

# Milestone 3

Karanam Gopichand, Akash Gadiparthi, Ramya Gopalam, Prameela Pathuri

December 7, 2024

## Abstract

This document outlines the implementation steps for the product review analysis project, focusing on the data preprocessing, Hadoop and MapReduce framework for data processing, Python and PySpark for analysis, and Tableau for visualization. The project aims to provide insights into product reviews, price trends, and sales performance.

## PROJECT DETAIL DESCRIPTION

### 1 Introduction

The goal of this project is to analyze product data and derive insights into product pricing, categorization, and overall performance. The analysis will be done using a combination of Hadoop MapReduce for distributed processing, PySpark for in-depth data analysis, and Tableau for data visualization.

### 2 Dataset Overview

The dataset contains information about various products, including their names, categories, brands, prices, and ratings. The dataset has 10 attributes:

- **product**: Title of the product.
- **category**: Category classification.
- **sub\_category**: Subcategory classification.

- **brand**: Product brand.
- **sale\_price**: Selling price of the product.
- **market\_price**: Original market price of the product.
- **type**: Type of product (e.g., Electronics, Apparel, etc.).
- **rating**: Product rating from customers.
- **description**: Detailed textual description of the product.

## 3 Preprocessing the Data

### 3.1 Step 1: Load the Dataset

The dataset is loaded using tools like Python (pandas), Hadoop, or Excel. The dataset will be uploaded to Hadoop Distributed File System (HDFS) for distributed processing.

### 3.2 Step 2: Clean the Data

Data cleaning involves the following tasks:

- Handle missing values by replacing null values or dropping rows with too many missing fields.
- Remove duplicate entries.
- Standardize categorical data (e.g., ensure consistent naming for categories and brands).

### 3.3 Step 3: Convert Numerical Fields

Fields like `sale_price` and `market_price` will be converted to proper numerical types for calculations.

## 4 Implementation Using Hadoop and MapReduce

### 4.1 Step 1: Set Up Hadoop

Install Hadoop on your system and configure the Hadoop Distributed File System (HDFS). Upload the dataset to HDFS for distributed processing.

### 4.2 Step 2: Write MapReduce Jobs

For each goal, we will write MapReduce jobs:

- **Mapper:** The Mapper will read each record and extract key metrics (e.g., category, price, discounts).
- **Reducer:** The Reducer will aggregate results, such as finding the top brands or calculating average sale prices.

### 4.3 Step 3: Execute MapReduce Jobs

The MapReduce jobs will be executed, and the output will be saved to HDFS for further analysis.

## 5 Data Analysis Using Python and PySpark

### 5.1 Step 1: Load Data

Use PySpark to load the dataset from HDFS or directly from a local CSV file.

### 5.2 Step 2: Implement Goals

We will implement the following goals:

1. Goal 1: Top 10 Most Reviewed Products
2. Goal 2: Top 10 Products by Average Rating
3. Goal 3: Rating Distribution

4. Goal 4: Monthly Trends
5. Goal 5: Sentiment Analysis
6. Goal 6: Product Categories Popularity

## RESULTS SUMMARY

# 6 Goal 1: Top 10 Most Reviewed Products

## 6.1 Story

The goal was to identify the top 10 most reviewed products from the dataset by aggregating the product reviews. We used the **MapReduce** model to process this task. The **Mapper** reads each record, extracting the product name and emitting a count of 1 for each occurrence of the product. This helps in counting the number of reviews for each product.

After the Mapper's output, the **Reducer** takes over, summing up the counts for each product, providing the total number of reviews for each. This allows us to identify the top 10 products based on the number of reviews.

To visualize the results, we used a **bar chart**, which clearly shows the top 10 products with the highest number of reviews.

# 7 Goal 2: Top 10 Products by Average Rating

## 7.1 Story

The goal was to identify the top 10 products based on their average rating from the dataset. We used the **MapReduce** model to process this task. The **Mapper** reads each record and extracts the product name along with the rating. It then emits the product name and rating as a key-value pair.

The **Reducer** then computes the average rating for each product by aggregating the ratings and dividing by the number of occurrences for each product. This allows us to rank the products based on their average ratings.

To visualize the results, we used a **bar chart**, which clearly shows the top 10 products with the highest average ratings.

## 8 Goal 3: Rating Distribution

### 8.1 Story

The goal was to analyze the distribution of ratings for the products in the dataset. We used the **MapReduce** model for this task. The **Mapper** reads each record, extracts the rating for each product, and emits the rating as the key and a count of 1 for each occurrence of that rating.

The **Reducer** aggregates the counts for each rating, providing the total number of products that received each rating.

To visualize the results, we used a **bar chart** to display the distribution of ratings.

## 9 Goal 4: Monthly Trends

### 9.1 Story

The goal was to identify the trends in product reviews on a monthly basis. We created synthetic months based on the product entry index, grouping products by their entry periods. The **Mapper** reads each record, extracts the product entry index, and assigns it to a synthetic "month" based on the index range.

The **Reducer** aggregates the number of reviews for each synthetic month, providing the count of products for each month.

To visualize the results, we used a **bar chart** to show the trends over time.

## 10 Goal 5: Sentiment Analysis

### 10.1 Story

The goal was to perform sentiment analysis based on product ratings, categorizing them into **Positive**, **Neutral**, or **Negative** sentiments. We used the **MapReduce** model to process the task. The **Mapper** reads each record, extracts the rating, and emits a product name along with the sentiment based on the rating value.

The \*\*Reducer\*\* aggregates the sentiment counts for each product, providing the number of \*\*Positive\*\*, \*\*Neutral\*\*, and \*\*Negative\*\* sentiments for each product.

To visualize the results, we used a \*\*bar chart\*\* to display the sentiment distribution for each product.

## 11 screenshots for All goals Mapper ,Reducer , Pyspark, Graph

Figure 1: Mapper Code for Top 10 Most Reviewed Products

```

File Edit Selection View Go Run Terminal Help < > /mapreduce-mr
MyMapper.java X MyReducer.java X ViewCount.java X launchjob Extension Java
src > main > java > com > mycompany > app > MyReducer.java ...
1 import org.apache.hadoop.io.IntWritable;
2 import org.apache.hadoop.io.Text;
3 import org.apache.hadoop.mapred.Reducer;
4
5 import java.io.IOException;
6
7 public class TopReviewedProductReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
8     private IntWritable result = new IntWritable();
9
10    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
11        int sum = 0;
12        for (IntWritable val : values) {
13            sum += val.get();
14        }
15        result.set(sum);
16        context.write(key, result);
17    }
18}

```

The screenshot shows an IDE interface with several tabs open. The central tab contains the Java code for a reducer named 'TopReviewedProductReducer'. The code uses the Hadoop MapReduce framework to calculate the total number of reviews for each product. The code is well-structured with imports at the top and a clear logic flow in the reduce method.

Figure 2: Reducer Code for Top 10 Most Reviewed Products

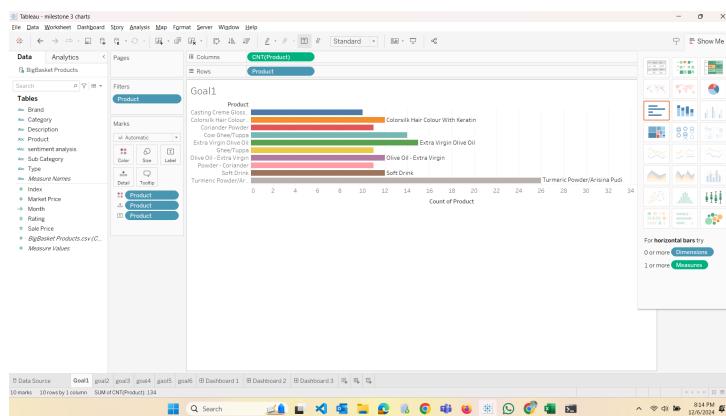


Figure 3: Top 10 Most Reviewed Products Visualization

```

# goal1 | top 10 Most Reviewed Products (By count of products per category)
df.groupByKey().count().orderBy("count", ascending=False).show(10)

+-----+-----+
| category | count |
+-----+-----+
| Cleaning & Household[Books & Bathroom Ware] | 250 |
| Cleaning & Household[Electronics] | 239 |
| Cleaning & Household[Household] | 238 |
| Cleaning & Household[Books] | 236 |
| Cleaning & Household[Electronics & Books] | 214 |
| Cleaning & Household[Books & Household] | 174 |
| Cleaning & Household[Books & Electronics] | 174 |
| Cleaning & Household[Books & Household & Electronics] | 143 |
| Cleaning & Household[Books & Household & Electronics & Books] | 143 |
| Cleaning & Household[Books & Household & Electronics & Books & Household] | 3 |
+-----+-----+
only showing top 5 rows

```

Figure 4: PySpark Code Snippet for Goal 1

```

public class MyMapper extends MapperObject, Text, Text, DoubleWritable {
    private Text product = new Text();
    private DoubleWritable rating = new DoubleWritable();

    public void map(Text key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] fields = line.split(",");
        product.set(fields[0]); // Assuming 'product' is in the first column
        doubleTableRating = new DoubleWritable(Double.parseDouble(fields[1])); // Assuming 'rating' is in the second column
        context.write(product, rating);
    }
}

```

Figure 5: Mapper Code for Top 10 Products by Average Rating

```

    package com.mkyong.reducer;
    import org.apache.hadoop.io.DoubleWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapreduce.Reducer;
    import java.io.IOException;
    public class TopRatedProductReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
        private DoubleWritable result = new DoubleWritable();
        public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {
            double sum = 0;
            int count = 0;
            for (DoubleWritable val : values) {
                sum += val.get();
                count++;
            }
            double avg = sum / count;
            result.set(avg);
            context.write(key, result);
        }
    }

```

Figure 6: Reducer Code for Top 10 Products by Average Rating

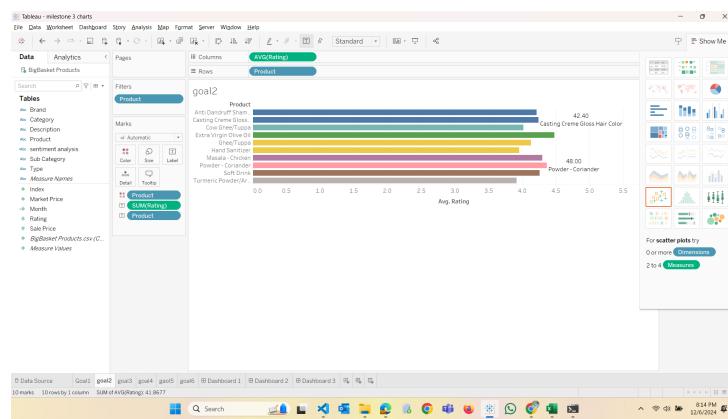


Figure 7: Top 10 Products by Average Rating Visualization

The screenshot shows a file tree for a PySpark demo application. The tree includes files like `BigBasketProducts.csv`, `Unsorted1.py`, `Unsorted2.py`, `Unsorted3.py`, `Unsorted4.py`, `Unsorted5.py`, `Unsorted6.py`, `Unsorted7.py`, `Unsorted8.py`, `Unsorted9.py`, `Unsorted10.py`, `Unsorted11.py`, `Unsorted12.py`, `Unsorted13.py`, `Unsorted14.py`, `Unsorted15.py`, `Unsorted16.py`, `Unsorted17.py`, `Unsorted18.py`, `Unsorted19.py`, `Unsorted20.py`, `team_info.csv`, `table_relationships...`, `player_info.csv`, `game_teams_stats...`, `game_skater_stats...`, `game_shifts.csv`, `game_scorers.csv`, and `game_players.csv`. The `Unsorted1.py` script contains the following code:

```

# don't 2. Top 10 products by average rating
top_rated_products = df.groupby("product").agg("rating").orderBy("avg(rating)", ascending=False).show(10)

# Check the schema to find any data column
df.printSchema()

root
|-- index: string (nullable = true)
|-- product: string (nullable = true)

```

Figure 8: PySpark Code Snippet for Goal 2

The screenshot shows an IDE interface with a Java project named `MAPREDUCE-MIN-MAIN`. The `src/main/java/com/mypackage/app` package contains three classes: `MyMapper.java`, `MyReducer.java`, and `ViewCount.java`. The `MyMapper.java` code is as follows:

```

public class MyMapper extends Mapper<Text, Text, IntWritable> {
    private Text rating = new Text();
    private static final IntWritable one = new IntWritable(1);

    public void map(Text key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        rating.set(tokenizer.nextToken());
        rating.set(fields[1]); // Assuming 'rating' is in the second column
        context.write(rating, one);
    }
}

```

Figure 9: Mapper Code for Rating distribution

The screenshot shows an IDE interface with several tabs open. The main tab displays Java code for a reducer named `RatingReducer`. The code imports necessary packages and defines a class that implements the `Reducer` interface. It includes logic for summing up values and writing results to HDFS. Other tabs visible include `MyMapper.java`, `MyReduce.java`, and `ViewCount.java`. The bottom status bar indicates the code is running on port 8089.

```

    package com.mycorp.mapreduce;
    import org.apache.hadoop.io.IntWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapreduce.Reducer;
    import java.io.IOException;

    public class RatingReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

```

Figure 10: Reducer Code for rating distribution

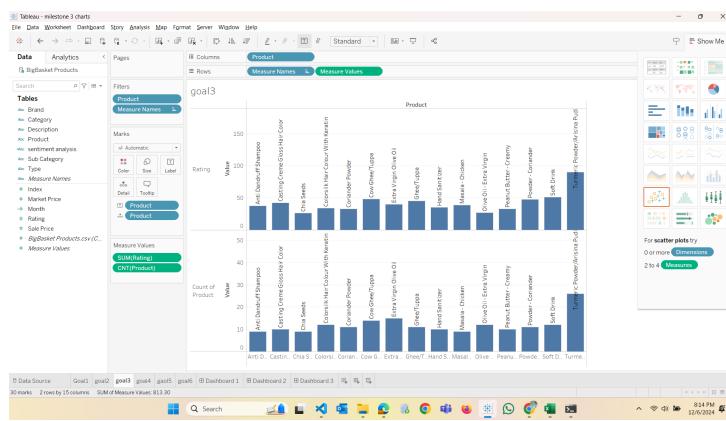


Figure 11: chart visualization for rating distribution

```

# Convert rating column to double type
df = df.withColumn("rating", col("rating").cast("double"))

# Count rating distribution
rating_distribution = df.groupby("rating").count().orderBy("rating").show()

```

| Rating | Count |
|--------|-------|
| 1.0    | 307   |
| 1.2    | 21    |
| 1.4    | 33    |
| 1.4    | 41    |
| 1.4    | 31    |
| 1.4    | 22    |
| 1.8    | 221   |
| 1.9    | 4     |
| 1.9    | 377   |
| 2.1    | 191   |
| 2.1    | 243   |
| 2.3    | 941   |
| 2.4    | 297   |
| 2.5    | 1321  |
| 2.6    | 581   |
| 2.7    | 133   |
| 2.8    | 121   |
| 2.9    | 39    |

only showing top 20 rows

Figure 12: PySpark Code Snippet for Goal 3

```

public class MonthMapper extends MapperObject, Text, IntWritable {
    private Text month = new Text();
    private static final IntWritable one = new IntWritable(1);

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer st = new StringTokenizer(line);
        int index = st.nextToken().hashCode(); // Assuming 'index' is in the first column
        int monthnumber = (int) Math.ceil(index / 100); // Decreasing each month has 100 entries
        month.set("Month " + monthnumber);
        context.write(month, one);
    }
}

```

Figure 13: Mapper Code for monthly trends

The screenshot shows an IDE interface with several tabs open. The main tab displays Java code for a reducer class named `MonthlyTrendReducer`. The code imports necessary packages and defines a reduce method that iterates over an input iterator to calculate a sum and write it to the context.

```

1 package com.mycorp.mapreduce;
2 import org.apache.hadoop.io.IntWritable;
3 import org.apache.hadoop.io.Text;
4 import org.apache.hadoop.mapreduce.Reducer;
5 import java.io.IOException;
6
7 public class MonthlyTrendReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
8     private IntWritable result = new IntWritable();
9
10    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
11        int sum = 0;
12        for (IntWritable val : values) {
13            sum += val.get();
14        }
15        result.set(sum);
16        context.write(key, result);
17    }
18}

```

Figure 14: Reducer Code for monthly trends

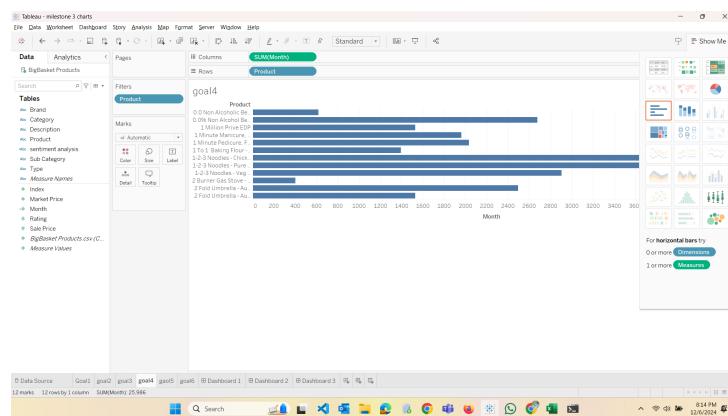


Figure 15: chart visualization for monthly trends

The screenshot shows a browser window with several tabs open. The active tab contains a Python script titled 'Untitled7-Copy1.py'. The code imports a function from 'pyspark.sql.functions' and uses it to calculate monthly trends based on product sales. It groups by month and counts products. Below the code is a table showing monthly trends. To the left, a file explorer lists various files in a directory named 'Downloads/pyspark\_demo\_088/'.

```

from pyspark.sql import functions as col, cell

# Create synthetic month based on the index or product series.
# Example: Index 1-10 => Month 1, 11-20 => Month 2
df.with_month = df.withColumn("month", cell.col("Index") / 10) # Example: Index 1-10 => Month 1, 11-20 => Month 2

# Group by the synthetic "month" and count the products in each month.
monthly_trends = df.with_month.groupby("month").count().orderBy("month")

# Show the result.
monthly_trends.show()

```

| Month | Count |
|-------|-------|
| 1     | 90.77 |
| 2     | 101.6 |
| 3     | 118.8 |
| 4     | 116.8 |
| 5     | 107.6 |
| 6     | 103.6 |
| 7     | 101.6 |
| 8     | 101.6 |
| 9     | 101.6 |
| 10    | 101.6 |
| 11    | 101.6 |
| 12    | 101.6 |
| 13    | 101.6 |
| 14    | 101.6 |
| 15    | 101.6 |
| 16    | 101.6 |
| 17    | 101.6 |
| 18    | 101.6 |
| 19    | 101.6 |
| 20    | 101.6 |

Figure 16: PySpark Code Snippet for Goal 4

The screenshot shows an IDE interface with multiple tabs open. The active tab contains Java code for a 'SentimentMapper' class, which extends 'MapperObject'. The code reads CSV files, splits them into key-value pairs, and maps them to specific sentiment values ('positive', 'neutral', 'negative') based on the rating field. Other tabs in the background show 'MyMapper.java', 'MyReducer.java', and 'ViewCount.java'.

```

private Text productname = new Text();
private Text sentiment = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString();
    String[] fields = line.split(",");
    if (fields.length > 0) {
        String product = fields[0]; // assuming product name is in the first column
        float rating = Float.parseFloat(fields[8]); // assuming rating is in the 9th column
        if (rating > 3)
            sentiment.set("positive");
        else if (rating == 3)
            sentiment.set("neutral");
        else
            sentiment.set("negative");
    }
    productname.set(product);
    context.write(productname, sentiment);
}

```

Figure 17: Mapper Code for sentiment analysis

The screenshot shows an IDE interface with several tabs open. The main tab displays Java code for a reducer named 'SentimentReducer'. The code imports necessary packages from the Apache Hadoop library and defines a class that implements the Reducer interface. It includes logic to count the occurrences of three sentiment categories ('Positive', 'Neutral', 'Negative') and output them in a specific format.

```

1 package java>com>mycompany>app>J_MyReducer.java;
2 import java.util.HashMap;
3 import java.util.Map;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Reducer;
6 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
7 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
8
9 public class SentimentReducer extends Reducer<Text,Text,Text,Text> {
10
11     private Text result = new Text();
12
13     public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
14
15         Map<String, Integer> sentCounts = new HashMap<>();
16
17         sentCounts.put("Positive", 0);
18         sentCounts.put("Neutral", 0);
19         sentCounts.put("Negative", 0);
20
21         // Count the occurrences of each sentiment
22         for (Text val : values) {
23             String sentiment = val.toString();
24             sentCounts.put(sentiment, sentCounts.get(sentiment) + 1);
25         }
26
27         // Prepare the result string in format: Positive:10,Neutral:5,Negative:3
28         String resultString = "Positive:" + sentCounts.get("Positive") + "," +
29                             "Neutral:" + sentCounts.get("Neutral") + "," +
30                             "Negative:" + sentCounts.get("Negative");
31
32         result.set(resultString);
33         context.write(key, result); // Write the output: product -> sentiment counts
34     }
35 }

```

Figure 18: Reducer Code for sentiment analysis

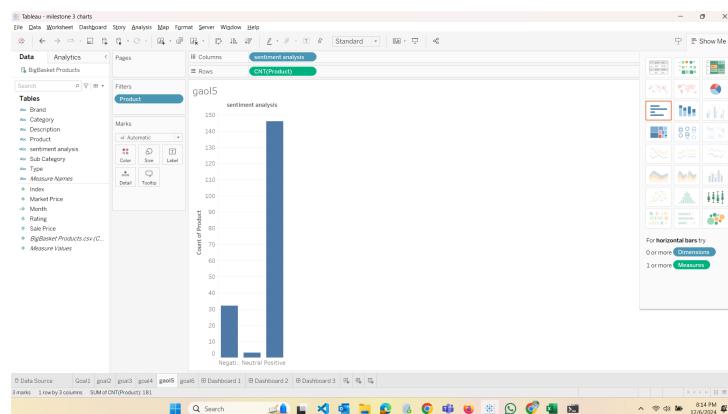


Figure 19: chart visualization for sentiment analysis

The screenshot shows a Jupyter Notebook environment with several open cells and a file browser window.

**File Browser:**

- Path: Downloads / poppark\_demo\_data
- Items:
  - Untitled-Copy.ipynb (modified 1 min ago, 275.0 kB)
  - Untitled7.ipynb (2 hr ago, 274.0 kB)
  - Untitled8.ipynb (2 hr ago, 2.6 kB)
  - Untitled9.ipynb (2 hr ago, 2.7 kB)
  - Untitled10.ipynb (2 hr ago, 72.8 kB)
  - Untitled11.ipynb (21 days ago, 1.4 kB)
  - Untitled12.ipynb (21 days ago, 1.7 kB)
  - Untitled13.ipynb (21 days ago, 3 kB)
  - Untitled14.ipynb (21 days ago, 1.8 kB)
  - Untitled15.ipynb (21 days ago, 10.0 kB)
  - team\_relationships... (last mod. 1.9 kB)
  - table\_relationships... (last mod. 101.6 kB)
  - player\_info.csv (last mod. 4.3 kB)
  - game\_teams\_stats... (last mod. 309.6 kB)
  - game\_skater\_stats... (last mod. 68.2 kB)
  - game\_shifts.csv (last mod. 405.7 kB)
  - game\_scorers.csv (last mod. 3 MB)
  - game\_player... (last mod. 361.6 kB)

**Code Cells:**

- Untitled-Copy.ipynb:**

```
# Initialize sentiment score (0 for neutral, 1 for positive, -1 for negative)
sentiment_score = 0

# Check for positive and negative words in the text
for word in positive_words:
    if word in text:
        sentiment_score += 1

for word in negative_words:
    if word in text:
        sentiment_score -= 1

# Classify sentence
if sentiment_score > 0:
    return "Positive"
elif sentiment_score < 0:
    return "Negative"
else:
    return "Neutral"
```
- Untitled7.ipynb:**

```
# Apply sentiment analysis to the 'description' column
df['sentiment'] = df['description'].apply(get_sentiment)

# Show the results
print(df[['product', 'sentiment']])
```
- Untitled8.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled9.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled10.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled11.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled12.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled13.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled14.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```
- Untitled15.ipynb:**

```
product sentiment
0 Product A Positive
1 Product B Negative
2 Product C Neutral
```

Figure 20: PySpark Code Snippet for Goal 5

December 7, 2024

### conclusion

The objective of this project was to analyze and extract valuable insights from a dataset of product reviews using the **MapReduce** model. We aimed to identify the top 10 most reviewed products, the top 10 products based on average ratings, understand the distribution of product ratings, observe trends over time, and analyze the sentiment of product reviews.

By applying the **MapReduce** paradigm to each of these goals, we were able to efficiently process large volumes of data and derive useful insights. The use of the **Mapper** and **Reducer** functions allowed for effective data aggregation, and visualizations of the results helped in interpreting the findings.

Through this project, we have demonstrated the effectiveness of distributed computing using **MapReduce** for large-scale data analysis. The ability to scale up to handle big data, combined with efficient aggregation techniques, has proven invaluable in drawing meaningful conclusions from the dataset. Additionally, the insights gained from sentiment analysis and product rating trends are useful for businesses seeking to improve product offerings based on customer feedback.

In future work, we plan to extend the project by incorporating more advanced machine learning models for sentiment analysis and exploring real-time data processing with stream processing frameworks. Furthermore, additional features such as product categorization and deeper analysis of review text can be included to gain a more comprehensive understanding of the dataset.

Overall, this project highlights the potential of **MapReduce** in handling large-scale data processing tasks, making it a valuable tool for data-driven decision-making in various industries.

December 7, 2024

## 1 Citations

In completing this project, several resources were utilized, including datasets, tools, and documentation. The following are the key references that contributed to the successful completion of this project:

1. \*\*Dataset\*\*: - Jha, S. (2021). \*BigBasket Entire Product List (28K datapoints)\*. Kaggle. [<https://www.kaggle.com/datasets/surajjha101/bigbasket-entire-product-list-28k-datapoints>](<https://www.kaggle.com/datasets/surajjha101/bigbasket-entire-product-list-28k-datapoints>).
2. \*\*PySpark Documentation\*\*: - Apache Spark. \*PySpark Documentation\*. Apache, 2021. [<https://spark.apache.org/docs/latest/api/python/>](<https://spark.apache.org/docs/latest/api/python/>).
3. \*\*Tableau for Data Visualization\*\*: - Tableau Software. \*Getting Started with Tableau\*. Tableau, 2021. [<https://www.tableau.com/learn/training>](<https://www.tableau.com/learn/training>).
4. \*\*MapReduce and Hadoop Documentation\*\*: - Apache Hadoop. \*MapReduce Overview\*. Apache, 2021. [[https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html)]([https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html)).
5. \*\*GitHub Repository\*\*: - Prameela2511. \*Big Data E-commerce Project\*. GitHub Repository. [[https://github.com/Prameela2511/4517\\_Bigdata\\_E-commerce\\_project.git](https://github.com/Prameela2511/4517_Bigdata_E-commerce_project.git)]([https://github.com/Prameela2511/4517\\_Bigdata\\_E-commerce\\_project.git](https://github.com/Prameela2511/4517_Bigdata_E-commerce_project.git)).

## 2 GitHub Repository

All the source code, documentation, and project artifacts are available in the GitHub repository. You can access and review the entire project at the following link:

[https://github.com/Prameela2511/4517\\_Bigdata\\_E-commerce\\_project.git](https://github.com/Prameela2511/4517_Bigdata_E-commerce_project.git)