INFOSYS SPRINGBOARD INTERNSHIP
FINAL REPORT

DOMAIN: ARTIFICIAL INTELLIGENCE

PROJECT TITLE: AI STYLIST

MENTOR NAME: ANIL KUMAR SHAW

TEAM MEMBERS:

1. Monish
2. Varsini
3. Sakshi
4. Bharath
5. Eldho Elias

TABLE OF CONTENTS

1. Introduction

Abstract:

The project "AI Stylist" explores the intersection of fashion and technology through data analysis and visualization. Utilizing Python and various data manipulation libraries, the project aims to provide insights into the most prominent brands and their product distributions. This document outlines the key components of the analysis, including data preprocessing, statistical summaries, and visualization techniques, to empower data-driven decision-making in the fashion industry.

Overview of the project

The fashion industry has evolved into a dynamic space where creativity meets technology, enabling brands to deliver personalized experiences to consumers. With the rise of artificial intelligence (AI), applications like virtual stylists and personalized recommendation systems are transforming how people shop for fashion. This project explores how data analysis and visualization can uncover trends and patterns that inform the development of an AI Stylist system.

An AI Stylist leverages data to suggest products tailored to individual preferences, making the shopping experience more engaging. The effectiveness of such systems relies on analyzing product datasets to identify brand popularity, category trends, and other key insights. This study focuses on using a JSON dataset containing fashion product information to derive meaningful patterns that can enhance AI-driven recommendations.

Python libraries such as Pandas and Matplotlib are used for data manipulation and visualization. The project objectives are:

1. Exploring the dataset’s structure and understanding its attributes.
2. Identifying key trends, including brand popularity and distribution.
3. Presenting findings visually to support intuitive understanding.

Through structured data analysis and visualization, this project demonstrates how insights from fashion product data can lay the groundwork for developing intelligent, customer-focused AI solutions.

2. Data Preprocessing

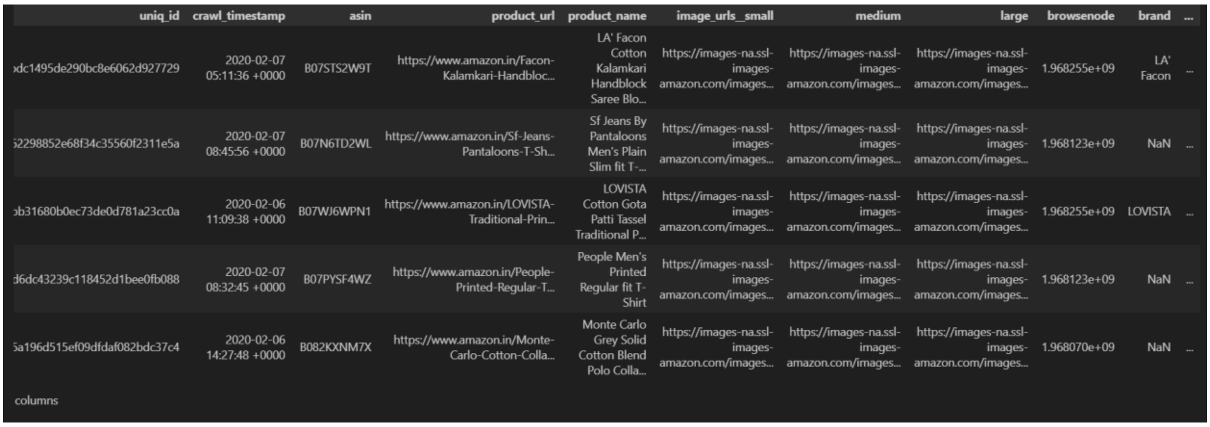
1.Data Loading and Preview:

The first step in the analysis involves loading the dataset using the Pandas library. The dataset, provided in JSON Lines format, is read into a DataFrame to facilitate data manipulation.

Code:

```
import pandas as pd
data=pd.read_json("C:/Users/sjvar/Downloads/fashion_products_data.ldjson", lines=True)
data.head()
```

This code snippet reads the dataset and displays the first few rows to provide an overview of its structure and contents. The data.head() function is particularly useful for verifying the successful import of the data.



uniq_id	crawl_timestamp	asin	product_url	product_name	image_urls__small	medium	large	browsenode	brand	...
dc1495de290bc8e6062d927729	2020-02-07 05:11:36 +0000	B07STS2W9T	https://www.amazon.in/Facon-Kalamkari-Handbloc...	LA' Facon Cotton Kalamkari Handblock Saree Blo...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	1.968255e+09	LA' Facon	...
2298852e68f34c35560f2311e5a	2020-02-07 08:45:56 +0000	B07N6TD2WL	https://www.amazon.in/Sf-Jeans-Pantaloons-T-Sh...	Sf Jeans By Pantaloons Men's Plain Slim fit T-...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	1.968123e+09	NaN	...
b31680b0ec73de0d781a23cc0a	2020-02-06 11:09:38 +0000	B07WJ6WPN1	https://www.amazon.in/LOVISTA-Traditional-Prin...	LOVISTA Cotton Gota Patti Tassel Traditional P...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	1.968255e+09	LOVISTA	...
d6dc43239c118452d1bee0fb088	2020-02-07 08:32:45 +0000	B07PYSF4WZ	https://www.amazon.in/People-Printed-Regular-T...	People Men's Printed Regular fit T-Shirt	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	1.968123e+09	NaN	...
a196d515ef09dfdaf082bdc37c4	2020-02-06 14:27:48 +0000	B082KXNM7X	https://www.amazon.in/Monte-Carlo-Cotton-Colla...	Monte Carlo Grey Solid Cotton Blend Polo Colla...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	1.968070e+09	NaN	...

columns

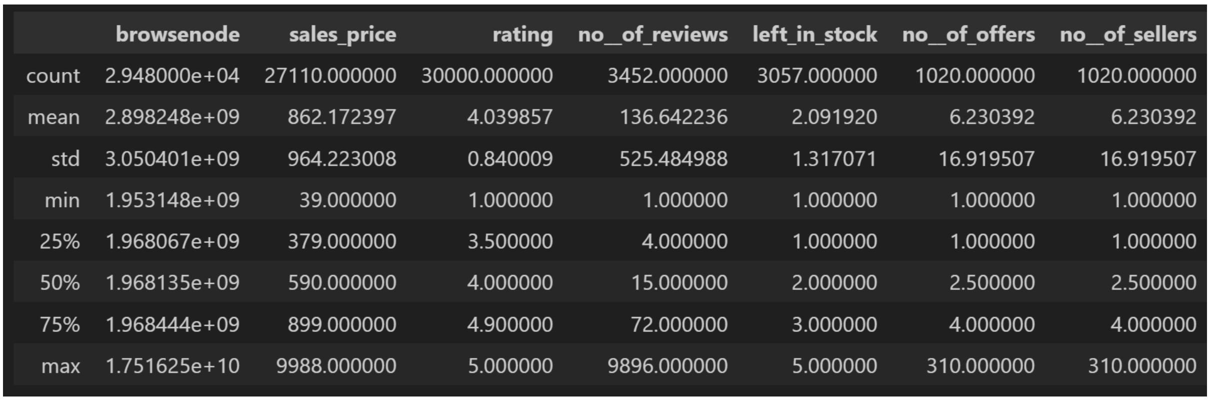
2. Descriptive Statistics:

To understand the dataset's statistical properties, we use the describe and info methods. These functions summarize the data distribution and provide details about data types and missing values.

Code:

```
data.describe()
data.info()
```

- data.describe(): Offers statistical insights such as mean, standard deviation, and quartiles for numeric columns.
- data.info(): Displays the structure of the dataset, including the number of entries, data types, and non-null counts.



	browsenode	sales_price	rating	no__of_reviews	left_in_stock	no__of_offers	no__of_sellers
count	2.948000e+04	27110.000000	30000.000000	3452.000000	3057.000000	1020.000000	1020.000000
mean	2.898248e+09	862.172397	4.039857	136.642236	2.091920	6.230392	6.230392
std	3.050401e+09	964.223008	0.840009	525.484988	1.317071	16.919507	16.919507
min	1.953148e+09	39.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	1.968067e+09	379.000000	3.500000	4.000000	1.000000	1.000000	1.000000
50%	1.968135e+09	590.000000	4.000000	15.000000	2.000000	2.500000	2.500000
75%	1.968444e+09	899.000000	4.900000	72.000000	3.000000	4.000000	4.000000
max	1.751625e+10	9988.000000	5.000000	9896.000000	5.000000	310.000000	310.000000

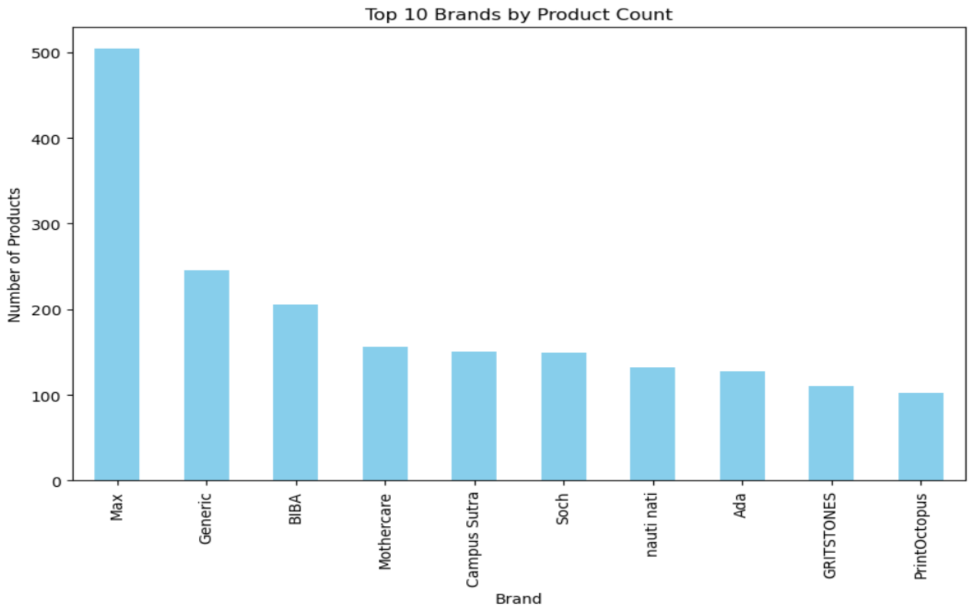
3. Data Visualization:

Top Brands by Product Count:

Visualizing data is a powerful way to uncover patterns and trends. Here, a bar chart is created to show the top 10 brands by product count.

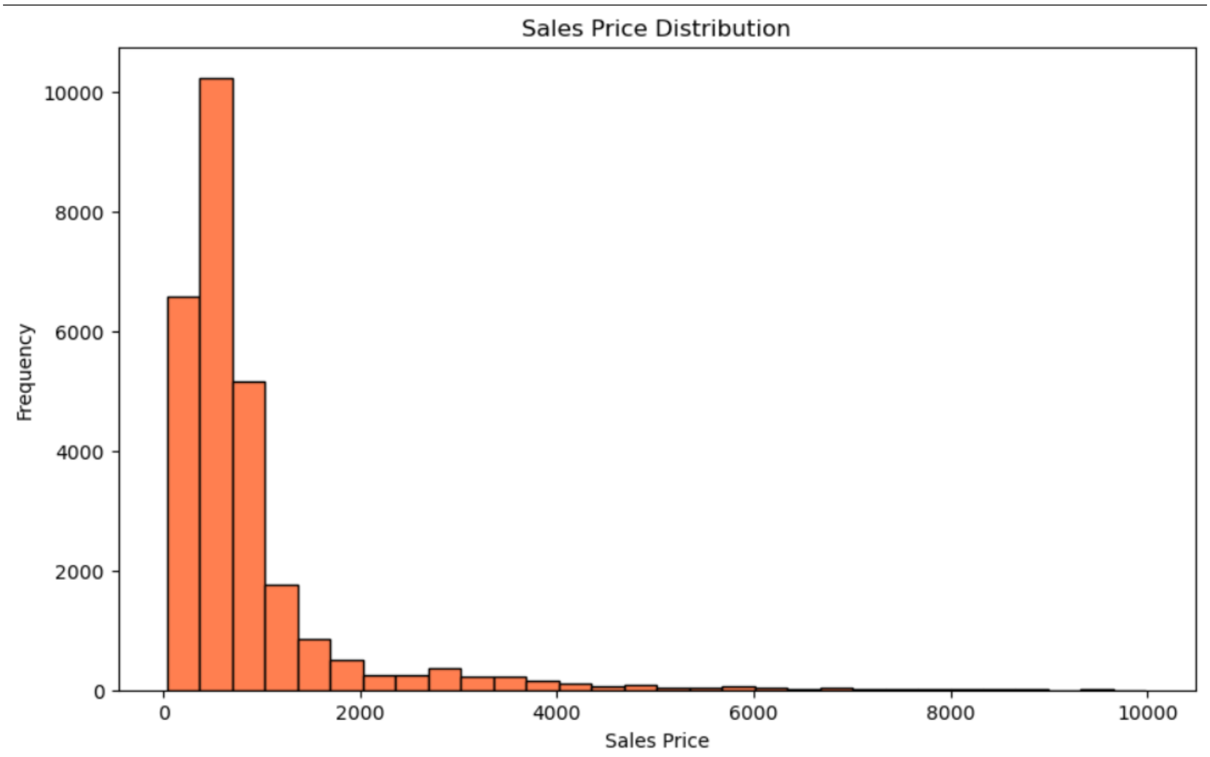
```
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))

# Count the occurrences of each brand and get the top 10
top_brands = data['brand'].value_counts().head(10)

# Create a bar chart
top_brands.plot(kind='bar', color='skyblue')

# Set titles and labels
plt.title('Top 10 Brands by Product Count')
plt.xlabel('Brand')
plt.ylabel('Number of Products')

# Show the plot
plt.show()
```

This code computes the frequency of each brand in the dataset, extracts the top 10, and plots the results using Matplotlib. The value_counts() method efficiently counts occurrences, while the plot() function generates the bar chart.



SALES PRICE DISTRIBUTION

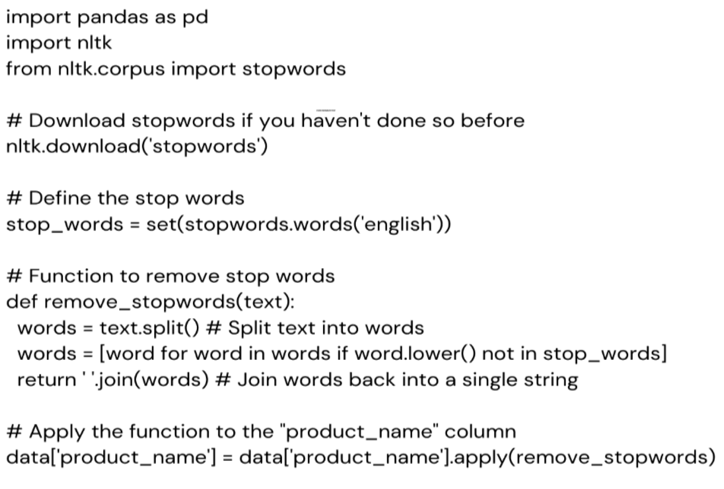
```
plt.figure(figsize=(10, 6))

# Plot histogram of sales price
data['sales_price'].dropna().plot(kind='hist', bins=30, color='coral', edgecolor='black')
plt.title('Sales Price Distribution')
plt.xlabel('Sales Price')
plt.ylabel('Frequency')
plt.show()
```



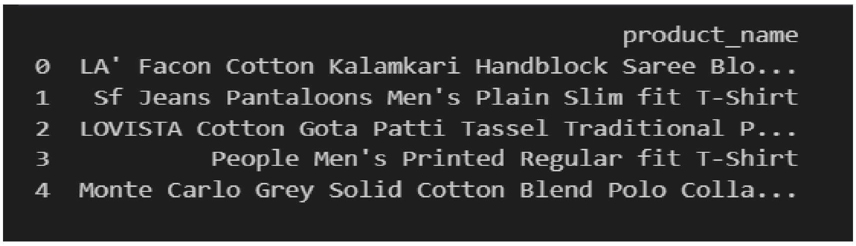
4. Text Preprocessing:

1.Why Stop Words Are Removed in Coding:

1. **Reduce Processing Load**:
   - Stop words make up a large portion of natural language but add little value to understanding the core meaning of text.
   - Removing them reduces the size of the data being processed, making algorithms faster.
2. **Enhance Model Accuracy**:
   - Models like classification or clustering focus on keywords that contribute to meaning. Removing stop words prevents them from overwhelming meaningful words.
3. **Simplify Algorithms**:
   - By removing unnecessary words, tasks such as keyword extraction or sentiment analysis become easier to compute.



```
import pandas as pd
import nltk
from nltk.corpus import stopwords

# Download stopwords if you haven't done so before
nltk.download('stopwords')

# Define the stop words
stop_words = set(stopwords.words('english'))

# Function to remove stop words
def remove_stopwords(text):
  words = text.split() # Split text into words
  words = [word for word in words if word.lower() not in stop_words]
  return ' '.join(words) # Join words back into a single string

# Apply the function to the "product_name" column
data['product_name'] = data['product_name'].apply(remove_stopwords)
```



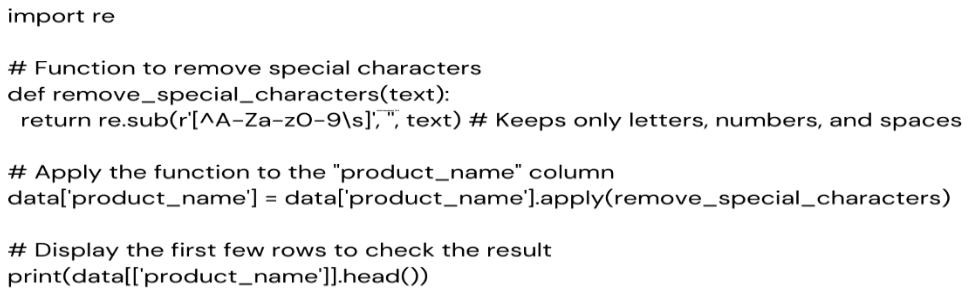
```
                                          product_name
0  LA' Facon Cotton Kalamkari Handblock Saree Blo...
1   Sf Jeans Pantaloons Men's Plain Slim fit T-Shirt
2  LOVISTA Cotton Gota Patti Tassel Traditional P...
3           People Men's Printed Regular fit T-Shirt
4  Monte Carlo Grey Solid Cotton Blend Polo Colla...
```

2.Why Remove Special Characters in Text Preprocessing?

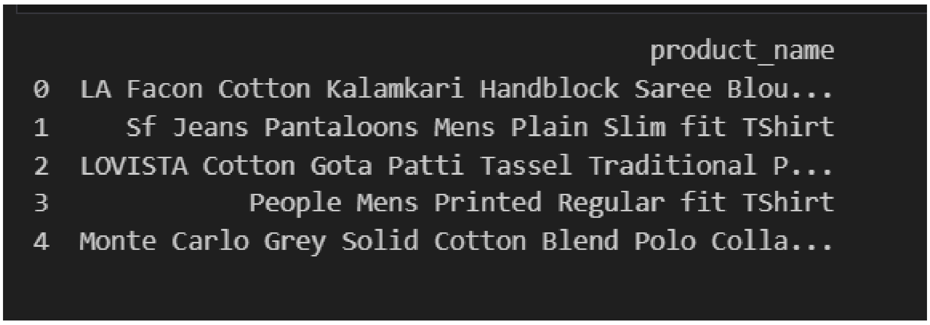
Special characters (like !, @, #, &, etc.) are often removed in text preprocessing for tasks like machine learning, natural language processing, or text analytics. Here's why:

Key Reasons:

1. **Noise Reduction**:
   - Special characters often don't add meaningful information.
   - For example, smart-phone! and smartphone mean the same thing, so the - and ! are unnecessary.
2. **Standardization**:
   - Helps create uniform text data by eliminating inconsistent symbols that may interfere with downstream processing.
3. **Avoid Errors**:
   - Special characters can cause issues in tokenization, feature extraction, or when working with regular expressions and other string operations.



```
import re

# Function to remove special characters
def remove_special_characters(text):
  return re.sub(r'[^A-Za-z0-9\s]', '', text) # Keeps only letters, numbers, and spaces

# Apply the function to the "product_name" column
data['product_name'] = data['product_name'].apply(remove_special_characters)

# Display the first few rows to check the result
print(data[['product_name']].head())
```



```
                                     product_name
0  LA Facon Cotton Kalamkari Handblock Saree Blou...
1     Sf Jeans Pantaloons Mens Plain Slim fit TShirt
2  LOVISTA Cotton Gota Patti Tassel Traditional P...
3             People Mens Printed Regular fit TShirt
4  Monte Carlo Grey Solid Cotton Blend Polo Colla...
```

3. Purpose of Removing Non-English Words or Text

In text preprocessing, filtering out non-English content is essential for tasks that involve analyzing English-only data. Here's why:

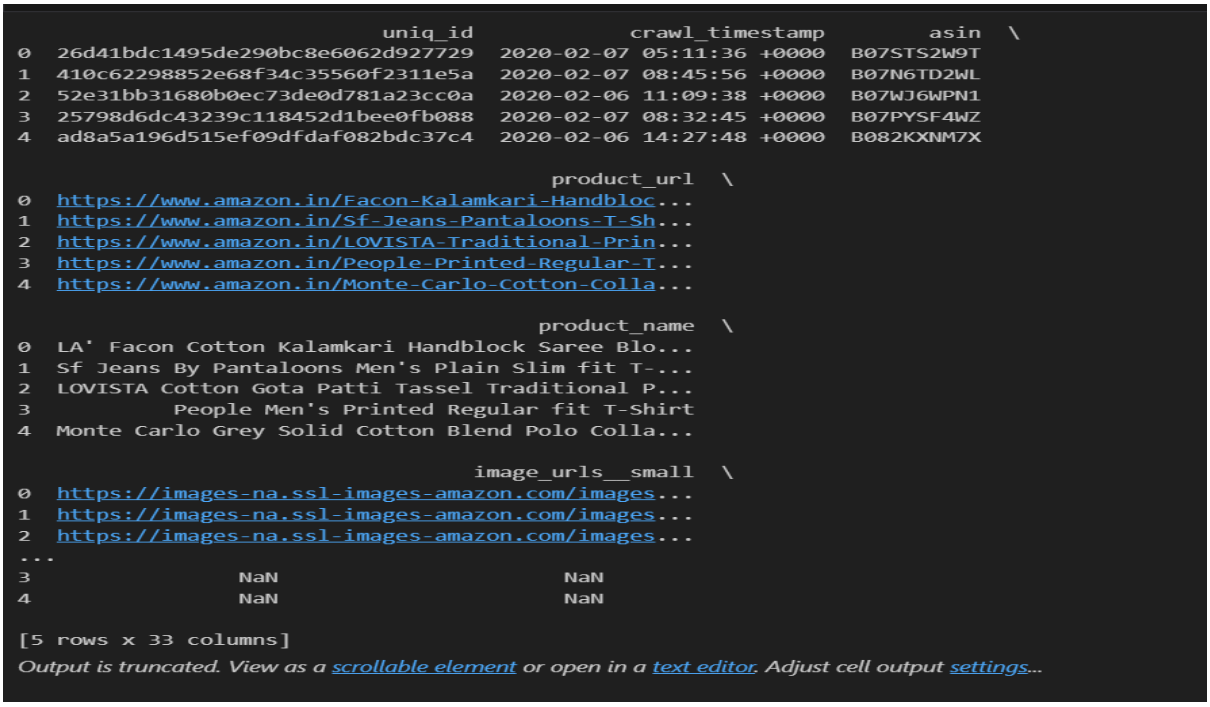
Why Remove Non-English Words/Text?

1. **Focus on Target Language**:
   - If the analysis is specific to English (e.g., keyword analysis, sentiment detection, or recommendation systems for English-speaking users), non-English words may introduce irrelevant data.

2. **Reduce Noise**:
   - Mixed-language content can confuse models, making it harder to extract meaningful patterns.

3. **Simplify Preprocessing**:
   - Ensures uniformity in the dataset, avoiding the need for additional steps like language detection or multilingual processing.
4. **Optimize Model Performance**:
   - Language-specific models (e.g., English sentiment analysis) perform best when trained and tested on text in the same language.
5. **Improved User Experience**:
   - For applications like search engines, chatbots, or e-commerce platforms catering to English speakers, excluding non-English content enhances relevance.

```
import pandas as pd
import re

# Load the JSON data into a DataFrame
# Replace the path with the actual file path to your JSON file
data = pd.read_json("C:/Users/sjvar/Downloads/fashion_products_data.ldjson", lines=True)

# Function to filter out non-English text based on ASCII characters
def remove_non_english(text):
 return text if re.match(r'^[\x00-\x7F]+$', text) else None

# Apply the function to the product_name column
data['product_name'] = data['product_name'].apply(remove_non_english)

# Drop rows with None in product_name column (non-English rows)
data = data.dropna(subset=['product_name'])

# Display the cleaned data
print(data.head())
```

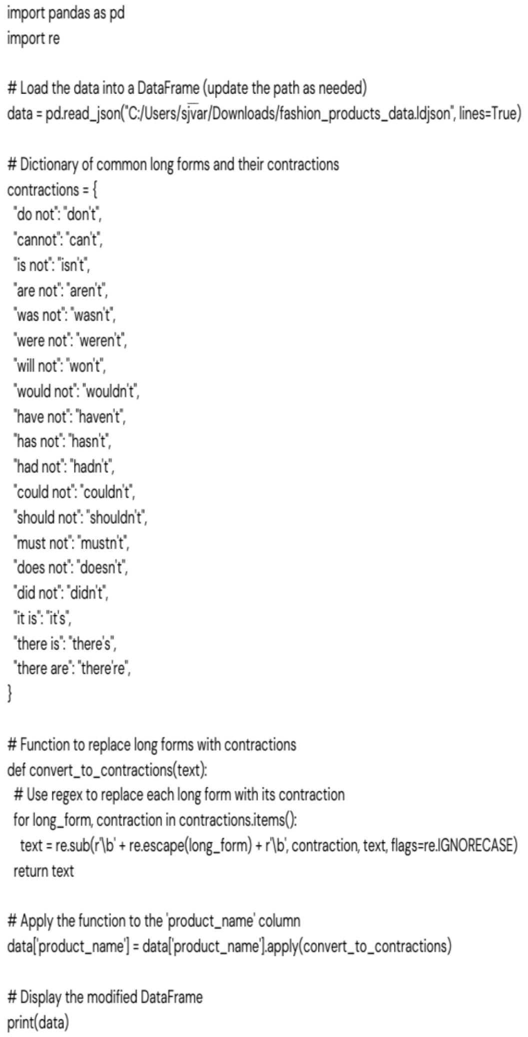


```
                            uniq_id            crawl_timestamp        asin  \
0  26d41bdc1495de290bc8e6062d927729  2020-02-07 05:11:36 +0000  B07STS2W9T
1  410c62298852e68f34c35560f2311e5a  2020-02-07 08:45:56 +0000  B07N6TD2WL
2  52e31bb31680b0ec73de0d781a23cc0a  2020-02-06 11:09:38 +0000  B07WJ6WPN1
3  25798d6dc43239c118452d1bee0fb088  2020-02-07 08:32:45 +0000  B07PYSF4WZ
4  ad8a5a196d515ef09dfdaf082bdc37c4  2020-02-06 14:27:48 +0000  B082KXNM7X

                                         product_url  \
0  https://www.amazon.in/Facon-Kalamkari-Handbloc...
1  https://www.amazon.in/Sf-Jeans-Pantaloons-T-Sh...
2  https://www.amazon.in/LOVISTA-Traditional-Prin...
3  https://www.amazon.in/People-Printed-Regular-T...
4  https://www.amazon.in/Monte-Carlo-Cotton-Colla...

                                        product_name  \
0  LA' Facon Cotton Kalamkari Handblock Saree Blo...
1  Sf Jeans By Pantaloons Men's Plain Slim fit T-...
2  LOVISTA Cotton Gota Patti Tassel Traditional P...
3           People Men's Printed Regular fit T-Shirt
4  Monte Carlo Grey Solid Cotton Blend Polo Colla...

                                   image_urls__small  \
0  https://images-na.ssl-images-amazon.com/images...
1  https://images-na.ssl-images-amazon.com/images...
2  https://images-na.ssl-images-amazon.com/images...
...
3                NaN                       NaN
4                NaN                       NaN

[5 rows x 33 columns]
```

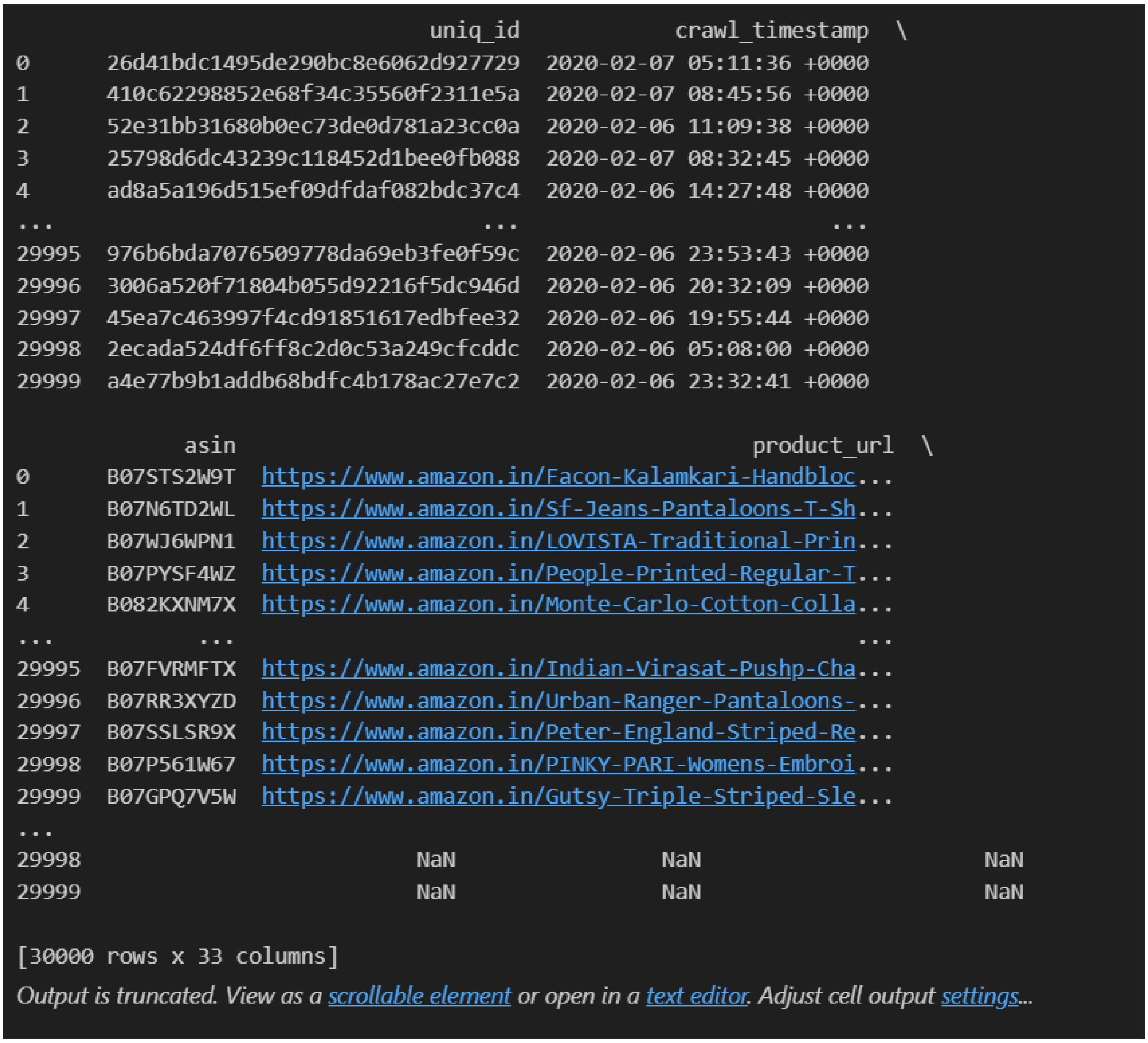
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

4.Why Use Contractions?

1. **Consistency**:
   - Standardizes text by replacing multiple representations of the same meaning (e.g., "do not" and "don't") with a single, consistent form.
2. **Alignment with Common Usage**:
   - Contractions are often used in casual text like product names, reviews, or social media content. Handling them improves relevance and user understanding.
3. **Improved Text Analysis**:
   - In tasks like sentiment analysis or language modeling, consistent contraction usage ensures better feature representation.



```
import pandas as pd
import re

# Load the data into a DataFrame (update the path as needed)
data = pd.read_json("C:/Users/sjvar/Downloads/fashion_products_data.ldjson", lines=True)

# Dictionary of common long forms and their contractions
contractions = {
  "do not": "don't",
  "cannot": "can't",
  "is not": "isn't",
  "are not": "aren't",
  "was not": "wasn't",
  "were not": "weren't",
  "will not": "won't",
  "would not": "wouldn't",
  "have not": "haven't",
  "has not": "hasn't",
  "had not": "hadn't",
  "could not": "couldn't",
  "should not": "shouldn't",
  "must not": "mustn't",
  "does not": "doesn't",
  "did not": "didn't",
  "it is": "it's",
  "there is": "there's",
  "there are": "there're",
}

# Function to replace long forms with contractions
def convert_to_contractions(text):
  # Use regex to replace each long form with its contraction
  for long_form, contraction in contractions.items():
    text = re.sub(r'\b' + re.escape(long_form) + r'\b', contraction, text, flags=re.IGNORECASE)
  return text

# Apply the function to the 'product_name' column
data['product_name'] = data['product_name'].apply(convert_to_contractions)

# Display the modified DataFrame
print(data)
```



```
                                 uniq_id            crawl_timestamp  \
0      26d41bdc1495de290bc8e6062d927729  2020-02-07 05:11:36 +0000
1      410c62298852e68f34c35560f2311e5a  2020-02-07 08:45:56 +0000
2      52e31bb31680b0ec73de0d781a23cc0a  2020-02-06 11:09:38 +0000
3      25798d6dc43239c118452d1bee0fb088  2020-02-07 08:32:45 +0000
4      ad8a5a196d515ef09dfdaf082bdc37c4  2020-02-06 14:27:48 +0000
...                                 ...                        ...
29995  976b6bda7076509778da69eb3fe0f59c  2020-02-06 23:53:43 +0000
29996  3006a520f71804b055d92216f5dc946d  2020-02-06 20:32:09 +0000
29997  45ea7c463997f4cd91851617edbfee32  2020-02-06 19:55:44 +0000
29998  2ecada524df6ff8c2d0c53a249cfcddc  2020-02-06 05:08:00 +0000
29999  a4e77b9b1addb68bdfc4b178ac27e7c2  2020-02-06 23:32:41 +0000

             asin                                        product_url  \
0      B07STS2W9T  https://www.amazon.in/Facon-Kalamkari-Handbloc...
1      B07N6TD2WL  https://www.amazon.in/Sf-Jeans-Pantaloons-T-Sh...
2      B07WJ6WPN1  https://www.amazon.in/LOVISTA-Traditional-Prin...
3      B07PYSF4WZ  https://www.amazon.in/People-Printed-Regular-T...
4      B082KXNM7X  https://www.amazon.in/Monte-Carlo-Cotton-Colla...
...           ...                                                ...
29995  B07FVRMFTX  https://www.amazon.in/Indian-Virasat-Pushp-Cha...
29996  B07RR3XYZD  https://www.amazon.in/Urban-Ranger-Pantaloons-...
29997  B07SSLSR9X  https://www.amazon.in/Peter-England-Striped-Re...
29998  B07P561W67  https://www.amazon.in/PINKY-PARI-Womens-Embroi...
29999  B07GPQ7V5W  https://www.amazon.in/Gutsy-Triple-Striped-Sle...
...
29998                          NaN                  NaN                      NaN
29999                          NaN                  NaN                      NaN

[30000 rows x 33 columns]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

5.Lemmatization

Lemmatization is a natural language processing (NLP) technique used to reduce words to their base or root form, called the **lemma**. Unlike stemming, which often chops off word endings arbitrarily, lemmatization takes into account the word's context and ensures that the reduced form is a meaningful word in the language.

For example:

- The words **"running"**, **"ran"**, and **"runs"** would all be lemmatized to **"run"**.
- The word **"better"** might be lemmatized to its root form **"good"**, depending on the part of speech.

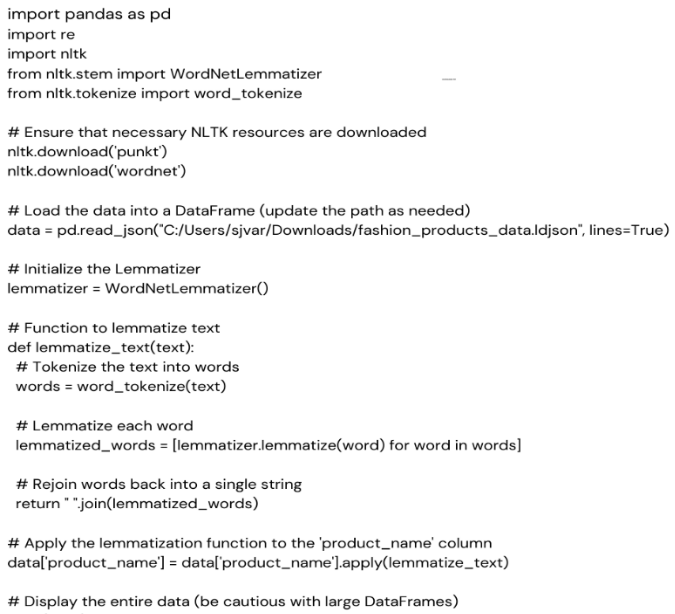
Why Use Lemmatization?

- To normalize words for better text analysis or preprocessing in NLP tasks.
- To reduce redundancy and variability in text data while retaining meaningful context.
- Helps improve the performance of tasks like sentiment analysis, information retrieval, and text classification.

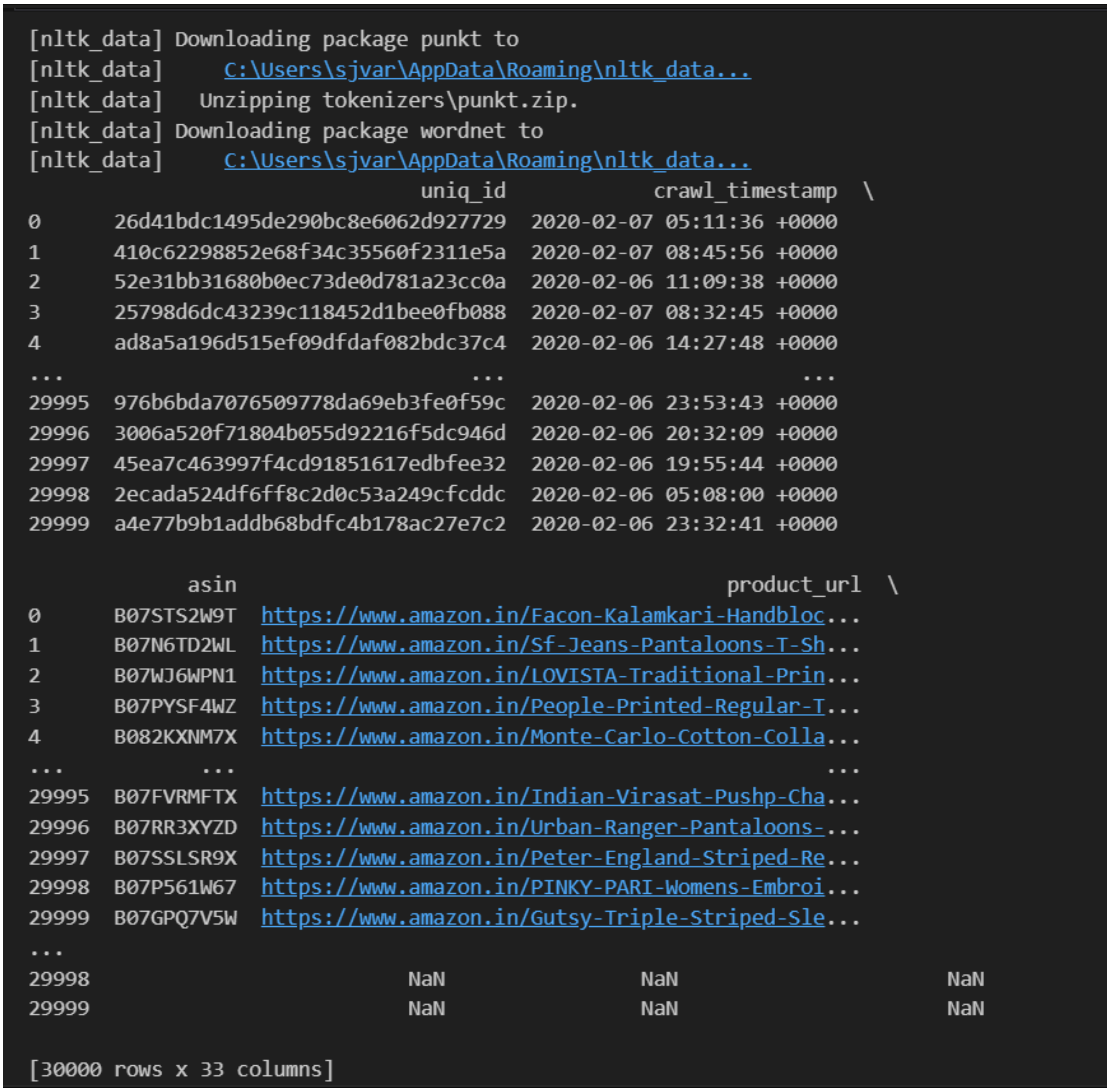
In the provided code:

1. **WordNetLemmatizer** is used to perform lemmatization.
2. Each word in the product_name column is tokenized and lemmatized individually.
3. Lemmatized words are rejoined into a single string and stored back in the column.

This process ensures that product names are standardized and easier to analyze, potentially reducing variability caused by different word forms.



```
import pandas as pd
import re
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Ensure that necessary NLTK resources are downloaded
nltk.download('punkt')
nltk.download('wordnet')

# Load the data into a DataFrame (update the path as needed)
data = pd.read_json("C:/Users/sjvar/Downloads/fashion_products_data.ldjson", lines=True)

# Initialize the Lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to lemmatize text
def lemmatize_text(text):
  # Tokenize the text into words
  words = word_tokenize(text)

  # Lemmatize each word
  lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

  # Rejoin words back into a single string
  return " ".join(lemmatized_words)

# Apply the lemmatization function to the 'product_name' column
data['product_name'] = data['product_name'].apply(lemmatize_text)

# Display the entire data (be cautious with large DataFrames)
print(data)
```



```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sjvar\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\sjvar\AppData\Roaming\nltk_data...
                                uniq_id            crawl_timestamp  \
0      26d41bdc1495de290bc8e6062d927729  2020-02-07 05:11:36 +0000
1      410c62298852e68f34c35560f2311e5a  2020-02-07 08:45:56 +0000
2      52e31bb31680b0ec73de0d781a23cc0a  2020-02-06 11:09:38 +0000
3      25798d6dc43239c118452d1bee0fb088  2020-02-07 08:32:45 +0000
4      ad8a5a196d515ef09dfdaf082bdc37c4  2020-02-06 14:27:48 +0000
...                                 ...                        ...
29995  976b6bda7076509778da69eb3fe0f59c  2020-02-06 23:53:43 +0000
29996  3006a520f71804b055d92216f5dc946d  2020-02-06 20:32:09 +0000
29997  45ea7c463997f4cd91851617edbfee32  2020-02-06 19:55:44 +0000
29998  2ecada524df6ff8c2d0c53a249cfcddc  2020-02-06 05:08:00 +0000
29999  a4e77b9b1addb68bdfc4b178ac27e7c2  2020-02-06 23:32:41 +0000

             asin                                product_url  \
0      B07STS2W9T  https://www.amazon.in/Facon-Kalamkari-Handbloc...
1      B07N6TD2WL  https://www.amazon.in/Sf-Jeans-Pantaloons-T-Sh...
2      B07WJ6WPN1  https://www.amazon.in/LOVISTA-Traditional-Prin...
3      B07PYSF4WZ  https://www.amazon.in/People-Printed-Regular-T...
4      B082KXNM7X  https://www.amazon.in/Monte-Carlo-Cotton-Colla...
...           ...                                                ...
29995  B07FVRMFTX  https://www.amazon.in/Indian-Virasat-Pushp-Cha...
29996  B07RR3XYZD  https://www.amazon.in/Urban-Ranger-Pantaloons-...
29997  B07SSLSR9X  https://www.amazon.in/Peter-England-Striped-Re...
29998  B07P561W67  https://www.amazon.in/PINKY-PARI-Womens-Embroi...
29999  B07GPQ7V5W  https://www.amazon.in/Gutsy-Triple-Striped-Sle...
...
29998                          NaN                NaN                         NaN
29999                          NaN                NaN                         NaN

[30000 rows x 33 columns]
```

5 Content Based Filtering:

Overview of Recommendation Methods

1. Bag of Words (BoW)
2. TF-IDF (Term Frequency-Inverse Document Frequency)
3. Word2Vec

Bag of Words (BoW):

What is Bag of Words?
Bag of Words (BoW) is a natural language processing (NLP) technique used to represent text data as numerical features. It involves converting textual descriptions or documents into fixed-length vectors based on the frequency of each word in the text.

In the context of collaborative filtering, BoW helps to extract features from product descriptions (e.g., names, specifications) and use these features for similarity-based recommendations.

Why Use Bag of Words?

- **Simplicity:** BoW is easy to implement and understand.
- **Textual Feature Extraction:** Useful when product data includes textual descriptions.
- **Versatility:** Works well when paired with similarity measures like cosine similarity to find similar products.

How Bag of Words Works

1. **Tokenization**: Splits text into individual words or tokens.
2. **Vocabulary Creation**: Creates a unique set of words from the corpus.
3. **Vectorization**: Represents each document as a fixed-length vector based on word frequency or presence.

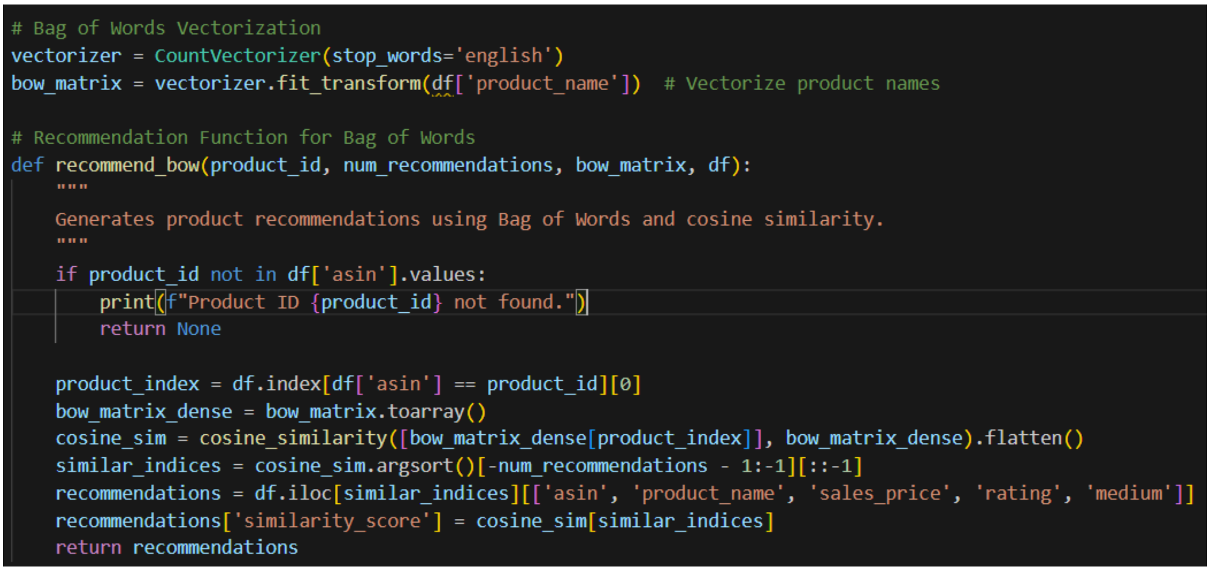
Advantages of Bag of Words

1. **Simple and Easy**: Easy to implement and understand.
2. **Effective Feature Extraction**: Useful for basic text analysis and recommendation systems.
3. **Versatile**: Works well with similarity measures like cosine similarity.

Limitations of Bag of Words

1. **Loses Context**: Ignores word order and semantics.
2. **High Dimensionality**: Requires large memory for extensive vocabularies.
3. **Sparse Data**: Results in sparse vectors, complicating computations.

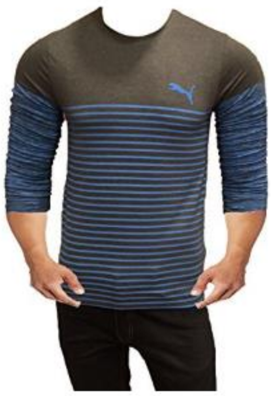
Code:



```
# Bag of Words Vectorization
vectorizer = CountVectorizer(stop_words='english')
bow_matrix = vectorizer.fit_transform(df['product_name'])  # Vectorize product names

# Recommendation Function for Bag of Words
def recommend_bow(product_id, num_recommendations, bow_matrix, df):
    """
    Generates product recommendations using Bag of Words and cosine similarity.
    """
    if product_id not in df['asin'].values:
        print(f"Product ID {product_id} not found.")
        return None

    product_index = df.index[df['asin'] == product_id][0]
    bow_matrix_dense = bow_matrix.toarray()
    cosine_sim = cosine_similarity([bow_matrix_dense[product_index]], bow_matrix_dense).flatten()
    similar_indices = cosine_sim.argsort()[-num_recommendations - 1:-1][::-1]
    recommendations = df.iloc[similar_indices][['asin', 'product_name', 'sales_price', 'rating', 'medium']]
    recommendations['similarity_score'] = cosine_sim[similar_indices]
    return recommendations
```

Output:

Query Image:



Recommendations:

TF-IDF (Term Frequency-Inverse Document Frequency)

Definition:
TF-IDF is a statistical technique used to measure the importance of a word in a document relative to a collection of documents (corpus). It helps to identify unique terms that are more relevant to a specific document while ignoring common terms across the corpus.

How It Works:

- **Term Frequency (TF):** Measures how often a word appears in a document.
- **Inverse Document Frequency (IDF):** Reduces the weight of common words across documents.

Why Use TF-IDF in Recommendations?

- Highlights unique words in product descriptions or reviews.
- Reduces noise from common words like "the" or "and."
- Improves similarity-based recommendations by providing better weightings.

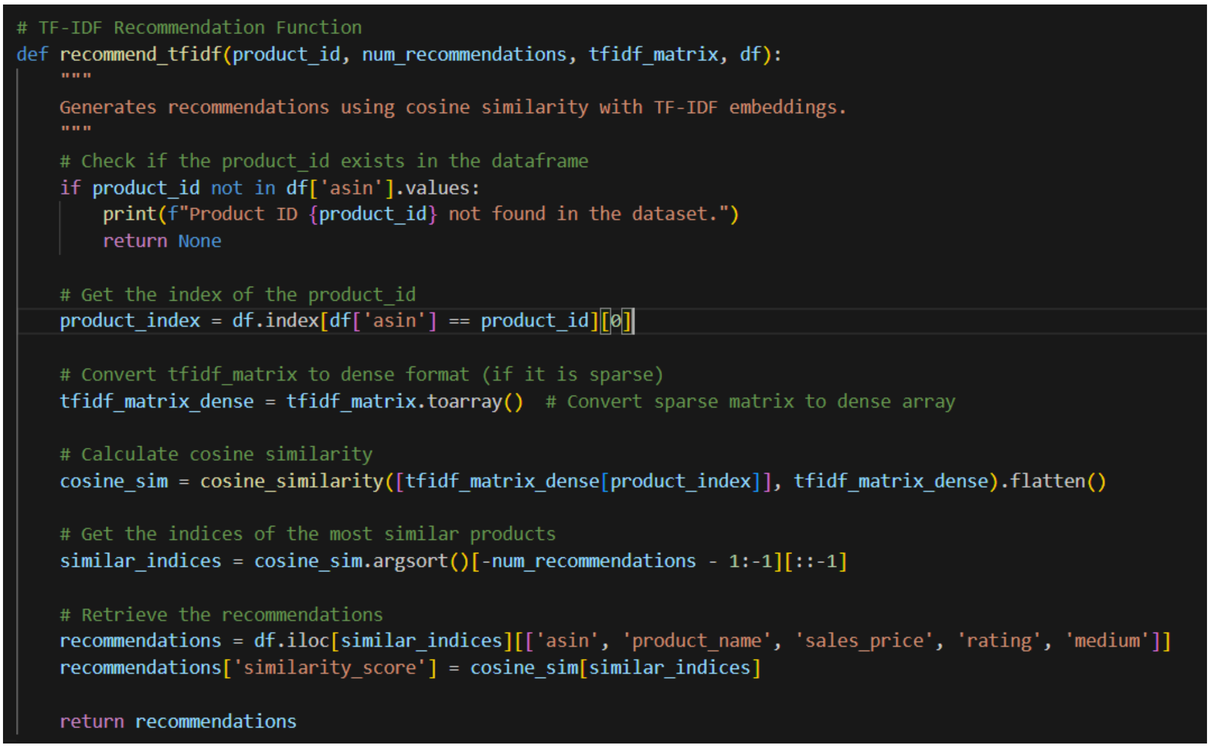
Advantages:

1. Balances word frequency and uniqueness.
2. Effective for text-heavy datasets.
3. Easy to implement and interpret.

Limitations:

1. Cannot capture the semantic meaning of words.
2. Computationally expensive for large datasets.

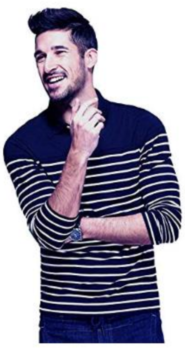
Code:



```python
# TF-IDF Recommendation Function
def recommend_tfidf(product_id, num_recommendations, tfidf_matrix, df):
    """
    Generates recommendations using cosine similarity with TF-IDF embeddings.
    """
    # Check if the product_id exists in the dataframe
    if product_id not in df['asin'].values:
        print(f"Product ID {product_id} not found in the dataset.")
        return None

    # Get the index of the product_id
    product_index = df.index[df['asin'] == product_id][0]

    # Convert tfidf_matrix to dense format (if it is sparse)
    tfidf_matrix_dense = tfidf_matrix.toarray()  # Convert sparse matrix to dense array

    # Calculate cosine similarity
    cosine_sim = cosine_similarity([tfidf_matrix_dense[product_index]], tfidf_matrix_dense).flatten()

    # Get the indices of the most similar products
    similar_indices = cosine_sim.argsort()[-num_recommendations - 1:-1][::-1]

    # Retrieve the recommendations
    recommendations = df.iloc[similar_indices][['asin', 'product_name', 'sales_price', 'rating', 'medium']]
    recommendations['similarity_score'] = cosine_sim[similar_indices]

    return recommendations
```

Output:

Query Image:



Recommendations

Word2Vec:

Definition:
Word2Vec is a word embedding technique that represents words as dense numerical vectors in a multi-dimensional space, capturing their semantic relationships. Developed using neural networks, Word2Vec learns the context of words based on their usage in a given text corpus.

How Word2Vec Works:

1. **Neural Network Training:**
   - Word2Vec uses either a **Skip-Gram** or **Continuous Bag of Words (CBOW)** model.
     - **Skip-Gram:** Predicts the context words from a target word.
     - **CBOW:** Predicts the target word from its surrounding context words.
2. **Vector Representation:**
   - Each word is mapped to a vector in a continuous vector space.
   - Words with similar meanings or contexts have vectors close to each other in this space.

Why Use Word2Vec in Recommendations?

- Captures semantic relationships (e.g., "king" and "queen" are close in vector space).
- Handles synonyms and context effectively.
- Provides a compact and meaningful representation of textual data.

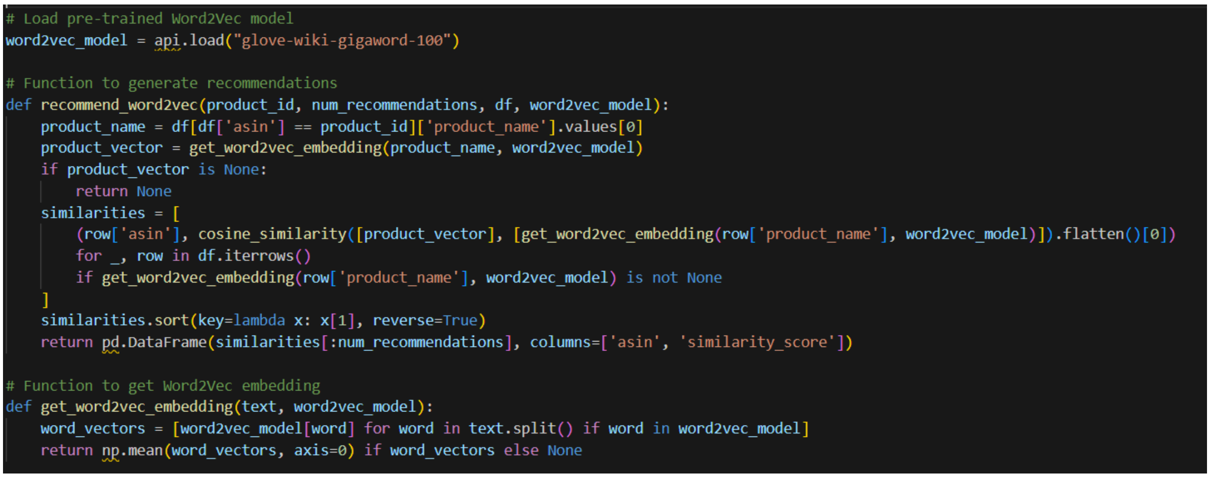
Advantages of Word2Vec:

1. Learns semantic and syntactic relationships.
2. Generates dense and compact word embeddings.
3. Handles large text datasets effectively.

Limitations:

1. Requires a large dataset for training to perform well.
2. Computationally intensive compared to simpler techniques like TF-IDF.

Code:



```
# Load pre-trained Word2Vec model
word2vec_model = api.load("glove-wiki-gigaword-100")

# Function to generate recommendations
def recommend_word2vec(product_id, num_recommendations, df, word2vec_model):
    product_name = df[df['asin'] == product_id]['product_name'].values[0]
    product_vector = get_word2vec_embedding(product_name, word2vec_model)
    if product_vector is None:
        return None
    similarities = [
        (row['asin'], cosine_similarity([product_vector], [get_word2vec_embedding(row['product_name'], word2vec_model)]).flatten()[0])
        for _, row in df.iterrows()
        if get_word2vec_embedding(row['product_name'], word2vec_model) is not None
    ]
    similarities.sort(key=lambda x: x[1], reverse=True)
    return pd.DataFrame(similarities[:num_recommendations], columns=['asin', 'similarity_score'])

# Function to get Word2Vec embedding
def get_word2vec_embedding(text, word2vec_model):
    word_vectors = [word2vec_model[word] for word in text.split() if word in word2vec_model]
    return np.mean(word_vectors, axis=0) if word_vectors else None
```

Output:

Query Image

Recommendations:1

Hybrid Recommendation System Using TF-IDF and Word2Vec

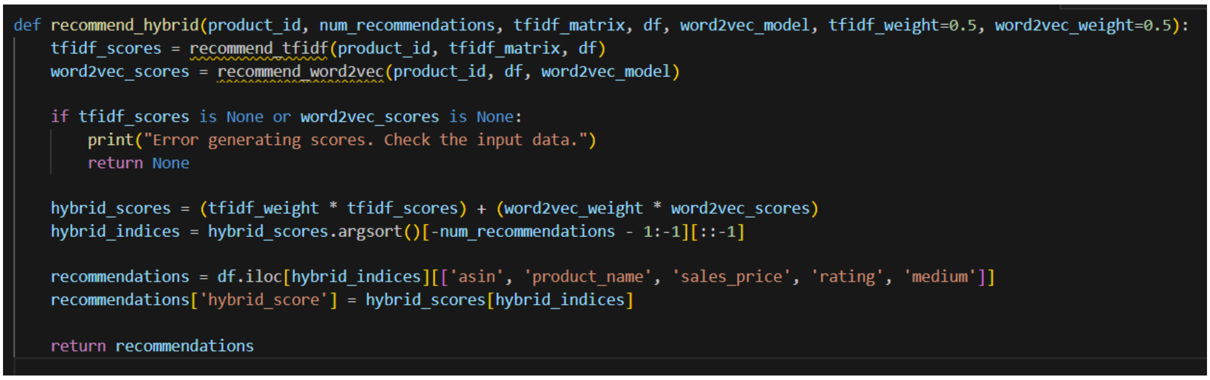
The code combines **TF-IDF** and **Word2Vec** to generate a hybrid recommendation system. This approach leverages the strengths of both methods:

- **TF-IDF** captures the relative importance of terms in product descriptions.
- **Word2Vec** captures the semantic relationships between words, enabling better context-based recommendations.

Advantages of the Hybrid Approach

- **Improved Accuracy:**
  Combines term relevance from TF-IDF and semantic context from Word2Vec.
- **Versatility:**
  Works well for both textual and semantic analysis.
- **Customization:**
  Weighting parameters allow adjusting the contribution of TF-IDF and Word2Vec based on the dataset's characteristics.

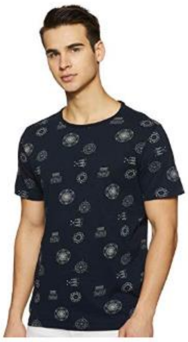
Code:



```
def recommend_hybrid(product_id, num_recommendations, tfidf_matrix, df, word2vec_model, tfidf_weight=0.5, word2vec_weight=0.5):
    tfidf_scores = recommend_tfidf(product_id, tfidf_matrix, df)
    word2vec_scores = recommend_word2vec(product_id, df, word2vec_model)

    if tfidf_scores is None or word2vec_scores is None:
        print("Error generating scores. Check the input data.")
        return None

    hybrid_scores = (tfidf_weight * tfidf_scores) + (word2vec_weight * word2vec_scores)
    hybrid_indices = hybrid_scores.argsort()[-num_recommendations - 1:-1][::-1]

    recommendations = df.iloc[hybrid_indices][['asin', 'product_name', 'sales_price', 'rating', 'medium']]
    recommendations['hybrid_score'] = hybrid_scores[hybrid_indices]

    return recommendations
```

Output:

Query Image



Recommendations:

Brand-Based Recommendation System

The **brand-based recommendation system** generates recommendations by filtering products that belong to the same brand as the selected product. This method is useful for customers who have a strong preference for certain brands, ensuring the recommendations align with their brand loyalty.

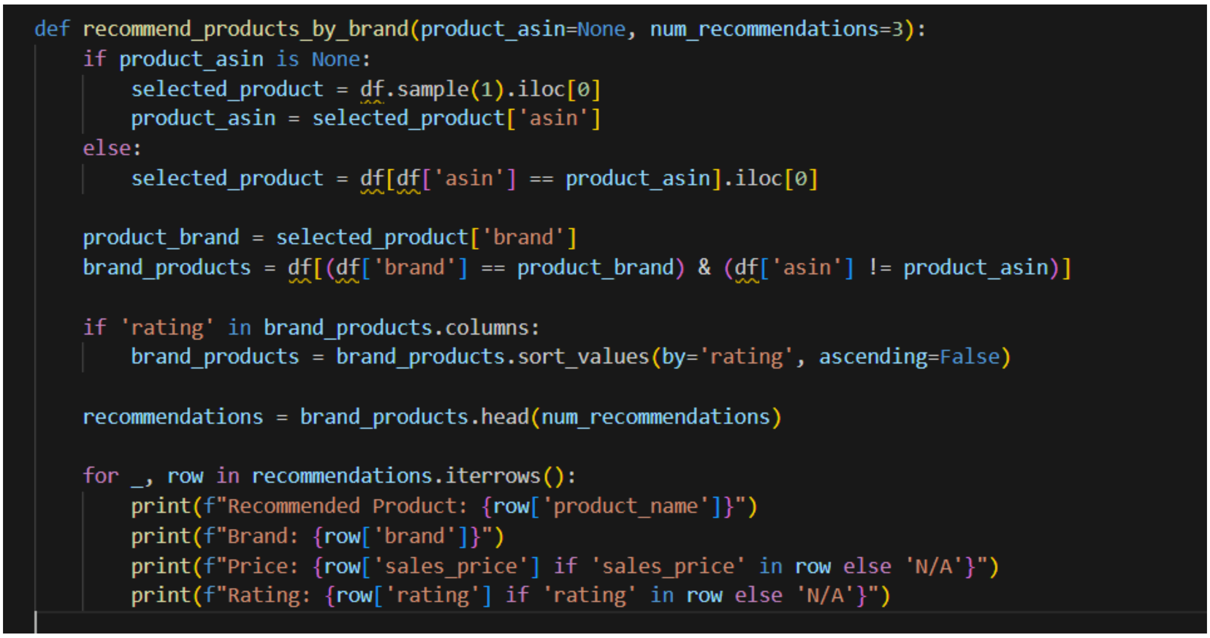
Key Points:

1. Products are filtered based on the brand of the input product.
2. Recommendations are sorted by rating (if available) to prioritize highly-rated items.
3. Relevant product details like name, price, and rating are displayed along with images.

Advantages:

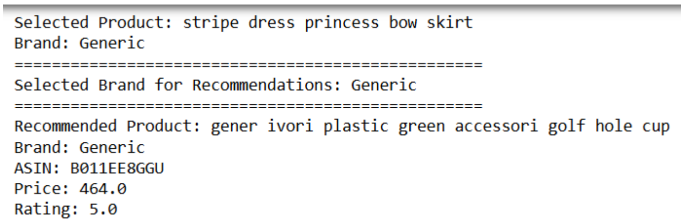
- Promotes brand loyalty by recommending similar products.
- Simple and efficient for real-time applications.
- Improves user experience by focusing on brand-specific products.

Core Code:

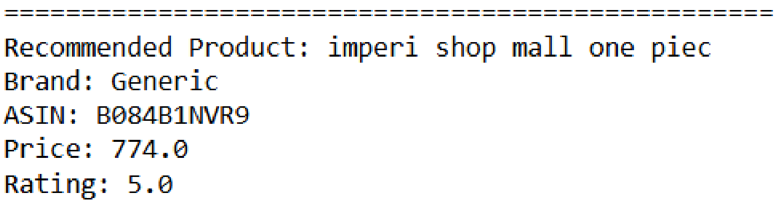


```
def recommend_products_by_brand(product_asin=None, num_recommendations=3):
    if product_asin is None:
        selected_product = df.sample(1).iloc[0]
        product_asin = selected_product['asin']
    else:
        selected_product = df[df['asin'] == product_asin].iloc[0]

    product_brand = selected_product['brand']
    brand_products = df[(df['brand'] == product_brand) & (df['asin'] != product_asin)]

    if 'rating' in brand_products.columns:
        brand_products = brand_products.sort_values(by='rating', ascending=False)

    recommendations = brand_products.head(num_recommendations)

    for _, row in recommendations.iterrows():
        print(f"Recommended Product: {row['product_name']}")
        print(f"Brand: {row['brand']}")
        print(f"Price: {row['sales_price'] if 'sales_price' in row else 'N/A'}")
        print(f"Rating: {row['rating'] if 'rating' in row else 'N/A'}")
```

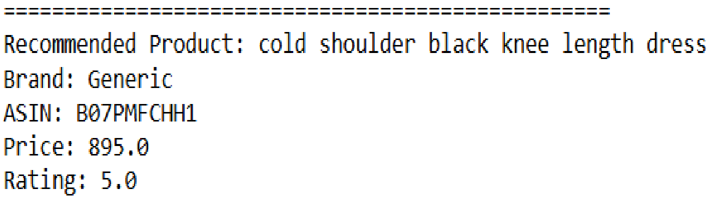
Output:



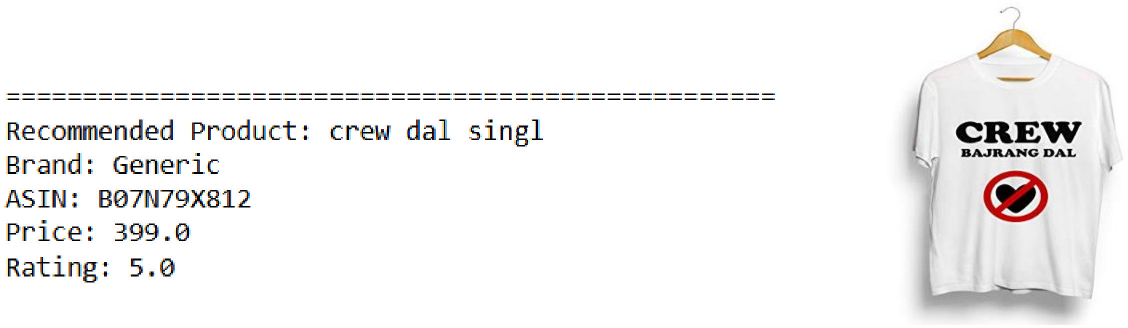
```
Selected Product: stripe dress princess bow skirt
Brand: Generic
==================================================
Selected Brand for Recommendations: Generic
==================================================
Recommended Product: gener ivori plastic green accessori golf hole cup
Brand: Generic
ASIN: B011EE8GGU
Price: 464.0
Rating: 5.0
```



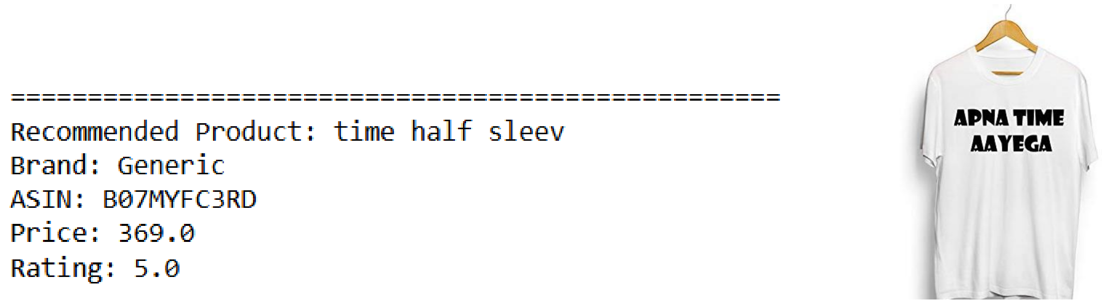
```
==================================================
Recommended Product: imperi shop mall one piec
Brand: Generic
ASIN: B084B1NVR9
Price: 774.0
Rating: 5.0
```



```
==================================================
Recommended Product: cold shoulder black knee length dress
Brand: Generic
ASIN: B07PMFCHH1
Price: 895.0
Rating: 5.0
```



```
==================================================
Recommended Product: crew dal singl
Brand: Generic
ASIN: B07N79X812
Price: 399.0
Rating: 5.0
```



```
==================================================
Recommended Product: time half sleev
Brand: Generic
ASIN: B07MYFC3RD
Price: 369.0
Rating: 5.0
```

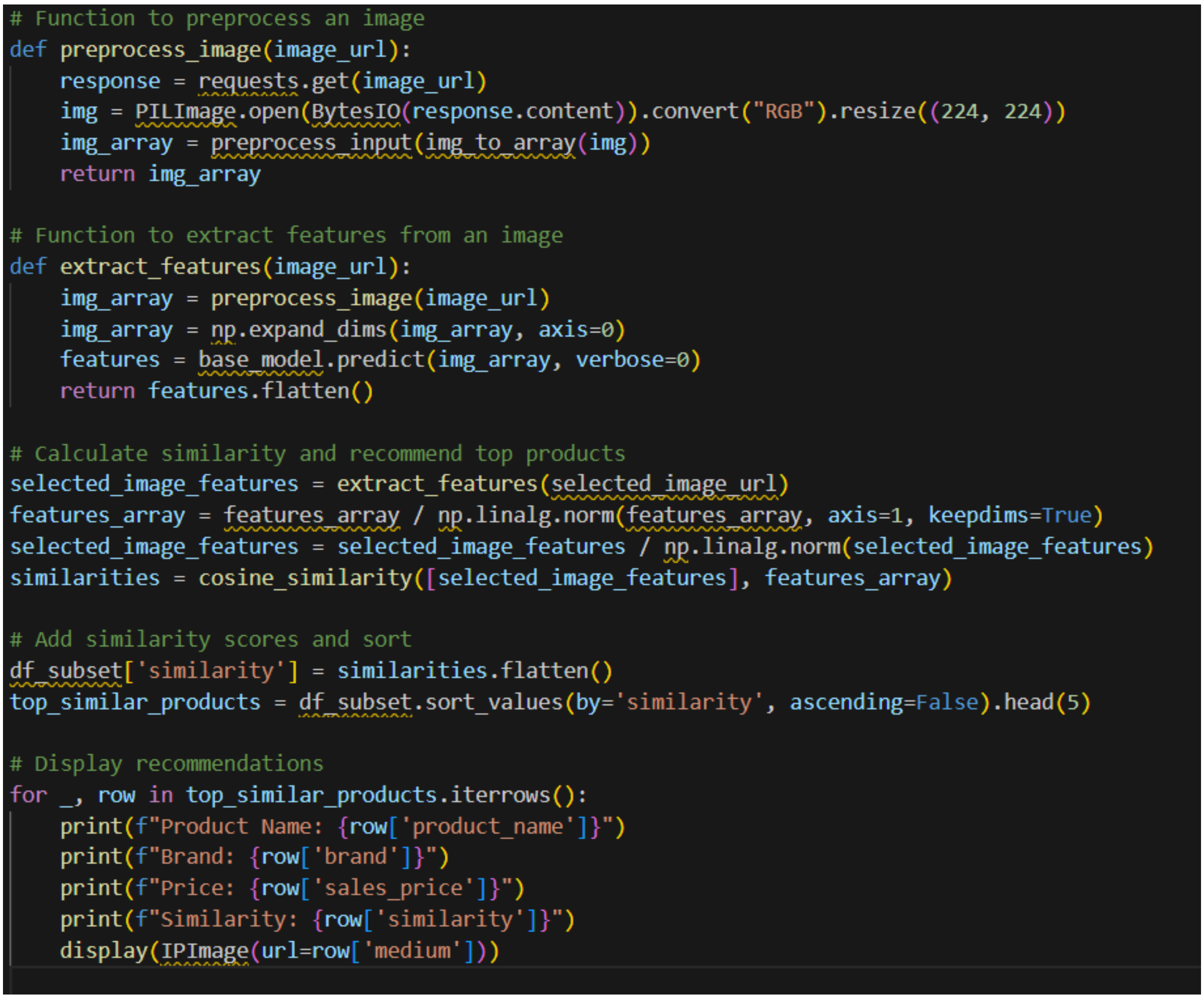
Image-Based Recommendation System

The **image-based recommendation system** recommends visually similar products by analyzing image features. A pre-trained deep learning model extracts meaningful features from images, which are then compared to determine similarity. This system enhances user experience by focusing on products that share similar visual aesthetics.

Key Points:

1. **Feature Extraction:** Uses a pre-trained EfficientNetB0 model to extract image features.
2. **Clustering:** Employs K-Means clustering to create synthetic labels for training a custom model.
3. **Similarity Computation:** Utilizes cosine similarity between image feature vectors to identify visually similar products.
4. **Recommendation:** Ranks products based on similarity scores and presents the top matches.

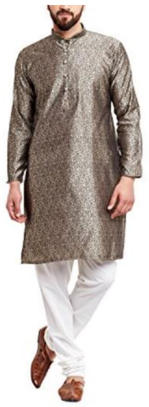
Core Code:



```
# Function to preprocess an image
def preprocess_image(image_url):
    response = requests.get(image_url)
    img = PILImage.open(BytesIO(response.content)).convert("RGB").resize((224, 224))
    img_array = preprocess_input(img_to_array(img))
    return img_array

# Function to extract features from an image
def extract_features(image_url):
    img_array = preprocess_image(image_url)
    img_array = np.expand_dims(img_array, axis=0)
    features = base_model.predict(img_array, verbose=0)
    return features.flatten()

# Calculate similarity and recommend top products
selected_image_features = extract_features(selected_image_url)
features_array = features_array / np.linalg.norm(features_array, axis=1, keepdims=True)
selected_image_features = selected_image_features / np.linalg.norm(selected_image_features)
similarities = cosine_similarity([selected_image_features], features_array)

# Add similarity scores and sort
df_subset['similarity'] = similarities.flatten()
top_similar_products = df_subset.sort_values(by='similarity', ascending=False).head(5)

# Display recommendations
for _, row in top_similar_products.iterrows():
    print(f"Product Name: {row['product_name']}")
    print(f"Brand: {row['brand']}")
    print(f"Price: {row['sales_price']}")
    print(f"Similarity: {row['similarity']}")
    display(IPImage(url=row['medium']))
```

Output:

Query Image:



Recommendations:

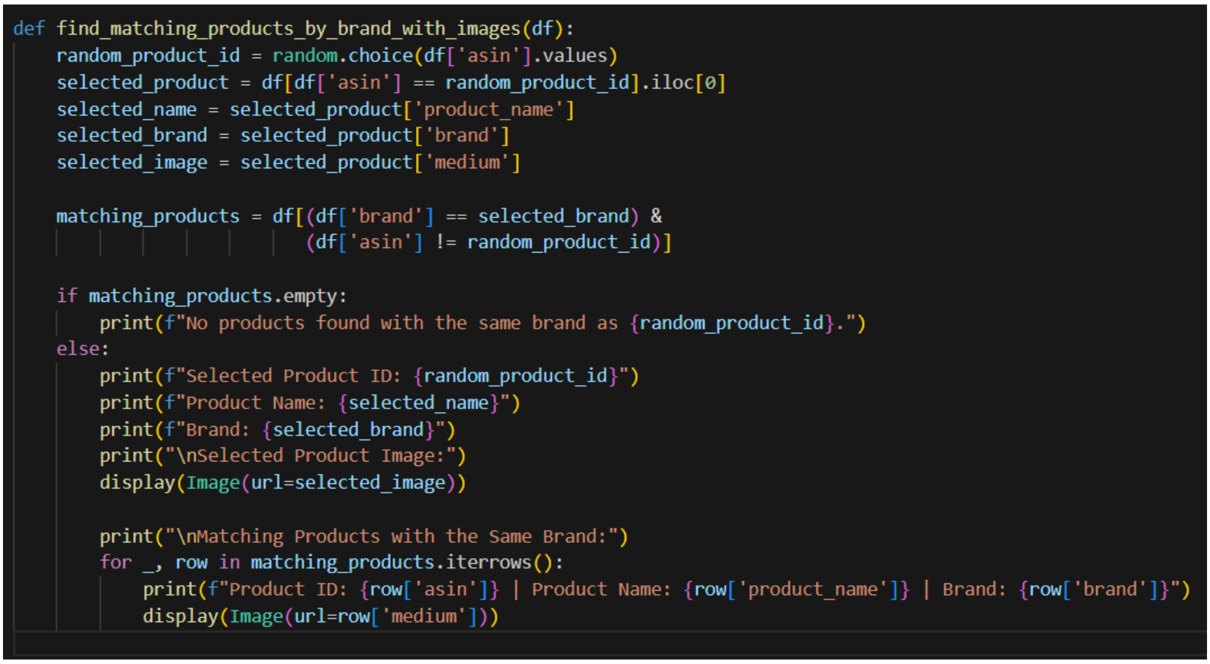
Combination of Product Name and Brand-Based Recommendation

This method combines product name and brand to enhance the relevance of recommendations. It identifies products from the same brand as the selected product and provides their details, including names and images. This approach ensures users find products with similar branding and potentially complementary features.

Concept Highlights:

1. **Selected Product:** A random or specific product is chosen from the dataset.
2. **Matching by Brand:** Filters out products of the same brand, excluding the selected product.
3. **Recommendation Details:** Displays matching product names, brands, and images for easy identification.

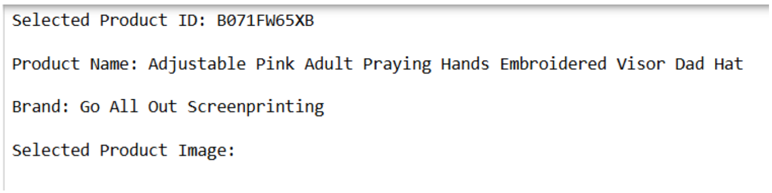
Core Code:



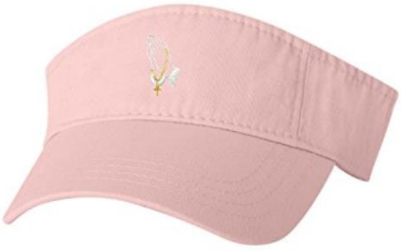
```python
def find_matching_products_by_brand_with_images(df):
    random_product_id = random.choice(df['asin'].values)
    selected_product = df[df['asin'] == random_product_id].iloc[0]
    selected_name = selected_product['product_name']
    selected_brand = selected_product['brand']
    selected_image = selected_product['medium']

    matching_products = df[(df['brand'] == selected_brand) &
                           (df['asin'] != random_product_id)]

    if matching_products.empty:
        print(f"No products found with the same brand as {random_product_id}.")
    else:
        print(f"Selected Product ID: {random_product_id}")
        print(f"Product Name: {selected_name}")
        print(f"Brand: {selected_brand}")
        print("\nSelected Product Image:")
        display(Image(url=selected_image))

        print("\nMatching Products with the Same Brand:")
        for _, row in matching_products.iterrows():
            print(f"Product ID: {row['asin']} | Product Name: {row['product_name']} | Brand: {row['brand']}")
            display(Image(url=row['medium']))
```

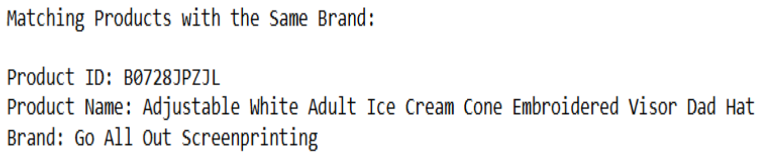
Output:

Query Image:

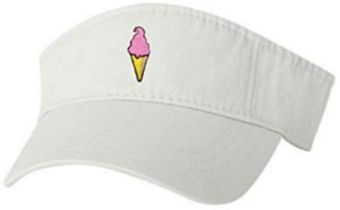


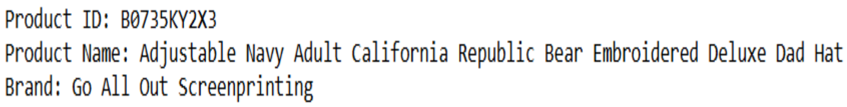
```
Selected Product ID: B071FW65XB

Product Name: Adjustable Pink Adult Praying Hands Embroidered Visor Dad Hat

Brand: Go All Out Screenprinting

Selected Product Image:
```



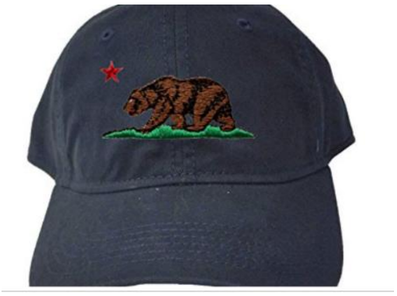


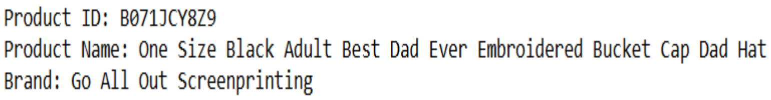
```
Matching Products with the Same Brand:

Product ID: B0728JPZJL
Product Name: Adjustable White Adult Ice Cream Cone Embroidered Visor Dad Hat
Brand: Go All Out Screenprinting
```



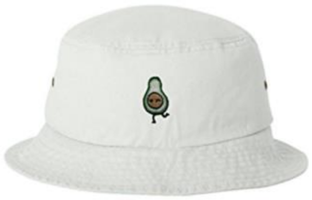


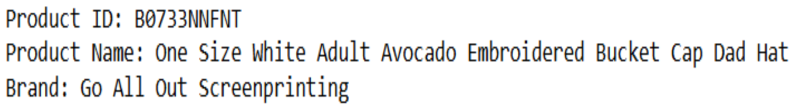
```
Product ID: B0735KY2X3
Product Name: Adjustable Navy Adult California Republic Bear Embroidered Deluxe Dad Hat
Brand: Go All Out Screenprinting
```



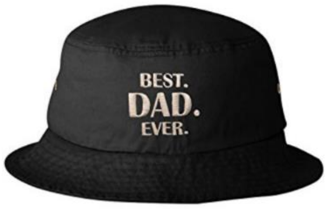


```
Product ID: B071JCY8Z9
Product Name: One Size Black Adult Best Dad Ever Embroidered Bucket Cap Dad Hat
Brand: Go All Out Screenprinting
```





```
Product ID: B0733NNFNT
Product Name: One Size White Adult Avocado Embroidered Bucket Cap Dad Hat
Brand: Go All Out Screenprinting
```



6. Collaborative Filtering

Overview

Recommendation systems are essential tools in e-commerce, streaming platforms, and other industries. They analyze user preferences and product attributes to suggest items users are likely to enjoy. This report delves into collaborative filtering as the core approach, augmented by advanced techniques leveraging pre-trained deep learning models like VGG16 and MobileNet for embedding generation and product recommendations.

Collaborative Filtering

Definition

Collaborative filtering (CF) is a recommendation approach that predicts a user's preference for an item based on past interactions and the preferences of similar users or items. It operates under the assumption that users who agreed in the past will agree in the future.

Types of Collaborative Filtering

1. User-Based Collaborative Filtering

- **Methodology**: Finds users with similar preferences ("neighbors") and recommends items preferred by these neighbors.
- **Advantages**:
  - Easy to understand and implement.
  - Works well for users with many interactions.
- **Challenges**:
  - Suffers from scalability issues in large datasets.
  - Cold start problem for new users.

2. Item-Based Collaborative Filtering

- **Methodology**: Identifies items that are similar based on user interaction patterns. Items similar to what a user has previously liked are recommended.
- **Advantages**:
  - More stable than user-based filtering.
  - Handles new users better than user-based approaches.
- **Challenges**:
  - Requires efficient computation of item similarity.

Metrics Used

- **Cosine Similarity**: Measures the cosine of the angle between two non-zero vectors (user/item embeddings). Values range from −1 to 1, where higher values indicate greater similarity.

Challenges of Collaborative Filtering

- **Cold Start Problem**: New users or items lack interaction data.
- **Data Sparsity**: Many users interact with only a small subset of items.
- **Scalability**: Computational demands grow with larger datasets.

Image-Based Recommendations Using Pre-Trained Models

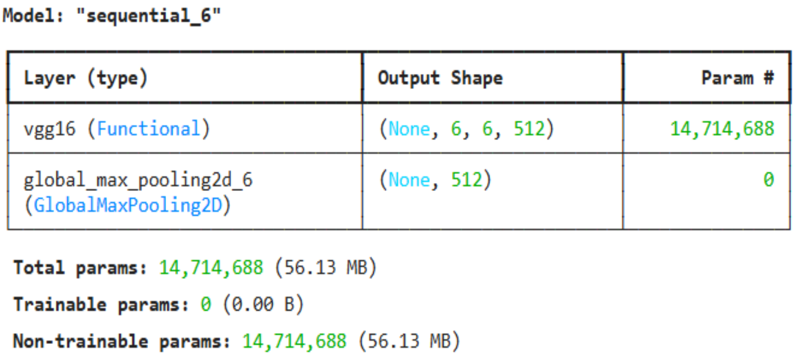
Why Image-Based Recommendations?

Collaborative filtering relies heavily on user interaction data, but product attributes like visual features can provide additional insights. By using pre-trained models like VGG16 and MobileNet, we can extract meaningful embeddings from product images to augment recommendations.

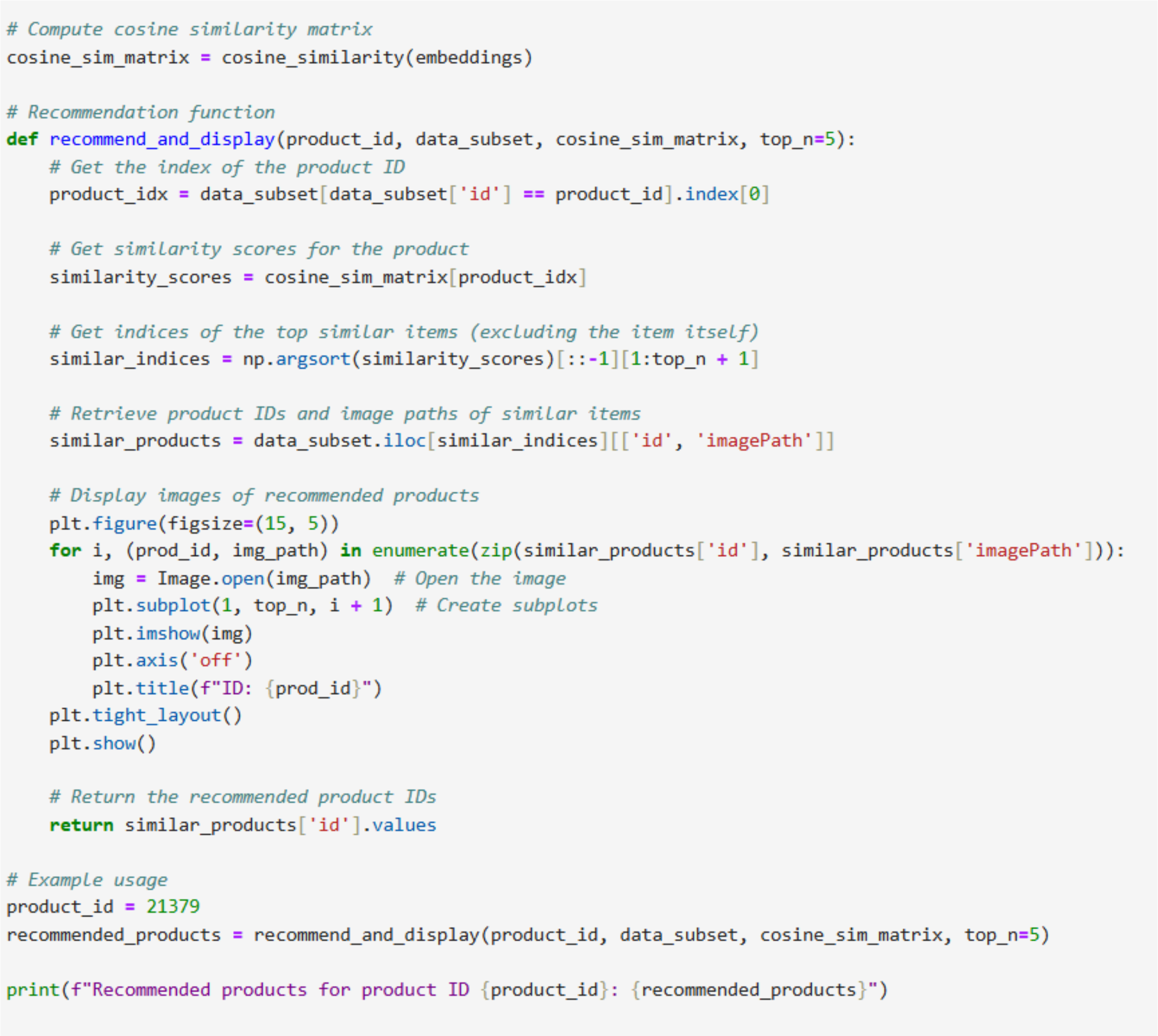
Pre-Trained Models

1. VGG16

- **Description**: VGG16 is a deep convolutional neural network architecture developed by the Visual Geometry Group (VGG) at Oxford. It has 16 weight layers and is known for its simplicity and effectiveness.
- **Features**:
    - Uses a uniform architecture with small 3x3 filters.
    - Trained on the ImageNet dataset.
    - Outputs feature maps that are effective for image-related tasks.
- **Advantages**:
    - Produces high-quality embeddings suitable for diverse tasks.
    - Well-documented and widely supported.
- **Challenges**:
    - Computationally expensive due to its depth.
    - Larger model size compared to alternatives like MobileNet.
- **Code:**



```
Model: "sequential_6"

┌──────────────────────────────────┬─────────────────────────────┬─────────────────┐
│ Layer (type)                     │ Output Shape                │         Param # │
├──────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ vgg16 (Functional)               │ (None, 6, 6, 512)           │      14,714,688 │
├──────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ global_max_pooling2d_6           │ (None, 512)                 │               0 │
│ (GlobalMaxPooling2D)             │                             │                 │
└──────────────────────────────────┴─────────────────────────────┴─────────────────┘

 Total params: 14,714,688 (56.13 MB)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 14,714,688 (56.13 MB)
```



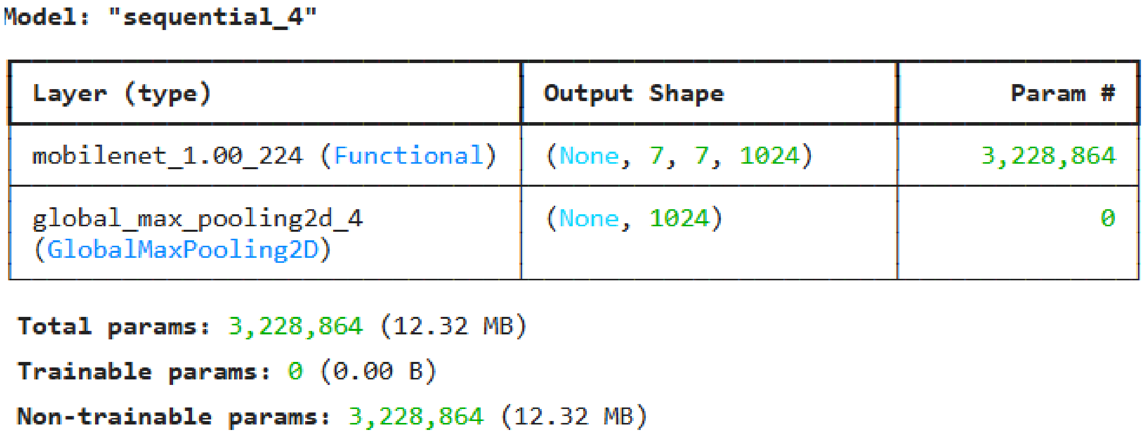
```python
# Compute cosine similarity matrix
cosine_sim_matrix = cosine_similarity(embeddings)

# Recommendation function
def recommend_and_display(product_id, data_subset, cosine_sim_matrix, top_n=5):
    # Get the index of the product ID
    product_idx = data_subset[data_subset['id'] == product_id].index[0]

    # Get similarity scores for the product
    similarity_scores = cosine_sim_matrix[product_idx]

    # Get indices of the top similar items (excluding the item itself)
    similar_indices = np.argsort(similarity_scores)[::-1][1:top_n + 1]

    # Retrieve product IDs and image paths of similar items
    similar_products = data_subset.iloc[similar_indices][['id', 'imagePath']]

    # Display images of recommended products
    plt.figure(figsize=(15, 5))
    for i, (prod_id, img_path) in enumerate(zip(similar_products['id'], similar_products['imagePath'])):
        img = Image.open(img_path)  # Open the image
        plt.subplot(1, top_n, i + 1)  # Create subplots
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"ID: {prod_id}")
    plt.tight_layout()
    plt.show()

    # Return the recommended product IDs
    return similar_products['id'].values

# Example usage
product_id = 21379
recommended_products = recommend_and_display(product_id, data_subset, cosine_sim_matrix, top_n=5)

print(f"Recommended products for product ID {product_id}: {recommended_products}")
```

Output:

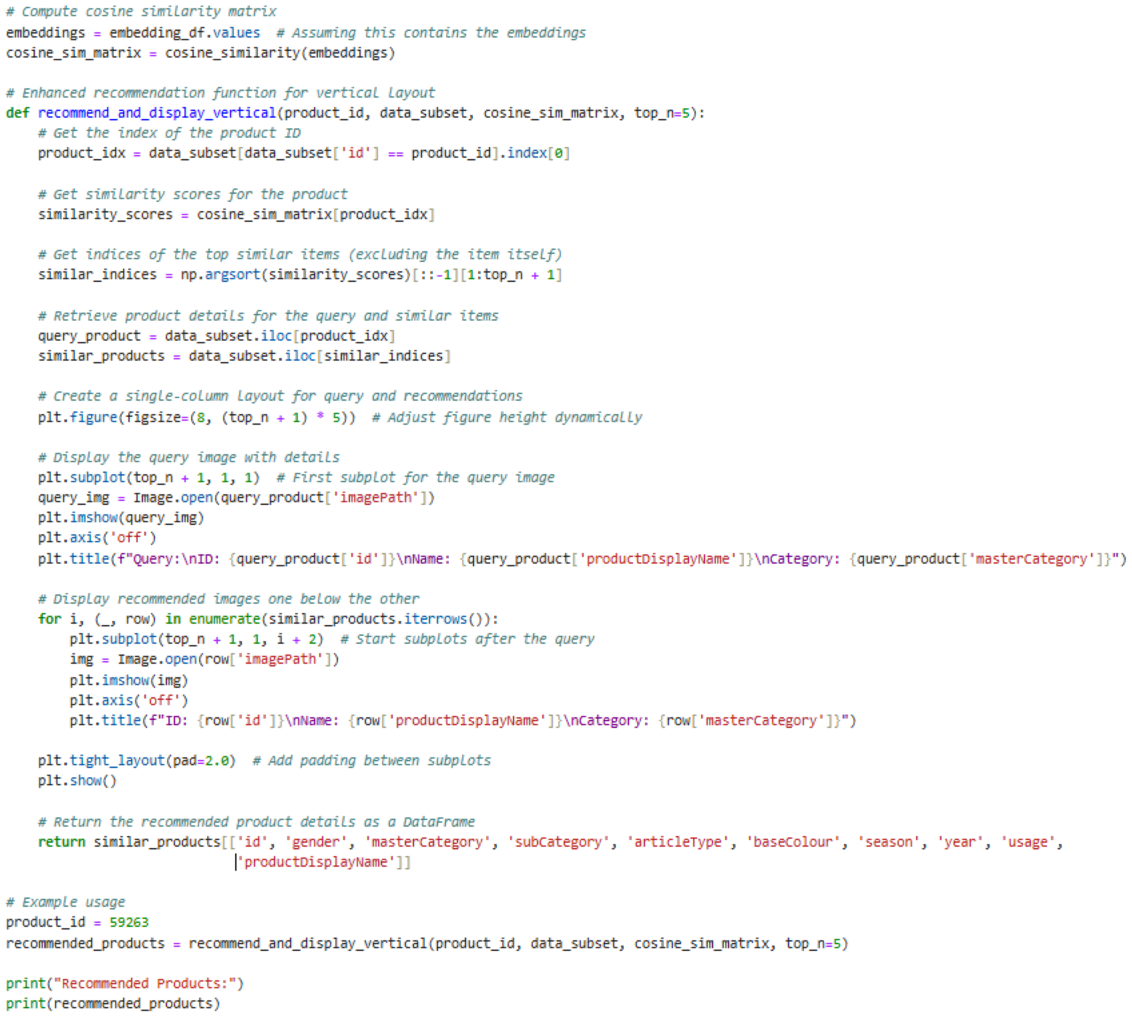
```
Recommended products for product ID 21379: [21385  7709 40371 16508  7707]
```

2. MobileNet

- **Description**: MobileNet is a lightweight convolutional neural network designed for mobile and embedded vision applications.
- **Features**:
  - Uses depthwise separable convolutions to reduce computational cost.
  - Trained on the ImageNet dataset for a variety of vision tasks.
- **Advantages**:
  - Optimized for speed and efficiency.
  - Suitable for deployment on resource-constrained devices.
- **Challenges**:
  - May not perform as well as larger models like VGG16 for some tasks.
- **Code:**



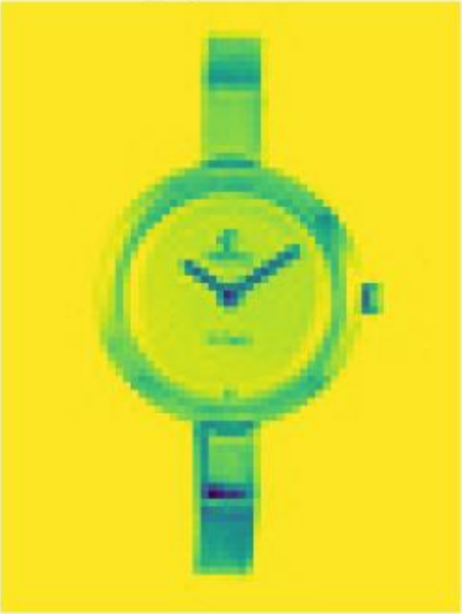
```
Model: "sequential_4"
┌──────────────────────────────────┬──────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape             │      Param # │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ mobilenet_1.00_224 (Functional)  │ (None, 7, 7, 1024)       │    3,228,864 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ global_max_pooling2d_4           │ (None, 1024)             │            0 │
│ (GlobalMaxPooling2D)             │                          │              │
└──────────────────────────────────┴──────────────────────────┴──────────────┘
 Total params: 3,228,864 (12.32 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 3,228,864 (12.32 MB)
```



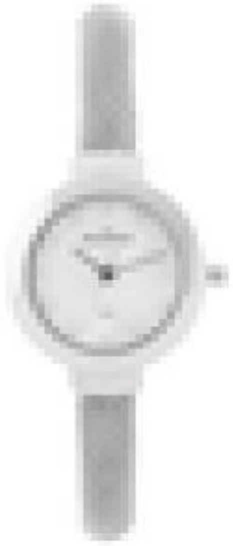
```python
# Compute cosine similarity matrix
embeddings = embedding_df.values  # Assuming this contains the embeddings
cosine_sim_matrix = cosine_similarity(embeddings)

# Enhanced recommendation function for vertical layout
def recommend_and_display_vertical(product_id, data_subset, cosine_sim_matrix, top_n=5):
    # Get the index of the product ID
    product_idx = data_subset[data_subset['id'] == product_id].index[0]

    # Get similarity scores for the product
    similarity_scores = cosine_sim_matrix[product_idx]

    # Get indices of the top similar items (excluding the item itself)
    similar_indices = np.argsort(similarity_scores)[::-1][1:top_n + 1]

    # Retrieve product details for the query and similar items
    query_product = data_subset.iloc[product_idx]
    similar_products = data_subset.iloc[similar_indices]

    # Create a single-column layout for query and recommendations
    plt.figure(figsize=(8, (top_n + 1) * 5))  # Adjust figure height dynamically

    # Display the query image with details
    plt.subplot(top_n + 1, 1, 1)  # First subplot for the query image
    query_img = Image.open(query_product['imagePath'])
    plt.imshow(query_img)
    plt.axis('off')
    plt.title(f"Query:\nID: {query_product['id']}\nName: {query_product['productDisplayName']}\nCategory: {query_product['masterCategory']}")

    # Display recommended images one below the other
    for i, (_, row) in enumerate(similar_products.iterrows()):
        plt.subplot(top_n + 1, 1, i + 2)  # Start subplots after the query
        img = Image.open(row['imagePath'])
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"ID: {row['id']}\nName: {row['productDisplayName']}\nCategory: {row['masterCategory']}")

    plt.tight_layout(pad=2.0)  # Add padding between subplots
    plt.show()

    # Return the recommended product details as a DataFrame
    return similar_products[['id', 'gender', 'masterCategory', 'subCategory', 'articleType', 'baseColour', 'season', 'year', 'usage',
                             'productDisplayName']]

# Example usage
product_id = 59263
recommended_products = recommend_and_display_vertical(product_id, data_subset, cosine_sim_matrix, top_n=5)

print("Recommended Products:")
print(recommended_products)
```

Output:

Query:
ID: 59263
Name: Titan Women Silver Watch
Category: Accessories



ID: 30036
Name: SKAGEN DENMARK Women White Mother of Pearl Dial Watch 107SSCS
Category: Accessories
Similarity: 0.80

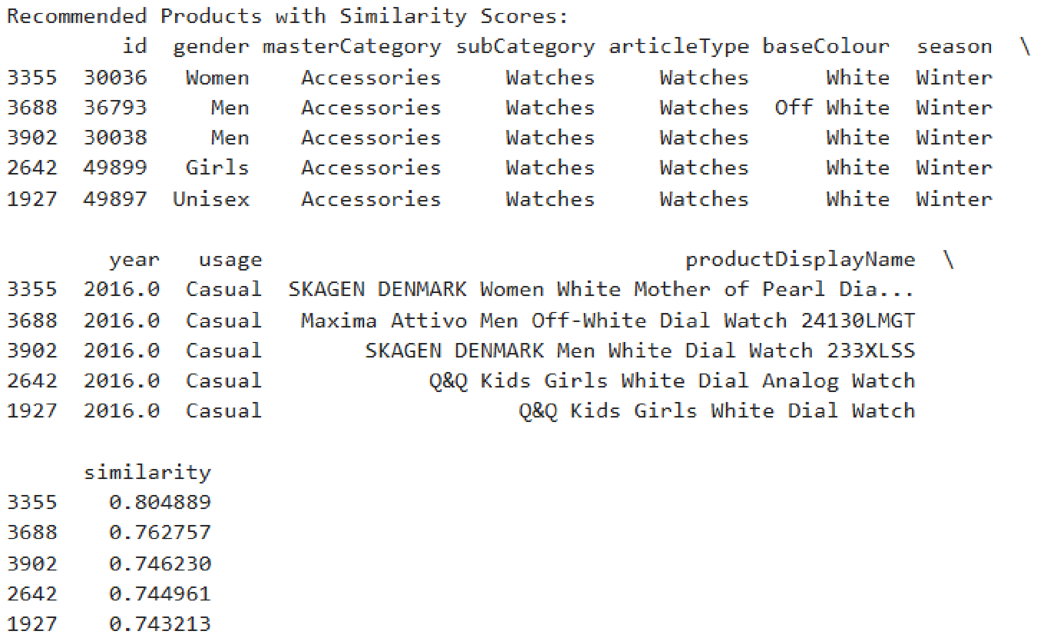


ID: 36793
Name: Maxima Attivo Men Off-White Dial Watch 24130LMGT
Category: Accessories
Similarity: 0.76

ID: 30038
Name: SKAGEN DENMARK Men White Dial Watch 233XLSS
Category: Accessories
Similarity: 0.75

ID: 49899
Name: Q&Q Kids Girls White Dial Analog Watch
Category: Accessories
Similarity: 0.74

ID: 49897
Name: Q&Q Kids Girls White Dial Watch
Category: Accessories
Similarity: 0.74

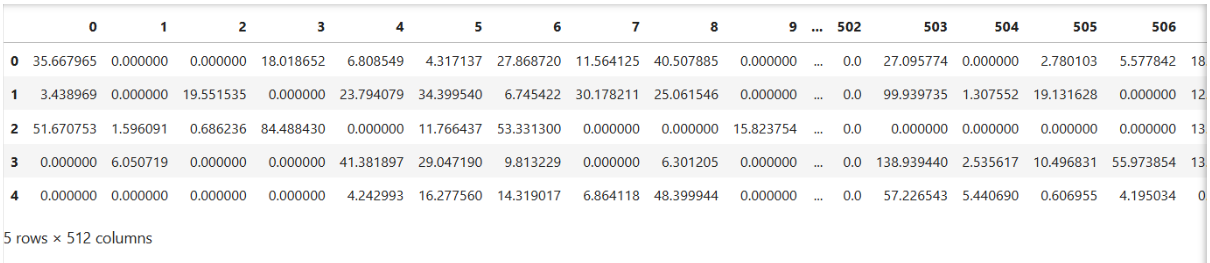


```
Recommended Products with Similarity Scores:
         id  gender masterCategory subCategory articleType baseColour  season  \
3355  30036   Women    Accessories     Watches     Watches      White  Winter
3688  36793     Men    Accessories     Watches     Watches  Off White  Winter
3902  30038     Men    Accessories     Watches     Watches      White  Winter
2642  49899   Girls    Accessories     Watches     Watches      White  Winter
1927  49897  Unisex    Accessories     Watches     Watches      White  Winter

        year   usage                             productDisplayName  \
3355  2016.0  Casual  SKAGEN DENMARK Women White Mother of Pearl Dia...
3688  2016.0  Casual   Maxima Attivo Men Off-White Dial Watch 24130LMGT
3902  2016.0  Casual       SKAGEN DENMARK Men White Dial Watch 233XLSS
2642  2016.0  Casual            Q&Q Kids Girls White Dial Analog Watch
1927  2016.0  Casual                   Q&Q Kids Girls White Dial Watch

      similarity
3355    0.804889
3688    0.762757
3902    0.746230
2642    0.744961
1927    0.743213
```

Implementation Steps

1. Feature Extraction

- **Objective**: Extract meaningful image embeddings using VGG16 and MobileNet.
- **Process**:
    1. Load the pre-trained model (VGG16 or MobileNet) with weights trained on ImageNet.
    2. Remove the top classification layer to retain only the feature extraction layers.
    3. Apply a global pooling layer to convert feature maps into compact embeddings.
    4. Generate embeddings for all product images.
- **Output**: A numerical vector (embedding) representing each image.



	0	1	2	3	4	5	6	7	8	9	...	502	503	504	505	506	
0	35.667965	0.000000	0.000000	18.018652	6.808549	4.317137	27.868720	11.564125	40.507885	0.000000	...	0.0	27.095774	0.000000	2.780103	5.577842	18
1	3.438969	0.000000	19.551535	0.000000	23.794079	34.399540	6.745422	30.178211	25.061546	0.000000	...	0.0	99.939735	1.307552	19.131628	0.000000	12
2	51.670753	1.596091	0.686236	84.488430	0.000000	11.766437	53.331300	0.000000	0.000000	15.823754	...	0.0	0.000000	0.000000	0.000000	0.000000	13
3	0.000000	6.050719	0.000000	0.000000	41.381897	29.047190	9.813229	0.000000	6.301205	0.000000	...	0.0	138.939440	2.535617	10.496831	55.973854	13
4	0.000000	0.000000	0.000000	0.000000	4.242993	16.277560	14.319017	6.864118	48.399944	0.000000	...	0.0	57.226543	5.440690	0.606955	4.195034	0

5 rows × 512 columns

2. Saving Embeddings

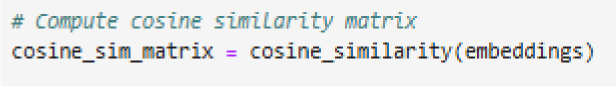
- **Storage**: Save embeddings in a CSV or .npy file, aligning each embedding with its corresponding product ID and metadata.

- **Advantages**: Allows efficient similarity computations and avoids repetitive extraction.

3. Computing Similarity

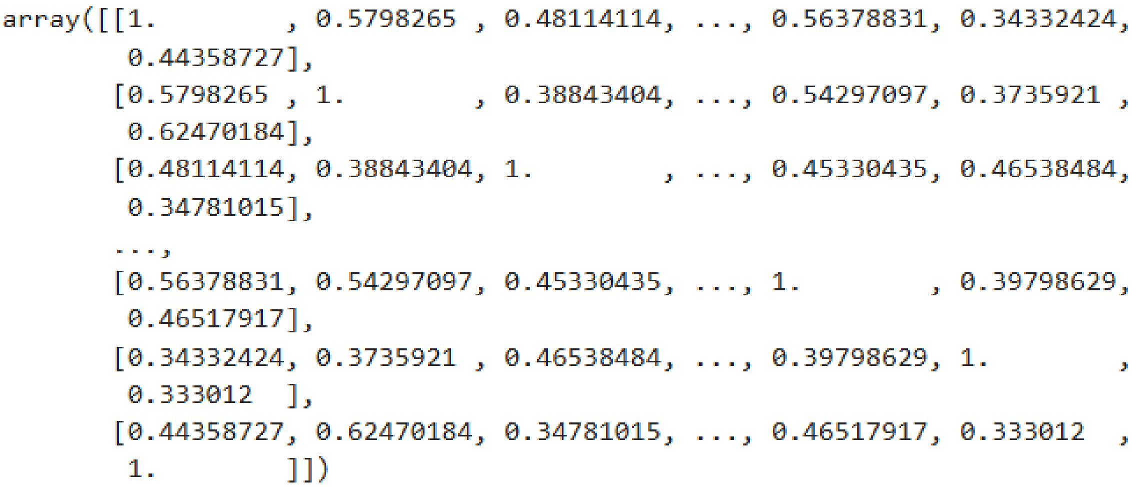
- **Method**: Use cosine similarity to compare embeddings and identify visually similar products.
- **Implementation**:
  - Compute pairwise similarity between embeddings.
  - Rank items based on similarity scores.

Code:



```
# Compute cosine similarity matrix
cosine_sim_matrix = cosine_similarity(embeddings)
```

Outputs:



```
array([[1.        , 0.5798265 , 0.48114114, ..., 0.56378831, 0.34332424,
        0.44358727],
       [0.5798265 , 1.        , 0.38843404, ..., 0.54297097, 0.3735921 ,
        0.62470184],
       [0.48114114, 0.38843404, 1.        , ..., 0.45330435, 0.46538484,
        0.34781015],
       ...,
       [0.56378831, 0.54297097, 0.45330435, ..., 1.        , 0.39798629,
        0.46517917],
       [0.34332424, 0.3735921 , 0.46538484, ..., 0.39798629, 1.        ,
        0.333012  ],
       [0.44358727, 0.62470184, 0.34781015, ..., 0.46517917, 0.333012  ,
        1.        ]])
```

4. Recommendation Pipeline

1. **Collaborative Filtering**:
   - Generate an initial list of recommendations using user or item interactions.
2. **Image-Based Refinement**:
   - For each recommended item, identify visually similar products using embeddings.
   - Integrate metadata (e.g., product category, brand) for contextual relevance.
3. **Display Results**:
   - Show recommended products along with their details (name, category, brand, similarity score).

Evaluation and Advantages

Evaluation Metrics

- **Accuracy**: Measure how well the system predicts user preferences.
- **Diversity**: Assess the variety in recommendations.
- **User Satisfaction**: Evaluate the relevance and appeal of recommendations.

Advantages of Combining Collaborative Filtering and Image-Based Methods

- **Cold Start Solution**: Image embeddings provide insights for new items with limited interactions.
- **Enhanced Relevance**: Combines behavioral data with visual cues for better recommendations.
- **Scalability**: Efficient embedding storage and retrieval allow for large-scale deployment.

Conclusion

This hybrid approach, leveraging collaborative filtering and pre-trained deep learning models like VGG16 and MobileNet, addresses common challenges in recommendation systems. Collaborative filtering provides a foundation based on user interactions, while image-based methods add depth by incorporating visual features. Together, these techniques create a robust, versatile recommendation pipeline suitable for diverse applications in e-commerce and beyond.