BIG DATA ANALYTICS

Data-driven model validated with real-world smart meter data for demand and response-based energy consumption for smart buildings with Big data Analytics

C3 REPORT

Submitted by
Group 10
Kalyani Pharkandekar IIT2020196
Priyamvada Priyadarshani IIB2020037
Dhamapurkar Nikita Suresh IIB2020042
Niharika Mangarai IIB2020011

Under the
Guidance & Supervision of
Prof. Dr. Sonali Aggarwal,
Mr. Shashi Shekhar Kumar and Mr. Ritesh Chandra

TABLE OF CONTENTS

A. Abstract

Due to increasing global energy demand and concerns about environmental security, it has become necessary to develop better and more efficient energy systems. A great way to achieve this can be found in smart homes with the latest technology and communications. This study leverages big data analytics to propose a data-driven approach to modeling and optimizing demand response based on energy use in smart buildings. Data from the world's smart meter will be used to develop and evaluate learning models that can predict energy demand and adjust energy use potential according to grid prices or peak demand times. The aim of this project is to demonstrate how big data-driven analytics can be used to improve energy management in smart buildings, thereby reducing energy costs and improving grid regulation. We used pyspark's MLib framework to predict future demand so we can match resources with performance.

B. Introduction

In today's modern world the main focus of our research and development is to make our lifestyle easier. Smart buildings are result of one of these technologies. In smart buildings, the whole building is "smart" as the name suggests. That is, it can detect our needs without we performing any action. For example, if we enter our apartment we won't be required to switch on lights or fans or Air conditioner or any of the electrical appliances that we generally use,and the automated system itself will switch on fan or Air Conditioner or room heater based on some data about the people living in the apartment and the weather etc.

As we can see, previous data storing and data prediction has a huge role in this problem. As data will be huge, we can only use Big Data Analysis for the processing. Currently we are processing historic data, but in future we might need this model for real time processing,i.e., even though now we only need to do batch processing on the data, in future we might need to do stream processing on the data. Because of this future scope, pySpark is the most suitable framework for this project, thus the framework we are going to use in this project.

The main purpose of this model, the project we are making is to analyze the data of power consumption and make the appropriate changes after analyzing the data.

C. Related work

The National Smart Grid Mission (NSGM), launched by the government of India in 2015, is at the forefront of changing the country's energy landscape through the implementation of smart grids. The aim of this mission is to improve the energy network, integrate renewable energy through distribution and provide technologies such as smart meters to customers, ultimately promoting energy efficiency and security.

At the same time, our work is of particular importance as it focuses on energy efficiency, especially for smart homes. Leveraging PySpark's extensive data analysis capabilities can add a layer of intelligence to your project. Integrating information on real-world intelligence metrics to ensure the model is based on real-world performance rather than relying on theoretical assumptions.

In line with NSGM's objectives, my project contributes to the goal of improving energy efficiency standards. A recent development in Bareilly, India, shows the aggressive installation of smart meters under the Smart Grid programme, a collaboration between the government and L&T (Larsen & Toubro). The initiative, which provides instant energy information to customers and assists operators with grid management, is an important step in the development of electricity.

As India reaches the milestone of 5 million smart meter installations, the state of Uttar has emerged as a leader demonstrating the widespread use of smart grids. A number of strategic initiatives and decisions reflect the country's determination, which is in line with the overall mission of NSGM.

Importantly, our project not only follows the vision of the National Smart Grid Mission, but also directly contributes to creating a smarter and more efficient energy system. May it be good for India.

D. Comparison with existing work or base paper

S.No	Paper Title	Dataset used	Methods/Approach used	Achieved Performance	Advantages and Disadvantages
1[5]	A Big Data Platform for Smart Meter Data Analytics	The SSE database	Modular Core-Broker-Client Architecture.The study utilizes Hadoop and a hybrid RDBMS with MapReduce integration for storing smart meter data, employing CSV files and HBase. Analytic options include Custom Hadoop MapReduce applications and MapReduce-based BI tools.	SMASH, operating under the broker pattern, demonstrates robust performance in data storage and querying, outperforming comparable products.	Innovative Architecture: Core-broker-client design enhances flexibility. Real-World Focus: Strengthens relevance with UK's electricity data.
2[6]	Smart meter data-driven evaluation of operational demand response potential of residential air conditioning loads	The ground truth data	discusses existing methods for ACLs extraction, including non-intrusive load monitoring (NILM), Markov-based, and regression methods	Results also demonstrate that great difference in terms of ACLs usage patterns (a total of 19 patterns), DR potential (a maximum of 0.7 kW) results from different DR duration and operational conditions	Accurate ACLs Disaggregation,Versatile Application . Implementation Complexity
3[4]	A Big Data Approach for Demand Response Management in Smart Grid Using the Prophet Model Electricity consumption three months (February–April 2013) (July 2009–June 2013)	Electricity consumption three months (February –April 2013) (July 2009–June 2013)	The ARIMA model is employed for time series data prediction, applicable to both linear and multiple-regression models. Prophet, developed by Facebook, addresses ARIMA limitations and works through an additive model with non-linear trends and seasonality	Performance metrics like mean square error and mean absolute error indicate Prophet's superior accuracy over ARIMA	Performance metrics like mean square error and mean absolute error indicate Prophet's superior accuracy over ARIMA

E. Architecture of proposed model

Spark Architecture

Apache Spark has resilient distributed database rates as its heart does faster processing due to the in-memory usage. It is a batch processing system at heart and converts the real-time data stream into window batches for processing. It is also lambda architecture. Some of its main components are hdfs or Hadoop over which it runs, yarn/mesos/standalone manager for resource management. SparkMl/Mlib for machine learning libraries, GraphX for graph processing, SparkSQL for SQL querying, and Spark streaming for data stream processing. All these extensions and their drivers and executors are handled by Spark Core responsible for overall management. We are using a standalone single-machine spark. Resilient Distributed Datasets (RDDs) and Directed Acyclic Graphs (DAGs) help us decrease computation cost of problem, make it fault tolerant and helps in optimization.

We utilized Google Colab, or Colaboratory, as a free cloud-based platform for our research. It allowed us to work with large datasets and complex tasks without the need for costly hardware. With the "!pip install pyspark" command, We installed PySpark, which is essential for big data processing and machine learning tasks.

Moreover, Colab is known for its cloud-based, scalable solution. It provides access to high-performance Spark clusters without the burden of infrastructure management. This was particularly valuable for our work, which involved large datasets and intricate machine-learning models.

By establishing a SparkSession, We seamlessly integrated PySpark's capabilities with Colab's collaborative features. This combination enabled us to efficiently preprocess data, train models, and perform in-depth analysis.

Master-Slave Architecture

The master-slave architecture has two components:

Driver: Central controller & responsible for analyzing the user's program code, converting it into work, and task scheduling on worker nodes.

Executors: Worker processes that perform actual work or program & are responsible for processing and managing the data stored in memory.

Resilient Distributed Dataset (RDD)

Resilient Distributed Dataset, or RDD(immutable), is regarded as the foundation of Apache Spark and is largely responsible for its immense popularity and success. Data may be partitioned and processed in parallel over a cluster of computers thanks to RDDs, which offer a basic distributed data abstraction. RDDs can retrieve and recalculate lost data using lineage information in the event of a node failure. Map, filter, reduce, and groupByKey are just a few of the many data transformation operations that RDDs support. Spark is an effective tool for data analysis because of these transformations, which are used to process and change data. An essential component of iterative machine learning algorithms is the ability of RDDs to be cached in memory. By eliminating the need to frequently read data from disc, this in-memory caching greatly enhances performance. On top of RDDs, Spark provides a robust ecosystem of libraries, including GraphX, Spark Streaming, MLlib (Machine Learning Library), and Spark SQL. For various use cases and areas, these libraries offer specialised functionalities. Transformations called on RDDs are not performed instantly upon call since they are lazy-assessed. Rather, they are executed and documented when a task is completed. Lazy evaluation aids in computation optimization. Operations known as actions start transformations in motion, return a value to the driver program, or write data to an external storage system.

Directed Acyclic Graph (DAG)

For transformations, Spark adds them to a DAG of computation and only when driver requests some data, does this DAG actually gets executed. Spark is lazy. Lineage Graph in Spark is a DAG, where vertices (representing RDDs or DataFrames) are connected by directed edges, illustrating dependencies. This acyclic structure ensures no circular dependencies. It plays a crucial role in Spark's fault tolerance by allowing recreation of lost RDDs through tracing back to their parent RDDs. Additionally, the Lineage Graph aids optimization by reducing data shuffling and increasing parallelism, resulting in faster execution and efficient cluster resource utilization.

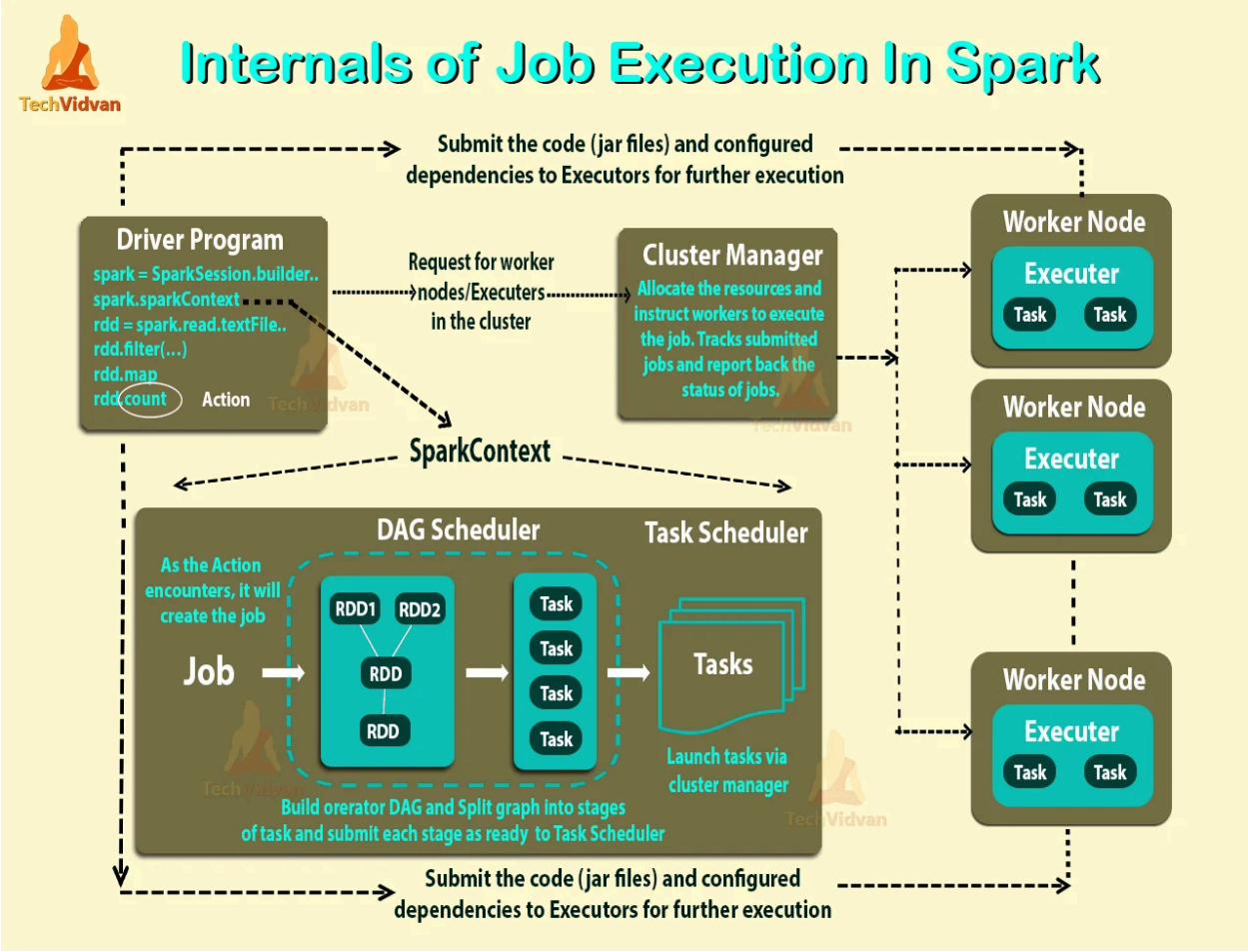


Fig: Spark Architecture & Internal Working Source: [Spark Architecture & Internal Working - TechVidvan]

In conclusion, the Spark architecture is a phenomenal & diversely equipped distributed data processor which is batch processing engine at heart but can do real stream processing as well. Master-slave architecture, flexible distribution of datasets, shared directed acyclic graphs, in-memory and checkpinting allow it to process huge and big datasets efficiently and fault-tolerantly.

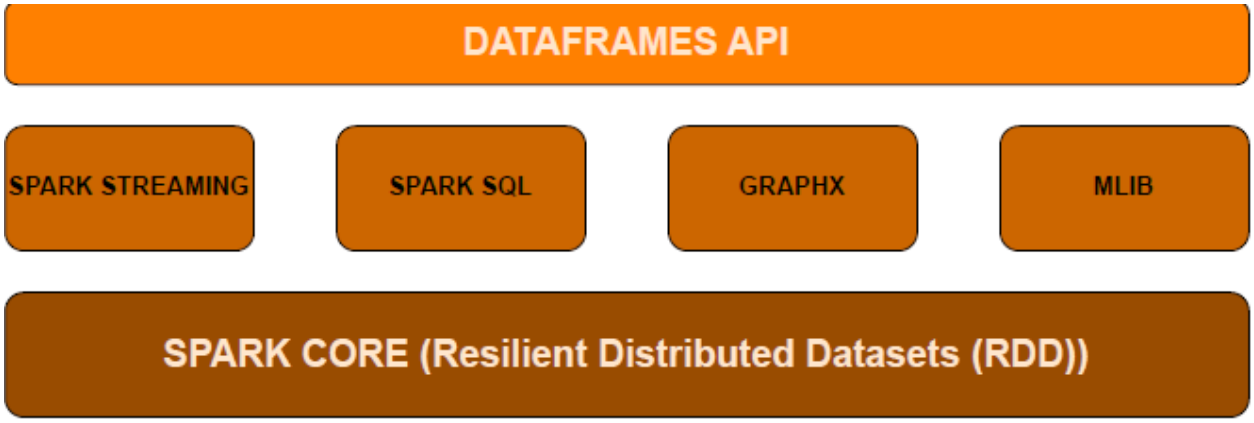


Fig 2: Framework of Apache Spark & its API

MLib

MLlib is a machine learning library built on the Apache Spark framework is lased with various tools that make it easy and scalable.

- Featurization: Operations for transformation, feature extraction, feature engineering, dimensionality reduction, Standard scaler and feature selection based on various techniques.
- Pipelines: Vector assembler, oneHotEncode, stringIndexer, and pipelines
- Persistence: it can allow for saving and loading algorithms, models, and Pipelines.
- ML Algorithms: Standard learning algorithms such as regression classification, and clustering etc.
- Utilities: It also allows for linear algebra, data handling, statistics etc.
- MLlib's architecture can be divided into three layers: Data layer, Algorithm layer, and Application layer.

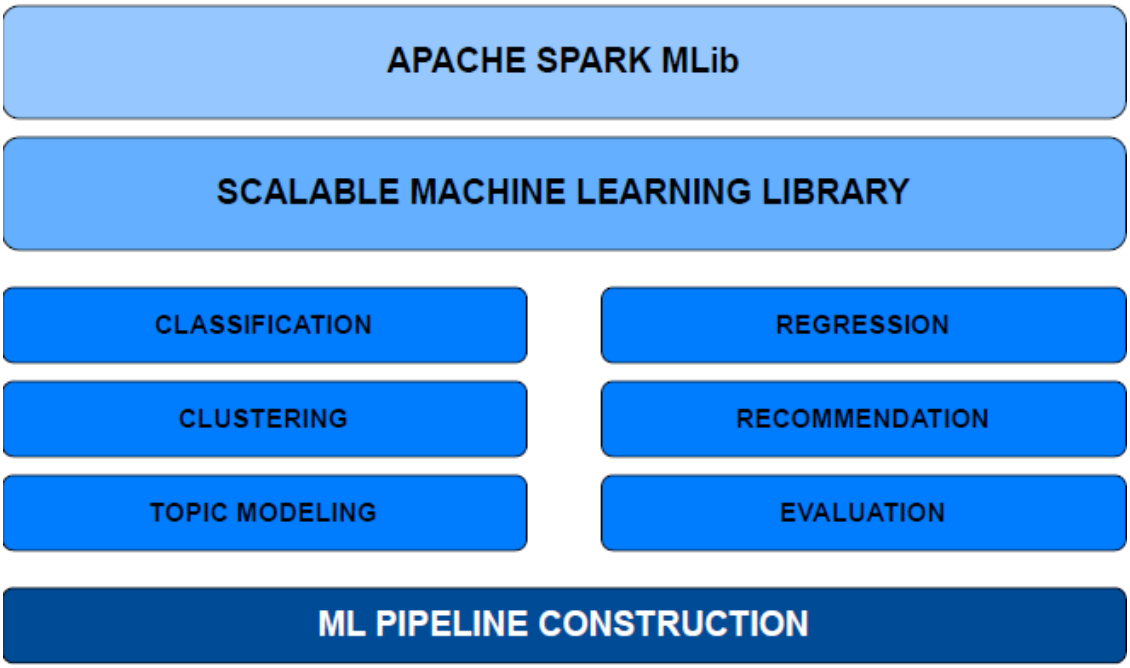


Fig 3. MLib Functionalities

Architecture diagram for the proposed methodology

The use of Spark and MLlib's prediction algorithm involves a systematic process. It begins with Data Retrieval and Preprocessing, where data is obtained locally or from various sources using Spark APIs. A careful cleaning and preprocessing stage follows, addressing missing values, outliers, and applying transformations to ensure data quality.

Next, in the Feature Engineering phase, key features representing different targets are identified. Dynamic transformation techniques, such as scaling, are applied to optimize the representation of these features. Moving on to Model Training and Selection, an appropriate ML algorithm is chosen from MLlib, considering the specific prediction task (e.g., classification, regression). The data is then divided into training and testing sets, and Spark's distributed resources are utilized for the training process. Model evaluation, using metrics like accuracy, AUC, or R^2, results in the selection of the most suitable model.

The subsequent Model Tuning and Optimization phase involves fine-tuning hyperparameters to enhance performance and generality. Techniques like grid search or random search are explored to optimize the model's configuration.

In the Model Deployment and Prediction phase, refined models are deployed for production, enabling predictions on new data. Spark's parallel processing capabilities expedite predictions on large datasets, and real-world data tests are conducted with ongoing performance monitoring.

Finally, in Model Development and Maintenance, a continuous monitoring process is established to assess performance against new data. The model is retrained with updated data as necessary, utilizing feedback to enhance accuracy and adapt to evolving data patterns. This holistic approach ensures the efficacy, adaptability, and longevity of predictive models in real-world applications.

Advantages of Spark and Mlib:

Scalability: Ability to efficiently process large datasets using Spark's distributed computing framework.Python’s Pandas cannot even contain the vast data that pyspark dataframe handles.
Performance: Leveraging parallelism
Flexibility: Various models and algorithms
Fault Tolerance: RDD & Lineage graph
By leveraging Spark and Mlib, creating, deploying and managing high-performance prediction algorithms becomes seamless and scalable even for large-scale data requests.

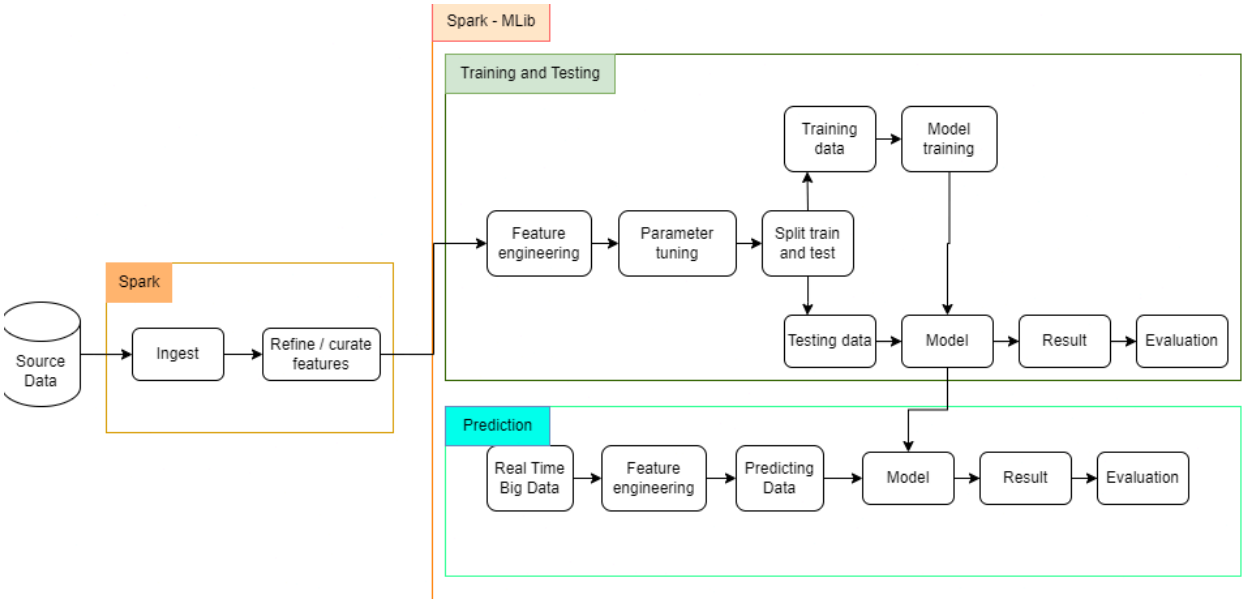


Fig 4: General Machine Learning methodology followed in spark

F. Dataset Description

About the dataset:

This compilation comprises energy consumption patterns recorded by smart meters in the Bareilly and Mathura districts of Uttar Pradesh, India, spanning from May 2019 to October 2021. The dataset consists of eight CSV files, each detailing power consumption, voltage, current, and frequency recorded by smart meters at three-minute intervals. This data serves as a valuable resource for conducting research and analysis related to energy models and products.

```
# Load data
df = spark.read.csv("CEEW - Smart meter data Bareilly 2020.csv", header=True, inferSchema=True)
# Load other datasets in a similar manner
```

```
[ ]
df.show()
```

```
+-------------------+-----+-------------------+-------------------+-----------+-----+
|        x_Timestamp|t_kWh|z_Avg Voltage (Volt)|z_Avg Current (Amp)|y_Freq (Hz)|meter|
+-------------------+-----+-------------------+-------------------+-----------+-----+
|2020-01-01 00:00:00|0.002|             251.26|               0.15|      49.97| BR02|
|2020-01-01 00:03:00|0.001|             251.23|               0.15|      49.94| BR02|
|2020-01-01 00:06:00|0.001|             251.55|               0.14|      49.94| BR02|
|2020-01-01 00:09:00|0.001|             251.97|               0.14|      50.09| BR02|
|2020-01-01 00:12:00|0.002|             252.03|               0.14|      50.08| BR02|
|2020-01-01 00:15:00|0.001|             251.78|               0.14|       50.0| BR02|
|2020-01-01 00:18:00|0.001|             251.75|               0.13|      49.97| BR02|
|2020-01-01 00:21:00|0.001|             251.95|               0.14|       50.0| BR02|
|2020-01-01 00:24:00|0.002|             251.81|               0.14|      49.96| BR02|
|2020-01-01 00:27:00|0.001|             252.05|               0.13|      49.95| BR02|
|2020-01-01 00:30:00|0.001|             252.15|               0.14|       50.0| BR02|
|2020-01-01 00:33:00|0.001|              252.1|               0.13|      49.98| BR02|
|2020-01-01 00:36:00|0.001|             252.31|               0.13|       50.0| BR02|
|2020-01-01 00:39:00|0.002|             252.57|               0.12|      50.02| BR02|
|2020-01-01 00:42:00|0.001|             252.74|               0.13|      50.04| BR02|
|2020-01-01 00:45:00|0.001|             252.84|               0.12|      50.05| BR02|
|2020-01-01 00:48:00|0.001|             252.78|               0.13|      50.04| BR02|
|2020-01-01 00:51:00|0.001|              253.0|               0.13|       50.0| BR02|
|2020-01-01 00:54:00|0.001|             253.29|               0.12|       50.0| BR02|
|2020-01-01 00:57:00|0.001|              253.4|               0.14|      49.98| BR02|
+-------------------+-----+-------------------+-------------------+-----------+-----+
only showing top 20 rows
```

Fig 5: Screenshot of code showing Rows of the spark dataframe

```
df.describe()
```

```
DataFrame[summary: string, t_kWh: string, z_Avg Voltage (Volt): string, z_Avg Current (Amp): string, y_Freq
(Hz): string, meter: string, hour: string, day_of_week: string]
```

Fig 6: Datatypes of the dataframe columns

Description of the files:

The provided files contain data on smart meters in Bareilly and Mathura for the years 2019, 2020, and 2021. They include details on electricity consumption, voltage, current, and frequency. Specifically, "CEEW - Smart Meter Data Bareilly Aggreated.csv" compiles daily electricity consumption data for 2019, 2020, and 2021 in Bareilly. The focus of the work has been on this aggregated file and "CEEW - Bareilly Smart Meter Data 2020.csv."

```
df.describe()
```

```
DataFrame[summary: string, t_kWh: string, z_Avg Voltage (Volt): string, z_Avg Current (Amp): string, y_Freq
(Hz): string, meter: string, hour: string, day_of_week: string]
```

CEEW - Bareilly Smart Meter Data 2020.csv & CEW - Smart Meter Data Bareilly Aggreated.csv were used for our experiments.

G. **Experiment details & Graphs**

a. Supervised Learning Ml on high-frequency three minutes interval bareilly2020 dataset

Flowchart:
Below is the flow followed for predicting the demand from features using machine learning models

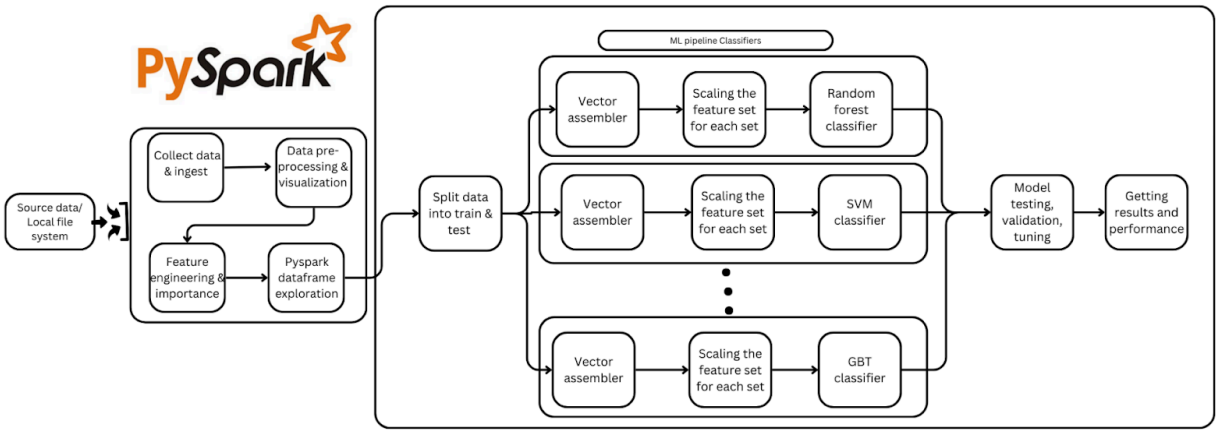


Fig 7: Proposed Methodology followed

Data preprocessing and feature engineering:

To complete the time analysis, we convert the "x_Timestamp" column to timestamp format. Then, important date and time-related attributes such as year, month, day, hour, minute, second, and day of the week are extracted from the time. These features improve the model's ability to capture physical patterns and structures. To make it easier to check the impact time, the code creates a custom window based on the "meter" column and sorts the data by timestamp. Additionally, a business line for calculating the time difference of the target difference ("t_kWh") of consecutive amounts has been introduced. This difference provides important time points for the modeling study. The resulting DataFrame supports these engineering tasks and provides a foundation for better model training and evaluation.

Data Exploration:

As you can see from the graph below, electricity consumption comparison by time of day (hour for each day of the week), where 1: "Monday", 2: "Tuesday", 3: "Wednesday" , 4: "Thursday", 5: "Friday", 6: "Saturday", 7: "Sunday";

We see that the busiest time every day is close to lunch time, especially in the evening after people work in the office, gas, television consumption etc. They return home for reasons. Additionally, electricity prices almost reach their highest level on Monday. Each observation can have many reasons.

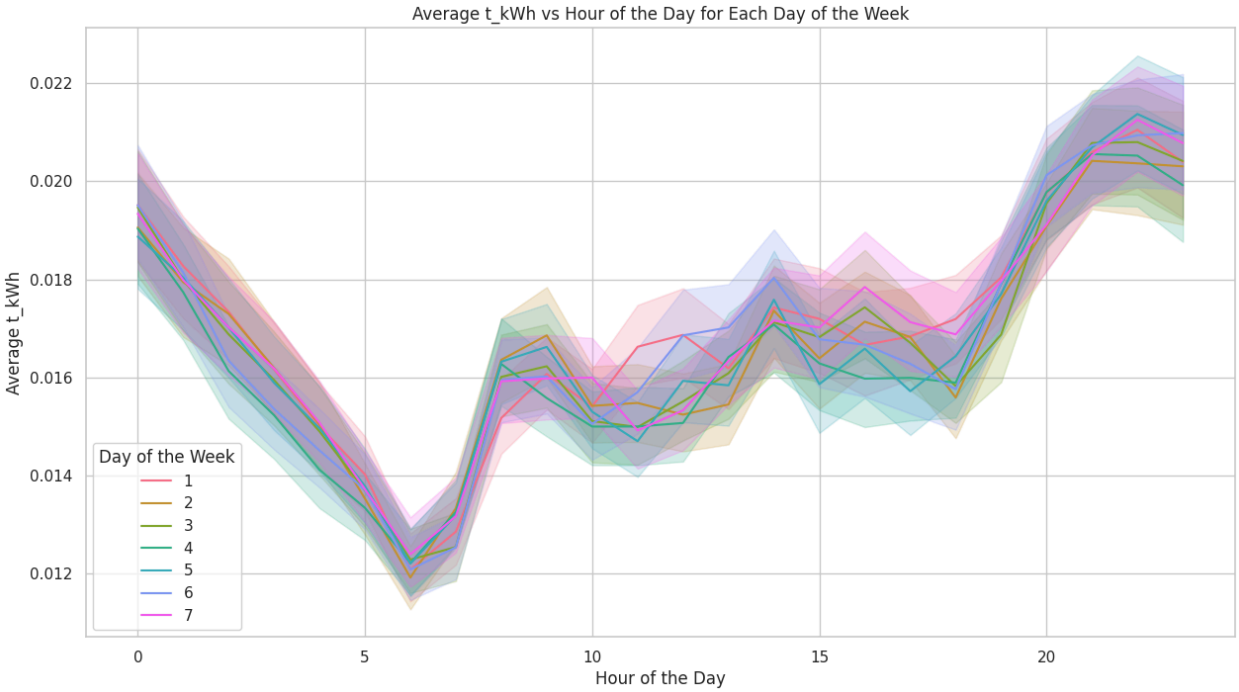


Fig 8: Average power consumption vs Time of the day for each day of the week

As you can see from the figure below, electricity consumption is compared to the time of day (hour) for each month of the year. We saw that electricity consumption is higher in summer nights due to the hot weather, thus AC power is needed. From May to September, summer and summer season people use fans, air conditioners, refrigerators, etc. forces you to use it. It is common in tropical countries of India, especially in hot cities like Bareilly. During the cold season, when the weather gets colder and electricity consumption increases in the morning, people turn on stoves, cook hot meals, and use geysers to get ready for work.

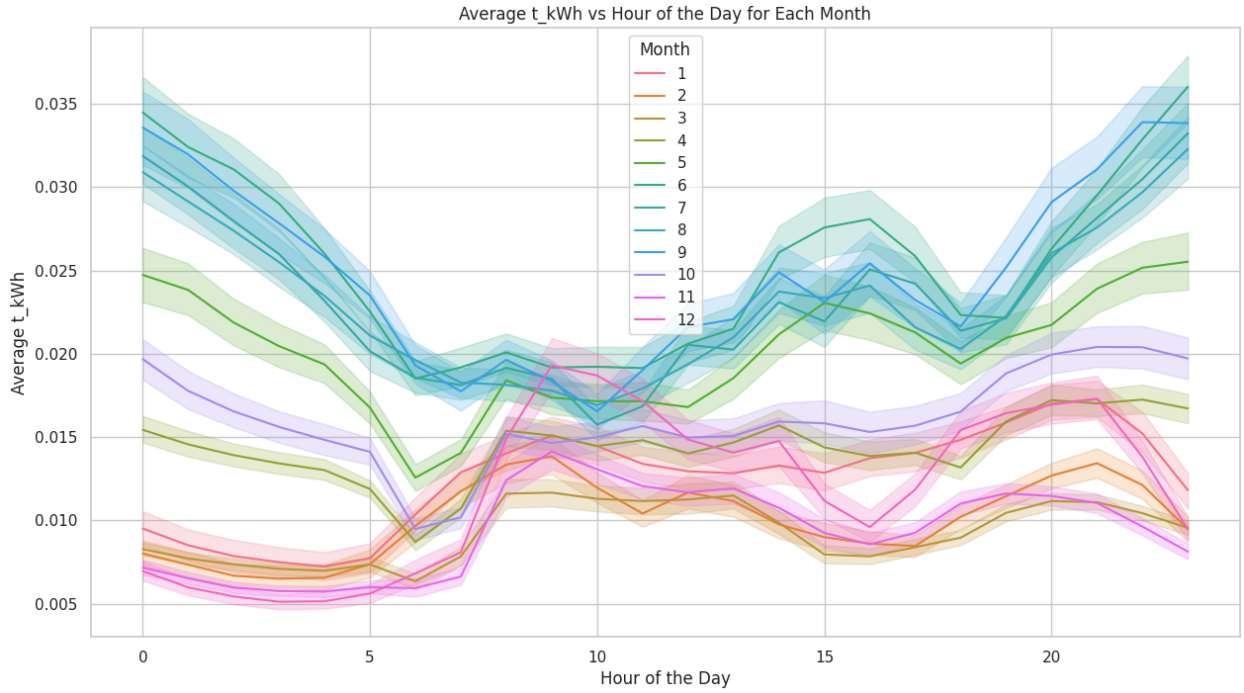


Fig 9: Average power consumption vs Time of the day for each month of the year

Demand-supply gap:

Demand and supply columns were calculated:

```
df2020 = df2020.withColumn("Supply", (col("z_Avg Voltage (Volt)") * col("z_Avg Current (Amp)"))/(1000))

df2020 = df2020.withColumn("demand", (col("t_kWh")*60/3 ))
```

```
]
  # Convert the 'x_Timestamp' column to a timestamp type
  df2020 = df2020.withColumn('x_Timestamp', col('x_Timestamp').cast('timestamp'))

  # Extract date and time-related features
  df2020 = df2020.withColumn('Year', F.year('x_Timestamp'))
  df2020 = df2020.withColumn('Month', F.month('x_Timestamp'))
  df2020 = df2020.withColumn('Day', F.dayofmonth('x_Timestamp'))
  df2020 = df2020.withColumn('Hour', F.hour('x_Timestamp'))
  df2020 = df2020.withColumn('Minute', F.minute('x_Timestamp'))
  df2020 = df2020.withColumn('Second', F.second('x_Timestamp'))
  df2020 = df2020.withColumn('DayOfWeek', F.dayofweek('x_Timestamp'))
  # Define a window specification for ordering the data by timestamp

  df2020 = df2020.withColumn("Supply", (col("z_Avg Voltage (Volt)") * col("z_Avg Current (Amp)"))/(1000))

  df2020 = df2020.withColumn("demand", (col("t_kWh")*60/3 ))

  window_spec = Window.partitionBy("meter").orderBy("x_Timestamp")

  # Add a lag column to calculate the difference in t_kWh between consecutive timestamps
  df2020 = df2020.withColumn("t_kWh_lag", F.lag("t_kWh").over(window_spec))

  # Calculate the difference in t_kWh between consecutive timestamps
  df2020 = df2020.withColumn("t_kWh_diff", col("t_kWh") - col("t_kWh_lag"))

  # Show the DataFrame with the added date, time, and difference columns
  df2020.show()
```

x_Timestamp	t_kWh	z_Avg Voltage (Volt)	z_Avg Current (Amp)	y_Freq (Hz)	meter	Year	Month	Day	Hour	Minute	Second	DayOfWeek	Supply	demand	t_kWh_lag	t_kWh_diff
2020-01-01 00:00:00	0.006	249.54	0.76	49.98	BR04	2020	1	1	0	0	0	4	0.1896504	0.12	NULL	NULL
2020-01-01 00:03:00	0.007	249.15	0.75	49.95	BR04	2020	1	1	0	3	0	4	0.18686250000000001	0.13999999999999999	0.006	0.001
2020-01-01 00:06:00	0.006	249.54	0.73	49.95	BR04	2020	1	1	0	6	0	4	0.18150719999999998	0.12	0.007	-0.001

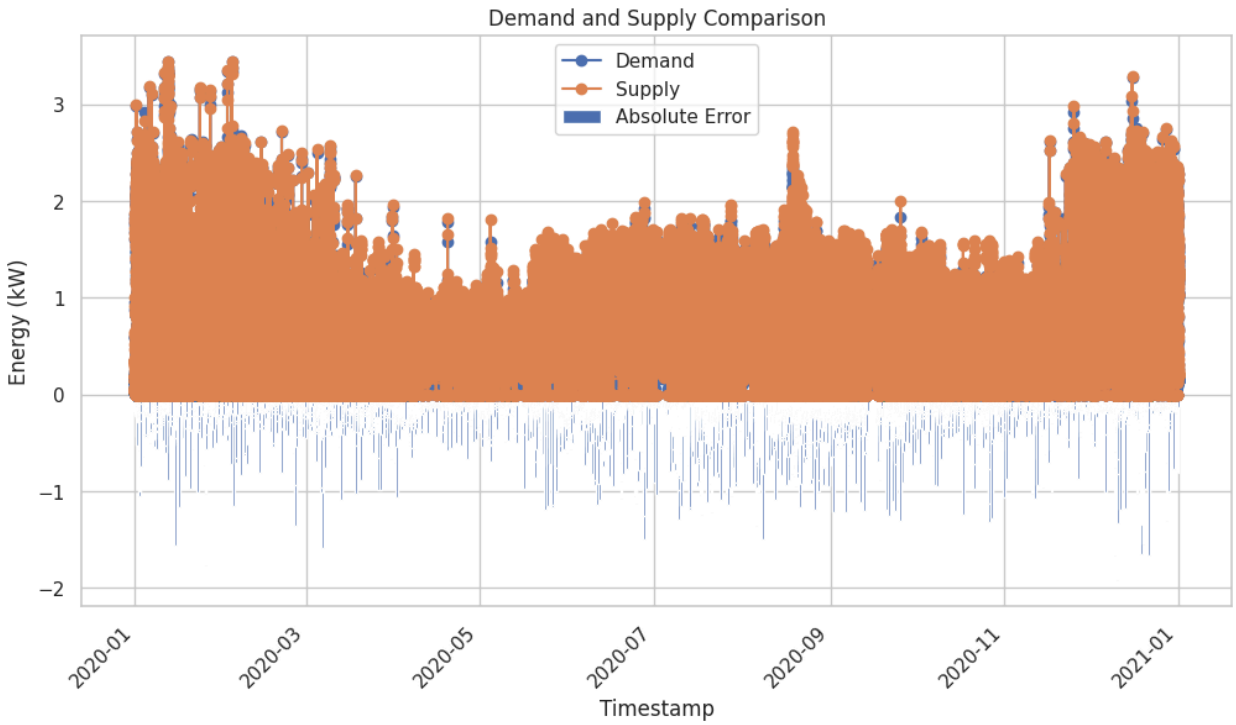


Fig 10: line plot of demand and supply vs timestamp

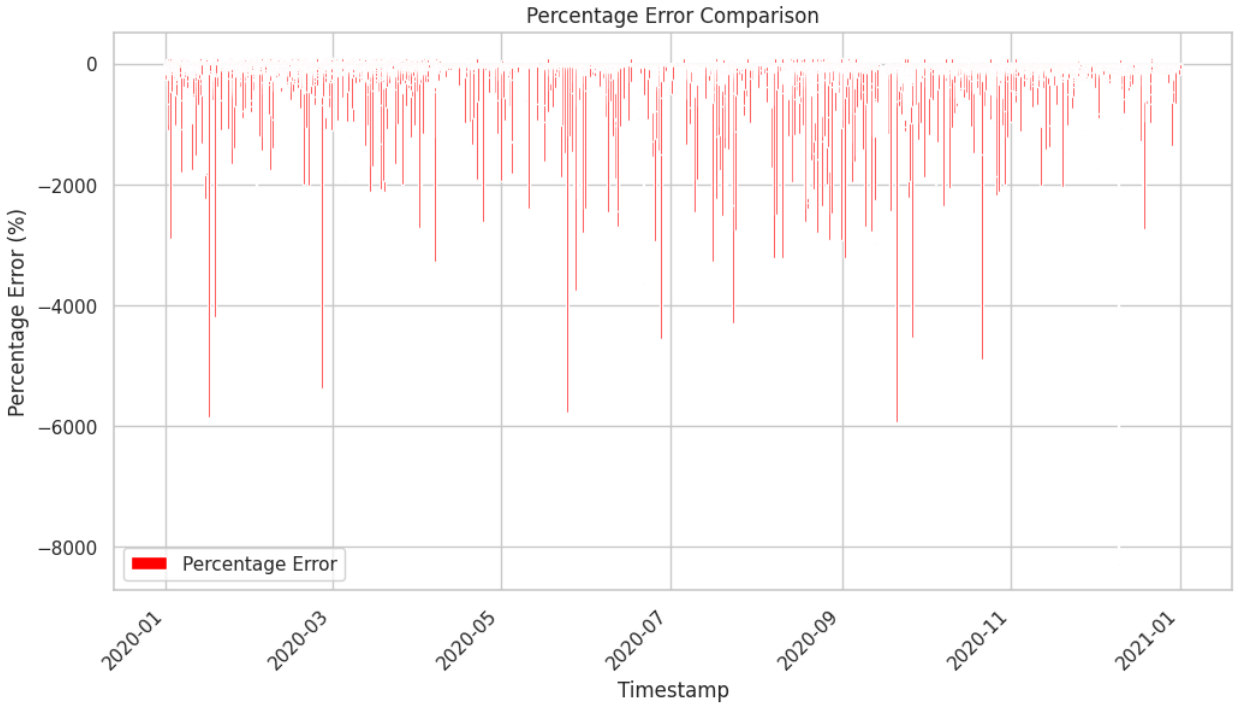


Fig 11: Percentage error plot

```
[86]
# Calculate average absolute error
average_absolute_error = df_pd['Absolute Error'].abs().mean()

# Calculate average percentage error
average_percentage_error = df_pd['Percentage Error'].abs().mean()

print(f'Average Absolute Error: {average_absolute_error:.4f} kW')
```

```
Average Absolute Error: 0.0432 kW
```

```
# Assuming 'Percentage Error' is the column name
percentage_error_column = 'Percentage Error'

# Filter out infinity values
filtered_df_pd = df_pd.replace([np.inf, -np.inf], np.nan).dropna(subset=[percentage_error_column])

# Calculate mean of absolute values
average_percentage_error = filtered_df_pd[percentage_error_column].abs().mean()

# Print the result
print(f"Average Percentage Error: {average_percentage_error}")
```

```
Average Percentage Error: 28.752038914066354
```

Fig 12: Screenshot of code for absolute error calculation between supply and demans

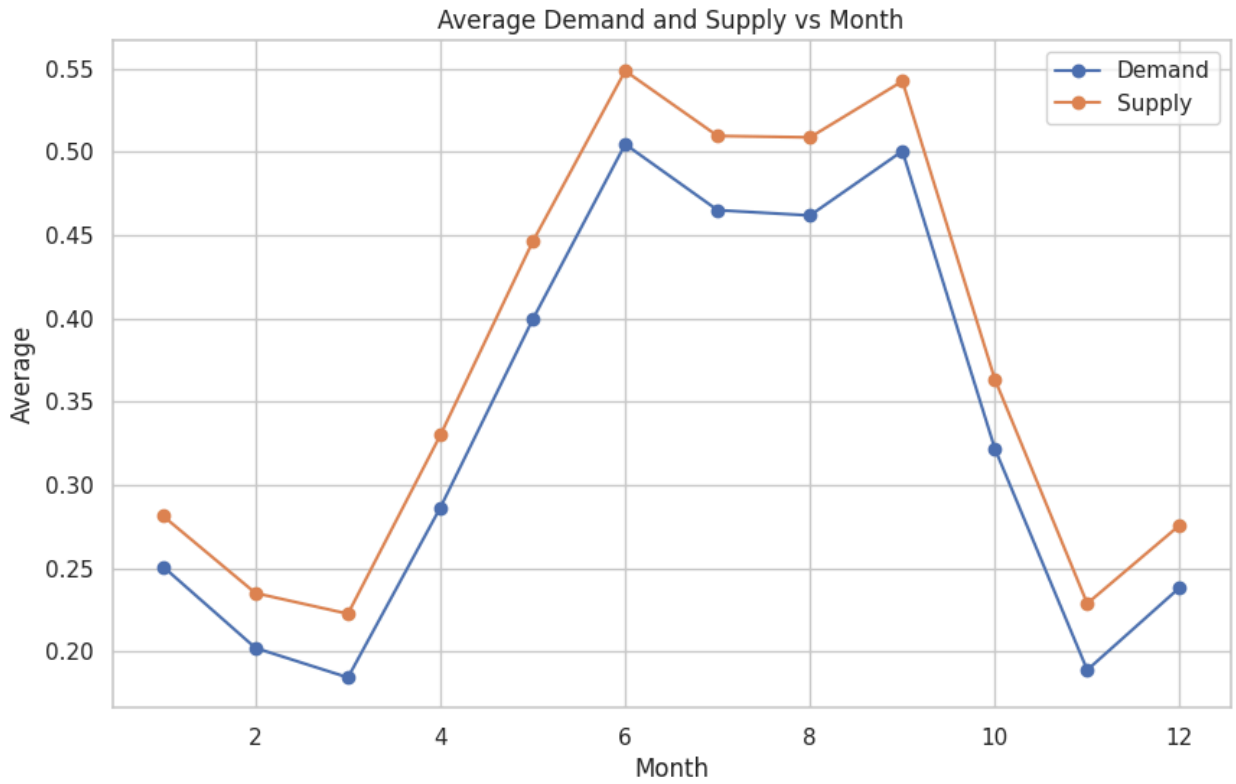


Fig 13: line plot of demand and supply vs month of the year

Mean Absolute Error: 0.0432 kW
Mean Absolute Percentage Error: 28.752038914066354

1. Feature Selection:

We calculated the correlation matrix of the features to find which features have a stronger influence on the power consumption whether it be positive or negative. Then we plotted a bar graph to display the Strength of relationship between the target electricity consumption (response final observed), and the features(demand predicted using these).

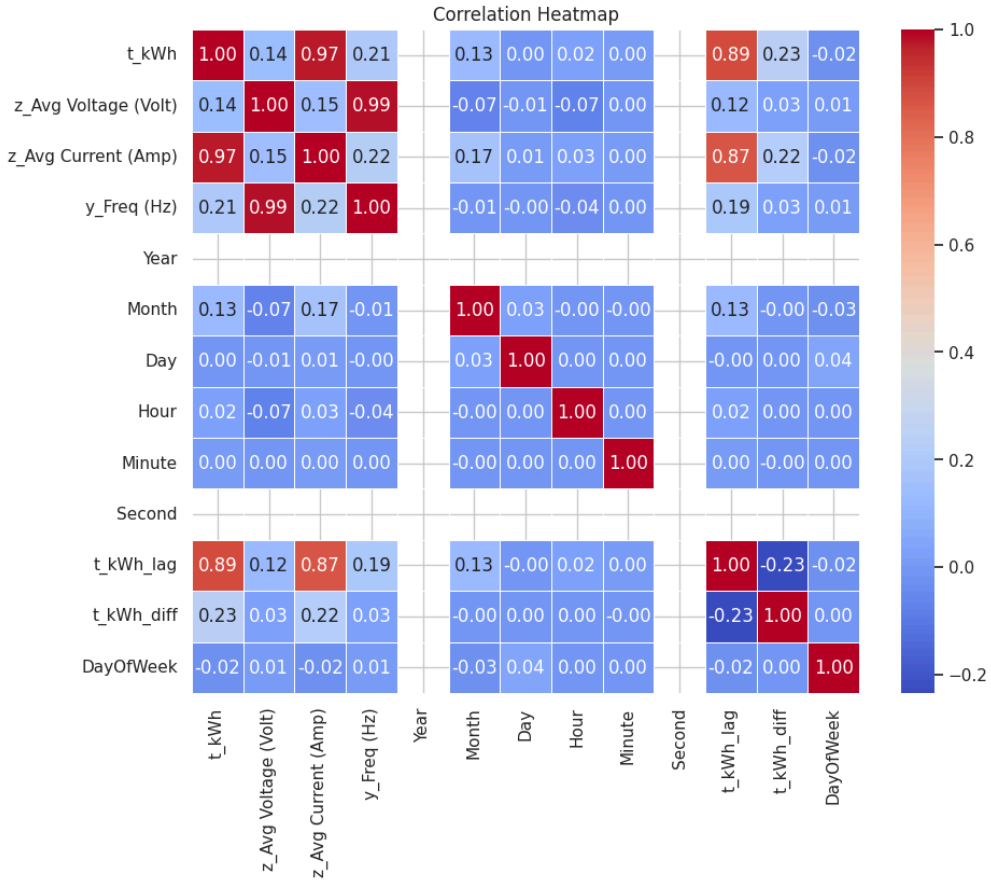


Fig 14: correlation matrix(heatmap)

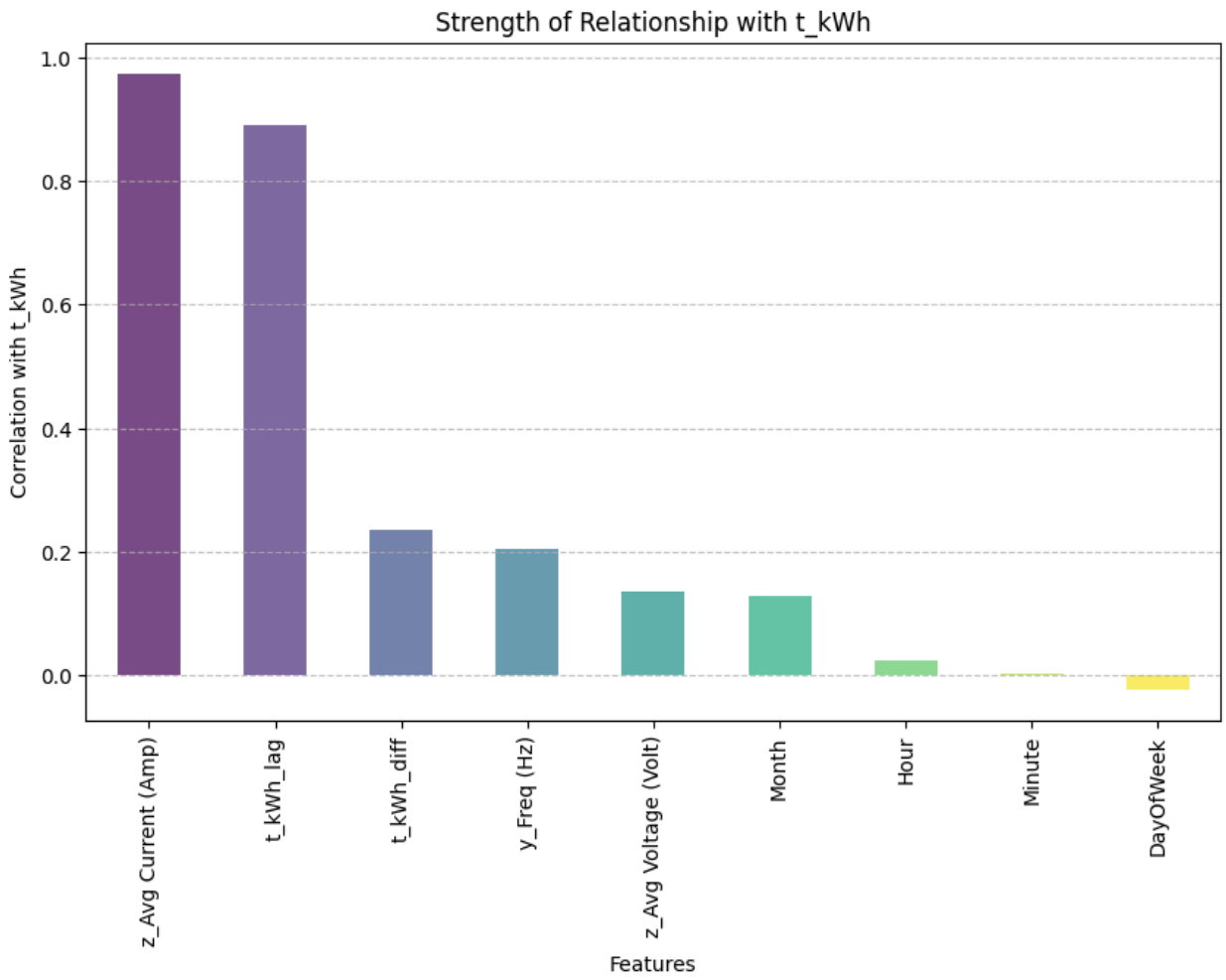


Fig 15: Strength of relationship of features with target variable

Data Scaling/Normalization:

Due to the unequal weights, some are in single digits and some are up to 4 digits. We must quantify and normalize the data so that the model will not be biased by the weight of one feature when the number of other features is insignificant. Because these features have different effects on different targets.

StandardScaler: Scaling ensures that all features contribute equally to the model and prevents one feature from becoming dominant due to its large size.

One-Hot Encoded Categorical Features: Machine learning models often use encoded input. Categorical features have separate categories and need to be converted to numbers. It represents each group as a binary vector and prevents the model from assuming any correlation between groups.

. **Pipeline**

Management Features: This is necessary because most machine learning algorithms in Spark require input to be entered into a linear vector.
Pipeline: Create pipelines to work and follow the construction steps. The pipeline ensures that changes are applied to training and testing data. They are particularly useful for maintaining clean and reworkable operations.

Training-testing separation
6627360 data samples split in 8:2 ratio

Model development (training, testing and cross-validation)
We submitted the model with 5 Random forest regressor with double cross-validation , decision tree regressor and GBT regressor and maxDepth, numTrees etc. Perform hyperparameter modification by defining grids like:

```
# Create pipelines for each model
rf_pipeline = Pipeline(stages=[rf_regressor])
dt_pipeline = Pipeline(stages=[dt_regressor])

# Set up the parameter grid for cross-validation
param_grid_rf = ParamGridBuilder().addGrid(rf_regressor.maxDepth, [5, 10]).addGrid(rf_regressor.numTrees, [10, 20]).build()
param_grid_dt = ParamGridBuilder().addGrid(dt_regressor.maxDepth, [5, 10]).build()

# Cross-Validation
```

The model overfits when it learns certain values, but has inconsistent properties of the training data, making it sensitive to small changes and breaking expansion for invisible data. Inconsistency occurs when a model fails to capture the interaction between traits and response variables. The bias-variance trade-off is important to find a balance between model simplicity (bias) and complexity (variance). This balance optimizes the model's ability to generalize to new data.

Cross validation (CV) is important to evaluate the performance of the model. The k-fold CV method divides the data into k parts, using a part for testing and the rest for training in each iteration.
Leave-one-out CV (k equals sample data) is another variable but can be calculated for large data sets.

Fig 16: Cross validation explanation Source:"Cross Validation" nghĩa là gì: Định Nghĩa, Ví Dụ trong Tiếng Anh (studytienganh.vn)

b. Time Series Approach over daily Bareilly aggregated dataset

Data exploration to find and observe seasonality, trend in data:

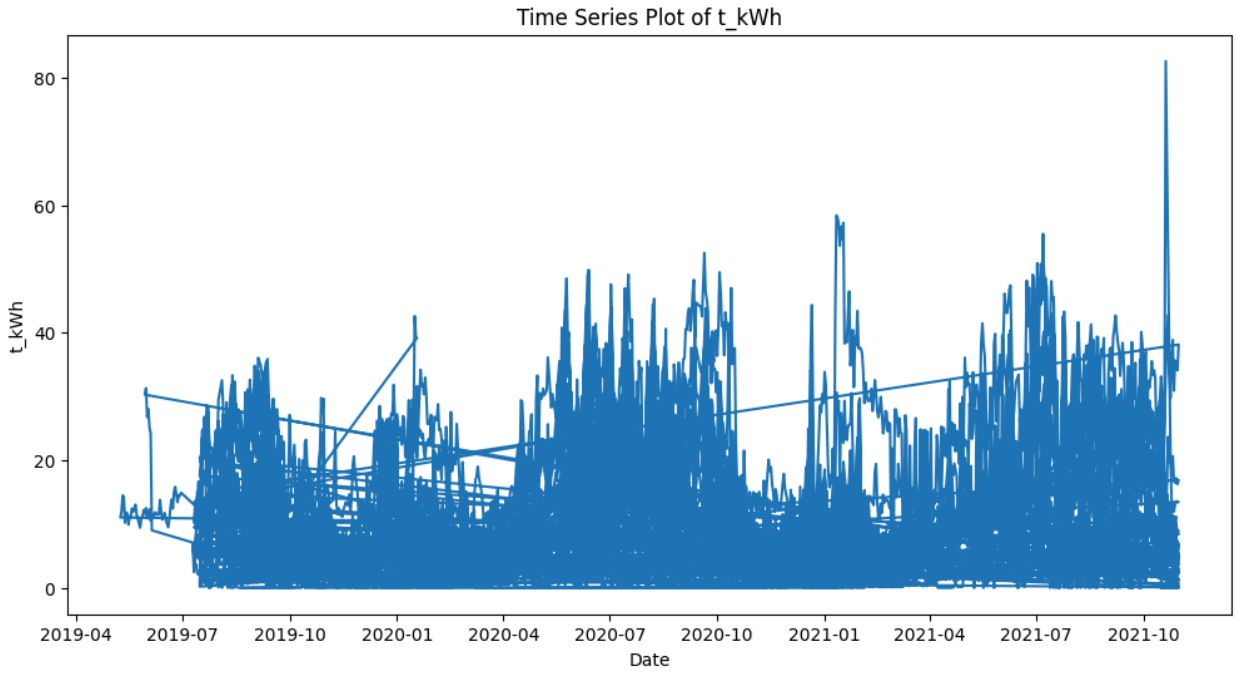


Fig 17: Time Series Plot of t_kWh. There can be observed a slight increasing trend in the series.(differencing might help).

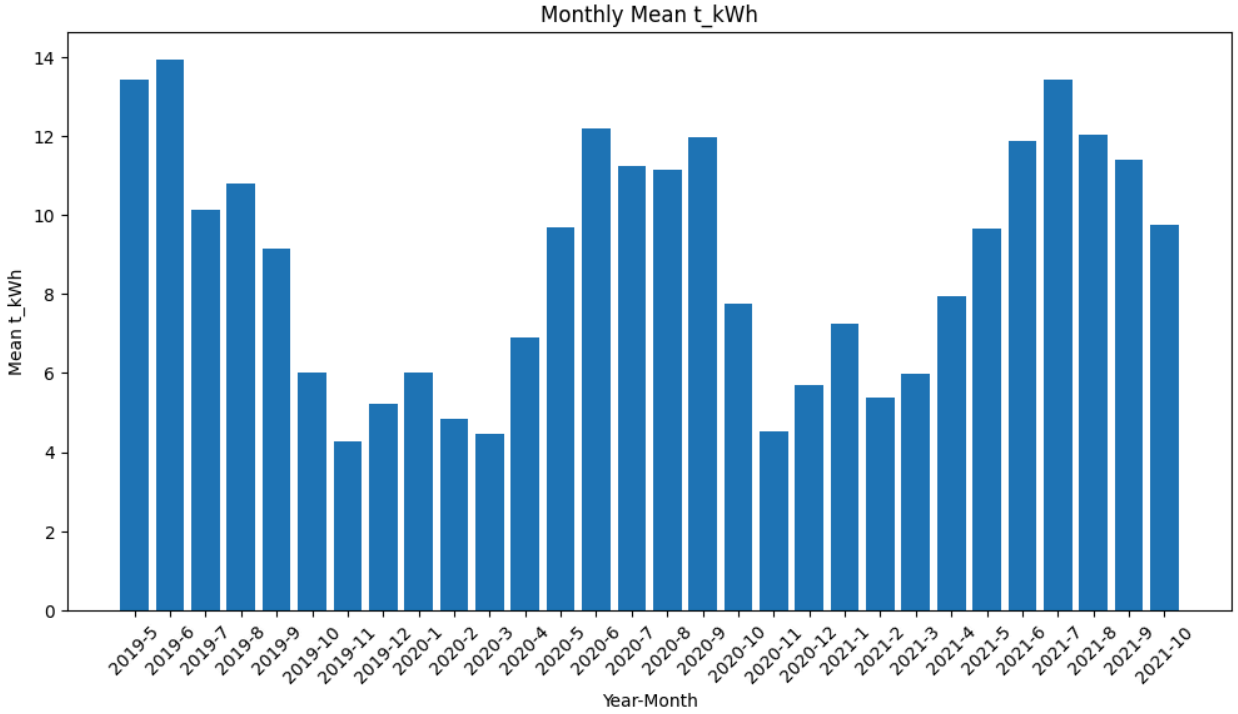


Fig 18: Monthly mean power consumptioon

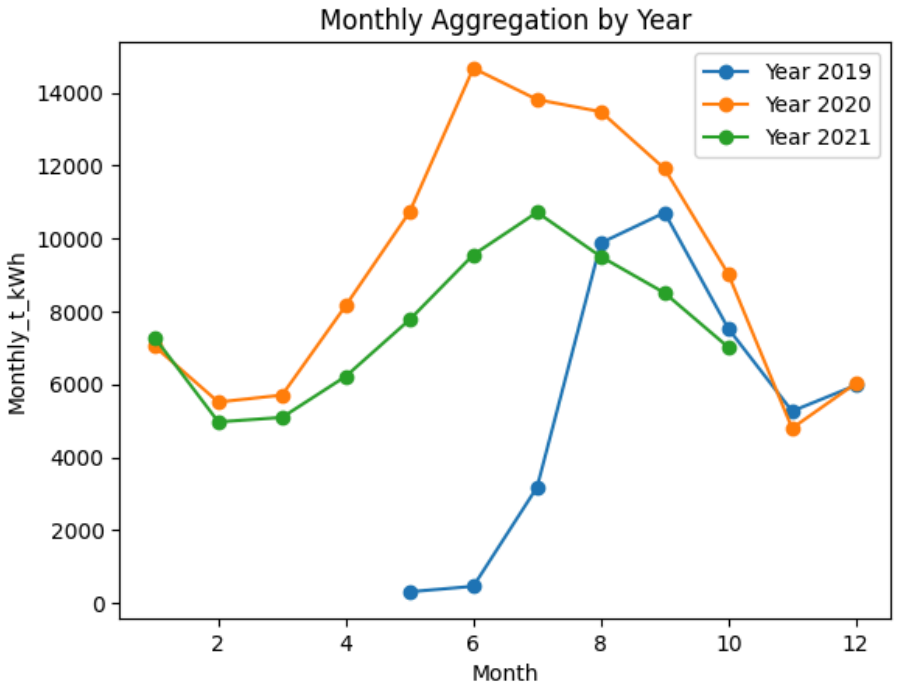


Fig 19: Monthly aggregated power consumptioon of each year vs month

Visible seasonality is observed when the energy consumption goes up during summer & humid months.

Time Series Analysis using ACF & PACF

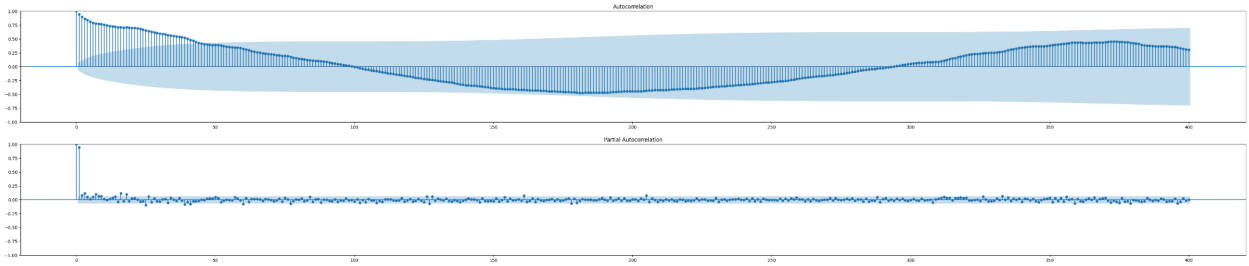


Fig 20:ACF &PACF plot of time series

The ACF is oscillating decay/dampened sine wave & doesn't shut down itself at some point.
The PACF is oscillating decay/dampened sine wave after lag 1 & doesn't shut down itself at some point.
It seems a ARMA(p, 1)

Stationarity:
We performed Adfuller test.

```
print("Critical Values:")
for key, value in critical_values.items():
    print(f"  {key}: {value}")

# Check for stationarity based on the ADF test
if p_value <= 0.05:
    print("The time series is likely stationary.")
else:
    print("The time series is likely non-stationary.")
```

```
ADF Statistic: -2.0388881548564126
P-Value: 0.26978647807030176
Critical Values:
  1%: -3.437914898140353
  5%: -2.864879373440662
  10%: -2.5685481313814624
The time series is likely non-stationary.
```

```
t_kWh_series.head()
```

```
0    11.044
```

Differencing:

Performed test on differenced data:

```
print("Differenced Series ADF Test Results:")
print(f"ADF Statistic: {adf_statistic}")
print(f"P-Value: {p_value}")
print("Critical Values:")
for key, value in critical_values.items():
    print(f"  {key}: {value}")

# Check for stationarity based on the ADF test
if p_value <= 0.05:
    print("The differenced series is stationary.")

else:
    print("The differenced series is likely non-stationary.")
```

```
Differenced Series ADF Test Results:
ADF Statistic: -10.317006704967888
P-Value: 3.0785148508656816e-18
Critical Values:
  1%: -3.437914898140353
  5%: -2.864879373440662
  10%: -2.5685481313814624
The differenced series is stationary.
```

The LagGather, MovingAverageSmoothing, and TrendGather classes apply the transformation, creating functions that can be integrated into Spark machine learning algorithms. Our method includes easy-to-choose regression types such as linear regression, tree regression, random forest regression, and gradient boosted regression tree, as well as different functions of time lags as predictors. Evaluation is done using the root mean square error (RMSE) metric.

We also added a validation variable to prevent overfitting and underfitting, thereby measuring bias and variance. Hyperparameter tuning is done by implementing a search grid of elasticNetParams such as numTrees, maxIter, MaxBins, maxDepth, subsamplingRate for tree regressors and adjusting the hyperparameters for linear regression to get the best answer.

Delay transformer autoregressive features (AR), moving average smoothing (MA) and variance (I) were combined to create an ARIMA model run on the Spark dataframe.

Batche processing:
Data is broken into smaller pieces, and each piece represents a portion of the data over time. In each cluster, a window of consecutive data is used for training or prediction.
We "slide" the window sequentially across the data set and then use the ARIMA and SARIMAX models. This will help speed up calculations. The batch size we chose is 100. Do ARIMA, SARIMAX. Bareilly2020 data is divided into 10 categories.

H. Results

Using the ML model to predict features (usually three minutes short):

The prediction performance of the regression model is the error measured against metrics such as the coefficient of determination (R^2).), root mean square error (RMSE), mean error (MAE), and mean percent error (MAP).

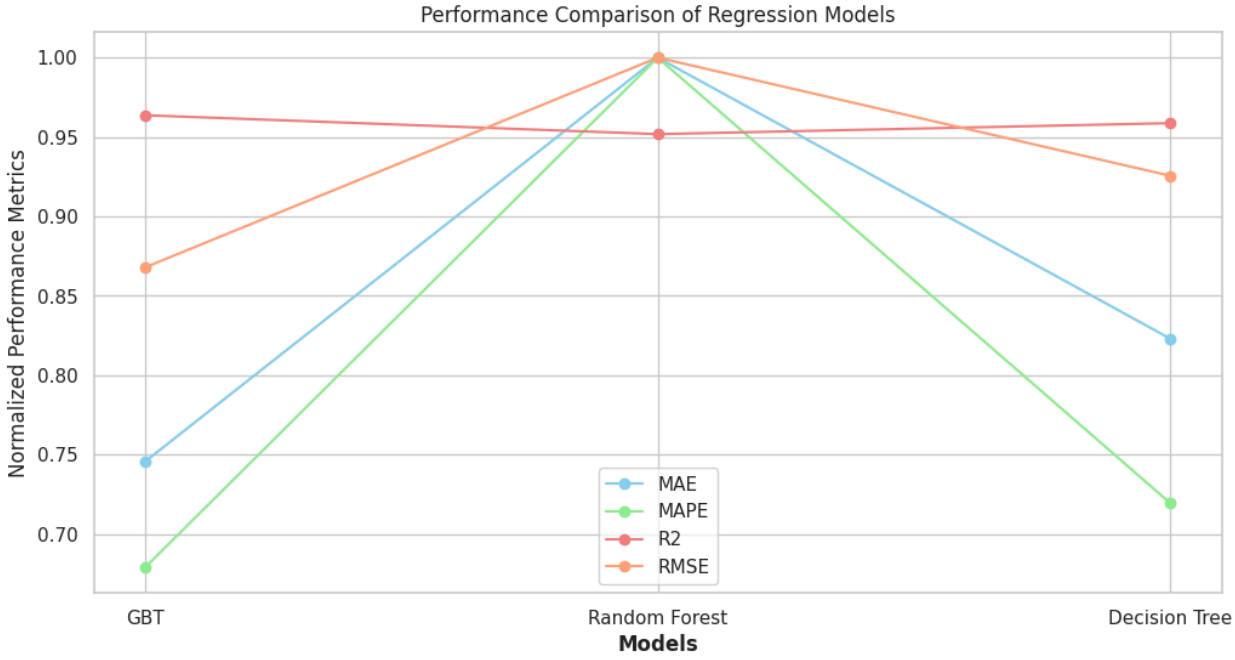
$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \in [0,+\infty) \qquad MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \in [0,+\infty)$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \in [0,+\infty)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \in [0,1] \qquad RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \in [0,+\infty)$$

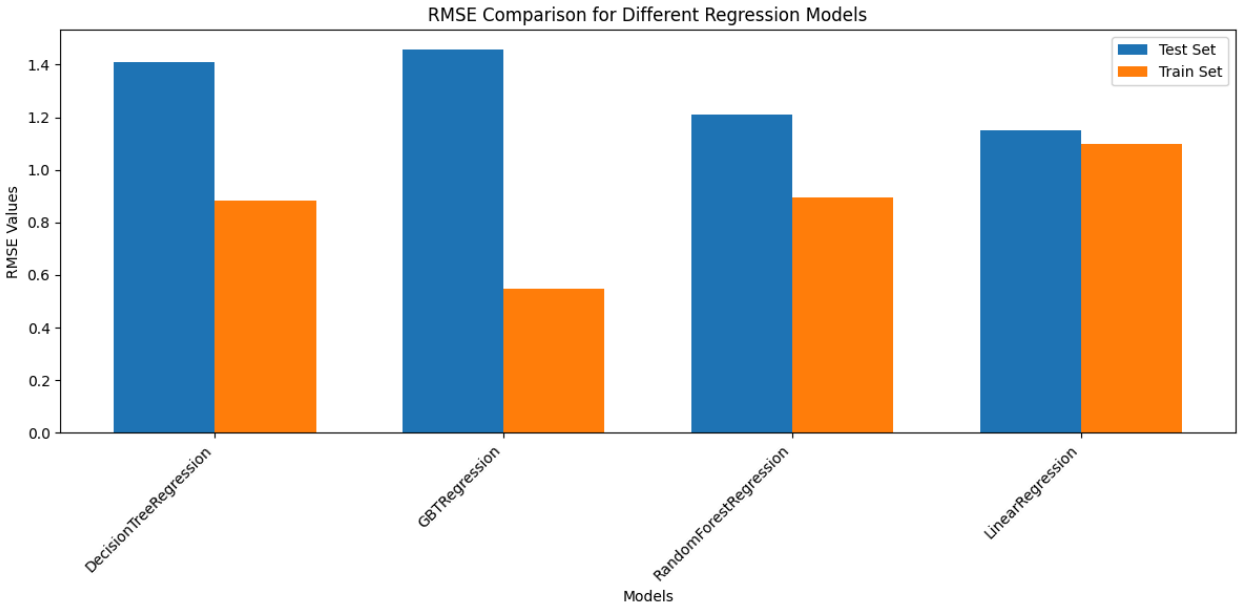
Fig Performance Metrics


Performance Comparison of Regression Models
MAE
MAPE
R2
RMSE
1.0
0.8
0.6
0.4
0.2
0.0
GBT
Random Forest
Decision Tree
Models

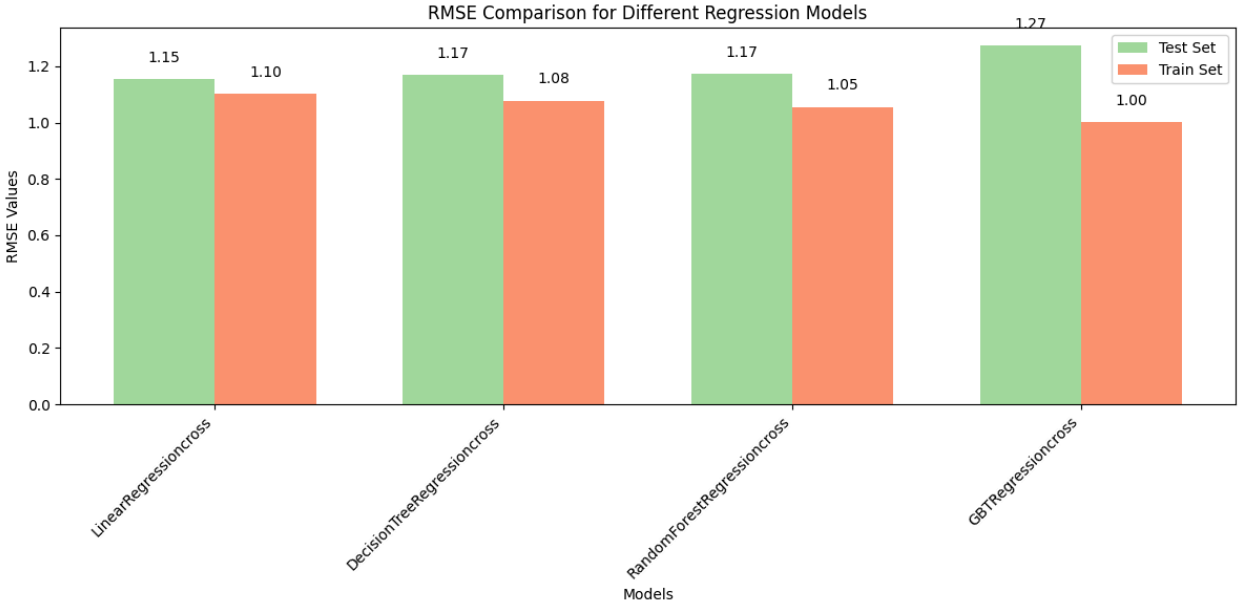

Performance Comparison of Regression Models
Normalized Performance Metrics
1.00
0.95
0.90
0.85
0.80
0.75
0.70
MAE
MAPE
R2
RMSE
GBT
Random Forest
Decision Tree
Models

Models	R^2	RMSE	MAE	MAPE
Decision tree	0.95873614	0.0028753424	0.00142654	16.70203181
Random Forest	0.95183497	0.0031064943	0.001733464	23.21827331
GBT	0.96372095	0.0026960789	0.001292445	15.7621761

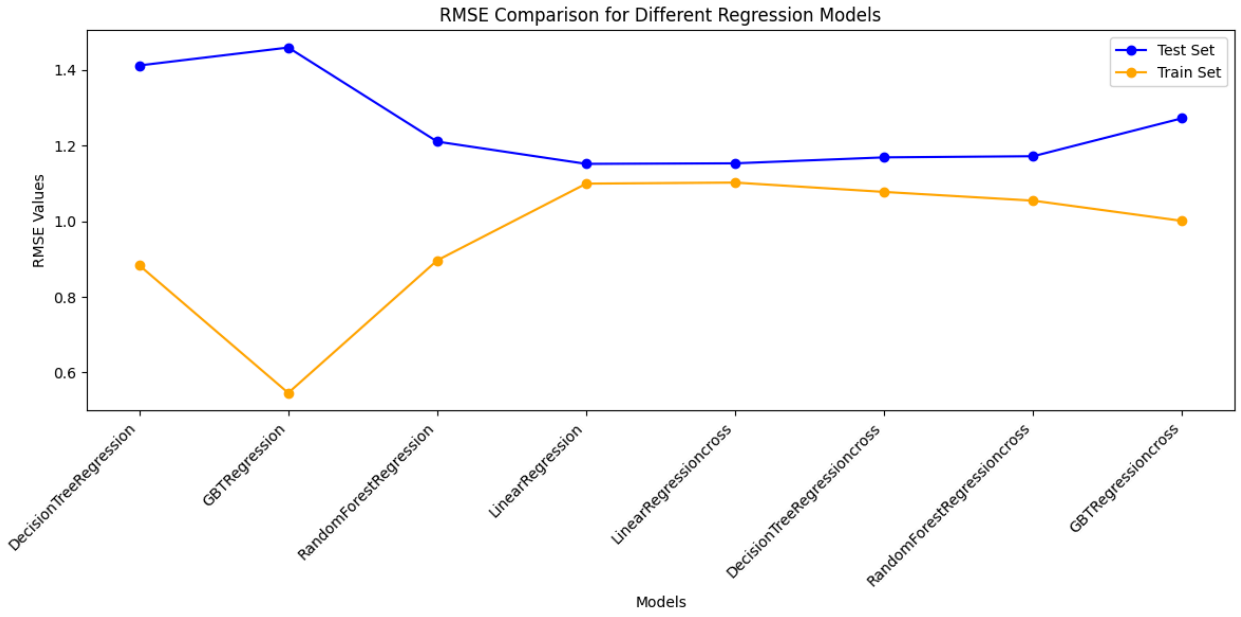
a. Time Series Analysis of (Daily data):



Time Series Analysis with cross validation



Combined Comparison line plot:



Models(Treating as Time-Series)	Train RMSE	Test RMSE
DecisionTreeRegression(DTR)	0.8835	1.4117
GBTRegression (GBTR)	0.5461	1.4591
Random Forest Regression(RFR)	0.8966	1.2104

Linear Regression(LR)	1.0994	1.1516
Cross-Validated DTR	1.0774	1.1688
Cross-Validated GBTR	1.0009	1.2723
Cross-Validated RFR	1.0544	1.1718
Cross-Validated LR	1.1021	1.1530

b. ARIMA

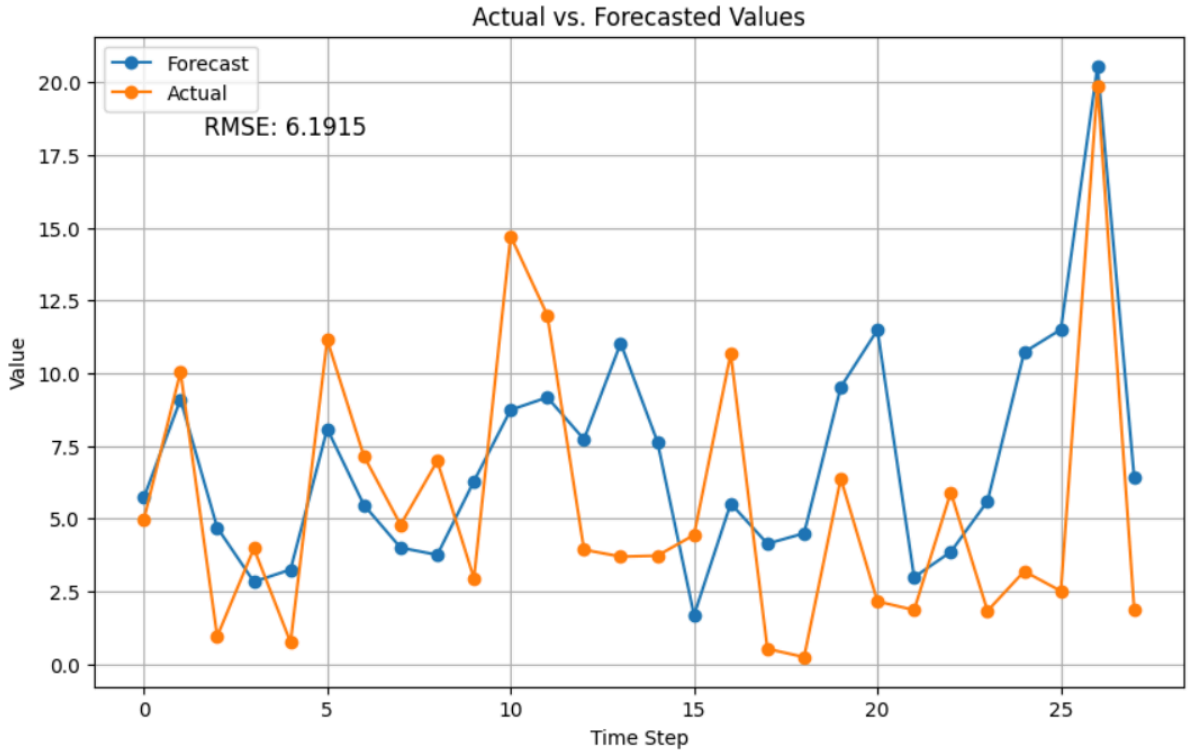


Fig 21(window size = 1000)

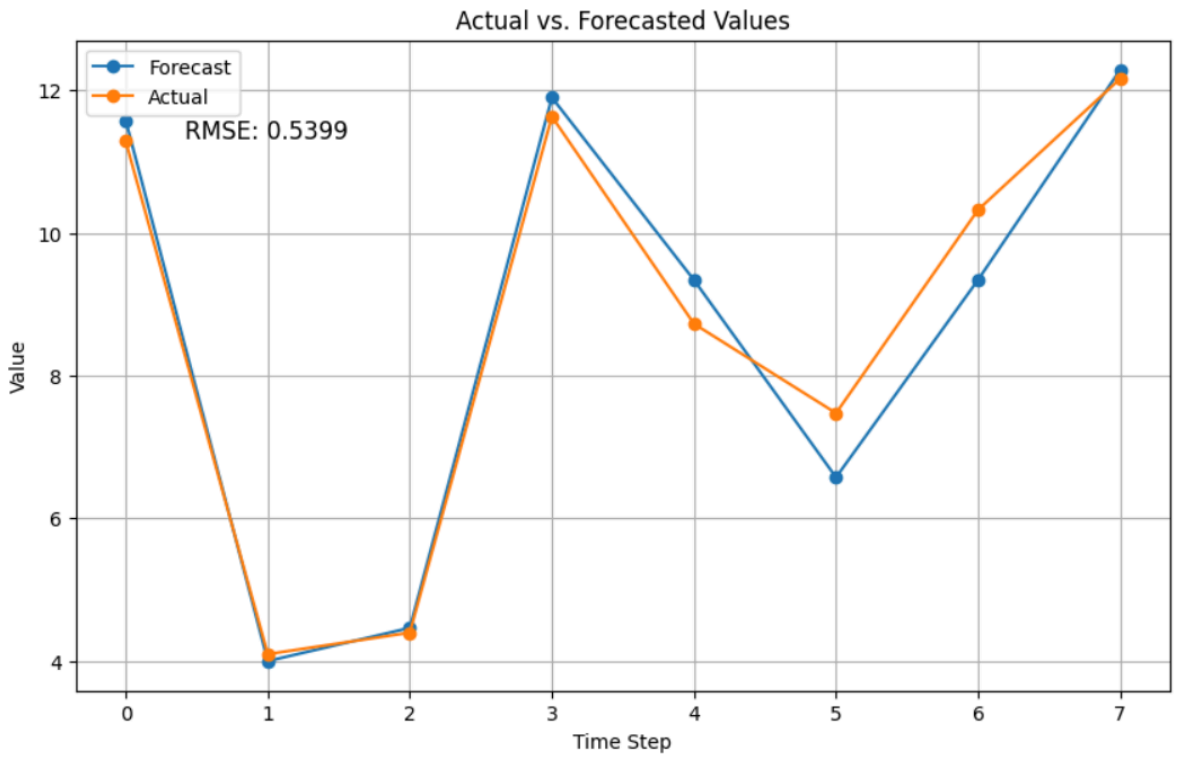
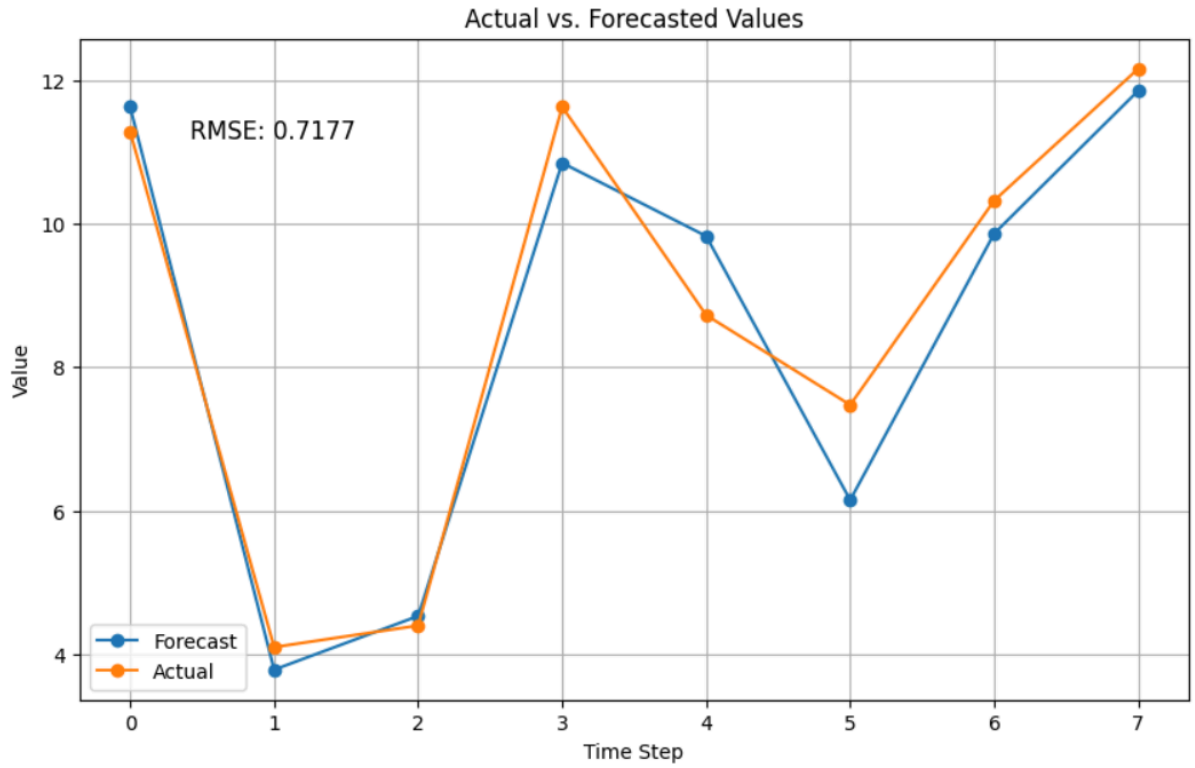


Fig : Actual vs Predicted of ARIMA
(window size =100)

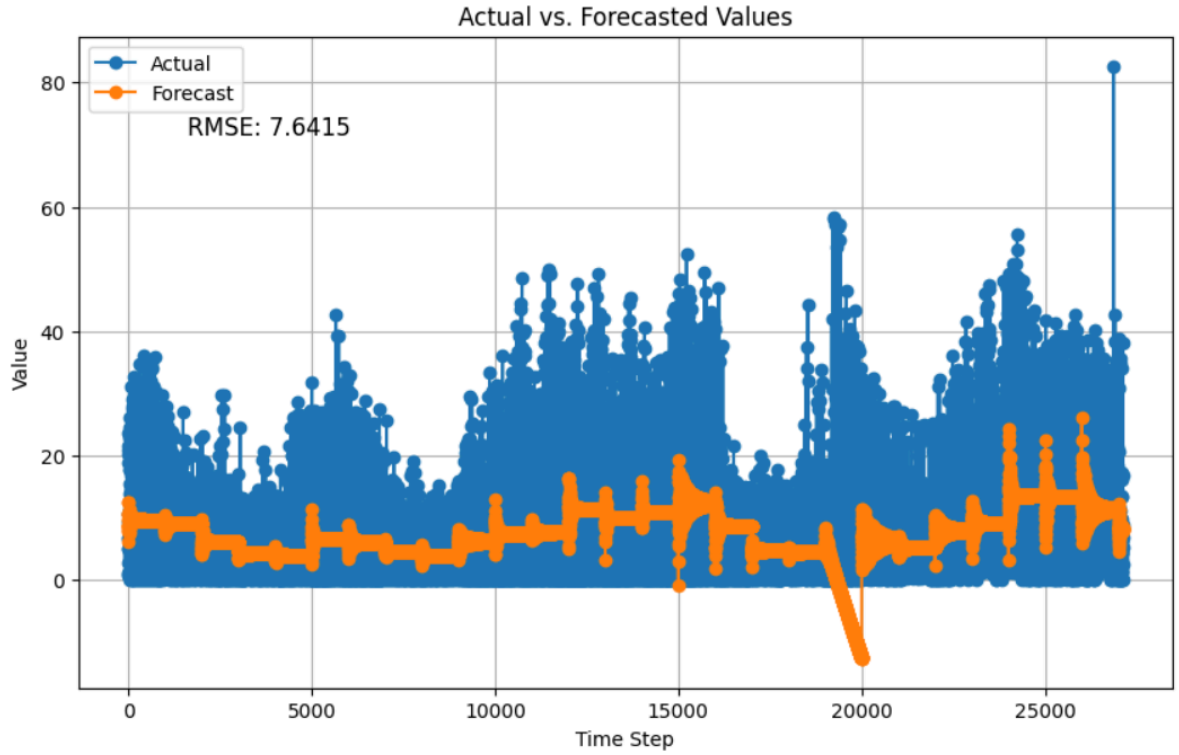
For ARIMA, the RMSE values were as follows:

Window Size 100: RMSE = 0.5399
Window Size 1000: RMSE = 6.1915

c. SARIMAX



Actual vs Predicted of SARIMAX(Window size: 100)



Actual vs Forecast values of SARIMAX(Window size:1000)

For SARIMAX, the RMSE values were:

Window Size 100: RMSE = 0.7177
Window Size 1000: RMSE = 7.6415

Models(Treating as Time-Series)	RMSE
ARIMA	0.5399
SARIMAX	0.7177

A lower RMSE suggests that the ARIMA model's predictions are, on average, closer to the actual values compared to the SARIMAX model.
Regardless of the window size, the ARIMA model demonstrated superior predictive accuracy compared to the more complex SARIMAX model.
Influence of window size on model performance, with smaller windows allowing for more adaptive and responsive predictions, while larger windows capture longer-term trends and seasonality.

Comparison of Prosposed model with LIterature survey

Literature Review Paper result:

1. A Big Data Approach for Demand Response Management in Smart Grid Using the Prophet Model by Sanju Kumari,Neeraj Kumar and Prashant Singh Rana Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala. Published: 12 July 2022, https://www.mdpi.com/2079-9292/11/14/2179

Model	MSE	RMSE	MAE	MAPE
ARIMA	1.06877	1.03381	0.6239	1.1932
Prophet	0.67546	0.82186	0.5308	1.0399

Paper/Model	Author	RMSE
2. LR(model) Machine learning based energy demand prediction, Energy Reports,Volume 9, Supplement 12, 2023, Pages 171-176, ISSN 2352-4847, https://doi.org/10.1016/j.egyr.2023.09.151. (paper)	Ammar Kamoona, Hui Song, Kian Keshavarzian, Kedem Levy, Mahdi Jalili, Richardt Wilkinson, Xinghuo Yu, Brendan McGrath, Lasantha Meegahapola,	3.67

Paper/Model	Author	RMSE
3. NLR(model) Machine learning based energy demand prediction, Energy Reports,Volume 9, Supplement 12, 2023, Pages 171-176, ISSN 2352-4847, https://doi.org/10.1016/j.egyr.2023.09.151. (paper)	Ammar Kamoona, Hui Song, Kian Keshavarzian, Kedem Levy, Mahdi Jalili, Richardt Wilkinson, Xinghuo Yu, Brendan McGrath, Lasantha Meegahapola,	3.07
4. ARIMA(time series analysis best performing)	Proposed Model	0.5399
5. Linear Regression (time series ie. regressor over ARIMA best performing model)	Proposed Model	1.15
6. GBT (features prediction best performing model)	Proposed Model	0.00269

LR 3.67 3.67 NLR 3.07 2.87

Final Result

In summary, GBT performed best during training but outperformed during RF testing without validation. Although GBT's trains are not inferior to RF's. So we can conclude that RF is the best.
By knowing, we see that there is no need for complex models such as GBT and RF because horizontal lines can provide the best performance.

In general, the first method gives better results; The RMSE for GBT matching is only about 0.0027 and the MAPE of the equipment is 28%, while GBT narrows the MAPE by only 15%. Well, we cannot say that the second model does not work very well, as the first method has additional features that leak a lot of important information such as voltage, average current. Because in real life we only have time records. The initial data model uses intelligent data connectivity to improve the practical needs of major energy users, including suitable products, electrical backup and storage systems of fire trucks. We can carefully monitor supply and time to predict demand. However, due to the limited data available for our specific research, we focus on predicting future demand ((customers)) which can be used to organize resources/production today so that we can meet supply demand. Since temperature and humidity also play an important role in this process, temperature, humidity, holidays, festivals, etc. is added together. Multiple time comparison test samples can be used for subsequent analysis.

Acknowledgments
Our progress and this report would not be possible without the support, guidance and collaboration of many individuals and partners. Our professors Dr. Sonali Agarwal and her mentors Mr. Shashi Shekhar

Kumar and Mr. We would like to thank Ritesh Chandra for his continuous support and expert guidance throughout the project along with all references to Big Data Analytics.

We are also grateful to the Indian Institute of Information Technology, Allahabad for providing the necessary resources and facilities. We thank the collective efforts of everyone who contributed directly or indirectly to this research as they played a significant role in our success.

I. References

7. Yang Bai, Haiwang Zhong and Qing Xia, "Real-time demand response potential evaluation: A smart meter driven method," 2016 IEEE Power and Energy Society General Meeting (PESGM), Boston, MA, 2016, pp. 1-5, doi: 10.1109/PESGM.2016.7742002.
8. Y. Chen, H. Hou, Z. Zhang, X. Li, J. Wang and A. Tang, "Review on Smart Meter Data Clustering and Demand Response Analytics," 2020 12th IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC), Nanjing, China, 2020, pp. 1-6, doi: 10.1109/APPEEC48164.2020.9220376.
9. Ning Qi, Lin Cheng, Helin Xu, Kuihua Wu, XuLiang Li, Yanshuo Wang, Rui Liu, Smart meter data-driven evaluation of operational demand response potential of residential air conditioning loads,Applied Energy, Volume 279, 2020,115708,ISSN 0306-2619,https://doi.org/10.1016/j.apenergy.2020.115708.
10. A Big Data Approach for Demand Response Management in Smart Grid Using the Prophet Model by Sanju Kumari,Neeraj Kumar and Prashant Singh Rana Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala. Published: 12 July 2022, https://www.mdpi.com/2079-9292/11/14/2179
11. A Big Data platform for smart meter data analytics , Author links open overlay panelTom Wilcox a, Nanlin Jin b,Available online 22 January 2019,https://www.sciencedirect.com/science/article/pii/S0166361518303749
12. Smart meter data-driven evaluation of operational demand response potential of residential air conditioning loads, Author links open overlay panel Ning Qi , Available online 8 October 2020.https://www.sciencedirect.com/science/article/abs/pii/S0306261920312022

Supplementary Material/Code:

1. C1_Bareily2020_python preprocessing.ipynb - Colaboratory (google.com)

2. SPARK_ENERGY_C1.ipynb - Colaboratory (google.com)

3. CheckNullandARIMAwithBatchProcessing.ipynb - Colaboratory (google.com)
4. CheckNullandARIMASARIMAXwithBatchProcessing.ipynb - Colaboratory (google.com)
5. BDA_PROJECT_C3_Time_series_smart - Colaboratory (google.com)
6. ML_models_on_Bareilly2020.ipynb - Colaboratory (google.com) | Batch1_ML_models_on_Bareilly2020.ipynb - Colaboratory (google.com) |
7. TSDA_BDA.ipynb - Colaboratory (google.com)
8. BDA_C3_TIME_SERIES_with_cross validation - Colaboratory (google.com)

Contribution:

It was a combined effort of all team members when it came to brain storming for ideas, we shared results, and from the feedback of each other tweaked the code, and proceeded further. Though we have divided the contributions as below, it is inevitable that all were part of it as whole, and efforts of everyone overlap, since everyone assisted each other and progressed collectively. Below distribution was based on solely division of writing work of already done work in the report.

Member	REPORT	CODE
Priyamvada(IIB2020037)	GHI	1. ML_models_on_Bareilly2020 2. BDA_PROJECT_C3_Time_series_smart 3. BDA_C3_TIME_SERIES_with_cross validation 4. TSDA_BDA 5. SPARK_ENERGY_C1 6. CheckNullandARIMAwithBatchProcessing 7. C1_Bareily2020_python preprocessing 8. CheckNullandARIMASARIMAXwithBatchProcessing
Nikita(IIB2020042)	EF	1. BDA_PROJECT_C3_Time_series_smart 2. CheckNullandARIMASARIMAXwithBatchProcessing 3. CheckNullandARIMAwithBatchProcessing 4. C1_Bareily2020_python preprocessing 5. SPARK_ENERGY_C1
Niharika(IIB2020011)	CD	1. BDA_PROJECT_C3_Time_series_smart 2. CheckNullandARIMAwithBatchProcessing

		3. CheckNullandARIMASARIMAXwithBatchProcessing 4. C1_Bareily2020_python preprocessing 5. SPARK_ENERGY_C1
Kalyani(IIT2020196)	AB	1. BDA_PROJECT_C3_Time_series_smart 2. C1_Bareily2020_python preprocessing 3. CheckNullandARIMAwithBatchProcessing 4. SPARK_ENERGY_C1