

# Graduation Project

## Fashion Recommenders Systems



## **Names of Contributors:**

**Mahmoud Salah Abd ElQader**

**Mohamed Elsayed Elsayed**

**Mariem Mahmoud Fathy**

**Maha Magdi Abdelhamid Hassan**

**Alaa Mohamed Shehab**

**Marina Magdy Ramzy**

**Ahmed Osama Abdellatif Alkadi**

## Contents

<b>1- Acknowledgement</b>	<b>3</b>
<b>2- Abstract</b>	<b>4</b>
<b>3- Introduction</b>	<b>4</b>
<b>4- Preprocessing the data for model 1</b>	<b>5</b>
<b>5- Frequently bought together model</b>	<b>9</b>
5.1- Association rule mining	9
5.2- Apriori algorithm	10
5.3- FP-Growth Algorithm	11
<b>6- Product Similarity Based Model</b>	<b>13</b>
6.1- TF-IDF Vectorization	13
6.2- Annoy Index	14
6.3- MobileNetV2	16
6.4- Fashion Clip	20
6.5- Align	23
<b>7- Preprocessing the data for model 3</b>	<b>25</b>
<b>8- User Personalization</b>	<b>34</b>
8.1- User Based Collaborative Filtering	34
8.2- Similarity Based on Embeddings	35
8.3- Two Towers Model	37
<b>9- References</b>	<b>40</b>



## **Acknowledgement**

**We would like to express our deepest gratitude to the Information Technology Institute (ITI) for their unwavering support and for granting us the invaluable nine-month scholarship in the field of Artificial Intelligence. The opportunity to immerse ourselves in such a dynamic and rapidly evolving field has been a transformative experience, equipping us with the knowledge and skills necessary to advance our careers and contribute meaningfully to the industry.**

**We would also like to extend our heartfelt thanks to TWENTRY TOO Company for their continuous assistance and generosity in sharing some of their data with us. This access to real-world problems was crucial for our graduation project, providing us with the practical experience needed to apply our theoretical learning in a tangible and impactful way.**

**Thank you both for your commitment to fostering the next generation of AI professionals.**

## **Abstract**

This project aims to develop three advanced recommendation systems for Twenty Too®, a company specializing in AI-as-a-service solutions for the fashion industry. These systems include: a frequently bought together model to suggest complementary products, a product similarity model to recommend visually and contextually similar items, and a personalization model tailored to individual user preferences based on demographics and browsing history. The objective is to boost basket size, increase in product salability, optimize dead inventory, Reduced time to market, and expedite the cash-to-cash cycle, thereby unleashing the full potential of fashion businesses through powerful AI-driven insights.

## **Introduction**

The fashion industry is rapidly evolving with the integration of artificial intelligence (AI) technologies, enabling businesses to optimize operations, enhance customer experience, and drive sales growth. Offering AI-as-a-service solutions designed to address key challenges in the fashion sector. This project, commissioned by Twenty Too, involves the development of three distinct recommendation systems aimed at improving various aspects of the customer journey and business efficiency.

The first system, a frequently bought together model, leverages historical purchase data to identify and suggest complementary products that customers are likely to purchase together. This model is designed to increase basket size and drive additional sales by presenting relevant product combinations.

The second system focuses on product similarity, utilizing advanced image and text analysis to recommend items that are visually and contextually similar. This model helps customers discover new products that align with their preferences, enhancing their shopping experience and increasing conversion rates.

The third system is a personalization model that tailors recommendations based on individual user demographics and browsing history. By understanding the unique preferences and behaviors of each customer, this model delivers highly relevant suggestions, fostering customer loyalty and satisfaction.

Together, these recommendation systems aim to transform the fashion retail experience, providing businesses with the tools to optimize inventory management, enhance product discovery, and deliver personalized shopping experiences. Through the implementation of these AI-driven solutions, Twenty Too empowers fashion businesses to unlock their full potential and stay competitive in a dynamic market.

## Data Preprocessing

The data cleaning and extraction is designed to streamline the process of preparing raw product data for analysis. It begins by downloading the necessary data files from Google Drive using unique file IDs, specifically fetching a file named `product_data.csv`. Once downloaded, the data is loaded into a pandas DataFrame, facilitating efficient manipulation.

The script contains several key functions:

**download\_files():** This function downloads the required data files from Google Drive using their unique file IDs. It ensures that the `product_data.csv` file is available locally for further processing.

**load\_data():** This function reads the downloaded CSV file into a pandas DataFrame. This DataFrame serves as the primary data structure for manipulation and analysis.

**extract\_price\_info(row, attribute):** This function extracts specific price information from the price column, which contains lists of dictionaries in string format. It retrieves attributes such as price value, currency, original status, valid from, and valid to dates, storing them in new columns.

**add\_price\_columns(df):** This function applies `extract_price_info` to the entire DataFrame, adding new columns for each extracted price attribute. This helps in organizing price information in a structured manner.

**extract\_feature(row, feature):** This function extracts specific features from the `other_attributes` column, which contains various key-value pairs in string format. It retrieves attributes such as product ID, SKU, product URL, review ratings, review counts, color, images, measurements, availability, care instructions, and original descriptions.

**add\_other\_attributes\_columns(df):** This function applies `extract_feature` to the entire DataFrame, adding new columns for each extracted attribute from the `other_attributes` column. This organizes additional product information in a structured manner.

**extract\_hexadecimal(object\_id\_string):** This function extracts the hexadecimal part of the object ID string from the ID column. This is useful for standardizing the ID format.

**clean\_data(df):** This function orchestrates the data cleaning process by calling the `add_price_columns` and `add_other_attributes_columns` functions. It also handles missing values, converts data types, extracts specific parts of string values, and removes unnecessary columns. This ensures the DataFrame is clean and ready for analysis.

**save\_data(df, file\_path):** This function saves the cleaned and organized DataFrame to a new CSV file at the specified file path. This makes the cleaned data available for further analysis or use in other parts of the project.

Finally, the cleaned and organized DataFrame is saved to a new CSV file, ensuring that the data is in a suitable format for subsequent analysis. This streamlined approach enhances the quality and reliability of the project's data handling processes.

# Data Preprocessing for Frequently Bought Together Model

## 1. Data Acquisition

**Objective:** Retrieve the necessary data files from Google Drive for further processing.

- **Methodology:**
  - **Google Drive Download:** The gdown library is employed to download files using their unique file IDs.
  - **Purpose:** Ensures all relevant data files are available locally, enabling efficient data manipulation and analysis.

## 2. Extracting Product IDs

**Objective:** Identify and extract unique product identifiers from specific columns in the dataset.

- **Methodology:**
  - **Regular Expressions:** Utilize regex patterns to search for alphanumeric sequences resembling product IDs.
    - **Pattern Used:** `[a-zA-Z0-9]{3,}` ensures only sequences with a minimum length, likely representing product IDs, are captured.
  - **Data Transformation:**
    - **Exploding Columns:** Rows with multiple product IDs are split into individual rows, ensuring each product ID is represented distinctly.
    - **Creation of New Columns:** Extracted product IDs are stored in new columns (`product_id_cleaned` and `product_id_cleaned2`), facilitating easier reference and analysis.
- **Impact on Model:**
  - **Enhanced Data Quality:** Accurate extraction of product IDs ensures that the subsequent analysis focuses on valid and relevant product identifiers.
  - **Granular Insights:** Exploding rows with multiple IDs allows the model to capture and analyze individual product interactions more precisely.

## 3. Chunk Processing

**Objective:** Process large datasets in manageable chunks to optimize memory usage and computational efficiency.

- **Methodology:**
  - **Reading in Chunks:** The dataset is read in chunks using a specified chunk size.
    - **Chunk Size:** Typically set to 100,000 rows to balance processing speed and memory constraints.

- **Iterative Processing:** Each chunk undergoes the same extraction and transformation processes to ensure uniformity across the entire dataset.
- **Impact on Model:**
  - **Efficient Processing:** Handling data in chunks prevents memory overflow and allows for smoother processing of large datasets.
  - **Consistency:** Ensures all parts of the dataset are processed uniformly, maintaining data integrity.

#### 4. Filtering Data

**Objective:** Focus on relevant events and interactions to hone in on meaningful data for the model.

- **Methodology:**
  - **Events of Interest:** Select events that indicate significant user interactions with products, such as clicks and views.
    - **Examples:** Events like "product\_similar\_items\_clicking" and "product\_details\_page\_view."
  - **Button Names of Interest:** Include interactions with specific buttons to capture user intent and preferences.
    - **Examples:** Buttons like "heart\_button" and "bag\_button."
- **Impact on Model:**
  - **Relevance:** Filtering ensures that only pertinent data is included, enhancing the model's ability to learn from meaningful interactions.
  - **Noise Reduction:** Excluding irrelevant data reduces noise, leading to clearer patterns and more accurate model predictions.

#### 5. Concatenating Filtered Data

**Objective:** Combine filtered data chunks into a comprehensive dataset.

- **Methodology:**
  - **Concatenation:** Merge all processed chunks into a single DataFrame, encompassing all relevant interactions and events.
- **Impact on Model:**
  - **Comprehensive Dataset:** A unified dataset provides a complete view of user interactions, crucial for uncovering patterns in frequently bought together items.
  - **Data Integrity:** Ensures all processed data is included, maintaining the dataset's comprehensiveness and accuracy.



## 6. Identifying Unique Product IDs

**Objective:** Ensure the dataset contains only unique product identifiers.

- **Methodology:**
  - **Removing Duplicates and Null Values:** Drop duplicate entries and null values from the dataset.
  - **Storage of Unique IDs:** Extract and store unique product IDs separately for focused analysis.
- **Impact on Model:**
  - **Data Quality:** Removing duplicates and null values ensures the dataset is clean, improving the model's learning process.
  - **Accurate Analysis:** Focusing on unique product IDs enhances the precision of identifying frequently bought together items.

## 7. Saving Processed Data

**Objective:** Save the processed data for future use in the frequently bought together model.

- **Methodology:**
  - **CSV Files:** Save the cleaned product IDs and the filtered dataset to CSV files.
    - **Files Created:** cleaned\_product\_ids.csv and all\_df\_filtered.csv.
- **Impact on Model:**
  - **Ease of Use:** CSV files ensure compatibility with various data analysis tools and facilitate easy sharing.
  - **Prepared Data:** Having the data preprocessed and saved allows for seamless integration into the modeling phase.

## Summary

The preprocessing steps outlined above are meticulously designed to prepare the dataset for the frequently bought together model. By downloading, extracting, transforming, filtering, and saving the data, we ensure that the dataset is clean, consistent, and ready for analysis using the Apriori and FP-Growth algorithms. These preprocessing steps are crucial for generating accurate and meaningful recommendations, enhancing the overall shopping experience through sophisticated AI-driven insights.

- **Data Quality:** Enhanced through meticulous extraction and cleaning processes.
- **Relevance:** Achieved by filtering for pertinent events and interactions.
- **Efficiency:** Ensured by processing data in manageable chunks.
- **Accuracy:** Improved by focusing on unique product IDs and removing duplicates and noise.

- **Preparedness:** Facilitated by saving processed data in accessible formats for seamless modeling and analysis.

## Model 1 Frequently bought together

In our Frequently Bought Together model, we implemented two approaches to identify frequently co-occurring items: the Apriori algorithm and the FP-Growth algorithm. Each method was selected to leverage its unique strengths in the process of discovering frequent itemsets.

### Approach 1: Association Rule Mining for Frequently Bought Together Model [3]

Association Rule Mining is a powerful data mining technique aimed at uncovering interesting relationships, patterns, and correlations within large datasets. It focuses on identifying strong associations between different items or variables in the data, presenting these associations in the form of if-then rules, commonly known as association rules. This method is particularly useful for recommending frequently bought together fashion items.

#### Key Concepts in Association Rule Mining

**Antecedent (If part):** The condition or itemset found in the dataset.

**Consequent (Then part):** The result or itemset derived from the antecedent.

**Support:** is the proportion of transactions in the dataset that contain a specific itemset.

**Importance:** It indicates the frequency with which the itemset appears in the data. Higher support values suggest that the rule is more common or significant within the dataset. Rules with low support are considered less relevant.

**Confidence:** is the ratio of the number of transactions containing both the antecedent and the consequent to the number of transactions containing only the antecedent.

**Importance:** It measures the reliability of the inference made by the rule. Confidence is not symmetric, meaning the confidence for  $\{X \rightarrow Y\}$  is not the same as  $\{Y \rightarrow X\}$ .

**Lift:** is the ratio of the observed support for a rule to the expected support if the items were independent.

**Importance:** Lift measures the strength of an association rule over random co-occurrence. A lift value greater than 1 indicates a positive association between the antecedent and the consequent, meaning they occur together more frequently than expected if they were independent. A lift value less than 1 indicates a negative association.

When applied to fashion recommender systems, Association Rule Mining can identify combinations of items that are frequently bought together. By analyzing transaction data, the system can generate recommendations that enhance the shopping experience by suggesting complementary products. For example, if the antecedent is a pair of jeans and the consequent is a t-shirt, a high confidence value for the rule {Jeans  $\rightarrow$  T-Shirt} indicates that customers who buy jeans often buy t-shirts as well. This insight can be used to recommend t-shirts to customers viewing or purchasing jeans, thereby increasing the likelihood of additional sales and improving customer satisfaction.

### **Approach 2: Apriori Algorithm [1]**

The Apriori algorithm is a classic method for mining frequent item sets and association rules. It operates by generating candidate item sets and iteratively increasing their size until no more frequent item sets are found.

Reason for Use:

Simplicity: Apriori's straightforward approach makes it easy to understand and implement.

Initial Insights: It provided a foundational understanding of the frequent item sets within our dataset.

Steps:

Set the minimum support threshold (min frequency required) for an itemset to be "frequent".

Identify frequent individual items (count the occurrence) of each individual item.

Generate candidate item sets of size 2 (create pairs of frequent items) discovered.

Prune infrequent item sets (eliminate item sets) that do not meet the threshold levels.

Generate item sets of larger sizes - combine the frequent item sets of size 3,4, and so on.

Repeat the pruning process (keep eliminating the item) sets that do not meet the threshold levels.

Iterate till no more frequent item sets can be generated.

Generate association rules that express the relationship between them - calculate measures to evaluate the strength & significance of these rules.

After the Apriori algorithm generates the item sets, the strength of the generated associations and relationships can be investigated. This enables the fashion recommender system to suggest comprehensive and personalized fashion recommendations to customers, enhancing their shopping experience and increasing sales.

Advantages:

Calculates large item sets.

Simple to understand and apply.

Disadvantages:

Expensive in terms of computational resources due to frequent database scans.

Computationally expensive with a large number of candidate rules.

### **Approach 3: FP-Growth Algorithm [2]**

After obtaining initial results with the Apriori algorithm, we employed the FP-Growth algorithm. FP-Growth improves performance by avoiding candidate generation and using a compact data structure called the FP-tree. It employs a divide-and-conquer strategy, leveraging a special data structure known as the frequent-pattern tree (FP-tree) to maintain item set association information.

Reason for Use:

**Efficiency:** Unlike Apriori, which requires multiple database scans for candidate generation, FP-Growth scans the database only twice, making it significantly faster.

**Scalability:** FP-Growth is more efficient and scalable, especially for large datasets, due to its compact memory representation of the database.

**Enhanced Performance:** The divide-and-conquer strategy of FP-Growth allows for effective mining of both long and short frequent patterns, further optimizing the performance of our model.

FP-tree Construction:

**Input Compression:** Compress the input database to create an FP-tree instance.

**Conditional Database Creation:** Divide the compressed database into a set of conditional databases.

Recursive Mining: Mine each conditional database separately, recursively looking for short patterns and concatenating them into long frequent patterns.

Database Partitioning:

For very large databases, partition the database into smaller projected databases and construct an FP-tree from each.

Steps:

Initial Database Scan:

Scan the database to find itemset occurrences and determine support count.

FP-tree Construction:

Create the root of the tree (null) then, scan the database again to examine transactions, ordering item sets by descending frequency count.

Mining the FP-Tree:

Examine the lowest nodes first to identify the frequency pattern of length 1, then traverse paths in the FP-tree to create a conditional pattern base, finally construct a Conditional FP-Tree and generate frequent patterns.

Advantages:

Efficient database scanning (only two scans).

Faster due to avoiding item pairing, compact memory storage.

Scalable for mining both long and short frequent patterns.

Disadvantages:

Complex and cumbersome to build, expensive in terms of computational resources.

May not fit into shared memory for very large databases.

Conclusion

By combining these two approaches, we leveraged the simplicity and initial insights provided by the Apriori algorithm, followed by the efficiency and scalability of the FP-Growth algorithm.

## Model 2 Product similarity [4]

Text Feature Extraction:

To prepare our data for the similarity model, we used two primary techniques: TF-IDF vectorization and one-hot encoding for text feature extraction.

Data Preparation:

We started by creating a combined text for each product, merging the values from the title, target\_audience, color, department, and Original\_Description columns. This combined text was saved in a new column, which served as the input for our feature extraction methods.

TF-IDF Vectorization:

$$w_{x,y} = tf_{x,y} \times \log \left( \frac{N}{df_x} \right)$$

**TF-IDF**

Term  $x$  within document  $y$

$tf_{x,y}$  = frequency of  $x$  in  $y$

$df_x$  = number of documents containing  $x$

$N$  = total number of documents

### 1- TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) vectorization was used to capture the significance of words in the product descriptions across the entire dataset. This technique transforms textual data into numerical vectors, where each dimension represents the importance of a word in the context of the document and the corpus. By focusing on the relative importance of words, TF-IDF helps to highlight the most relevant terms for distinguishing different products.

Architecture:

Initialization: Configure parameters like max\_features to limit the number of features. Which in our case the max\_features equal to 5000.

Fit: Learn the vocabulary and IDF (inverse document frequency) from the training set.

Transform: Transform documents to TF-IDF matrix based on learned vocabulary and IDF values.

One-Hot Encoding:

In addition to textual data, we had categorical features such as target\_audience, color, and department. These features were one-hot encoded, a process that converts categorical variables into a binary matrix where each column represents a unique category. This encoding allows the categorical data to be integrated into the numerical feature space used by the model.

OneHotEncoder: The OneHotEncoder from the scikit-learn library encodes categorical integer features as a one-hot numeric array. This is useful for converting categorical data into a binary matrix that can be used by machine learning algorithms.

Architecture:

Initialization: Configure parameters such as categories to define the expected categories which are color, target\_audience and department.

Fit: Learn the unique categories from the data.

Transform: Transform the categorical data into a binary matrix

Combined Feature Set:

The TF-IDF matrix and the one-hot encoded matrix were horizontally stacked to form a combined feature set, integrating both textual and categorical information into a unified representation.

## 2- Annoy

Annoy (Approximate Nearest Neighbors Oh Yeah): is an efficient technique developed by Spotify for finding approximate nearest neighbors in high-dimensional spaces. It is particularly useful in scenarios where exact nearest neighbor search becomes computationally expensive due to the high dimensionality of the data. Annoy builds a forest of trees, each constructed by randomly selecting a subset of the data and recursively splitting it by hyperplanes. This allows for rapid querying of nearest neighbors with a trade-off between search speed and accuracy, making it an excellent choice for our high-dimensional data where traditional techniques like cosine similarity would be less effective.

After deciding to use the Annoy Index, we passed the combined feature set to the model, using angular metrics for distance measurement and a maximum depth of 20 for the trees. We ran the model on our combined features to build the index.

Approximate Nearest Neighbor (ANN) techniques balance a slight reduction in accuracy for significantly improved search efficiency in machine learning tasks. These techniques are particularly effective in scenarios where computational speed is crucial, such as recommendation systems and image recognition. ANN methods are used to efficiently find the closest points in high-dimensional spaces, with applications in data mining, machine learning, and computer vision.

How internally ANN works[5] :

### 1. Index Construction

Tree-Based Structures:

Many ANN algorithms, such as Annoy and KD-Trees, use tree-based structures to partition the data space. These trees are constructed by recursively dividing the data into smaller regions using hyperplanes.

Annoy: Builds a forest of trees where each tree is constructed by randomly selecting a subset of the data and splitting it based on randomly chosen hyperplanes. This randomness helps in balancing the tree and making the search process efficient.

## 2. Index Optimization

### Dimensionality Reduction:

To improve efficiency, ANN algorithms often employ dimensionality reduction techniques, such as Principal Component Analysis (PCA), to reduce the complexity of the data. This reduces the computational load during the search phase.

## 3. Search Process

### Approximate Search:

Instead of performing an exhaustive search over all data points, ANN algorithms use the index structure to quickly narrow down the search space. For example, tree-based methods use hierarchical partitioning to eliminate large portions of the space, while graph-based methods traverse the graph to find approximate nearest neighbors.

### Heuristic Approaches:

ANN algorithms often use heuristic approaches to balance speed and accuracy. For example, in Annoy, the depth of the trees and the number of trees in the forest are parameters that can be adjusted to control the trade-off between search speed and accuracy.

## 4. Algorithmic Variants

### Randomized Approaches:

Randomized approaches, such as the one used in Annoy, involve creating multiple trees with different random splits. This randomness helps in exploring different regions of the data space and improves the likelihood of finding approximate neighbors quickly.



## Image Feature Extraction

### MobileNetV2 [6][7]

MobileNetV2 is an advanced convolutional neural network architecture designed to efficiently handle image classification tasks while minimizing computational resources. Its architecture integrates several innovative components to enhance performance and efficiency:

#### MobileNetV2 Architecture

##### 1. Depthwise Separable Convolutions

MobileNetV2 uses depthwise separable convolutions as its core building block. This approach consists of two main steps:

**Depthwise Convolution:** Applies a single convolutional filter per input channel. This reduces the number of computations compared to standard convolutions, which apply multiple filters across all channels simultaneously.

**Pointwise Convolution:** Uses a 1x1 convolution to combine the outputs of the depthwise convolution. This step mixes information across channels, allowing the model to learn more complex features.

##### 2. Linear Bottlenecks

The concept of Linear Bottlenecks is introduced in MobileNetV2 to improve the model's performance. Bottlenecks are layers where the output is passed through a linear layer (without a non-linearity) before it is used as input to the next layer. This design helps to preserve the information through the bottleneck, avoiding the loss of data that can occur with activation functions like ReLU.

##### 3. Inverted Residuals

MobileNetV2 employs Inverted Residual Blocks which consist of three main components:

**Depthwise Separable Convolution:** As described above, used to reduce computational complexity.

**Linear Layer:** A 1x1 convolution that projects the feature maps to a higher dimension before applying the depthwise convolution.

**Shortcut Connection:** A residual connection that skips the depthwise separable convolution and adds the input directly to the output of the block. This helps in preserving important features and improving gradient flow during training.

##### 4. Linear Activation

Unlike previous architectures that used ReLU activation functions, MobileNetV2 uses linear activation functions in the bottleneck layers. This design choice helps in maintaining more information through the network layers and supports better performance, especially in deep networks.

##### 5. Global Average Pooling

At the end of the network, MobileNetV2 uses Global Average Pooling to reduce the spatial dimensions of the feature maps to a single vector per channel. This vector is then passed to a fully connected layer to produce the final classification output.

### Feature Extraction Using MobileNetV2

In this process, MobileNetV2 is utilized to extract features from images, which are then used to compute similarities between products.

#### Feature Extraction Process

**Image Fetching and Preprocessing:** Images are loaded and preprocessed to match the input requirements of MobileNetV2. This includes resizing and normalization steps to ensure consistency.

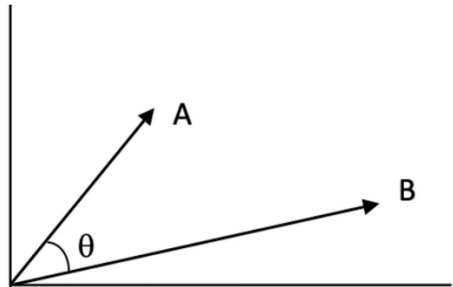
**Feature Extraction:** For each image, features are extracted using MobileNetV2. These features are high-dimensional vectors that represent various aspects of the image.

**Storing Features:** Extracted features are appended to a DataFrame along with the product ID. This DataFrame accumulates feature vectors for all processed images.

## Similarity Computation [8]

Cosine similarity is a metric used to measure how similar two vectors are in a high-dimensional space. It is widely used in data science and machine learning due to its effectiveness in evaluating similarity between feature vectors, especially in high-dimensional datasets such as image features.

Definition: Cosine similarity calculates the cosine of the angle between two vectors. It is defined as:


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

where  $\mathbf{A}$  and  $\mathbf{B}$  are the vectors being compared,  $\mathbf{A} \cdot \mathbf{B}$  represents their dot product, and  $\|\mathbf{A}\|$  and  $\|\mathbf{B}\|$  are the magnitudes (norms) of the vectors.

Range: The cosine similarity value ranges from -1 to 1. A value of 1 indicates that the vectors are identical in direction, 0 indicates orthogonality (no similarity), and -1 indicates opposite directions.

To find similar images, we use cosine similarity. This is particularly suitable for our high-dimensional image features because it focuses on the orientation of the vectors rather than their magnitude.

Cosine Similarity Calculation: For a given image, its feature vector is compared to all other image feature vectors using cosine similarity. This produces a similarity score for each pair of images.

Top-K Similar Images: The goal is to identify the most similar images to a specified image. This involves:

Retrieving Features: Extracting the feature vector of the specified image.

Calculating Similarities: Computing the cosine similarity between this vector and all other vectors in the DataFrame.

Sorting and Selecting: Sorting the similarity scores in descending order to find the top N most similar images. The indices corresponding to these top scores are used to retrieve the image IDs of the most similar images.

Image features extracted from MobileNet V2 are represented as vectors. Cosine similarity is then applied to measure the similarity between these image feature vectors, identifying visually similar products.

### **Vector Concatenation and Cosine Similarity (Text + Image):**

By concatenating TF-IDF vectors (from text descriptions) and MobileNet V2 image feature vectors, a combined feature vector is created for each item.

Cosine similarity is applied to this combined vector to measure the similarity between items based on both textual and visual characteristics. This approach leverages the complementary nature of text and image features to enhance similarity modeling.

### **RoBERTa Large for Text and EfficientNet-B5 for Image:**

RoBERTa Large is a transformer-based model pretrained on a large corpus of text data, capable of capturing complex semantic relationships in textual descriptions.

EfficientNet-B5 is a state-of-the-art CNN architecture known for its balance between accuracy and computational efficiency in image processing tasks.

By applying RoBERTa Large to text descriptions and EfficientNet-B5 to images, high-dimensional embeddings are obtained for each item. Similarity between items is then computed using cosine similarity on these embeddings, capturing nuanced relationships based on both textual semantics and visual content.

### **CLIP (Contrastive Language-Image Pretraining):**

CLIP is a transformative model that jointly learns from text and images by aligning them in a shared embedding space.

CLIP uses a transformer-based architecture to encode both text and image inputs, enabling direct comparison and retrieval of semantically related items across modalities.

By leveraging CLIP, the similarity model benefits from a unified representation of items that encapsulates their textual descriptions and visual features, facilitating more accurate and context-aware recommendations based on comprehensive understanding.

## Fashion Clip [9]

### Overview

FashionCLIP is a state-of-the-art model adapted from CLIP (Contrastive Language-Image Pre-training), specifically fine-tuned for the fashion industry. It is designed to learn and extract meaningful representations from both images and textual descriptions, providing a robust framework for a variety of fashion-related tasks. This documentation outlines how FashionCLIP can be utilized to extract features from image and description pairs effectively.

### Key Features

1. **Multi-modal Embeddings:** FashionCLIP generates embeddings for both images and textual descriptions, mapping them into a shared latent space. This allows for meaningful comparisons and retrieval tasks.
2. **Generalization:** Trained on a diverse fashion dataset, FashionCLIP captures broad fashion concepts, making it applicable to various fashion-related applications.
3. **Zero-shot Capabilities:** The model can handle unseen datasets and descriptions, demonstrating its versatility and adaptability in real-world scenarios.

### Applications

FashionCLIP can be applied to numerous tasks within the fashion industry, including:

- **Product Search:** Enhancing search functionality by comparing textual queries with visual features of products.
- **Recommendation Systems:** Generating recommendations based on image and text similarity.
- **Content-Based Retrieval:** Facilitating retrieval of similar fashion items based on user-provided descriptions or images.
- **Classification:** Assisting in categorizing fashion items by comparing their visual and textual features.

### Feature Extraction Process

1. **Input Preparation:** The process begins by preparing pairs of images and descriptions. Each pair consists of a fashion item's image and its corresponding textual description.
2. **Embedding Generation:**
  - **Image Embeddings:** FashionCLIP's vision encoder processes the image to produce a high-dimensional vector representation.
  - **Text Embeddings:** The text encoder generates a corresponding vector from the description.

3. **Similarity Measurement:** The embeddings of images and descriptions are aligned in a shared latent space. This alignment enables comparison and retrieval based on the proximity of these vectors.
4. **Use Case Implementation:** Extracted features can be utilized for various downstream tasks such as similarity search, recommendation generation, and classification.

### **CLIP Architecture [10]**

The CLIP (Contrastive Language-Image Pre-training) model, the foundation of FashionCLIP, consists of two main components:

1. **Vision Encoder:**

- **Architecture:** The vision encoder is typically based on a deep convolutional neural network or a vision transformer. It processes raw images to extract high-level visual features.
- **Output:** The output is a dense vector that captures the visual characteristics of the image. This vector is later normalized and used in the shared latent space.

2. **Text Encoder:**

- **Architecture:** The text encoder is generally a transformer-based model. It converts textual descriptions into dense vector representations.
- **Output:** The output vector captures the semantic meaning of the text and is projected into the same latent space as the image vectors.

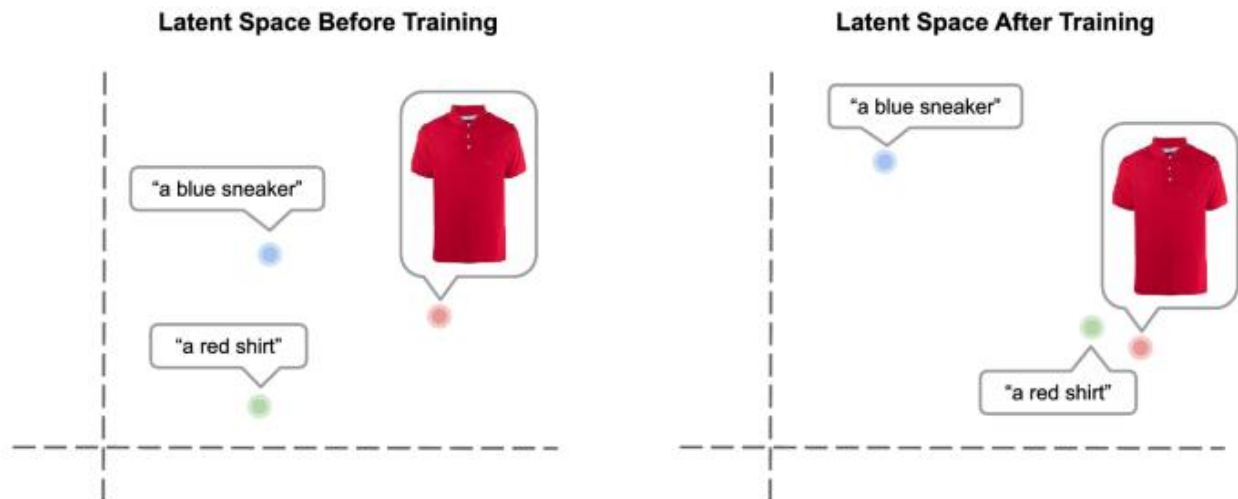
3. **Contrastive Learning Objective:**

- **Training:** Both encoders are trained using a contrastive learning approach, where the model learns to align image and text pairs in a shared vector space. Matching image-text pairs are positioned close together, while non-matching pairs are pushed apart.
- **Loss Function:** The model employs a cross-entropy loss to maximize the similarity of matching pairs and minimize the similarity of non-matching pairs in the latent space.

4. **Latent Space Alignment:**

- **Shared Space:** Both image and text embeddings are projected into a common vector space, allowing for direct comparison and similarity measurement.

- **Applications:** This alignment enables various applications such as image retrieval using text queries and vice versa.



## Evaluation

To ensure the quality of the extracted features, evaluate the model's performance on:

- **Retrieval Accuracy:** Check how well the features support accurate retrieval of similar items.
- **Generalization:** Test the model's ability to generalize features to new and unseen datasets.
- **Robustness:** Assess the consistency and reliability of the features across different tasks and contexts.

## Benefits

- **Scalability:** The model can handle large-scale fashion datasets efficiently.
- **Flexibility:** FashionCLIP's embeddings are versatile, supporting a wide range of fashion-related applications.
- **Cost-Effectiveness:** By leveraging pre-trained models, FashionCLIP reduces the need for extensive task-specific training.

## ALIGN [11]

**ALIGN (Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision)** is a state-of-the-art model designed to enhance visual and vision-language representation learning. It addresses the limitations of existing models that rely on heavily curated datasets by leveraging a large-scale, noisy dataset to improve scalability and performance.

### Purpose

ALIGN aims to improve representation learning for both visual and vision-language tasks using a significantly larger and noisier dataset compared to traditional models. By minimizing data cleaning, the model scales efficiently while maintaining high-quality representations.

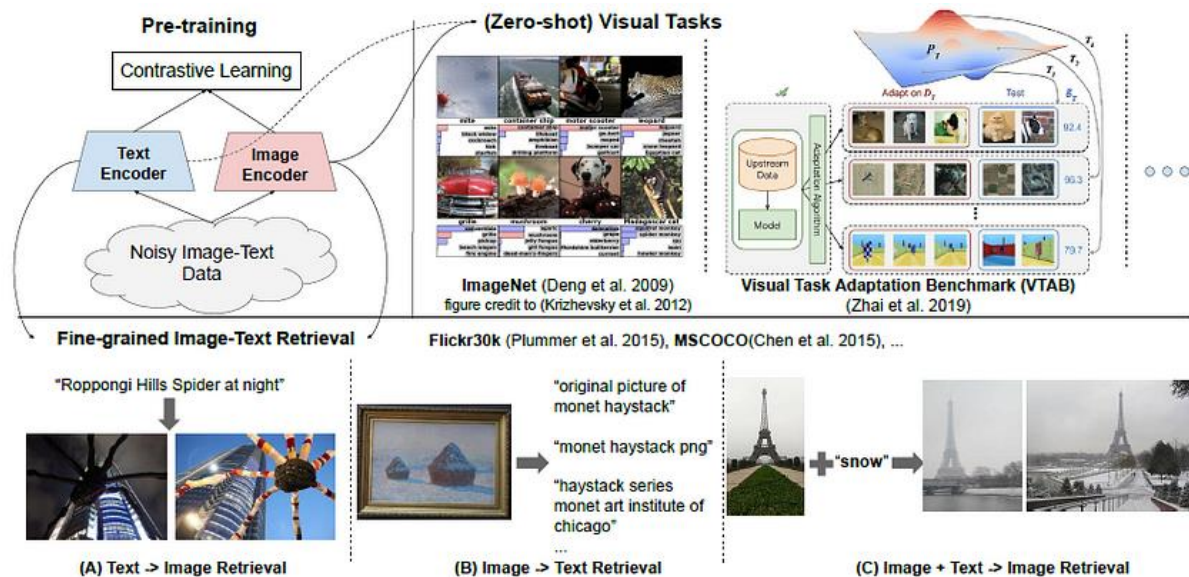
### Architecture

#### Dual-Encoder System

- **Image Encoder:** EfficientNet
- **Text Encoder:** BERT

The model utilizes a dual-encoder architecture where images and text are processed separately by distinct encoders. This architecture aligns visual and textual representations into a shared latent space using a contrastive loss function.

#### Training Data



- **Dataset:** Over one billion noisy image-text pairs
- **Data Filtering:** Minimal filtering to retain large-scale data



## Key Points and Advantages

### Noisy Dataset

ALIGN leverages a vast dataset with minimal filtering, allowing it to scale beyond the constraints of heavily curated datasets used by traditional models.

### Simple Learning Scheme

Despite the noise in the dataset, the sheer volume of data enables ALIGN to achieve high-quality representations, compensating for the lack of data cleaning.

## Advantages

- **State-of-the-Art Performance:** Excels in zero-shot image classification and cross-modal retrieval tasks.
- **Efficiency:** Outperforms more complex models, such as those using cross-attention mechanisms, in both performance and practical application.
- **Versatility:** Effective across various tasks, including image classification, image-to-text retrieval, text-to-image retrieval, and combined image-text queries.

## Performance

- **Zero-Shot Image Classification:** Achieves 76.4% top-1 accuracy on ImageNet without any training samples.
- **Image-to-Text Retrieval:** Surpasses previous state-of-the-art methods by over 7% in R@1 metrics on benchmarks like Flickr30K and MSCOCO.

# Data Preprocessing in the Third Model of Personalization

## Introduction

The third model in our personalization project focuses on predicting and filling missing age values to enhance user profiles and improve recommendation accuracy. This document provides a detailed explanation of each preprocessing step, its purpose, and the impact on the model's performance and effectiveness.

## Step-by-Step Preprocessing

### 1. Download Files

**Objective:** Obtain the necessary data files from remote storage.

**Method:** Use a script to download specific CSV files from Google Drive.

**Rationale:** Automating the file download ensures that the data used is the most current, reducing the risk of discrepancies.

**Impact on Model:** Ensures the model is trained and tested on the latest data, which is crucial for accuracy and relevance in recommendations.

### 2. Load Data

**Objective:** Load the downloaded data files into memory for processing.

**Method:** Use pandas to read CSV files into DataFrames.

**Rationale:** Efficient data manipulation and analysis require data to be loaded into a format suitable for processing.

**Impact on Model:** Proper data loading facilitates smooth execution of subsequent preprocessing steps, ensuring the data is readily available for analysis and modeling.

### 3. Clean Data

**Objective:** Remove duplicate entries and rows with missing critical information.

**Method:**

- Drop duplicate rows using `df.drop_duplicates()`.
- Drop rows with missing values in the 'Answer' column using `df.dropna(subset=['Answer'])`.

**Rationale:** Removing duplicates and incomplete entries improves data quality and reduces noise.

**Impact on Model:** Clean data leads to more reliable model training and evaluation, as duplicates and missing values can skew results and reduce model accuracy.

### 4. Extract Age from 'Answer' Column

**Objective:** Derive the age of users from their responses in the 'Answer' column.

**Method:**

- Use regular expressions to parse birthdates from the 'Answer' column.
- Calculate age based on the extracted birthdate.

**Rationale:** Converting textual data into numeric age values enhances the dataset's completeness.

**Impact on Model:** Accurate age information enriches the user profile, leading to more personalized recommendations and improving model performance.

## 5. Add Age and Age Group

**Objective:** Categorize users into age groups based on their extracted ages.

**Method:**

- Define age ranges and create age groups using `pd.cut()`.

**Rationale:** Grouping users into categories allows the model to apply age-specific recommendations.

**Impact on Model:** Age grouping provides additional context for personalization, enabling the model to make more nuanced recommendations based on age demographics.

## 6. Extract Unique Users

**Objective:** Identify unique users and their associated demographic information.

**Method:**

- Select relevant columns and remove duplicates using `df.drop_duplicates(subset=['User_ID'])`.

**Rationale:** Ensuring each user is represented once prevents redundancy and ensures accurate data representation.

**Impact on Model:** Accurate and unique user records ensure that each user is treated individually, which improves the precision of personalized recommendations.

## 7. Fill Missing Ages

**Objective:** Predict and fill missing age values for users.

**Method:**

- Group data by 'User\_ID' and apply forward and backward filling.
- Use linear regression to predict missing ages based on other features.

**Rationale:** Predicting and filling missing ages enhances data completeness and allows the model to utilize all available information.

**Impact on Model:** Filling missing ages improves the dataset's completeness, leading to better model training and more accurate predictions.

## 8. Merge Age Data with Main DataFrame

**Objective:** Integrate the filled age data back into the main dataset.

**Method:**

- Drop existing age columns in the main DataFrame to avoid conflicts.
- Map predicted age and age group values back to the main DataFrame.

**Rationale:** Merging ensures that the enriched age data is utilized throughout the dataset.

**Impact on Model:** Integrating accurate age data into the main DataFrame ensures that the model benefits from the improved data quality for all subsequent analyses and predictions.

## 9. Convert Columns to Numeric

**Objective:** Ensure all relevant columns are in the appropriate numeric format.

**Method:**

- Convert specified columns to numeric types using `pd.to_numeric()`.

**Rationale:** Numeric data is necessary for many machine learning algorithms that require numerical inputs.

**Impact on Model:** Properly formatted numeric data ensures that the model can effectively process and analyze the data, leading to better performance.

## 10. Merge Additional Data

**Objective:** Combine additional data with the main DataFrame for comprehensive analysis.

**Method:**

- Merge columns based on common attributes and handle datatype inconsistencies.

**Rationale:** Merging additional data provides a richer dataset, which can improve model accuracy.

**Impact on Model:** A comprehensive dataset allows the model to make more informed predictions by leveraging additional context and information.

## 11. Filter Data

**Objective:** Focus on relevant events and interactions.

**Method:**

- Filter the dataset based on specific events and button names of interest.

**Rationale:** Filtering isolates meaningful interactions, reducing noise and improving focus.

**Impact on Model:** Filtering ensures that the model is trained on relevant and significant interactions, enhancing its ability to make accurate recommendations.

## 12. Apply Interaction Mapping

**Objective:** Categorize interactions based on event and button name.

**Method:**

- Map combinations of event names and button names to interaction categories and assign weights.

**Rationale:** Categorizing interactions provides context for the model to differentiate between various types of user behavior.

**Impact on Model:** Categorization improves the model's ability to understand and leverage different types of interactions, enhancing the personalization of recommendations.

### 13. Handle Missing Values in 'City' and 'Region'

**Objective:** Ensure no missing values in critical location fields.

**Method:**

- Fill missing 'region' values with corresponding 'city' values and vice versa.
- Apply forward and backward filling within user groups.

**Rationale:** Complete location data is essential for accurate geographical-based recommendations.

**Impact on Model:** Filling missing location data ensures that the model has complete information for making location-based recommendations, improving its relevance and accuracy.

**Conclusion**

This detailed preprocessing methodology ensures that the dataset is prepared for effective model training and evaluation. Each step is designed to improve data quality, completeness, and relevance, ultimately leading to a more accurate and personalized recommendation system. By understanding the impact of each preprocessing step, we ensure that the model benefits from high-quality data, resulting in improved user satisfaction and engagement.

## Recommender System Of Personalization

**Overview**

This documentation outlines the recommender system designed to provide personalized product recommendations. The system addresses different user scenarios, including existing users, new users with event data, and new users without demographic data. It integrates various recommendation techniques to ensure relevance and engagement.

**System Components**

### 1. Recommendations for Existing Users

**Function:** recommend\_for\_existing\_user

**Description:** This function provides recommendations based on the historical interactions of existing users. It utilizes user-based collaborative filtering to identify products that a target user has not interacted with but are popular among users with similar preferences.

**Process:**

1. **Extract User History:**
  - Retrieve products previously interacted with by the target user.
2. **Identify Similar Users:**
  - Find users who have interacted with the same products as the target user.
3. **Aggregate Recommendations:**
  - Recommend products from the interaction history of similar users, excluding items already interacted with by the target user.

**Mathematical Background:**

- **User Similarity Measures:**
  - **Cosine Similarity:** Measures the cosine of the angle between two user vectors based on their interaction history, providing a measure of similarity.
  - **Pearson Correlation:** Assesses the linear relationship between the interaction patterns of users.

**Impact on Model:**

- **Personalization:** Offers recommendations tailored to user preferences based on similar user interactions, improving relevance.

## 2. Recommendations for New Users with Event Data

**Function:** `recommend_for_new_user_with_events`

**Description:** This function provides recommendations for new users who have interacted with a specific product. It uses item-based collaborative filtering to recommend products based on similar user interactions and falls back to popular items if no similar users are found.

**Process:**

1. **Find Similar Users:**
  - Identify users who have interacted with the same product as the new user.
2. **Aggregate Recommendations:**
  - Recommend products from the interaction history of similar users, excluding the product already interacted with by the new user.

### 3. Fallback Mechanism:

- If no similar users are found, recommend the most popular items.

#### Mathematical Background:

- **Item-Based Similarity Measures:**

- **Item Similarity:** Determines similarity between items based on user interaction patterns.
- **Popularity Calculation:** Measures the frequency of interactions for each item to rank them.

#### Impact on Model:

- **Effective Recommendations:** Provides relevant suggestions even with limited data by leveraging similar user interactions and ensuring engagement through popular items.

### 3. Recommendations for New Users without Demographic Data

**Function:** `recommend_popular_items`

**Description:** This function recommends popular items to new users when no historical or demographic data is available. It relies on item popularity across the entire dataset.

#### Process:

1. **Calculate Popularity:**

- Identify and rank items based on their frequency of interactions.

2. **Recommend Top Products:**

- Suggest the most popular items to the new user.

#### Mathematical Background:

- **Frequency Count:** Counts the number of interactions for each product.
- **Ranking Algorithm:** Orders items based on their interaction frequency to determine popularity.

#### Impact on Model:

- **Engagement:** Ensures new users receive suggestions for widely liked items, increasing the likelihood of engagement.

### 4. Logging New User Interactions

**Function:** `log_user_interaction`

**Description:** This function logs interactions of new users with products and updates the dataset accordingly. It captures details such as user ID, product interacted with, event name, and button interaction.

#### Process:

1. **Create Interaction Record:**

- Generate a new record with the user's interaction details.

2. **Update Dataset:**

- Append the new interaction record to the existing dataset.

**Impact on Model:**

- **Data Enrichment:** Adds new interaction data to the dataset, allowing for future updates and improvements in recommendations.

**Main Function: process\_user**

**Description:** The process\_user function determines the appropriate recommendation strategy based on the user's status (existing or new) and available data.

**Process:**

1. **Existing User:**

- Recommends products based on historical interactions.

2. **New User with Event Data:**

- Recommends products based on similar user interactions and adds the new user's interaction data to the dataset.

3. **New User without Data:**

- Recommends popular items and logs the new user's interaction.

**Impact on Model:**

- **Flexibility:** Adapts to different user scenarios, ensuring relevant recommendations are provided while updating the dataset for future interactions.

**Summary**

This recommender system employs a combination of user-based and item-based collaborative filtering techniques, along with popularity-based recommendations, to cater to various user scenarios. Each component is designed to enhance the relevance of suggestions, improve user engagement, and adapt to the availability of data.

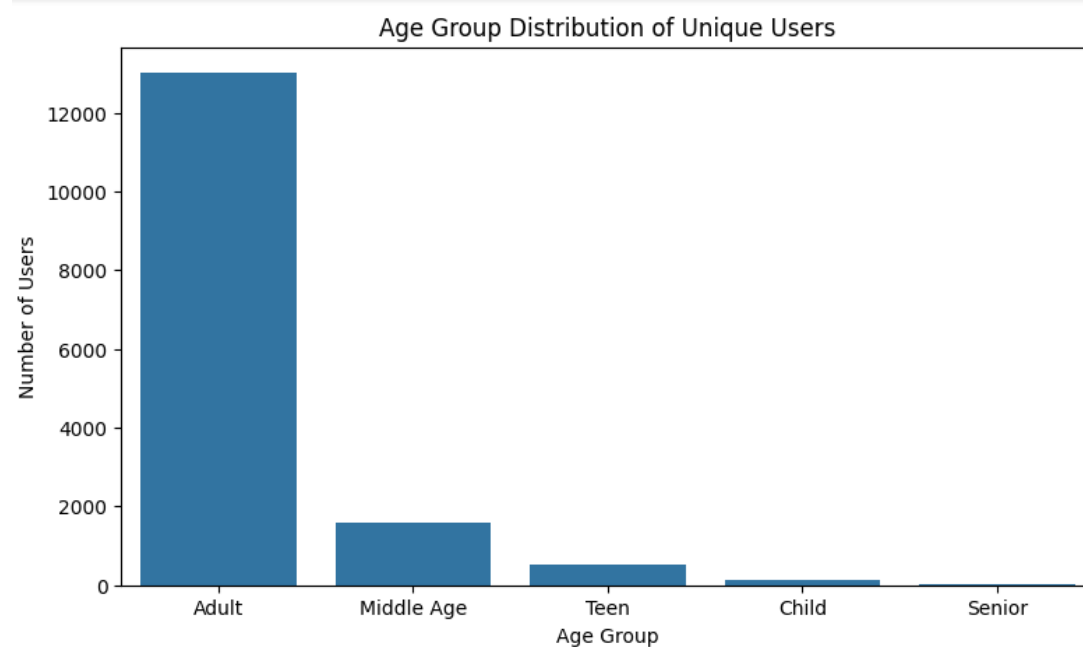
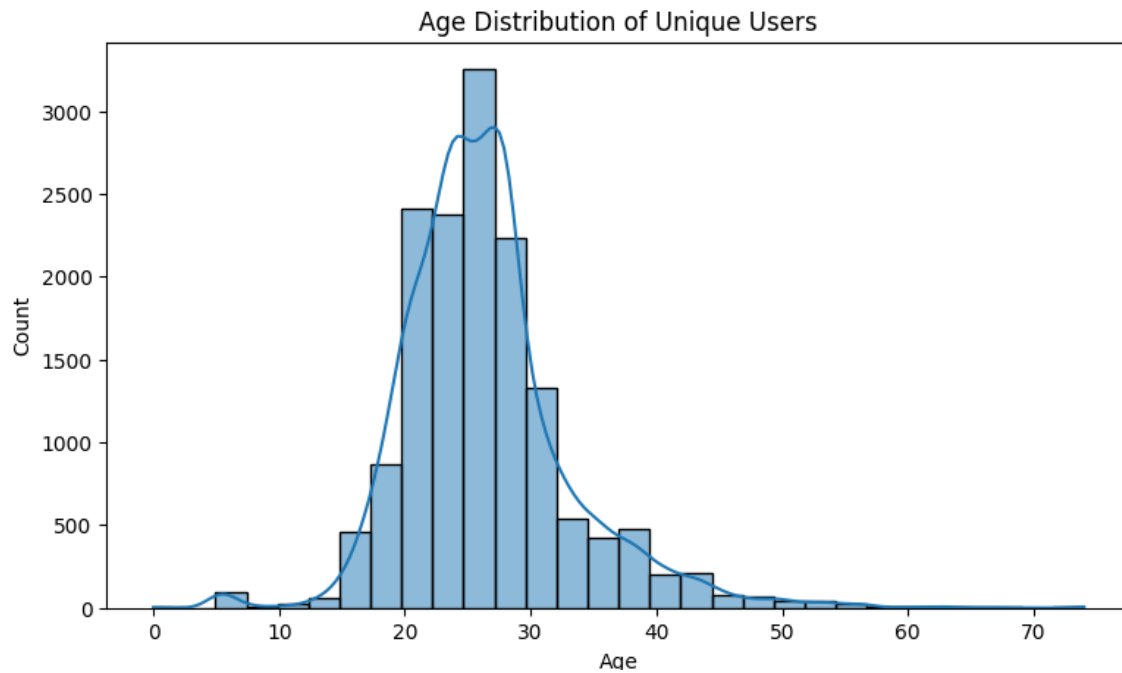
**Key Techniques:**

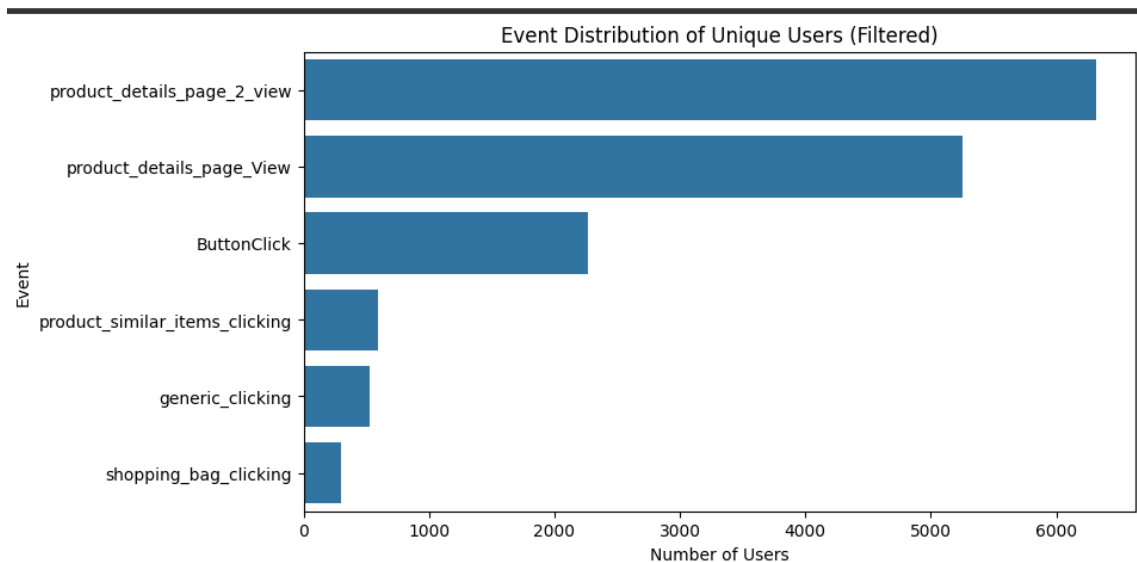
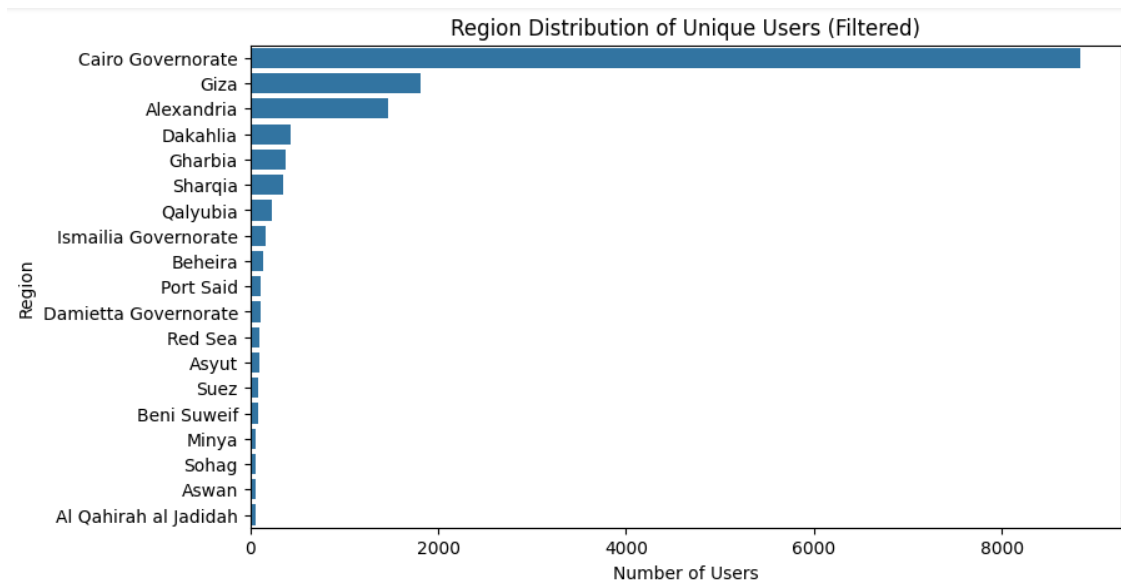
- **User-Based Collaborative Filtering**
- **Item-Based Collaborative Filtering**
- **Popularity-Based Recommendations**
- **Interaction Logging**



Each technique is chosen to address specific user needs, ensuring a comprehensive approach to personalized product recommendations.

Visualizations:



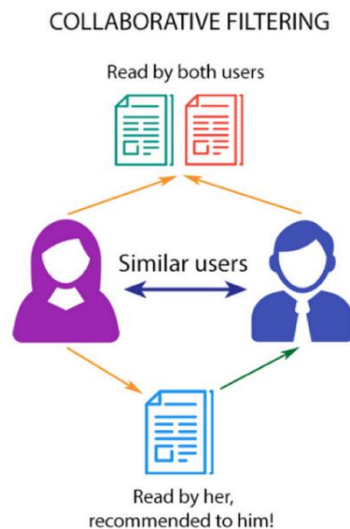


## Model 3 User personalization model

### Model three: User-User Personalization Model

#### Description

This model uses collaborative filtering based on direct user interaction data to provide personalized item recommendations. It focuses on identifying similar users based on their interaction patterns with items and then recommending items based on the preferences of these similar users.



## Methods for User Personalization that we used

### User-Based Collaborative Filtering Using Interaction Data to Calculate Similarity

#### Overview

User-Based Collaborative Filtering is a recommendation system approach that identifies users with similar interaction patterns and leverages their preferences to generate personalized recommendations. This method directly uses user interaction data, such as events, clicks to compute the similarity between users. Based on these similarities, the system recommends items that similar users have interacted with, but the target user has not yet experienced.

#### Components

##### 1. Interaction Data:

- User-item interaction dataset.: matrix where rows represent users, items, interaction strength.

##### 2. Similarity Calculation:

- **Method:** Cosine similarity between users' interaction vectors.

- **Process:**
  - Construct a user-item interaction matrix.
  - For each pair of users, calculate the cosine similarity between their interaction vectors. The cosine similarity measures the cosine of the angle between two non-zero vectors, which provides a measure of how similar the users are based on their interactions.

Store these similarities in a user-user similarity matrix, where each entry  $(i, j)$  represents the similarity between user  $i$  and user  $j$ .

### 3. Recommendation Generation:

- **Identify Similar Users:** For a given user, find the top  $N$  users with the highest similarity scores.
- **Aggregate Preferences:** Combine the interaction data of these similar users.
- **Exclude Known Interactions:** Remove items the target user has already interacted with.
- **Rank and Recommend:** Rank the remaining items based on aggregated interaction strengths and recommend the top items.

#### Advantages:

- Simple and easy to implement.
- Effective with rich interaction data.
- No need for complex models.

#### Disadvantages:

- Computationally expensive with large datasets.
- Cold start problem for new users/items.
- Limited to interaction data, lacking semantic understanding.

## Similarity Based on Embeddings

**Overview:** User-Based Collaborative Filtering using embeddings leverages advanced models (transformer) to create embeddings of users and items. These embeddings capture complex relationships and similarities in a high-dimensional space. By computing similarity between user embeddings, this approach generates personalized recommendations based on the preferences of similar users.

## Components

### 1. Embeddings Generation:

- **Source:** Item descriptions, titles
- **Models** like CLIP Fashion (same as model 2) can be used to generate embeddings for items based on their descriptions or titles.

### 2. User Embeddings Calculation:

- **Process:** Compute user embeddings as the average of the embeddings of the items they have interacted with.

### 3. Similarity Calculation:

- **Method:** Cosine similarity between user embeddings.
- **Matrix:** Create a similarity matrix where each entry represents the cosine similarity between two users based on their embeddings.

### 4. Recommendation Generation:

- **Identify Similar Users:** For a given user, find the top N users with the highest similarity scores using the similarity matrix.
- **Aggregate Preferences:** Combine the interaction data from these similar users.
- **Exclude Known Interactions:** Remove items the target user has already interacted with.
- **Rank and Recommend:** Rank the remaining items based on aggregated interaction strengths and recommend the top items.

### Advantages:

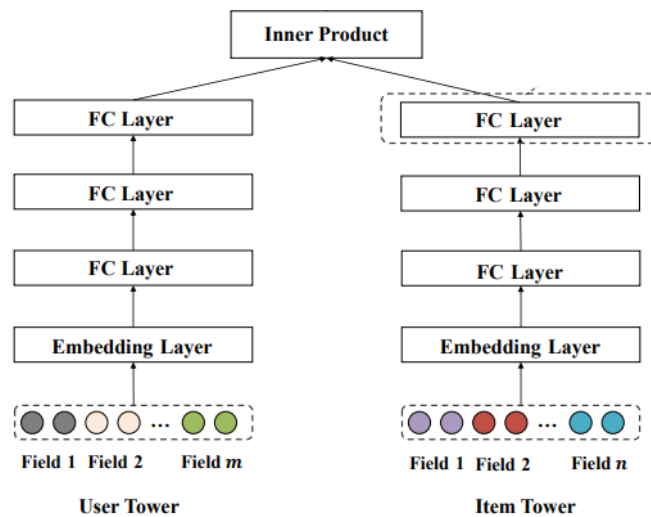
- Captures semantic relationships between items.
- Often yields better recommendations.
- Flexible with various feature types.

### Disadvantages:

- Complex and resource-intensive.
- Depends on quality of embeddings.
- Requires substantial data for meaningful embeddings

## Two-Tower Model

**Overview:** The Two-Tower Model, also known as the Dual-Tower Architecture, This model leverages deep learning to learn separate representations for users and items, which are then combined to generate personalized recommendations. The two towers—one for users and one for items—allow the model to capture complex patterns and interactions between users and items.



### Components:

#### 1. User Tower:

- **Purpose:** Processes user-specific data to create user embeddings.
- **Input:** User features, such as historical interactions, or preferences.
- **Architecture:** Typically consists of several layers of neural networks that transform user input data into a dense vector representation (embedding).

#### 2. Item Tower:

- **Purpose:** Processes item-specific data to create item embeddings.
- **Input:** Item features, such as metadata.
- **Architecture:** Similar to the user tower, it uses neural network layers to generate embeddings for items.

#### 3. Interaction Layer:

- **Purpose:** Computes the interaction between user and item embeddings to generate prediction scores.
- **Method:** Can use dot products, cosine similarity, or other methods to calculate the affinity between user and item embeddings.

- **Output:** A score that predicts the likelihood of user-item interaction or preference.

#### 4. Training:

- **Objective:** The model is trained to minimize the loss function, which measures the discrepancy between predicted interaction scores and actual interactions.
- **Loss Function:** mean squared error (MSE) for interaction strength predictions.
- **Purpose:** Trains the Two-Tower Model to optimize the embedding representations and interaction predictions.
- **Components:**

**Loss Function:** Measures the error between predicted and actual interactions (mean squared error).

**Optimizer:** Algorithm (Adam) used to update model parameters based on the loss function.

**Inputs:** Training data (user-item pairs, interaction outcomes).

**Outputs:** Trained model parameters.

#### 5-Recommendation Generation:

- **Prediction:** For a given user, compute interaction scores with all items using the trained model.
- **Ranking:** Rank items based on predicted scores.
- **Recommendation:** Recommend the top M items with the highest scores to the user.

#### Advantages:

- High accuracy with deep learning.
- Scalable to large datasets.
- Provides highly personalized recommendations.

#### Disadvantages:

- Complex and resource-heavy.
- Needs significant data for effective training.
- Risk of overfitting with complex models.

**Choosing similarity based on embeddings for user personalization is a strong choice for several reasons:**

1. **Semantic Understanding:** Embeddings capture the underlying semantic relationships between items, enabling the system to understand and recommend items based on their content and context, not just interaction history.
2. **Enhanced Relevance:** By learning detailed representations of items and users, embeddings often lead to more accurate and relevant recommendations compared to methods that rely solely on interaction data.
3. **Improved Handling of Cold Start:** While embeddings still face cold start challenges, they can somewhat mitigate the issue by leveraging item content and features to make educated guesses about new users or items.



## References:

<https://www.ibm.com/topics/apriori-algorithm> [1]

<https://www.javatpoint.com/fp-growth-algorithm-in-data-mining> [2]

<https://www.educative.io/answers/what-is-association-rule-mining> [3]

[https://www.activeloop.ai/resources/glossary/approximate-nearest-neighbors-ann/#:~:text=Approximate%20Nearest%20Neighbors%20\(ANN\)%20is,machine%20learning%2C%20and%20computer%20vision](https://www.activeloop.ai/resources/glossary/approximate-nearest-neighbors-ann/#:~:text=Approximate%20Nearest%20Neighbors%20(ANN)%20is,machine%20learning%2C%20and%20computer%20vision). [4]

<https://www.activeloop.ai/resources/glossary/annoy-approximate-nearest-neighbors-oh-yeah/> [5]

<https://arxiv.org/abs/1801.04381> [6]

<https://www.analyticsvidhya.com/blog/2023/12/what-is-mobilenetv2/> [7]

<https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db> [8]

J. Paluchasz, "Understanding OpenAI's CLIP Model," \*Medium\*, Medium, Jan. 17, 2021. [Online]. Available: <https://medium.com/@paluchasz/understanding-openais-clip-model-6b52bade3fa3>. [9]

[2] P. J. Chia, G. Attanasio, F. Bianchi, et al., "Contrastive language and vision learning of general fashion concepts," \*Sci. Rep.\*, vol. 12, no. 18958, pp. 1-13, Nov. 2022, doi: 10.1038/s41598-022-23052-9. [10]

C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li, and T. Duerig, "Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision," \*arXiv preprint arXiv:2102.01418\*, Feb. 2021. [11]