Recommender

KaDo Project

Introduction:

Recommender systems have become a very important part of the retail, social networking, and entertainment industries. From providing advice on songs for you to try, suggesting books for you to read, or finding clothes to buy, recommender systems have greatly improved the ability of customers to make choices more easily.

The goal of this project is to build recommendation engines for companies so they can offer one or more items to their clients, depending on their preferences.

This report will explain to you in detail all what we did...

Recommendation system

The database that we were given contains all the purchases made during a year for different items divided in 3 categories.

Depending on several factors, such as the kind of data we have, the ability to scale and the recommendation transparency., we can decide which methodology and option we can use to build our engine.

In our case, and since the database haven't information about the users and we have no rating for the items that users purchase so it's hard to know if the user did like the product or not, in this case, it's highly recommended to use/ best choice will be) collaborative filtering because it doesn't require any information about the users or items.

Collaborative Filtering

Collaborative filtering is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users.
All we need for the collaborative filtering, is a rating of some kind for each user/item interaction that occurred where available. for that there are two kinds of data available for this type of interaction: explicit and implicit.

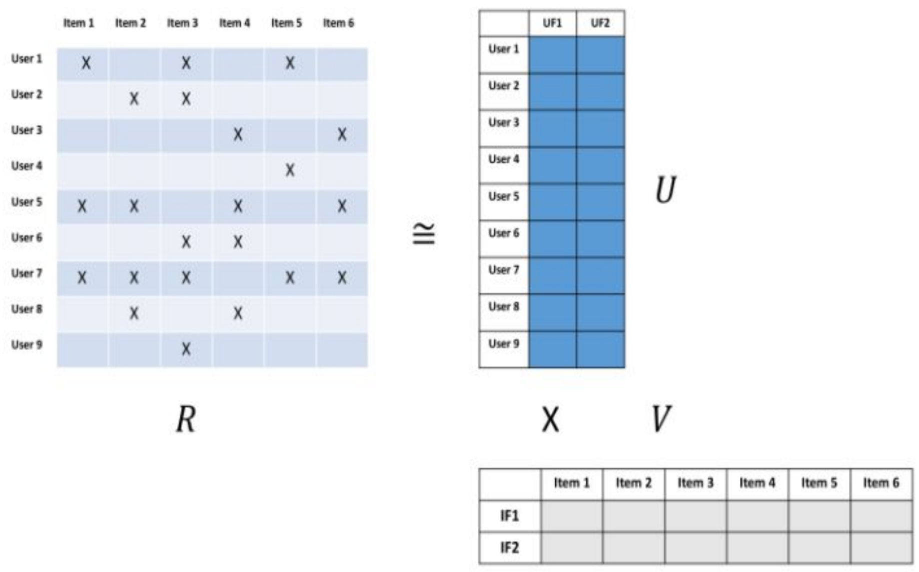
- Explicit: A score, such as a rating or a like ()
- Implicit: Not as obvious in terms of preference, such as a click, view, or purchase

And since more data usually means a better model, implicit feedback is where our efforts should be focused. While there are a variety of ways to tackle collaborative filtering with implicit feedback, we choose to use the method included in **Spark's library** used for collaborative filtering, alternating least squares (ALS).

Alternating Least Squares

Simply, we can take a large matrix of user/item interactions and figure out the latent (or hidden) features that relate them to each other in a much smaller matrix of user features and item features? That’s exactly what ALS is trying to do through matrix factorization.

In more details and the image below demonstrates, assume we have an original ratings matrix R of size, where M is the number of users and N is the number of items. This matrix is quite **sparse**, since most users only interact with a few items each. We can factorize this matrix into two separate smaller matrices: one with dimensions MxK which will be our latent **user feature** vectors for each user (U) and a second with dimensions KxN, which will have our latent **item feature** vectors for each item (V). Multiplying these two feature matrices together approximates the original matrix, but now we have two matrices that are dense including a number of latent features K for each of our items and users.



	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	X		X		X	
User 2		X	X			
User 3				X		X
User 4					X	
User 5	X	X		X		X
User 6			X	X		
User 7	X	X	X		X	X
User 8		X		X		
User 9			X			

	UF1	UF2
User 1		
User 2		
User 3		
User 4		
User 5		
User 6		
User 7		
User 8		
User 9		

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
IF1						
IF2						

Once we defined the algorithms that we are going to use now ::::::

Steps:

1. Processing the Data

Processing the Data

The data we are using to build our recommendation engine, has more than 700 00 rows and we need to take all of the transactions for each customer and put these into a format ALS can use. This means we need each unique customer ID (CLI_ID) in the rows of the matrix, and each unique item ID in the columns of the matrix. The values of the matrix should be the total number of purchases for each item by each customer (Quantity).

Our dataset includes the ticket ID for each user CLI_ID, along with the LIBELLE an item price, the family (famille), the universe (univers), the month of purchase.

there are few necessary steps that we done before we start using our dataset:

1. check to see if there are any missing values in the data (with simple command line)
2. make a lookup table that keeps track of each item ID and the Libel of the that Item
3. Group purchase quantities together by stock code and item ID
4. Change any sums that equal zero to one.
5. Only include customers with a positive purchase total to eliminate possible errors
6. Set up our sparse ratings matrix

Here is the libraries that we will use for data processing are:

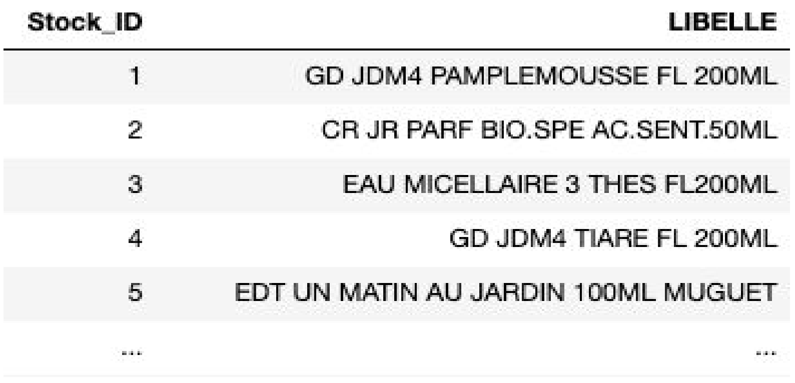
- **Pandas**: a data analysis and manipulation tool.
- **Scipy.sparse**: to store data that contains a large number of zero-valued elements can both save a significant amount of memory and speed up the processing of that data
- **numpy**: python extension for manipulating matrices or multidimensionnel tables
- **Random**
- **implicit:**
- **sklearn:**

1- Lookup table

This table will be helpful for telling us what each item (LABELLE) is we need to, such as that StockCode 5 is EDT UN MATIN AU JARDIN 100ML MUGUET.

To build this table it's quite simple, first we need to remove all the duplicates of LIBELLE the our dataset then all the columns that we won't use such as CLI_ID, PRIX_NET... and put it in item_lookup dataFrame, once this is done add a new column SOCKETCode to item_lookup where we use the map(hash) function to generate a code for each LIBELLE

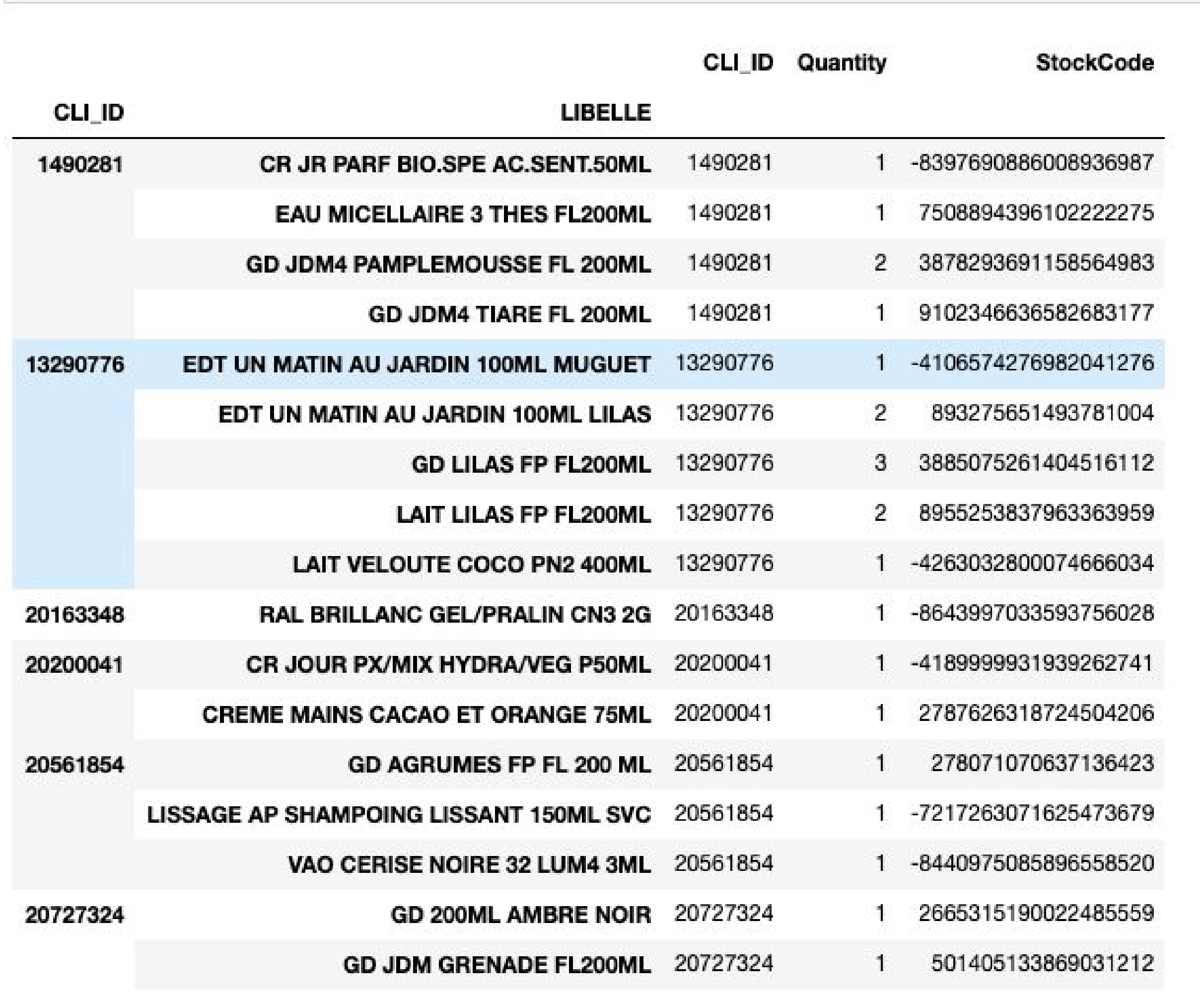
and since we have about 1484 different items, we get a table with 1484 rows



Stock_ID	LIBELLE
1	GD JDM4 PAMPLEMOUSSE FL 200ML
2	CR JR PARF BIO.SPE AC.SENT.50ML
3	EAU MICELLAIRE 3 THES FL200ML
4	GD JDM4 TIARE FL 200ML
5	EDT UN MATIN AU JARDIN 100ML MUGUET
...	...

2- Group purchase quantities together by stock code and client ID

For this step we need to start by convert to integer for customer ID, get rid of unnecessary info, after that group together the Libelle Id and the Client Id
Finally Replace a sum of zero purchases with a one to indicate purchased and only get customers where purchase totals were positive.
If we look at our final resulting matrix of grouped purchases, we see the following:



		CLI_ID	Quantity	StockCode
CLI_ID	LIBELLE			
1490281	CR JR PARF BIO.SPE AC.SENT.50ML	1490281	1	-8397690886008936987
	EAU MICELLAIRE 3 THES FL200ML	1490281	1	7508894396102222275
	GD JDM4 PAMPLEMOUSSE FL 200ML	1490281	2	3878293691158564983
	GD JDM4 TIARE FL 200ML	1490281	1	9102346636582683177
13290776	EDT UN MATIN AU JARDIN 100ML MUGUET	13290776	1	-4106574276982041276
	EDT UN MATIN AU JARDIN 100ML LILAS	13290776	2	893275651493781004
	GD LILAS FP FL200ML	13290776	3	3885075261404516112
	LAIT LILAS FP FL200ML	13290776	2	8955253837963363959
	LAIT VELOUTE COCO PN2 400ML	13290776	1	-4263032800074666034
20163348	RAL BRILLANC GEL/PRALIN CN3 2G	20163348	1	-8643997033593756028
20200041	CR JOUR PX/MIX HYDRA/VEG P50ML	20200041	1	-4189999931939262741
	CREME MAINS CACAO ET ORANGE 75ML	20200041	1	2787626318724504206
20561854	GD AGRUMES FP FL 200 ML	20561854	1	278071070637136423
	LISSAGE AP SHAMPOING LISSANT 150ML SVC	20561854	1	-7217263071625473679
	VAO CERISE NOIRE 32 LUM4 3ML	20561854	1	-8440975085896558520
20727324	GD 200ML AMBRE NOIR	20727324	1	2665315190022485559
	GD JDM GRENADE FL200ML	20727324	1	501405133869031212

For each client we have all the items he purchased with the quantity.

“”Instead of representing an **explicit rating**, the purchase quantity can represent a **confidence** in terms of how strong the interaction was. Items with a larger number of purchases by a customer can carry more weight in our ratings matrix of purchases.””

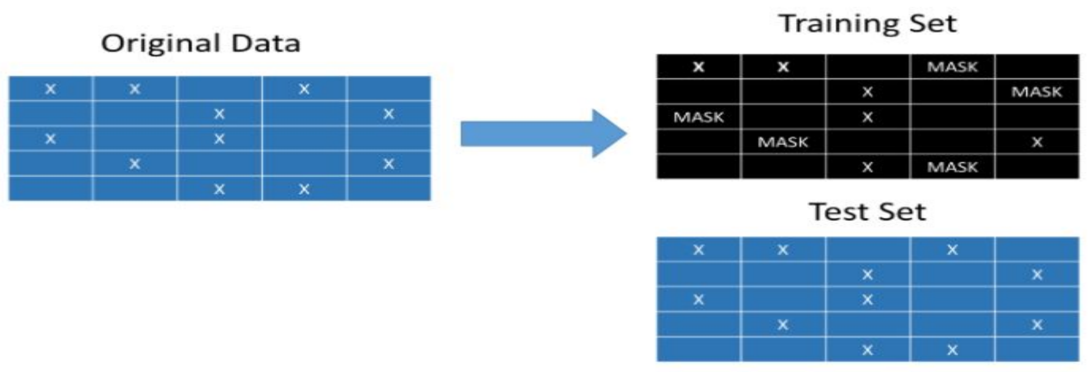
The last step we did is to create the sparse ratings matrix of users and items, but first what's a sparse matrix? Sparse matrices are distinct from matrices with mostly non-zero values, which are referred to as dense matrices.

to create a sparse matrix we needed to get our unique clients and items that were purchased with the quantity, then we had to get the associated row and column indices when we check our final matrix object:

For that we have 853514 customers with 1484 items. For these user/item interactions, 5911149 of these items had a purchase. In terms of sparsity of the matrix, that makes 99.53 % of the interaction matrix is sparse.

Creating a Training and Validation Set

With collaborative filtering, we need all of the user/item interactions to find the proper matrix factorization. The best method is to hide a certain percentage of the user/item interactions from the model during the training phase chosen at random. Then, check during the test phase how many of the items that were recommended the user actually ended up purchasing in the end.



Our test set is an exact copy of our original data. The training set, however, will mask a random percentage of user/item interactions and act as if the user never purchased the item (making it a sparse entry with a zero). We then checked in the test set which items were recommended to the user that they ended up actually purchasing. If the users frequently ended up purchasing the items most recommended to them by the system, we can conclude the system seems to be working.

As an additional check, we also compared our system to simply recommending the most popular items to every user.

After that we created a function that helped us to separate our training and testing sets, this function will take in the original user-item matrix and "mask" a percentage of the original ratings where a user-item interaction has taken place for use as a test set. The test set will contain all of the original ratings, while the training set replaces the specified percentage of them with a zero in the original ratings matrix.

this function will take as a parameters:

ratings: the original ratings matrix from which we want to generate a train/test set.

pct_test: The percentage of user-item interactions where an interaction took place that you want to mask in the training set

And it returns:

training_set: The altered version of the original data with a certain percentage of the user-item pairs that originally had interaction set back to zero.

test_set: A copy of the original ratings matrix, unaltered, so it can be used to see how the rank order compares with the actual interactions.

user_inds: From the randomly selected user-item indices, which user rows were altered in the training data.

Once we have our train/test split, we start to implement the alternating least squares algorithm.

Implementing ALS for Implicit Feedback

Implicit weighted ALS is designed for alternating least squares and implicit feedback based collaborative filtering. It takes as a parameters:

training_set: Our matrix of ratings
lambda_val: Used for regularization during alternating least squares.
alpha: The parameter associated with the confidence matrix
iterations: The number of times to alternate between both user feature vector and item feature vector
rank_size: The number of latent features in the user/item feature vectors.
seed: Set the seed for reproducible results

It returns:
The feature vectors for users and items. The dot product of these feature vectors should give you the expected "rating" at each point in your original matrix.

Evaluating the Recommender System

Since our training set had 20% of the purchases masked, This will allow us to evaluate the performance of our recommender system. Essentially, we need to see if the order of recommendations given for each user matches the items they ended up purchasing. A commonly used metric for this kind of problem is the area under the Receiver Operating Characteristic (or ROC) curve. A greater area under the curve means we are recommending items that end up being purchased near the top of the list of recommended items.

In order to do that, we needed to write a function that can calculate a mean area under the curve (AUC) for any user that had at least one masked item.

First, we made a simple function that can calculate our AUC.

This simple function will output the area under the curve using sklearn's metrics.

parameters

predictions: your prediction output

test: the actual target result you are comparing to

returns:

AUC (area under the Receiver Operating Characterisic curve)

Now, we utilize this helper function inside of a second function that will calculate the AUC for each user in our training set that has at least one item masked. It should also calculate AUC for the most popular items for our users to compare.

This function will calculate the mean AUC by user for any user that had their user-item matrix altered.

parameters:

training_set: The training set resulting from make_train, where a certain percentage of the original user/item interactions are reset to zero to hide them from the model

predictions: The matrix of your predicted ratings for each user/item pair as output from the implicit MF.These should be stored in a list, with user vectors as item zero and item vectors as item one.

altered_users: The indices of the users where at least one user/item pair was altered from make_train function

test_set: The test set constructed earlier from make_train function

returns: The mean AUC (area under the Receiver Operator Characteristic curve) of the test set only on user-item interactions

We can now use this function to see how our recommender system is doing. To use this function, we will need to transform our output from the ALS function to csr_matrix format and transpose the item vectors.

```
(0.803, 0.851)
```

We can see that our recommender system beat popularity. Our system had a mean AUC of 0.803, while the popular item benchmark had a lower AUC of 0.814. You can go back and tune the hyperparameters if you wish to see if you can get a higher AUC score. Ideally, you would have separate train, cross-validation, and test sets so that you aren't overfitting while tuning the hyperparameters, but this setup is adequate to demonstrate how to check that the system is working.

A Recommendation Example

We now have our recommender system trained and have proven it beats the benchmark of popularity. An AUC of 0.803 means the system is recommending items the user in fact had purchased in the test set far more frequently than items the user never ended up purchasing. To see an example of how it works, to examine the recommendations given to a particular user and decide subjectively if they make any sense.

First, however, we need to find a way of retrieving the items already purchased by a user in the training set. Initially, we will create an array of our customers and items we made earlier.

Now, we can create a function that will return a list of the item descriptions from our earlier created item lookup table.

This just tells me which items have been already purchased by a specific user in the training set. It takes as parameters:

customer_id: Input the customer's id

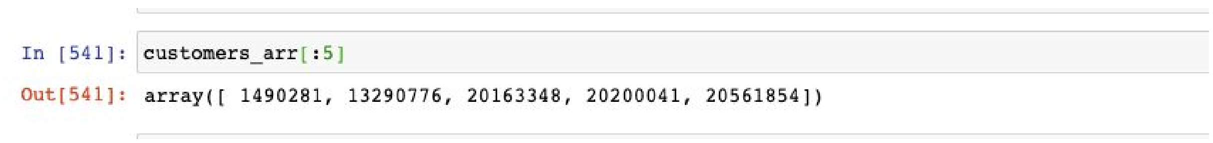
mf_train: The initial ratings training set used

customers_list: The array of customers used in the ratings matrix

products_list: The array of products used in the ratings matrix

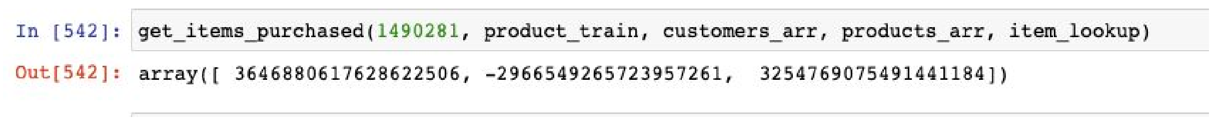
item_lookup: A simple pandas dataframe of the unique product ID/product descriptions available

It returns a list of item IDs and item libels for a particular customer that were already purchased in the training set



```
In [541]: customers_arr[:5]
Out[541]: array([ 1490281, 13290776, 20163348, 20200041, 20561854])
```

we can see that the first customer listed has an ID of 1490281. Let's examine their purchases from the training set.



```
In [542]: get_items_purchased(1490281, product_train, customers_arr, products_arr, item_lookup)
Out[542]: array([ 3646880617628622506, -2966549265723957261,  3254769075491441184])
```

We can see that the customer purchased a CR JR PARF BIO.SPE AC.SENT.50ML, GD JDM4 PAMPLEMOUSSE FL 200ML,EAU MICELLAIRE 3 THES FL200ML. What items does the recommender system say this customer should purchase? We need to create another function that does this. Let's also import the MinMaxScaler from scikit-learn to help with this.

This function will return the top recommended items to our users

parameters:

customer_id: Input the customer's id

mf_train: The training matrix you used for matrix factorization fitting

user_vecs: the user vectors from your fitted matrix factorization

item_vecs: the item vectors from your fitted matrix factorization

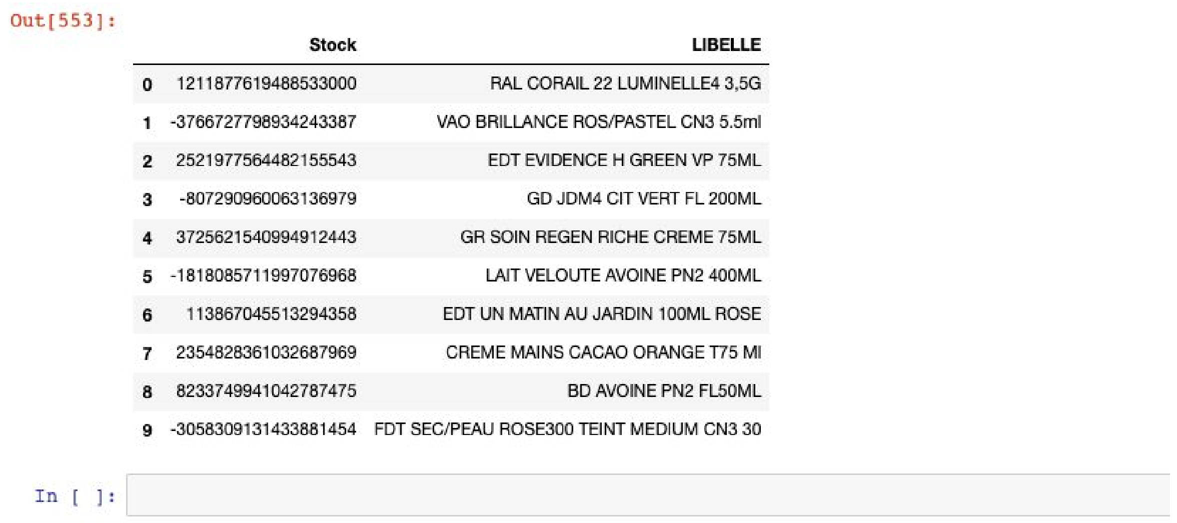
customer_list: an array of the customer's ID

item_list: an array of the products

item_lookup: A simple pandas dataframe of the unique product ID/product descriptions available

num_items: The number of items you want to recommend in order of best recommendations. Default is 10.

And returns the top n recommendations chosen based on the user/item vectors for items never interacted with/purchased



Out[553]:

	Stock	LIBELLE
0	1211877619488533000	RAL CORAIL 22 LUMINELLE4 3,5G
1	-3766727798934243387	VAO BRILLANCE ROS/PASTEL CN3 5.5ml
2	2521977564482155543	EDT EVIDENCE H GREEN VP 75ML
3	-807290960063136979	GD JDM4 CIT VERT FL 200ML
4	3725621540994912443	GR SOIN REGEN RICHE CREME 75ML
5	-1818085711997076968	LAIT VELOUTE AVOINE PN2 400ML
6	113867045513294358	EDT UN MATIN AU JARDIN 100ML ROSE
7	2354828361032687969	CREME MAINS CACAO ORANGE T75 MI
8	8233749941042787475	BD AVOINE PN2 FL50ML
9	-3058309131433881454	FDT SEC/PEAU ROSE300 TEINT MEDIUM CN3 30

In [ ]: