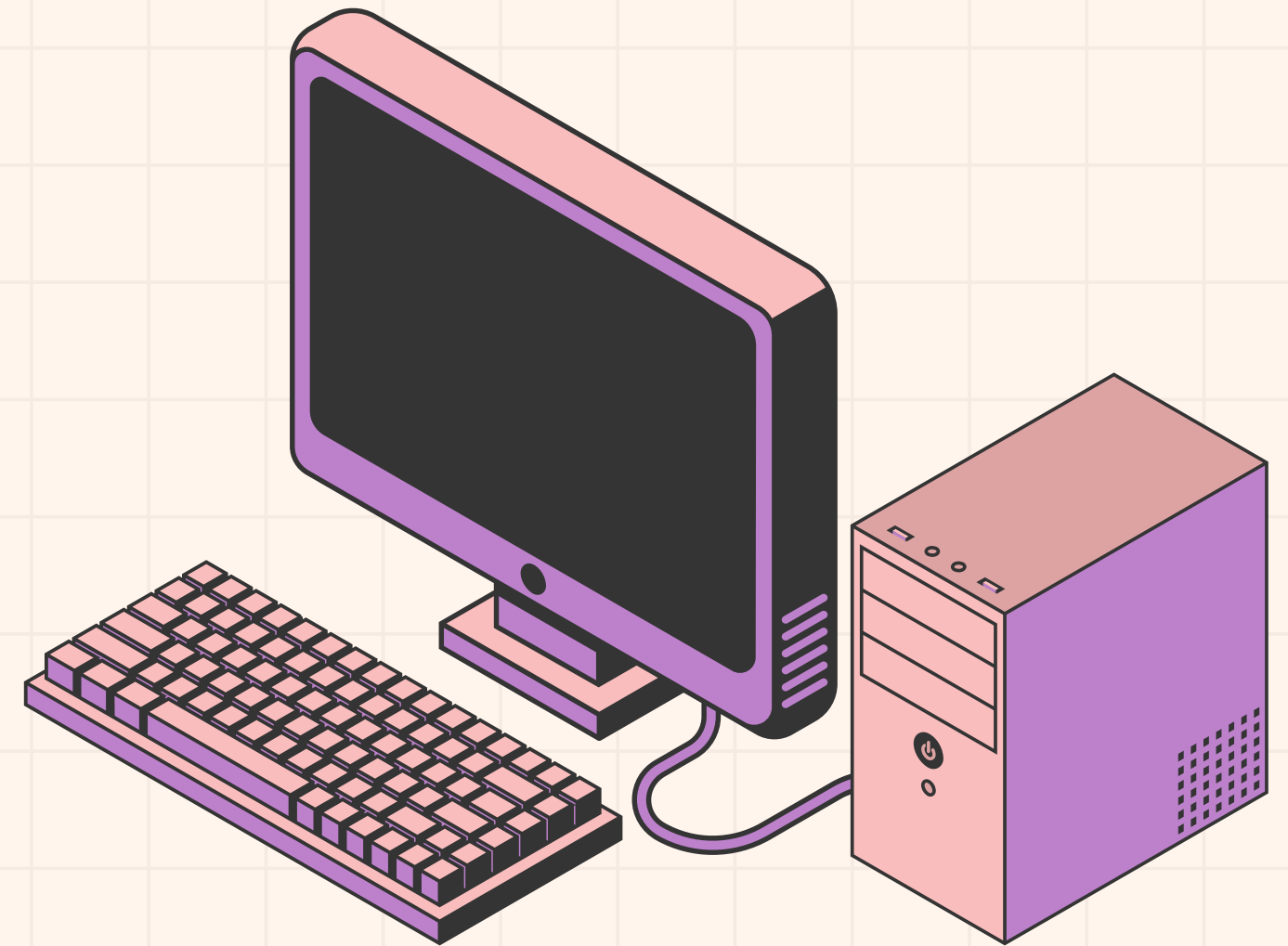


# CS431

BIG DATA  
ANALYTICS

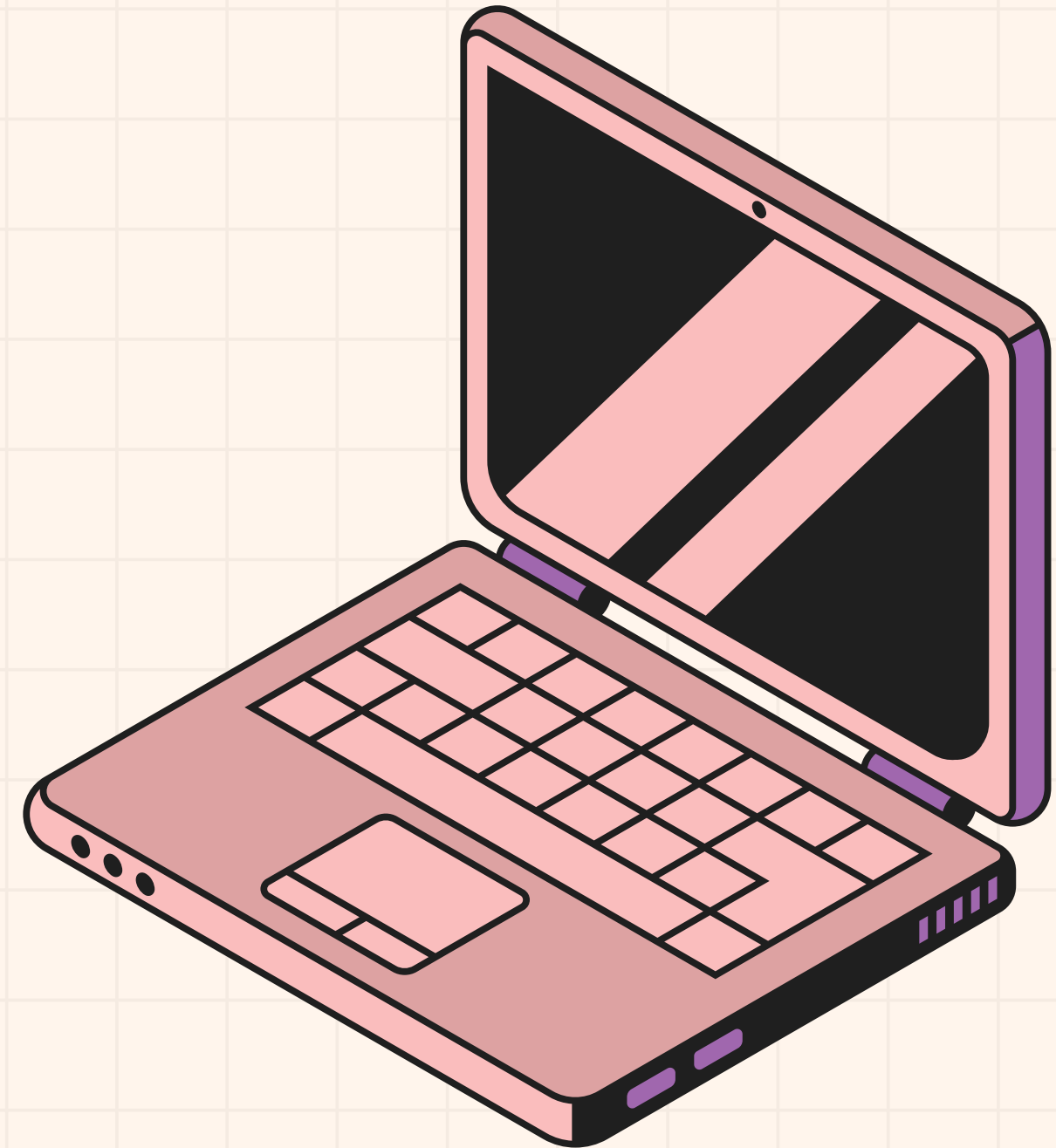
## Customer Segmentation on Online Retail Data

- Shrey Sinha 2101AI31
- Lalit Chandra Routhu 2101AI17
- Swapnil Srivastava 2101AI34
- Aryan Dabad 2101AI36
- Sidhant Senapati 2101AI38



# INDEX

1. Dataset Insights
2. Preprocessing
3. Purchase Patterns
4. Clustering
5. Market Basket Analysis
6. References



# THE DATASET

Variables Table

| Variable Name | Role    | Type        | Description   | Units    | Missing Values |
|---------------|---------|-------------|---|----------|----------------|
| InvoiceNo     | ID      | Categorical | a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation |          | no             |
| StockCode     | ID      | Categorical | a 5-digit integral number uniquely assigned to each distinct product  |          | no             |
| Description   | Feature | Categorical | product name  |          | no             |
| Quantity      | Feature | Integer     | the quantities of each product (item) per transaction   |          | no             |
| InvoiceDate   | Feature | Date        | the day and time when each transaction was generated  |          | no             |
| UnitPrice     | Feature | Continuous  | product price per unit  | sterling | no             |
| CustomerID    | Feature | Categorical | a 5-digit integral number uniquely assigned to each customer  |          | no             |
| Country       | Feature | Categorical | the name of the country where each customer resides   |          | no             |

# THE DATASET

## Online Retail Data

A transactional data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail

## Example Record

| InvoiceNo | StockCode | Description                        | Quantity | InvoiceDate      | UnitPrice | CustomerID | Country        |
|-----------|-----------|------------------------------------|----------|------------------|-----------|------------|----------------|
| 536365    | 85123A    | WHITE HANGING HEART T-LIGHT HOLDER | 6        | 01-12-2010 08:26 | 2.55      | 17850      | United Kingdom |

# PREPROCESSING

- Filter out the header: Remove the first row of the input dataset, assuming it contains column headers.
- Format the timestamp: Add the missing seconds field to make the timestamp consumable by Hive.
- Calculate Total Price: Add a new column for each record that computes the total price as  $\text{Quantity} * \text{Unit Price}$ .
- Write to HDFS: Save the transformed data into a new HDFS subdirectory for further processing.

# PREPROCESSING

## Reducer Class

```
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class RetailSalesReducer extends Reducer<NullWritable, Text, NullWritable, Text> {
    @Override
    protected void reduce(NullWritable key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
        for (Text value : values) {
            context.write(NullWritable.get(), value);
        }
    }
}
```

## Mapper Class

```
public class RetailSalesMapper extends Mapper<LongWritable, Text, NullWritable, Text> {
    private static final String HEADER =
        "TransactionID,Date,Time,Quantity,UnitPrice";

    @Override
    protected void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
        String line = value.toString();
        if (line.equals(HEADER)) return; // Skip header

        String[] fields = line.split(",");
        if (fields.length < 5) return;

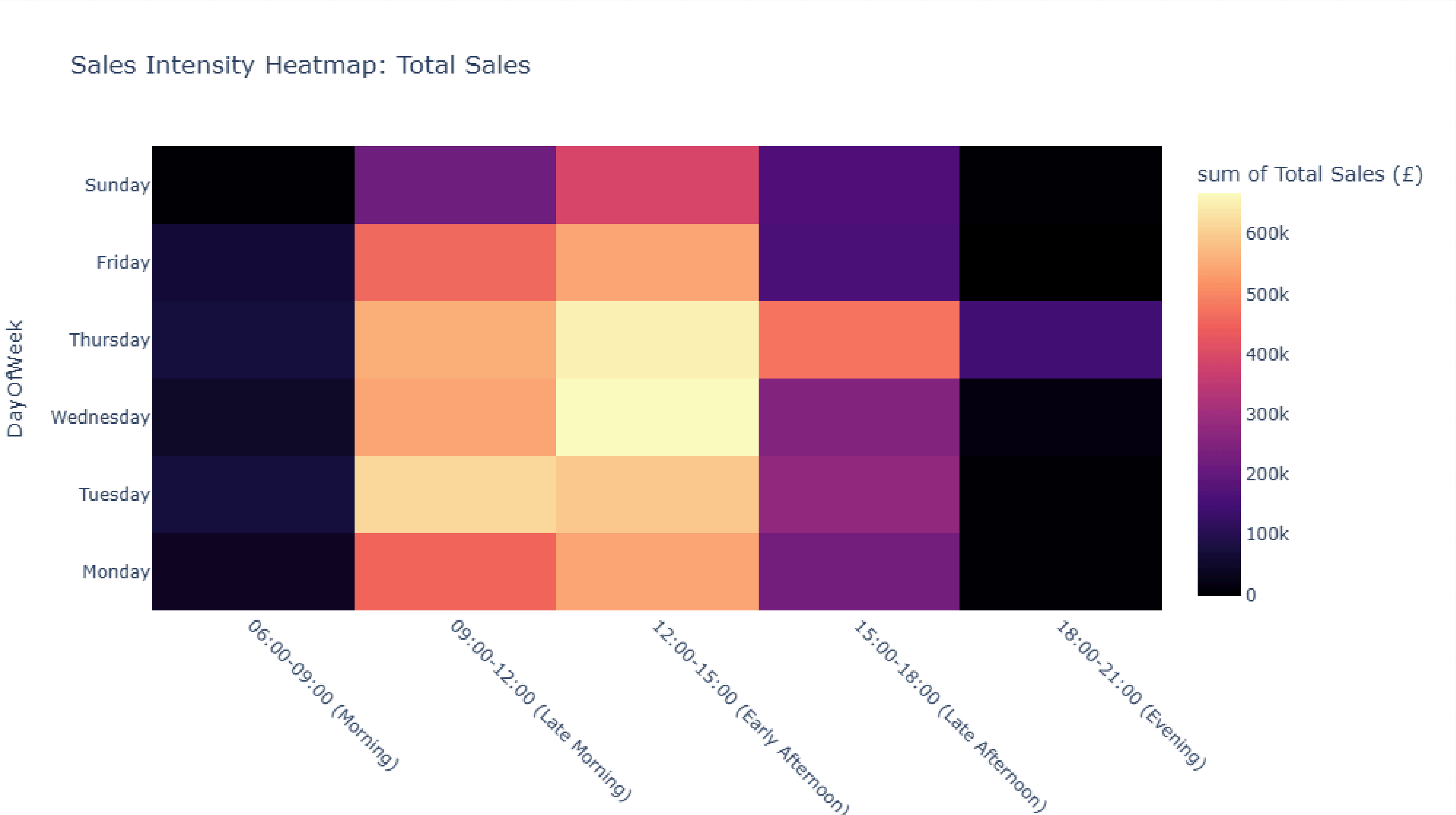
        try {
            String transactionId = fields[0];
            String date = fields[1];
            String time = fields[2];
            int quantity = Integer.parseInt(fields[3]);
            double unitPrice = Double.parseDouble(fields[4]);

            // Add seconds to time if missing
            if (time.split(":").length == 2) {
                time = time + ":00";
            }

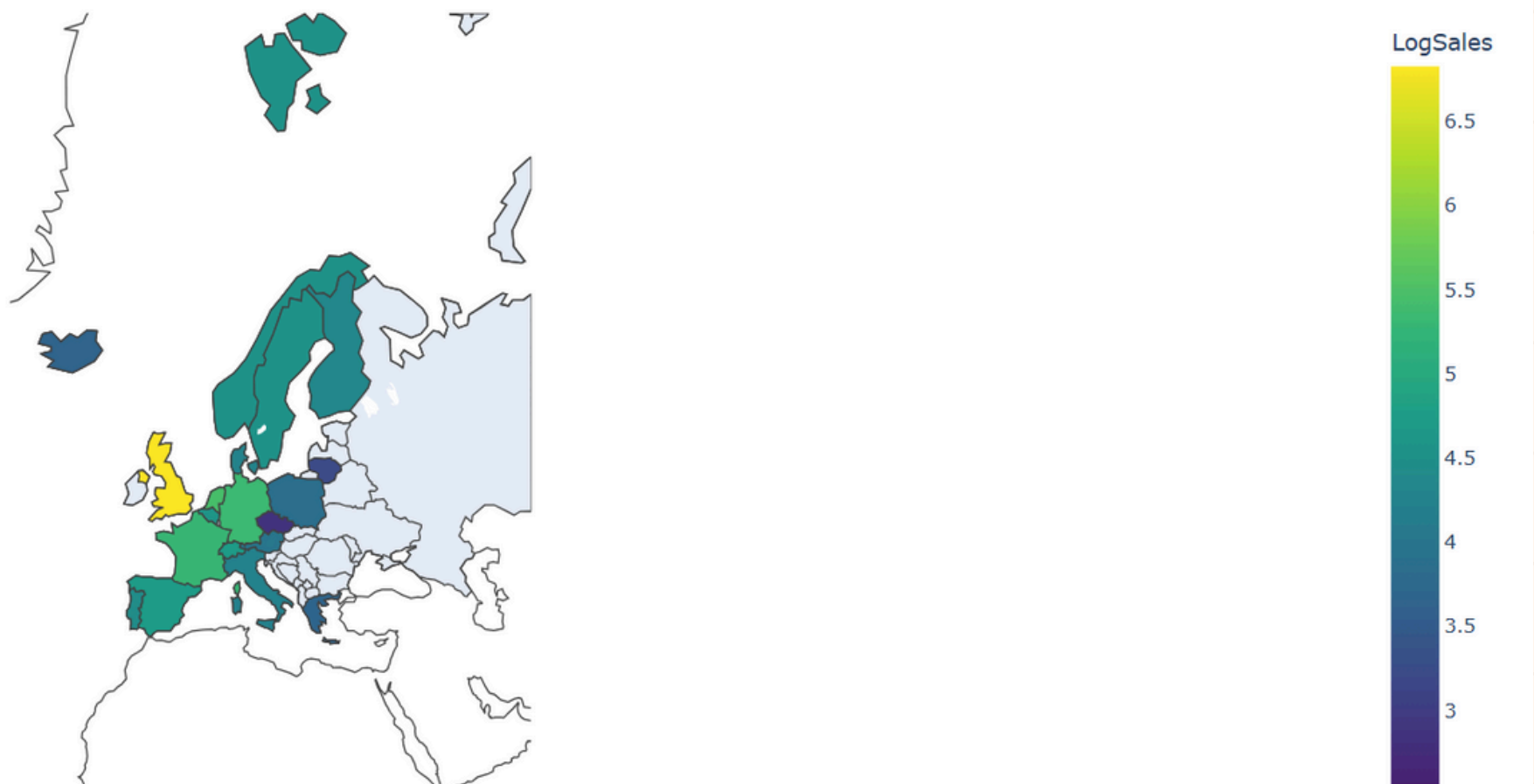
            // Calculate Total Price
            double totalPrice = quantity * unitPrice;

            // Write output as a single transformed record
            String transformedLine = String.join(",", transactionId, date, time,
                String.valueOf(quantity),
                String.valueOf(unitPrice),
                String.valueOf(totalPrice));
            context.write(NullWritable.get(), new Text(transformedLine));
        } catch (NumberFormatException e) {
            // Handle any parsing errors
        }
    }
}
```

# PURCHASE PATTERNS



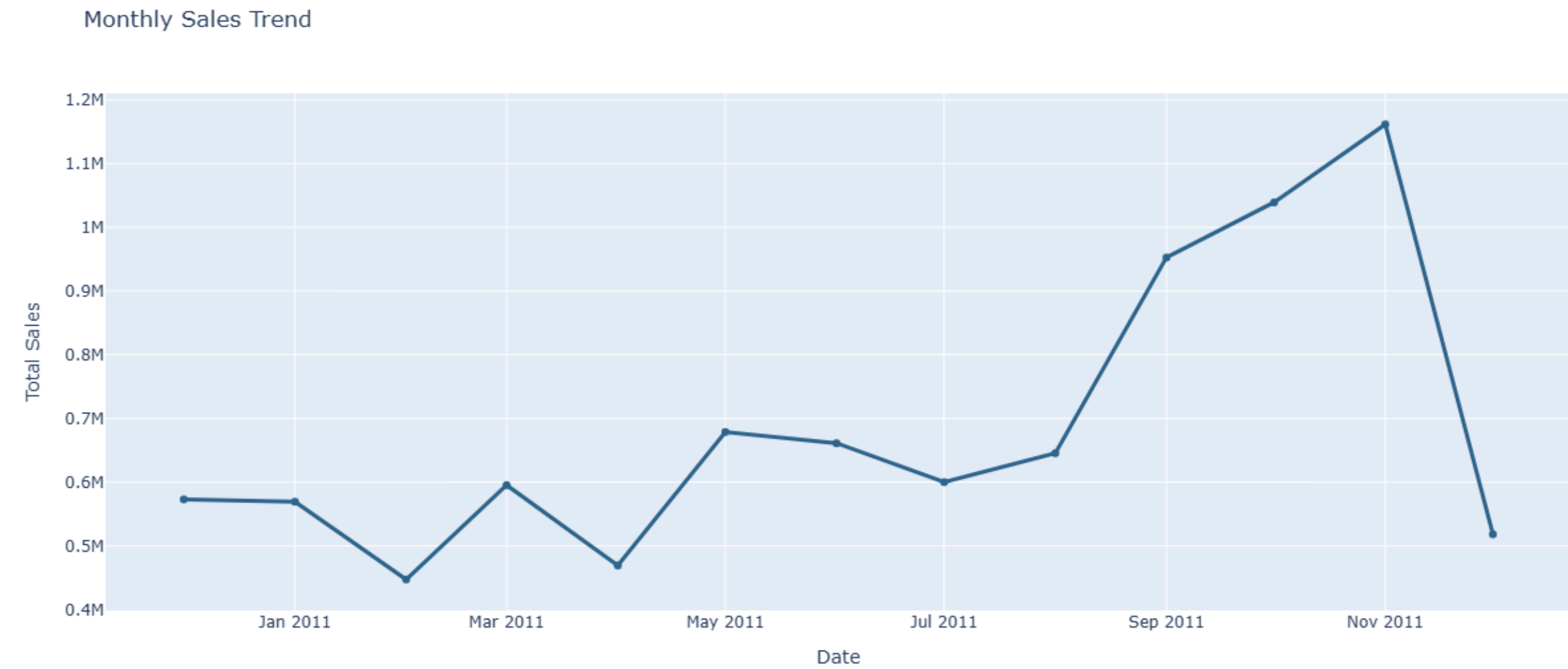
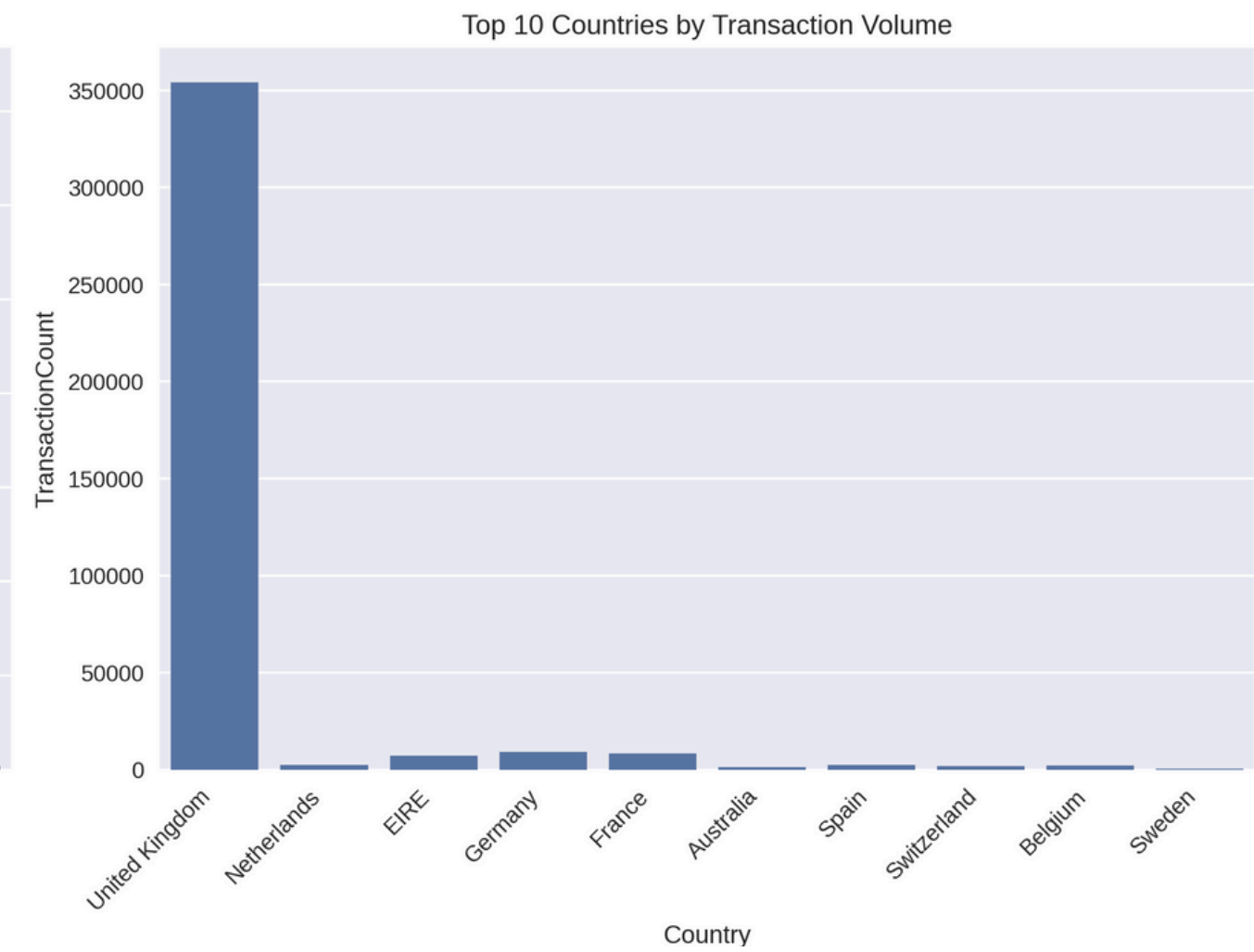
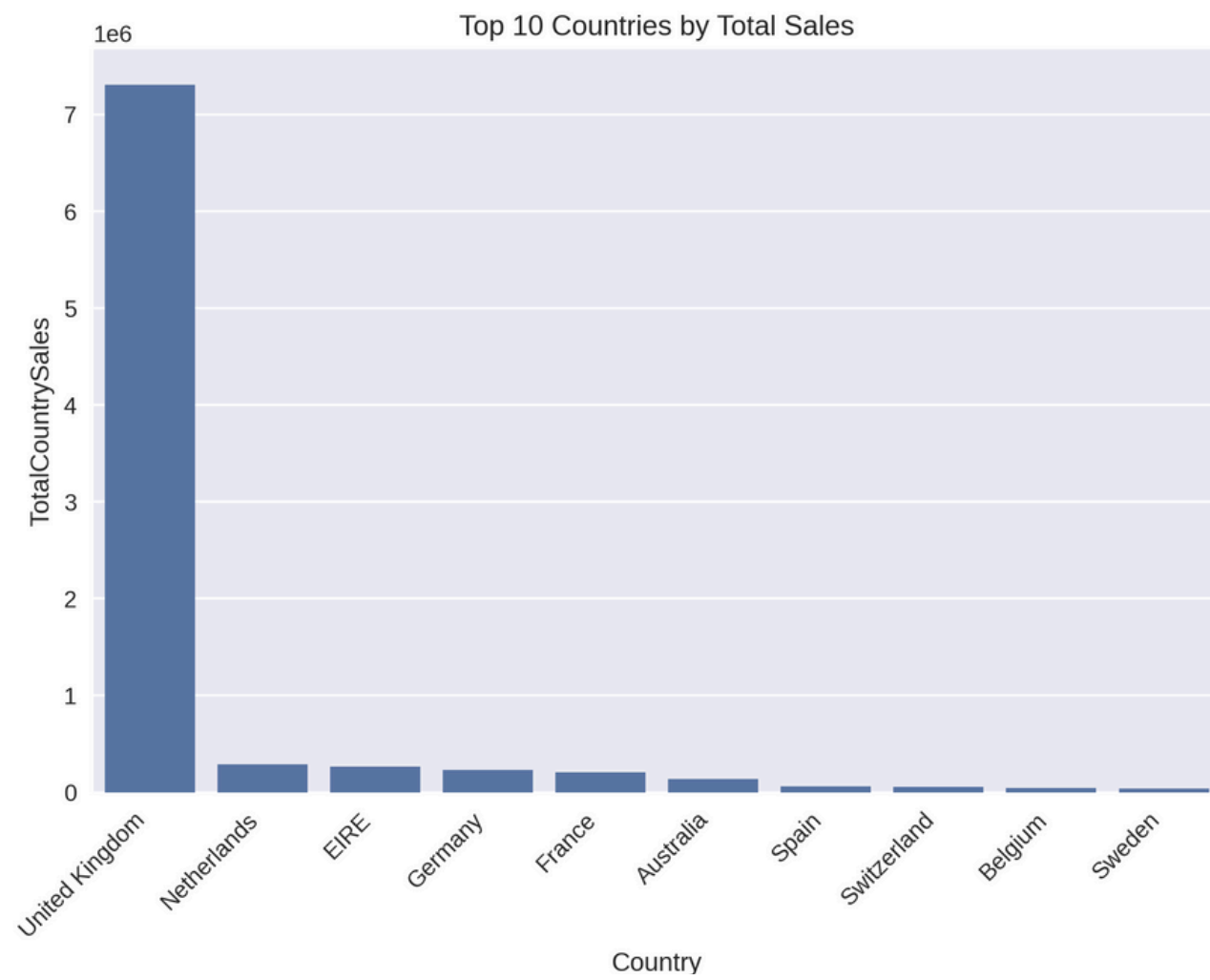
European Online Retail Sales (Log Scaled)



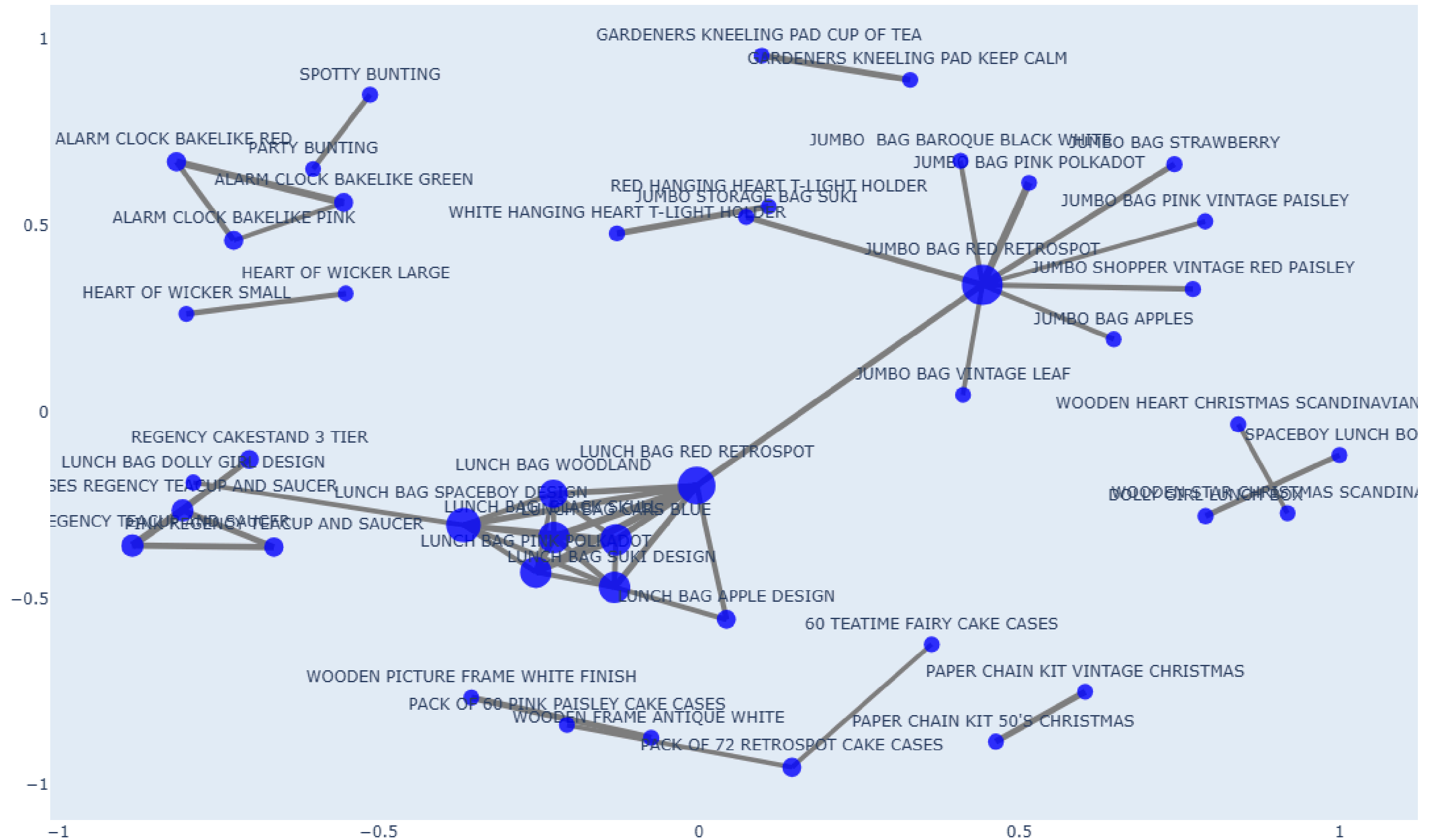
```
%sql
-- Sales Metrics by country for top 5 countries
WITH InvoiceAmount AS (Select country, customerid, invoiceno, count(distinct Stockcode) as NumItems, sum(totalprice) as InvoiceTotal from retailsales where
totalprice > 0 and Country in ('United Kingdom','Netherlands','EIRE','Germany','France') group by country,invoiceno,customerid)
select Country, count(distinct customerid) as NumCustomers, count(distinct invoiceno) as NumTransactions, ceil(avg(NumItems)) as AvgNumItems, ceil(min(InvoiceTotal
)) as MinAmtperCustomer, ceil(max(InvoiceTotal)) as MaxAmtperCustomer, ceil(avg(InvoiceTotal)) as AvgAmtperCustomer, ceil(std(InvoiceTotal)) as StdDevAmtperCustomer
from InvoiceAmount
group by Country
order by NumCustomers desc
```

| Country        | NumCustomers | NumTransactions | AvgNumItems | MinAmtperCustomer | MaxAmtperCustomer | AvgAmtperCustomer | StdDevAmtperCustomer |
|----------------|--------------|-----------------|-------------|-------------------|-------------------|-------------------|----------------------|
| United Kingdom | 3,920        | 18,018          | 27          | 1                 | 168,470           | 501               | 1,782                |
| Germany        | 94           | 457             | 20          | 3                 | 9,342             | 501               | 609                  |
| France         | 87           | 392             | 22          | 5                 | 8,896             | 535               | 685                  |
| Netherlands    | 9            | 94              | 26          | 3                 | 20,278            | 3,037             | 4,468                |
| EIRE           | 3            | 288             | 28          | 2                 | 16,775            | 985               | 1,613                |

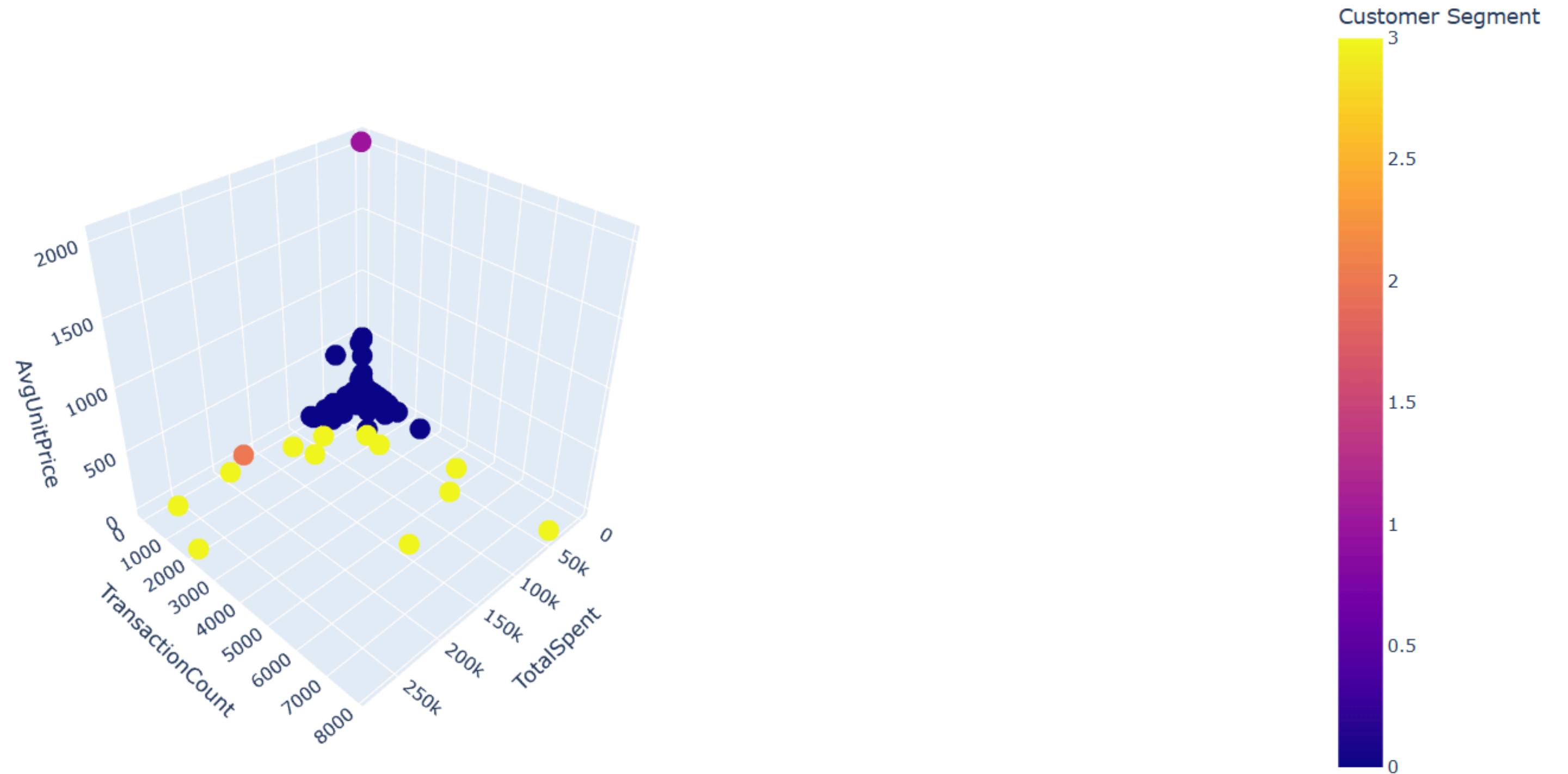




## Improved Product Co-purchase Network



# CUSTOMER SEGMENTATION



# ASSOCIATION RULE MINING

Used Spark MLlib's FP-Growth association rule mining algorithm to perform Market Basket Analysis

- Minimum support: 0.7% (to identify relevant frequent itemsets).
- Confidence level: 80%

[2319,2238] => [2320], Confidence: 0.8857

[2266,2072] => [2238], Confidence: 0.8571

## **StockCode Analysis:**

- 2319: Items are primarily notepads, shopping lists, and notebooks.
- 2238: Items are mostly lunch bags.
- 2320: Items are jumbo grocery bags.

## **Customer Behavior:**

- Customers purchasing notepads/notebooks and lunch bags also purchase jumbo grocery bags more than 88% of the time.