Assignment 1- Report
Team-62

Introduction

This report details the processing and aggregation of electric vehicle data from the Electric_Vehicle_Data.csv file. The goal is to preprocess the data by handling missing values and generalizing certain attributes. The processed data is then aggregated based on the new generalized attributes to extract meaningful insights.

Data Loading and Initial Processing

1. **Data Loading**:
    - The dataset is loaded from the file Electric_Vehicle_Data.csv using Pandas.
    - The pd.set_option functions are used to display all columns and adjust the display width to ensure that the entire dataset is visible for inspection.
2. **Handling Missing Values**:
    - The code identifies columns with missing values using data.isnull().sum().
    - It prints out the columns that contain missing values, which provides a preliminary view of data completeness.
    - Columns with missing values are dropped using data.dropna(), ensuring that only complete records are retained.

Attribute-Oriented Induction:

1.Attribute Removal

The columns 'Legislative District', 'VIN (1-10)', 'Base MSRP', 'Vehicle Location', and 'State' are removed from the dataset. These columns are deemed unnecessary for the analysis.

2.Attribute Generalization

1. **Generalizing 'Model Year'**:
    - The 'Model Year' column is generalized into three categories:
        - 'Old': For years 2015 and earlier.
        - 'Moderate': For years between 2016 and 2019.
        - 'New': For years 2020 and later.
    - A new column 'Generalized Model Year' is added to the dataset to reflect these categories.
2. **Generalizing 'Electric Range'**:
    - The 'Electric Range' is categorized into:
        - 'Low Range': For ranges less than 50 miles.
        - 'Moderate Range': For ranges between 50 and 150 miles.
        - 'High Range': For ranges greater than 150 miles.
    - A new column 'Generalized Electric Range' is added.
3. **Generalizing 'CAFV Eligibility'**:
    - The 'Clean Alternative Fuel Vehicle (CAFV) Eligibility' column is generalized into:

        - 'Eligible': If the vehicle is eligible.
        - 'Not Eligible': If the vehicle is not eligible.
    - A new column 'Generalized CAFV Eligibility' is created.
4. **Generalizing 'Make'**:
    - The 'Make' column is categorized into:
        - 'Popular EV Brands': Includes brands such as TESLA, NISSAN, and CHEVROLET.
        - 'Other Brands': All other brands.
    - A new column 'Generalized Make' is added.

Data Aggregation

1. **Aggregating Data**:
    - The dataset is grouped by a combination of the following columns: 'Model Year', 'Generalized Model Year', 'Make', 'Generalized Make', 'Electric Vehicle Type', 'Electric Range', 'Generalized Electric Range', and 'Generalized CAFV Eligibility'.
    - The count of occurrences for each combination is computed using groupby().size(), and the result is stored in a new DataFrame called aggregated_data.
2. **Summarizing Counts**:
    - The aggregated data is further summarized by grouping based on the generalized attributes: 'Generalized Model Year', 'Generalized Make', 'Electric Vehicle Type', 'Generalized Electric Range', and 'Generalized CAFV Eligibility'.
    - The total count for each combination is computed and stored in a DataFrame called summary_counts.

BUC Algorithm Implementation

1. In-Memory Implementation of BUC Algorithm

Code Overview

The in-memory implementation of the BUC algorithm is designed to process the entire dataset within the available main memory. The code performs the following steps:

1. **Aggregation Function**:
    - aggregate(df, group_by_cols): Groups the DataFrame df by the columns specified in group_by_cols and counts the occurrences for each group. The result is a DataFrame with the count of each combination of values.
2. **BUC Function**:
    - buc(df, dimensions, minsup=1, dim_index=0): A recursive function that applies the BUC algorithm. It aggregates data for each combination of dimensions, processing each dimension one by one. The function includes

cases where dimensions are grouped by 'ALL', indicating aggregation without the current dimension.

3. **Execution**:
    - The data_small DataFrame is created with selected columns: 'Model Year', 'Make', and 'Electric Vehicle Type'.
    - The buc function is called with the specified dimensions and minimum support (minsup=2).
    - The results are combined and aggregated into a final DataFrame, buc_result.

Results

- The aggregated results are printed, and the execution time is recorded.
- Example output displays the first few rows of the aggregated data and the time taken for execution.

Execution Time

The execution time of the in-memory BUC algorithm is reported, reflecting the time taken to process the data and compute the results using the available memory.

2. Out-of-Memory Implementation of BUC Algorithm

Code Overview

The out-of-memory implementation handles large datasets that may not fit into the main memory by using paging (chunk processing). The steps include:

1. **BUC with Paging**:
    - buc_out_of_memory(input_file, dimensions, output_file, minsup=1, chunk_size=10000): Processes the dataset in chunks, performs BUC on each chunk, and writes intermediate results to temporary files. The results from all chunks are merged and saved to the specified output_file.
2. **Temporary File Handling**:
    - Data is read in chunks from input_file and saved to temporary CSV files. Each chunk is processed independently to compute the BUC results.
    - After processing, the temporary files are removed, and results are appended to the output_file.
3. **Execution**:
    - The buc_out_of_memory function is called with the input file, dimensions, output file path, minimum support, and chunk size.
    - The results are read from the output file and displayed.

Results

- The results from the out-of-memory BUC algorithm are displayed, including the first few rows of the final output DataFrame.
- The execution time is recorded and reported.

Execution Time

The execution time of the out-of-memory BUC algorithm is reported, reflecting the time taken to process the data in chunks and write the results to the output file.

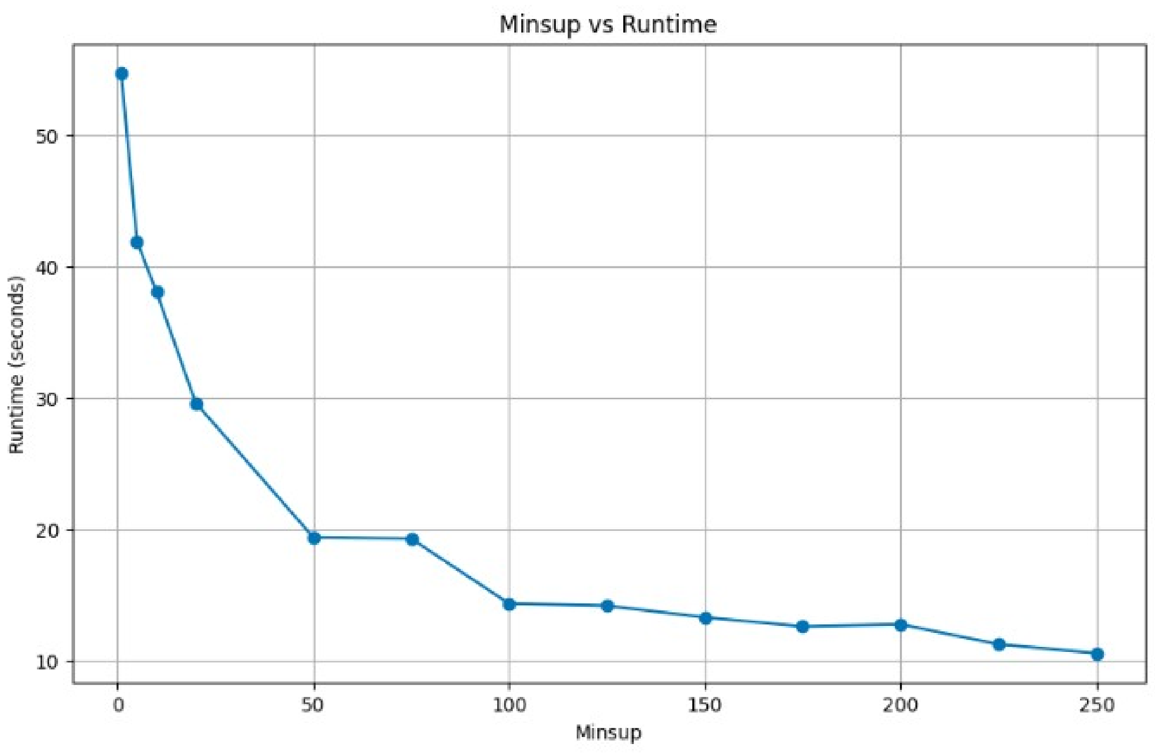
Conclusion

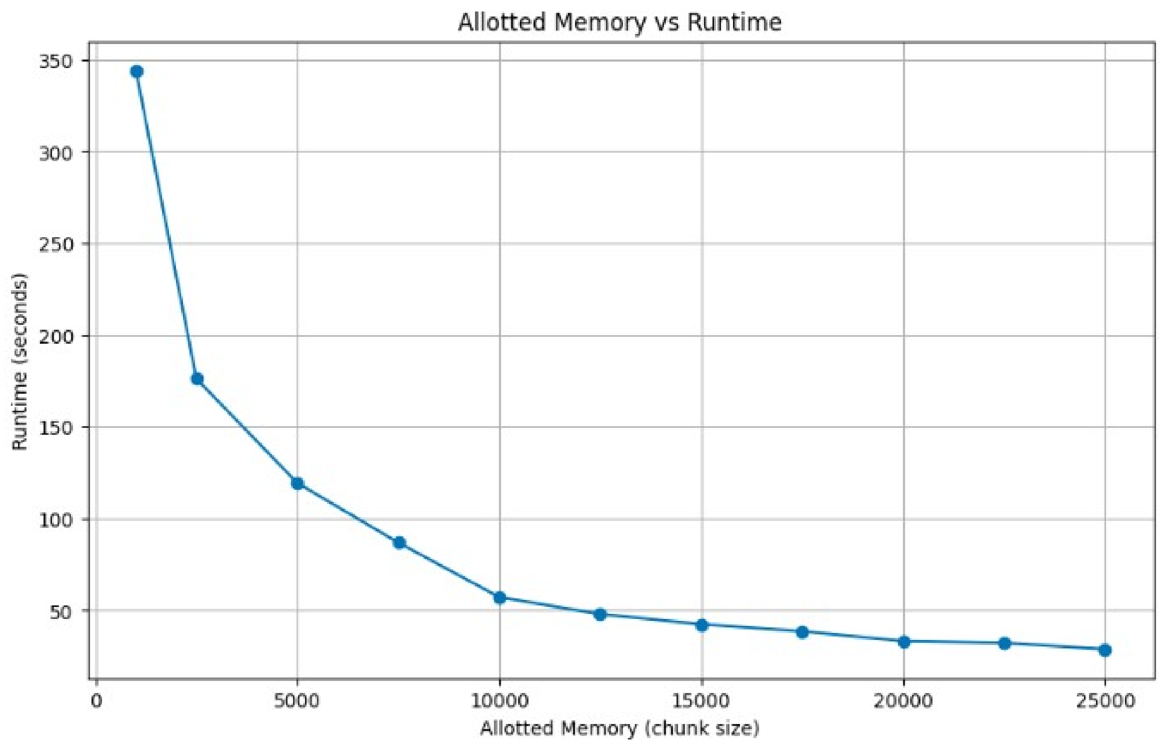
Both implementations of the BUC algorithm are effective for different scenarios:

- **In-Memory Implementation**: Suitable for datasets that fit into main memory, providing faster execution with straightforward processing.
- **Out-of-Memory Implementation**: Designed for larger datasets that exceed memory limits, using paging to handle data efficiently and prevent memory overflow.

The execution times for both approaches provide insights into their performance under different conditions, helping to select the appropriate method based on data size and system constraints.

Based on the performance analysis:





1. **Minsup vs Runtime:**As the minimum support (minsup) increases, the runtime typically decreases. This is because higher minsup values lead to more aggressive pruning, which reduces the number of partitions that need to be processed.

2. **Memory vs Runtime:**As the allocated memory increases(by increasing chunk size), the runtime generally decreases until a certain point. Larger memory allocations allow more data to be processed in memory, reducing the need for frequent I/O operations. However, once the chunk size is large enough to fit most of the data in memory, further increases may not significantly reduce runtime.

Optimization Technique:

The performance of the BUC algorithm is sensitive to the order in which dimensions are processed. By dynamically ordering the dimensions based on their discriminating power, we can enhance the pruning efficiency of the algorithm. This optimization seeks to order dimensions in such a way that the most discriminating (i.e., those that result in the smallest partitions) dimensions are processed first, maximizing the chances of early pruning and reducing the computational load.

Dynamic Dimension Ordering:

1. Dimension Cardinality: Higher cardinality dimensions should be processed earlier as they are more likely to result in smaller partitions, leading to better pruning opportunities.
2. Skewness: Dimensions with uniform distribution (less skew) are preferred early since they distribute data more evenly across partitions.

Impact of the Optimization

1. Early Pruning: By processing the most discriminating dimensions first, the algorithm can prune large sections of the search space early, leading to significant performance improvements. This reduces the number of partitions that need to be further processed.
2. Reduced Computation: Since more irrelevant or less frequent partitions are eliminated early on, the overall number of operations required for aggregation decreases, speeding up the computation.
3. Impact on Real Datasets: On real datasets where some dimensions are highly skewed, dynamic dimension ordering can prevent the algorithm from getting stuck processing large partitions with little pruning, improving performance in such cases.

Performance Comparison

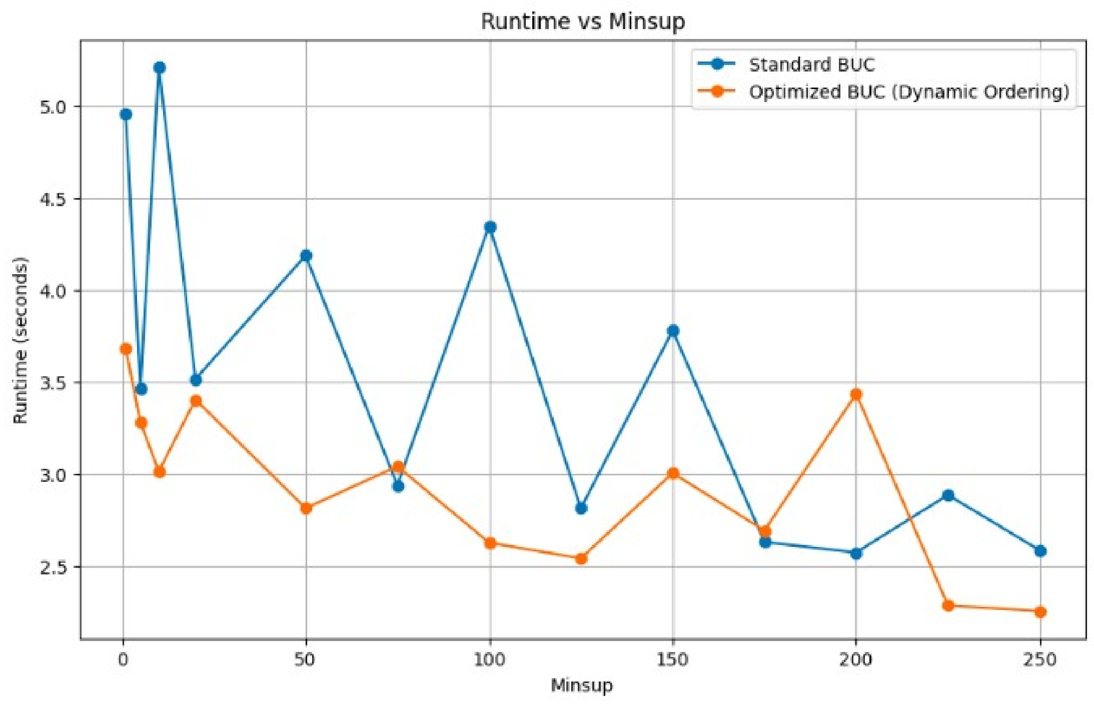
You can compare the runtime and memory usage between the standard BUC and the optimized dynamic BUC using various datasets and configurations to quantitatively assess the impact of this optimization. In typical scenarios, the dynamic BUC should show noticeable improvements, especially on datasets with varied cardinalities and skewness across dimensions. This optimization is crucial when dealing with large and complex datasets, as it directly tackles the primary bottlenecks in the original BUC algorithm.

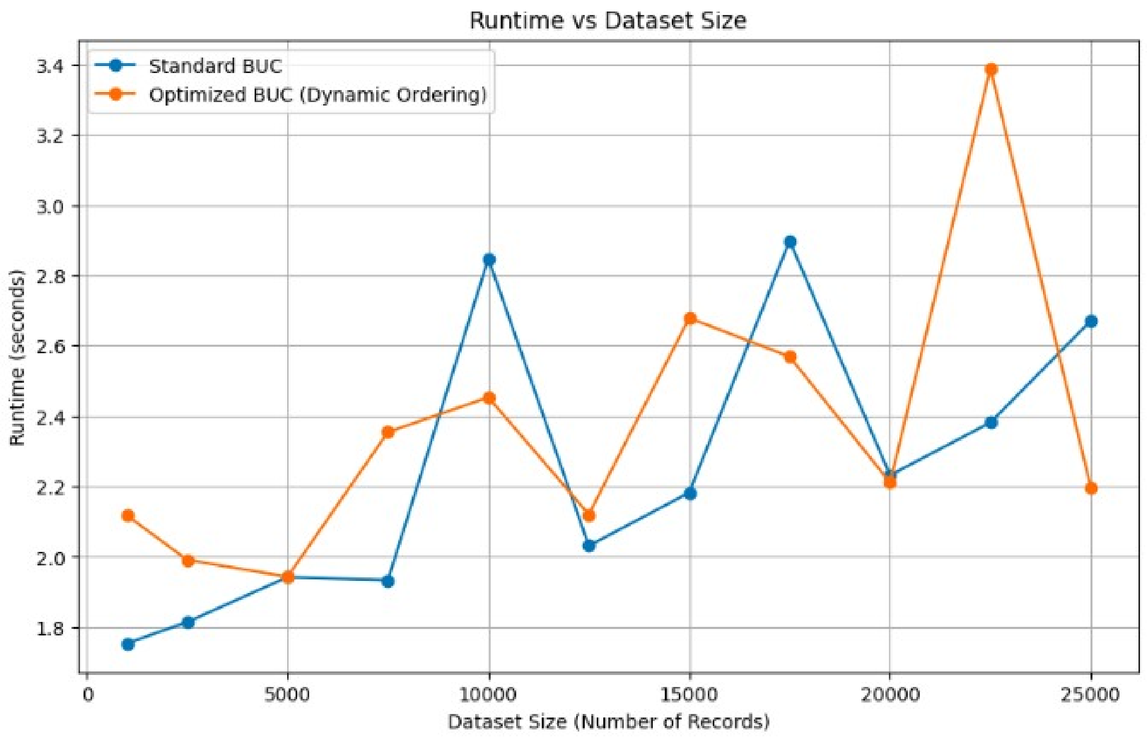
To compare the performance of the optimized BUC algorithm (using dynamic dimension ordering) with the standard BUC algorithm, we can run both algorithms on the same dataset and plot the following graphs:

1. **Runtime vs. Minsup** : This shows how the runtime changesas the minimum support (minsup) varies.

2. **Runtime vs. Dataset Size** : Compares how the runtimescales as the size of the dataset increases. After running the code, the following graphs are generated:

After running the code, the following graphs are generated:





1**. Runtime vs. Minsup** : This graph will show how both the standard and optimized BUC algorithms perform as the minimum support value changes. The optimized BUC should demonstrate faster runtimes due to better pruning when higher cardinality dimensions are processed first.

2. **Runtime vs. Dataset Size** : This graph will illustrate how both algorithms scale as the dataset size increases. The optimized BUC is expected to outperform the standard BUC, especially on larger datasets.

Comparison of BUC and AOI

Both the Bottom-Up Cube (BUC) algorithm and Attribute-Oriented Induction (AOI) are techniques used in data analysis and mining, but they serve different purposes and are suitable for different types of tasks. Below is a detailed comparison based on various criteria.

a. The Primary Purposes and Use Cases of Each Technique

BUC Algorithm:

● Purpose:BUC is designed to efficiently compute datacubes, particularly sparse and iceberg cubes, which involve aggregating data across multiple dimensions.

● Use Cases:It is commonly used in OLAP (Online AnalyticalProcessing) environments where multidimensional analysis is required, such as in sales analysis, inventory management, and financial reporting.PAOI (Attribute-Oriented Induction):

● Purpose:AOI is a data generalization method usedto extract characteristic rules by abstracting specific values into more generalized concepts. It is often used in data summarization and concept description.

● Use Cases:AOI is typically applied in data miningtasks such as classification, concept description, and knowledge discovery, where the goal is to derive high-level patterns or rules from the data.

b. The Types of Insights or Patterns Each Method is Best Suited to Discover

BUC Algorithm:

● Insights/Patterns:BUC is well-suited to discover detailed aggregate patterns across multiple dimensions. For example, it can identify total sales per product category, region, and time period.

● Example:In the context of the electric vehicle dataset, BUC can be used to compute the total number of vehicles sold for each combination of Make, Model, and Electric Vehicle Type.

AOI:

● Insights/Patterns:AOI is best suited to uncover generalized patterns or characteristic rules that describe the overall structure of the data. It simplifies the data by removing less relevant attributes or generalizing them.

● Example:Using AOI on the electric vehicle dataset might result in rules like "`Popular EV Brands”` (e.g.,`'TESLA', 'NISSAN', 'CHEVROLET'`) are primarily associated with `Battery Electric Vehicle type`

c. The Computational Efficiency and Scalability of Each Approach

BUC Algorithm:

● Efficiency:BUC is computationally intensive, especially with a large number of dimensions, as it computes aggregates for every possible combination of dimensions.

● Scalability:While BUC can handle large datasets,its performance can degrade rapidly as the number of dimensions increases due to the exponential growth in the number of group-bys.

● Optimization:Techniques like dynamic dimension or deringand iceberg cubing can improve BUC's efficiency by reducing unnecessary computations.

AOI:

● Efficiency:AOI is generally more efficient than BUC as it focuses on generalizing data rather than computing all possible aggregates. The complexity of AOI primarily depends on the number of attributes and the levels of generalization.

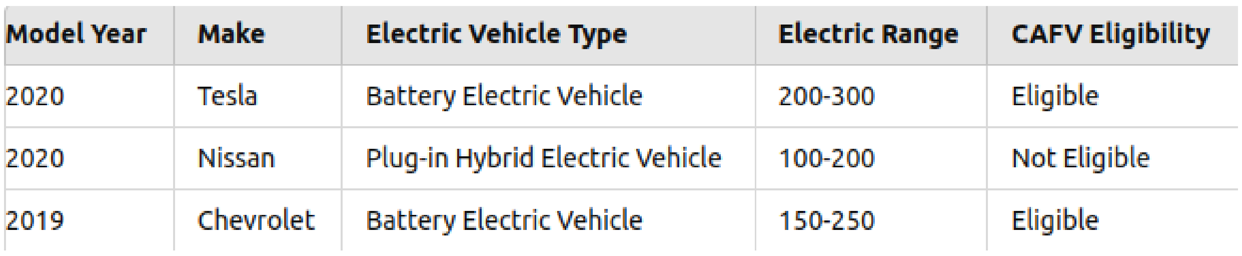
● Scalability:AOI is highly scalable, especially in scenarios where the dataset contains a large number of specific values that can be generalized into fewer categories.

d. The Interpretability of the Results Produced by Each Method

BUC Algorithm:

● Interpretability:The results of BUC are highly detailed and require a good understanding of the underlying data structure. Each result represents a specific aggregate value for a particular combination of dimensions.

● Example:



Model Year	Make	Electric Vehicle Type	Electric Range	CAFV Eligibility
2020	Tesla	Battery Electric Vehicle	200-300	Eligible
2020	Nissan	Plug-in Hybrid Electric Vehicle	100-200	Not Eligible
2019	Chevrolet	Battery Electric Vehicle	150-250	Eligible

AOI:

● Interpretability:The results of AOI are typicallymore abstract and easier to interpret, especially for non-technical users. The generalization makes the patterns more comprehensible by reducing the complexity of the data.

● Example:AOI might produce a rule like "New popular EV brands are having the most sales"

e. Scenarios Where One Method Might Be Preferable Over the Other

BUC Algorithm:

● Preferred Scenario:BUC is preferable when detailed multidimensional analysis is required, particularly when you need to understand the interaction between multiple dimensions at a granular level.

● Example:An analyst looking to explore how different combinations of Make, Model, and Electric Vehicle Type influence sales volume would benefit from using BUC.

AOI:

- Preferred Scenario:AOI is more suitable when the goal is to extract high-level, generalized insights or when the dataset is too large or complex to analyze at a detailed level.
- Example:If the goal is to identify broad trends in electric vehicle adoption across different periods and simplify the data, AOI would be more effective.