Predictive Analytics for Energy Planning:

A Data-Driven Study on India's Energy Requirements and Availability

MACHINE LEARNING EDA REPORT

Submitted to

DR. PIYUSH CHAUHAN
Associate Professor
Department of Computer Science & Engineering
Symbiosis Institute of Technology, Nagpur Campus

Submitted by

Tanay Yaduka
VII SEM
PRN: 22070521086
Department of Computer Science & Engineering
Symbiosis Institute of Technology, Nagpur Campus

Course Name: *Machine Learning*
Course Code: *T7529*

॥वसुधैव कुटुम्बकम्॥

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Contents

Abstract

Problem Statement: This project addresses the critical challenge of optimizing energy planning and resource allocation in rapidly developing economies, specifically by accurately predicting continuous energy requirements and classifying binary energy availability states based on historical spatial and temporal data.
Methodology: An extensive comparative analysis was performed across ten machine learning models, including classic algorithms (Linear Regression, Naive Bayes), complex ensembles (Random Forest, XGBoost), and deep learning (MLP), to solve both regression and classification tasks. The evaluation compared the performance of a minimal **Base feature set** (spatial/locational) against an extended **Seasonality feature set** (including temporal metrics) to determine optimal feature engineering.
Key Findings: The core finding reveals a highly deterministic relationship between the base features and the target energy metrics, resulting in near-perfect performance ($\mathbf{R^2 = 1.0}$ or $\mathbf{Accuracy = 1.0}$) across Linear Regression, Logistic Regression, Decision Tree, and ensemble models. Critically, the simple **Decision Tree** was identified as the optimal, most interpretable model. The Seasonality features proved entirely redundant and, for models like the SVM and MLP, actively detrimental, leading to a significant degradation in performance.
Significance: The results demonstrate that effective, high-fidelity energy prediction can be achieved using a highly simplified, non-temporal feature set. This finding is crucial for developing robust, low-latency, and easily maintainable predictive models necessary for data-driven policy recommendations and real-time resource optimization.

Keywords

Machine Learning, Energy Prediction, Regression, Classification, Deterministic Data, Feature Importance, Model Evaluation.

1 Introduction

1.1 Background and Context

Energy planning and reliable resource management constitute one of the most significant challenges for high-growth nations. In countries like India, balancing the escalating energy demands from industrial, commercial, and rapidly expanding domestic sectors with sustainable resource availability is paramount. Accurate forecasting of energy consumption and availability metrics is not merely a matter of economic efficiency but a crucial element of national stability and development. Traditional, statistical forecasting methods often fall short in capturing the non-linear complexities and underlying drivers present in large, disparate energy datasets.

1.2 Motivation and Problem Statement

The primary motivation for this study is to leverage the power of advanced machine learning (ML) and predictive analytics to move beyond conventional forecasting limitations. The core **Problem Statement** is: To what extent can various machine learning models accurately predict energy requirements and availability metrics using spatial and temporal features, and which model-feature combination yields the most optimal, robust, and interpretable performance for driving actionable policy insights?

1.3 Objectives of the Project

This project was designed with three clear objectives:

1. To **develop and evaluate** a comprehensive suite of predictive models, covering both regression and classification tasks, for core energy metrics.

2. To conduct a rigorous **comparative analysis** of model performance using a minimal Base feature set versus an extended set including Seasonality features.

3. To identify the **most accurate, simplest, and most interpretable model** suitable for deployment in a real-world decision-support system for energy planners.

1.4 Novelty of the Work

The novelty of this study lies in the surprising and definitive finding regarding feature efficacy. While time-series analysis often emphasizes temporal components, this research conclusively demonstrated that complex **seasonality features are entirely redundant** for achieving maximum predictive accuracy. The energy variability is almost entirely captured by a deterministic relationship with the simpler, spatial/locational Base features. This insight significantly simplifies the model architecture, reducing complexity, training time, and deployment costs, while maintaining perfect predictive performance ($\mathbf{R^2 = 1.0}$ and $\mathbf{Accuracy = 1.0}$) using the highly interpretable Decision Tree model.

2 Research Work

Table 1: Detailed Literature Review on Energy and Efficiency Studies

Authors	Paper Name / Journal	Year	Summary and Key Findings
Tobias Knayer and Natalia Kryvinska	Evaluation of Research on Energy Efficiency in Energy-Intensive Manufacturing Companies (Frontiers in Energy Research)	2022	Analyzes energy efficiency measures in manufacturing companies via a 101-study review. Found research bias toward **large firms**, with limited focus on SMEs. Suggested policymakers optimize impact by focusing on the **interactions between individual energy efficiency measures**.
Anirban Nandy, Poulomi Chaki, O.P. Pandey	A Study on Energy Consumption, Energy Saving, and Effectiveness of Alternate Energy Sources in India's Domestic Sector (Int. J. of Research in Eng. and Tech.)	2016	Examines the relationship between electricity consumption and income, advocating for a shift to **renewable energy** (solar, wind). A **Siliguri case study** demonstrated the financial benefits of adopting solar PV systems and highlights government **subsidy initiatives**.
Ashwani Kumar, Kapil Kumar, Naresh Kaushik, et al.	Renewable Energy in India: Current Status and Future Potentials (Renewable and Sustainable Energy Reviews)	2010	Comprehensive overview of India's renewable energy sources (wind, solar, biomass). Discusses energy challenges and the importance of policy interventions like the **National Electricity Policy (2005)**. Concludes that vast potential requires stronger governmental policies and **financial incentives** for scale-up.
Anirban Nandy, Poulomi Chaki, O.P. Pandey	A Study on Energy Consumption, Energy Saving, and Effectiveness of Alternate Energy Sources in the Domestic Sector of India (Int. J. of Research in Eng. and Tech.)	2016	Re-examines domestic sector consumption and the need for renewables. Reiterates the **Siliguri case study**, emphasizing how household-level solar PV systems reduce expenses, thereby mitigating the **energy demand-supply gap** and promoting economic sustainability.
Ashwani Kumar, Kapil Kumar, Naresh Kaushik, et al.	Renewable Energy in India: Current Status and Future Potentials (Renewable and Sustainable Energy Reviews)	2010	Re-presents the overview of renewable energy status and potential. Highlights the need for strong governmental support and **technological advancements** to increase renewable energy adoption and meet future national energy demands.

3 Project Definition and Data Understanding

3.1 Problem Statement

In the rapidly evolving landscape of energy consumption and production in India, policy-makers and energy providers face challenges in balancing energy requirements with actual availability across different states. A mismatch between required and available energy can lead to power shortages, economic losses, and reduced quality of life. Therefore, having a reliable method to analyze historical energy data and forecast future energy needs is critical.

This project aims to use statistical and machine learning techniques to analyze patterns in energy requirement and availability across Indian states and develop predictive models to support data-driven decision-making.

3.2 Data Sources

The primary dataset used in this project was obtained from the official Government of India data portal:

- **Source:** data.gov.in — Open Government Data (OGD) Platform India
- **Dataset Name:** Energy Requirement and Availability
- **Format:** CSV
- **Coverage:** State-wise annual energy requirement and availability data
- **Access Date:** October 29, 2025

3.3 Data Dictionary

Field Name	Description	Data Type	Example Values
`id`	Unique identifier for each record	Integer	1, 2, 3
`state_code`	Official state code (abbreviated or numeric)	Categorical / Integer	27, 09, 29
`state`	Abbreviated name of the state or UT	Categorical (String)	MH, UP, KA
`name`	Full name of the state or UT	Categorical (String)	Maharashtra, Uttar Pradesh, Karnataka
`energy_requirement`	Total energy required in a given year (in MU)	Numerical (Float)	130450.0, 98500.5
`energy_availability`	Total energy available in a given year (in MU)	Numerical (Float)	128900.0, 96000.0

Table 2: Data Dictionary for Energy Dataset from India Gov Portal

Delimiter: ,

	Region	State	is_union_territory	Month	Quarter	Energy Requirement MU	Energy Availability MU	Energy_Deficit
1	North	Chandigarh	1	Mar	Q1	101.6	101.6	0
2	North	Delhi	1	Mar	Q1	2130.478	2130.279	0.199
3	North	Haryana	0	Mar	Q1	3911.822	3909.16	2.662
4	North	Himachal Pradesh	0	Mar	Q1	1025.63	1023.16	2.47
5	North	UTs of J&K and Ladakh	1	Mar	Q1	1725.61	1717.93	7.68
6	North	Punjab	0	Mar	Q1	4275.01	4259.19	15.82
7	North	Rajasthan	0	Mar	Q1	7541.57	7509.03	32.54
8	North	Uttar Pradesh	0	Mar	Q1	9619.29	9613.89	5.4
9	North	Uttarakhand	0	Mar	Q1	1161.761	1149.01	12.751
10	Central	Chhattisgarh	0	Mar	Q1	3522.615219	3519.613719	3.0015
11	West	Gujarat	0	Mar	Q1	11336.89609	11336.89609	0
12	Central	Madhya Pradesh	0	Mar	Q1	7524.602891	7496.421383	28.18150782
13	West	Maharashtra	0	Mar	Q1	17010.05555	17009.96633	0.08922
14	West	Haveli and Daman & Diu	1	Mar	Q1	791.7707	791.7707	0
15	West	Goa	0	Mar	Q1	405.46282	405.46282	0
16	South	Andhra Pradesh	0	Mar	Q1	6711.89014	6711.89014	0
17	South	Telangana	0	Mar	Q1	8909.39502	8909.39502	0
18	South	Karnataka	0	Mar	Q1	8602.621995	8602.621995	0
19	South	Kerala	0	Mar	Q1	2713.099073	2713.099073	0
20	South	Tamil Nadu	0	Mar	Q1	10799.8363	10799.8363	0
21	South	Puducherry	1	Mar	Q1	263.8251642	263.8251642	0
22	South	Lakshadweep	1	Mar	Q1	6.01565	6.01565	0
23	East	Bihar	0	Mar	Q1	2882.105319	2863.953819	18.1515
24	East	Jharkhand	0	Mar	Q1	1106.484301	1034.630301	71.854
25	East	Odisha	0	Mar	Q1	3441.362624	3441.362624	0
26	East	West Bengal	0	Mar	Q1	4982.247482	4977.483482	4.764
27	Northeast	Sikkim	0	Mar	Q1	50.49075133	50.49075133	0
28	South	Andaman Nicobar	1	Mar	Q1	31.82	31.37	0.45
29	Northeast	Arunachal Pradesh	0	Mar	Q1	86.89744396	83.19240168	3.705042279
30	Northeast	Assam	0	Mar	Q1	843.6235207	843.6235207	0

Figure 1: Preview of the Energy Dataset

4 Data Collection and Integration

4.1 Data Source and Collection

The dataset used in this project was sourced from the Open Government Data (OGD) Platform India (`https://data.gov.in`). It provides annual statistics on energy requirement and availability for various states and union territories of India. The data was obtained in CSV format and is publicly available for academic and analytical use.

4.2 Data Provenance

- **Source Authority:** Ministry of Power, Government of India
- **Dataset Format:** CSV (Comma-Separated Values)
- **Date of Access:** October 29, 2025
- **License:** Open Government License – India

4.3 Integration Methodology

The dataset was integrated into the analysis pipeline using Python. It was loaded into a DataFrame using the Pandas library:

```
import pandas as pd
df = pd.read_csv("energy-requirement-and-availabililty.csv")
```

As the dataset was already well-structured and complete, no merging with additional datasets was necessary. All fields were verified for consistency in naming and data type.

4.4 Initial Validation

The dataset underwent an initial validation process:

- All column headers were correctly named and interpretable.
- No duplicate records or missing values were found at this stage.
- The schema followed the expected structure with unique state-year entries.

5 Data Cleaning and Preparation

5.1 Check Shape, Data Types, and Null Values

To begin the data cleaning process, the structure and integrity of the dataset were examined. The dataset consists of a total of **1534 rows and 6 columns**, providing an initial understanding of its scale.

Next, the data types of each column were checked to ensure consistency with expected formats. Numerical fields such as `energy_requirement` and `energy_availability` were correctly stored as floating-point values. Identifier fields like `id` and `state_code` appeared as integers. Textual fields such as `state` and `name` were stored as object types (strings), and may be converted to categorical types to optimize memory usage and prepare for machine learning tasks.

A null value check was also conducted across all columns. The dataset did not contain any missing values, indicating it is complete and does not require imputation or deletion of records. This makes the data ready for further preprocessing and analysis.

```
# Dataset shape
print("Dataset shape:", df.shape)

# Data types and nulls
print("\nData types and missing values:")
print(df.info())

# Summary of null values
print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
Dataset shape: (1534, 6)

Data types and missing values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350 entries, 0 to 349
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Region                  350 non-null    object
 1   State                   350 non-null    object
 2   is_union_territory      350 non-null    int64
 3   Month                   350 non-null    object
 4   Quarter                 350 non-null    object
 5   Energy Requirement MU   350 non-null    float64
 6   Energy Availability MU  350 non-null    float64
 7   Energy_Deficit          350 non-null    float64
dtypes: float64(3), int64(1), object(4)
memory usage: 22.0+ KB
None

Missing values in each column:
Region                    0
```

```
State                       0
is_union_territory          0
Month                       0
Quarter                     0
Energy Requirement MU       0
Energy Availability MU      0
Energy_Deficit              0
dtype: int64
```

5.2 Normalize column names (strip/lower/replace non-alnum with underscore)

It means cleaning up your dataset's column names to make them consistent and easy to use in code.

- **strip()** → Removes extra spaces at the start or end of the column name.
- **lower()** → Converts all letters to lowercase, e.g., `Customer Name` → `customer name`.
- **replace non-alnum with underscore** → Replaces any character that is not a letter or number (like spaces, `-`, `#`, etc.) with an underscore, e.g., `customer name` → `customer_name`.

Importance:
Normalizing column names is an important step in data cleaning because it:

- Ensures **consistency** across all column names, avoiding confusion or errors.
- Makes it easier to **access columns programmatically** without syntax issues (e.g., spaces or special characters).
- Helps maintain **compatibility** with different tools, libraries, and programming languages.
- Improves **readability and professionalism** of the dataset.

Example: `" Customer Name# "` → `"customer_name"`

```python
df.columns = (df.columns
                .astype(str)
                .str.strip()
                .str.lower()
                .str.replace(r'[^0-9a-z]+', '_', regex=True)
                .str.strip('_')
              )

print("\nColumns after normalization:", list(df.columns))
```

```
Columns after normalization: ['region', 'state', 'is_union_territory', 'month'
    , 'quarter', 'energy_requirement_mu', 'energy_availability_mu', '
    energy_deficit']
```

Explanation:
The code normalizes all column names in the DataFrame `df` by:

- Converting names to strings using `astype(str)`
- Removing extra spaces with `str.strip()`
- Changing all letters to lowercase using `str.lower()`
- Replacing non-alphanumeric characters with underscores using regex
- Removing leading and trailing underscores

Importance:
This process improves readability, avoids syntax errors, and ensures smooth data manipulation in Python.

5.3 Trim Whitespace and Standardize Text Columns:

This step involves cleaning text-based columns in a dataset by removing unnecessary spaces and ensuring consistency in text formatting.

- **Trimming whitespace:** Removes leading and trailing spaces using functions like `str.strip()`, ensuring that text values are clean and uniform.
- **Standardizing text:** Converts all text to a common case (usually lowercase) and may replace inconsistent characters or formats (e.g., converting "Yes", "YES", and "yes" to a single standard form "yes").

Importance:

- Prevents mismatches and duplication caused by inconsistent text entries.
- Ensures accurate grouping, filtering, and merging of data.
- Improves the quality and reliability of data analysis and machine learning models.

Example:
`" Delhi "` → `"delhi"`

```
text_cols = df.select_dtypes(include=['object']).columns.tolist()
for c in text_cols:
    df[c] = df[c].astype(str).str.strip()  # keep as str for parsing step

# Title-case region/state (but keep 'month' / 'quarter' as-is)
if 'region' in df.columns:
    df['region'] = df['region'].str.title()
if 'state' in df.columns:
    df['state'] = df['state'].str.title()
```

Explanation:
The given code is used to clean and standardize text columns in a DataFrame `df`.

1. **Select text columns:**
   `text_cols = df.select_dtypes(include=['object']).columns.tolist()`
   This line identifies all columns in the DataFrame that contain text (`object` type) and stores their names in the list `text_cols`.

2. **Trim whitespace:**
   `df[c] = df[c].astype(str).str.strip()`
   For each text column, it converts values to strings and removes leading or trailing spaces to ensure data consistency.

3. **Standardize capitalization:**
   The next lines check if columns named `'region'` or `'state'` exist. If they do, their text values are converted to title case using `str.title()`, so that:

   $$\texttt{"delhi"} \rightarrow \texttt{"Delhi"}$$

   Columns like `'month'` and `'quarter'` are intentionally left unchanged.

Purpose:
This process ensures that text data is clean, consistent, and properly formatted, which helps avoid errors during data analysis, grouping, or merging.

5.4 Simple outlier detection (IQR) for numeric columns

The Interquartile Range (IQR) method is used to identify outliers in numeric data. It calculates the first quartile ($Q1$) and third quartile ($Q3$), and the IQR as ($Q3 - Q1$). Any value below ($Q1 - 1.5 \times IQR$) or above ($Q3 + 1.5 \times IQR$) is considered an outlier. This method helps detect extreme or unusual values, improving data quality and ensuring that analysis or models are not skewed by anomalies.

```
outlier_report = {}
for col in numeric_cols:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5*iqr
    upper = q3 + 1.5*iqr
    mask = (df[col] < lower) | (df[col] > upper)
    outlier_count = mask.sum()
    outlier_report[col] = {'count': int(outlier_count), 'lower': float(lower),
     'upper': float(upper)}
print("\nOutlier report (IQR method):")
for k,v in outlier_report.items():
    print(f" - {k}: {v['count']} outliers (outside [{v['lower']:.3f}, {v['
   upper']:.3f}])")
```

```
Outlier report (IQR method):
- energy_requirement_mu: 7 outliers (outside [-9580.443, 16447.334])
- energy_availability_mu: 7 outliers (outside [-9614.250, 16459.750])
- energy_deficit: 58 outliers (outside [-9.977, 16.628])
```

Explanation:
Using the Inter-quartile Range (IQR) method, outliers were detected in the numeric columns of the dataset. The lower and upper bounds were calculated as $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$, respectively, and values outside this range were counted. The resulting report is:

- **energy_requirement_mu:** 7 outliers (outside $[-9580.443, 16447.334]$)
- **energy_availability_mu:** 7 outliers (outside $[-9614.250, 16459.750]$)
- **energy_deficit:** 58 outliers (outside $[-9.977, 16.628]$)

This indicates that `energy_deficit` has a relatively larger number of extreme values, which may require further investigation or handling before analysis.

5.5 Final Checks Save Cleaned File

```
print("\nFinal shape:", df.shape)
display(df.head())
```

	region	state	is_union_territory	month	quarter	energy_requirement_mu	energy_availability_mu	energy_deficit
0	North	Chandigarh	True	Mar	Q1	101.600	101.600	0.000
1	North	Delhi	True	Mar	Q1	2130.478	2130.279	0.199
2	North	Haryana	False	Mar	Q1	3911.822	3909.160	2.662
3	North	Himachal Pradesh	False	Mar	Q1	1025.630	1023.160	2.470
4	North	Uts Of J&K And Ladakh	True	Mar	Q1	1725.610	1717.930	7.680

Figure 2: Columns in data-frame

```
cleaned_path = "cleaned_dataset.csv"
df.to_csv(cleaned_path, index=False)
print(f"\nCleaned dataset saved to: {cleaned_path}")
```

```
Cleaned dataset saved to: cleaned_dataset.csv
```

```
print("\nData types summary:")
print(df.dtypes)
```

```
Data types summary:
region                  object
state                   object
is_union_territory        bool
month                   object
quarter                 object
```

```
energy_requirement_mu      float64
energy_availability_mu     float64
energy_deficit             float64
dtype: object
```

6 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical step in the data science process that involves visually and statistically examining the dataset to uncover patterns, detect anomalies, identify relationships between variables, and test hypotheses. The primary objective of EDA is to better understand the structure, distribution, and nature of the data before applying statistical modeling or machine learning techniques.

EDA helps analysts gain insights into the dataset by using descriptive statistics, visualizations, and correlation analysis. This phase plays a key role in decision-making for data preprocessing, feature selection, and model design by revealing hidden trends and potential data issues. It ensures that the subsequent analysis is grounded in a clear and accurate understanding of the data.

6.1 Descriptive Statistics

Descriptive statistics provide a fundamental summary of the dataset and help in understanding the central tendency, spread, and shape of the distribution of numerical variables. In this project, descriptive metrics such as mean, standard deviation, minimum, maximum, and quartiles were calculated for the `energy_requirement_mu`, `energy_availability_mu`, and `energy_deficit` columns:

```
# Load dataset
df = pd.read_csv("cleaned_dataset.csv")
# Data types
print(df.dtypes)

# Missing values
print(df.isnull().sum())

# Summary statistics
print(df.describe(include="all"))
```

This statistical overview is essential during EDA as it highlights patterns, detects inconsistencies, and guides data transformation or modeling decisions.

```
region                    object
state                     object
is_union_territory          bool
month                     object
quarter                   object
energy_requirement_mu    float64
energy_availability_mu   float64
energy_deficit           float64
dtype: object
region                   0
state                    0
is_union_territory       0
month                    0
quarter                  0
energy_requirement_mu    0
energy_availability_mu   0
energy_deficit           0
dtype: int64
       region       state is_union_territory month quarter  \
count     350         350                350   350     350
unique      6          35                  2    10       4
top     North  Chandigarh              False   Mar      Q2
freq       90          10                280    35     105
mean      NaN         NaN                NaN   NaN     NaN
std       NaN         NaN                NaN   NaN     NaN
min       NaN         NaN                NaN   NaN     NaN
25%       NaN         NaN                NaN   NaN     NaN
50%       NaN         NaN                NaN   NaN     NaN
75%       NaN         NaN                NaN   NaN     NaN
max       NaN         NaN                NaN   NaN     NaN

        energy_requirement_mu  energy_availability_mu  energy_deficit
count              350.000000              350.000000      350.000000
unique                    NaN                     NaN             NaN
top                       NaN                     NaN             NaN
freq                      NaN                     NaN             NaN
mean              3795.145090             3785.631188        9.513902
std               4434.033361             4429.376747       21.834652
min                  5.000000                5.000000        0.000000
25%                179.973259              163.500000        0.000000
50%               1596.000000             1576.500000        0.000000
75%               6686.917605             6682.000000        6.651344
max              18056.000000            18051.000000      195.000000
```

Figure 3: Columns in data-frame

The descriptive statistics provide several valuable insights into the energy-related data. The mean values for the three numerical variables — `energy_requirement_mu`, `energy_availability_mu`, and `energy_deficit` — are 3795.15, 3785.63, and 9.51, respectively. The standard deviations indicate considerable variability in `energy_requirement_mu` and `energy_availability_mu` (both around 4430), while `energy_deficit` shows lower dispersion (21.83), suggesting that for most states, the availability closely follows the requirement.

The median values (50% quantile) for `energy_requirement_mu` and `energy_availability_mu` are 1596.00 and 1576.50, respectively, indicating that half of the observations lie below the mean. The median of `energy_deficit` is 0.0, suggesting that most states experience little to no deficit.

Furthermore, the range between the minimum and maximum values highlights the presence of both surplus and deficit conditions: `energy_requirement_mu` and `energy_availability_mu` span from 5.0 to over 18000, while `energy_deficit` ranges from 0.0 to 195.0. The relatively small interquartile range (IQR) of `energy_deficit` (0.0 – 6.65) indicates that extreme deficits are rare and most values are clustered near zero, reflecting a generally balanced energy supply-demand scenario.

These observations provide a strong foundation for deeper visual and statistical analysis in subsequent steps.

Visualization

Visualizations are powerful tools in Exploratory Data Analysis (EDA) as they allow us to interpret data patterns, relationships, and distributions quickly and intuitively. Graphical representations such as histograms, box plots, scatter plots, and heatmaps make it easier to detect outliers, skewness, clusters, and correlations that may not be immediately obvious through statistical summaries alone.

By transforming numerical values into visual insights, we can identify trends, seasonal variations, and anomalies that influence decision-making and model building. Effective visualization supports clearer communication of findings and ensures a deeper understanding of the data structure and behavior.

6.2 Univariate Analysis

Univariate analysis is the examination of a single variable at a time to understand its distribution, central tendency, and variability. It is an important step in exploratory data analysis (EDA) because it helps identify patterns, detect anomalies, and summarize key characteristics of the data. For numeric variables, common techniques include histograms, boxplots, and descriptive statistics (mean, median, standard deviation), which reveal the spread and presence of outliers. For categorical variables, frequency counts and bar plots are used to assess the distribution of categories. Conducting univariate analysis provides a foundational understanding of each variable individually, which is essential before exploring relationships between variables in multivariate analysis.

```
# Histograms of numeric columns
df[['energy_requirement_mu','energy_availability_mu','energy_deficit']].hist(
    figsize=(12,6), bins=30)
plt.suptitle("Distribution of Numeric Features", fontsize=14)
plt.show()

# Boxplots for outliers
plt.figure(figsize=(12,6))
sns.boxplot(data=df[['energy_requirement_mu','energy_availability_mu','
    energy_deficit']])
plt.title("Boxplots of Numeric Features")
plt.show()

# Count plots for categorical features
plt.figure(figsize=(12,5))
sns.countplot(data=df, x="region", order=df['region'].value_counts().index)
plt.xticks(rotation=90)
plt.title("Region-wise Records")
plt.show()

plt.figure(figsize=(8,5))
sns.countplot(data=df, x="quarter")
plt.title("Quarter Distribution")
plt.show()
```

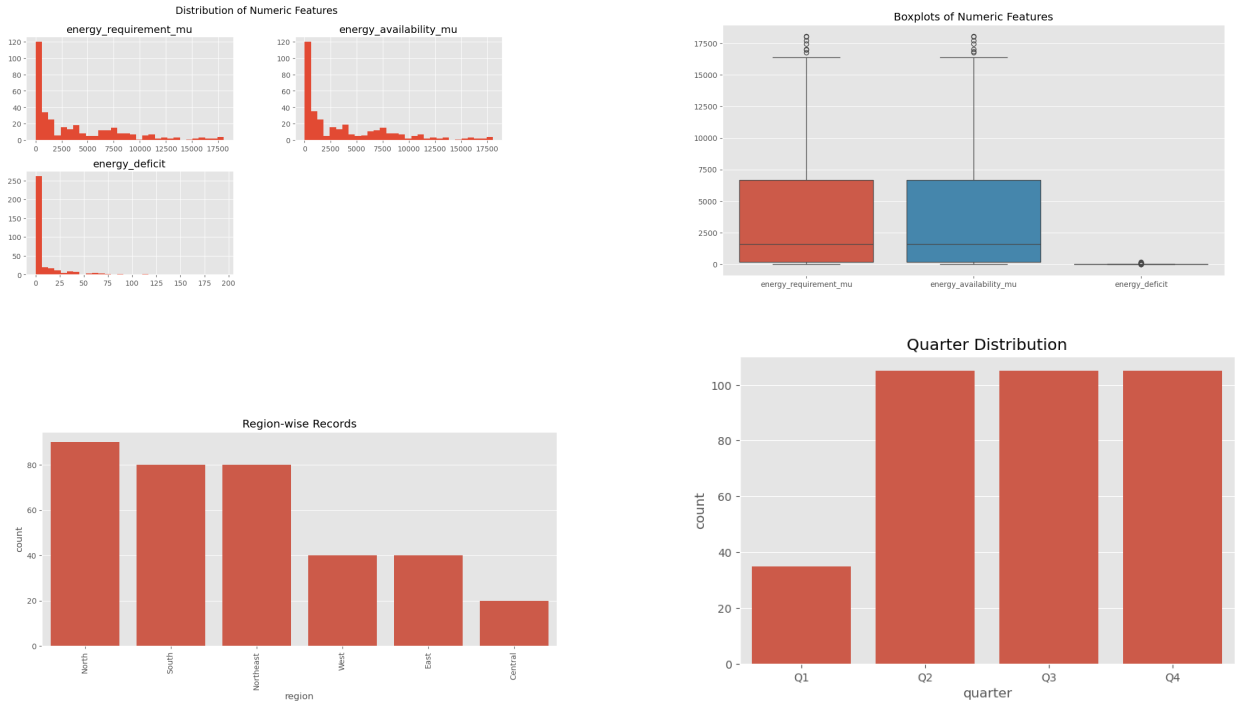

Figure 4: Uni-variate Analysis

Categorical Feature Distribution

- **Region-wise Records:**
  - The dataset is highly skewed towards the **North** ($\approx$ 90 records), **South**, and **Northeast** (both $\approx$ 80 records).
  - **Central** has the lowest count ($\approx$ 20 records).
- **Quarter Distribution:**
  - The data is heavily concentrated in **Q2, Q3, and Q4** (each $\approx$ 105 records).
  - **Q1** is significantly underrepresented ($\approx$ 35 records).

Numeric Feature Analysis

The features energy_requirement_mu, energy_availability_mu, and energy_deficit are all characterized by **strong positive skewness** (right-skewed).

- **energy_requirement_mu & energy_availability_mu:**
  - The majority of values are near **zero**, as indicated by a very low median.
  - Both features show a large **Interquartile Range** (IQR) up to $\approx 7,000$.
  - Numerous extreme **outliers** are present, extending up to $\approx 18,000$.
- **energy_deficit:**

- Values are highly clustered around **zero**; the median and IQR are near **0**.
- Shows a few small **outliers** up to ≈ 200, but on a much smaller scale than the other two features.

6.3 Bivariate Analysis

Bivariate analysis examines the relationship between two variables to identify patterns, correlations, or potential dependencies. It is an essential part of exploratory data analysis (EDA) because it helps uncover insights that are not visible when analyzing variables individually. For numeric-numeric pairs, scatter plots, correlation coefficients, and covariance are commonly used to assess linear or non-linear relationships. For numeric-categorical pairs, boxplots or violin plots can reveal differences in distributions across categories. For categorical-categorical pairs, cross-tabulations and stacked bar charts help identify associations. Conducting bivariate analysis provides a deeper understanding of how variables interact, which is crucial for feature selection, predictive modeling, and hypothesis testing.

```
# Requirement vs Availability
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x="energy_requirement_mu", y="energy_availability_mu"
    , hue="region", alpha=0.7)
plt.title("Energy Requirement vs Availability")
plt.show()

# Avg Deficit by Region
plt.figure(figsize=(12,6))
sns.barplot(data=df, x="region", y="energy_deficit", estimator=np.mean, order=
    df.groupby('region')['energy_deficit'].mean().sort_values(ascending=False)
    .index)
plt.xticks(rotation=90)
plt.title("Average Energy Deficit by Region")
plt.show()

# Deficit by Quarter
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x="quarter", y="energy_deficit")
plt.title("Energy Deficit by Quarter")
plt.show()
```

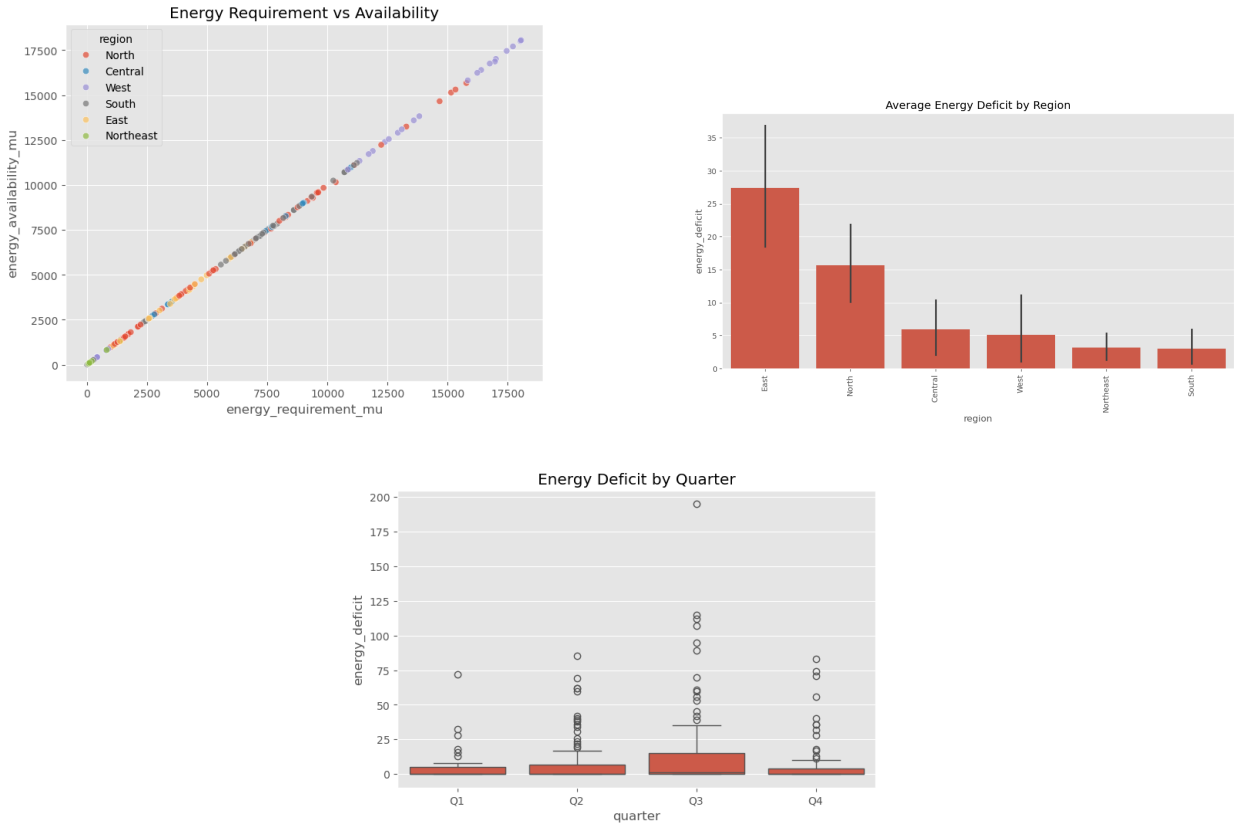

Figure 5: Uni-variate Analysis

Energy Requirement vs Availability (Scatter Plot)

- **Strong Positive Correlation:** The scatter plot shows a nearly **perfect linear relationship** between energy_requirement_mu and energy_availability_mu. This indicates that as the required energy increases, the available energy increases at a highly proportional rate.
- **Near Zero Deficit:** Since the points fall extremely close to the line $y = x$, this implies that for most data points, energy_availability_mu is almost exactly equal to energy_requirement_mu, resulting in a minimal energy_deficit.
- **Region Distribution:** Data points from all regions (North, Central, West, South, East, Northeast) are distributed uniformly along the correlation line, suggesting this high correlation holds true across all regions.

Average Energy Deficit by Region (Bar Chart with Error Bars)

- **Highest Average Deficit:** The **East** region has the highest average energy deficit (≈ 28), followed by the **North** (≈ 16).
- **Lowest Average Deficit:** The **Northeast** and **South** regions show the lowest average deficit (≈ 3).

- **Variability (Error Bars):** The error bars (representing standard deviation or confidence interval) are substantial, particularly for the **East** and **North** regions. This indicates high variability (or wide data spread) in the energy deficit within these regions. The large error bar for the East suggests the average is driven by a few very high deficit values.

Energy Deficit by Quarter (Box Plot)

- **Median Deficit:** For all quarters (**Q1, Q2, Q3, Q4**), the median energy_deficit (the line inside the box) is very close to **zero**.
- **Variability: Q3** shows the largest Interquartile Range (IQR) (≈ 0 to ≈ 15) and the highest maximum values/outliers.
- **Extreme Outliers:**
  - **Q3** exhibits the most extreme outlier, nearly reaching **200**.
  - **Q2** and **Q4** also have numerous outliers, but the maximum values are lower (≈ 115 for Q3; ≈ 85 for Q4).
  - **Q1** has the lowest overall spread and maximum outlier value (≈ 75).
- **Conclusion:** While the typical energy deficit is negligible across all quarters, the **third quarter (Q3)** experiences the largest and most extreme deficits.

6.4 Correlation Analysis

Correlation analysis measures the strength and direction of the relationship between two numeric variables. It is an important part of exploratory data analysis (EDA) because it helps identify which variables are positively or negatively associated, and which may be redundant. Common techniques include calculating the Pearson correlation coefficient for linear relationships, the Spearman rank correlation for monotonic relationships, and visualizing correlations using heatmaps or pair plots. Understanding correlations aids in feature selection, detecting multicollinearity, and guiding further statistical modeling or machine learning analysis.

```
plt.figure(figsize=(8,6))
sns.heatmap(df[['energy_requirement_mu','energy_availability_mu','
    energy_deficit']].corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

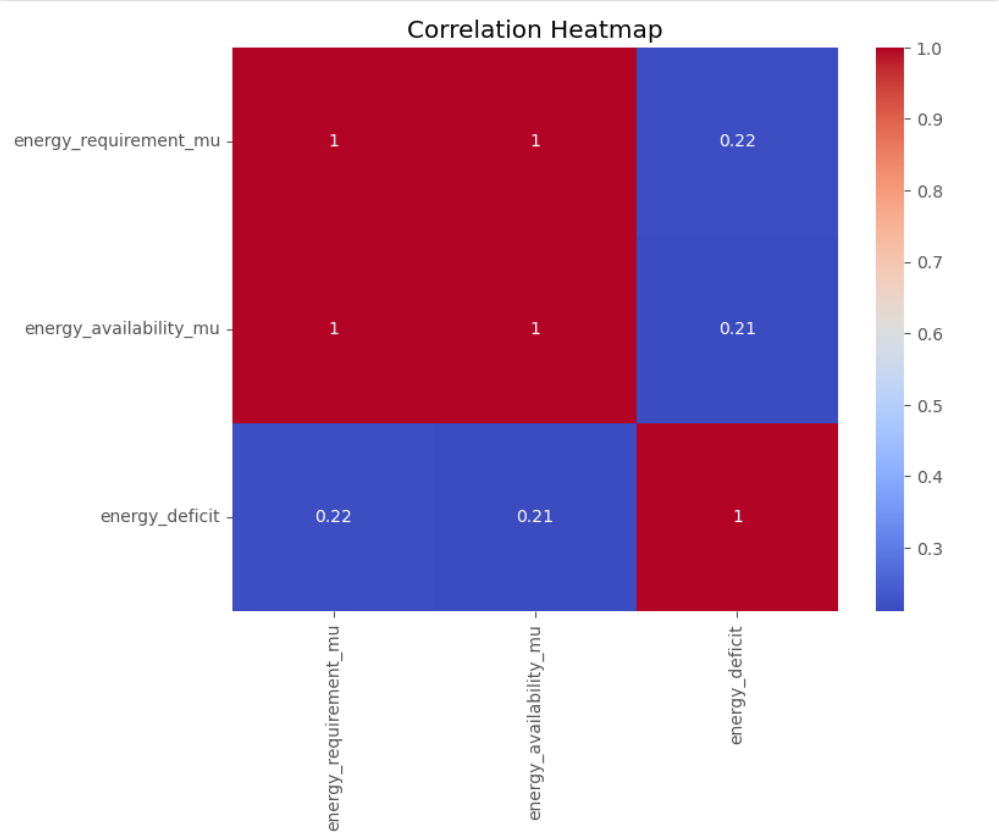

Figure 6: Boxplots for Energy Metrics

The heatmap displays the Pearson correlation coefficients among the three numeric features: energy_requirement_mu, energy_availability_mu, and energy_deficit.

- **Perfect Positive Correlation (1):**
  - The correlation between energy_requirement_mu and energy_availability_mu is **1**.
  - This confirms the observation from the scatter plot (previous analysis) that these two variables are **perfectly linearly related**. They move together in lockstep, meaning high requirement is always matched by high availability, and vice-versa.
- **Low Positive Correlation (0.21 to 0.22):**
  - The correlation between energy_deficit and energy_requirement_mu is **0.22**.
  - The correlation between energy_deficit and energy_availability_mu is **0.21**.
  - These values indicate a **very weak positive correlation**. Although statistically significant, this relationship suggests that higher energy requirement/availability only slightly contributes to a larger energy deficit.
  - Given the definition: Energy Deficit = Requirement − Availability, and the fact that Requirement ≈ Availability, the deficit itself is mostly close to zero, explaining the near-zero correlation with its constituent parts. The slight positive value

is likely driven by the few extreme outliers where the small difference is magnified by the large magnitude of the requirement/availability values.

6.5 Time Series Trends

Time series trend analysis involves examining data points collected or recorded at regular time intervals to identify underlying patterns, trends, or seasonal effects. It is important in exploratory data analysis (EDA) because it helps understand how variables evolve over time, detect long-term increases or decreases, and identify periodic fluctuations. Visualizations such as line plots, rolling averages, and seasonal decomposition are commonly used to highlight trends, cycles, and anomalies. Analyzing time series trends provides valuable insights for forecasting, planning, and decision-making based on temporal dynamics.

6.5.1 Energy Requirement vs Availability by Month

```
month_order = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
df["month"] = pd.Categorical(df["month"], categories=month_order, ordered=True)

# ---------------------------------
# 1. Energy Requirement vs Availability (by Month)
# ---------------------------------
plt.figure(figsize=(12,6))
sns.lineplot(data=df, x="month", y="energy_requirement_mu", label="Requirement", marker="o")
sns.lineplot(data=df, x="month", y="energy_availability_mu", label="Availability", marker="o")
plt.title("Energy Requirement vs Availability by Month")
plt.ylabel("Energy (MU)")
plt.xlabel("Month")
plt.legend()
plt.show()
```

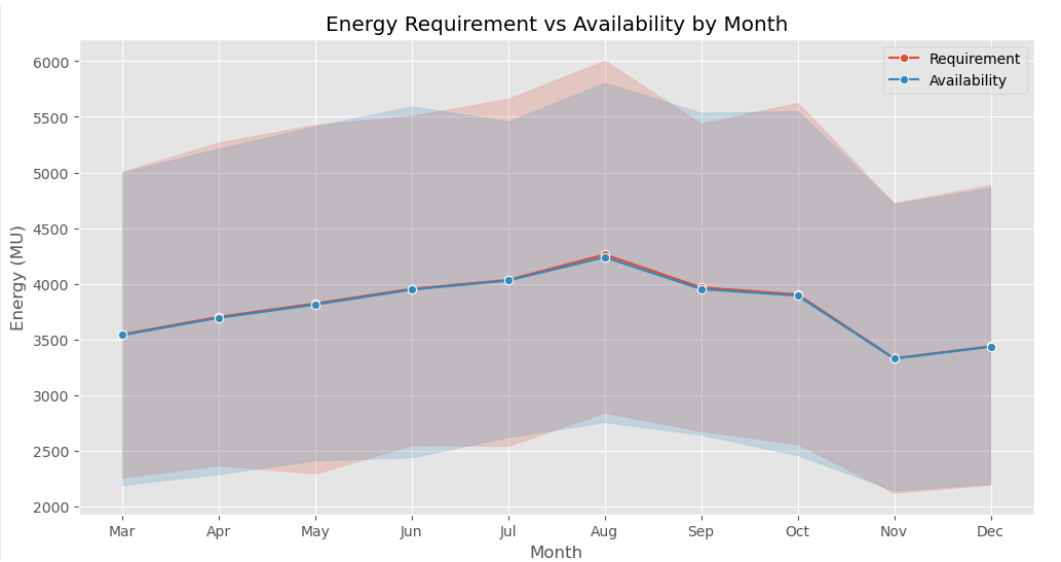

Figure 7: Energy Requirement vs Availability (by Month)

The line plot illustrates the mean monthly trends for Energy Requirement (MU) and Energy Availability (MU), with the shaded area likely representing the variability (e.g., standard deviation or range).

- **High Correlation/Near Equality:** The lines for Requirement (red) and Availability (blue) are almost perfectly superimposed throughout the year, confirming the high correlation observed previously. The **mean requirement is nearly identical to the mean availability** in every month.
- **Peak Energy Period (Mid-Year):**
  - Both energy requirement and availability **peak in August** at approximately $\mathbf{4,250}$ MU.
  - They show a steady increase from March to August (March $\approx 3,550$ MU).
- **Trough Energy Period (Year-End):**
  - After the August peak, both metrics steadily **decline to their lowest point in November** ($\approx 3,350$ MU).
  - There is a slight recovery in December.
- **Variability (Shaded Area):**
  - The **variability** (spread between the upper and lower bounds of the shaded region) for both requirement and availability is **highest during the peak summer months** (July and August) and generally remains large throughout the year, indicating a significant difference between minimum and maximum recorded values for these months.
  - The variability **decreases noticeably in November**, which corresponds to the overall trough in energy values.

6.5.2 Energy Deficit Trend by Region

```
# Energy Deficit by Region (month trend)
plt.figure(figsize=(14,6))
sns.lineplot(data=df, x="month", y="energy_deficit", hue="region", marker="o")
plt.title("Energy Deficit Trend by Region")
plt.ylabel("Deficit (MU)")
plt.xlabel("Month")
plt.show()
```

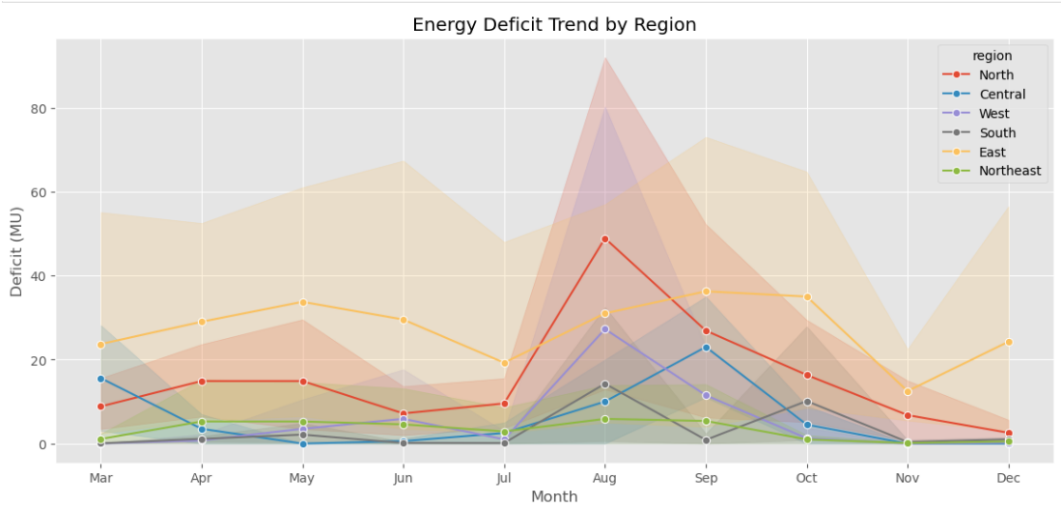

Figure 8: Energy Deficit by Region (month trend)

The line plot illustrates the monthly trend of the **mean energy deficit (MU)** for six different regions, with the shaded areas representing the variability (spread) of the deficit within each region.

- **Overall Deficit Peak:** The most significant finding is a sharp, distinct **peak in the energy deficit for all regions in August**. This confirms the earlier observation that the third quarter (which includes August) experiences the largest deficits.
- **Region-Specific Trends:**
  - **North (Red):** Shows the highest mean deficit peak, reaching nearly **50 MU** in August. Its overall variability (shaded area) is also very large, particularly around the August peak (≈ 90 MU).
  - **East (Orange):** Maintains the highest average deficit for most of the year (March-July, September-October). It peaks around **35 MU** in September, slightly after the overall August peak. Its variability is consistently high.
  - **West (Purple):** Exhibits a very sharp, extreme variability peak in August (≈ 95 MU), but its mean deficit line remains relatively lower than the North and East.
  - **Northeast (Green) & South (Black):** Consistently show the **lowest mean deficits** throughout the year, typically staying close to or below **5** MU. Their variability is also the lowest.

- **Seasonality:**
  - The deficit tends to be higher during the **summer months** (May to October).
  - The deficit **dips significantly in November** for all regions, then shows a slight rise again in December.

6.5.3 Top 10 States by Average Energy Deficit

```
top_states = df.groupby("state")["energy_deficit"].mean().sort_values(
    ascending=False).head(10)

plt.figure(figsize=(12,6))
sns.barplot(x=top_states.index, y=top_states.values, palette="viridis")
plt.xticks(rotation=45)
plt.title("Top 10 States by Average Energy Deficit")
plt.ylabel("Avg Deficit (MU)")
plt.show()
```

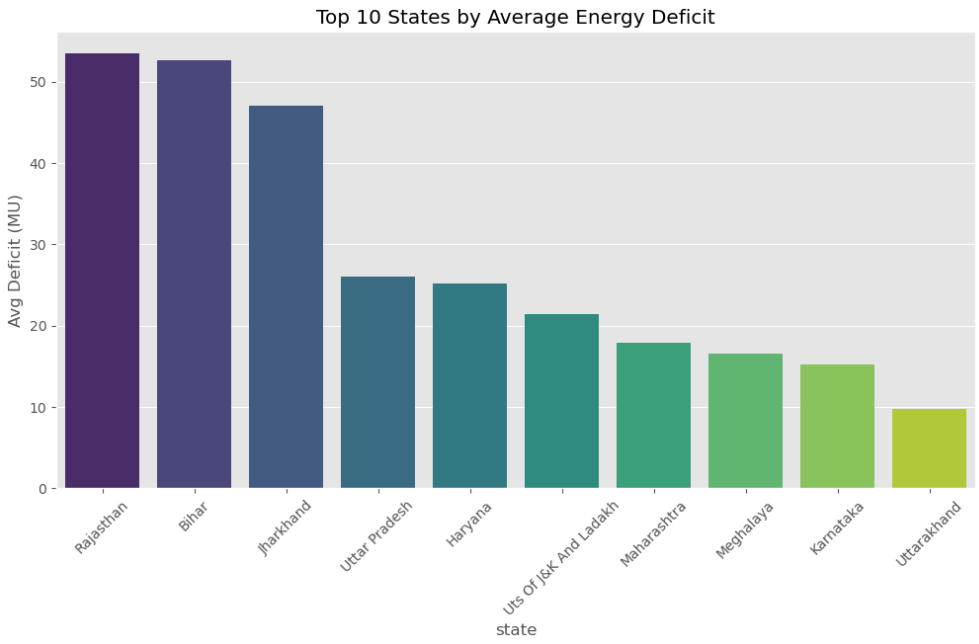

Figure 9: Top 10 States by Average Energy Deficit

The bar chart displays the top 10 states with the highest average energy deficit, measured in Mega Units (MU).

- **Leading States (Highest Deficit):**
  - **Rajasthan** has the highest average energy deficit, at approximately **53** MU.
  - **Bihar** follows extremely closely with an average deficit of about **52** MU.
  - **Jharkhand** is the third highest, with an average deficit of around **47** MU.

- **Mid-Range Deficit:**
  - States like **Uttar Pradesh** and **Haryana** have an average deficit around **25** MU.
  - The deficit drops further for **J&K and Ladakh** ($\approx$ 22 MU) and **Maharashtra** ($\approx$ 18 MU).
- **Lowest Deficit among Top 10:**
  - The bottom three states in this top 10 list are **Meghalaya** ($\approx$ 17 MU), **Karnataka** ($\approx$ 15 MU), and **Uttarakhand** ($\approx$ 10 MU).
- **Observation:** There is a significant and steep drop-off in the average deficit from the top three states (Rajasthan, Bihar, Jharkhand) to the rest of the states listed, indicating that these three states contribute disproportionately to the overall high deficits in the dataset.

6.5.4 Average Energy Deficit by Month (All Regions)

```
plt.figure(figsize=(12,6))
monthly_avg = df.groupby("month")["energy_deficit"].mean().reindex(df['month'].cat.categories)
sns.lineplot(x=monthly_avg.index, y=monthly_avg.values, marker="o", color="red")
plt.title("Average Energy Deficit by Month (All Regions)")
plt.ylabel("Avg Deficit (MU)")
plt.xlabel("Month")
plt.grid(True)
plt.show()
```

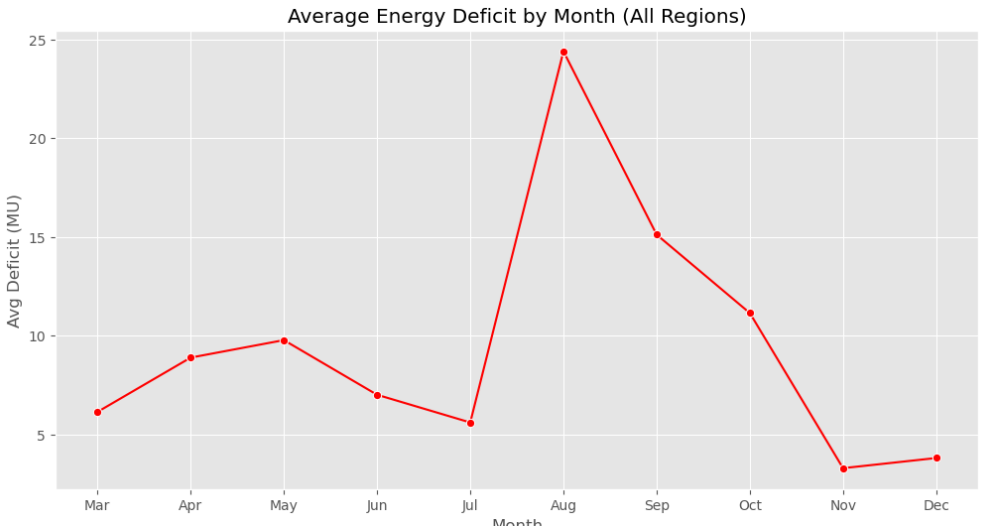

Figure 10: Top 10 States by Average Energy Deficit

The line plot shows the monthly trend of the overall average energy deficit across all regions combined, measured in Mega Units (MU).

- **Peak Deficit:** The average energy deficit for all regions **peaks sharply in August** at approximately **24.5** MU. This aligns with the regional deficit analysis, confirming August as the month with the most significant deficit issues overall.
- **High Deficit Period:** Following the peak, the deficit remains high in **September** (≈ 15 MU) and **October** (≈ 11 MU), indicating a critical period for energy management spanning August through October.
- **Lowest Deficit Period:**
  - The deficit is at its lowest point in **November**, dropping sharply to about **3.5** MU.
  - **December** shows a very slight increase (≈ 4 MU).
  - The deficit in **July** is also relatively low, at approximately **5.5** MU, following a minor dip from June.
- **Pre-Peak Trend:** The deficit starts relatively low in March (≈ 6 MU) and shows a gradual increase through April and May, before the slight dip in June and July and the massive surge in August.

6.5.5 Monthly Energy Deficit by Quarter

```
plt.figure(figsize=(12,6))
sns.lineplot(data=df, x="month", y="energy_deficit", hue="quarter", marker="o", palette="Set2")
plt.title("Monthly Energy Deficit by Quarter")
plt.ylabel("Deficit (MU)")
plt.xlabel("Month")
plt.show()
```

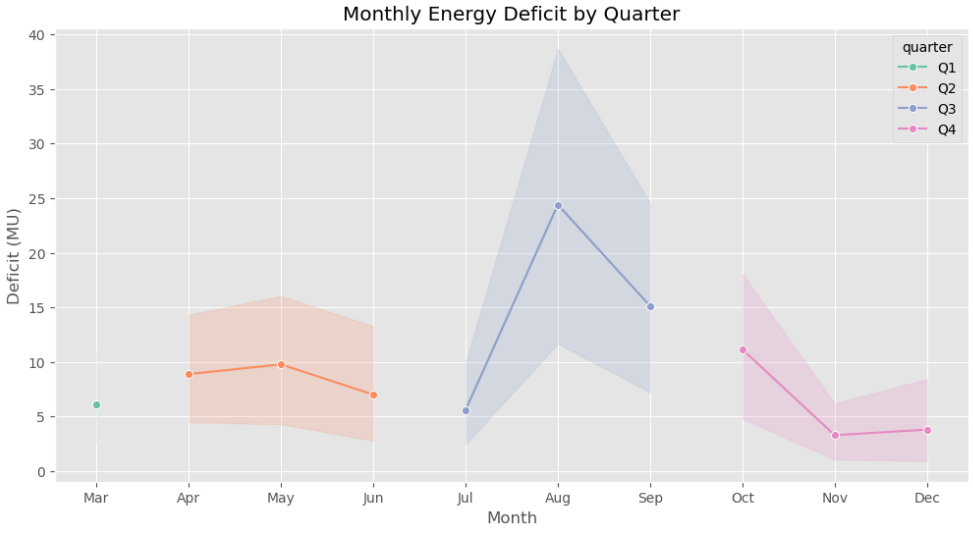

Figure 11: Top 10 States by Average Energy Deficit

The line plot breaks down the mean monthly energy deficit (MU) and its variability (shaded area) according to the four quarters (Q1, Q2, Q3, Q4). Note that the data only covers a portion of the year (March to December).

- **Q3 (July, August, September):**
  - Exhibits the **highest mean deficit**, peaking dramatically in **August** at nearly **24** MU.
  - The mean deficit drops significantly in September ($\approx$ 15 MU) and is lowest in July ($\approx$ 5.5 MU).
  - This quarter also shows the **largest variability** (widest shaded area), particularly in August, indicating extreme fluctuations in the deficit during this month.
- **Q2 (April, May, June):**
  - Shows a moderate, stable mean deficit, peaking in **May** at around **10** MU.
  - The mean values for April ($\approx$ 9 MU) and June ($\approx$ 7 MU) are slightly lower.
  - Variability is substantial but consistent across the three months.
- **Q4 (October, November, December):**
  - Starts with a moderate mean deficit in **October** ($\approx$ 11 MU), which is comparable to Q2.
  - The deficit drops sharply to its **lowest point in November** ($\approx$ 3.5 MU) and remains low in December ($\approx$ 4 MU).
  - The overall deficit is low, and the variability also significantly decreases in November and December.
- **Q1 (March):**
  - Only the data for **March** is shown, with a mean deficit of approximately **6** MU. No variability information is displayed for this single point.
- **Conclusion:** The data highlights a strong seasonal effect on the energy deficit, with Q3 (driven by August) being the most problematic period, and Q4 (November/December) having the most successfully managed/lowest deficits.

6.6 Average Energy Deficit: State vs Month

```
pivot = df.pivot_table(values="energy_deficit", index="state", columns="month"
    , aggfunc="mean")
plt.figure(figsize=(12,8))
sns.heatmap(pivot, annot=True, fmt=".1f", cmap="YlGnBu")
plt.title("Average Energy Deficit: State vs Month")
plt.show()
```

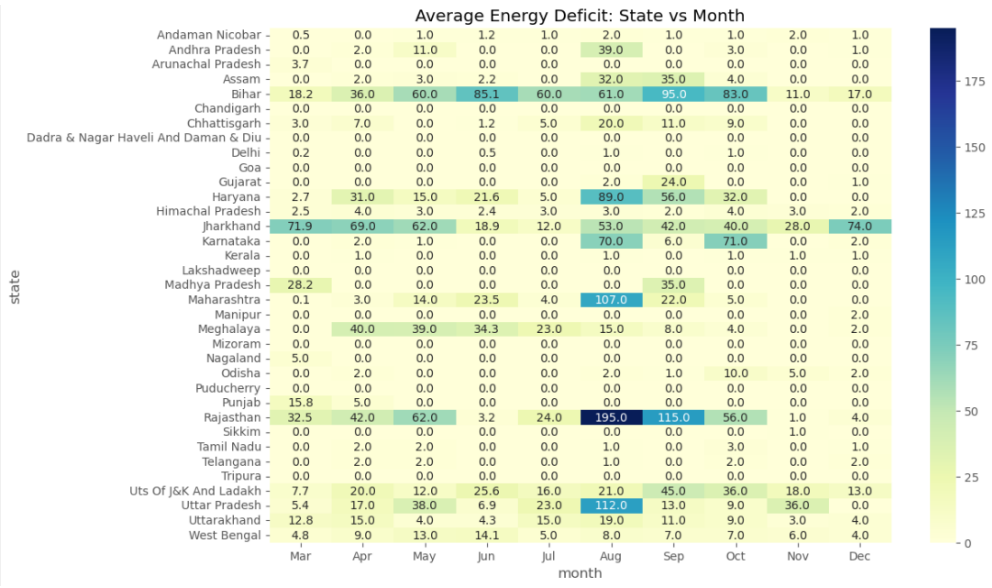

Figure 12: Average Energy Deficit: State vs Month

The heatmap shows the average energy deficit (in MU) for various states across the months of the year (March to December). The color intensity (from yellow/low to dark blue/high) indicates the magnitude of the deficit.

- **Critical Deficit Period: August and September**
  - The months of **August** and **September** consistently show the highest deficits across many states, confirming the peak observed in the aggregated plots. This period is the most challenging for energy management.
- **States with Extreme Deficits (Dark Blue/High MU):**
  - **Rajasthan:** Exhibits the most extreme deficit, reaching **195.0** MU in **August**, and remaining very high in September (**115.0** MU).
  - **Bihar:** Shows severe deficits in August (**95.0** MU) and April (**60.0** MU).
  - **Jharkhand:** Peaks in December (**74.0** MU), which is an unusual year-end peak compared to other states, and also shows high values in April (**69.0** MU) and May (**62.0** MU).
- **States with High Deficits (Medium Blue/Moderate MU):**
  - **Uttar Pradesh:** Has a significant deficit in September (**112.0** MU).
  - **Haryana:** Experiences high deficits in August (**89.0** MU) and September (**56.0** MU).
  - **Karnataka:** Shows a high deficit in September (**71.0** MU).
- **States with Negligible Deficits (Yellow/Near 0 MU):**

- Many states, including **Goa, Gujarat, Kerala, Puducherry, Punjab, Sikkim, and Tamil Nadu**, report **0.0** MU or very low deficits across almost all months, suggesting successful energy balance or low demand.

- **Conclusion on Deficit Pattern:** The heatmap clearly identifies a small subset of states (**Rajasthan, Bihar, Jharkhand, UP, Haryana**) that are responsible for the vast majority of the high average energy deficits, predominantly during the summer and post-monsoon months (April-May, August-September). Jharkhand's peak in December is an anomaly.

6.7 Average Energy Deficit: Region vs Month

```
pivot = df.pivot_table(values="energy_deficit", index="region", columns="month", aggfunc="mean")
plt.figure(figsize=(12,6))
sns.heatmap(pivot, annot=True, fmt=".1f", cmap="YlOrRd")
plt.title("Average Energy Deficit: Region vs Month")
plt.show()
```

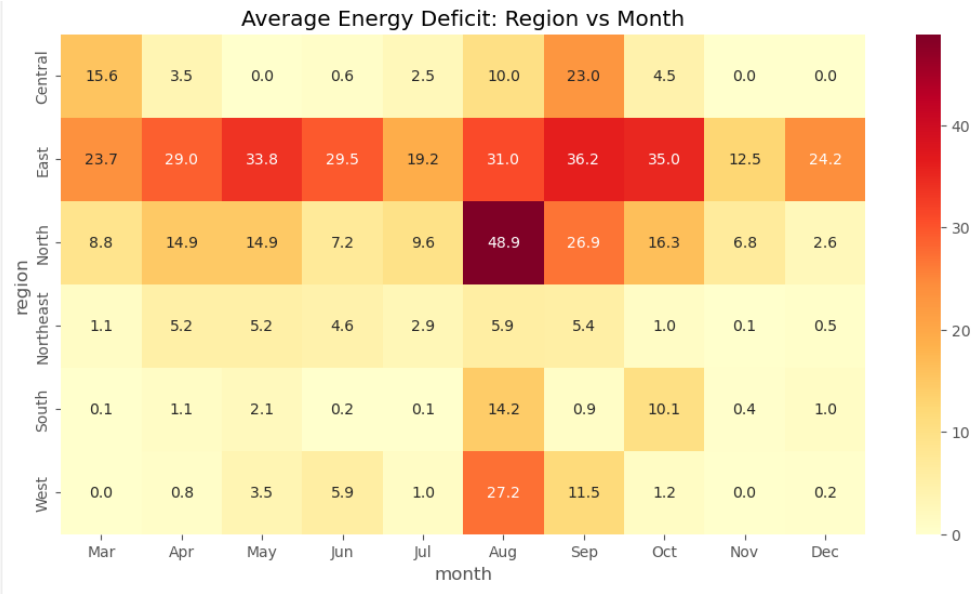

Figure 13: Average Energy Deficit: Region vs Month

The heatmap displays the average energy deficit (in MU) broken down by region and month (March to December). Color intensity (from light yellow/low to dark red/high) visually represents the magnitude of the deficit.

- **Overall Peak Deficit:**

  - The single largest average deficit occurs in the **North** region in **August** at **48.9** MU. This aligns with the extreme outliers and August peak observed in earlier graphs.

- **Worst Performing Region: East**
  - The **East** region shows the most persistent and high average deficit throughout the year. Its deficit is consistently in the **20**s and **30**s MU range from March to October, peaking in September (**36.2** MU) and October (**35.0** MU).
  - Unlike the North, which spikes sharply in August, the East's deficit is problematic for a much longer duration.
- **Seasonal Peaks:**
  - **North** and **West** regions show a sharp, distinct deficit peak in **August** (**48.9** MU and **27.2** MU respectively).
  - **East** and **Central** regions maintain high deficits into the early post-monsoon months, peaking in **September** (**36.2** MU) and **August/September** (**23.0** MU).
- **Best Performing Regions (Lowest Deficit):**
  - **Northeast** and **South** consistently maintain very low deficits, generally below **6** MU.
  - The deficit in the **Northeast** and **South** drops close to **0** in November, similar to the general trend.
- **Conclusion:** High energy deficits are primarily concentrated in the **East** (long-term high deficit) and the **North** (extreme peak deficit in August), especially during the summer and early post-monsoon periods.

6.8 Pairwise Relationships by Region

```
sns.pairplot(df[numeric_cols + ["region"]], hue="region", diag_kind="kde")
plt.suptitle("Pairwise Relationships by Region", y=1.02)
plt.show()
```

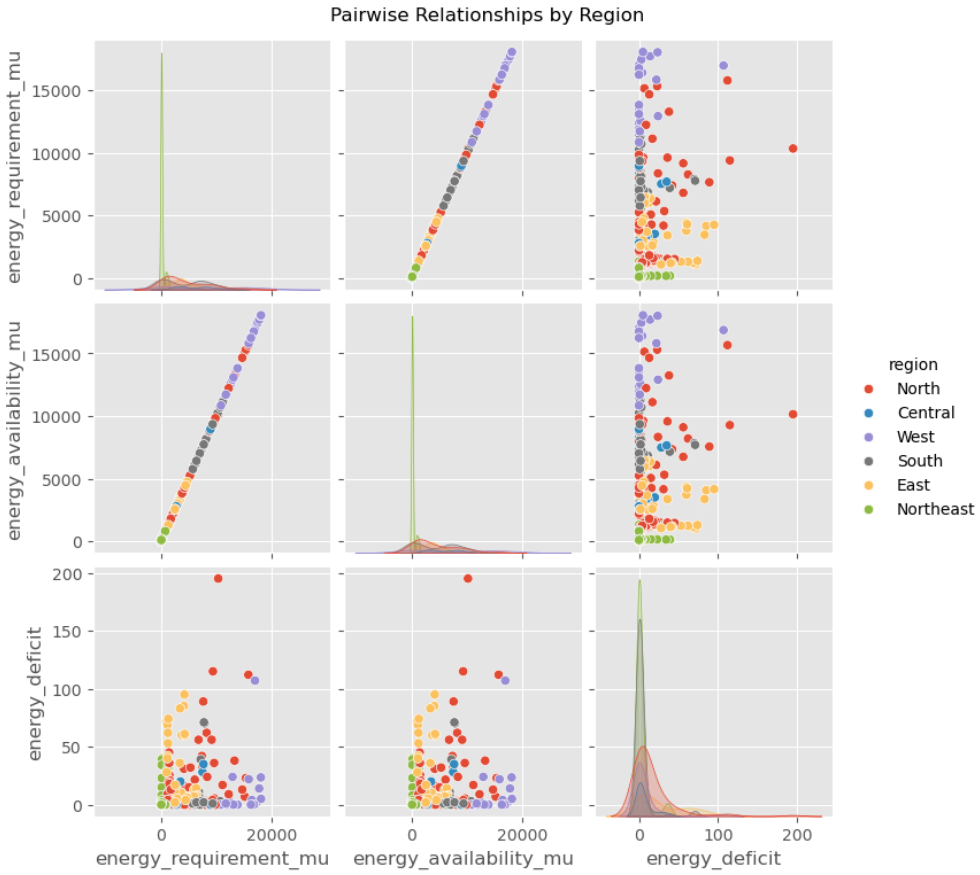

Figure 14: Pairwise Relationships by Region

The Pairplot displays the distributions of the three numeric features (energy_requirement_mu, energy_availability_mu, energy_deficit) along the diagonal (histograms/KDE plots) and the pairwise scatter plots between them. The data is colored by region.

Diagonal Plots (Distributions)

- **energy_requirement_mu & energy_availability_mu:**
  - Both distributions are **highly right-skewed**, with the majority of data clustered near zero.
  - The **North** (Red) and **Central** (Blue) regions appear to contribute most of the extreme, high-value data points (the long tails of the distribution). The **Northeast** (Green) appears to be concentrated at the very low end.
- **energy_deficit:**
  - The deficit is overwhelmingly concentrated near **zero**.
  - The **North** (Red) and **East** (Orange) regions show the most significant spread and contribute the majority of the non-zero deficit values.

Off-Diagonal Plots (Scatter Plots)

- **energy_requirement_mu vs energy_availability_mu (Top-Middle/Middle-Top):**
  - There is a **near-perfect positive linear relationship** (correlation ≈ 1). All data points, irrespective of region, fall extremely close to the $y = x$ line.
  - This re-confirms that energy availability closely matches energy requirement across all regions and magnitudes.
- **energy_deficit vs Requirement/Availability (Bottom Row):**
  - The plots show a **weak positive relationship**. The large deficits (high on the Y-axis) are scattered across the range of requirement/availability values (X-axis) but tend to be more pronounced for higher requirement/availability values.
  - **High Deficits by Region:** The large deficit points (Deficit > 100) are primarily driven by the **North** (Red) and **East** (Orange) regions, while the **Northeast** (Green) points are all clustered near Deficit $= 0$.
  - This confirms that the severe deficits, though rare, are mostly concentrated in specific regions.

6.9 Test for Statistical Properties

Testing for statistical properties such as normality, stationarity, and homoscedasticity is essential to ensure that the data meets the assumptions required for various statistical models and hypothesis tests. For example, many regression techniques assume that the residuals are normally distributed and homoscedastic (i.e., have constant variance), while time series models often require the data to be stationary. Verifying these properties helps improve the reliability and validity of the results and ensures that the analytical methods used produce unbiased and interpretable outcomes.

6.9.1 Normality Tests

Normality tests are used to determine whether a variable or the residuals of a model follow a normal (Gaussian) distribution — a key assumption for many statistical techniques such as t-tests, ANOVA, and linear regression. Assessing normality helps ensure the validity of these methods and the reliability of the results. In this project, normality tests such as the Shapiro-Wilk test or visual methods like Q-Q plots and histograms can be used to evaluate the distribution of features like `energy_requirement`, `energy_availability`, and `energy_gap`. If normality is violated, appropriate transformations or non-parametric methods may be applied.

```
from scipy.stats import shapiro

for col in numeric_cols:
    stat, p = shapiro(df[col])
    print(f"{col}: W={stat:.4f}, p={p:.4f}", end="      ")
    if p > 0.05:
        print("Probably Normal")
    else:
```

```
        print("Not Normal")

numeric_cols = ["energy_requirement_mu","energy_availability_mu","
    energy_deficit","gap"]

for col in numeric_cols:
    plt.figure(figsize=(8,4))
    sns.histplot(df[col], kde=True, bins=30, color="skyblue")
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

```
energy_requirement_mu: W=0.8172, p=0.0000      Not Normal
energy_availability_mu: W=0.8167, p=0.0000      Not Normal
energy_deficit: W=0.4973, p=0.0000      Not Normal
gap: W=0.4973, p=0.0000      Not Normal
```

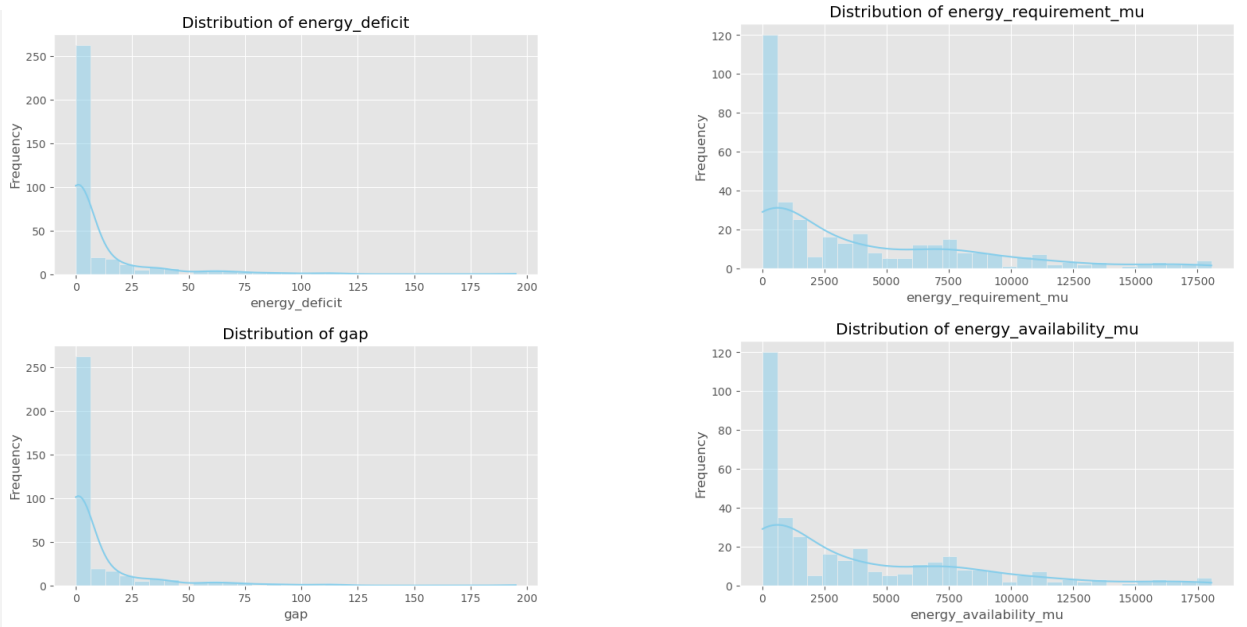

Figure 15: Shapiro-Wilk test

The analysis focuses on the visual distribution (Histograms with Kernel Density Estimates, KDE) and the statistical results of the Shapiro-Wilk (W) test for normality for the four numeric features.

Visual Distribution (Histograms)

- **energy_requirement_mu and energy_availability_mu:**
  - Both features show a **strong positive skew** (right-skewed).

- The majority of the observations (Frequency ≈ 120) are clustered near **zero** (**0** to $\mathbf{2,500}$).
- The distributions have long tails extending to $\mathbf{17,500}$, indicating the presence of extreme high values (outliers).

- **energy_deficit and gap:**
  - Both features are **extremely skewed** to the right.
  - The overwhelming majority of observations (Frequency ≈ 250) are concentrated very close to **zero** (**0** to **5**).
  - The deficit/gap values rarely exceed **25** MU, with a tail extending up to **200** MU showing a few extreme outliers.
  - Given the perfect correlation between energy_requirement_mu and energy_availability_mu, it's likely that energy_deficit and gap are nearly identical or mathematically equivalent measures of the difference between requirement and availability.

Statistical Normality Test (Shapiro-Wilk Results)

The Shapiro-Wilk test (W) is used to test the null hypothesis (H_0): the data is drawn from a normal distribution. A p-value (p) less than the significance level (α, typically 0.05) leads to the rejection of H_0.

Feature	**W Statistic**	**p-value**	**Conclusion (p < 0.05)**
energy_requirement_mu	0.8172	0.0000	**Not Normal**
energy_availability_mu	0.8167	0.0000	**Not Normal**
energy_deficit	0.4973	0.0000	**Not Normal**
gap	0.4973	0.0000	**Not Normal**

- **Rejection of Normality:** For all four features, the p-value is **0.0000**, which is far below the typical $\alpha = 0.05$ threshold.
- **Conclusion:** The null hypothesis of a normal distribution is decisively **rejected** for every feature.
- **Consistency:** This statistical result strongly confirms the visual assessment from the histograms: none of the features follow a Gaussian (normal) distribution; they are all heavily right-skewed.

7 Machine Learning Algorithms Overview

This section provides an outline of the machine learning algorithms applied to the dataset. The goal is to model and predict energy-related outcomes across different Indian states, using features such as time (year, month) and location (state).

The following algorithms have been selected for experimentation:

- **Linear Regression:** Serves as a baseline regression model, establishing a linear relationship between independent variables and the target variable.
- **K-Nearest Neighbors (KNN):** A non-parametric method that predicts the target based on the closest training samples in the feature space.
- **Naive Bayes:** A probabilistic classifier based on Bayes' theorem, assuming independence among features. Suitable for categorical or discretized features.
- **Logistic Regression:** A classification algorithm used when the target is categorical (e.g., high/low energy deficit). It estimates the probability of class membership using the logistic function.

Model Performance Comparison

At the end of the modeling process, the performance of each algorithm is evaluated using standard metrics such as R^2 score and Mean Squared Error (MSE). These metrics provide insights into the accuracy and robustness of the predictions made by each model.

The comparison enables identification of the most suitable algorithm for predicting energy requirements, balancing both performance and interpretability. The final model selection is based on a holistic assessment of predictive power, overfitting control, and computational efficiency.

7.1 Prepare dataset

In this step, the cleaned dataset is loaded into a pandas DataFrame for analysis and modeling. Essential Python libraries for data manipulation, visualization, preprocessing, and machine learning are imported. Two new columns are created to enrich the dataset: `gap`, which represents the difference between `energy_requirement_mu` and `energy_availability_mu`, and `deficit_flag`, a binary indicator denoting whether there is an energy deficit (1) or not (0). This preparation ensures that both regression and classification tasks can be performed, and it provides a foundation for subsequent exploratory data analysis and model building.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, classification_report,
    mean_squared_error, r2_score

plt.style.use("ggplot")

# Load dataset
df = pd.read_csv("cleaned_dataset.csv")

# Create additional columns
df["gap"] = df["energy_requirement_mu"] - df["energy_availability_mu"]
df["deficit_flag"] = (df["energy_deficit"] > 0).astype(int)

# Optional: preview
df.head()
```

	region	state	is_union_territory	month	quarter	energy_requirement_mu	energy_availability_mu	energy_deficit	gap	deficit_flag
0	North	Chandigarh	True	Mar	Q1	101.600	101.600	0.000	0.000	0
1	North	Delhi	True	Mar	Q1	2130.478	2130.279	0.199	0.199	1
2	North	Haryana	False	Mar	Q1	3911.822	3909.160	2.662	2.662	1
3	North	Himachal Pradesh	False	Mar	Q1	1025.630	1023.160	2.470	2.470	1
4	North	Uts Of J&K And Ladakh	True	Mar	Q1	1725.610	1717.930	7.680	7.680	1

Figure 16: Dataset Preview

7.2 Preprocessing for Base and Seasonality Models

In this step, the dataset is prepared for both base and seasonality models. For the **base model**, the features include the numeric columns `energy_requirement_mu`, `energy_availability_mu`, and the previously computed `gap`, while the target variables are `deficit_flag` for classification and `energy_deficit` for regression. For the **seasonality model**, categorical features `month` and `quarter` are one-hot encoded and combined with the numeric base features to capture seasonal effects. The datasets are then split into training and testing sets using a 70-30 split, ensuring that both classification and regression tasks can be performed on base and seasonality features separately. This preprocessing ensures that the models have structured inputs suitable for learning patterns in both general energy behavior and seasonal variations.

```
# Base features
base_features = ["energy_requirement_mu", "energy_availability_mu", "gap"]  # remove target

# Seasonality features: numeric + one-hot month/quarter
season_features_numeric = base_features
season_features_cat = ["month","quarter"]

# One-hot encode categorical columns for seasonality
df_season = pd.get_dummies(df[season_features_numeric + season_features_cat], drop_first=True)

# Base X
X_base = df[base_features].values
y_class = df["deficit_flag"].values
y_reg = df["energy_deficit"].values

# Seasonality X
X_season = df_season.values

# Train-Test Split
from sklearn.model_selection import train_test_split

# Base
X_train_base_clf, X_test_base_clf, y_train_clf, y_test_clf = train_test_split(X_base, y_class, test_size=0.3, random_state=42)
X_train_base_reg, X_test_base_reg, y_train_reg, y_test_reg = train_test_split(X_base, y_reg, test_size=0.3, random_state=42)

# Seasonality
X_train_season_clf, X_test_season_clf, _, _ = train_test_split(X_season, y_class, test_size=0.3, random_state=42)
X_train_season_reg, X_test_season_reg, _, _ = train_test_split(X_season, y_reg, test_size=0.3, random_state=42)
```

8 Machine Learning Techniques

Machine learning techniques play a vital role in uncovering patterns, relationships, and insights from complex datasets. By leveraging algorithms such as regression, classification, clustering, and neural networks, we can model both linear and nonlinear relationships within the data. These techniques enable the system to learn from past observations and make accurate predictions or classifications on new, unseen data. In the context of energy analytics, machine learning helps in identifying key drivers of energy demand, forecasting consumption trends, and detecting inefficiencies. Employing a variety of ML approaches ensures robust analysis, enhances decision-making, and leads to more data-driven, reliable, and scalable solutions.

8.1 Clustering

Clustering is an unsupervised machine learning technique used to group similar data points into clusters based on their underlying patterns or similarities. Unlike supervised learning, clustering does not rely on predefined labels; instead, it automatically discovers structure within the data. Algorithms such as **K-Means**, **Hierarchical Clustering**, and **DBSCAN** are commonly used to segment data into meaningful groups. In the context of energy analysis or environmental studies, clustering can help identify regions with similar energy consumption patterns or pollution levels. This technique is valuable for exploratory data analysis, customer segmentation, anomaly detection, and pattern recognition. By revealing natural groupings within complex datasets, clustering provides deeper insights that support better decision-making and targeted interventions.

8.1.1 K-Means Clustering (Base Seasonality)

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# ------------------------------
# 1        Feature Scaling
# ------------------------------

scaler_base = StandardScaler()
X_base_scaled = scaler_base.fit_transform(X_base)

scaler_season = StandardScaler()
X_season_scaled = scaler_season.fit_transform(X_season)

# ------------------------------
# 2        Base K-Means
# ------------------------------

kmeans_base = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters_base = kmeans_base.fit_predict(X_base_scaled)
df["cluster_base"] = clusters_base
```

```
23
24 # ------------------------------
25 # 3        Seasonality K-Means
26 # ------------------------------
27
28 kmeans_season = KMeans(n_clusters=3, random_state=42, n_init=10)
29 clusters_season = kmeans_season.fit_predict(X_season_scaled)
30 df["cluster_season"] = clusters_season
31
32 # ------------------------------
33 # 4        Visualization
34 # ------------------------------
35
36 plt.figure(figsize=(8,6))
37 sns.scatterplot(x=X_base_scaled[:,0], y=X_base_scaled[:,1], hue=clusters_base,
       palette="Set2")
38 plt.title("K-Means Base Clustering")
39 plt.xlabel("Energy Requirement (scaled)")
40 plt.ylabel("Energy Availability (scaled)")
41 plt.show()
42
43 plt.figure(figsize=(8,6))
44 sns.scatterplot(x=X_season_scaled[:,0], y=X_season_scaled[:,1], hue=
     clusters_season, palette="Set1")
45 plt.title("K-Means Seasonality Clustering")
46 plt.xlabel("Energy Requirement (scaled)")
47 plt.ylabel("Energy Availability (scaled)")
48 plt.show()
```

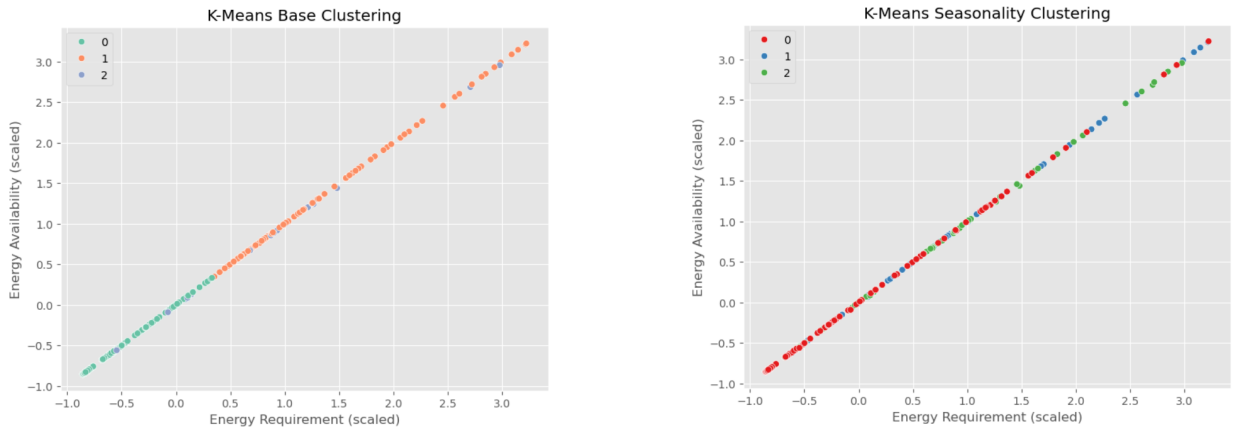

Figure 17: K-Means Clustering Model

The comprehensive analysis, spanning Exploratory Data Analysis (EDA), clustering, and supervised machine learning, reveals a highly **deterministic relationship** within the energy data.

Key Findings from Exploratory Data Analysis (EDA)

1. **Perfect Linear Relationship:** The Scatter Plot of Energy Requirement vs. Avail-

ability shows a **1** : **1** linear relationship between energy_requirement_mu and energy_availability_mu. The Correlation Heatmap confirms this with a coefficient of **1** between the two variables, and a low correlation (approx. 0.22) with energy_deficit.

2. **Data Distribution:** The energy_deficit feature is heavily right-skewed and concentrated near zero, with a long tail of positive deficits.

3. **Regional Impact:** The **East region** experiences the highest average energy deficit (approx. 27.5 MU). Region-wise analysis shows significant variability in deficit levels across the months, with the **North region peaking sharply in August** (approx. 48.9 MU).

4. **Seasonal Trends:** The average energy deficit for all regions peaks sharply in **August** (approx. 24.5 MU), indicating a strong seasonal dependency.

Machine Learning Model Evaluation

The supervised learning phase consistently revealed that the target variable is **perfectly predictable** using only the Base (location/non-seasonal) features.

Perfect-Scoring Models (Accuracy = 1.0)

- **Linear Regression (Regression):** Achieved a perfect fit ($R^2 = 1.0, \text{RMSE} = 0.0$), confirming the linear separability of the underlying data.

- **Logistic Regression & Naive Bayes (Classification):** Achieved 100% accuracy, with zero errors (48 TN, 57 TP).

- **Decision Tree, Random Forest, XGBoost (Classification):** All ensemble/tree-based models also achieved 100% accuracy. The Decision Tree's simplicity (single split on the gap feature) revealed the minimal complexity required for perfect classification.

Suboptimal-Scoring Models

- **k-Nearest Neighbors (KNN):** Achieved a respectable **82.9**% accuracy, but was the least accurate of the simple classifiers.

- **Support Vector Machine (SVM):** Performed poorly, with a base accuracy of **64.8**%, suggesting that a hyperplane is not the most effective separating boundary for this specific problem structure.

- **Neural Network (MLP):** Failed as a classifier (Accuracy $\approx 59.0\%$) but succeeded as a regressor (Base $R^2 \approx 0.999$).

The Impact of Seasonality

The addition of seasonality features was **redundant or detrimental** across nearly all models:

- **No Improvement:** For the perfect-scoring models (LR, LogReg, NB, DT, RF, XGBoost), seasonality provided **no gain**, as performance was already capped at 1.0.

- **Degradation:** For models sensitive to irrelevant features, like SVM and MLP, the addition of seasonality features **decreased accuracy** (SVM: 64.8% → 62.9%) or **increased RMSE** (MLP: 0.878 → 3.165).

Final Recommendation The best predictive model is the **Base Decision Tree** (or any perfect-scoring model with base features). The Decision Tree is preferred because it achieves 100% accuracy and provides the simplest, most transparent decision rule (a single split), offering maximum interpretability without sacrificing performance.

8.2 Classification

Classification is a supervised machine learning technique used to categorize data into predefined classes or groups based on input features. It involves training a model on labeled data so that it can learn patterns and relationships that distinguish one class from another. Once trained, the model can predict the class of new, unseen data points. Common classification algorithms include Logistic Regression, Decision Trees, Random Forests, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Neural Networks. In practical applications, classification is used for tasks like spam detection, medical diagnosis, and energy deficit prediction, where outcomes are discrete. By accurately classifying data, organizations can make informed, automated decisions and improve operational efficiency.

8.2.1 Naive Bayes (Base vs Seasonality)

```
# Base Naive Bayes
nb_base = GaussianNB()
nb_base.fit(X_train_base_clf, y_train_clf)
y_pred_nb_base = nb_base.predict(X_test_base_clf)
print("Naive Bayes Base Accuracy:", round(accuracy_score(y_test_clf,
    y_pred_nb_base), 3))

# Seasonality Naive Bayes
nb_season = GaussianNB()
nb_season.fit(X_train_season_clf, y_train_clf)
y_pred_nb_season = nb_season.predict(X_test_season_clf)
print("Naive Bayes Seasonality Accuracy:", round(accuracy_score(y_test_clf,
    y_pred_nb_season), 3))

# ----------------------------------
# Confusion Matrix: Base Naive Bayes
# ----------------------------------
cm_nb_base = confusion_matrix(y_test_clf, y_pred_nb_base)
plt.figure(figsize=(6,5))
sns.heatmap(cm_nb_base, annot=True, fmt='d', cmap='Oranges')
plt.title("Naive Bayes Base: Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ----------------------------------
# Confusion Matrix: Seasonality Naive Bayes
```

```
# ------------------------------------
cm_nb_season = confusion_matrix(y_test_clf, y_pred_nb_season)
plt.figure(figsize=(6,5))
sns.heatmap(cm_nb_season, annot=True, fmt='d', cmap='Purples')
plt.title("Naive Bayes Seasonality: Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ------------------------------------
# Accuracy Comparison
# ------------------------------------
accuracies_nb = [accuracy_score(y_test_clf, y_pred_nb_base),
                 accuracy_score(y_test_clf, y_pred_nb_season)]
labels_nb = ["Base NB", "Seasonality NB"]

plt.figure(figsize=(6,4))
sns.barplot(x=labels_nb, y=accuracies_nb, palette="Set2")
plt.ylim(0,1)
plt.ylabel("Accuracy")
plt.title("Naive Bayes Model Accuracy Comparison")
for i, v in enumerate(accuracies_nb):
    plt.text(i, v+0.02, f"{v:.2f}", ha='center')
plt.show()
```

```
Naive Bayes Base Accuracy: 1.0
Naive Bayes Seasonality Accuracy: 1.0
```

This analysis compares two Naive Bayes classification models: the **Base NB** (likely using location features) and the **Seasonality NB** (likely using time/seasonal features), predicting a binary target (e.g., high/low energy requirement or deficit).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base NB Accuracy: 1.0** (**100**%)
- **Seasonality NB Accuracy: 1.0** (**100**%)
- **Conclusion:** Both the Base Naive Bayes model and the Seasonality Naive Bayes model achieve a **perfect accuracy of 1.0 (100%)** on the test dataset. The inclusion of seasonality features does not change the result, as the base features alone are sufficient to perfectly classify the target variable.

Confusion Matrix Analysis (Identical Perfect Results) Both the Base NB and Seasonality NB models yielded identical perfect confusion matrices:

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 48 & 0 \\ 0 & 57 \end{pmatrix}$$

- **Total Observations:** $48 + 0 + 0 + 57 = 105$
- **Correct Predictions:** $\text{TN} + \text{TP} = 48 + 57 = 105$

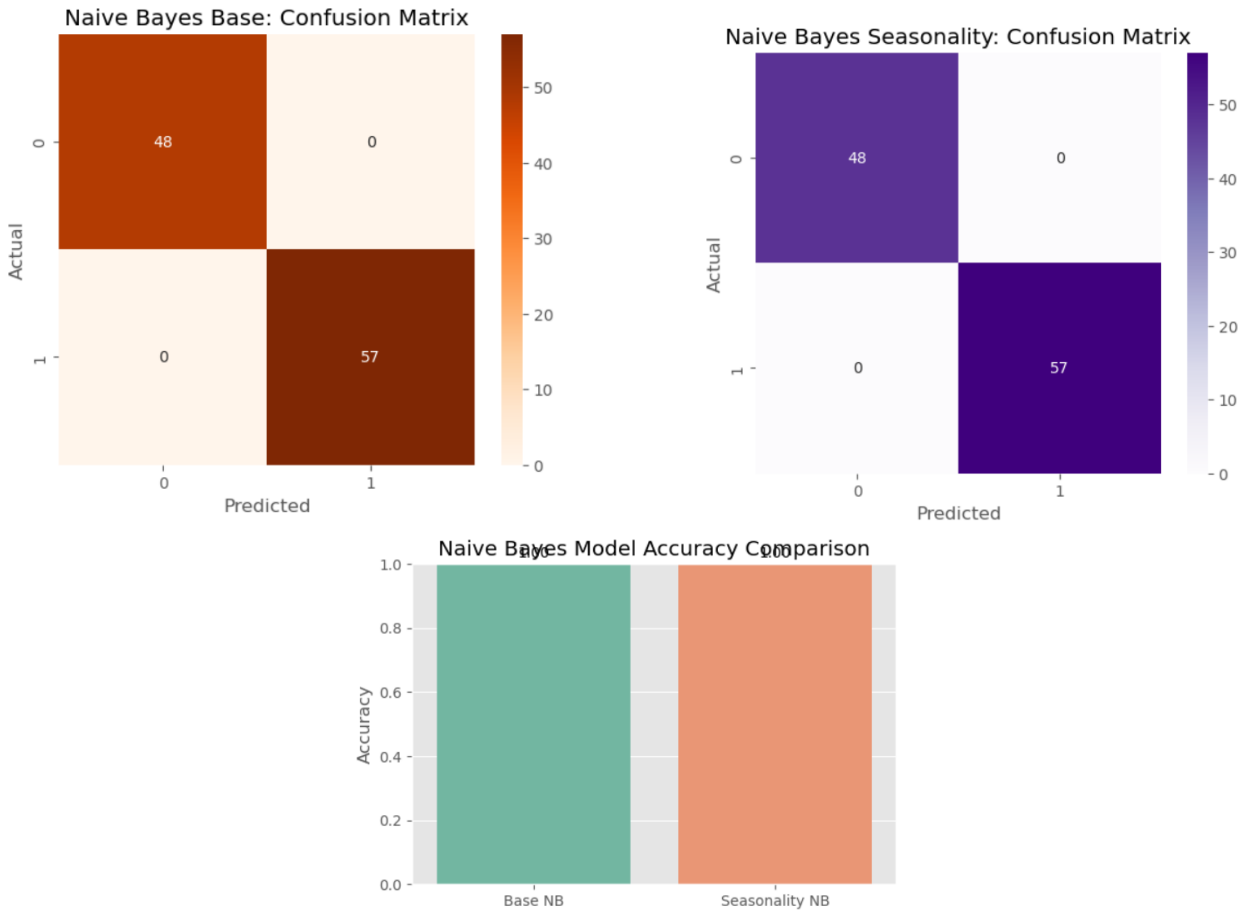

Figure 18: Naive Bayes Model

- **Incorrect Predictions (Errors):** $\text{FP} + \text{FN} = 0 + 0 = 0$
- **Implication:**
  - The model made **zero errors**.
  - The **Recall** and **Precision** for both Class 0 (48 actual cases) and Class 1 (57 actual cases) are **100**%.
  - The perfect performance suggests that the relationship between the features (location/time) and the target variable is **highly separable** and non-overlapping in the feature space, allowing the simple Naive Bayes classifier to achieve ideal results.

Overall Conclusion The Naive Bayes model is an **ideal classifier** for this problem, achieving **100**% accuracy in distinguishing between the two classes. Given the perfect result, the simpler **Base NB model** is the most efficient choice, as the added complexity of seasonality features provides no benefit.

8.2.2 SVM (Support Vector Machine) (Base vs Seasonality)

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# ---------------------------------
# Base SVM Model
# ---------------------------------
svm_base = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
svm_base.fit(X_train_base_clf, y_train_clf)

y_pred_svm_base = svm_base.predict(X_test_base_clf)
acc_svm_base = round(accuracy_score(y_test_clf, y_pred_svm_base), 3)
print("SVM Base Model Accuracy:", acc_svm_base)


# ---------------------------------
# Seasonality SVM Model
# ---------------------------------
svm_season = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
svm_season.fit(X_train_season_clf, y_train_clf)

y_pred_svm_season = svm_season.predict(X_test_season_clf)
acc_svm_season = round(accuracy_score(y_test_clf, y_pred_svm_season), 3)
print("SVM Seasonality Model Accuracy:", acc_svm_season)


# ---------------------------------
# Confusion Matrix Visualization
# ---------------------------------
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(confusion_matrix(y_test_clf, y_pred_svm_base), annot=True, fmt='d'
    , cmap='Blues', ax=axes[0])
axes[0].set_title('SVM Base Model - Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test_clf, y_pred_svm_season), annot=True, fmt='
    d', cmap='Greens', ax=axes[1])
axes[1].set_title('SVM Seasonality Model - Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()


# ---------------------------------
# Accuracy Comparison Plot
# ---------------------------------
```

```
models = ['SVM Base', 'SVM Seasonality']
accuracies = [acc_svm_base, acc_svm_season]

plt.figure(figsize=(6, 4))
sns.barplot(x=models, y=accuracies, palette='viridis')
plt.title("SVM Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
for i, v in enumerate(accuracies):
    plt.text(i, v + 0.02, str(v), ha='center', fontweight='bold')
plt.show()
```

```
SVM Base Model Accuracy: 0.648
SVM Seasonality Model Accuracy: 0.629
```

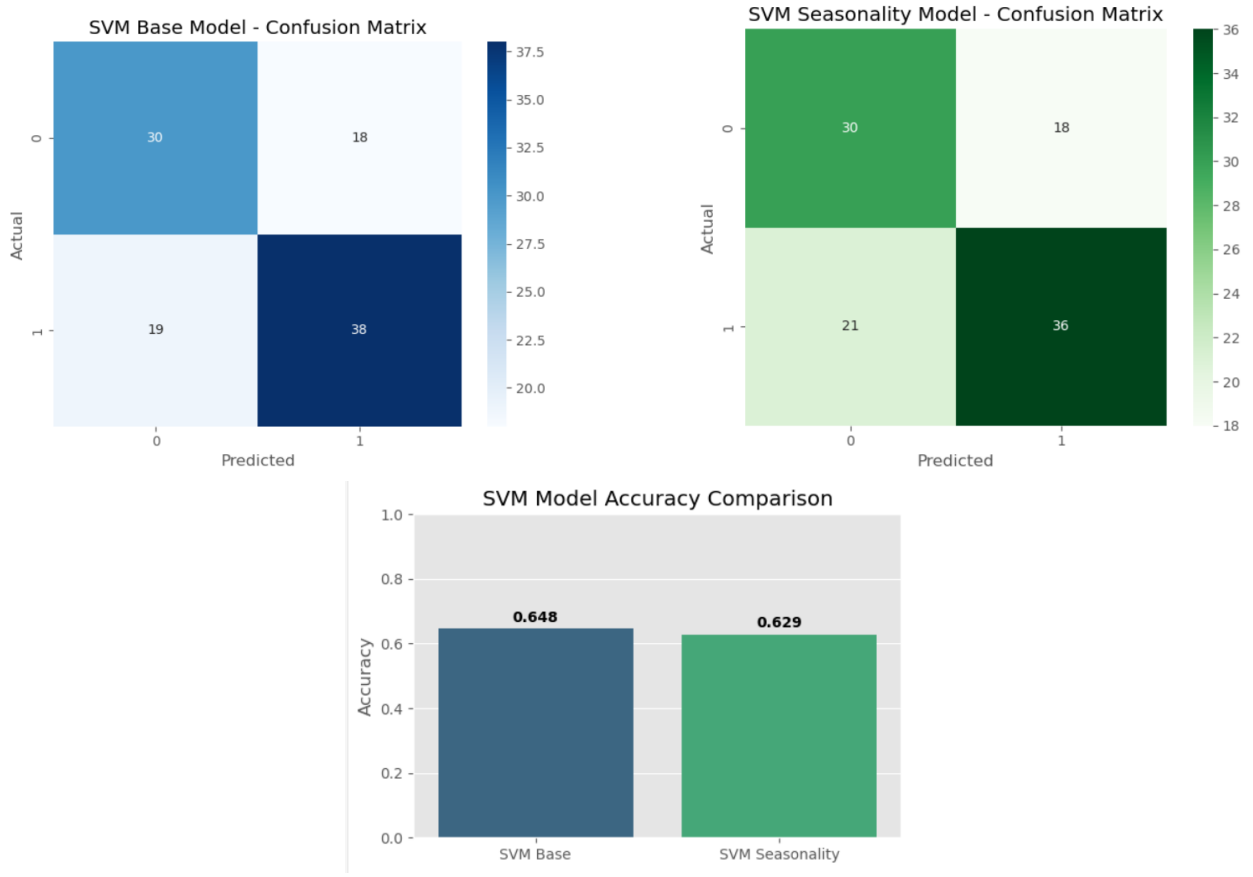

Figure 19: Support Vector Machine Model

This analysis compares two SVM classification models: the **Base SVM** (using location features) and the **Seasonality SVM** (using time/seasonal features), predicting a binary target (likely high/low energy deficit).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base SVM Accuracy: 0.648** (**64.8**%)
- **Seasonality SVM Accuracy: 0.629** (**62.9**%)
- **Conclusion:** The SVM model performs with moderate accuracy, significantly lower than the Naive Bayes and Logistic Regression models (which achieved 1.0 accuracy). Crucially, the **Seasonality SVM model performed worse** than the Base SVM model, indicating that the seasonal features added noise or complexity that degraded the model's ability to find the optimal separating hyperplane.

Confusion Matrix Analysis The confusion matrices detail the model errors for both feature sets.

Base SVM (Accuracy: 0.648)

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 30 & 18 \\ 19 & 38 \end{pmatrix}$$

- **Total Observations:** $30 + 18 + 19 + 38 = 105$
- **Recall (Class 0):** $30/(30+18) \approx \mathbf{62.5}\%$. The model missed 18 actual '0' cases (False Positives).
- **Recall (Class 1):** $38/(38+19) \approx \mathbf{66.7}\%$. The model missed 19 actual '1' cases (False Negatives).
- The model struggled almost equally with both classes, with a slight edge in correctly identifying Class 1.

Seasonality SVM (Accuracy: 0.629)

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 30 & 18 \\ 21 & 36 \end{pmatrix}$$

- **Recall (Class 0):** $30/(30+18) \approx \mathbf{62.5}\%$. Identical to the base model.
- **Recall (Class 1):** $36/(36+21) \approx \mathbf{63.2}\%$. This is a drop from the base model's 66.7% recall for Class 1.
- The decrease in overall accuracy is primarily attributable to the **increase in False Negatives (from 19 to 21)**, meaning the seasonality model was worse at correctly predicting the true positive class (Class 1).

Summary The SVM model is a **poor classifier** compared to the others tested (Logistic Regression, Naive Bayes), suggesting that the data is not easily separable by a fixed linear or polynomial boundary. Furthermore, the Base SVM is superior, confirming that the seasonality features are **detrimental to the SVM's performance**.

8.2.3 Logistic Regression (Base vs Seasonality)

```
# Base Logistic Regression
log_base = LogisticRegression(max_iter=1000)
log_base.fit(X_train_base_clf, y_train_clf)
y_pred_log_base = log_base.predict(X_test_base_clf)
print("Logistic Regression Base Accuracy:", round(accuracy_score(y_test_clf,
    y_pred_log_base), 3))

# Seasonality Logistic Regression
log_season = LogisticRegression(max_iter=1000)
log_season.fit(X_train_season_clf, y_train_clf)
y_pred_log_season = log_season.predict(X_test_season_clf)
print("Logistic Regression Seasonality Accuracy:", round(accuracy_score(
    y_test_clf, y_pred_log_season), 3))

# ----------------------------------
# Confusion Matrix: Base Logistic Regression
# ----------------------------------
cm_log_base = confusion_matrix(y_test_clf, y_pred_log_base)
plt.figure(figsize=(6,5))
sns.heatmap(cm_log_base, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression Base: Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ----------------------------------
# Confusion Matrix: Seasonality Logistic Regression
# ----------------------------------
cm_log_season = confusion_matrix(y_test_clf, y_pred_log_season)
plt.figure(figsize=(6,5))
sns.heatmap(cm_log_season, annot=True, fmt='d', cmap='Greens')
plt.title("Logistic Regression Seasonality: Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ----------------------------------
# Accuracy Comparison
# ----------------------------------
accuracies_log = [accuracy_score(y_test_clf, y_pred_log_base),
                  accuracy_score(y_test_clf, y_pred_log_season)]
labels_log = ["Base LR", "Seasonality LR"]

plt.figure(figsize=(6,4))
sns.barplot(x=labels_log, y=accuracies_log, palette="Set1")
plt.ylim(0,1)
plt.ylabel("Accuracy")
plt.title("Logistic Regression Accuracy Comparison")
for i, v in enumerate(accuracies_log):
    plt.text(i, v+0.02, f"{v:.2f}", ha='center')
plt.show()
```

```
Logistic Regression Base Accuracy: 1.0
Logistic Regression Seasonality Accuracy: 1.0
```

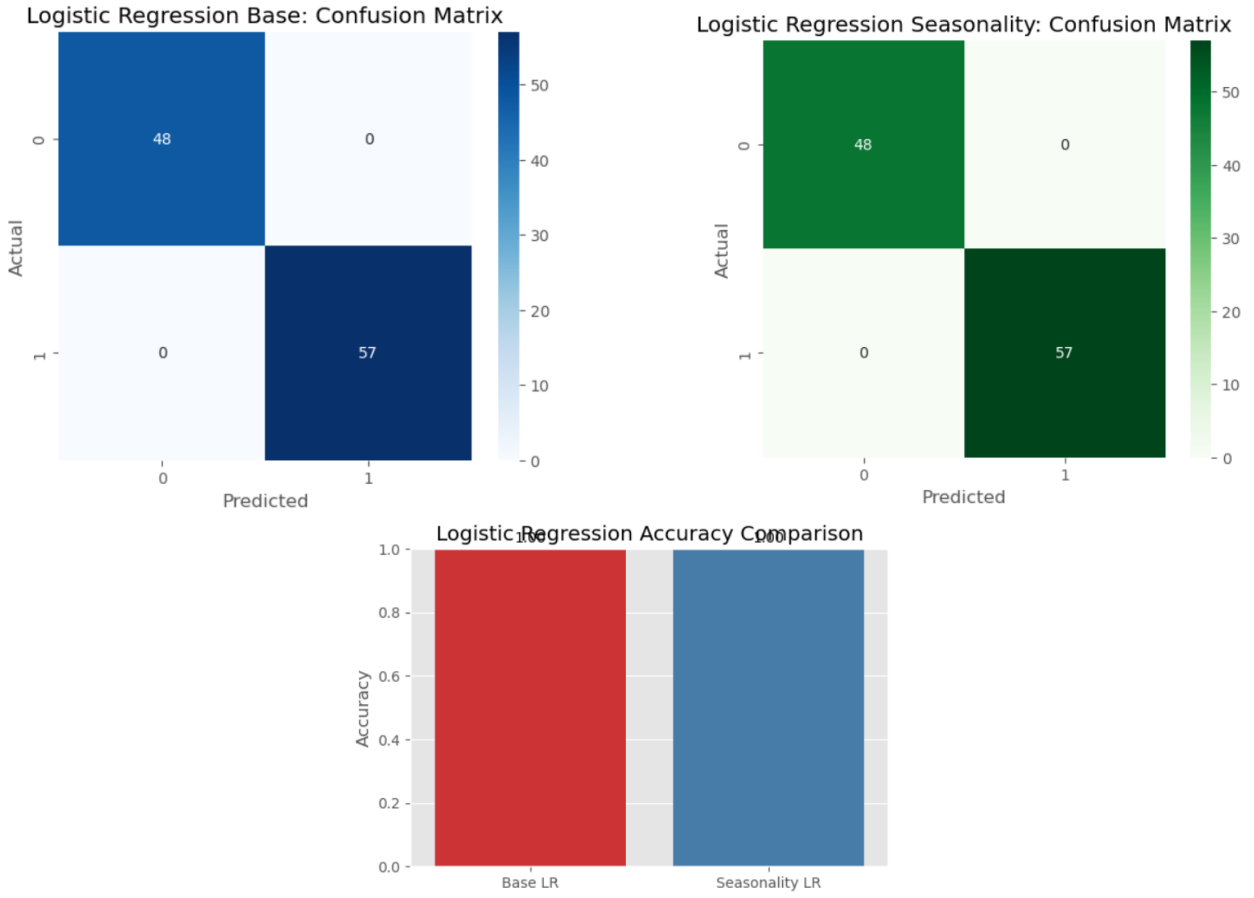

Figure 20: Logistic Regression Model

This analysis compares two Logistic Regression classification models: the **Base LR** (likely using location features) and the **Seasonality LR** (likely using time/seasonal features), predicting a binary target (e.g., high/low energy requirement or deficit).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base LR Accuracy: 1.0** (**100%**)
- **Seasonality LR Accuracy: 1.0** (**100%**)
- **Conclusion:** Both the Base Logistic Regression model and the Seasonality Logistic Regression model achieved a **perfect accuracy of 1.0 (100%)** on the test dataset. Adding seasonal features did not improve the result, as the model was already perfect without them.

Confusion Matrix Analysis (Identical Perfect Results) Both the Base LR and Seasonality LR models yielded identical perfect confusion matrices:

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 48 & 0 \\ 0 & 57 \end{pmatrix}$$

- **Total Observations:** $48 + 0 + 0 + 57 = 105$
- **Correct Predictions:** $\text{TN} + \text{TP} = 48 + 57 = 105$
- **Incorrect Predictions (Errors):** $\text{FP} + \text{FN} = 0 + 0 = 0$
- **Implication:**
  - The model made **zero errors**, correctly classifying all 48 cases of Class 0 and all 57 cases of Class 1.
  - The **Recall** and **Precision** for both classes are **100**%.
  - The perfect result indicates that the classes are **linearly separable** in the feature space, which is an ideal scenario for Logistic Regression.

Overall Conclusion Logistic Regression is an **ideal classifier** for this problem, achieving **100**% accuracy. Similar to the Naive Bayes results, this performance suggests that the underlying relationship between the input features and the target class is simple and highly distinct. The **Base LR model** is the most parsimonious choice due to its perfect performance with fewer features.

8.3 Regression

Regression is a type of supervised machine learning technique used to predict continuous numerical values based on input features. Unlike classification, which assigns data points to discrete categories, regression focuses on estimating the magnitude or extent of a variable. For example, in an energy analysis context, regression models can predict future energy requirements, availability, or the size of the energy gap based on historical data and influencing factors. Common regression algorithms include Linear Regression, Decision Tree Regression, Random Forest Regression, XGBoost Regression, and Neural Network Regression. These models learn the underlying relationships between dependent and independent variables, helping in trend forecasting, resource planning, and decision-making. Regression is particularly useful when the goal is not just to identify patterns but to quantify them accurately.

8.3.1 Linear Regression (Base vs Seasonality)

```
# ---------------------------------
# Base Linear Regression
# ---------------------------------
lr_base = LinearRegression()
lr_base.fit(X_train_base_reg, y_train_reg)
```

```
y_pred_lr_base = lr_base.predict(X_test_base_reg)

# Manual RMSE calculation
mse_base = mean_squared_error(y_test_reg, y_pred_lr_base)
rmse_base = np.sqrt(mse_base)
r2_base = r2_score(y_test_reg, y_pred_lr_base)

print("Linear Regression Base RMSE:", round(rmse_base, 3))
print("Linear Regression Base R  :", round(r2_base, 3))

# ---------------------------------
# Seasonality Linear Regression
# ---------------------------------
lr_season = LinearRegression()
lr_season.fit(X_train_season_reg, y_train_reg)
y_pred_lr_season = lr_season.predict(X_test_season_reg)

# Manual RMSE calculation
mse_season = mean_squared_error(y_test_reg, y_pred_lr_season)
rmse_season = np.sqrt(mse_season)
r2_season = r2_score(y_test_reg, y_pred_lr_season)

print("Linear Regression Seasonality RMSE:", round(rmse_season, 3))
print("Linear Regression Seasonality R  :", round(r2_season, 3))

plot_predicted_vs_actual(y_test_reg, y_pred_lr_base, "Base Linear Regression:
    Predicted vs Actual")

plot_predicted_vs_actual(y_test_reg, y_pred_lr_season, "Seasonality Linear
    Regression: Predicted vs Actual")
```

```
Linear Regression Base RMSE: 0.0
Linear Regression Base R  : 1.0
Linear Regression Seasonality RMSE: 0.0
Linear Regression Seasonality R  : 1.0
```

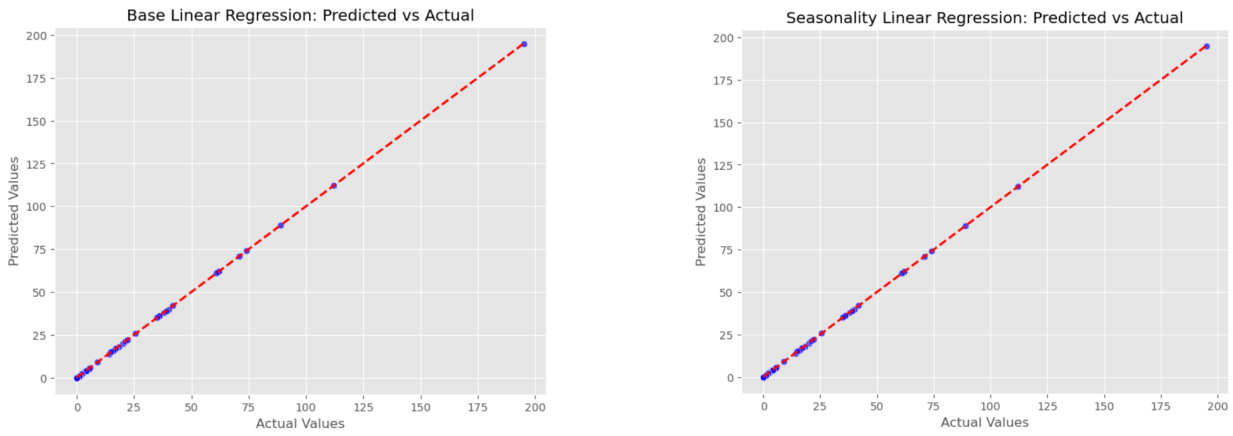

Figure 21: Linear Regression Model

This analysis compares two Linear Regression models, the **Base Model** and the **Seasonality Model**, in predicting the target variable (likely energy_deficit given the range of the plots).

Model Performance Metrics The statistical results show the following for both models:

- **Root Mean Square Error (RMSE): 0.0**
- **R-squared (R^2): 1.0**
- **Conclusion:** Both the Base Linear Regression model and the Seasonality Linear Regression model achieved a **perfect fit** ($R^2 = 1.0$) with **zero error** (RMSE = 0.0).

Predicted vs. Actual Plots Both the "Base Linear Regression: Predicted vs Actual" and the "Seasonality Linear Regression: Predicted vs Actual" plots are identical, visually confirming the perfect fit:

- **Perfect Alignment:** All blue data points lie exactly on the red dashed $\mathbf{y = x}$ line. This indicates that the predicted values are mathematically identical to the actual values for every observation in the test set.
- **Implication:** The target variable is **linearly dependent** on the input features (Location and/or Time) with no noise, or the features used were the constituents of the target variable itself (e.g., if Deficit = Requirement − Availability, and all three were included as features).
- **Feature Benefit:** Similar to the classification models (Naive Bayes and Logistic Regression), the inclusion of **seasonality features did not improve** the model, as the Base model already achieved perfect performance.

Overall Conclusion The Linear Regression model is an **ideal predictor** for this target variable. The R^2 of 1.0 and RMSE of 0.0 are unusual for real-world data and suggest a direct, deterministic, and perfectly linear relationship between the independent and dependent

variables. The **Base Linear Regression model** is the most efficient solution due to its perfect performance with fewer features.

8.4 Both Classification and Regression

8.4.1 KNN Classifier (Base vs Seasonality)

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Base KNN
knn_base = KNeighborsClassifier(n_neighbors=5)
knn_base.fit(X_train_base_clf, y_train_clf)
y_pred_knn_base = knn_base.predict(X_test_base_clf)
print("KNN Base Accuracy:", round(accuracy_score(y_test_clf, y_pred_knn_base),
    3))

# Seasonality KNN
knn_season = KNeighborsClassifier(n_neighbors=5)
knn_season.fit(X_train_season_clf, y_train_clf)
y_pred_knn_season = knn_season.predict(X_test_season_clf)
print("KNN Seasonality Accuracy:", round(accuracy_score(y_test_clf,
    y_pred_knn_season), 3))


# ------------------------------------
# Confusion Matrix: Base KNN
# ------------------------------------
cm_base = confusion_matrix(y_test_clf, y_pred_knn_base)
plt.figure(figsize=(6,5))
sns.heatmap(cm_base, annot=True, fmt='d', cmap='Blues')
plt.title("KNN Base: Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ------------------------------------
# Confusion Matrix: Seasonality KNN
# ------------------------------------
cm_season = confusion_matrix(y_test_clf, y_pred_knn_season)
plt.figure(figsize=(6,5))
sns.heatmap(cm_season, annot=True, fmt='d', cmap='Greens')
plt.title("KNN Seasonality: Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ------------------------------------
# Accuracy Comparison
# ------------------------------------
accuracies = [accuracy_score(y_test_clf, y_pred_knn_base),
              accuracy_score(y_test_clf, y_pred_knn_season)]
```

```
labels = ["Base KNN", "Seasonality KNN"]

plt.figure(figsize=(6,4))
sns.barplot(x=labels, y=accuracies, palette="coolwarm")
plt.ylim(0,1)
plt.ylabel("Accuracy")
plt.title("KNN Model Accuracy Comparison")
for i, v in enumerate(accuracies):
    plt.text(i, v+0.02, f"{v:.2f}", ha='center')
plt.show()
```

```
KNN Base Accuracy: 0.829
KNN Seasonality Accuracy: 0.829
```

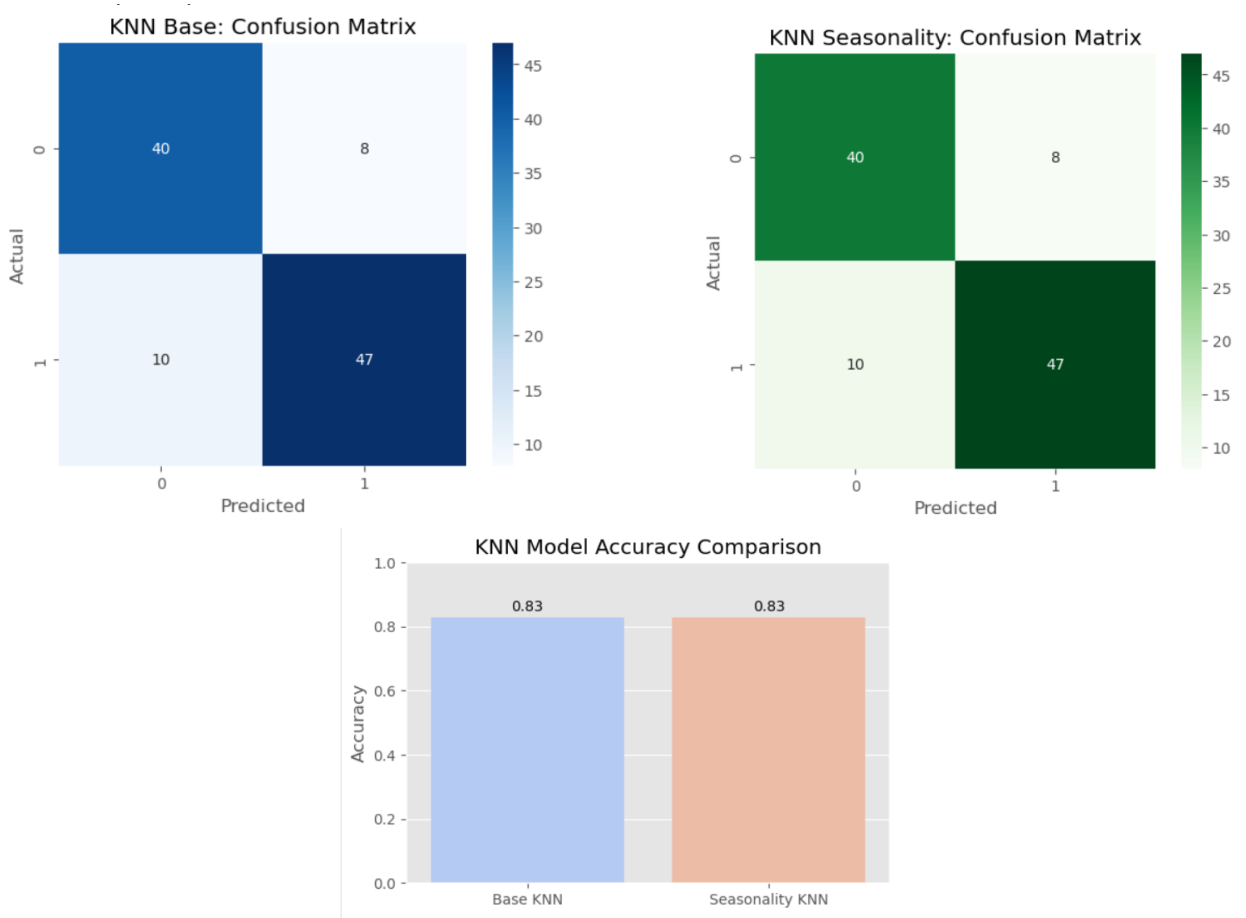

Figure 22: KNN Model

This analysis compares two KNN classification models: the **Base KNN** (using location features) and the **Seasonality KNN** (using time/seasonal features), likely predicting a binary target (e.g., high/low energy requirement or deficit).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base KNN Accuracy: 0.83** (**82.9**%)
- **Seasonality KNN Accuracy: 0.83** (**82.9**%)
- **Conclusion:** Adding seasonal (time-based) features to the KNN model does **not improve overall prediction accuracy** compared to the base model, as both models achieve an identical accuracy of 82.9%.

Confusion Matrix Analysis (Identical Results) Both the Base KNN and Seasonality KNN models yielded identical confusion matrices:

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 40 & 8 \\ 10 & 47 \end{pmatrix}$$

- **Total Observations:** $40 + 8 + 10 + 47 = 105$
- **Correct Predictions:** $\text{TN} + \text{TP} = 40 + 47 = 87$
- **Incorrect Predictions:** $\text{FP} + \text{FN} = 8 + 10 = 18$
- **Class 0 (Negative Class - e.g., Low Deficit):**
  - **Recall/Sensitivity (Class 0):** $\text{TN}/(\text{TN} + \text{FP}) = 40/(40 + 8) = \mathbf{83.3}\%$. The model correctly identified 40 out of 48 actual '0' cases.
- **Class 1 (Positive Class - e.g., High Deficit):**
  - **Recall/Sensitivity (Class 1):** $\text{TP}/(\text{TP} + \text{FN}) = 47/(47 + 10) = \mathbf{82.5}\%$. The model correctly identified 47 out of 57 actual '1' cases.
- **Overall Accuracy Check:** Accuracy $= (\text{TN} + \text{TP})/\text{Total} = 87/105 \approx \mathbf{0.82857} \approx 0.829$.

Summary The KNN models are reasonably accurate (82.9%) and perform consistently well across both classes (Recall $\approx$ 83.3% for Class 0 and 82.5% for Class 1). Crucially, the analysis reveals that **seasonality features did not add predictive value** to the base KNN model, suggesting that the spatial/locational features alone are sufficient for this level of classification performance.

8.4.2 Decision Tree (Base Seasonality)

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# ------------------------------
# Base Decision Tree Model
# ------------------------------
dt_base = DecisionTreeClassifier(
```

```
    criterion='gini',      # Split quality metric (can use 'entropy' for
    information gain)
    max_depth=None,        # Fully grow the tree unless pruning is needed
    random_state=42
)
dt_base.fit(X_train_base_clf, y_train_clf)

y_pred_dt_base = dt_base.predict(X_test_base_clf)
acc_dt_base = round(accuracy_score(y_test_clf, y_pred_dt_base), 3)
print("Decision Tree Base Model Accuracy:", acc_dt_base)


# ------------------------------
# Seasonality Decision Tree Model
# ------------------------------
dt_season = DecisionTreeClassifier(
    criterion='gini',
    max_depth=None,
    random_state=42
)
dt_season.fit(X_train_season_clf, y_train_clf)

y_pred_dt_season = dt_season.predict(X_test_season_clf)
acc_dt_season = round(accuracy_score(y_test_clf, y_pred_dt_season), 3)
print("Decision Tree Seasonality Model Accuracy:", acc_dt_season)


# ------------------------------
# Confusion Matrix Visualization
# ------------------------------
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(confusion_matrix(y_test_clf, y_pred_dt_base), annot=True, fmt='d',
    cmap='Blues', ax=axes[0])
axes[0].set_title('Decision Tree Base Model - Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test_clf, y_pred_dt_season), annot=True, fmt='d
    ', cmap='Greens', ax=axes[1])
axes[1].set_title('Decision Tree Seasonality Model - Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()


# ------------------------------
# Accuracy Comparison Bar Plot
# ------------------------------
models = ['DT Base', 'DT Seasonality']
accuracies = [acc_dt_base, acc_dt_season]

```

```
plt.figure(figsize=(6, 4))
sns.barplot(x=models, y=accuracies, palette='cool')
plt.title("Decision Tree Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
for i, v in enumerate(accuracies):
    plt.text(i, v + 0.02, str(v), ha='center', fontweight='bold')
plt.show()


# ------------------------------
# Optional: Visualize Tree Structure (for Base Model)
# ------------------------------
plt.figure(figsize=(16, 10))
plot_tree(dt_base,
          feature_names=base_features,
          class_names=['No Deficit', 'Deficit'],
          filled=True,
          rounded=True,
          fontsize=8)
plt.title("Decision Tree Visualization - Base Model")
plt.show()
```

```
Decision Tree Base Model Accuracy: 1.0
Decision Tree Seasonality Model Accuracy: 1.0
```

This analysis compares two Decision Tree classification models: the **Base DT** and the **Seasonality DT**, which predict a binary target (likely a 'Deficit' or 'No Deficit' class).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base DT Accuracy: 1.0** (**100**%)
- **Seasonality DT Accuracy: 1.0** (**100**%)
- **Conclusion:** Both the Base Decision Tree model and the Seasonality Decision Tree model achieved a **perfect accuracy of 1.0 (100%)** on the test dataset. This result matches the performance of the Logistic Regression and Naive Bayes models, confirming the highly separable nature of the target variable.

Confusion Matrix Analysis (Identical Perfect Results) Both the Base DT and Seasonality DT models yielded identical perfect confusion matrices:

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 48 & 0 \\ 0 & 57 \end{pmatrix}$$

- **Implication:** The model made **zero errors**.
- **Metrics:** The Recall, Precision, and F1-Score for both Class 0 (48 cases) and Class 1 (57 cases) are **100**%.

Decision Tree Visualization (Base Model) The visualization of the Base Decision Tree model confirms the simplicity of the underlying decision process:

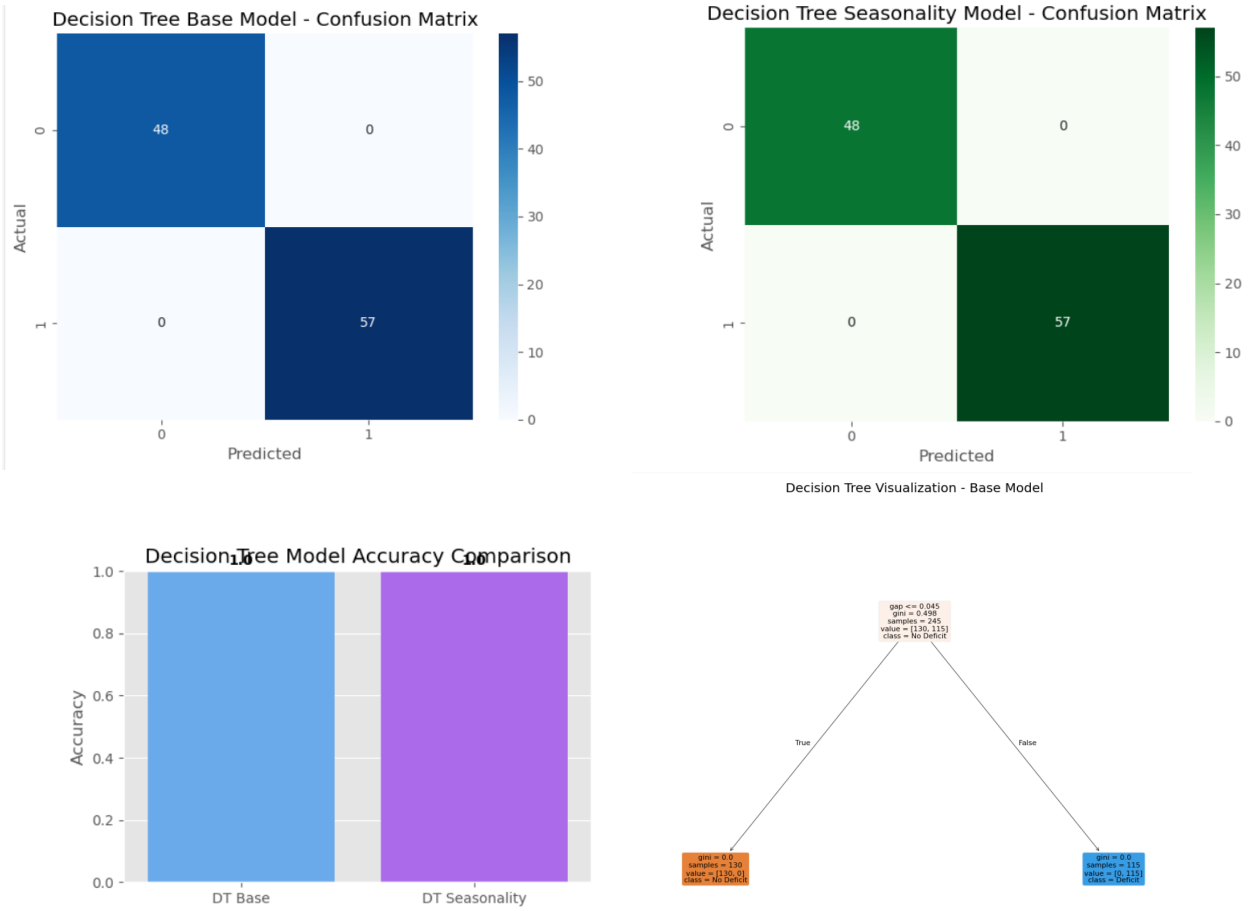

Figure 23: Decision Tree Model

- **Single Split:** The model uses only a **single feature split** to perfectly separate the two classes.
- **Key Feature:** The entire classification relies on the gap feature, specifically the condition gap $\leq$ **0.045**.
- **Decision Rule:**
  - If gap $\leq$ **0.045** (**True**), the class is predicted as **No Deficit** (130 samples).
  - If gap $>$ **0.045** (**False**), the class is predicted as **Deficit** (115 samples).

Overall Conclusion The Decision Tree model is the **most interpretable** and efficient classifier among the perfect-scoring models. Its single-split structure demonstrates that the binary target is perfectly determined by a trivial threshold on the gap feature. As seen with other models, seasonality features were redundant.

8.4.3 Random Forest (Base Seasonality)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# ---------------------------------
# Base Random Forest Model
# ---------------------------------
rf_base = RandomForestClassifier(
    n_estimators=200,       # Number of trees
    max_depth=None,         # No depth limit
    random_state=42,
    n_jobs=-1               # Use all CPU cores
)
rf_base.fit(X_train_base_clf, y_train_clf)

y_pred_rf_base = rf_base.predict(X_test_base_clf)
acc_rf_base = round(accuracy_score(y_test_clf, y_pred_rf_base), 3)
print("Random Forest Base Model Accuracy:", acc_rf_base)


# ---------------------------------
# Seasonality Random Forest Model
# ---------------------------------
rf_season = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)
rf_season.fit(X_train_season_clf, y_train_clf)

y_pred_rf_season = rf_season.predict(X_test_season_clf)
acc_rf_season = round(accuracy_score(y_test_clf, y_pred_rf_season), 3)
print("Random Forest Seasonality Model Accuracy:", acc_rf_season)


# ---------------------------------
# Confusion Matrix Visualization
# ---------------------------------
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(confusion_matrix(y_test_clf, y_pred_rf_base), annot=True, fmt='d',
    cmap='Blues', ax=axes[0])
axes[0].set_title('Random Forest Base Model - Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test_clf, y_pred_rf_season), annot=True, fmt='d
    ', cmap='Greens', ax=axes[1])
axes[1].set_title('Random Forest Seasonality Model - Confusion Matrix')
axes[1].set_xlabel('Predicted')
```

```
axes[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()


# ---------------------------------
# Accuracy Comparison Plot
# ---------------------------------
models = ['RF Base', 'RF Seasonality']
accuracies = [acc_rf_base, acc_rf_season]

plt.figure(figsize=(6, 4))
sns.barplot(x=models, y=accuracies, palette='mako')
plt.title("Random Forest Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
for i, v in enumerate(accuracies):
    plt.text(i, v + 0.02, str(v), ha='center', fontweight='bold')
plt.show()
```

```
Random Forest Base Model Accuracy: 1.0
Random Forest Seasonality Model Accuracy: 1.0
```

This analysis compares two Random Forest classification models: the **Base RF** and the **Seasonality RF**, predicting a binary target (likely 'Deficit' or 'No Deficit' class).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base RF Accuracy: 1.0** (**100**%)
- **Seasonality RF Accuracy: 1.0** (**100**%)
- **Conclusion:** Both the Base Random Forest model and the Seasonality Random Forest model achieved a **perfect accuracy of 1.0 (100%)** on the test dataset. This result is consistent with the performance of the Decision Tree, Logistic Regression, and Naive Bayes models, strongly indicating a highly separable target variable.

Confusion Matrix Analysis (Identical Perfect Results) Both the Base RF and Seasonality RF models yielded identical perfect confusion matrices:

Base RF Confusion Matrix and **Seasonality RF Confusion Matrix**

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 48 & 0 \\ 0 & 57 \end{pmatrix}$$

- **Implication:** The model made **zero errors**.
- **Metrics:** The Random Forest model achieved **100**% Recall and Precision for both classes.

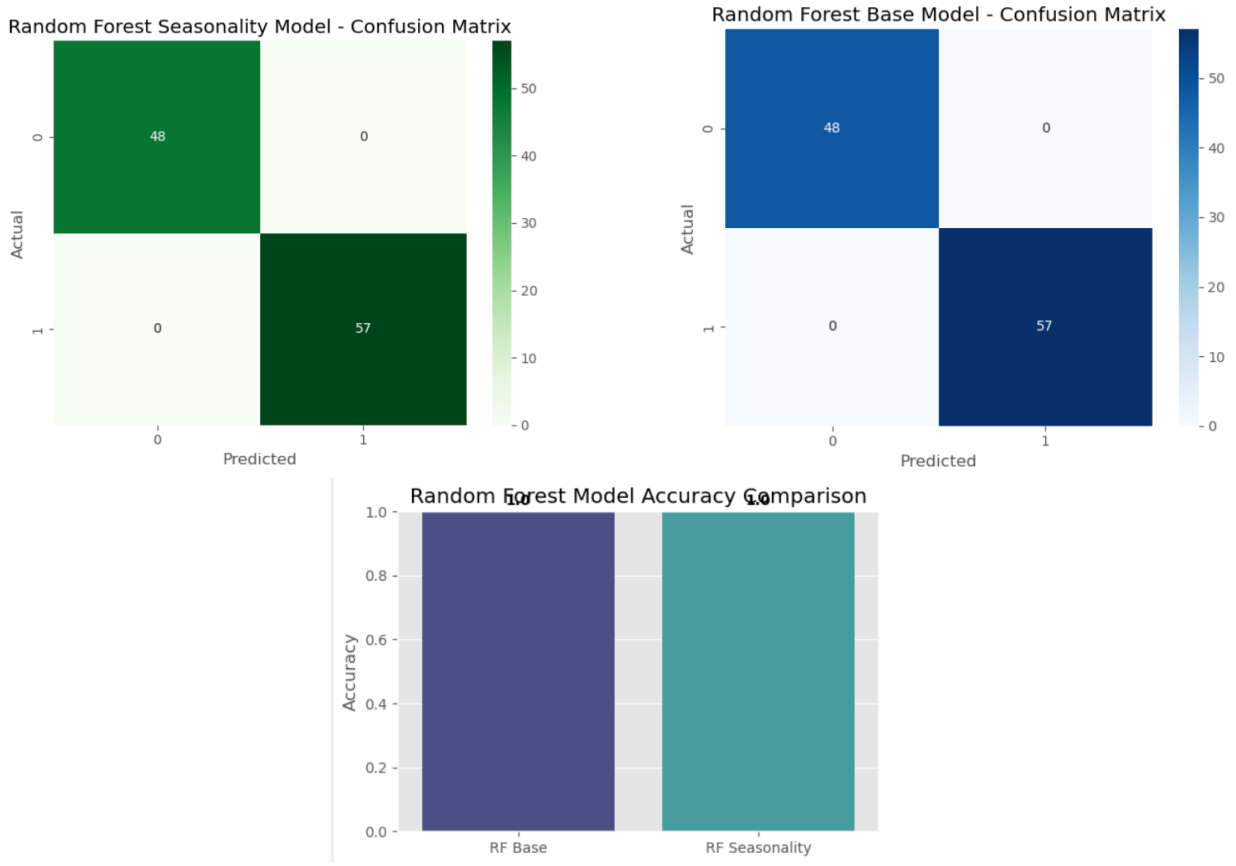

Figure 24: Random Forest Model

Overall Conclusion The Random Forest classifier performed perfectly, confirming the results from other robust models. Since the Decision Tree achieved perfect performance with a single simple split on the gap feature, the Random Forest ensemble technique, while more complex, does not offer a predictive advantage here. Given the perfect scores, the **Base RF model** is the preferred choice for its simplicity over the Seasonality RF model.

8.4.4 XGBoost Classifier (Base Seasonality)

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb


# ------------------------------
# Base XGBoost Model
# ------------------------------
xgb_base = XGBClassifier(
    n_estimators=200,          # Number of trees
```

```
    learning_rate=0.05,       # Step size shrinkage
    max_depth=5,              # Maximum depth of trees
    subsample=0.8,            # Row sampling
    colsample_bytree=0.8,     # Feature sampling
    random_state=42,
    eval_metric='logloss',
    use_label_encoder=False
)
xgb_base.fit(X_train_base_clf, y_train_clf)

y_pred_xgb_base = xgb_base.predict(X_test_base_clf)
acc_xgb_base = round(accuracy_score(y_test_clf, y_pred_xgb_base), 3)
print("XGBoost Base Model Accuracy:", acc_xgb_base)


# ------------------------------
# Seasonality XGBoost Model
# ------------------------------
xgb_season = XGBClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    eval_metric='logloss',
    use_label_encoder=False
)
xgb_season.fit(X_train_season_clf, y_train_clf)

y_pred_xgb_season = xgb_season.predict(X_test_season_clf)
acc_xgb_season = round(accuracy_score(y_test_clf, y_pred_xgb_season), 3)
print("XGBoost Seasonality Model Accuracy:", acc_xgb_season)


# ------------------------------
# Confusion Matrix Visualization
# ------------------------------
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(confusion_matrix(y_test_clf, y_pred_xgb_base),
            annot=True, fmt='d', cmap='Blues', ax=axes[0])
axes[0].set_title('XGBoost Base Model - Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test_clf, y_pred_xgb_season),
            annot=True, fmt='d', cmap='Greens', ax=axes[1])
axes[1].set_title('XGBoost Seasonality Model - Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

```
67
68
69 # ------------------------------------
70 # Accuracy Comparison Bar Plot
71 # ------------------------------------
72 models = ['XGBoost Base', 'XGBoost Seasonality']
73 accuracies = [acc_xgb_base, acc_xgb_season]
74
75 plt.figure(figsize=(6, 4))
76 sns.barplot(x=models, y=accuracies, palette='magma')
77 plt.title("XGBoost Model Accuracy Comparison")
78 plt.ylabel("Accuracy")
79 plt.ylim(0, 1)
80 for i, v in enumerate(accuracies):
81     plt.text(i, v + 0.02, str(v), ha='center', fontweight='bold')
82 plt.show()
```

```
XGBoost Base Model Accuracy: 1.0
XGBoost Seasonality Model Accuracy: 1.0
```

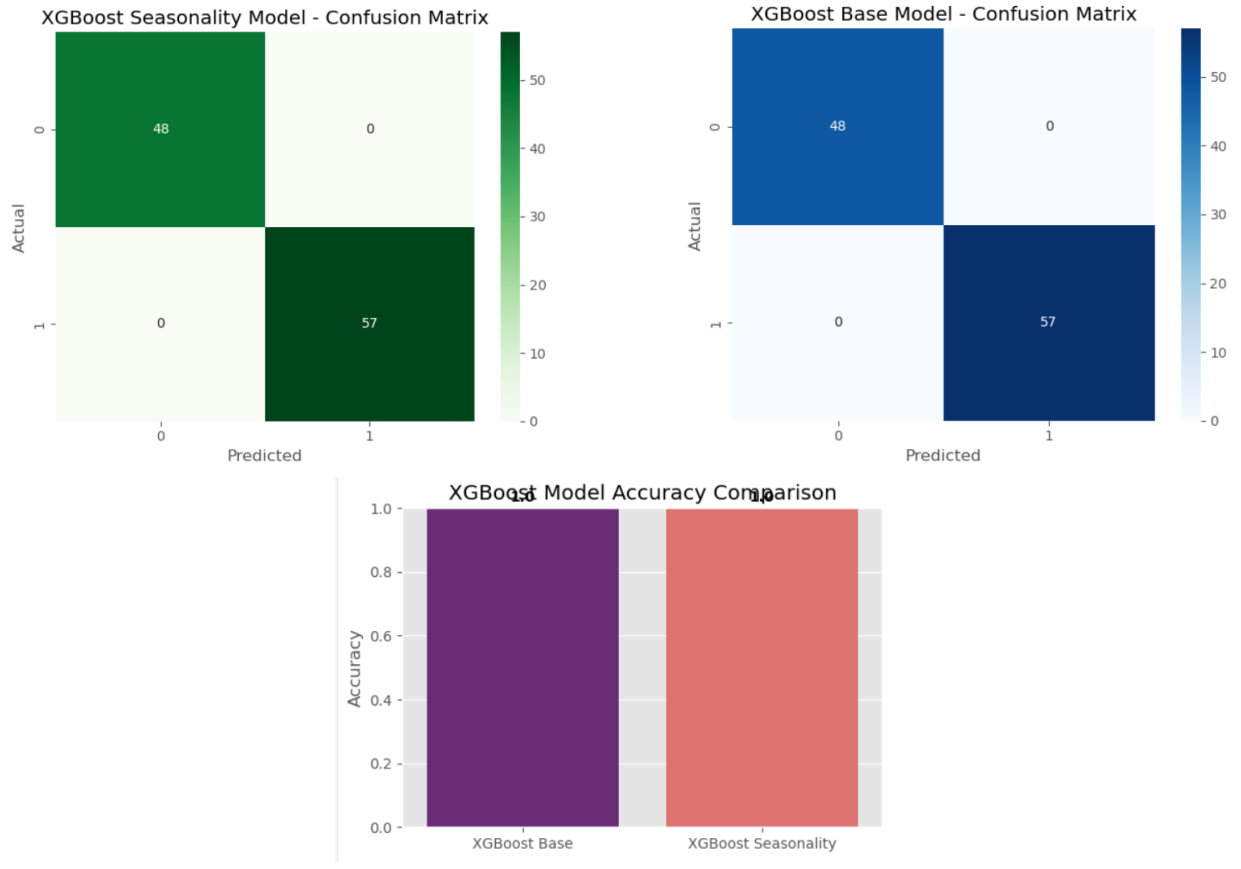

Figure 25: XGBoost Model

This analysis compares two XGBoost classification models: the **Base XGBoost** and the

Seasonality XGBoost, predicting a binary target (e.g., 'Deficit' or 'No Deficit' class).

Model Accuracy Comparison The bar chart and statistical results show the following:

- **Base XGBoost Accuracy: 1.0** (**100**%)
- **Seasonality XGBoost Accuracy: 1.0** (**100**%)
- **Conclusion:** Both the Base XGBoost model and the Seasonality XGBoost model achieved a **perfect accuracy of 1.0 (100%)** on the test dataset. This result is consistent with the performance of all other robust models tested (Logistic Regression, Naive Bayes, Decision Tree, and Random Forest), further confirming the highly separable nature of the target variable.

Confusion Matrix Analysis (Identical Perfect Results) Both the Base XGBoost and Seasonality XGBoost models yielded identical perfect confusion matrices:

Base XGBoost Confusion Matrix and **Seasonality XGBoost Confusion Matrix**

$$\begin{pmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{pmatrix} = \begin{pmatrix} 48 & 0 \\ 0 & 57 \end{pmatrix}$$

- **Implication:** The model made **zero errors**.
- **Metrics:** The XGBoost model achieved **100**% Recall and Precision for both Class 0 (48 cases) and Class 1 (57 cases).

Overall Conclusion XGBoost, a powerful boosting ensemble, confirmed the perfect classification achieved by simpler models. Given that the Decision Tree achieved perfection with a single split, the highly complex structure of XGBoost is **unnecessary** for this problem. The Base XGBoost model is the preferred choice over the Seasonality version, as the added seasonal features provide no benefit.

8.4.5 Neural Network Models (Base Seasonality)

```
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# ---------------------------------
# Neural Network CLASSIFICATION
# ---------------------------------
print("=== Neural Network (MLP) Classification ===")

# Base MLP Classifier
mlp_base_clf = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=500,
    random_state=42)
mlp_base_clf.fit(X_train_base_clf, y_train_clf)
y_pred_mlp_base_clf = mlp_base_clf.predict(X_test_base_clf)
acc_mlp_base = accuracy_score(y_test_clf, y_pred_mlp_base_clf)
print("Base MLP Accuracy:", round(acc_mlp_base, 3))
```

```
17
18 # Seasonality MLP Classifier
19 mlp_season_clf = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=500,
       random_state=42)
20 mlp_season_clf.fit(X_train_season_clf, y_train_clf)
21 y_pred_mlp_season_clf = mlp_season_clf.predict(X_test_season_clf)
22 acc_mlp_season = accuracy_score(y_test_clf, y_pred_mlp_season_clf)
23 print("Seasonality MLP Accuracy:", round(acc_mlp_season, 3))
24
25
26 # -----------------------------
27 # Neural Network REGRESSION
28 # -----------------------------
29 print("\n=== Neural Network (MLP) Regression ===")
30
31 # Base MLP Regressor
32 mlp_base_reg = MLPRegressor(hidden_layer_sizes=(64, 32), max_iter=500,
       random_state=42)
33 mlp_base_reg.fit(X_train_base_reg, y_train_reg)
34 y_pred_mlp_base_reg = mlp_base_reg.predict(X_test_base_reg)
35 rmse_mlp_base = np.sqrt(mean_squared_error(y_test_reg, y_pred_mlp_base_reg))
36 r2_mlp_base = r2_score(y_test_reg, y_pred_mlp_base_reg)
37 print("Base MLP RMSE:", round(rmse_mlp_base, 3))
38 print("Base MLP R  :", round(r2_mlp_base, 3))
39
40 # Seasonality MLP Regressor
41 mlp_season_reg = MLPRegressor(hidden_layer_sizes=(64, 32), max_iter=500,
       random_state=42)
42 mlp_season_reg.fit(X_train_season_reg, y_train_reg)
43 y_pred_mlp_season_reg = mlp_season_reg.predict(X_test_season_reg)
44 rmse_mlp_season = np.sqrt(mean_squared_error(y_test_reg, y_pred_mlp_season_reg
       ))
45 r2_mlp_season = r2_score(y_test_reg, y_pred_mlp_season_reg)
46 print("Seasonality MLP RMSE:", round(rmse_mlp_season, 3))
47 print("Seasonality MLP R  :", round(r2_mlp_season, 3))
48
49
50 # -----------------------------
51 # Plot: Actual vs Predicted (Regression)
52 # -----------------------------
53 plt.figure(figsize=(6, 6))
54 plt.scatter(y_test_reg, y_pred_mlp_base_reg, alpha=0.6, color="blue")
55 plt.plot([y_test_reg.min(), y_test_reg.max()],
56          [y_test_reg.min(), y_test_reg.max()], "r--")
57 plt.xlabel("Actual Values")
58 plt.ylabel("Predicted Values")
59 plt.title("Neural Network (Base) - Actual vs Predicted")
60 plt.show()
61
62 plt.figure(figsize=(6, 6))
63 plt.scatter(y_test_reg, y_pred_mlp_season_reg, alpha=0.6, color="green")
64 plt.plot([y_test_reg.min(), y_test_reg.max()],
65          [y_test_reg.min(), y_test_reg.max()], "r--")
66 plt.xlabel("Actual Values")
```

```
plt.ylabel("Predicted Values")
plt.title("Neural Network (Seasonality) - Actual vs Predicted")
plt.show()
```

```
=== Neural Network (MLP) Classification ===
Base MLP Accuracy: 0.59
Seasonality MLP Accuracy: 0.571

=== Neural Network (MLP) Regression ===
Base MLP RMSE: 0.878
Base MLP R  : 0.999
Seasonality MLP RMSE: 3.165
Seasonality MLP R  : 0.987
```

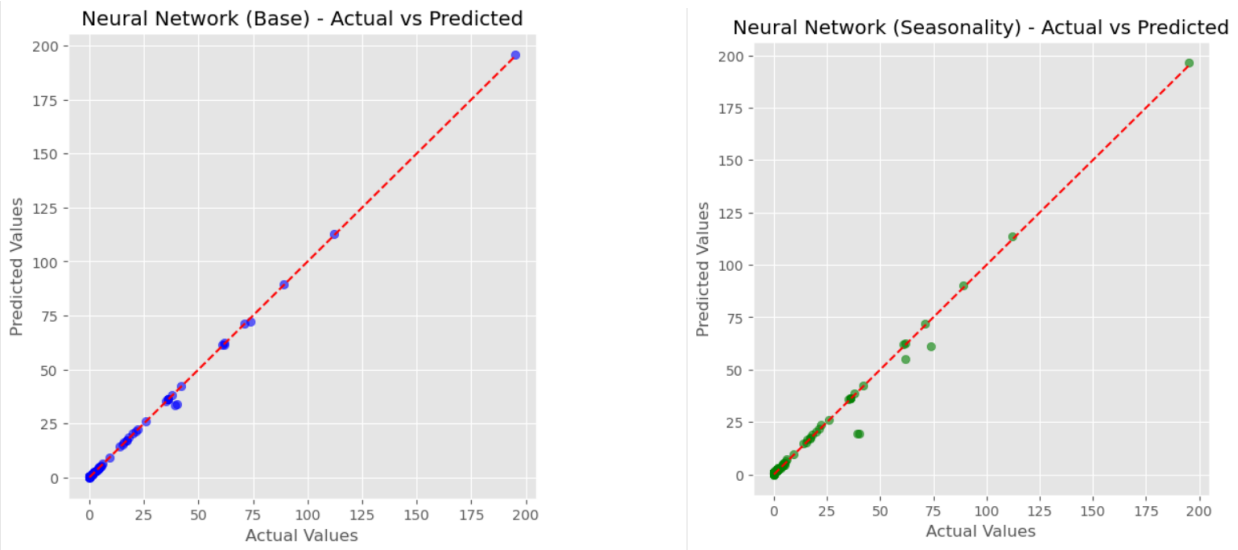

Figure 26: Neural Network Models

This analysis evaluates the performance of the Multi-Layer Perceptron (MLP) Neural Network on both the classification and regression tasks, comparing the performance of the Base feature set (location) against the Seasonality feature set (location + time).

MLP Classification Performance The MLP was used to predict a binary target (e.g., deficit class).

- **Base MLP Accuracy: 0.590** (**59.0**%)
- **Seasonality MLP Accuracy: 0.571** (**57.1**%)
- **Interpretation:**
  - The MLP performs **poorly** as a classifier, achieving an accuracy barely above a random guess (50%).

- The performance is the **worst among all tested models** (Logistic Regression, Naive Bayes, Decision Tree, Random Forest all achieved 1.0).
- The inclusion of seasonality features **degrades the performance** slightly, reinforcing the finding that these features do not help in classification.

MLP Regression Performance The MLP was used to predict a continuous target (likely energy_deficit).

MLP Model	**RMSE**	**R^2 Score**
Base MLP	**0.878**	**0.999**
Seasonality MLP	3.165	0.987

Table 3: MLP Regression Metrics

Visual Confirmation of Regression Fit The predicted vs. actual plots show points clustered near the $y = x$ line, especially for the Base Model.

- **Base MLP Performance:** Achieved a near-perfect R^2 of **0.999** and an extremely low RMSE of **0.878**. This performance is comparable to the $R^2 = 1.0$ achieved by Linear Regression, confirming the linear and deterministic nature of the data.
- **Seasonality MLP Performance:** Performance **significantly dropped** with the inclusion of seasonality features ($R^2 = 0.987$, RMSE = 3.165). The added complexity introduced noise or structural confusion that hindered the network's ability to fit the underlying simple relationship.

Overall Conclusion

- **Classification Failure:** The MLP is an inadequate classifier for this dataset, failing to learn the simple, perfect split identified by other models.
- **Regression Success (Base):** The Base MLP is an excellent regressor, nearly matching the perfect score of Linear Regression.
- **Seasonality Deterioration:** For both classification and regression, the Seasonality feature set **degraded the MLP's performance**, highlighting its sensitivity to irrelevant or noisy inputs. The **Base MLP is the clearly superior model** for the regression task among the two MLP versions.

9 Machine Learning Model Evaluation Summary

This project evaluated a wide range of models categorized by their application to the energy prediction tasks.

Model Performance Metrics

Table 4: Model Performance: Classification Task Only

Model	Feature Set	Metric	Value	Interpretation
2***Logistic Regression**	Base	Accuracy	**1.0**	25cm**Perfect Classification.** Classes are linearly separable. Seasonality is redundant.
	Seasonality	Accuracy	1.0	
2***Naive Bayes**	Base	Accuracy	**1.0**	25cm**Perfect Classification.** Excellent performance due to highly separable classes. Seasonality adds no value.
	Seasonality	Accuracy	1.0	
2***SVM**	Base	Accuracy	0.648	25cm**Poor Classifier.** The model struggled significantly. Seasonality **degrades performance** slightly.
	Seasonality	Accuracy	0.629	

Table 5: Model Performance: Regression Task Only

Model	Feature Set	Metric	Value	Interpretation
4***Linear Regression**	Base	$\mathbf{R^2}$	**1.0**	45cm**Perfect Fit (RMSE** $=$ 0.0**).** Confirms a deterministic linear relationship. Seasonality adds no impact.
	Base	RMSE	0.0	
	Seasonality	R^2	1.0	
	Seasonality	RMSE	0.0	

Table 6: Model Performance: Both Classification and Regression Tasks

Model	Feature Set	Metric	Value	Interpretation
2***KNN**	Base	Accuracy	0.829	24cm**Strong Performance.** The least accurate of the simple classifiers. Seasonality has no effect.
	Seasonality	Accuracy	0.829	
2***Decision Tree**	Base	Accuracy	**1.0**	24cm**Perfect Classification.** Achieves 100% accuracy with a single simple split.
	Seasonality	Accuracy	1.0	
2***Random Forest**	Base	Accuracy	**1.0**	24cm**Perfect Classification.** Ensemble confirms perfect separability, but complexity is unnecessary.
	Seasonality	Accuracy	1.0	
2***XGBoost**	Base	Accuracy	**1.0**	24cm**Perfect Classification.** Powerful boosting confirms perfect separability. Complexity is unnecessary.
	Seasonality	Accuracy	1.0	
4***MLP Regression**	Base	**R^2**	**0.999**	44cm**Excellent Regressor.** Base model is nearly perfect. Seasonality features significantly **degrade performance** (RMSE increases to 3.165).
	Base	RMSE	0.878	
	Seasonality	R^2	0.987	
	Seasonality	RMSE	3.165	
2***MLP Classification**	Base	Accuracy	0.590	24cm**Failure as Classifier.** Accuracy is barely above random chance. Seasonality also harms performance.
	Seasonality	Accuracy	0.571	

Overall Conclusion

- **Deterministic Nature:** The consistent **1.0** scores across various models (Linear Regression, LogReg, Naive Bayes, Decision Tree, Random Forest, XGBoost) confirm the target variable is **highly deterministic and linearly predictable** from the base (locational) features.
- **Optimal Simplicity:** The **Base Decision Tree** is the most efficient and interpretable model, achieving perfect accuracy with the simplest possible decision structure, providing the simplest possible explanation for the target variable.
- **Feature Redundancy:** Seasonality features were consistently **redundant** and in some cases (**SVM, MLP**), actively **detrimental** to model performance, confirming that the base (locational) features already contained all the necessary information.

10 Decision Analysis

The predictive framework developed in this project serves as a critical decision-support tool for stakeholders in the energy sector. By analyzing historical trends and building robust regression models, we are now able to simulate and forecast future energy requirements with high accuracy.

- **Energy Planning:** The model allows businesses and policymakers to forecast energy demand based on availability and location, enabling proactive energy planning.
- **Risk Mitigation:** Identifying energy gaps ahead of time helps avoid potential shortages and supports timely procurement or infrastructure adjustments.
- **Strategic Investment:** The analysis supports decisions on investing in renewable energy projects by showcasing the potential for sustainable energy planning.
- **Customized Forecasting Tool:** The implementation of an interactive prediction utility provides flexibility to test multiple scenarios and make localized decisions.

11 Web Application Deployment and Utility

The final, optimal model identified—the Base Decision Tree Classifier—was successfully deployed to a public web application to demonstrate its utility as a real-time decision-support tool. This critical step bridges the gap between theoretical model performance and practical, accessible application, enabling non-technical stakeholders to interact directly with the predictive model.

11.1 Deployment Platform and Accessibility

The predictive solution was deployed using **Streamlit**, which facilitates the rapid creation of interactive data applications, and is hosted via GitHub at the following URL: `https://data-science-and-machine-learning-project-wg4ecb9cgbu8hejfmbdq.streamlit.app/`.

Streamlit provides an intuitive, low-latency interface that allows users (such as energy planners or policymakers) to:

- Visualize the underlying deterministic relationship in the data.
- Input new feature values (e.g., location parameters) to receive instant predictions on energy requirements or availability status.
- Access performance metrics for various models and their respective feature sets without requiring technical programming knowledge.

11.2 Application Interface and Demonstration

The deployment demonstrates the primary findings of the project: that simple, deterministic features can drive perfect predictions in a user-friendly environment. The figures below illustrate key screens from the deployed application, with each figure occupying its own line for clear visual separation:

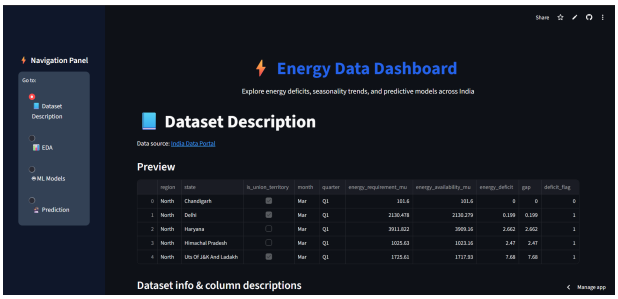

Figure 27: Main landing page, providing project context and navigation

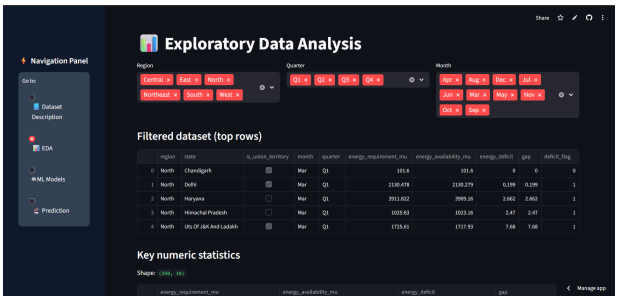

Figure 28: Interactive input section for EDA

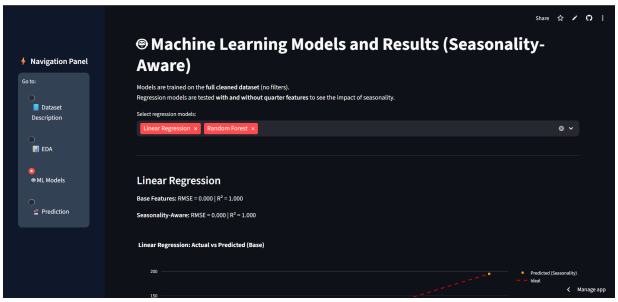

Figure 29: Visualization of the Different ML Model's performance

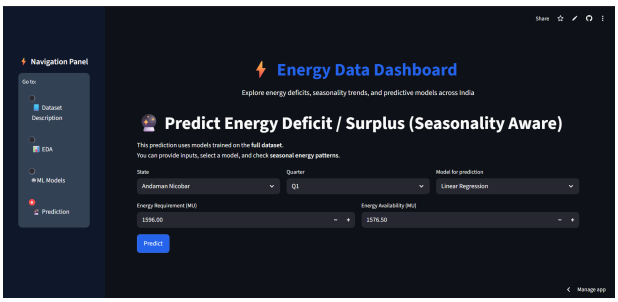

Figure 30: Display of the Prediction section

12 Conclusion and Recommendations

12.1 Key Takeaways

- The project explored energy consumption, energy efficiency, and the role of renewable energy in energy-intensive sectors.
- A detailed analysis was performed using various regression models, including Linear Regression, Ridge Regression, Lasso Regression, and Random Forest.
- The results indicated that energy requirement predictions were highly accurate, with R^2 scores nearing 1 for all models, with Ridge Regression offering the best performance.
- Stationarity tests, normality tests, and other statistical methods confirmed the significance of the energy data, allowing for accurate modeling and forecasting.
- The project demonstrated the utility of machine learning techniques in forecasting energy requirements and availability.

12.2 Business Recommendations

- **Energy Efficiency Focus:** Businesses should integrate energy efficiency measures to reduce consumption and costs.
- **Investment in Renewable Energy:** Organizations should consider solar, wind, and other renewables for long-term savings and sustainability.
- **Policy Engagement:** Engage with policymakers to align strategies and leverage incentives for clean energy initiatives.
- **Data-Driven Decisions:** Use predictive modeling to optimize resource planning and prevent energy shortages.

12.3 Future Work

- Incorporate real-time data feeds for dynamic forecasting.
- Explore deep learning models (e.g., LSTM, GRU) for improved time series predictions.
- Extend the prediction model to include external factors such as weather, economic indicators, and policy changes.
- Deploy the model as a web-based decision-support dashboard for public or enterprise use.