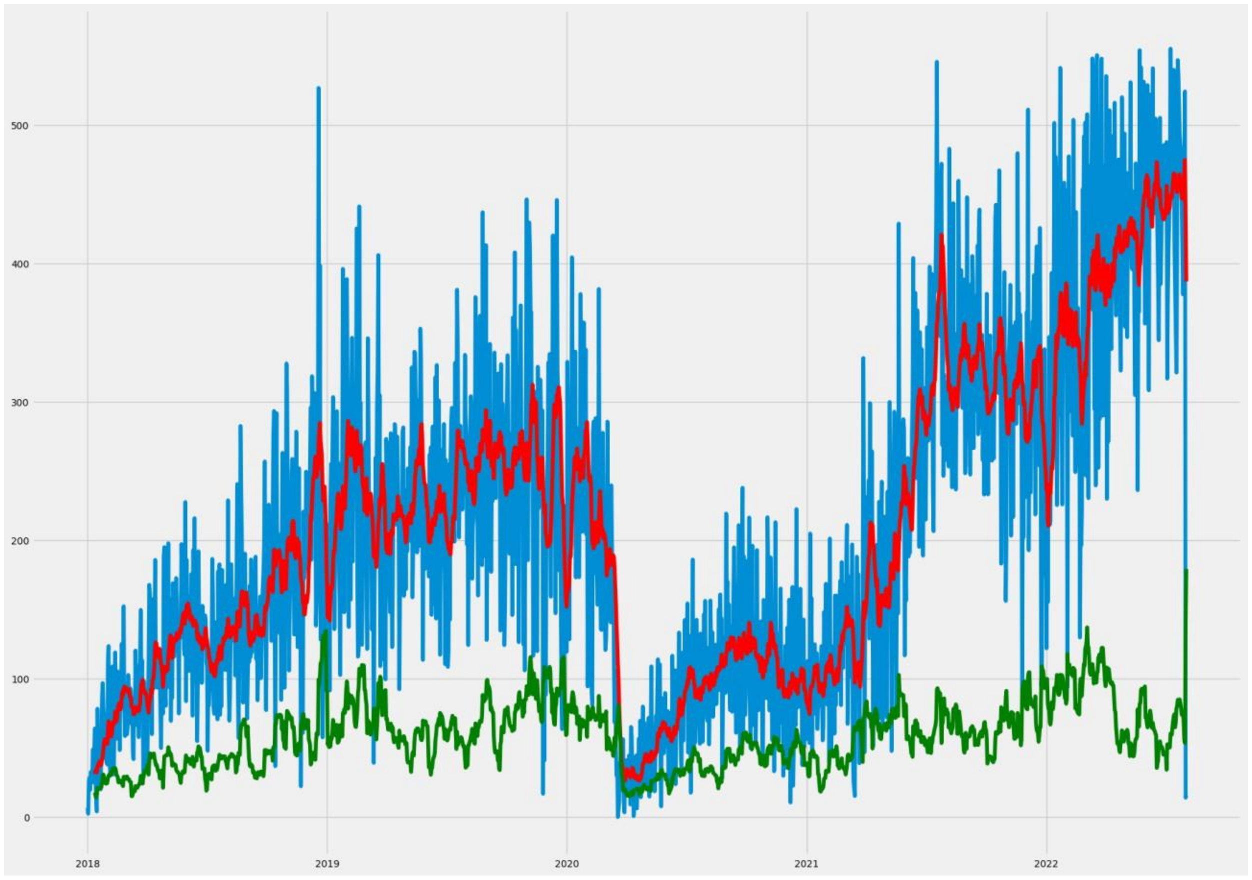
```
ADF Statistic: -1.904
p-value: 0.33
##################################
Critical Values:
1%: -3.434
5%: -2.863
10%: -2.568
##################################
ADF > Critical Values
Failed to reject null hypothesis, time series is non-stationary.
```

From the results of adfuller test, we found that the data is not stationary and failed to reject the null hypothesis. To make the dataset stationary the moving average of the energy_kwh is calculated over a rolling window of 12. The standard deviation is also calculated.

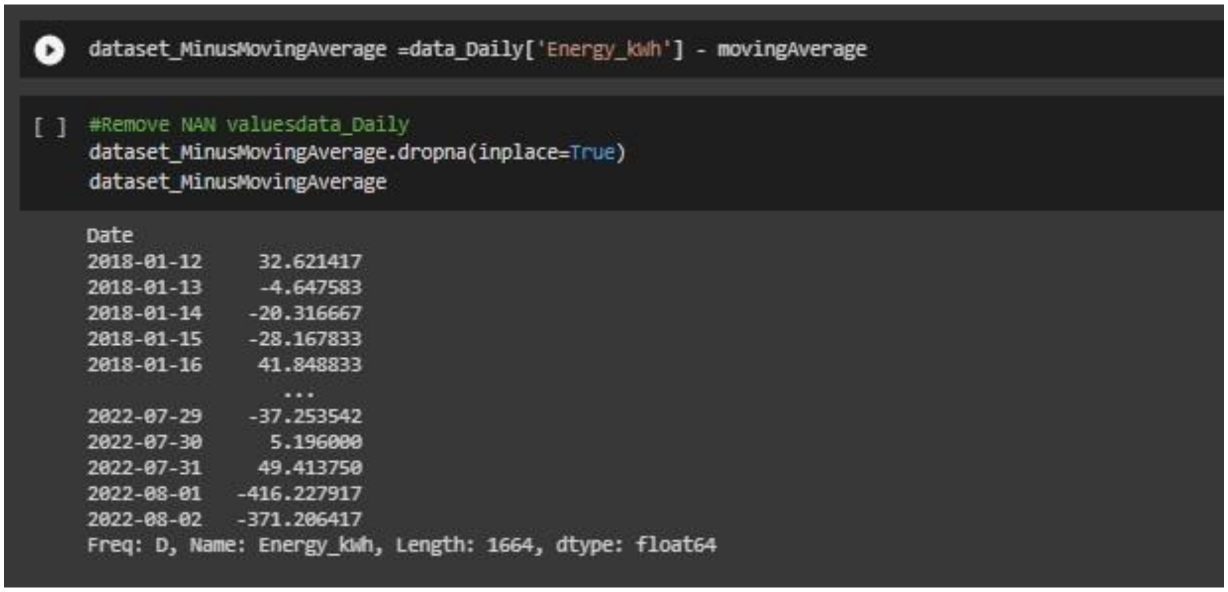


```
[ ] movingAverage = data_Daily['Energy_kWh'].rolling(window=12).mean()
    movingSTD = data_Daily['Energy_kWh'].rolling(window=12).std()
    plt.plot(data_Daily['Energy_kWh'])
    plt.plot(movingAverage, color='red')
    plt.plot(movingSTD, color='green')
```

The plot for the energy, moving average and the moving standard deviation is presented below.



The moving average is subtracted from the energy column to make the data stationary. Then we are checking for null values and dropping them.

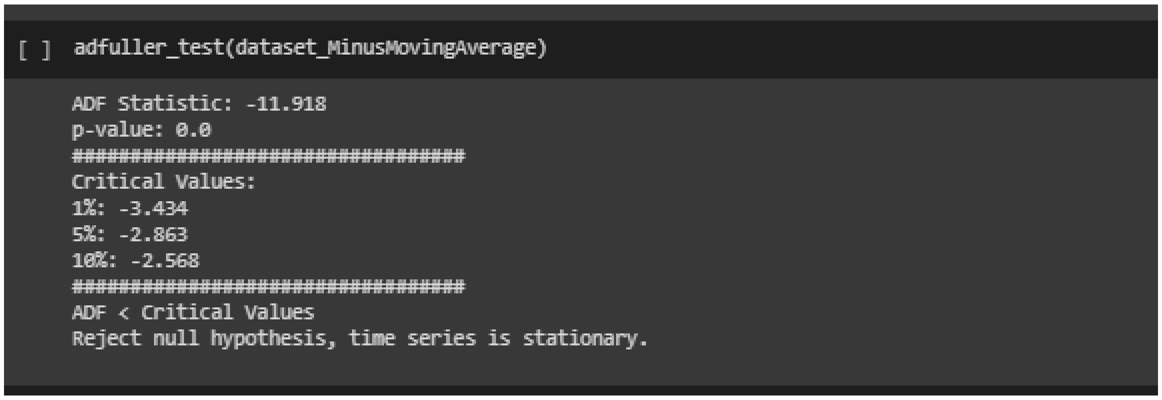


```
dataset_MinusMovingAverage =data_Daily['Energy_kWh'] - movingAverage
```

```
#Remove NAN valuesdata_Daily
dataset_MinusMovingAverage.dropna(inplace=True)
dataset_MinusMovingAverage
```

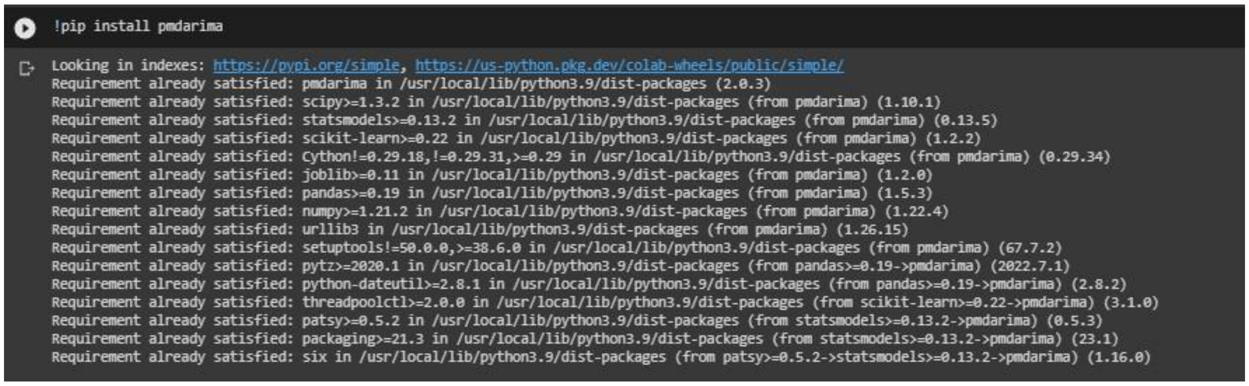
```
Date
2018-01-12     32.621417
2018-01-13     -4.647583
2018-01-14    -20.316667
2018-01-15    -28.167833
2018-01-16     41.848833
                 ...
2022-07-29    -37.253542
2022-07-30      5.196000
2022-07-31     49.413750
2022-08-01   -416.227917
2022-08-02   -371.206417
Freq: D, Name: Energy_kWh, Length: 1664, dtype: float64
```

Verifying if the data is stationary or not using the adfuller test after performing computations to make the data stationary.



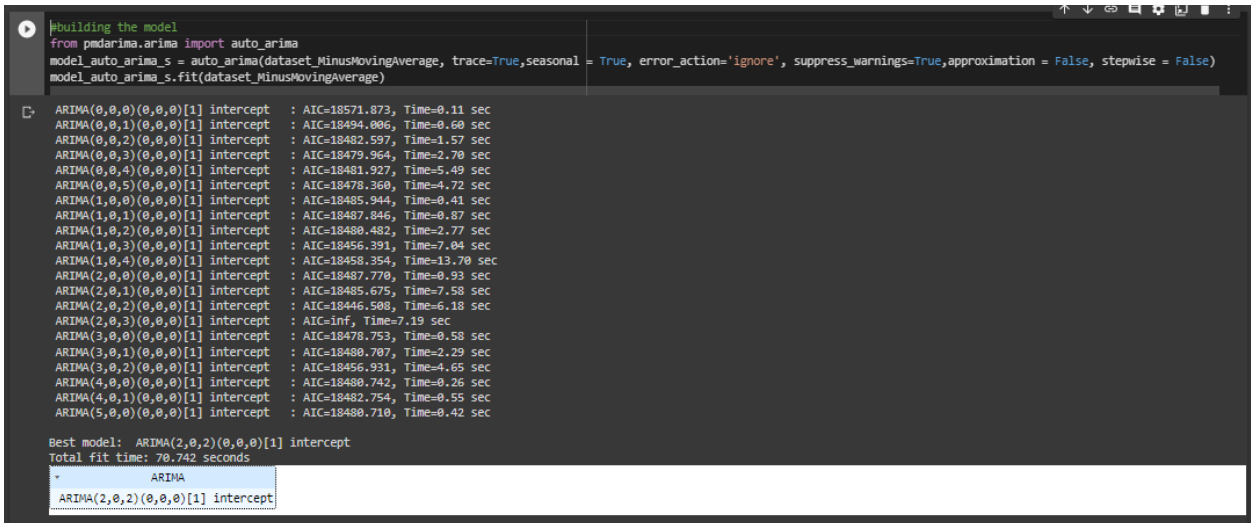
```
[ ]  adfuller_test(dataset_MinusMovingAverage)

     ADF Statistic: -11.918
     p-value: 0.0
     ##################################
     Critical Values:
     1%: -3.434
     5%: -2.863
     10%: -2.568
     ##################################
     ADF < Critical Values
     Reject null hypothesis, time series is stationary.
```

Installing Arima model using pip.



```
!pip install pmdarima

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pmdarima in /usr/local/lib/python3.9/dist-packages (2.0.3)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (1.10.1)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (0.13.5)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (1.2.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (0.29.34)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (1.2.0)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (1.5.3)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (1.22.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (1.26.15)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.9/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.19->pmdarima) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=0.22->pmdarima) (3.1.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.9/dist-packages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.9/dist-packages (from statsmodels>=0.13.2->pmdarima) (23.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

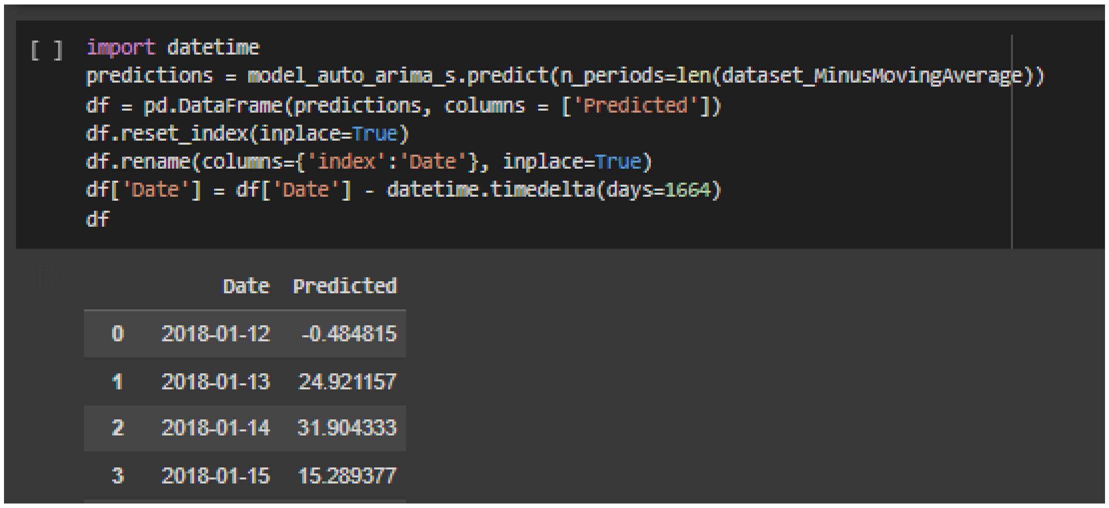
Building the model importing auto_arima form installed Arima. The model is provided with the stationary dataset i.e., dataset_MinusMovingAverage. Trace is given as true. Trace refers to auto-correlation matrix. This will check if the sufficient parameters are available to build the model. Suppress warnings will suppress the warnings in the output. Approximation false will not allow the model to take approximations in calculations.



```
#building the model
from pmdarima.arima import auto_arima
model_auto_arima_s = auto_arima(dataset_MinusMovingAverage, trace=True,seasonal = True, error_action='ignore', suppress_warnings=True,approximation = False, stepwise = False)
model_auto_arima_s.fit(dataset_MinusMovingAverage)
```

```
ARIMA(0,0,0)(0,0,0)[1] intercept   : AIC=18571.873, Time=0.11 sec
ARIMA(0,0,1)(0,0,0)[1] intercept   : AIC=18494.006, Time=0.60 sec
ARIMA(0,0,2)(0,0,0)[1] intercept   : AIC=18482.597, Time=1.57 sec
ARIMA(0,0,3)(0,0,0)[1] intercept   : AIC=18479.964, Time=2.70 sec
ARIMA(0,0,4)(0,0,0)[1] intercept   : AIC=18481.927, Time=5.49 sec
ARIMA(0,0,5)(0,0,0)[1] intercept   : AIC=18478.360, Time=4.72 sec
ARIMA(1,0,0)(0,0,0)[1] intercept   : AIC=18485.944, Time=0.41 sec
ARIMA(1,0,1)(0,0,0)[1] intercept   : AIC=18487.846, Time=0.87 sec
ARIMA(1,0,2)(0,0,0)[1] intercept   : AIC=18480.482, Time=2.77 sec
ARIMA(1,0,3)(0,0,0)[1] intercept   : AIC=18456.391, Time=7.04 sec
ARIMA(1,0,4)(0,0,0)[1] intercept   : AIC=18458.354, Time=13.70 sec
ARIMA(2,0,0)(0,0,0)[1] intercept   : AIC=18487.770, Time=0.93 sec
ARIMA(2,0,1)(0,0,0)[1] intercept   : AIC=18485.675, Time=7.58 sec
ARIMA(2,0,2)(0,0,0)[1] intercept   : AIC=18446.508, Time=6.18 sec
ARIMA(2,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=7.19 sec
ARIMA(3,0,0)(0,0,0)[1] intercept   : AIC=18478.753, Time=0.58 sec
ARIMA(3,0,1)(0,0,0)[1] intercept   : AIC=18480.707, Time=2.29 sec
ARIMA(3,0,2)(0,0,0)[1] intercept   : AIC=18456.931, Time=4.65 sec
ARIMA(4,0,0)(0,0,0)[1] intercept   : AIC=18480.742, Time=0.26 sec
ARIMA(4,0,1)(0,0,0)[1] intercept   : AIC=18482.754, Time=0.55 sec
ARIMA(5,0,0)(0,0,0)[1] intercept   : AIC=18480.710, Time=0.42 sec

Best model:  ARIMA(2,0,2)(0,0,0)[1] intercept
Total fit time: 70.742 seconds
            ARIMA
ARIMA(2,0,2)(0,0,0)[1] intercept
```

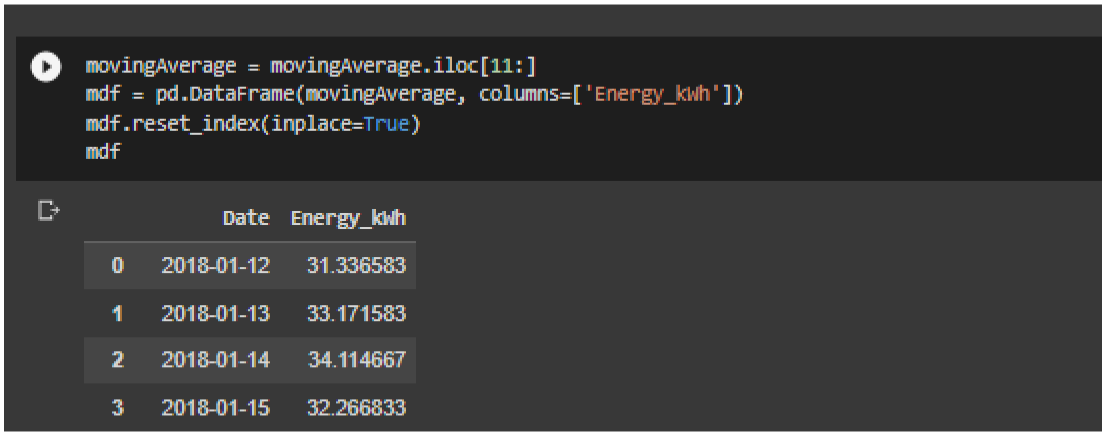
The below code will predict values for the provident input data for the model. Then the predicted data is added to a dataframe using pandas. The data is rearranged by subtracting the count of the input data from the predicted to align them in place.



```
import datetime
predictions = model_auto_arima_s.predict(n_periods=len(dataset_MinusMovingAverage))
df = pd.DataFrame(predictions, columns = ['Predicted'])
df.reset_index(inplace=True)
df.rename(columns={'index':'Date'}, inplace=True)
df['Date'] = df['Date'] - datetime.timedelta(days=1664)
df
```

	Date	Predicted
0	2018-01-12	-0.484815
1	2018-01-13	24.921157
2	2018-01-14	31.904333
3	2018-01-15	15.289377

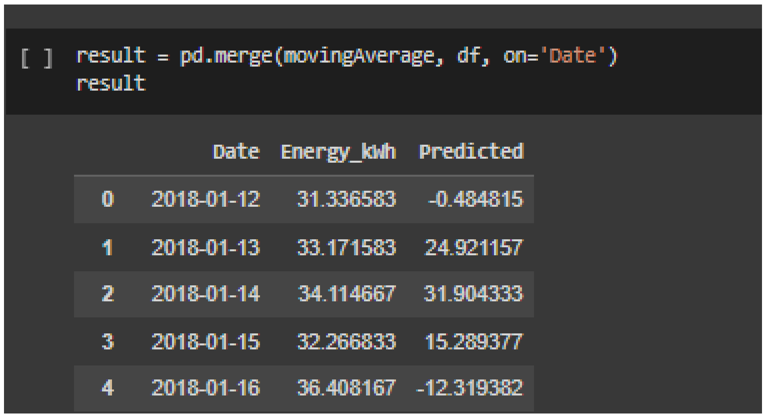
The top 12 value in the moving average dataframe will become nulls as we are using rolling window size as 12. So, the top 12 values are removed from the data. Then a new data frame is created for the resultant and index of the data frame is reset.



```
movingAverage = movingAverage.iloc[11:]
mdf = pd.DataFrame(movingAverage, columns=['Energy_kWh'])
mdf.reset_index(inplace=True)
mdf
```

```
         Date   Energy_kWh
0  2018-01-12   31.336583
1  2018-01-13   33.171583
2  2018-01-14   34.114667
3  2018-01-15   32.266833
```

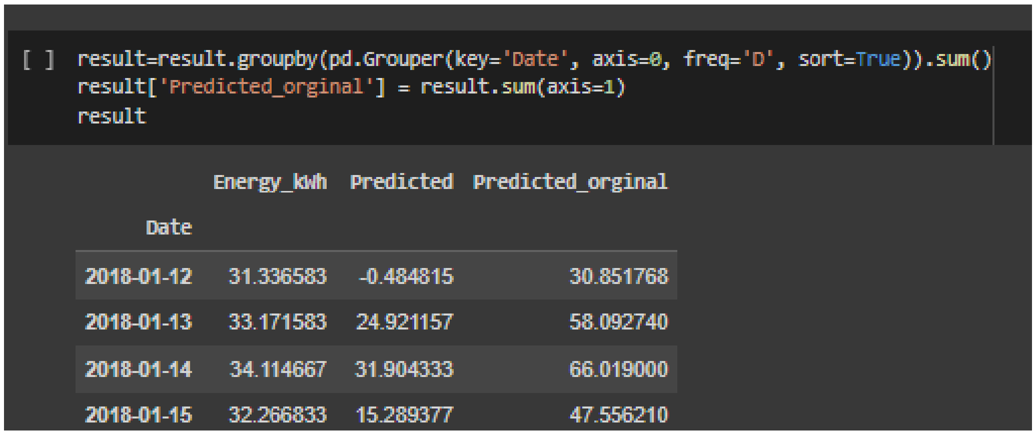
The predicted values and the actual values are merged and formed a single dataframe. For easy analysis.



```
[ ] result = pd.merge(movingAverage, df, on='Date')
    result
```

```
         Date   Energy_kWh   Predicted
0  2018-01-12   31.336583    -0.484815
1  2018-01-13   33.171583    24.921157
2  2018-01-14   34.114667    31.904333
3  2018-01-15   32.266833    15.289377
4  2018-01-16   36.408167   -12.319382
```

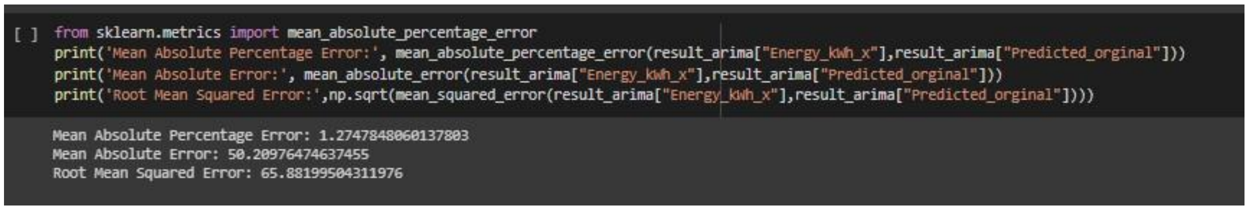
The predicted value is again added with the removed moving average and show in the table.



```
[ ] result=result.groupby(pd.Grouper(key='Date', axis=0, freq='D', sort=True)).sum()
    result['Predicted_orginal'] = result.sum(axis=1)
    result
```

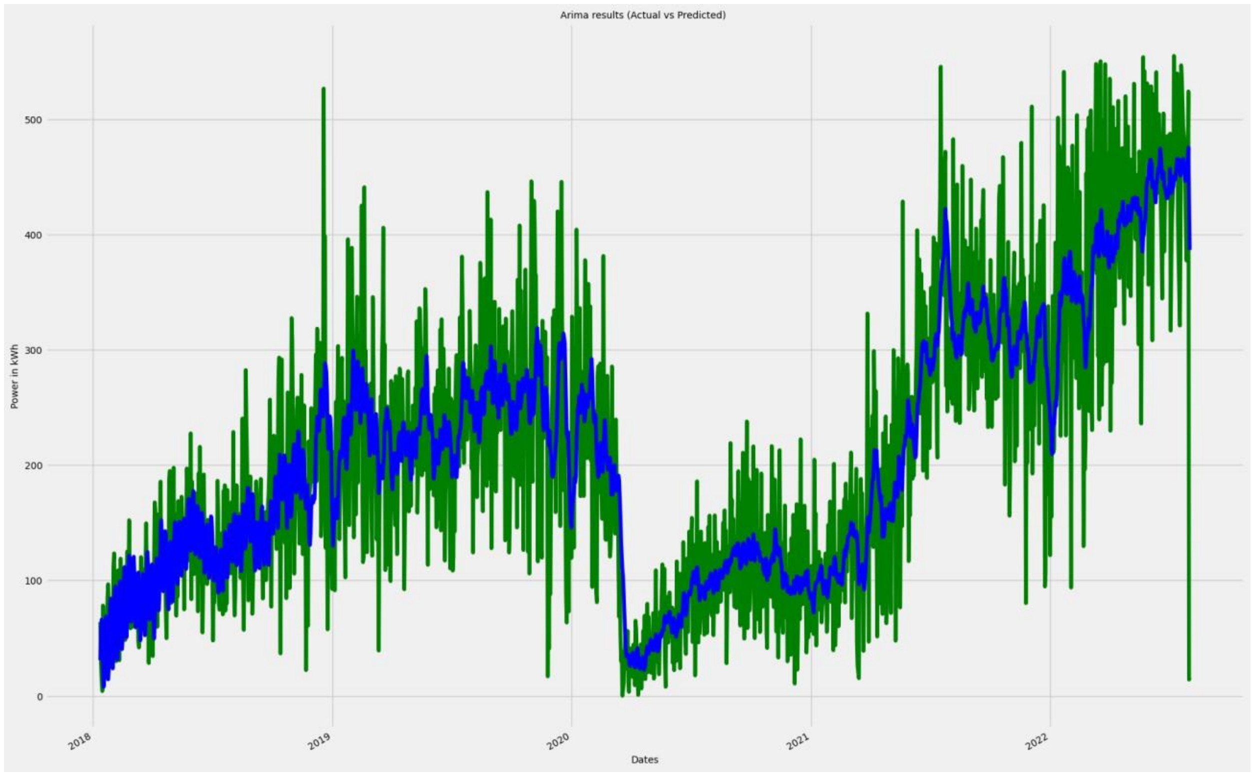
```
            Energy_kWh   Predicted   Predicted_orginal
Date
2018-01-12  31.336583    -0.484815           30.851768
2018-01-13  33.171583    24.921157           58.092740
2018-01-14  34.114667    31.904333           66.019000
2018-01-15  32.266833    15.289377           47.556210
```

The performance of the model is evaluated below. From the results the erorr percentage, absolute error and the mean squared errors are high. So, we found that the model can not be used for predicting the load analysis.



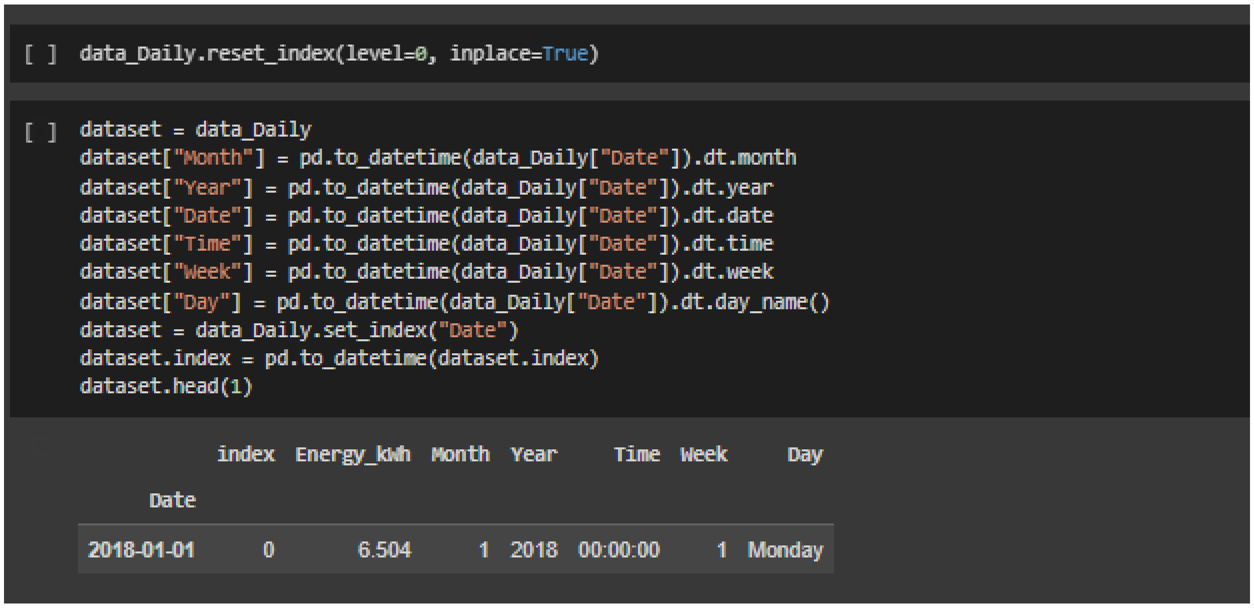
```
[ ] from sklearn.metrics import mean_absolute_percentage_error
    print('Mean Absolute Percentage Error:', mean_absolute_percentage_error(result_arima["Energy_kWh_x"],result_arima["Predicted_orginal"]))
    print('Mean Absolute Error:', mean_absolute_error(result_arima["Energy_kWh_x"],result_arima["Predicted_orginal"]))
    print('Root Mean Squared Error:',np.sqrt(mean_squared_error(result_arima["Energy_kWh_x"],result_arima["Predicted_orginal"])))

    Mean Absolute Percentage Error: 1.2747848060137803
    Mean Absolute Error: 50.20976474637455
    Root Mean Squared Error: 65.88199504311976
```

The plot for the original values and the predicted values is provided below. In the figure the green line is the actual value and the blue line is the predicted value. It is clearly evident from the figure that the predicted values are looking like the average of the actual data but not the close enough to the actual values.



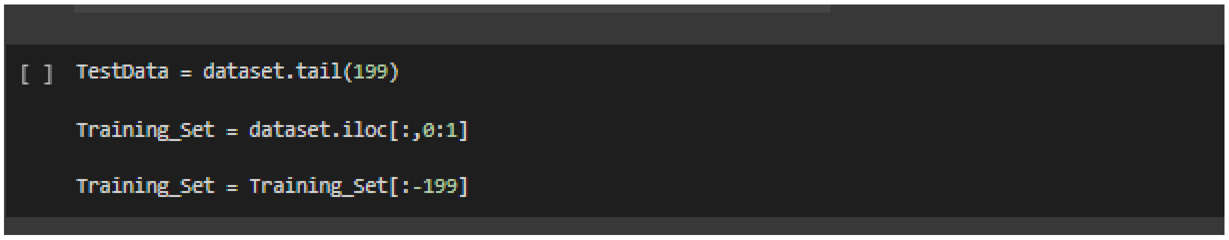
5.2.2 LSTM Model

Resting the index of the dataset and adding the date dimension parameters to the dataset. The date dimension table will have the columns date, hour, day of week, quarter, month, year, week of year.



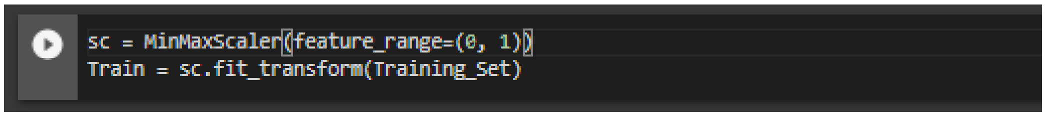
```
[ ] data_Daily.reset_index(level=0, inplace=True)

[ ] dataset = data_Daily
    dataset["Month"] = pd.to_datetime(data_Daily["Date"]).dt.month
    dataset["Year"] = pd.to_datetime(data_Daily["Date"]).dt.year
    dataset["Date"] = pd.to_datetime(data_Daily["Date"]).dt.date
    dataset["Time"] = pd.to_datetime(data_Daily["Date"]).dt.time
    dataset["Week"] = pd.to_datetime(data_Daily["Date"]).dt.week
    dataset["Day"] = pd.to_datetime(data_Daily["Date"]).dt.day_name()
    dataset = data_Daily.set_index("Date")
    dataset.index = pd.to_datetime(dataset.index)
    dataset.head(1)

                 index  Energy_kWh  Month  Year      Time  Week     Day
          Date
    2018-01-01       0       6.504      1  2018  00:00:00     1  Monday
```

The last 200 records of the dataframe are assigned to the testData variable which is going to be used for testing the model. Rest of the data is used for training the model.



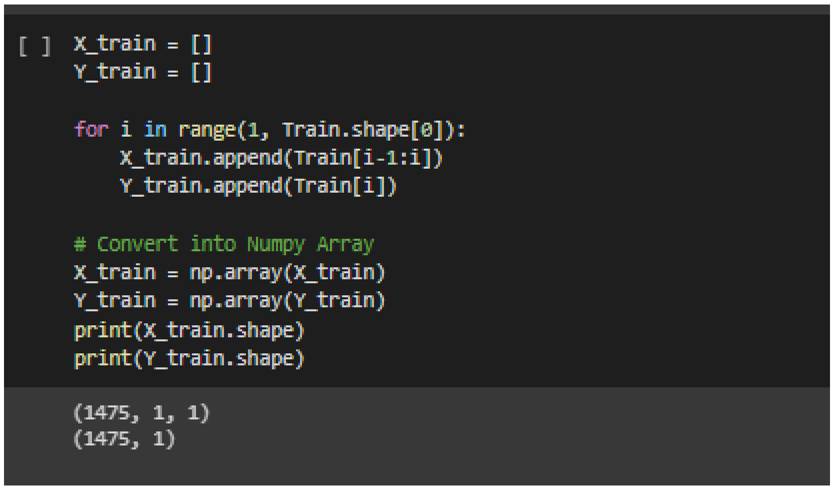
```
[ ] TestData = dataset.tail(199)

    Training_Set = dataset.iloc[:,0:1]

    Training_Set = Training_Set[:-199]
```

The energy column in the training dataframe is scaled and the new values will be in the range of 0 and 1. It will help in preserving the shape of the data and help in preventing the outliers.



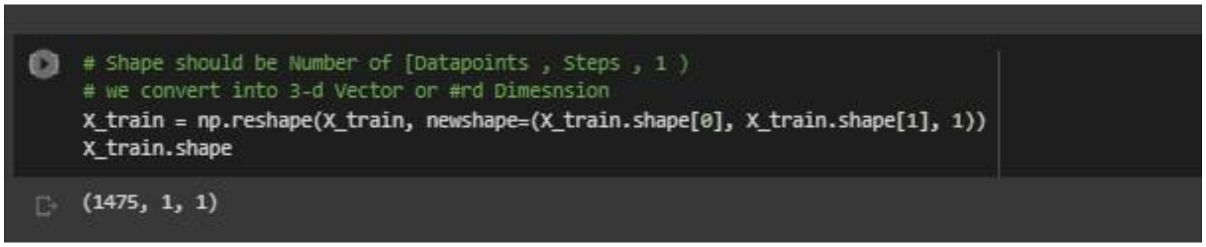
```
sc = MinMaxScaler(feature_range=(0, 1))
Train = sc.fit_transform(Training_Set)
```

X_train and Y_train has been crerated for the training data. Xtrain is a 3D vector and Y train is a 2D vector. This can be seen in the console. The shape of x train is in 3D and the shape of y train is in 2D.



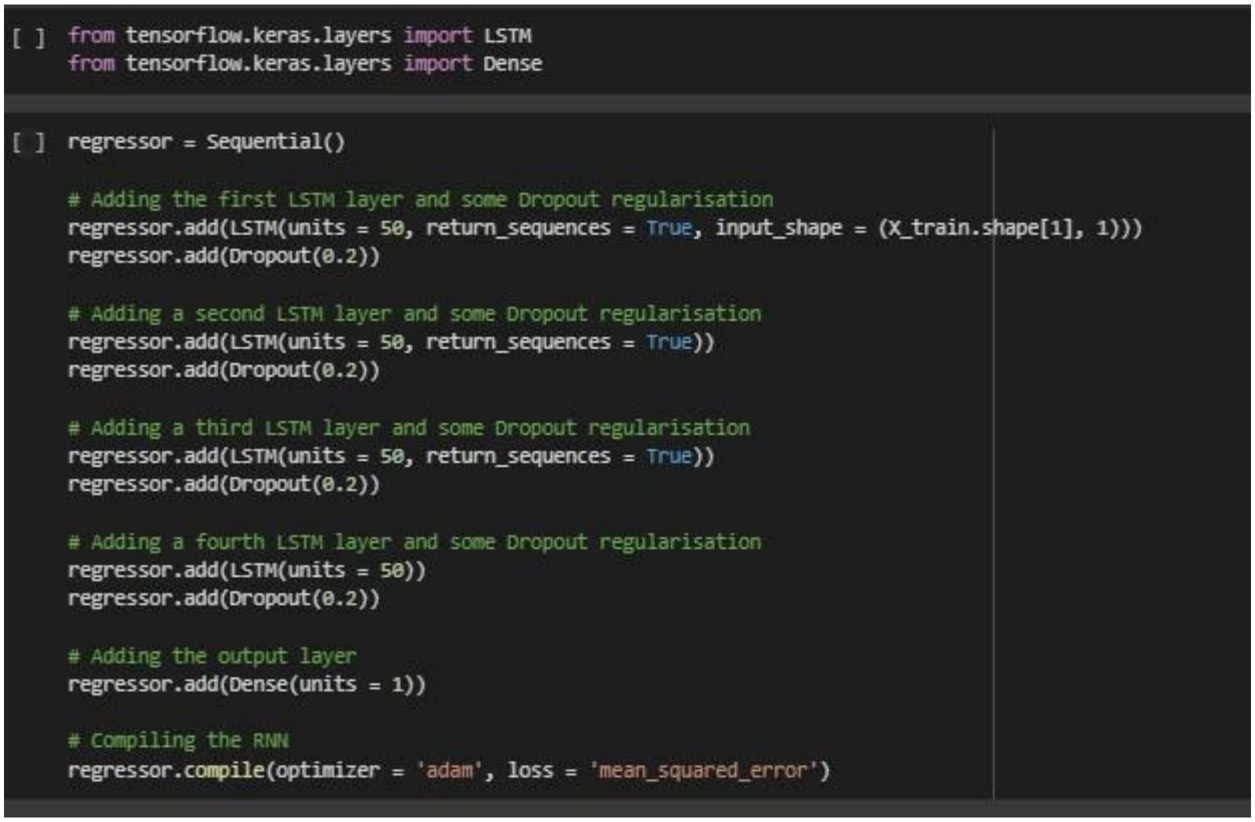
```
[ ] X_train = []
    Y_train = []

    for i in range(1, Train.shape[0]):
        X_train.append(Train[i-1:i])
        Y_train.append(Train[i])

    # Convert into Numpy Array
    X_train = np.array(X_train)
    Y_train = np.array(Y_train)
    print(X_train.shape)
    print(Y_train.shape)

    (1475, 1, 1)
    (1475, 1)
```

The X_train is reshaped. The shape of the number should be in the format (datapoints, steps, 1).



```
# Shape should be Number of [Datapoints , Steps , 1 )
# we convert into 3-d Vector or #rd Dimesnsion
X_train = np.reshape(X_train, newshape=(X_train.shape[0], X_train.shape[1], 1))
X_train.shape

(1475, 1, 1)
```

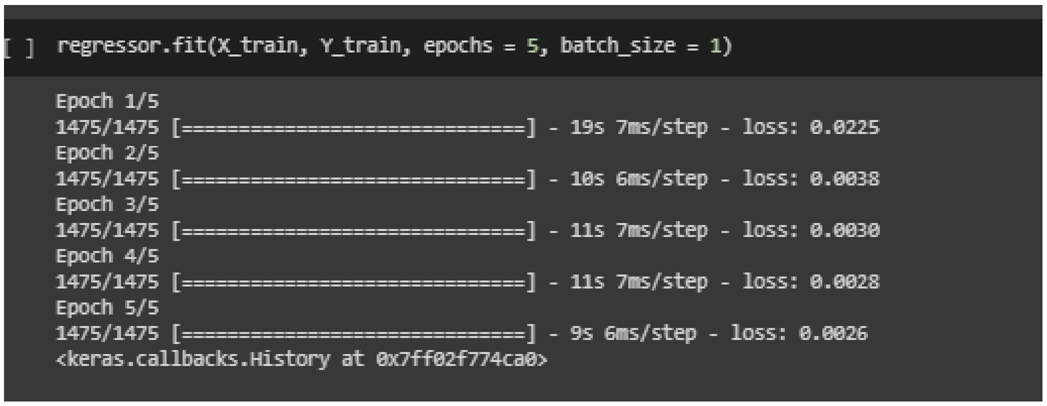
The model is created with four layers. Each model is given with 50 units as input and the drop is given as 0.2. dropout helps in preventing the overfitting of model. The dense layer is used by the model to learn the non linear mapping between the actual and predicted data. The model is then compiled with Adam optimizer and the loss function as mean squared error.



```
[ ] from tensorflow.keras.layers import LSTM
    from tensorflow.keras.layers import Dense
```

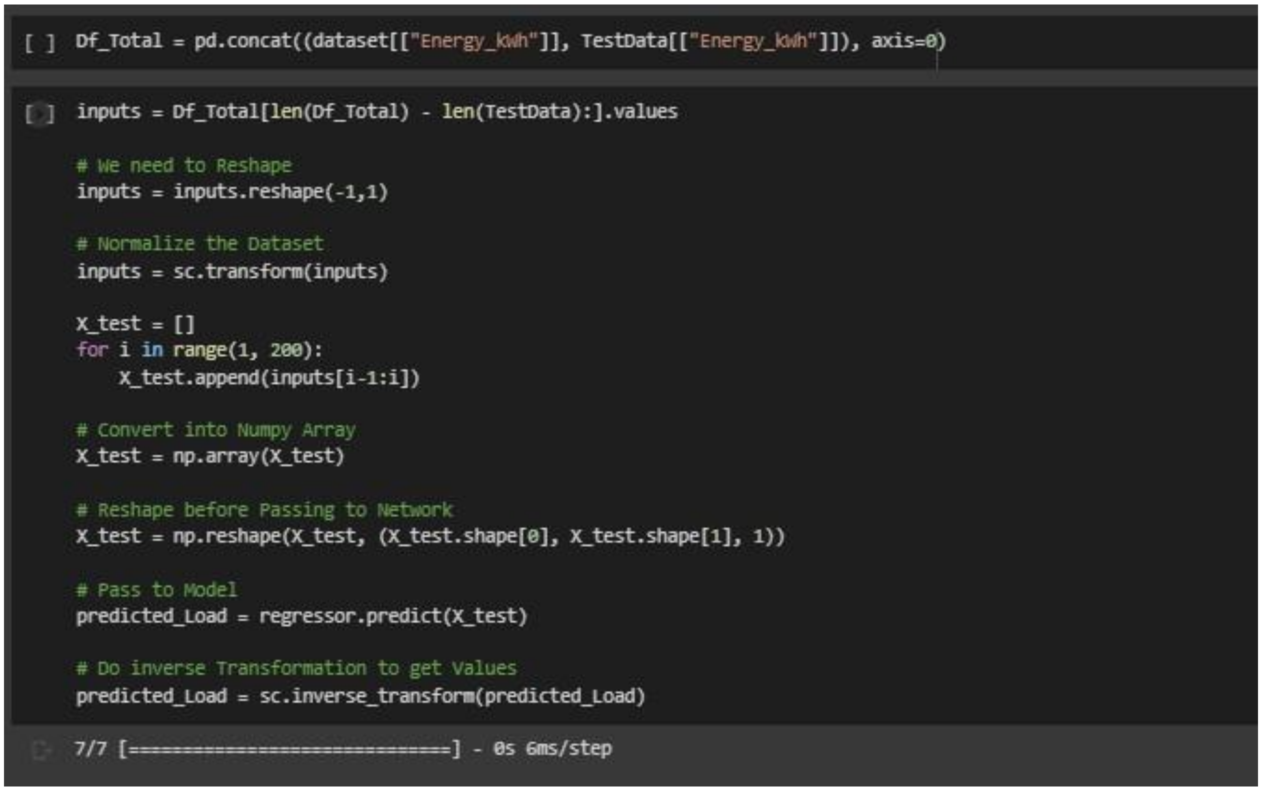
```
[ ] regressor = Sequential()

    # Adding the first LSTM layer and some Dropout regularisation
    regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
    regressor.add(Dropout(0.2))

    # Adding a second LSTM layer and some Dropout regularisation
    regressor.add(LSTM(units = 50, return_sequences = True))
    regressor.add(Dropout(0.2))

    # Adding a third LSTM layer and some Dropout regularisation
    regressor.add(LSTM(units = 50, return_sequences = True))
    regressor.add(Dropout(0.2))

    # Adding a fourth LSTM layer and some Dropout regularisation
    regressor.add(LSTM(units = 50))
    regressor.add(Dropout(0.2))

    # Adding the output layer
    regressor.add(Dense(units = 1))

    # Compiling the RNN
    regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

The Model is trained with the training data and ran with an epoch 5 and batch size of 1.



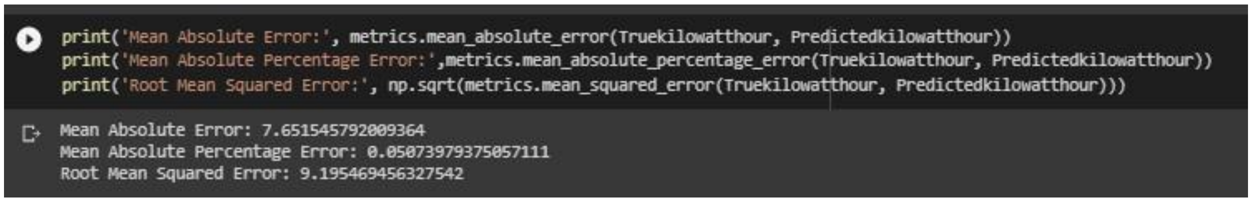
```
[ ] regressor.fit(X_train, Y_train, epochs = 5, batch_size = 1)

    Epoch 1/5
    1475/1475 [==============================] - 19s 7ms/step - loss: 0.0225
    Epoch 2/5
    1475/1475 [==============================] - 10s 6ms/step - loss: 0.0038
    Epoch 3/5
    1475/1475 [==============================] - 11s 7ms/step - loss: 0.0030
    Epoch 4/5
    1475/1475 [==============================] - 11s 7ms/step - loss: 0.0028
    Epoch 5/5
    1475/1475 [==============================] - 9s 6ms/step - loss: 0.0026
    <keras.callbacks.History at 0x7ff02f774ca0>
```

The values in the energy column of testing data is extracted and then the data is reshaped. The reshaped inputs are then transformed to normalize the inputs. The extracted data is added to x_test variable and then it is converted to an array. The 2D and 3D vectors of x_test are reshaped again and then passed to the predict method of the regressor. Then the result is inverse transformed to obtain the predicted values.



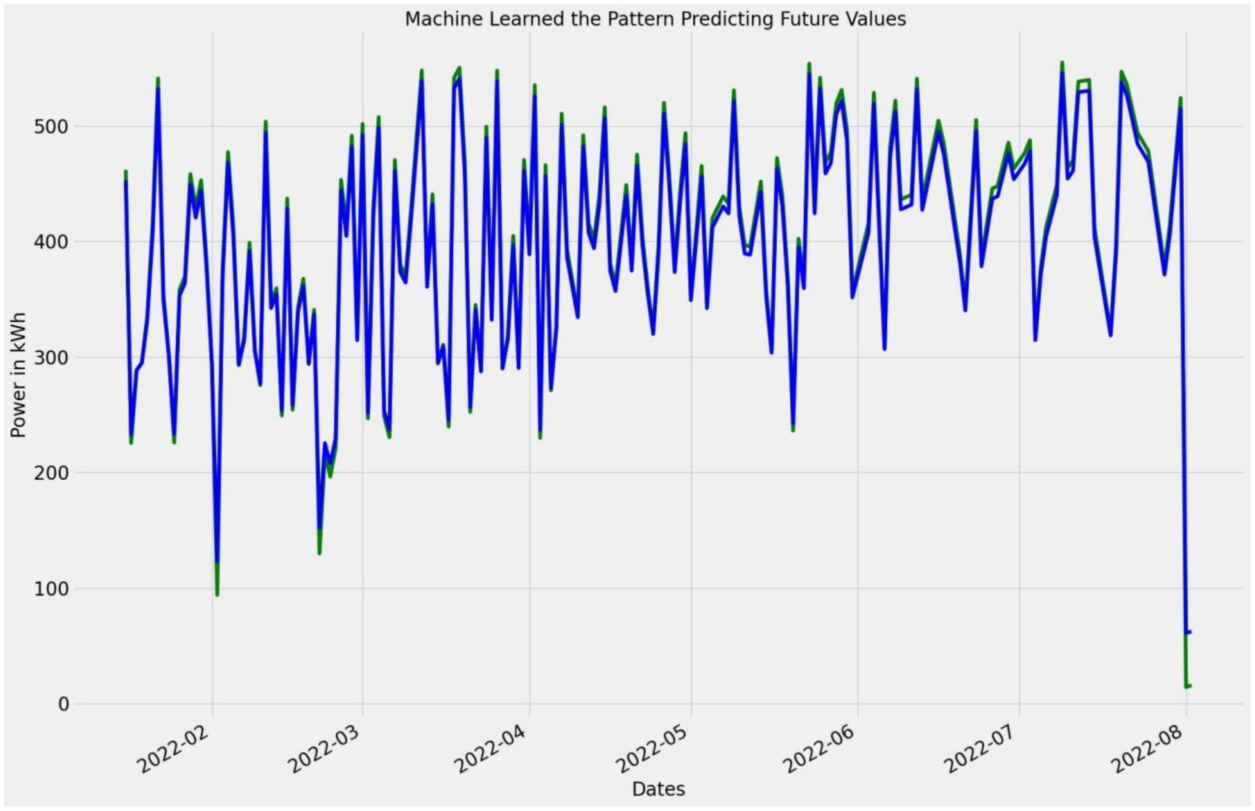
```
[ ] Df_Total = pd.concat((dataset[["Energy_kWh"]], TestData[["Energy_kWh"]]), axis=0)

[ ] inputs = Df_Total[len(Df_Total) - len(TestData):].values

    # We need to Reshape
    inputs = inputs.reshape(-1,1)

    # Normalize the Dataset
    inputs = sc.transform(inputs)

    X_test = []
    for i in range(1, 200):
        X_test.append(inputs[i-1:i])

    # Convert into Numpy Array
    X_test = np.array(X_test)

    # Reshape before Passing to Network
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

    # Pass to Model
    predicted_Load = regressor.predict(X_test)

    # Do inverse Transformation to get Values
    predicted_Load = sc.inverse_transform(predicted_Load)

    7/7 [==============================] - 0s 6ms/step
```

The performance of the model is evaluated by calculating the mean absolute error, mean absolute percentage error, and root mean squared error. From the results, the model has performed well. The error percentage and error values are very less compared to the previous model.



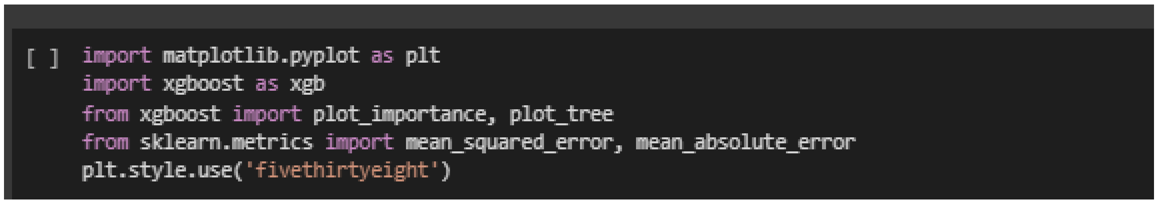
```
print('Mean Absolute Error:', metrics.mean_absolute_error(Truekilowatthour, Predictedkilowatthour))
print('Mean Absolute Percentage Error:',metrics.mean_absolute_percentage_error(Truekilowatthour, Predictedkilowatthour))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Truekilowatthour, Predictedkilowatthour)))

Mean Absolute Error: 7.651545792009364
Mean Absolute Percentage Error: 0.05073979375057111
Root Mean Squared Error: 9.195469456327542
```

The graph for the actual and predicted values is plotted below. The blue line represents the predicted values, and the green line represents the actual values. From the graph it is evident that the predicted values are much closer to the actual values.



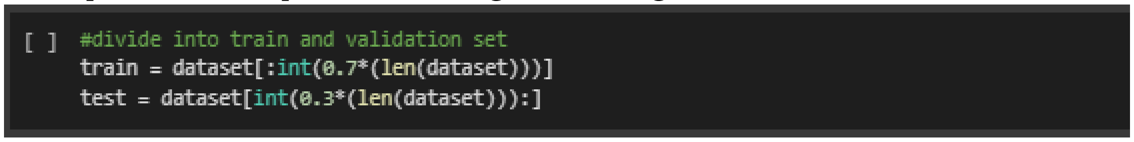
5.2.3 XGBoost

The necessary modules are imported.



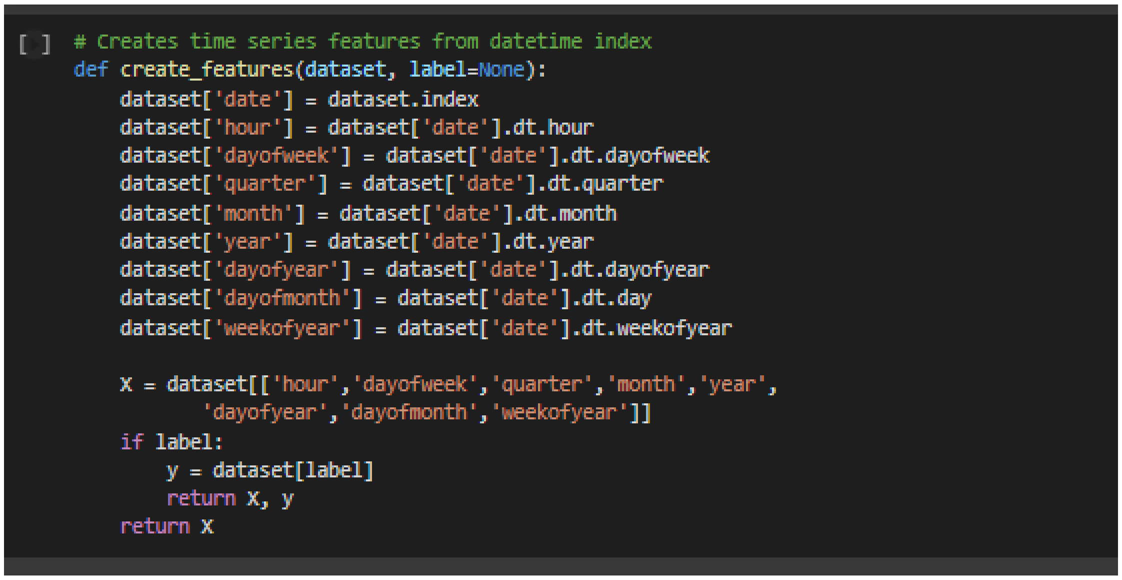
```
[ ] import matplotlib.pyplot as plt
    import xgboost as xgb
    from xgboost import plot_importance, plot_tree
    from sklearn.metrics import mean_squared_error, mean_absolute_error
    plt.style.use('fivethirtyeight')
```

The input data set is spitted into testing and training datasets in the ratio of 7 and 3.



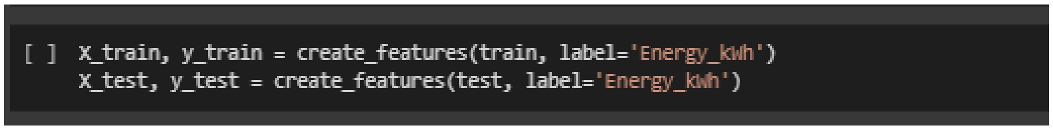
```
[ ] #divide into train and validation set
    train = dataset[:int(0.7*(len(dataset)))]
    test = dataset[int(0.3*(len(dataset))):]
```

Created a method for adding the date dimension parameters to the dataset. The date dimension table will have the columns date, hour, day of week, quarter, month, year, week of year. This method will return only x if the dataset does not have any labels. If the dataset has label, then this method will return the dataset along with the label.



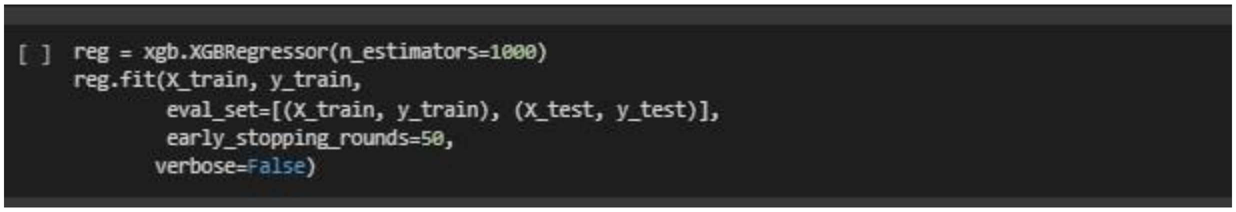
```
[ ] # Creates time series features from datetime index
    def create_features(dataset, label=None):
        dataset['date'] = dataset.index
        dataset['hour'] = dataset['date'].dt.hour
        dataset['dayofweek'] = dataset['date'].dt.dayofweek
        dataset['quarter'] = dataset['date'].dt.quarter
        dataset['month'] = dataset['date'].dt.month
        dataset['year'] = dataset['date'].dt.year
        dataset['dayofyear'] = dataset['date'].dt.dayofyear
        dataset['dayofmonth'] = dataset['date'].dt.day
        dataset['weekofyear'] = dataset['date'].dt.weekofyear

        X = dataset[['hour','dayofweek','quarter','month','year',
               'dayofyear','dayofmonth','weekofyear']]
        if label:
            y = dataset[label]
            return X, y
        return X
```

The date dimension attributes are added to both training dataset and testing dataset using the above method.



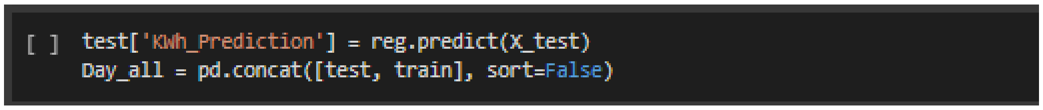
```
[ ] X_train, y_train = create_features(train, label='Energy_kWh')
    X_test, y_test = create_features(test, label='Energy_kWh')
```

The model is fed with 1000 estimators. The model us asked to evaluate with 50 stopping rounds. This parameter specifies the number of iterations with no improvement on the validation set after which the training will be stopped.



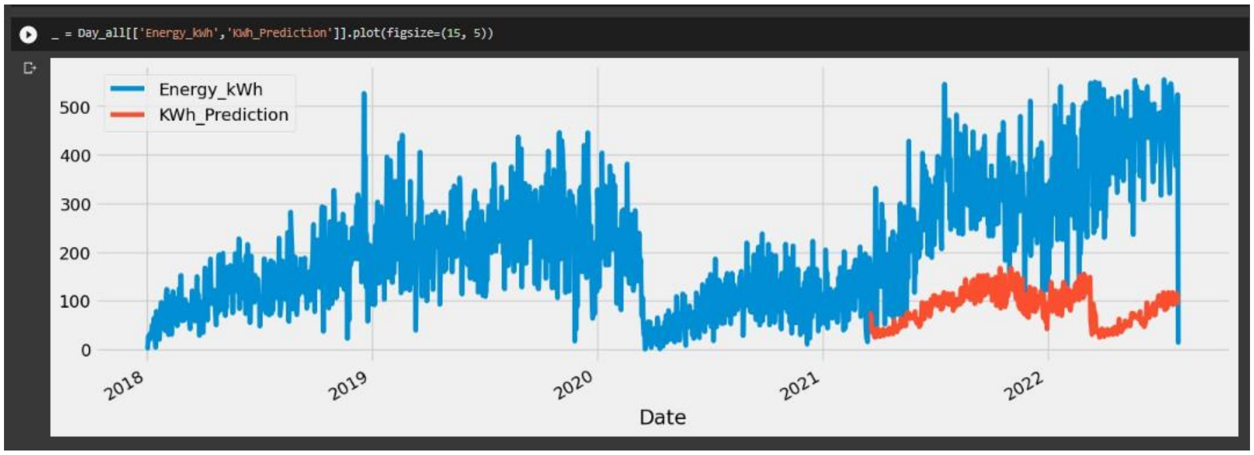
```
[ ]  reg = xgb.XGBRegressor(n_estimators=1000)
     reg.fit(X_train, y_train,
             eval_set=[(X_train, y_train), (X_test, y_test)],
             early_stopping_rounds=50,
            verbose=False)
```

Predictions of the model has been obtained for the testing data.



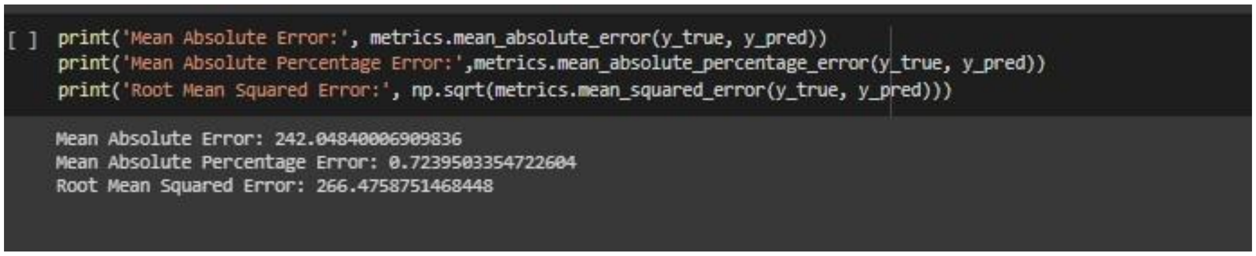
```
[ ]  test['KWh_Prediction'] = reg.predict(X_test)
     Day_all = pd.concat([test, train], sort=False)
```

The plot has been created for the actual and predicted values. Blue data is the actual data and the orange is the predicted data. From the figure it is evident that the predictions are not close to the actual values.



```
_ = Day_all[['Energy_kWh','KWh_Prediction']].plot(figsize=(15, 5))
```

The evaluation metrics are performed on the predictions and found the mean absolute error and mean absolute percentage error and the root mean squared error. The values are much higher compared to the LSTM model. Therefore, this model is not fit for predicting the load of the charging stations.



```
[ ]  print('Mean Absolute Error:', metrics.mean_absolute_error(y_true, y_pred))
     print('Mean Absolute Percentage Error:',metrics.mean_absolute_percentage_error(y_true, y_pred))
     print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_true, y_pred)))

     Mean Absolute Error: 242.04840006909836
     Mean Absolute Percentage Error: 0.7239503354722604
     Root Mean Squared Error: 266.4758751468448
```

Chapter 6

6.1 Conclusion:

From the results of the XGBoost, Arima and LSTM models, LSTM model is the more efficient model which is predicting 93% close to the actual values. Therefore, LSTM model is suitable for predicting the future load of the charging stations.

6.2 Project management:

Work completed:

- Responsibilities took by Sreekar Thanda are Data gathering and Pre-processing and building ARIMA model.
- Data Cleaning and work on LSTM model is done by Vishnu Sudireddy.
- Vydya Abhiram worked on plotting the data and finding the dependencies and worked on LSTM model in identifying the best epoch and optimizer combination.
- Sai Vishwak worked on XGBoost model and ARIMA model.

6.3 References:

[1] Anwar, Tahreem Sharma, Bhaskar Chakraborty, Koushik Sirohia, Himanshu. (2018). Introduction to Load Forecasting. International Journal of Pure and Applied Mathe- matics. 119. 1527-1538.

[2] http://engineering.electrical-equipment.org/electrical-distribution/electric-load-forecasting-advantages-challenges.html

[3] https://www.kaggle.com/jeanmidev/smart-meters-in-london

[4] E. S. Xydas, C. E. Marmaras, L. M. Cipcigan, A. S. Hassan and N. Jenkins, ” Electric vehicle load forecasting using data mining methods, ” IET Hybrid and Electric Vehicles Conference 2013 (HEVC 2013), London, 2013, pp. 1-6.

[5] https://darksky.net/dev

[6] F. Xie, M. Huang, W. Zhang and J. Li, ” Research on electric vehicle charging station load forecasting, ” 2011 International Conference on Advanced Power System Automation and Protection, Beijing, 2011, pp. 2055-2060.

[7] https://acorn.caci.co.uk

[8] K. Goswami, A. Ganguly and A. K. Sil, ” Day Ahead Forecasting and Peak Load Management using Multivariate Auto Regression Technique, ” 2018 IEEE Applied Signal Processing Conference (ASPCON), Kolkata, India, 2018, pp. 279-282.

[9] https://otexts.com/fpp2/arima.html

[10] Demand Calculation Method for Electric Vehicle Charging Station Locating and Deployment (researchgate.net)

[11] https://www.researchgate.net/publication/267928696_Electric_vehicles'_energy_consumption_measurement_and_estimation

[12] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7001349

[13]
https://www.researchgate.net/publication/283028174_Deploying_public_charging_stations_for_electric_vehicles_on_urban_road_networks

[14] C. Hor, S. J. Watson and S. Majithia, " Daily Load Forecasting and Maximum Demand Estimation using ARIMA and GARCH, " 2006 International Conference on Probabilistic Methods Applied to Power Systems, Stockholm, 2006, pp. 1-6.

[15] https://towardsdatascience.com/understanding-neural-networks-19020b758230

[16] V. Dehalwar, A. Kalam, M. L. Kolhe and A. Zayegh, " Electricity load forecasting for Urban area using weather forecast information, " 2016 IEEE International Conference on Power and Renewable Energy (ICPRE), Shanghai, 2016, pp. 355-359.

[17] https://www.sciencedirect.com/topics/social-sciences/regression-model

[18]https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning 6a6e67336aa1

[19]
https://www.researchgate.net/publication/224160777_Electric_Energy_and_Power_Consumption_by_Light-Duty_Plug-In_Electric_Vehicles