Hierarchical Time Series Clustering

Zachary Olea

November 2024

Abstract

Hierarchical time series data provides essential insights across various domains by allowing multilevel analysis of temporally organized information. This paper explores hierarchical time series clustering through both synthetic simulations and real-world datasets, using the M5 Walmart competition data as a practical example. A combination of dimension reduction techniques (PCA, autoencoders, and VAEs) and clustering algorithms (K-means, DBSCAN, GMMs) were employed to identify latent groupings within large-scale time series. Additionally, Dynamic Time Warping (DTW) was used to capture shape-based similarities in temporal patterns, further enhancing clustering accuracy. My findings reveal that unsupervised clustering can effectively uncover hidden hierarchies, often revealing more homogeneous groupings than traditional predefined hierarchies.

Contents

1 Introduction

1.1 Time Series Data

Time series data comprises observations recorded sequentially over time, represented as:

$$\{X_t\}_{t=1}^{n} = X_1, X_2, X_3, \ldots, X_n, \tag{1}$$

where X_t denotes the observed value at time t and n is the total number of observations. In visualizing time series, the temporal dimension is typically placed on the x-axis, while the values of the series are plotted on the y-axis. Time series analysis provides a framework for identifying patterns, such as seasonality, and developing models to forecast future values based on past observations.

In practice, industry analysts frequently work with multiple variables observed over time. Such data can be organized as a *multivariate time series*, in which each observation comprises a vector of values for distinct variables. Formally, a multivariate time series at time t can be represented as:

$$\mathbf{X}_t = \begin{pmatrix} X_{1,t} \\ X_{2,t} \\ \vdots \\ X_{m,t} \end{pmatrix}, \tag{2}$$

where $\mathbf{X}_t$ is a vector containing m variables at time t, and $X_{i,t}$ represents the value of the i-th variable at t. Across n time points, the multivariate time series can be expressed as:

$$\mathbf{X} = \begin{pmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,n} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,1} & X_{m,2} & \cdots & X_{m,n} \end{pmatrix}. \tag{3}$$

Another structure of interest in time series analysis is the *panel time series*, which involves several independent time series measured over the same time span. This can be represented as:

$$X^i = \{X_t^i\}_{t=1}^{n} = X_1^i, X_2^i, \ldots, X_n^i, \quad \text{for } i = 1, 2, \ldots, k, \tag{4}$$

where X^i denotes the i-th time series, with X_t^i as the observed value at time t. Each series X^i can be expressed individually as:

$$\begin{aligned} X^1 &= \{X_1^1, X_2^1, \ldots, X_n^1\}, \\ X^2 &= \{X_1^2, X_2^2, \ldots, X_n^2\}, \\ &\vdots \\ X^k &= \{X_1^k, X_2^k, \ldots, X_n^k\}. \end{aligned} \tag{5}$$

This allows for flexible modeling across time series types, from single-variable trends to complex, multidimensional data and panel structures.

1.2 Hierarchical Time Series

1.2.1 Hierarchical Modeling

In time series analysis, hierarchies provide a structured framework for organizing data across multiple levels, facilitating insights into complex, nested groupings such as geographical divisions (e.g., countries, regions, cities) or categorical segments (e.g., product types, business units). Often, multiple hierarchical structures can coexist within the same dataset, each converging at a common top level while differing in their underlying levels of aggregation. For example, one hierarchy may organize data spatially by geographical region, while another may categorize by product type; both ultimately contribute to the overall dataset at the highest level.

For a hierarchy of time series, let $\{X_t^{(l)}\}_{t=1}^n$ denote a time series at level l, where $l = 1, 2, \ldots, L$ represents increasing levels of detail. At each level, $X_t^{(l)}$ aggregates or decomposes information relative to the preceding level. For example, at level $l = 1$, $X_t^{(1)}$ may represent a total sales figure, while at level $l = 2$, $\{X_t^{(2,1)}, X_t^{(2,2)}, \ldots, X_t^{(2,m)}\}$ captures sales by individual product categories.

1.2.2 Time granularity

Alongside hierarchies, time series data can also be organized at various temporal granularities, such as annual, quarterly, monthly, or daily levels, allowing for analyses at both broad and detailed time scales. For example, a yearly time series $\{X_t^{(Y)}\}_{t=1}^N$ can be derived by aggregating from a monthly series $\{X_t^{(M)}\}_{t=1}^{12N}$, where each value $X_t^{(Y)}$ represents the cumulative or average measure across the corresponding months. These granularities enable flexibility in analysis, accommodating both long-term trends and short-term fluctuations. Furthermore, multi-scale models can integrate multiple granularities within a unified framework, allowing relationships across temporal scales to enhance forecast precision and reveal patterns unique to each level of aggregation.

1.2.3 Applications Across Industries

Hierarchies are critical in time series data analysis across numerous industries, as they provide the structure necessary to capture patterns and relationships at different levels of detail. In retail and e-commerce, for example, sales data can be organized hierarchically from individual stores or zip codes up to cities, states, and regions, enabling companies to analyze and forecast demand at both granular and aggregated levels. Asset management firms utilize hierarchical time series to manage data across multiple dimensions, such as distribution channels or investment strategies, in order to track and forecast client cash flows, portfolio returns, and performance variations within each segment. These hierarchical structures allow organizations to make more precise, tailored decisions at every level, supporting both localized actions and broader strategic initiatives.

1.2.4 Relevance Across Stakeholders

Hierarchical time series data serves as a critical tool for diverse stakeholders, offering them insights that guide both operational and strategic decisions. Executives rely on hierarchical data to shape corporate strategy by identifying trends at different organizational levels, from regional performance down to individual stores or customer segments, which helps in resource allocation and long-term planning. In corporate finance, hierarchical forecasts of revenues and expenses across departments or product lines are essential for accurate budgeting and financial planning. Supply chain managers utilize hierarchical demand forecasts—whether by product category, region, or store—to optimize logistics, improve inventory control, and reduce costs. Marketing analysts leverage hierarchical time series to assess campaign impacts across different market segments, allowing them to allocate resources effectively and maximize return on investment. By capturing these multiple layers of detail, hierarchical time series data enhances decision-making precision, supports risk management, and drives sustained growth across all levels of an organization.

1.3 Uncovering Hidden Hierarchies

While predefined hierarchies (e.g., those structured by geographic or organizational levels) provide a foundational view of data, they may not always capture the most meaningful patterns within complex datasets. Unsupervised methods allow us to uncover *hidden hierarchies* or relationships within time series data that are not apparent in existing structures. By clustering data without predefined categories, we can reveal new groupings and hierarchies based on intrinsic similarity patterns. For example, customer behavior data may yield clusters that are more representative of underlying purchasing patterns than traditional demographic segmentation, allowing organizations to identify emergent groupings and adjust strategies accordingly.

Predefined hierarchies may encompass substantial heterogeneity within groups, where individual time series can exhibit markedly different behaviors, thereby constraining the effectiveness of forecasting and analytical models. By employing unsupervised clustering methods, we can create more homogeneous groupings that share similar temporal characteristics, enhancing predictive accuracy. For instance, in retail data, clustering stores based on similar seasonal demand profiles rather than geographic location can yield more accurate inventory management and demand forecasting. These refined groupings facilitate the development of tailored strategies for each cluster.

2 Methodology

Clustering time series directly can be computationally intensive, especially when attempting to group based on all temporal dimensions. To reduce this complexity, dimensionality reduction techniques like Principal Component Analysis (PCA) or autoencoders can transform time series data into lower-dimensional

representations. These compressed representations retain critical patterns while significantly reducing computation time. Once the dimensionality is reduced, clustering algorithms such as k-means, DBSCAN, or Gaussian Mixture Models (GMM) can be applied. Although these methods are not hierarchical, they offer an efficient way to reveal distinct clusters within large datasets. For a more time-aware approach, Dynamic Time Warping (DTW) is often used to align time series by minimizing temporal distance, making it particularly useful for clustering tasks where the shape of the time series over time is more relevant than exact time-aligned matching.

2.1 Dimension Reduction

Principal Component Analysis (PCA) PCA is a linear dimensionality reduction technique that identifies the directions (principal components) of maximum variance in the data. By projecting the data onto a subset of these components, PCA reduces dimensionality while retaining the most significant variance. Mathematically, PCA seeks to solve the eigenvalue problem:

$$\mathbf{C}\mathbf{v} = \lambda\mathbf{v} \tag{6}$$

where $\mathbf{C}$ is the covariance matrix of the data, $\mathbf{v}$ represents the eigenvectors (principal components), and λ denotes the corresponding eigenvalues. The principal components are ordered by the magnitude of their eigenvalues, and a subset is selected to represent the data in reduced dimensions. PCA is particularly effective for data with linear correlations; however, for data with non-linear correlations, PCA may be less effective as it cannot capture complex, non-linear patterns within the data structure.

Autoencoders Autoencoders are a class of artificial neural networks used for unsupervised learning of efficient codings. They consist of an encoder that maps the input data to a latent space and a decoder that reconstructs the input from the latent representation. The network is trained to minimize the reconstruction error:

$$L = \|\mathbf{X} - \mathbf{X}'\|^2 \tag{7}$$

where $\mathbf{X}$ is the input data, and $\mathbf{X}'$ is the reconstructed data. I used a PyTorch implementation for autoencoding:

```
class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, encoding_dim),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, input_dim),
            nn.Sigmoid()
```

```
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

Variational Autoencoders (VAEs) VAEs are a probabilistic extension of autoencoders that model the latent space with a probability distribution, typically Gaussian. They consist of an encoder that outputs parameters of the distribution (mean and variance) and a decoder that generates data from samples drawn from this distribution. The training objective combines the reconstruction loss and a regularization term, which is the Kullback-Leibler divergence between the learned latent distribution and a prior distribution:

$$L = E_{q(\mathbf{z}|\mathbf{X})}[\log p(\mathbf{X}|\mathbf{z})] - \mathrm{KL}(q(\mathbf{z}|\mathbf{X}) \parallel p(\mathbf{z})) \tag{8}$$

where $q(\mathbf{z}|\mathbf{X})$ is the approximate posterior, and $p(\mathbf{z})$ is the prior.

```
class VAE(nn.Module):
    def __init__(self, input_dim, latent_dim):
        super(VAE, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2_mean = nn.Linear(64, latent_dim)
        self.fc2_log_var = nn.Linear(64, latent_dim)
        self.fc3 = nn.Linear(latent_dim, 64)
        self.fc4 = nn.Linear(64, input_dim)

    def encode(self, x):
        h1 = torch.relu(self.fc1(x))
        return self.fc2_mean(h1), self.fc2_log_var(h1)

    def reparameterize(self, mu, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h3 = torch.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        return self.decode(z), mu, log_var

```

2.2 Clustering Algorithms

After reducing the dimensionality of the time series data, clustering algorithms are applied to group similar time series based on their transformed representations.

K-Means Clustering K-Means is a partitioning algorithm that divides data into k clusters by minimizing the within-cluster sum of squares. It iteratively assigns each data point to the nearest cluster centroid and updates the centroids based on the mean of assigned points. The objective function is:

$$J = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu_i\|^2 \tag{9}$$

where C_i is the set of points in cluster i, and μ_i is the centroid of cluster i. K-Means is efficient but assumes clusters are spherical and of similar size.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) DBSCAN is a density-based clustering algorithm that identifies clusters as regions of high point density, separated by regions of low density. It requires two parameters: ϵ (the maximum distance between two points to be considered neighbors) and minPts (the minimum number of points required to form a dense region).

Gaussian Mixture Models (GMMs) GMMs are probabilistic models that assume data is generated from a mixture of several Gaussian distributions with unknown parameters. Each component is defined by a mean and covariance matrix, and the model estimates the probability that a data point belongs to each component. The parameters are typically estimated using the Expectation-Maximization (EM) algorithm, which iteratively maximizes the likelihood function:

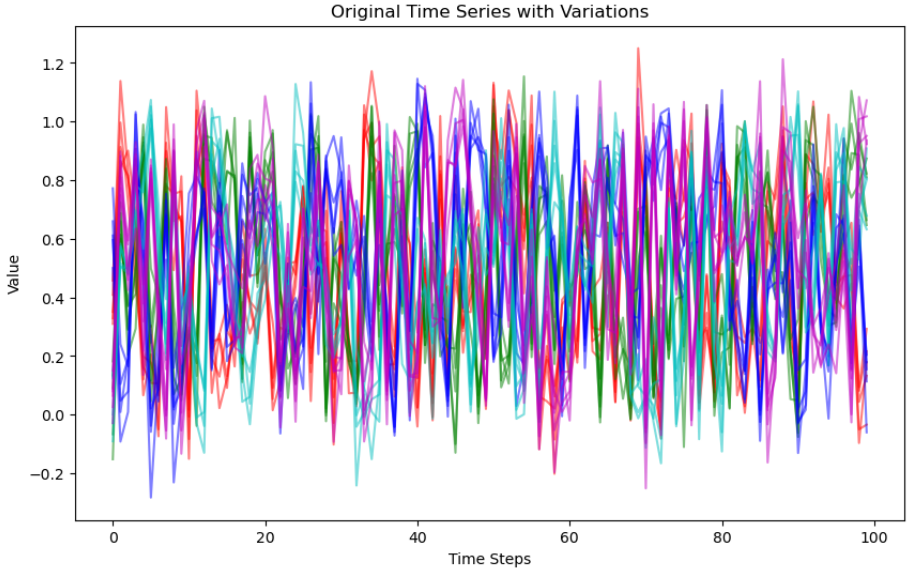
$$L = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{k} \pi_j \mathcal{N}(\mathbf{x}_i | \mu_j, \mathbf{\Sigma}_j) \right) \tag{10}$$

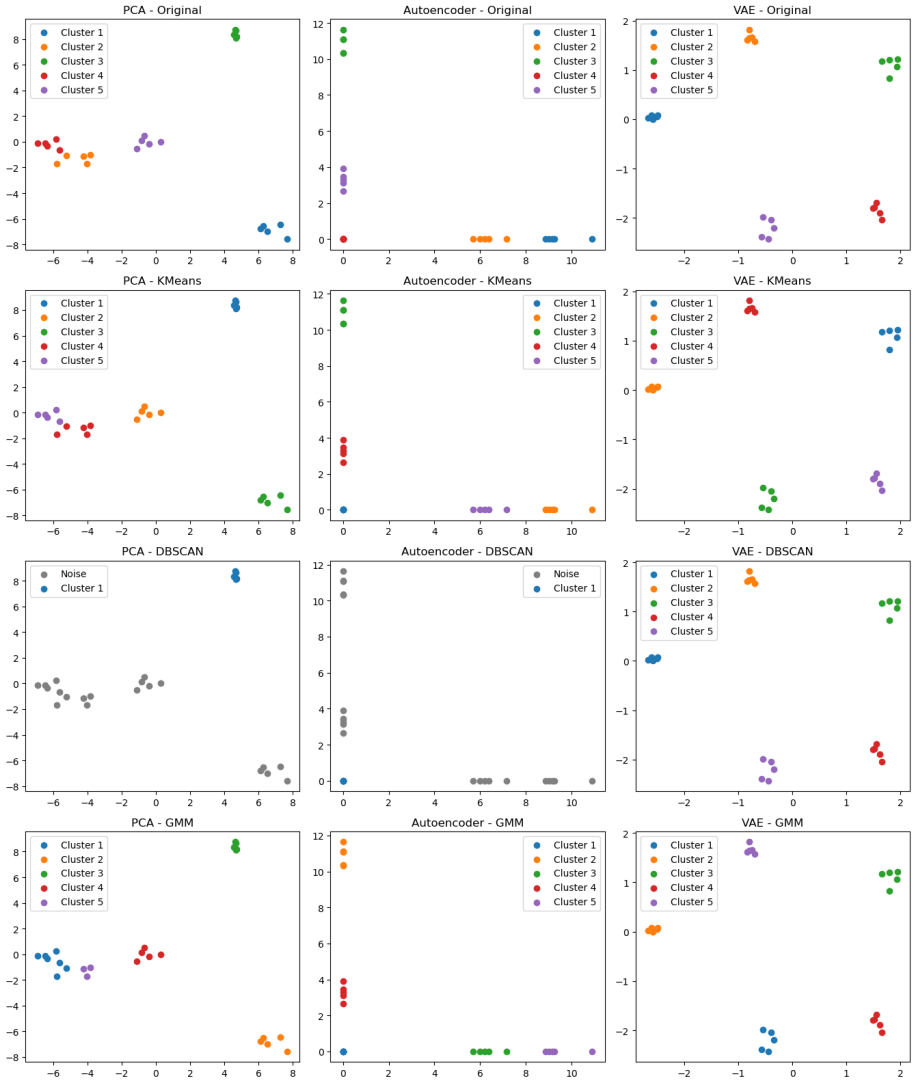
where π_j are the mixing coefficients, $\mathcal{N}$ is the Gaussian distribution, μ_j and $\mathbf{\Sigma}_j$ are the mean and covariance of component j, respectively.

2.3 Simulation 1: Dimension Reduction with Clustering

In this work, I generated synthetic time series data by adding Gaussian noise to a set of base series, producing multiple variations for each series to simulate realistic fluctuations. After scaling the data for uniformity, I applied Principal Component Analysis (PCA) as an initial dimensionality reduction technique to capture the primary variance components within the dataset. Following this, I constructed an autoencoder and a variational autoencoder (VAE), both trained to encode the data into a compressed, low-dimensional space, effectively learning latent representations that capture temporal dependencies within the series.

For each representation, I applied clustering algorithms (KMeans, DBSCAN, and Gaussian Mixture Models) to segment the data into distinct groups based on their encoded features. This approach allowed me to compare the efficacy of each encoding method in preserving the structural patterns of the synthetic time series within the reduced dimensions. Finally, I visualized the original and encoded clusters, assessing the clustering results across methods to highlight how well each representation retained separability among the base series and their variations.





The hue in the first row represents the original synthetic groupings we aim to replicate. KMeans accurately recovers these groups across all three dimensionality reduction techniques, while DBSCAN fails except with the VAE, which performs well across methods. PCA with GMM misclassifies two series but succeeds with the encoded versions. Given the randomized data and stochastic nature of KMeans, these results serve as an illustrative rather than definitive comparison of clustering alignment with the original groupings, highlighting potential variability across runs.

2.4 Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) is a similarity measure designed for time series alignment, allowing sequences of varying lengths to be compared by non-linearly warping the time axis. Unlike Euclidean distance, which compares sequences point-by-point, DTW accommodates temporal distortions, making it highly effective for clustering and analysis.

Given two time series $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$, DTW constructs a cost matrix D, with each element $D(i, j)$ representing the cumulative cost of aligning x_i with y_j. The optimal warping path $W = (w_1, w_2, \ldots, w_L)$ minimizes the total alignment cost:

$$\text{DTW}(X, Y) = \min_{W} \left\{ \sum_{k=1}^{L} d(x_{i_k}, y_{j_k}) \right\} \tag{11}$$

where $d(x_{i_k}, y_{j_k})$ is typically the squared Euclidean distance. The warping path is subject to boundary, continuity, and monotonicity constraints to ensure meaningful alignment.

DTW's ability to handle temporal shifts, accelerations, and decelerations allows it to capture similarity in shape across sequences with different time patterns. It accommodates sequences of varying lengths, preserving data integrity and making it particularly useful for sparse series.

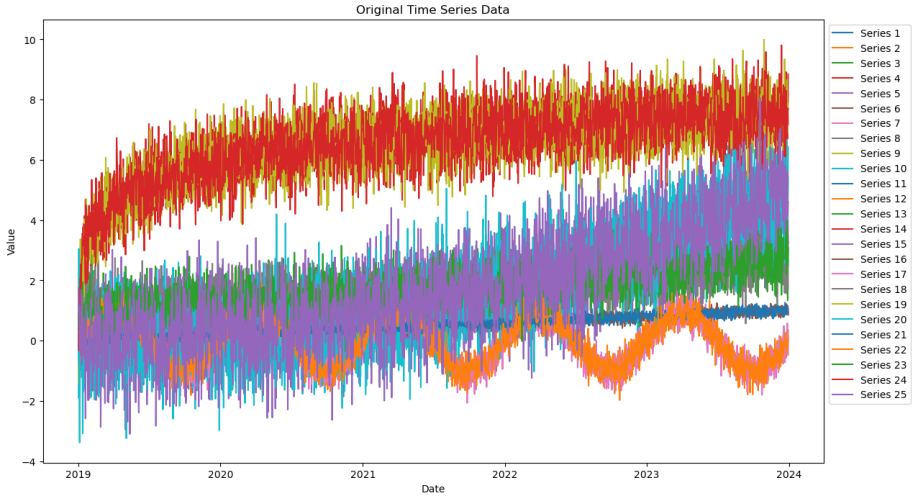
In clustering, DTW serves as a distance metric that enables grouping of time series with similar shapes. Algorithms like K-Medoids can leverage DTW to identify representative medoids within clusters, yielding more meaningful clusters when temporal misalignments are present. Hierarchical clustering using DTW constructs dendrograms that reflect the temporal dynamics of the data, revealing nested relationships and providing insights at various levels of granularity.

2.5 Simulation 2: Dynamic Time Warping

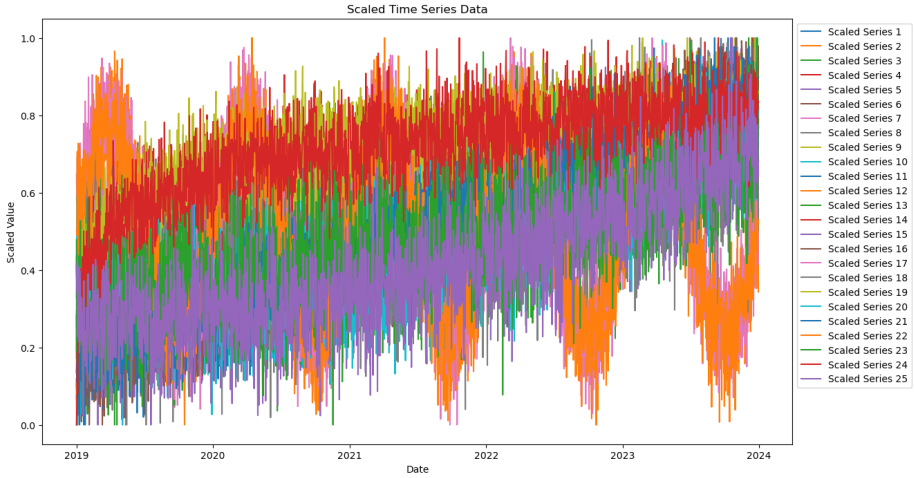
I begin by generating 25 synthetic time series, each representing daily observations over a 5-year period. To capture diverse temporal behaviors, each time series is assigned to one of five distinct states, representing various trend patterns and dynamics commonly observed in time series data. Specifically, these states include (1) a linear trend with added noise, (2) a sinusoidal wave with periodic oscillations and random fluctuations, (3) exponential growth, (4) logarithmic growth, and (5) a quadratic trend with increasing acceleration over time. Each state offers a unique pattern of temporal progression, emulating different real-world scenarios.

To further diversify the data, each state is paired with a specific level of randomness, represented as the standard deviation of noise, which ranges from low (0.1) to high (1.0). This variability introduces realistic deviations in each time series, reflecting levels of unpredictability that often characterize time series

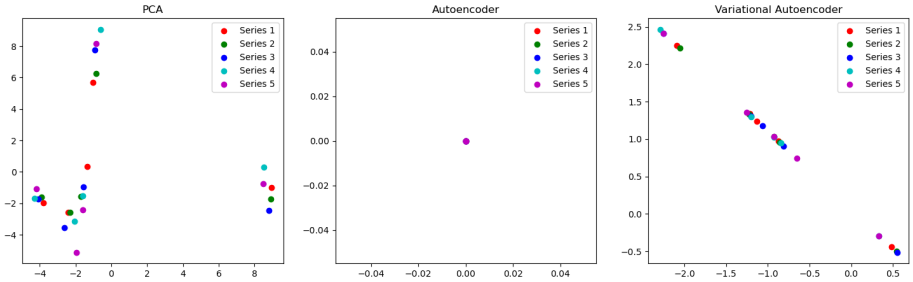
data. The combination of distinct patterns and varied noise levels provides a comprehensive dataset for testing models' ability to handle heterogeneity in time series characteristics, thereby offering insights into model robustness and generalizability across different time-dependent patterns and noise conditions.



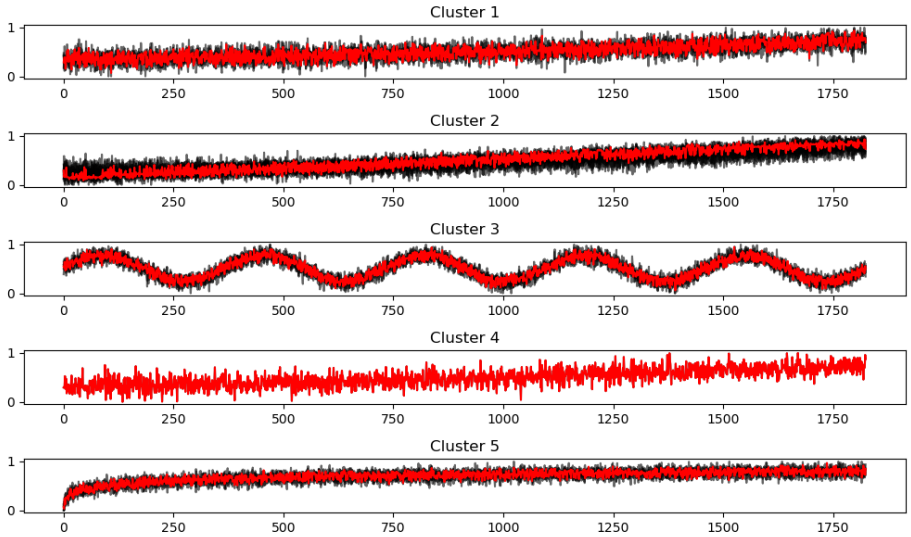
Each time series is subsequently normalized to have a minimum value of 0 and a maximum value of 1, ensuring consistency in scale across all series. This normalization enhances comparability and standardizes the data range.



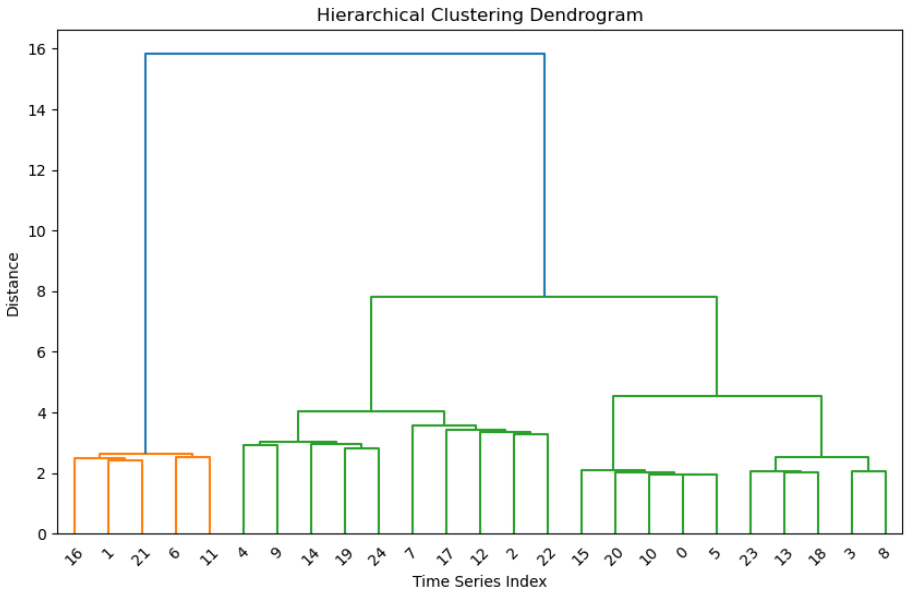
For this simulation, dimension reduction to two dimensions did not result in any useful boundaries that would result in original groupings to be identified through clustering.



However, using DTW, I was able to isolate most of the 5 groupings through clustering.



Lastly, we can see from the Dendrogram, that it was able to isolate 5 distinct groups, and we can see all series were correctly identified here, with the numbered series being 5 apart for each group. We can see through the hierarchy, we gained new information as well, in that one branch is isolated (the sine series) and that the other 4 are more closely related.



3 Real World Example

To demonstrate hierarchical time series clustering on real world data, I will use the M5 Competition dataset. The M5 Competition, organized by Walmart and Kaggle, challenged participants to develop accurate forecasting models for daily sales across various products, stores, and states, aiming to improve inventory and resource planning. This dataset is particularly valuable for exploring hierarchical clustering as it includes multiple predefined hierarchies based on product categories and geographical divisions. In this paper, I focus on clustering techniques to generate a new, data-driven hierarchy and compare it to the existing hierarchies within the dataset.

3.1 Data - M5 Competition Walmart

The M5 Competition dataset contains Walmart transactional sales data and is structured across several files, each offering distinct but complementary information:

1. `calendar.csv` - Contains date-related data, including time granularities (year, month, and day) and information on holidays and promotional events that may affect sales patterns.

2. `sales_train_validation.csv` - Provides daily sales quantities from Day 1 to Day 1913, along with associated metadata on products (e.g., item, department, category) and geographical locations (e.g., store and state).

3. `sales_train_evaluation.csv` - This file includes daily sales quantities of products from Day 1 through Day 1941. I exclude this file for analyses as `sell_prices.csv` does not contain data for the additional days.

4. `sample_submission.csv` - An example submission template for the competition, containing the format for forecasting sales for the next 28 days.

5. `sell_prices.csv` - Records the daily selling price of each product for the days where sales were recorded up to Day 1913. Price information is linked to specific items, stores, and weeks.

3.2 Transformation to Recreate Transaction Table

To recreate the transaction table required for analysis, I performed a series of transformations that consolidated data across multiple files. The steps included:

1. **Unpivoting Sales Data**: The `sales_train_evaluation` data was unpivoted to transform daily sales from a wide format (one column per day) into a long format, with each row representing a single day's sales for each product.

2. **Joining with Calendar Data**: The `calendar.csv` file was joined with the unpivoted sales data on the day identifier (`d`) to incorporate date information and relevant events, such as holidays, providing additional context for the sales data.

3. **Joining with Price Data**: The `sell_prices.csv` file was joined using keys (`wm_yr_wk`, `store_id`, and `item_id`) to add product pricing data for the respective days. This allowed us to calculate net sales values by multiplying quantity sold by price.

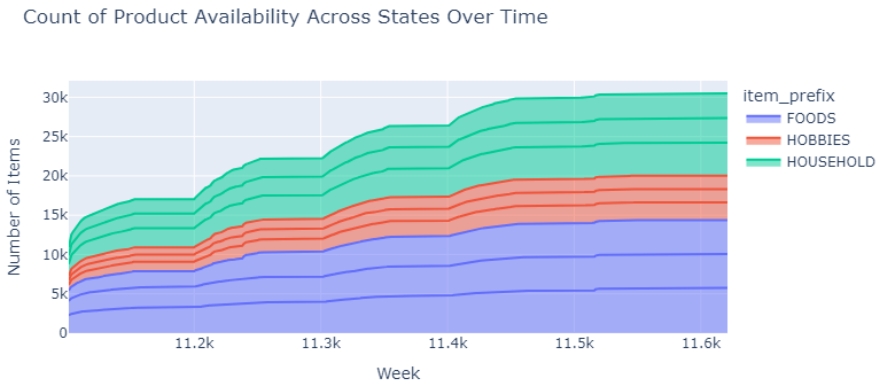
4. **Adding Availability Indicators**: To account for periods when a product was or wasn't available for sale, I created flags indicating whether each observation fell within the date range for which pricing data was available for that product at the given store.

5. **Calculating Net Sales**: Finally, a new column was computed to capture the net sales (i.e., revenue) by multiplying `sell_price` by `sales` for each entry, providing a comprehensive view of transactional sales data with temporal, product, and pricing information.

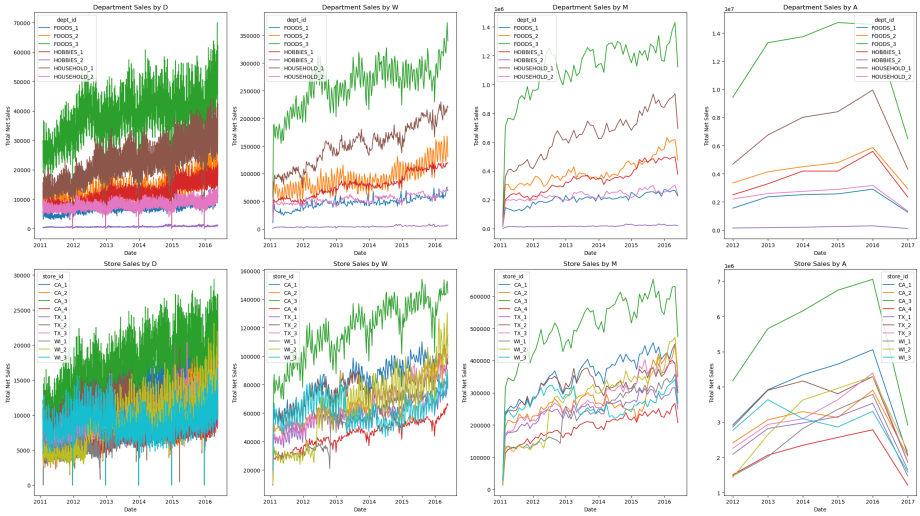
These transformations yielded a consolidated transaction table, integrating time-series sales data with temporal, product, and pricing details, setting up the dataset for hierarchical clustering and comparison to predefined structures.

3.2.1 Exploratory Data Analysis

The transaction table created from our transformation contains 59,181,090 rows and 28 columns, though there is some redundancy, such as day names and weekday identifiers, which can be removed for efficiency. In the original dataset, the daily sales table was pivoted in a way that obscured product availability, with zeros replacing missing values. Consequently, there is no differentiation between a product genuinely having zero sales and a product not being available on that day. To address this, I included an availability indicator, as discussed previously. After filtering out days when products were not available, the transaction table was reduced to 46,881,677 rows.



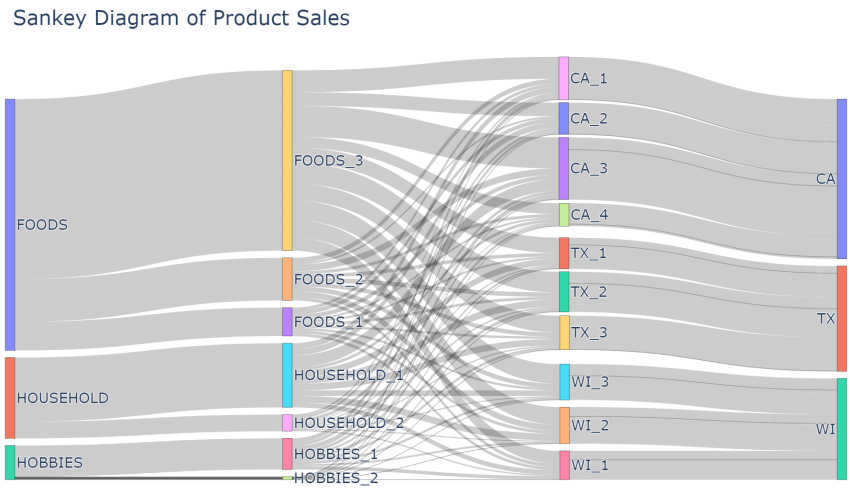
The plot above illustrates product availability over time. One notable observation is that, while not all items are available in the earliest time frames, all items are consistently available towards the end of the dataset. This indicates that the dataset focuses on existing product lines, potentially introducing *survivorship bias*—as only currently active products are represented, which could influence model outcomes and limit generalizability.

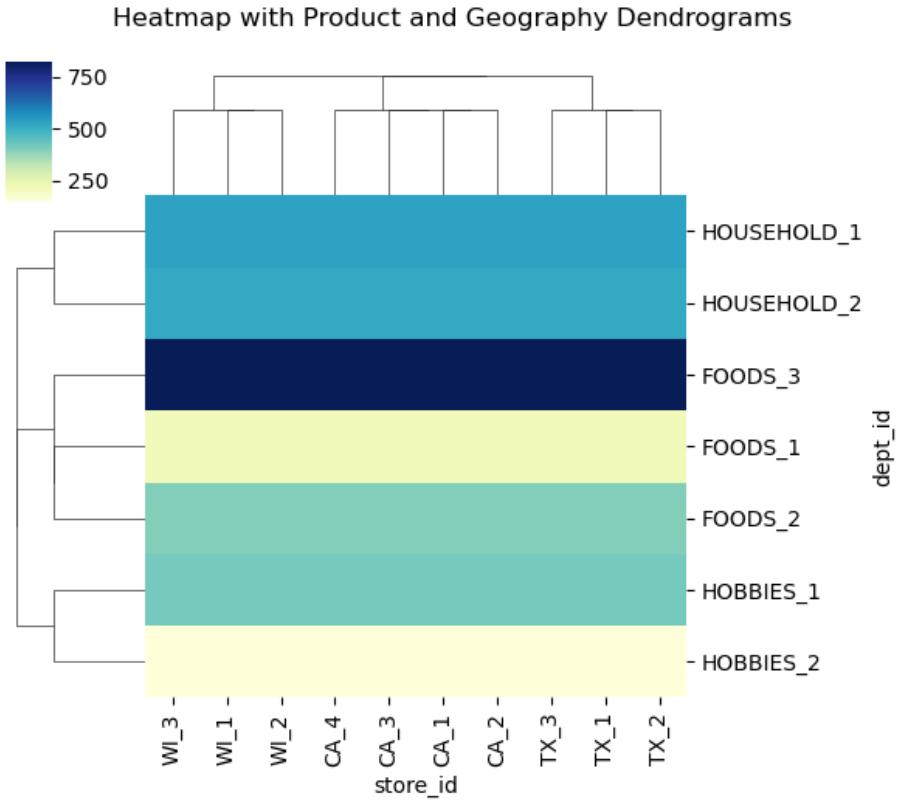


The subset of data used in this analysis includes transactions from 10 stores across 3 states (4 in California, 3 in Texas, and 3 in Wisconsin). There are 3,049 unique items spanning 7 departments and 3 categories ("Food," "Household," "Hobbies"), amounting to 30,490 unique time series at the lowest hierarchical level. This level, often referred to as the "bottom" or "child" level in hierarchical time series analysis, captures sales data for individual items in specific stores.

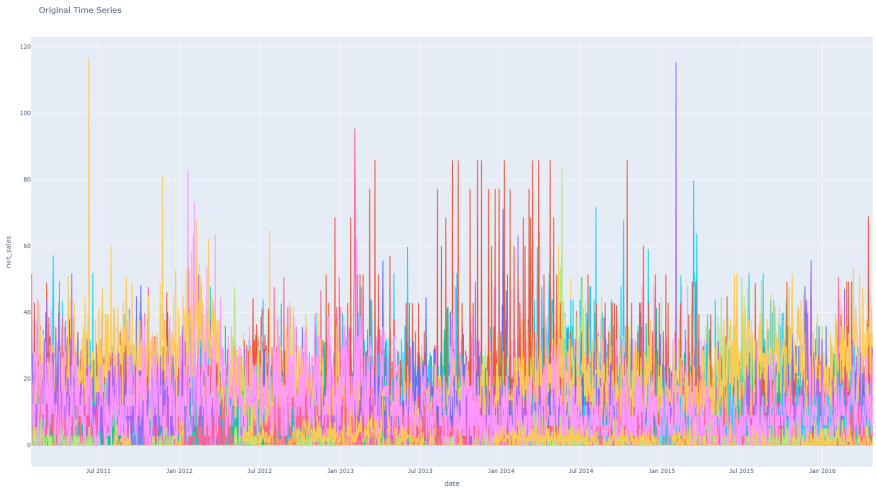
3.2.2 Predefined Hierarchies

To better understand and visualize the predefined hierarchies in the dataset, we can represent them through a Sankey diagram and a dendrogram. The *Sankey diagram* provides a flow-based visualization, illustrating the connections between different hierarchical levels by showing the distribution of items across categories, departments, and stores. In contrast, the *dendrogram* offers a hierarchical tree structure that highlights the nested relationships, making it easier to observe how items group under various levels, from categories down to individual products. By analyzing these visualizations, we gain insights into the inherent structure of the dataset and evaluate the coherence of the existing hierarchies, providing a foundation for comparison with clusters derived from unsupervised methods.

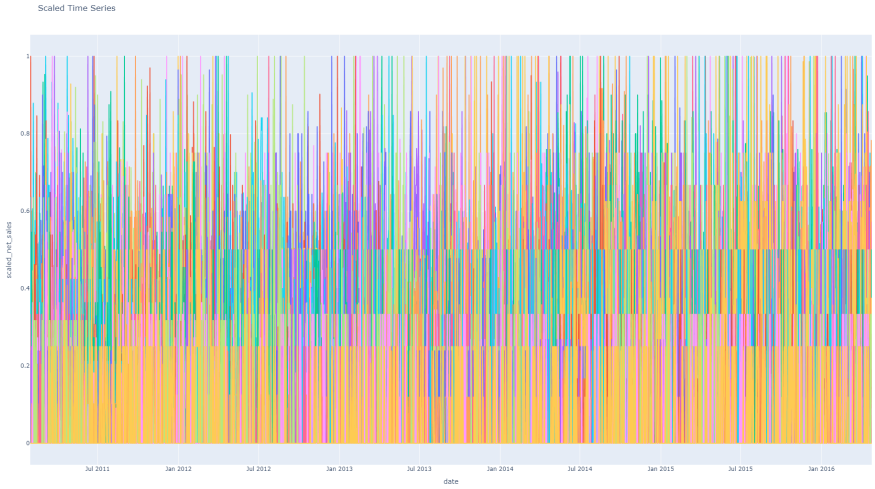

Sankey Diagram of Product Sales
FOODS
HOUSEHOLD
HOBBIES
FOODS_3
FOODS_2
FOODS_1
HOUSEHOLD_1
HOUSEHOLD_2
HOBBIES_1
HOBBIES_2
CA_1
CA_2
CA_3
CA_4
TX_1
TX_2
TX_3
WI_3
WI_2
WI_1
CA
TX
WI


Heatmap with Product and Geography Dendrograms
750
500
250
HOUSEHOLD_1
HOUSEHOLD_2
FOODS_3
FOODS_1
FOODS_2
HOBBIES_1
HOBBIES_2
dept_id
WI_3
WI_1
WI_2
CA_4
CA_3
CA_1
CA_2
TX_3
TX_1
TX_2
store_id

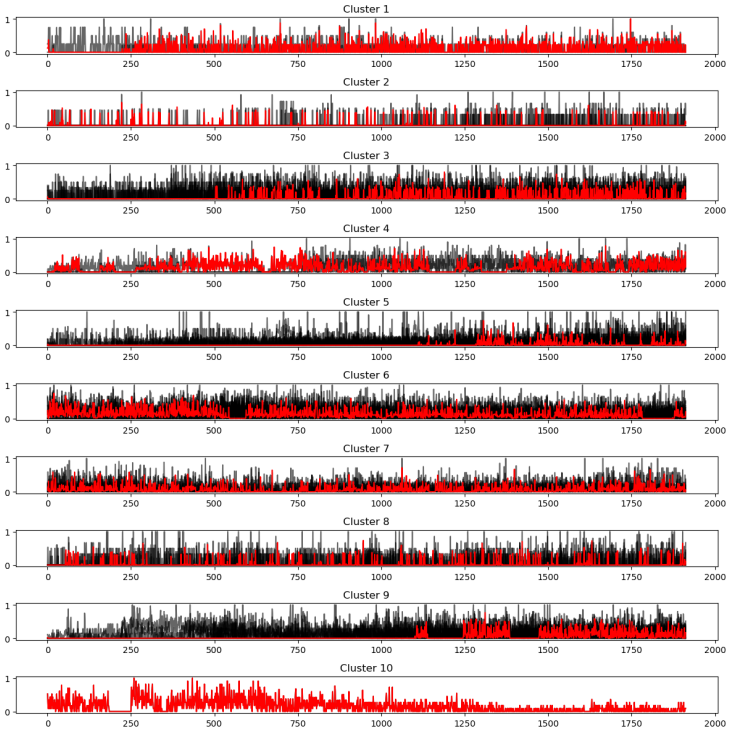
3.2.3 Clustered Hierarchies



Due to computational constraints, I randomly selected 100 unique IDs from the transaction table shown in the figure above. I then applied Min-Max scaling to bring each series between 0 and 1, normalizing the data.



Next, I ran K-means clustering with 10 clusters and using DTW and Dynamic Time Warping Barycenter Averaging (DBA) to calculate centroids. Finally, I visualized each cluster, plotting the individual series in an opaque black to show density and their centroids in red, giving a clear snapshot of how the time series group together and where each cluster's central trend lies.



Additionally, I provided below how the *Store IDs* and *Department IDs* compare to how they are grouped in these clusters. We can see that the clusters are finding patterns between stores and products, which is the goal of what we are trying to find, hierarchies beyond the predefined.

clusters store id	0	1	2	3	4	5	6	7	8	9
CA_1	0	0	1	0	8	1	1	0	4	0
CA_2	0	0	1	0	3	6	1	0	4	1
CA_3	0	0	1	1	2	1	0	0	1	0
CA_4	1	2	1	0	2	0	0	0	0	0
TX_1	1	0	1	0	2	0	3	4	0	0
TX_2	0	1	1	0	4	0	3	1	1	0
TX_3	0	0	2	1	2	1	0	0	2	0
WI_1	0	0	0	1	1	0	0	1	3	0
WI_2	0	1	4	0	1	1	0	3	0	0
WI_3	0	1	3	0	3	1	0	1	2	0

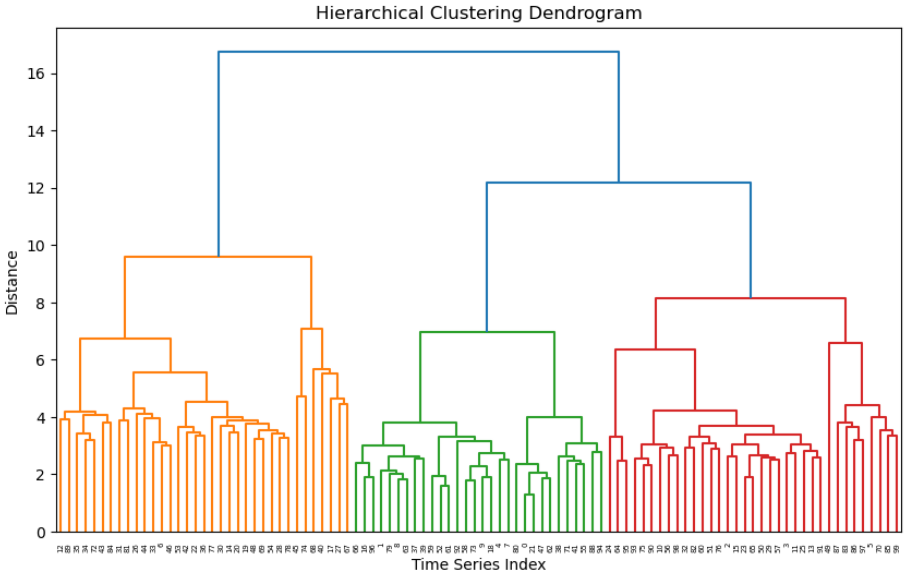
clusters dept id	0	1	2	3	4	5	6	7	8	9
$FOODS_1$	0	0	1	0	2	0	0	1	2	0
$FOODS_2$	0	1	0	0	2	1	2	4	3	0
$FOODS_3$	1	0	4	1	9	2	0	0	5	0
$HOBBIES_1$	1	1	3	0	5	2	3	1	2	0
$HOBBIES_2$	0	0	1	0	1	2	1	0	0	0
$HOUSEHOLD_1$	0	3	2	0	3	1	1	2	2	1
$HOUSEHOLD_2$	0	0	4	2	6	3	1	2	3	0

After converting the scaled data into a PyTorch tensor, I computed the pairwise Dynamic Time Warping (DTW) distances between every time series in the tensor.

```
def generalized_dtw(tensor_data):
    num_series = tensor_data.shape[0]
    dtw_distances = torch.zeros((num_series, num_series))
    for i in range(num_series):
        for j in range(i + 1, num_series):
            dist = dtw(tensor_data[i].numpy(), tensor_data[j].numpy())
            dtw_distances[i, j] = dist
            dtw_distances[j, i] = dist
    return dtw_distances

ts_tensor = torch.tensor(scaled_time_series_data, dtype=torch.float32)
gdtw_distances = generalized_dtw(ts_tensor)
```

By iterating through each pair of time series, this function fills a symmetric distance matrix (*gdtw_distances*) with DTW values, capturing how similar or different each pair is. With this distance matrix, I extracted the upper triangular values for hierarchical clustering, applied Ward's method, and then made a visualization of the dendrogram to see the structure among the time series.



4 Conclusion

While this paper provides a robust framework for hierarchical time series clustering, further exploration into Bayesian hierarchical models could deepen interpretability and adaptability in complex datasets. Specifically, integrating Bayesian clustering could allow for probabilistic modeling of hierarchical relationships, enabling a more nuanced handling of uncertainty and variability within clusters. Such methods could dynamically adjust to changing time series patterns and offer valuable insights for applications requiring high interpretabil-

ity, such as in risk-sensitive domains like finance or supply chain optimization. Future research might also investigate the use of Bayesian nonparametric models, such as Dirichlet processes, to discover flexible and potentially overlapping clusters within large-scale hierarchical data, enhancing scalability and resilience in model performance across diverse applications.