**MILESTONE 2**

**Objective**:

The objective of this project is to develop a recommendation system that enhances user experience by suggesting relevant products based on historical interactions with products and available product data. The system aims to leverage machine learning techniques to analyse customer behaviour and provide personalised recommendations.

The 2nd milestone focuses on implementing approaches like TF-IDF and cosine similarity approach and machine learning algorithms like XGBoost and Random Forest to provide personalized recommendations to users. The report focuses on the feature engineering, feature selection, training process and the output interpretation carried out for each approach.

**Type of tool:**

A recommendation engine will be developed based on the user behaviour, product interactions and pricing trends.

**Tech Stack**

* **Programming Language:** Python
* **Libraries & Frameworks:** Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn
* **Visualization Tools:** Matplotlib, Seaborn

**Data Collection:**

The data used is open-source datasets available on Kaggle. The data describes the user’s interactions on an ecommerce website/application, providing information regarding the time spent on the website, the products viewed/bought, price, name of the product, etc through the following attributes:

1. **event\_time** (timestamp of interaction)
2. **event\_type** (type of user interaction)
3. **product\_id** (unique identifier for products)
4. **category\_id** (unique category identifier)
5. **category\_code** (product classification)
6. **brand** (product brand)
7. **price** (product price)
8. **user\_id** (unique identifier for users)
9. **user\_session** (unique identifier for user sessions)

Three csv files are used, 2 csv files contain information related to user’s interactions with an electronic store/website and the other csv file contains information related to users’ interactions with a multi category store.

**Project Timeline**

**Milestone 1: Data Collection, Preprocessing, and EDA (Feb 5, 2025 - Feb 21, 2025)**

1. Week 1 (Feb 5 - Feb 11): Identify and acquire dataset, verify accessibility, and document dataset details.
2. Week 2 (Feb 12 - Feb 18): Handle missing values, address outliers, and apply feature scaling.
3. Week 3 (Feb 19 - Feb 21): Perform exploratory data analysis (EDA), generate visualizations, and identify patterns.

**Milestone 2: Feature Engineering, Feature Selection, and Data Modeling (Feb 21, 2025 - March 21, 2025)**

1. Week 4 (Feb 22 - Feb 28)**:** Engineer new features and encode categorical variables.
2. Week 5 (March 1 - March 7): Perform feature selection and dimensionality reduction.
3. Week 6 (March 8 - March 14): Split data into training and testing sets, train initial models.
4. Week 7 (March 15 - March 21): Tune hyperparameters and evaluate models using performance metrics.

**Milestone 3: Evaluation, Interpretation, Tool Development, and Presentation (March 24, 2025 - April 23, 2025)**

1. Week 8 (March 24 - March 30): Assess model performance and interpret results.
2. Week 9 (April 1 - April 7): Identify biases and refine models if needed.
3. Week 10 (April 8 - April 14): Develop an interactive tool (dashboard, conversational agent, or reporting system).
4. Week 11 (April 15 - April 23): Finalize report, prepare presentation, and deliver findings.

**Feature Engineering:**

Three different approaches have been used to create a recommendation system, TF-IDF vectorizer and Cosine similarity approach, XGBoost approach and the random forest approach. The process of feature engineering slightly differs for these approaches.

1. TF-IDF vectorizer and Cosine similarity:

The ‘merged\_df’ csv has the following columns:

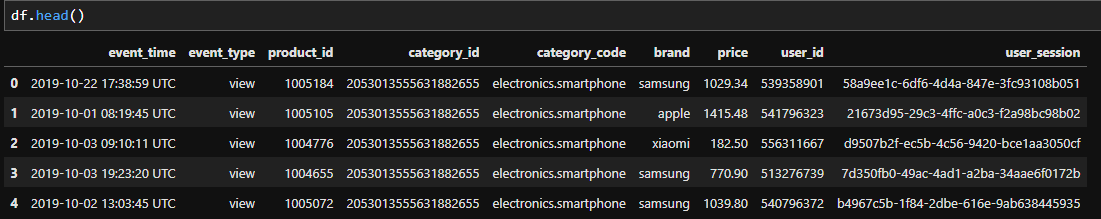


Fig 1. Visualizing the dataframe

Before building the recommendation system the following feature engineering steps were carried out:

* 1. Creating ‘main\_category’ and ‘sub\_category’ columns from ‘category\_code’ column:

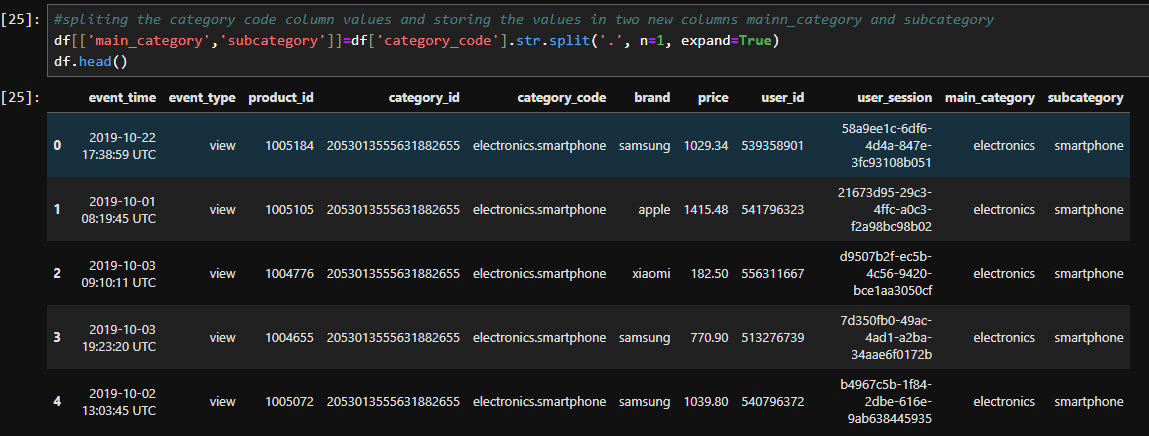


Fig 2. Dataframe after splitting the column category\_code

This step will allow to recommend more details about the products like the main\_category and subcategory (name of the product). Apart from this, using these two new columns the recommendation system will be more efficient and will be able to provide valid recommendations to the user.

* 1. Adding the ‘price\_scaled’ and ‘interaction\_score’ columns:

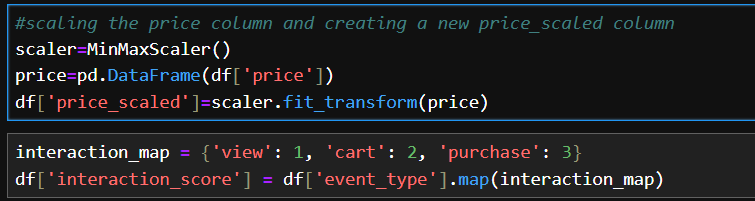


Fig 3. Scaling and encoding the numerical and categorical columns

The ‘price’ column is scaled down to have a price range from 0 to 1, to decrease the training time and to deal with the outliers in the columns, scaling down these values will help mitigate the variance introduced by the outliers.

Adding interaction\_score column quantifies user engagement, mapping event\_type to values: 1 for "view," 2 for "cart," 3 for "purchase." This numeric representation reflects interest levels, enabling Random Forest and XGBRanker models to rank products by relevance. Interaction\_score acts as target variable during training and ground truth for evaluation metrics like Precision@K and NDCG@K. Converting categorical data to continuous scale enhances personalization and accuracy, ensuring recommendations align with user behavior effectively.

* 1. Combining the categorical variables:

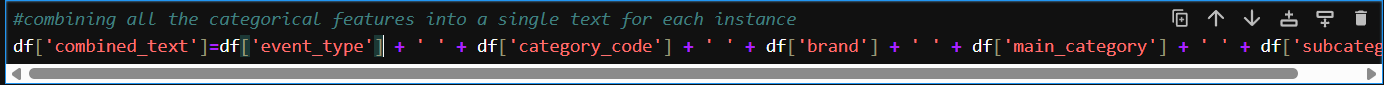


Fig 4. Combining the categorical columns

This is a crucial step as it transforms all the categorical variables into uniform strings, that enables TF-IDF vectorization for content-based recommendation. It enhances the similarity computation as the contextual relationships between event type and product attributes are captured. This feature engineering supports cosine similarity analysis, aligning product recommendations with user preferences effectively.

1. XGBoost and Random Forest:
   1. Aggregating user\_product\_stats:

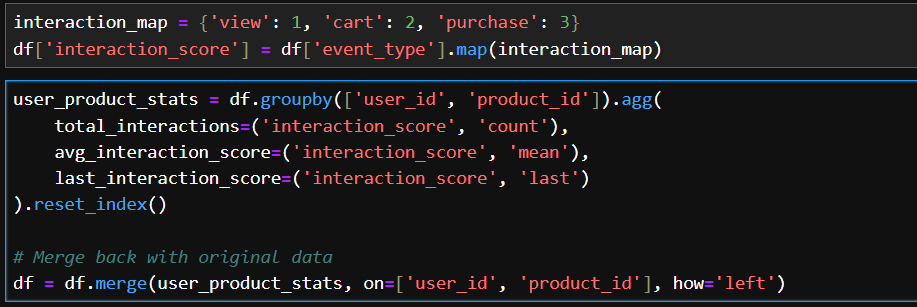


Fig 5. Feature engineering for XGBoost dataframe

This step involves grouping by user\_id and product\_id, calculating total\_interactions as count, avg\_interaction\_score as mean, and last\_interaction\_score as latest value of interaction\_score. This step captures user-product interaction patterns numerically and by combining them with behavioral features enhances the model input. Total\_interactions reflects frequency, avg\_interaction\_score indicates average engagement, and last\_interaction\_score preserves recency.

* 1. Encoding Categorical variables:

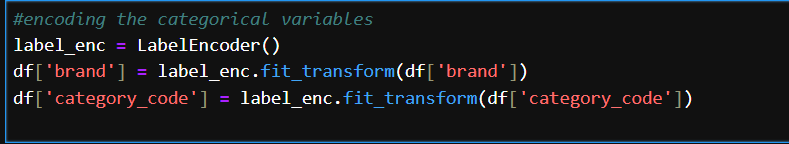


Fig 6. Encoding categorical variables

This is an important step as it involves converting the categorical variables(textual features) into numerical values, enabling machine learning models to process them effectively. The numeric format preserves categorical distinctions without assuming ordinality, facilitating feature use in Random Forest and XGBRanker.

**Feature Selection:**

**Correlation Analysis:**

Majority of the columns in the dataframe are text/categorical columns, although checking the correlation among the numerical columns is an important step while building a content based recommendation system as it measures the strength and direction of these relationships.

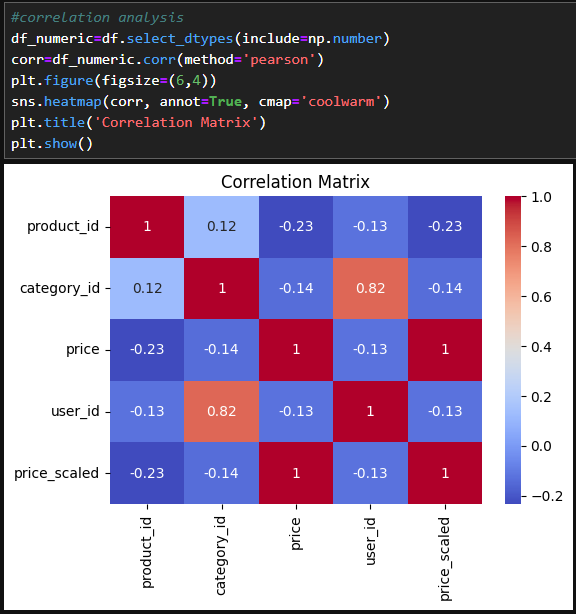


Fig 7. Heatmap for feature selection

From the matrix, we observe that category\_id and user\_id have a strong positive correlation (0.82), indicating that user preferences may align closely with specific categories. Similarly, price and price\_scaled are perfectly correlated, as expected since one is a scaled transformation of the other. Other variables, such as product\_id and price, show weak correlations (-0.23), suggesting minimal linear dependence. This analysis is crucial for identifying redundant features and understanding relationships between variables, which can guide feature selection or engineering in building a content-based recommendation system.

Feature selection process differs a bit for TF-IDF approach and the XGBoost and Random Forest approach. For TF-IDF, apart from the above-mentioned features, the column ‘combined\_text’ (mentioned in feature engineering) is considered as it is a combination of the most important features. It summarizes all the textual imformation.

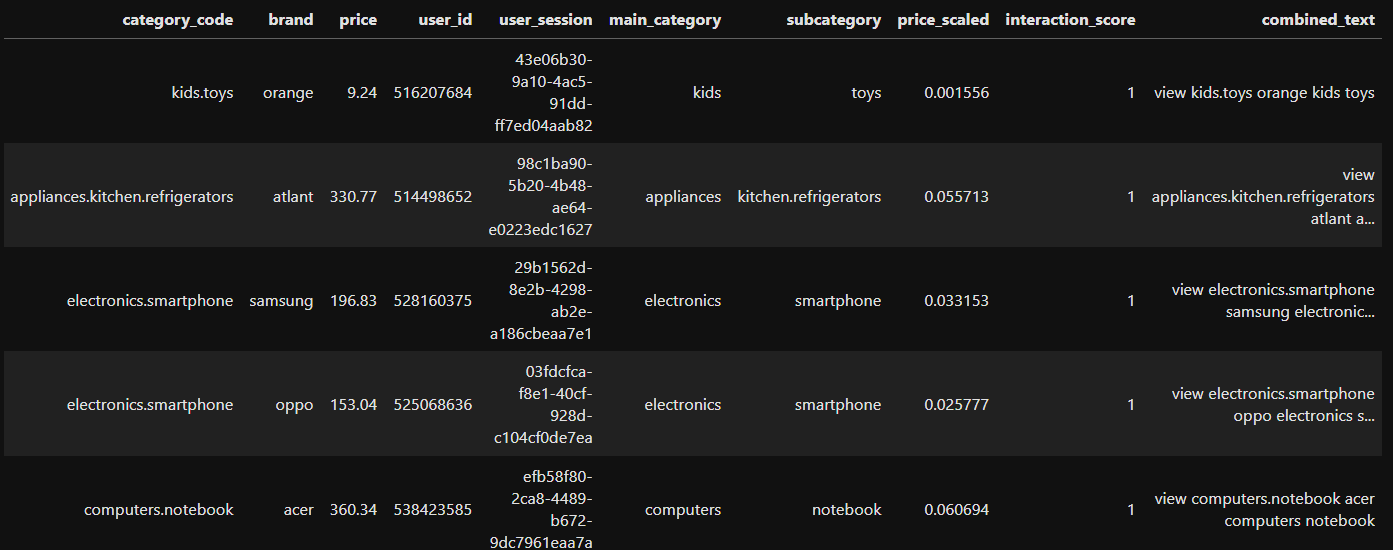


Fig 8. Displaying the ‘combined\_text’ column

For the Random Forest and XGBoost approach, it is necessary to consider the features created in the feature engineering process like the total\_interactions, avg\_interaction\_score and last\_interaction\_score.

**Data Splitting**

1. **TF-IDF**

For the TF-IDF approach there is no splitting of the dataset because of the way the products are recommended. In this approach, the cosine similarity of the products with each other is calculated and on the basis of highest cosine similarity value, the products are recommended. The greater the number of instances, more accurate the cosine similarity value and thus the recommendations are more relevant.

1. **XGBoost**

Before feeding the data to the XGBoost model, the dataset was split into training and testing sets. The train\_test\_split class was used provided by sklearn. 20% of the original dataset was reserved for testing.

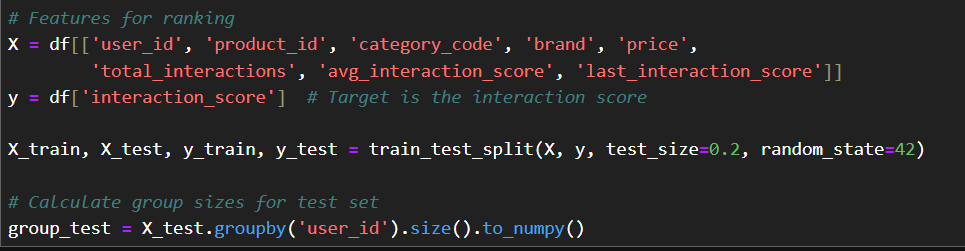


Fig 9. Splitting the dataset into train and test sets

Computing group\_test involves grouping X\_test by user\_id, calculating the size of each group, and converting results to a NumPy array. This process quantifies interactions per user in the test set, creating a group size vector. It is essential for XGBRanker’s pairwise ranking objective, group\_test ensures the model recognizes user-specific interaction counts, enabling accurate ranking of products within each user’s context during evaluation, supporting effective recommendation generation. The same process is also carried out for the training set, discussed later in the training section.

1. **Random Forest**

Before feeding the data to the Random Forest model, the dataset was split into training and testing sets. The train\_test\_split class was used provided by sklearn. 20% of the original dataset was reserved for testing.

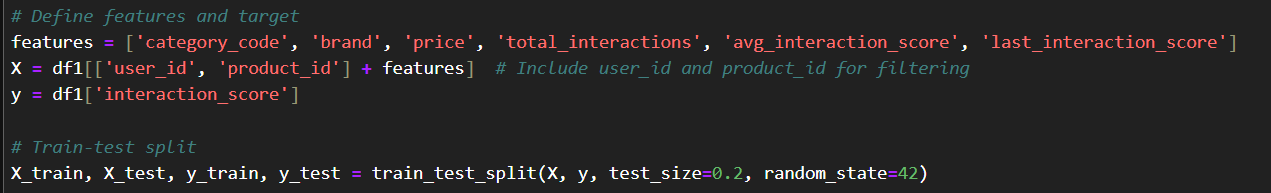


Fig 10. Splitting the dataset into train and test sets for random forest

**Model training**

1. **TF-IDF**

The process of training in this approach is a bit different from the traditional machine learning model training. Following steps were to taken to get the product recommendations for users

1. Implementing the TF-IDF vectorizer:

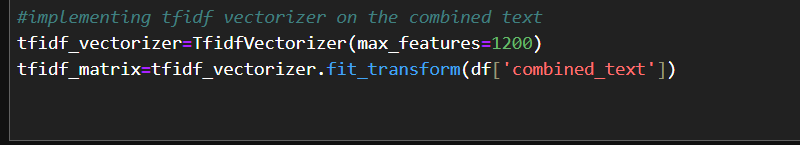


Fig 11. Vectorizing the ‘combined\_test’ column

The TfidfVectorizer is initialized with max\_features=1200 (due to computational limitations) restricts vocabulary to 1200 key terms. The fit\_transform methos is exeuted on combined\_text transforms concatenated categorical data into a TF-IDF matrix, assigning weights based on term frequency and inverse document frequency. This process generates a sparse feature representation, capturing textual importance for training. Limiting features reduces dimensionality, enhancing computational efficiency while retaining critical information for similarity-based recommendation in the TF-IDF model training phase.

1. Extracting the feature names

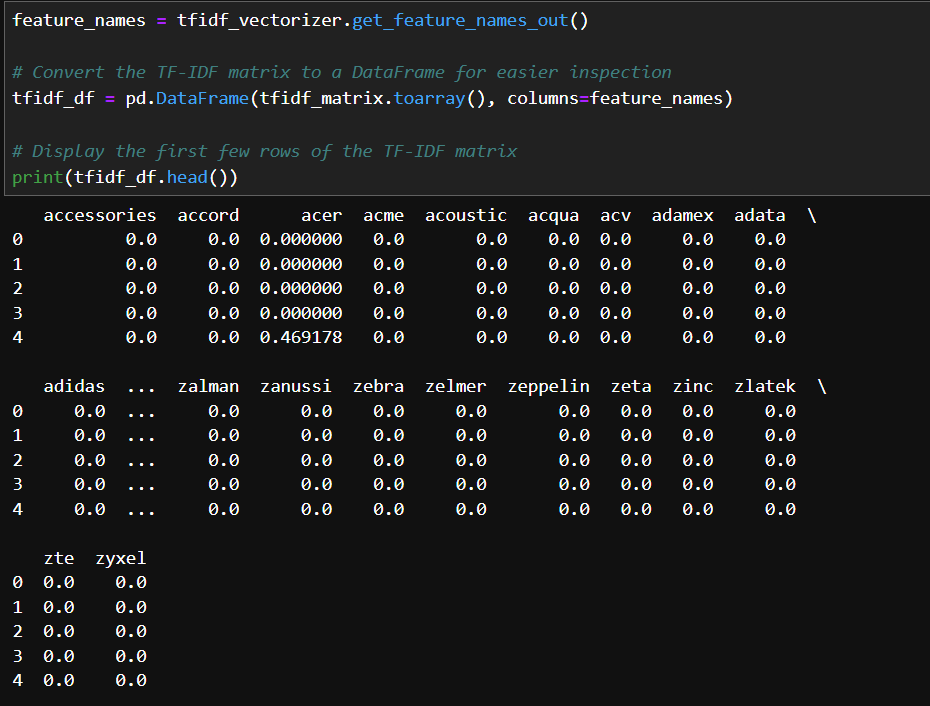


Fig 12. Displaying the feature vector

The feature\_names from tfidf\_vectorizer are extracted using get\_feature\_names\_out(). The tfidf\_matrix is then converted to a dense array and creating tfidf\_df with feature\_names as columns facilitates inspection. Displaying tfidf\_df.head() shows the first few rows, revealing TF-IDF weights for each term per instance. This step aids in understanding feature representation, verifying vectorization output before training the TF-IDF recommendation model.

1. Combining tf-idf matrix with the numerical columns

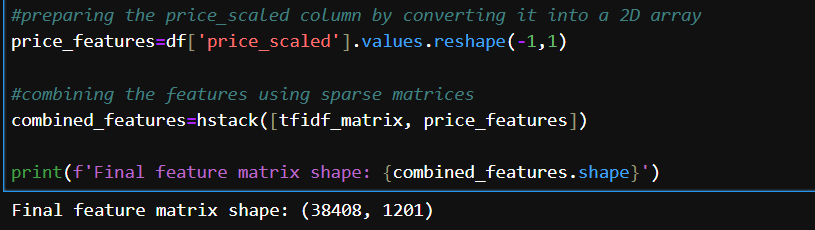


Fig 13. Stacking feature vector with price\_scaled (numerical feature)

Combining tfidf\_matrix with price\_scaled enhances feature set for TF-IDF training. Converting price\_scaled to a 2D array via reshape(-1,1) ensures compatibility. Using hstack merges sparse tfidf\_matrix with numerical price\_features, creating combined\_features. Printing combined\_features.shape confirms the resulting matrix dimensions, integrating textual and price data. This step enriches the model with both categorical and numerical insights, improving recommendation quality in the TF-IDF training phase.

1. Computing the cosine similarity

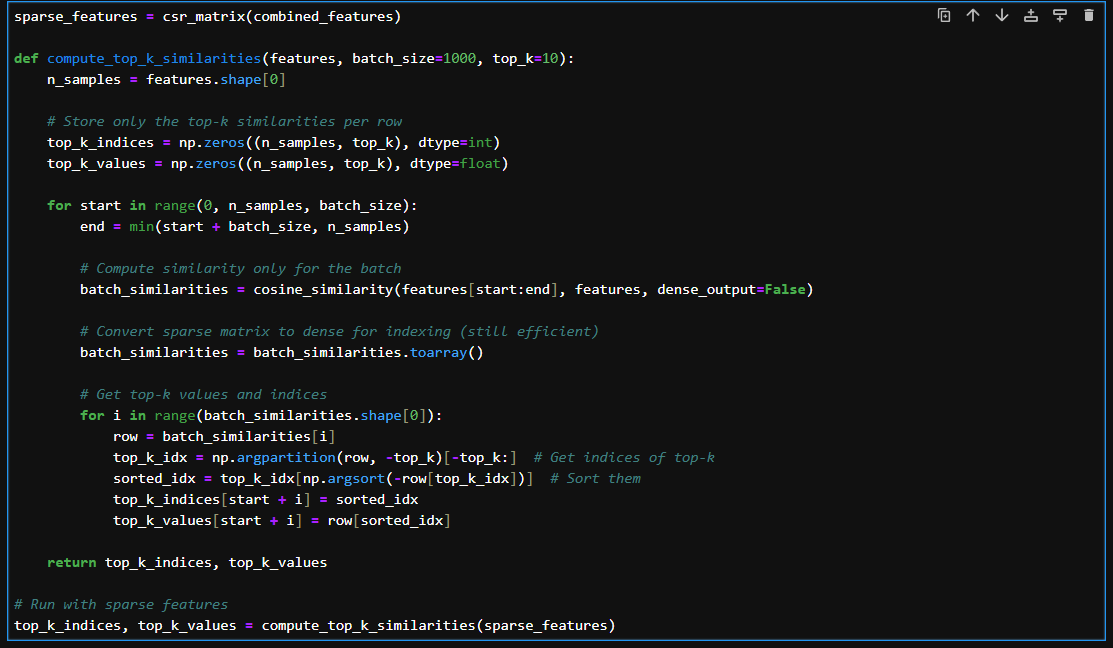


Fig 14. Computing cosine similarity

Converting combined\_features to csr\_matrix creates sparse\_features, optimizing memory usage for sparse data in our TF-IDF pipeline. We define compute\_top\_k\_similarities to process features in batches of 1000, calculating cosine similarities efficiently. The function stores the top 10 similarities per row in top\_k\_indices and top\_k\_values, leveraging argpartition and sorting for speed. Executing this with sparse\_features produces similarity rankings critical for training. This approach ensures scalability in similarity computation, enabling our TF-IDF model to identify and rank closely related products effectively for recommendation purposes.

1. User profile creation (for personal recommendation)

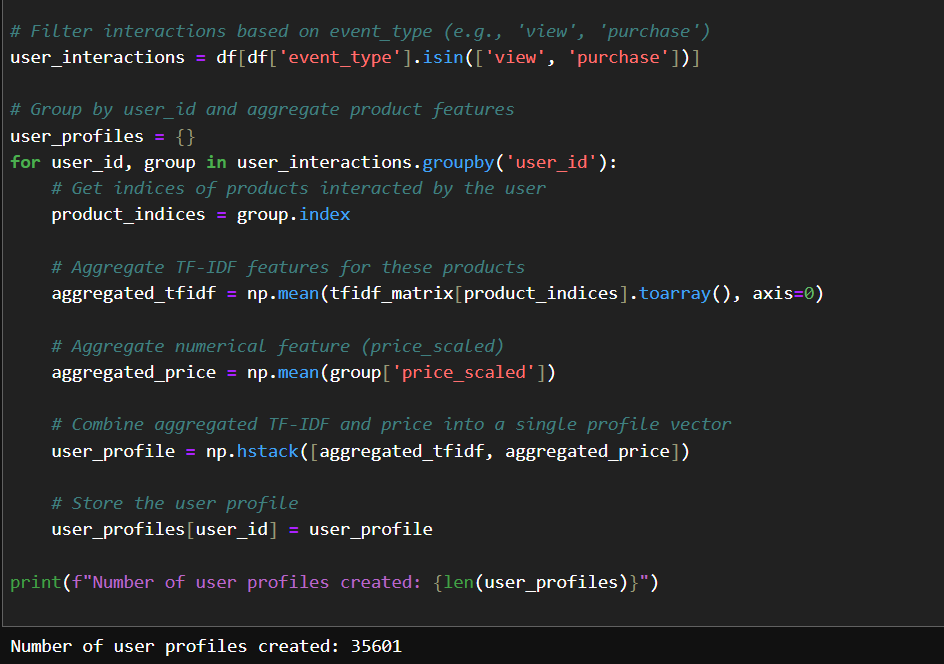


Fig 15. User profile creation

Filtering df for event\_type values "view" and "purchase" creates user\_interactions, focusing on significant user actions. Grouping by user\_id, we aggregate features into user\_profiles. For each group, extracting product\_indices allows averaging tfidf\_matrix rows into aggregated\_tfidf. Calculating aggregated\_price from price\_scaled adds numerical context. Combining these into user\_profile via np.hstack forms a unified vector per user. Storing profiles in a dictionary and printing the count tracks profile creation, supporting personalized TF-IDF recommendation training.

1. Recommending products to users



Fig 16. Recommendation algorithm for TF-IDF vector

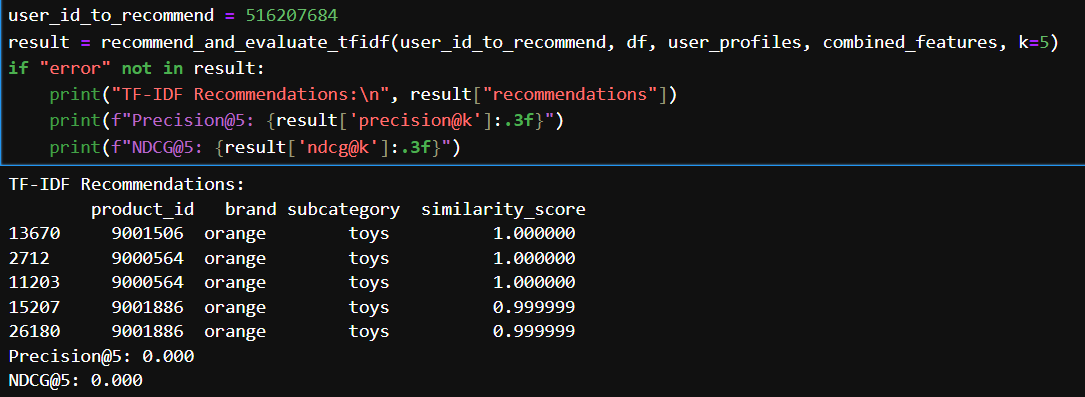


Fig 17. Displaying recommendations for a particular user

Defining recommend\_and\_evaluate\_tfidf computes TF-IDF recommendations for user 516207684 by retrieving user\_profile\_vector from user\_profiles, calculating cosine similarities with combined\_features, and selecting top K products (excluding user’s interactions). Extracting product\_id, brand, and subcategory, adding similarity\_score, and evaluating with interaction\_score yields metrics. The output lists toys from brand "orange" with high similarity scores, but Precision@5 and NDCG@5 are 0.000. This occurs because recommended products lack interaction\_score >= 2 (cart or purchase) in user\_data. If the user only viewed these products (interaction\_score = 1), binary relevance becomes 0, resulting in zero metrics, indicating poor alignment with significant user actions and highlighting recommendation challenges.

1. **XGBoost**

The XGBoost library in python provides a specific module- XGBRanker that is the most suitable for recommendation systems. The XGBRanker is a gradient boosting algorithm designed specifically for ranking tasks. In this implementation, the model will rank the products best suitable for recommendation. Using category\_code, brand, price, total\_interactions, avg\_interaction\_score, and last\_interaction\_score as features, with interaction\_score as the target, recommend\_and\_evaluate\_xgboost generates XGBRanker recommendations

1. Hyperparameter tuning

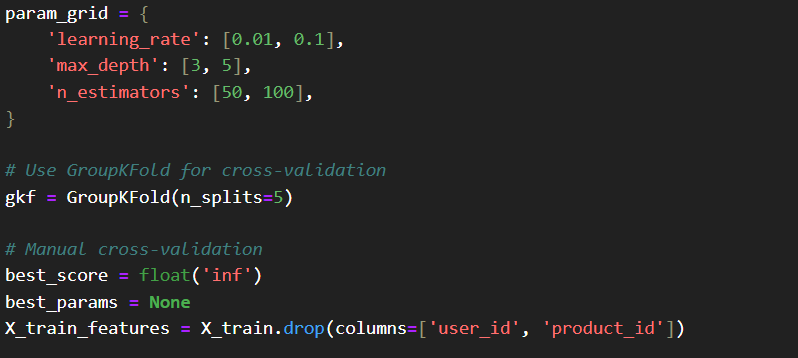


Fig 18. Hyperparameter tuning XGBoost model

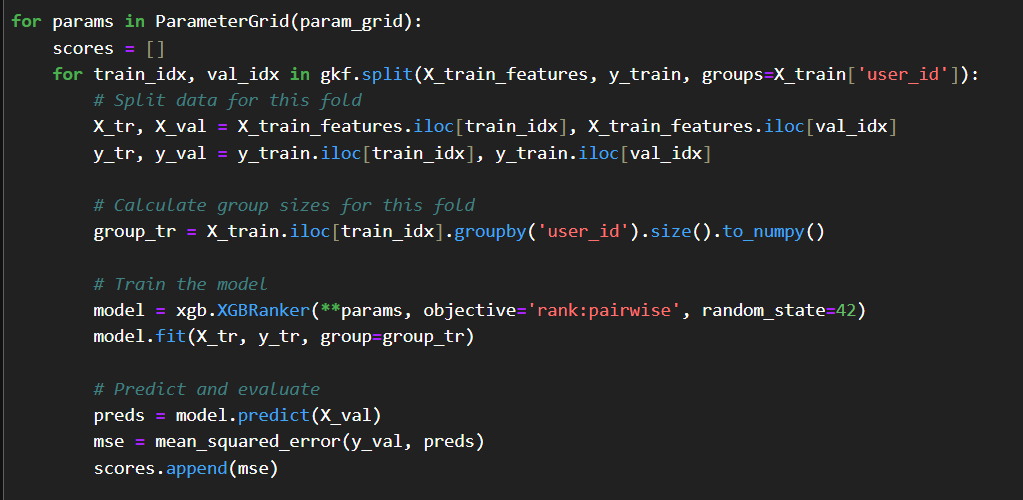


Fig 19. Iterating through the parameter grid

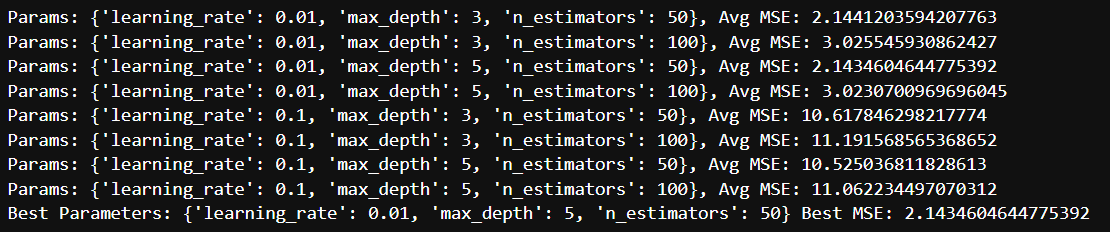


Fig 20. Displaying the best\_model

Defining param\_grid specifies hyperparameter combinations for learning\_rate, max\_depth, and n\_estimators. Using GroupKFold with 5 splits ensures user-based cross-validation. Iterating through ParameterGrid, each fold splits X\_train\_features and y\_train by user\_id groups. Calculating group\_tr sizes, training XGBRanker with rank:pairwise objective, and predicting on validation sets computes MSE scores. Averaging scores across folds identifies best\_params with the lowest best\_score. This process optimizes XGBRanker for ranking, enhancing recommendation accuracy.

1. Extracting the best model and training it on the grouped data

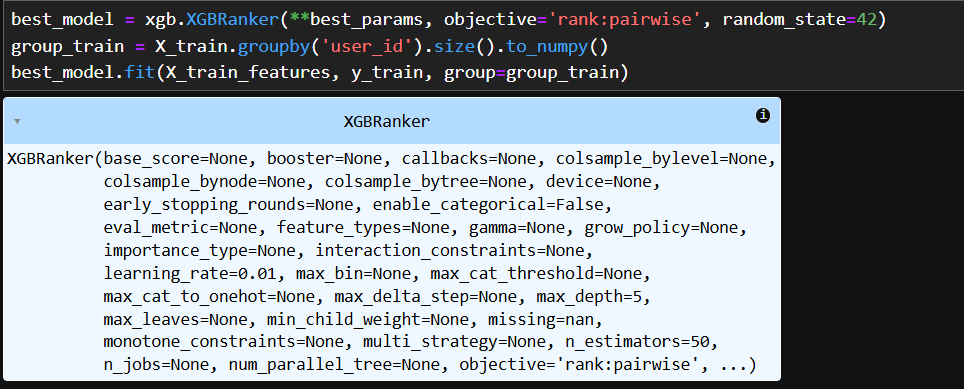


Fig 21. Training the model on grouped data

Initializing best\_model as XGBRanker with best\_params, rank:pairwise objective, and random\_state=42 sets up the optimized model. Computing group\_train by grouping X\_train by user\_id and converting sizes to a NumPy array defines user group sizes. Fitting best\_model on X\_train\_features and y\_train with group\_train trains the model for ranking tasks. This step finalizes XGBRanker training, leveraging user-specific grouping to enhance product recommendation accuracy.

1. Recommending products using XGBoost

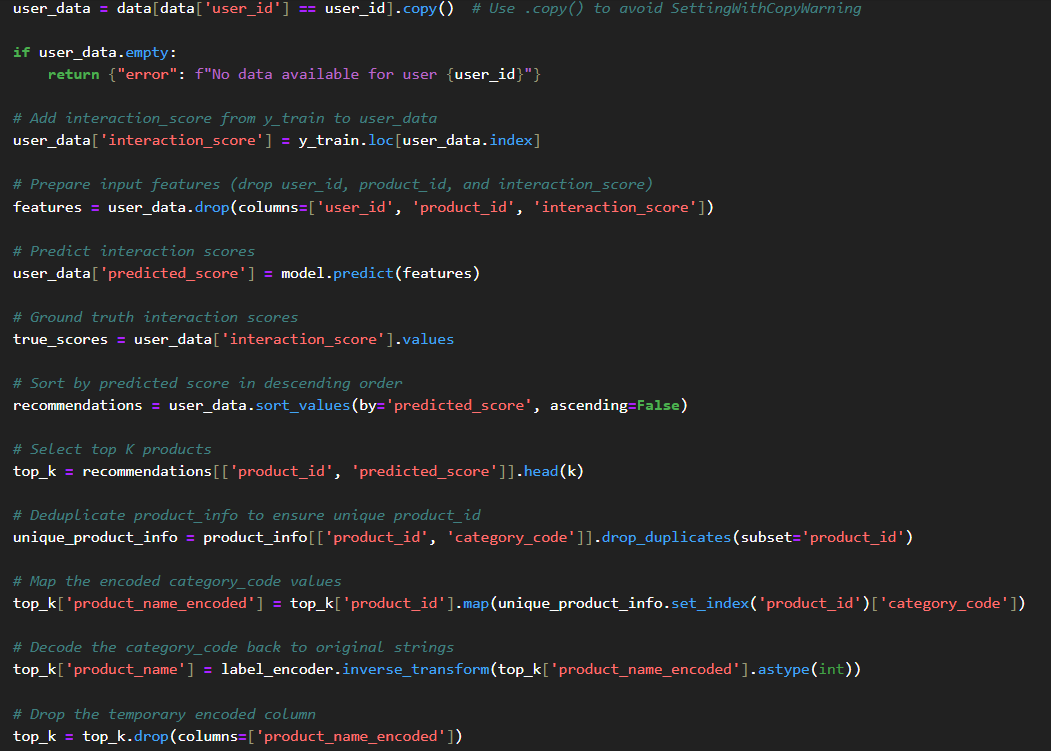


Fig 22. Recommendation algorithm for XGBoost model

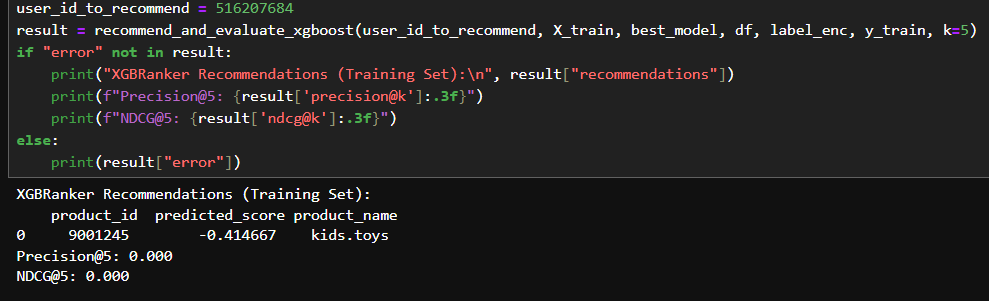


Fig 23. Displaying the recommendation from XGBoost

Defining recommend\_and\_evaluate\_xgboost generates XGBRanker recommendations for user 516207684 using training data. We filter X\_train, add interaction\_score from y\_train, predict scores with best\_model, and rank products. Mapping category\_code to names via label\_encoder, we select top K products. Precision@5 and NDCG@5 evaluate relevance and ranking. The output shows one recommendation, product\_id 9801245 (kids.toys), with a negative predicted\_score. Precision@5 and NDCG@5 are 0.000, as the recommended product’s interaction\_score is likely below 2 (e.g., only viewed), making binary relevance zero. This indicates XGBRanker’s recommendations fail to align with significant user actions like cart or purchase, highlighting potential model limitations on this training set.

1. Random Forest

The Random Forest algorithm, implemented via scikit-learn, excels in recommendation systems by modeling complex patterns. As an ensemble method, it predicts product rankings effectively. Using category\_code, brand, price, total\_interactions, avg\_interaction\_score, and last\_interaction\_score as features, with interaction\_score as the target, recommend\_and\_evaluate\_rf generates Random Forest recommendations.

1. Train the model

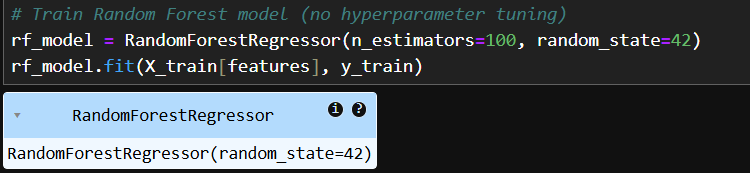


Fig 24. Training the Random Forest model

The model was trained using a 100 estimators, the features it was trained on are 'category\_code', 'brand', 'price', 'total\_interactions', 'avg\_interaction\_score', 'last\_interaction\_score'.

1. Recommending products to user

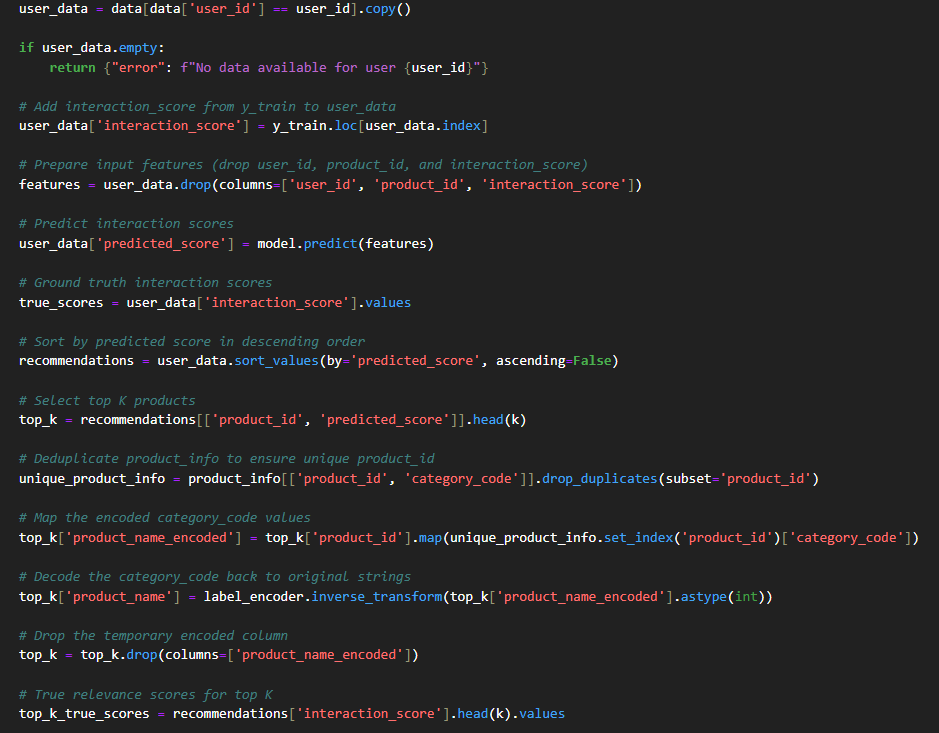


Fig 25. Recommendation algorithm for Random Forest

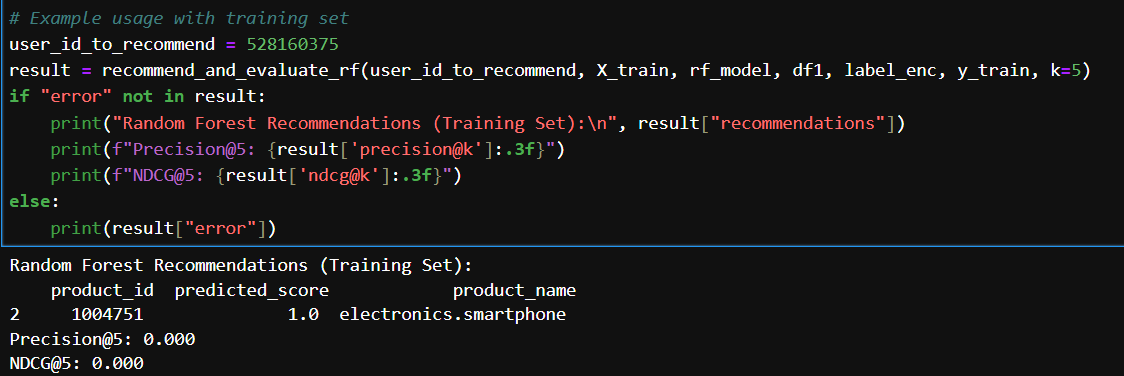
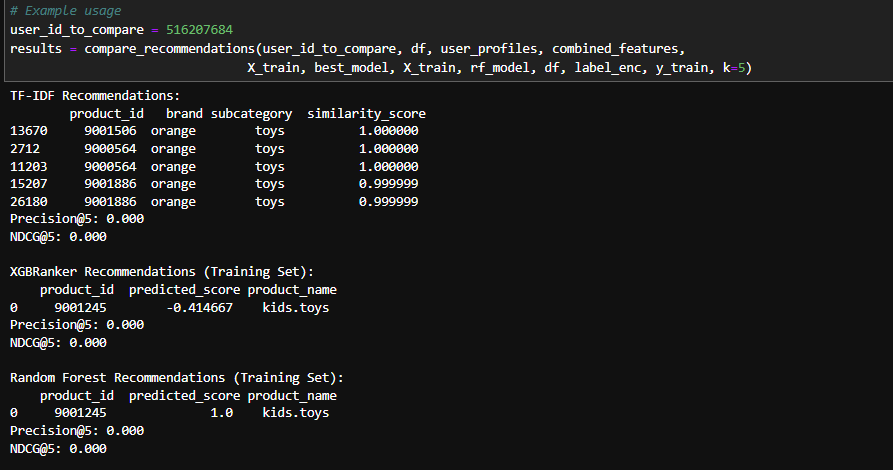


Fig 26. Recommendations from Random Forest

Defining recommend\_and\_evaluate\_rf generates Random Forest recommendations for user 528160375 using training data with features category\_code, brand, price, total\_interactions, avg\_interaction\_score, and last\_interaction\_score, targeting interaction\_score. We filter X\_train, add interaction\_score from y\_train, predict scores with rf\_model, and rank products. Mapping category\_code to names via label\_encoder, we select top K products. The output shows one recommendation, product\_id 1004751 (electronics.smartphone), with predicted\_score 1.0. Precision@5 and NDCG@5 are 0.000, as the interaction\_score is likely below 2 (e.g., only viewed), making binary relevance zero. This indicates Random Forest’s recommendations fail to capture significant user actions like cart or purchase.

**Comparing Recommendations from each model**



Comparing TF-IDF, XGBRanker, and Random Forest for user 516207684 reveals distinct recommendation outcomes on a smaller dataset, constrained by computational limitations, potentially impacting results. TF-IDF recommends toys from "orange" with high similarity scores but yields zero Precision@5 and NDCG@5, indicating no significant user actions like cart or purchase. XGBRanker suggests a kids.toys product with a negative score, also showing zero metrics due to low interaction\_score. Random Forest proposes an electronics.smartphone with a score of 1.0, yet metrics remain zero, reflecting similar relevance issues. Using compare\_recommendations, this analysis highlights TF-IDF’s content-based strength, XGBRanker’s ranking focus, and Random Forest’s pattern recognition. The limited dataset may contribute to zero metrics, as sparse user interactions reduce the likelihood of capturing meaningful engagements, underscoring challenges in achieving accurate recommendations across all methods on the training set.