

**Book Recommandation System**

**Assigning Teacher:**

* **Mm DIF**

**Group:**

* **Messaoudi Nabehet**
* **Melissa Meski**

**School Year**

**2023/2024**

##### **Machine Learning Project Report**

**What’s Recommendation System?**

A recommendation system, driven by artificial intelligence and machine learning, utilizes Big Data to provide personalized suggestions or recommendations to users. These suggestions are based on various criteria, such as past purchases, search history, and demographic information. Recommender systems play a crucial role in helping users discover products and services that align with their preferences, enhancing the overall user experience. These systems are trained to comprehend user preferences, past decisions, and product characteristics by analyzing data related to user interactions, including impressions, clicks, likes, and purchases. The predictive capabilities of recommender systems make them highly valuable for content and product providers, enabling them to guide consumers towards products or services tailored to their individual interests, spanning a wide range from books and videos to health classes and clothing.

**Types of Recommendation Systems:**

While there are a vast number of recommender algorithms and techniques, most fall into these broad categories**:**

* **collaborative filtering** :
* *Memory Based :*

1. User-User CF
2. Item-Item CF

* *Model Based:*

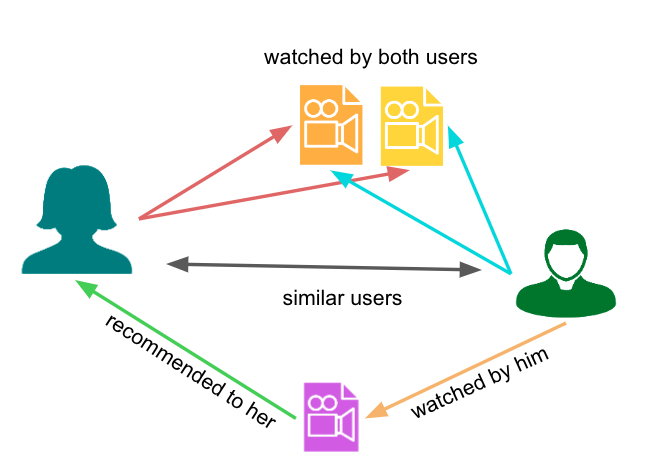
1. SVD

* **content filtering**
* **Collaborative filtering** :

Collaborative Filtering is a recommendation system technique that leverages user behavior data to make predictions about a user's preferences. It encompasses two main approaches: User-Based Collaborative Filtering and Item-Based Collaborative Filtering.

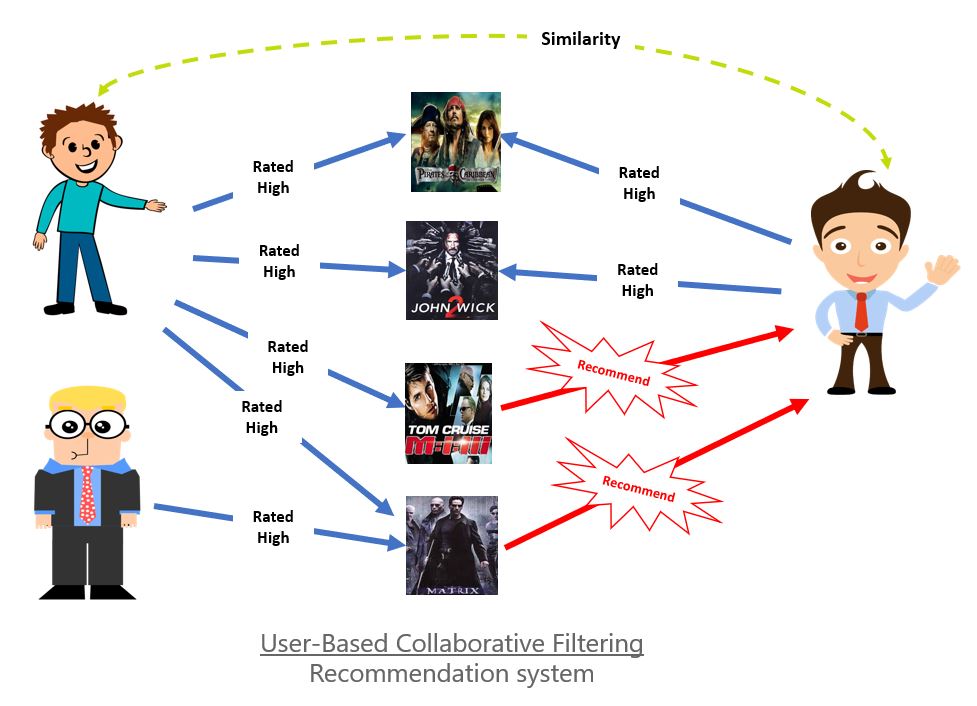
***User-Based Collaborative Filtering*:**

User-Based Collaborative Filtering recommends items to a user based on the preferences and behaviors of users with similar tastes. The algorithm identifies users who have made similar choices in the past, predicting that they will likely agree on additional selections. For instance, if User A and User B share similar movie preferences, the system may suggest a movie to User A based on the choices of User B.



***Item-Based Collaborative Filtering*:**

Item-Based Collaborative Filtering recommends items to a user by analyzing the characteristics and relationships between items. Instead of focusing on similar users, this approach identifies items that are similar to those the user has interacted with previously. For example, if a user has liked a certain movie, the algorithm recommends other movies with similar attributes, irrespective of whether other users have liked them.



**Collaborative Filtering Approaches:**

**Memory-Based vs.**

**Model-Based**

Within collaborative filtering, two fundamental approaches—Memory-Based and Model-Based—shape the foundation of recommendation systems.

***Memory-Based Collaborative Filtering:***

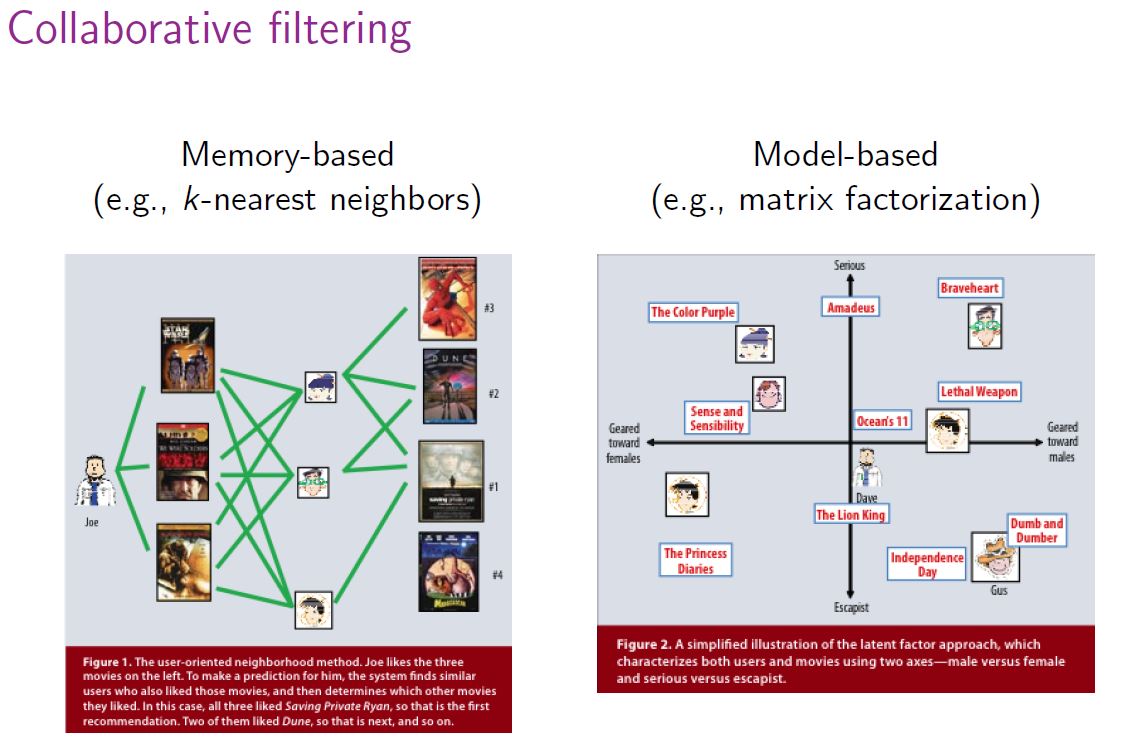
Memory-Based Collaborative Filtering relies directly on user-item interaction data for recommendations. It comprises two main strategies: User-Based Collaborative Filtering, which suggests items based on the preferences of users with similar tastes, and Item-Based Collaborative Filtering, which recommends items by analyzing the relationships between items. While Memory-Based methods are conceptually simple and intuitive, they can face challenges with computational efficiency and scalability, particularly when dealing with large and sparse datasets.

***Model-Based Collaborative Filtering:***

On the other hand, Model-Based Collaborative Filtering involves building predictive models from user-item interaction data. Techniques such as matrix factorization and latent factor models fall under this category. Unlike Memory-Based approaches, Model-Based methods create models during a training phase, enhancing computational efficiency and scalability.

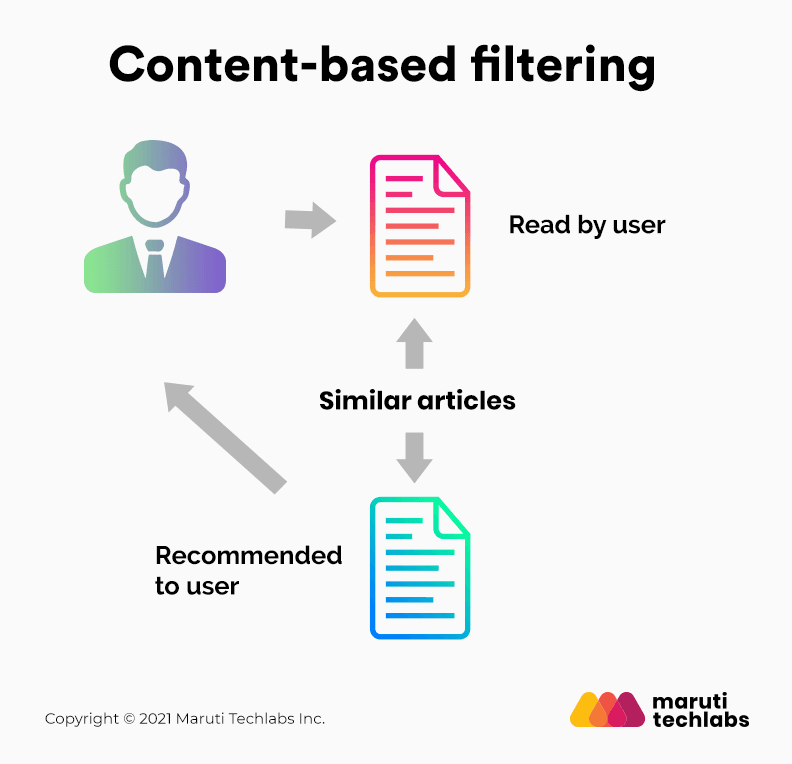
They often outperform Memory-Based methods in accuracy, especially in scenarios involving complex relationships or sparse datasets.

In summary, the choice between Memory-Based and Model-Based Collaborative Filtering depends on factors like computational efficiency, scalability, and the characteristics of the available data. While Memory-Based methods offer simplicity, Model-Based methods provide improved scalability and accuracy, making them suitable for handling more extensive and intricate recommendation scenarios.



* **Content filtering** :

Content filtering, in stark contrast to collaborative filtering, adopts an approach that relies on the intrinsic attributes or features of an item to generate recommendations aligned with a user's preferences. This method, often referred to as the "content" part, leverages information about a user and their interactions with items. Key attributes such as a user's age, the cuisine category of a restaurant, or the average review score for a movie are considered. By establishing similarities between these item and user features, content filtering models the likelihood of a new interaction, predicting items that may resonate with the user based on their past preferences. This personalized recommendation strategy is particularly effective in scenarios where user-item interactions and attributes play a pivotal role, offering a tailored and relevant user experience.



**Which approaches were chosen for our project?**

* ***Content-based filtering :***

Recommending books similar to a given book by analyzing the content and characteristics of the books (title, author, and category-name).

* ***Collaborative filtering approach (user-based CF + item-based):***

Suggesting books that similar users have rated before. The algorithm identifies users who have similar history and characteristics and recommends other books rated by those users. This approach leverages the collective preferences of users to make personalized recommendations.

**Dataset’s Information:**

* **Dataset N°1:** [**https://www.kaggle.com/datasets/saurabhbagchi/books-dataset**](https://www.kaggle.com/datasets/saurabhbagchi/books-dataset)**:** 
  + - **It contains 3 CSV files :**

1. Books.csv: contains the books and their info ***without GENRE***.
2. Users.csv: contains the users and their info.
3. Ratings.csv: contains the ratings given by users for books.

* **Dataset N°2:**

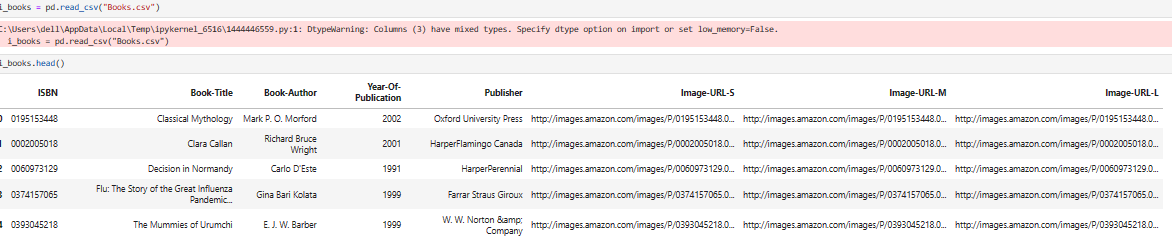
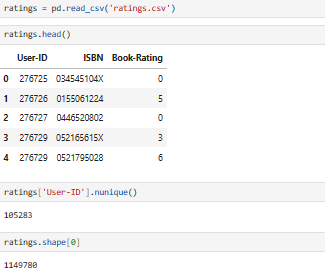
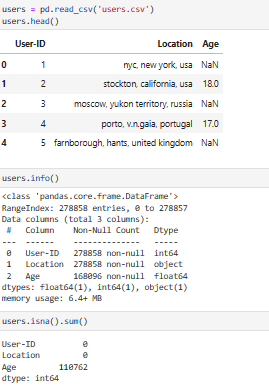
[**https://app.gigasheet.com/spreadsheet/amazon-kindle-books-dataset-2023-130k-books/11d599ca\_4dfe\_4ff6\_81ef\_a271242a8e5d?referrerId=https%3A%2F%2Fwww.gigasheet.com%2Fsample-data%2Famazon-kindle-books-dataset-2023-130k-books**](%20https:/app.gigasheet.com/spreadsheet/amazon-kindle-books-dataset-2023-130k-books/11d599ca_4dfe_4ff6_81ef_a271242a8e5d?referrerId=https%3A%2F%2Fwww.gigasheet.com%2Fsample-data%2Famazon-kindle-books-dataset-2023-130k-books)**:**

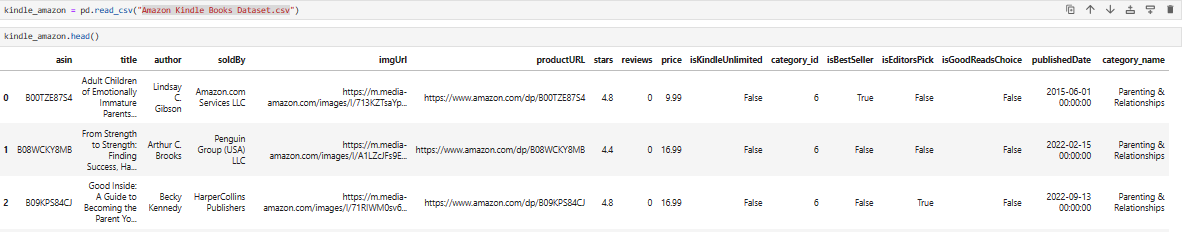
* + - **It contains 1 CSV file :**

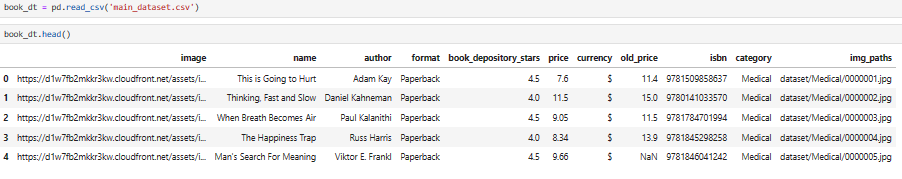
1. Amazon Kindle Books Dataset.csv: contains the books and their info ***with category***.

* **Dataset N°3:**
  + - **It contains 1 CSV file :**

1. Main-dataset.csv: contains the books and their ***info with Category***.



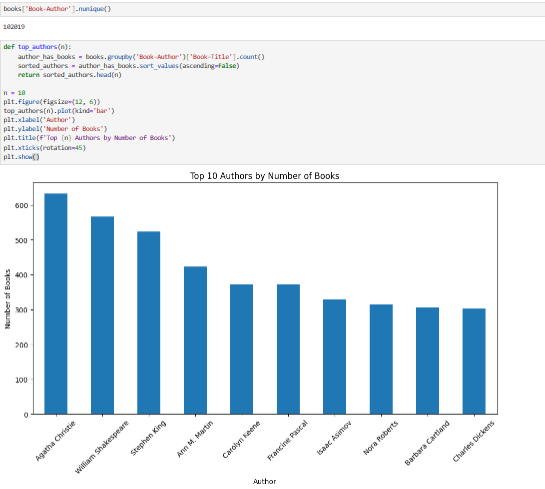


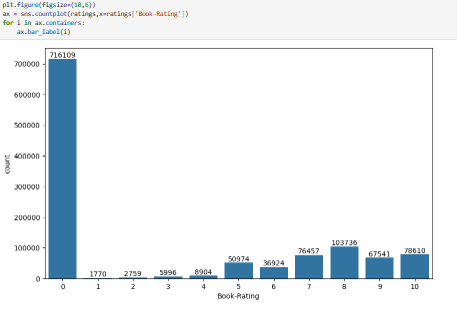


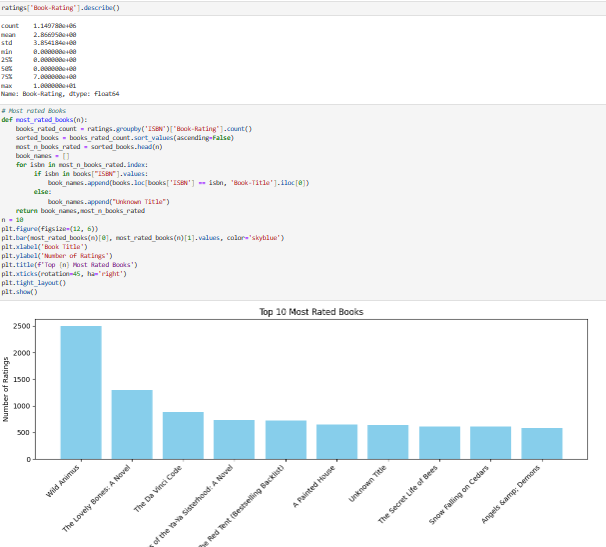
**Project’s Steps:**

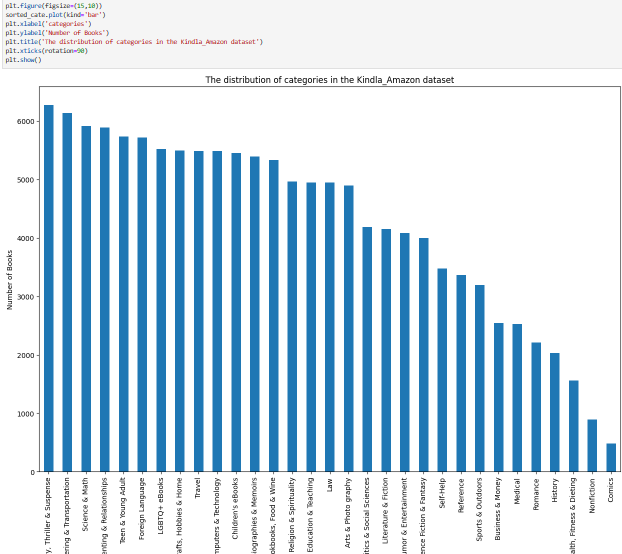
1. Dataset Visualization :

* Using Matplotlib library to visualize our datasets.



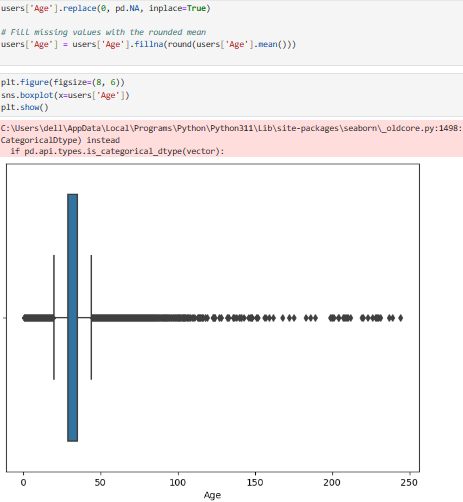






1. Preprocessing :
   * + EDA Analysis :

* Handling Missing Values.
* Handling Outliers.
* Handling Duplicate Rows.
* Dropping Useless columns.





* + - Extra operations:
* Renaming Columns.
* Changing the categories name so they match between the datasets.
* Merging 2 categories together (few samples in a category so we merge it with another one that have approximately similar concept.
* **Merging the 2 datasets (main.csv and kindle\_amazon.csv).**
  + - Handling Categorical data(Vectorizing):
* ***CountVectorizer:*** It is a text preprocessing technique widely employed in natural language processing and machine learning applications. Implemented through the scikit-learn library in Python, CountVectorizer facilitates the conversion of a collection of text documents into a matrix of token counts. The process involves creating an instance of the CountVectorizer class, where parameters such as **max\_features** can be specified to limit the number of considered features. The fit\_transform method is then utilized to both learn the vocabulary from the provided text data and transform the text into a matrix where each row corresponds to a document, and each column represents a unique word. The resulting matrix, often used as input for various machine learning algorithms, reveals the frequency of each word in the respective documents.
* ***TF-IDF***: An acronym for Term *Frequency-Inverse Document Frequency*, stands as a pivotal text preprocessing technique in natural language processing and information retrieval. Unlike CountVectorizer, TF-IDF takes into account the significance of terms in a document within the context of an entire corpus. It assigns higher weights to terms that are frequent in a specific document but relatively rare across the entire collection. This uniqueness factor helps in identifying key terms that distinguish documents from one another. The TF-IDF algorithm involves computing the product of the term frequency (TF) and the inverse document frequency (IDF) for each term in the corpus.
* ***TF-IDF with n-grams parameter*** provides a powerful and flexible approach to text representation in natural language processing tasks. TF-IDF is effective in capturing the importance of terms, while n-grams maintain the sequential context of words. By incorporating n-grams into the TF-IDF framework, we create a feature set that not only considers the significance of individual terms but also accounts for the contextual relationships between them. This hybrid approach is particularly beneficial in tasks where the order of words is crucial, such as sentiment analysis or language modeling.
  + - Implementing classification models to predict the category-name for the targeted dataset(Books.csv):
* ***Naive Bayes:*** It stands as a robust and widely employed algorithm for category classification, particularly in the domain of natural language processing. This probabilistic classifier is adept at assigning documents to predefined categories based on the observed frequency of words or features. Despite its "naive" assumption of feature independence, Naive Bayes delivers efficient and accurate results in various text classification tasks. In the context of category classification, the algorithm undergoes a training phase on labeled data to learn the probability distribution of words within each category. During classification, it then estimates the likelihood of a document belonging to a particular category, making it invaluable for tasks such as topic categorization, spam detection, and sentiment analysis.
* ***Logistic Regression:*** In the context of category classification, Logistic Regression models the relationship between the input features (such as word frequencies in a document) and the probability of a document belonging to a specific category. Through a training process, the algorithm learns the optimal coefficients that best define this relationship for each category. During classification, Logistic Regression calculates the probability distribution across all categories and assigns the document to the one with the highest probability.
* ***LBFGS solver:***  (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) *solver employed in logistic regression* plays a pivotal role in efficiently optimizing model parameters. This numerical optimization algorithm is particularly adept at navigating the parameter space to find the optimal coefficients for the logistic regression model. Notably, LBFGS is well-suited for large-scale problems, offering a practical solution when the full Hessian matrix becomes computationally challenging to compute or store. Its quasi-Newton approach iteratively refines the parameter estimates, striking a balance between computational efficiency and memory limitations.
  + - Varying the splitting techniques :
* ***train\_test\_split:*** It serves as a pivotal step in the development and evaluation of machine learning models. It involves partitioning a dataset into two subsets: a training set used for training the model and a testing set employed for assessing its performance. The Test-Train Split helps in gauging the model's ability to generalize to new, unseen data.
* ***Stratified K-Fold Cross-Validation:*** It emerges as a vital strategy for model evaluation, especially when confronted with imbalanced class distributions. This technique partitions the dataset into K folds, maintaining the relative distribution of each class across all folds. During the iterative process, each fold serves alternately as both the training and testing set. This ensures that the model encounters diverse instances of each class, enhancing its ability to generalize to various scenarios. Stratified K-Fold Cross-Validation is particularly useful when imbalances exist in the target classes, as it provides a more accurate reflection of the model's performance across the entire dataset.
* Dimensionality reduction:
* ***Singular Value Decomposition (SVD):*** It is a powerful technique for dimensionality reduction in machine learning. It is particularly useful when working with large datasets and aiming to capture the most significant patterns and features in the data. Apply SVD to the centered data matrix *X*: *X*=*U*Σ*VT :*

*U* is an *m*×*m* matrix containing the left singular vectors.

Σ is an *m*×*n* diagonal matrix containing singular values.

*VT* is *n*×*n* matrix containing the right singular vectors.

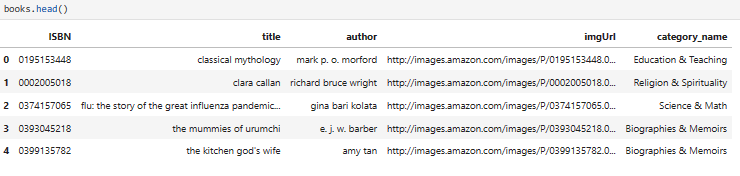
Retain only the top *k* singular values and their corresponding

columns in *U* and *V*.

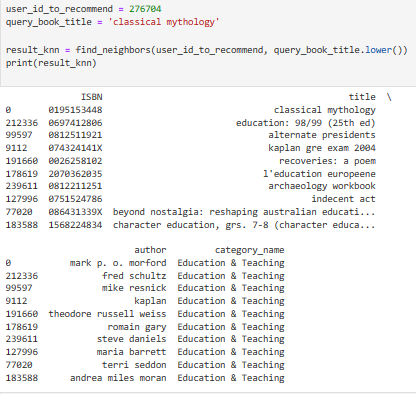
This effectively reduces the dimensions from *n* to *k*, where *k* is

the desired reduced dimensionality.

* + - The classification Metrics to evaluate the models:
* ***F1 Score:*** It is a vital metric in model evaluation, especially in situations where class imbalances pose challenges. Calculated as the harmonic mean of precision and recall, the F1 Score offers a balanced assessment of a model's performance. This makes it particularly useful when the distribution of classes is uneven, providing a single numerical value that encapsulates both the precision (the accuracy of positive predictions) and recall (the ability to capture all positive instances) of a classification model. Its ability to consider both false positives and false negatives makes the F1 Score a robust measure in scenarios where achieving a balance between precision and recall is critical.
* ***Accuracy:*** It stands as a fundamental metric for evaluating the overall correctness of a machine learning model. It measures the proportion of correctly predicted instances among the total instances in a dataset. While accuracy is a straightforward and intuitive metric, it may not be sufficient in scenarios with imbalanced class distributions. In situations where certain classes are underrepresented, accuracy alone might not provide a comprehensive view of a model's performance. Nevertheless, accuracy remains a valuable measure for assessing general predictive correctness, particularly in cases where class imbalances are not a significant concern.
  + - The combinations Implemented :
* **CountVectorizer + train\_test\_split (size=0.2) + MNBayes**
  + - Accuracy on the training set : 0.6877697364052852
    - Accuracy on the test set: 0.5421950733683022
    - F1-score: 0.52
* **TFIDF + train\_test\_split (size=0.2) + MNBayes**
  + - Exactitude sur l'ensemble de test : 0.6596673527654207
    - F1-score sur l'ensemble de test : 0.6384298006226967
    - Exactitude sur l'ensemble de test : 0.5286833543589403
    - F1-score sur l'ensemble de test : 0.5002624198031068
* **TFIDF + STKFOLD(n=10) + MNBayes**
  + - Average accuracy on the training set: 0.657801237928719
    - Average accuracy on the test set: 0.5331120111546379
    - Average F1 score on the training set: 0.637349619155015
    - Average F1 score on the test set: 0.5059537666383706
* **TFIDF(ngram\_range = (1,3)) + STKFOLD(n=10) + MNBayes**
  + - Average accuracy on the training set: 0.849370891707057
    - Average accuracy on the test set: 0.5411526459066464
    - Average F1 score on the training set: 0.839205125159316
    - Average F1 score on the test set: 0.5100842449463473
* **Logistic Regression (solver='lbfgs') + CountVectorizer + train\_test\_split(0.2):**
  + - Number of iterations: [50]
    - Accuracy on the training set: 0.774209879822057
    - Accuracy on the test set: 0.5488347387291681
    - Average F1 score on the training set: 0.773426566954482
    - Average F1 score on the test set: 0.5462313441717696
* **Logistic Regression (solver='lbfgs') + TFIDF + train\_test\_split(0.2):**
  + - Accuracy on the training set: 0.6457738529978089
    - Accuracy on the test set: 0.539140827302304
    - Average F1 score on the training set: 0.642308612003171
    - Average F1 score on the test set: 0.5342479542830446
* **Logistic Regression (solver='lbfgs') + TFIDF + SKFOLD:**
  + - Average accuracy on the training set: 0.632570878427727
    - Average accuracy on the test set: 0.534539539207224
    - Average F1 score on the training set: 0.734112836638569
    - Average F1 score on the test set: 0.5197391349216004
* The chosen Vectorizer **: CountVectorizer**
* The chosen splitting technique : **SKFOLD**
* The chosen model (LR/MNB) **: LOGISTIC REGRESSION**
  + - * Best Accuracy with Logistic Regression: 0.7466967664829692
      * Average Accuracy with Logistic Regression: 0.5542659849943562
      * Best Accuracy with Naive Bayes: 0.5529513312529049
      * Average Accuracy with Naive Bayes: 0.5466702078215258
    - ***Final step :***
* ***Using the technique (CV + SKFOLD + LR) for predicting categories and apply modifications on the (Books.csv) dataset then save it to work with it.***

******

**Recommender System Task:**

* ***Content-Based Filtering:*** The idea is to recommend books similar to a given book that a user clicked on it (in the webapp). The similarity between books is based on the book’s title, author and category. We’ve implemented a supervised and unsupervised models :
  + - * + ***The models used:***
    - ***KNN classifier:***

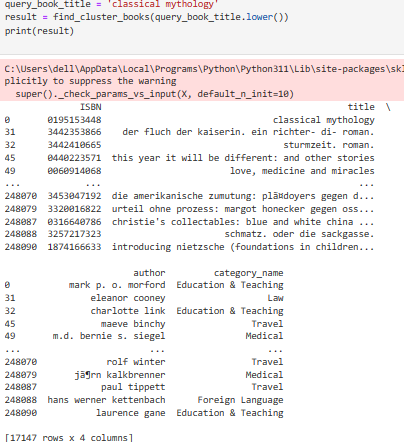
Represent each book ***(title+author+category)*** as a vector in a high-dimensional space using a suitable text representation technique like TF-IDF or CountVectorizer. Each dimension in the vector corresponds to a unique term.

Using SVD to reduce the ***Books.csv*** dimensionality.

Creating a KNN model that computes distances between vectors and specifying metric='cosine' to instruct the algorithm to use cosine similarity when computing distances: Compute the cosine similarity between all pairs of book vectors. Cosine similarity measures the cosine of the angle between two vectors and provides a value between -1 (completely dissimilar) and 1 (identical).

When a new book is queried for recommendations, represent it as a vector using the same CountVectorizer representation and use the KNN model to find the k-nearest neighbors of the queried book vector.

Books with closer vectors (higher cosine similarity) are considered more similar and are likely to be recommended, such that these books shouldn’t be rated by this user.



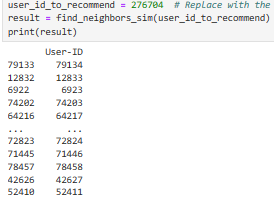
* + - ***Kmeans clustering model:***
* Using same preprocessing techniques implemented with KNN classifier.
* Utilize the K-Means clustering algorithm to group books into clusters based on their feature vectors. The number of clusters (k) was predefined based on the **number of unique categories present in the dataset (33).**
* Assign each book to a specific cluster based on **the cluster centroids** calculated by the K-Means algorithm. Books within the same cluster are considered more similar to each other.
* When a new book is queried for recommendations, represent it as a vector using the same text representation technique (e.g., CountVectorizer). Assign this book to the nearest cluster based on the centroids obtained from the K-Means model. Books in the same cluster are assumed to share similar features and thematic content.

***REMARK:***

* This technique was just an extra task that we want to add to our project since the models implemented were not included in the project‘s models choices.
* The model that have been deployed in the webapp is ***KNN*** , this is due to facility of implementing it (defining the parameter K) comparing to

***Kmeans*** which is a bit challenging to define the parameter (n\_clusters).

* ***User-Based collaborative filtering:*** The idea is to find a similar users to a given user, then predicting to that user the most frequent items (books) read by those similar users. We’ve combined two models to accomplish this task:
  + - ***KNN classifier:*** It was used to find similar users :



* Making a new column ***“History”*** that contains the unique categories of the books read by each user***.***
* Represent each user (age + location+ history) as a vector in a high-dimensional space.

Employing a text representation technique CountVectorizer, where each dimension corresponds to a unique term.

* Utilizing (SVD) to reduce the dimensionality of user vectors.
* Establishing a (KNN) model designed to compute distances between user vectors. Specifying the metric as 'cosine' to leverage cosine similarity when measuring distances between users.
* Users with closer vectors (higher cosine similarity) are deemed more similar in terms of their preferences. Providing recommendations by suggesting similar users based on the neighbors identified through the KNN model.
* ***REMARK*:** the similar users returned by the KNN classifier were grouped in a dataframe where each user have all the books that he rated (positively >5), this dataframe will be the input to our unsupervised model so it’ll be trained on it (basket of books).

***Association rules:*** The idea is to sift through massive volumes of data and find intriguing relationships between features. There are 3 main algorithms:

1. **Apriori:**  This algorithm detects the most frequent itemsets or elements in a transaction database and establishes association rules between the items. The method employs a “bottom-up” strategy, in which frequent subsets are expanded one item at a time (candidate generation), and groups of candidates are checked against the data. When no more successful rules can be obtained from the data, the algorithm stops. The algorithm examines three crucial aspects while constructing association rules between components or items: support, confidence, and lift. Each of these elements is discussed below:
2. **Association rule:** For example, X Y is a depiction of discovering Y on a basket that contains X.
3. **Itemset:** For example, X, Y is a representation of the list of all objects that comprise the association rule.
4. **Support:** Transactions containing the itemset as a percentage of total transactions
5. **Confidence:** Given X, what is the likelihood of Y occurring?
6. **Lift:** Confidence ratio to baseline likelihood of occurrence of Y

*“Among association rule learning algorithms, this is the simplest and most straightforward algorithm. The resulting rules are simple to understand and express to the end-user.”*

* The two major shortcomings of the Apriori algorithm are:
* The size of candidate itemsets could be extremely large
* High costs on counting support since we have to scan the itemset database over and over again.
* To overcome these challenges, the biggest breakthrough of FP Growth is that No candidate generation is required!

1. **FP Growth**: FP tree is the core concept of the whole FP Growth algorithm. Briefly speaking, the FP tree is the **compressed representation** of the itemset database. The tree structure not only reserves the itemset in DB but also keeps track of the association between itemsets.

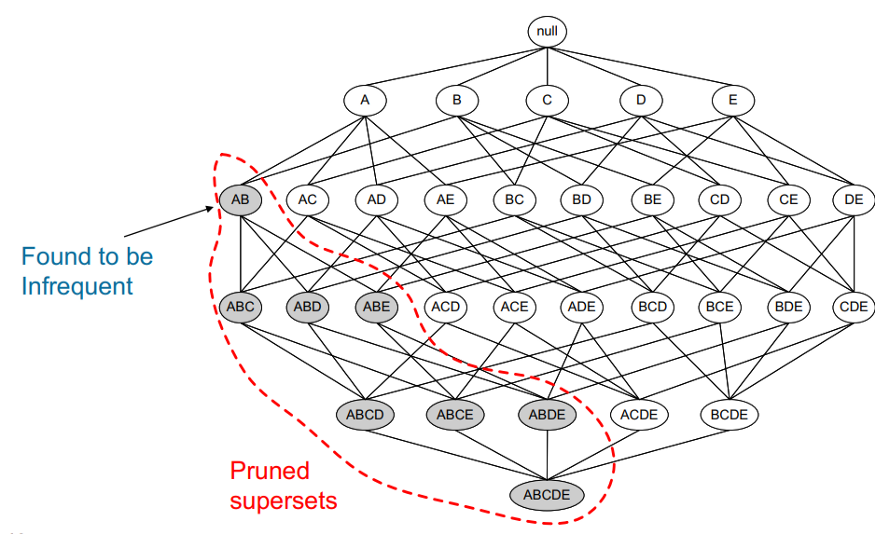
* The tree is constructed by taking each itemset and mapping it to a path in the tree one at a time. The whole idea behind this construction is that :

More frequently occurring items will have better chances of sharing items.

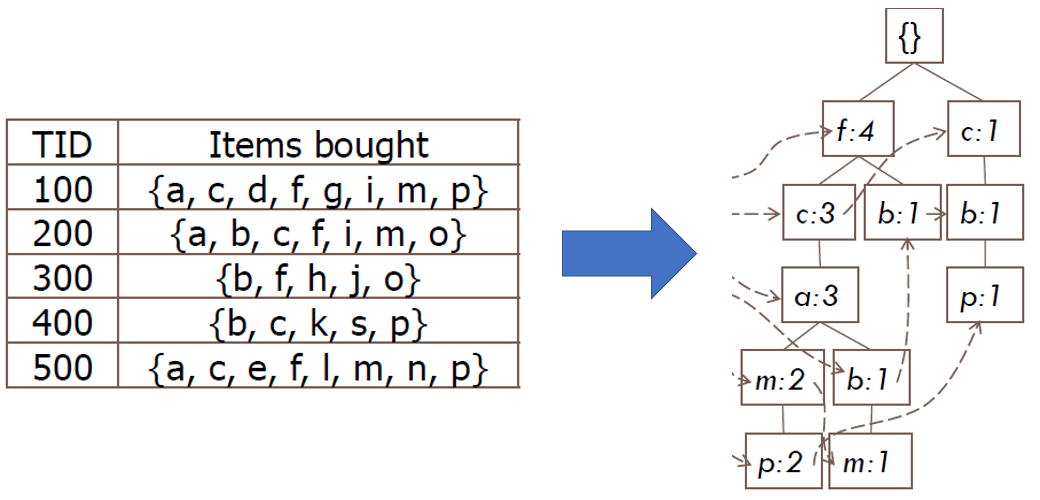
* We then mine the tree recursively to get the frequent pattern. Pattern growth, the name of the algorithm, is achieved by concatenating the frequent pattern generated from the conditional FP trees.

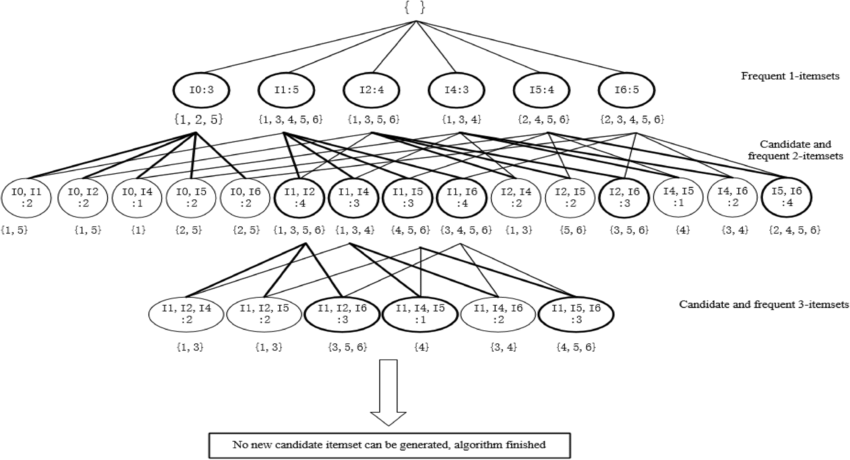
1. **Eclat algorithm:** The ECLAT algorithm stands for **Equivalence Class Clustering and bottom-up Lattice Traversal**. It is one of the popular methods of [Association Rule mining](https://en.wikipedia.org/wiki/Association_rule_learning). It is a more efficient and scalable version of the Apriori algorithm. While the Apriori algorithm works in a horizontal sense imitating the Breadth-First Search of a graph, the ECLAT algorithm works in a vertical manner just like the Depth-First Search of a graph. This vertical approach of the ECLAT algorithm makes it a faster algorithm than the Apriori algorithm.

* **How the algorithm work?:**  
  The basic idea is to use Transaction Id Sets(tidsets) intersections to compute the support value of a candidate and avoiding the generation of subsets which do not exist in the prefix tree. In the first call of the function, all single items are used along with their tidsets. Then the function is called recursively and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs. This process is continued until no candidate item-tidset pairs can be combined.
* **Advantages over Apriori algorithm:**
* **Memory Requirements:** Since the ECLAT algorithm uses a Depth-First Search approach, it uses less memory than Apriori algorithm.
* **Speed:** The ECLAT algorithm is typically faster than the Apriori algorithm.
* **Number of Computations:** The ECLAT algorithm does not involve the repeated scanning of the data to compute the individual support values.



**APRIORI ALGORITHM**

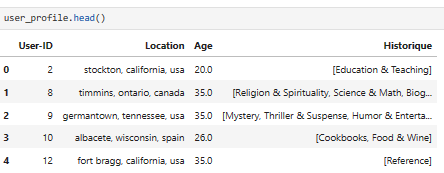
**FPGROWTH ALGORITHM**



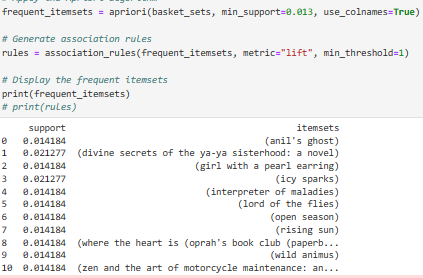
**ECLAT ALGORITHM**

**How these algorithms were implemented in our project?**

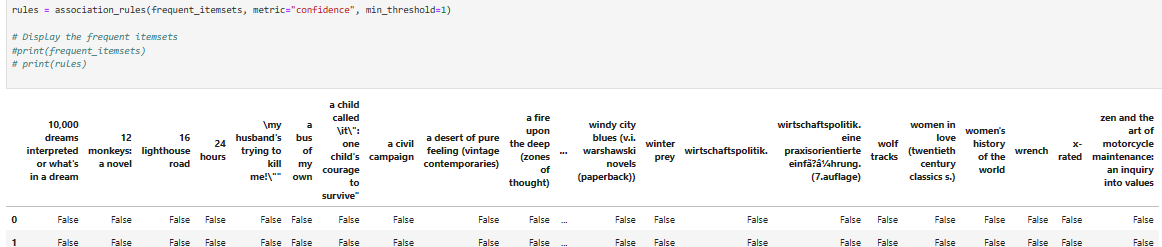
* **Input:** The dataframe created in the step before (from each user get the set of books that he read).By this we make a set of lists to be analyzed like a ***Basket***.
* **Preprocessing**: Convert the data to a suitable format for each algorithm:
* ***TransactionEncoder:*** The **TransactionEncoder** in the **mlxtend** library is responsible for converting a transaction dataset into a one-hot encoded format, which is a requirement for association rule mining algorithms like **Apriori.** This encoder transforms a list of lists or an array of transactions into a binary matrix, where each row corresponds to a transaction, and each column represents a unique item in the entire dataset.
* ***Get\_dummies:*** **get\_dummies** is a function in the pandas library in Python that is used for one-hot encoding categorical variables. One-hot encoding is a process of converting categorical variables into a binary matrix, also known as a one-hot encoded matrix. This is a common preprocessing step in machine learning when dealing with categorical data. It was used with **FPgrowth.**
* **Eclat\_instance.df\_bin: df\_bin** generates a binary dataframe, that can be used for other analyzes (***specified for Eclat algorithm).***
* ***Fine tuning parameters:***
* **min\_support**: The minimum support threshold, specifying the minimum fraction of transactions an itemset must be present in to be considered frequent. It is a value between 0 and 1.
* **Metric**: The metric used to evaluate the generated association rules. In our case, "lift" is used, which measures the ratio of the observed support to that expected if the two rules were independent. Lift values greater than 1 indicate a positive association.



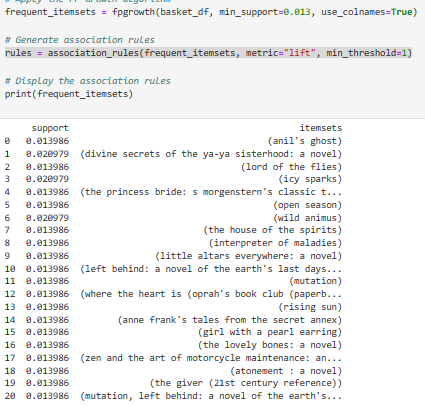
* ***The input dataframe***



* **The result by Apriori**



* **Result of Transaction Encoder preprocessing**



* **the result of FPgrowth**

****

* **the input that Eclat algorithm takes**
* ***REMARK:*** The algorithm chosen to be deployed in the webapp was ***Apriori.*** Basically all the algorithms gave **the same result**!!