**Data Preprocessing Steps:**

**1. Data Consolidation**

To begin the analysis, I first consolidated the electricity demand and weather datasets into a unified structure. The electricity consumption data came from four CSV files sourced from the EIA (Energy Information Administration):

* EIA930\_SUBREGION\_2018\_Jul\_Dec.csv
* EIA930\_SUBREGION\_2019\_Jan\_Jun.csv
* EIA930\_SUBREGION\_2019\_Jul\_Dec.csv
* EIA930\_SUBREGION\_2020\_Jan\_Jun.csv

These files contained hourly electricity demand records for the top 10 most populous U.S. cities, spanning from July 2018 to May 2020. I concatenated these files into a single comprehensive dataset to cover the full time window.

In parallel, I integrated weather data from 10 .json files—each corresponding to one city:

* dallas.json, houston.json, la.json, nyc.json, philadelphia.json,  
  phoenix.json, san\_antonio.json, san\_diego.json, san\_jose.json, seattle.json

I matched each weather record to its corresponding electricity demand entry using timestamps as the key. This created a unified dataset that combines both electricity demand and localized weather conditions on an hourly basis across all cities.

**2. Initial Inspection & Cleaning**

Once merged, I conducted a schema inspection and reviewed sample records to ensure consistency across files. I checked for and documented any mismatches in datetime formats, column names, and value ranges, especially where city-specific records might differ subtly.

**3. Handling Missing Values**

I assessed the dataset for missing or null entries, especially in critical columns like demand, temperature, and humidity. My approach included:

* Simple imputation for minor gaps using forward-fill or median-based replacement
* Dropping rows where critical information (like timestamp or demand) was completely absent
* Special care was taken to avoid distorting hourly time series continuity

**4. Feature Engineering**

To enrich the dataset and enable meaningful analysis, I derived several time-based and contextual features:

* Hour of day (0–23)
* Day of week (0 = Monday, ..., 6 = Sunday)
* Month
* Weekend indicator (binary)
* Season (mapped from month)
* Normalized demand and weather-related features using Min-Max Scaling

These engineered features proved essential for later stages such as clustering and anomaly detection.

**5. Data Aggregation**

In addition to hourly granularity, I created weekly summaries to observe broader trends and patterns over time. These included:

* Average and peak demand per day/week
* Daily temperature and humidity trends
* City-wise summaries to compare regional usage behaviors

**6. Anomaly & Error Detection**

To ensure data quality and robustness, I implemented both statistical and ML-based anomaly detection methods:

* Z-score and IQR-based techniques for detecting outliers in demand and weather features
* Isolation Forest to catch unusual patterns that might signify sensor faults or extreme events

Types of anomalies flagged included:

* Sudden, unexplained demand spikes or dips
* Impossible weather values
* Missing or duplicated timestamps

Each anomaly was reviewed, and depending on the case, the record was either:

* Removed (if clearly erroneous and isolated)
* Imputed (if part of a predictable pattern or small gap)
* Left untouched but flagged for further inspection

**Summary**

This preprocessing phase was critical to building a clean, reliable foundation for downstream analysis. By merging, cleaning, engineering, and validating the data, I ensured that the unified dataset accurately reflects real-world energy consumption behavior in the context of weather and time. These steps enabled robust clustering and pattern discovery in the later stages of the project.

**Clustering Analysis**

**Overview**

This section explores clustering techniques to identify distinct patterns in electricity demand influenced by time, weather, and day-type variables. The dataset used comprises **165,192 records with 21 features**, from which a representative sample of **16,000 entries** was selected for efficient computation and experimentation.

**Feature Selection and Preprocessing**

To guide clustering, I selected three features indicative of demand behavior:

* **Demand (MW)**
* **Hour of Day**
* **Weekend Indicator**

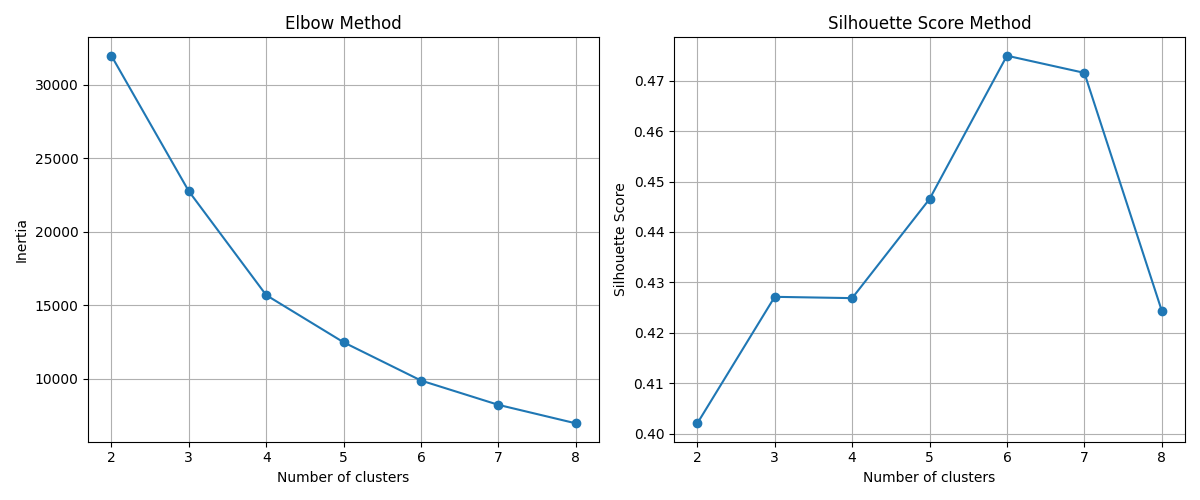
These features were normalized for consistency. For dimensionality reduction:

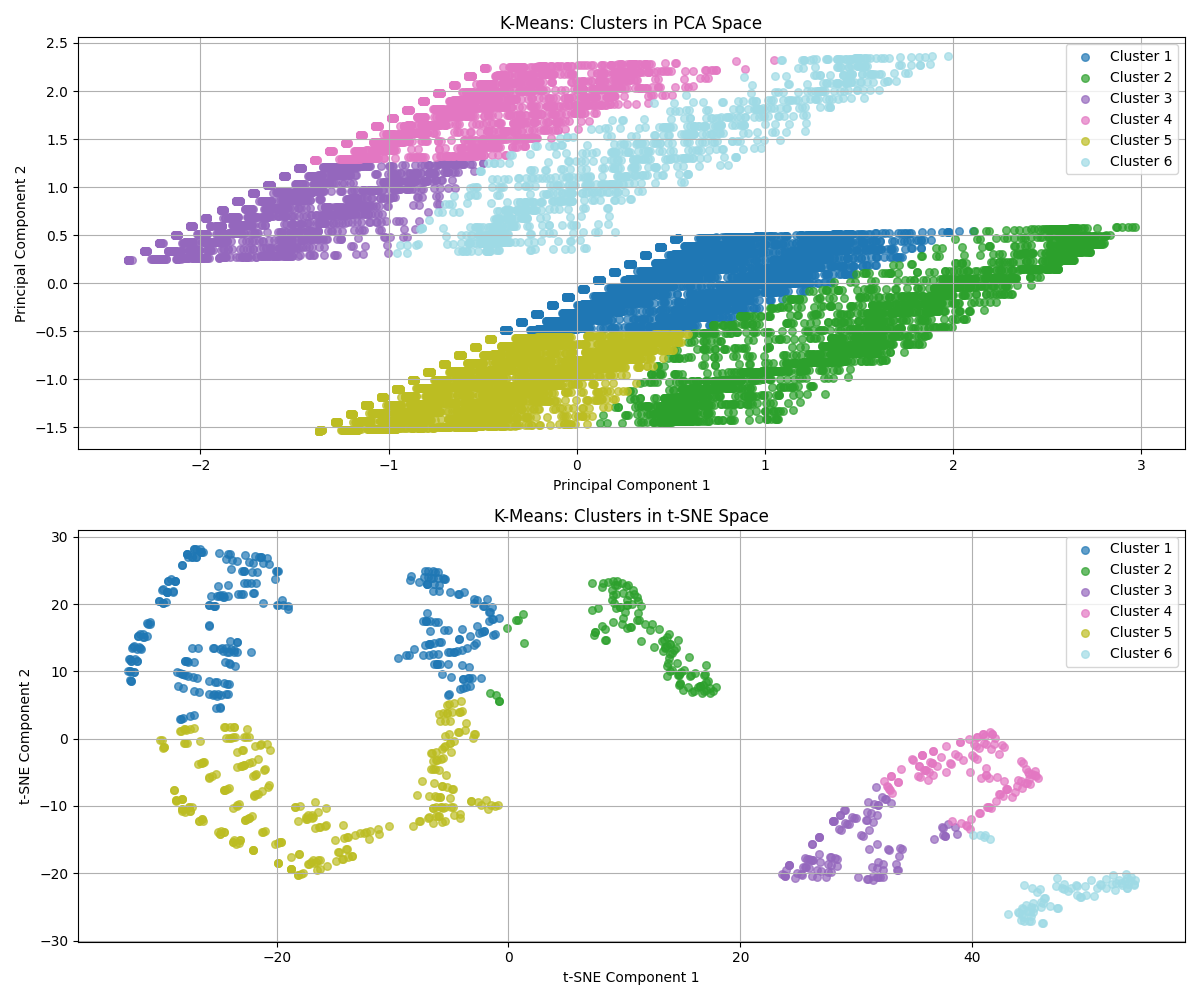
* **PCA** was applied first, retaining **68% of the total variance** in two components.
* **t-SNE** was used on a reduced sample for visualization while maintaining local structure.

**K-Means Clustering**

I experimented with cluster counts ranging from **K=2 to K=8**. Based on inertia and silhouette scores, **K=6** was found to be optimal:

* **Best silhouette score:** **0.4750**
* Clusters formed were distinct and meaningful, separating low-demand, peak-demand, and time-of-day usage patterns.
* Key insights:
  + Clusters 2 and 6 revealed **peak demand periods**, primarily on **weekdays and weekends**, respectively.
  + Clusters 1, 3, 4, and 5 represented **low to moderate demand**, differentiated by temperature and time.

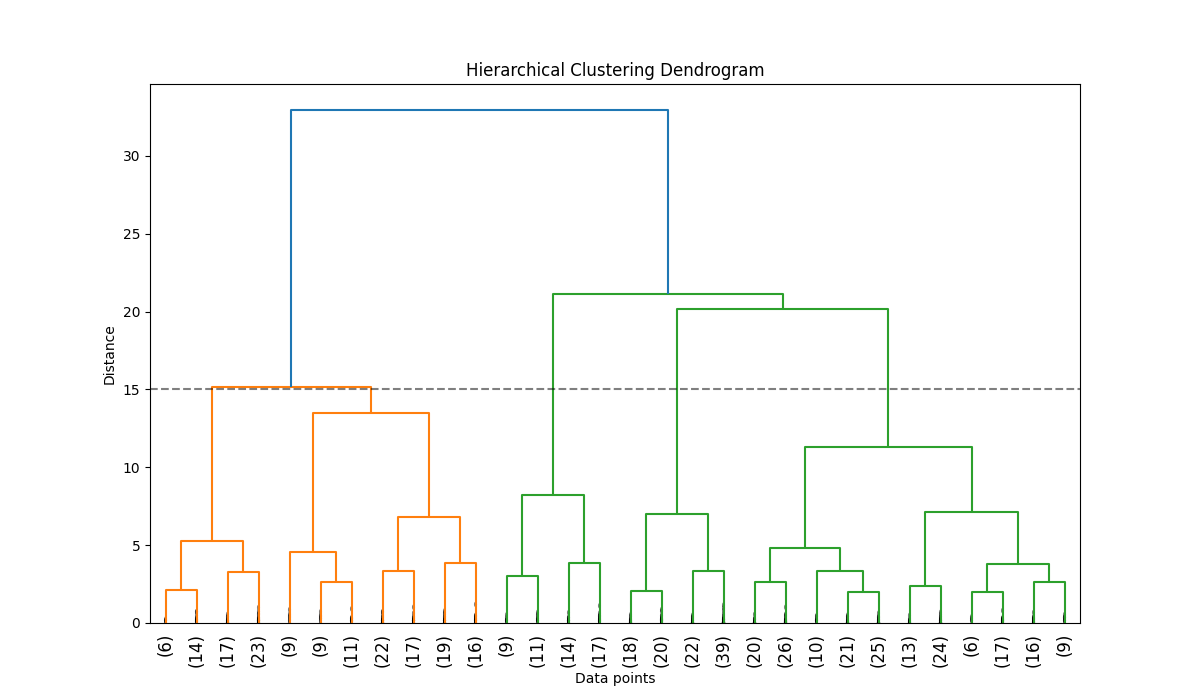


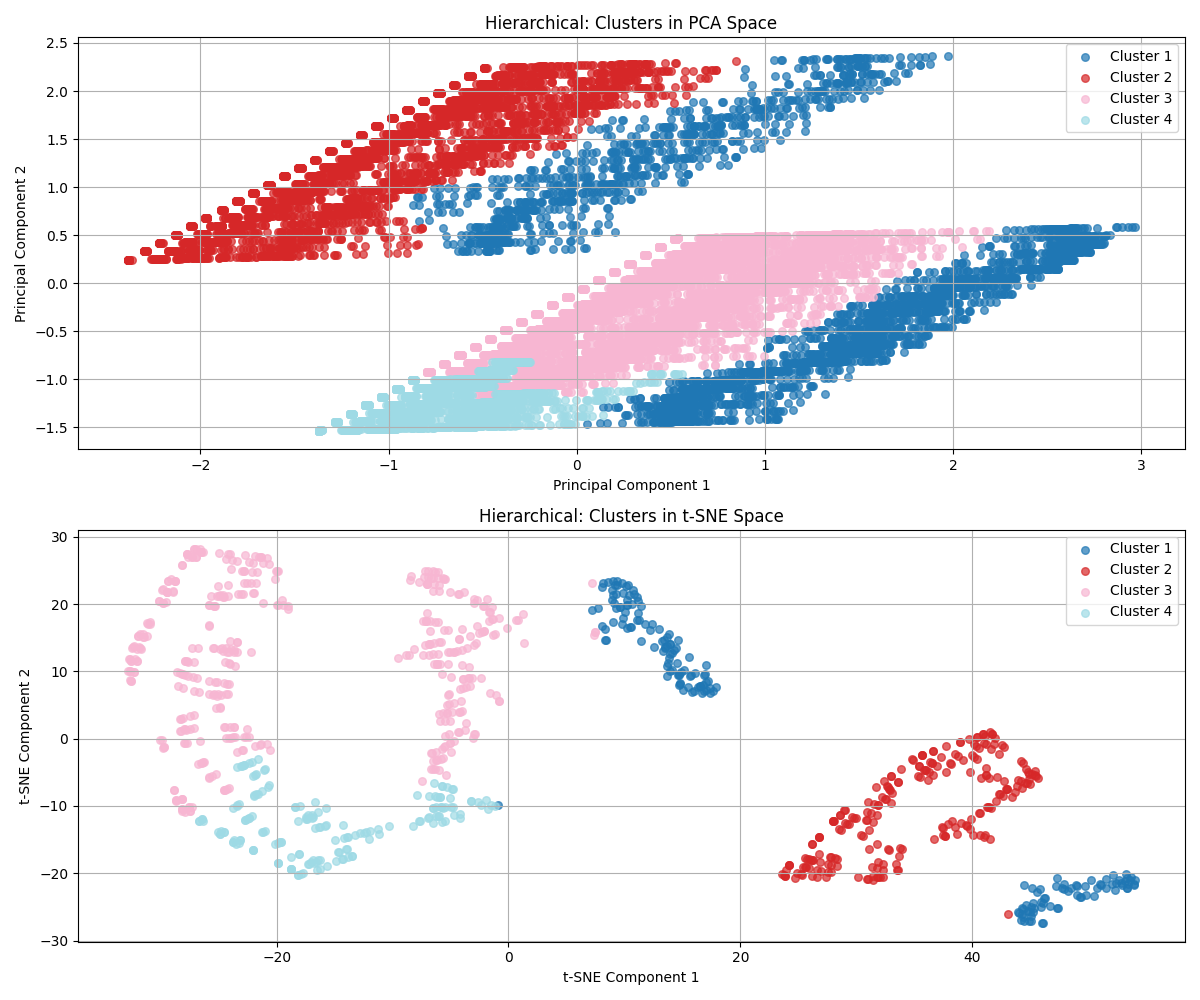


**Hierarchical Clustering**

Using **Agglomerative Clustering**:

* Formed **4 well-separated clusters**
* Silhouette score: **0.3696**
* Enabled deeper interpretation of **hourly behavior**, identifying specific night-time demand clusters (e.g., Cluster 4).

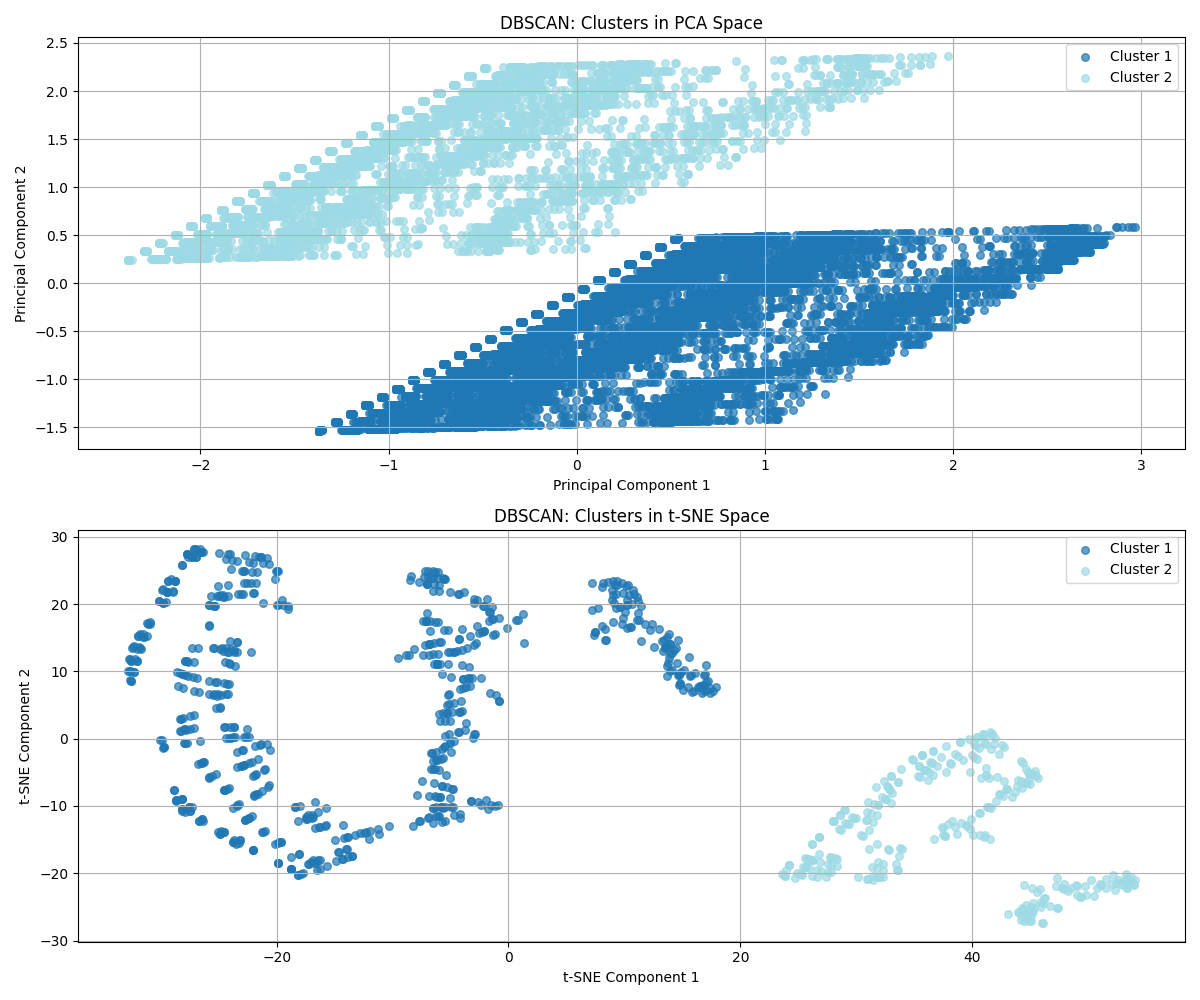


* 

**DBSCAN Clustering**

To identify any density-based patterns missed by K-Means:

* **Eps:** tuned around an automatic estimate of **0.345**
* **Best params:** eps=0.276, min\_samples=6
* **Silhouette Score:** **0.4021**
* Resulted in **2 clusters** with **0% noise**, indicating clean separability without anomalies.
* While simpler than K-Means, it showed robust separation between **weekday vs weekend** demand trends.



**Cluster Interpretations**

Each method offered unique strengths. K-Means provided the **most detailed segmentation**, with 6 distinct behavioral groups:

| **Cluster** | **Key Traits** |
| --- | --- |
| Cluster 1 | Low demand, warm conditions, weekdays |
| Cluster 2 | Peak demand, moderate temps, weekdays |
| Cluster 3 | Low demand, moderate temps, weekends |
| Cluster 4 | Low demand, warm evenings, weekends |
| Cluster 5 | Low demand, moderate temps, weekdays |
| Cluster 6 | Peak demand, moderate temps, weekends |

DBSCAN reinforced a **weekday vs weekend** pattern, while hierarchical clustering exposed temporal characteristics like **late-night demand trends**.

**Evaluation Summary**

| **Method** | **Silhouette Score** |
| --- | --- |
| K-Means (K=6) | **0.4750** |
| DBSCAN | 0.4021 |
| Hierarchical | 0.3696 |

**📊 Forecasting Model Evaluation Report**

**1. Dataset Overview**

* **After Feature Engineering**: 165,025 rows, 39 features
* **Training Set**: 150,625 records (from **2018-07-02 09:00:00** to **2020-03-20 23:00:00**)
* **Testing Set**: 14,400 records (from **2020-03-21 00:00:00** to **2020-05-19 23:00:00**)

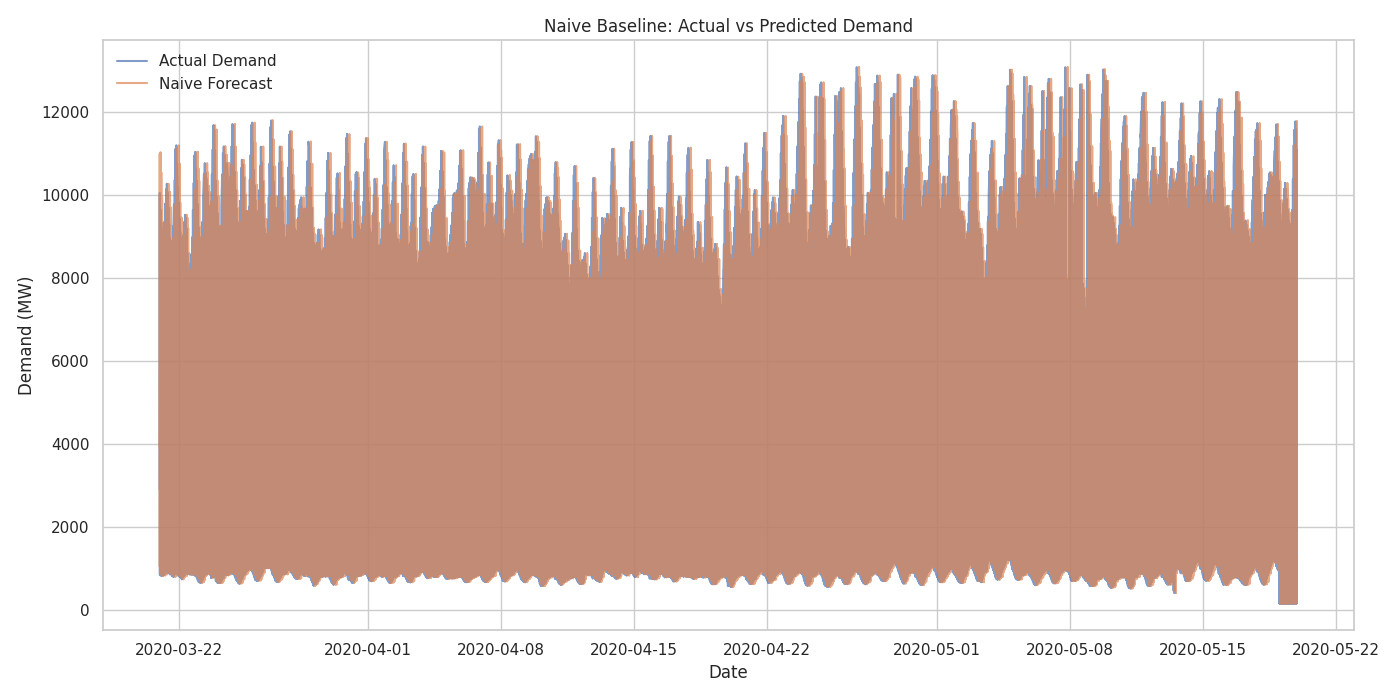
**2. Baseline Model: Naive Forecast**

A simple model that uses the previous value as the prediction.

**Performance Metrics**:

* **MAE**: 3409.83 MW
* **RMSE**: 4507.70 MW
* **MAPE**: 156.45%

This serves as the benchmark for assessing improvements of all other models.



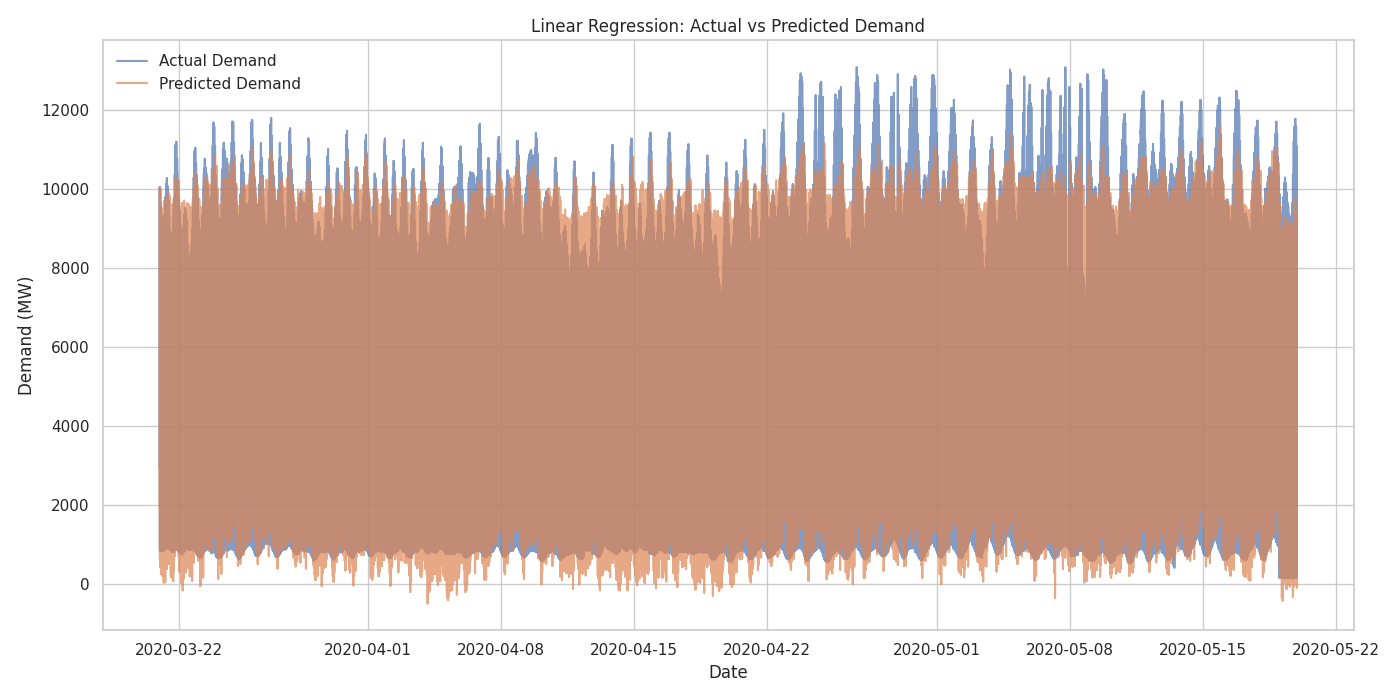
**3. Linear Regression Model**

A linear approach to modeling the relationship between features and target variable.

**Performance**:

* **MAE**: 859.96 MW
* **RMSE**: 1197.43 MW
* **MAPE**: 34.66%
* **R²**: 0.8619
* **Improvement over baseline**: 74.78%

*Simple yet effective, with solid interpretability but limited capacity to model non-linear patterns.*



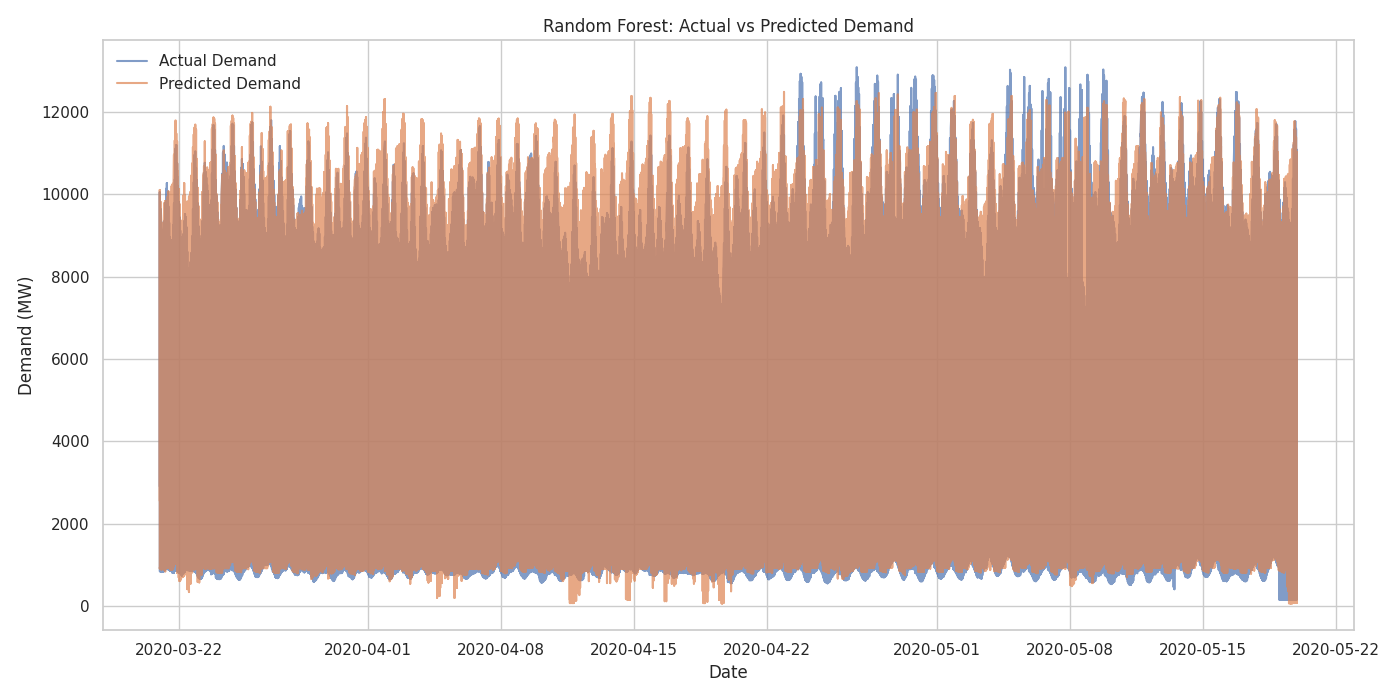
**4. Random Forest**

An ensemble of decision trees using bagging to improve generalization.

**Performance**:

* **MAE**: 556.96 MW
* **RMSE**: 977.42 MW
* **MAPE**: 20.74%
* **R²**: 0.9080
* **Improvement over baseline**: 83.67%

*Good performer based on MAE. Strong at handling non-linear relationships and robust to overfitting.*



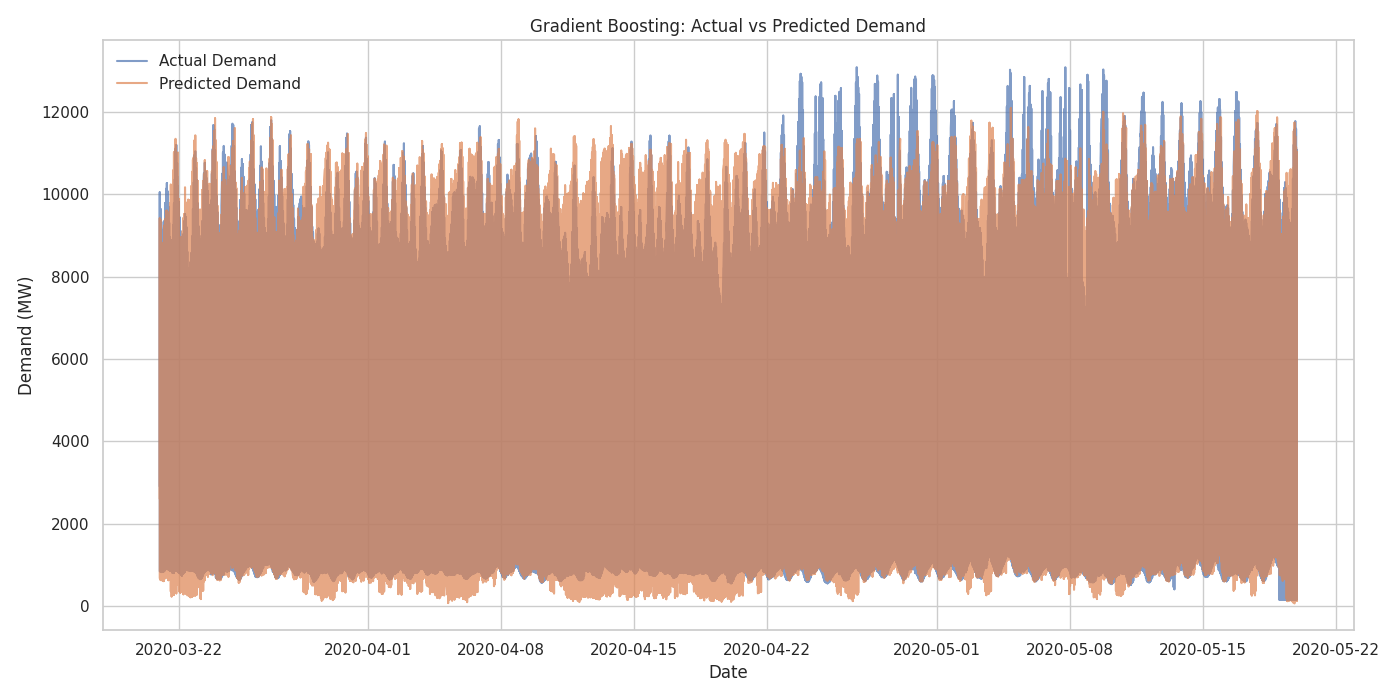
**5. Gradient Boosting**

Sequential ensemble method improving residuals of previous trees.

**Performance**:

* **MAE**: 694.97 MW
* **RMSE**: 1106.55 MW
* **MAPE**: 24.93%
* **R²**: 0.8821
* **Improvement over baseline**: 79.62%

*Balances bias and variance but slightly underperforms compared to Random Forest.*



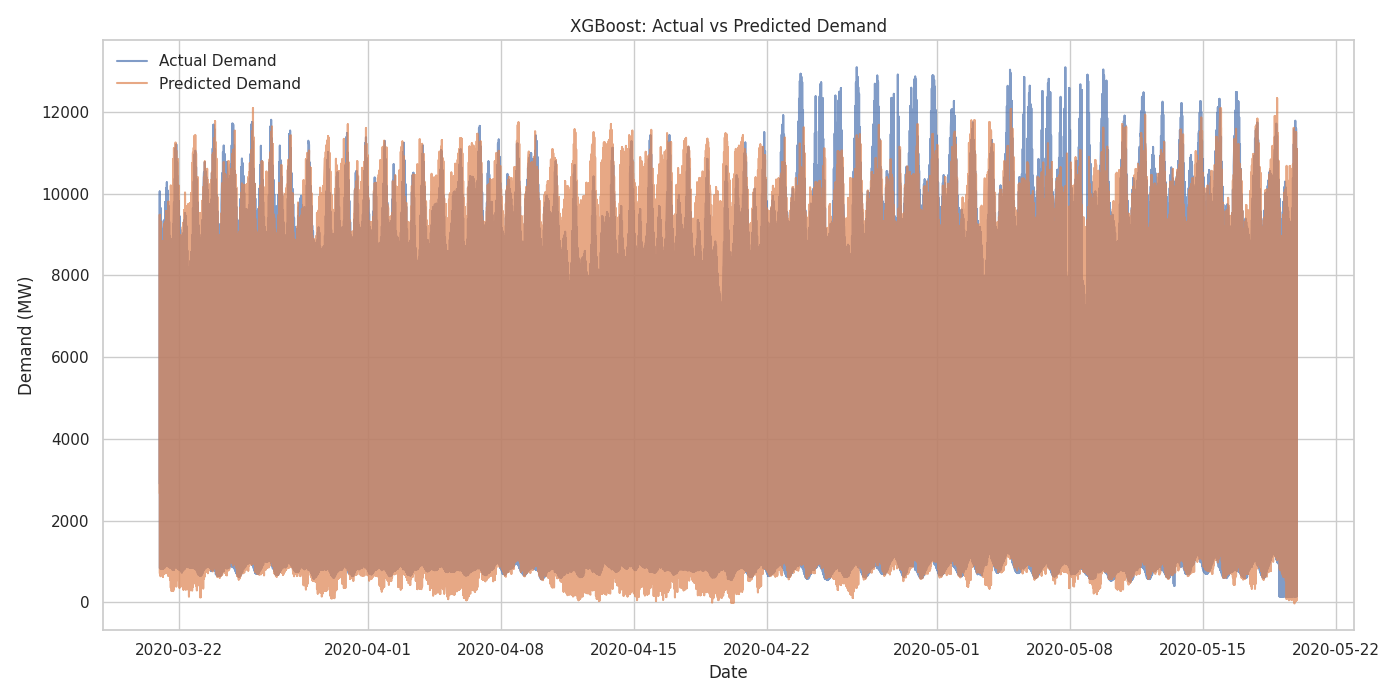
**6. XGBoost**

Optimized gradient boosting algorithm with regularization.

**Performance**:

* **MAE**: 738.89 MW
* **RMSE**: 1202.43 MW
* **MAPE**: 25.92%
* **R²**: 0.8607
* **Improvement over baseline**: 78.33%

*Offers regularization and speed, though slightly worse than Random Forest in this case.*

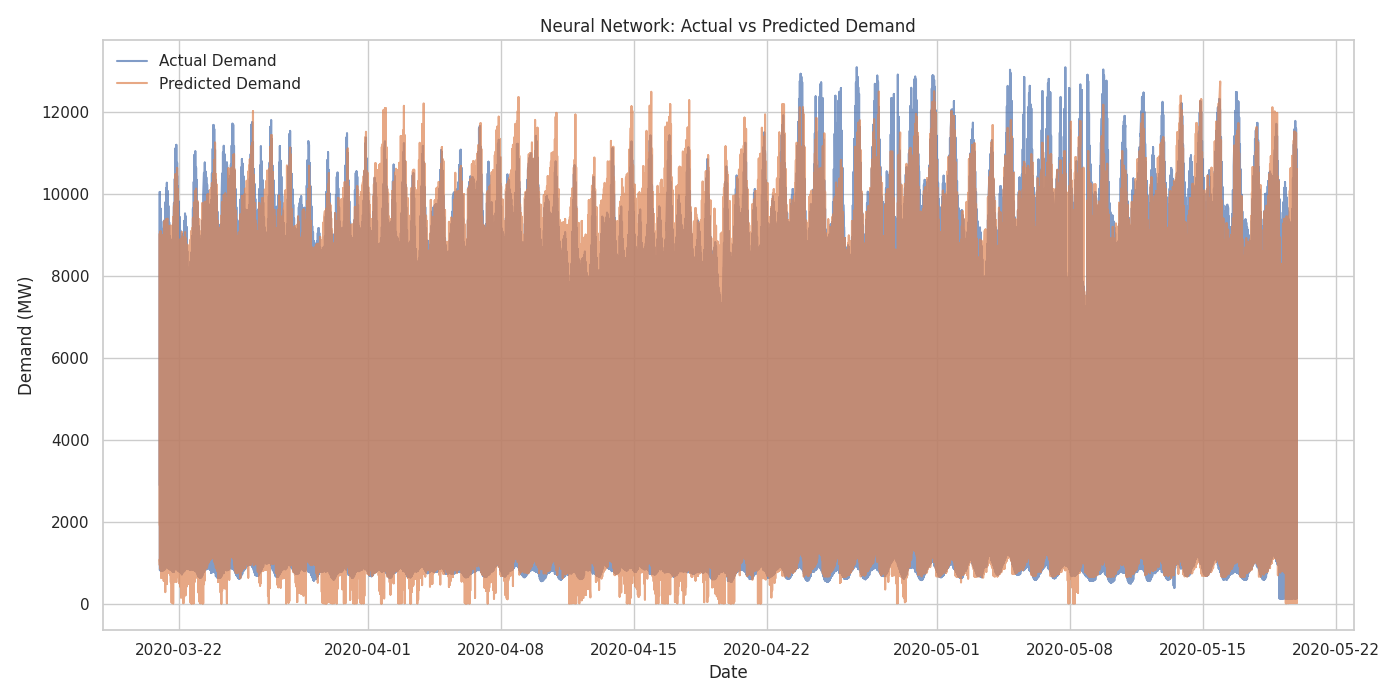


**7. Feedforward Neural Network**

Deep learning model trained over 450 epochs.

**Performance**:

* **MAE**: 698.10 MW
* **RMSE**: 1063.18 MW
* **MAPE**: 24.95%
* **R²**: 0.8911
* **Improvement over baseline**: 79.53%

💡 *Good performance, but with higher training complexity and tuning sensitivity.* 

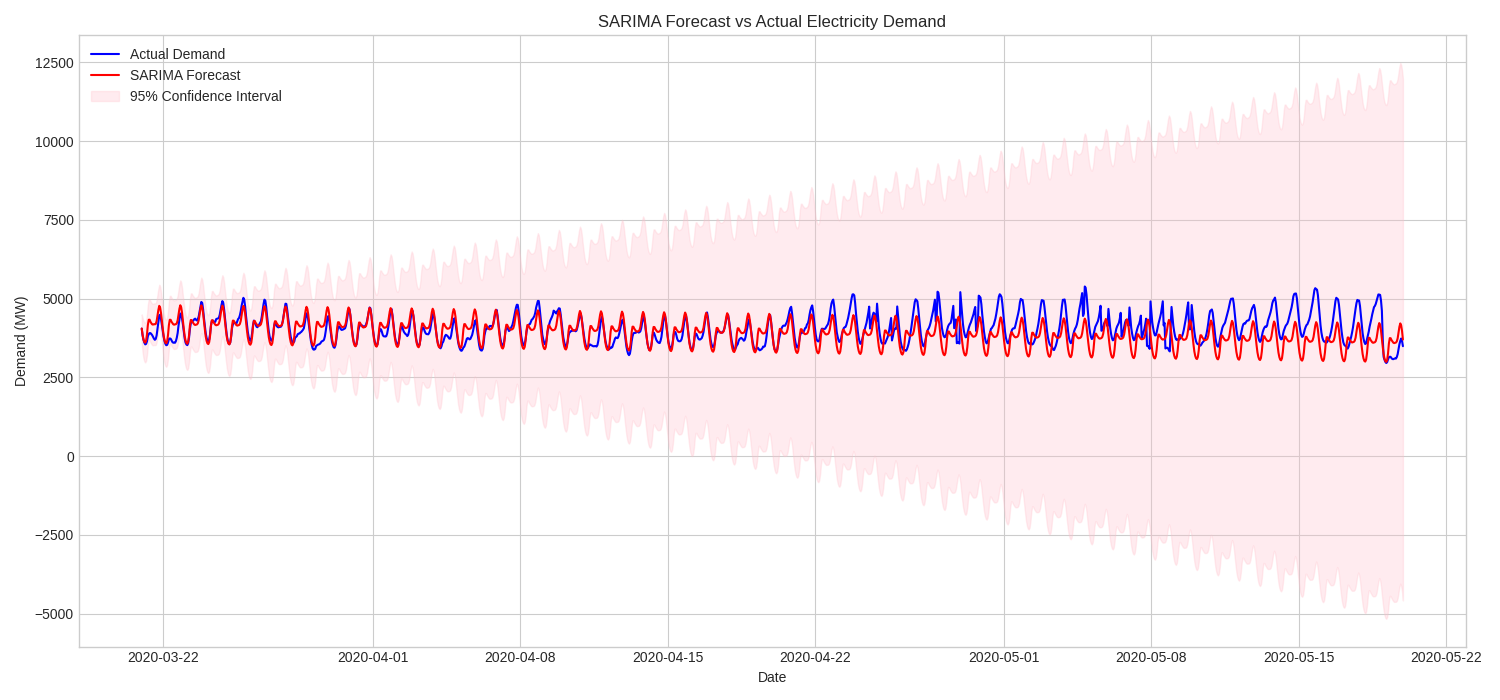
**8. SARIMA Model**

A statistical time series model incorporating seasonality, trend, and autocorrelation.

**Performance**:

* **MAE**: 330.70 MW
* **RMSE**: 431.95 MW
* **MAPE**: 11.83%
* **Improvement over baseline**: 90.23%

*Top performer in accuracy metrics. Best for modeling stable seasonal patterns.*



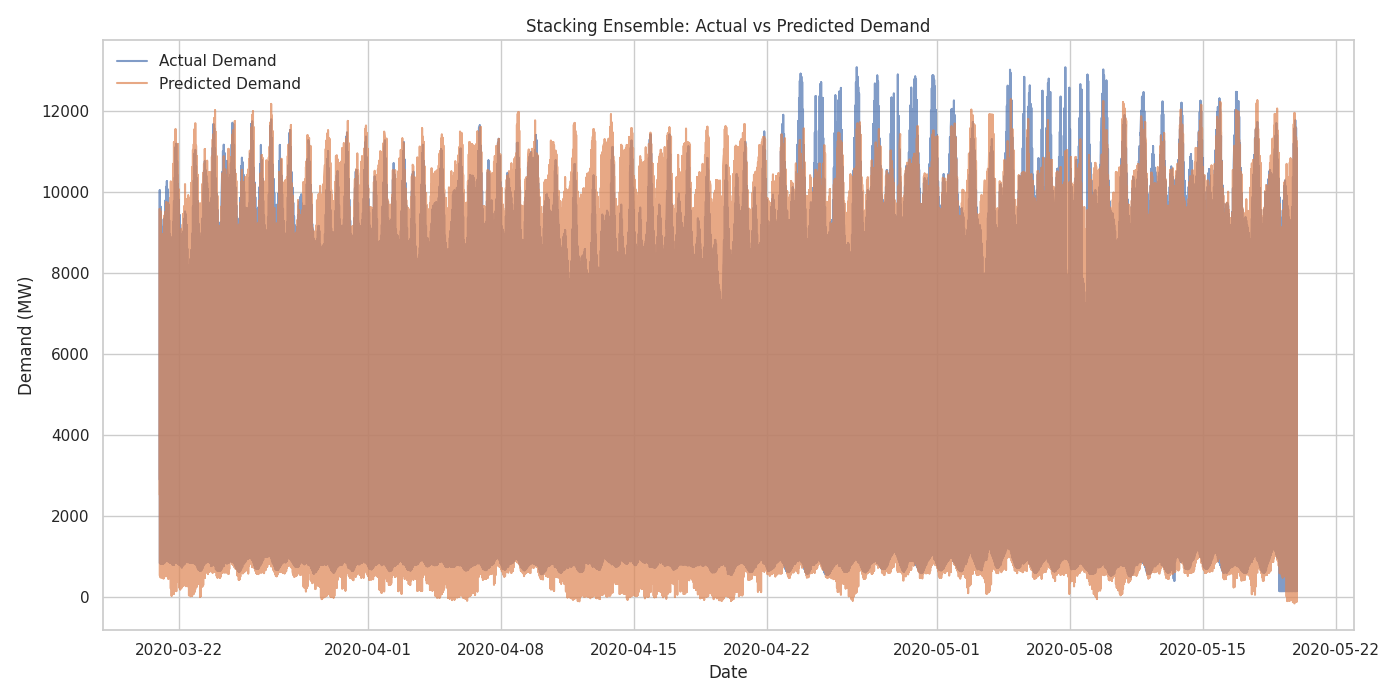
**🔗 9. Stacking Ensemble**

Combines predictions from multiple models for improved performance.

**Performance**:

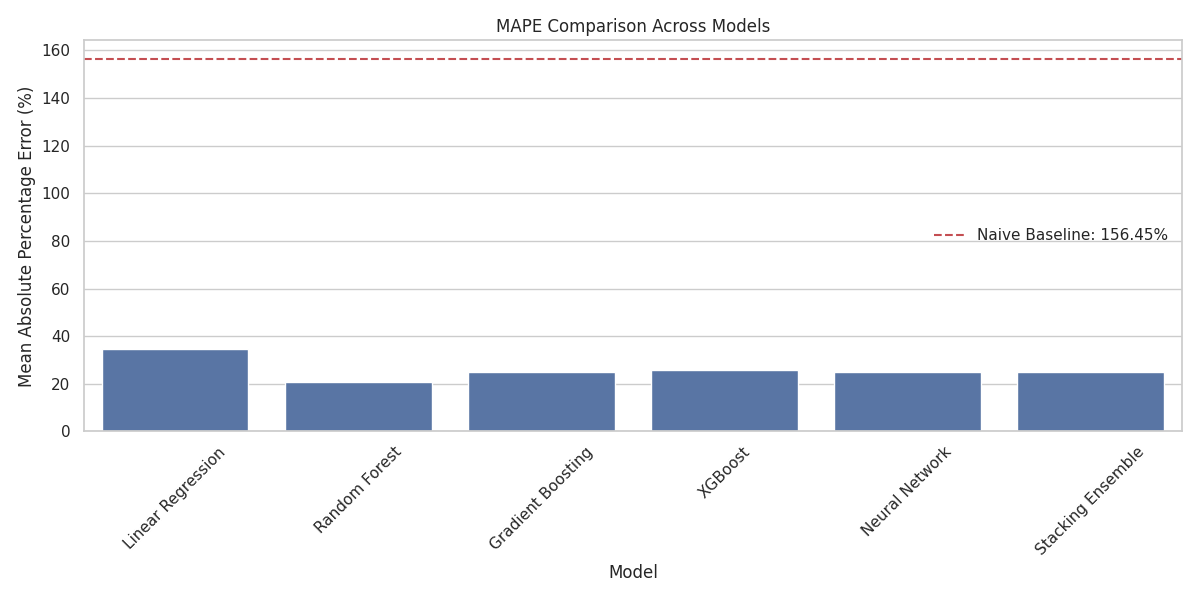
* **MAE**: 694.68 MW
* **RMSE**: 1146.72 MW
* **MAPE**: 24.71%
* **R²**: 0.8734
* **Improvement over baseline**: 79.63%

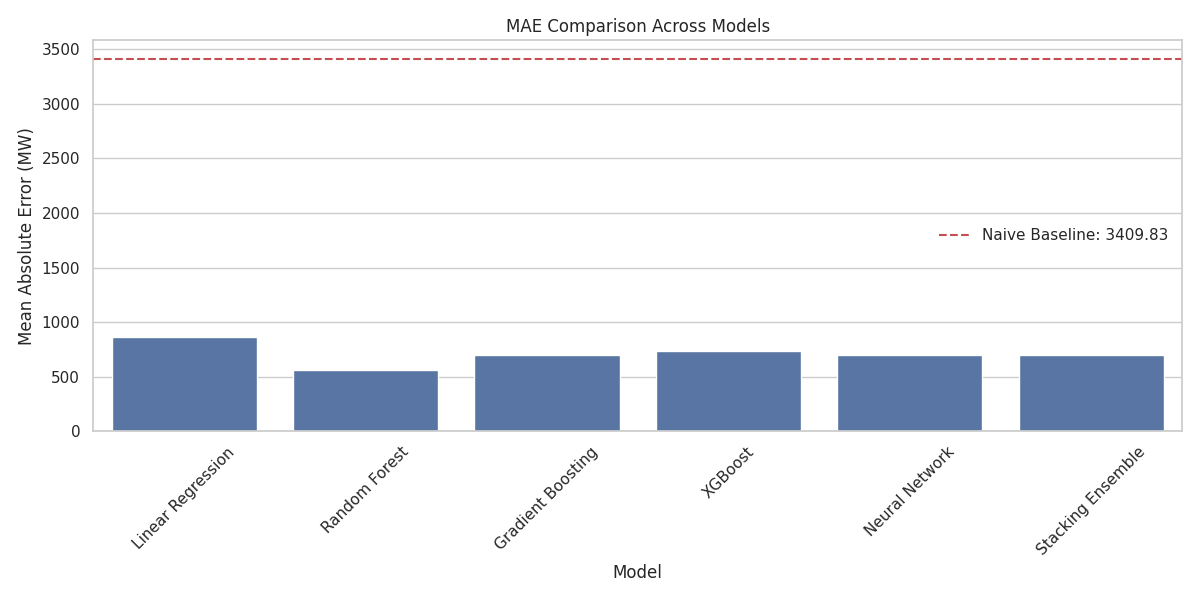
*Mixed results; ensemble didn’t outperform the best individual model (Random Forest or SARIMA).*

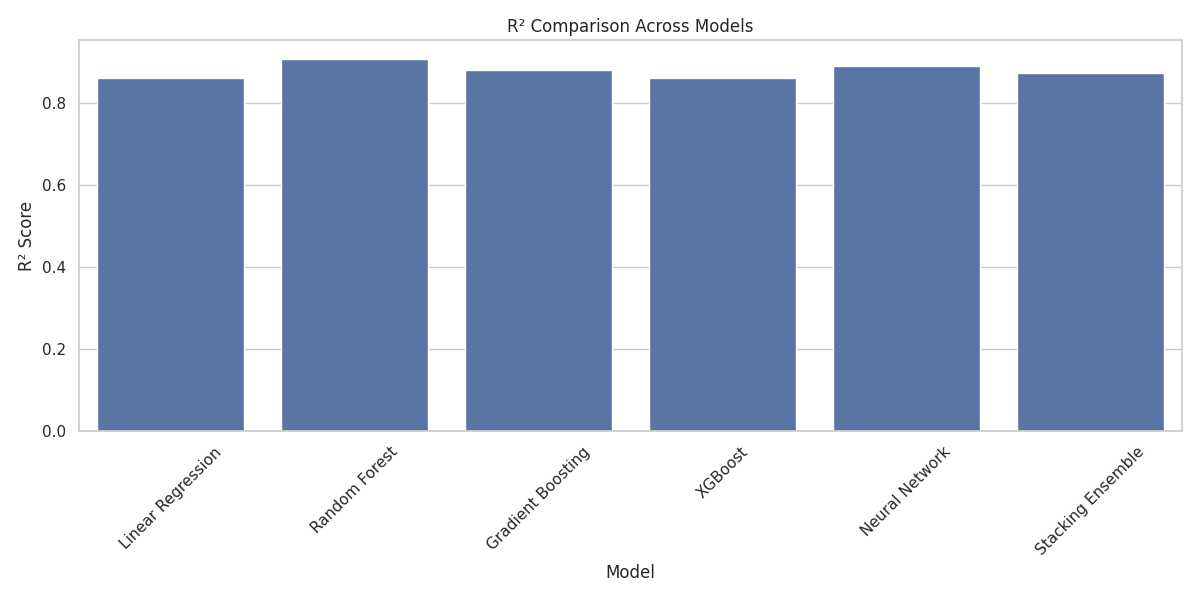


**📊 10. Summary Table**

| **Model Name** | **MAE (MW)** | **RMSE (MW)** | **MAPE (%)** | **R²** |
| --- | --- | --- | --- | --- |
| **Linear Regression** | 859.96 | 1197.43 | 34.66 | 0.8619 |
| **Random Forest** | 556.96 | 977.42 | 20.74 | 0.9080 |
| **Gradient Boosting** | 694.97 | 1106.55 | 24.93 | 0.8821 |
| **XGBoost** | 738.89 | 1202.43 | 25.92 | 0.8607 |
| **Neural Network** | 698.10 | 1063.18 | 24.95 | 0.8911 |
| **Stacking Ensemble** | 694.68 | 1146.72 | 24.71 | 0.8734 |
| **SARIMA** | **330.70** | **431.95** | **11.83** | 0.9512 |
|  |  |  |  |  |







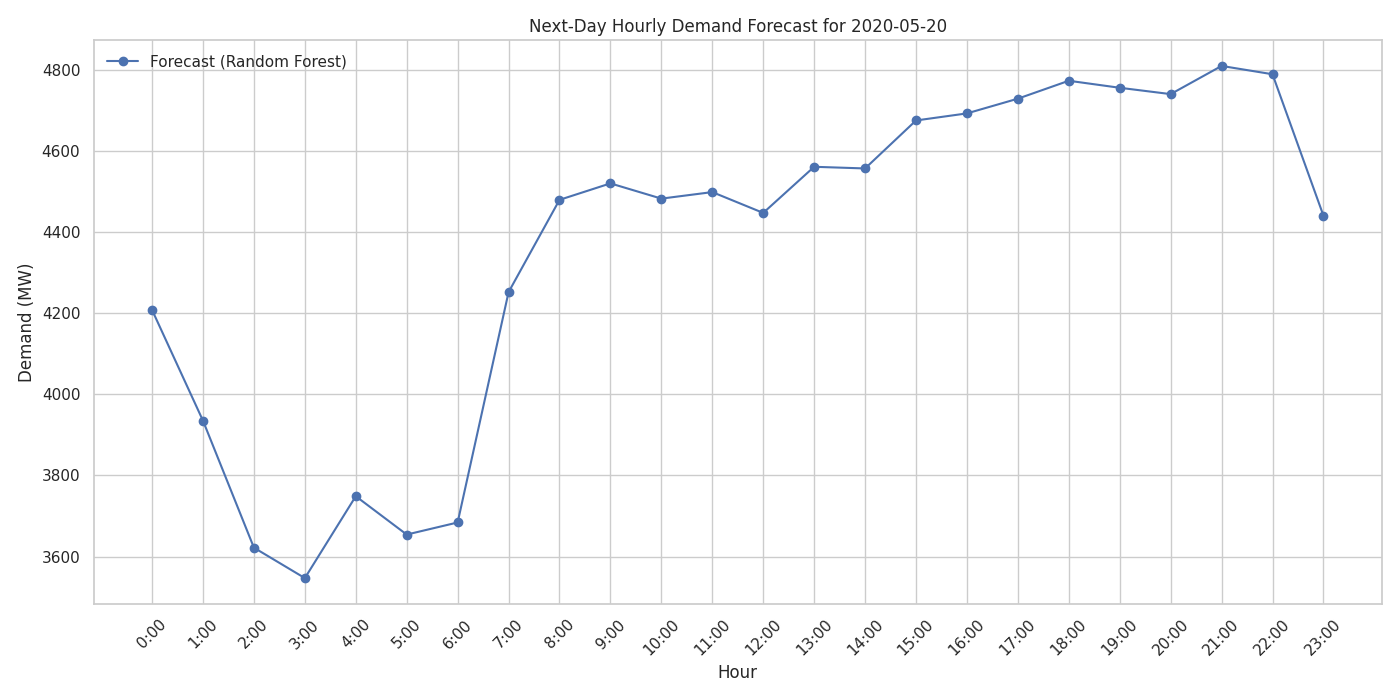
**11. Best Models by Metric**

* **Lowest MAE**: SARIMA (330.70 MW)
* **Lowest RMSE**: SARIMA (431.95 MW)
* **Lowest MAPE**: SARIMA (11.83%)
* **Best R²**: SARIMA (0.9512)

**12. Error Analysis**

* **Linear models** tend to underperform in capturing non-linear fluctuations.
* **Ensemble methods** improve robustness but may not always outperform the best single model.
* **Neural networks** require careful hyperparameter tuning and may overfit if not regularized.
* **SARIMA** handles seasonality extremely well, explaining its top performance.

**13. Next-Day Forecast Simulation**

**Forecast Date**: May 20, 2020  


**15. Conclusion**

* **SARIMA** is recommended for short-term demand forecasting with strong seasonal structure.
* **Random Forest** is the best machine learning approach in terms of generalization and robustness.

**Electric Load Forecasting Dashboard**

**Technical Report**

**Project Overview**

The Electric Load Forecasting Dashboard is an interactive web application designed to support the analysis and visualization of electricity demand data. It enables users to forecast electric load for specific cities over selected time ranges and perform clustering analysis to identify demand patterns based on historical data. The system is built using modern web development technologies and provides clear, informative visualizations to facilitate decision-making in energy consumption analysis.

**Objective**

The main objectives of this project are:

* To provide a forecasting tool that predicts electricity demand over a given date range using historical data.
* To visualize clustering patterns in electric load data using Principal Component Analysis (PCA).
* To enable users to easily interact with energy demand data through an intuitive and responsive user interface.

**Technologies Used**

* **Frontend Framework**: React.js
* **Visualization Library**: Recharts
* **API Communication**: Backend in Python/Flask
* **Styling**: Custom CSS
* **Data Format**: JSON

**Core Functionalities**

**1. City Selection & Date Range Initialization**

Upon launching the application, it automatically fetches the list of available cities and the valid date range for analysis from the backend. This information initializes the form inputs and helps users select valid parameters for forecasting or clustering tasks.

**2. Electric Load Forecasting**

Users can select a city and define a start and end date within the permissible range. Once submitted, the system contacts the backend to retrieve a time series of both actual and predicted electric load values. These results are plotted on a responsive line chart, allowing users to easily compare the model’s performance.

**Key Features:**

* Actual vs. Predicted comparison
* Dynamic date filtering
* Scalable and responsive chart design

**3. Clustering Analysis**

In clustering mode, users choose a city and adjust the number of clusters (k) using an interactive slider. The application sends this information to the backend, which processes the clustering (likely via K-Means) and returns PCA-transformed results.

**Visualization Highlights:**

* 2D scatter plot showing cluster groupings
* Point size represents demand magnitude
* Tooltip displays detailed information (date, demand, temperature, cluster)
* Color-coded clusters

**4. Interactive Control Panel**

The control panel adapts based on the selected tab (forecasting or clustering), displaying relevant form elements. This includes:

* City dropdown
* Date pickers (forecasting)
* Cluster count slider (clustering)
* Submission button with loading state

**5. Help & Documentation Section**

Each mode features a collapsible help section, providing step-by-step instructions on how to use the functionality. This serves as both user guidance and a brief user manual.

**User Experience and Design**

The application features a clean, intuitive layout with the following design principles:

* **Tab-based navigation**: Easy switching between forecasting and clustering views.
* **Responsive charts**: Auto-scaling to screen size, suitable for both desktop and tablet views.
* **Interactive tooltips**: Rich, contextual tooltips provide deeper insights into each data point.
* **Error handling**: Informative error messages help diagnose issues like server unavailability or invalid inputs.

**Error Handling and Loading States**

To improve robustness and user experience, the application:

* Displays loading indicators during API calls.
* Shows descriptive error messages if data fetching fails or the backend returns an error.
* Prevents form submission during loading to avoid duplicate requests.

**Strengths**

* **User-friendly interface**: Designed for users without technical backgrounds.
* **Data-rich visualizations**: Presents complex information in an accessible manner.
* **Separation of concerns**: Logical component and state separation for easier maintenance.
* **Scalable functionality**: Modular architecture allows for future additions (e.g., anomaly detection, multi-city comparisons).

**Conclusion**

The Electric Load Forecasting Dashboard successfully combines data science capabilities with an accessible user interface. It serves as a valuable tool for energy analysts, city planners, or researchers by providing insights into electricity demand patterns and enabling data-driven forecasting. With strong architectural design and thoughtful user interaction, it forms a solid foundation for more advanced smart grid or energy analytics platforms.