**APRIORI ALGORITHM :**

* **Step 1:** 
  + Create a list of all the elements that appear in every transaction and create a frequency table.
* **Step 2:** 
  + Set the minimum level of support. Only those elements whose support exceeds or equals the threshold support are significant.
* **Step 3:** 
  + All potential pairings of important elements must be made, bearing in mind that AB and BA are interchangeable.
* **Step 4:** 
  + Tally the number of times each pair appears in a transaction.
* **Step 5:** 
  + Only those sets of data that meet the criterion of support are significant.
* **Step 6:** 
  + Now, suppose you want to find a set of three things that may be bought together. A rule, known as self-join, is needed to build a three-item set. The item pairings OP, OB, PB, and PM state that two combinations with the same initial letter are sought from these sets.
  + OPB is the result of OP and OB.
  + PBM is the result of PB and PM.
* **Step 7:** 
  + When the threshold criterion is applied again, you'll get the significant itemset.

**FP GROWTH ALGORITHM :**

* **Data Preprocessing:**
  + Load your transactional dataset, which consists of transactions where each transaction contains a list of items.
* **Initialization:** 
  + Scan the dataset to count the frequency of each item (itemset of size 1).Eliminate infrequent items that do not meet the minimum support threshold.
  + Create a list of frequent 1-itemsets.
* **Iterative Process:** 
  + Generate candidate itemsets of size (k+1) from frequent itemsets of size k.
  + Prune candidates that contain infrequent subsets.
  + Count the support for each candidate itemset by scanning the dataset.
* **Frequent Itemset Generation:** 
  + Select candidate itemsets with support greater than or equal to the minimum support threshold to become frequent itemsets of size (k+1).
  + The frequent itemsets of size (k+1) are used as input for the next iteration.
* **Repeat Step 3 and Step 4:** 
  + Continue generating candidate itemsets, pruning, and counting support until no more frequent itemsets can be found.Each iteration finds frequent itemsets of larger sizes, starting from 2-itemsets, 3-itemsets, and so on.
* **Association Rule Generation:** 
  + Once all frequent itemsets are found, association rules can be generated.
  + For each frequent itemset, generate all possible non-empty subsets.
  + Calculate the confidence for each rule and keep those that satisfy the minimum confidence threshold.
* **Output Frequent Itemsets and Association Rules:** 
  + The algorithm returns a list of frequent itemsets and a list of association rules that meet the minimum support and minimum confidence thresholds, respectively.
* **Visualization and Interpretation (optional):** 
  + Visualize and interpret the discovered frequent itemsets and association rules to gain insights into patterns and relationships within the dataset.

**PAGE RANK ALGORITHM :**

* **Initialization:** 
  + Assign an initial PageRank score to each web page. Commonly, you can start with equal scores for all pages (e.g., 1/N, where N is the number of pages).
* **Iterative Computation:**
  + Perform a series of iterations until convergence. In each iteration:
  + For each web page A, calculate its new PageRank score based on the scores of pages linking to A. The score of a page is divided among the pages it links to.
  + A page's PageRank is the sum of a fraction of the PageRank of each linking page, divided by the number of outbound links on the linking page. This fraction is often referred to as the "damping factor" (typically set to around 0.85) and can be seen as the probability of a random web surfer following a link.
  + Mathematically, the PageRank of page A in the current iteration (PR(A)) is calculated as:
    - PR(A) = (1 - d) / N + d \* Σ (PR(B) / L(B))
      * where:
      * d is the damping factor (typically around 0.85).
      * N is the total number of pages.
      * Σ represents the sum over all pages B that link to page A.
      * PR(B) is the PageRank of page B.
      * L(B) is the number of outbound links on page B.
* **Convergence Check:** 
  + Check for convergence by comparing the PageRank scores from the current iteration with the scores from the previous iteration. If the scores have converged or a predefined number of iterations have been reached, stop the process.
* **Normalization:** 
  + Optionally, you can normalize the PageRank scores so that they sum up to 1.
* **Output:** 
  + The final PageRank scores represent the importance of each web page.

**OLAP CUBE AND OPERATIONS :**

1. OLAP CUBE :

* An OLAP cube, also known as a multidimensional cube or hypercube, is a data structure that stores and manages multidimensional data in a compact and efficient manner. It allows users to view, analyze, and interact with data from various dimensions, such as time, geography, product categories, and more. The cube is a key component of OLAP systems and is typically used to support complex analytical queries.
* Key characteristics of an OLAP cube include:
  + **Dimensions:** 
    - These are the attributes by which data can be categorized or sliced. Common dimensions include time, geography, products, and customers.
  + **Measures:** 
    - Measures are the numeric values or facts that are being analyzed. Examples include sales revenue, profit, or quantities sold.
  + **Hierarchies:**
    - Dimensions can be organized in hierarchies, which define levels of granularity. For instance, a time dimension can have hierarchies like year, quarter, month, and day.
  + **Cells:** 
    - Each cell in the cube represents the intersection of specific dimension members. It stores a measure value or aggregated data.

1. OLAP OPERATIONS

* **Slice:** 
  + Slicing an OLAP cube involves selecting a single dimension from the cube to view a "slice" of data. For example, you can slice a sales cube by the "time" dimension to see sales data for a specific period.
* **Dice:** 
  + Dicing involves selecting two or more dimensions to view a subset of the cube. This allows users to examine data from various perspectives. For instance, you can dice the cube to view sales data for a specific product category and time period.
* **Roll-Up (Drill-Up):** 
  + Roll-up operations involve moving up the hierarchy in a dimension to see data at a higher level of aggregation. For example, you can drill up from monthly sales data to quarterly or annual sales.
* **Drill-Down (Drill-Through):** 
  + Drill-down operations involve moving down the hierarchy to see data at a more detailed level. For example, you can drill down from annual sales data to monthly or daily sales.
* **Pivot:** 
  + Pivoting allows users to switch dimensions. You can pivot the view of the data to analyze it from a different angle.
* **Rotate:** 
  + Rotation allows users to change the orientation of the cube, effectively changing the way dimensions are displayed.
* **Consolidation:** 
  + Consolidation operations are used to aggregate data across dimensions, helping to provide an overview of data from multiple dimensions.

**STAR SCHEMA:**

* Centralized Fact Table: In a Star Schema, there is one centralized fact table at the center of the schema. The fact table typically contains measures or metrics (e.g., sales revenue) and foreign keys to dimension tables.
* Dimension Tables: Surrounding the fact table are dimension tables. Dimension tables contain descriptive data and attributes. Each dimension table is related to the fact table through foreign keys.
* Denormalized: In a Star Schema, dimension tables are often denormalized, meaning they might contain redundant data for the sake of faster query performance. Denormalization simplifies queries and reduces the number of joins.
* Simplicity: Star schemas are easy to understand, query, and maintain. They are well-suited for ad-hoc reporting and data warehousing tools that require straightforward querying.
* Performance: Star schemas are known for their fast query performance. Since dimension tables are denormalized, there are fewer joins, making queries more efficient.
* Suitable for Data Warehouses: Star schemas are commonly used in data warehousing and business intelligence environments, where quick and intuitive access to data is essential.

**SNOWFLAKE SCHEMA :**

* Normalized Dimension Tables: In a Snowflake Schema, dimension tables are normalized, which means data is organized efficiently to minimize redundancy. This leads to more tables, smaller tables, and complex relationships.
* Hierarchical Structure: Dimension tables in a Snowflake Schema may have a more hierarchical structure due to normalization. For example, a region dimension might have separate tables for countries, states, and cities.
* Complexity: Snowflake schemas are more complex than star schemas due to the normalization of dimension tables. This complexity can make queries and data retrieval more complicated.
* Storage Efficiency: Snowflake schemas can be more storage-efficient because of the normalization. Redundant data is reduced, and storage space may be conserved.
* Maintainability: Snowflake schemas can be harder to maintain because of their complex structure. Updates and changes may require more effort.
* Suitable for Complex Data Structures: Snowflake schemas are useful when dealing with data sources that have intricate, hierarchical relationships and when storage space is a significant concern.