Table of Content

Abstract 2

Problem Statement 2

Goals and Non-Goals 2

Data Sources 3

Data Integration and Explorations 4

Data Integration 4

Exploratory Data Analysis 4

Summary Statistics and Outlier Detection 5

Bias Exploration 8

Security Checklist, Privacy, and Other Risk 9

Data Preparation and Preprocessing 10

Balancing the Dataset 10

Train and Testing Model 11

Future Enhancements 11

Data source incorporation, batch processing, and behavioral layers 12

References 13

Appendix 14

Python code via AWS Sagemaker

ECOGridAI – Pipelining Energy Usage
Daily Monitoring of the Grid

Authors: Duy Nguyen, Jordan Torres, Kirsten Drennen
Company Name: EcoGrid AI
Company Industry: Energy Optimization & Efficiency
Company Size: Mid-sized clean-tech company focused on AI-driven household energy solutions.

Abstract

Rising energy costs and workplace inefficiencies are driving the need for smarter energy solutions. EcoGrid AI is an AI-powered platform that analyzes real-time energy grid data, smart meters, and patterns to automate energy-saving decisions. By detecting waste and burnout risk, EcoGrid AI recommends smart shutdowns and efficiency boosts, aiming to reduce costs, lower carbon footprints, and enhance wellbeing.

Problem Statement

Both businesses and individual consumers struggle with energy inefficiency. According to the Electric Power Research Institute (EPRI) poor power quality is estimated to cost losses ranging from $119 billion to $188 billion (Ignatova, 2015). Unnecessary power consumption, poor scheduling, and lack of real-time information leads to higher costs and an increased carbon footprint. Consumers often leave appliances running longer than needed, while businesses fail to optimize resources and equipment usage.

EcoGrid AI analyzes smart meter data, grid trends, and workplace activity to automate energy-saving decisions and optimize work schedules. By detecting wasteful energy consumption and patterns linked to inefficiency, our system provides smart recommendations that help both customers and businesses reduce energy consumption while improving sustainability.

Goals

- Predict and monitor real-time energy grid conditions that will help to inform poor power quality
- Monitor the grid in real time to analyze consumption and inform strategy to help reduce stress on the grid during peak hours
- Inform strategies for proactive power shut-offs to mitigate wildfires

Non-Goals

- Not the end all be all for the company/consumer energy choices – We are providing suggestions
- Data-driven not machine – We are not developing smarter appliances but giving insight into your current setup

- Not in control of your power grids – We are not providing insights, not changing the grids infrastructure or cost of wattages to customers

Data Sources

Data will be sourced from the U.S. Energy Information Administration, extracted by an application programming interface (API), and stored on AWS service S3 Bucket to be processed in AWS Sagemaker. Daily usage of megawatt hours were ingested for the period of 1/1/2024 to 12/31/2024 for the following

- Electricity Overview (demand, forecast demand, generation, and total interchange)
- Electricity demand by region
- Electricity generation by energy source
- Electricity interchange with neighboring countries

Data Exploration, Security, Privacy, and Other Risks

GitHub Repository Information

The GitHub repository provides details instructions to install, extract, and store the initial packages and data required for this model. Additionally, a detail Jupyter notebook has been provided to run the model detailed in this design document.

Resources:
GitHub Repository - https://github.com/dnguyen-1/ADS508TeamProject
Data Preparation & Model Notebook - https://github.com/dnguyen-1/ADS508TeamProject/blob/main/gridAI_training.ipynb

Exploratory Data Analysis

A one-time application programming interface (API) was created between the EIA website and AWS SageMaker to communicate and retrieve data. Once retrieved, data was saved as a .JSON file and uploaded to an S3 Bucket, Amazon S3 is an object storage service that allows for scalability, security, and performance; highly important attributes since our datasets are over a million records (Amazon Web Services, n.d.). Users will also need to register and create an API key to retrieve data from the EIA website. A Jupyter notebook has been created to retrieve data and stored in the GitHub Repository.

Resources:
EIA API Key Registration - https://www.eia.gov/opendata/register.php (Copyrights and Reuse Policy)
Data API and Storage - https://github.com/dnguyen-1/ADS508TeamProject/blob/main/data/dataimport.ipynb

Data Integration

To support scalable querying and analysis, the team used Amazon Athena alongside Amazon S3. Raw data was extracted from the U.S. Energy Information Administration (EIA) via a public API. This data, which included subregion usage, energy source generation, and neighboring grid interchange, was initially stored as .json and .csv files. These files were uploaded to a structured S3 bucket, organized by dataset type and date.

To ensure compatibility with Athena, all files were consistently formatted using UTF-8 encoding, standard delimiters, and a header row to support schema inference. External tables were then defined in Athena using the OpenCSVSerde library, with each table mapped directly to its corresponding S3 path. This setup enabled SQL queries to be executed across millions of records without loading data into memory.

Athena played a central role in the exploratory analysis phase. It was used to inspect distributions, identify outliers, verify missing values, and perform aggregations across time intervals. The use of Athena's serverless architecture also minimized infrastructure overhead while allowing rapid iteration during preprocessing and modeling.

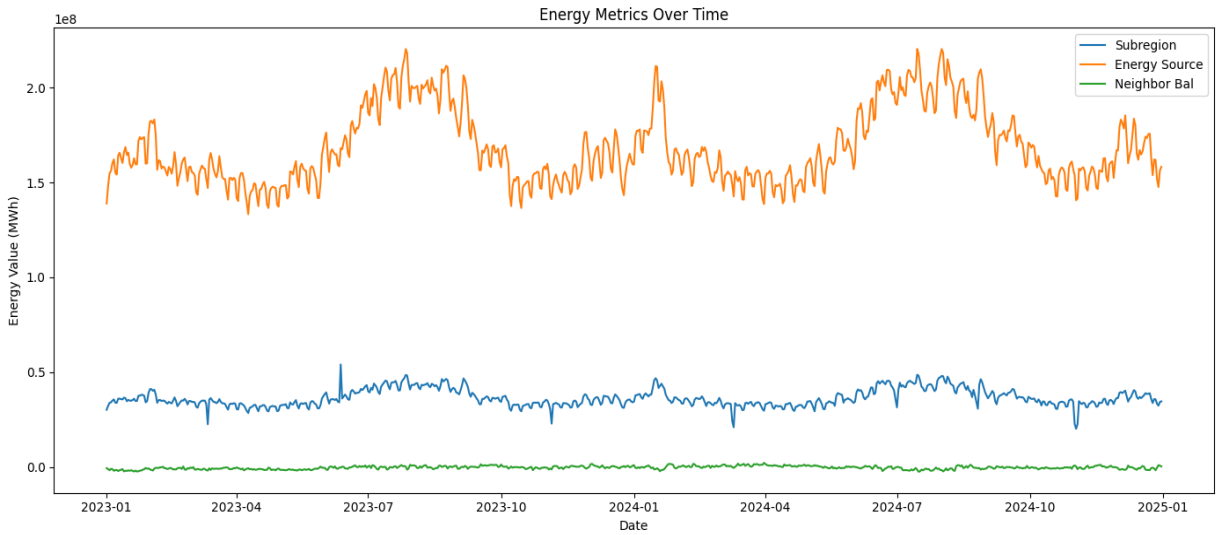
Summary Statistics and Outlier Detection

Exploratory data analysis (EDA) was conducted on the four datasets obtained from the EIA: Subregion, Energy Source, and Neighboring Balancing. Each dataset was inspected for structure, missing values, data types, and statistical properties.

All datasets were confirmed to be complete with no missing values. The data types were also as expected, with "period" as timestamps and "value" fields as integers. The Subregion dataset contains 302,719 rows, the Energy Source dataset 1.42 million rows, and the Neighbor Balancing dataset 1.23 million rows. All datasets span the years 2023 to 2024.

The Subregion dataset includes 302,719 records of hourly load data across various time zones with values ranging from 0 to 785,360 MWh and an average of approximately 87,875 MWh. The Energy Source dataset contains over 1.42 million entries of generation values extending from –24,448 MWH to 7,156,358 MWh, which captures both the positive generations as well as the potential losses and negative net values. The Neighboring Balancing Authorities dataset includes 1.23 million records displaying the energy interchange between regions. The values here are centered around zero (mean = -181 MWh), indicating symmetrical net imports/exports over time.

Figure 1. Energy Metrics Over Time



To better understand energy behavior over time, the total value for each dataset was plotted by date from early 2023 through the end of 2024 (Figure 1). The Energy source dataset (orange) shows clear seasonal generation patterns, with periodic peaks during the summer and winter months, which is likely due to the demand for cooling and heating. The Subregion values (blue) are relatively stable but exhibit weekly variability and an occasional spike in consumption, reflecting the fluctuation in load across regional grids.

Figure 2a. Subregion Value Distribution

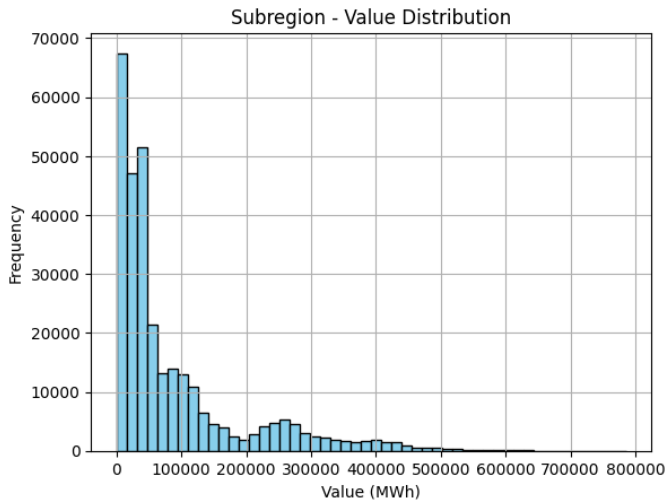


Figure 2b. Energy Source Value Distribution

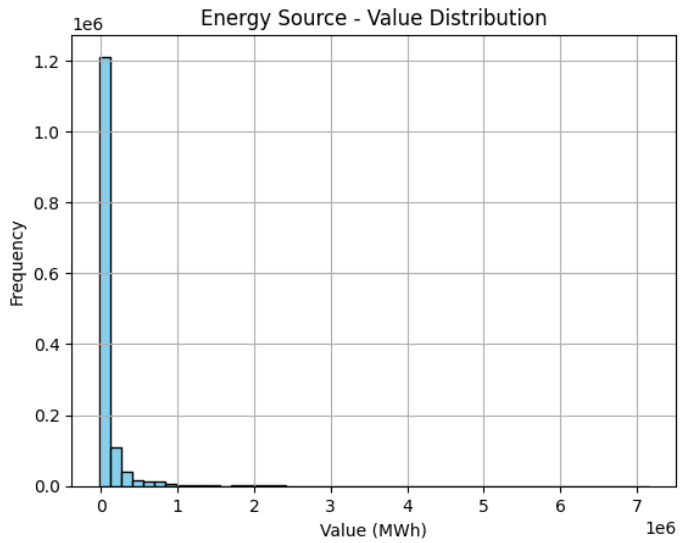
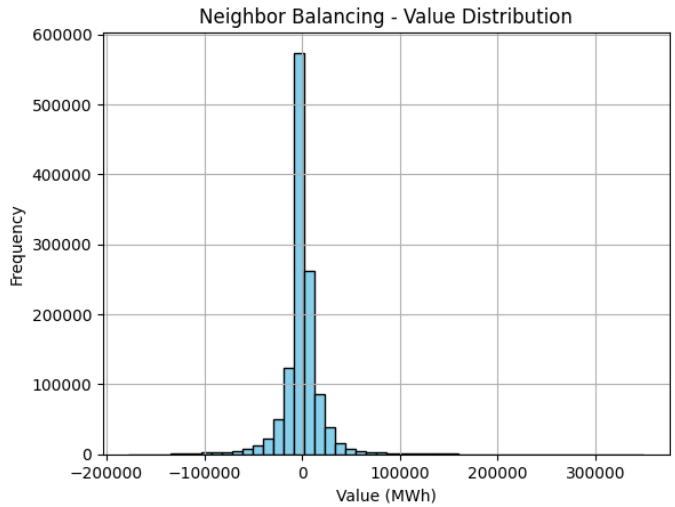
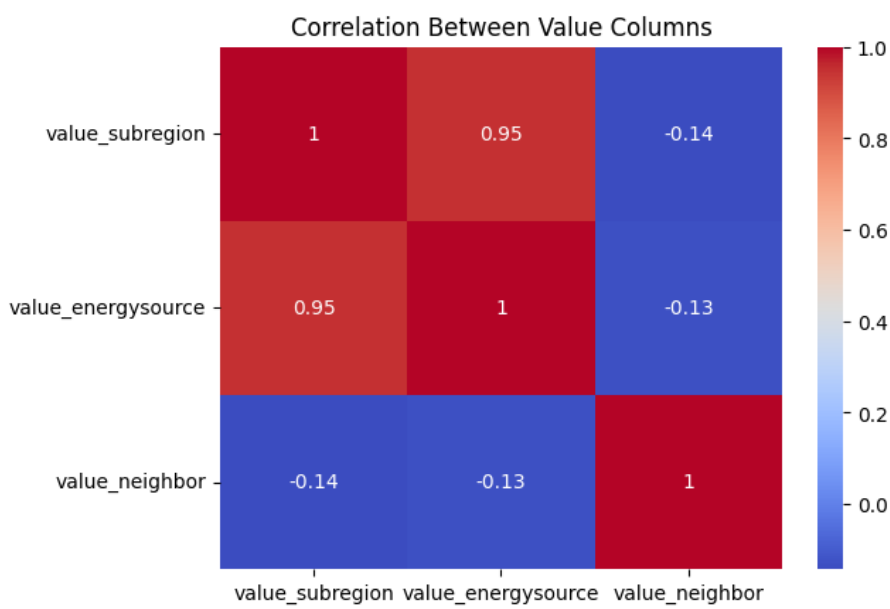


Figure 2c. Neighbor Balancing Value Distribution



The distribution of energy values for the three datasets was also visualized using histograms (Figure 2). The subregion values (Figure 2a) followed a heavily right-skewed distribution, with most observations falling below 100,000 MWh, indicating many low-energy-use periods with fewer high-demand instances. Energy source values (Figure 2b) were more dispersed and multimodal, consistent with the presence of multiple generation types also following a right-skewed distribution with a high concentration below 500,000 MWh with a few extreme values reaching beyond 7 million MWh. The neighbor balancing distribution (Figure 2c) is observed to be symmetric and centered around 0, which reflects the energy exchange between neighboring balancing authorities.

Figure 3. Correlation between Value Columns



Lastly, an aggregation of each dataset was performed by day and computed using Pearson Correlation Coefficients among Subregion, Energy Source, and Neighboring Balancing Values

(Figure 3). The resulting heatmap showed a strong positive correlation (0.95) between subregion consumption and total energy generation, indicating the close link between regional consumption and overall generation patterns. Neighbor balancing exhibited a weaker correlation to both metrics, showcasing its role in smoothing out the supply and demand mismatches across regions. These findings suggest that while all three datasets contribute valuable context, subregion and energy source data are likely to be the most predictive inputs in forecasting regional electricity demand.

Bias Exploration

Although the datasets used in this project are comprehensive and clean, several sources of bias may influence the integrity of the model's predictions. One concern is the geographic bias due to the over representation of specific balancing authorities and fuel types, which may cause the model to overweigh high-frequency regions. The datasets also display a distributional bias in the subregion and energy source datasets through highly skewed value distributions. This skewness can cause the model to favor large-magnitude events, resulting in reduced sensitivity to low usage or baseline patterns. Together, these biases have the ability to inhibit fair, reliable, and generalizable model performance.

Measuring Impact

To evaluate the impact of this project, we identified two key metrics that align with our original goals. First, we anticipate a measurable reduction in peak grid usage as a result of our real-time forecasting models and smart recommendations. By helping users shift their consumption outside high-demand windows, EcoGridAI can support grid stability and reduce stress on infrastructure. Second, we expect increased user participation in energy-saving programs. If the AI-driven insights are clear, timely, and useful, more customers are likely to engage with the platform and adopt cost-saving behaviors.

Security Checklist, Privacy, and Other Risks

Our system does not process any Protected Health Information (PHI), and it does not store or process credit card data at any point. While the project currently operates without sensitive personal data, it may incorporate limited Personally Identifiable Information (PII) in the future, such as user IDs or region-based usage patterns, if incentive-based programs are implemented. This PII would be minimal and non-sensitive, and it would be used solely for energy consumption tracking and program customization.

To support data ingestion and analysis, we use a secure token-based connection with the U.S. Energy Information Administration (EIA) API. Data pulled through this API is stored in an Amazon S3 bucket for exploration, transformation, and modeling. No third parties are involved, and the data remains within a controlled AWS environment.

Regional and structural bias must also be considered when working with energy data. Energy source bias may occur, as fossil fuels are often overrepresented in public grid data compared to renewables. Regional bias is also a concern since areas with better infrastructure

tend to contribute more data, which can skew the dataset toward those regions. In addition, confirmation bias may arise when established time-based efficiency tips, such as avoiding appliance use from 4 to 9 p.m., are overemphasized in the modeling phase. This could reinforce common assumptions and ignore underexplored timeframes.

Finally, we acknowledge the ethical implications of tracking energy usage. Transparency and user consent are core to our project's design. Users should be fully informed about how their data is collected, stored, and used. By clearly disclosing data handling practices and complying with data privacy regulations, we aim to build trust while developing a system that can scale responsibly.

Energy Modeling Using SageMaker's Built-in DeepAR

Data Preparation

The data preparation process began with data scrubbing, where we focused on identifying and handling missing values in the datasets obtained from the U.S. Energy Information Administration (EIA). In rare cases where missing values were found, we opted to drop incomplete rows to maintain data integrity rather than introduce assumptions through imputation. Fortunately, after running .isnull().sum() checks, all datasets were confirmed to be complete, allowing us to proceed confidently into feature engineering.

Feature selection centered on isolating the key numeric fields from each dataset. Specifically, we retained the value_subregion, value_energysource, and value_neighbor columns, which represented electricity demand, generation by source, and interchange activity between neighboring grids. Non-essential fields such as identifiers (respondent, respondent-name) were excluded, as they did not contribute predictive value and could introduce unnecessary complexity.

We then created new features to improve the model's ability to recognize temporal patterns. Using the period column, we derived calendar-based fields like month, day of the week, and weekend vs. weekday flags to capture cyclical demand behavior. We merged the three main datasets on the period column to form a unified data frame. This consolidated format enabled consistent alignment across datasets and made it easier to build a time series suitable for forecasting.

For feature transformation, the most important step was converting the period column to a datetime object using pd.to_datetime(). This enabled us to resample the data, plot trends over time, and aggregate values by different temporal resolutions. Since our key features were already numeric, no additional encoding or normalization was required. However, we noted that future model experimentation may benefit from additional transformations depending on algorithm needs.

Regarding data balancing, we chose not to apply balancing techniques during this phase. Since our prediction target involves continuous numerical values, rather than classification labels, there was no need to balance for class representation. That said, we remain aware that some energy sources and regions are overrepresented, and we may address this in future versions using techniques like weighting or downsampling if model bias is observed.

Finally, for training and evaluation, we applied an 80/20 train-test split using train_test_split() from sklearn.model_selection, with a fixed random seed for reproducibility. For future time series models where temporal continuity is essential, we may adjust this to a chronological split to prevent data leakage and maintain sequence integrity.

Model Training

The team will be using SageMaker's built-in DeepAR algorithm to train a demand forecasting model for energy usage. DeepAR leverages machine learning to train, tune, and optimize forecasts, and is capable of learning from multiple time series simultaneously. It also supports "cold-start" forecasting, meaning it can generate predictions even for series with little or no historical data (Kafritsas, 2022). Given that our dataset has simple attributes, using a built-in algorithm like DeepAR is an efficient and cost-effective approach to model deployment. The model will be trained on daily energy usage values, measured in megawatt-hours, capturing grid demand and usage by fuel type over a two-year period. Model performance will be evaluated by visually overlaying forecasted values with actual data, as well as by calculating accuracy and root mean square error (RMSE).

We selected DeepAR due to its optimization in time series forecasting and ability to handle multiple datasets related to time series datasets (Gomede, 2024). This fits our business goals of predicting grid energy usage across different sources, balancing authorities, and fuel types. DeepAR also has the ability to handle seasonality, trends, and potential noise in energy consumption patterns. This allows for more accurate predictions with complex datasets.

For parameter tuning, we will begin with DeepAR's default hyperparameters and make adjustments according to model performance. These key parameters include context length (using 30 to aid the model in learning from previous month's data) and prediction length (set to 7 to predict a weekly forecast).These parameters were chosen to balance accuracy and training time. We will be using a single ml.m5.xlarge instance for training. This instance type is sufficient given the scale of the dataset and the efficiency of the built-in DeepAR container (AWS, 2023). While the GPU instances can reduce the training time, cost- efficiency was prioritized.

Model Evaluation

The model performance will be analyzed using a combination of RMSE and visual inspections. RMSE is useful for understanding the magnitude of errors (Brownlee, 2021). Visualization aids in confirming the model's capture of key trends and anomalies to ensure that the model is aligning with the business needs, such as demand planning as well as load balancing.

Future Enhancements

As we continue to develop our platform, there are several routes to enhance our model pipeline and overall functionality. We first plan to incorporate additional data sources such as real-time weather information to improve forecasting accuracy. Weather patterns, especially temperature and humidity, as these variables significantly influence energy demand (NOAA, 2023). By integrating this data, our model will be able to make context-aware predictions. In addition to weather, adding demographic data, such as population density and usage by sector,

will aid in the fine-tuning of forecasts for specific areas. This will improve our ability to recommend efficient energy-saving strategies (U.S. Census Bureau, 2023).

We also aim to transition from batch processing, which was used in this project for the sake of time and cost efficiency, to a real-time streamlining process to deliver actionable insights while minimizing the delays. By using technologies such as Apache Kafka or Amazon Kinesis, we would be able to analyze incoming data streams instantly while triggering real-time alerts or recommendations (AWS, 2023) (Apache,2023). This implementation would allow both businesses and consumers to respond immediately to energy usage and demand warnings to increase efficiency and responsiveness. In collaboration with these readings, implementing notification systems such as SMS will ensure that users receive timely updates based on their energy usage patterns.

Lastly, we plan to develop a behavioral layer that learns from each individual user's historical energy patterns. This approach is influenced by already established methods in recommender systems and behavior modeling, which have proven effective in tailoring user experiences across industries (Ricci et al.,2015). By leveraging this data, our AI would be able to give personable suggestions tailored to a user's lifestyle (Gkatzikis et al., 2013). This personalization will support better decision-making and improve the program by aligning recommendations with user habits and preferences.

References

Amazon Web Services. (2023). Time series forecasting with DeepAR in Amazon SageMaker. AWS Machine Learning Blog. https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html.

Amazon Web Services. (n.d.). What is Amazon S3? AWS Documentation. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Apache. (2023). Apache Kafka documentation. Apache Software Foundation. https://kafka.apache.org/documentation/.

AWS. (2023). Amazon Kinesis Data Streams. Amazon Web Services. https://aws.amazon.com/kinesis/data-streams/.

Fregly, C., & Barth, A. (2021). Data science on AWS. O'Reilly.

Gkatzikis, L., Koutsopoulos, I., & Salonidis, T. (2013). The role of aggregators in smart grid demand response markets. IEEE Journal on Selected Areas in Communications, 31(7), 1247–1257. https://doi.org/10.1109/JSAC.2013.130709.

Ignatova, V. (2015 October 16). "Why Poor Power Quality Costs Billions Annually and What Can Be Done About It"

Kafritsas, N. (2022, November 14). DeepAR: Mastering time-series forecasting with deep learning. Medium. https://medium.com/p/bc717771ce85.

Machine Learning Mastery. (n.d.). Regression metrics for machine learning. https://machinelearningmastery.com/regression-metrics-for-machine-learning/.

NOAA. (2023). Weather and climate data. National Oceanic and Atmospheric Administration. https://www.noaa.gov.

Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender systems handbook. Springer. https://doi.org/10.1007/978-1-4899-7637-6.

U.S. Census Bureau. (2023). Data tools and apps. https://www.census.gov/data.html.

Unknown. (n.d.). Revolutionizing forecasting: Harnessing the power of DeepAR for enhanced predictive accuracy. Medium. https://medium.com/the-modern-scientist/revolutionizing-forecasting-harnessing-the-power-of-deepar-for-enhanced-predictive-accuracy-and-a3e1d1102703.

In [1]:
```
#import required packages
import boto3
import requests
import json
import pandas as pd
import seaborn as sns
from datetime import datetime
import time
from pyathena import connect
import sagemaker
from sqlalchemy.engine import create_engine
import warnings
from sagemaker import get_execution_role
from sagemaker.estimator import Estimator
from sagemaker import image_uris
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

```
/opt/conda/lib/python3.11/site-packages/pydantic/_internal/_fields.py:192: UserWarni
ng: Field name "json" in "MonitoringDatasetFormat" shadows an attribute in parent "B
ase"
  warnings.warn(
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/
config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/sagemaker-use
r/.config/sagemaker/config.yaml
```

Check s3 for files

In [2]:
```
# Create an S3 client
s3 = boto3.client('s3')

# List objects with the prefix
response = s3.list_objects_v2(Bucket='ecogridaidata', Prefix='eia_electricity/eia_d

# Print the file keys
for obj in response.get('Contents', []):
    print(obj['Key'])
```

```
eia_electricity/eia_data_demand_20250324_225715.json
eia_electricity/eia_data_demand_20250325_021240.json
eia_electricity/eia_data_energy_source_20250323_200432.json
eia_electricity/eia_data_neighboring_bal_20250324_224805.json
eia_electricity/eia_data_subregion_20250323_191441.json
eia_electricity/eia_data_subregion_20250325_021741.json
```

Note: If data is not available, locate "RTC:ADS508TeamProject/installation/Installation & Set-Up.ipynb" notebook to run refesh data from the EIA website

Load data stored as a json file in s3 "ecogridaidata" into a dataframe for verification and processing

In [3]:
```
# S3 path to your file
s3_uri = "s3://ecogridaidata/eia_electricity/eia_data_subregion_20250323_191441.jso
s3_uri2 = "s3://ecogridaidata/eia_electricity/eia_data_energy_source_20250323_20043
s3_uri3 = "s3://ecogridaidata/eia_electricity/eia_data_neighboring_bal_20250324_224
s3_uri4 = "s3://ecogridaidata/eia_electricity/eia_data_demand_20250325_021240.json"

# Read json directly into pandas DataFrame
df_subregion = pd.read_json(s3_uri, storage_options={"anon": False})
df_energysource = pd.read_json(s3_uri2, storage_options={"anon": False})
df_neighbor_bal = pd.read_json(s3_uri3, storage_options={"anon": False})
df_demand = pd.read_json(s3_uri4, storage_options={"anon": False})

# Preview the data
df_subregion.head()
```

Out[3]:

	period	subba	subba-name	parent	parent-name	timezone	value	value-units
0	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Arizona	247213	megawatthours
1	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Central	247876	megawatthours
2	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Eastern	248481	megawatthours
3	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Mountain	247213	megawatthours
4	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Pacific	246697	megawatthours

In [4]:
```
df_energysource.head()
```

Out[4]:

	period	respondent	respondent-name	fueltype	type-name	timezone	timezone-description	value	val
0	2024-12-31	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Arizona	Arizona	23782	megaw
1	2024-12-31	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Central	Central	23309	megaw
2	2024-12-31	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Eastern	Eastern	22893	megaw
3	2024-12-31	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Mountain	Mountain	23782	megaw
4	2024-12-31	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Pacific	Pacific	24422	megaw

In [5]:
```
df_neighbor_bal.head()
```

Out[5]:

	period	fromba	fromba-name	toba	toba-name	timezone	value	value-units
0	2024-12-31	AECI	Associated Electric Cooperative, Inc.	MISO	Midcontinent Independent System Operator, Inc.	Arizona	584	megawatthours
1	2024-12-31	AECI	Associated Electric Cooperative, Inc.	MISO	Midcontinent Independent System Operator, Inc.	Central	359	megawatthours
2	2024-12-31	AECI	Associated Electric Cooperative, Inc.	MISO	Midcontinent Independent System Operator, Inc.	Eastern	321	megawatthours
3	2024-12-31	AECI	Associated Electric Cooperative, Inc.	MISO	Midcontinent Independent System Operator, Inc.	Mountain	584	megawatthours
4	2024-12-31	AECI	Associated Electric Cooperative, Inc.	MISO	Midcontinent Independent System Operator, Inc.	Pacific	1111	megawatthours

In [6]:
```
df_demand.head()
```

Out[6]:

	period	respondent	respondent-name	type	type-name	timezone	timezone-description
0	2025-03-24	AECI	Associated Electric Cooperative, Inc.	DF	Day-ahead demand forecast	Central	Central
1	2025-03-24	AECI	Associated Electric Cooperative, Inc.	DF	Day-ahead demand forecast	Eastern	Eastern
2	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Arizona	Arizona
3	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Central	Central
4	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Eastern	Eastern

Create Athena Database

In [7]:
```
# Save to CSV for Athena readability
df_subregion.to_csv("s3://ecogridaidata/eia_electricity/subregion.csv", index=False
df_energysource.to_csv("s3://ecogridaidata/eia_electricity/data_csv/energysource.cs
df_neighbor_bal.to_csv("s3://ecogridaidata/eia_electricity/data_csv/neighboring_bal
df_demand.to_csv("s3://ecogridaidata/eia_electricity/demand.csv", index=False)
```

In [8]:
```
# Setup
sess = sagemaker.Session()
bucket = 'ecogridaidata'
role = sagemaker.get_execution_role()
region = boto3.Session().region_name

s3_staging_dir = f"s3://{bucket}/athena/staging"
database_name = "ecodataaidatabase"

# Connect
conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)
cursor = conn.cursor()

# Create DB if needed
cursor.execute("CREATE DATABASE IF NOT EXISTS ecodataaidatabase")
```

Out[8]:
```
<pyathena.cursor.Cursor at 0x7f51c7ba80d0>
```

In [9]:
```
# Subregion Table
query = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {database_name}.subregion (
    period string,
    subba string,
    `subba-name` string,
    parent string,
    `parent-name` string,
    timezone string,
    value int,
    `value-units` string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar" = "\\""
)
LOCATION 's3://{bucket}/athena/subregion/'
TBLPROPERTIES ('skip.header.line.count'='1');
"""

cursor = conn.cursor()
cursor.execute(query)
```

Out[9]:
```
<pyathena.cursor.Cursor at 0x7f51c5fc5250>
```

In [10]:
```
# Test Query Subregion
query = f"SELECT * FROM {database_name}.subregion LIMIT 10;"
df = pd.read_sql(query, conn)
df.head()
```

Out[10]:

	period	subba	subba-name	parent	parent-name	timezone	value	value-units
0	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Arizona	247213	megawatthours
1	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Central	247876	megawatthours
2	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Eastern	248481	megawatthours
3	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Mountain	247213	megawatthours
4	2024-12-31	PGAE	Pacific Gas and Electric	CISO	California Independent System Operator	Pacific	246697	megawatthours

In [11]:

```
# Energy Source Table
query = """
CREATE EXTERNAL TABLE IF NOT EXISTS ecodataaidatabase.energysource (
    period string,
    respondent string,
    `respondent-name` string,
    fueltype string,
    `type-name` string,
    timezone string,
    `timezone-description` string,
    value int,
    `value-units` string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar" = "\\""
)
LOCATION 's3://ecogridaidata/athena/energysource/'
TBLPROPERTIES ('skip.header.line.count'='1');
"""
cursor = conn.cursor()
cursor.execute(query)
```

Out[11]:

```
<pyathena.cursor.Cursor at 0x7f51c5d78290>
```

In [12]:
```
# Neighboring Table
query = """
CREATE EXTERNAL TABLE IF NOT EXISTS ecodataaidatabase.neighboring_bal (
    period string,
    fromba string,
    `fromba-name` string,
    toba string,
    `toba-name` string,
    timezone string,
    value int,
    `value-units` string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar" = "\\""
)
LOCATION 's3://ecogridaidata/athena/neighboring_bal/'
TBLPROPERTIES ('skip.header.line.count'='1');
"""
cursor = conn.cursor()
cursor.execute(query)
```

Out[12]:
```
<pyathena.cursor.Cursor at 0x7f51c5c6e990>
```

In [13]:
```
# Demand Table
query = """
CREATE EXTERNAL TABLE IF NOT EXISTS ecodataaidatabase.demand (
    period string,
    respondent string,
    `respondent-name` string,
    type string,
    `type-name` string,
    timezone string,
    `timezone-description` string,
    value int,
    `value-units` string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar" = "\\""
)
LOCATION 's3://ecogridaidata/athena/demand/'
TBLPROPERTIES ('skip.header.line.count'='1');
"""
# Run the query
cursor = conn.cursor()
cursor.execute(query)
```

Out[13]:
```
<pyathena.cursor.Cursor at 0x7f51c5d79650>
```

In [14]:
```
# Verification
df_tables = pd.read_sql(f"SHOW TABLES IN {database_name}", conn)
```

```
print(df_tables)
```

```
         tab_name
0          demand
1    energysource
2 neighboring_bal
3       subregion
```

In [15]:
```
if database_name in df_tables.values:
    ingest_create_athena_db_passed = True
```

Exploratory Daya Analysis

In [16]:
```
for name, df in zip(
    ['Subregion', 'Energy Source', 'Neighbor Balancing', 'Demand'],
    [df_subregion, df_energysource, df_neighbor_bal, df_demand]
):
    print(f"\n=== {name} Dataset ===")
    print("Shape:", df.shape)
    print("Null values:\n", df.isnull().sum())
    print("Data types:\n", df.dtypes)
```

```
=== Subregion Dataset ===
Shape: (302719, 8)
Null values:
 period         0
subba          0
subba-name     0
parent         0
parent-name    0
timezone       0
value          0
value-units    0
dtype: int64
Data types:
 period         object
subba          object
subba-name     object
parent         object
parent-name    object
timezone       object
value           int64
value-units    object
dtype: object

=== Energy Source Dataset ===
Shape: (1425802, 9)
Null values:
 period                  0
respondent              0
respondent-name         0
fueltype                0
type-name               0
timezone                0
timezone-description    0
value                   0
value-units             0
dtype: int64
Data types:
 period                  object
respondent              object
respondent-name         object
fueltype                object
type-name               object
timezone                object
timezone-description    object
value                    int64
value-units             object
dtype: object

=== Neighbor Balancing Dataset ===
Shape: (1233232, 8)
Null values:
 period         0
fromba         0
fromba-name    0
toba           0
toba-name      0
```

```
timezone       0
value          0
value-units    0
dtype: int64
Data types:
 period         object
fromba         object
fromba-name    object
toba           object
toba-name      object
timezone       object
value           int64
value-units    object
dtype: object

=== Demand Dataset ===
Shape: (1038828, 7)
Null values:
 period                  0
respondent              0
respondent-name         0
type                    0
type-name               0
timezone                0
timezone-description    0
dtype: int64
Data types:
 period                  object
respondent              object
respondent-name         object
type                    object
type-name               object
timezone                object
timezone-description    object
dtype: object
```

Datasets do no contain any null values

In [17]:
```
dfs = [df_subregion, df_energysource, df_neighbor_bal, df_demand]
for df in dfs:
    df['period'] = pd.to_datetime(df['period'])

for name, df in zip(
    ['Subregion', 'Energy Source', 'Neighbor Balancing', 'Demand'],
    dfs
):
    years = df['period'].dt.year.unique()
    print(f"\n{name} years in data:", sorted(years))
```

```
Subregion years in data: [2023, 2024]

Energy Source years in data: [2023, 2024]

Neighbor Balancing years in data: [2023, 2024]

Demand years in data: [2025]
```

In [18]:
```
print(df_demand.columns)
```

```
Index(['period', 'respondent', 'respondent-name', 'type', 'type-name',
       'timezone', 'timezone-description'],
      dtype='object')
```

In [19]:
```
df_demand.head()
```

Out[19]:

	period	respondent	respondent-name	type	type-name	timezone	timezone-description
0	2025-03-24	AECI	Associated Electric Cooperative, Inc.	DF	Day-ahead demand forecast	Central	Central
1	2025-03-24	AECI	Associated Electric Cooperative, Inc.	DF	Day-ahead demand forecast	Eastern	Eastern
2	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Arizona	Arizona
3	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Central	Central
4	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Eastern	Eastern

In [20]:
```
df_demand.dtypes
```

Out[20]:
```
period                  datetime64[ns]
respondent                      object
respondent-name                 object
type                            object
type-name                       object
timezone                        object
timezone-description            object
dtype: object
```

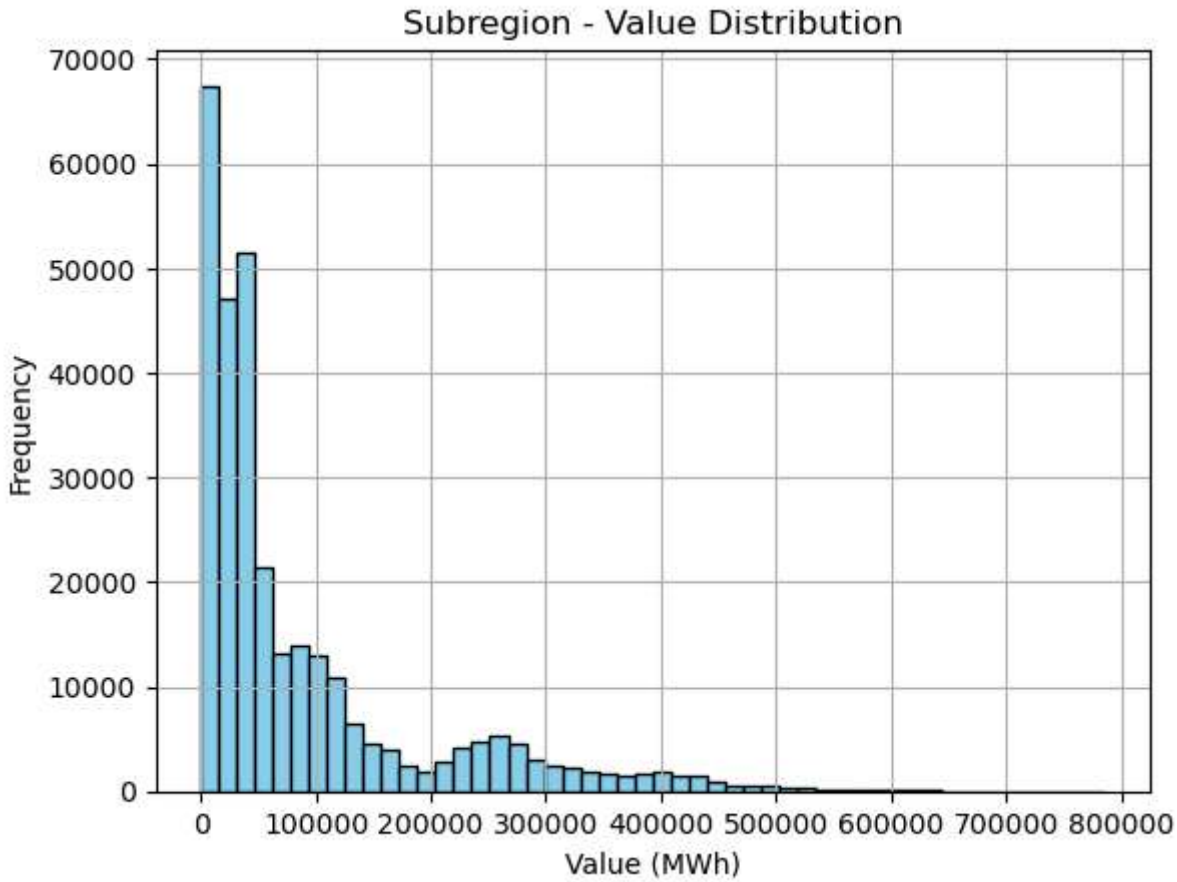
In [21]:
```
for name, df in zip(
    ['Subregion', 'Energy Source', 'Neighbor Balancing', 'Demand'],
    [df_subregion, df_energysource, df_neighbor_bal, df_demand]
):
    print(f"\n{name} value stats:")
    if 'value' in df.columns:
        print(df['value'].describe())
    else:
        print("'value' column not found.")
```

```
Subregion value stats:
count    302719.000000
mean      87875.291197
std      108917.572342
min           0.000000
25%       18723.000000
50%       42308.000000
75%      109606.500000
max      785360.000000
Name: value, dtype: float64

Energy Source value stats:
count    1.425802e+06
mean     8.635683e+04
std      3.142995e+05
min     -2.444800e+04
25%      1.011000e+03
50%      9.493000e+03
75%      4.901500e+04
max      7.156358e+06
Name: value, dtype: float64

Neighbor Balancing value stats:
count    1.233232e+06
mean    -1.812310e+02
std      2.222270e+04
min     -1.772330e+05
25%     -5.127000e+03
50%      0.000000e+00
75%      4.588000e+03
max      3.494280e+05
Name: value, dtype: float64

Demand value stats:
'value' column not found.
```

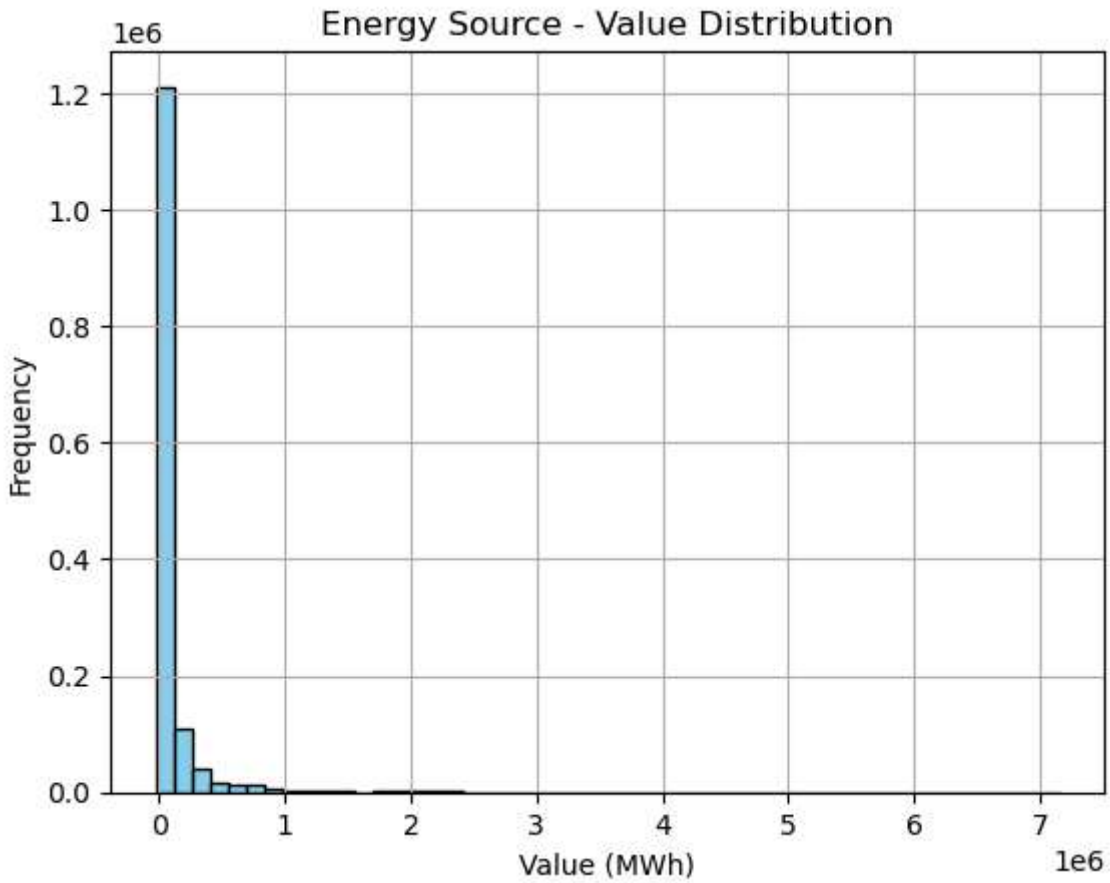
In [22]:

```python
#Value Distribution

for name, df in zip(
    ['Subregion', 'Energy Source', 'Neighbor Balancing'],
    [df_subregion, df_energysource, df_neighbor_bal]
):
    if 'value' in df.columns:
        print(f"\n{name} value distribution:")
        plt.hist(df['value'], bins=50, color='skyblue', edgecolor='black')
        plt.title(f'{name} - Value Distribution')
        plt.xlabel('Value (MWh)')
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.show()
    else:
        print(f"'{name}' has no 'value' column.")
```

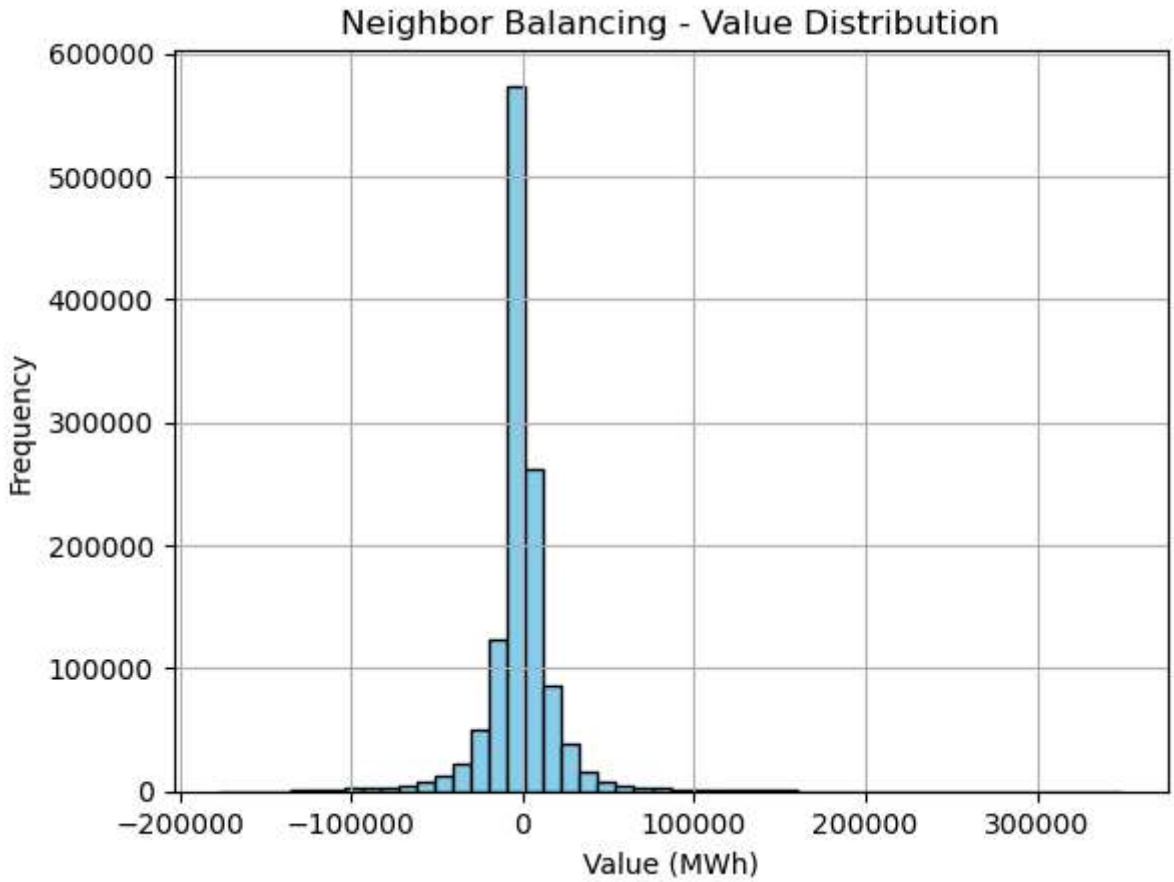
```
Subregion value distribution:
```




```
Energy Source value distribution:
```

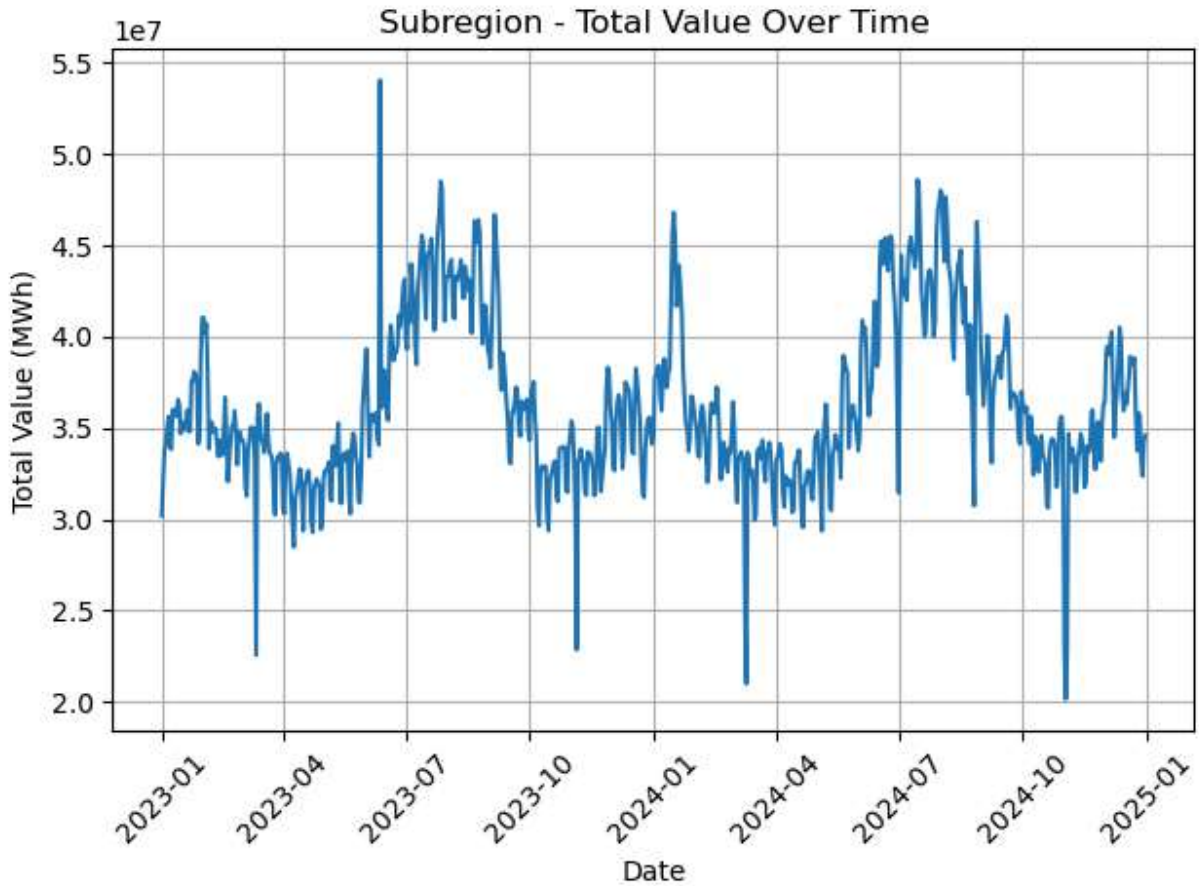



```
Neighbor Balancing value distribution:
```

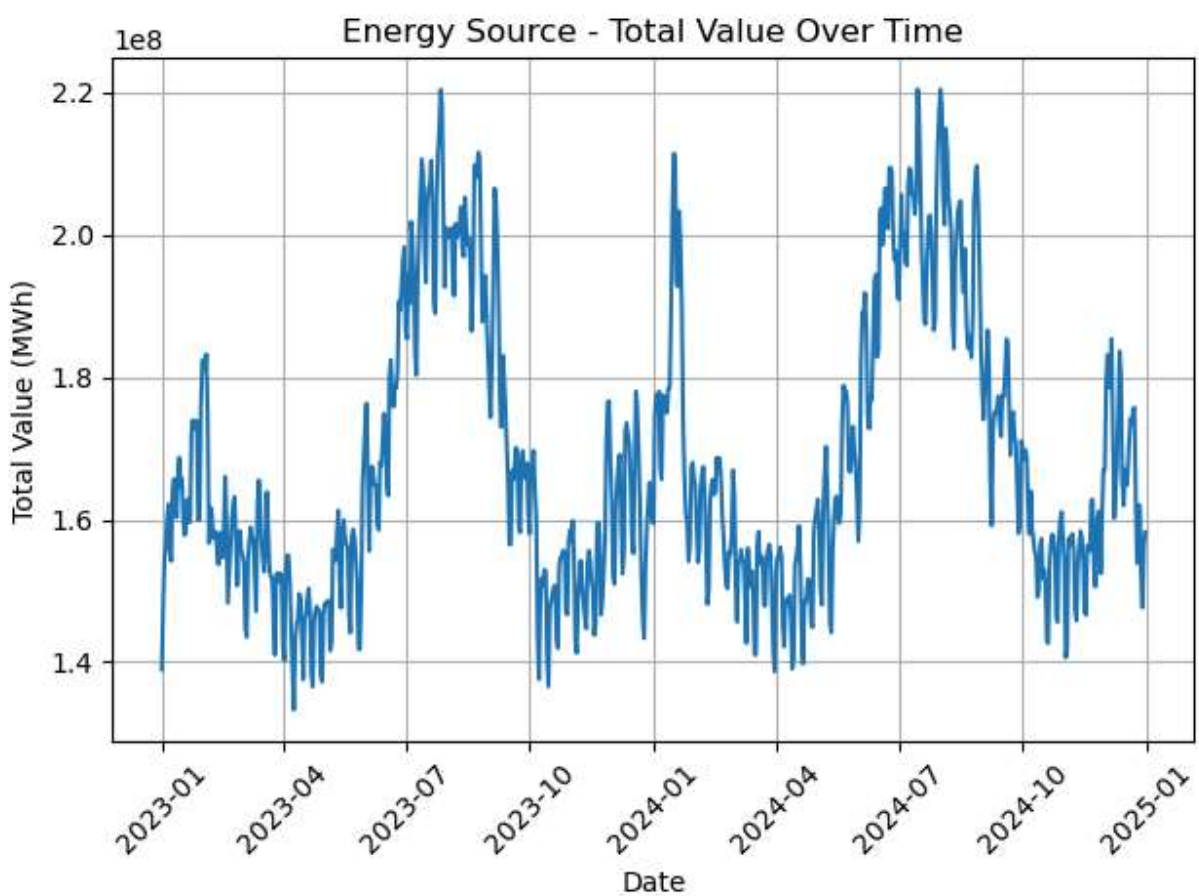


In [23]:
```
for name, df in zip(
    ['Subregion', 'Energy Source', 'Neighbor Balancing'],
    [df_subregion, df_energysource, df_neighbor_bal]
):
    if 'value' in df.columns:
        print(f"\n{name} - Total Value Over Time:")
        df_time = df.copy()
        df_time['period'] = pd.to_datetime(df_time['period'])

        df_time_grouped = df_time.groupby('period')['value'].sum().reset_index()

        plt.plot(df_time_grouped['period'].values, df_time_grouped['value'].values)
        plt.title(f'{name} - Total Value Over Time')
        plt.xlabel('Date')
        plt.ylabel('Total Value (MWh)')
        plt.xticks(rotation=45)
        plt.grid(True)
        plt.tight_layout()
        plt.show()
```

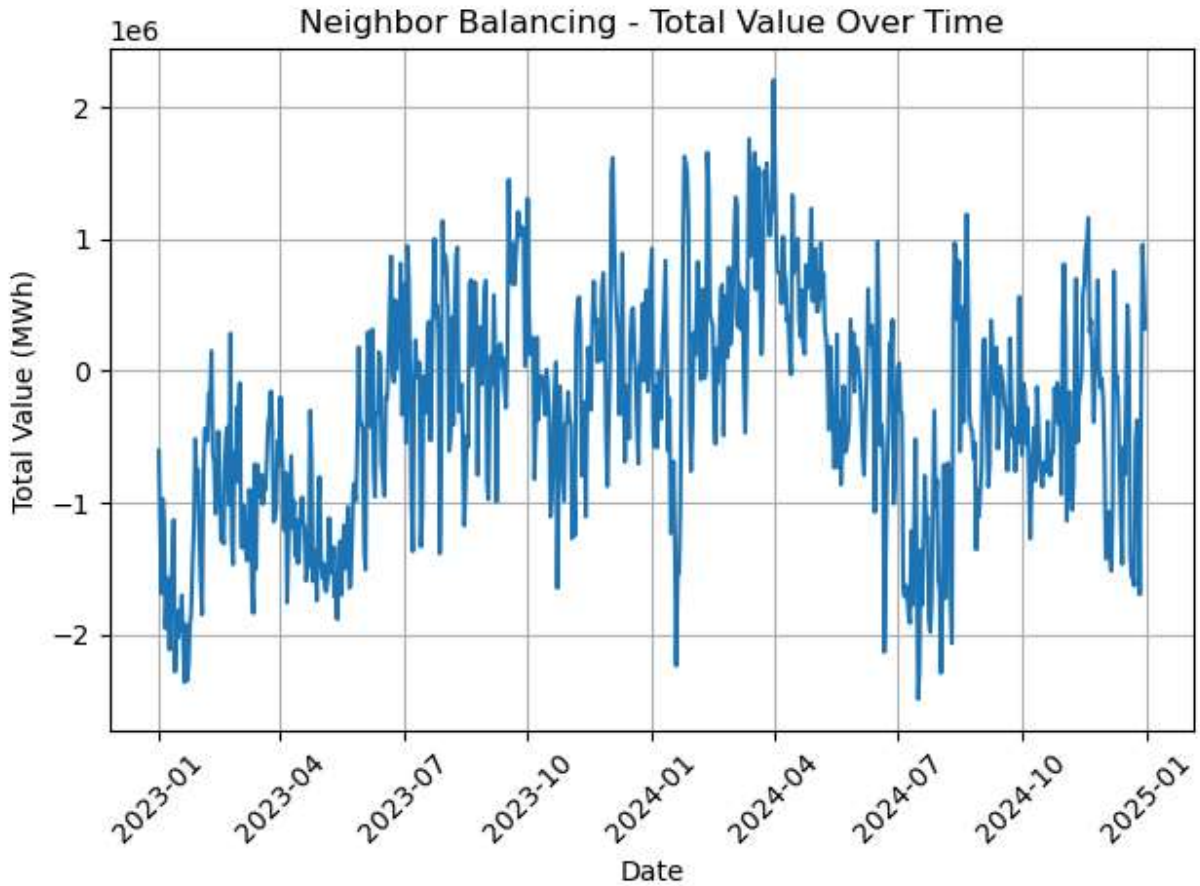
```
Subregion - Total Value Over Time:
```




```
Energy Source - Total Value Over Time:
```




```
Neighbor Balancing - Total Value Over Time:
```



In [24]:
```
#merge "value" across all datasets

df_subregion_time = df_subregion.groupby('period')['value'].sum().reset_index()
df_energysource_time = df_energysource.groupby('period')['value'].sum().reset_index
df_neighbor_time = df_neighbor_bal.groupby('period')['value'].sum().reset_index()
```

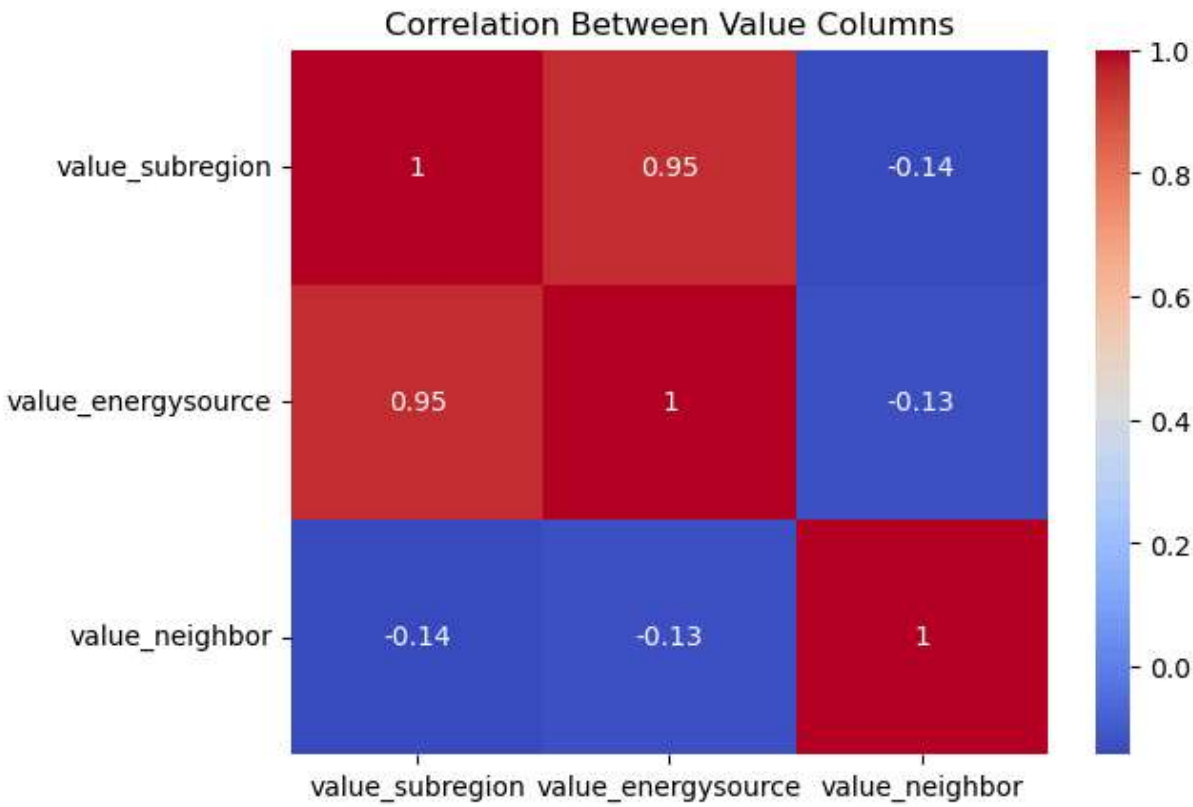
In [25]:
```
#merge for plotting

merged = df_subregion_time.merge(df_energysource_time, on='period', suffixes=('_sub
merged = merged.merge(df_neighbor_time, on='period')
merged.rename(columns={'value': 'value_neighbor'}, inplace=True)
```

In [26]:
```
print(merged.columns)
print(merged.dtypes)
```

```
Index(['period', 'value_subregion', 'value_energysource', 'value_neighbor'], dtype
='object')
period                datetime64[ns]
value_subregion                int64
value_energysource             int64
value_neighbor                 int64
dtype: object
```

In [27]:
```
correlation_matrix = merged[['value_subregion', 'value_energysource', 'value_neighb

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

```
plt.title('Correlation Between Value Columns')
plt.show()
```



Data Preparation

In [28]:
```
# --- Subregion Dataset ---
query_subregion = "SELECT * FROM ecodataaidatabase.subregion"
df_subregion = pd.read_sql(query_subregion, conn)

# Clean: drop unnecessary columns
df_subregion.drop(columns=['subba-name', 'parent-name', 'value-units'], inplace=Tru

# Scrub: drop duplicates and rows missing critical fields
df_subregion.drop_duplicates(inplace=True)
df_subregion.dropna(subset=['period', 'value'], inplace=True)

# Feature engineering: convert period to datetime and create new features
df_subregion['period'] = pd.to_datetime(df_subregion['period'])
df_subregion['month'] = df_subregion['period'].dt.month
df_subregion['weekday'] = df_subregion['period'].dt.weekday

# Preview
print("Subregion DataFrame after cleaning:")
display(df_subregion.head())
```

```
Subregion DataFrame after cleaning:
```

	period	subba	parent	timezone	value	month	weekday
0	2024-12-31	PGAE	CISO	Arizona	247213	12	1
1	2024-12-31	PGAE	CISO	Central	247876	12	1
2	2024-12-31	PGAE	CISO	Eastern	248481	12	1
3	2024-12-31	PGAE	CISO	Mountain	247213	12	1
4	2024-12-31	PGAE	CISO	Pacific	246697	12	1

In [29]:
```
# --- Energysource Dataset ---
query_energysource = "SELECT * FROM ecodataaidatabase.energysource"
df_energysource = pd.read_sql(query_energysource, conn)

# Clean: drop unnecessary columns
df_energysource.drop(columns=['respondent-name', 'type-name', 'timezone-description

# Scrub: drop duplicates and rows missing critical fields
df_energysource.drop_duplicates(inplace=True)
df_energysource.dropna(subset=['period', 'value', 'fueltype'], inplace=True)

# Feature engineering: convert period to datetime and add time features
df_energysource['period'] = pd.to_datetime(df_energysource['period'])
df_energysource['month'] = df_energysource['period'].dt.month
df_energysource['weekday'] = df_energysource['period'].dt.weekday

# Preview
print("Energysource DataFrame after cleaning:")
display(df_energysource.head())
```

```
Energysource DataFrame after cleaning:
```

	period	respondent	fueltype	timezone	value	month	weekday
0	2023-12-12	SE	NG	Eastern	442347	12	1
1	2023-12-12	SE	NG	Mountain	440939	12	1
2	2023-12-12	SE	NG	Pacific	439671	12	1
3	2023-12-12	SE	NUC	Arizona	168001	12	1
4	2023-12-12	SE	NUC	Central	168034	12	1

In [30]:
```
# --- Neighboring Balancing Dataset ---
query_neighboring = "SELECT * FROM ecodataaidatabase.neighboring_bal"
df_neighboring_bal = pd.read_sql(query_neighboring, conn)

# Clean: drop unnecessary columns
df_neighboring_bal.drop(columns=['fromba-name', 'toba-name', 'value-units'], inplac

# Scrub: drop duplicates and rows missing critical fields
df_neighboring_bal.drop_duplicates(inplace=True)
df_neighboring_bal.dropna(subset=['period', 'value', 'fromba', 'toba'], inplace=Tru
```

```
# Feature engineering: convert period to datetime and add new features
df_neighboring_bal['period'] = pd.to_datetime(df_neighboring_bal['period'])
df_neighboring_bal['month'] = df_neighboring_bal['period'].dt.month
df_neighboring_bal['weekday'] = df_neighboring_bal['period'].dt.weekday

# Preview
print("Neighboring Balancing DataFrame after cleaning:")
display(df_neighboring_bal.head())
```

```
Neighboring Balancing DataFrame after cleaning:
```

	period	fromba	toba	timezone	value	month	weekday
0	2024-12-31	AECI	MISO	Arizona	584	12	1
1	2024-12-31	AECI	MISO	Central	359	12	1
2	2024-12-31	AECI	MISO	Eastern	321	12	1
3	2024-12-31	AECI	MISO	Mountain	584	12	1
4	2024-12-31	AECI	MISO	Pacific	1111	12	1

In [31]:

```
# --- Demand Dataset ---
query_demand = "SELECT * FROM ecodataaidatabase.demand"
df_demand = pd.read_sql(query_demand, conn)

# Scrub: drop duplicates and rows missing critical fields
df_demand.drop_duplicates(inplace=True)
df_demand.dropna(subset=['period', 'value'], inplace=True)

# Feature engineering: convert period to datetime and add new features
df_demand['period'] = pd.to_datetime(df_demand['period'])
df_demand['month'] = df_demand['period'].dt.month
df_demand['weekday'] = df_demand['period'].dt.weekday

# Preview
print("Demand DataFrame after cleaning:")
display(df_demand.head())
```

```
Demand DataFrame after cleaning:
```

period	respondent	respondent-name	type	type-name	timezone	timezone-description	value	value-units	mont

In [32]:

```
#Split datasets into train, validation, and test sets

# Define Train/Validation/Test Split Dates
train_cutoff = '2024-10-31'
val_cutoff = '2024-11-30'

# Split by Date
def time_split(df, date_col='period'):
    df[date_col] = pd.to_datetime(df[date_col])
    df = df.sort_values(date_col)
    df_train = df[df[date_col] <= train_cutoff]
```

```
    df_val = df[(df[date_col] > train_cutoff) & (df[date_col] <= val_cutoff)]
    df_test = df[df[date_col] > val_cutoff]
    return df_train, df_val, df_test

#Apply Split to all datasets
dfs_cleaned = {
    'subregion': df_subregion,
    'energysource': df_energysource,
    'neighbor_bal': df_neighboring_bal,
    'demand': df_demand
}

split_datasets = {}

for name, df in dfs_cleaned.items():
    train_df, val_df, test_df = time_split(df)
    split_datasets[name] = {
        'train': train_df,
        'val': val_df,
        'test': test_df
    }
    print(f"{name} split: train({len(train_df)}), val({len(val_df)}), test({len(tes
```

```
subregion split: train(277742), val(12112), test(12865)
energysource split: train(1302730), val(60202), test(62870)
neighbor_bal split: train(1130527), val(50080), test(52625)
demand split: train(0), val(0), test(0)
```

In [33]:

```
# Test Demand Query
df = pd.read_sql("SELECT * FROM ecodataaidatabase.demand LIMIT 10;", conn)
df.head()
```

Out[33]:

	period	respondent	respondent-name	type	type-name	timezone	timezone-description	value	value-units
0	2025-03-24	AECI	Associated Electric Cooperative, Inc.	DF	Day-ahead demand forecast	Central	Central	None	None
1	2025-03-24	AECI	Associated Electric Cooperative, Inc.	DF	Day-ahead demand forecast	Eastern	Eastern	None	None
2	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Arizona	Arizona	None	None
3	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Central	Central	None	None
4	2025-03-24	AVA	Avista Corporation	DF	Day-ahead demand forecast	Eastern	Eastern	None	None

In [34]:

```
#prepare dataset for forecasting
# Step 1: Setup
session = sagemaker.Session()
role = get_execution_role()
bucket = 'ecogridaidata'
prefix = 'eia_electricity/cleaned'
region = boto3.Session().region_name
container = image_uris.retrieve(region=region, framework='forecasting-deepar')

# Step 2: Preprocess dataset
df = split_datasets['subregion']['train'].copy()

# Format timestamp and item_id
df['timestamp'] = pd.to_datetime(df['period'])
df['item_id'] = df['subba'] + '_' + df['parent'] + '_' + df['timezone']
df['target'] = df['value']

# Select and reorder required columns
df = df[['timestamp', 'item_id', 'target']]

# Group each time series by item_id
def format_series(group):
    return {
        "start": str(group['timestamp'].min()),
        "target": group.sort_values('timestamp')['target'].tolist()
    }
```

```
json_series = df.groupby('item_id').apply(format_series).tolist()
json_lines = '\n'.join([str(obj).replace("'", '"') for obj in json_series])
```

```
[04/15/25 03:09:37] INFO     Same images used for training and inference. Def
                             scope: inference.

                    INFO     Ignoring unnecessary instance type: None.
```

In [35]:
```
df
```

Out[35]:

	timestamp	item_id	target
302718	2023-01-01	WR_SWPP_Pacific	70108
302434	2023-01-01	ZONA_NYIS_Arizona	35990
302435	2023-01-01	ZONA_NYIS_Central	36031
302436	2023-01-01	ZONA_NYIS_Eastern	36075
302437	2023-01-01	ZONA_NYIS_Mountain	35990
...	...	...	...
25253	2024-10-31	PS_PJM_Central	108208
25260	2024-10-31	RECO_PJM_Mountain	3778
25252	2024-10-31	PS_PJM_Arizona	108784
25254	2024-10-31	PS_PJM_Eastern	107909
25259	2024-10-31	RECO_PJM_Eastern	3761

277742 rows × 3 columns

In [36]:
```
#save as csv and load to s3
s3_key = f'{prefix}/train/train.json'
s3_path = f's3://{bucket}/{s3_key}'

s3 = boto3.client('s3')
s3.put_object(Bucket=bucket, Key=s3_key, Body=json_lines)
```

Out[36]:
```
{'ResponseMetadata': {'RequestId': 'A36TBZDCF4T379ZS',
  'HostId': '9Z7L2zLDa2EABEVjlx/SJ9Q6SUmss4ar5x3kM4dpXVI/+y6CAmhqMAMBOJB3u+Yx7O1uq
grzBCKHJno6uuWT6tyDuUMmeizk',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amz-id-2': '9Z7L2zLDa2EABEVjlx/SJ9Q6SUmss4ar5x3kM4dpXVI/+y6CA
mhqMAMBOJB3u+Yx7O1uqgrzBCKHJno6uuWT6tyDuUMmeizk',
   'x-amz-request-id': 'A36TBZDCF4T379ZS',
   'date': 'Tue, 15 Apr 2025 03:09:39 GMT',
   'x-amz-server-side-encryption': 'AES256',
   'etag': '"94c4e621a2b51e645041deb2461b79b0"',
   'x-amz-checksum-crc32': 'IHYn3w==',
   'x-amz-checksum-type': 'FULL_OBJECT',
   'content-length': '0',
   'server': 'AmazonS3'},
  'RetryAttempts': 0},
 'ETag': '"94c4e621a2b51e645041deb2461b79b0"',
 'ChecksumCRC32': 'IHYn3w==',
 'ChecksumType': 'FULL_OBJECT',
 'ServerSideEncryption': 'AES256'}
```

In [37]:
```python
#train the model
hyperparameters = {
    "time_freq": "D",
    "epochs": "20",
    "early_stopping_patience": "10",
    "mini_batch_size": "64",
    "learning_rate": "0.001",
    "num_cells": "40",
    "num_layers": "3",
    "likelihood": "gaussian",
    "context_length": "30",
    "prediction_length": "14"
}

estimator = Estimator(
    image_uri=container,
    role=role,
    instance_count=1,
    instance_type="ml.m5.xlarge",
    output_path=f's3://{bucket}/{prefix}/output',
    sagemaker_session=session,
    hyperparameters=hyperparameters
)

estimator.fit({'train': s3_path})
```

```
[04/15/25 03:09:38] INFO     SageMaker Python SDK will collect telemetry to h
                             understand our user's needs, diagnose issues, an
                             additional features.
                             To opt out of telemetry, please disable via Tele
                             parameter in SDK defaults config. For more infor
                             to
                             https://sagemaker.readthedocs.io/en/stable/overv
                             guring-and-using-defaults-with-the-sagemaker-pyt
```

```
INFO     Creating training-job with name:
         forecasting-deepar-2025-04-15-03-09-38-360
```

```
2025-04-15 03:09:41 Starting - Starting the training job...
..25-04-15 03:09:56 Starting - Preparing the instances for training.
.............10:35 Downloading - Downloading the training image.
Docker entrypoint called with argument(s): trainmpleted. Training in progress..
Running default environment configuration script
Running custom environment configuration script
/opt/amazon/lib/python3.8/site-packages/mxnet/model.py:97: SyntaxWarning: "is" with
a literal. Did you mean "=="?
  if num_device is 1 and 'dist' not in kvstore:
[04/15/2025 03:13:12 INFO 140116693747520] Reading default configuration from /opt/a
mazon/lib/python3.8/site-packages/algorithm/resources/default-input.json: {'_kvstor
e': 'auto', '_num_gpus': 'auto', '_num_kv_servers': 'auto', '_tuning_objective_metri
c': '', 'cardinality': 'auto', 'dropout_rate': '0.10', 'early_stopping_patience':
'', 'embedding_dimension': '10', 'learning_rate': '0.001', 'likelihood': 'student-
t', 'mini_batch_size': '128', 'num_cells': '40', 'num_dynamic_feat': 'auto', 'num_ev
al_samples': '100', 'num_layers': '2', 'test_quantiles': '[0.1, 0.2, 0.3, 0.4, 0.5,
0.6, 0.7, 0.8, 0.9]'}
[04/15/2025 03:13:12 INFO 140116693747520] Merging with provided configuration from
/opt/ml/input/config/hyperparameters.json: {'context_length': '30', 'early_stopping_
patience': '10', 'epochs': '20', 'learning_rate': '0.001', 'likelihood': 'gaussian',
'mini_batch_size': '64', 'num_cells': '40', 'num_layers': '3', 'prediction_length':
'14', 'time_freq': 'D'}
[04/15/2025 03:13:12 INFO 140116693747520] Final configuration: {'_kvstore': 'auto',
'_num_gpus': 'auto', '_num_kv_servers': 'auto', '_tuning_objective_metric': '', 'car
dinality': 'auto', 'dropout_rate': '0.10', 'early_stopping_patience': '10', 'embeddi
ng_dimension': '10', 'learning_rate': '0.001', 'likelihood': 'gaussian', 'mini_batch
_size': '64', 'num_cells': '40', 'num_dynamic_feat': 'auto', 'num_eval_samples': '10
0', 'num_layers': '3', 'test_quantiles': '[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9]', 'context_length': '30', 'epochs': '20', 'prediction_length': '14', 'time_fre
q': 'D'}
Process 7 is a worker.
[04/15/2025 03:13:12 INFO 140116693747520] Detected entry point for worker worker
[04/15/2025 03:13:12 INFO 140116693747520] Using early stopping with patience 10
[04/15/2025 03:13:12 INFO 140116693747520] random_seed is None
[04/15/2025 03:13:12 INFO 140116693747520] [cardinality=auto] `cat` field was NOT fo
und in the file `/opt/ml/input/data/train/train.json` and will NOT be used for train
ing.
[04/15/2025 03:13:12 INFO 140116693747520] [num_dynamic_feat=auto] `dynamic_feat` fi
eld was NOT found in the file `/opt/ml/input/data/train/train.json` and will NOT be
used for training.
[04/15/2025 03:13:12 INFO 140116693747520] Training set statistics:
[04/15/2025 03:13:12 INFO 140116693747520] Integer time series
[04/15/2025 03:13:12 INFO 140116693747520] number of time series: 420
[04/15/2025 03:13:12 INFO 140116693747520] number of observations: 277742
[04/15/2025 03:13:12 INFO 140116693747520] mean target length: 661.2904761904762
[04/15/2025 03:13:12 INFO 140116693747520] min/mean/max target: 0.0/88107.4487473986
7/785360.0
[04/15/2025 03:13:12 INFO 140116693747520] mean abs(target): 88107.44874739867
[04/15/2025 03:13:12 INFO 140116693747520] contains missing values: no
[04/15/2025 03:13:12 INFO 140116693747520] Small number of time series. Doing 2 pass
es over dataset with prob 0.7619047619047619 per epoch.
[04/15/2025 03:13:12 INFO 140116693747520] No test channel found not running evaluat
ions
/opt/amazon/lib/python3.8/site-packages/algorithm/core/date_feature_set.py:44: Futur
eWarning: weekofyear and week have been deprecated, please use DatetimeIndex.isocale
ndar().week instead, which returns a Series.  To exactly reproduce the behavior of w
```

```
eek and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar
().week)
  return index.weekofyear / 51.0 - 0.5
[04/15/2025 03:13:12 INFO 140116693747520] #memory_usage::<batchbuffer> = 7.95654296
875 mb
/opt/amazon/python3.8/lib/python3.8/subprocess.py:848: RuntimeWarning: line bufferin
g (buffering=1) isn't supported in binary mode, the default buffer size will be used
  self.stdout = io.open(c2pread, 'rb', bufsize)
[04/15/2025 03:13:12 INFO 140116693747520] nvidia-smi: took 0.031 seconds to run.
[04/15/2025 03:13:12 INFO 140116693747520] nvidia-smi identified 0 GPUs.
[04/15/2025 03:13:12 INFO 140116693747520] Number of GPUs being used: 0
[04/15/2025 03:13:12 INFO 140116693747520] Create Store: local
#metrics {"StartTime": 1744686792.739004, "EndTime": 1744686792.83969, "Dimensions":
{"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metrics":
{"get_graph.time": {"sum": 99.66564178466797, "count": 1, "min": 99.66564178466797,
"max": 99.66564178466797}}}
[04/15/2025 03:13:12 INFO 140116693747520] Number of GPUs being used: 0
[04/15/2025 03:13:13 INFO 140116693747520] #memory_usage::<model> = 28 mb
#metrics {"StartTime": 1744686792.8397665, "EndTime": 1744686793.0103648, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"initialize.time": {"sum": 271.2209224700928, "count": 1, "min": 271.2209224700
928, "max": 271.2209224700928}}}
[03:13:13] /opt/brazil-pkg-cache/packages/AIAlgorithmsMXNet/AIAlgorithmsMXNet-1.3.x_
Cuda_11.1.x.406.0/AL2_x86_64/generic-flavor/src/src/operator/nn/mkldnn/mkldnn_base.c
c:74: Allocate 10240 bytes with malloc directly
[04/15/2025 03:13:13 INFO 140116693747520] Epoch[0] Batch[0] avg_epoch_loss=10.60549
5
[04/15/2025 03:13:13 INFO 140116693747520] #quality_metric: host=algo-1, epoch=0, ba
tch=0 train loss <loss>=10.60549545288086
[04/15/2025 03:13:14 INFO 140116693747520] Epoch[0] Batch[5] avg_epoch_loss=10.25684
0
[04/15/2025 03:13:14 INFO 140116693747520] #quality_metric: host=algo-1, epoch=0, ba
tch=5 train loss <loss>=10.256839752197266
[04/15/2025 03:13:14 INFO 140116693747520] Epoch[0] Batch [5]#011Speed: 414.38 sampl
es/sec#011loss=10.256840
[04/15/2025 03:13:15 INFO 140116693747520] Epoch[0] Batch[10] avg_epoch_loss=10.0369
62
[04/15/2025 03:13:15 INFO 140116693747520] #quality_metric: host=algo-1, epoch=0, ba
tch=10 train loss <loss>=9.773109436035156
[04/15/2025 03:13:15 INFO 140116693747520] Epoch[0] Batch [10]#011Speed: 397.87 samp
les/sec#011loss=9.773109
[04/15/2025 03:13:15 INFO 140116693747520] processed a total of 681 examples
#metrics {"StartTime": 1744686793.0104263, "EndTime": 1744686795.0394442, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"epochs": {"sum": 20.0, "count": 1, "min": 20, "max": 20}, "update.time": {"su
m": 2028.9301872253418, "count": 1, "min": 2028.9301872253418, "max": 2028.930187225
3418}}}
[04/15/2025 03:13:15 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=335.6262502611504 records/second
[04/15/2025 03:13:15 INFO 140116693747520] #progress_metric: host=algo-1, completed
5.0 % of epochs
[04/15/2025 03:13:15 INFO 140116693747520] #quality_metric: host=algo-1, epoch=0, tr
ain loss <loss>=10.036962335759943
[04/15/2025 03:13:15 INFO 140116693747520] best epoch loss so far
[04/15/2025 03:13:15 INFO 140116693747520] Saved checkpoint to "/opt/ml/model/state_
2337351f-ea98-4357-b106-c064e330c53a-0000.params"
```

```
#metrics {"StartTime": 1744686795.0395284, "EndTime": 1744686795.056787, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"state.serialize.time": {"sum": 16.594409942626953, "count": 1, "min": 16.59440
9942626953, "max": 16.594409942626953}}}
[04/15/2025 03:13:15 INFO 140116693747520] Epoch[1] Batch[0] avg_epoch_loss=10.29657
2
[04/15/2025 03:13:15 INFO 140116693747520] #quality_metric: host=algo-1, epoch=1, ba
tch=0 train loss <loss>=10.296571731567383
[04/15/2025 03:13:16 INFO 140116693747520] Epoch[1] Batch[5] avg_epoch_loss=9.840743
[04/15/2025 03:13:16 INFO 140116693747520] #quality_metric: host=algo-1, epoch=1, ba
tch=5 train loss <loss>=9.84074322382609
[04/15/2025 03:13:16 INFO 140116693747520] Epoch[1] Batch [5]#011Speed: 297.52 sampl
es/sec#011loss=9.840743
[04/15/2025 03:13:17 INFO 140116693747520] Epoch[1] Batch[10] avg_epoch_loss=9.44529
9
[04/15/2025 03:13:17 INFO 140116693747520] #quality_metric: host=algo-1, epoch=1, ba
tch=10 train loss <loss>=8.970765399932862
[04/15/2025 03:13:17 INFO 140116693747520] Epoch[1] Batch [10]#011Speed: 311.08 samp
les/sec#011loss=8.970765
[04/15/2025 03:13:17 INFO 140116693747520] processed a total of 647 examples
#metrics {"StartTime": 1744686795.0568552, "EndTime": 1744686797.5026999, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 2445.7828998565674, "count": 1, "min": 2445.782899856567
4, "max": 2445.7828998565674}}}
[04/15/2025 03:13:17 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=264.5263469720392 records/second
[04/15/2025 03:13:17 INFO 140116693747520] #progress_metric: host=algo-1, completed
10.0 % of epochs
[04/15/2025 03:13:17 INFO 140116693747520] #quality_metric: host=algo-1, epoch=1, tr
ain loss <loss>=9.445298758420078
[04/15/2025 03:13:17 INFO 140116693747520] best epoch loss so far
[04/15/2025 03:13:17 INFO 140116693747520] Saved checkpoint to "/opt/ml/model/state_
795abb4e-ea92-49fb-81d5-5a30c744960b-0000.params"
#metrics {"StartTime": 1744686797.5027645, "EndTime": 1744686797.5205293, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"state.serialize.time": {"sum": 17.307043075561523, "count": 1, "min": 17.30704
3075561523, "max": 17.307043075561523}}}
[04/15/2025 03:13:17 INFO 140116693747520] Epoch[2] Batch[0] avg_epoch_loss=9.470450
[04/15/2025 03:13:17 INFO 140116693747520] #quality_metric: host=algo-1, epoch=2, ba
tch=0 train loss <loss>=9.470450401306152
[04/15/2025 03:13:18 INFO 140116693747520] Epoch[2] Batch[5] avg_epoch_loss=9.583484
[04/15/2025 03:13:18 INFO 140116693747520] #quality_metric: host=algo-1, epoch=2, ba
tch=5 train loss <loss>=9.583484013875326
[04/15/2025 03:13:18 INFO 140116693747520] Epoch[2] Batch [5]#011Speed: 405.08 sampl
es/sec#011loss=9.583484
[04/15/2025 03:13:19 INFO 140116693747520] processed a total of 619 examples
#metrics {"StartTime": 1744686797.5206003, "EndTime": 1744686799.2790194, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1758.3611011505127, "count": 1, "min": 1758.361101150512
7, "max": 1758.3611011505127}}}
[04/15/2025 03:13:19 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=352.0119747359104 records/second
[04/15/2025 03:13:19 INFO 140116693747520] #progress_metric: host=algo-1, completed
15.0 % of epochs
[04/15/2025 03:13:19 INFO 140116693747520] #quality_metric: host=algo-1, epoch=2, tr
ain loss <loss>=9.756263542175294
```

```
[04/15/2025 03:13:19 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:19 INFO 140116693747520] Epoch[3] Batch[0] avg_epoch_loss=9.460284
[04/15/2025 03:13:19 INFO 140116693747520] #quality_metric: host=algo-1, epoch=3, ba
tch=0 train loss <loss>=9.460284233093262
[04/15/2025 03:13:20 INFO 140116693747520] Epoch[3] Batch[5] avg_epoch_loss=9.595903
[04/15/2025 03:13:20 INFO 140116693747520] #quality_metric: host=algo-1, epoch=3, ba
tch=5 train loss <loss>=9.595903396606445
[04/15/2025 03:13:20 INFO 140116693747520] Epoch[3] Batch [5]#011Speed: 409.40 sampl
es/sec#011loss=9.595903
[04/15/2025 03:13:21 INFO 140116693747520] Epoch[3] Batch[10] avg_epoch_loss=9.64633
0
[04/15/2025 03:13:21 INFO 140116693747520] #quality_metric: host=algo-1, epoch=3, ba
tch=10 train loss <loss>=9.706842231750489
[04/15/2025 03:13:21 INFO 140116693747520] Epoch[3] Batch [10]#011Speed: 396.29 samp
les/sec#011loss=9.706842
[04/15/2025 03:13:21 INFO 140116693747520] processed a total of 651 examples
#metrics {"StartTime": 1744686799.2790923, "EndTime": 1744686801.212481, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1932.9204559326172, "count": 1, "min": 1932.920455932617
2, "max": 1932.9204559326172}}}
[04/15/2025 03:13:21 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=336.7772039207125 records/second
[04/15/2025 03:13:21 INFO 140116693747520] #progress_metric: host=algo-1, completed
20.0 % of epochs
[04/15/2025 03:13:21 INFO 140116693747520] #quality_metric: host=algo-1, epoch=3, tr
ain loss <loss>=9.646330139853738
[04/15/2025 03:13:21 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:21 INFO 140116693747520] Epoch[4] Batch[0] avg_epoch_loss=10.11667
8
[04/15/2025 03:13:21 INFO 140116693747520] #quality_metric: host=algo-1, epoch=4, ba
tch=0 train loss <loss>=10.116678237915039
[04/15/2025 03:13:22 INFO 140116693747520] Epoch[4] Batch[5] avg_epoch_loss=9.944225
[04/15/2025 03:13:22 INFO 140116693747520] #quality_metric: host=algo-1, epoch=4, ba
tch=5 train loss <loss>=9.94422467549642
[04/15/2025 03:13:22 INFO 140116693747520] Epoch[4] Batch [5]#011Speed: 389.44 sampl
es/sec#011loss=9.944225
[04/15/2025 03:13:23 INFO 140116693747520] processed a total of 634 examples
#metrics {"StartTime": 1744686801.2125568, "EndTime": 1744686803.03536, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1822.4895000457764, "count": 1, "min": 1822.489500045776
4, "max": 1822.4895000457764}}}
[04/15/2025 03:13:23 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=347.85705515621913 records/second
[04/15/2025 03:13:23 INFO 140116693747520] #progress_metric: host=algo-1, completed
25.0 % of epochs
[04/15/2025 03:13:23 INFO 140116693747520] #quality_metric: host=algo-1, epoch=4, tr
ain loss <loss>=9.95140905380249
[04/15/2025 03:13:23 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:23 INFO 140116693747520] Epoch[5] Batch[0] avg_epoch_loss=9.650592
[04/15/2025 03:13:23 INFO 140116693747520] #quality_metric: host=algo-1, epoch=5, ba
tch=0 train loss <loss>=9.650591850280762
[04/15/2025 03:13:24 INFO 140116693747520] Epoch[5] Batch[5] avg_epoch_loss=9.645081
[04/15/2025 03:13:24 INFO 140116693747520] #quality_metric: host=algo-1, epoch=5, ba
tch=5 train loss <loss>=9.645080725351969
[04/15/2025 03:13:24 INFO 140116693747520] Epoch[5] Batch [5]#011Speed: 416.63 sampl
es/sec#011loss=9.645081
```

```
[04/15/2025 03:13:24 INFO 140116693747520] processed a total of 629 examples
#metrics {"StartTime": 1744686803.035429, "EndTime": 1744686804.7758543, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1739.8598194122314, "count": 1, "min": 1739.859819412231
4, "max": 1739.8598194122314}}}
[04/15/2025 03:13:24 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=361.5033156086143 records/second
[04/15/2025 03:13:24 INFO 140116693747520] #progress_metric: host=algo-1, completed
30.0 % of epochs
[04/15/2025 03:13:24 INFO 140116693747520] #quality_metric: host=algo-1, epoch=5, tr
ain loss <loss>=9.664748668670654
[04/15/2025 03:13:24 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:25 INFO 140116693747520] Epoch[6] Batch[0] avg_epoch_loss=9.735014
[04/15/2025 03:13:25 INFO 140116693747520] #quality_metric: host=algo-1, epoch=6, ba
tch=0 train loss <loss>=9.735013961791992
[04/15/2025 03:13:25 INFO 140116693747520] Epoch[6] Batch[5] avg_epoch_loss=9.814679
[04/15/2025 03:13:25 INFO 140116693747520] #quality_metric: host=algo-1, epoch=6, ba
tch=5 train loss <loss>=9.81467898686727
[04/15/2025 03:13:25 INFO 140116693747520] Epoch[6] Batch [5]#011Speed: 411.86 sampl
es/sec#011loss=9.814679
[04/15/2025 03:13:26 INFO 140116693747520] Epoch[6] Batch[10] avg_epoch_loss=9.81108
2
[04/15/2025 03:13:26 INFO 140116693747520] #quality_metric: host=algo-1, epoch=6, ba
tch=10 train loss <loss>=9.806766319274903
[04/15/2025 03:13:26 INFO 140116693747520] Epoch[6] Batch [10]#011Speed: 393.95 samp
les/sec#011loss=9.806766
[04/15/2025 03:13:26 INFO 140116693747520] processed a total of 667 examples
#metrics {"StartTime": 1744686804.7759233, "EndTime": 1744686806.7067137, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1930.3345680236816, "count": 1, "min": 1930.334568023681
6, "max": 1930.3345680236816}}}
[04/15/2025 03:13:26 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=345.5200377730267 records/second
[04/15/2025 03:13:26 INFO 140116693747520] #progress_metric: host=algo-1, completed
35.0 % of epochs
[04/15/2025 03:13:26 INFO 140116693747520] #quality_metric: host=algo-1, epoch=6, tr
ain loss <loss>=9.81108231977983
[04/15/2025 03:13:26 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:27 INFO 140116693747520] Epoch[7] Batch[0] avg_epoch_loss=9.889005
[04/15/2025 03:13:27 INFO 140116693747520] #quality_metric: host=algo-1, epoch=7, ba
tch=0 train loss <loss>=9.889004707336426
[04/15/2025 03:13:27 INFO 140116693747520] Epoch[7] Batch[5] avg_epoch_loss=9.848986
[04/15/2025 03:13:27 INFO 140116693747520] #quality_metric: host=algo-1, epoch=7, ba
tch=5 train loss <loss>=9.848985513051352
[04/15/2025 03:13:27 INFO 140116693747520] Epoch[7] Batch [5]#011Speed: 384.88 sampl
es/sec#011loss=9.848986
[04/15/2025 03:13:28 INFO 140116693747520] processed a total of 636 examples
#metrics {"StartTime": 1744686806.7067745, "EndTime": 1744686808.5134547, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1806.3783645629883, "count": 1, "min": 1806.378364562988
3, "max": 1806.3783645629883}}}
[04/15/2025 03:13:28 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=352.0659560441736 records/second
[04/15/2025 03:13:28 INFO 140116693747520] #progress_metric: host=algo-1, completed
40.0 % of epochs
[04/15/2025 03:13:28 INFO 140116693747520] #quality_metric: host=algo-1, epoch=7, tr
```

```
ain loss <loss>=9.775450801849365
[04/15/2025 03:13:28 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:28 INFO 140116693747520] Epoch[8] Batch[0] avg_epoch_loss=9.475825
[04/15/2025 03:13:28 INFO 140116693747520] #quality_metric: host=algo-1, epoch=8, ba
tch=0 train loss <loss>=9.475825309753418
[04/15/2025 03:13:29 INFO 140116693747520] Epoch[8] Batch[5] avg_epoch_loss=9.648054
[04/15/2025 03:13:29 INFO 140116693747520] #quality_metric: host=algo-1, epoch=8, ba
tch=5 train loss <loss>=9.648054440816244
[04/15/2025 03:13:29 INFO 140116693747520] Epoch[8] Batch [5]#011Speed: 407.55 sampl
es/sec#011loss=9.648054
[04/15/2025 03:13:30 INFO 140116693747520] Epoch[8] Batch[10] avg_epoch_loss=9.58260
3
[04/15/2025 03:13:30 INFO 140116693747520] #quality_metric: host=algo-1, epoch=8, ba
tch=10 train loss <loss>=9.504060554504395
[04/15/2025 03:13:30 INFO 140116693747520] Epoch[8] Batch [10]#011Speed: 399.09 samp
les/sec#011loss=9.504061
[04/15/2025 03:13:30 INFO 140116693747520] processed a total of 672 examples
#metrics {"StartTime": 1744686808.513524, "EndTime": 1744686810.4390507, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1924.9608516693115, "count": 1, "min": 1924.960851669311
5, "max": 1924.9608516693115}}}
[04/15/2025 03:13:30 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=349.08114662134403 records/second
[04/15/2025 03:13:30 INFO 140116693747520] #progress_metric: host=algo-1, completed
45.0 % of epochs
[04/15/2025 03:13:30 INFO 140116693747520] #quality_metric: host=algo-1, epoch=8, tr
ain loss <loss>=9.582602674310857
[04/15/2025 03:13:30 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:30 INFO 140116693747520] Epoch[9] Batch[0] avg_epoch_loss=9.639515
[04/15/2025 03:13:30 INFO 140116693747520] #quality_metric: host=algo-1, epoch=9, ba
tch=0 train loss <loss>=9.639514923095703
[04/15/2025 03:13:31 INFO 140116693747520] Epoch[9] Batch[5] avg_epoch_loss=9.696150
[04/15/2025 03:13:31 INFO 140116693747520] #quality_metric: host=algo-1, epoch=9, ba
tch=5 train loss <loss>=9.696150302886963
[04/15/2025 03:13:31 INFO 140116693747520] Epoch[9] Batch [5]#011Speed: 406.94 sampl
es/sec#011loss=9.696150
[04/15/2025 03:13:32 INFO 140116693747520] Epoch[9] Batch[10] avg_epoch_loss=9.77104
5
[04/15/2025 03:13:32 INFO 140116693747520] #quality_metric: host=algo-1, epoch=9, ba
tch=10 train loss <loss>=9.860919189453124
[04/15/2025 03:13:32 INFO 140116693747520] Epoch[9] Batch [10]#011Speed: 410.42 samp
les/sec#011loss=9.860919
[04/15/2025 03:13:32 INFO 140116693747520] processed a total of 651 examples
#metrics {"StartTime": 1744686810.4391108, "EndTime": 1744686812.3494596, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1909.8989963531494, "count": 1, "min": 1909.898996353149
4, "max": 1909.8989963531494}}}
[04/15/2025 03:13:32 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=340.8400210408952 records/second
[04/15/2025 03:13:32 INFO 140116693747520] #progress_metric: host=algo-1, completed
50.0 % of epochs
[04/15/2025 03:13:32 INFO 140116693747520] #quality_metric: host=algo-1, epoch=9, tr
ain loss <loss>=9.771045251326127
[04/15/2025 03:13:32 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:32 INFO 140116693747520] Epoch[10] Batch[0] avg_epoch_loss=9.58288
2
```

```
[04/15/2025 03:13:32 INFO 140116693747520] #quality_metric: host=algo-1, epoch=10, b
atch=0 train loss <loss>=9.582881927490234
[04/15/2025 03:13:33 INFO 140116693747520] Epoch[10] Batch[5] avg_epoch_loss=9.87788
7
[04/15/2025 03:13:33 INFO 140116693747520] #quality_metric: host=algo-1, epoch=10, b
atch=5 train loss <loss>=9.87788724899292
[04/15/2025 03:13:33 INFO 140116693747520] Epoch[10] Batch [5]#011Speed: 412.09 samp
les/sec#011loss=9.877887

2025-04-15 03:13:42 Uploading - Uploading generated training model[04/15/2025 03:13:
34 INFO 140116693747520] processed a total of 603 examples
#metrics {"StartTime": 1744686812.3495188, "EndTime": 1744686814.096717, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1746.8862533569336, "count": 1, "min": 1746.886253356933
6, "max": 1746.8862533569336}}}
[04/15/2025 03:13:34 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=345.16526071170637 records/second
[04/15/2025 03:13:34 INFO 140116693747520] #progress_metric: host=algo-1, completed
55.0 % of epochs
[04/15/2025 03:13:34 INFO 140116693747520] #quality_metric: host=algo-1, epoch=10, t
rain loss <loss>=9.67171607017517
[04/15/2025 03:13:34 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:34 INFO 140116693747520] Epoch[11] Batch[0] avg_epoch_loss=10.0929
97
[04/15/2025 03:13:34 INFO 140116693747520] #quality_metric: host=algo-1, epoch=11, b
atch=0 train loss <loss>=10.092996597290039
[04/15/2025 03:13:35 INFO 140116693747520] Epoch[11] Batch[5] avg_epoch_loss=9.68541
6
[04/15/2025 03:13:35 INFO 140116693747520] #quality_metric: host=algo-1, epoch=11, b
atch=5 train loss <loss>=9.685415585835775
[04/15/2025 03:13:35 INFO 140116693747520] Epoch[11] Batch [5]#011Speed: 402.49 samp
les/sec#011loss=9.685416
[04/15/2025 03:13:36 INFO 140116693747520] Epoch[11] Batch[10] avg_epoch_loss=9.6212
54
[04/15/2025 03:13:36 INFO 140116693747520] #quality_metric: host=algo-1, epoch=11, b
atch=10 train loss <loss>=9.544259643554687
[04/15/2025 03:13:36 INFO 140116693747520] Epoch[11] Batch [10]#011Speed: 400.30 sam
ples/sec#011loss=9.544260
[04/15/2025 03:13:36 INFO 140116693747520] processed a total of 660 examples
#metrics {"StartTime": 1744686814.0967913, "EndTime": 1744686816.0379696, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"update.time": {"sum": 1940.7570362091064, "count": 1, "min": 1940.757036209106
4, "max": 1940.7570362091064}}}
[04/15/2025 03:13:36 INFO 140116693747520] #throughput_metric: host=algo-1, train th
roughput=340.0578146305509 records/second
[04/15/2025 03:13:36 INFO 140116693747520] #progress_metric: host=algo-1, completed
60.0 % of epochs
[04/15/2025 03:13:36 INFO 140116693747520] #quality_metric: host=algo-1, epoch=11, t
rain loss <loss>=9.621253793889826
[04/15/2025 03:13:36 INFO 140116693747520] loss did not improve
[04/15/2025 03:13:36 INFO 140116693747520] Loading parameters from best epoch (1)
#metrics {"StartTime": 1744686816.0380323, "EndTime": 1744686816.0464973, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"state.deserialize.time": {"sum": 8.116722106933594, "count": 1, "min": 8.11672
2106933594, "max": 8.116722106933594}}}
[04/15/2025 03:13:36 INFO 140116693747520] stopping training now
```

```
[04/15/2025 03:13:36 INFO 140116693747520] #progress_metric: host=algo-1, completed
100 % of epochs
[04/15/2025 03:13:36 INFO 140116693747520] Final loss: 9.445298758420078 (occurred a
t epoch 1)
[04/15/2025 03:13:36 INFO 140116693747520] #quality_metric: host=algo-1, train final
_loss <loss>=9.445298758420078
[04/15/2025 03:13:36 INFO 140116693747520] Worker algo-1 finished training.
[04/15/2025 03:13:36 WARNING 140116693747520] wait_for_all_workers will not sync wor
kers since the kv store is not running distributed
[04/15/2025 03:13:36 INFO 140116693747520] All workers finished. Serializing model f
or prediction.
#metrics {"StartTime": 1744686816.0465643, "EndTime": 1744686816.1746893, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"get_graph.time": {"sum": 127.45881080627441, "count": 1, "min": 127.4588108062
7441, "max": 127.45881080627441}}}
[04/15/2025 03:13:36 INFO 140116693747520] Number of GPUs being used: 0
#metrics {"StartTime": 1744686816.1747582, "EndTime": 1744686816.2330425, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"finalize.time": {"sum": 185.84465980529785, "count": 1, "min": 185.84465980529
785, "max": 185.84465980529785}}}
[04/15/2025 03:13:36 INFO 140116693747520] Serializing to /opt/ml/model/model_algo-1
[04/15/2025 03:13:36 INFO 140116693747520] Saved checkpoint to "/opt/ml/model/model_
algo-1-0000.params"
#metrics {"StartTime": 1744686816.2331126, "EndTime": 1744686816.242627, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"model.serialize.time": {"sum": 9.477853775024414, "count": 1, "min": 9.4778537
75024414, "max": 9.477853775024414}}}
[04/15/2025 03:13:36 INFO 140116693747520] Successfully serialized the model for pre
diction.
[04/15/2025 03:13:36 INFO 140116693747520] No test data passed, skipping evaluation.
#metrics {"StartTime": 1744686816.2426713, "EndTime": 1744686816.2520661, "Dimension
s": {"Algorithm": "AWS/DeepAR", "Host": "algo-1", "Operation": "training"}, "Metric
s": {"setuptime": {"sum": 4.340887069702148, "count": 1, "min": 4.340887069702148,
"max": 4.340887069702148}, "totaltime": {"sum": 23768.449544906616, "count": 1, "mi
n": 23768.449544906616, "max": 23768.449544906616}}}

2025-04-15 03:13:56 Completed - Training job completed
Training seconds: 220
Billable seconds: 220
```