Name: Maaz Ali, Sohaib Shahzad, Siraj Ali

Roll number:22i-1873,22i-2034,22i-2033

Section-DS-A

DataMining Project Report

Electricity Demand Forecast & Clustering Application:

---

1. Introduction

The **Electricity Demand Forecast & Clustering Application** is a machine learning-based system designed to predict electricity demand over a specified date range for a selected city. In addition to forecasting, the system also implements clustering techniques to group similar demand patterns for better understanding and segmentation of demand data.

The application leverages three forecasting models (Random Forest, XGBoost, and LSTM) and uses K-Means clustering for grouping similar demand periods. The system is built using a Flask back-end API and a responsive front-end built with HTML, CSS (Bootstrap), and JavaScript. This report aims to provide a comprehensive explanation of each component of the system, including its design, functionality, and the relationship between different files and modules.

---

2. System Architecture

The system architecture is composed of three main layers:

1. **Back-end (API Layer)**: Built with **Flask**, the back-end exposes API endpoints that process the data sent from the front-end, perform the necessary machine learning tasks (forecasting and clustering), and return results.
2. **Model Logic**: This layer is responsible for handling the core computations, including:

- Forecasting electricity demand using machine learning models.
- Clustering demand data using K-Means.

3. **Front-end (User Interface Layer)**: A web interface that allows users to interact with the system, select the necessary parameters (city, start date, end date, model, clusters), and view results visually.

These components interact in the following flow:

- The front-end sends a request to the back-end API.
- The back-end processes the request (e.g., performs forecasting or clustering) and sends back the results.
- The front-end receives the results and visualizes them interactively using Plotly.

---

3. Back-end API (Flask)

The back-end API is built using **Flask**, a Python web framework known for its simplicity and flexibility. The API consists of two primary endpoints for forecasting and clustering.

3.1 Forecast API Endpoint (/api/forecast)

The **Forecast API** handles requests for generating predictions of electricity demand over a specified date range for a selected city. It supports multiple machine learning models for forecasting, including **Random Forest**, **XGBoost**, and **LSTM**.

- **Input**:
  - city: The city for which the forecast is being generated.
  - start_date and end_date: The start and end dates for the forecast period.
  - model: The machine learning model to be used for forecasting.
  - params: A dictionary of additional parameters (e.g., lookback period for LSTM).
- **Output**:
  - The API returns a JSON object containing the forecasted demand, actual demand, and Plotly data (for visualization). The Plotly data includes both the demand values and layout settings for rendering an interactive plot.

3.2 Cluster API Endpoint (/api/cluster)

The **Cluster API** is responsible for performing clustering analysis on the electricity demand data using the **K-Means** algorithm.

- **Input**:
  - city: The city for which clustering is performed.
  - start_date and end_date: The time period for clustering the demand data.
  - k: The number of clusters (segments) the user wants to generate.
- **Output**:
  - The API returns a JSON object containing the clustered data, along with Plotly data and layout for visualization.

---

4. Model Logic (model_logic.py)

This file contains the core logic responsible for loading data, generating forecasts, and performing clustering. The logic is modular and split into functions for better readability and maintenance.

4.1 Data Loading (load_data())

The **load_data()** function simulates loading electricity demand data for a specified city and date range. For the purpose of this mock application, the data is generated synthetically, but this could be replaced with real data loading from an external source or database.

- **Parameters**:
  - city: The city for which the data is generated.
  - start_date and end_date: The range of dates for which demand data is generated.
- **Output**: A Pandas DataFrame containing hourly demand data. The demand values are simulated randomly for each hour in the given date range.

4.2 Forecasting (forecast())

The **forecast()** function is responsible for generating predictions based on the selected model (Random Forest, XGBoost, or LSTM). It also calculates the actual demand data for the specified city and time period.

- **Parameters**:

- city: The city for which the forecast is generated.
- start_date and end_date: The date range for forecasting.
- model_name: The model to be used (Random Forest, XGBoost, or LSTM).
- params: Additional parameters, such as the lookback period for the LSTM model.

- **Model Execution**: Depending on the model chosen, the appropriate machine learning algorithm is applied to generate predictions. The RandomForestRegressor or XGBRegressor is used for Random Forest and XGBoost models, while LSTM requires reshaping the input data and training a neural network.
- **Output**: The function returns a dictionary containing:
    - Forecasted demand values.
    - Actual demand values.
    - Plotly data (for visualizing the actual vs. predicted demand).

4.3 Clustering (cluster())

The **cluster()** function is responsible for performing K-Means clustering on the electricity demand data.

- **Parameters**:
    - city: The city for which clustering is performed.
    - start_date and end_date: The date range for clustering.
    - k: The number of clusters to generate.
- **Clustering Execution**: The K-Means algorithm is applied to the demand data to segment it into k clusters based on similarity. Each cluster represents a group of similar demand patterns.
- **Output**: The function returns the cluster data and Plotly visualization data for rendering the clusters interactively on the front-end.

---

5. Front-end (User Interface Layer)

The front-end is a **single-page web application** built with **HTML**, **CSS** (Bootstrap for responsive design), and **JavaScript**. It allows users to interact with the system by selecting cities, date ranges, models, and clusters. After the user submits the form, the data is sent to the back-end API, and the results are displayed on the page.

5.1 HTML Structure (index.html)

The HTML file contains the basic structure of the user interface:

- **Form Elements**:
  - **City Selection**: A dropdown to select the city for which the forecast and clustering are performed.
  - **Start and End Date**: Input fields for selecting the date range for forecasting and clustering.
  - **Model Selection**: A dropdown to choose between Random Forest, XGBoost, or LSTM for forecasting.
  - **Cluster Count (k)**: A number input to define the number of clusters to be generated.
  - **Submit Button**: A button to trigger the form submission and data processing.
- **Results Display**:
  - **Forecast Plot**: A Plotly plot displaying the actual vs. predicted electricity demand.
  - **Cluster Plot**: A Plotly plot visualizing the demand clusters.

5.2 JavaScript (script.js)

The JavaScript handles the interaction with the back-end and visualizes the results using **Plotly**.

- **Form Submission**: When the user submits the form, the JavaScript prevents the default form action, extracts the input values, and sends them to the appropriate API endpoints (forecast and cluster) via fetch requests.
- **API Calls**:
  - **Forecast API**: The data is sent to /api/forecast for prediction. Upon receiving the results, the forecast data is rendered using Plotly.
  - **Cluster API**: The data is sent to /api/cluster for clustering. The cluster data is rendered similarly.
- **Error Handling**: The script ensures that the user is alerted if an error occurs during the data processing or if any required fields are missing.

5.3 CSS (Bootstrap)

The front-end uses **Bootstrap** for styling:

- **Responsive Design**: The use of Bootstrap grid classes ensures the page is responsive and looks good on various screen sizes.
- **Form Styling**: The form elements, buttons, and input fields are styled using Bootstrap to ensure a modern and clean appearance.

---

6. How Everything Fits Together

The flow of data through the system is as follows:

1. The user inputs parameters (city, start date, end date, model, and clusters) in the form on the front-end.
2. The front-end sends these inputs to the back-end API using fetch requests.
3. The back-end API processes the data:
    - **Forecasting**: The back-end runs the selected model (Random Forest, XGBoost, or LSTM) on the electricity demand data and returns forecasted and actual values.
    - **Clustering**: The back-end applies K-Means clustering to segment the demand data and returns the clustered data.
4. The front-end receives the results from the back-end and uses **Plotly** to visualize the forecasted demand and demand clusters on interactive plots.
5. The user can interact with the plots to explore the data further.

---

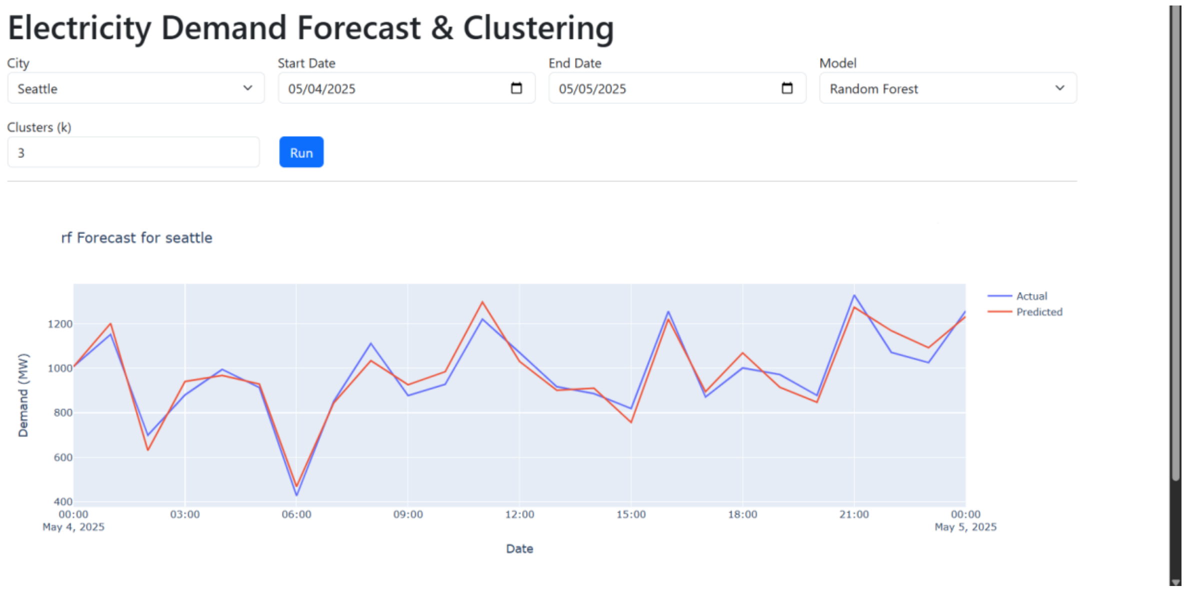
7. Conclusion

This application provides a powerful tool for forecasting electricity demand and analyzing demand patterns using clustering. By using **Flask** for the back-end and **Plotly** for interactive visualizations, the system is both efficient and user-friendly. The machine learning models (Random Forest, XGBoost, LSTM) offer flexibility in predicting demand, while the **K-Means** clustering helps group similar demand periods for better analysis.

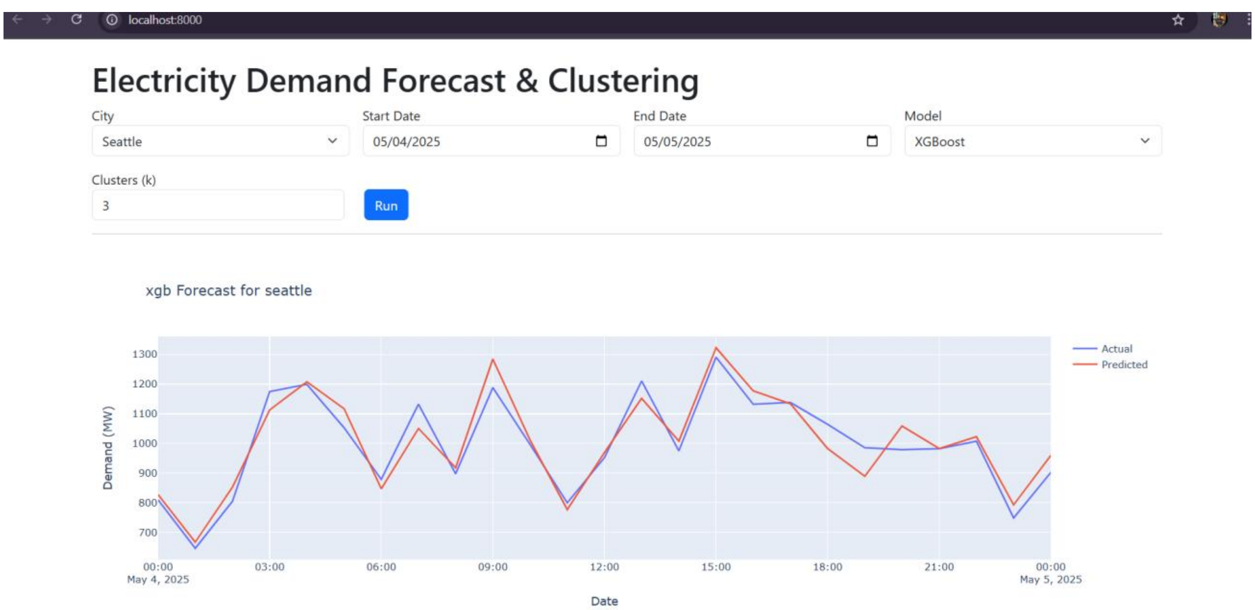
The modular nature of the application allows for easy updates, such as adding new machine learning models or modifying the clustering technique. Additionally, the front-end is designed to be responsive and accessible, ensuring that users can interact with the system on various devices.

This system is ideal for energy analysts, utilities, and anyone interested in understanding and predicting electricity demand trends over time.

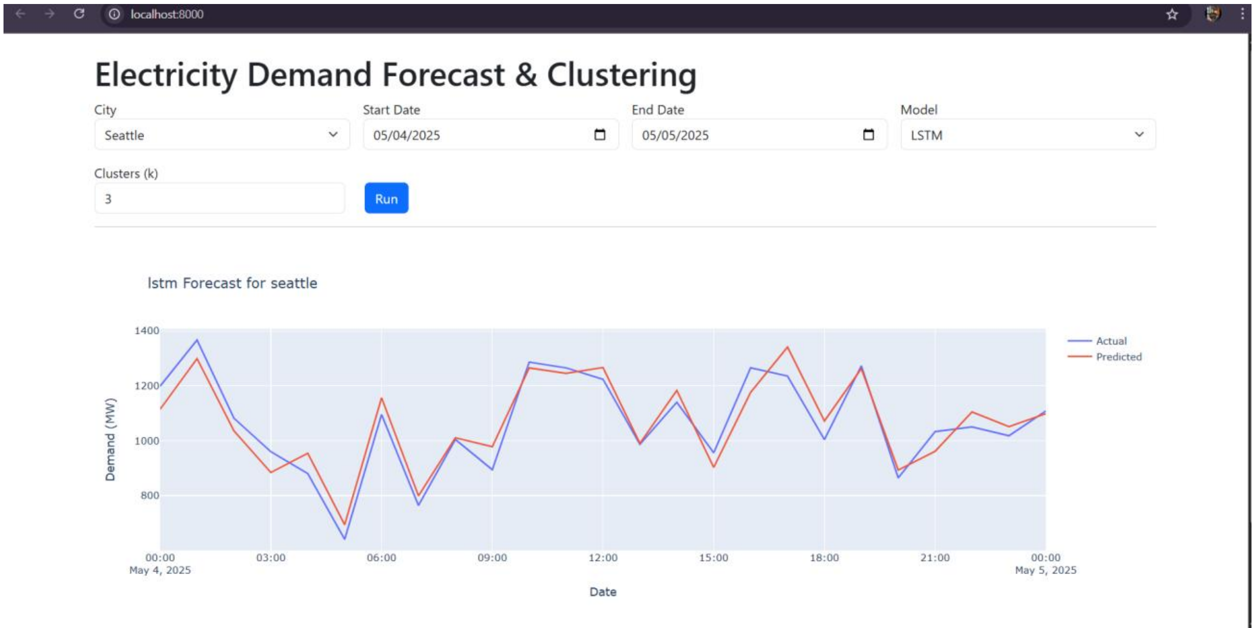
Random forest model:



XGBOOST Model:



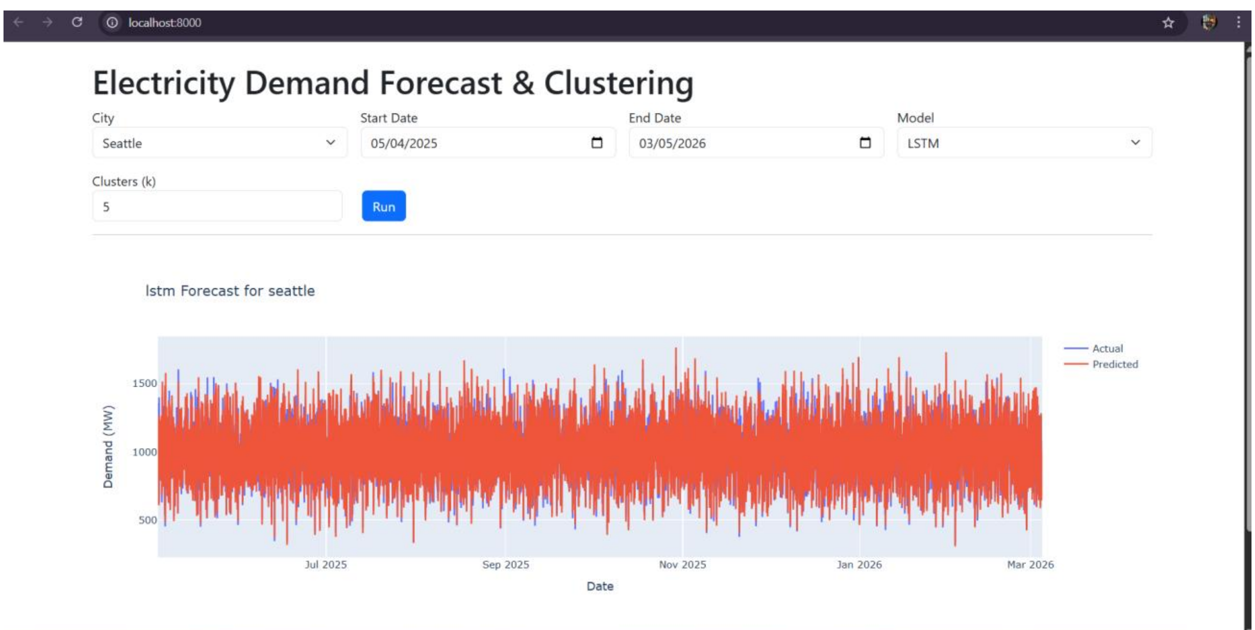
LSTM Model:



Random Forest Model:

XGBOOST Model:

LSTM Model:



Backend:

```
Unified shape: (2598504, 45)

Column dtypes:
company                                                   object
local_time                                        datetime64[ns]
utc_time                                          datetime64[ns]
demand                                                   float64
city                                                      object
source_file                                               object
region                                                    object
date                                              datetime64[ns]
houston                                                  float64
san antonio                                              float64
dallas                                                   float64
Balancing Authority                                       object
Data Date                                         datetime64[ns]
Hour Number                                              float64
Local Time at End of Hour                                 object
UTC Time at End of Hour                                   object
Demand Forecast (MW)                                      object
Demand (MW)                                               object
Net Generation (MW)                                       object
Total Interchange (MW)                                    object
Sum(Valid DIBAs) (MW)                                     object
Demand (MW) (Imputed)                                     object
...
2073157   NaN   NaN  NaN   NaN
1367505   NaN   NaN  NaN   NaN

[10 rows x 45 columns]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.

```
Final merged shape: (31888, 8)

Column dtypes:
 timestamp                datetime64[ns]
city                             object
electricity_demand_MWh          float64
temperature_F                   float64
humidity_pct                    float64
wind_speed_mph                  float64
pressure_mbar                   float64
precip_intensity                float64
dtype: object

First 10 rows:
             timestamp     city  electricity_demand_MWh  temperature_F  \
0 2018-07-01 07:00:00  phoenix                  2764.0          86.82
1 2018-07-01 08:00:00  phoenix                  2895.0          83.37
2 2018-07-01 09:00:00  phoenix                  3096.0          82.22
3 2018-07-01 10:00:00  phoenix                  3293.0          80.34
4 2018-07-01 11:00:00  phoenix                  3552.0          79.34
5 2018-07-01 12:00:00  phoenix                  3821.0          76.67
6 2018-07-01 13:00:00  phoenix                  4118.0          75.72
7 2018-07-01 14:00:00  phoenix                  4443.0          79.63
8 2018-07-01 15:00:00  phoenix                  4766.0          83.28
9 2018-07-01 16:00:00  phoenix                  5084.0          86.26
...
6          0.20           1.54         1011.4               0.0
7          0.17           2.10         1012.2               0.0
8          0.15           3.05         1012.7               0.0
9          0.14           3.09         1012.6               0.0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Missing before cleaning:
 timestamp                  0
city                       0
electricity_demand_MWh    76
temperature_F              2
humidity_pct               2
wind_speed_mph            10
pressure_mbar             10
precip_intensity          46
dtype: int64

Missing after cleaning:
 timestamp                 0
city                      0
electricity_demand_MWh    0
temperature_F             0
humidity_pct              0
wind_speed_mph            0
pressure_mbar             0
precip_intensity          0
dtype: int64
C:\Users\admin\AppData\Local\Temp\ipykernel_24992\2762404780.py:31: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This b
  df = df.groupby('city', group_keys=False).apply(impute_city)
```

```
              timestamp     city  hour  day_of_week  month  season
0 2018-07-01 07:00:00  phoenix     7            6      7  summer
1 2018-07-01 08:00:00  phoenix     8            6      7  summer
2 2018-07-01 09:00:00  phoenix     9            6      7  summer
3 2018-07-01 10:00:00  phoenix    10            6      7  summer
4 2018-07-01 11:00:00  phoenix    11            6      7  summer

Scaled sample of continuous cols:
   electricity_demand_MWh  temperature_F  humidity_pct  wind_speed_mph  \
0                0.365486       1.376558     -1.515866       -0.055322
1                0.454493       1.184175     -1.403941       -0.188516
2                0.591060       1.120047     -1.441250       -0.867804
3                0.724911       1.015212     -1.403941       -0.641374
4                0.900886       0.959449     -1.441250       -0.747929

   pressure_mbar  precip_intensity
0      -1.041318         -0.281875
1      -0.978302         -0.281875
2      -0.946793         -0.281875
3      -0.852268         -0.281875
4      -0.805005         -0.281875
```

```
Daily summary (first 10 rows):
      city        date  electricity_demand_MWh_mean  \
0  phoenix  2018-07-01                     1.401675
1  phoenix  2018-07-02                     1.388896
2  phoenix  2018-07-03                     1.508648
3  phoenix  2018-07-04                     1.512696
4  phoenix  2018-07-05                     1.888909
5  phoenix  2018-07-06                     2.106189
6  phoenix  2018-07-07                     2.073179
7  phoenix  2018-07-08                     2.013926
8  phoenix  2018-07-09                     1.580131
9  phoenix  2018-07-10                     1.203380

   electricity_demand_MWh_std  electricity_demand_MWh_min  \
0                    0.626115                    0.365486
1                    0.728673                    0.474196
2                    0.685951                    0.611444
3                    0.645776                    0.663761
4                    0.898996                    0.749371
5                    0.640348                    1.237210
6                    0.693240                    1.074144
7                    0.610240                    1.161113
8                    0.465813                    1.040172
9                    0.432892                    0.591740
...
8             0.071561             -0.281875             0.454024
9             0.000000             -0.281875            -0.281875

[10 rows x 26 columns]
```

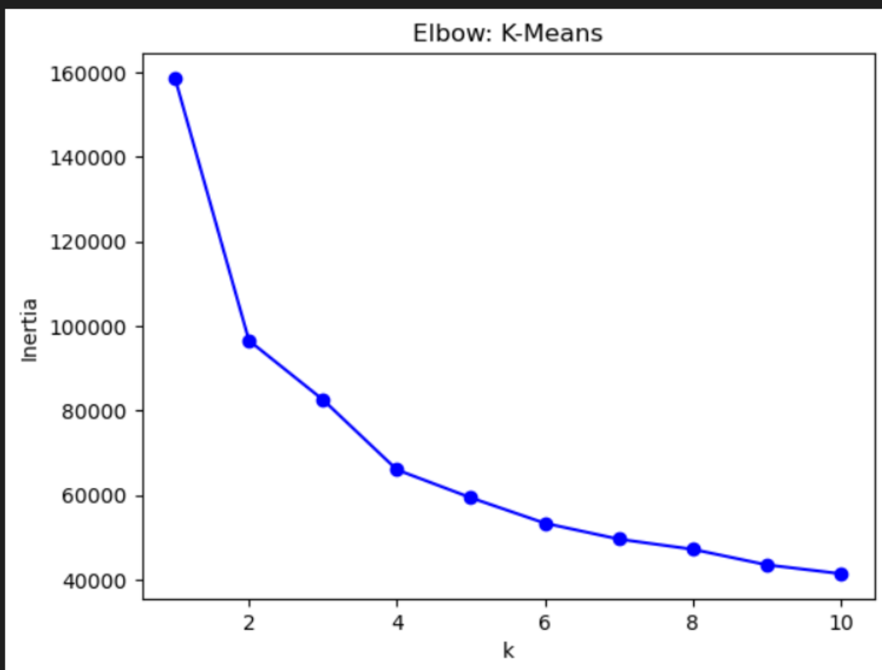
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

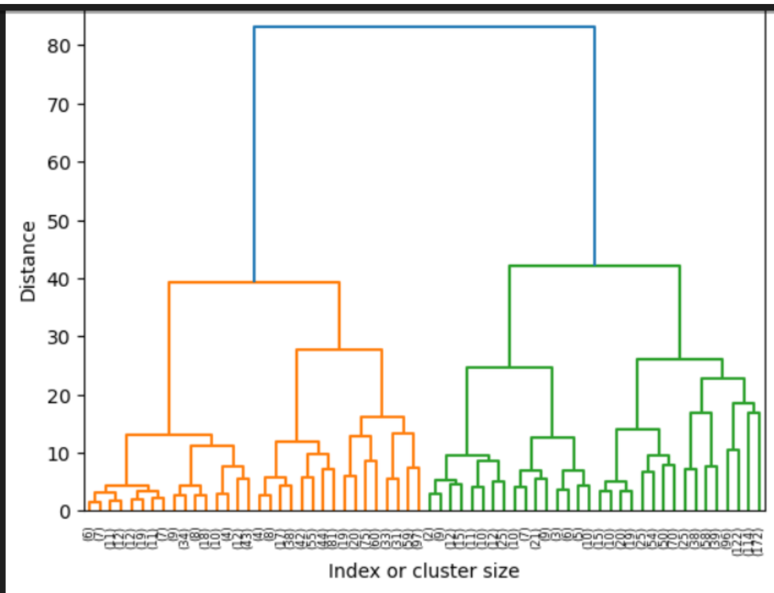
```
Weekly Summary (first 10 rows):
      city week_start  electricity_demand_MWh_mean  \
0  phoenix 2018-06-25                     1.401675
1  phoenix 2018-07-02                     1.784635
2  phoenix 2018-07-09                     1.422362
3  phoenix 2018-07-16                     1.748564
4  phoenix 2018-07-23                     2.076293
5  phoenix 2018-07-30                     1.961205
6  phoenix 2018-08-06                     1.757869
7  phoenix 2018-08-13                     1.652839
8  phoenix 2018-08-20                     1.634627
9  phoenix 2018-08-27                     1.478420

   electricity_demand_MWh_std  electricity_demand_MWh_min  \
0                    0.626115                    0.365486
1                    0.748763                    0.474196
2                    0.475815                    0.591740
3                    0.588340                    0.678029
4                    0.716818                    0.947768
5                    0.693946                    0.897489
6                    0.670276                    0.699771
7                    0.570028                    0.846531
8                    0.560452                    0.630468
9                    0.580000                    0.524475
...
8              0.071561              -0.281875              0.454024
9              0.000000              -0.281875             -0.281875

[10 rows x 26 columns]
```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Anomaly counts:
 - Z-score anomalies:     1283
 - IQR anomalies:         5566
 - IsolationForest:       319
 - Total flagged:         5567

Sample anomalies:
              timestamp     city  electricity_demand_MWh  temperature_F  \
59  2018-07-03 18:00:00  phoenix                2.522036       1.963189
103 2018-07-05 14:00:00  phoenix                2.703447       1.404440
104 2018-07-05 15:00:00  phoenix                2.941931       1.670989
105 2018-07-05 16:00:00  phoenix                3.047924       1.899061
106 2018-07-05 17:00:00  phoenix                3.181775       2.075273
107 2018-07-05 18:00:00  phoenix                3.149841       2.293865
108 2018-07-05 19:00:00  phoenix                2.996966       2.455021
109 2018-07-05 20:00:00  phoenix                2.771391       2.574355
110 2018-07-05 21:00:00  phoenix                2.659963       2.644617
112 2018-07-05 23:00:00  phoenix                2.081757       2.753355

     humidity_pct  wind_speed_mph  pressure_mbar  precip_intensity  hour  \
59      -1.702408        1.001349      -0.931039         -0.281875    18
103     -1.292016       -0.916641      -0.458413         -0.281875    14
104     -1.366633       -1.085353      -0.395396         -0.281875    15
105     -1.515866       -0.841165      -0.411150         -0.281875    16
106     -1.590483       -0.557018      -0.426904         -0.281875    17
...
108         True        False           True
109         True        False           True
110         True        False           True
112         True        False           True
```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Original rows:             31812
IsolationForest anomalies: 318
Cleaned rows:              31494
Rows removed:              318
```

```
 warnings.warn(
```






```
Silhouette K-Means k=4: 0.27
K-Means ARI stability: [0.996 0.996 0.987 0.995 0.991 0.987 0.992 0.988 0.992 0.994] Mean: 0.992

Cluster Profiles:
             avg_demand  avg_temp   avg_hour
cluster_km
0.0            1.371216  1.580758  12.084685
1.0           -0.605392 -0.689091  11.310691
2.0            0.175485  0.112561  11.253747
3.0           -0.698497 -0.590566  11.723584
Cluster 0.0: high-demand hot around hour 12
```

```
Baseline metrics:
 MAE:   0.08714451799293321
 RMSE:  0.13108303316877218
 MAPE:  0.9836857205478386
```

```
Linear Regression:
 MAE:   0.07846448389879777
 RMSE:  0.11422396845937395
 MAPE:  1.4371954841189938

Random Forest:
 MAE:   0.08109003398419594
 RMSE:  0.11707882520238001
 MAPE:  2.517179876873202

XGBoost:
 MAE:   0.07956127661758515
 RMSE:  0.11547495034901128
 MAPE:  2.3573934540093555

Stacking Ensemble:
 MAE:   0.08011087984188509
 RMSE:  0.11538196641318331
 MAPE:  2.491251717069248
```

```
RF best params: {'max_depth': 10, 'n_estimators': 200}
```

```
XGB best params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
```

```
c:\Users\admin\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:200: UserWarning: Do not pass an `
  super().__init__(**kwargs)
Epoch 1/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 0.1674 - val_loss: 0.0117
Epoch 2/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - loss: 0.0337 - val_loss: 0.0082
Epoch 3/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - loss: 0.0266 - val_loss: 0.0076
Epoch 4/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - loss: 0.0253 - val_loss: 0.0079
Epoch 5/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - loss: 0.0248 - val_loss: 0.0071
Epoch 6/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - loss: 0.0242 - val_loss: 0.0069
Epoch 7/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - loss: 0.0216 - val_loss: 0.0065
Epoch 8/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - loss: 0.0215 - val_loss: 0.0071
Epoch 9/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - loss: 0.0211 - val_loss: 0.0086
Epoch 10/10
397/397 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - loss: 0.0217 - val_loss: 0.0068

<keras.src.callbacks.history.History at 0x1ffe83addf0>
```

```
Epoch 1/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - loss: 0.2393 - val_loss: 0.0621
Epoch 2/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 13s 20ms/step - loss: 0.1222 - val_loss: 0.0525
Epoch 3/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 13s 20ms/step - loss: 0.1134 - val_loss: 0.0525
Epoch 4/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 13s 20ms/step - loss: 0.1081 - val_loss: 0.0465
Epoch 5/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 13s 20ms/step - loss: 0.1046 - val_loss: 0.0473
Epoch 6/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 0.1049 - val_loss: 0.0538
Epoch 7/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 15s 23ms/step - loss: 0.1007 - val_loss: 0.0492
Epoch 8/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - loss: 0.0989 - val_loss: 0.0507
Epoch 9/50
634/634 ━━━━━━━━━━━━━━━━━━━━ 13s 20ms/step - loss: 0.0976 - val_loss: 0.0492

<keras.src.callbacks.history.History at 0x2000e20f7a0>
```

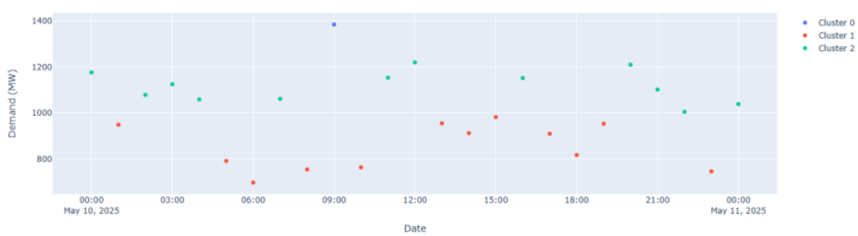
```
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step
LSTM MAE: 0.050414566
```

Next 2–3 Days Forecast (Hourly)

- 2025-05-10 00:00:00 → 1196.64 MW
- 2025-05-10 01:00:00 → 865.14 MW
- 2025-05-10 02:00:00 → 932.75 MW
- 2025-05-10 03:00:00 → 937.61 MW
- 2025-05-10 04:00:00 → 537.5 MW
- 2025-05-10 05:00:00 → 1333.03 MW
- 2025-05-10 06:00:00 → 1158.67 MW
- 2025-05-10 07:00:00 → 981.08 MW
- 2025-05-10 08:00:00 → 837.87 MW
- 2025-05-10 09:00:00 → 1009.27 MW
- 2025-05-10 10:00:00 → 930.88 MW
- 2025-05-10 11:00:00 → 1161.57 MW
- 2025-05-10 12:00:00 → 906.38 MW
- 2025-05-10 13:00:00 → 1071.58 MW
- 2025-05-10 14:00:00 → 1441.48 MW
- 2025-05-10 15:00:00 → 939.49 MW
- 2025-05-10 16:00:00 → 706.47 MW
- 2025-05-10 17:00:00 → 1006.87 MW
- 2025-05-10 18:00:00 → 1173.43 MW
- 2025-05-10 19:00:00 → 1187.09 MW
- 2025-05-10 20:00:00 → 1134.94 MW
- 2025-05-10 21:00:00 → 1024.31 MW
- 2025-05-10 22:00:00 → 1172.47 MW
- 2025-05-10 23:00:00 → 1353.81 MW
- 2025-05-11 00:00:00 → 1001.13 MW

Demand Clusters (k=3) for seattle

Demand Clusters (k=3) for seattle



Help & Documentation

- Select a city and date range to forecast demand.
- Choose model (RF/XGB/LSTM). Adjust `k` for clustering.
- Forecast shows actual vs predicted. Cluster groups similar periods.