Retail e-Commerce Analysis & Recommender System Documentation

Written by:

Oukessou Houda

Maliki Meryem

Berrissoul Saad

Errimy Hatim

Deloitte.

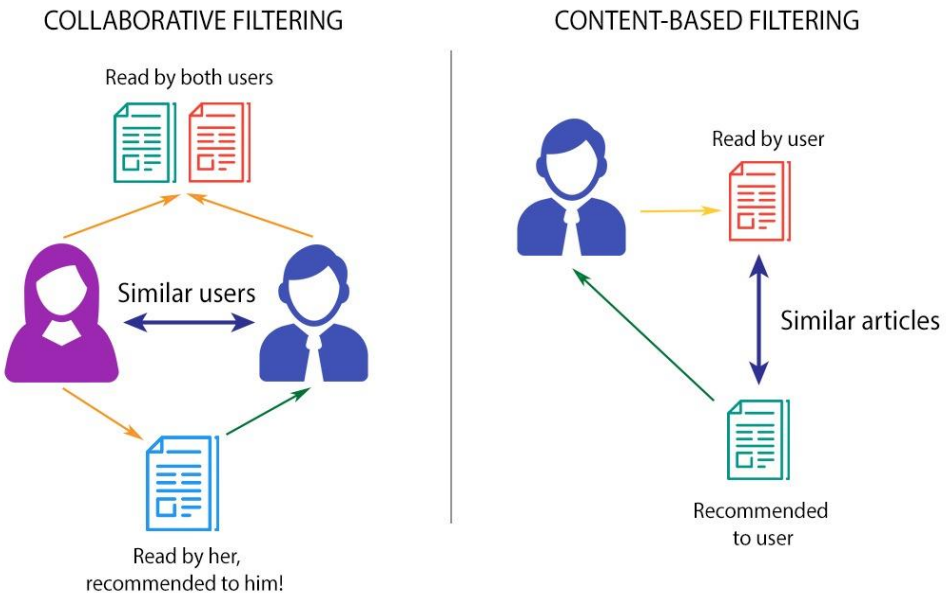
Recommendation System:

Introduction:

An eCommerce company wants to make better use of their data with the goal of better decision-making and optimized item recommendations for visitors of the website to increase sales.

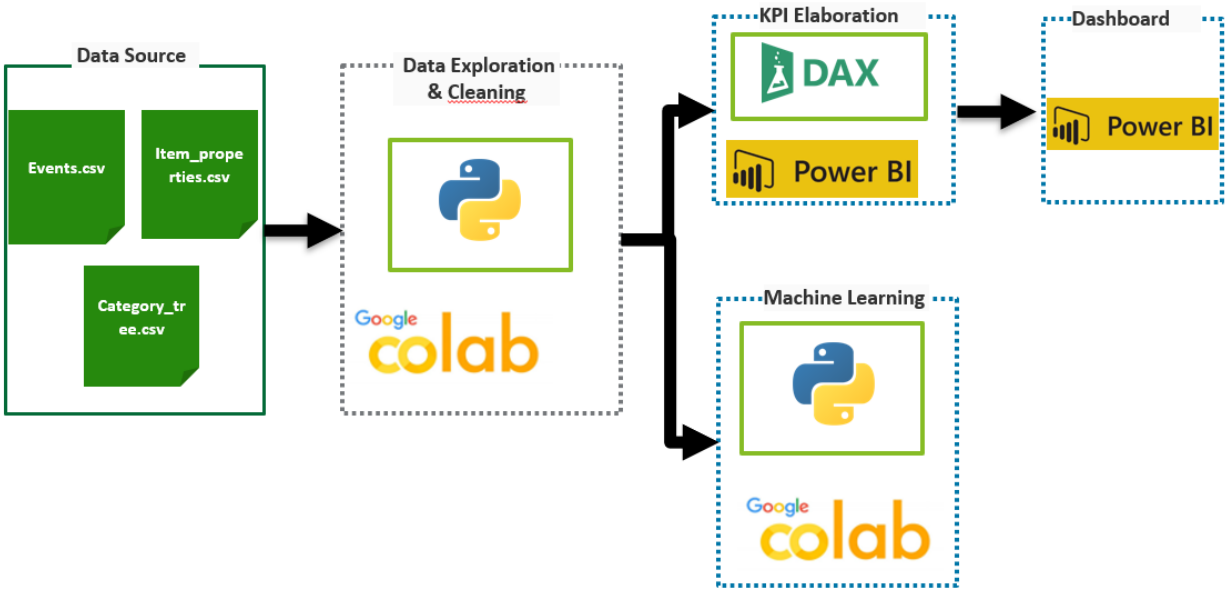
Solution:

In order to make better recommendations for the visitors, we were tasked with making an item recommendation system based on the data the client provided.
There are three approaches when making a recommendation system:
Collaborative filtering: This type of system uses the past behavior of users to recommend items that similar users have liked.
Content-based filtering: This type of system uses the characteristics of items that a user has interacted with in the past to recommend similar items.
Hybrid: To generate suggestions, this kind of system combines content-based filtering and collaborative filtering techniques.
The figure below showcases how these systems behave:



Collaborative filtering is good when recommending for visitors who have a good number of interactions with items but falls short when recommending for visitors who have a low number of interactions.
In contrary, Content-based filtering is good when recommending items similar to the ones the visitor have already interacted with but fails when recommending to visitors who have interacted with a lot of different items.
For this reason, we decided to use a hybrid system, Users who have interacted more frequently can benefit from collaborative filtering, which captures patterns based on user behavior. For users with fewer interactions, content-based filtering, which relies on the features of items, can be more effective.

Architecture:

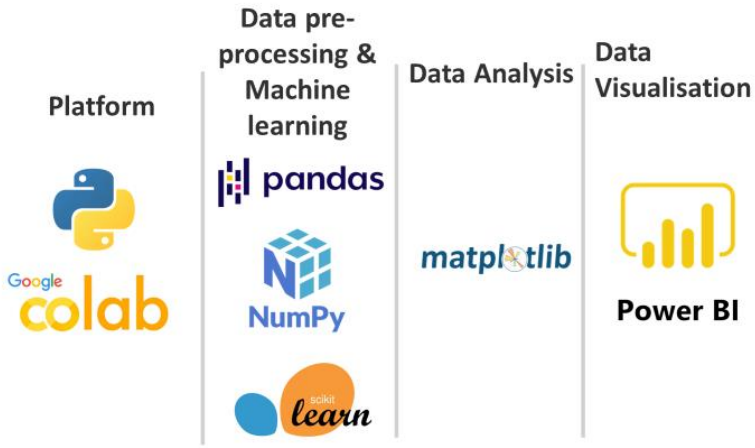


In this architecture, we first start by taking the csv data files provided with the project resources and import them in the Google Colab Platform where the data is cleaned and processed to prepare it for the machine learning and dashboard coming parts.

To answer the client needs, we opted for **Google Colab** as Jupyter Notebook in the cloud, for ease of collaboration, and **Python** as the primary programing language for this project.

Libraries such as **Pandas** and **Numpy** were used for data manipulation and numeric operations, **Matplotlib** allows data analysis, and **Scikit** learn was used for the model creation.

And finally **Power BI** as data visualization and reporting tool.



Hybrid Recommendation System:

Data pre-processing:

To create a collaborative filtering system, we used Python as the programming language and Google Colab as a Jupyter Notebook on the cloud, the first step was the data cleaning & pre-processing, which was contained in four files: *'events.csv', 'category_tree.csv', 'item_properties_part1.csv'* and *item_properties_part2.csv'*.

The file *'event.csv'* consists of 5 columns: **timestamp, visitorid, event, itemid, transactionid.** It represents the data of each event that happened between visitors and items, with the timestamp when the event occurred and the type of the event ('view', 'addtocart','transaction'), there are **2.7 million** rows in this file.

After removing the rows with no value in the **events** column, and replacing the NaN values with 0 in the **transactionid** column, we changed the format of the timestamp into a more readable and easier to analyse format.

The '*item_properties_part2.csv'* is a continuation of the '*item_properties_part1.csv'*, they both consist of the following columns: **timestamp, itemid, property, value.** These files represent the properties of each item, approximatly **20 million** rows.

Before performing any cleaning or pre-processing operation, we merged the data from both files into a single dataframe, the data didn't contain any null values, nor did it contain duplicate rows. The format of the **timestamp** column was changed in the same way as the **timestamp** column in the '*event.csv*' file.

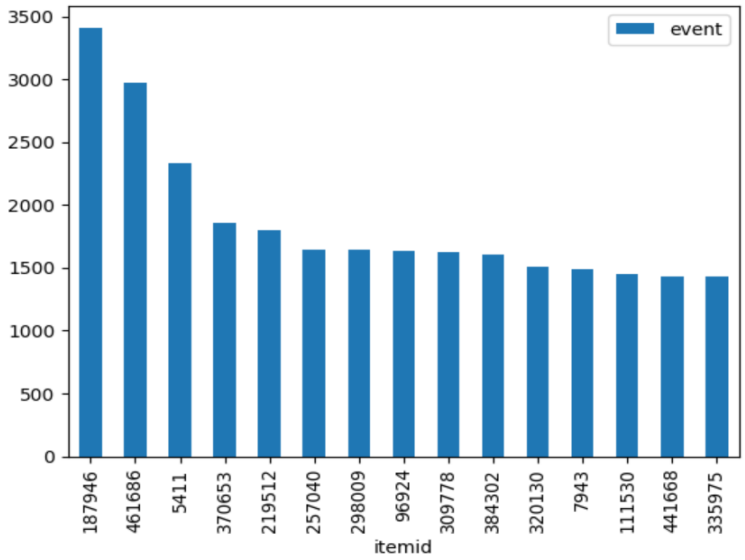
The *'category_tree.csv'* file consists of two columns: **categoryid** and **visitortid**, the file contains **1669** rows.

We removed duplicate rows and replaced the null values of the **parentid** column to '0'.

After completing the cleaning and the pre-processing of the data, we saved and loaded the new data that will be useful in the analysis and model creation.

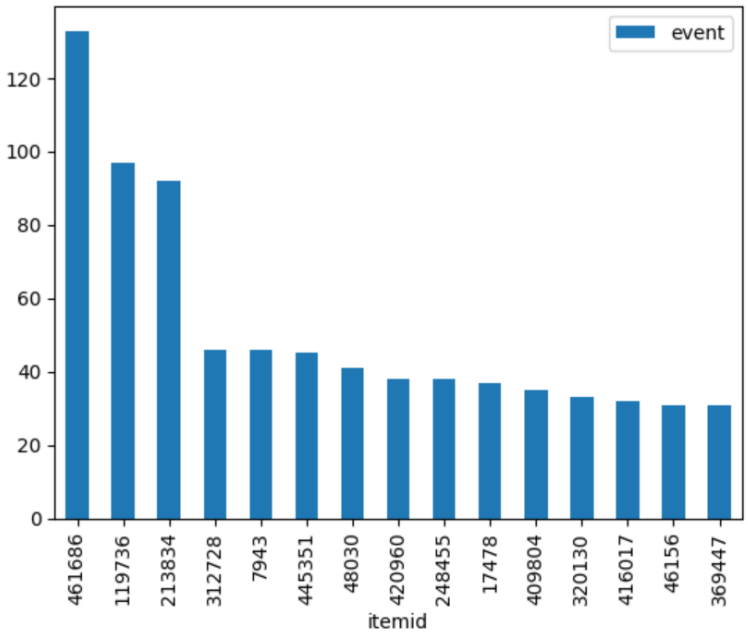
Data analysis:

For a better understanding of the data, we analyzed the data using matplotlib, starting by identifying the most popular products (the products with whom the visitors interacted with the most):



Best-Selling Items

After ranking the items by interactions, let's get more specific and rank them by transactions, this allows us to identify the items that get bought the most.

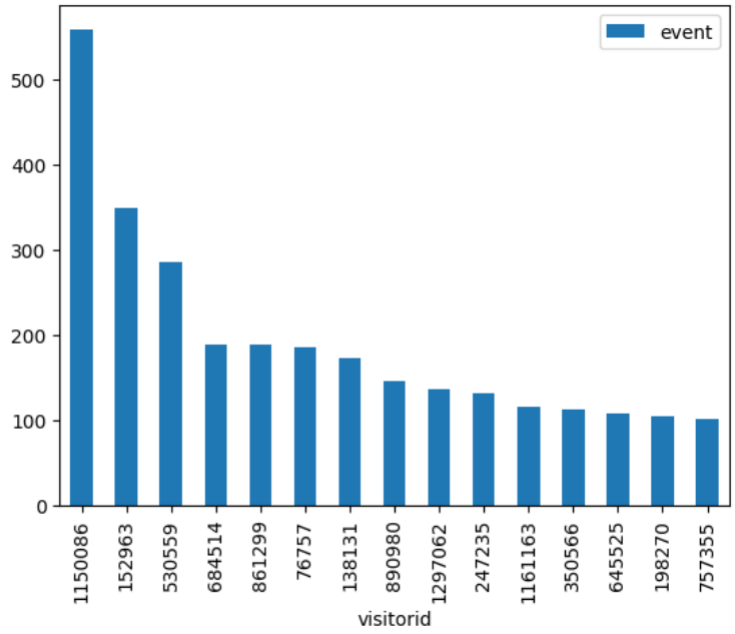


Notice:

Here we can see that there's a difference between the best-selling items, and the items with whom the visitors interacted the most.

Most Transactions per Visitor

Ranking visitors by the number of the transactions they made, this will allow us to identify these users and gain more insights about their behavior:



Machine Learning - Collaborative Filtering:

Data Preparation:

To make a Collaborative Filtering based recommendation system, we need as much data about user behavior as possible, the file events.csv contains all the data needed.

We used Pandas and SciPy for data manipulation.

After importing the libraries and loading the data, we added a new column called **['interaction_value'],** This column will allow the model to know how users behave with each item. It assigns a numeric interaction value based on the **[event]** column's values. If the event is *'view'*, the interaction value is 0.4; if it's *'addtocart'*, the value is 0.6; otherwise (*'transaction'*), it's 0.8. This column represents the strength of interaction between a visitor and an item.

Next, we made a matrix based on the interaction value between each user and item as shown below:

```
(1, 2101)     0.4
(1, 2421)     0.4
(1, 4112)     0.4
(1, 5444)     0.4
(1, 8199)     0.4
(1, 11416)    0.4
(1, 15434)    0.4
(1, 22492)    0.4
(1, 24683)    0.8
```

Model:

For the creation of the model, we used Surprise, a Python scikit for building and analyzing recommender systems, and more precisely the Singular Value Decomposition (SVD) model, since it's the most commonly used when making recommendation systems.

The model was trained using 80% of the data, the rest was used for testing purposes.

When evaluating the model, we experimented with different interaction values and used the Mean Absolute Error (MAE) and the Root Squared Absolute Error (RMSE) for the evaluation, the results were as follows:

For 'view'=0.2, 'addtocart'=0.5 and 'transaction'=0.9:

```
RMSE: 0.0900
MAE:  0.0484
```

For ‘view’=0.3, ‘addtocart’=0.6 and ‘transaction’=0.9:

```
RMSE: 0.0848
MAE:  0.0472
```

For ‘view’=0.4, ‘addtocart’=0.6 and ‘transaction’=0.8:

```
RMSE: 0.0684
MAE:  0.0411
```

For ‘view’=0.4, ‘addtocart’=0.6 and ‘transaction’=0.9:

```
RMSE: 0.0731
MAE:  0.0423
```

The best performing model was the one based on the interaction values of ‘view’=0.4, ‘addtocart’=0.6 and ‘transaction’=0.8.

Recommendations:

Now that the model is ready for use, we made a function that gets a ‘visitorid’ as an input, it outputs the top 10 best items to recommend to that visitor, the figure below shows the best items recommendations for the ‘visitorid’=1:

```
Top Recommendations for User 1
Item: 133472 Estimated Rating: 0.501883503370965
Item: 193990 Estimated Rating: 0.5005390579557638
Item: 224991 Estimated Rating: 0.5000306493486697
Item: 192582 Estimated Rating: 0.4969437262512333
Item: 28789 Estimated Rating: 0.49516420492126617
Item: 188377 Estimated Rating: 0.4937422080911881
Item: 118024 Estimated Rating: 0.49130570312626887
Item: 181830 Estimated Rating: 0.4910667480199934
Item: 213834 Estimated Rating: 0.49089194769614075
Item: 176669 Estimated Rating: 0.48953638250304427
```

Machine Learning – Content-Based Filtering:

Goal:

We will use the Content-based filtering. a method rooted in the idea that users are likely to appreciate products similar to the ones they have recently bought. This recommendation system doesn't rely on the data of other users, allowing it to be implemented without the need for a data collection phase.

Approach :

1. Data Exploration :

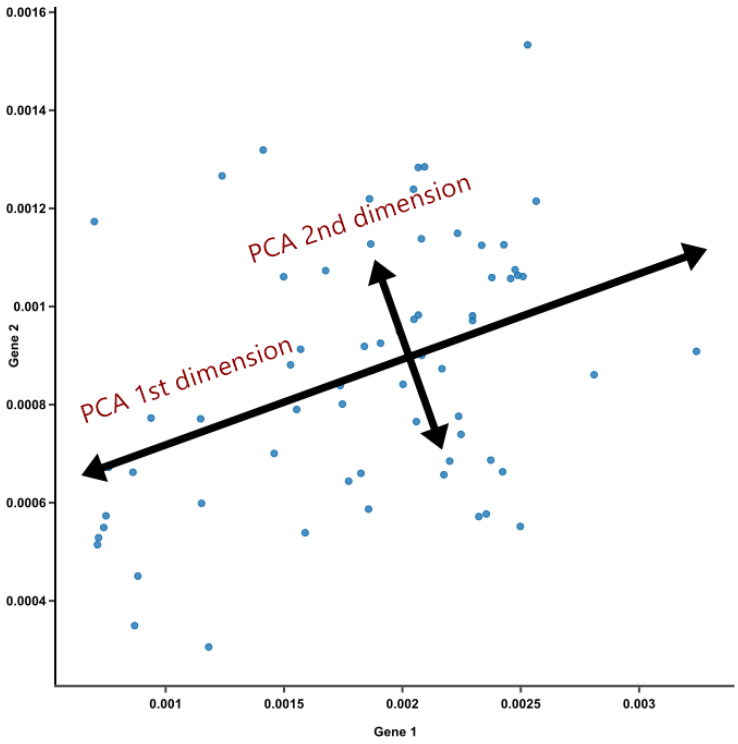
Initially, we will read these files and transform them into pandas dataframes. Subsequently, we will combine them into a single dataframe that consolidates all the data. To optimize the dataset's size and reduce training costs, we intend to retain only the latest item property entry for each item based on the timestamp. Additionally, we will remove any missing values, as we lack information regarding these hashed item properties.

2. Data Preprocessing :

For numerical values beginning with 'n', we will convert them into float values. For text values, we will create embedding vectors. For example, "Hello world 2017!" would be transformed into "24214 44214 n2017.000" and then further reduced to [24214, 44214] after eliminating numbers. Subsequently, we will aggregate these values using the mean. With this preprocessing complete, we will construct the dataframe used for building the Recommender System, utilizing these properties as features in a Vector Space representation. Additionally, we will exclude the "available" column, as it is known to have no influence on product properties, which helps reduce potential bias in our model.

A crucial aspect of our preprocessing phase involves limiting the dataset to just 10,000 items for model construction. The original dataset contains a substantial 417,000 items, which is considerably large. For resource constraints, we are opting to reduce this number significantly.

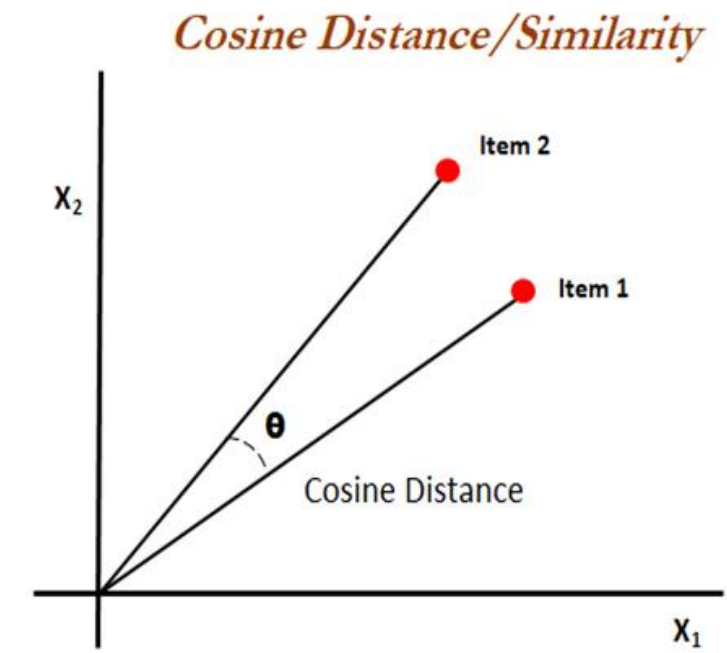
The current vector space we are operating within is exceptionally large, with 1000 dimensions. Consequently, we plan to employ dimensionality reduction techniques, specifically PCA (Principal Component Analysis), retaining only the first 4 principal components. Remarkably, we find that this reduction doesn't result in significant information loss, as we still retain over 90% of the data's variance.



Next, we will eliminate any outliers from our dataset and then apply a MinMaxScaler to normalize the data, bringing it into a [0, 1] scale.

3. Model Building – part 1 :

The step is to know the similarities between each item. To do this, we will calculate the cosine similarities of every pair of items. If 2 data vectors are closed, the angle between these 2 vectors is small and cosine similarity is high.



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

The function **get_recommendations** will receive the itemid, cosine similarity matrix, and a number of the recommended items as the inputs. Then, it will return a list of recommended items.

```
# Demonstration
itemid = 24
print('Top 5 items similar to item',itemid,':')
get_recommendations(itemid, cosine_sim, 5)
```

```
Top 5 items similar to item 24 :

array([328539,  41314,  62021, 157038, 306801], dtype=int64)
```

4. *Model Building – part 2 :*

Later, we will construct a user profile for each individual user, allowing us to precisely discern their preferences derived from their interactions with the platform. This user profile will serve as a foundation for providing tailored recommendations.

We will use a rating scale from 1 to 10, view : 1, addToCart: 5, transaction: 10, and we will keep just the highest rating for each pair (user, item) since we won't need the low values because each transaction follow these steps : view -> addToCart -> transaction

As we are using the user interaction as a rating we will keep just the high value, for example, if a user has viewed an item before buying it, we will keep just the buy event which matches a 10 rating, this will help us reducing the data size, and also working with accurate values.

We will create a function that takes as input : visitorid, it generate his rating vector, then using the property-item matrix, it generate his User Profile, this function will serve as to predict the user rating for an item he didn't interact with.

```
# Demonstration
visitorid = 10
itemid = 24
print('Predicted rating for item',itemid,'by visitor',visitorid,':',predict_rating(visitorid, itemid))
```

```
Predicted rating for item 24 by visitor 10 : 5.94
```

5. Conclusion :

Our model's performance is suboptimal for several reasons. Firstly, our dataset's size is insufficient to construct a robust model, and we had to reduce it for resource constraints. Secondly, the dataset is imbalanced, with limited data available for many users. A significant portion of users has only provided a single rating, contributing to substantial bias in our model.

Data Modeling:

Before delving into data visualization within the geographic context, it's essential to introduce the mapping aspect. which involves representing data points on a map to provide valuable insights when dealing with location-based data.

1) Data Dictionary:

Cleansed_Category Table

Table Name:	Cleansed_Category
Columns :	• CategoryId • ParentId
Description:	This table displays the item's categories and their parents

Cleansed_item_pro Table:

Table Name:	Cleansed_item_pro
Columns :	• CategoryID • Itemid • Property • Timestamp
Description:	This table displays the item's properties such as the category, the id of the item and some hashed properties, and finally the time because all properties within it are time dependent

Cleansed_event Table:

Table Name:	Cleansed_item_pro
Columns :	• event • Itemid • TransactionId • VisitorId • Timestamp (Année,Trimestre, Mois, Jour, hour)
Description:	This table comprises entries that capture diverse user engagements and their associated identifiers, including instances of clicks, additions to the shopping cart, and finalized transactions (View, AddToCart and transaction). These interactions were amassed over four and a half months.

Derived Tables:

Transactions Table:

Table Name:	Transaction
Columns :	• event • Itemid • TransactionId • VisitorId • Timestamp (Année,Trimestre, Mois, Jour, hour)
Description:	This table is derived from the cleansed_event table with a filter where the event= transaction

AddToCart Table:

Table Name:	AddToCart
Columns :	• event • Itemid • TransactionId • VisitorId • Timestamp (Année,Trimestre, Mois, Jour, hour)
Description:	This table is derived from the cleansed_event table with a filter where the event= AddToCart

View Table:

Table Name:	View
Columns :	• event • Itemid • TransactionId • VisitorId • Timestamp (Année,Trimestre, Mois, Jour, hour)
Description:	This table is derived from the cleansed_event table with a filter where the event= View

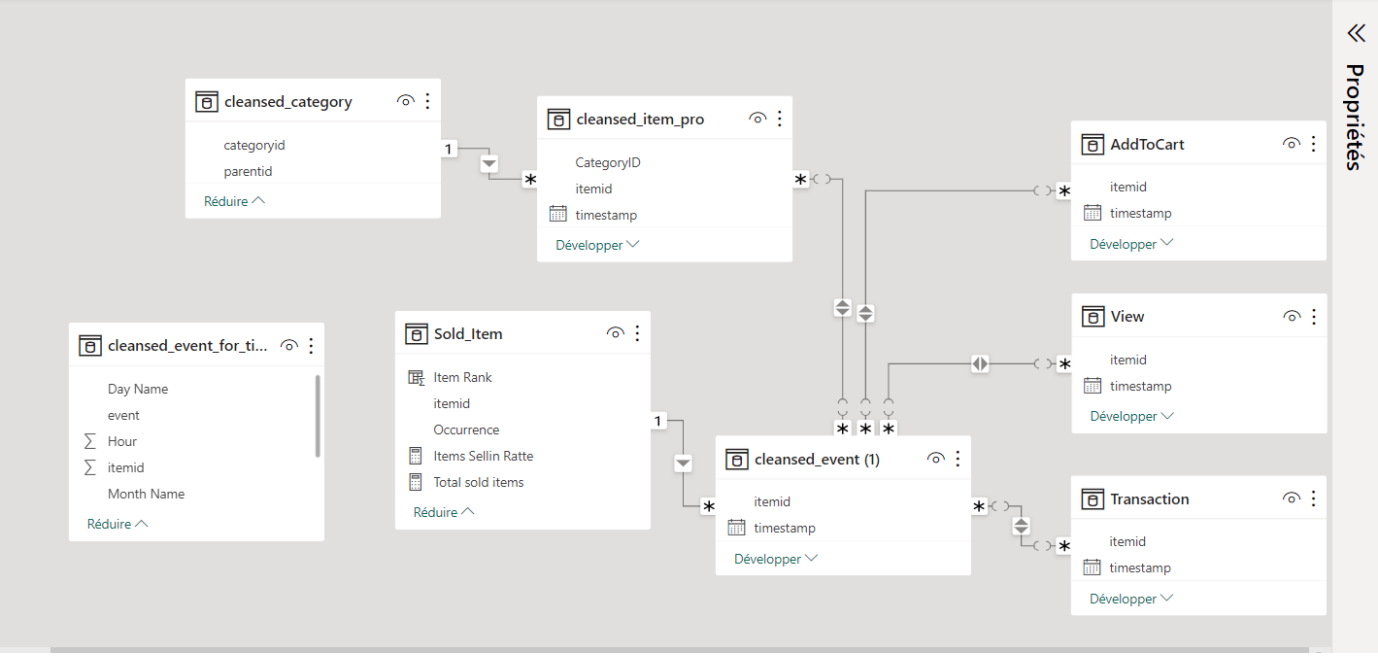
Sold_Items Table:

Table Name:	Sold_Items
Columns :	• event • Itemid
Description:	This table presents the sold items from the Transaction Table and displays their rank and occurrence (how many times they were bought)

These derived tables were created for more ease at creating the visualizations and calculating insight KPI's

2) Entity-Relationship Diagram (ERD):

We created an entity-relationship diagram to visually represent the relationships between tables in our data model.



3) Table's Relationships:

The following table represents the different relationships between the table existing in the data model by displaying the related columns of every relation as well as the cardinality

Relationship	First table's column	Cardinality	Second table's column
	Cleansed_event[itemId]	Many-to-many(*:*)	View[itemId]

	Cleansed_event[itemId]	Many-to-many(*:*)	AddToCart [itemId]
	Cleansed_event[itemId]	Many-to-many(*:*)	Transaction[itemId]
	Cleansed_event[itemId]	Many-to-many(*:*)	Cleansed_item_pro [itemId]
	Cleansed_event[itemId]	Many-to-one(*:1)	Sold_item [itemId]
	Cleansed_item_pro [CategoryID]	Many-to-one(*:1)	Cleansed_category [categoryid]

Key Performance Indicators:

→ **Conversion (A to T):**

This KPI illustrates the percentage of the number of items bought out of the number of items added to the cart.

We added a measure in Power BI called **Conversion (A to T)**.

```
Conversion (A to T) = DIVIDE(COUNTROWS('Transaction'), (COUNTROWS('AddToCart')),0)
```

→ **Conversion (V to A):**

This KPI illustrates the percentage of the number of items added to the cart out of the number of items viewed.

We added a measure in Power BI called **Conversion (V to A)**.

```
Conversion (V to A) = DIVIDE(COUNTROWS('AddToCart'), (COUNTROWS('View')),0)
```

→ **Items Selling Rate:**

This KPI illustrates the percentage of the number of items bought out of the total of existing items

We added a measure in Power BI called **Items Selling Rate**.

```
Items Sellin Ratte = DIVIDE(COUNTROWS(Sold_Item), (COUNTROWS('cleansed_item_pro')),0)
```

→ **Average Items Viewed per Visitor:**

This KPI illustrates the average number of items viewed per visitor:

We added a measure in Power BI called **AverageItemsViewedperVisitor**.

```
AverageItemsViewedperVisitor = DIVIDE(COUNTROWS(FILTER('cleansed_event (1)', 'cleansed_event (1)'[event] = "view")), COUNTROWS(SUMMARIZE('cleansed_event (1)', 'cleansed_event (1)'[visitorid])))
```

→ **Average Items Added to Cart per Visitor:**

This KPI illustrates the average number of items added to cart per visitor:

We added a measure in Power BI called **AverageItemsAddedToCartperVisitor**.

```
AverageItemsAddedToCartperVisitor = DIVIDE(COUNTROWS('AddToCart'),
COUNTROWS(SUMMARIZE('AddToCart', 'AddToCart'[visitorid])))
```

→ **Average Items Bought per Visitor:**

This KPI illustrates the average number of items added to cart per visitor:

We added a measure in Power BI called **AverageItemsBoughtperVisitor**.

```
AverageItemsBoughtperVisitor = DIVIDE(COUNTROWS('Transaction'),
COUNTROWS(SUMMARIZE('Transaction', 'Transaction'[visitorid])))
```

→ **Average Views Before Transaction:**

This KPI measures the average number of views before purchasing an item.
To create this KPI, we created a new csv file in Google Colab called *viewsbeforetransactionTable* with the following code:

```
#Average number of views before transaction
import pandas as pd
import numpy as np

df= pd.read_csv('/content/cleansed_event (1).csv')

transaction_df = df[df['event'] == 'transaction']
view_df = df[df['event'] == 'view']

merged_df = pd.merge(view_df, transaction_df[['visitorid', 'itemid']], on=['visitorid', 'itemid'], how='inner'

result = merged_df.groupby(['visitorid', 'itemid']).size().reset_index(name='views_before_transaction')

# result.to_csv("viewsbeforetransactionTable.csv",index=False)
```

We then imported it to PowerBI and created a new measure with the following DAX syntax:

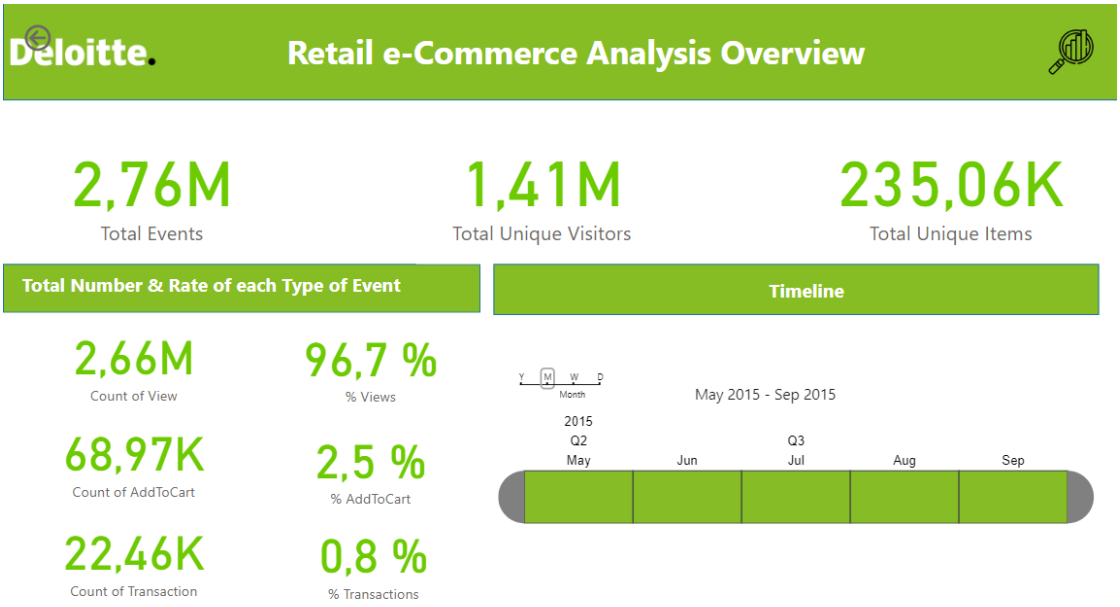
```
AverageViewsBeforeTransaction =
AVERAGEX(
    viewsbeforetransactionTable,
    [views_before_transaction]
)
```

Dashboard

We chose to divide our Dashboard into three pages:

Page 1: Overview

In this first page, we have represented the visualizations that provide a high-level summary of key metrics and insights to give a quick understanding of the overall situation.



We have three types of events:

- **View:** when the visitor clicks on an item
- **AddToCart:** when the visitor adds an item to its cart
- **Transaction:** when the visitor buys an item

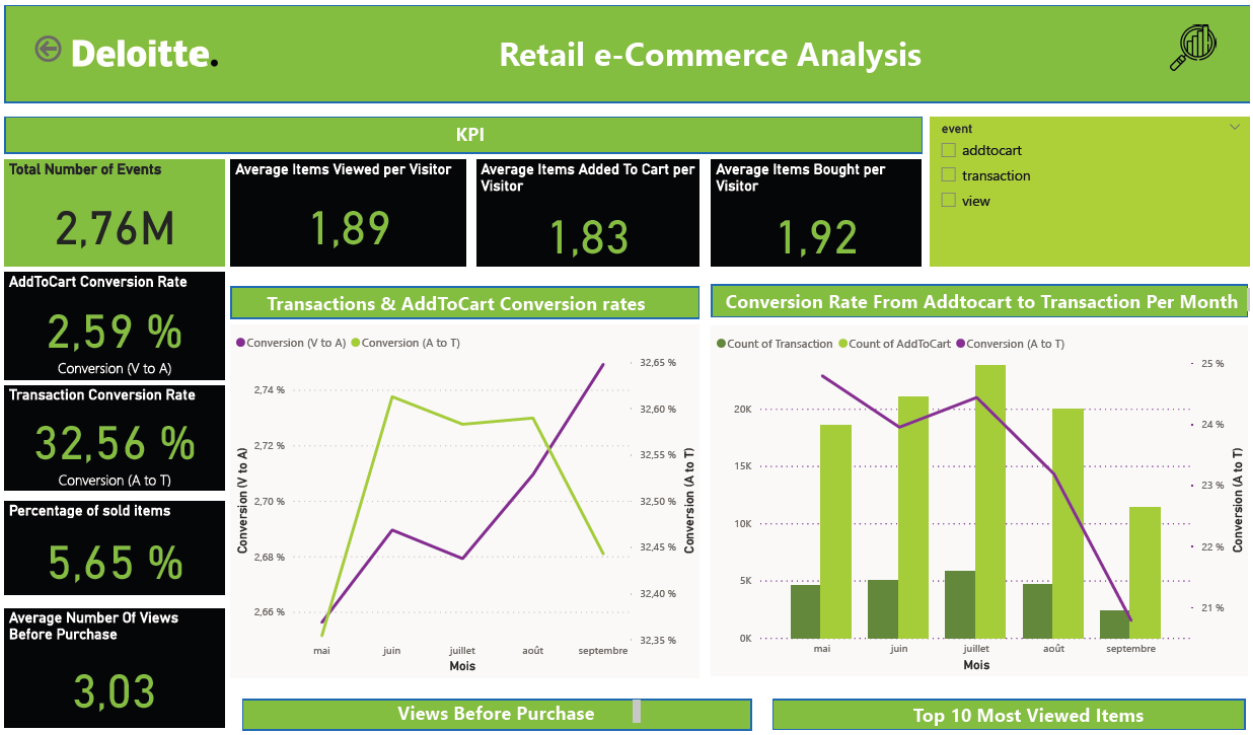
In total, we have 2 760 000 events including 2 660 000 views which make up approximately 96.7% of all events, 68 970 addtocart accounting for about 2.5% of all events and 22 460 transactions representing around 0.8% of all events.

These events are carried out by 1 410 000 visitors on 235 060 unique items.

The data extends over a period of 4 and a half months (from 2015-05-03 to 2015-09-18).

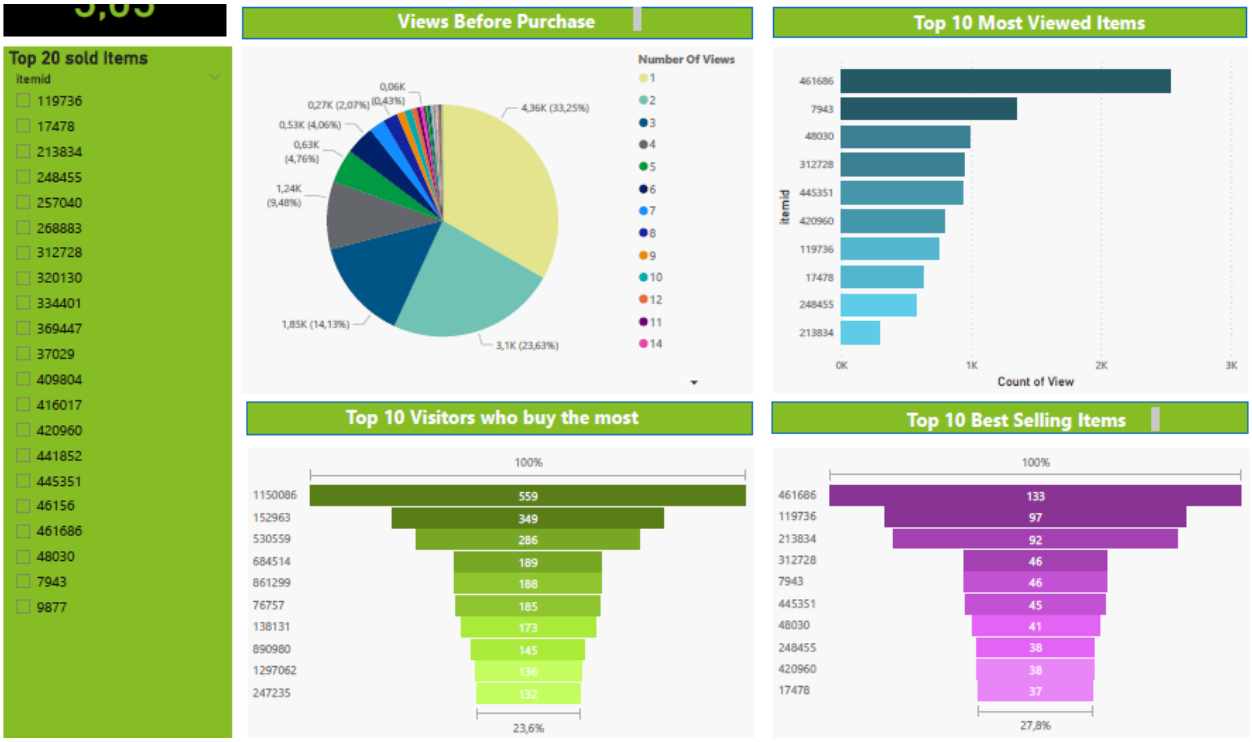
Page 2: Analysis

We tried to dedicate this page to showcasing key insights and visualizations from our data analysis.



We gathered the previously created KPIs in the KPI section which we can interpret as follows:

- The AddToCart Conversion Rate indicates that, approximatively, 2 out of 100 visitors, who viewed an item added it to their carts.
- The transaction conversion rate shows that, approximatively, 32 out of 100 visitors, who expressed initial interest by adding items to their carts followed through with a purchase.
- On average each visitor views, adds to cart and eventually buys 2 items.
- Before buying an item, visitors view it on average three times.



From the visualizations, we can extract the following observations:

- Visitors are more likely to buy a product if they have viewed it multiple times, as this suggests that they are interested in the product and have taken the time to learn more about it. Visitors may be viewing items on multiple devices, such as their phone, computer or tablet. This could also explain why some customers view a product multiple times before purchasing it.
- The 10 best selling items are the items 461686, 119736, 233834, 312728, 7943, 445351, 48030, 248455, 420960, 17478. Therefore, we suggest recommending those items to new visitors because it can create a sense of trust and assurance and consequently, they may be more inclined to purchase items that others have already found valuable.

Page 3: Events Time Distribution

On this third page, we have displayed visualizations with a temporal scope, enabling us to analyze the distribution of data over time.

- **Event Count Heatmap by Day and Hour:**

The heatmap shows the count of events by day and hour. The darker the color, the higher the count of events.

The heatmap shows that most events occur on weekdays between 16h and midnight, with a peak between 19H and midnight. There are fewer events on weekends, and the fewest events occur on Sunday mornings.

Sunday mornings are the least busy time for the events. This could be because people are sleeping in or spending time with family and friends.

- **Event Count Heatmap by Month and Day:**

This heatmap shows the count of events by month and day.

The heatmap shows that most of the events occur in June and July, with a peak in July. The are fewer events in August and May, and the fewest events occur in September.

July is the peak time for the events. This could be because people want to buy new things during their vacations.

September is the least busy time for the events. This could be because people are preparing for the start of the school year and consequently, they are too busy to browse products on the website.

- **Total events by day:**

The histogram shows the distribution of the number of events per day. The x-axis shows the name of the days, and the y-axis shows the number of events.

Most events occur on Tuesday, Monday and Wednesday and Saturday records the fewest events.

- **Total events by month:**

The histogram shows the distribution of the number of events per month. The x-axis shows the name of the months, and the y-axis shows the number of events.

Most events occur on July, June, May and August with a peak in July. September records the fewest events.

- **Total events by hour:**

The histogram shows the distribution of the number of events per hour. The x-axis shows the hour, and the y-axis shows the number of events.

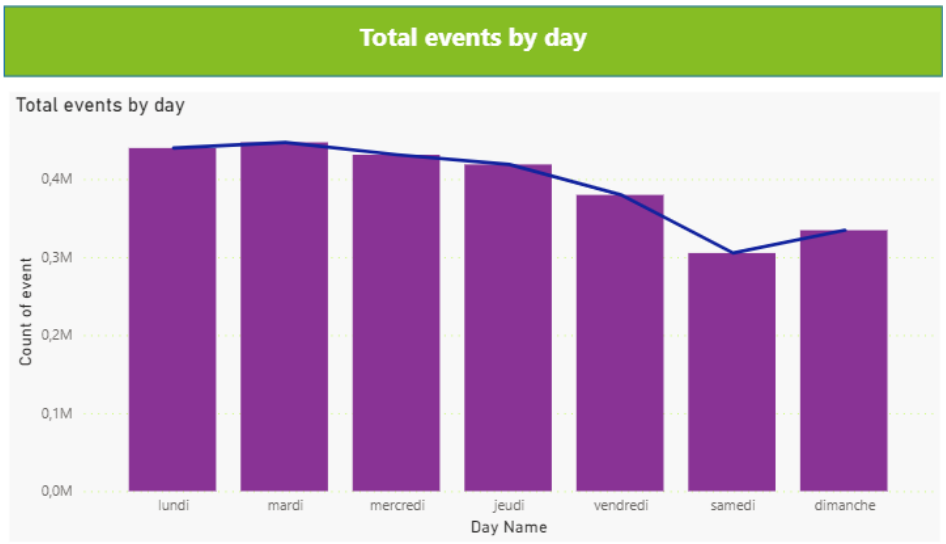
Most events occur between 17H and 20H and least events occur in the mornings (between 6H and 13H).

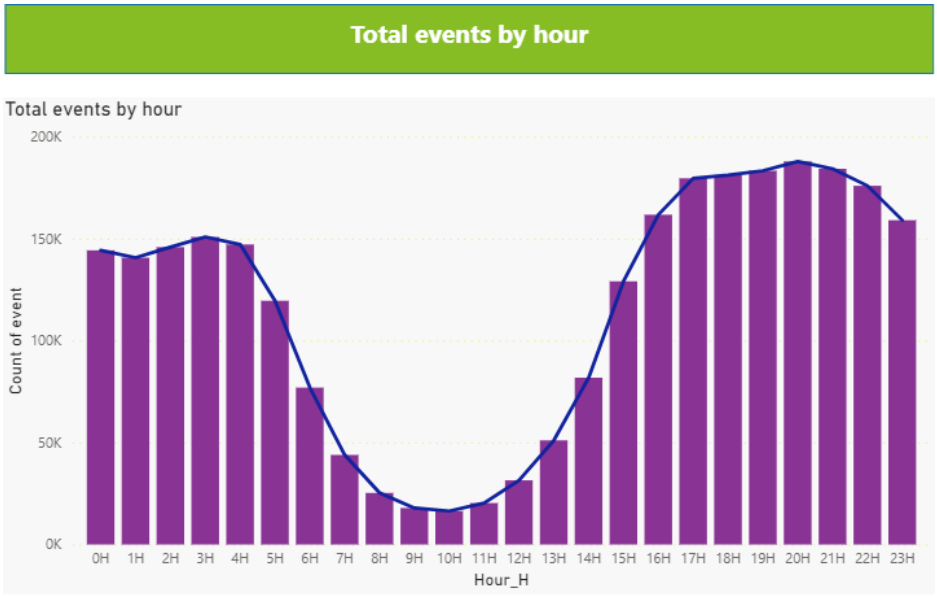
Insights:

After analyzing the data, we gathered useful recommendations for the client

Interactions by Visitors by Day & Hour:

The two charts below give a clear view of the moments when the website is interacted with the most:





It's clear that the day with the least interactions are Friday, Saturday and Sunday, the hours where the visitors interact with items the least are between 6AM and 2PM, the days and hours mentionned are the best choice when planning to shut down the system for an update, and best avoided when planning for promotions and advertising.

Data quality:

When working with the items data, we noticed that nearly 50% of the items that appeared in the transactions are missing their category, which hinders the quality of the recommendation system and reduces the quality of the insights that can be extracted from the item properties.

We suggest improving the quality of the data about the items, by adding the category column in the '*item_properties.csv*' file, separating it from the property's column, the same can be done for the availability property.

This will increase the level of detail that can be extracted from the data, which will lead to better decision making, and a more precise recommendation system.

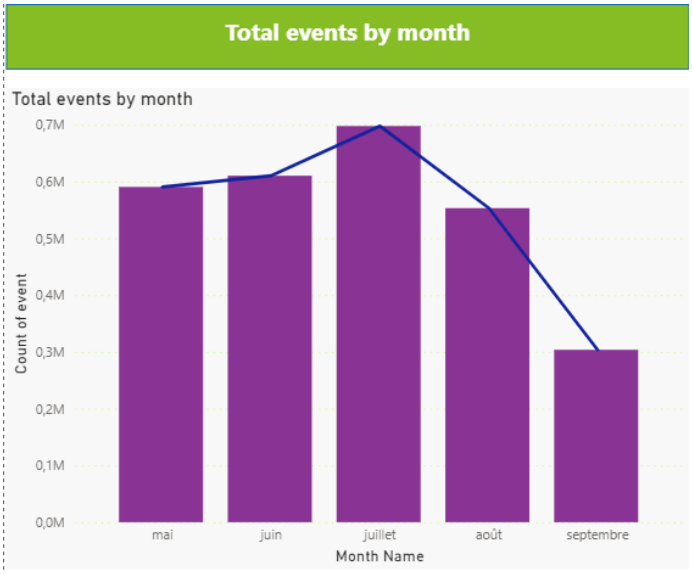
Reducing the Number of Items:

The total number of items is **417 053**, the total number of items with whom the visitors interacted (including: 'views', 'addtocart', 'transaction') is **235 060** items, which means that nearly 50% of the items didn't get a single interaction for the entire duration.

More items mean more warehouse space (which increases the rent), and more defective items. It's also important to note that the lesser the number of options a client has the clearer it is to decide (according to research, link: Increase your eCommerce Sales With Fewer Options | VWO), according to the reasons aforementioned, it would be better to reduce the total number of items.

Improve engagement:

The bar chart below represents the total number of events by month, July is the best performing month, but august is the month with the least number of events (we only have data until 16 September)



There are many reasons that can cause the number of events to fall off in August such as seasonal items, the availability of the items, the reduction of the advertising budget or increased competition, we suggest to the client to diversify their marketing strategies during August. Consider offering special promotions or exclusive deals to entice customers. Additionally, focusing on online platforms and social media can help maintain engagement during this period, ensuring a steady flow of customers despite the seasonal dip in events.