INTRODUCTION

INTRODUCTION

1.1 OBJECTIVE:

Project Overview

Project Goal:

Energy IQ is a data-driven initiative aimed at optimizing energy consumption and promoting a greener environment within Manyata Tech Park. The project's primary objective is to reduce electricity usage across various industries within the park, contributing to a more sustainable and environmentally conscious ecosystem.

1.2 Problem Statement: The industrial sector is a significant contributor to global electricity consumption, accounting for approximately 42% of worldwide usage (as of 2018). Large companies like Samsung, TSMC, Alphabet, Microsoft, Intel, and Facebook are among the top consumers of electricity, highlighting the scale of the challenge. Manyata Tech Park, as a thriving hub of various industries, faces similar challenges. Understanding and optimizing energy consumption within the park is crucial for reducing environmental impact and promoting sustainability.

1.3 Solution:

Energy IQ tackles this challenge through a two-pronged approach:

Predictive Analytics: Developing a machine learning model to forecast monthly electricity consumption for individual tenants within Manyata Tech Park. This model leverages historical energy data, weather information, and tenant-specific features to provide accurate predictions.

Power BI Dashboard: Creating a user-friendly and interactive Power BI dashboard that visualizes historical data, model predictions, and provides actionable insights for energy optimization.

1.4 Benefits:

Reduced Electricity Consumption: By accurately forecasting energy demand, tenants can proactively manage their consumption, potentially leading to significant reductions in electricity usage.

Cost Savings: Lower energy consumption translates into lower electricity bills, resulting in financial savings for tenants and the park as a whole.

Environmental Impact Reduction: Minimizing electricity consumption contributes to reducing greenhouse gas emissions and promoting a more sustainable environment.

Improved Operational Efficiency: The insights derived from Energy IQ can lead to more efficient building operations, optimized HVAC systems, and better resource management.

Enhanced Transparency & Accountability: The dashboard provides a transparent view of energy consumption for individual tenants and the park overall, promoting accountability and encouraging responsible energy use.

1.5 Project Scope:

This initial phase of Energy IQ focuses on:

Target Industries: The project will analyze electricity consumption across a representative sample of industries present in Manyata Tech Park.

Data Period: The model will be trained using historical data from [Specify the time period, e.g., 2015-2024], and predictions will be generated for the next month.

Dashboard Functionality: The Power BI dashboard will provide a range of visualizations, interactive filters, and key performance indicators (KPIs) to enable effective analysis and decision-making.

1.8 Project Deliverables:

Predictive Model: A trained machine learning model capable of predicting monthly energy consumption for tenants.

Power BI Dashboard: A fully functional interactive dashboard for visualizing historical data, predictions, and insights.

Project Report: This comprehensive report documenting the project methodology, results, and future directions.

MAJOR MODULES:

- Data Generation
- Data Preprocessing
- ML Training
- Training and Evaluation
- Deployment and Visualization

Functionalities of Modules

- **Historical Data Gathering**: Collect data on electricity consumption for each tenant within Manyata Tech Park.

Include:
Tenant Name
Industry Type
Date (daily, weekly, or monthly)
Electricity Reading (kWh)
Other relevant features (e.g., building size, number of employees, HVAC usage, etc.)

- **Weather Data**: Collect relevant weather data for Bangalore

Include:

Date

Temperature (average, maximum, minimum)

Precipitation

Humidity

Other relevant weather parameters

2. Data Preprocessing

- Data Cleaning:
- Handle missing values data.
- Detect and treat outliersNormalization/Standardization
- Feature EncodingConsider:
- One-hot encoding
- Label encoding
- Ordinal encoding
- Feature Selection
- Correlation analysis
- Feature importance analysis from model training
- Data Splitting

3. Model Training & Evaluation:

- LightGBM:
- Split the data into training and testing sets (80% training, 20% testing).
- Trained the LightGBMRegressor model using the training data.
- Evaluated the model using metrics like Mean Squared Error (MSE) and R-squared.

4.Model Deployment:

- Model Saving: The trained LightGBMRegressor model was saved using pickle for future use in the Power BI dashboard.
- Build an interactive Power BI dashboard to:

- Show historical data trends, weather impact, and predicted values.
- Allow filtering by tenant, time period, and industry type.
- Display key metrics like total consumption, average consumption, and forecast accuracy.

SOFTWARE REQUIREMENT ANALYSIS

SOFTWARE REQUIREMENT ANALYSIS

System Analysis is a detailed study of the various operations performed by a system and their relationships within and outside of the system. Analysis begins when a developer begins a study of the program using an existing system. During analysis, data are collected on the various files, decision points and transactions handled by the present system. The commonly used tools in the system are Data Flow Diagram, interviews, etc. Training, experience and common sense are required for collection of relevant information needed for the development of the system. The success of the system depends largely on how clearly the problem is defined, thoroughly investigated and properly carried out through the choice of solution. A good analysis model should provide not only the mechanisms of problem understanding but also the frame work of the solution. Thus, it should be studied thoroughly by collecting data about the system. Then the proposed system should be analyzed thoroughly in accordance with the needs.

PROPOSED SYSTEM

The proposed model for predicting electricity consumption using a LightGBM Regression approach seeks to improve accuracy by incorporating robust data preprocessing techniques, such as feature engineering and selection, to capture key tenant-related factors like area acquired, total employees, and building age. The model's performance will be enhanced through hyperparameter optimization, ensuring that the relationship between variables is accurately captured. Regularization techniques will be used to prevent overfitting, and adaptive mechanisms will be implemented to adjust predictions based on new incoming data. Cross-validation will be applied for more reliable model evaluation, with the deployment focusing on real-time forecasting of energy usage per tenant, supported by continuous retraining to improve prediction accuracy.

SOFTWARE REQUIREMENT SPECIFICATIONS

SOFTWARE REQUIREMENT SPECIFICATIONS

A well-structured software requirement specification (SRS) outlines the functionality and performance of the system, ensuring that all system operations are clearly defined and that the technology stack is chosen accordingly.

An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

3.1 FUNCTIONAL REQUIREMENTS:

Data Input: Accept historical electricity usage data from Manyata Tech Park tenants.
Model Training: Train a regression model using input features like industry type, floor area, and weather data.
Electricity Forecasting: Predict future electricity consumption for tenants based on historical data and external factors.
Model Persistence: Save and load trained machine learning models for future use.
Visualization: Provide graphical representations of actual vs. predicted electricity readings.
Dashboard Integration: Present data in an interactive Power BI dashboard.
Monthly Prediction Updates: Generate monthly electricity usage forecasts for each tenant.
Error Calculation: Calculate and display error metrics like Mean Squared Error and R-squared.
User Input for Custom Predictions: Allow users to input specific parameters (e.g., tenant or date) to get custom predictions.

3.2 NON-FUNCTIONAL REQUIREMENTS:

Performance: Ensure the model runs efficiently and predictions are generated within seconds.

Scalability: The model should support adding more tenants or data without a significant drop in performance.

Security: Protect the integrity of the model and ensure that sensitive tenant data is secure.

Accuracy: Maintain prediction accuracy with error metrics below a defined threshold.

Reliability: Ensure the system runs smoothly with minimal downtime for model training and predictions.

Usability: Provide a simple, intuitive interface for users to interact with the model and view forecasts.

Maintainability: The code and model should be easy to maintain and update as new data or requirements emerge.

HARDWARE & SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS	
Processor	Intel Core i3 or above (Recommended: i5 or higher)
RAM	4GB (Recommended: 8GB for faster processing)
Storage	500GB HDD or SSD
SOFTWARE REQUIREMENTS	
Operating System	Windows, Linux, or macOS
Programming Language	Python 3.x for data processing, machine learning model development, and analytics.
Machine Learning Libraries	Scikit-learn (for machine learning),Pickle
Front-End	Dashboard(Power BI)
Database	CSV-based storage for weather, Manyata tenant detail,predicted electricity Reading data storage
IDE	Any Python-compatible IDE such as PyCharm, VS Code, or Jupyter Notebook for writing, testing, and deploying the project code.

SOFTWARE PROFILE

SOFTWARE PROFILE

5.1 POWER BI

Microsoft Power BI is an interactive data visualization software product developed by Microsoft with a primary focus on business intelligence. It is part of the Microsoft Power Platform. Power BI is a collection of software services, apps, and connectors that work together to turn various sources of data into static and interactive data visualizations. Data may be input by reading directly from a database, webpage, PDF, or structured files such as spreadsheets, CSV, XML, JSON, XLSX, and SharePoint Key components.

Key components of the Power BI ecosystem are as follows:

Power BI Desktop

The Windows desktop-based application for PCs, primarily for designing and publishing reports to the service.

Power BI Service

The SaaS-based (software as a service) online service. This was formerly known as Power BI for Office 365, now referred to as PowerBI.com, or simply Power BI.

Power BI Mobile Apps

The Power BI Mobile apps for Android and iOS devices, as well as for Windows phones and tablets.

Power BI Gateway

Gateways are used to sync external data in and out of Power BI and are required for automated refreshes. In enterprise mode, it can also be used by Microsoft Power Automate (previously called Flows) and PowerApps in Office 365.

Power BI Embedded

Power BI REST API can be used to build dashboards and reports into the custom applications that serves Power BI users, as well as non-Power BI users.

Power BI Report Server

An on-premises Power BI is a reporting product for companies that choose not to store data in the cloud-based Power BI Service.

Power BI Premium

Capacity-based offering that includes flexibility to publish reports broadly across an enterprise without requiring recipients to be licensed individually per user. This provides greater scale and performance than shared capacity in the Power BI Service.

Power BI Visuals Marketplace

A marketplace of custom visuals and R-powered visuals.

Power BI Dataflow

A Power Query implementation in the cloud that can be used for data transformations to make a common Power BI Dataset, which can then can be made available for report developers through Microsoft's Common Data Service. For example, it can be used as an alternative to doing transformations in SSAS, and may ensure that several report developers use data that has been transformed in a similar way.

Paginated reports for Power BI, which can be built with Power BI Report Builder, are a special type of SSRS reports with pagination formatting which can give better control of the layout of reports which need to be printed to paper or PDF. This is in contrast to regular Power BI reports which instead are optimized for presentation or interactivity and exploration on a screen. Paginated reports can, as of 2022, not be made with the regular Power BI Desktop report builder software. Instead, the standalone Power BI Report Builder has to be used, which can be viewed as a descendant of the SQL Server Reporting Services (SSRS) Microsoft Report Builder for Microsoft SQL Server introduced in 2004. It is also similar to the Report Designer in SQL Server Data Tools.

Power BI Paginated reports are saved in the Report Definition Language (.rdl file format), as opposed to the .pbix file of regular Power BI reports. The RDL format is based on XML, and was proposed by Microsoft as a benchmark for defining reports with SSRS

Paginated reports may be more suitable than regular Power BI reports, and may include printing of invoices or other repeated printouts of reports with a similar layout but different content, or for printing reports where text would otherwise overflow due to being cut off by scrollbars.

5.2 PANDAS

Pandas (styled as **pandas**) is a software library written for the for Python programming language data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "**panel data**", an econometrics term for data sets that include observations over multiple time periods for the same individuals, as well as a play on the phrase "Python data analysis". Wes McKinney started building what would become Pandas at AQR Capital while he was a researcher there from 2007 to 2010.

The development of Pandas introduced into Python many comparable features of working with DataFrames that were established in the R programming language. The library is built upon another library, NumPy.

Developer Wes McKinney started working on Pandas in 2008 while at AQR Capital Management out of the need for a high performance, flexible tool to perform quantitative analysis on financial data. Before leaving AQR he was able to convince management to allow him to open source the library.

Another AQR employee, Chang She, joined the effort in 2012 as the second major contributor to the library.

In 2015, Pandas signed on as a fiscally sponsored project of NumFOCUS, a 50 nonprofit charity in the United States.

5.3VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a free, open-source, lightweight code editor developed by Microsoft, designed for a wide range of programming languages and platforms. It is highly extensible and provides a powerful, yet user-friendly interface, making it popular among developers for tasks ranging from small scripting to full-scale application development.

Key Features:
Cross-Platform: VS Code is available on Windows, macOS, and Linux, allowing developers to use the same tool across different operating systems.

Built-in IntelliSense: It offers smart code completions based on variable types, function definitions, and imported modules. This makes coding faster and reduces syntax errors.

Integrated Git: VS Code has integrated Git version control, allowing developers to manage code repositories, stage changes, commit code, and even push to remote repositories directly from the editor.

Debugger: It has a built-in debugger that supports various languages and allows for setting breakpoints, stepping through code, and inspecting variables, making debugging straightforward.

Extensions and Plugins: One of VS Code's most powerful features is its vast marketplace of extensions. Developers can enhance the editor with language support, themes, linters, debuggers, and other tools like Docker, Python, or JavaScript frameworks.

Terminal Integration: VS Code comes with an integrated terminal, so developers can run command-line tools, execute scripts, or manage projects without leaving the editor.

Code Formatting and Linting: Extensions like Prettier, ESLint, and others provide automatic code formatting and linting, ensuring consistent coding styles and catching syntax errors early.

Live Share: This feature allows for real-time collaborative coding, where multiple developers can work on the same project simultaneously, making it great for pair programming or remote collaboration.

Customizable User Interface: VS Code is highly customizable, allowing users to modify the layout, color schemes, keyboard shortcuts, and settings to suit their workflow preferences.

Lightweight and Fast: Unlike full Integrated Development Environments (IDEs) like Visual Studio, VS Code is relatively lightweight and starts quickly, yet it still provides many powerful features typically found in full IDEs.

Typical Use Cases:
Web Development: With support for HTML, CSS, JavaScript, and frameworks like React, Angular, and Vue.js.
Python Development: Through the Python extension, VS Code provides debugging, linting, testing, and virtual environment support.
Version Control: Integrated Git support makes it ideal for projects managed in GitHub, GitLab, or Bitbucket.

Collaborative Programming: The Live Share feature allows developers to collaborate remotely in real time.
Data Science: Through Jupyter Notebooks integration, it is increasingly used by data scientists for Python-based analytics and machine learning.

VS Code strikes a balance between being lightweight and offering the flexibility of an IDE, making it a versatile and popular choice for developers across different fields and languages.

5.4 DRAWI IO

- **Cloud-Based and Desktop Versions**: Draw.io is available as both a web application and a desktop application, allowing users to work online or offline. The web-based version integrates with popular cloud storage services like Google Drive, OneDrive, Dropbox, and GitHub, making it easy to save, share, and access diagrams from anywhere.

Features:

1. **Wide Range of Diagram Types**:
    a. **Flowcharts**: For mapping processes and workflows.
    b. **Entity-Relationship Diagrams (ERDs)**: For database structure representation.
    c. **Unified Modeling Language (UML) Diagrams**: For visualizing software architecture and object-oriented designs.
    d. **Org Charts**: For representing organizational hierarchies.
    e. **Network Diagrams**: For network architecture visualization.

f. **Venn Diagrams**: For showing logical relationships between sets.
g. **Mind Maps**: To brainstorm and organize ideas.

2. **Drag-and-Drop Interface**: Draw.io has a highly intuitive user interface where users can easily drag and drop shapes, connectors, images, and text to create complex diagrams. This makes the tool accessible for users of all skill levels, from beginners to advanced users.
3. **Templates and Pre-built Libraries**: The platform provides a vast collection of templates and shape libraries for various diagram types. You can quickly start from pre-built templates or create diagrams from scratch. Shape libraries cover categories such as:
    a. Basic shapes (rectangles, circles, etc.)
    b. Flowchart symbols (process, decision, start/end, etc.)
    c. UML components (classes, objects, etc.)
    d. Network and cloud architecture components (servers, databases, routers, etc.)
4. **Collaboration and Sharing**:
    a. **Real-Time Collaboration**: Multiple users can work on the same diagram simultaneously, making it ideal for team projects and remote work.
    b. **Sharing**: Diagrams can be easily shared via link or exported in various formats, including PNG, JPEG, SVG, and PDF.
    c. **Version Control**: You can track changes in the diagram over time, making it easy to revert to previous versions.
5. **File Formats and Integration**: Draw.io allows users to import and export diagrams in multiple formats, such as:
    a. **XML**: To retain full editing capability for later use in Draw.io.
    b. **PDF**: For high-quality, non-editable sharing.
    c. **VSDX (Visio)**: For compatibility with Microsoft Visio diagrams.
    d. **Images**: Export as PNG, JPEG, or SVG for use in reports and presentations.

It integrates with cloud storage solutions like Google Drive, OneDrive, and GitHub, allowing users to save and sync diagrams across different platforms. It also has a Confluence plugin, making it a preferred choice for teams using Atlassian's tools.

6. **Offline Mode**: Draw.io offers a desktop version for offline work. You can save diagrams locally and open them later without needing an internet connection, which is useful for privacy and working in environments without connectivity.
7. **Customizable Elements**: Users can customize diagrams extensively with options to:
    a. Modify shapes, colors, borders, and fonts.
    b. Add custom images, logos, and icons.
    c. Use custom connectors, arrows, and line styles to represent various relationships between diagram elements.
8. **Keyboard Shortcuts**: For power users, Draw.io provides a range of keyboard shortcuts to streamline the diagram creation process. This can greatly improve efficiency when building large or complex diagrams.
9. **Open-Source and Free**: Unlike many other diagramming tools, Draw.io is open-source, which means it is free to use without any licensing fees. There are no premium tiers, making all features accessible without limitations.

Use Cases:

1. **Business Process Mapping**: Visualize workflows, decision-making processes, and business operations using flowcharts.
2. **Software Design and Development**: Create UML diagrams, ERDs, and class diagrams to design software architectures.
3. **Project Management**: Use Gantt charts, process diagrams, and mind maps to plan, organize, and track projects.
4. **Network Architecture**: Visualize network infrastructure with detailed network diagrams showing routers, servers, and connections.

5. **Database Design**: Use ER diagrams to map out databases, relationships between entities, and data models.
6. **Educational Tools**: Teachers and students use Draw.io to create diagrams for presentations, reports, and research projects.

Benefits:

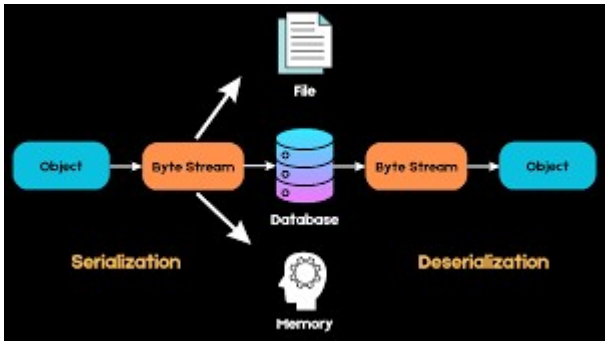
- **Ease of Use**: No steep learning curve, making it accessible for new users.
- **Collaborative**: Ideal for team-based projects, especially in software development, engineering, and business.
- **Cross-Platform Compatibility**: Works seamlessly across different operating systems (Windows, macOS, Linux) and integrates with popular cloud platforms.
- **Cost-Efficient**: Since it's free and open-source, it's an excellent option for individuals, small businesses, and large organizations alike.

5.5 Pickle

Introduction to Pickle in Python

Python's Pickle module is a popular format used to serialize and deserialize data types. This format is native to Python, meaning Pickle objects cannot be loaded using any other programming language.

Pickle comes with its own advantages and drawbacks compared to other serialization formats.



Advantages of using Pickle to serialize objects

Unlike serialization formats like JSON, which cannot handle tuples and datetime objects, Pickle can serialize almost every commonly used built-in Python data type. It also retains the exact state of the object which JSON cannot do.

Pickle is also a good choice when storing recursive structures since it only writes an object once.

Pickle allows for flexibility when deserializing objects. You can easily save different variables into a Pickle file and load them back in a different Python session, recovering your data exactly the way it was without having to edit your code.

unpack a PKL file, you need to follow these steps:

Import the pickle Module: The pickle module is part of Python's standard library, so you don't need to install anything extra.

Open the PKL File: Open the file in binary mode for reading.

Load the Data: Use pickle.load to deserialize the data from the file.

5.6 LightGBM

The LightGBM framework supports different algorithms including GBT, GBDT, GBRT, GBM, MART[6][7] and RF.[8] LightGBM has many of XGBoost's advantages, including sparse optimization, parallel training, multiple loss functions, regularization, bagging, and early stopping. A major difference between the two lies in the construction of trees. LightGBM does not grow a tree level-wise — row by row — as most other implementations do.[9] Instead it grows trees leaf-wise. It chooses the leaf it believes will yield the largest decrease in loss.[*citation needed*] Besides,

LightGBM does not use the widely used sorted-based decision tree learning algorithm, which searches the best split point on sorted feature values, as XGBoost or other implementations do. Instead, LightGBM implements a highly optimized histogram-based decision tree learning algorithm, which yields great advantages on both efficiency and memory consumption. The LightGBM algorithm utilizes two novel techniques called Gradient-Based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) which allow the algorithm to run faster while maintaining a high level of accuracy.

LightGBM works on Linux, Windows, and macOS and supports C++, Python, R, and C#. The source code is licensed under MIT License and available on GitHub.

LightGBM, short for **Light Gradient-Boosting Machine**, is a free and open-source distributed gradient-boosting framework for machine learning, originally developed by Microsoft. It is based on decision tree algorithms and used for ranking, classification and other machine learning tasks. The development focus is on

5.7 Scikit-learn

Scikit-learn, also known as sklearn, is an open-source, machine learning and data modeling library for Python. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python libraries, NumPy and SciPy.

Scikit-learn was first released in 2010, and it has since gained a prominent place in the Python machine learning ecosystem. It implements numerous data modeling and machine learning algorithms, and provides consistent Python APIs. It supports a standardized and concise model interface across models. For example, Scikit-learn makes use of a simple fit/predict workflow model for its classification algorithms. Scikit-learn integrates well with many other Python libraries, such as matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more. You can pass NumPy arrays and Pandas dataframes directly to Scikit-learn's algorithms.

It provides a comprehensive set of supervised and unsupervised learning algorithms, covering areas such as:

- **Classification:** Identifying which category an object belongs to.
- **Regression:** Predicting a continuous-valued attribute associated with an object.
- **Clustering:** Automatic grouping of similar objects into sets, with models like k-means.
- **Dimensionality Reduction:** Reducing the number of attributes in data for summarization, visualization and feature selection, with models like Principal Component Analysis (PCA).
- **Model Selection:** Comparing, validating and choosing parameters and models.

5.8 MATPLOTLIB

Matplotlib is one of the most widely used plotting libraries in Python, providing powerful tools for creating static, animated, and interactive visualizations. It is highly popular in data science, machine learning, engineering, and research communities due to its flexibility and extensive customization options. Matplotlib works well for various types of charts, including line plots, bar charts, histograms, scatter plots, and more.

Overview:

- **Library Type**: Open-source, 2D plotting library for Python.
- **Primary Use**: Creating publication-quality plots and visualizations, suitable for both simple and complex datasets.
- **Core Package**: It forms the basis for many other popular plotting libraries like Seaborn and pandas' plotting tools, making it a fundamental part of the Python data visualization ecosystem.

Key Features:

Comprehensive Plotting Capabilities: Matplotlib provides tools to create a wide range of visualizations:

Line plots: For time-series data or trends.
Bar charts: For comparing quantities across categories.
Histograms: To show the distribution of a dataset.
Scatter plots: For visualizing the relationship between two variables.
Pie charts: For representing proportions.
3D plots: With `mpl_toolkits.mplot3d`, you can create 3D plots for complex data.

Subplotting and Multi-Plot Figures: You can create multiple plots on the same figure, allowing for side-by-side comparisons and more advanced visualizations. The `subplots()` function is particularly helpful in creating grids of plots within a single figure window.

Customizability: Matplotlib is highly customizable. Nearly every element of a plot (axes, ticks, labels, lines, grids, legends, etc.) can be modified, allowing for intricate fine-tuning:

Color: Adjust the color of the plot elements, including lines, markers, and backgrounds.
Line styles: Choose from dashed, solid, or custom line styles.
Fonts and Labels: Control font size, font family, axis labels, and plot titles.
Legends and Annotations: Add legends, annotate points, or highlight specific areas of the plot.
Axis control: Set axis limits, ticks, and scales (linear, logarithmic).

Interactive Plots: Although Matplotlib primarily produces static plots, it can also create interactive plots when combined with libraries like `IPywidgets` in Jupyter Notebooks or by using the interactive backend (`%matplotlib notebook`).

Integration with Other Libraries: Matplotlib integrates well with:

pandas: You can directly plot DataFrames and Series objects, leveraging the data processing capabilities of pandas with Matplotlib's customization.
NumPy: As Matplotlib was designed to work well with NumPy arrays, it is ideal for visualizing scientific computations and large datasets.
Seaborn: Seaborn builds on Matplotlib, offering a higher-level interface for statistical plots and attractive color themes. You can easily combine Seaborn's aesthetics with Matplotlib's detailed customizations.

Export Options: Matplotlib supports exporting figures in a variety of formats, such as:

PNG: For web or display use.
JPEG: For reports or lower-resolution images.
SVG: For scalable vector graphics (ideal for publication).

PDF: For print or document-based visualizations.

Object-Oriented API: Matplotlib supports two styles of programming:

Pyplot API (state-based): Simpler and more accessible for quick plots, where each function call alters the state of the current figure.
Object-oriented API: More suitable for complex and intricate plots, where you explicitly create `Figure` and `Axes` objects and control every detail of the plot.

Interactive Widgets in Jupyter Notebooks: In Jupyter Notebooks, Matplotlib can render interactive widgets that enable users to zoom, pan, and explore data visually, facilitating exploratory data analysis.

Matplotlib Themes and Styles: Matplotlib includes built-in styles such as `seaborn`, `ggplot`, `fivethirtyeight`, and more. These predefined themes help users quickly change the appearance of their plots with a single line of code (`plt.style.use('ggplot')`).

Animations: Matplotlib allows you to create animations using `FuncAnimation`. This is useful for demonstrating changes in data over time, such as a moving trend line or a growing bar chart.

Use Cases:

- **Data Analysis**: Matplotlib is widely used in data science for visualizing data distributions, correlations, trends, and outliers. Histograms, box plots, and scatter plots are commonly used to understand data during exploratory data analysis (EDA).
- **Scientific Computing**: Matplotlib is ideal for scientific visualization, making it easy to graph results from numerical computations and simulations.
- **Financial Analysis**: Matplotlib can be used for time-series analysis, such as plotting stock prices, sales trends, or other financial data over time.

- **Machine Learning**: Data scientists use Matplotlib to visualize model performance, including loss curves, confusion matrices, and ROC curves, during machine learning model development.
- **Presentation and Reporting**: Matplotlib can generate publication-quality figures,
- which are often used in academic papers, business reports, and presentations.
- **Dashboards**: While not a full-fledged dashboarding tool, Matplotlib can be integrated with web frameworks like Flask or Django to display visualizations as part of a web-based
- **Limitations:**
- **3D Plots**: While Matplotlib supports basic 3D plotting, its 3D capabilities are limited compared to other libraries like Plotly or Mayavi.
- **Interactive Visualizations**: For highly interactive plots or complex dashboards, libraries like Plotly or Bokeh are often preferred.
- **Learning Curve**: Although powerful, Matplotlib's extensive customization options can lead to a steeper learning curve compared to higher-level libraries like Seaborn or Plotly.

5.9 GOOGLE COLAB

Google Colab (Colaboratory) is a free, cloud-based platform developed by Google that allows users to write and execute Python code directly in their web browser. It is particularly popular for

machine learning, data science, and artificial intelligence projects, providing access to powerful computing resources such as GPUs and TPUs without requiring any setup. Colab integrates seamlessly with Google Drive, making it easy to save and share projects.

Key Features:

Cloud-Based Environment: Google Colab operates entirely in the cloud, meaning users don't need to install or configure any software. All code execution happens on Google's servers, making it accessible from any device with an internet connection.

Free Access to GPUs and TPUs: One of Colab's most attractive features is free access to **hardware** accelerators like GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units), which are critical for training large machine learning models quickly.

Jupyter Notebook Interface: Colab is built on the Jupyter Notebook interface, providing an interactive environment where users can combine code, visualizations, and text in a single document. This makes it ideal for data analysis, experimentation, and presentations.

Pre-installed Libraries: Colab comes with many popular Python libraries pre-installed, such as TensorFlow, Keras, PyTorch, NumPy, pandas, and scikit-learn, saving users from having to install them manually.

Integration with Google Drive: Users can easily save their notebooks to Google Drive, enabling automatic backups and easy sharing. Colab also supports importing datasets from Drive or Google Sheets.

Collaboration: Colab allows multiple users to collaborate on the same notebook in real-time, similar to Google Docs. This makes it perfect for group projects, remote teamwork, or educational purposes.

Code Execution: Code cells in Colab can be executed independently, and outputs like tables, charts, and graphs are displayed inline, enabling interactive analysis and visualization.

Supports External Data: Colab provides multiple ways to import data, including uploading local files, accessing Google Drive files, and connecting to external databases or cloud storage services.

Easy Sharing: Notebooks can be shared with others via a link, making collaboration and peer review straightforward. Users can control access levels, such as view-only or editing permissions.

Customizable Environment: Users can install additional Python packages using pip or apt-get to customize their environment based on their project needs.

Typical Use Cases:

Machine Learning and AI: Training and evaluating machine learning models using TensorFlow, Keras, or PyTorch with access to high-performance hardware (GPUs/TPUs).

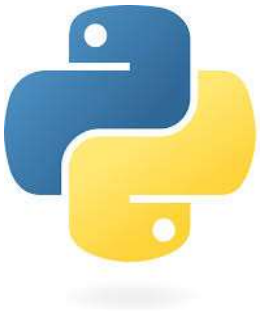
Data Science: Performing data analysis, visualization, and reporting using libraries like pandas, Matplotlib, and Seaborn.

Education: Google Colab is widely used in educational settings for teaching programming, data science, and AI, thanks to its accessibility and collaborative features.

Prototyping: Rapid prototyping and experimentation for machine learning projects without needing local infrastructure.

Google Colab simplifies Python coding and provides powerful tools for anyone working in data science, machine learning, and related fields, offering high-performance resources without the need for specialized hardware.

5.10 Python



Python is a high-level, interpreted programming language known for its readability and simplicity. Created by Guido van Rossum and released in 1991, Python emphasizes code readability, allowing developers to express concepts in fewer lines of code compared to languages like C++ or Java. Its design philosophy encourages the use of significant whitespace, making it accessible to beginners and appealing to experienced programmers.

Key Features of Python

1. Easy to Learn and Use

Python's syntax is clear and intuitive, making it an ideal choice for beginners. The language's focus on readability reduces the complexity of the code, allowing developers to concentrate on problem-solving rather than getting bogged down in syntax.

2. Interpreted Language

Python is an interpreted language, which means that code is executed line by line. This allows for quick testing and debugging, as errors can be identified immediately, facilitating a faster development cycle.

3. Dynamically Typed

In Python, variables do not need explicit declaration to reserve memory space. The declaration happens automatically when a value is assigned to a variable. This flexibility allows for rapid development and reduces the amount of code needed.

4. Extensive Standard Library

Python boasts a comprehensive standard library that supports many programming tasks. This library includes modules for file I/O, system calls, web development, data manipulation, and more, enabling developers to perform complex tasks without the need for third-party libraries.

5. Versatile and Platform-Independent

Python is cross-platform, meaning that Python programs can run on various operating systems such as Windows, macOS, and Linux without requiring changes to the codebase. This versatility allows developers to write code once and run it anywhere.

6. Large Community and Ecosystem

Python has a vast and active community that contributes to its growth and improvement. This community support results in a wealth of resources, including documentation, tutorials, forums,

and third-party libraries that extend Python's functionality, such as NumPy for numerical computations, Pandas for data analysis, and TensorFlow for machine learning.

Applications of Python

Python is widely used across different domains, showcasing its versatility:

1. Web Development

Python is popular for web development, with frameworks like Django and Flask providing tools for building robust and scalable web applications. These frameworks facilitate the rapid development of applications by providing built-in functionalities for handling requests, connecting to databases, and managing user authentication.

2. Data Science and Analysis

Python has become the go-to language for data scientists and analysts due to libraries like Pandas, NumPy, and Matplotlib. These libraries enable data manipulation, analysis, and visualization, making Python an essential tool for data-driven decision-making in various industries.

3. Machine Learning and Artificial Intelligence

Python is a prominent language in machine learning and AI, with libraries like TensorFlow, Keras, and Scikit-learn providing frameworks for building and training machine learning models. Its simplicity and efficiency make it suitable for prototyping and experimentation in AI research.

4. Automation and Scripting

Python is frequently used for automation tasks, allowing users to write scripts that automate repetitive tasks such as file handling, web scraping, and data extraction. The language's readability and ease of use make it an ideal choice for writing quick scripts to enhance productivity.

5. Game Development

Python is also utilized in game development, with libraries such as Pygame allowing developers to create games with rich graphics and interactive features. Its simplicity makes it an excellent choice for prototyping game ideas.

Python's combination of readability, simplicity, and versatility has made it one of the most popular programming languages in the world. Its extensive standard library and strong community support further enhance its appeal, allowing developers to tackle a wide range of projects, from simple scripts to complex web applications and machine learning algorithms. As technology continues to evolve, Python's relevance remains steadfast, making it an invaluable tool for programmers and developers across various domains. Whether you are a beginner looking to start your coding journey or an experienced professional working on advanced applications, Python offers the tools and capabilities needed to succeed in today's fast-paced technological landscape.

SYSTEM DESIGN

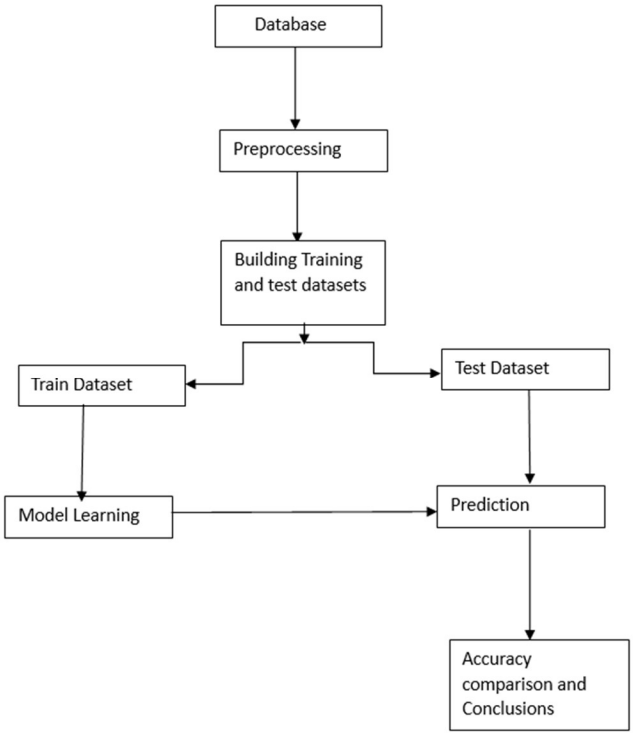
DATA FLOW DIAGRAM

Data Flow diagram:

A data-flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

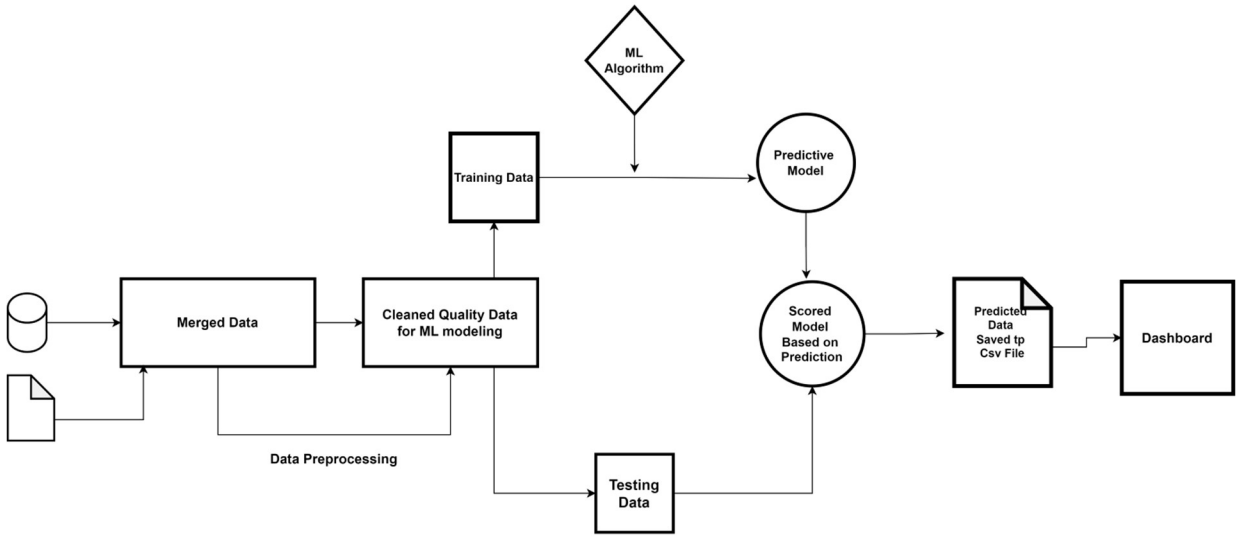
On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing or ordering of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored.



SYSTEM ARCHITECTURE:

System architecture refers to the conceptual model that defines the structure, behavior, and more views of a system. It provides a comprehensive framework for the design and development of a system, outlining how its components work together and how the system fulfills its requirements.



Data Gathering: Start with raw data from various sources (represented by the database and file icons).

Data Preparation: The raw data is cleaned, merged, and prepared for machine learning (ML) modeling.

Model Training: An ML algorithm is selected and trained using the prepared data.

Model Evaluation: The trained model is tested on a separate dataset to assess its performance.

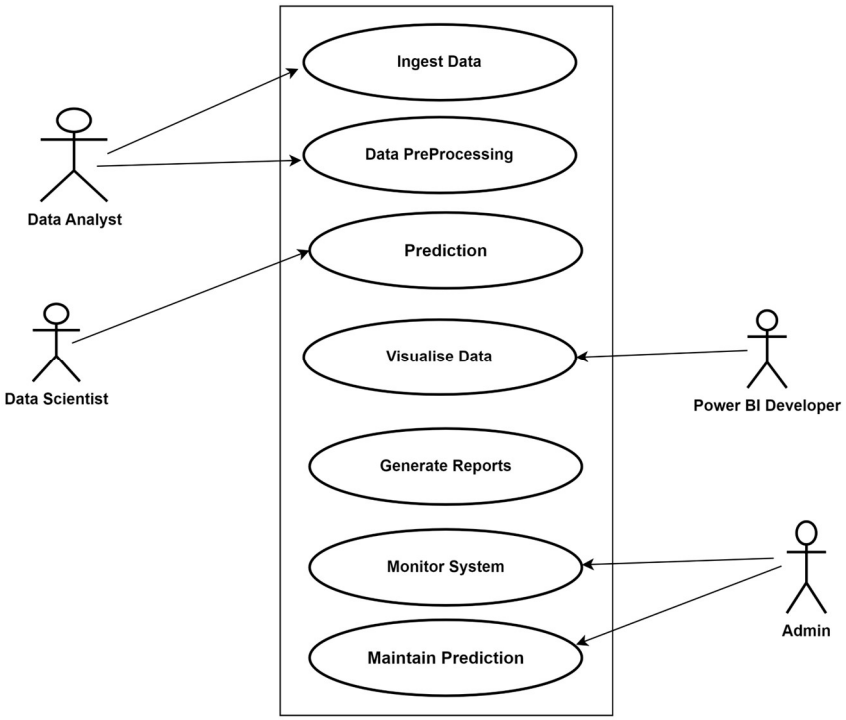
Prediction: The trained model is used to make predictions on new data.

Visualization: The results are presented on a dashboard, providing a visual summary of the predictions.

USE CASE DIAGRAM:

A Use Case Diagram is a visual representation of the interactions between users (actors) and a system, showcasing the functional requirements of the system. It is a type of Unified Modeling Language (UML) diagram that provides an overview of how different users will interact with the system and what functionalities the system will provide.

Use Case Diagrams are an essential tool in the field of software engineering and system design. They effectively capture the interactions between users and the system, providing valuable insights into system requirements and functionalities. By clearly defining actors, use cases, and relationships, Use Case Diagrams enhance communication among stakeholders and serve as a foundation for further development activities. Whether used in the initial stages of system design or as a reference throughout the development process, Use Case Diagrams play a vital role in creating successful software systems.



Key Actors:

Data Analyst: The person analyzing the data and generating insights.

Power BI Analyst The interface for visualization and reporting.

Data Scientist: The engine that processes the data and performs predictions.

Admin (Maintenance): A system admin responsible for maintaining the platform, ensuring data flows smoothly, and handling any technical issues.

Main Use Cases:

Ingest Data: Collect electricity usage and weather data from external sources.

Data Preprocessing: Process and clean the ingested data (e.g., ETL processes).

Predict Usage: Use Python models to predict future electricity readings based on Visualize

Use Power BI to create dashboards showing actual vs predicted readings.

Generate Reports: Power BI generates reports for the stakeholders

.

Monitor System: Admin monitors the system for data flow and any errors.

Maintain Predictions: Admin maintains and updates the prediction models if required.

SAMPLE TEST CASES

SAMPLE TEST CASES

7.1 CODING, TESTING, AND IMPLEMENTATION

The main interfaces in the system include:

- Web server and application server interface.
- Application server and database server interface.

It is essential to ensure that all interactions between these servers' function correctly. This involves verifying that errors are handled appropriately; if the database or web server returns any error messages for queries made by the application server, the application server should catch and display these error messages clearly to the users. Additionally, testing should address scenarios where a user interrupts a transaction mid-process or if the connection to the web server is reset during an operation.

7.2 TESTING AND ERRORS

Compatibility testing is a critical aspect of our web application. The following compatibility tests will be conducted:

- **Browser Compatibility**: Given the reliance of web applications on different browsers, it is crucial to ensure that the application is compatible across various browsers, including Internet Explorer, Firefox, Chrome, Safari, Opera, and others. We will test the application with different versions of these browsers, especially focusing on areas where JavaScript or AJAX is used for user interface functionalities.
- **Operating System Compatibility**: Functionality may vary across operating systems. The application will be tested on multiple operating systems, including Windows, macOS, Linux, and Unix, to ensure all features perform as expected.
- **Mobile Browsing**: Since users may access the application on mobile devices, it will be tested on various mobile browsers to ensure responsive design and functionality.

- **Printing Options**: Testing will include the application's printing capabilities to ensure documents can be printed correctly from different browsers and operating systems.

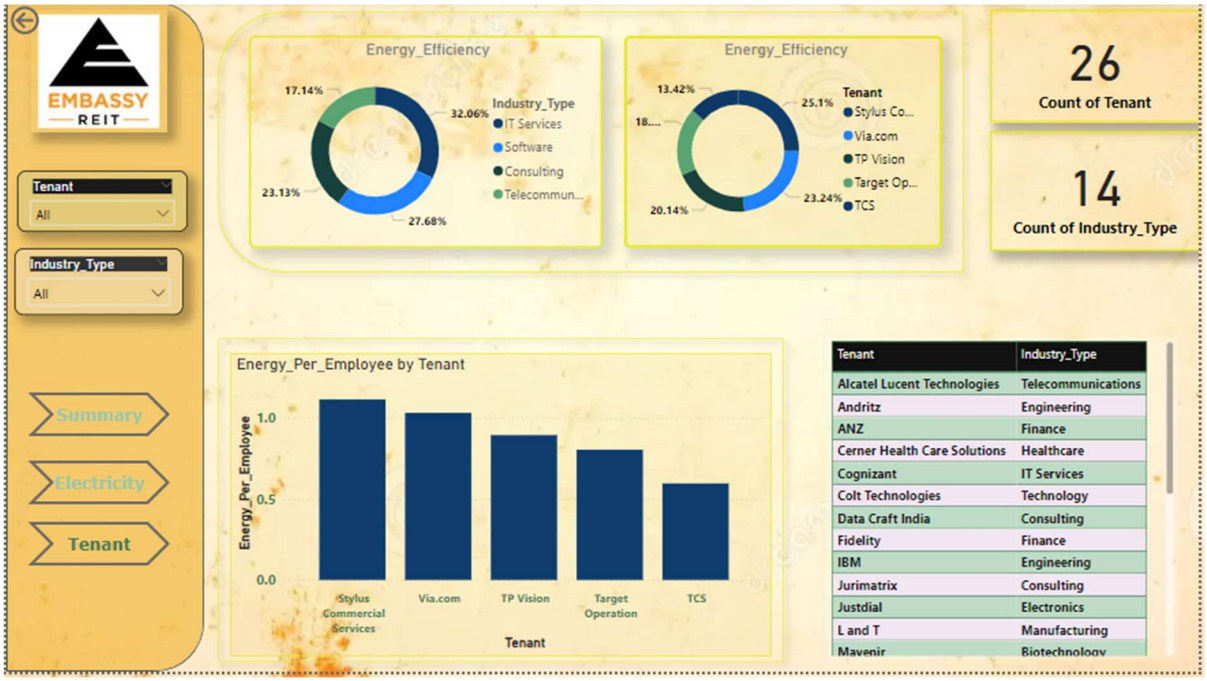
7.3 SAMPLE TEST CASES DONE

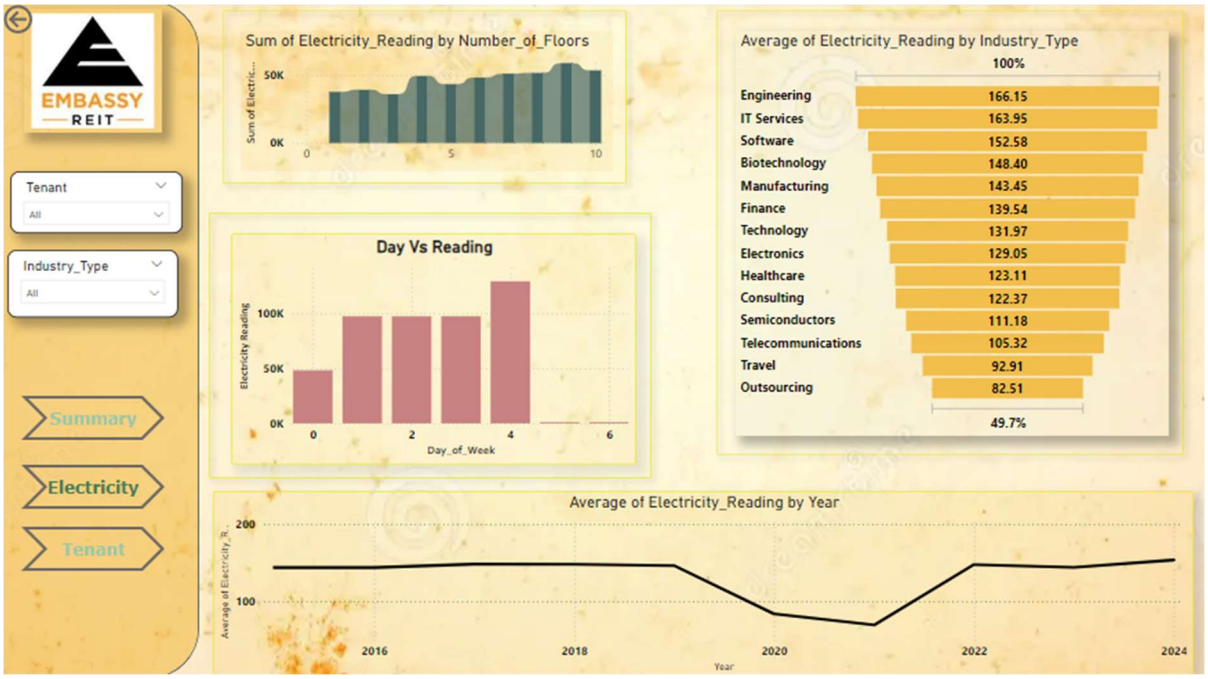
In this section, we present sample test cases executed during the system testing phase. These test cases ensure that the web application functions as expected, covering aspects such as invalid input handling, form completion, and error message prompts. Each test case documents the action taken, expected outcome, and actual system behavior, helping to verify the application's robustness and user experience across different scenarios. This ensures all components work seamlessly before the system is deployed.

Sl.No.	Test Case Description	Action Taken
1	Model Prediction: Input Valid Data	Test the model's ability to generate predictions for a specific tenant and date using valid data.
2	Model Evaluation: Compare Predictions to Historical Data	Verify that the model's predictions for a given period match historical data within an acceptable range of error.
3	Power BI Data Export: Downloading Data	Test the functionality of exporting data (if applicable) from the Power BI dashboard. Verify that the downloaded data is in the correct format and contains the expected information.
4	Power BI Visualizations: Verify Chart and Graph Display	Check that charts and graphs are displayed correctly, with appropriate labels, axes, and data.
5	Incorrect Department entry	Error message notified to the user for entering the correct club ID.

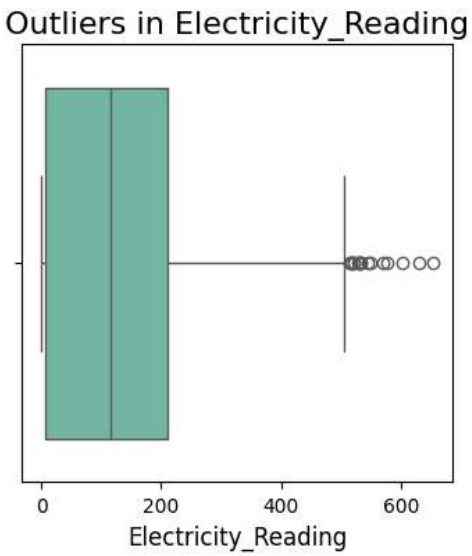
SCREENSHOTS

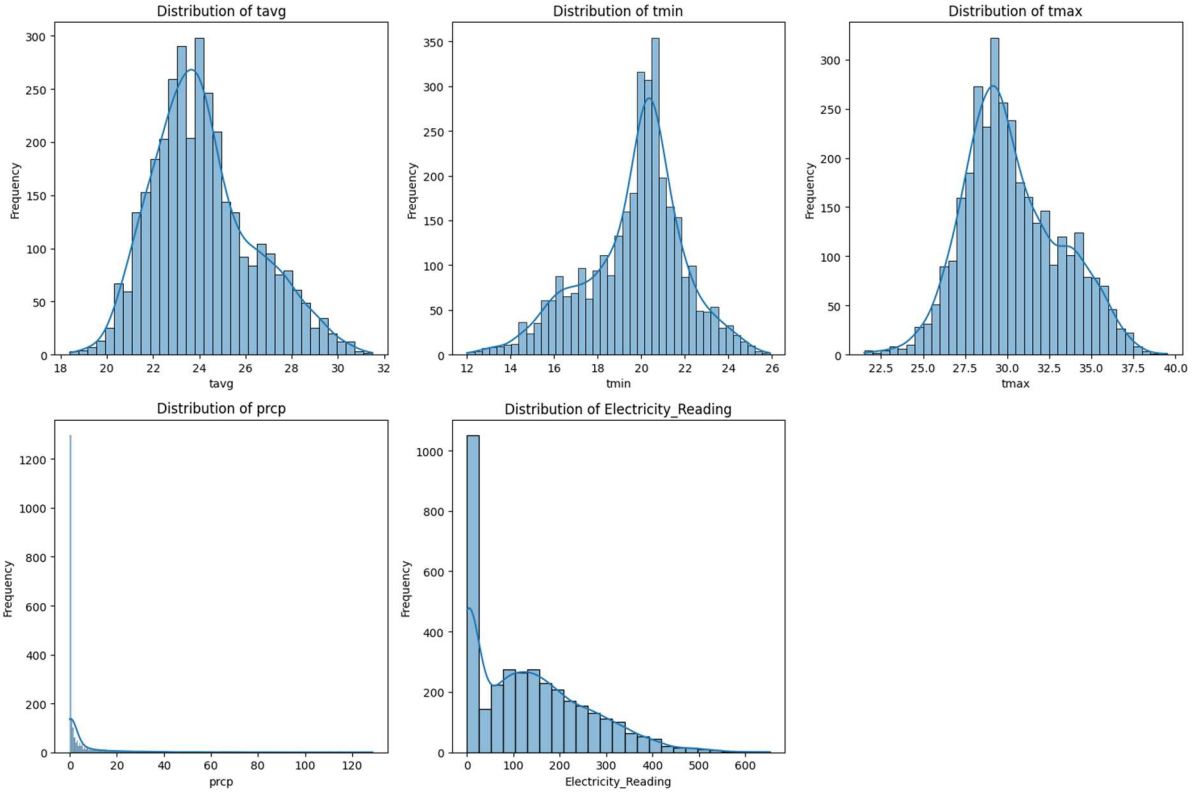

EMBASSY REIT
Tenant
All
Industry_Type
All
Summary
Electricity
Tenant
Andritz
Tenant with Minimum Energy Last Month
Energy_Efficiency
0.50
0.00
0.83
1.00
26
Count of Tenant
652.89
Max of Electricity_Reading
Total Employees
570K
132.00
Average of Electricity_Rea...
Date
Thursday, October 03, 2...
Tenant
Cognizant
Predicted Value
13.09
Sum of Predicted_Electricity_Reading (kWh)
Predicted Forecast
15
10
Reading
Oct 03

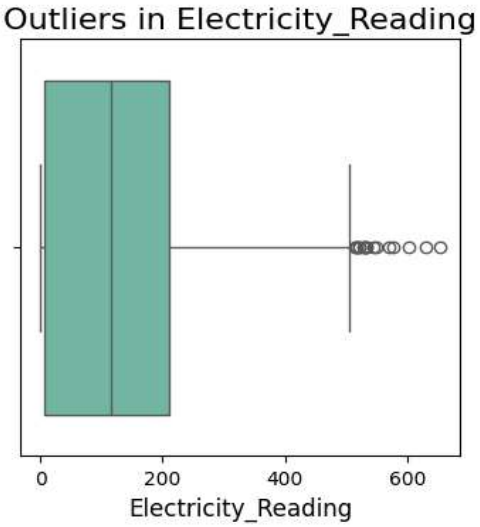

EMBASSY REIT
Tenant
All
Industry_Type
All
Summary
Electricity
Tenant
Energy_Efficiency
17.14%
32.06%
23.13%
27.68%
Industry_Type
IT Services
Software
Consulting
Telecommun...
Energy_Efficiency
13.42%
18....
25.1%
23.24%
20.14%
Tenant
Stylus Co...
Via.com
TP Vision
Target Op...
TCS
26
Count of Tenant
14
Count of Industry_Type
Energy_Per_Employee by Tenant
Energy_Per_Employee
1.0
0.5
0.0
Stylus Commercial Services
Via.com
TP Vision
Target Operation
TCS
Tenant
Tenant | Industry_Type
Alcatel Lucent Technologies | Telecommunications
Andritz | Engineering
ANZ | Finance
Cerner Health Care Solutions | Healthcare
Cognizant | IT Services
Colt Technologies | Technology
Data Craft India | Consulting
Fidelity | Finance
IBM | Engineering
Jurimatrix | Consulting
Justdial | Electronics
L and T | Manufacturing
Mavenir | Biotechnology

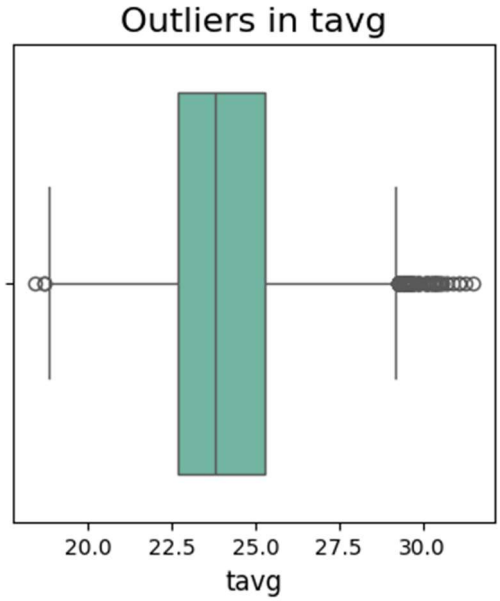

EMBASSY REIT
Tenant
All
Industry_Type
All
Summary
Electricity
Tenant
Sum of Electricity_Reading by Number_of_Floors
Sum of Electric...
50K
0K
0
5
10
Day Vs Reading
Electricity Reading
100K
50K
0K
0
2
4
6
Day_of_Week
Average of Electricity_Reading by Industry_Type
100%
Engineering 166.15
IT Services 163.95
Software 152.58
Biotechnology 148.40
Manufacturing 143.45
Finance 139.54
Technology 131.97
Electronics 129.05
Healthcare 123.11
Consulting 122.37
Semiconductors 111.18
Telecommunications 105.32
Travel 92.91
Outsourcing 82.51
49.7%
Average of Electricity_Reading by Year
Average of Electricity_R...
200
100
2016
2018
2020
2022
2024
Year

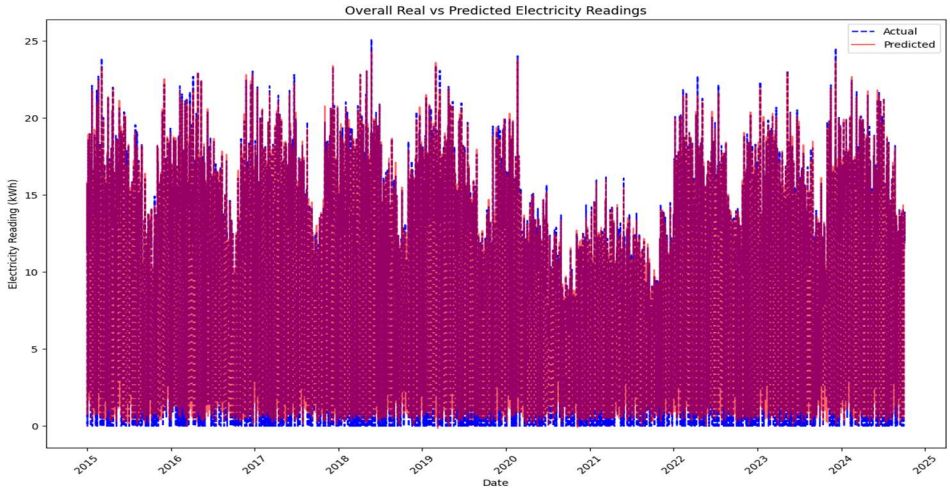

Date
Thursday, October 03, 2...
Tenant
Cognizant
Predicted Value
171.34
Predicted Reading Measure(kWh)

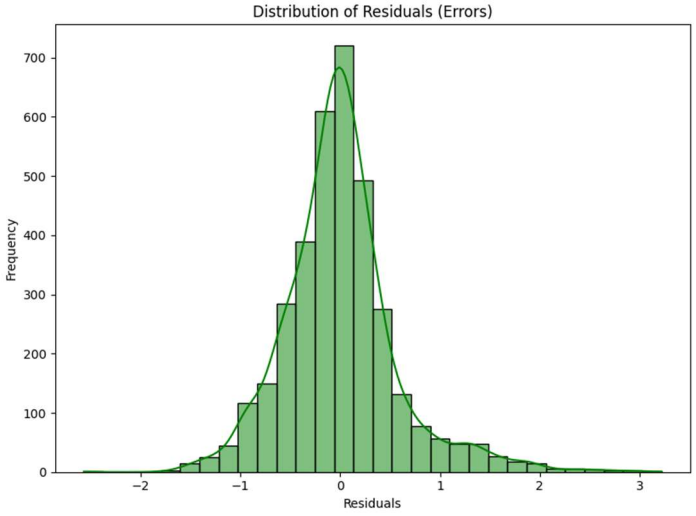

Outliers in Electricity_Reading
0
200
400
600
Electricity_Reading

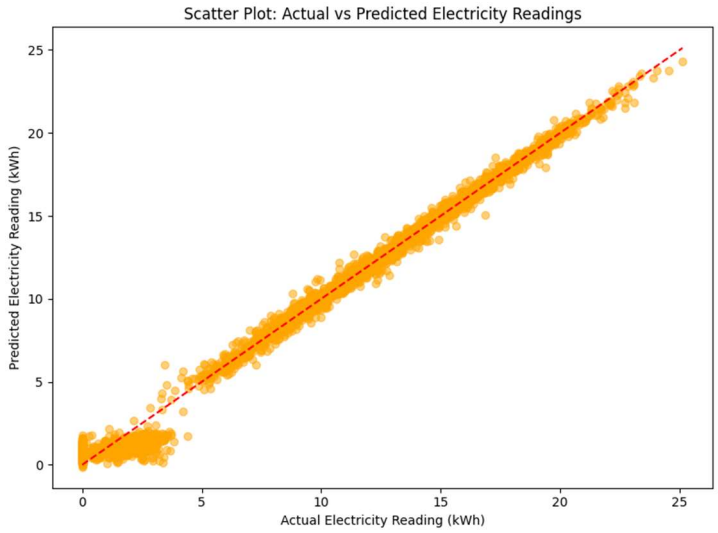

Distribution of tavg
Frequency
tavg
Distribution of tmin
Frequency
tmin
Distribution of tmax
Frequency
tmax
Distribution of prcp
Frequency
prcp
Distribution of Electricity_Reading
Frequency
Electricity_Reading

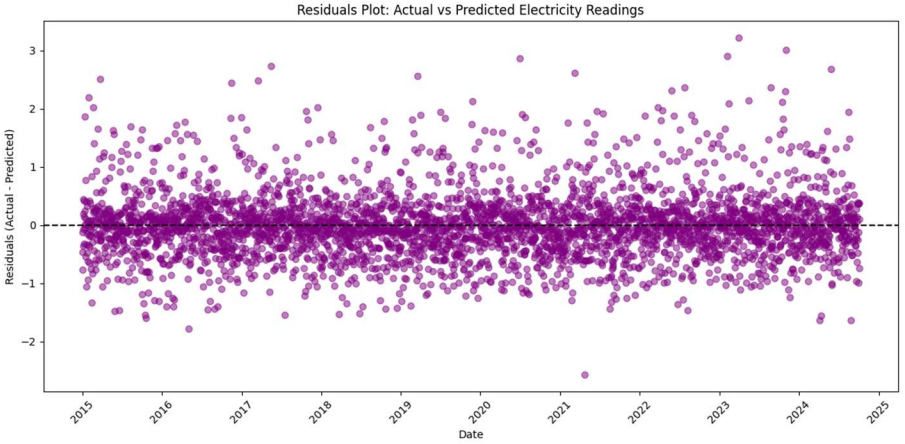

Outliers in Electricity_Reading
0
200
400
600
Electricity_Reading

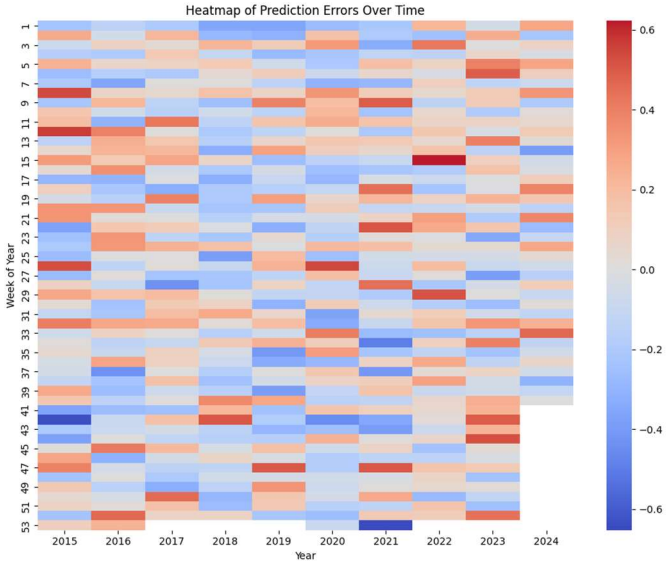

Outliers in tavg
20.0
22.5
25.0
27.5
30.0
tavg

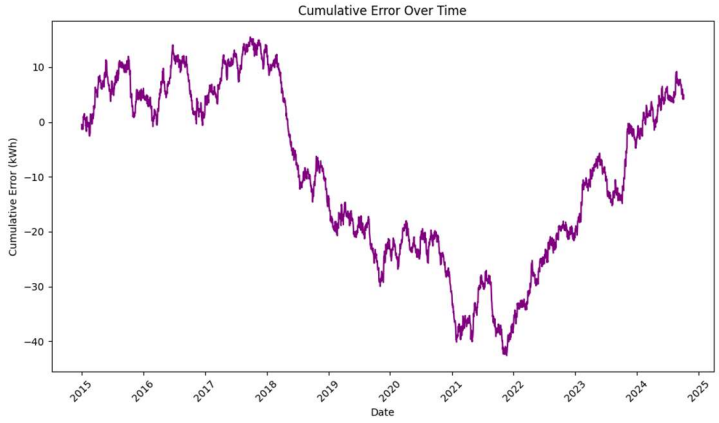

Overall Real vs Predicted Electricity Readings
Actual
Predicted
Electricity Reading (kWh)
Date
0
5
10
15
20
25
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025

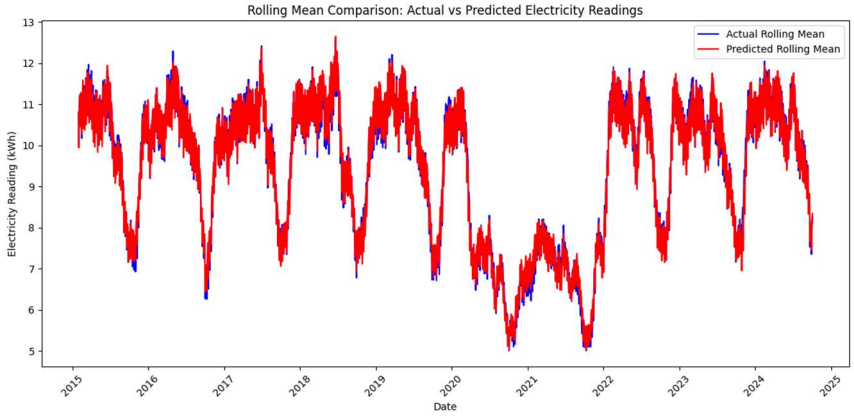

Distribution of Residuals (Errors)
Frequency
Residuals
0
100
200
300
400
500
600
700
−2
−1
0
1
2
3

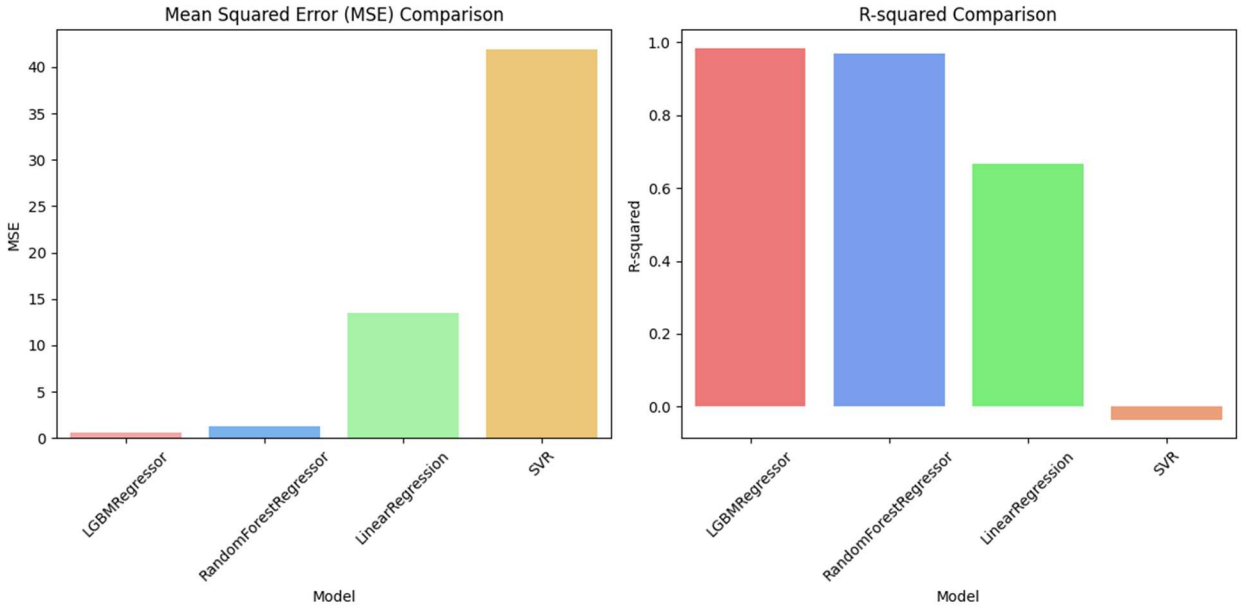

Scatter Plot: Actual vs Predicted Electricity Readings
Predicted Electricity Reading (kWh)
Actual Electricity Reading (kWh)
0
5
10
15
20
25


Residuals Plot: Actual vs Predicted Electricity Readings
Residuals (Actual - Predicted)
Date
3
2
1
0
−1
−2
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025


Heatmap of Prediction Errors Over Time
Week of Year
Year
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


Cumulative Error Over Time
Cumulative Error (kWh)
Date
10
0
−10
−20
−30
−40
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025


Rolling Mean Comparison: Actual vs Predicted Electricity Readings
Actual Rolling Mean
Predicted Rolling Mean
Electricity Reading (kWh)
Date
13
12
11
10
9
8
7
6
5
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025


Mean Squared Error (MSE) Comparison
MSE
0
5
10
15
20
25
30
35
40
LGBMRegressor
RandomForestRegressor
LinearRegression
SVR
Model
R-squared Comparison
R-squared
0.0
0.2
0.4
0.6
0.8
1.0
LGBMRegressor
RandomForestRegressor
LinearRegression
SVR
Model

CODING

CODING

Data Generation

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

# Parameters

start_date = '2015-01-01'

end_date = pd.to_datetime('today').normalize() # Today's date

total_area_sqft = 5.3 * 1e6 # Total area of Manyata Tech Park in square feet

number_of_tenants = 26 # Number of tenants

# Generate date range

date_range = pd.date_range(start=start_date, end=end_date, freq='D')

# Create DataFrame for date

data = pd.DataFrame(date_range, columns=['Date'])

# Tenant names and their corresponding industry types

tenant_industry_dict = {

'Alcatel Lucent Technologies': 'Telecommunications',
```

'Andritz': 'Engineering',

'ANZ': 'Finance',

'Cerner Health Care Solutions': 'Healthcare',

'Cognizant': 'IT Services',

'Colt Technologies': 'Technology',

'Data Craft India': 'Consulting',

'Fidelity': 'Finance',

'IBM': 'Engineering',

'Jurimatrix': 'Consulting',

'L and T': 'Manufacturing',

'Mavenir': 'Biotechnology',

'Monsanto Holdings': 'Telecommunications',

'Nokia Siemens Network India': 'Technology',

'Northern Operation Services': 'Outsourcing',

'Nvidia Graphics': 'Semiconductors',

'Stylus Commercial Services': 'IT Services',

'Target Operation': 'Consulting',

'Netscout System Software': 'Software',

'Software Software Software': 'Travel',

'Via.com': 'Software',

```
'TP Vision': 'Software',

'Justdial': 'Electronics',

'Philips': 'Technology',

'Novell': 'Software',

'TCS': 'Telecommunications', # Added TCS to ensure 26 tenants

}

# Ensure we have the correct number of tenants

if len(tenant_industry_dict) != number_of_tenants:

raise ValueError(f"Expected {number_of_tenants} tenants, but found {len(tenant_industry_dict)}.")

# Convert the dictionary to a DataFrame

tenant_data = pd.DataFrame(list(tenant_industry_dict.items()), columns=['Tenant', 'Industry_Type'])

# Assign random tenant and industry type to each date entry

data = data.assign(Tenant=np.random.choice(tenant_data['Tenant'], size=len(data)))

data = data.merge(tenant_data, on='Tenant', how='left')

# Randomly assign number of floors (1 to 10)

data['Number_of_Floors'] = np.random.randint(1, 11, size=len(data))

# Randomly assign number of employees

data['Number_of_Employees'] = np.random.randint(50, 500, size=len(data))

# Apply shutdown period: 2020-03-01 to 2021-12-31

shutdown_start = pd.to_datetime('2020-03-01')
```

```
shutdown_end = pd.to_datetime('2021-12-31')

data.loc[(data['Date'] >= shutdown_start) & (data['Date'] <= shutdown_end), 'Number_of_Employees'] = 0

# Set number of employees to 0 on weekends (Saturday and Sunday)

data['Day_of_Week'] = data['Date'].dt.dayofweek

data.loc[data['Day_of_Week'].isin([5, 6]), 'Number_of_Employees'] = 0 # 5 = Saturday, 6 = Sunday

# Generate acquired area for each tenant

acquired_area_per_tenant = np.random.rand(number_of_tenants)

acquired_area_per_tenant = (acquired_area_per_tenant / acquired_area_per_tenant.sum()) * total_area_sqft # in square feet

# Create a mapping for acquired area per tenant

area_mapping = dict(zip(tenant_data['Tenant'], acquired_area_per_tenant))

# Assign acquired area to each entry in the DataFrame based on the tenant

data['Acquired_Area'] = data['Tenant'].map(area_mapping)

# Calculate base electricity reading based on acquired area, number of floors, and number of employees

base_electricity_reading = (

(data['Number_of_Employees'] * 0.2) + # Example: each employee contributes to reading

(data['Acquired_Area'] / 1000 * 0.1) + # Area contribution to electricity reading (0.1 kWh per sqft)

(data['Number_of_Floors'] * 5) # Additional reading per floor

)

# Assign base electricity reading
```

```
data['Base_Electricity_Reading'] = base_electricity_reading

# Define industry type multipliers as a range (min, max)

industry_multiplier_ranges = {

'Manufacturing': (1.3, 1.7),

'Engineering': (1.3, 1.7),

'Healthcare': (1.2, 1.5),

'Finance': (1.1, 1.3),

'IT Services': (1.3, 1.7),

'Technology': (1.0, 1.2),

'Consulting': (0.9, 1.1),

'Biotechnology': (1.2, 1.3),

'Telecommunications': (0.9, 1.1),

'Semiconductors': (1.1, 1.3),

'Electronics': (1.4, 1.6),

'Outsourcing': (0.9, 1.1),

'Travel': (0.9, 1.1),

'Software': (1.2, 1.5),

}

# Apply the adjustments based on the industry type using the defined ranges

for industry, (min_multiplier, max_multiplier) in industry_multiplier_ranges.items():
```

```
random_multiplier = np.random.uniform(min_multiplier, max_multiplier)

data.loc[data['Industry_Type'] == industry, 'Base_Electricity_Reading'] *= random_multiplier

# Define weekday multipliers

day_multipliers = {

0: 0.5, # Monday

1: 1.0, # Tuesday

2: 1.0, # Wednesday

3: 1.0, # Thursday

4: (1.2, 1.5), # Friday (Increase on Fridays)

5: 0.0, # Saturday (No employees)

6: 0.0, # Sunday (No employees)

}

# Apply day multipliers to Base Electricity Reading

for day, multiplier in day_multipliers.items():

if isinstance(multiplier, tuple):

# For Friday, randomly select a multiplier within the specified range

random_multiplier = np.random.uniform(multiplier[0], multiplier[1])

data.loc[data['Day_of_Week'] == day, 'Base_Electricity_Reading'] *= random_multiplier

else:

data.loc[data['Day_of_Week'] == day, 'Base_Electricity_Reading'] *= multiplier
```

```
# Apply seasonal adjustments to electricity readings

data['Electricity_Reading'] = data['Base_Electricity_Reading']

seasonal_multiplier = 1 # Default multiplier

data['Month'] = data['Date'].dt.month

# Adjust reading based on season

data.loc[data['Month'].isin([2, 3, 4,5,6]), 'Electricity_Reading'] *= 2.0 # Highest in Feb-Apr

data.loc[data['Month'].isin([7, 8,]), 'Electricity_Reading'] *= 1.5 # Moderate in May-Sep

data.loc[data['Month'].isin([9,10]), 'Electricity_Reading'] *= 0.9 # Least in Oct-Nov

data.loc[data['Month'].isin([12,11,1]), 'Electricity_Reading'] *= 1.8 # More than moderate in Dec-Jan

# Add noise to electricity readings

data['Electricity_Reading'] += np.random.normal(0, 5, len(data))

# Ensure non-negative electricity readings

data['Electricity_Reading'] = data['Electricity_Reading'].clip(lower=0)

# Clean up the DataFrame

#data.drop(columns=['Day_of_Week', 'Month'], inplace=True) # Drop temporary columns

# Display the first few rows of the DataFrame

print(data.info())

# Optionally save to CSV

data.to_csv('manyata_tech_park_data.csv', index=False)
```

PreProcessing , Training –Testing

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import pickle

import numpy as np

import pandas as pd

from lightgbm import LGBMRegressor

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

manyata=pd.read_csv("/content/manyata_tech_park_data.csv")

manyata.info()

manyata['Date'] = pd.to_datetime(manyata['Date'],errors='coerce')

w=pd.read_csv("/content/weather_combined.csv")

w.info()

w['date']=pd.to_datetime(w['date'])

w.rename(columns={'date': 'Date'}, inplace=True)

merged_data = pd.merge(manyata, w, on='Date', how='left')
```

```
merged_data.info()

merged_data.head()

t=merged_data

t.isnull().sum()

def plot_distributions(df, columns):

"""

Plots distribution plots for the specified columns in a DataFrame.

Parameters:

df (DataFrame): The DataFrame containing the data.

columns (list): List of column names to plot distributions for.

"""

plt.figure(figsize=(15, 10)) # Set the figure size for better visibility

for i, col in enumerate(columns, 1):

plt.subplot(2, 3, i) # Create a subplot grid (2 rows, 3 columns)

sns.histplot(df[col], kde=True) # Plot the distribution with KDE curve

plt.title(f'Distribution of {col}')

plt.xlabel(col)

plt.ylabel('Frequency')

plt.tight_layout() # Adjust subplots to fit in the figure area

plt.show()
```

```
# List of columns to plot

columns_to_plot = ['tavg', 'tmin', 'tmax', 'prcp', 'Electricity_Reading']

# Call the function with the DataFrame and the list of columns

plot_distributions(t, columns_to_plot)

def fill_missing_values(df):

"""

Fill missing values for specific columns in the DataFrame with different imputation strategies.

Columns and Strategies:

- 'tavg': Fill with the median.

- 'tmax': Fill with the median.

- 'tmin': Fill with the mean.

Parameters:

df (DataFrame): The DataFrame with missing values to be filled.

Returns:

DataFrame: The DataFrame with missing values filled.

"""

df['tavg'].fillna(df['tavg'].median(), inplace=True)

df['tmax'].fillna(df['tmax'].median(), inplace=True)

df['tmin'].fillna(df['tmin'].mean(), inplace=True)

return df
```

```
# Call the function to fill missing values

t = fill_missing_values(t)

t.isnull().sum()

final=t

final.to_csv('final.csv', index=False)

encoder = LabelEncoder()

column_mapping={}

for column in ['Tenant', 'Industry_Type']:

t[column] = encoder.fit_transform(t[column])

column_mapping[column] = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))

t.corr()

tsa=t

t.drop(columns=['Base_Electricity_Reading','prcp','tmin','tmax'], inplace=True)

def plot_feature_relationships(df, target_column='Electricity_Reading'):

"""

Plots the correlation between each feature and the target column (Electricity_Reading) as a bar chart.

Parameters:

df (DataFrame): The DataFrame containing the features and target.

target_column (str): The target column to calculate correlation against.

"""
```

```
# Calculate the correlation of all columns with the target column

correlations = df.corr()[target_column].drop(target_column)

# Sort correlations by absolute value to show the strongest relationships at the top

correlations_sorted = correlations.abs().sort_values(ascending=False)

# Create a bar plot to show the relationships

plt.figure(figsize=(6, 8))

sns.barplot(x=correlations_sorted.index, y=correlations_sorted.values, palette='Blues_d')

# Add plot title and labels

plt.title(f'Correlation of Features with {target_column}', fontsize=16)

plt.xticks(rotation=90, fontsize=12)

plt.ylabel('Correlation Coefficient', fontsize=14)

plt.xlabel('Features', fontsize=14)

plt.tight_layout()

plt.show()

# Call the function to plot feature relationships with 'Electricity_Reading'

plot_feature_relationships(t, target_column='Electricity_Reading')

def detect_and_visualize_outliers(df, columns):

"""

Detects and visualizes outliers for specified columns using box plots.

Parameters:
```

df (DataFrame): The DataFrame containing the features to check for outliers.

columns (list): A list of column names for which to detect and visualize outliers.

Returns:

outliers_info (dict): A dictionary containing the number of outliers for each column.

"""

outliers_info = {}

for column in columns:

Calculate Q1 (25th percentile) and Q3 (75th percentile)

Q1 = df[column].quantile(0.25)

Q3 = df[column].quantile(0.75)

IQR = Q3 - Q1

Define outlier boundaries

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

Count outliers

outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]

outliers_info[column] = len(outliers)

Visualization using box plot

plt.figure(figsize=(4, 4))

sns.boxplot(x=df[column], palette='Set2')

```
plt.title(f'Outliers in {column}', fontsize=16)

plt.xlabel(column, fontsize=12)

plt.show()

return outliers_info

# Example usage:

columns_to_check = ['tavg','Number_of_Floors','Day_of_Week','Number_of_Employees','Acquired_Area'
,'Electricity_Reading',]

outliers_info = detect_and_visualize_outliers(t, columns_to_check)

# Output the number of outliers in each column

for col, count in outliers_info.items():

print(f'Total number of outliers in {col}: {count}')

def print_outliers(df, columns):

"""

Prints the outliers for each specified column based on the IQR method.

Parameters:

df (DataFrame): The DataFrame containing the features to check for outliers.

columns (list of str): A list of column names for which to detect and print outliers.

Returns:

outliers_dict (dict): A dictionary containing DataFrames with the outlier values for each column.

"""
```

```
outliers_dict = {}

for column in columns:

# Calculate Q1 (25th percentile) and Q3 (75th percentile)

Q1 = df[column].quantile(0.25)

Q3 = df[column].quantile(0.75)

IQR = Q3 - Q1

# Define outlier boundaries

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

# Filter out the outliers

outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]

# Store outliers in the dictionary

outliers_dict[column] = outliers[[column]]

# Print outlier values

print(f"Outliers in {column}:")

print(outliers[[column]])

print() # Add a blank line for better readability between columns

return outliers_dict

# Example usage:

columns = ['Electricity_Reading', 'tavg']
```

```
outliers_dict = print_outliers(t, columns)

t.drop(index=3389, inplace=True)

# Optionally, you can reset the index if needed

t.reset_index(drop=True, inplace=True)

temp=t.drop(columns=['Tenant','Date','Industry_Type'])

# Calculate skewness for relevant columns

skewness_info = {}

columns_to_check = ['Electricity_Reading','tavg']

for column in columns_to_check:

skewness_value = t[column].skew()

skewness_info[column] = skewness_value

# Display the skewness of each column

for column, skewness_value in skewness_info.items():

print(f"Skewness of {column}: {skewness_value}")

import numpy as np

# Applying square root transformation

t['Electricity_Reading'] = np.sqrt(t['Electricity_Reading'])

# Check the new skewness values

print("Skewness after Square Root Transformation:")

print("Electricity_Reading:", t['Electricity_Reading'].skew())
```

```
df=t

df=df.drop(columns=['Date'])

print(column_mapping)

x=df.drop(columns=['Electricity_Reading'])

y=df['Electricity_Reading']

x.head()

y.head()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

from lightgbm import LGBMRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear_model import LinearRegression

from sklearn.svm import SVR

import pickle

# Assuming 't' is your DataFrame
```

```
x = t[['Industry_Type', 'Number_of_Floors', 'Number_of_Employees', 'Day_of_Week', 'Acquired_Area', 'Month', 'tavg']]

y = t['Electricity_Reading']

# Train-Test Split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# List of models to evaluate

models = {

"LGBMRegressor": LGBMRegressor(learning_rate=0.1, n_estimators=150),

"RandomForestRegressor": RandomForestRegressor(n_estimators=150),

"LinearRegression": LinearRegression(),

"SVR": SVR()

}

# Store results

results = {

'Model': [],

'MSE': [],

'R-squared': []

}

# Train, predict and evaluate each model
```

```
for model_name, model in models.items():

# Train the model

model.fit(x_train, y_train)

# Predict

y_pred = model.predict(x_test)

# Evaluate

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

# Store results

results['Model'].append(model_name)

results['MSE'].append(mse)

results['R-squared'].append(r2)

print(f"{model_name}:")

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}\n")

# Convert results to DataFrame for visualization

results_df = pd.DataFrame(results)

# Visualization

plt.figure(figsize=(12, 6))
```

```
# Custom color palette

mse_palette = ['#FF9999', '#66B2FF', '#99FF99', '#FFCC66'] # Red, Blue, Green, Orange

r2_palette = ['#FF6666', '#6699FF', '#66FF66', '#FF9966'] # Darker shades for R-squared

# Plot MSE values

plt.subplot(1, 2, 1)

sns.barplot(x='Model', y='MSE', data=results_df, palette=mse_palette)

plt.title('Mean Squared Error (MSE) Comparison')

plt.xticks(rotation=45)

# Plot R-squared values

plt.subplot(1, 2, 2)

sns.barplot(x='Model', y='R-squared', data=results_df, palette=r2_palette)

plt.title('R-squared Comparison')

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()

tenant_mapping = {

0: 'ANZ', 1: 'Alcatel Lucent Technologies', 2: 'Andritz', 3: 'Cerner Health Care Solutions',

4: 'Cognizant', 5: 'Colt Technologies', 6: 'Data Craft India', 7: 'Fidelity', 8: 'IBM',

9: 'Jurimatrix', 10: 'Justdial', 11: 'L and T', 12: 'Mavenir', 13: 'Monsanto Holdings',
```

```
14: 'Netscout System Software', 15: 'Nokia Siemens Network India', 16: 'Northern Operation Services',

17: 'Novell', 18: 'Nvidia Graphics', 19: 'Philips', 20: 'Software Software Software',

21: 'Stylus Commercial Services', 22: 'TCS', 23: 'TP Vision', 24: 'Target Operation', 25: 'Via.com'

}

# Assuming 't' is your DataFrame containing all tenant data, and you want to use specific features

x = t[['Industry_Type', 'Number_of_Floors', 'Number_of_Employees', 'Day_of_Week', 'Acquired_Area', 'Month', 'tavg']] # Features

y = t['Electricity_Reading'] # Target (Electricity usage)

# Step 1: Train-Test Split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Step 2: Train the model

model = LGBMRegressor(learning_rate=0.1, n_estimators=150)

model.fit(x_train, y_train)

# Step 3: Save the model

with open('electricity_model2.pkl', 'wb') as file:

pickle.dump(model, file)

# Step 4: Load the model

with open('electricity_model2.pkl', 'rb') as file:
```

```
loaded_model = pickle.load(file)

# Step 5: Evaluate the model

y_pred = loaded_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

# Step 6: Predict electricity readings for October for each tenant

october_forecasts = []

for tenant_code in t['Tenant'].unique():

# Prepare October data for each tenant

october_data_tenant = pd.DataFrame({

'Industry_Type': [t[t['Tenant'] == tenant_code]['Industry_Type'].iloc[0]] * 31,

'Number_of_Floors': [t[t['Tenant'] == tenant_code]['Number_of_Floors'].iloc[0]] * 31,

'Number_of_Employees': [t[t['Tenant'] == tenant_code]['Number_of_Employees'].iloc[0]] * 31,

'Day_of_Week': np.tile(np.arange(1, 8), 31)[:31], # Days in October

'Acquired_Area': [t[t['Tenant'] == tenant_code]['Acquired_Area'].iloc[0]] * 31,

'Month': [10] * 31, # October

'tavg': [25] * 31 # Average temperature in October
```

```
})

# Make predictions for October

october_predictions_tenant = loaded_model.predict(october_data_tenant)

# Store the predictions

october_forecasts.append(pd.DataFrame({

'Tenant': [tenant_mapping[tenant_code]] * 31,

'Date': pd.date_range(start='2024-10-01', periods=31),

'Predicted_Electricity_Reading (kWh)': october_predictions_tenant

}))

# Combine all predictions for October into one DataFrame

october_forecasts = pd.concat(october_forecasts)

# Step 7: Display the results

print(october_forecasts)

october_forecasts.to_csv('october_tenant_forecasts_final.csv', index=False)

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming the same 'comparison_df' from earlier

# Group data by Date to visualize overall electricity usage across tenants
```

```
comparison_grouped = comparison_df.groupby('Date').sum().reset_index()

# Plot actual and predicted values across the dataset timeline

plt.figure(figsize=(12, 8))

# Plot the actual values

plt.plot(comparison_grouped['Date'], comparison_grouped['Actual_Electricity_Reading (kWh)'],

label='Actual', linestyle='--', color='blue')

# Plot the predicted values with transparency to avoid overlap issues

plt.plot(comparison_grouped['Date'], comparison_grouped['Predicted_Electricity_Reading
(kWh)'],

label='Predicted', color='red', alpha=0.6) # Adding transparency with alpha

# Customize the plot

plt.title('Overall Real vs Predicted Electricity Readings')

plt.xlabel('Date')

plt.ylabel('Electricity Reading (kWh)')

plt.xticks(rotation=45)

plt.legend(loc='upper right')

# Display the plot

plt.tight_layout()

plt.show()
```

```
# Calculate residuals

comparison_grouped['Residuals'] = comparison_grouped['Actual_Electricity_Reading (kWh)'] - comparison_grouped['Predicted_Electricity_Reading (kWh)']

# Plot residuals

plt.figure(figsize=(12, 6))

plt.scatter(comparison_grouped['Date'], comparison_grouped['Residuals'], color='purple', alpha=0.5)

plt.axhline(0, linestyle='--', color='black')

plt.title('Residuals Plot: Actual vs Predicted Electricity Readings')

plt.xlabel('Date')

plt.ylabel('Residuals (Actual - Predicted)')

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()

""" This plot shows the difference between the actual electricity readings and the predicted values. The ideal scenario is to have the residuals randomly scattered around zero, indicating that the model is not systematically over or underpredicting. **The graph shows that the residuals are mostly randomly scattered, but there are some patterns that suggest the model may not be perfect.**"""

plt.figure(figsize=(8, 6))

sns.histplot(comparison_grouped['Residuals'], bins=30, kde=True, color='green')
```

```
plt.title('Distribution of Residuals (Errors)')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.tight_layout()

plt.show()

"""Here are a few interpretations based on the plot:

**Symmetry**: The residuals are symmetrically distributed around zero, suggesting that the errors are fairly distributed across the predictions. This is a good sign for model accuracy.

**Bell-shaped curve**: The normality (bell-shaped curve) of the residuals means the model errors are mostly small and follow a normal distribution, which is what you generally aim for in regression models.

**Outliers**: There are some residuals on both tails of the distribution, but they appear to be relatively few, meaning the majority of your predictions are close to the actual values.

"""

plt.figure(figsize=(8, 6))

plt.scatter(comparison_grouped['Actual_Electricity_Reading (kWh)'],
comparison_grouped['Predicted_Electricity_Reading (kWh)'],

color='orange', alpha=0.5)

plt.plot([comparison_grouped['Actual_Electricity_Reading (kWh)'].min(),
comparison_grouped['Actual_Electricity_Reading (kWh)'].max()],
```

```
[comparison_grouped['Actual_Electricity_Reading (kWh)'].min(),
comparison_grouped['Actual_Electricity_Reading (kWh)'].max()],

color='red', linestyle='--') # Line showing perfect prediction

plt.title('Scatter Plot: Actual vs Predicted Electricity Readings')

plt.xlabel('Actual Electricity Reading (kWh)')

plt.ylabel('Predicted Electricity Reading (kWh)')

plt.tight_layout()

plt.show()

comparison_grouped['Actual_Rolling'] = comparison_grouped['Actual_Electricity_Reading
(kWh)'].rolling(window=30).mean()

comparison_grouped['Predicted_Rolling'] = comparison_grouped['Predicted_Electricity_Reading
(kWh)'].rolling(window=30).mean()

plt.figure(figsize=(12, 6))

plt.plot(comparison_grouped['Date'], comparison_grouped['Actual_Rolling'], label='Actual
Rolling Mean', color='blue')

plt.plot(comparison_grouped['Date'], comparison_grouped['Predicted_Rolling'], label='Predicted
Rolling Mean', color='red')

plt.title('Rolling Mean Comparison: Actual vs Predicted Electricity Readings')

plt.xlabel('Date')

plt.ylabel('Electricity Reading (kWh)')

plt.xticks(rotation=45)
```

```
plt.legend(loc='upper right')

plt.tight_layout()

plt.show()

"""This plot shows the actual and predicted electricity readings, smoothed using a rolling mean. This helps to visualize the overall trend and how the model is capturing the seasonal patterns. **The graph shows that the model is generally capturing the overall trend and seasonal patterns, although there are some discrepancies.**"""

comparison_grouped['Cumulative_Error'] = comparison_grouped['Residuals'].cumsum()

plt.figure(figsize=(10, 6))

plt.plot(comparison_grouped['Date'], comparison_grouped['Cumulative_Error'], color='purple')

plt.title('Cumulative Error Over Time')

plt.xlabel('Date')

plt.ylabel('Cumulative Error (kWh)')

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()

""" This plot shows the cumulative error of the model over time. It helps to visualize how the model is accumulating errors. **The graph shows that the cumulative error is fluctuating, but it does not seem to be diverging drastically, which is a good sign.**"""

# Create a column for year and week

comparison_grouped['Year'] = comparison_grouped['Date'].dt.year
```

```
comparison_grouped['Week'] = comparison_grouped['Date'].dt.isocalendar().week

# Pivot the data to create a heatmap of errors by week and year

error_pivot = comparison_grouped.pivot_table(values='Residuals', index='Week', columns='Year', aggfunc='mean')

plt.figure(figsize=(10, 8))

sns.heatmap(error_pivot, cmap='coolwarm', center=0)

plt.title('Heatmap of Prediction Errors Over Time')

plt.xlabel('Year')

plt.ylabel('Week of Year')

plt.tight_layout()

plt.show()
```

DAX CODE

- Energy_Per_Employee = SUM('final'[Electricity_Reading]) / SUM('final'[Number_of_Employees])

- Energy_Per_Square_Foot = SUM('final'[Electricity_Reading]) / SUM('final'[Acquired_Area])

- Last Month Filter =

```
                EOMONTH(TODAY(), -1) + 1
```

- Minimum Energy Consumption Last Month =

```
    CALCULATE(
    MIN(final[Electricity_Reading]),
    final[Date] >= EOMONTH(TODAY(), -2) + 1 &&
    final[Date] <= EOMONTH(TODAY(), -1)
```

Tenant with Minimum Energy Last Month =

```
    CALCULATE(
    FIRSTNONBLANK(final[Tenant], 1),
    FILTER(
    final,
    final[Electricity_Reading] = [Minimum Energy Consumption Last Month] &&
    final[Date] >= EOMONTH(TODAY(), -2) + 1 &&
    final[Date] <= EOMONTH(TODAY(), -1)
    )
    )
   )
```

CONCLUSION & FUTURE ENHANCEMENT

CONCLUSION

This project has successfully provided valuable insights into the electricity consumption patterns of tenants at Manyata Tech Park. By analyzing historical data, the project highlighted trends, seasonal variations, and other consumption behaviors that could influence decision-making for tenants. The forecasting model developed offers a data-driven approach for predicting future electricity usage, allowing tenants to better plan their consumption, optimize energy usage, and potentially reduce costs. These insights are particularly beneficial for property managers aiming to improve energy efficiency across the park, as well as for individual tenants striving for cost-effective solutions to meet sustainability goals.

The ability to predict electricity usage also helps in balancing the load more effectively, identifying potential peak demand periods, and enabling preventive measures to avoid over-consumption or outages. The use of machine learning techniques in this project opens up opportunities for more advanced energy management strategies, giving stakeholders a foundation for long-term energy planning and sustainability initiatives.

FUTURE ENHANCEMENTS

While the current version of the ENERGY iQ: Sustainable Energy Solution for Manyata Tech Park is functional and meets its primary objectives, several potential enhancements could further improve its capabilities and user experience. The following suggestions outline possible future developments:

1. Dynamic Dataset Integration (Real-Time Data)

- **IoT Integration**: Utilize IoT sensors across tenant buildings to collect real-time electricity usage data, which can feed into the system dynamically.

- **Automatic Data Ingestion**: Connect the system to live databases or APIs (like weather data APIs) to fetch real-time data on factors such as temperature, energy prices, or tenant occupancy. This can improve forecasting accuracy.

2. Deploy the Dashboard on the Web

- **Interactive Web Dashboard**: Host your Power BI dashboard on a platform like Power BI Service, allowing remote access and real-time interaction. You could also build a custom web application using frameworks like Flask or Django to deploy your dashboard, enabling customized user interfaces and interaction.
- **User Authentication and Role-based Access**: Allow various stakeholders (e.g., building management, tenants) to access specific parts of the dashboard based on their role, providing a more personalized and secure experience.
- **Mobile-Friendly Dashboard**: Optimize the dashboard for mobile devices so that users can track energy consumption on the go.

3. Incorporate Multiple Energy Sources

- **Extend to Other Energy Types**: Add energy types such as **fuel**, **solar power**, or **battery storage** to the dataset, and predict their consumption alongside electricity. This can help provide a holistic view of overall energy usage and efficiency.
- **Energy Mix Analysis**: Allow the system to provide insights on the mix of energy sources being consumed (e.g., electricity vs. fuel) and show how shifts in energy mix impact costs and environmental factors.

4. Predictive Maintenance for Energy Systems

- **Equipment Monitoring and Failure Prediction**: Implement predictive maintenance for energy systems, using data like past energy consumption patterns and equipment health to predict when systems may need repair or replacement. This can help avoid costly downtimes.

- **Anomaly Detection**: Incorporate machine learning models to detect anomalies in energy consumption, which may indicate faulty equipment or inefficient energy use.

5. Carbon Footprint & Sustainability Reporting

- **Carbon Emission Tracking**: Integrate a **carbon footprint calculator** that uses electricity and fuel consumption data to estimate the environmental impact of the tech park's energy usage.
- **Sustainability Targets**: Implement dashboards that allow tenants and building managers to set and track sustainability targets (e.g., reducing carbon emissions by 10% over 5 years).

6. Cost Prediction and Budgeting

- **Cost Forecasting**: In addition to electricity usage, add a module to predict the **future costs of energy consumption**, taking into account variable energy prices and tenant demand. This can help management teams with budgeting and financial planning.
- **Energy Cost Optimization**: Provide suggestions on how to optimize energy consumption across the park, such as shifting high-energy activities to off-peak hours to reduce costs.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Books Referred:

1. Software Engineering by Roger Pressman
2. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Geron Aurelien
3. Machine Learning using Python by Manaranjan Pradhan , U Dinesh Kumar
4. Practical Time Series Forecasting with R: A Hands-On Guide by Galit Shmueli
5. Python for Data Analysis by Wes McKinney
6. Forecasting: Principles and Practice by Rob J. Hyndman & George Athanasopoulos
7. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron
8. Storytelling with Data by Cole Nussbaumer Knaflic
9. Time Series Analysis and Its Applications by Robert H. Shumway and David S. Stoffer
10. Data Science for Business by Foster Provost and Tom Fawcett

2. Websites referred:

1. Coursera (Machine Learning and Time Series Forecasting) www.coursera.org
2. Kaggle (Datasets and Competitions) www.kaggle.com
3. Towards Data Science (Tutorials and Case Studies) www.towardsdatascience.com
4. DataCamp (Online Data Science and Machine Learning Courses www.datacamp.com
5. Analytics Vidhya (Data Science Blog and Community www.analyticsvidhya.com
6. Stack Overflow (Coding Questions and Solutions) www.stackoverflow.com
7. Manyata Buisness Park
   https://www.embassyofficeparks.com/ourportfolio/bangalore/embassy-manyata/
8. ChatGPT

   https://chatgpt.com/c/670566e9-25d8-8002-a2a5-7d70daf34dc5