FLUENTGRID
seamless digital transformation

Fluentgrid Internship Task Report

Report by:
J.Ritesh

June 13, 2023

Table of Contents

1 OVERVIEW OF THE COMPANY

1.1 Introduction of the Organization

Fluentgrid Limited is an Indian software product company, which was founded in 1998. It provides digital transformation services for power, water, and gas distribution utilities and smart cities and communities. The company received the 2012 IBM Beacon Award and was also the winner of the Fierce Innovation Award 2015.

1.2 Vision and Mission of the Organization

Fluentgrid's vision is to be the leading provider of smart energy and infrastructure solutions globally.Its mission is to facilitate the global movement towards smart grids and smarter cities for a sustainable planet and happy people. Its values are innovation, customer focus, integrity, and teamwork.

1.3 Future Plans of the Organization

-Fluentgrid's future plans include: Expanding its product offerings: Fluentgrid plans to launch new products in the areas of smart metering, smart grid analytics, and smart city management.
-Entering new markets: Fluentgrid plans to enter new markets in Europe, Asia, and Latin America.
-Continuing to invest in research and development: Fluentgrid plans to invest 100 million dollars in research and development over the next five years.

Figure 1

2 INTRODUCTION

Power consumption analysis involves assessing and quantifying the amount of electrical power used by systems and devices.It plays a crucial role in various fields, including electronics, energy management, and sustainability. By measuring voltage, current, and energy usage, engineers can identify areas of inefficiency, energy-intensive components or processes, and propose strategies for improvement. Power consumption analysis enables informed decision-making in designing energy-efficient devices, optimizing power usage in industries, and reducing energy consumption in buildings. By understanding power consumption patterns and implementing energy-saving measures, individuals and organizations can contribute to a sustainable future.

2.1 Problem Statement

The objective of this report is to develop a predictive model that accurately forecasts the given April month dataset. The dataset contains various variables, and the goal is to utilize this data to generate accurate predictions for the specified time period. By addressing this problem, we aim to provide valuable insights and improve forecasting accuracy for the given April month dataset.

2.2 Description of the problem statement

- rtimeid: This variable represents the timestamp indicating the time at which the measurement was taken. It helps in tracking the energy consumption and other measurements over time.

- kWh: It stands for kilowatt-hours, which is a unit of energy. This variable represents the total amount of energy consumed at the given timestamp. It indicates the cumulative energy usage up to that point in time.

- VB, VY, VR: These variables represent voltage values measured in volts. They may refer to voltage measurements in different phases or at different points in the electrical system. For example, VB could represent voltage in the B phase, VY in the Y phase, and VR in the R phase.

- IY, IB, IN: These variables represent current values measured in amps. Similar to voltage, they may refer to current measurements in different phases or at different points in the electrical system. For example, IY could represent current in the Y phase, IB in the B phase, and IN in the neutral wire.

- kVarh: It stands for kilovolt-ampere reactive hours, which is a unit of reactive power consumption. Reactive power is the power that oscillates between the source and the load without performing useful work. kVarh represents the cumulative reactive power consumption at the given timestamp.

- IR: This variable represents the reactive current, which is the current component associated with reactive power. It is measured in amps and provides information about the reactive power flow in the system.

- kWhLead: It represents the kilowatt-hours consumed with a leading power factor. A leading power factor indicates that the current wave leads the voltage wave in phase. It is typically observed in certain types of loads or circuits.

- VY: This variable specifically represents the voltage in the Y phase. It provides information about the voltage measurement in a specific phase of the electrical system.

- IN: This variable specifically represents the current in the neutral wire. The neutral wire carries the return current in a balanced three-phase electrical system, and measuring the current in the neutral wire provides insights into the overall current balance.

- kWhLag: It represents the kilowatt-hours consumed with a lagging power factor. A lagging power factor indicates that the current wave lags behind the voltage wave in phase. Similar to kWhLead, it is observed in certain types of loads or circuits.

2.3 Useful definitions and Terminologies

Voltage: Voltage is the electrical potential difference between two points in a circuit. It represents the force or pressure that drives electric charges to move.

Current: Current refers to the flow of electric charge in a circuit. It represents the rate at which charges move through a conductor.

Resistance: Resistance is the opposition encountered by electric current when flowing through a material. It is a measure of how effectively a material restricts the flow of electrons.

Power factor: Power factor can be thought of as a measure of how effectively electrical energy is being used. It tells us how well the voltage and current in an electrical circuit are working together, with a higher power factor indicating more efficient energy usage.

Leading power factor: A leading power factor means that the current leads the voltage in a circuit, which is common in systems with capacitive loads like motors.

Lagging power factor: A lagging power factor means that the current lags behind the voltage, typically seen in systems with inductive loads such as transformers.

2.4 What is Data Analysis?

Data analysis is the process of examining, cleaning, transforming, and organizing data to uncover meaningful patterns, insights, and trends.It involves using various statistical and analytical techniques to extract valuable information from data and make data-driven decisions. Data analysis helps us understand the story that data tells us, allowing us to identify relationships between variables, detect anomalies, make predictions, and draw meaningful conclusions. It plays a crucial role in various fields, including business, science, healthcare, and social sciences, enabling us to gain valuable insights and drive informed decision-making based on objective evidence.

2.5 How Data Analysis is used in this dataset?

In this dataset, data analysis can be used to gain insights into energy consumption patterns. By analyzing the data, we can identify trends and patterns in energy usage over time, such as daily, weekly, or monthly fluctuations. We can calculate metrics like average consumption, peak consumption, and total consumption to understand overall energy usage.Data analysis can also help identify abnormal or anomalous energy consumption, indicating potential issues or inefficiencies. Additionally, correlations between different variables in the dataset, such as voltage and current, can be explored to understand the relationships between them. Overall, data analysis enables us to uncover valuable information and make data-driven decisions regarding energy consumption and efficiency.

2.6 What is Machine Learning?

Machine Learning (ML) is a subfield of artificial intelligence that focuses on developing algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. ML algorithms learn from data and improve their performance over time through experience. The goal of ML is to enable machines to automatically discover patterns, make accurate predictions, and make intelligent decisions based on input data.

2.7 How does Machine Learning work?

- A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.
- Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem.
- The below block diagram explains the working of Machine Learning algorithm:

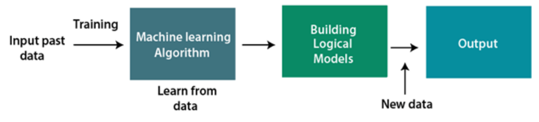


Figure 2

2.8 Need For Machine Learning

Machine learning is needed because it allows computers to process and analyze large, complex datasets that humans cannot handle manually. It automates tasks, recognizes patterns, makes predictions, and provides personalized experiences. Machine learning plays a vital role in fraud detection, cybersecurity, healthcare diagnostics, and language processing. It enhances efficiency, improves decision-making, and enables businesses to extract valuable insights from data. Overall, machine learning addresses the challenges posed by big data and empowers technology to learn, adapt, and make intelligent decisions.

2.9 Classification of Machine Learning

There some variations of how to define the types of Machine Learning Algorithms but commonly they can be divided into categories according to their purpose and the main categories are the following:

- Supervised learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

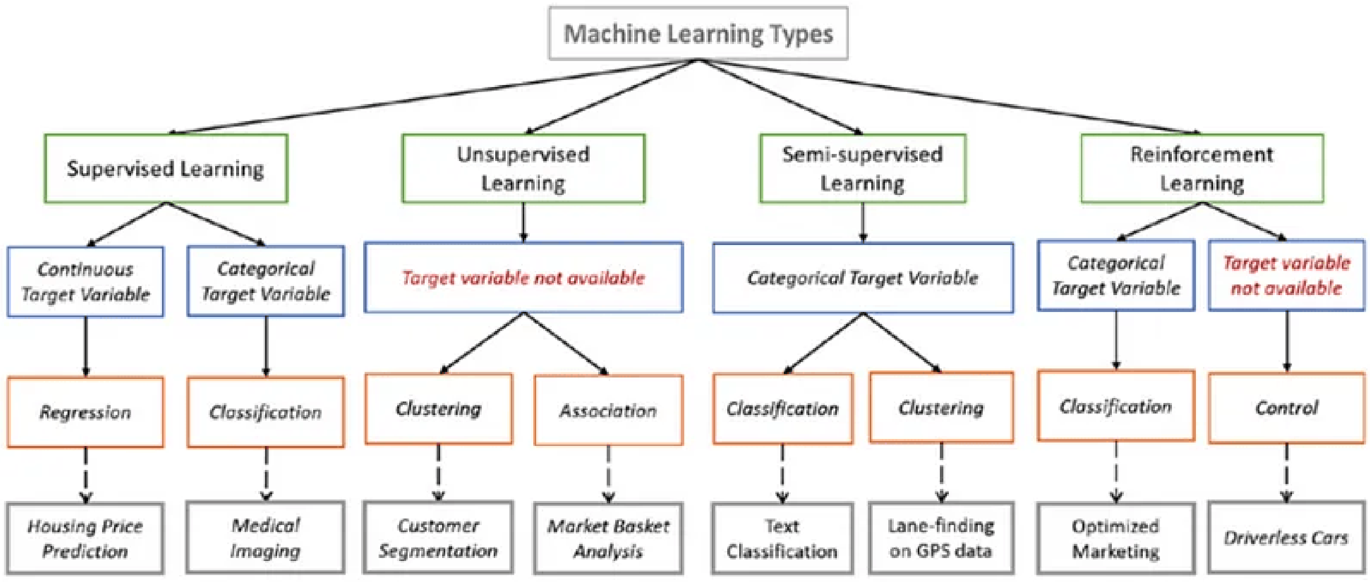


Figure 3

2.9.1 Supervised Learning

- I like to think of supervised learning with the concept of function approximation, where basically we train an algorithm and in the end of the process we pick the function that best describes the input data, the one that for a given X makes the best estimation of y (X -> y). Most of the time we are not able to figure out the true function that always make the correct predictions and other reason is that the algorithm rely upon an assumption made by humans about how the computer should learn and this assumptions introduce a bias, Bias is topic I'll explain in another post.
- Here the human experts acts as the teacher where we feed the computer with training data containing the input/predictors and we show it the correct answers (output) and from the data the computer should be able to learn the patterns.
- Supervised learning algorithms try to model relationships and dependencies between the target prediction output and the input features such that we can predict the output values for new data based on those relationships which it learned from the previous data sets.

List of Common Algorithms in Supervised Learning

- Nearest Neighbor
- Naive Bayes
- Decision Trees
- Linear Regression
- Support Vector Machines (SVM)
- Neural Networks

2.9.2 Unsupervised Learning

- The computer is trained with unlabeled data.
- Here there's no teacher at all, actually the computer might be able to teach you new things after it learns patterns in data, these algorithms a particularly useful in cases where the human expert doesn't know what to look for in the data.
- They are the family of machine learning algorithms which are mainly used in pattern detection and descriptive modeling. However, there are no output categories or labels here based on which the algorithm can try to model relationships. These algorithms try to use techniques on the input data to mine for rules, detect patterns, and summarize and group the data points which help in deriving meaningful insights and describe the data better to the users.

List of Common Algorithms in Unsupervised Learning

- k-means clustering
- Association Rules

2.9.3 Semi-supervised Learning

In the previous two types, either there are no labels for all the observation in the dataset or labels are present for all the observations. Semi-supervised learning falls in between these two. In many practical situations, the cost to label is quite high, since it requires skilled human experts to do that. So,in the absence of labels in the majority of the observations but present in few, semi-supervised algorithms are the best candidates for the model building. These methods exploit the idea that even though the group memberships of the unlabeled data are unknown, this data carries important information about the group parameters.

2.9.4 Reinforcement Learning

- This method aims at using observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk. Reinforcement learning algorithm (called the agent) continuously learns from the environment in an iterative fashion. In the process, the agent learns from its experiences of the environment until it explores the full range of possible states.
- Reinforcement Learning is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

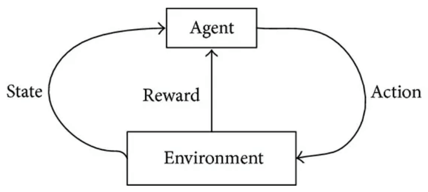


Figure 4

There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms.In the problem, an agent is supposed decide the best action to select based on his current state. When this step is repeated, the problem is known as a Markov Decision Process.

In order to produce intelligent programs (also called agents), reinforcement learning goes through the following steps:

- Input state is observed by the agent.
- Decision making function is used to make the agent perform an action.
- After the action is performed, the agent receives reward or reinforcement from the environment.
- The state-action pair information about the reward is stored.

List of Common Algorithms in Reinforcement Learning

- Q-Learning
- Temporal Difference (TD)
- Deep Adversarial Networks

2.10 Some of the Machine Learning Applications

Image and Object Recognition: ML is used for image classification, object detection, and facial recognition tasks. It enables computers to accurately identify and categorize images, making it useful in areas like self-driving cars, surveillance systems, and medical imaging.

Natural Language Processing (NLP): ML is applied in NLP to understand and process human language. It is used in applications such as sentiment analysis, language translation, chatbots, and voice assistants like Siri and Alexa.

Recommender Systems: ML algorithms are used to create personalized recommendations for users in various domains like e-commerce, streaming platforms, and online advertising. These algorithms analyze user behavior and preferences to suggest relevant products, movies, or content.

Fraud Detection: ML is used in financial institutions to detect fraudulent transactions by analyzing patterns and anomalies in large volumes of data. It helps in identifying potential fraud cases quickly and accurately.

Healthcare: ML is applied in healthcare for various purposes, including disease diagnosis, patient monitoring, drug discovery, and personalized medicine. ML models can analyze patient data, medical images, and genetic information to assist in diagnosis and treatment decisions.

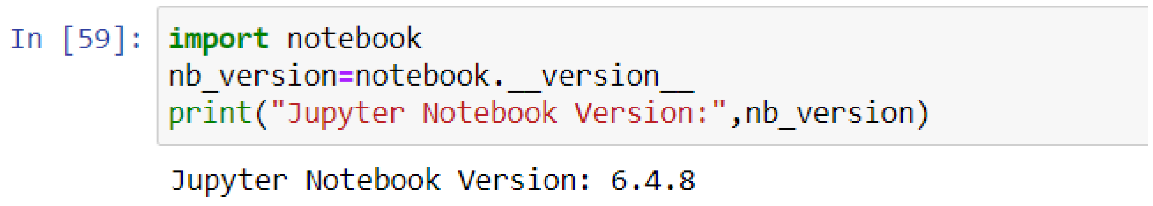
2.11 How ML is used in this dataset?

ML is used in this dataset for energy consumption analysis and prediction.By leveraging ML algorithms, patterns and relationships within the data can be discovered, enabling accurate forecasting of future energy consumption.This information is valuable for energy providers, allowing them to plan and allocate resources effectively, optimize energy production and distribution, and avoid unnecessary costs. ML models can analyze various factors influencing energy consumption, such as historical usage patterns and time of day, to provide insights into energy demand fluctuations.

3 SYSTEM REQUIREMENTS

3.1 Jupyter notebook version

Interactive coding and documentation platform for data analysis, machine learning, and scientific computing.

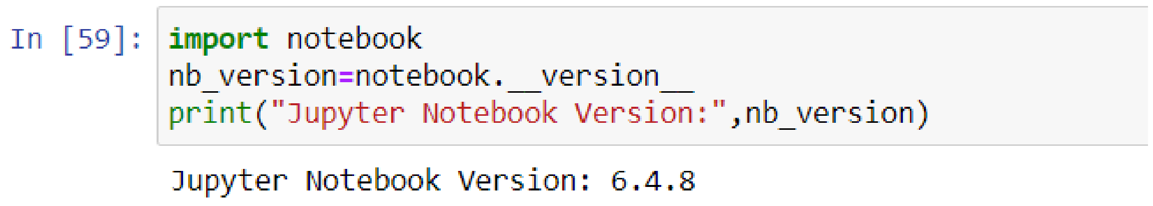


```
In [59]: import notebook
         nb_version=notebook.__version__
         print("Jupyter Notebook Version:",nb_version)

         Jupyter Notebook Version: 6.4.8
```

Figure 5

3.2 Python Version

High-level, versatile programming language known for its readability and extensive range of libraries and frameworks.



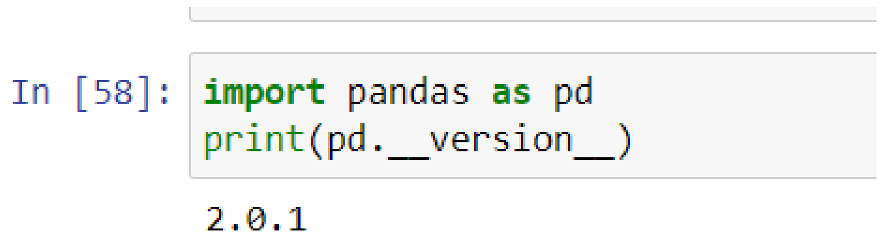
```
In [59]: import notebook
         nb_version=notebook.__version__
         print("Jupyter Notebook Version:",nb_version)

         Jupyter Notebook Version: 6.4.8
```

Figure 6

3.3 Libraries used and its version

3.3.1 Pandas

Python library for data manipulation and analysis.

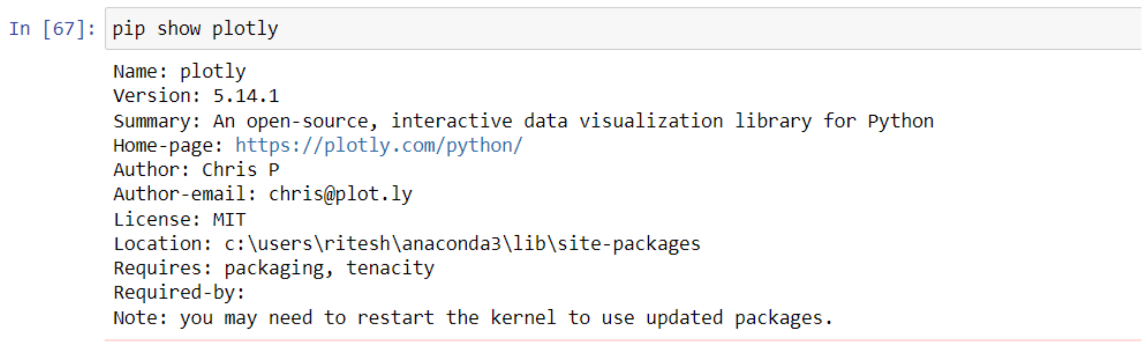


```
In [58]: import pandas as pd
         print(pd.__version__)

         2.0.1
```

Figure 7

3.3.2 Plotly

Interactive data visualization library for Python.

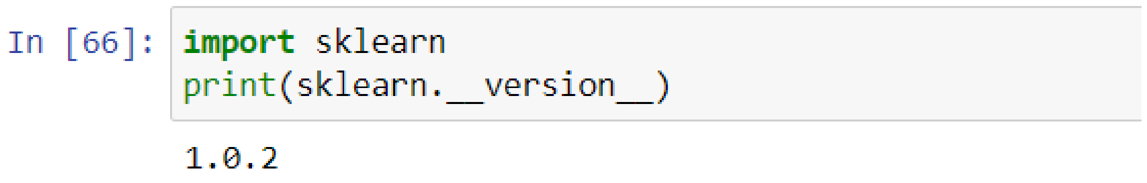


```
In [67]: pip show plotly

Name: plotly
Version: 5.14.1
Summary: An open-source, interactive data visualization library for Python
Home-page: https://plotly.com/python/
Author: Chris P
Author-email: chris@plot.ly
License: MIT
Location: c:\users\ritesh\anaconda3\lib\site-packages
Requires: packaging, tenacity
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

Figure 8

3.3.3 scikit-learn

Machine learning library for Python.



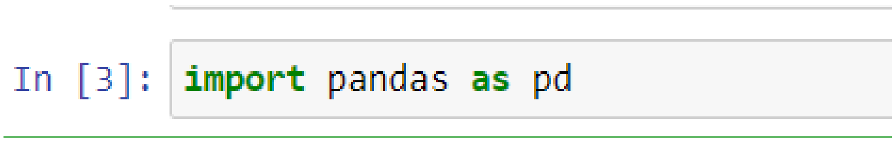
```
In [66]: import sklearn
         print(sklearn.__version__)

1.0.2
```

Figure 9

3.4 Implementation of libraries in dataset

3.4.1 Pandas

Pandas is used in this dataset for various data handling tasks such as reading the data from an Excel file (`pd.read_excel()`), data cleaning by dropping columns with null values and replacing zeros with means (`drop()`, `replace()`, `fillna()`), datetime conversion and setting the index (`pd.to_datetime()`, `set_index()`), resampling and aggregation (`resample()`, `sum()`, `mean()`, `max()`, `min()`), deriving a new column based on calculations (PF calculation), splitting the data into input features and target variable (`drop()`), and creating future predictions (`pd.date_range()`, creating and populating a future DataFrame).

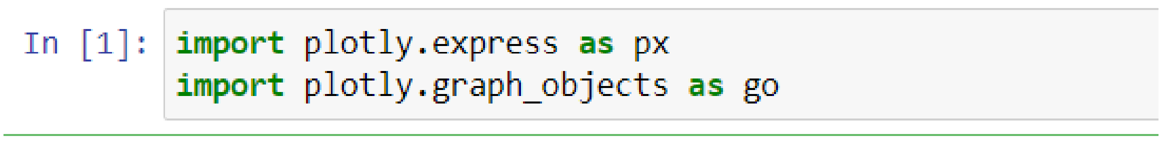


```
In [3]: import pandas as pd
```

Figure 10

3.4.2 Plotly

Plotly is used in this dataset for visualizing the data and creating interactive plots. It is used to create line plots (`px.line()`), customize the plot layout (`update_layout()`), and display the plot (`show()`). Additionally, Plotly is used to create line plots with markers and annotations (`go.Scatter()`). These visualizations help in analyzing and interpreting the data, allowing for a better understanding of patterns, trends, and relationships within the dataset.

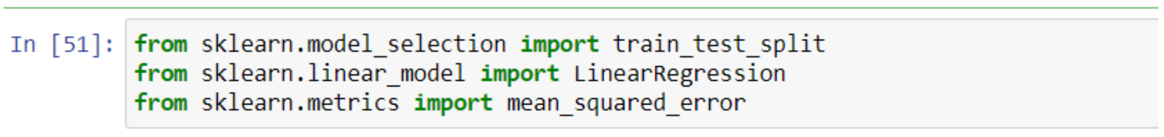


```
In [1]: import plotly.express as px
        import plotly.graph_objects as go
```

Figure 11

3.4.3 scikit-learn

`scikit-learn` is utilized in this dataset for various machine learning tasks. Firstly, the data is split into training and testing sets using the `train_test_split` function. Then, regression model, like Linear Regression, is trained using the `fit` method. The trained models are employed to make predictions on the testing set through the `predict` method. The performance of the models is assessed using the mean squared error (MSE) metric computed with the `mean_squared_error` function from `scikit-learn`. The comprehensive set of tools and algorithms offered by `scikit-learn` makes it a valuable library for analyzing and modeling the provided dataset.



```
In [51]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error
```

Figure 12

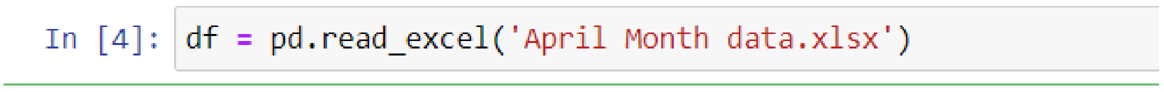
4 DATA ANALYSIS

4.1 Data Collection and Preparation

4.1.1 Data Source

Description: The code reads data from an Excel file named 'April Month data.xlsx' and assigns it to the DataFrame df. If the file is not found or cannot be read, it will raise an error.

Note: Ensure that the 'April Month data.xlsx' file is present in the specified location.

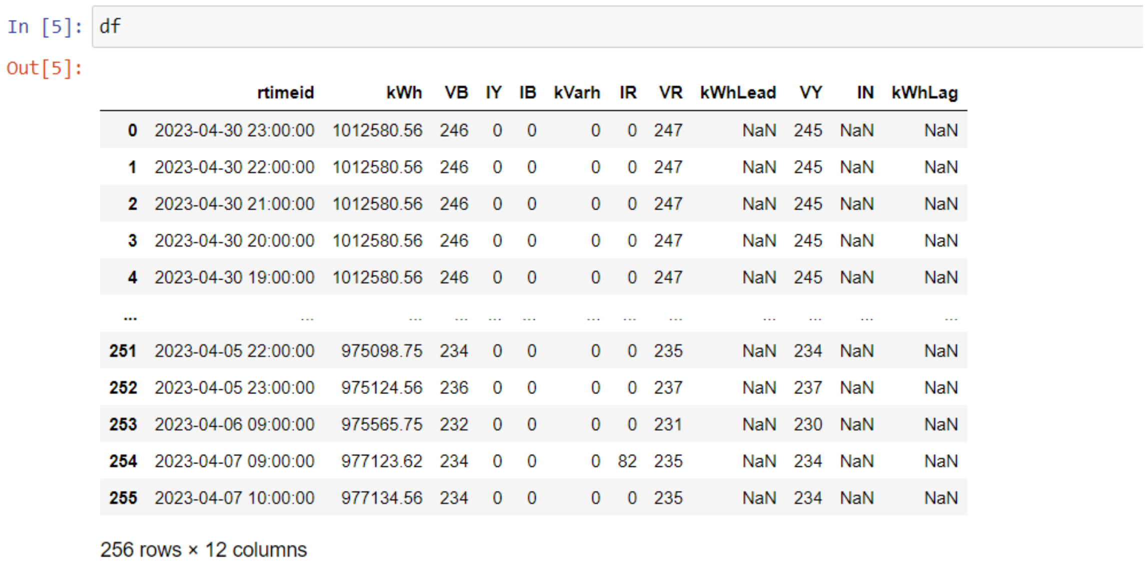


```
In [4]: df = pd.read_excel('April Month data.xlsx')
```

Figure 13

4.1.2 Data Collection

Description: The code displays the contents of the DataFrame 'df', showing the collected data. If there is an issue with the DataFrame or it contains no data, an empty DataFrame or an error message will be displayed.



```
In [5]: df
Out[5]:
```

	rtimeid	kWh	VB	IY	IB	kVarh	IR	VR	kWhLead	VY	IN	kWhLag
0	2023-04-30 23:00:00	1012580.56	246	0	0	0	0	247	NaN	245	NaN	NaN
1	2023-04-30 22:00:00	1012580.56	246	0	0	0	0	247	NaN	245	NaN	NaN
2	2023-04-30 21:00:00	1012580.56	246	0	0	0	0	247	NaN	245	NaN	NaN
3	2023-04-30 20:00:00	1012580.56	246	0	0	0	0	247	NaN	245	NaN	NaN
4	2023-04-30 19:00:00	1012580.56	246	0	0	0	0	247	NaN	245	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
251	2023-04-05 22:00:00	975098.75	234	0	0	0	0	235	NaN	234	NaN	NaN
252	2023-04-05 23:00:00	975124.56	236	0	0	0	0	237	NaN	237	NaN	NaN
253	2023-04-06 09:00:00	975565.75	232	0	0	0	0	231	NaN	230	NaN	NaN
254	2023-04-07 09:00:00	977123.62	234	0	0	0	82	235	NaN	234	NaN	NaN
255	2023-04-07 10:00:00	977134.56	234	0	0	0	0	235	NaN	234	NaN	NaN

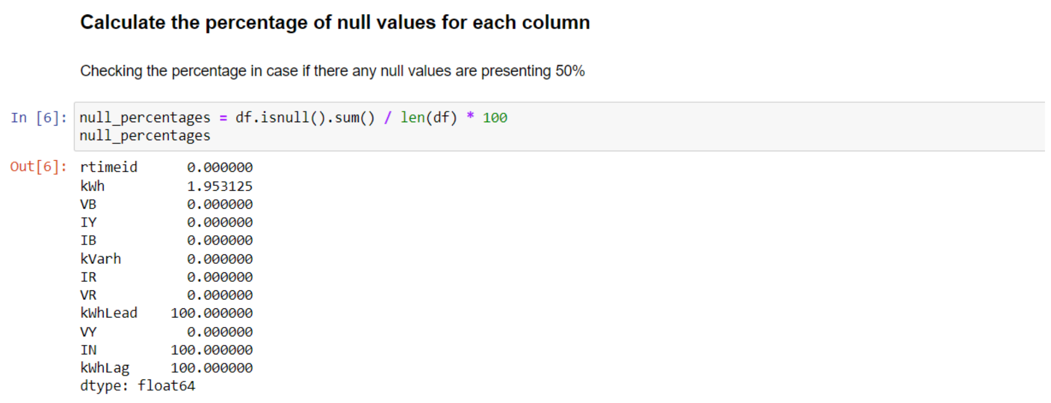
256 rows × 12 columns

Figure 14

4.1.3 Data Cleaning

- **Missing values**
  - **Description:** The code calculates the percentage of missing values in each column and identifies columns with more than 50% missing values. These columns

are then dropped from the DataFrame. If there are no missing values or if all columns have more than 50% missing values, the original DataFrame will remain unchanged.



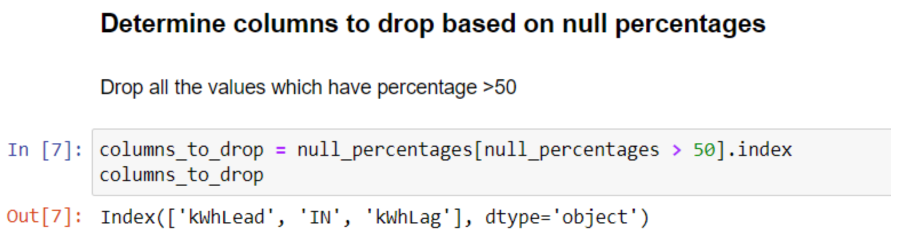
Calculate the percentage of null values for each column

Checking the percentage in case if there any null values are presenting 50%

```
In [6]: null_percentages = df.isnull().sum() / len(df) * 100
        null_percentages
```

```
Out[6]: rtimeid       0.000000
        kWh           1.953125
        VB            0.000000
        IY            0.000000
        IB            0.000000
        kVarh         0.000000
        IR            0.000000
        VR            0.000000
        kWhLead     100.000000
        VY            0.000000
        IN          100.000000
        kWhLag      100.000000
        dtype: float64
```

Figure 15



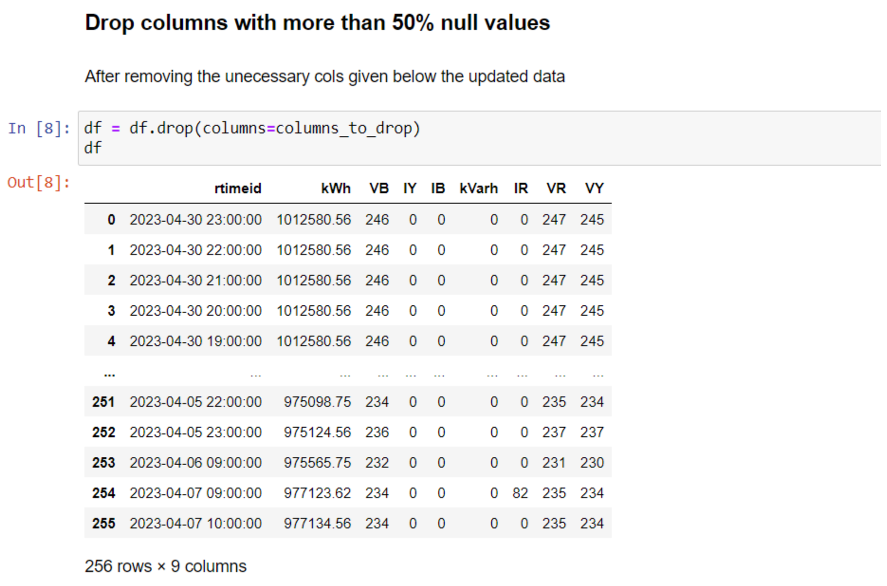
Determine columns to drop based on null percentages

Drop all the values which have percentage >50

```
In [7]: columns_to_drop = null_percentages[null_percentages > 50].index
        columns_to_drop
```

```
Out[7]: Index(['kWhLead', 'IN', 'kWhLag'], dtype='object')
```

Figure 16



Drop columns with more than 50% null values

After removing the unecessary cols given below the updated data

```
In [8]: df = df.drop(columns=columns_to_drop)
        df
```

Out[8]:

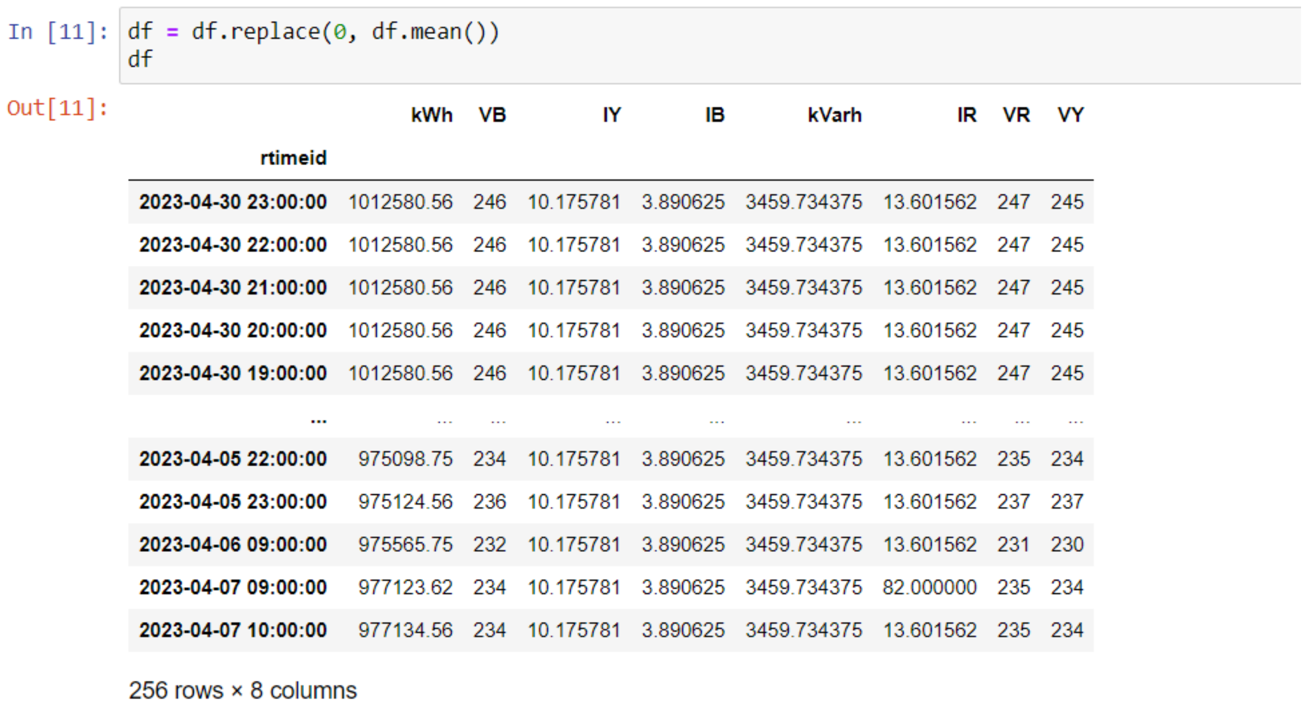
	rtimeid	kWh	VB	IY	IB	kVarh	IR	VR	VY
0	2023-04-30 23:00:00	1012580.56	246	0	0	0	0	247	245
1	2023-04-30 22:00:00	1012580.56	246	0	0	0	0	247	245
2	2023-04-30 21:00:00	1012580.56	246	0	0	0	0	247	245
3	2023-04-30 20:00:00	1012580.56	246	0	0	0	0	247	245
4	2023-04-30 19:00:00	1012580.56	246	0	0	0	0	247	245
...	...	...	...	...	...	...	...	...	...
251	2023-04-05 22:00:00	975098.75	234	0	0	0	0	235	234
252	2023-04-05 23:00:00	975124.56	236	0	0	0	0	237	237
253	2023-04-06 09:00:00	975565.75	232	0	0	0	0	231	230
254	2023-04-07 09:00:00	977123.62	234	0	0	0	82	235	234
255	2023-04-07 10:00:00	977134.56	234	0	0	0	0	235	234

256 rows × 9 columns

Figure 17

- **Note:**Since the whole columns have the null values it is not required to consider for our analysis because the data might not be usefull and also it leads to the lower accuracy rate so if the the null values are more than 50 percent i had removed those columns. In case if the null values are less than 50 percent then we can replace it with the mean values since that data might be useful for the futher prediction.

- **Zero Values**
  - **Description:** The code replaces zero values in the DataFrame with the column-wise mean. If there are no zero values or if the DataFrame has no numeric columns, the original DataFrame will remain unchanged.



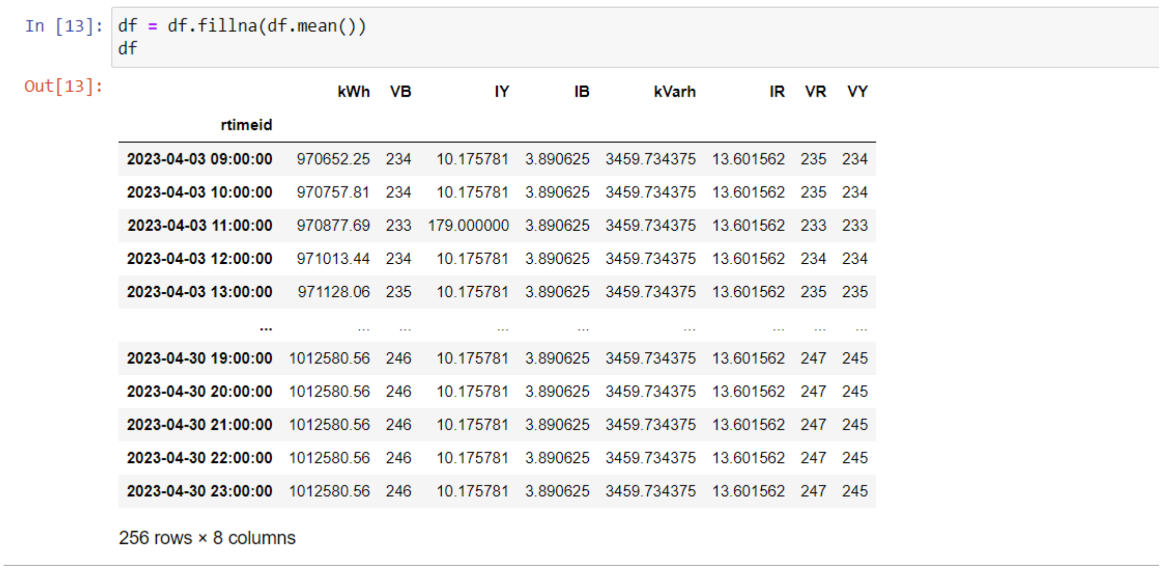
```
In [11]: df = df.replace(0, df.mean())
         df
```

```
Out[11]:
                            kWh   VB         IY        IB        kVarh         IR   VR   VY
             rtimeid
2023-04-30 23:00:00  1012580.56  246  10.175781  3.890625  3459.734375  13.601562  247  245
2023-04-30 22:00:00  1012580.56  246  10.175781  3.890625  3459.734375  13.601562  247  245
2023-04-30 21:00:00  1012580.56  246  10.175781  3.890625  3459.734375  13.601562  247  245
2023-04-30 20:00:00  1012580.56  246  10.175781  3.890625  3459.734375  13.601562  247  245
2023-04-30 19:00:00  1012580.56  246  10.175781  3.890625  3459.734375  13.601562  247  245
                 ...         ...  ...        ...       ...          ...        ...  ...  ...
2023-04-05 22:00:00   975098.75  234  10.175781  3.890625  3459.734375  13.601562  235  234
2023-04-05 23:00:00   975124.56  236  10.175781  3.890625  3459.734375  13.601562  237  237
2023-04-06 09:00:00   975565.75  232  10.175781  3.890625  3459.734375  13.601562  231  230
2023-04-07 09:00:00   977123.62  234  10.175781  3.890625  3459.734375  82.000000  235  234
2023-04-07 10:00:00   977134.56  234  10.175781  3.890625  3459.734375  13.601562  235  234

256 rows × 8 columns
```

Figure 18

- **Note:**The zeros in the data represent missing values or data entry errors, it might be appropriate to fill them with the mean value.So filling in missing values with the mean can help maintain the overall distribution and reduce the impact of missing data on subsequent analyses.

-Filling Nan Values with mean: The zeros in the data represent missing values or data entry errors, it might be appropriate to fill them with the mean value.So filling in missing values with the mean can help maintain the overall distribution and reduce the impact of missing data on subsequent analyses.



```
In [13]: df = df.fillna(df.mean())
         df
```

Out[13]:

rtimeid	kWh	VB	IY	IB	kVarh	IR	VR	VY
2023-04-03 09:00:00	970652.25	234	10.175781	3.890625	3459.734375	13.601562	235	234
2023-04-03 10:00:00	970757.81	234	10.175781	3.890625	3459.734375	13.601562	235	234
2023-04-03 11:00:00	970877.69	233	179.000000	3.890625	3459.734375	13.601562	233	233
2023-04-03 12:00:00	971013.44	234	10.175781	3.890625	3459.734375	13.601562	234	234
2023-04-03 13:00:00	971128.06	235	10.175781	3.890625	3459.734375	13.601562	235	235
...	...	...	...	...	...	...	...	...
2023-04-30 19:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 20:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 21:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 22:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 23:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245

256 rows × 8 columns

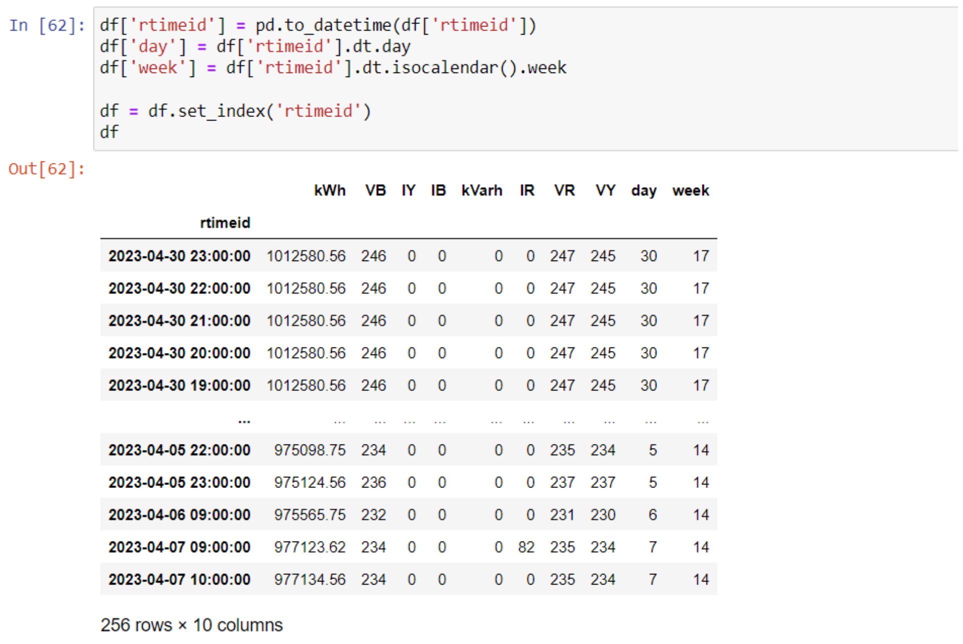
Figure 19

Note: The reason to use this method is not to leave the null values as it ease because it causes inappropriate results or we can say it produces an inaccurate result.The meaning here is in the given data if the given data has less than 50 percent of the null values.

4.1.4 Data Formating

- **Date and Time**
  - **Description:** The code converts the `rtimeid` column to datetime format and sets it as the index of the DataFrame, enabling easier time-based analysis. If the `rtimeid` column does not exist or cannot be converted to datetime format, an error will be raised.
  - **Note:**
    ->The code snippet `df['day'] = df['rtimeid'].dt.day` extracts the day component from the `'rtimeid'` column of the DataFrame `df` using the `dt` accessor provided by pandas. It creates a new column named `'day'` and assigns the day values to it.
    ->The code snippet `df['week'] = df['rtimeid'].dt.isocalendar().week` extracts the week number from the `'rtimeid'` column using the `isocalendar()` function provided by pandas. It returns a tuple containing ISO year, ISO week number, and ISO weekday for each datetime value. By accessing the `'week'` component of the tuple, it assigns the week numbers to the `'week'` column in the DataFrame.
    ->Including 'day' and 'week' as features in the dataset can capture the influence

of specific days and weeks on energy consumption. For example, certain days of the week or weeks of the month might exhibit higher or lower energy consumption due to various factors such as weekends, holidays, or specific events. By incorporating these features into the predictive model, it can potentially improve the accuracy of energy consumption predictions by capturing such temporal patterns and trends.

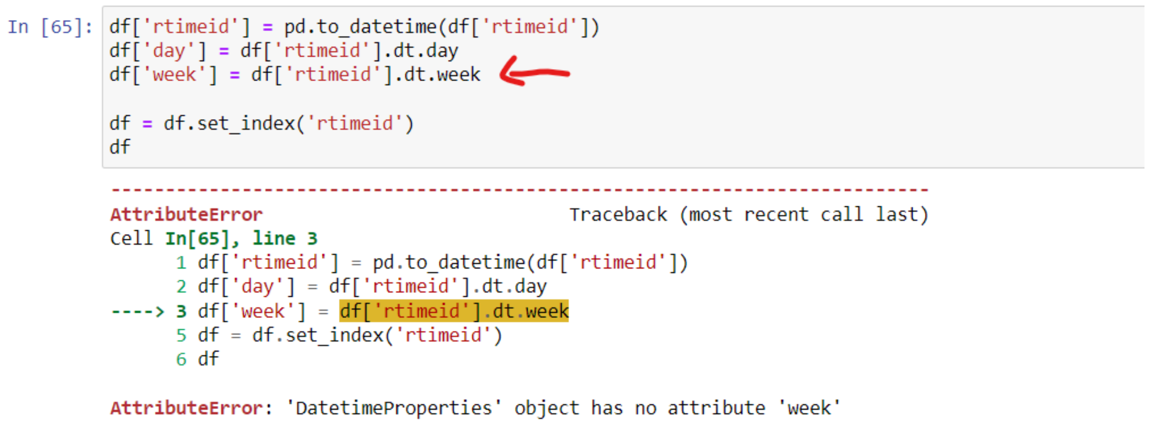


```
In [62]: df['rtimeid'] = pd.to_datetime(df['rtimeid'])
         df['day'] = df['rtimeid'].dt.day
         df['week'] = df['rtimeid'].dt.isocalendar().week

         df = df.set_index('rtimeid')
         df
Out[62]:
                           kWh   VB  IY  IB  kVarh  IR   VR   VY  day  week
            rtimeid
2023-04-30 23:00:00  1012580.56  246   0   0      0   0  247  245   30    17
2023-04-30 22:00:00  1012580.56  246   0   0      0   0  247  245   30    17
2023-04-30 21:00:00  1012580.56  246   0   0      0   0  247  245   30    17
2023-04-30 20:00:00  1012580.56  246   0   0      0   0  247  245   30    17
2023-04-30 19:00:00  1012580.56  246   0   0      0   0  247  245   30    17
                ...         ...  ...  ..  ..    ...  ..  ...  ...  ...   ...
2023-04-05 22:00:00   975098.75  234   0   0      0   0  235  234    5    14
2023-04-05 23:00:00   975124.56  236   0   0      0   0  237  237    5    14
2023-04-06 09:00:00   975565.75  232   0   0      0   0  231  230    6    14
2023-04-07 09:00:00   977123.62  234   0   0      0  82  235  234    7    14
2023-04-07 10:00:00   977134.56  234   0   0      0   0  235  234    7    14

256 rows × 10 columns
```

Figure 20

– **Important point**

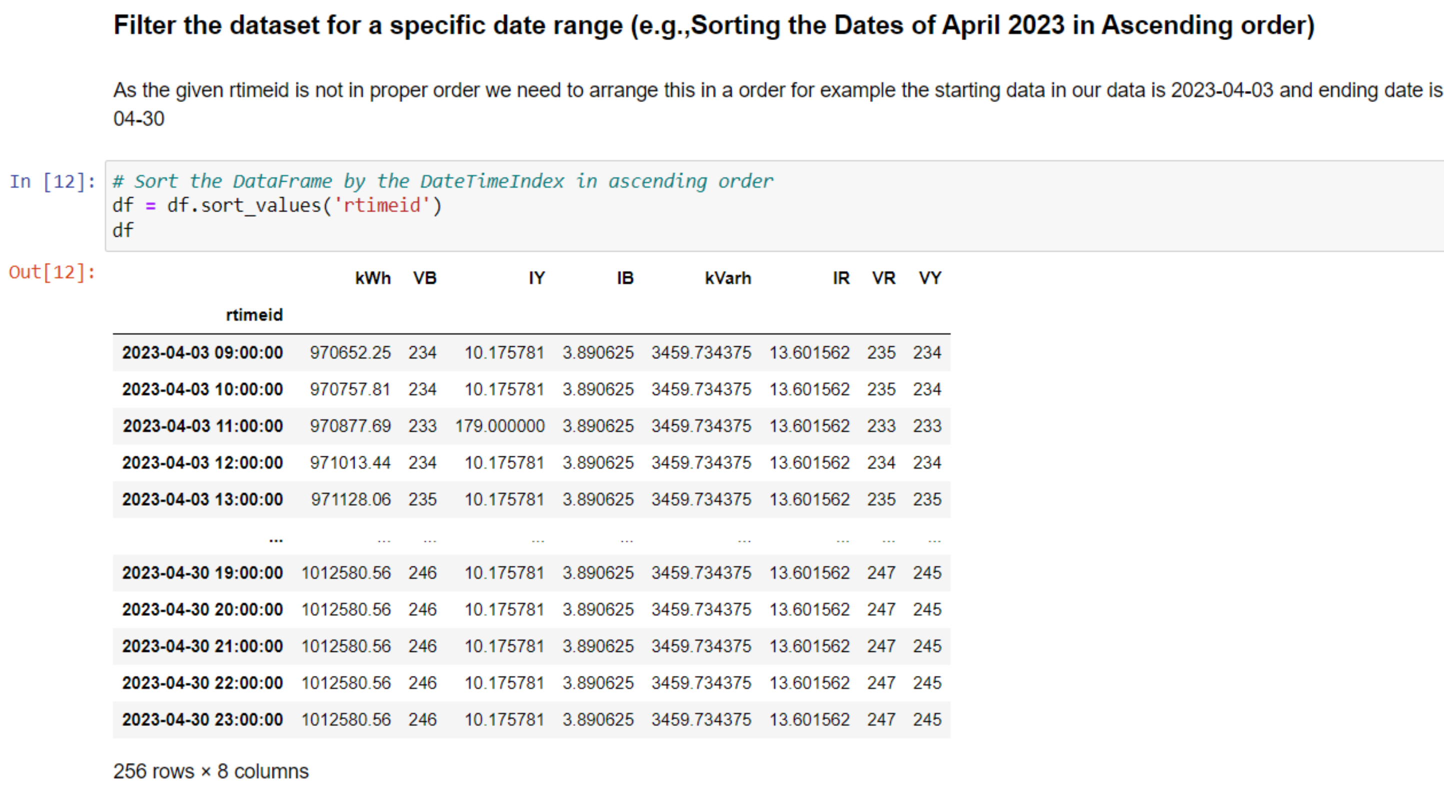
The code snippet `df['week']=df['rtimeid'].dt.week` attempts to extract the week number from the `'rtimeid'` column using the week attribute available through the `dt` accessor provided by pandas. However, the 'week' attribute is not available directly through the `dt` accessor in the version of pandas being used.



```
In [65]: df['rtimeid'] = pd.to_datetime(df['rtimeid'])
         df['day'] = df['rtimeid'].dt.day
         df['week'] = df['rtimeid'].dt.week

         df = df.set_index('rtimeid')
         df

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[65], line 3
      1 df['rtimeid'] = pd.to_datetime(df['rtimeid'])
      2 df['day'] = df['rtimeid'].dt.day
----> 3 df['week'] = df['rtimeid'].dt.week
      5 df = df.set_index('rtimeid')
      6 df

AttributeError: 'DatetimeProperties' object has no attribute 'week'
```

Figure 21

- **Fliter Data**
  - **Description:** The code `df = df.sort_values('rtimeid')` sorts the DataFrame `df` based on the values in the `'rtimeid'` column in ascending order.
    By sorting the DataFrame, the rows are rearranged so that the data is ordered chronologically based on the values in the `'rtimeid'` column. The DataFrame is modified in-place, meaning that the original DataFrame is reorganized rather than creating a new sorted DataFrame.



Filter the dataset for a specific date range (e.g.,Sorting the Dates of April 2023 in Ascending order)

As the given rtimeid is not in proper order we need to arrange this in a order for example the starting data in our data is 2023-04-03 and ending date is
04-30

```
In [12]: # Sort the DataFrame by the DateTimeIndex in ascending order
         df = df.sort_values('rtimeid')
         df
```

Out[12]:

	kWh	VB	IY	IB	kVarh	IR	VR	VY
rtimeid								
2023-04-03 09:00:00	970652.25	234	10.175781	3.890625	3459.734375	13.601562	235	234
2023-04-03 10:00:00	970757.81	234	10.175781	3.890625	3459.734375	13.601562	235	234
2023-04-03 11:00:00	970877.69	233	179.000000	3.890625	3459.734375	13.601562	233	233
2023-04-03 12:00:00	971013.44	234	10.175781	3.890625	3459.734375	13.601562	234	234
2023-04-03 13:00:00	971128.06	235	10.175781	3.890625	3459.734375	13.601562	235	235
...	...	...	...	...	...	...	...	...
2023-04-30 19:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 20:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 21:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 22:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245
2023-04-30 23:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245

256 rows × 8 columns

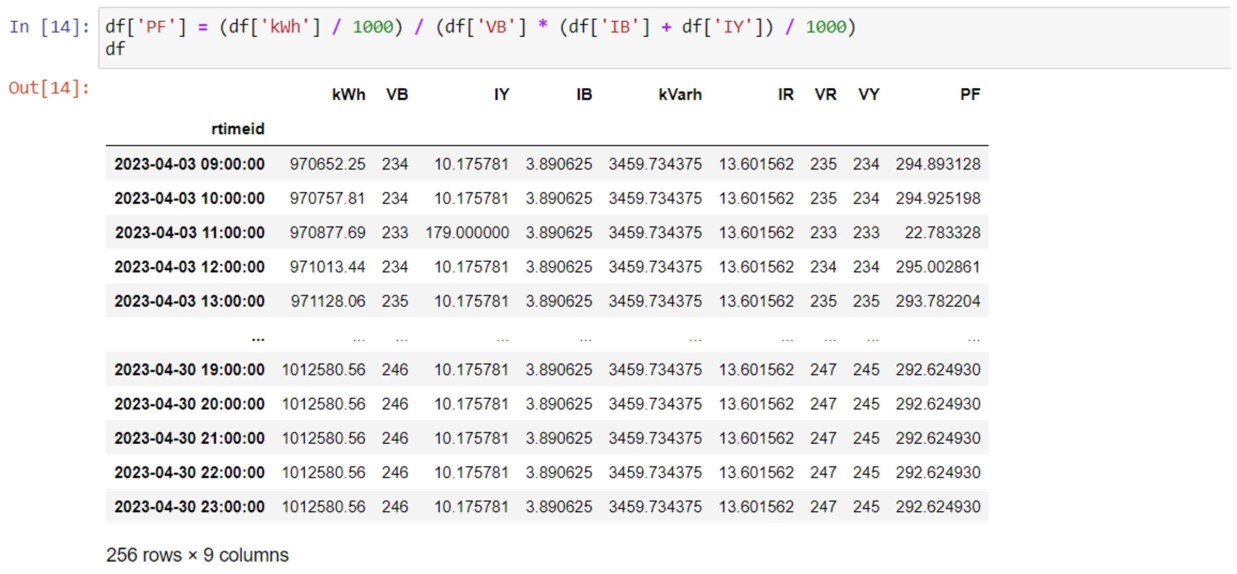
Figure 22

  - **Note:** Sorting the DataFrame ensures that the data is arranged in chronological order based on the DateTimeIndex. This is important for conducting sequential analysis and observing any patterns or trends over time.

4.2 Power Factor Analysis

4.2.1 Calculating power factor

The code calculates the Power Factor (PF) using the provided formulas and adds a new column ’PF’ to the DataFrame. If the required columns (’kWh’, ’VB’, ’IB’, ’IY’) are missing or have invalid values, an error will be raised.



```
In [14]: df['PF'] = (df['kWh'] / 1000) / (df['VB'] * (df['IB'] + df['IY']) / 1000)
         df
```

Out[14]:

rtimeid	kWh	VB	IY	IB	kVarh	IR	VR	VY	PF
2023-04-03 09:00:00	970652.25	234	10.175781	3.890625	3459.734375	13.601562	235	234	294.893128
2023-04-03 10:00:00	970757.81	234	10.175781	3.890625	3459.734375	13.601562	235	234	294.925198
2023-04-03 11:00:00	970877.69	233	179.000000	3.890625	3459.734375	13.601562	233	233	22.783328
2023-04-03 12:00:00	971013.44	234	10.175781	3.890625	3459.734375	13.601562	234	234	295.002861
2023-04-03 13:00:00	971128.06	235	10.175781	3.890625	3459.734375	13.601562	235	235	293.782204
...	...	...	...	...	...	...	...	...	...
2023-04-30 19:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245	292.624930
2023-04-30 20:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245	292.624930
2023-04-30 21:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245	292.624930
2023-04-30 22:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245	292.624930
2023-04-30 23:00:00	1012580.56	246	10.175781	3.890625	3459.734375	13.601562	247	245	292.624930

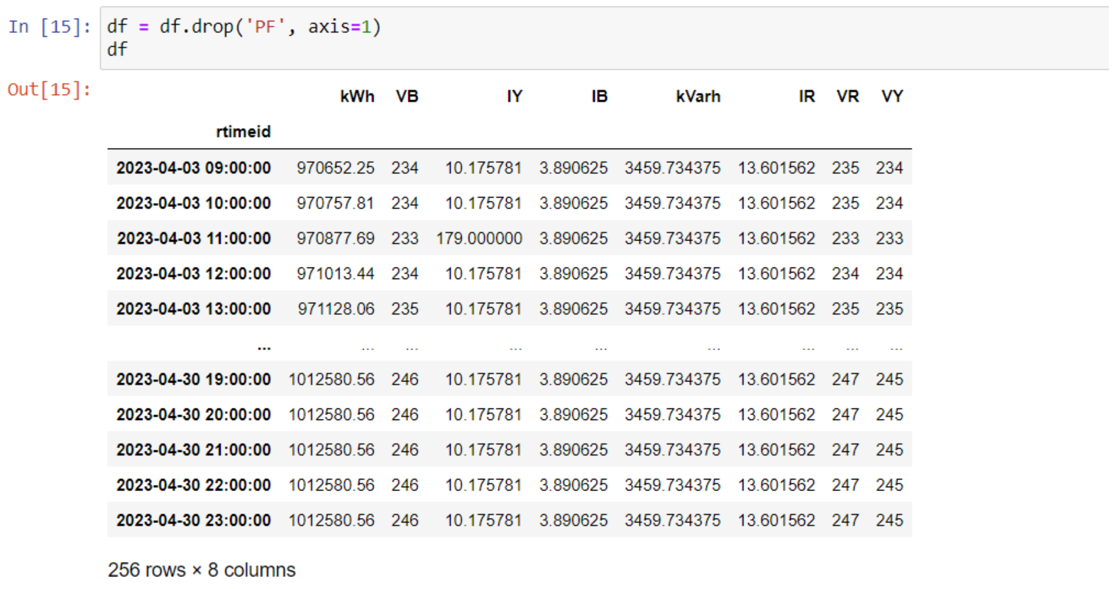
256 rows × 9 columns

Figure 23

- **Important Points**
  ->The reason to calculate this is to identify how efficiently the electrical power is being utilized in a system.
  So we can find the power factor by using factors like current and voltage in different phases. Since 1 kilowatt-hour (kWh) is equal to 1 kilowatt (kW) consumed in 1 hour, dividing kWh by 1000 converts it from watt-hours to kilowatt-hours (kWh to kW). The VB column represents voltage values, and the (IB + IY) expression represents the sum of current values. In the power factor calculation, these values should typically be expressed in kilovolts (kV) and kiloamperes (kA), respectively. Dividing them by 1000 adjusts the units from volts (V) and amperes (A) to kilovolts (kV) and kiloamperes (kA).

  ->From the above a power factor value of 294.893128 is quite high and seems unusual. Typically, power factor values range from -1 to 1, where a value of 1 indicates a purely resistive load,0 indicates a purely reactive load, and -1 indicates a load with a 180-degree phase shift.
  ->A power factor value of 1 indicates a purely resistive load. In this case, the current and voltage waveforms are in phase, meaning the load consumes real power without any reactive power.An Example of purely resistive loads include light bulbs.
  ->A power factor value of 0 indicates a purely reactive load. This means that the load

consumes reactive power without any real power. Reactive power is required to support the magnetic or electric fields in inductive or capacitive components. Examples of purely reactive loads include ideal inductors or capacitors.
->A power factor value of -1 indicates a load with a 180-degree phase shift between voltage and current waveforms. This usually occurs in certain types of electronic circuits or systems. In such cases, the load may return power back to the source rather than consuming it.Example Solar power systems.
So it is better to drop the PF column from the dataframe as it gives the unusual data



```
In [15]: df = df.drop('PF', axis=1)
         df
```

```
Out[15]:
                           kWh   VB          IY        IB        kVarh         IR   VR   VY
             rtimeid
 2023-04-03 09:00:00   970652.25  234   10.175781  3.890625  3459.734375  13.601562  235  234
 2023-04-03 10:00:00   970757.81  234   10.175781  3.890625  3459.734375  13.601562  235  234
 2023-04-03 11:00:00   970877.69  233  179.000000  3.890625  3459.734375  13.601562  233  233
 2023-04-03 12:00:00   971013.44  234   10.175781  3.890625  3459.734375  13.601562  234  234
 2023-04-03 13:00:00   971128.06  235   10.175781  3.890625  3459.734375  13.601562  235  235
                 ...         ...  ...         ...       ...          ...        ...  ...  ...
 2023-04-30 19:00:00  1012580.56  246   10.175781  3.890625  3459.734375  13.601562  247  245
 2023-04-30 20:00:00  1012580.56  246   10.175781  3.890625  3459.734375  13.601562  247  245
 2023-04-30 21:00:00  1012580.56  246   10.175781  3.890625  3459.734375  13.601562  247  245
 2023-04-30 22:00:00  1012580.56  246   10.175781  3.890625  3459.734375  13.601562  247  245
 2023-04-30 23:00:00  1012580.56  246   10.175781  3.890625  3459.734375  13.601562  247  245

256 rows × 8 columns
```

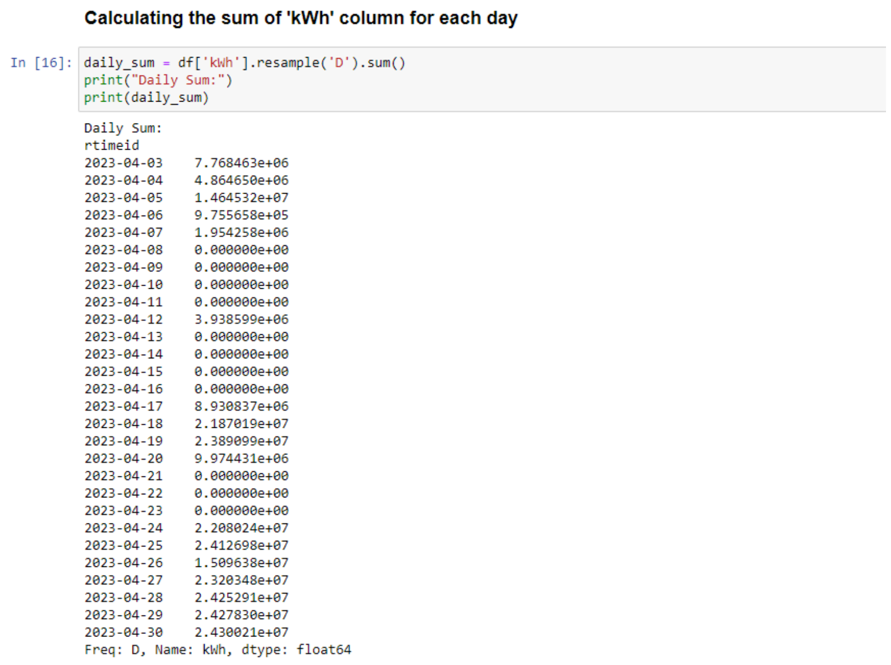
Figure 24

4.3 Power Consumption Analysis

4.3.1 Calculating the sum of 'kWh' column for each day

Description: The code `df['kWh'].resample('D').sum()` calculates the daily sum of the `'kWh'` values in the DataFrame.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `resample('D')` is a method that allows us to change the frequency of the data. In this case, `'D'` stands for daily frequency, meaning we want to group the data by day.
- `sum()` is used to calculate the sum of the `'kWh'` values within each day. It adds up all the energy consumption values for each day.



Calculating the sum of 'kWh' column for each day

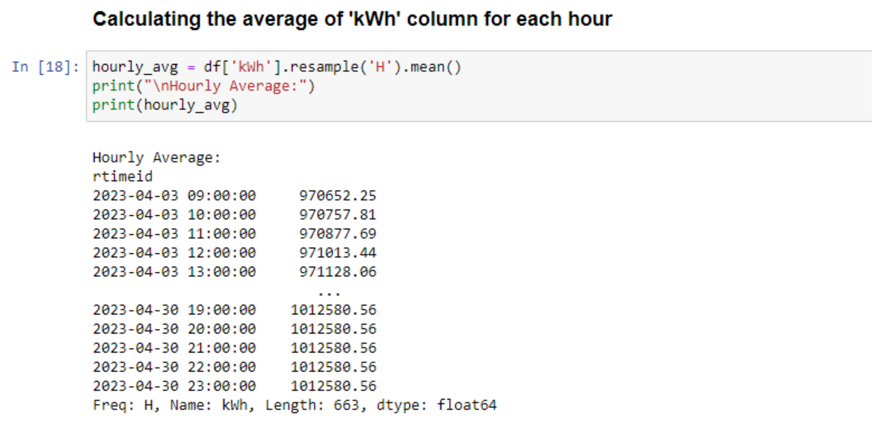
```
In [16]: daily_sum = df['kWh'].resample('D').sum()
         print("Daily Sum:")
         print(daily_sum)

Daily Sum:
rtimeid
2023-04-03    7.768463e+06
2023-04-04    4.864650e+06
2023-04-05    1.464532e+07
2023-04-06    9.755658e+05
2023-04-07    1.954258e+06
2023-04-08    0.000000e+00
2023-04-09    0.000000e+00
2023-04-10    0.000000e+00
2023-04-11    0.000000e+00
2023-04-12    3.938599e+06
2023-04-13    0.000000e+00
2023-04-14    0.000000e+00
2023-04-15    0.000000e+00
2023-04-16    0.000000e+00
2023-04-17    8.930837e+06
2023-04-18    2.187019e+07
2023-04-19    2.389099e+07
2023-04-20    9.974431e+06
2023-04-21    0.000000e+00
2023-04-22    0.000000e+00
2023-04-23    0.000000e+00
2023-04-24    2.208024e+07
2023-04-25    2.412698e+07
2023-04-26    1.509638e+07
2023-04-27    2.320348e+07
2023-04-28    2.425291e+07
2023-04-29    2.427830e+07
2023-04-30    2.430021e+07
Freq: D, Name: kWh, dtype: float64
```

Figure 25

4.3.2 Calculating the average of ’kWh’ column for each hour

Description: The code `df['kWh'].resample('H').mean()` calculates the hourly average of the `'kWh'` values in the DataFrame.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `resample('H')` is a method that allows us to change the frequency of the data. In this case, `'H'` stands for hourly frequency, meaning we want to group the data by hour.
- `mean()` is used to calculate the average of the `'kWh'` values within each hour. It calculates the mean energy consumption for each hour.



Calculating the average of 'kWh' column for each hour

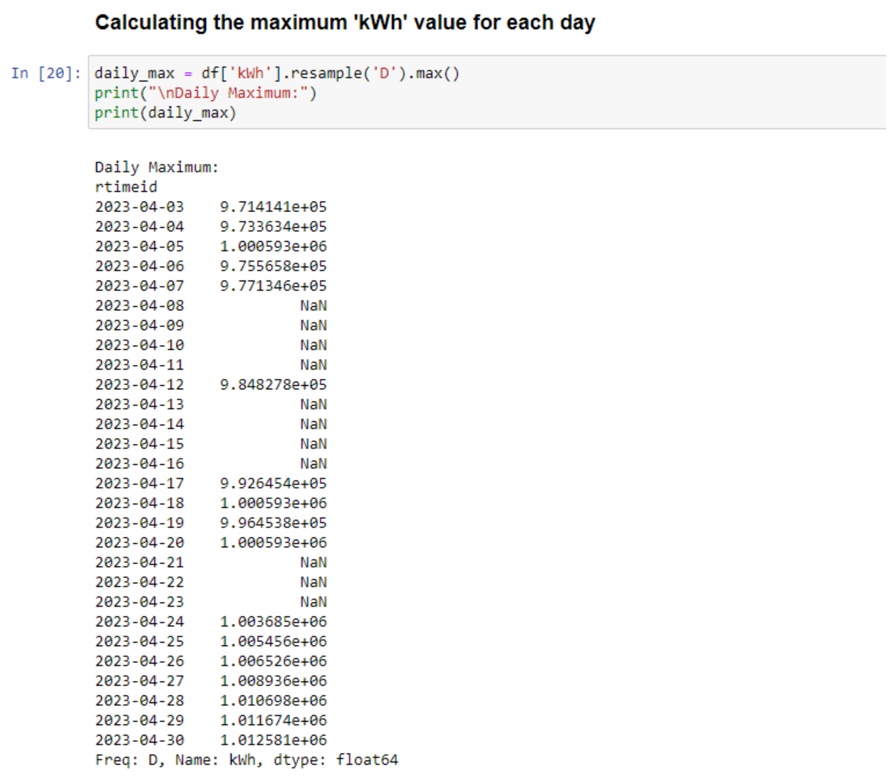
```
In [18]: hourly_avg = df['kWh'].resample('H').mean()
         print("\nHourly Average:")
         print(hourly_avg)

Hourly Average:
rtimeid
2023-04-03 09:00:00     970652.25
2023-04-03 10:00:00     970757.81
2023-04-03 11:00:00     970877.69
2023-04-03 12:00:00     971013.44
2023-04-03 13:00:00     971128.06
                          ...
2023-04-30 19:00:00    1012580.56
2023-04-30 20:00:00    1012580.56
2023-04-30 21:00:00    1012580.56
2023-04-30 22:00:00    1012580.56
2023-04-30 23:00:00    1012580.56
Freq: H, Name: kWh, Length: 663, dtype: float64
```

Figure 26

4.3.3 Calculating the maximum 'kWh' value for each day

Description: The code `df['kWh'].resample('D').max()` calculates the daily maximum value of the `'kWh'` column in the DataFrame.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `resample('D')` is a method that allows us to change the frequency of the data. In this case, `'D'` stands for daily frequency, meaning we want to group the data by day.
- `max()` is used to find the maximum value of the `'kWh'` column within each day. It identifies the highest energy consumption recorded in a day.



Calculating the maximum 'kWh' value for each day

```
In [20]: daily_max = df['kWh'].resample('D').max()
         print("\nDaily Maximum:")
         print(daily_max)
```

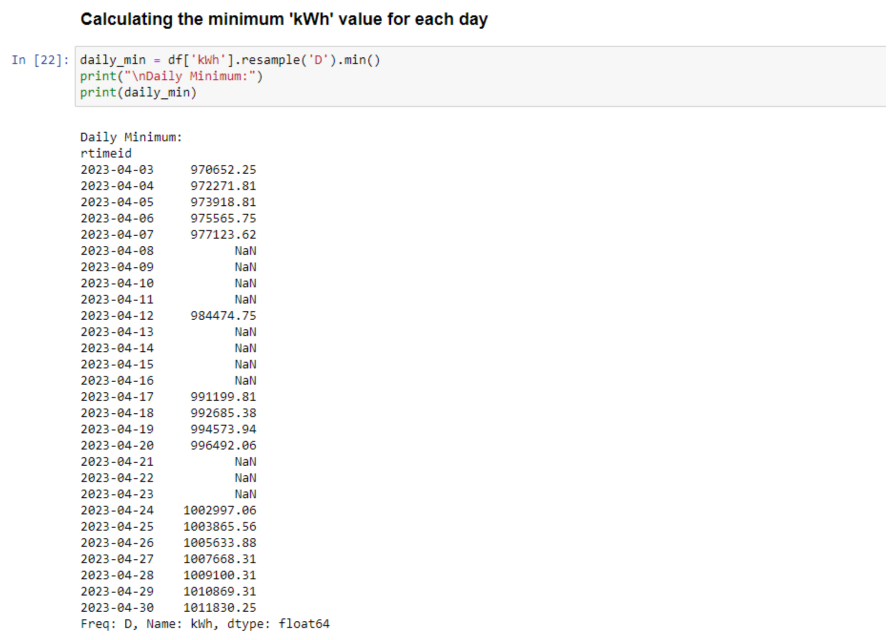
```
Daily Maximum:
rtimeid
2023-04-03    9.714141e+05
2023-04-04    9.733634e+05
2023-04-05    1.000593e+06
2023-04-06    9.755658e+05
2023-04-07    9.771346e+05
2023-04-08             NaN
2023-04-09             NaN
2023-04-10             NaN
2023-04-11             NaN
2023-04-12    9.848278e+05
2023-04-13             NaN
2023-04-14             NaN
2023-04-15             NaN
2023-04-16             NaN
2023-04-17    9.926454e+05
2023-04-18    1.000593e+06
2023-04-19    9.964538e+05
2023-04-20    1.000593e+06
2023-04-21             NaN
2023-04-22             NaN
2023-04-23             NaN
2023-04-24    1.003685e+06
2023-04-25    1.005456e+06
2023-04-26    1.006526e+06
2023-04-27    1.008936e+06
2023-04-28    1.010698e+06
2023-04-29    1.011674e+06
2023-04-30    1.012581e+06
Freq: D, Name: kWh, dtype: float64
```

Figure 27

4.3.4 Calculating the minimum 'kWh' value for each day

Description: The code `df['kWh'].resample('D').min()` calculates the daily minimum value of the `'kWh'` column in the DataFrame.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `resample('D')` is a method that allows us to change the frequency of the data. In this case, `'D'` stands for daily frequency, meaning we want to group the data by day.
- `min()` is used to find the minimum value of the `'kWh'` column within each day. It identifies the lowest energy consumption recorded in a day.



Calculating the minimum 'kWh' value for each day

```
In [22]: daily_min = df['kWh'].resample('D').min()
         print("\nDaily Minimum:")
         print(daily_min)
```

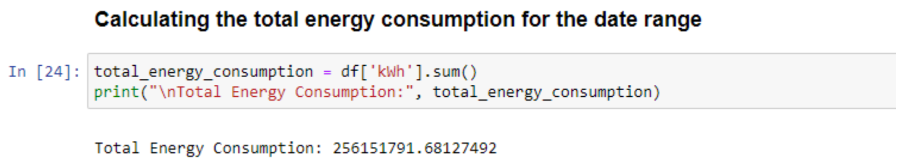
```
Daily Minimum:
rtimeid
2023-04-03     970652.25
2023-04-04     972271.81
2023-04-05     973918.81
2023-04-06     975565.75
2023-04-07     977123.62
2023-04-08           NaN
2023-04-09           NaN
2023-04-10           NaN
2023-04-11           NaN
2023-04-12     984474.75
2023-04-13           NaN
2023-04-14           NaN
2023-04-15           NaN
2023-04-16           NaN
2023-04-17     991199.81
2023-04-18     992685.38
2023-04-19     994573.94
2023-04-20     996492.06
2023-04-21           NaN
2023-04-22           NaN
2023-04-23           NaN
2023-04-24    1002997.06
2023-04-25    1003865.56
2023-04-26    1005633.88
2023-04-27    1007668.31
2023-04-28    1009100.31
2023-04-29    1010869.31
2023-04-30    1011830.25
Freq: D, Name: kWh, dtype: float64
```

Figure 28

4.3.5 Calculating the total energy consumption for the date range

Description: The code `df['kWh'].sum()` calculates the total energy consumption by summing up all the values in the `'kWh'` column of the DataFrame.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `sum()` is a function that adds up all the values in the `'kWh'` column.



Calculating the total energy consumption for the date range

```
In [24]: total_energy_consumption = df['kWh'].sum()
         print("\nTotal Energy Consumption:", total_energy_consumption)
```

```
Total Energy Consumption: 256151791.68127492
```

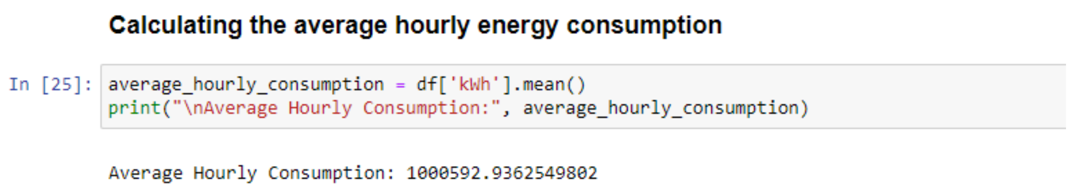
Figure 29

Note: The value "Total Energy Consumption: 256151791.68127492" indicates the total amount of energy consumed over the given period of time. In this case, it represents the total energy consumption recorded in the dataset.
In the context of the given dataset, it implies that the cumulative energy consumed by the load or system, as recorded in the dataset, is approximately 256,151,791.68 kilowatt-hours. This value provides an overall measure of the energy usage during the period covered by the dataset.

4.3.6 Calculating the average hourly energy consumption

Description: The code `df['kWh'].mean()` calculates the average hourly consumption of energy.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `mean()` is a function that calculates the arithmetic mean of all the values in the `'kWh'` column.


Calculating the average hourly energy consumption

```
In [25]: average_hourly_consumption = df['kWh'].mean()
         print("\nAverage Hourly Consumption:", average_hourly_consumption)

Average Hourly Consumption: 1000592.9362549802
```

Figure 30

hl**Note:**The value "Average Hourly Consumption: 1000592.9362549802" represents the average amount of energy consumed per hour. It indicates the average rate at which energy is being consumed by the load or system over the given period of time.
In this case, the average hourly consumption value is approximately 1,000,592.94 kilowatt-hours per hour. This means that, on average, the load or system is consuming energy at a rate of approximately 1,000,592.94 kilowatt-hours per hour.
The average hourly consumption is a useful metric to understand the typical energy consumption pattern and rate of energy usage over time.

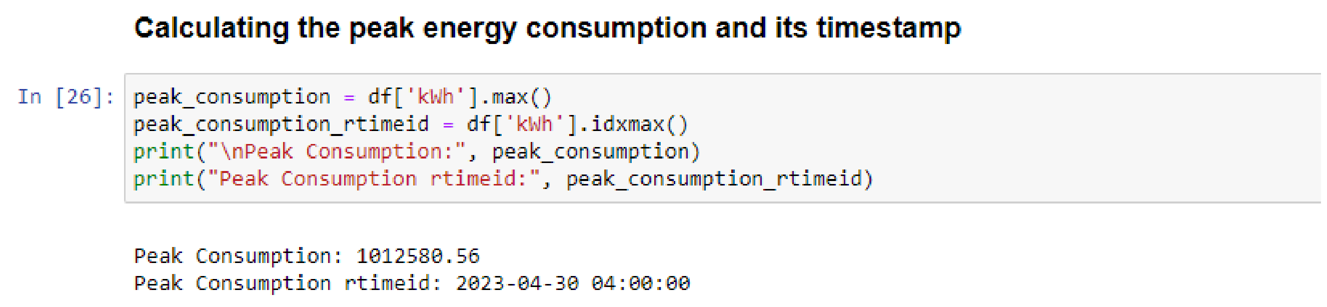
4.3.7 Calculating the peak energy consumption and its timestamp

Description: The code `df['kWh'].max()` calculates the maximum value of energy consumption, representing the peak consumption.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `max()` is a function that finds the highest value in the `'kWh'` column, indicating the peak energy consumption.

The result of `df['kWh'].max()` is stored in the variable `peak_consumption`, representing the peak energy consumption value.

Additionally, `df['kWh'].idxmax()` finds the index (time) corresponding to the maximum value in the `'kWh'` column. It identifies the specific time when the peak consumption occurred.



Calculating the peak energy consumption and its timestamp

```
In [26]: peak_consumption = df['kWh'].max()
         peak_consumption_rtimeid = df['kWh'].idxmax()
         print("\nPeak Consumption:", peak_consumption)
         print("Peak Consumption rtimeid:", peak_consumption_rtimeid)
```

```
Peak Consumption: 1012580.56
Peak Consumption rtimeid: 2023-04-30 04:00:00
```

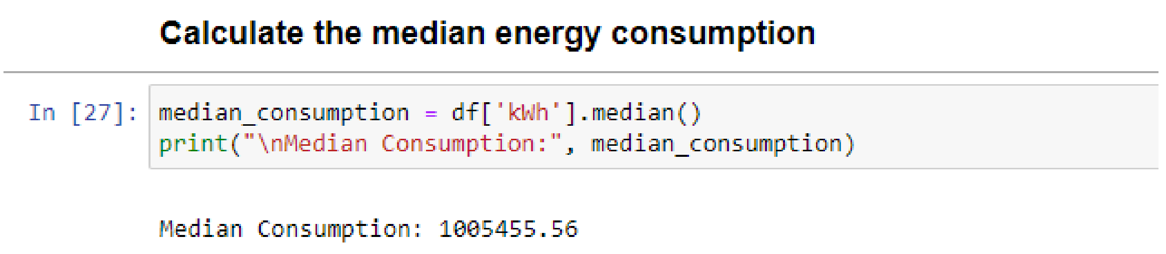
Figure 31

Note: The values "Peak Consumption: 1012580.56" and "Peak Consumption rtimeid: 2023-04-30 04:00:00" represent the highest energy consumption recorded during the given period and the corresponding timestamp when the peak consumption occurred. In this case, the peak consumption value is 1,012,580.56 kilowatt-hours. This indicates the maximum amount of energy consumed within the dataset. The peak consumption rtimeid, which is 2023-04-30 04:00:00, represents the specific date and time when the peak consumption occurred. This information helps to identify the time period during which the highest energy demand was observed. Knowing the peak consumption and the corresponding timestamp is important for various purposes, such as identifying periods of high energy demand.

4.3.8 Calculate the median energy consumption

Description: The code `df['kWh'].median()` calculates the median value of energy consumption. The median is a statistical measure that helps understand the central tendency of a dataset. It represents the value below and above which 50 percent of the energy consumption values lie.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `median()` is a function that calculates the middle value in a sorted list of values. In this case, it finds the median energy consumption value.



Calculate the median energy consumption

```
In [27]: median_consumption = df['kWh'].median()
         print("\nMedian Consumption:", median_consumption)
```

```
Median Consumption: 1005455.56
```

Figure 32

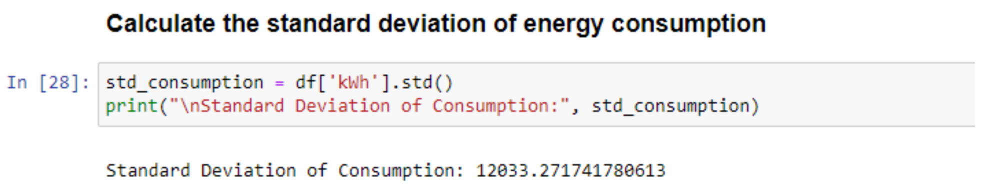
Note: In the given context, the median consumption of 1005455.56 means that approximately half of the consumption values in the dataset are below this value, and the other half are above it. It can be useful in comparing consumption patterns or assessing the typical level of energy usage.

4.3.9 Calculate the standard deviation of energy consumption

Description: The code `df['kWh'].std()` calculates the standard deviation of energy consumption. The standard deviation helps assess how spread out the energy consumption values are. A higher standard deviation indicates a greater variability, while a lower standard deviation suggests that the values are closer to the mean.

- `df['kWh']` refers to the column named `'kWh'` in the DataFrame. It represents the energy consumption values.
- `std()` is a function that calculates the standard deviation of a set of values. The standard deviation measures the dispersion or variability of the values from the mean.

The result of `df['kWh'].std()` is stored in the variable `std_consumption`, representing the standard deviation of energy consumption.



Calculate the standard deviation of energy consumption

```
In [28]: std_consumption = df['kWh'].std()
         print("\nStandard Deviation of Consumption:", std_consumption)

         Standard Deviation of Consumption: 12033.271741780613
```

Figure 33

Note: A standard deviation of 12033.271741780613 means that, on average, the consumption values in the dataset deviate from the mean consumption by approximately 12033.27 units.
The standard deviation is useful for understanding the dispersion of consumption data and assessing the stability or fluctuation in energy usage. It provides insights into the degree of variability and can help identify unusual or abnormal consumption patterns.

4.3.10 Calculate the coefficient of variation of energy consumption

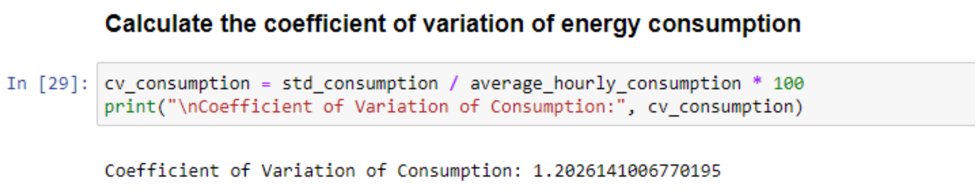
Description: `cv_consumption = std_consumption average_hourly_consumption` $\times 100$
calculates the coefficient of variation (CV) of energy consumption.

The coefficient of variation allows us to understand the relative variability of energy consumption data. A higher CV indicates a higher relative variability, suggesting that the energy consumption values are more spread out compared to the mean. Conversely, a lower CV implies a lower relative variability, indicating that the consumption values are closer to the mean.

- `std_consumption` represents the standard deviation of energy consumption, which measures how much the consumption values deviate from the average.
- `average_hourly_consumption` represents the average (mean) of energy consumption, which provides a central value for the data.

The code calculates the coefficient of variation by dividing the standard deviation (`std_consumption`) by the average consumption (`average_hourly_consumption`), and then multiplying the result by 100.

The calculated coefficient of variation is stored in the variable `cv_consumption`.



Calculate the coefficient of variation of energy consumption

```
In [29]: cv_consumption = std_consumption / average_hourly_consumption * 100
         print("\nCoefficient of Variation of Consumption:", cv_consumption)
```

```

Coefficient of Variation of Consumption: 1.2026141006770195
```

Figure 34

Note:A coefficient of variation of 1.2026 suggests that the consumption values in your dataset have a moderate degree of variability, with the standard deviation being approximately 1.2026 times the mean. This means that the consumption values are relatively close to the average consumption, but still exhibit some variability around it.

4.4 Data Visualization

4.4.1 Plot the daily sum of energy consumption

Description: The code snippet `fig2 = px.line(daily_sum, x=daily_sum.index, y=daily_sum.values)` creates a line plot to visualize the daily sum of energy consumption.

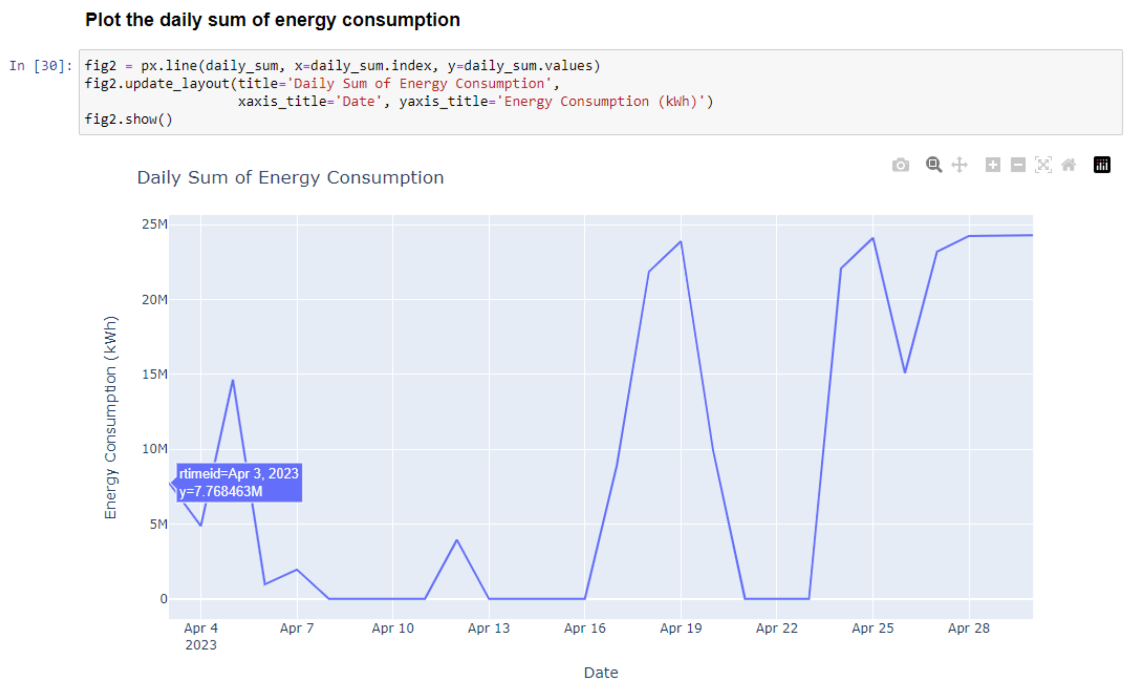
The variable `daily_sum` contains the daily sum of energy consumption, which was calculated earlier using the `resample` function.

The code utilizes the `px.line()` function from the Plotly Express library to create the line plot. Within the function, the data is provided as `daily_sum`, and the x and y values are specified as `daily_sum.index` and `daily_sum.values`, respectively. This sets the dates from the index of the `daily_sum` DataFrame as the x-axis values and the corresponding daily sum values as the y-axis values.

The resulting line plot is assigned to the variable `fig2`.

To enhance the plot's visual presentation, the code uses the `update_layout()` function to set the title as "Daily Sum of Energy Consumption" and label the x-axis as "Date" and the y-axis as "Energy Consumption (kWh)".

Finally, `fig2.show()` is called to display the plot.



Plot the daily sum of energy consumption

```
In [30]: fig2 = px.line(daily_sum, x=daily_sum.index, y=daily_sum.values)
         fig2.update_layout(title='Daily Sum of Energy Consumption',
                           xaxis_title='Date', yaxis_title='Energy Consumption (kWh)')
         fig2.show()
```

Figure 35

Note: From the above graph, my observations are:

- During April 3rd, the total energy consumed is 7.768463M, where 'M' represents the Meg a metric (10^6), indicating that 7.768463 million units of energy have been consumed.
- Therefore, we can see that the graphs oscillate up and down, which means that energy consumption is not constant and varies.

4.4.2 Plot the hourly average of energy consumption

Description: The code `fig3` creates a line plot to visualize the hourly average of energy consumption.

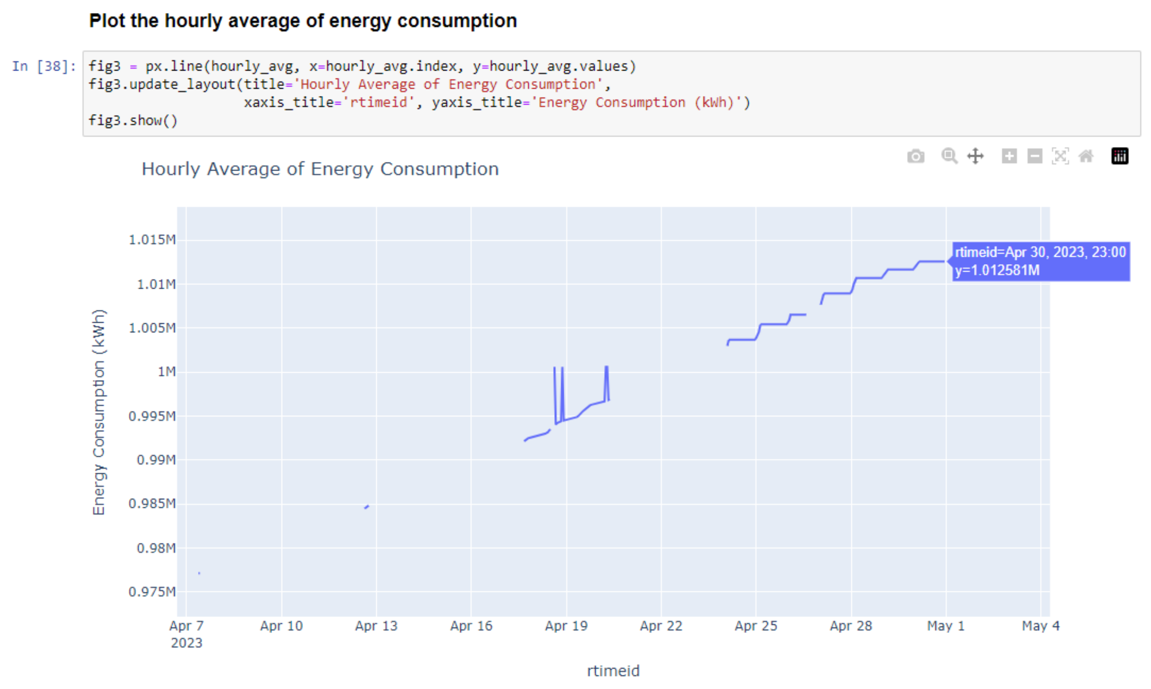
`px.line()` is a function from the Plotly Express library used to create a line plot. Inside `px.line()`, we provide the data and specify the x and y values for the plot.

The resulting line plot is assigned to the variable `fig3`. `update_layout()` is used to customize the plot's layout with the following settings:

- Title: `'Hourly Average of Energy Consumption'`
- X-axis label: `'rtimeid'` (timestamp)
- Y-axis label: `'Energy Consumption (kWh)'`

Finally, `fig3.show()` is called to display the plot.

The line plot visually represents the hourly average of energy consumption over time. The x-axis displays the timestamps, while the y-axis represents the corresponding average energy consumption values. By analyzing the line plot, we can observe the fluctuations and patterns in the hourly energy consumption data and gain insights into the average energy usage throughout the day.



Plot the hourly average of energy consumption

```
In [38]: fig3 = px.line(hourly_avg, x=hourly_avg.index, y=hourly_avg.values)
         fig3.update_layout(title='Hourly Average of Energy Consumption',
                            xaxis_title='rtimeid', yaxis_title='Energy Consumption (kWh)')
         fig3.show()
```

Figure 36

Note: From the above graph, my observations are:

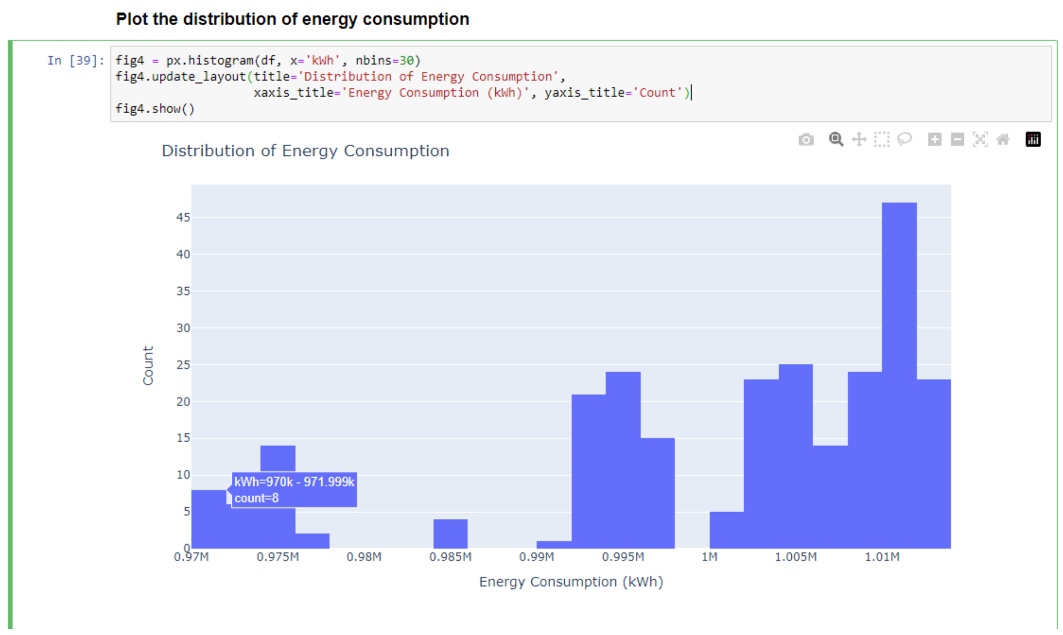
The hourly average consumption ranged from approximately 970,652 kWh to 1,012,580 kWh.The peak consumption of 1,012,580 kWh occurred on April 30 at 23:00. The unit "kilowatt-hour" (kWh) signifies the energy consumed when a device with a power consumption of 1 kilowatt operates for one hour. In this case, the recorded energy consumption of 1,012,580 kWh indicates a substantial usage of electricity during the specified hour.

- **4.4.3 Plot the distribution of energy consumption**

Description: The code `fig4 = px.histogram(df, x='kWh', nbins=30)` creates a histogram to visualize the distribution of energy consumption.

- The DataFrame `df` contains the data on energy consumption.
- To create the histogram, we use the `px.histogram()` function from the Plotly Express library.

- Inside `px.histogram()`, we specify the data to plot as `df` and set `x='kWh'` to indicate the 'kWh' column as the values to be plotted.
- Additionally, we set `nbins=30` to divide the data into 30 bins, determining the number of bars in the histogram.
- The resulting histogram plot is assigned to the variable `fig4`, which can be further customized if desired.
  - By using the `update_layout()` function, we modify the plot's layout, setting the title as 'Distribution of Energy Consumption'. The x-axis is labeled as 'Energy Consumption (kWh)', representing the measured quantity, and the y-axis is labeled as 'Count', indicating the frequency of occurrence.
  - Finally, `fig4.show()` is called to display the histogram plot.



Plot the distribution of energy consumption

```
In [39]: fig4 = px.histogram(df, x='kWh', nbins=30)
         fig4.update_layout(title='Distribution of Energy Consumption',
                            xaxis_title='Energy Consumption (kWh)', yaxis_title='Count')
         fig4.show()
```

Figure 37

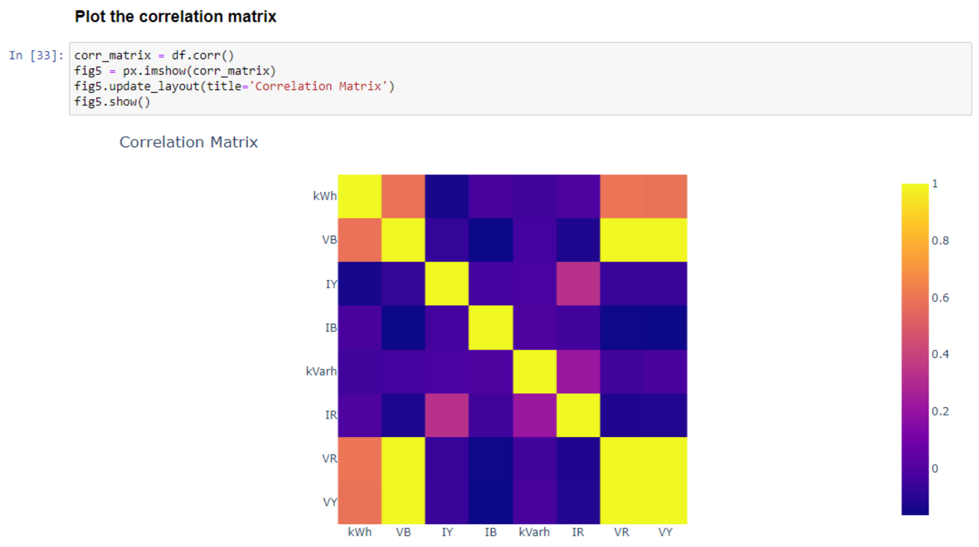
Note: From the above graph,My observations are:

- The histogram plot shows the distribution of energy consumption in the range from 970,000 kWh to 971,990 kWh.
- The x-axis represents the range of energy consumption values, ranging from 970,000 kWh to 971,990 kWh.The y-axis represents the count or frequency of occurrences for each energy consumption range.
- In this particular histogram plot, there are 8 bars or bins representing different ranges of energy consumption values.
- Each bar in the histogram indicates the number of occurrences or data points falling within that specific range of energy consumption.
- So based on the histogram, we can observe the distribution of energy consumption values within the given range. We can see how many occurrences fall within each range and identify any patterns of energy consumption.

4.4.4 Plot the correlation matrix

Description: The code `corr_matrix = df.corr()` calculates the correlation matrix for the given DataFrame `df`, which contains the correlation coefficients between different variables.

- The `corr()` function calculates the pairwise correlation between all numerical columns in the DataFrame `df`.
- The resulting correlation matrix, stored in the variable `corr matrix`, is a square matrix where each cell represents the correlation coefficient between two variables.
- The `px.imshow()` function from the Plotly Express library is used to create an image plot, which visualizes the correlation matrix as a color-coded grid.
- The `update layout()` function is used to set the title of the plot as 'Correlation Matrix'.
- Finally, `fig5.show()` is called to display the correlation matrix plot.



Plot the correlation matrix

```
In [33]: corr_matrix = df.corr()
         fig5 = px.imshow(corr_matrix)
         fig5.update_layout(title='Correlation Matrix')
         fig5.show()
```

Figure 38

Note: From the above graph, my observations are:

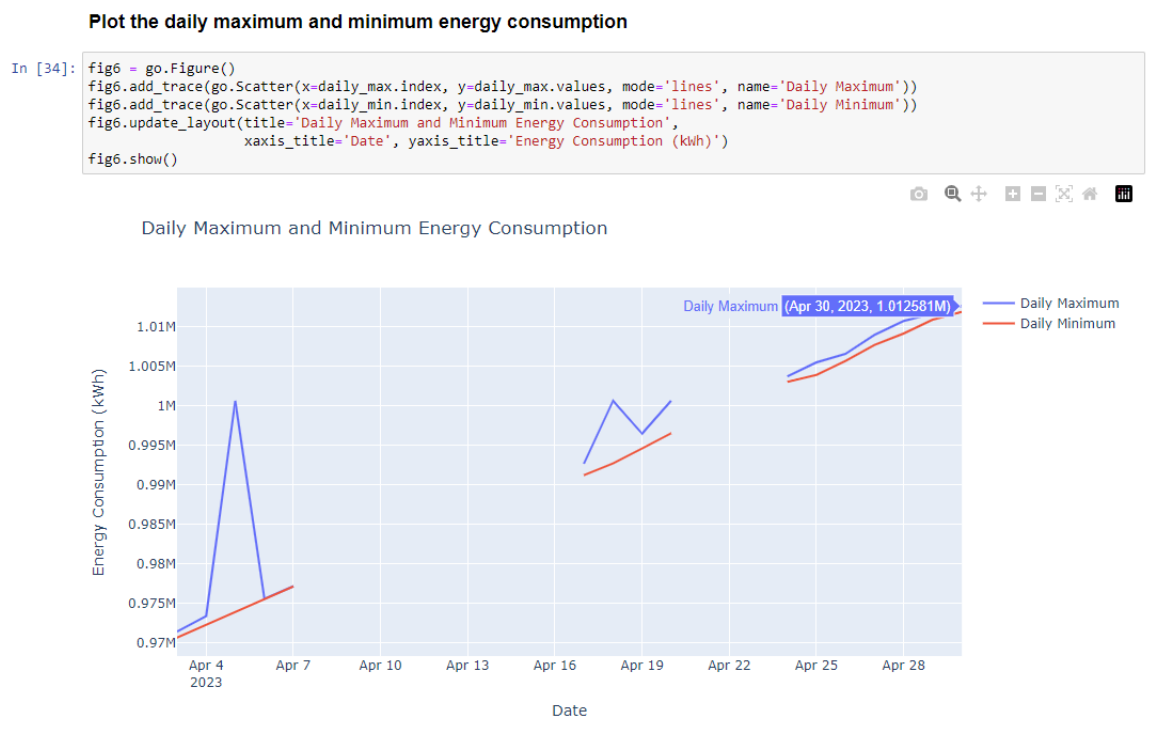
- The correlation values range from -1 to 1. A value of -1 means variables move in opposite directions, 1 means they move together, and 0 means there is no relationship.
- The closer the correlation value is to -1 or 1, the stronger the relationship. Values closer to 0 indicate a weak relationship.

4.4.5 Plot Daily Maximum and Minimum energy consumption

Description: The code `fig6 = go.Figure()` initializes a new figure to create a line plot visualizing the daily maximum and minimum energy consumption.

- The `go.Figure()` function from the Plotly library is used to create a blank figure object, which serves as the container for our plot.

- The `add_trace()` function is used to add individual line traces to the figure. In this case, we add two traces: one for the daily maximum energy consumption and another for the daily minimum energy consumption.
- For each trace, we provide the x and y values for the plot. The x-values are the dates from the index of the `daily_max` and `daily_min` DataFrames, representing the dates on the x-axis. The y-values are the corresponding daily maximum and minimum energy consumption values.
- The `mode='lines'` parameter specifies that the data points should be connected with lines to create a line plot.
- The `name` parameter is used to provide labels for the line traces, which will be displayed in the legend.
- The `update_layout()` function is used to set the title of the plot as 'Daily Maximum and Minimum Energy Consumption', and label the x-axis as 'Date' and the y-axis as 'Energy Consumption (kWh)'.
- Finally, `fig6.show()` is called to display the line plot.



Plot the daily maximum and minimum energy consumption

```
In [34]: fig6 = go.Figure()
         fig6.add_trace(go.Scatter(x=daily_max.index, y=daily_max.values, mode='lines', name='Daily Maximum'))
         fig6.add_trace(go.Scatter(x=daily_min.index, y=daily_min.values, mode='lines', name='Daily Minimum'))
         fig6.update_layout(title='Daily Maximum and Minimum Energy Consumption',
                            xaxis_title='Date', yaxis_title='Energy Consumption (kWh)')
         fig6.show()
```

Figure 39: Maximum Value

Plot the daily maximum and minimum energy consumption

```
In [34]: fig6 = go.Figure()
         fig6.add_trace(go.Scatter(x=daily_max.index, y=daily_max.values, mode='lines', name='Daily Maximum'))
         fig6.add_trace(go.Scatter(x=daily_min.index, y=daily_min.values, mode='lines', name='Daily Minimum'))
         fig6.update_layout(title='Daily Maximum and Minimum Energy Consumption',
                            xaxis_title='Date', yaxis_title='Energy Consumption (kWh)')
         fig6.show()
```

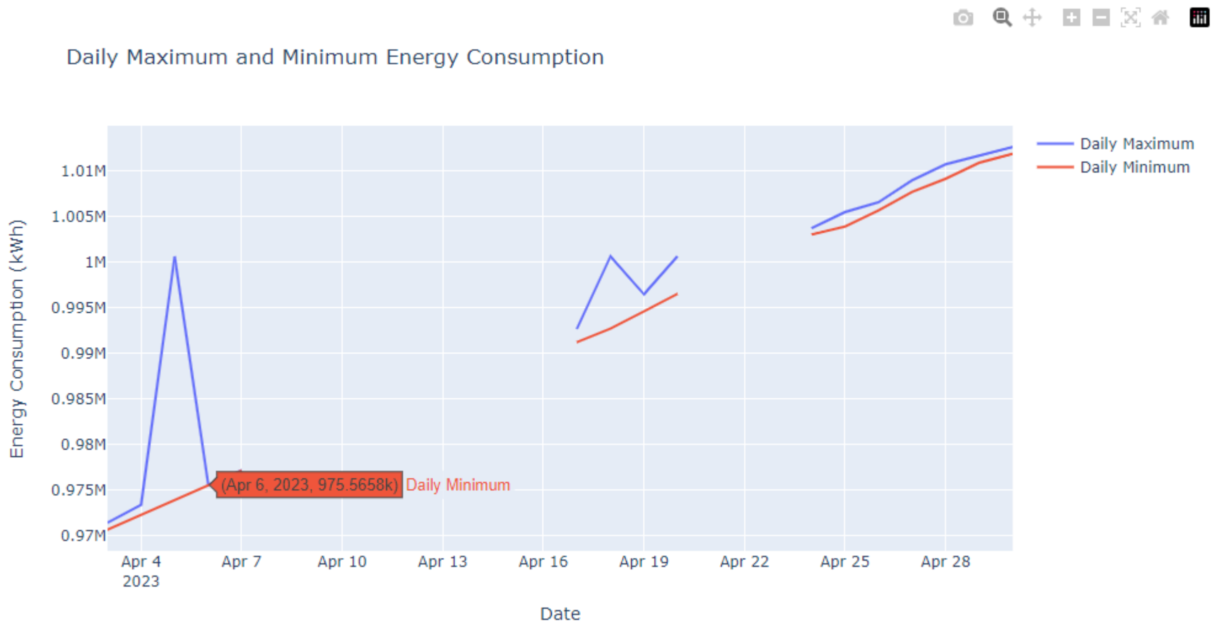


Figure 40: Minimum Value

Note:

- The daily energy consumption varied throughout the month, with some days having zero consumption.
- The highest daily consumption was observed on April 26, reaching 24,120,185 kWh.
- The lowest daily consumption was on April 6, with 975,566.75 kWh.

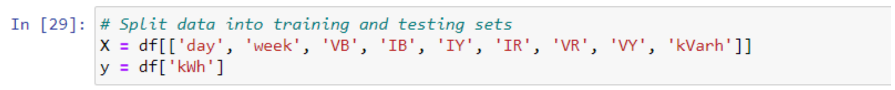
5 Model Evaluation

5.1 Split the dataset into features (X) and target variable (y)

Desciption: The X_train variable contains the features that will be used to train the model. In this case, the features are the day of the week, the week of the year, the voltage between buses VB, the current in bus IB, the current in bus IY, the reactive current in bus IR, the voltage across bus VR, the current across bus VY, and the reactive power kVarh.

The y_train variable contains the target variable, which is the amount of energy consumed in kWh.

The model will be trained to predict the amount of energy consumed based on the features in X_train. Once the model is trained, it will be evaluated on the y_test variable to see how well it performs.



```
In [29]: # Split data into training and testing sets
         X = df[['day', 'week', 'VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh']]
         y = df['kWh']
```

Figure 41

5.2 Split the data into training and test sets

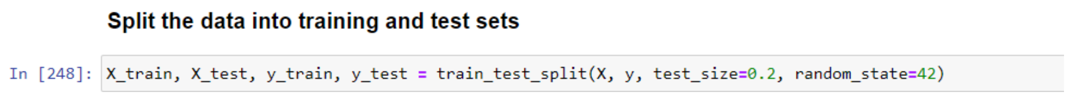
The code `train_test_split()` is used to split the data into training and testing sets for machine learning.

The variables X and y represent the features (inputs) and the target variable (output) respectively. In this case, X contains the 'day' and 'week' features, and y contains the energy consumption values (kWh).

The `train_test_split()` function is called with the arguments X, y, `test_size`=0.2, and `random_state`=42. Here's what each argument means:

X and y: The features and target variable to be split. `test_size`=0.2: It specifies that 20`random_state`=42: It sets a random seed to ensure that the split is reproducible, meaning the same split will be obtained each time the code is run. The function splits the data into two sets: the training set and the testing set.

The training set, denoted by X_train and y_train, contains a subset of the data (80The testing set, denoted by X_test and y_test, contains the remaining portion of the data (20The training and testing sets are assigned to their respective variables (X_train, X_test, y_train, y_test).



Split the data into training and test sets

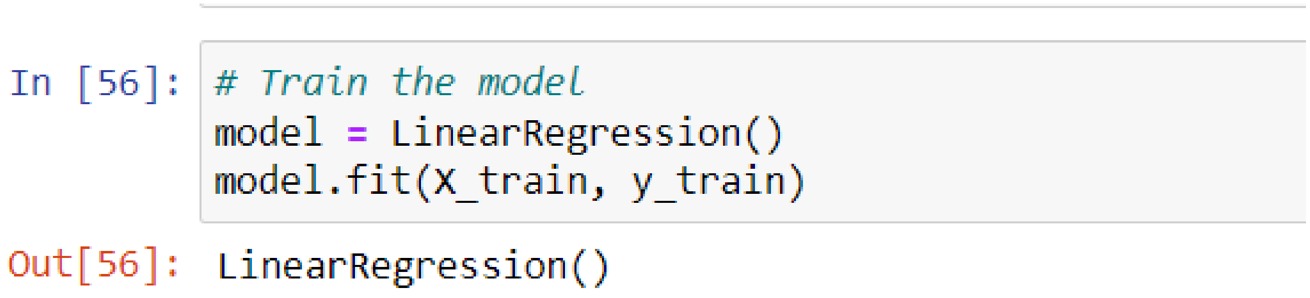
```
In [248]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 42

5.3 Train The Model

- The first line, model = LinearRegression(), creates a new instance of the LinearRegression class. This class represents a linear regression model, which is a statistical model that predicts a continuous value based on a set of independent variables.
- The second line, model.fit(X_train, y_train), trains the model on the training data. The X_train variable contains the independent variables, and the y_train variable contains the dependent variable. The fit() method uses the training data to learn the coefficients of the linear regression model.

The following code creates a model variable and trains it on the X_train and y_train data:



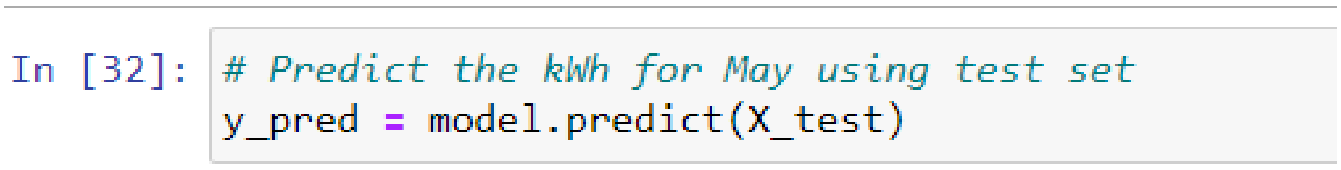
```
In [56]: # Train the model
         model = LinearRegression()
         model.fit(X_train, y_train)

Out[56]: LinearRegression()
```

Figure 43

5.4 Predict the kWh for May using test set

Description: y_pred, represents the predicted values of energy consumption (kWh) for the test set.

- The trained linear regression model ($model$) is used to predict the energy consumption for the test set.
- The $predict$ function is called on the model with X_test as the input. X_test contains the features (day, week, VB, IB, IY, IR, VR, VY, kVarh) of the test set.
- The model uses the provided features to make predictions on the energy consumption.
- The predicted values are stored in the y_pred variable.

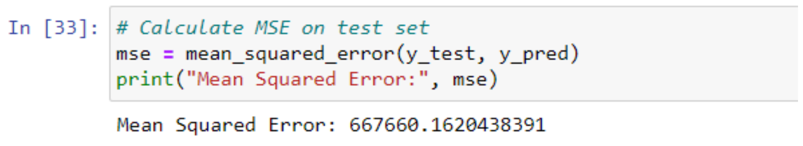


```
In [32]: # Predict the kWh for May using test set
         y_pred = model.predict(X_test)
```

Figure 44

5.5 Calculate Mean Squared Error on test set

- The variable mse stands for Mean Squared Error. It is a metric that measures the average squared difference between the actual values (y_test) and the predicted values (y_pred).
- The mean_squared_error() function is called with two arguments: y_test and y_pred. It calculates the squared differences between the corresponding elements of y_test and y_pred, and then computes the average of these squared differences.



```
In [33]: # Calculate MSE on test set
         mse = mean_squared_error(y_test, y_pred)
         print("Mean Squared Error:", mse)

         Mean Squared Error: 667660.1620438391
```

Figure 45

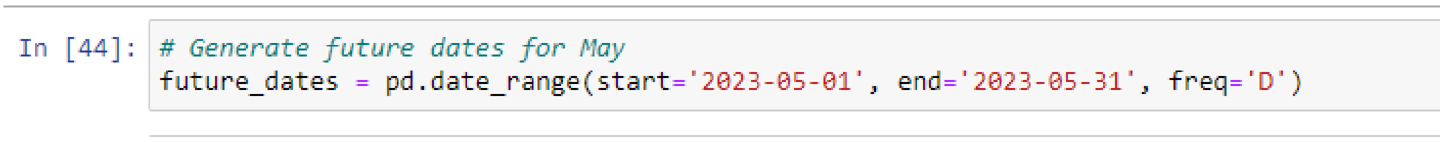
Note: The mean squared error (MSE) measures the average squared difference between the actual energy consumption values and the predicted values. In this case, the MSE of 667660.1620438 suggests that, on average, the predicted energy consumption values deviate from the actual values by approximately 667660.1620438 squared units.

5.6 Generate Future Dates For May

Description: The `pd.date_range()` function takes three arguments:

- `start`: The starting date of the range.
- end: The ending date of the range.
- `freq`: The frequency of the dates in the range. In this case, the frequency is D, which means that the dates will be generated one day apart.

The code then assigns the list of dates to the variable `future_dates`. This variable can then be used to access the dates in the future.



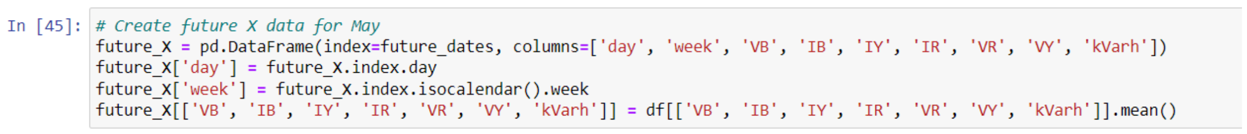
```
In [44]: # Generate future dates for May
         future_dates = pd.date_range(start='2023-05-01', end='2023-05-31', freq='D')
```

Figure 46

5.7 Create Future X Data For May

- future_X = pd.DataFrame(index=future_dates, columns=['day', 'week', 'VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh']) This line of code creates a new DataFrame called future_X. The index argument specifies the dates for the DataFrame, and the columns argument specifies the names of the columns.

- Populate the day column with the corresponding values from the future_dates list. future_X['day'] = future_X.index.day This line of code populates the day column in future_X with the corresponding values from the future_dates list.
- Populate the week column with the corresponding values from the future_dates list. future_X['week'] = future_X.index.isocalendar().week This line of code populates the week column in future_X with the corresponding values from the future_dates list.
- Populate the VB, IB, IY, IR, VR, VY, and kVarh columns with the mean values from the df DataFrame. future_X[['VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh']] = df[['VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh']].mean() This line of code populates the VB, IB, IY, IR, VR, VY, and kVarh columns in future_X with the mean values from the df DataFrame.



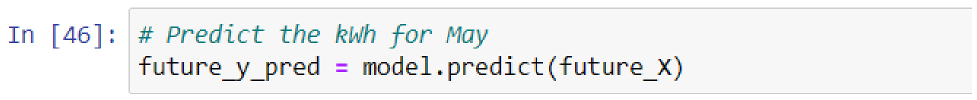
```
In [45]: # Create future X data for May
         future_X = pd.DataFrame(index=future_dates, columns=['day', 'week', 'VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh'])
         future_X['day'] = future_X.index.day
         future_X['week'] = future_X.index.isocalendar().week
         future_X[['VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh']] = df[['VB', 'IB', 'IY', 'IR', 'VR', 'VY', 'kVarh']].mean()
```

Figure 47

5.8 Predict the kWh for May

Description: The code `future_y_pred = model.predict(future_X)` uses a trained machine learning model (`model`) to predict the energy consumption (kWh) for the month of May.

- The variable `future_X` contains the features or input data for the prediction. In this case, it is a DataFrame with the index set to a range of dates in May and columns representing the corresponding day and week of each date.
- The machine learning model (`model`), which has been trained on historical data, is used to make predictions based on the provided features.
- The `predict()` function is called on the `model` object, passing `future_X` as the input. This function applies the trained model to the input data and generates predictions for each sample in `future_X`.
- The resulting predictions are stored in the variable `future_y_pred`, which represents the predicted energy consumption (kWh) for each date in May



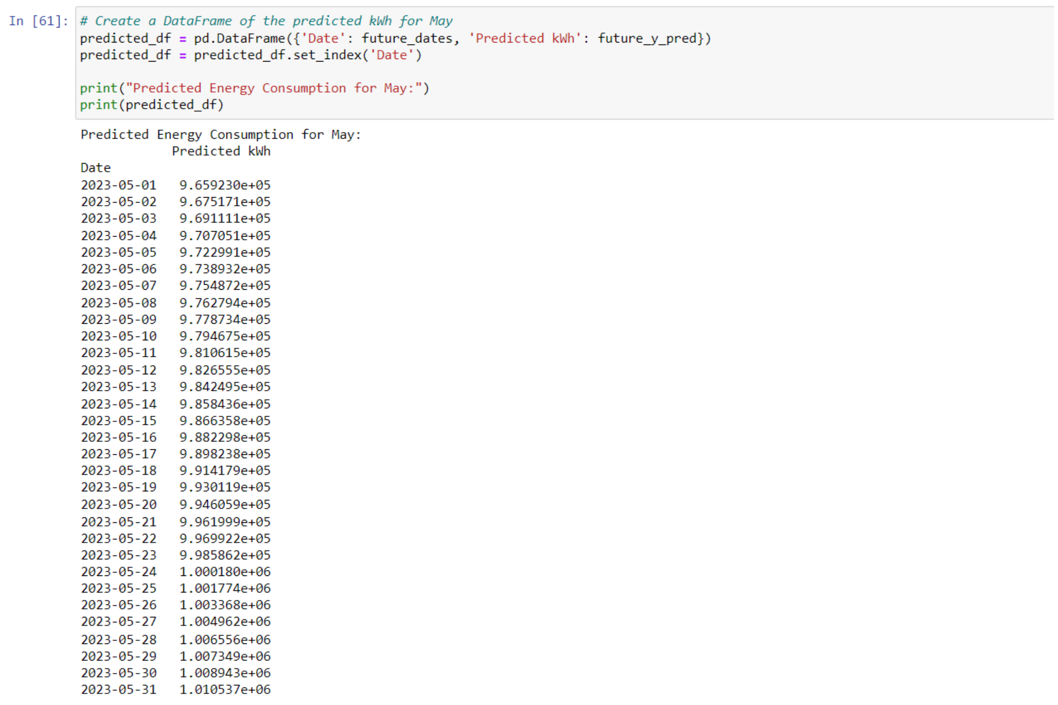
```
In [46]: # Predict the kWh for May
         future_y_pred = model.predict(future_X)
```

Figure 48

5.9 Create a DataFrame Of The Predicted kWh For May

- The code creates a DataFrame called `predicted_df` to store the predicted energy consumption for the month of May.
- It starts by using the `pd.DataFrame()` function to create the DataFrame. The DataFrame has two columns: 'Date' and 'Predicted kWh'.

- The ’Date’ column is filled with the dates in May, and the ’Predicted kWh’ column is filled with the corresponding predicted energy consumption values.
- Next, the code uses the `set_index()` function to set the ’Date’ column as the index of the DataFrame. This step is performed to make it easier to access and manipulate the data based on the dates.
- Finally, the code prints a message indicating that it is displaying the predicted energy consumption for May (`predicted_df`)



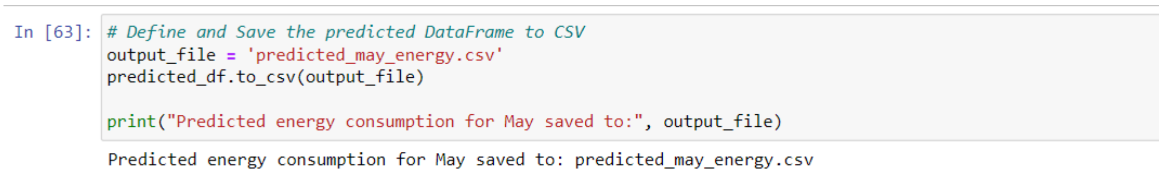
```
In [61]: # Create a DataFrame of the predicted kWh for May
         predicted_df = pd.DataFrame({'Date': future_dates, 'Predicted kWh': future_y_pred})
         predicted_df = predicted_df.set_index('Date')

         print("Predicted Energy Consumption for May:")
         print(predicted_df)
```

```
Predicted Energy Consumption for May:
            Predicted kWh
Date
2023-05-01   9.659230e+05
2023-05-02   9.675171e+05
2023-05-03   9.691111e+05
2023-05-04   9.707051e+05
2023-05-05   9.722991e+05
2023-05-06   9.738932e+05
2023-05-07   9.754872e+05
2023-05-08   9.762794e+05
2023-05-09   9.778734e+05
2023-05-10   9.794675e+05
2023-05-11   9.810615e+05
2023-05-12   9.826555e+05
2023-05-13   9.842495e+05
2023-05-14   9.858436e+05
2023-05-15   9.866358e+05
2023-05-16   9.882298e+05
2023-05-17   9.898238e+05
2023-05-18   9.914179e+05
2023-05-19   9.930119e+05
2023-05-20   9.946059e+05
2023-05-21   9.961999e+05
2023-05-22   9.969922e+05
2023-05-23   9.985862e+05
2023-05-24   1.000180e+06
2023-05-25   1.001774e+06
2023-05-26   1.003368e+06
2023-05-27   1.004962e+06
2023-05-28   1.006556e+06
2023-05-29   1.007349e+06
2023-05-30   1.008943e+06
2023-05-31   1.010537e+06
```

Figure 49

5.10 Define and Save the predicted DataFrame to CSV

- First, the variable `output_file` is set to the file name ’predicted_may_energy.csv’. This will be the name of the file where the predicted data will be saved.
- Then, the DataFrame `predicted_df`, which contains the predicted energy consumption values for each date in May, is saved to the CSV file using the `to_csv()` function. This function converts the DataFrame into a CSV (Comma-Separated Values) file format and writes it to the specified file name.
- Finally, the code prints a message indicating that the predicted energy consumption for May has been saved to the file specified by `output_file`.

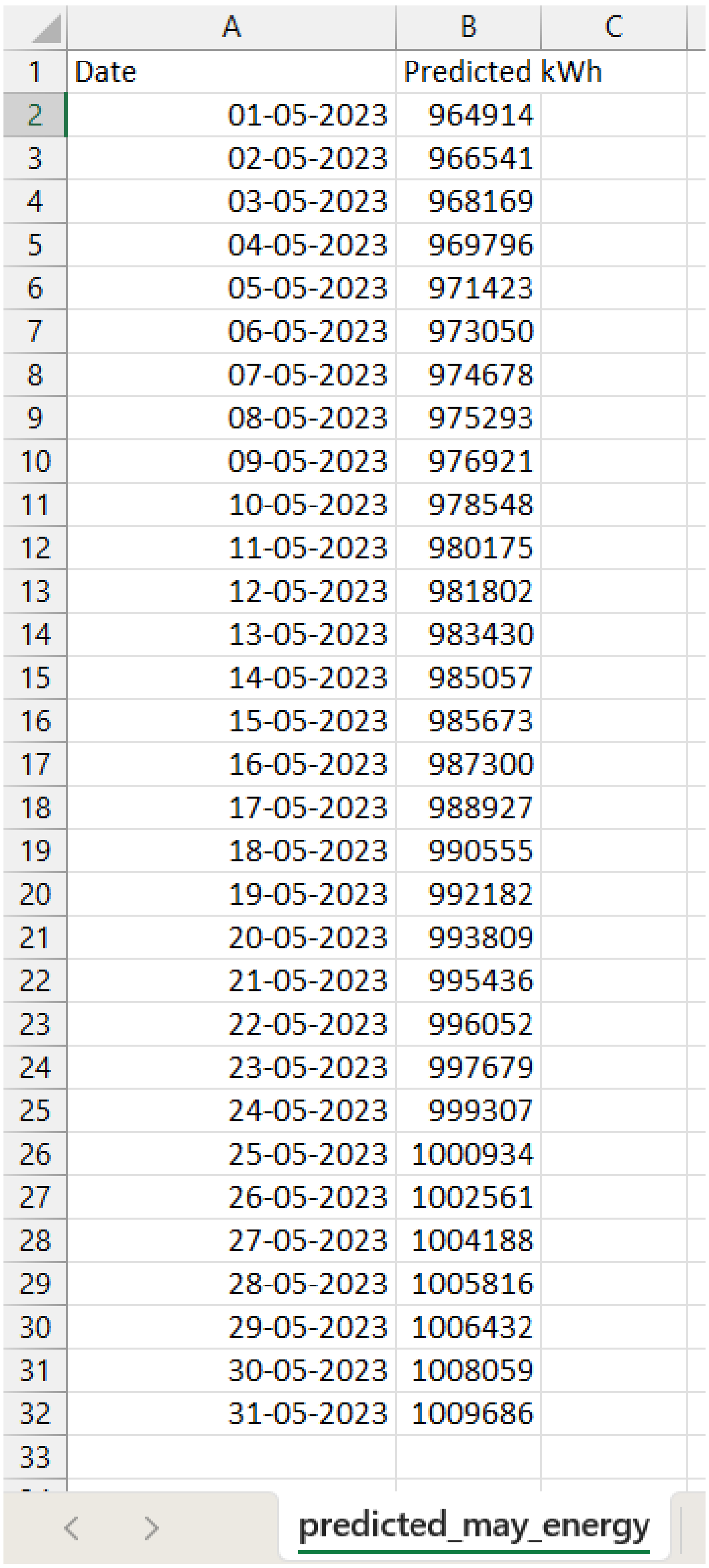


```
In [63]: # Define and Save the predicted DataFrame to CSV
         output_file = 'predicted_may_energy.csv'
         predicted_df.to_csv(output_file)

         print("Predicted energy consumption for May saved to:", output_file)
```

```
Predicted energy consumption for May saved to: predicted_may_energy.csv
```

Figure 50

6 Output



	A	B	C
1	Date	Predicted kWh	
2	01-05-2023	964914	
3	02-05-2023	966541	
4	03-05-2023	968169	
5	04-05-2023	969796	
6	05-05-2023	971423	
7	06-05-2023	973050	
8	07-05-2023	974678	
9	08-05-2023	975293	
10	09-05-2023	976921	
11	10-05-2023	978548	
12	11-05-2023	980175	
13	12-05-2023	981802	
14	13-05-2023	983430	
15	14-05-2023	985057	
16	15-05-2023	985673	
17	16-05-2023	987300	
18	17-05-2023	988927	
19	18-05-2023	990555	
20	19-05-2023	992182	
21	20-05-2023	993809	
22	21-05-2023	995436	
23	22-05-2023	996052	
24	23-05-2023	997679	
25	24-05-2023	999307	
26	25-05-2023	1000934	
27	26-05-2023	1002561	
28	27-05-2023	1004188	
29	28-05-2023	1005816	
30	29-05-2023	1006432	
31	30-05-2023	1008059	
32	31-05-2023	1009686	
33			

Figure 51

7 Limitations

- The analysis is based on a single month of data, which may not be representative of long-term energy consumption patterns. The findings may not accurately capture seasonal variations or long-term trends.

- The analysis utilizes linear regression, assuming a linear relationship between the input variables and energy consumption. This oversimplification may not capture the complex dynamics and potential non-linearities present in energy consumption data.

- The dataset used for the analysis may lack important variables that can influence energy consumption, such as weather conditions, economic factors, or specific events. The absence of these variables may lead to incomplete or biased conclusions.

- Although efforts have been made to address outliers, there may still be extreme values or anomalies present in the data that could impact the accuracy and reliability of the analysis.

- Linear regression assumes that the statistical properties of the data remain constant over time. However, energy consumption data often exhibits non-stationary behavior, such as trends or seasonality. Ignoring these patterns may result in inaccurate predictions.

- Missing values in the dataset have been imputed using mean values. Imputation introduces potential biases and may not fully reflect the true values of the missing data points.

- The analysis and conclusions are specific to the dataset and time period under consideration. Extrapolating the findings to other time periods or different contexts should be done with caution.

8 Conclusion

- In conclusion, the energy consumption analysis using linear regression has provided insights into the relationship between various features (day, week, VB, IB, IY, IR, VR, VY, kVarh) and energy consumption (kWh). The model was trained on a dataset from April and used to predict energy consumption for the month of May.

- The linear regression model, despite its simplicity, allows us to estimate energy consumption based on the provided features. However, it's important to note that the model's performance is indicated by the obtained mean squared error (MSE) value of 667660.1620438. The MSE measures the average squared difference between the predicted and actual energy consumption values.

- In addition to, it is important to note that the analysis was conducted using data from a single month, specifically April. With only one month of data available, the dataset might not capture long-term patterns, seasonal trends, or other time-dependent dynamics adequately.

- Given the limited data availability, linear regression was chosen as a straightforward approach to estimate energy consumption based on the provided features. Linear regression assumes a linear relationship between the features and the target variable. However, it may not capture more complex temporal patterns or seasonal variations that could be present in energy consumption data.

- For a comprehensive analysis and accurate predictions of energy consumption, a larger dataset spanning a longer period would be beneficial. With more data, time series analysis techniques become more applicable. Time series analysis takes into account the sequential nature of the data, identifies patterns over time, and can provide more accurate forecasts by considering seasonality, trends, and other temporal factors.

- Therefore, with only one month of data available, the choice between linear regression and time series analysis may not significantly impact the results. However, if more data becomes available in the future, it is advisable to explore time series analysis methods to capture the temporal patterns and improve the accuracy of energy consumption predictions.

9 References

Visit the below links :

- https://shrturl.app/GCH-7TxW
- https://shrturl.app/cfv_3vxD
- https://shrturl.app/qjQepWV2
- https://shrturl.app/YsxOlmN_
- https://shrturl.app/RALB26iN
- https://shrturl.app/JO9MemZK
- http://surl.li/hwwvk
- http://surl.li/hxtct