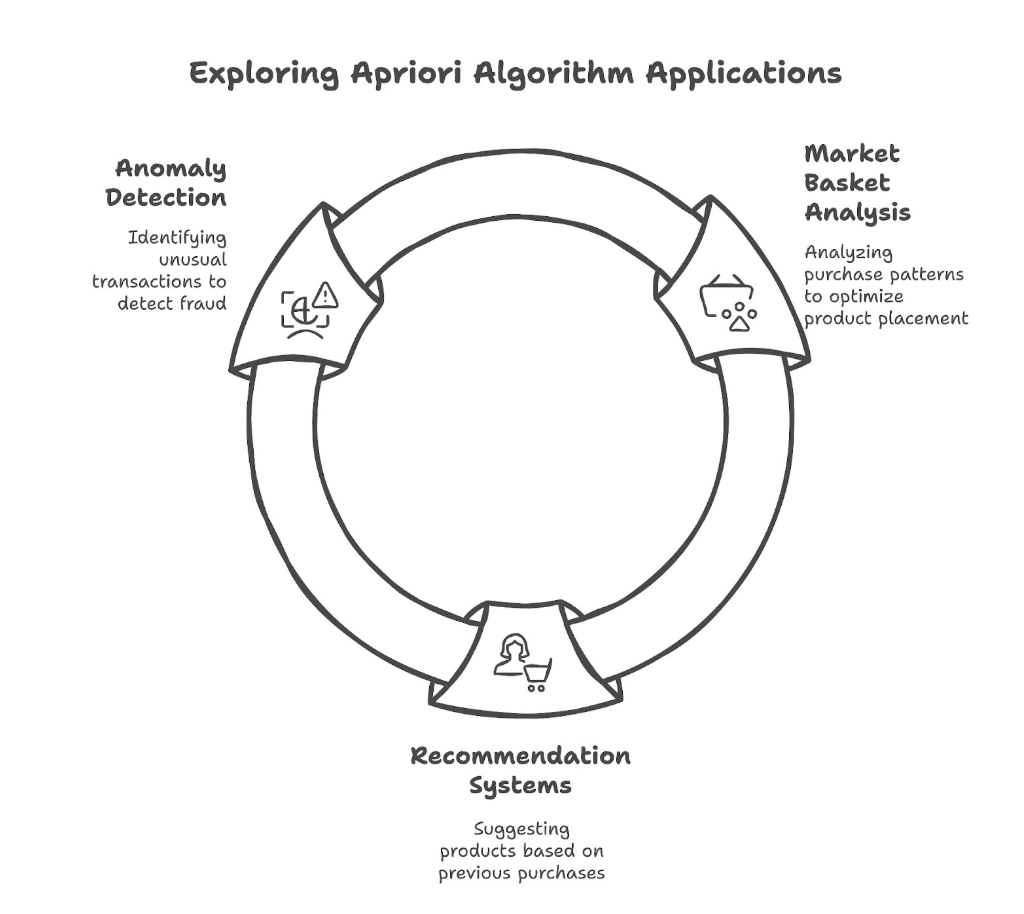
**1. Introduction**

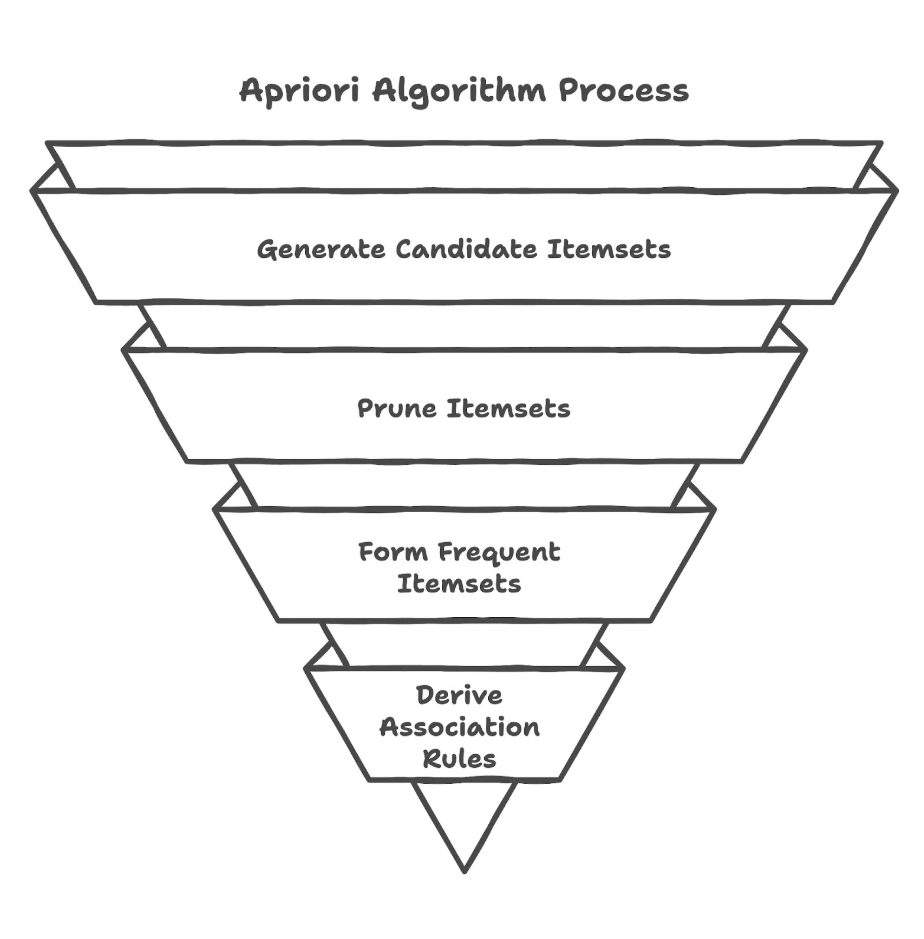
This project introduces an interactive, web-based application that leverages association rule mining to understand digital banking service usage. By analyzing patterns in how customers use banking services in combination, banks can better tailor their offerings, improve digital experiences, and design marketing campaigns. The tool is built using Python, Streamlit, Plotly, and the Apriori algorithm, and it supports visualization and download of results in a user-friendly interface.



**2. Objective**

The main goals of this project include:

* Detecting patterns in customer interactions with multiple banking services.
* Revealing service combinations that frequently occur together.
* Providing insights into sequential service usage (e.g., if a customer uses X, how likely are they to use Y?).
* Empowering business users with intuitive visual reports and actionable data-driven recommendations.



**3. Methodology**

**Background: Apriori Algorithm - Mathematical Logic**

The Apriori algorithm is a classic algorithm in data mining for learning association rules. It is based on the principle that if an itemset is frequent, then all of its subsets must also be frequent. The algorithm operates on transactional datasets and applies a level-wise search for frequent itemsets.

**Key Concepts:**

* **Itemset**: A group of one or more items (in this case, banking services).
* **Support**: The proportion of transactions in which an itemset appears.

Support(A) = count(A) / Total Transactions

* **Confidence**: The likelihood that itemset B appears in a transaction that contains itemset A.

Confidence(A → B) = Support(A ∪ B) / Support(A)

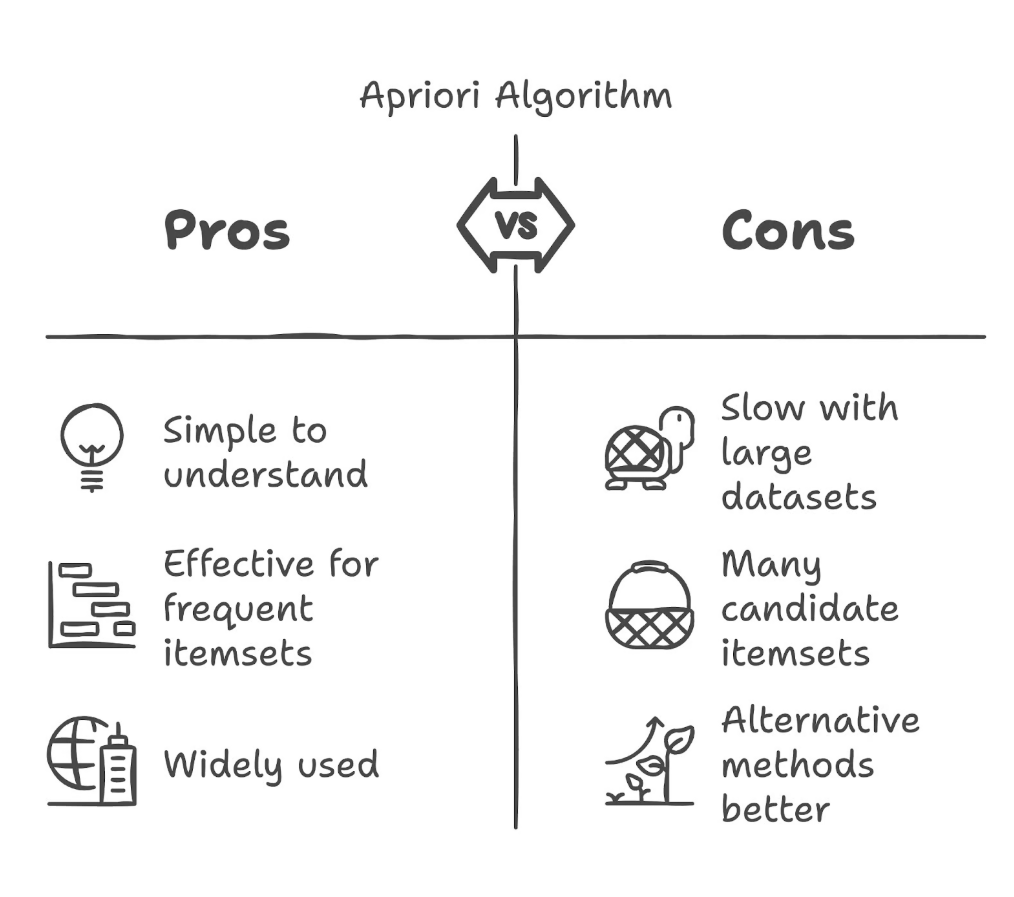
* **Lift**: The ratio of the observed support to that expected if A and B were independent.

Lift(A → B) = Confidence(A → B) / Support(B)

**Steps in the Apriori Algorithm:**

1. **Generate Candidate Itemsets**: Begin with single-item sets. Calculate their support. Retain only those that meet the min\_support threshold.
2. **Iterative Expansion**: Combine frequent itemsets of size *k* to generate candidates of size *k+1*. Prune those with infrequent subsets.
3. **Rule Generation**: For each frequent itemset, generate all non-empty subsets and compute confidence and lift for potential rules.
4. **Prune Rules**: Retain only those rules that satisfy min\_confidence and min\_lift.

This algorithm is well-suited for understanding customer behavior in digital banking, as it reveals combinations and directional relationships between service usages.



**3.1 Data Understanding**

The data must be structured as a session-based service log in CSV format. Each row corresponds to one banking session, with a single column named services, containing a comma-separated list of service identifiers. This simulates a market basket scenario adapted to digital banking.

Example rows:

services

"bill\_payment,fund\_transfer,balance\_inquiry"

"qr\_payment,mobile\_topup"

The DataLoader class is responsible for validating the file format and structure.

**3.2 Data Exploration**

Exploratory analysis is performed visually through the Streamlit interface. Two primary visualizations are used:

* **Bar Chart of Frequencies**: Shows how often each service is used across all sessions. This helps identify core vs. peripheral banking features.
* **Network Graph**: Illustrates services that frequently co-occur. Edges are weighted by the confidence of association rules, allowing quick recognition of strong dependencies.  
  These visual tools enable non-technical stakeholders to understand trends quickly.

**3.3 Data Cleaning**

Robust data validation and cleaning ensure quality input. Performed in loader.py, key steps include:

* Ensuring the column services exists.
* Dropping missing or invalid rows.
* Removing leading/trailing spaces and quotes from service names.
* Splitting strings into service lists for use in analysis.
* Transforming long format (transaction + item) into basket format (if necessary).

Clean data ensures reliable pattern detection and prevents bias due to formatting errors.

**3.4 Model Development**

Two analysis models are supported:

**1. AprioriAnalyzer (from apriori.py)**:

* Uses mlxtend's apriori and association\_rules functions.
* Input is one-hot encoded transaction matrix.
* Outputs frequent itemsets and association rules.

**2. BankingPatternAnalysis (custom)**:

* Implements Apriori logic manually with better control.
* Uses Python set operations and custom thresholding.
* Allows fine-tuned filtering based on support, confidence, and lift.

Both methods are modular and can be extended. Users define min\_support and min\_confidence via UI sliders.

**3.5 Model Evaluation**

Evaluation is embedded into the interpretation of metrics:

* **Support**: How often a combination appears across all sessions.
* **Confidence**: Given a service is used, how likely is the next one.
* **Lift**: How much stronger the rule is compared to random occurrence.

Example:  
If support = 0.25 and confidence = 0.75 for A → B, then 25% of sessions contain A & B, and 75% of sessions that contain A also include B.

These metrics are shown in formatted tables for clarity.

**3.6 Improvement on the Model**

Suggested model and system improvements include:

* **Alternative Algorithms**: Use FP-Growth for faster computation on larger datasets.
* **Session Metadata Integration**: Include customer demographics, channel type, or time to improve segmentation.
* **Temporal Pattern Mining**: Analyze service sequences over time.
* **Real-time Processing**: Integrate with stream processing frameworks for live dashboards.
* **Evaluation Metrics**: Introduce precision, recall, and F1-score for rule evaluation, especially in supervised tasks.

**3.7 Final Result**

The final deliverables of the application are:

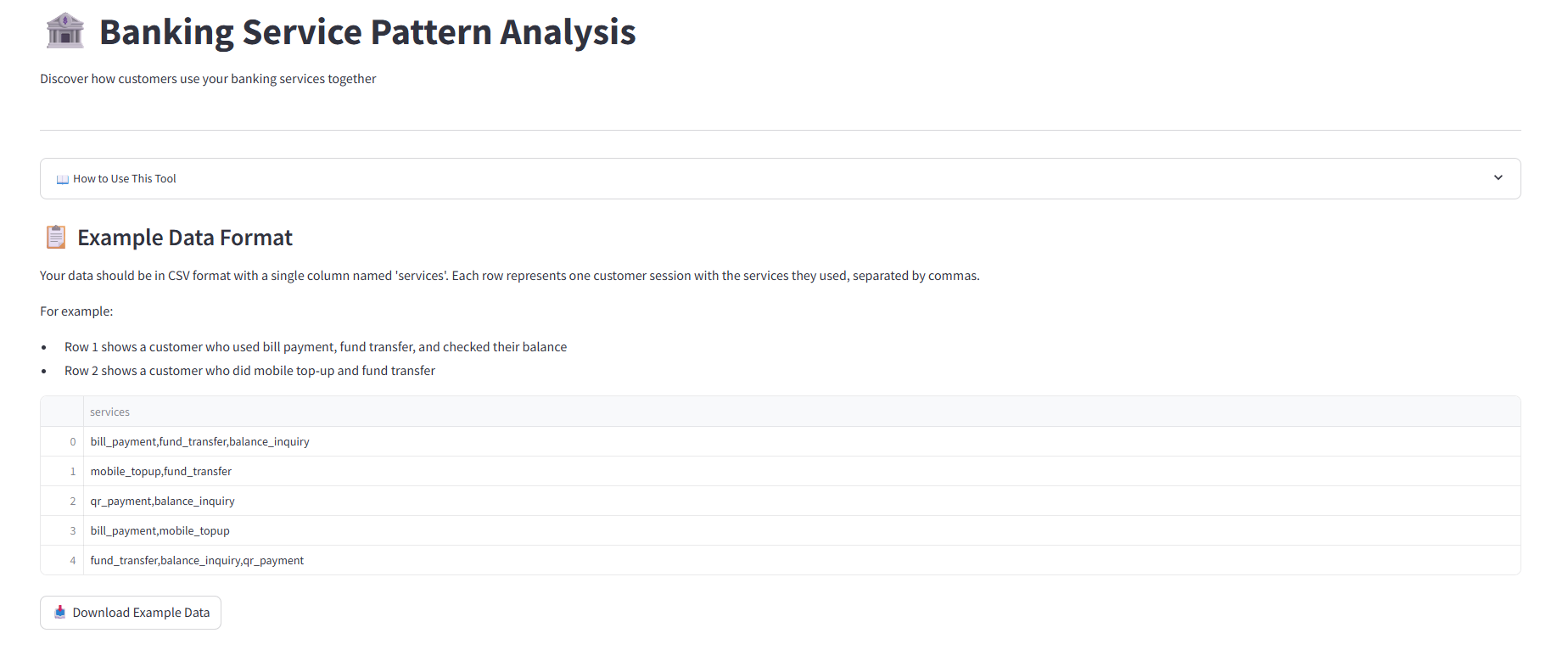
* **Interactive UI**: A Streamlit dashboard for uploading data, selecting thresholds, and triggering analysis.
* **Visual Outputs**:
  + Frequency plots of services
  + Network graphs showing related services
* **Downloadable Results**:
  + CSV of frequent itemsets with support
  + CSV of association rules with confidence and lift
* **Example Dataset**: Built-in sample for testing and demonstration.

These features ensure the tool is accessible and insightful to both analysts and business teams.

**5. Guideline**

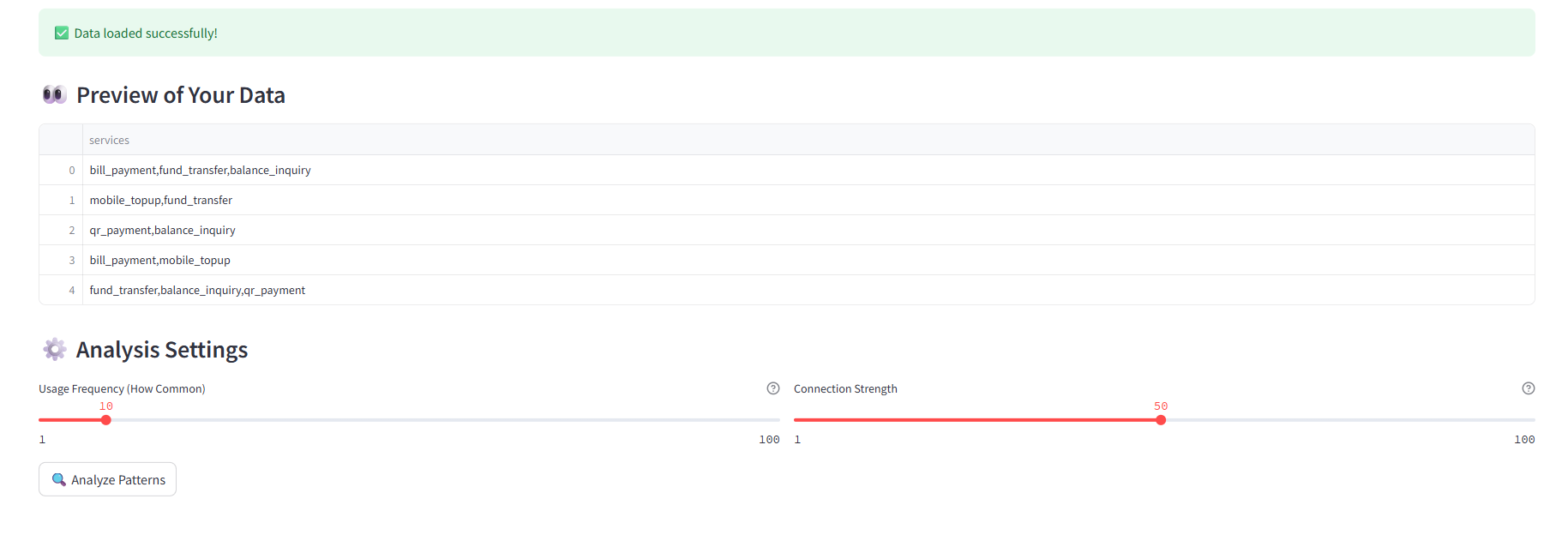
**5.1 Uploading Data**

* The platform expects a CSV file with one column named services.
* Each row is a session showing what services were used together.
* You can download a sample CSV using the "Download Example Data" button.



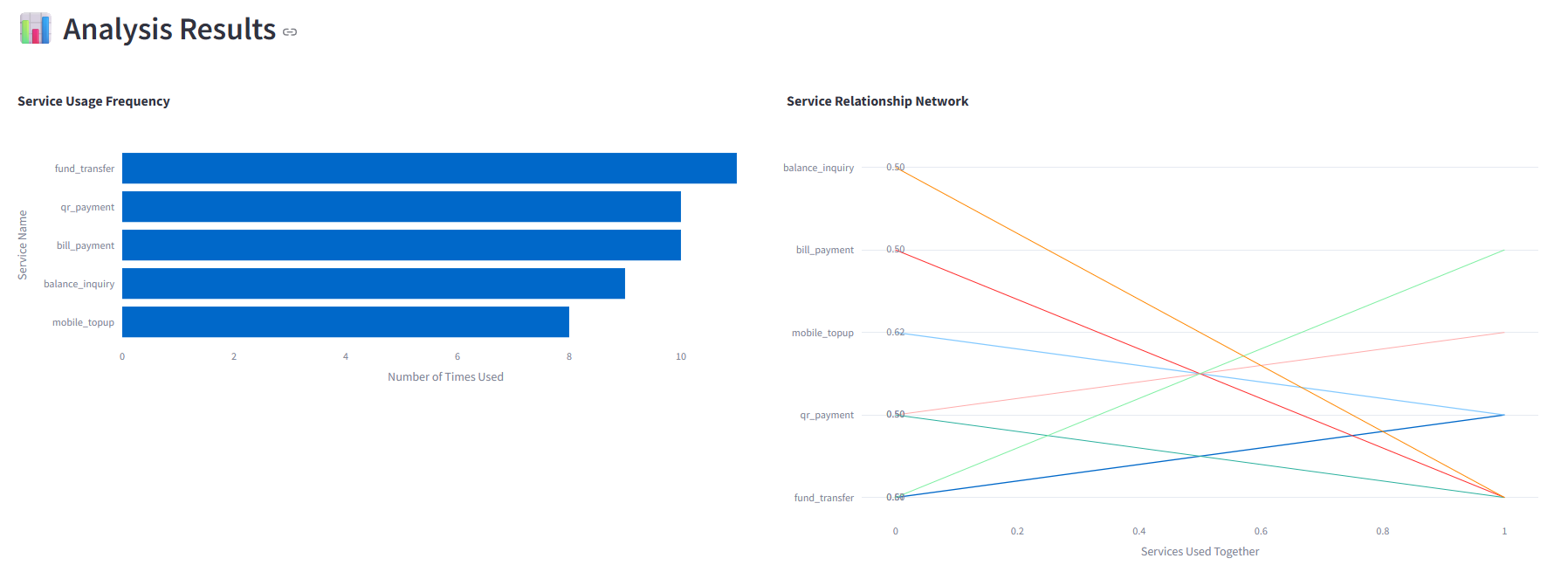
**5.2 Previewing and Setting Parameters**

* After loading data, a preview table appears.
* Use sliders to configure:
  + **Usage Frequency (Support)**: Minimum % of sessions a pattern must appear in.
  + **Connection Strength (Confidence)**: Likelihood that one service leads to another.
* Click **Analyze Patterns** to run the analysis.



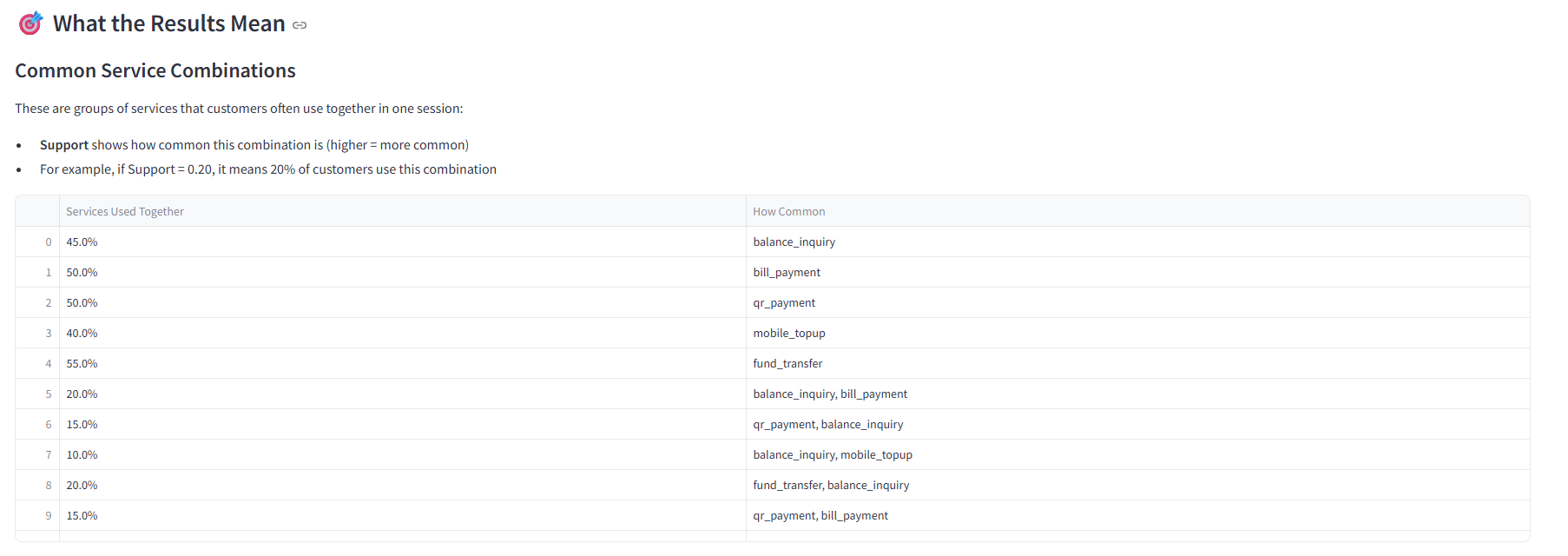
**5.3 Viewing Analysis Results**

* The bar chart shows how often each service is used.
* The network graph highlights the strength of relationships between services.
* Hover over edges to see correlation values.



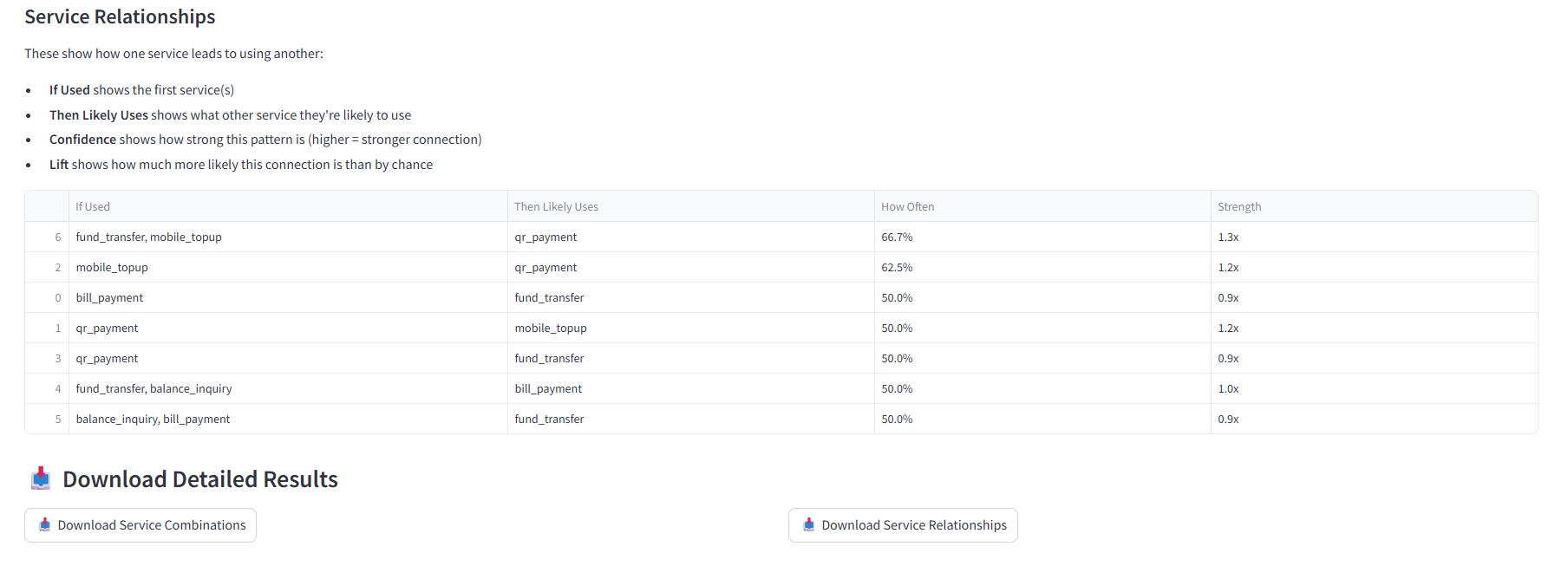
**5.4 Interpreting Frequent Combinations**

* This table lists service sets used together frequently.
* Support shows how often the group appears across sessions.
* Helps identify natural service bundles.



**5.5 Understanding Service Relationships**

* This section shows rules like "If A is used, B is likely to follow."
* Key metrics:
  + **Confidence**: How often the rule is true.
  + **Lift**: How much more likely the relationship is than by chance.
* You can download both combination and rule results as CSV.



These steps guide the user through the entire flow: from loading data to interpreting patterns and exporting results.

**6. Conclusion**

In conclusion, this project delivers a modular, user-friendly application for pattern mining in digital banking. It demonstrates how association rule mining, when combined with good UI/UX and visualization, can empower banks to make informed decisions. Its extendable architecture supports scaling and future enhancements, making it a solid base for customer behavior analytics in financial services.