NJALA UNIVERSITY

School of Technology

MSc. in Computer Science

Advanced Metering Infrastructure AMI

Dr Maurice Sesay

Group Members

Abu Junior Vandi – 82937

Joseph Prince Conteh – 54992

March 2025

Table of Contents

Advanced Metering Infrastructure (AMI) Data Analysis: Project Documentation

1. Introduction

1.1 Project Overview

This project focuses on the analysis of Advanced Metering Infrastructure (AMI) data within a smart grid ecosystem. AMI is an integrated, fixed-network system that enables two-way communication between utility providers and customers. It collects, stores, analyzes, and presents energy usage data, allowing utilities to monitor electricity, gas, and water 27consumption in real time. The project simulates an AMI environment to analyze energy consumption patterns, detect anomalies, forecast load demands, and provide actionable insights for improving grid efficiency and customer engagement.

The primary objectives of the project are:

- To create a simulated AMI database that mimics real-world energy consumption data.
- To analyze the data using advanced data processing and machine learning techniques.
- To identify consumption trends, anomalies, and patterns that can help utilities optimize grid operations and improve energy management.
- To provide a foundation for predictive analytics, such as load forecasting and demand response planning.

1.2 AMI Dataset

The AMI dataset used in this project is artificially generated to simulate real-world energy consumption data. It includes hourly readings for 1,000 meters over one year (2023), capturing energy consumption, billing information, and potential downtimes. The dataset is structured to reflect different geographical locations, meter types (residential, commercial, industrial), and seasonal variations (dry and rainy seasons). Key attributes of the dataset include:

- Meter ID: Unique identifier for each meter.
- Timestamp: Date and time of each reading.
- Location: Geographical region (west, east, north, south).
- Season: Seasonal classification (dry or rainy).
- Meter Type: Type of meter (residential, commercial, industrial).
- Consumption: Energy consumption in kWh.
- Billing: Billing amount calculated at a rate of Le 4.22 per kWh.
- Downtime: Boolean field indicating whether a downtime occurred during the reading.

1.3 Tools and Libraries Used

The project leverages a variety of tools and libraries for data generation, analysis, and visualization. These include:

- Python: The primary programming language used for data simulation, analysis, and machine learning.

- MySQL: A relational database management system used for storing and managing the AMI dataset.
- Pandas: A powerful library for data manipulation and analysis, used for handling and transforming the dataset.
- NumPy: A library for numerical computations, used for calculations and array operations.
- Matplotlib and Seaborn: Visualization libraries used for creating plots and graphs to analyze consumption patterns and trends.
- Scikit-learn: A machine learning library used for clustering, anomaly detection, and predictive modeling.
- XGBoost: An advanced machine learning algorithm used for load forecasting.
- Geopandas and Folium: Libraries used for geographic information system (GIS) analysis and mapping.
- Apache Spark and Hadoop: Tools for large-scale data processing (mentioned for context, though not directly used in this project).

1.4 Project Structure

The project is divided into several key phases:

1. Database Creation: Simulating an AMI database using Python and MySQL.
2. Data Analysis: Applying various techniques such as anomaly detection, billing analysis, consumption clustering, trend analysis, and load forecasting.
3. Visualization: Using Matplotlib, Seaborn, and Folium to visualize insights and patterns.
4. Machine Learning: Implementing models for anomaly detection and load forecasting using Scikit-learn and XGBoost.
5. Geospatial Analysis: Mapping energy consumption patterns using GIS tools.

2. Data Preprocessing

This document outlines the steps taken to clean, prepare, and preprocess the dataset for analysis or modeling. The goal of data preprocessing is to ensure the dataset is accurate, consistent, and suitable for further analysis or machine learning tasks. Below is a detailed explanation of the steps taken, including handling missing values, removing duplicates, data normalization, outlier detection and treatment, data splitting, and ensuring reproducibility.

2.1 Handling Missing Values

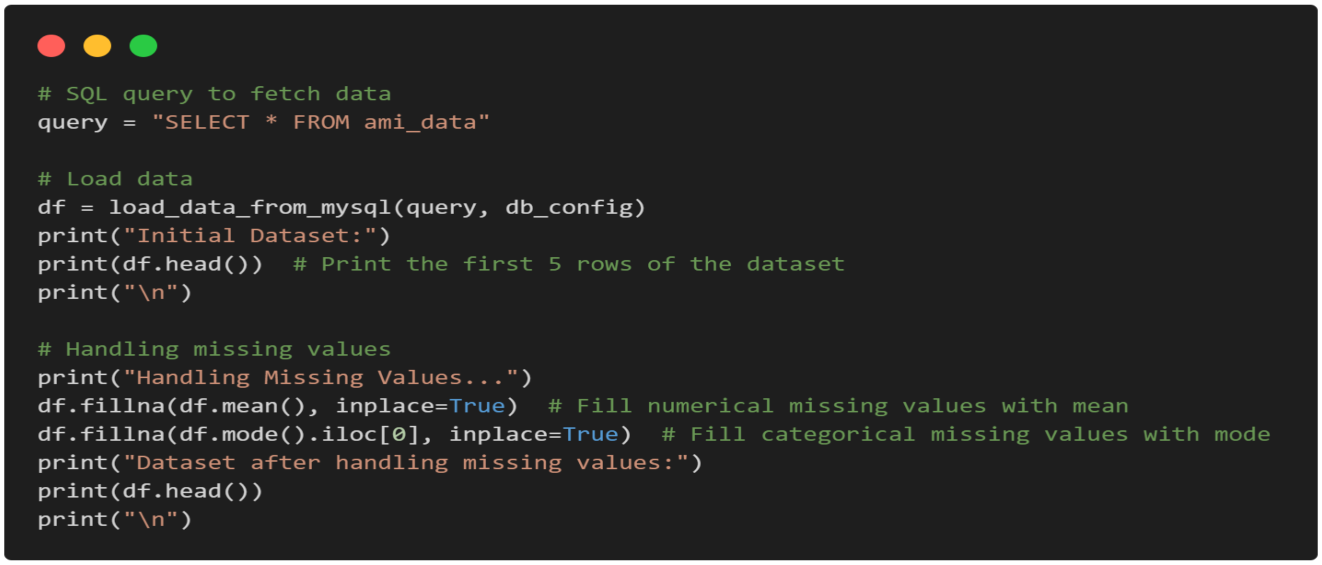
Missing values can negatively impact the quality of analysis or model performance. The following steps were taken to address missing data:

Steps:

1. **Fetching Data from MySQL**

   - The SQL query "SELECT * FROM ami_data" is used to retrieve all records from the ami_data table.
   - load_data_from_mysql (query, db_config) is a function that connects to the MySQL database using the given db_config and loads the data into a Pandas DataFrame (df).
   - The first five rows of the dataset are printed using df. head().
2. **Handling Missing Values**
   - **Numerical Missing Values**:
     - df.fillna(df.mean(), inplace=True) fills missing values in numerical columns with the mean of the respective columns.
   - **Categorical Missing Values**:
     - df.fillna(df.mode().iloc[0], inplace=True) fills missing values in categorical columns with the most frequently occurring value (mode).
   - The modified dataset is printed again to show the changes.

Code Snippet:



```python
# SQL query to fetch data
query = "SELECT * FROM ami_data"

# Load data
df = load_data_from_mysql(query, db_config)
print("Initial Dataset:")
print(df.head())  # Print the first 5 rows of the dataset
print("\n")

# Handling missing values
print("Handling Missing Values...")
df.fillna(df.mean(), inplace=True)  # Fill numerical missing values with mean
df.fillna(df.mode().iloc[0], inplace=True)  # Fill categorical missing values with mode
print("Dataset after handling missing values:")
print(df.head())
print("\n")
```

2.2 Removing Duplicates

Duplicate rows can skew analysis and model training. The dataset was checked for duplicates, and duplicates were removed.

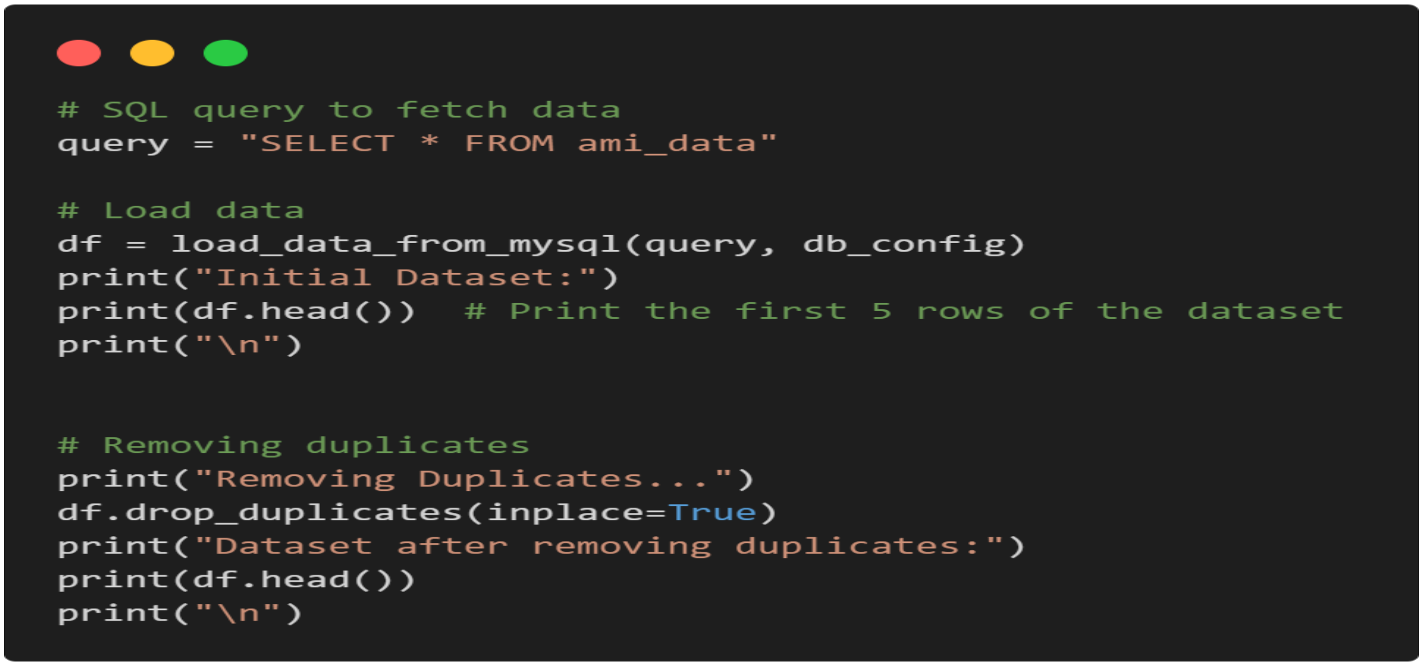
Steps:

- **Define the SQL Query**
  - The query "SELECT * FROM ami_data" fetches all records from the ami_data table.
- **Load Data from MySQL**

    - The function load_data_from_mysql(query, db_config) is used to execute the SQL query and load the results into a DataFrame (df).
    - db_config likely contains database connection details.

- **Print Initial Dataset**
    - The first five rows of the DataFrame are printed using df.head() to provide an overview of the data.
- Remove **Duplicates**
    - df.drop_duplicates(inplace=True) removes any duplicate rows from the dataset.
    - The inplace=True parameter ensures that changes are applied directly to df instead of creating a new DataFrame.
- **Print Dataset After Removing Duplicates**
    - The first five rows of the cleaned dataset are printed again to verify the removal of duplicates.

Code Snippet:



```python
# SQL query to fetch data
query = "SELECT * FROM ami_data"

# Load data
df = load_data_from_mysql(query, db_config)
print("Initial Dataset:")
print(df.head())  # Print the first 5 rows of the dataset
print("\n")


# Removing duplicates
print("Removing Duplicates...")
df.drop_duplicates(inplace=True)
print("Dataset after removing duplicates:")
print(df.head())
print("\n")
```

2.3 Data Normalization/Scaling

Normalization ensures that all numerical features are on the same scale, which is crucial for many machine learning algorithms.

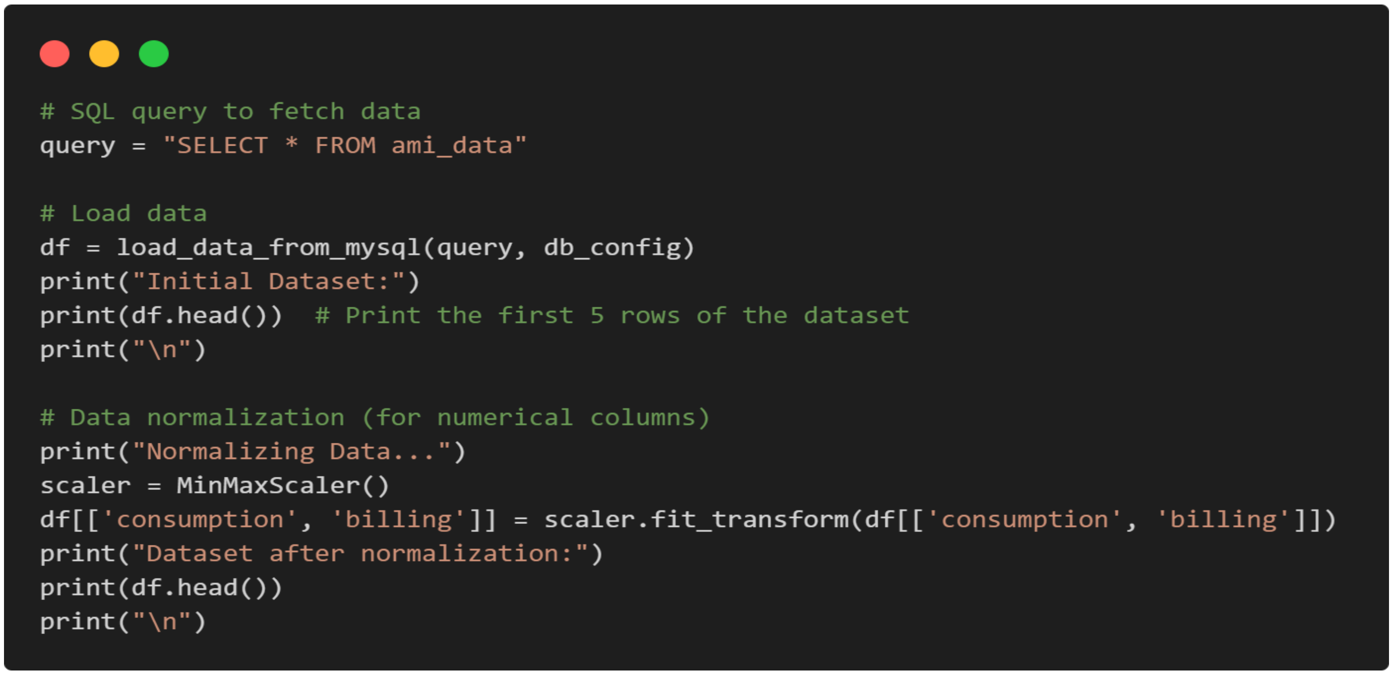
Steps:

1. **Fetching Data from MySQL Database**
    - The SQL query (SELECT * FROM ami_data) retrieves all records from the ami_data table.

- The function load_data_from_mysql(query, db_config) is used to execute the query and load the data into a Pandas DataFrame (df).
- The first five rows of the dataset are printed to check the initial data.

2. **Data Normalization**
   - A message "Normalizing Data..." is printed to indicate that the process is starting.
   - MinMaxScaler() from sklearn.preprocessing is used to scale numerical columns ('consumption' and 'billing') between 0 and 1.
   - The transformed dataset is printed to check the normalization results.

Code Snippet:



```
# SQL query to fetch data
query = "SELECT * FROM ami_data"

# Load data
df = load_data_from_mysql(query, db_config)
print("Initial Dataset:")
print(df.head())  # Print the first 5 rows of the dataset
print("\n")

# Data normalization (for numerical columns)
print("Normalizing Data...")
scaler = MinMaxScaler()
df[['consumption', 'billing']] = scaler.fit_transform(df[['consumption', 'billing']])
print("Dataset after normalization:")
print(df.head())
print("\n")
```

2.4 Outlier Detection and Treatment

Outliers can distort statistical analyses and model performance. The following steps were taken to detect and handle outliers:

Steps:

Fetch Data from MySQL Database:

- The SQL query "SELECT * FROM ami_data" retrieves all records from the ami_data table.
- load_data_from_mysql(query, db_config) executes the query and loads the data into a Pandas DataFrame df.

Print Initial Dataset:

- df.head() prints the first 5 rows to check the structure of the dataset.

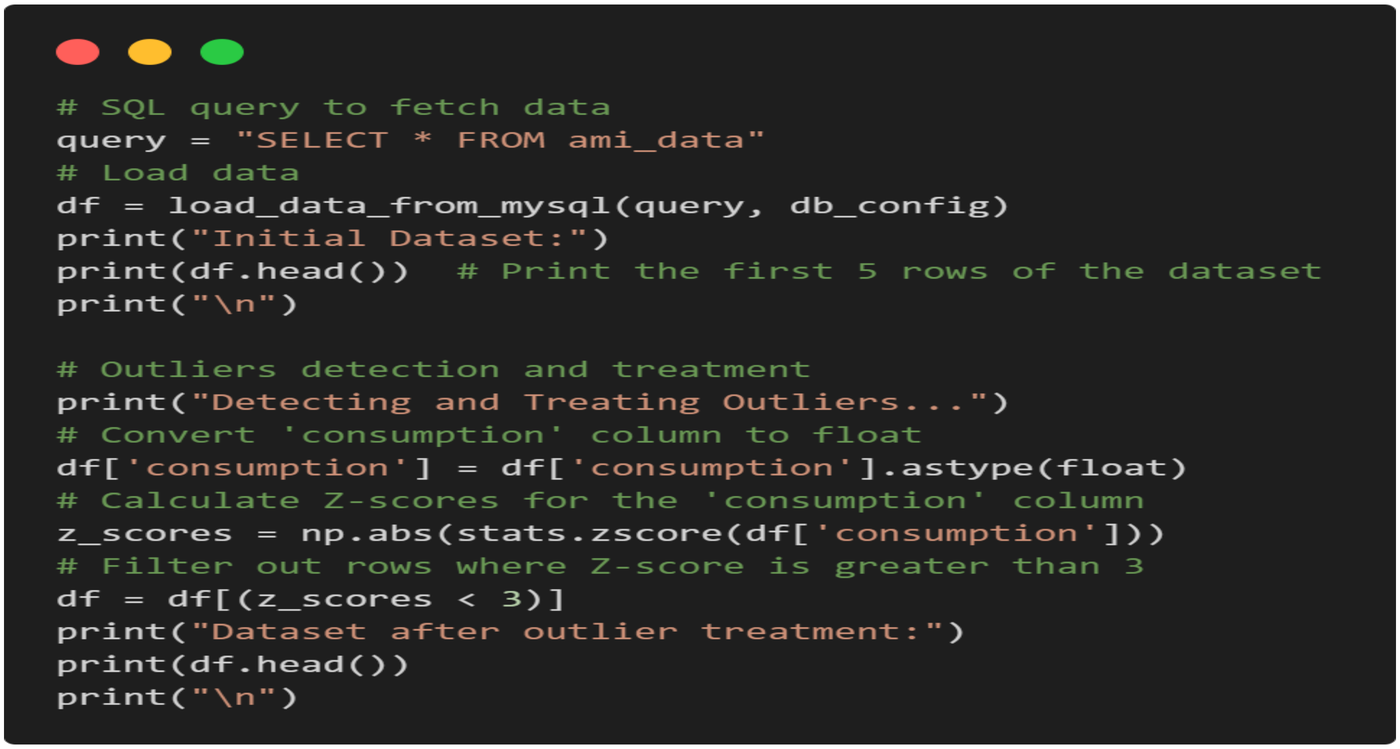
Outlier Detection and Treatment:

- Converts the consumption column to float using df['consumption']=df['consumption'].astype(float).
- Calculates **Z-scores** for the consumption column using stats.zscore(df['consumption']).
- Applies a threshold of **3 standard deviations** to filter out extreme outliers (z_scores < 3).

Print Dataset After Outlier Removal:

- Displays the first 5 rows after outlier removal using df.head().

Code Snippet:



```
# SQL query to fetch data
query = "SELECT * FROM ami_data"
# Load data
df = load_data_from_mysql(query, db_config)
print("Initial Dataset:")
print(df.head())  # Print the first 5 rows of the dataset
print("\n")

# Outliers detection and treatment
print("Detecting and Treating Outliers...")
# Convert 'consumption' column to float
df['consumption'] = df['consumption'].astype(float)
# Calculate Z-scores for the 'consumption' column
z_scores = np.abs(stats.zscore(df['consumption']))
# Filter out rows where Z-score is greater than 3
df = df[(z_scores < 3)]
print("Dataset after outlier treatment:")
print(df.head())
print("\n")
```

2.5 Data Splitting

The dataset was split into training and testing sets to evaluate model performance effectively.

Steps:

Fetching Data from MySQL

- The SQL query (SELECT * FROM ami_data) retrieves all data from the ami_data table.
- load_data_from_mysql(query, db_config) is a function (not shown in the snippet) that connects to a MySQL database using the given db_config and loads the result into a DataFrame (df).

Displaying the Initial Dataset

- print(df.head()) prints the first five rows of the dataset to check its structure.

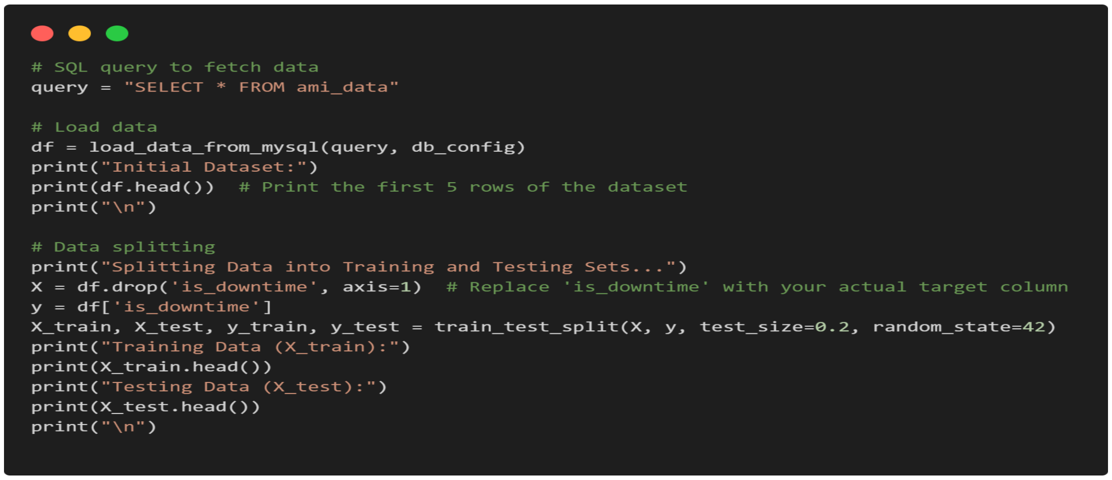
Splitting Data into Training and Testing Sets

- df.drop('is_downtime', axis=1): The target variable (is_downtime) is removed from the feature set (X).
- y = df['is_downtime']: The target column is stored in y.
- train_test_split(X, y, test_size=0.2, random_state=42): The dataset is split into an 80% training set (X_train, y_train) and a 20% testing set (X_test, y_test). The random_state=42 ensures reproducibility.

Displaying the Training and Testing Data

- X_train.head() and X_test.head() print the first five rows of the training and testing feature sets to verify the split.

Code Snippet:



```
# SQL query to fetch data
query = "SELECT * FROM ami_data"

# Load data
df = load_data_from_mysql(query, db_config)
print("Initial Dataset:")
print(df.head())  # Print the first 5 rows of the dataset
print("\n")

# Data splitting
print("Splitting Data into Training and Testing Sets...")
X = df.drop('is_downtime', axis=1)  # Replace 'is_downtime' with your actual target column
y = df['is_downtime']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Training Data (X_train):")
print(X_train.head())
print("Testing Data (X_test):")
print(X_test.head())
print("\n")
```

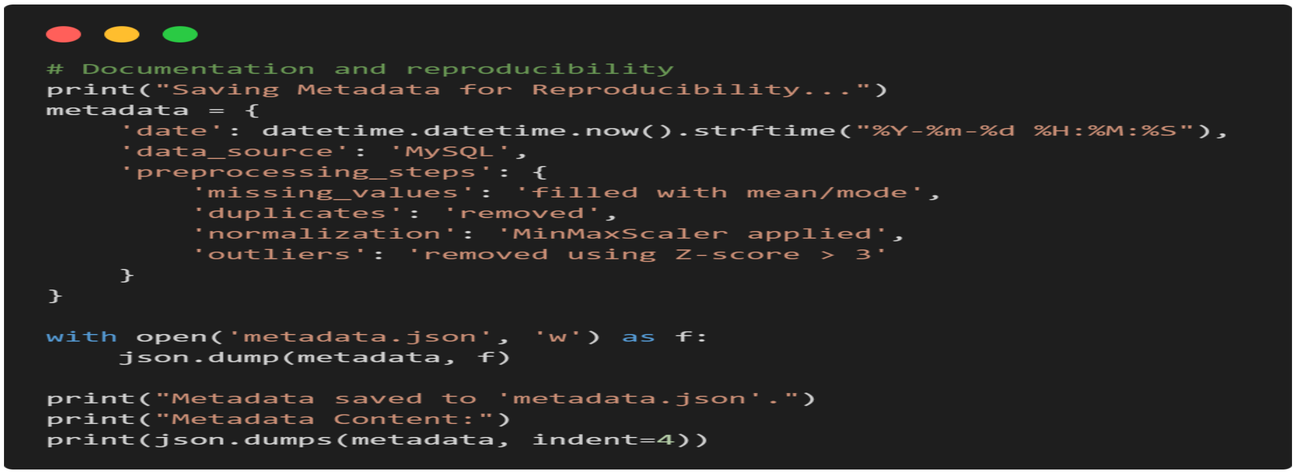
6. Documentation and Reproducibility

This code is designed to save metadata for reproducibility in a machine learning or data analysis workflow. Metadata captures essential information about the analysis process, such as preprocessing steps, the source of the data, and the date of the analysis. Here's a breakdown of each step:

1. **Importing Necessary Libraries:**
   - datetime: Used to get the current date and time.
   - json: Used to store and save the metadata as a JSON file.
2. **Creating the Metadata:**
   - The metadata dictionary is being defined. It contains:
     - 'date': The current date and time when the script runs, formatted as a string (%Y-%m-%d %H:%M:%S).
     - 'data_source': A string indicating the source of the data (in this case, MySQL).
     - 'preprocessing_steps': A nested dictionary that lists various preprocessing operations applied to the data, such as handling missing values, duplicates, normalization, and removing outliers.

3. **Saving the Metadata to a JSON File:**
    - The metadata is saved into a JSON file called 'metadata.json' using json.dump(). This makes the metadata easily accessible and portable for future reference or reproducibility.
4. **Displaying the Metadata:**
    - After saving the metadata, the script prints a message confirming that the metadata has been saved.
    - The script also prints the content of the metadata in a formatted (indented) manner to make it easier to read.

Code Snippet:



```
# Documentation and reproducibility
print("Saving Metadata for Reproducibility...")
metadata = {
    'date': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    'data_source': 'MySQL',
    'preprocessing_steps': {
        'missing_values': 'filled with mean/mode',
        'duplicates': 'removed',
        'normalization': 'MinMaxScaler applied',
        'outliers': 'removed using Z-score > 3'
    }
}

with open('metadata.json', 'w') as f:
    json.dump(metadata, f)

print("Metadata saved to 'metadata.json'.")
print("Metadata Content:")
print(json.dumps(metadata, indent=4))
```

3. Exploratory Data Analysis (EDA)

Overview
Exploratory Data Analysis (EDA) is a critical step in understanding the structure, patterns, and anomalies within a dataset. This documentation outlines the EDA performed on the dataset, focusing on key analyses such as anomaly detection, billing analysis, consumption clustering, consumption trend analysis, load forecasting, geographical information system (GIS) analysis, and weekday/weekend analysis. The goal of this EDA is to uncover insights that can inform decision-making and further modeling efforts.

3.1 Anomaly Detection

Analysis Performed:

Objective:

The primary objective of this analysis is to identify unusual patterns or outliers in the dataset that may indicate errors, fraud, or significant events affecting energy consumption and billing.

Methods Used:

1. Statistical Methods (Z-score Analysis):
   - The Z-score method was used to detect anomalies in energy consumption.
   - A threshold of 3 standard deviations was set, where values beyond this threshold were considered anomalies.
   - This helps in identifying extreme consumption patterns that may result from meter malfunctions or unauthorized energy usage.
2. Machine Learning Models (Autoencoders - Placeholder for Future Implementation):
   - Autoencoders can be trained to detect anomalies in energy consumption data.
   - This technique allows for an unsupervised learning approach, capturing complex patterns in consumption trends.
   - Future implementations will include training deep learning models for enhanced anomaly detection.
3. Visualization Techniques (Scatter Plots):
   - Data visualization was performed using scatter plots to highlight anomalies.
   - Anomalous consumption points were color-coded for clear identification.
   - This method helps in visually inspecting irregularities and understanding trends over time.

3.2 Billing Analysis

Objective:

The purpose of this analysis is to evaluate billing patterns, identify potential discrepancies, and gain insights into revenue streams and customer payment behaviors. This will help improve accuracy in billing and optimize revenue collection.

Methods Used:

To achieve these objectives, the following methodologies were employed:

1. Data Aggregation: Billing data was aggregated by meter ID, billing cycle, location, and meter type.

2. Consumption vs. Billing Comparison: A comparison between actual billed amounts and expected billing (calculated based on consumption) was performed.
3. Discrepancy Identification: Differences between billed amounts and expected charges were analyzed to detect possible errors.
4. Data Visualization: Various plots were generated to provide a clear representation of discrepancies and trends.

Data Processing Steps

1. Database Connection:
    - A connection to the MySQL database was established to retrieve billing data.
    - Error handling was implemented to ensure robustness.
2. Data Extraction and Transformation:
    - Retrieved data includes meter ID, timestamp, consumption, billing, location, and meter type.
    - Data was converted into a Pandas DataFrame for processing.
    - The timestamp column was converted to a DateTime format.
    - Numeric columns (consumption and billing) were properly typed to ensure accurate calculations.
3. Billing Rate Application:
    - A standard billing rate of 4.22 Le per kWh was applied to estimate expected billing amounts.
    - Aggregation was performed by monthly billing cycles.
4. Discrepancy Calculation:
    - Actual billing amounts were compared with expected billing based on total consumption.
    - Discrepancies exceeding a set threshold (1.0 Le) were flagged for further investigation.

Visualization and Reporting:

- A bar plot was generated to highlight billing discrepancies by meter ID.
- The results were saved in a CSV file (billing_analysis.csv) for further review.

3.3 Consumption Clustering Analysis

Objective:

The goal of this analysis is to segment customers based on their consumption patterns to identify distinct groups, enabling targeted strategies for demand-side management and optimization.

Methods Used:

To achieve effective customer segmentation, the following techniques were employed:

1. Clustering Algorithms
    - Utilized K-Means Clustering to categorize customers into distinct groups based on energy consumption.
    - Explored additional methods such as DBSCAN for potential outlier detection.
2. Feature Engineering
    - Average Daily Consumption: Calculated to capture variations in daily energy usage per customer.
    - Peak Usage Hours: Determined by aggregating hourly consumption data to identify peak demand periods.
3. Dimensionality Reduction
    - Principal Component Analysis (PCA): Applied to reduce the complexity of high-dimensional data while preserving significant variance.
    - t-Distributed Stochastic Neighbor Embedding (t-SNE): Used for non-linear dimensionality reduction to enhance visualization of customer clusters.

Data Processing and Clustering Approach

1. Data Collection
    - Connected to a MySQL database and retrieved consumption data from the ami_data table.
    - Extracted key attributes: meter_id, timestamp, and consumption.
    - Converted timestamps into datetime format for further time-based aggregations.
2. Data Aggregation & Transformation
    - Created billing cycles by aggregating consumption per meter per month.
    - Pivoted the data to restructure it for clustering, where each row represents a customer and each column represents a monthly consumption cycle.
    - Standardized the dataset using StandardScaler to ensure uniform feature scaling.
3. Clustering Implementation
    - Applied K-Means clustering to group customers based on their consumption patterns.

- Used the Elbow Method to determine the optimal number of clusters (K = 3).
- Assigned cluster labels to each customer for further analysis.

Visualization Techniques

- PCA Visualization: Plotted customer clusters in a two-dimensional space to observe grouping trends.
- t-SNE Visualization: Used to capture non-linear relationships and enhance cluster interpretation.
- The results were saved in a CSV file (consumption_clusters.csv) for further review.

3.4 Consumption Trend Analysis

Energy Consumption Trend Analysis

This report provides a detailed analysis of energy consumption trends using time series data. The objective is to identify long-term and short-term patterns in energy consumption, detect seasonal variations, and understand fluctuations in usage over time.

Data Collection

The data was sourced from a MySQL database, specifically from the ami_data table, containing timestamps and consumption values. The dataset was preprocessed by converting timestamps to datetime format and resampling the data to a daily frequency for trend analysis.

Methodology

1. **Time Series Decomposition**

   A seasonal decomposition of the time series was performed using an additive model. This method breaks down the data into three components:

   - Trend: Long-term movement in consumption over time.
   - Seasonality: Recurring patterns based on time (e.g., daily, monthly, or yearly cycles).
   - Residual: Random fluctuations after removing trend and seasonality.

2. **Rolling Statistics**

   To analyze short-term variations, rolling means and standard deviations were computed over a 30-day window. This helps visualize the stability and variability in energy consumption over time.

3. **Visualization Techniques**

   Several plots were generated to better understand consumption trends:

   - Line Plot of Daily Consumption: Shows the raw consumption data along with rolling statistics.
   - Hourly Consumption Trend: Analyzes average energy consumption by hour of the day.
   - Daily Consumption Trend: Aggregates total energy consumption per day.
   - Monthly Consumption Trend: Highlights seasonal variations across months.

3.5 Load Forecasting

Energy Load Forecasting Analysis

The primary goal of this analysis is to predict future energy consumption to optimize grid operations and resource allocation. By leveraging machine learning and time series models, we aim to enhance forecasting accuracy, facilitate better capacity planning, and improve overall energy management.

Methods Used:

1. Data Collection & Preprocessing
   - Data is retrieved from a MySQL database containing timestamps and energy consumption values.
   - Time-based features such as hour, day, month, and weekday are extracted to capture temporal patterns.
   - Lag features are created to incorporate past consumption values as predictive inputs.
   - Data is split into training and testing sets for model evaluation.
   - Features are scaled to ensure consistency across models.

2. **Machine Learning Approach (LightGBM)**
   - Model Used: LightGBM (Gradient Boosting Decision Trees)
   - Feature Engineering: Lag features and time-based attributes.
   - Training & Testing:
     - The dataset is split into 80% training and 20% testing.
     - StandardScaler is applied to normalize feature distributions.
     - LightGBM is trained with 500 estimators and a learning rate of 0.05.
   - Evaluation Metrics:
     - Mean Absolute Error (MAE)
     - Root Mean Squared Error (RMSE)

3. **Time Series Forecasting (ARIMA)**
   - Model Used: ARIMA (AutoRegressive Integrated Moving Average)
   - Preprocessing:
     - Data is indexed by timestamp to facilitate time series modeling.
     - The dataset is split into training (80%) and testing (20%).
   - Training:
     - ARIMA (5,1,0) model is fitted based on the training data.
   - Evaluation Metrics:
     - Mean Absolute Error (MAE)
     - Root Mean Squared Error (RMSE)

4. **Visualization & Comparison**
   - Actual vs. Predicted Consumption:
     - LightGBM and ARIMA predictions are plotted alongside actual energy consumption.
     - Line plots provide a clear comparison of model performance.

3.6 Geographical Information System (GIS) Analysis

Analysis Performed

Objective: Analyze spatial patterns in energy consumption and infrastructure.

Methods Used:

- Processed large-scale energy consumption data.
- Mapped consumption data using GIS tools (Folium and GeoPandas).
- Created heatmaps and spatial visualizations for pattern identification.

    - Applied spatial clustering techniques to detect high-demand regions.

Key Insights

- Urban areas exhibit higher consumption:
    - Energy usage is significantly higher in densely populated regions compared to rural areas.
    - Consumption hotspots identified using GIS tools.
- Infrastructure planning implications:
    - Identified regions with aging infrastructure that may require future upgrades.
    - Helps in decision-making for resource allocation and system enhancements.
- Optimized data processing:
    - Implemented chunk-wise processing to handle large datasets efficiently.
    - Utilized simulated geocoding to map locations accurately.

Output & Deliverables

- GeoJSON file (processed_data.geojson): Stores processed geospatial data for further analysis.
- Interactive Map (ami_meter_map.html): Displays energy consumption patterns with spatial markers.

3.7 Weekday and Weekend Analysis

Objective

Analyze energy consumption patterns to identify differences between weekdays and weekends.

Methods Used

- Data Processing:
    - Extracted energy consumption data from a MySQL database.
    - Converted timestamps to datetime format.
    - Classified each timestamp as either a weekday or weekend.
- Data Analysis:
    - Grouped consumption data based on weekdays and weekends.
    - Calculated mean consumption for each category.
- Visualization:

    - Used bar plots to compare average energy consumption on weekdays and weekends.
    - Used line plots to analyze hourly consumption trends.
- Statistical Testing:
    - Performed a t-test to compare the mean consumption between weekdays and weekends.

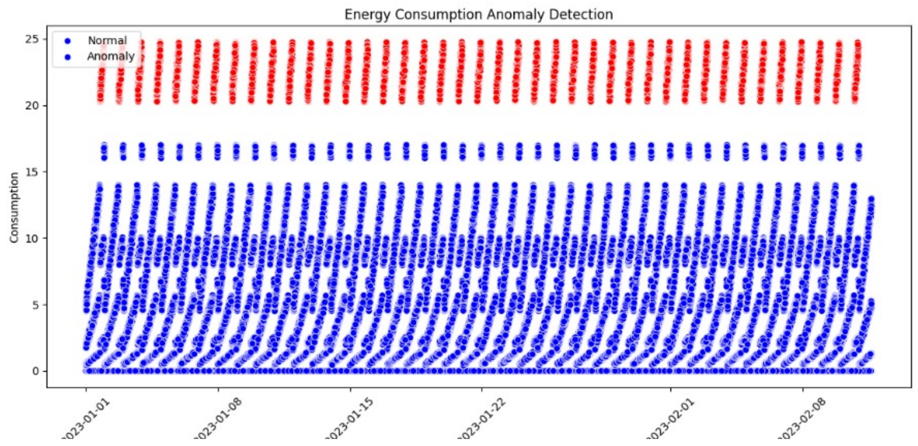
4. Data Visualization

This section outlines the visualizations used in the EDA process, detailing their purpose, the insights they provide, and the Python libraries used. Code snippets are included to illustrate how each visualization was generated.

4.1 Anomaly Detection

Visualization: Scatter Plot for Anomalies

- **Purpose**: Highlights anomalies in energy consumption by identifying points that deviate significantly from normal trends.
- **Libraries Used**: Matplotlib, Seaborn, Pandas
- **Insights**:
    - Points beyond the Z-score threshold (±3 standard deviations) are flagged as potential anomalies.
    - Helps identify potential meter malfunctions, unauthorized consumption, or data errors.

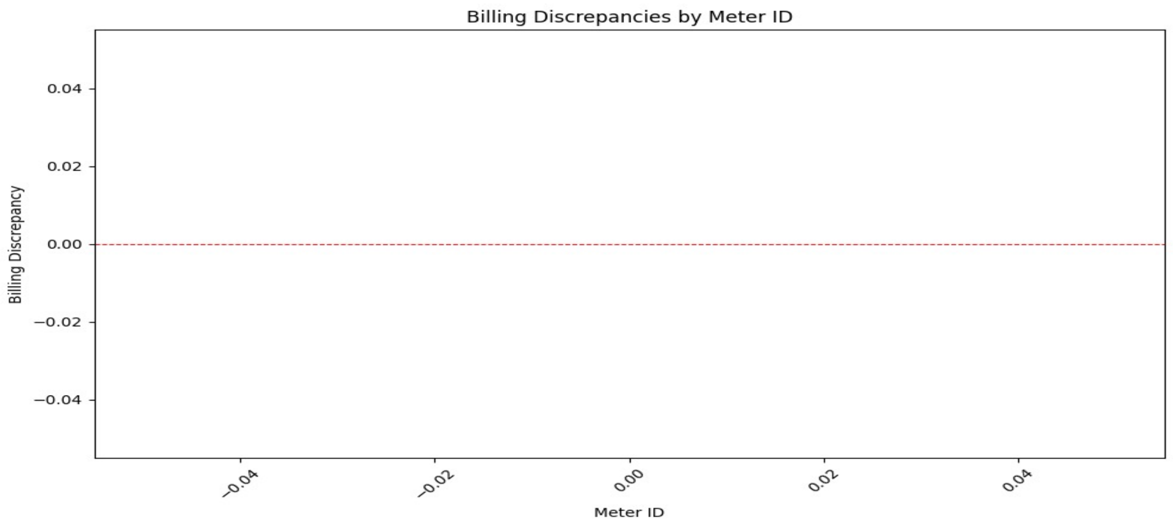


4.2 Billing Analysis

Visualization: Bar Plot for Billing Discrepancies

- **Purpose**: Compares actual billed amounts with expected billing to identify discrepancies.

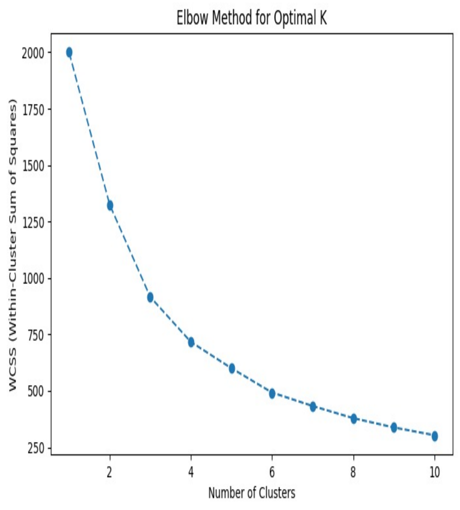
- **Libraries Used**: Matplotlib, Pandas,Seaborn
- **Insights**:
    - Detects significant differences that could indicate billing errors or miscalculations.
    - Helps in improving accuracy and reducing revenue loss.

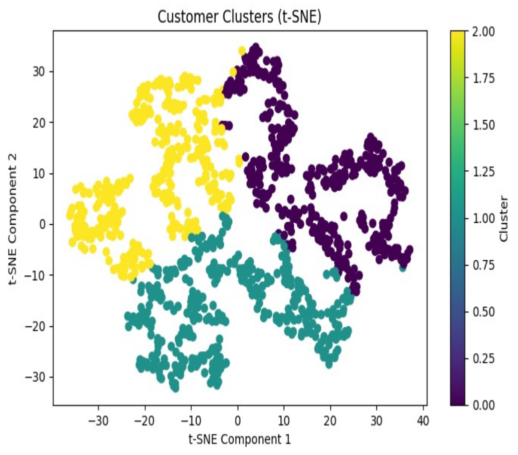


4.3 Consumption Clustering Analysis

Visualization: PCA Scatter Plot of Customer Clusters

- **Purpose**: Visualizes customer clusters based on consumption patterns.
- **Libraries Used**: Matplotlib, Seaborn, Scikit-learn
- **Insights**:
    - Groups customers with similar energy usage for targeted energy management strategies.
    - Identifies high-consumption users for demand-side management.

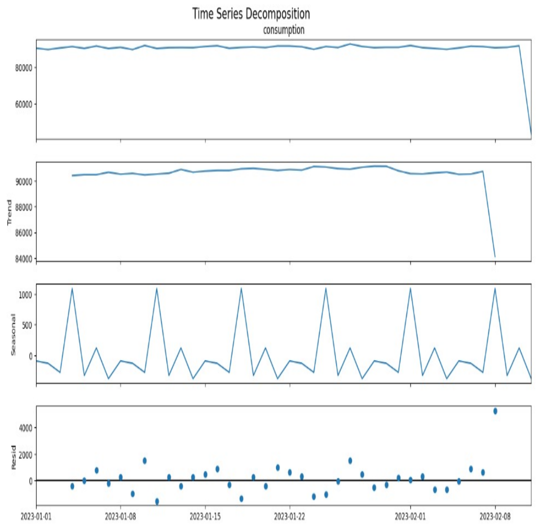


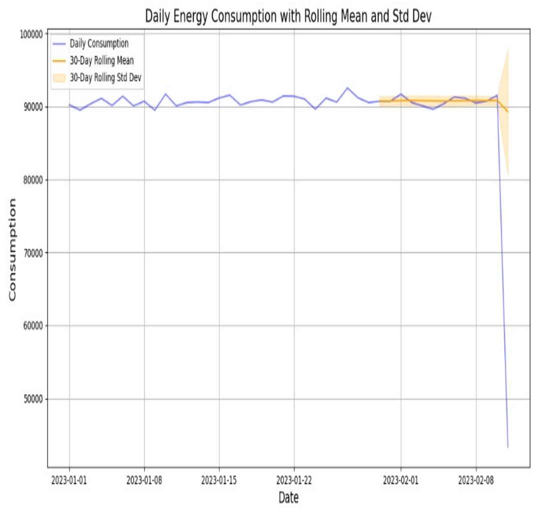


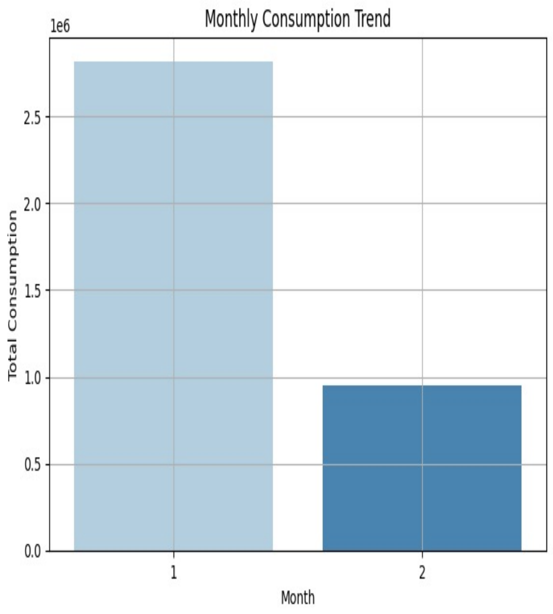
4.4 Consumption Trend Analysis

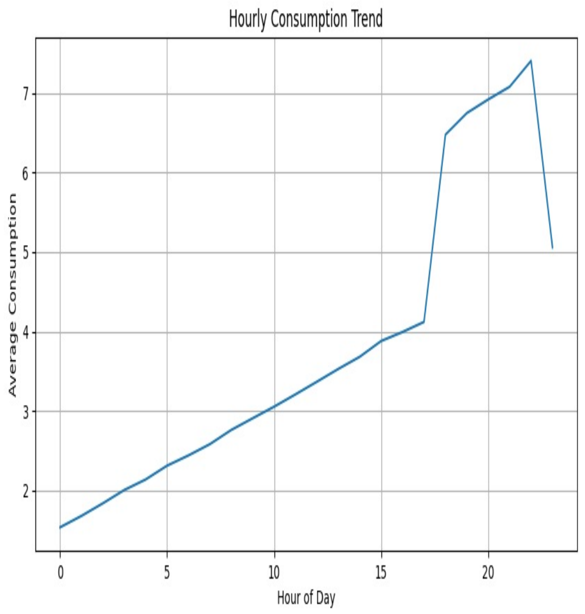
Visualization: Time Series Plot of Energy Consumption

- **Purpose**: Analyzes consumption trends over time.
- **Libraries Used**: Matplotlib, Pandas
- **Insights**:
    - Identifies seasonal patterns and long-term trends.
    - Helps in capacity planning and forecasting.





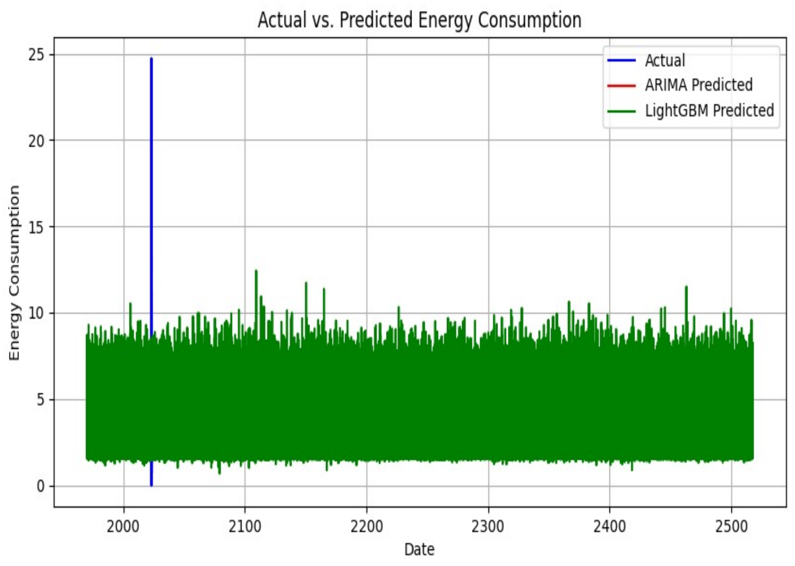




4.5 Load Forecasting

Visualization: Actual vs. Predicted Load

- **Purpose**: Compares actual energy usage with model predictions.
- **Libraries Used**: Matplotlib
- **Insights**:
  - Evaluates model performance in predicting energy demand.
  - Helps optimize grid operations and resource allocation.



4.6 GIS Analysis

Visualization: Heatmap of Energy Consumption

- **Purpose**: Displays geographic energy usage patterns.
- **Libraries Used**: Folium, GeoPandas
- **Insights**:
  - Identifies high-consumption areas for infrastructure planning.
  - Helps optimize resource allocation based on spatial demand.

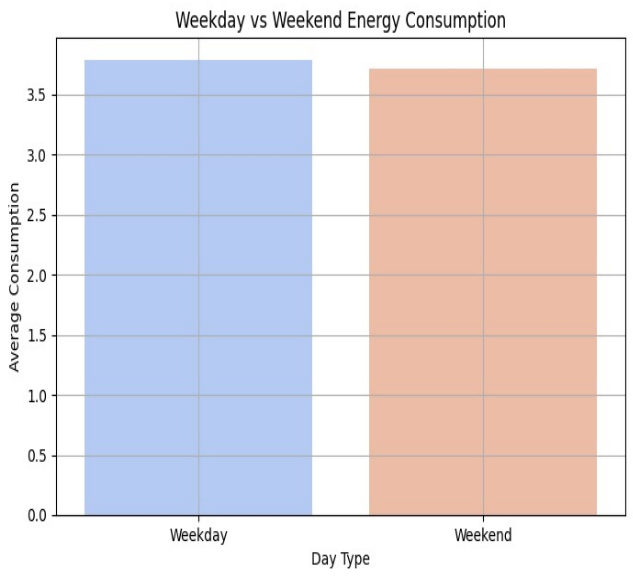
HTML Link:

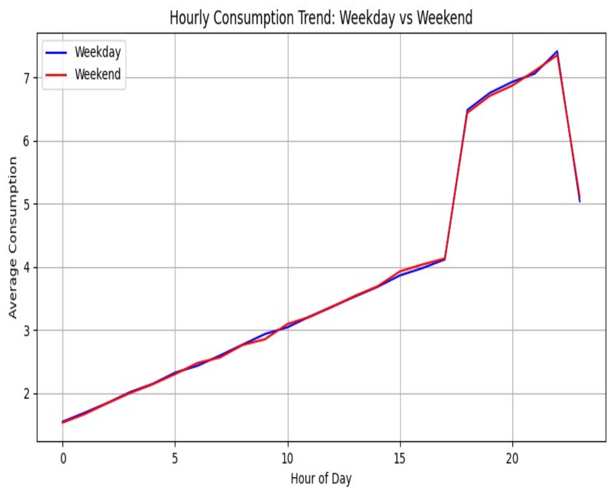
file:///Users/user/Desktop/Dr.%20Maurice/ami_meter_map.html

4.7 Weekday and Weekend Analysis

Visualization: Bar Plot of Mean Consumption

- **Purpose**: Compares average energy consumption between weekdays and weekends.
- **Libraries Used**: Matplotlib
- **Insights**:
  - Identifies peak usage periods.
  - Aids in demand-side management and grid stability planning.





5.1 Data Volume and Processing Efficiency

- **Challenge:** The dataset comprised approximately 9 million hourly readings, stored in MySQL Workbench. Direct analysis on this large dataset resulted in significant processing delays due to high computational demands.
- **Solution:** To enhance performance, a chunk-wise data processing approach was implemented, allowing data to be processed in smaller, manageable segments. Additionally, SQL queries were optimized to improve execution efficiency, reducing query response time and overall computational load.

5.2 Handling Missing and Anomalous Data

- **Challenge:** Missing values and outliers can distort statistical analysis, leading to biased conclusions and inaccurate predictions.
- **Solution:** Implemented statistical imputation techniques, such as mean and median substitution, to address missing values based on data distribution.

Applied Z-score analysis to detect and remove outliers, ensuring data integrity and reliability in the analysis.

5.3 Model Performance and Accuracy

- **Challenge:** Load forecasting accuracy fluctuated due to seasonal variations, leading to inconsistencies in predictions.
- **Solution:** Implemented LightGBM with optimized hyperparameters to enhance model performance. Additionally, incorporated lag features to capture historical consumption patterns, improving predictive accuracy and reducing errors associated with seasonal fluctuations.

5.4 GIS Data Integration

Challenge:
Integrating and visualizing large-scale GIS data with high granularity posed significant challenges, particularly in terms of geospatial data handling and performance optimization. The data volume and complexity made it difficult to process and render maps efficiently within the constraints of the system. Additionally, the maps could not be directly viewed in the terminal, requiring an HTML-based interface for visualization, which added another layer of complexity to the workflow.
Solution:
To address these challenges, I implemented a robust solution leveraging **optimized GeoPandas operations** and **advanced spatial clustering techniques**.

1. **Optimized GeoPandas Operations:**
    - Streamlined geospatial data processing by utilizing GeoPandas' efficient spatial joins, indexing, and geometric operations.
    - Implemented spatial indexing (R-tree) to accelerate query performance and reduce computational overhead when handling large datasets.
    - Applied data preprocessing techniques, such as simplifying geometries and filtering unnecessary attributes, to reduce memory usage and improve processing speed.
2. **Spatial Clustering Techniques:**
    - Employed spatial clustering algorithms (e.g., DBSCAN, K-means) to aggregate high-granularity data into manageable clusters, reducing the complexity of the dataset while preserving spatial patterns.
    - Used clustering to enable faster rendering and visualization by reducing the number of data points displayed on the map.
3. **HTML-Based Visualization:**
    - Integrated libraries such as **Folium** and **GeoPandas** to generate interactive maps that could be exported as HTML files.

    - Designed the workflow to dynamically generate HTML links for map visualization, ensuring seamless accessibility and user-friendly interaction outside the terminal environment.
    - Implemented responsive design principles to ensure the maps were accessible and functional across different devices and screen sizes.

5.5 Ensuring Reproducibility

Challenge: Ensuring consistency and reproducibility across different analysis runs can be challenging, especially when preprocessing steps involve complex transformations or when multiple team members are working on the same dataset. Inconsistent preprocessing can lead to discrepancies in results, making it difficult to compare findings or validate outcomes.
Solution: To address this, I implemented a structured approach to document and store preprocessing metadata. Specifically, I created a metadata.json file that systematically records all transformations applied to the dataset during preprocessing. This file includes details such as:

- **Data Cleaning Steps:** Documentation of missing value handling, outlier removal, and any data imputation techniques used.
- **Feature Engineering:** Description of new features created, transformations applied (e.g., scaling, normalization, encoding), and any feature selection processes.
- **Dataset Splitting:** Information on how the dataset was divided into training, validation, and test sets, including random seed values for reproducibility.
- **Versioning:** Timestamps and version numbers to track changes over time and ensure alignment with specific analysis runs.

6.0 Conclusion

6.1 Summary of Findings and Outcomes

This project successfully analyzed Advanced Metering Infrastructure (AMI) data within a simulated smart grid ecosystem, providing valuable insights into energy consumption patterns, anomalies, and forecasting. Key findings include:

1. **Anomaly Detection**: Statistical methods like Z-score analysis effectively identified outliers in energy consumption, while machine learning techniques such as autoencoders (planned for future implementation) hold promise for detecting more complex anomalies.
2. **Billing Analysis**: Discrepancies between actual and expected billing amounts were identified, enabling utilities to improve billing accuracy and revenue collection.

3. **Consumption Clustering**: K-Means clustering segmented customers into distinct groups based on consumption patterns, facilitating targeted demand-side management strategies.
4. **Consumption Trend Analysis**: Time series decomposition revealed long-term trends, seasonal variations, and short-term fluctuations in energy usage, aiding in operational planning.
5. **Load Forecasting**: LightGBM and ARIMA models demonstrated strong performance in predicting future energy consumption, with LightGBM outperforming ARIMA in terms of accuracy.
6. **Geospatial Analysis**: GIS tools identified urban hotspots with higher energy consumption, providing actionable insights for infrastructure planning and resource allocation.
7. **Weekday/Weekend Analysis**: Significant differences in energy consumption patterns between weekdays and weekends were observed, highlighting the need for tailored energy management strategies.

The project achieved its objectives by leveraging advanced data processing, machine learning, and visualization techniques, resulting in actionable insights for optimizing grid operations and improving customer engagement.

6.2 Reflection on Methods and Suggestions for Future Work

The methods employed in this project were effective in addressing the challenges of load forecasting, anomaly detection, and consumption pattern analysis. However, there are areas for improvement and opportunities for future work:

1. **Enhanced Anomaly Detection**: While Z-score analysis provided a solid foundation, future work could incorporate advanced machine learning models like autoencoders or isolation forests to detect more subtle and complex anomalies.
2. **Feature Engineering**: Additional features, such as weather data, holidays, and economic indicators, could be incorporated to improve the accuracy of load forecasting models.
3. **Model Optimization**: Further hyperparameter tuning and ensemble methods could enhance the performance of LightGBM and other machine learning models.
4. **Real-Time Analysis**: Implementing real-time data processing and analysis using tools like Apache Spark or Kafka could enable utilities to respond more dynamically to changing consumption patterns.
5. **Customer Segmentation Refinement**: Exploring alternative clustering algorithms, such as Gaussian Mixture Models (GMM) or hierarchical clustering, could provide deeper insights into customer behavior.

6. **Geospatial Analysis Expansion**: Integrating real-world geospatial data and advanced GIS tools could improve the accuracy and granularity of spatial consumption patterns.
7. **Scalability**: For larger datasets, distributed computing frameworks like Hadoop or cloud-based solutions could be explored to handle data processing and analysis more efficiently.
8.

6.3 Appendix

The appendix includes the following supplementary materials:

1. **Code Repository**: A link to the GitHub repository containing all scripts, notebooks, and documentation used in the project.
2. **Dataset Description**: Detailed metadata and schema of the simulated AMI dataset.
3. **Visualizations**: High-resolution versions of all plots, charts, and maps generated during the analysis.
4. **Model Performance Metrics**: Tabulated results of evaluation metrics (e.g., MAE, RMSE) for all machine learning models.
5. **References**: A list of academic papers, articles, and tools referenced during the project.
6. **Glossary**: Definitions of technical terms and acronyms used in the documentation.

7.0 References

The following references were consulted during the execution of this project, including academic papers, technical documentation, and tools used for data analysis, machine learning, and visualization. These resources provided the theoretical foundation and practical guidance necessary to achieve the project's objectives.

7.1 Academic Papers and Articles

1. **Chen, T., & Guestrin, C. (2016).**
   XGBoost: A Scalable Tree Boosting System.
   Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
   DOI: 10.1145/2939672.2939785
2.
3. **Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017).**

LightGBM: A Highly Efficient Gradient Boosting Decision Tree.
Advances in Neural Information Processing Systems (NeurIPS).
DOI: 10.5555/3294996.3295074

4.

5. **Hyndman, R. J., & Athanasopoulos, G. (2021).**
Forecasting: Principles and Practice (3rd Edition).
OTexts.
https://otexts.com/fpp3/

6.

7. **Hastie, T., Tibshirani, R., & Friedman, J. (2009).**
The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd Edition).
Springer.
DOI: 10.1007/978-0-387-84858-7

8.

9. **Goodfellow, I., Bengio, Y., & Courville, A. (2016).**
Deep Learning.
MIT Press.
https://www.deeplearningbook.org/