

THE APRIORI ALGORITHM: BIG DATA MINING IN MARKET BASKET ANALYSIS



CAC 2

23rd October 2024

SANJAY R

2348055.

MDS571

Big Data Analytics

Department of Statistics and Data Science

THE APRIORI ALGORITHM: BIG DATA MINING IN MARKET BASKET ANALYSIS

❖ Introduction:

The Apriori algorithm is a foundational data mining technique used for extracting frequent itemsets and generating association rules in large transactional datasets. This algorithm plays a crucial role in market basket analysis, enabling businesses to uncover relationships between items purchased together. By leveraging these insights, organizations can optimize inventory management, enhance product recommendations, and design effective marketing strategies.

In this case study, we implement the Apriori algorithm using Hadoop MapReduce to handle the challenges posed by massive datasets. Hadoop's distributed computing framework ensures scalability and efficiency, making it an ideal platform for processing large volumes of data. The study focuses on analyzing transaction data to identify frequent itemsets, calculate support, confidence, and lift metrics, and derive meaningful association rules.

❖ Market Basket Analysis:

Market Basket Analysis (MBA) is a data mining technique used to identify patterns or relationships between items frequently purchased together in transactional datasets. This analysis is particularly useful in the retail and e-commerce sectors for understanding customer purchasing behavior and optimizing business strategies.

The goal of MBA is to uncover association rules of the form $A \rightarrow B$, where the presence of item A in a transaction implies a likelihood of item B also being purchased. The strength of these associations is quantified using metrics such as **support**, **confidence**, and **lift**.

❖ Examples of Market Basket Analysis:

- **Retail Sector:** In a supermarket, MBA may reveal that customers who purchase bread are likely to buy butter as well. The association rule $bread \rightarrow butter$ can help managers place these items closer together to increase sales.
- **E-Commerce:** On online platforms, MBA might show that customers who purchase a smartphone frequently buy a phone case or screen protector. This insight can be used to recommend accessories during the checkout process.
- **Healthcare:** In pharmacy data, MBA can identify that customers buying flu medication are more likely to purchase vitamin supplements. Pharmacies can utilize this information to bundle these items for promotions.
- **Food Delivery:** Analysis might reveal that customers ordering pizza often include soft drinks or desserts in their orders. Restaurants can use this knowledge to create combo offers.

By applying Market Basket Analysis, businesses can enhance customer experience through personalized recommendations, improve cross-selling strategies, and optimize inventory management. These advantages make MBA a cornerstone of modern data-driven decision-making.

❖ Importance of the Case Study:

The significance of this case study lies in demonstrating the practical application of the Apriori algorithm in a big data environment. With the exponential growth of data, traditional single-machine implementations face limitations in terms of processing speed and scalability. This case study addresses these challenges by utilizing the parallel processing capabilities of Hadoop.

The insights gained from frequent itemset mining have far-reaching implications in various domains such as retail, e-commerce, and healthcare. By identifying frequently co-occurring items, businesses can make data-driven decisions to improve customer experience and increase revenue. Additionally, the study highlights the adaptability of Hadoop MapReduce for real-world data mining tasks, showcasing its potential for future applications in big data analytics.

✓ Use Cases of Market Basket Analysis:

Market Basket Analysis has become an essential tool for businesses to understand customer purchasing behavior and optimize their strategies. By analyzing customer purchase history and identifying patterns of frequently purchased items, retailers can gain valuable insights to enhance their product offerings and increase sales. Below are some specific use cases of Market Basket Analysis:

✓ Cross-Selling and Upselling:

By identifying complementary products that are frequently purchased together, businesses can use this information to cross-sell or upsell to customers. For example, suggesting a phone case and screen protector to a customer purchasing a smartphone.

✓ Product Bundling:

Market Basket Analysis helps businesses identify products that can be bundled together to create attractive packages for customers. This not only increases sales but also assists in clearing out excess inventory.

✓ Inventory Management:

Understanding which products are frequently purchased together allows businesses to optimize their inventory management by stocking the right quantities of each product, thereby reducing overstock or understock scenarios.

✓ Pricing Strategy:

By analyzing the relationships between different products and their pricing, businesses can determine the optimal price for products, ensuring competitive yet profitable pricing strategies.

✓ Store Layout Optimization:

Analyzing customer purchasing patterns enables businesses to optimize their store layout, encouraging customers to purchase complementary products or increasing the sales of slow-moving items by placing them strategically.

These use cases demonstrate the potential of Market Basket Analysis in enhancing business operations and improving customer satisfaction.

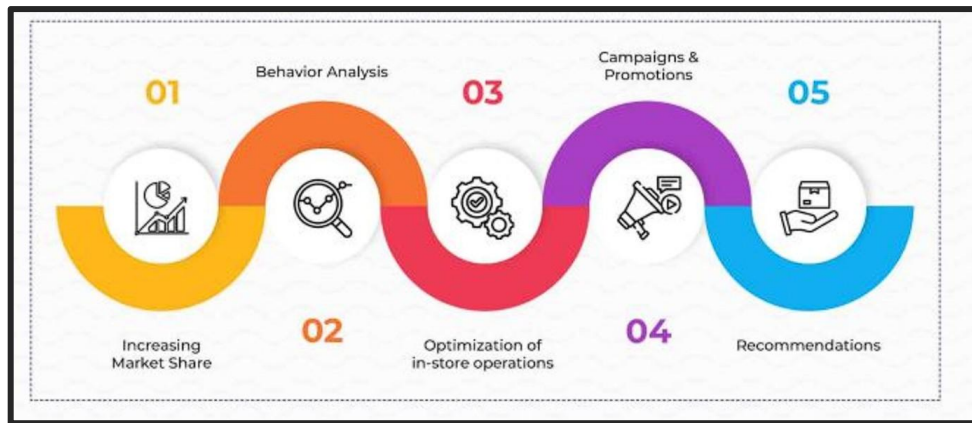


Figure 1: Credit: Javapoint.com

✓ Tools and Technologies Used:

In this study, we utilized advanced tools and technologies to implement Market Basket Analysis effectively on large datasets. The tools and technologies used include:

✓ Hadoop MapReduce:

Hadoop MapReduce is a powerful framework for processing and analyzing vast amounts of data in a distributed computing environment. It enables parallel processing by dividing tasks into smaller chunks, which are executed across multiple nodes. In this study, MapReduce was employed to implement the Apriori algorithm for mining frequent itemsets and generating association rules.

✓ Apache Pig:

Apache Pig is a high-level platform for processing and analyzing large datasets. It provides a scripting language, Pig Latin, which simplifies the development of data analysis programs. In this case study, Pig was used for data preprocessing and transformation, making the data suitable for analysis using MapReduce.

❖ Dataset Used:

The Market Basket Analysis dataset, publicly available on Kaggle, has been used for this project. This dataset contains transactional data from a retail store, which is used to perform Market Basket Analysis.

❖ Number of Attributes: 7

The dataset consists of the following attributes:

- **Itemname:** The product name of the item purchased.
- **Quantity:** The quantity of each product purchased per transaction.
- **Date:** The day and time when each transaction occurred.
- **Price:** The price of each product.
- **CustomerID:** A unique 5-digit number assigned to each customer.
- **Country:** The name of the country where the retail store is located.

- **Bill No.:** The unique identification number for each bill or transaction.

❖ Dataset Preview

A sample record from the dataset is as follows:

- **Bill No.:** 21663
- **Itemname:** 4187
- **Quantity:** 691
- **Date:** 19642
- **Price:** 1268
- **CustomerID:** 2499
- **Country:** 31

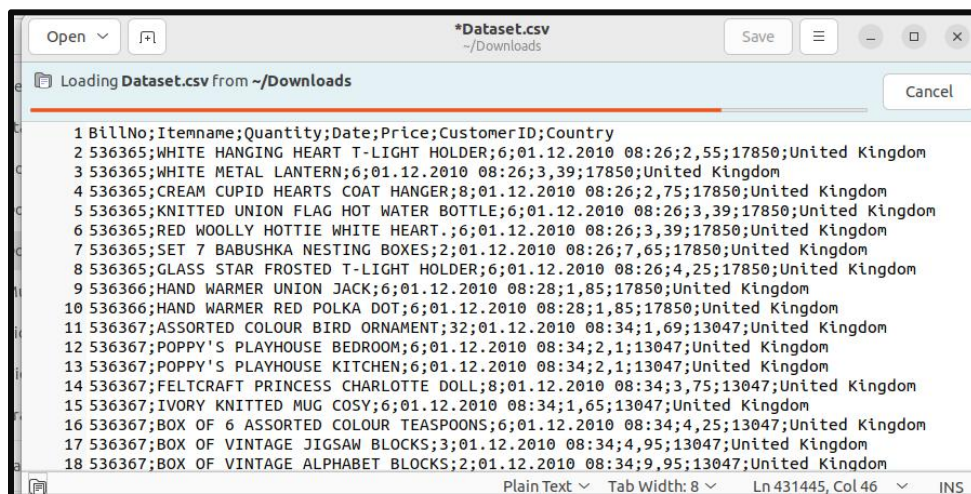


Figure 2: Dataset preview

❖ Dataset Statistics:

The dataset contains a total of 5,35,260 tuples, each representing a unique transaction or purchase event.

❖ Pre-processing:

Data preprocessing is an important step in the data analysis pipeline that involves transforming raw data into a format that can be used for analysis. This is done in order to improve data quality and model processing.

We performed the following steps to make our dataset suitable for modelling:

1. Removing and filling null values as required
2. Removing unnecessary rows or noise in the data
3. Formatting certain columns in the required format

❖ Frequent Itemset and Association Rule Generation:

Association rule learning is a rule-based machine learning method used to discover interesting relationships between variables in large databases. It aims to identify strong rules discovered in databases.

For example, if item A is bought by customer c_1 , then the likelihood of item B being bought by the same customer in the same transaction increases. This can be represented as:

$A \Rightarrow B$, where A is the antecedent and B is the consequent.

In a large dataset with many transactions, we can have various combinations of itemsets such as:

$A \Rightarrow B$, $A \& B \Rightarrow C$, $A \& B \& C \Rightarrow D$, and so on.

To find promising rules, we need to evaluate these itemsets using specific metrics. These metrics are defined as follows:

❖ Apriori Algorithm Implementation in Hadoop MapReduce

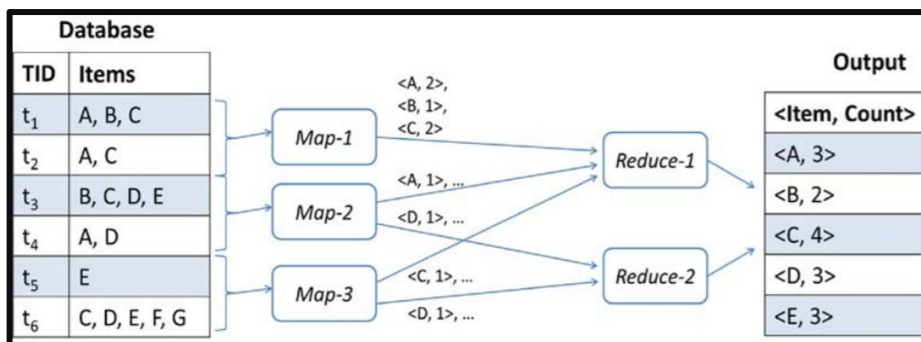
✓ Part 1 : Just finding frequent item-sets

This section describes the implementation of the Apriori algorithm in Hadoop MapReduce, focusing on the Mapper

and Reducer phases for finding frequent itemsets. The implementation consists of two phases: Phase 1 for generating frequent 1-itemsets and Phase 2 for generating frequent k-itemsets.

✓ Phase 1 Mapper

The goal of the Phase 1 Mapper is to process the transaction data and generate frequent 1-itemsets by emitting key-value pairs where the key is a combination of the country and item, and the value is 1. The Mapper then counts the occurrences of each item and computes its support in the reducer phase.



```
public static class Phase1Mapper extends Mapper<LongWritable, Text, Text,
    IntWritable> { private final static IntWritable ONE = new IntWritable(1);
```

```
@Override
```

```
protected void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
```

```
    String line = value.toString();
```

```
    if (line.startsWith("BillNo")) return; // Skip header
```

```
    String[] fields =
```

```
    line.split(";"); if
```

```
    (fields.length > 7) return;
```

```

String country = fields[6].trim();
String[] items =
fields[1].trim().split(","); for
(String item : items) –
    String cleanItem =
    item.trim(); if
    (!cleanItem.isEmpty()) –
        context.write(new Text(country + ":" + cleanItem), ONE);

```

✓ **Explanation:**

1. `line.startsWith("BillNo")` ensures the header is skipped. 2. `fields[6].trim()` extracts the country information. 3. `String[] items = fields[1].trim().split(",")` splits the comma-separated items in the transaction. 4. For each item, the Mapper writes a key-value pair where the key is the concatenation of the country and the item, and the value is 1 (indicating the occurrence of that item in the transaction).

✓ **Phase 1 Reducer:**

The Phase 1 Reducer aggregates the counts of the 1-itemsets from the Mapper and filters them based on the minimum support threshold. If the count of an itemset meets or exceeds the minimum support, the itemset is written to the output.

```

public static class Phase1Reducer extends Reducer<Text, IntWritable, Text,
    IntWritable> – private int minSupport;

```

```

@Override
protected void setup(Context context) –
    minSupport = context.getConfiguration().getInt("minSupport", 2);
"

```

```

@Override
protected void reduce(Text key, Iterable<IntWritable> values,
    Context context) throws IOException, InterruptedException
–

```

```

    int sum = 0;
    for (IntWritable val : values) –
        sum += val.get(); // Count the occurrences of the itemset
    "

```

```

    if (sum >= minSupport) –
        context.write(key, new IntWritable(sum));
    "

```

```

"

```


✓ **Explanation:**

1. minSupport = context.getConfiguration().getInt("minSupport", 2) reads the minimum support from the configuration. 2. The Reducer aggregates the values for each itemset and sums their counts. 3. If the summed count exceeds the minimum support, the itemset is written to the output.

✓ **Phase 2 Mapper:**

The Phase 2 Mapper is responsible for processing the frequent 1-itemsets generated in Phase 1 and generating frequent 2-itemsets. It reads the transaction data, extracts the items, and generates combinations of itemsets of size 2.

```
public static class Phase2Mapper extends Mapper<LongWritable, Text, Text,
    IntWritable> – private final static IntWritable ONE = new IntWritable(1);
```

```
@Override
```

```
protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException –
```

```
    String line = value.toString();
    if (line.startsWith("BillNo")) return;
```

```
    String[] fields =
    line.split(";"); if
    (fields.length < 7) return;
```

```
    String country = fields[6].trim();
    String[] items =
    fields[1].trim().split(","); List<String>
    cleanItems = new ArrayList<>();
```

```
    for (String item : items) –
        String cleanItem =
        item.trim(); if
        (!cleanItem.isEmpty()) –
            cleanItems.add(cleanItem);
    "
```

```
    if (cleanItems.size() <= 2) –
        Set<Set<String>> combinations =
        generateCombinations(cleanItems, 2); for (Set<String>
        itemset : combinations) –
            List<String> sortedItems = new ArrayList<>(itemset);
            Collections.sort(sortedItems); // Sort items for consistent key
            format context.write(new Text(country + ":" + String.join(",",
            sortedItems)), ONE);
```

```
private Set<Set<String>> generateCombinations(List<String> items,
    int k) – Set<Set<String>> combinations = new HashSet<>();
    generateCombinationsHelper(items, new HashSet<>(), 0, k,
    combinations); return combinations;
```


”

```
private void generateCombinationsHelper(List<String> items,
Set<String> current, int start, int k,
    Set<Set<String>>
combinations) – if
(current.size() == k) –
    combinations.add(new HashSet<>(current)); // Add
    combination to set return;
”
for (int i = start; i < items.size(); i++) –
```

```

current.add(items.get(i));
generateCombinationsHelper(items, current, i + 1, k,
combinations); current.remove(items.get(i));    //
Backtrack

```

✓ **Explanation:**

1. generateCombinations(cleanItems, 2) generates all combinations of size 2 from the list of items in a transaction. 2. generateCombinationsHelper() uses backtracking to generate combinations and add them to a set. 3. String.join(",", sortedItems) joins the sorted itemset into a string that will be used as the key in the Reducer.

✓ **Phase 2 Reducer:**

The Phase 2 Reducer works similarly to the Phase 1 Reducer, except that it processes 2-itemsets instead of 1-itemsets. It aggregates the counts for the 2-itemsets and filters them based on the minimum support threshold.

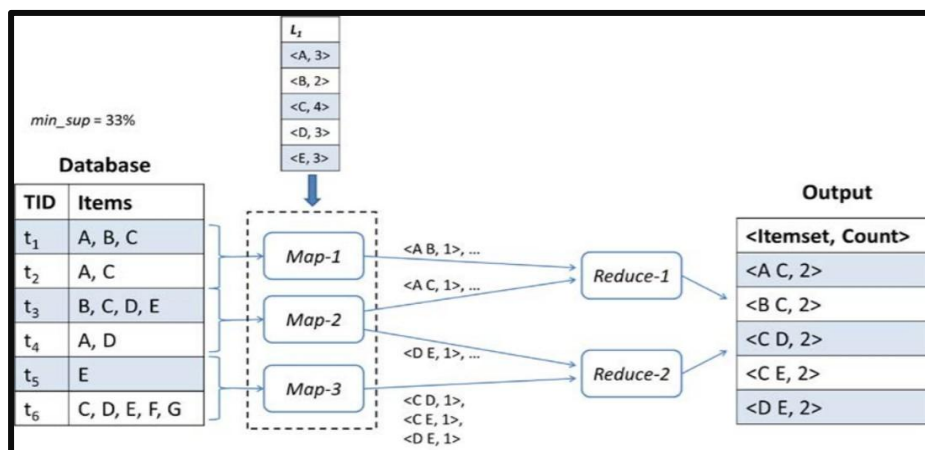


Figure 4: Phase 2

```

public static class Phase2Reducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
    private int minSupport;

```

```

    @Override

```

```

    protected void setup(Context context) {

```

```

        minSupport = context.getConfiguration().getInt("minSupport", 2);

```

```

    }

```

```

@Override
protected void reduce(Text key, Iterable<IntWritable> values,
    Context context) throws IOException, InterruptedException
{
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get(); // Count occurrences of the itemset
    }

    if (sum >= minSupport) {
        context.write(key, new IntWritable(sum));
    }
}

```

✓ **Explanation:**

1. The Reducer aggregates the counts for each 2-itemset. 2. It checks if the itemset meets the minimum support threshold and writes the itemset to the output if it does.

✓ **Main Driver:**

The main driver configures and initiates two MapReduce jobs: one for finding frequent 1-itemsets and another for finding frequent 2-itemsets.

```

public static void main(String[] args) throws
    Exception {
    if (args.length < 2) {
        System.err.println("Usage: AprioriDriver <input path> <output path>");
        System.exit(-1);
    }
}

```

```

Configuration conf = new Configuration();
conf.setInt("minSupport", 8); // Set minimum support

```

```

// Job 1: Frequent 1-itemsets
Job job1 = Job.getInstance(conf, "Frequent 1-itemsets");
job1.setJarByClass(AprioriDriver.class);
job1.setMapperClass(Phase1Mapper.class);
job1.setReducerClass(Phase1Reducer.class);
job1.setOutputKeyClass(Text.class);
job1.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job1, new Path(args[0]));
Path phase1Output = new Path(args[1] + "/frequent1");
FileOutputFormat.setOutputPath(job1, phase1Output);

```

```

if (!job1.waitForCompletion(true)) {
    System.exit(1);
}

```

```

// Job 2: Frequent k-itemsets
conf.setInt("k", 2); // Set k for frequent 2-itemsets
Job job2 = Job.getInstance(conf, "Frequent k-itemsets");

```

```

job2.setJarByClass(AprioriDriver.class);
job2.setMapperClass(Phase2Mapper.class);
job2.setReducerClass(Phase2Reducer.class);
job2.setOutputKeyClass(Text.class);
job2.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job2,
phase1Output);
FileOutputFormat.setOutputPath(job2, new Path(args[1] + "/frequent2"));

System.exit(job2.waitForCompletion(true) ? 0 : 1);

```

This implementation demonstrates how to execute Apriori on Hadoop using MapReduce to find frequent item- sets for a given dataset.

✓ Output:

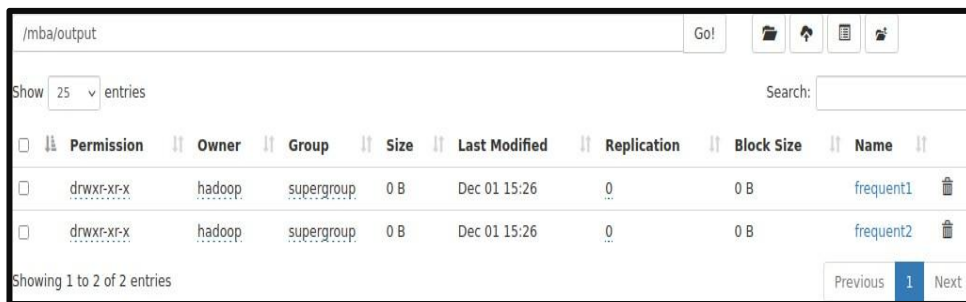


Figure 5: The output directory after running the code will have 2 folders as shown above.

Open		
1	Australia:BAKING SET SPACEBOY DESIGN	8
2	Australia:LUNCH BAG RED RETROSPOT	8
3	Australia:LUNCH BAG SPACEBOY DESIGN	8
4	Australia:PARTY BUNTING	8
5	Australia:RED TOADSTOOL LED NIGHT LIGHT	9
6	Australia:ROSES REGENCY TEACUP AND SAUCER	8
7	Australia:SET OF 3 CAKE TINS PANTRY DESIGN	9
8	Austria:POSTAGE	14
9	Belgium:60 CAKE CASES DOLLY GIRL DESIGN	9
10	Belgium:ALARM CLOCK BAKELIKE GREEN	12
11	Belgium:ALARM CLOCK BAKELIKE PINK	9
12	Belgium:ALARM CLOCK BAKELIKE RED	13
13	Belgium:BREAD BIN DINER STYLE RED	8
14	Belgium:CHARLOTTE BAG APPLES DESIGN	10
15	Belgium:CHARLOTTE BAG DOLLY GIRL DESIGN	8
16	Belgium:CHARLOTTE BAG PINK POLKADOT	8
17	Belgium:CHILDRENS CUTLERY DOLLY GIRL	8
18	Belgium:CHILDRENS CUTLERY SPACEBOY	11
19	Belgium:CIRCUS PARADE LUNCH BOX	10
20	Belgium:COFFEE MUG APPLES DESIGN	9
21	Belgium:DOLLY GIRL LUNCH BOX	23
22	Belgium:IVORY KITCHEN SCALES	12
23	Belgium:JUMBO BAG RED RETROSPOT	10

(a) 1-frequent items

Open		
1	Germany:COFFEE,SET 3 RETROSPOT TEA	11
2	Germany:COFFEE,SUGAR	11
3	Germany:SET 3 RETROSPOT TEA,SUGAR	11
4	United Kingdom:1 HANGER,HOOK	89
5	United Kingdom:1 HANGER,MAGIC GARDEN	89
6	United Kingdom:5 SUMMER B'DRAW LINERS,FLOWER FAIRY	24
7	United Kingdom:ACRYLIC HANGING JEWEL,BLUE	27
8	United Kingdom:ACRYLIC JEWEL ICICLE,PINK	18
9	United Kingdom:AIRLINE LOUNGE,METAL SIGN	290
10	United Kingdom:ART LIGHTS,FUNK MONKEY	60
11	United Kingdom:B,C PAINTED LETTERS	69
12	United Kingdom:B,NURSERY A	69
13	United Kingdom:BACK DOOR,KEY FOB	317
14	United Kingdom:BATHROOM SCALES,TROPICAL BEACH	12
15	United Kingdom:BILLBOARD FONTS DESIGN,WRAP	29
16	United Kingdom:BIRTHDAY CARD,ELEPHANT	145
17	United Kingdom:BIRTHDAY CARD,RETRO SPOT	255
18	United Kingdom:BLACK TEA,COFFEE	44
19	United Kingdom:BLACK TEA,SUGAR JARS	44
20	United Kingdom:BREAKFAST IN BED,TRAY	71
21	United Kingdom:C PAINTED LETTERS,NURSERY A	69
22	United Kingdom:CHOCOLATE SPOTS,SWISS ROLL TOWEL	92
23	United Kingdom:CHOCOLATE SPOTS,LARGE CAKE TOWEL	8
24	United Kingdom:CHRISTMAS GARLAND STARS,TREES	48
25	United Kingdom:COAL,BLACK FEATHER PEN	160

(b) 2-frequent items

Figure 6: Part 1 : Output

❖ Part 2 : Partitioning the results based on the location of the retail stores.

✓ Phase 1: Mapper for Frequent 1-itemsets

In Phase 1, we focus on identifying frequent 1-itemsets. The 'Phase1Mapper' class processes each transaction, extracts the country and items, and emits key-value pairs with the format country:item as the key and 1 as the value.

```
public static class Phase1Mapper extends Mapper<LongWritable, Text, Text,
    IntWritable> { – private final static IntWritable ONE = new IntWritable(1);
```

```
@Override
```

```
protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException –
```

```
    String line = value.toString();
    if (line.startsWith("BillNo")) return;
```

```
    String[] fields =
    line.split(";"); if
    (fields.length < 7) return;
```

```
    String country = fields[6].trim();
    String[] items =
    fields[1].trim().split(","); for
    (String item : items) –
        String cleanItem =
        item.trim(); if
        (!cleanItem.isEmpty()) –
            // Key format: country:item
            context.write(new Text(country + ":" + cleanItem), ONE);
```

This mapper:

- Skips any lines that start with "BillNo".
- Splits the input line to extract the country and items.
- Emits the country-item pairs as keys and 1 as the value.

✓ Phase 1: Reducer for Frequent 1-itemsets

The 'Phase1Reducer' class receives the country-item key-value pairs from the mapper and aggregates them by summing the occurrences. It then filters the results to keep only those itemsets with a frequency greater than or equal to the minimum support threshold.

```
public static class Phase1Reducer extends Reducer<Text, IntWritable, Text,
    IntWritable> { – private int minSupport;
```

```
@Override
```

```
protected void setup(Context context) –
    minSupport = context.getConfiguration().getInt("minSupport", 2);
"
```

```
@Override
```

```

protected void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException,
InterruptedException
    – int sum = 0;
    for (IntWritable val :
        values) – sum +=
        val.get();
    "
    if (sum <= minSupport) –
        context.write(key, new IntWritable(sum));

```

The reducer:

- Sums the occurrences of each country-item pair.
- Filters out any itemsets whose frequency is below the minimum support.
- Outputs the frequent 1-itemsets (country:item and their counts).

✓ **Phase 2: Mapper for Frequent k-itemsets:**

In Phase 2, the mapper processes transactions to generate combinations of items. The ‘Phase2Mapper’ class extracts the items from the transaction, generates 2-item combinations (for simplicity, the value of k is set to 2), and emits them as key-value pairs, where the key is the sorted itemset and the value is 1.

```

public static class Phase2Mapper extends Mapper<LongWritable, Text, Text,
IntWritable> – private final static IntWritable ONE = new IntWritable(1);

```

@Override

```

protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException –

```

```

    String line = value.toString();
    if (line.startsWith("BillNo")) return;

```

```

    String[] fields =
    line.split(";"); if
    (fields.length < 7) return;

```

```

    String country = fields[6].trim();
    String[] items =
    fields[1].trim().split(","); List<String>
    cleanItems = new ArrayList<>();

```

```

    for (String item : items) –
        String cleanItem =
        item.trim(); if
        (!cleanItem.isEmpty()) –
            cleanItems.add(cleanItem);
    "

```

```

    if (cleanItems.size() <= 2) –

```

```

Set<Set<String>> combinations =
generateCombinations(cleanItems, 2); for (Set<String>
itemset : combinations) –
    List<String> sortedItems = new ArrayList<>(itemset);
    Collections.sort(sortedItems);
    context.write(new Text(country + ":" + String.join(",",
sortedItems)), ONE);
"

private Set<Set<String>> generateCombinations(List<String> items,
int k) – Set<Set<String>> combinations = new HashSet<>();
generateCombinationsHelper(items, new HashSet<>(), 0, k,
combinations); return combinations;
"

private void generateCombinationsHelper
(List<String> items, Set<String> current, int start, int k,
Set<Set<String>> combinations) –
if (current.size() == k) –
    combinations.add(new
HashSet<>(current)); return;
"

for (int i = start; i < items.size(); i++) –
    current.add(items.get(i));
    generateCombinationsHelper(items, current, i + 1, k, combinations);
    current.remove(items.get(i));

```

This mapper:

- Extracts the items from each transaction and generates 2-item combinations.
- Emits these combinations as keys and 1 as the value.

✓ Phase 2: Reducer for Frequent k-itemsets:

The 'Phase2Reducer' class processes the 2-item combinations, sums their occurrences, and filters out those that do not meet the minimum support threshold.

```

public static class Phase2Reducer extends Reducer<Text, IntWritable, Text,
IntWritable> –

```

```

    private int minSupport;

```

```

    @Override

```

```

    protected void setup(Context context) –

```

```

        minSupport = context.getConfiguration().getInt("minSupport", 2);
    "

```

```

    @Override

```

```

    protected void reduce(Text key, Iterable<IntWritable> values,

```

```

        Context context) throws IOException, InterruptedException
    –

```

```

        int sum = 0;

```



```

    for (IntWritable val :
        values) – sum +=
        val.get();
    "
    if (sum <= minSupport) –
        context.write(key, new IntWritable(sum));

```

The reducer:

- Sums the occurrences of each itemset.
- Filters out itemsets whose frequency is below the minimum support.
- Outputs the frequent k-itemsets (country:itemset and their counts).

✓ **Location Partitioner:**

The ‘LocationPartitioner’ class assigns each key to a partition based on the country, ensuring that itemsets from the same country are processed by the same reducer.

```

public static class LocationPartitioner extends Partitioner<Text,
    IntWritable> – @Override
    public int getPartition(Text key, IntWritable value, int
        numPartitions) – String[] parts = key.toString().split(":");
        String country = parts[0];

        // Assign partitions based on country
        return Math.abs(country.hashCode()) % numPartitions;
    "
    "

```

The partitioner:

- Uses the country (extracted from the key) to determine which partition the data should go to.
- Ensures data is grouped by country for more efficient processing.

✓ **Driver for Job Configuration:**

The ‘AprioriDriver’ class configures and runs the two MapReduce jobs for frequent itemset mining.

```

public static void main(String[] args) throws
    Exception – if (args.length < 2) –
        System.err.println("Usage: AprioriDriver <input path>
        <output path>"); System.exit(-1);
    "

```

```

Configuration conf = new Configuration();

```

```
conf.setInt("minSupport", 8);
```

```
// Job 1: Frequent 1-itemsets
```

```
Job job1 = Job.getInstance(conf, "Frequent 1-itemsets with Partitioning");  
job1.setJarByClass(AprioriDriver.class);
```

```
job1.setMapperClass(Phase1Mapper.class);  
job1.setPartitionerClass(LocationPartitioner.class);  
job1.setReducerClass(Phase1Reducer.class);
```

```
job1.setOutputKeyClass(Text.class);  
job1.setOutputValueClass(IntWritable.class);
```

```
FileInputFormat.addInputPath(job1, new  
Path(args[0])); Path phase1Output = new  
Path(args[1] + "/frequent1");  
FileOutputFormat.setOutputPath(job1,  
phase1Output);
```

```
job1.setNumReduceTasks(3);
```

```
if  
    (!job1.waitForCompletion(true)  
    e)) – System.exit(1);  
"
```

```
// Job 2: Frequent k-itemsets
```

```
conf.setInt("k", 3);  
Job job2 = Job.getInstance(conf, "Frequent k-itemsets with Partitioning");  
job2.setJarByClass(AprioriDriver.class);
```

```
job2.setMapperClass(Phase2Mapper.class);  
job2.setPartitionerClass(LocationPartitioner.class);  
job2.setReducerClass(Phase2Reducer.class);
```

```
job2.setOutputKeyClass(Text.class);  
job2.setOutputValueClass(IntWritable.class);
```

```
FileInputFormat.addInputPath(job2,  
phase1Output); Path phase2Output = new  
Path(args[1] + "/frequentk");  
FileOutputFormat.setOutputPath(job2,  
phase2Output);
```

```
job2.setNumReduceTasks(3);  
System.exit(job2.waitForCompletion(true) ?  
0 : 1);  
"
```

✓ Output:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	Dec 01 15:36	3	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	99 B	Dec 01 15:36	3	128 MB	part-r-00000
-rw-r--r--	hadoop	supergroup	2.33 KB	Dec 01 15:36	3	128 MB	part-r-00001
-rw-r--r--	hadoop	supergroup	0 B	Dec 01 15:36	3	128 MB	part-r-00002

Figure 7: The output directory after running the code will have 2 folders as shown above. The second folder will have 3 output files based on the location wise results.

```
Germany:COFFEE,SET 3 RETROSPOT TEA 11
Germany:COFFEE,SUGAR 11
Germany:SET 3 RETROSPOT TEA,SUGAR 11
```

(a) part-r-00000

```
United Kingdom:1 HANGER,HOOK 89
United Kingdom:1 HANGER,MAGIC GARDEN 89
United Kingdom:5 SUMMER B'DRAW LINERS,FLOWER FAIRY 24
United Kingdom:ACRYLIC HANGING JEWEL,BLUE 27
United Kingdom:ACRYLIC JEWEL ICICLE,PINK 18
United Kingdom:AIRLINE LOUNGE,METAL SIGN 290
```

(b) part-r-00001

Figure 8: Part 2 : Output. Here we get 2 files since only 2 locations have exceeded the minimum support of 8 in the frequent item set mining algorithm.

❖ Part 3 : Finding frequent itemsets as well as generating rules based on confidence level and support value.

Only the Phase 2 reducer logic changes here.

✓ Phase 2 Reducer: Distributed Cache and Confidence Calculation Purpose of Distributed Cache:

In Hadoop, the distributed cache is used to share files across all nodes participating in a job. In the case of Phase 2, the item support values from the first phase (frequent 1-itemsets) are loaded into a distributed cache for quick access by the Phase 2 Reducer. This cache allows the reducer to efficiently access the support values for individual items when calculating the confidence and lift of itemsets.

The Phase 2 Reducer uses the distributed cache to store the output of Phase 1 (frequent 1-itemsets). This approach allows the reducer to efficiently access the support count of individual items, which is essential for calculating confidence and lift for frequent 2-itemsets. By loading this data into memory, the computation avoids repeated and expensive I/O operations during the reducer's execution, thus improving performance.

✓ Implementation of Distributed Cache:

The distributed cache is implemented by reading the frequent 1-itemsets from HDFS during the reducer's setup phase. This data is stored in a 'HashMap' for quick lookups. Each line in the Phase 1 output file is parsed to extract the item and its support count, which are then added to the map. Below is the verbatim code for the distributed cache setup:

```

@Override
protected void setup(Context context) –
    minSupport = context.getConfiguration().getInt("minSupport", 2);
    totalTransactions = context.getConfiguration().getInt("totalTransactions",
    43328);

    // Load item support from Phase 1 output into
    the map try –
        Path phase1OutputPath = new
        Path("hdfs://localhost:9000/mba/output`assoc/frequent1"); FileSystem fs =
        FileSystem.get(context.getConfiguration());
        FileStatus[] statuses =
        fs.listStatus(phase1OutputPath); for (FileStatus
        status : statuses) –
            if (status.isFile()) –
                BufferedReader br = new BufferedReader(new InputStreamReader
                (fs.open(status.getPath())));
                String line;
                while ((line = br.readLine()) !=
                null) – String[] parts =
                line.split("\t"); if
                (parts.length != 2) –
                    String item = parts[0].trim();
                    int support = Integer.parseInt(parts[1]);
                    itemSupportMap.put(item, support);

                br.close()
            ;
    " catch (IOException e) –
        e.printStackTrace();

```

Support, Confidence, and Lift Calculation:

In the context of the Apriori algorithm, support, confidence, and lift are used to evaluate the strength of the relationships between itemsets.

✓ Support:

Support represents the frequency of occurrence of an itemset in the dataset. For each itemset S , the support is calculated as:

$$\text{Support}(S) = \frac{\text{Count of transactions containing } S}{\text{Total number of transactions}}$$

In the Phase 2 Reducer, the support for a k -itemset is simply the count of occurrences of the itemset in the input data. This count is compared to the minimum support threshold specified in the configuration. If the support meets or exceeds this threshold, the itemset is considered frequent and passed to the output.

✓ Confidence:

Confidence measures the likelihood that an item A will be bought given that item B is bought, i.e., $P(A|B)$. It is computed as the ratio of the support of the itemset $A \cup B$ to the support of the item B :

$$\text{Confidence}(A \rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(B)}$$

In the reducer, confidence is calculated for item pairs (or larger itemsets) using the item support values from Phase 1 stored in the distributed cache. The confidence of a rule is only calculated if the itemset's support meets the minimum support threshold.

;

❖ Output:

1 HANGER,HOOK Support: 0.002307976366322009 Confidence: 2.307976366322009E-5 Lift: 4.3328 100	
1 HANGER,MAGIC GARDEN Support: 0.002307976366322009 Confidence: 2.307976366322009E-5 Lift: 4.3328 100	
5 SUMMER B'DRAW LINERS,FLOWER FAIRY Support: 5.539143279172822E-4 Confidence: 2.307976366322009E-5 Lift: 75.2222222222223	24
ACRYLIC HANGING JEWEL,BLUE Support: 6.231536189069424E-4 Confidence: 1.832804761491007E-5 Lift: 31.62456747404844	27
ACRYLIC JEWEL ICICLE,PINK Support: 4.154357459379616E-4 Confidence: 1.9782654568474362E-5 Lift: 45.345892203035056	18
AIRLINE LOUNGE,METAL SIGN Support: 0.006762370753323486 Confidence: 2.307976366322009E-5 Lift: 0.28113523404144874	293
ART LIGHTS,FUNK MONKEY Support: 0.0014078655834564254 Confidence: 2.307976366322009E-5 Lift: 11.644181671593659	61
B,C PAINTED LETTERS Support: 0.0016155834564254062 Confidence: 2.307976366322009E-5 Lift: 8.842448979591838	70
B,NURSERY A Support: 0.0016155834564254062 Confidence: 2.307976366322009E-5 Lift: 8.842448979591838	70
BACK DOOR,KEY FOB Support: 0.007339364844903988 Confidence: 2.3079763663220087E-5 Lift: 0.13050916889563605	318
BATHROOM SCALES,TROPICAL BEACH Support: 2.769571639586411E-4 Confidence: 2.307976366322009E-5 Lift: 300.8888888888889	12
BILLBOARD FONTS DESIGN,WRAP Support: 7.154726735598227E-4 Confidence: 2.3079763663220087E-5 Lift: 41.108159392789375	31
BIRTHDAY CARD,ELEPHANT Support: 0.0037620014771048743 Confidence: 8.70833675255758E-6 Lift: 0.23216735253772292	163
BIRTHDAY CARD,RETRO SPOT Support: 0.0062084564254062035 Confidence: 1.4371426910662508E-5 Lift: 0.2321673525377229	269
BLACK TEA,COFFEE Support: 0.001038589364844904 Confidence: 2.307976366322009E-5 Lift: 1.8270293063461944	45
BLACK TEA,SUGAR JARS Support: 0.001038589364844904 Confidence: 2.307976366322009E-5 Lift: 9.725701459034793	45
BREAKFAST IN BED,TRAY Support: 0.0017771418020679469 Confidence: 2.307976366322009E-5 Lift: 7.307809074042841	77
C PAINTED LETTERS,NURSERY A Support: 0.0016155834564254062 Confidence: 2.307976366322009E-5 Lift: 8.842448979591838	70
CHOCOLATE SPOTS,SWISS ROLL TOWEL Support: 0.0021464180206794683 Confidence: 2.307976366322009E-5 Lift: 3.451055356431701	93
CHOCOLATE SPOTS,LARGE CAKE TOWEL Support: 1.8463810930576072E-4 Confidence: 2.307976366322009E-5 Lift: 676.9999999999999	8
CHRISTMAS GARLAND STARS,TREES Support: 0.0011078286558345643 Confidence: 2.307976366322009E-5 Lift: 18.805555555555557	48
COAL BLACK,FEATHER PEN Support: 0.003992799113737075 Confidence: 2.307976366322009E-5 Lift: 0.4480337514347463	173
COFFEE,SET 3 RETROSPOT TEA Support: 0.009878138847858198 Confidence: 1.874409648549943E-5 Lift: 0.15600819503904886	428
COFFEE,SUGAR Support: 0.009878138847858198 Confidence: 1.874409648549943E-5 Lift: 0.15600819503904886	428
COFFEE,SUGAR JARS Support: 0.0022848966026587886 Confidence: 4.335667177720661E-6 Lift: 0.15600819503904884	99
COFFEE,WHITE TEA Support: 0.0012463072378138848 Confidence: 2.3649093696658153E-6 Lift: 0.15600819503904884	54
CUPCAKE SINGLE HOOK,METAL SIGN Support: 0.00537758493353028 Confidence: 2.3079763663220087E-5 Lift: 0.3535305732795901	233

Figure 9: Association Rules for k=2 i.e. 2 items bought together

Open	+	*part-r-00000(6)	Save	≡	–	□	×
~/Downloads							
1		1 HANGER,HOOK,MAGIC GARDEN Support: 0.002307976366322009					
		Confidence: 2.307976366322009E-5 Lift: 4.3328 100					
2		B,C PAINTED LETTERS,NURSERY A Support:					
		0.0016155834564254062 Confidence: 2.307976366322009E-5					
		Lift: 8.842448979591838 70					
3		BLACK TEA,COFFEE,SUGAR JARS Support: 0.001038589364844904					
		Confidence: 2.307976366322009E-5 Lift:					
		1.8270293063461944 45					
4		COFFEE,SET 3 RETROSPOT TEA,SUGAR Support:					
		0.009878138847858198 Confidence: 1.874409648549943E-5					
		Lift: 0.15600819503904886 428					
5		COFFEE,SUGAR JARS,WHITE TEA Support: 0.001246307237813884					
		Confidence: 2.3649093696658153E-6 Lift:					
		0.08509537911220846 54					
6		DECORATION,METAL,WOBBLY CHICKEN Support:					
		8.308714918759232E-4 Confidence: 9.550247033056588E-6					
		Lift: 2.368718137958369 36					
7		DECORATION,METAL,WOBBLY RABBIT Support:					
		0.0011770679468242244 Confidence: 1.3529516630163499E-5					
		Lift: 3.3556840287743563 51					
8		DOUGHNUTS,GREETING CARD,SQUARE Support:					

Figure 10: Generalizing. Here we have set the value of k to be 3. So how 3 items bought together what are their confidence values and lift values can be noted. P.S : the maximum limit of k depends on the type of dataset and transactions made for items to be present together.

❖ Interpretation of Part 3: Association Rule Mining in part 3

In Figure 9 we have the first row as

1 HANGER`HOOK — Support: 0.002307976336223209
— Confidence: 2.307976336223209E-5 — Lift: 4.3328 100

The first row contains the following information:

- **Itemset:** HANGER HOOK
This represents an itemset in the dataset.
- **Support:** 0.002307976336223209
Support is the fraction of transactions that contain this itemset. Here, it means that HANGER HOOK appears in approximately 0.23% of all transactions.
- **Confidence:** $2.307976336223209 \times 10^{-5}$

Confidence is the probability that if a customer buys this item (HANGER HOOK), they will also buy another item in the rule. The value $2.307976336223209 \times 10^{-5}$ is very small, indicating low confidence for this rule.

- **Lift:** 4.3328
Lift is a measure of how much more likely the items in the rule are to appear together than by random chance. **A lift value of 4.3328 suggests that the occurrence of HANGER HOOK is more than four times as likely as it would be if the items were independent.**

Improving Business Using Association Rule Mining

Association Rule Mining provides valuable insights into customer purchasing behavior by identifying itemsets frequently bought together. From the analysis results presented, businesses can implement the following strategies to improve sales and customer satisfaction:

- **Cross-Selling Opportunities:** The rule with itemset HANGER HOOK demonstrates a lift value of 4.3328. This indicates that customers are over four times more likely to purchase HOOK alongside HANGER than by random chance. Businesses can leverage this by promoting these items together, for example, through bundle offers or targeted advertising.
- **Improving Product Placement:** The high lift values for various itemsets, such as DECORATION, METAL, WOBBLY CHICKEN, suggest strong co-purchase likelihoods. Placing these products in close proximity within the store or online platform can increase visibility and sales of related items.
- **Targeted Marketing Campaigns:** Although the confidence value for some rules (e.g., HANGER HOOK) is low, businesses can use insights about frequent itemsets to design specific promotions. For instance, discounts or loyalty points can be offered to customers who purchase items like HANGER to encourage further purchases.
- **Inventory Management:** Itemsets with significant support values, such as COFFEE, SUGAR JARS, WHITE TEA, indicate popular combinations. Retailers can ensure adequate stock of these items to prevent shortages and maintain customer satisfaction.
- **Personalized Recommendations:** By analyzing rules with high confidence and lift values, businesses can enhance recommendation systems. **For instance, suggesting COFFEE when customers purchase SUGAR JARS can lead to higher conversion rates in online stores.**

- **Product Bundling:** For itemsets like DECORATION, METAL, WOBBLY RABBIT (Lift: 3.3556), creating pre- packaged bundles or themed kits can encourage customers to purchase more items together, thus increasing average order value.

❖ Additional Business Analysis in the Retail Store using Apache Pig

✓ Country-wise Transaction Count:

The purpose of this analysis is to calculate the number of unique transactions for each country. This is useful to understand how many transactions have been conducted in each country.

```
raw`data = LOAD '/home/hadoop/Downloads/Dataset(1).csv' USING PigStorage(';')
    AS ( BillNo:chararray,
        Items:chararray,
        Date:chararray,
        Price:float,
        Quantity:int,
        CustomerID:chararra
        y,
        Country:chararray
    );

filtered`data = FILTER raw`data BY

BillNo != 'BillNo'; country`bills = GROUP

filtered`data BY (Country, BillNo);

country`unique`trans = FOREACH country`bills GENERATE

group.Country as Country; country`counts = GROUP country`unique`trans

BY Country;

final`count = FOREACH country`counts
    GENERATE group as Country,
    COUNT(country`unique`trans) as

TransactionCount; ordered`result = ORDER

final`count BY TransactionCount DESC;

STORE ordered`result INTO '/home/hadoop/Downloads/country`wise' USING
PigStorage(',');
```

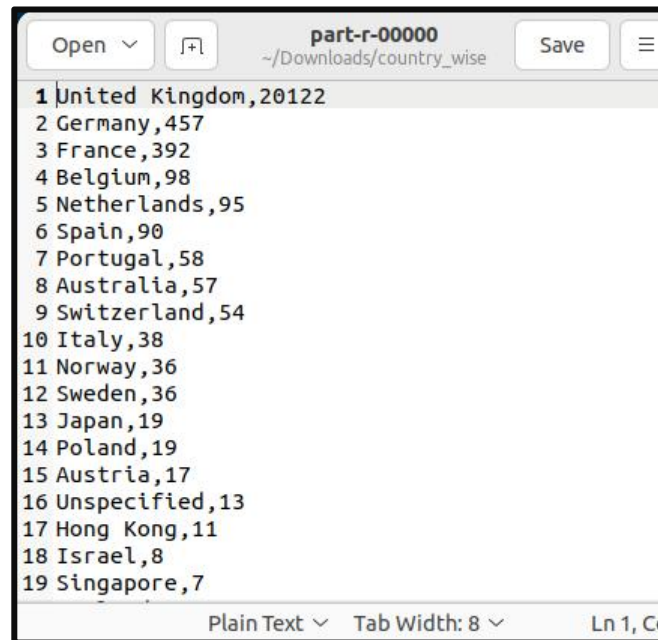


Figure 11: Country Wise in-store transactions

✓ Average Basket Size per Country:

This analysis calculates the average number of items purchased in each transaction for every country, which helps identify the typical basket size in different countries.

```

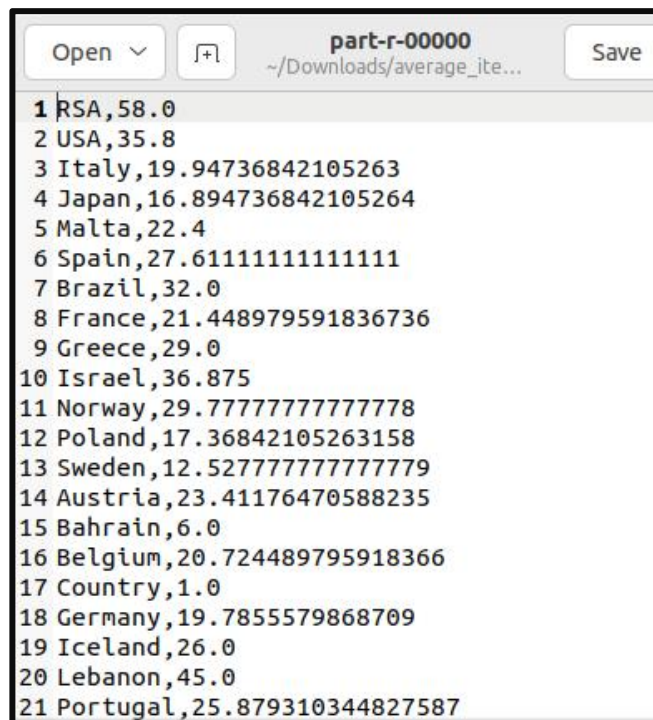
retail`data = LOAD '/home/hadoop/Downloads/Dataset.csv' USING PigStorage('\t')
AS ( BillNo:chararray,
    ItemName:chararra
y, Quantity:int,
    Date:chararray,
    Price:float,
    CustomerID:chararra
y,
    Country:chararray
);

grouped`by`transaction = GROUP retail`data BY (BillNo,
Country); count`items`per`transaction = FOREACH
grouped`by`transaction GENERATE
    FLATTEN(group) AS (BillNo, Country),
    COUNT(retail`data.ItemName) AS
    CountOfItemName;

grouped`by`country = GROUP count`items`per`transaction BY
Country; average`item`count`per`country = FOREACH
grouped`by`country GENERATE
    group AS Country,
    AVG(count`items`per`transaction.CountOfItemName) AS AverageBasketSize;

```

```
STORE average`item`count`per`country
INTO '/home/hadoop/Downloads/average`item`count`by`country' USING
PigStorage(',');
```



Rank	Country	Average Basket Size
1	RSA	58.0
2	USA	35.8
3	Italy	19.94736842105263
4	Japan	16.894736842105264
5	Malta	22.4
6	Spain	27.61111111111111
7	Brazil	32.0
8	France	21.448979591836736
9	Greece	29.0
10	Israel	36.875
11	Norway	29.77777777777778
12	Poland	17.36842105263158
13	Sweden	12.527777777777779
14	Austria	23.41176470588235
15	Bahrain	6.0
16	Belgium	20.724489795918366
17	Country	1.0
18	Germany	19.7855579868709
19	Iceland	26.0
20	Lebanon	45.0
21	Portugal	25.879310344827587

Figure 12: Average Basket size

✓ Most Popular Items per Country:

This script identifies the most popular items sold in each country by calculating the total quantity of each item sold in each country and selecting the top 5 items.

```
retail`data = LOAD '/home/hadoop/Downloads/Dataset.csv' USING PigStorage('\t')
AS ( BillNo:chararray,
    ItemName:chararray,
    Quantity:int,
    Date:chararray,
    Price:float,
    CustomerID:chararra
y,
    Country:chararray
);

grouped`by`country`item = GROUP retail`data BY (Country,
ItemName); count`items`by`country = FOREACH
grouped`by`country`item GENERATE
    group.Country AS Country,
    group.ItemName AS ItemName,
    SUM(retail`data.Quantity) AS TotalQuantity;

sorted`by`popularity = ORDER count`items`by`country BY
TotalQuantity DESC; grouped`by`country = GROUP
```

```
sorted`by`popularity BY Country; top`5`items`by`country =
```

```
FOREACH grouped`by`country –
```

```
    sorted`data` = LIMIT sorted`by`popularity 5;
```

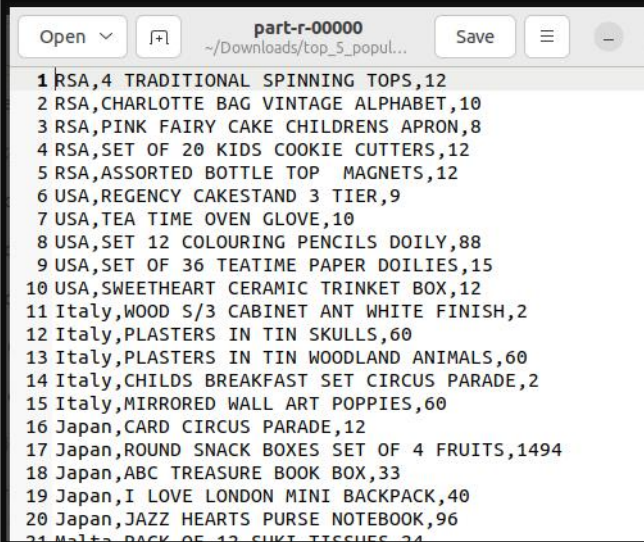
```
    GENERATE FLATTEN(sorted`data`);
```

```
“
```

```
STORE top`5`items`by`country
```

```
INTO '/home/hadoop/Downloads/top`5`popular`items`by`country' USING
```

```
PigStorage(',');
```



Rank	Country	Item Name	Quantity
1	RSA	4 TRADITIONAL SPINNING TOPS	12
2	RSA	CHARLOTTE BAG VINTAGE ALPHABET	10
3	RSA	PINK FAIRY CAKE CHILDRENS APRON	8
4	RSA	SET OF 20 KIDS COOKIE CUTTERS	12
5	RSA	ASSORTED BOTTLE TOP MAGNETS	12
6	USA	REGENCY CAKESTAND 3 TIER	9
7	USA	TEA TIME OVEN GLOVE	10
8	USA	SET 12 COLOURING PENCILS DOILY	88
9	USA	SET OF 36 TEATIME PAPER DOILIES	15
10	USA	SWEETHEART CERAMIC TRINKET BOX	12
11	Italy	WOOD S/3 CABINET ANT WHITE FINISH	2
12	Italy	PLASTERS IN TIN SKULLS	60
13	Italy	PLASTERS IN TIN WOODLAND ANIMALS	60
14	Italy	CHILDS BREAKFAST SET CIRCUS PARADE	2
15	Italy	MIRRORED WALL ART POPPIES	60
16	Japan	CARD CIRCUS PARADE	12
17	Japan	ROUND SNACK BOXES SET OF 4 FRUITS	1494
18	Japan	ABC TREASURE BOOK BOX	33
19	Japan	I LOVE LONDON MINI BACKPACK	40
20	Japan	JAZZ HEARTS PURSE NOTEBOOK	96
21	Malta	PACK OF 12 SWEET TISSUES	24

Figure 13: Popular items sold

✓ Revenue by Country:

This analysis calculates the total revenue generated by each country by multiplying the quantity and price of each item sold, and then summing the revenue per country.

```
retail`data` = LOAD '/home/hadoop/Downloads/Dataset.csv' USING PigStorage('\t')
    AS ( BillNo:chararray,
        ItemName:chararra
        y, Quantity:int,
        Date:chararray,
        Price:float,
        CustomerID:chararra
        y,
        Country:chararray
    );
```

```

revenue`data = FOREACH retail`data
    GENERATE Country,
    (Quantity * Price) as revenue;

grouped`by`country = GROUP revenue`data

BY Country;

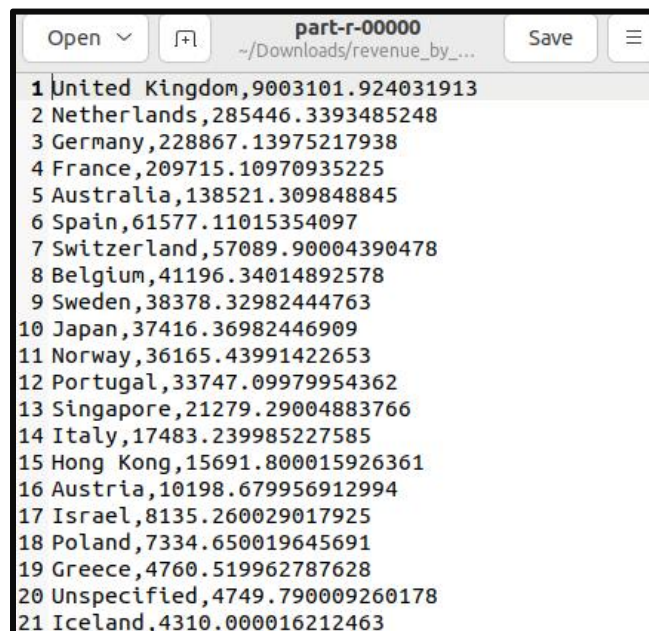
country`revenue = FOREACH grouped`by`country
    GENERATE group as Country,
    SUM(revenue`data.revenue) as TotalRevenue;

ordered`results = ORDER country`revenue BY

TotalRevenue DESC;

STORE ordered`results INTO '/home/hadoop/Downloads/revenue`by`country' USING
PigStorage(',');

```



Rank	Country	Revenue
1	United Kingdom	9003101.924031913
2	Netherlands	285446.3393485248
3	Germany	228867.13975217938
4	France	209715.10970935225
5	Australia	138521.309848845
6	Spain	61577.11015354097
7	Switzerland	57089.90004390478
8	Belgium	41196.34014892578
9	Sweden	38378.32982444763
10	Japan	37416.36982446909
11	Norway	36165.43991422653
12	Portugal	33747.09979954362
13	Singapore	21279.29004883766
14	Italy	17483.239985227585
15	Hong Kong	15691.800015926361
16	Austria	10198.679956912994
17	Israel	8135.260029017925
18	Poland	7334.650019645691
19	Greece	4760.519962787628
20	Unspecified	4749.790009260178
21	Iceland	4310.000016212463

Figure 14: Revenue generated in store location wise

❖ Summary of the analyses done in Apache Pig:

The analysis conducted using Apache Pig on the retail store data has provided notable points into customer purchasing behavior and sales performance across different regions. Key findings from the analysis include:

- **Revenue Analysis:** The United Kingdom (UK) generated the highest revenue among all countries, indicating strong market performance and a high volume of sales in the region.
- **Popular Items per Country:** Popular items were analyzed by country, revealing the most frequently purchased products in each region.
- **Average Basket Size:** An average basket size was found that, in South Africa (RSA), was the highest in contrast with the United States of America. This implies that lot of opportunities are then generated toward targeted promotions and upselling in these branches.

- **Transaction Volume:** The UK, also had the highest number of transactions, followed by Germany, which suggests a high level of customer engagement and frequent purchases in these countries, providing an insight into regional customer behavior and potential areas for business expansion.

❖ Conclusion:

The analysis performed with the Hadoop framework proves that big data tools can draw meaningful insights from large datasets. By using MapReduce, we efficiently processed and analyzed transaction data to discover patterns like frequent itemsets, support, confidence, and lift values. These insights will guide business decisions such as optimizing inventory, targeted marketing, and customer segmentation, which enhances overall business performance. The study emphasizes the importance of association rule mining in product relationship identification and improving sales strategies.

❖ Limitations:

While the results of this analysis are insightful, there are certain limitations to be considered:

- The calculated confidence and lift values suggest some item associations but do not guarantee causality, limiting their predictive capabilities.
- Data preprocessing was simplified, and errors or inconsistencies in the original data might have affected the results.
- External factors such as seasonality, promotions, and market trends were not considered, which could have impacted itemset relationships.

Future research and implementation are expected to overcome these limitations by incorporating additional contextual data and using advanced algorithms for better scalability and accuracy.

❖ References:

1. **Apriori Algorithm Explained — Association Rule Mining — Finding Frequent Itemset — Edureka.** Available at: <https://www.youtube.com/watch?v=guVvtZ7ZClw>
2. **Market Basket Analysis Algorithms with MapReduce.** Woo, Jongwook. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 3.6 (2013): 445-452. Available at: <https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1107>
3. **Market Basket Analysis Dataset.** Available at: <https://www.kaggle.com/datasets/aslanahmedov/market-basket-analysis>
4. **Market Basket Analysis Algorithm with MapReduce Using HDFS.** Available at: <https://library.ndsu.edu/ir/bitstream/handle/10365/26367/Market>
5. **Market Basket Analysis in R with Hadoop.** Available at: <https://stackoverflow.com/questions/41172825/market-basket-analysis-in-r-with-hadoop>
6. **Market Basket Analysis Algorithm with Map/Reduce of Cloud Computing.** Woo, Jongwook, and Yuhang Xu. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2011. Available at: Here

