

# Automotive Sales Analytics Platform

## Project Idea:

The project's main goal is to derive useful insights from an automotive company's sales data by utilizing exploratory data analysis (EDA) and recency-frequency-monetary (RFM) analysis methodologies. In order to provide practical suggestions for improving customer happiness, inventory control, and sales tactics, ASAP looks for patterns, trends, and customer behavior.

To gain a more in-depth understanding of the dataset, we'll employ the big data idea of the 5Vs.

### Volume:

The dataset has 18 rows and 18 properties, resulting in a large amount of data. As we investigate order fulfillment in an automobile firm, the amount of the dataset is critical to understanding patterns, trends, and optimizing the order fulfillment process.

### Velocity:

In the context of order fulfillment, velocity, or the rate at which data is generated, is very important. The steady intake of sales data from diverse clients and areas helps to the dataset's ongoing growth. This high data velocity provides useful insights for the automaker to modify and streamline their order fulfillment tactics in real-time.

### Variety:

The dataset's properties are numerous, including order numbers, quantity ordered, pricing, customer details, and more. This structured data with various categories and established formats emphasizes the variation inherent in a car company's order fulfillment procedure.

### Veracity:

Veracity, which emphasizes the dependability and quality of data, is crucial for making educated judgments in order fulfillment. The dataset is thoroughly preprocessed to ensure correctness and reliability. By resolving any discrepancies or inaccuracies, we want to improve the data's authenticity, resulting in more trustworthy insights.

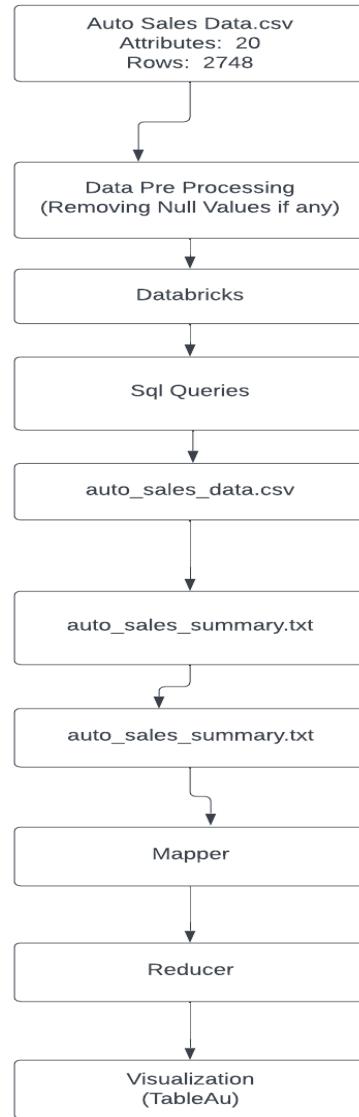
### Value:

The ultimate objective is to maximize order fulfillment by taking advantage of analytics to extract value from the dataset. The car manufacturer is able to increase customer value, increase efficiency, and make well-informed decisions thanks to thorough analysis and insights from the data.

## **Tools and Technologies Used:**

- **Microsoft Excel:** Microsoft Excel supports the storage of a wide range of data types within spreadsheet cells or ranges.
- **Databricks:** It is a tool for transforming complex data into formats that can be reused.
- **MySQL:** MySQL is a relational database management system (RDBMS) that makes it easier to extract useful information from a variety of queries.
- **Hadoop:** Based on Java, Hadoop uses simple programming techniques to distribute large datasets among machine clusters.
- **Map& Reduce:** Mapper tasks in Hadoop are in charge of breaking down jobs into smaller subtasks and handling intermediate data, whereas Reducers are in charge of organizing and condensing this intermediate data into more manageable units during the data processing pipeline.

## Architecture Diagram:



## Architecture Summary:

The auto sales dataset includes 2748 records that document a variety of sales information. It includes entries such as ORDERNUMBER, QUANTITYORDERED, PRICEEACH, and more. Our goal is to use SQL queries to import and extract important insights by utilizing Databricks. The output file, called "auto\_sales.csv" and saved as "auto\_sales.txt," will serve as a primary source for our research projects involving cars. By putting the "Map Reduce" technique into practice, we can further hone our analysis and ensure accuracy and precision when assessing auto sales trends and patterns.

## **Goals:**

The following are the five objectives and their corresponding goals.

1. To calculate evaluate the average days since the last order in different cities.
2. To understand the distribution of sales across different deal sizes. To find out the total average amount with respective Transfers.
3. To Identify the top customers based on the quantity of products purchased.
4. To Examine the trend in sales over different months and years.
5. To Analyze the average price of products sold in different countries.
6. To Understand the contribution of each product line to the total sales.

## **Project Description:**

### **1) Project Setup:**

#### **Setting up the Project Environment:**

##### **1. Installing VS Code and Java Extensions:**

Installing VS Code as well as necessary Java modules. For coding, debugging, and project management, VS Code will be your primary Integrated Development Environment (IDE). To facilitate easy Java development, ensure that Java-related capabilities are smoothly integrated within VS Code.

##### **2. Setting up the Java Environment:**

Check the accuracy of your Java installation on your machine. Confirm the presence and proper configuration of Java. Check that the system environment variables are correctly configured and that the Java Development Kit (JDK) is properly added and available.

## **Using the Automotive Sales Analytics Platform:**

1. **Dataset's Availability:** I have confirmed the dataset's availability from the designated source, ensuring proper access permissions and repository storage.
2. **Dataset Structure Evaluation:** The dataset structure has been assessed, with columns checked for accuracy, data types verified, missing values handled, clear column labels ensured, and unique IDs and linkages between tables identified.
3. **Data Analysis:** We have successfully loaded the dataset, produced basic statistics, and conducted preliminary investigations for data analysis. To evaluate the dataset's structure and acquire insights into its content and properties, data quality checks, visualizations, and descriptive analysis were performed.

## **Data Retrieval:**

1. We have used VS Code for Java Programming.
2. Initialize a new Java file within Visual Studio Code (VS Code) for programming purposes.

3. **Map Reduce Compatibility:** After extracting data with Java, structure it into ordered key-value pairs to ensure interoperability with Hadoop MapReduce. To ease smooth use of MapReduce tools, save the processed files in a standard, readable format such as CSV.

## **Data Preprocessing:**

Inspect for missing data and plan how to handle it, taking into account imputation or removal. Remove duplicate entries to ensure data consistency. Filters should be used to extract relevant subsets that are aligned with project objectives.

## **Hadoop MapReduce:**

Make Java MapReduce applications to work with preprocessed data. Create a Mapper function to extract key-value pairs. For data processing and aggregation, use a Reducer function. Ensure that the output formatting is optimum for analysis and visualization.

## **Data post-processing and Analysis:**

Perform a thorough examination of MapReduce outputs to reveal patterns and useful insights. Recognize key data patterns and trends in the processed data. Perform further computations and filtering to ensure compliance with project-specific standards. Learn about intricate patterns and trends to make more informed decisions. Conduct further analyses to get useful information from the improved data.

## **Visual Presentation:**

For Tableau compatibility, save processed data in CSV format. Import into Tableau for interactive visual representation. Create bespoke visuals that are in line with project goals. Create charts that compare bridge condition, age, traffic impact, type categorization, and load. Use Tableau for detailed data visualization.

## **Result Summary:**

**Goal 1:** To calculate evaluate the average days since the last order in different cities.

Screenshot of the Quokka IDE interface showing the MyMapper.java file in the editor.

File Path: mapreduce-mvn-main/src/main/java/com/mycompany/app

```
1 package com.mycompany.app;
2
3 import java.io.IOException;
4 import org.apache.hadoop.io.IntWritable;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Mapper;
8
9
10 public class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
11
12     @Override
13     public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
14         String[] row = value.toString().split(",");
15         // Goal 1
16         context.write(new Text(row[1]), new IntWritable(Integer.parseInt(row[6])));
17     }
18 }
19 }
```

IDE Status Bar: Ln 15, Col 18 Spaces: 4 UTF-8 ⚡ Java ⚡ Quokka 859 PM 12/2/2023

Screenshot of the Quokka IDE interface showing the MyReducer.java file in the editor.

File Path: mapreduce-mvn-main/src/main/java/com/mycompany/app

```
1 package com.mycompany.app;
2
3 import java.io.IOException;
4 import org.apache.hadoop.io.IntWritable;
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Reducer;
7
8
9 public class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
10
11     @Override
12     public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
13         // Goal 1
14         int sum = 0;
15         int count = 0;
16
17         for (IntWritable value : values) {
18             sum += value.get();
19             count++;
20         }
21         int average = sum / count;
22         context.write(key, new IntWritable(average));
23     }
24 }
```

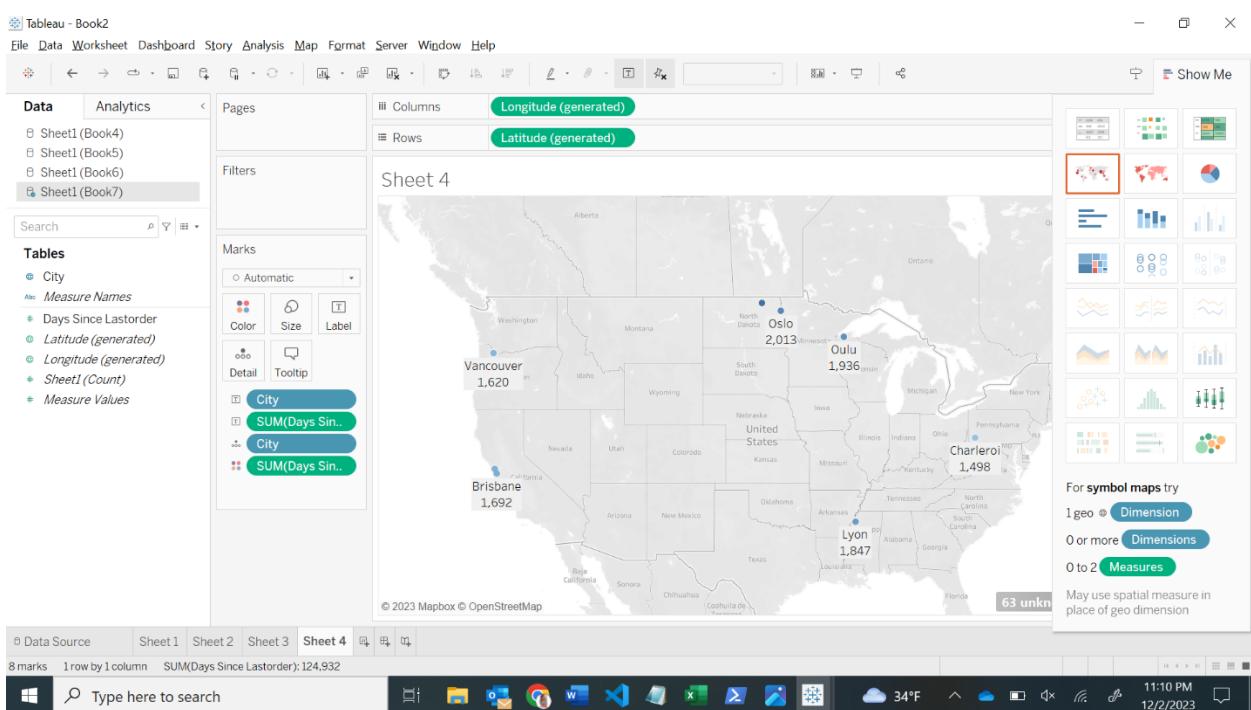
IDE Status Bar: Ln 22, Col 54 Spaces: 4 UTF-8 ⚡ Java ⚡ Quokka 9:00 PM 12/2/2023

Screenshot of a Windows file explorer window showing the directory structure of a MapReduce project and its output files.

```

data > output > part-r-00000
1 Aarhus 1648
2 Allentown 1378
3 Barcelona 2483
4 Bergamo 2283
5 Bergen 1468
6 Boras 1689
7 Boston 1921
8 Brickhaven 1713
9 Bridgewater 1489
10 Brisbane 1692
11 Bruxelles 1848
12 Burbank 1714
13 Burlingame 1918
14 Cambridge 1629
15 Charleroi 1498
16 Chatswood 1681
17 Cowes 2180
18 Dublin 1939
19 Espoo 1568
20 Frankfurt 1697
21 Genvee 1362
22 Glen Waverly 1689
23 Glendale 2358
24 Graz 1565
25 Helsinki 1745
26 Kobenhavn 1924
27 Koin 1480
28 Las Vegas 1770
29 Lille 1875
30 Liverpool 1404
31 London 2111
32 Lule 1319
33 Lyon 1847
34 Madrid 1888
35 Makati City 2080
36 Manchester 1803
37 Marsailles 1673

```



**Story:** Analyze the dataset to get the average number of days since the last order in each city. Investigate patterns to identify any geographic tendencies in client ordering behavior. Create an approach for calculating and evaluating the average number of days since the last order. Use city-level computations to gain insight about ordering frequency across multiple sites. Inform decision-making and improve operational efficiency by presenting findings.

**Goal 2:** To understand the distribution of sales across different deal sizes. To find out the total average amount with respective Transfers.

```

File Edit Selection View Go ... < > mapreduce-mvn-main
EXPLORER ... J MyReducer.java 1 J MyMapper.java 1 part-r-00000 J ViewCount.java
MAPREDUCE-MVN-MAIN
src > main > java > com > mycompany > app > J MyMapper.java > MyMapper > map(LongWritable, Text, Context)
6 import org.apache.hadoop.io.IntWritable;
7 import org.apache.hadoop.io.LongWritable;
8 import org.apache.hadoop.io.Text;
9 import org.apache.hadoop.mapreduce.Mapper;
10
11 public class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
12     // Goal 1
13     // @Override
14     public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
15         String[] row = value.toString().split("\t");
16         // Goal 1
17         // context.write(new Text(row[14]), new IntWritable(Integer.parseInt(row[6])));
18     }
19 }
20
21     // Goal 2
22     @Override
23     protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
24         try {
25             String[] columns = value.toString().split(regex:"\t");
26
27             if (columns.length >= 20) {
28                 String dealSize = columns[19].trim();
29                 double sales = Double.parseDouble(columns[4].trim());
30                 context.write(new Text(dealSize), new IntWritable((int) sales));
31             }
32         } catch (Exception e) {
33             // Handle parsing errors or other exceptions
34         }
35     }
36
37     // Goal 3
38     // @Override
39     // protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
40     //     try {
41     //         String[] columns = value.toString().split("\t");
42     //     }
43     // }
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

```

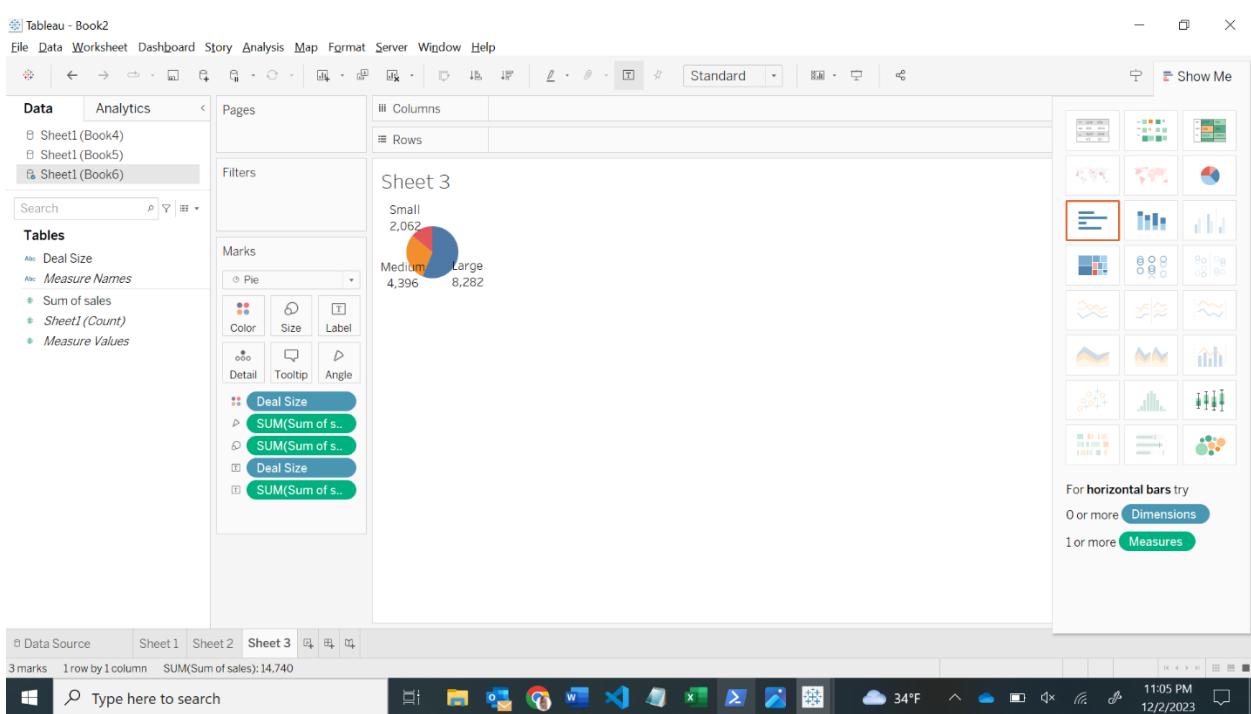
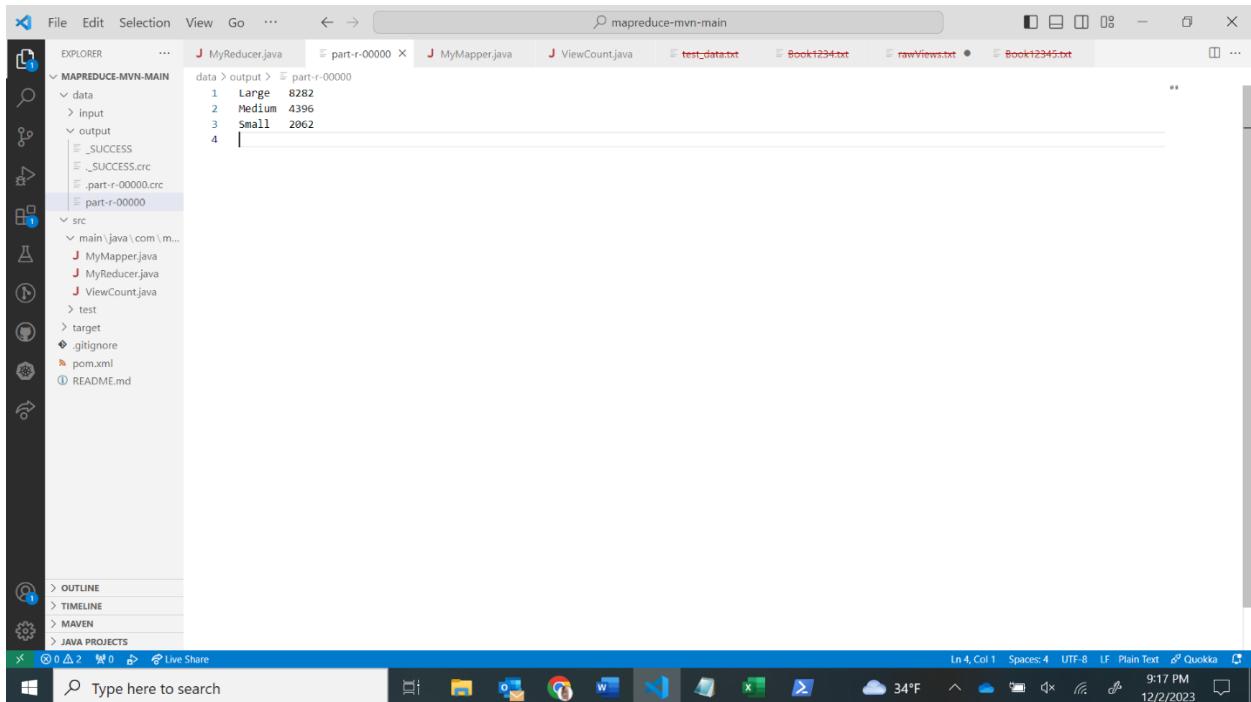
This screenshot shows the MyMapper.java code within an IDE. The code is a Mapper implementation that processes input rows split by tabs. It extracts deal sizes from the 19th column and sales from the 4th column, then writes them as key-value pairs where the key is the deal size and the value is a sales count. The code includes comments for goals 1, 2, and 3, and handles parsing exceptions.

```

File Edit Selection View Go ... < > mapreduce-mvn-main
EXPLORER ... J MyReducer.java 1 J MyMapper.java 1 part-r-00000 J ViewCount.java
MAPREDUCE-MVN-MAIN
src > main > java > com > mycompany > app > J MyReducer.java > MyReducer > reduce(Text, Iterable<IntWritable>, Context)
25
26     // sum / count;
27     // context.write(key, new IntWritable(average));
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

```

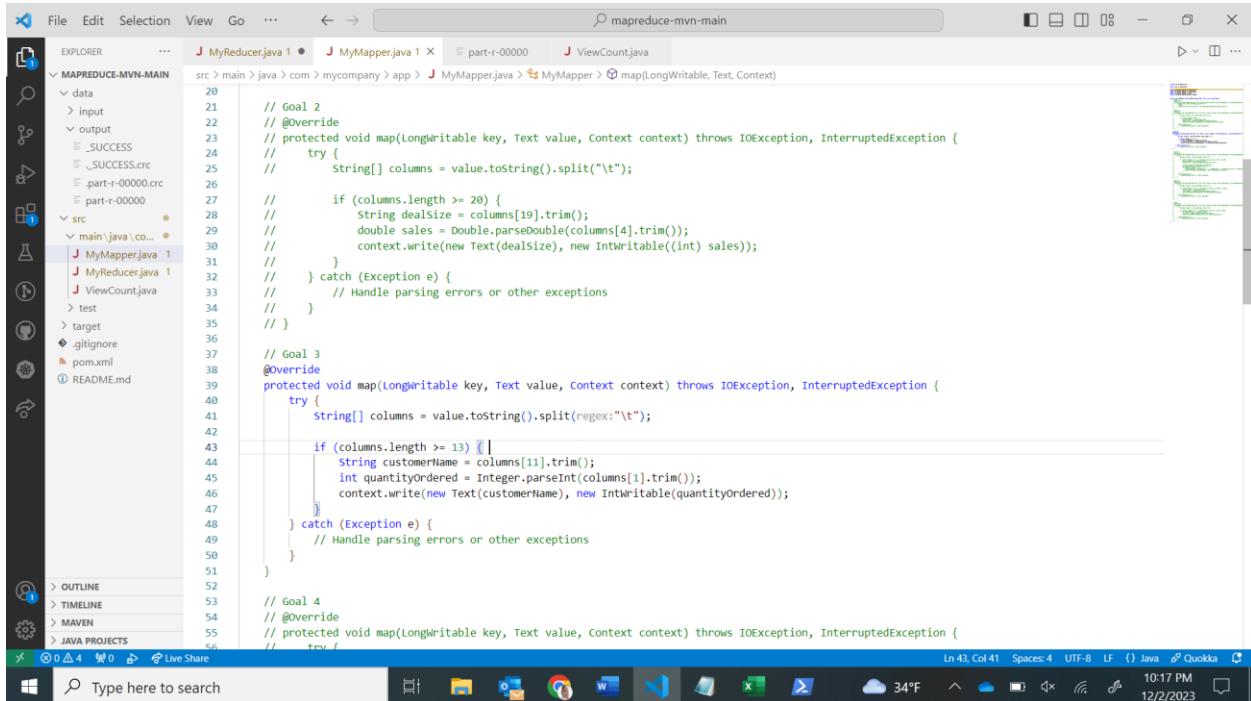
This screenshot shows the MyReducer.java code within an IDE. The code is a Reducer implementation that iterates over the values for each key (deal size). It calculates the average sales by summing all values and dividing by the count. The code includes comments for goals 2 and 3, and handles exceptions.



**Story:** Investigate the dataset to learn about the distribution of sales across various transaction sizes. Analyze transactional data to estimate the total average amount connected with various deal sizes and subsequent transfers. Develop insights into the financial landscape, identifying significant transaction categories and their impact on overall sales. Use this data to improve

transaction tactics, financial planning, and overall business performance. Present insights for informed decision-making and strategic financial management.

### Goal 3: To Identify the top customers based on the quantity of products purchased.



```
20
21 // Goal 2
22 // @Override
23 // protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
24 //     try {
25 //         String[] columns = value.toString().split("\t");
26 //
27 //         if (columns.length >= 20) {
28 //             String dealSize = columns[19].trim();
29 //             double sales = Double.parseDouble(columns[4].trim());
30 //             context.write(new Text(dealSize), new IntWritable((int) sales));
31 //         }
32 //     } catch (Exception e) {
33 //         // Handle parsing errors or other exceptions
34 //     }
35 //
36 // Goal 3
37 // @Override
38 // protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
39 //     try {
40 //         String[] columns = value.toString().split(regex:"\t");
41 //
42 //         if (columns.length >= 13) {
43 //             String customerName = columns[11].trim();
44 //             int quantityOrdered = Integer.parseInt(columns[1].trim());
45 //             context.write(new Text(customerName), new IntWritable(quantityOrdered));
46 //         }
47 //     } catch (Exception e) {
48 //         // Handle parsing errors or other exceptions
49 //     }
50 // }
51 //
52 // Goal 4
53 // @Override
54 // protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
55 //     try {
56 // 
```

File Edit Selection View Go ... ↶ ↷ mapreduce-mvn-main

EXPLORER MAPREDUCE-MVN-MAIN

- data
  - > input
  - < output
    - \_SUCCESS
    - \_SUCCESS.crc
    - .part-r-00000.crc
    - .part-r-00000
- src
  - main.java\com\mycompany\app
    - | J MyMapper.java 1
    - | J MyReducer.java 1
    - | J ViewCount.java
  - > test
  - > target
  - |.gitignore
  - pom.xml
  - README.md

OUTLINE TIMELINE MAVEN JAVA PROJECTS

```

36 // for (IntWritable value : values) {
37 //     totalsales += value.get();
38 //     count++;
39 //
40 //
41 //     if (count > 0) {
42 //         int averageSales = totalsales / count;
43 //         context.write(key, new IntWritable(averageSales));
44 //     }
45 // } catch (Exception e) {
46 //     // Handle exceptions
47 // }
48 //
49 }

50 /**
51 * Goal 3
52 * @Override
53 * protected void reduce(Text key, Iterable<IntWritable> values, Context context)
54 * throws IOException, InterruptedException {
55 * try {
56 *     int totalQuantity = 0;
57 *
58 *     for (IntWritable value : values) {
59 *         totalQuantity += value.get();
60 *     }
61 *
62 *     context.write(key, new IntWritable(totalQuantity));
63 * } catch (Exception e) {
64 *     // Handle exceptions
65 * }
66 *
67 }

68 // Goal 4
69 // @Override
70 // protected void reduce(Text key, Iterable<IntWritable> values, Context context)
71 // throws IOException, InterruptedException {
72

```

Ln 52, Col 8 Spaces: 4 UTF-8 LF Markdown ⚡ Quokka 10:17 PM 12/2/2023

File Edit Selection View Go ... ↶ ↷ mapreduce-mvn-main

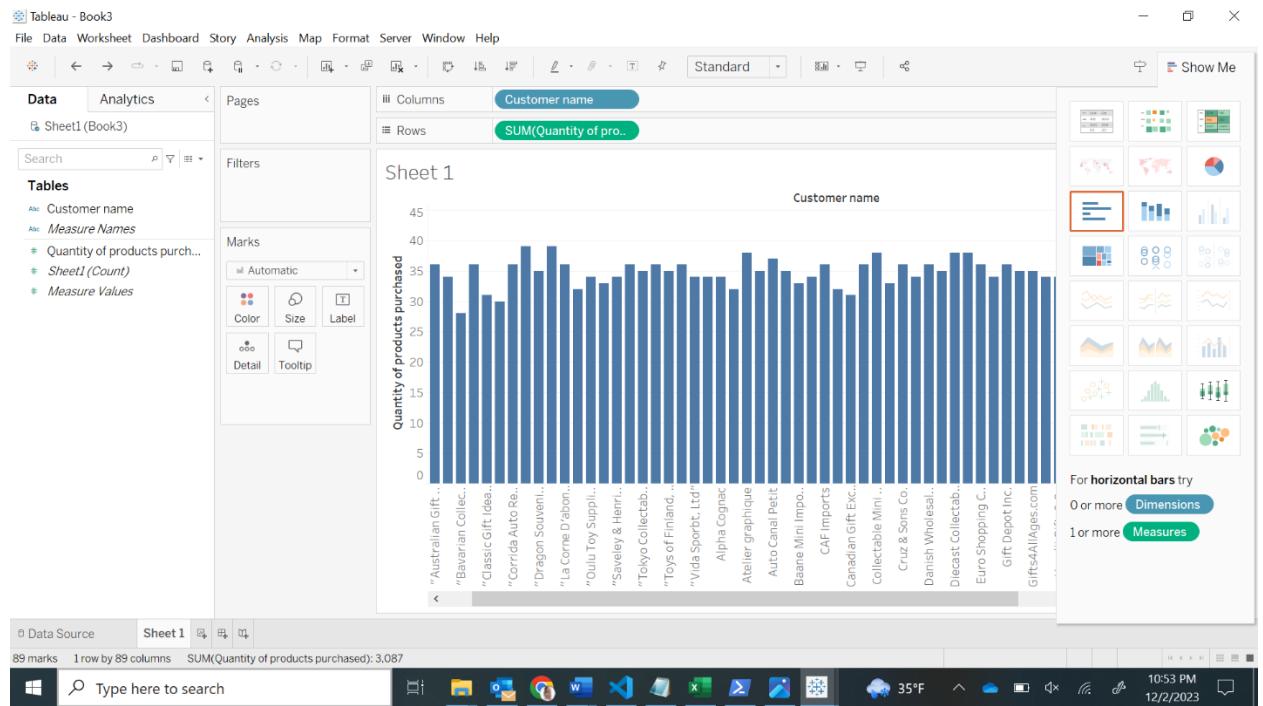
EXPLORER MAPREDUCE-MVN-MAIN

- data
  - > output
    - \* part-r-00000
    - 1 "AV Stores, Co." 1778
    - 2 "Anna's Decorations, Ltd" 1469
    - 3 "Australian Collectables, Ltd" 705
    - 4 "Australian Collectors, Co." 1926
    - 5 "Australian Gift Network, Co" 545
    - 6 "Bavarian Collectables Imports, Co." 401
    - 7 "Blauer See Auto, Co." 811
    - 8 "Classic Gift Ideas, Inc" 668
    - 9 "Clover Collections, Co." 490
    - 10 "Corrida Auto Replicas, Ltd" 1163
    - 11 "Double Decker Gift Stores, Ltd" 357
    - 12 "Dragon Souvenirs, Ltd." 1524
    - 13 "Iberia Gift Imports, Corp." 589
    - 14 "La Corne D'abondance, Co." 836
    - 15 "Norway Gifts By Mail, Co." 787
    - 16 "Oulu Toy Supplies, Inc." 1110
    - 17 "Royal Canadian Collectables, Ltd." 1051
    - 18 "Saviley & Henriot, Co." 1428
    - 19 "Stylish Desk Decors, Co." 937
    - 20 "Tokyo Collectables, Ltd" 1150
    - 21 "Tomi Speziallitten, Ltd" 936
    - 22 "Toys of Finland, Co." 1051
    - 23 "UK Collectables, Ltd." 1046
    - 24 "Vida Sport, Ltd" 1078
    - 25 "Volvo Model Replicas, Co" 647
    - 26 Alpha Cognac 687
    - 27 Amica Models & Co. 843
    - 28 Atelier graphique 270
    - 29 Auto Assoc. & Cie. 637
    - 30 Auto Canal Petit 1001
    - 31 Auto-Moto Classics Inc. 287
    - 32 Baane Mini Imports 1082
    - 33 Boards & Toys Co. 102
    - 34 CAF Imports 468
    - 35 Cambridge Collectables Co. 357
    - 36 Canadian Gift Exchange Network 703
    - classic Legends, Inc 720
  - > input
  - < output
    - \_SUCCESS
    - \_SUCCESS.crc
    - .part-r-00000.crc
    - .part-r-00000

OUTLINE TIMELINE MAVEN JAVA PROJECTS

part-r-00000 x J MyMapper.java x part-r-00000 x J MyReducer.java x viewCount.java x test\_data.txt x Book12345.txt x rawViews.txt x Book12345.txt

Ln 4, Col 1 Spaces: 4 UTF-8 LF Markdown ⚡ Quokka 9:22 PM 12/2/2023



**Story:** Discover the most valued consumers by recognizing individuals who have purchased the most products. Analyze consumer transaction data to identify top buyers, allowing for more personalized engagement and targeted marketing initiatives. By recognizing and prioritizing high-impact clients based on their purchase history, this effort attempts to improve customer connections, drive loyalty, and optimize business outcomes.

## Goal4: To Examine the trend in sales over different months and years.

```

File Edit Selection View Go ... < > mapreduce-mvn-main
EXPLORER J MyReducer.java 1 J MyMapper.java 1 part-r-00000 J ViewCount.java
src/main/java/com/mycompany/app/J MyReducer.java > MyReducer > reduce(Text, Iterable<IntWritable>, Context)
53 // @Override
54 // protected void reduce(Text key, Iterable<IntWritable> values, Context context)
55 // throws IOException, InterruptedException {
56 // try {
57 // int totalQuantity = 0;
58 //
59 // for (IntWritable value : values) {
60 // totalQuantity += value.get();
61 // }
62 //
63 // context.write(key, new IntWritable(totalQuantity));
64 // } catch (Exception e) {
65 // // Handle exceptions
66 // }
67 //
68 // }
69 //
70 // Goal 4
71 // @Override
72 protected void reduce(Text key, Iterable<IntWritable> values, Context context)
73 throws IOException, InterruptedException {
74 try {
75 int totalsales = 0;
76
77 for (IntWritable value : values) {
78 totalsales += value.get();
79
80 context.write(key, new IntWritable(totalsales));
81 } catch (Exception e) {
82 // Handle exceptions
83 }
84 }
85
86 // Goal 5
87 // @Override
88 // protected void reduce(Text key, Iterable<IntWritable> values, Context context)

```

Ln 78, Col 14 Spaces: 4 UTF-8 LF ⚡ Java ⚡ Quokka ⚡ 10:20 PM 12/2/2023

```

File Edit Selection View Go ... < > mapreduce-mvn-main
EXPLORER J MyReducer.java 1 J MyMapper.java 1 part-r-00000 J ViewCount.java
src/main/java/com/mycompany/app/J MyMapper.java > MyMapper
40 // try {
41 // String[] columns = value.toString().split("\t");
42 //
43 // if (columns.length >= 13) {
44 // String customerName = columns[11].trim();
45 // int quantityOrdered = Integer.parseInt(columns[1].trim());
46 // context.write(new Text(customerName), new IntWritable(quantityOrdered));
47 // }
48 // } catch (Exception e) {
49 // // Handle parsing errors or other exceptions
50 // }
51 //
52 // Goal 4
53 // @Override
54 protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
55 try {
56 String[] columns = value.toString().split("\\t");
57
58 if (columns.length >= 6) {
59 String orderDate = columns[5].trim();
60 double sales = Double.parseDouble(columns[4].trim());
61
62 String[] dateParts = orderDate.split("/");
63 if (dateParts.length == 3) {
64 String yearMonth = dateParts[2] + "-" + dateParts[1]; // Assuming the format is dd/MM/yyyy
65 context.write(new Text(yearMonth), new IntWritable((int) sales));
66 }
67 }
68 } catch (Exception e) {
69 // Handle parsing errors or other exceptions
70 }
71 }
72
73 // Goal 5
74 // @Override
75

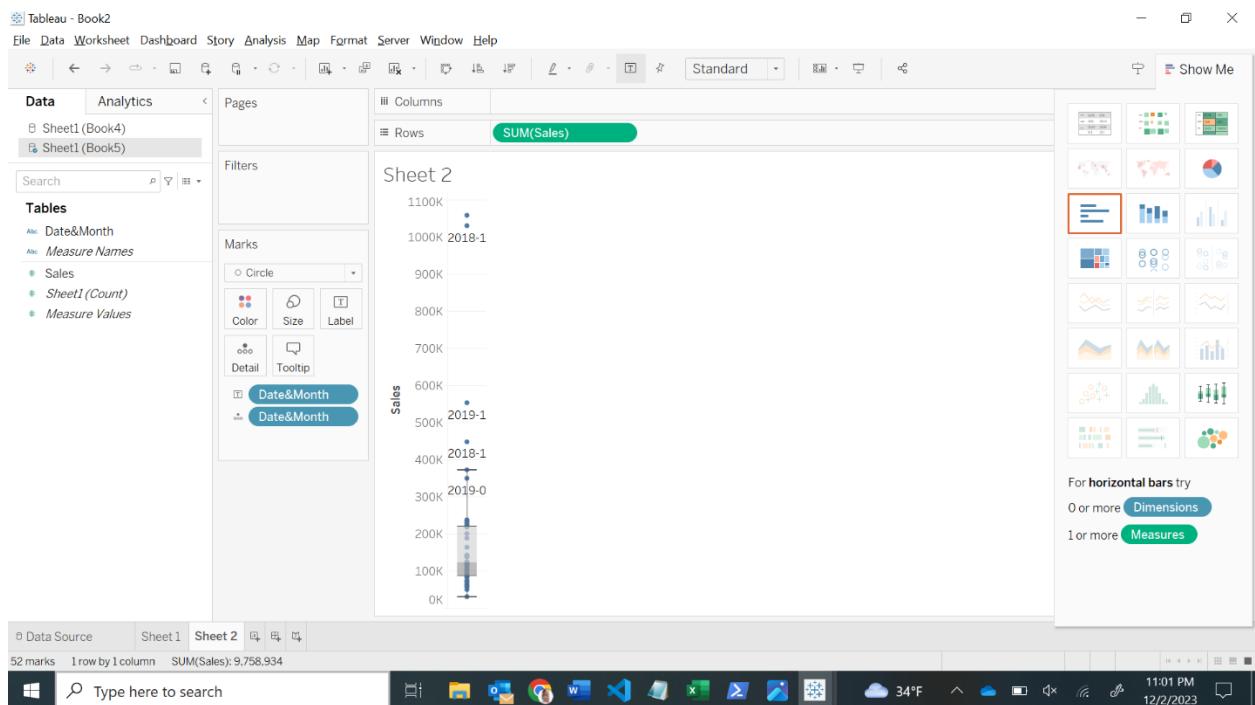
```

Ln 73, Col 1 Spaces: 4 UTF-8 LF ⚡ Java ⚡ Quokka ⚡ 10:20 PM 12/2/2023

```

mapreduce-mvn-main
File Edit Selection View Go ... < > mapreduce-mvn-main
EXPLORER J MyReducer.java part-r-00000 J MyMapper.java ViewCount.java test_data.txt Book12345.txt rawView.txt Book12345.txt
MAPREDUCE-MVN-MAIN
data > output > part-r-00000
1 2018-01 99388
2 2018-02 81953
3 2018-03 73156
4 2018-04 136881
5 2018-05 121201
6 2018-06 49787
7 2018-07 70421
8 2018-08 57404
9 2018-09 140872
10 2018-1 30427
11 2018-10 448398
12 2018-11 1829698
13 2018-12 236414
14 2018-2 58866
15 2018-3 82635
16 2018-4 64699
17 2018-5 71443
18 2018-6 118275
19 2018-7 117289
20 2018-8 140376
21 2018-9 123865
22 2019-01 187639
23 2019-02 226506
24 2019-03 88217
25 2019-04 93597
26 2019-05 55912
27 2019-06 226068
28 2019-07 235108
29 2019-08 349381
30 2019-09 102002
31 2019-1 105013
32 2019-10 552852
33 2019-11 1058562
34 2019-12 372747
35 2019-2 84872
36 2019-3 117492
37 2019-4 112517

```



**Story:** Examining the dataset to track and analyze sales trends over time. Investigate trends and changes in sales data to get insights into seasonal variances and long-term performance. This investigation seeks to comprehend the dynamics of sales over time in order to facilitate educated decision-making and strategic planning for future business ventures.

## Goal 5: To Analyze the average price of products sold in different countries.

The screenshot shows the Microsoft Visual Studio Code interface with the file `MyReducer.java` open. The code implements a reducer to calculate the average price of products sold in different countries. It uses regular expressions to extract the country name and price from the input values. The code includes exception handling for parsing errors.

```
src > main > java > com > mycompany > app > J MyMapper.java > ↗ MyMapper
  09 // String[] dateParts = value.toString().split("/");
  10 // if (dateParts.length == 3) {
  11 //     String yearMonth = dateParts[2] + "-" + dateParts[1]; // Assuming the format is dd/MM/yyyy
  12 //     context.write(new Text(yearMonth), new IntWritable((int) sales));
  13 // }
  14 // } catch (Exception e) {
  15 //     // Handle parsing errors or other exceptions
  16 // }
  17 // }

// Goal 5
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    try {
        String[] columns = value.toString().split("\t");
        if (columns.length >= 17) {
            String country = columns[16].trim();
            int priceEach = Integer.parseInt(columns[2].trim());
            context.write(new Text(country), new IntWritable(priceEach));
        }
    } catch (Exception e) {
        // Handle parsing errors or other exceptions
    }
}

// Goal 6
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    try {
        String[] columns = value.toString().split("\t");
        int totalSales = 0;
        for (IntWritable value : values) {
            totalSales += value.get();
        }
        context.write(key, new IntWritable(totalSales));
    } catch (Exception e) {
        // Handle exceptions
    }
}

// Goal 5
@Override
protected void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
    try {
        int totalPrices = 0;
        int count = 0;
        for (IntWritable value : values) {
            totalPrices += value.get();
            count++;
        }
        if (count > 0) {
            int averagePrice = totalPrices / count;
            context.write(key, new IntWritable(averagePrice));
        }
    } catch (Exception e) {
        // Handle exceptions
    }
}

// Goal 6
@Override
```

The screenshot shows the Microsoft Visual Studio Code interface with the file `MyReducer.java` open. The code implements both the `map` and `reduce` functions. The `map` function calculates the total sales for each country. The `reduce` function calculates the average price per country. Both functions include exception handling.

```
src > main > java > com > mycompany > app > J MyReducer.java > ↗ MyReducer > ↗ reduce(Text, Iterable<IntWritable>, Context)
  74 // int totalSales = 0;
  75 // for (IntWritable value : values) {
  76 //     totalSales += value.get();
  77 // }
  78 // context.write(key, new IntWritable(totalSales));
  79 // } catch (Exception e) {
  80 //     // Handle exceptions
  81 // }
  82 // }

// Goal 5
@Override
protected void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
    try {
        int totalPrices = 0;
        int count = 0;
        for (IntWritable value : values) {
            totalPrices += value.get();
            count++;
        }
        if (count > 0) {
            int averagePrice = totalPrices / count;
            context.write(key, new IntWritable(averagePrice));
        }
    } catch (Exception e) {
        // Handle exceptions
    }
}

// Goal 6
@Override
```

Screenshot of an IDE (IntelliJ IDEA) showing the project structure and output of a MapReduce job.

**Project Explorer:**

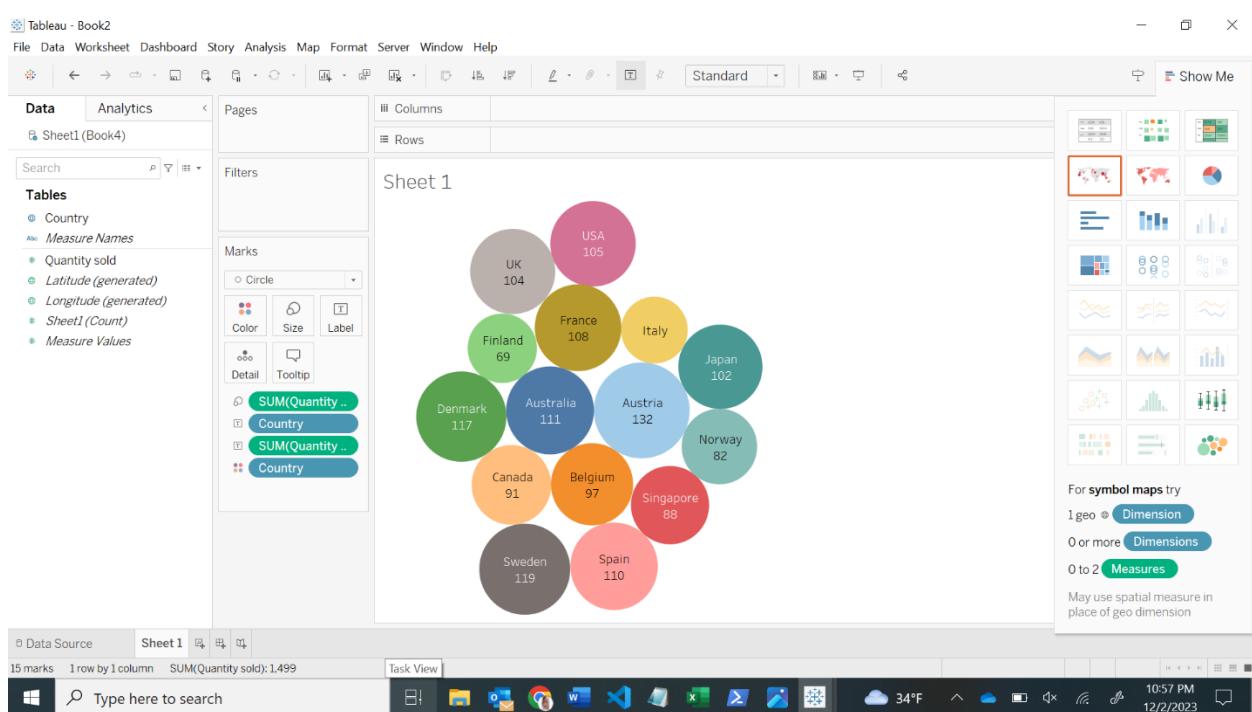
- src
  - main.java
    - J MyMapper.java
    - J MyReducer.java
    - J ViewCount.java
- test
- target
- .gitignore
- pom.xml
- README.md

**Output:**

```

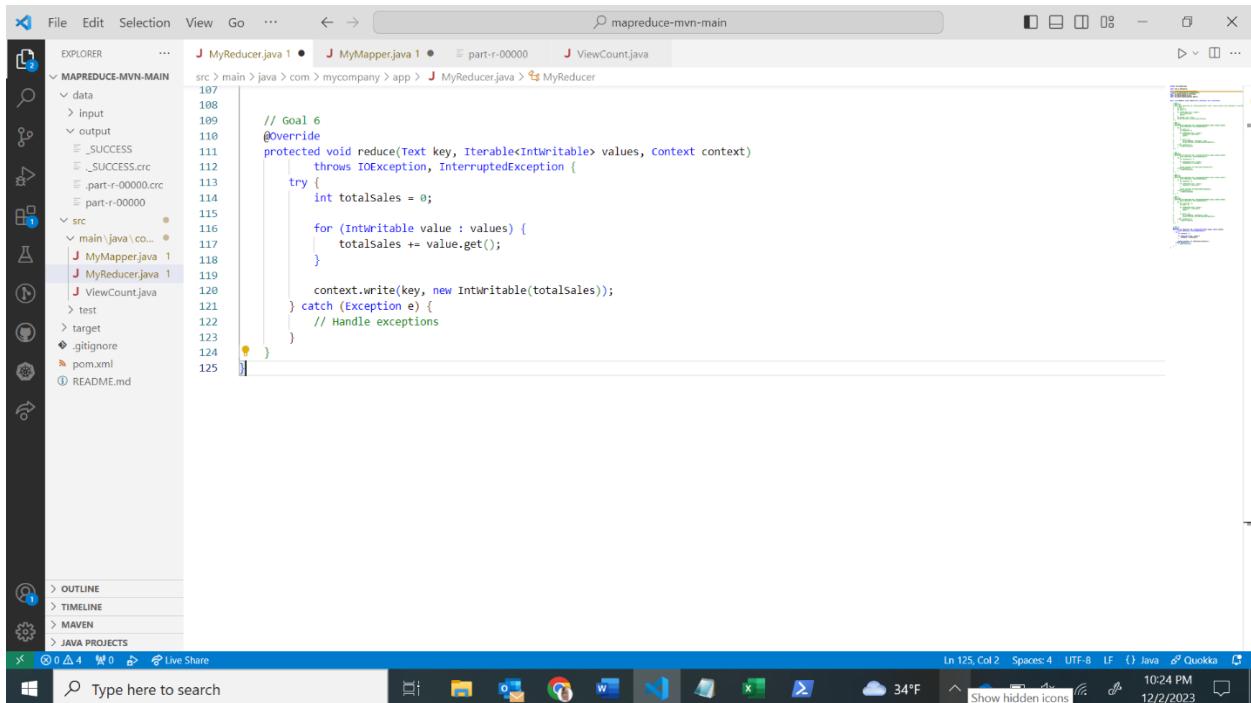
data > output > part-r-00000
1 Australia 111
2 Austria 132
3 Belgium 97
4 Canada 91
5 Denmark 117
6 Finland 69
7 France 108
8 Italy 64
9 Japan 102
10 Norway 82
11 Singapore 88
12 Spain 110
13 Sweden 119
14 UK 104
15 USA 105
16

```



**Story:** Investigate and evaluate the average pricing of things sold, noting differences between countries. The purpose is to get insights into pricing dynamics, geographical preferences, and market trends by evaluating this data. This study aids in informed decision-making by enabling for smart pricing adjustments and targeted marketing tactics adapted to the economic landscape of each country.

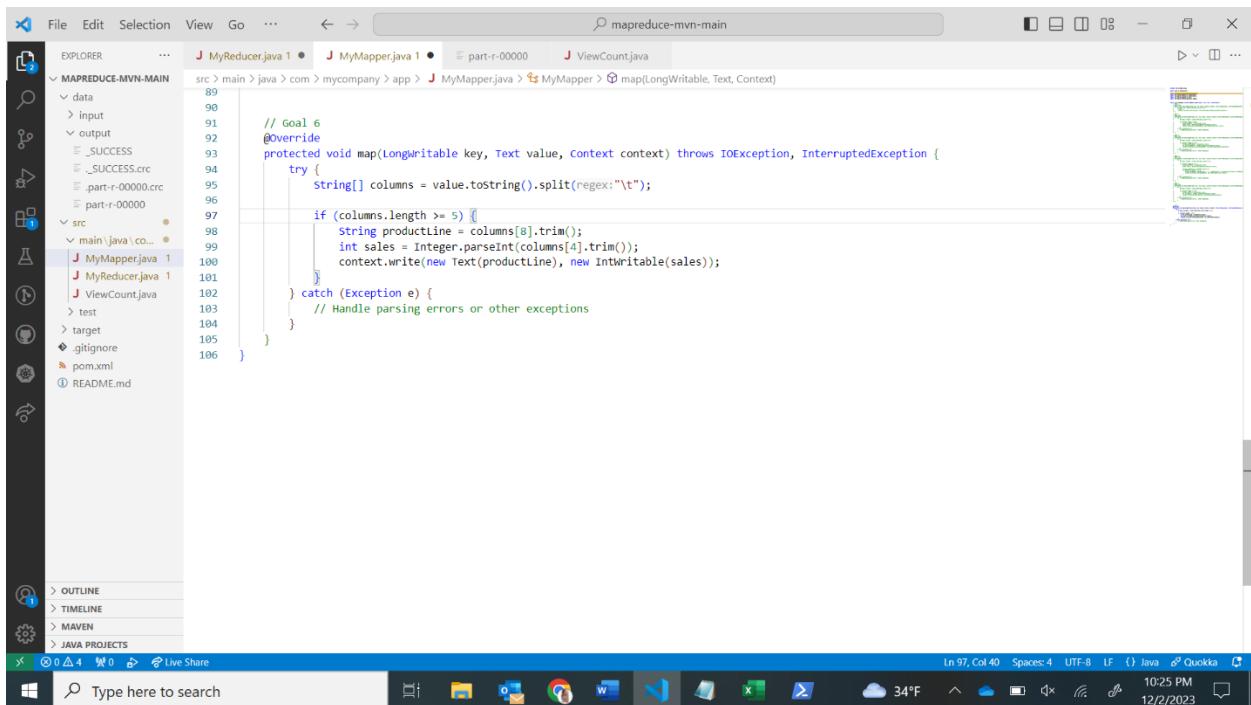
## Goal 6: To Understand the contribution of each product line to the total sales.



```
// Goal 6
@Override
protected void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
    try {
        int totalsales = 0;

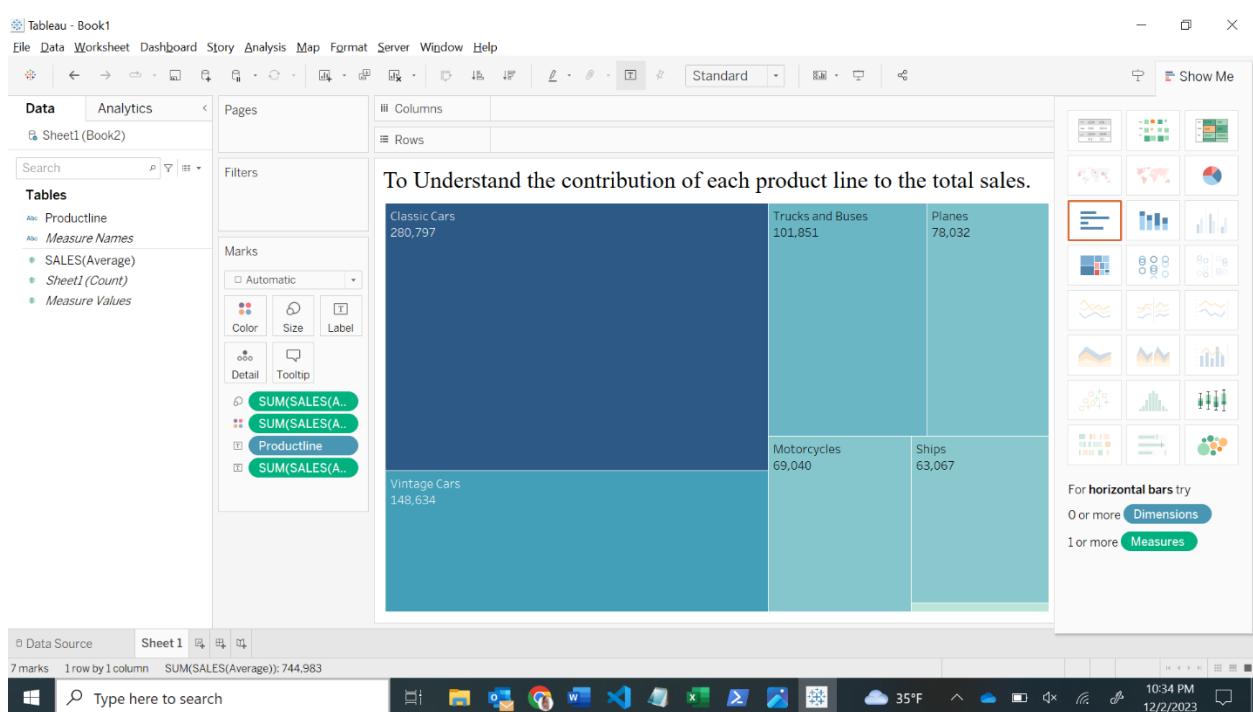
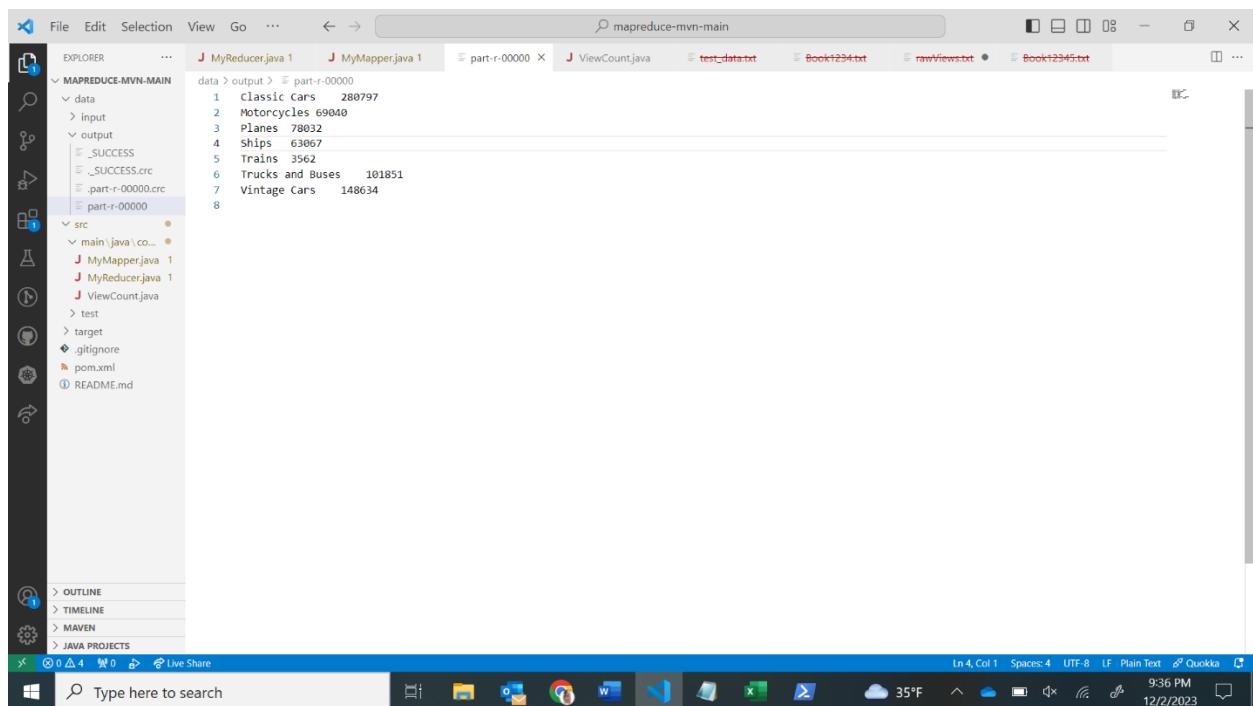
        for (IntWritable value : values) {
            totalsales += value.get();
        }

        context.write(key, new IntWritable(totalsales));
    } catch (Exception e) {
        // Handle exceptions
    }
}
```



```
// Goal 6
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    try {
        String[] columns = value.toString().split(regex: "\t");

        if (columns.length >= 5) {
            String productline = columns[8].trim();
            int sales = Integer.parseInt(columns[4].trim());
            context.write(new Text(productline), new IntWritable(sales));
        }
    } catch (Exception e) {
        // Handle parsing errors or other exceptions
    }
}
```



**Story:** Investigate the statistics to learn how each product line contributes to overall sales. Analyze sales data to determine the relative impact of different product lines on total revenue. This investigation intends to provide insights into the performance of various product categories, allowing strategic decisions for optimizing sales tactics and resource allocation. Ultimately, the

goal is to comprehend and improve each product line's contribution to the overall performance of the company.

## **Conclusion:**

The analysis of several goals generated practical information. Understanding customer behavior, sales dispersion, and top customers improves targeted strategies. Examining sales trends over time and examining pricing dynamics by area provide significant market knowledge. Finally, understanding each product line's contribution improves inventory and marketing efforts, enabling educated decision-making for long-term business success.

**GitHub:** <https://github.com/Achyuthareddy-gantla/ByteBrilliance>

## **Citations:**

1. <https://www.factorywarrantylist.com/car-sales-by-country.html>
2. <https://www.statista.com/topics/1487/automotive-industry/#topicOverview>
3. <https://www.oica.net/category/sales-statistics/>