RETAIL TRANSACTION OPTIMIZATION USING ASSOCIATION RULE MINING

225505N - Anujan N
CM3720 - Machine Learning
Department of Computational Mathematics
University of Moratuwa

PRESENTATION OUTLINE

- Problem Definition
- Proposed Solution
- Dataset Overview
- Data Preprocessing & Cleaning
- Feature Engineering
- Exploratory Data Analysis
- Customer Segmentation
- Hyperparameter Tuning
- Model Evaluation Metrics

PROBLEM DEFINITION

Local Rice & Curry shops face operational challenges:

- Food waste from unpopular dishes
- Inefficient item layout causing long queues
- Inventory shortages and missed revenue
- No data-driven insights for optimization

PROPOSED SOLUTION

Apply Association Rule Mining using Apriori Algorithm to discover purchasing patterns.

Approach

- Market Basket Analysis to find item associations
- Customer Segmentation using K-Means Clustering
- Segmented Apriori for cluster-specific pattern

Benefits

- Optimize layout and reduce customer wait time
- Improve inventory management
- Increase sales through data-driven decisions

DATASET OVERVIEW

Synthetic dataset representing 5,076 customer transactions over 7 days.

Dataset Structure:

- Transaction_ID: Unique identifier (T0001 - T5076)
- Items: Comma-separated list of dishes
- Time_Stamp: Purchase time (11:00 AM - 2:30 PM)

Menu Categories:

- Base Starches: Red Rice, White Rice, Fried Rice, String Hoppers
- Curries: Dhal, Pumpkin, Jackfruit, Potato, Mallum
- Proteins: Chicken, Fish, Egg variants
- Condiments: Pol Sambol, Chilli Paste, Papadam

DATASET PREVIEW

```
Transaction_ID,Items,Time_Stamp      You, 6 hours ago • initial commit …
T0001,"Red Rice, Coconut Sambol, Chicken, Pol Sambol",2025-11-16 11:00 AM
T0002,"Fried Rice, Chilli Paste, Gotukola",2025-11-16 11:00 AM
T0003,"Red Rice, Chilli Paste, Dried Fish, Mallum, Fried Egg",2025-11-16 11:00 AM
T0004,"Red Rice, Chicken, Fried Egg, Pumpkin",2025-11-16 11:00 AM
T0005,"String Hoppers, Devilled Chicken, Chicken",2025-11-16 11:00 AM
T0006,"String Hoppers, Fried Egg, Potato Tempered, Dhal, Chicken Curry, Devilled Chicken",2025-11-16 11:00 AM
T0007,"String Hoppers, Gotukola, Brinjal Moju, Kiri Hodi, Papadam, Devilled Chicken",2025-11-16 11:01 AM
T0008,"Fried Rice, Fried Fish, Jackfruit, Fried Fish",2025-11-16 11:01 AM
T0009,"White Rice, Chicken Curry, Fried Egg, Chicken",2025-11-16 11:01 AM
T0010,"String Hoppers, Dhal, Brinjal Moju",2025-11-16 11:02 AM
T0011,"White Rice, Chop Suey, Dhal, Chicken Curry",2025-11-16 11:02 AM
T0012,"Fried Rice, Fried Egg, Fish, Pol Sambol, Dried Fish",2025-11-16 11:02 AM
T0013,"White Rice, Fried Fish, Chilli Paste, Dried Fish, Potato Tempered, Dhal",2025-11-16 11:02 AM
T0014,"Fried Rice, Fried Fish, Potato Tempered, Brinjal Moju, Jackfruit",2025-11-16 11:03 AM
```

DATA PREPROCESSING: MISSING VALUES

Missing Value Analysis:

- Transaction_ID: 0 missing
- Items: 18 missing (critical data)
- Time_Stamp: 12 missing

Handling Strategy:

- Dropped 18 rows with missing Items (cannot analyze empty baskets)
- Forward-filled 12 missing Time_Stamp values
- Final dataset: 5,058 clean transactions

DATA PREPROCESSING: MISSING VALUES

```
df_cleaned = df.dropna(subset=['Items'])
df_cleaned['Time_Stamp'] = df_cleaned['Time_Stamp'].fillna(method='ffill')
print(f"Rows dropped: {len(df) - len(df_cleaned)}")
print(f"Final shape: {df_cleaned.shape}")
```

✓ 0.0s

```
Rows dropped: 50
Final shape: (5026, 3)
```

DATA STANDARDIZATION

Synonym Merging:

- "Fried Fish", "Fish Ambul Thiyal", "Dried Fish" -> "Fish"
- "Chicken Curry" -> "Chicken"
- "pol_sambol", "Coconut Sambol" -> "Pol Sambol

Transformation:

- Converted Items string to lists
- - Applied standardization mapping
- One-Hot Encoding using TransactionEncoder
- Result: Binary matrix (5058 x 15 items)

```
df_cleaned['Items_List'] = df_cleaned['Items'].str.split(', ')

item_mapping = {
    'Fried Fish': 'Fish', 'Fish Ambul Thiyal': 'Fish', 'Dried Fish': 'Fish',
    'Chicken Curry': 'Chicken', 'pol_sambol': 'Pol Sambol',
    'Coconut Sambol': 'Pol Sambol'
}

df_cleaned['Items_Standardized'] = df_cleaned['Items_List'].apply(
    lambda items: [item_mapping.get(item.strip(), item.strip()) for item in items] if items else []
)

print(f"Standardization complete. Sample:")
print(df_cleaned['Items_Standardized'].head(3).tolist())
```

✓ 0.0s

```
Standardization complete. Sample:
[['Red Rice', 'Pol Sambol', 'Chicken', 'Pol Sambol'], ['Fried Rice', 'Chilli Paste', 'Gotukola'], ['Red Ric
```

FEATURE ENGINEERING

1. Base_Starch Classification:
    - Rice-Based: Red Rice, White Rice, Fried Rice
    - Noodle-Based: String Hoppers
    - Other: No primary starch

```python
rice_based = ['Red Rice', 'White Rice', 'Fried Rice']
noodle_based = ['String Hoppers']

def classify_base_starch(items):
    if any(item in rice_based for item in items): return 'Rice-Based'
    if any(item in noodle_based for item in items): return 'Noodle-Based'
    return 'Other'

df_cleaned['Base_Starch'] = df_cleaned['Items_Standardized'].apply(classify_base_starch)
print(f"\nBase Starch Distribution:\n{df_cleaned['Base_Starch'].value_counts()}")
```

```
Base Starch Distribution:
Base_Starch
Rice-Based      3752
Noodle-Based    1274
Name: count, dtype: int64
```

FEATURE ENGINEERING

2. Time_Bin Temporal Analysis:

- Early Lunch: < 12:00 PM
- Peak Lunch: 12:00 PM - 1:00 PM
- Late Lunch: > 1:00 PM

```python
import pandas as pd

df_cleaned['Time_Stamp'] = pd.to_datetime(df_cleaned['Time_Stamp'], format='%Y-%m-%d %I:%M %p')
df_cleaned['Hour'] = df_cleaned['Time_Stamp'].dt.hour

df_cleaned['Time_Bin'] = df_cleaned['Hour'].apply(
    lambda h: 'Early Lunch' if h < 12 else 'Peak Lunch' if h < 13 else 'Late Lunch'
)

print(f"Time Binning:\n{df_cleaned['Time_Bin'].value_counts()}")
```

```
Time Binning:
Time_Bin
Late Lunch     2157
Peak Lunch     1487
Early Lunch    1382
Name: count, dtype: int64
```

FEATURE ENGINEERING

3. Is_Vegetarian Dietary Flag:

- True if no Chicken, Fish, or Egg in transaction

```
non_veg_items = ['Chicken', 'Devilled Chicken', 'Fish', 'Fried Egg']

df_cleaned['Is_Vegetarian'] = df_cleaned['Items_Standardized'].apply(
    lambda items: not any(item in non_veg_items for item in items)
)

print(f"Vegetarian Distribution:\n{df_cleaned['Is_Vegetarian'].value_counts()}")
```

```
Vegetarian Distribution:
Is_Vegetarian
False    4082
True      944
Name: count, dtype: int64
```

CUSTOMER SEGMENTATION (K-MEANS)

Why Segment?

- Different customer groups have different preferences
- Enables targeted association rule mining
- Avoids "averaged" rules that fit nobody well

K-Means Configuration:

- n_clusters = 3 (optimal via elbow method)
- Input: One-hot encoded transaction matrix
- andom_state = 42 (reproducible results)

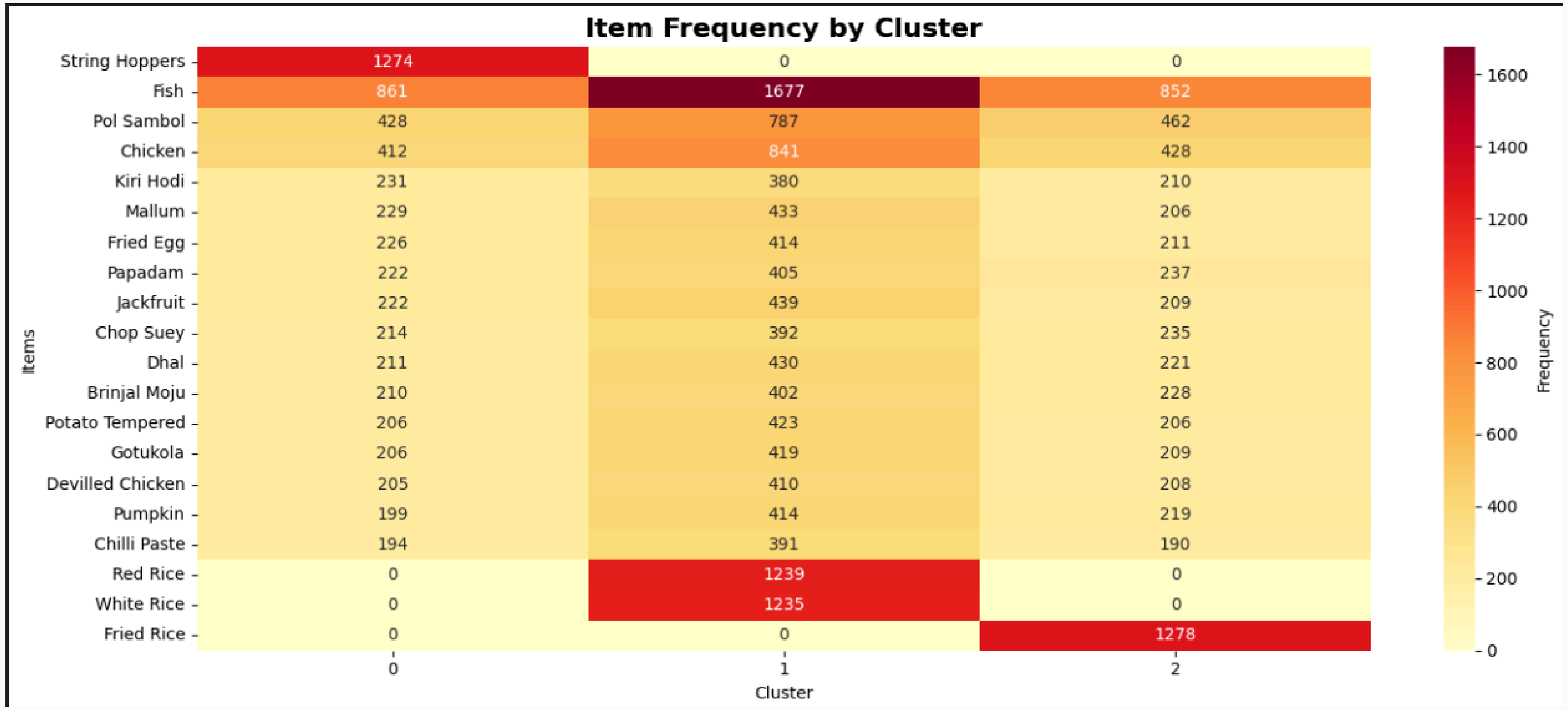
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_cleaned['Cluster'] = kmeans.fit_predict(df_encoded)

print(f"Cluster Distribution:\n{df_cleaned['Cluster'].value_counts().sort_index()}")
for i in range(3):
    count = len(df_cleaned[df_cleaned['Cluster'] == i])
    print(f"Cluster {i}: {count} ({count/len(df_cleaned)*100:.1f}%)")
```

```
Cluster Distribution:
Cluster
0    1274
1    2474
2    1278
Name: count, dtype: int64
Cluster 0: 1274 (25.3%)
Cluster 1: 2474 (49.2%)
Cluster 2: 1278 (25.4%)
```

EXPLORATORY DATA ANALYSIS



Key Findings: --->

Cluster 0: String Hopper Enthusiasts (Noodle-Based Lovers)

- **Size**: 1274 transactions (~25%)
- **Top 5 Items**:
  1. Fish: 691 times
  2. Pol Sambol: 381 times
  3. Chicken: 372 times
  4. String Hoppers: 335 times
  5. Fried Rice: 321 times
- **Dominant Base Starch**: Noodle-Based (1274 transactions)
- **Vegetarian Transactions**: 225
- **Most Active Time**: Late Lunch
- **Protein Preferences**:
  - Fish: 691 times
  - Chicken: 372 times
  - Fried Egg: 228 times
  - Devilled Chicken: 209 times

Cluster 1: Rice Lovers (Balanced Meat Eaters)

- **Size**: 2474 transactions (~50~%)
- **Top 5 Items**:
  1. Fish: 1345 times
  2. Pol Sambol: 764 times

Vegetarian Behavior:

- all clusters have ~ 20% vegetarian transactions

HYPERPARAMETER TUNING

Grid Search Configuration:

- Support values tested: [0.025, 0.05, 0.07]
- Confidence values tested: [0.4, 0.5, 0.6]
- Total combinations: 9 per cluste

Selection Criteria:

- Maximize number of rules with Lift > 1.0
- Different optimal parameters per cluster
- Adaptive to cluster size and pattern strength

Why Cluster-Specific Tuning?

- Smaller clusters need lower support for pattern discovery
- Larger clusters can use higher support for stronger patterns

```
Cluster 0:
  sup=0.03, conf=0.40 → 3 rules
  sup=0.03, conf=0.50 → 3 rules
  sup=0.03, conf=0.60 → 0 rules
  sup=0.05, conf=0.40 → 0 rules
  sup=0.05, conf=0.50 → 0 rules
  sup=0.05, conf=0.60 → 0 rules
  sup=0.07, conf=0.40 → 0 rules
  sup=0.07, conf=0.50 → 0 rules
  sup=0.07, conf=0.60 → 0 rules

Cluster 1:
  sup=0.03, conf=0.40 → 46 rules
  sup=0.03, conf=0.50 → 46 rules
  sup=0.03, conf=0.60 → 0 rules
  sup=0.05, conf=0.40 → 19 rules
  sup=0.05, conf=0.50 → 19 rules
  sup=0.05, conf=0.60 → 0 rules
  sup=0.07, conf=0.40 → 19 rules
  sup=0.07, conf=0.50 → 19 rules
  sup=0.07, conf=0.60 → 0 rules

Cluster 2:
  sup=0.03, conf=0.40 → 0 rules
  sup=0.03, conf=0.50 → 0 rules
  sup=0.03, conf=0.60 → 0 rules
  sup=0.05, conf=0.40 → 0 rules
  sup=0.05, conf=0.50 → 0 rules
  sup=0.05, conf=0.60 → 0 rules
  sup=0.07, conf=0.40 → 0 rules
```

OPTIMAL PARAMETERS SELECTION

```python
print("Best Hyperparameters:")
print("="*70)

best_params = {}
for cluster_id in sorted(tuning_results.keys()):
    best = max(tuning_results[cluster_id], key=lambda x: x['rules'])
    best_params[cluster_id] = best
    print(f"\nCluster {cluster_id}: Support={best['support']}, Confidence={best['confidence']}, Rules={best['rules']}")
```

```
Best Hyperparameters:
======================================================================

Cluster 0: Support=0.025, Confidence=0.4, Rules=3

Cluster 1: Support=0.025, Confidence=0.4, Rules=46

Cluster 2: Support=0.025, Confidence=0.4, Rules=0
```

APRIORI ALGORITHM & PRUNING

Anti-Monotonicity Principle:

- If an itemset is infrequent (below min_support), ALL its supersets are also infrequent.

Example:

- If {Fish} is infrequent -> Skip {Fish, Pol Sambol}
- Skip ALL supersets containing Fish
- Massive computational savings

Efficiency Impact:

- 15 items = 32,767 possible itemsets
- With pruning: Check only ~100 itemsets
- 99% reduction in search space

TOP RULES

```
Top 5 Rules per Cluster
==============================================================

Cluster 0:
  Pol Sambol, Dhal → Fish
    Support: 0.0259, Confidence: 0.5593, Lift: 1.0093
  String Hoppers, Pol Sambol, Dhal → Fish
    Support: 0.0259, Confidence: 0.5593, Lift: 1.0093
  Pol Sambol, Dhal → String Hoppers, Fish
    Support: 0.0259, Confidence: 0.5593, Lift: 1.0093

Cluster 1:
  Pol Sambol, Potato Tempered → Red Rice
    Support: 0.0283, Confidence: 0.5833, Lift: 1.1648
  Brinjal Moju, Pol Sambol → Red Rice
    Support: 0.0255, Confidence: 0.5833, Lift: 1.1648
  Pol Sambol, Papadam → Red Rice
    Support: 0.0259, Confidence: 0.5818, Lift: 1.1618
  Chicken, Potato Tempered → Red Rice
    Support: 0.0259, Confidence: 0.5766, Lift: 1.1513
  Brinjal Moju, Chicken → Red Rice
    Support: 0.0251, Confidence: 0.5741, Lift: 1.1463

Cluster 2:
  No strong rules
```

EVALUATION METRICS

1. Support:
    - Support = (Transactions with itemset) / (Total transactions)
    - Measures: How FREQUENTLY the pattern occurs
    - Example: Support(Fish, Pol Sambol) = 0.15 means 15% of transaction
2. Confidence:
    - Confidence = P(Consequent | Antecedent)
    - Measures: How RELIABLY antecedent predicts consequent
    - Example: Confidence(Fish -> Pol Sambol) = 0.75 means 75% reliability
3. Lift:
    - Lift = Confidence / Support(Consequent)
    - Lift > 1.0: Positive correlation (bought together MORE than random)
    - Lift = 1.0: No correlation (independent) | Lift < 1.0: Negative correlation

```
Association Rules Evaluatio
==============================

Cluster 0: 3 rules
  Avg Support: 0.0259
  Avg Confidence: 0.5593
  Avg Lift: 1.0093

Cluster 1: 46 rules
  Avg Support: 0.0680
  Avg Confidence: 0.5321
  Avg Lift: 1.0552

Cluster 2: No rules
```

THANK YOU