INFOSYS SPRINGBOARD INTERNSHIP REPORT

PROJECT NAME: AI STYLIST

DOMAIN: ARTIFICIAL INTELLIGENCE

MENTOR NAME: ANIL KUMAR SHAW

TEAM MEMBERS:

1. Sushant
2. Varsha Peddireddy
3. Lakshmi Prasanna Mudige
4. Vaithees

Table of Contents

Abstract

The advent of artificial intelligence has revolutionized the fashion industry, introducing innovative solutions like AI-powered personal stylists. This project explores the development of an **AI Stylist**, a system capable of recommending fashion items tailored to user preferences and styles. By combining Artificial Intelligence techniques and advanced feature extraction methods, the AI Stylist analyses image data to identify trends, suggest complementary pieces, and provide a personalized shopping experience.

Using VGG16 for feature extraction and cosine similarity for recommendation generation, the AI Stylist processes extensive datasets of fashion items, creating embeddings that capture the essence of each item. This enables the system to recommend similar or matching items, aligning with a user's style or enhancing their wardrobe.

The AI Stylist exemplifies collaborative and data-driven innovation, merging computer vision, machine learning, and user-centric design principles. This work highlights the transformative potential of artificial intelligence in reshaping fashion retail, improving customer engagement, and fostering creativity in personal styling.

INTRODUCTION

The project leverages advanced machine learning techniques, specifically feature extraction and similarity analysis, to build a personalized recommendation system. By utilizing pre-trained models like MobileNetV2, the system extracts meaningful features from images of fashion items. These features are then compared using cosine similarity to identify products that align with a user's preferences. The integration of two robust datasets, Content-based filtering fashion_ products_ LD Json and **Styles.csv**, ensures a comprehensive understanding of the fashion domain, enabling effective recommendations tailored to various styles, categories, and trends.

Datasets

1. **Content-based filtering [Fashion Products (fashion_ products_ LD Json)]**
   a recommender system that uses machine learning to suggest items to a user based on their interests and previous interactions
   This dataset comprises a large collection of fashion-related data in LDJSON (Line Delimited JSON) format. Each record contains metadata and visual information about fashion products, including:
   - Product ID: Unique identifier for each item.
   - Categories: Classification of items into groups such as shirts, dresses, or accessories.
   - Product Details: Descriptions, sizes, materials, and brand information.
   - Image URLs: Links to high-resolution images of products.

This dataset serves as a foundational resource for training the AI Stylist to understand the nuances of fashion categories, item relationships, and customer behaviour patterns. The image data supports feature extraction, while the textual metadata enhances contextual understanding.

2. **Collaborative filtering [Myntra dataset]**
   Collaborative filtering is a technique that recommends items to a user based on the preferences of similar users. It works by finding a group of users with similar tastes to a particular user, and then combining the items they like to create a ranked list of recommendations.
   It includes **Styles.csv** and **Images** datasets offers a structured view of fashion products, emphasizing attributes relevant to stylistic and aesthetic analysis. The dataset includes:

Style ID: A unique identifier for each product style.

- Product Type: Categories such as tops, bottoms, or footwear.
- Visual Features: Colour, pattern, and design descriptors.
- Price: Pricing information for budget-aware recommendations.
- Images: Associated image paths used for visual processing

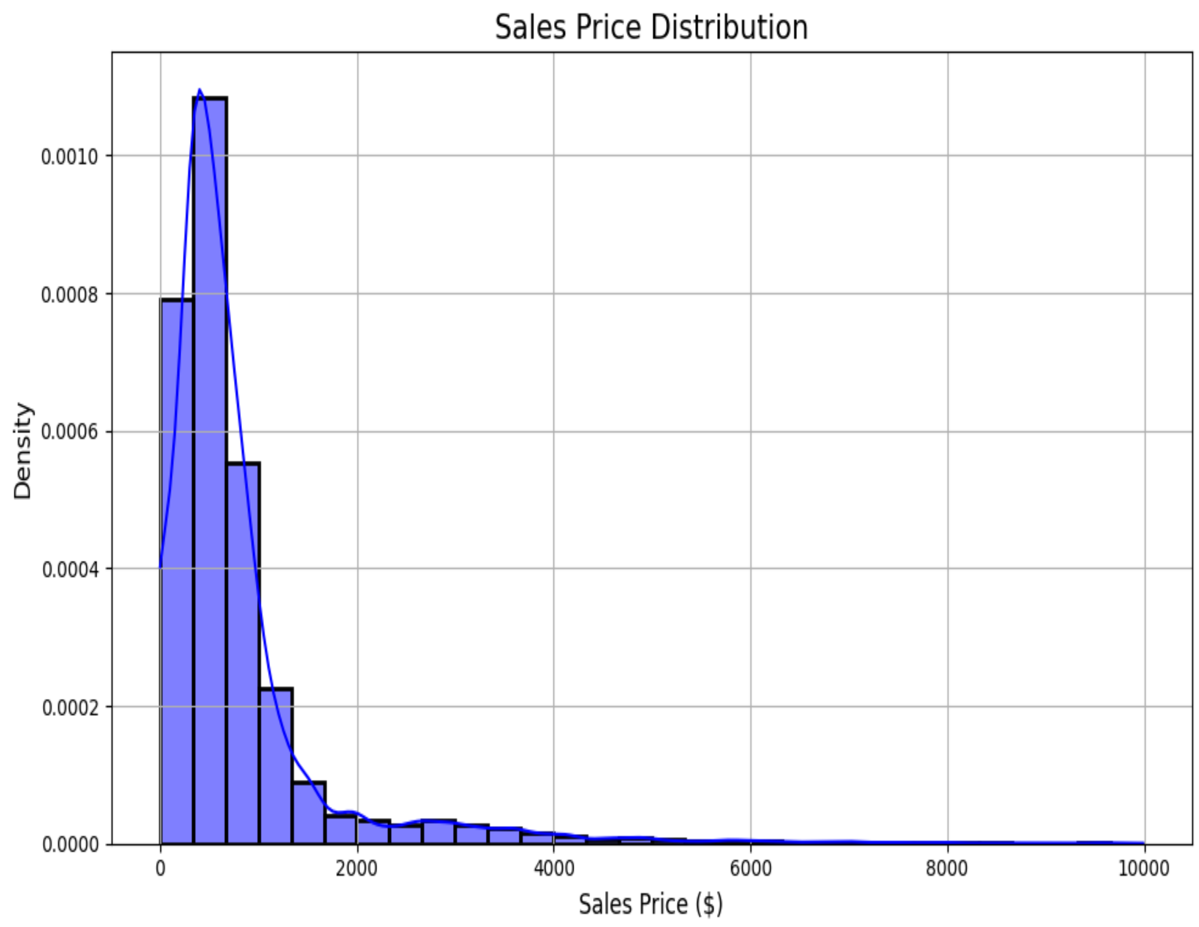
Visualize Distribution of Key Variables

Sales Price Distribution

The **Sales Price Distribution** provides an overview of how product prices are spread across various ranges within a dataset.

Purpose of Sales Price Distribution Visualization

Identify Pricing Trends, Customer Segmentation, Market Insights, Decision Making

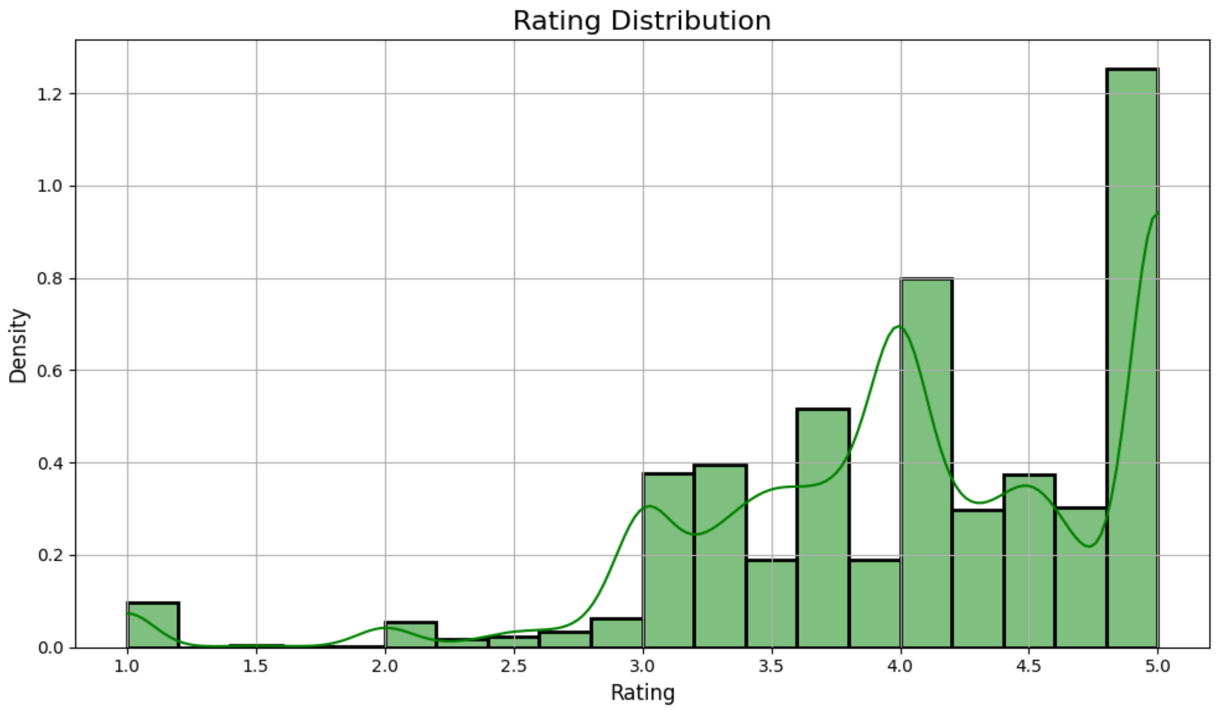


Rating Distribution

The **Rating Distribution** provides an analysis of how customer ratings are spread across products in the dataset.

Purpose of Rating Distribution Visualization

Understand Customer Sentiment, Identify Popular Products, Spot Issues, Track Performance Trends

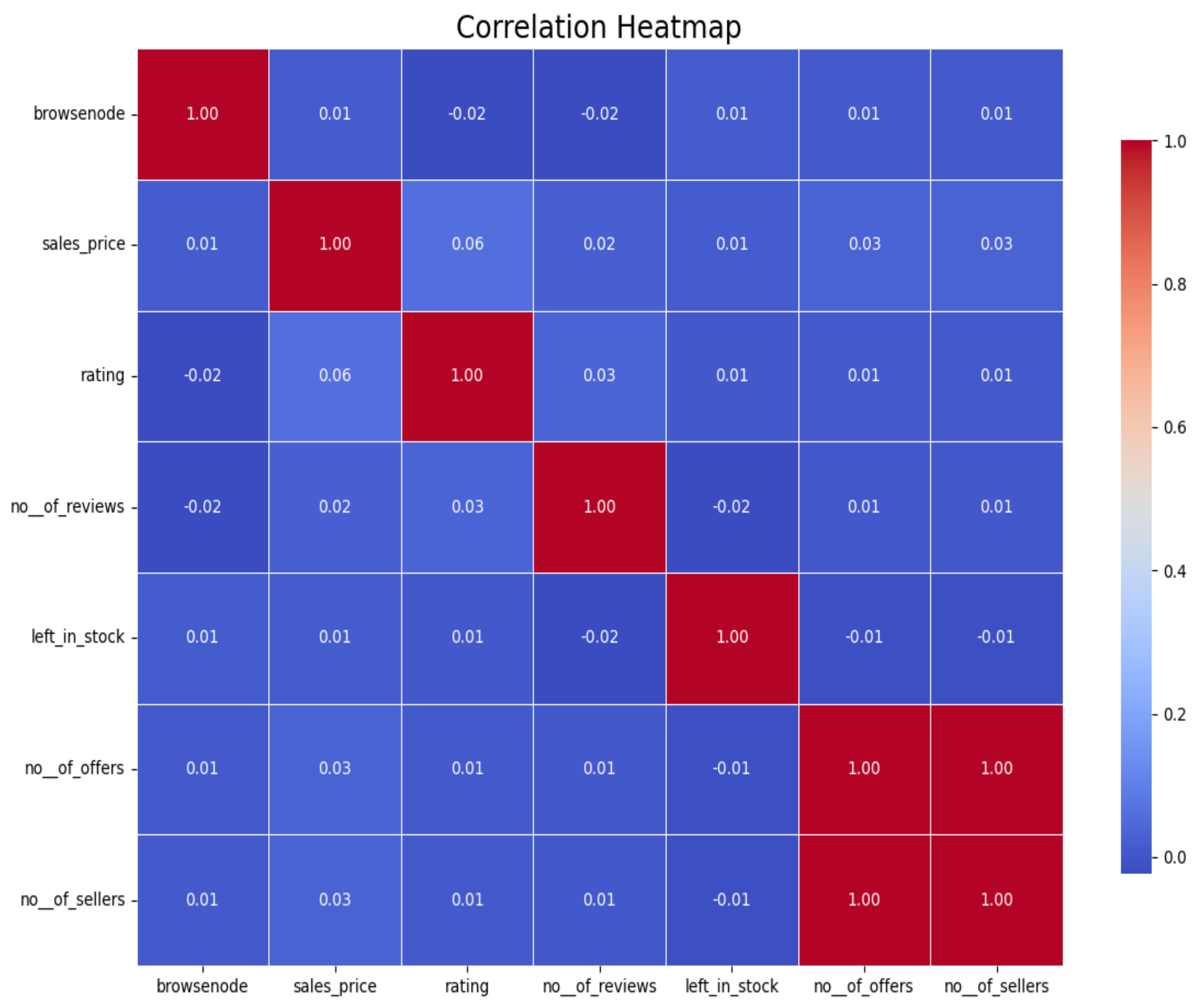


Correlation Heatmap for Numerical Columns

A **correlation heatmap** is a graphical representation of the relationships between different variables in a dataset**.**

Purpose of Correlation Heatmap

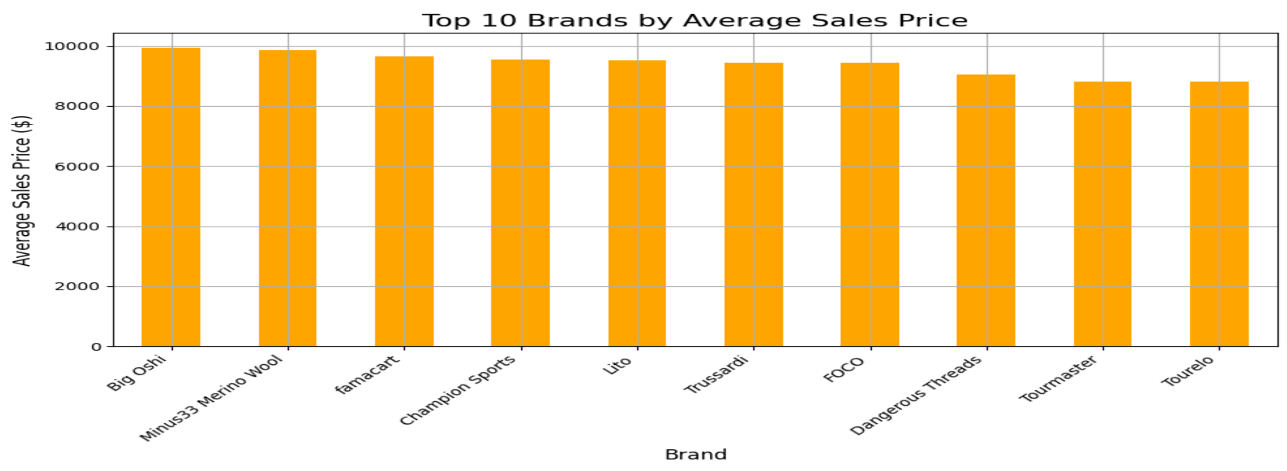
Understand Relationships, Feature Selection, Insights for Decision-Making.



	browsenode	sales_price	rating	no__of_reviews	left_in_stock	no__of_offers	no__of_sellers
browsenode	1.00	0.01	-0.02	-0.02	0.01	0.01	0.01
sales_price	0.01	1.00	0.06	0.02	0.01	0.03	0.03
rating	-0.02	0.06	1.00	0.03	0.01	0.01	0.01
no__of_reviews	-0.02	0.02	0.03	1.00	-0.02	0.01	0.01
left_in_stock	0.01	0.01	0.01	-0.02	1.00	-0.01	-0.01
no__of_offers	0.01	0.03	0.01	0.01	-0.01	1.00	1.00
no__of_sellers	0.01	0.03	0.01	0.01	-0.01	1.00	1.00

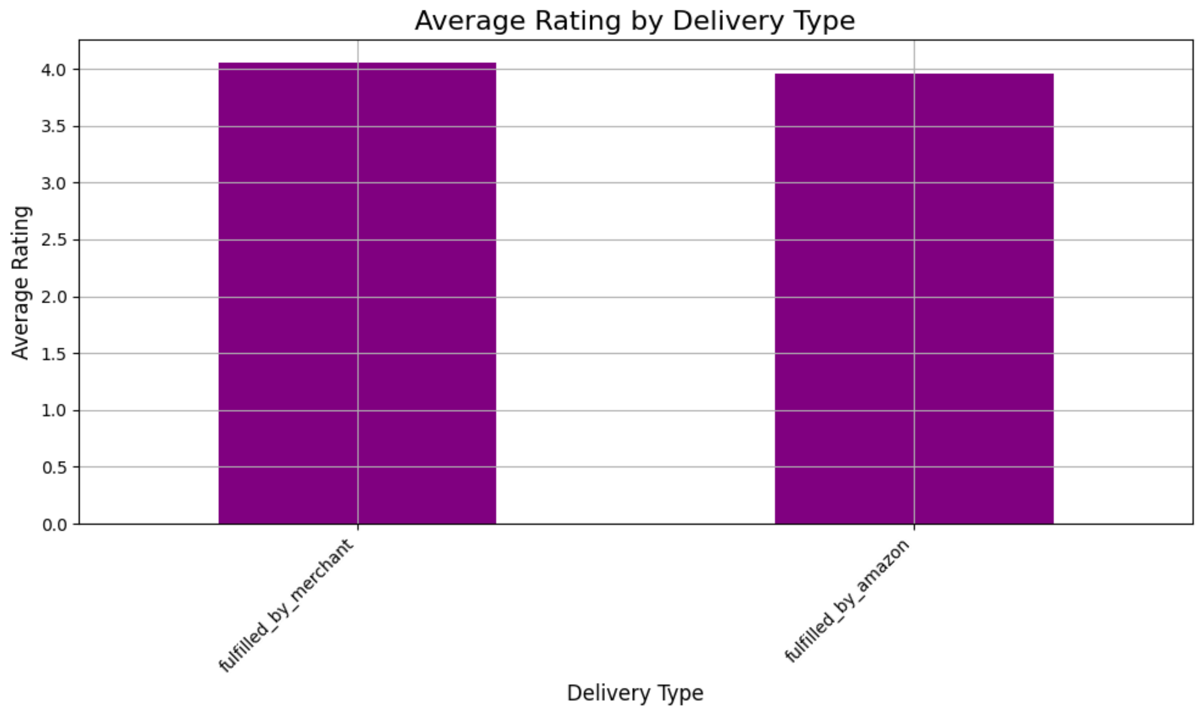
Grouping and Aggregation Analysis

Grouping and aggregation analysis is a data analysis technique used to organize data into meaningful groups and summarize it through aggregate functions like sum, mean, median, count, or other statistical metrics.



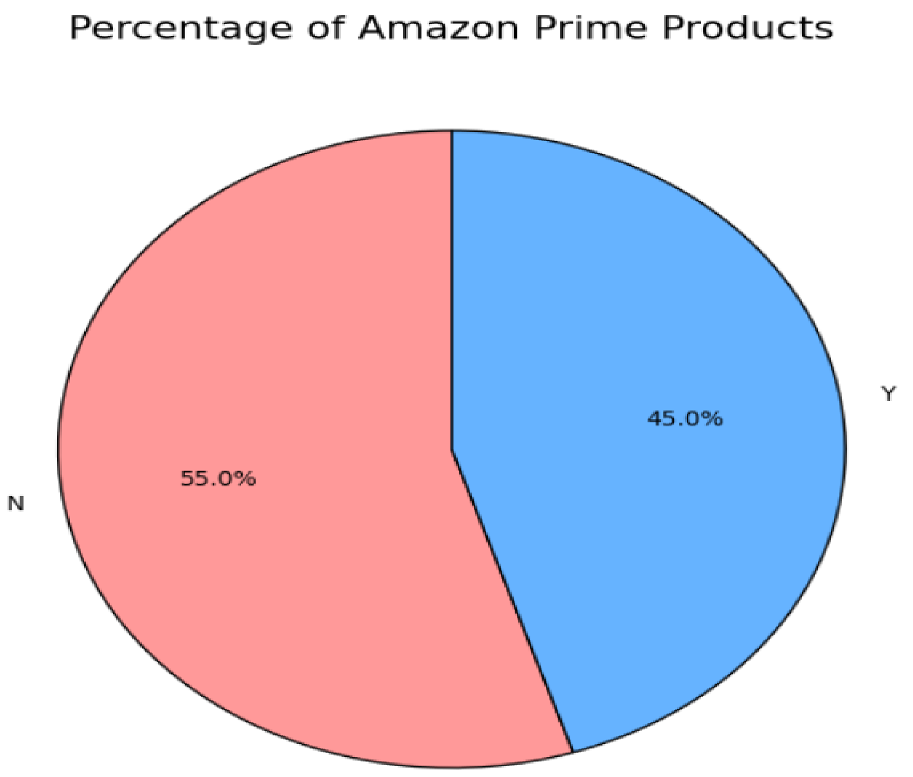
Purpose of Grouping and Aggregation Analysis

Identify Patterns, Summarize Large Datasets, Support Decision-Making.



Analysis:

1. **Delivery Types**:
    - Two delivery types are compared:
        - **fulfilled_ by_ merchant**
        - **fulfilled_ by_ amazon**
2. **Average Ratings**:
    - Both delivery types have an **average rating of approximately 4.0**.
    - This indicates that customers rate products similarly regardless of whether they are fulfilled by the merchant or Amazon.
3. **Uniformity**:
    - The bar heights are nearly identical, suggesting no significant difference in customer satisfaction between the two delivery methods.



Analysis:

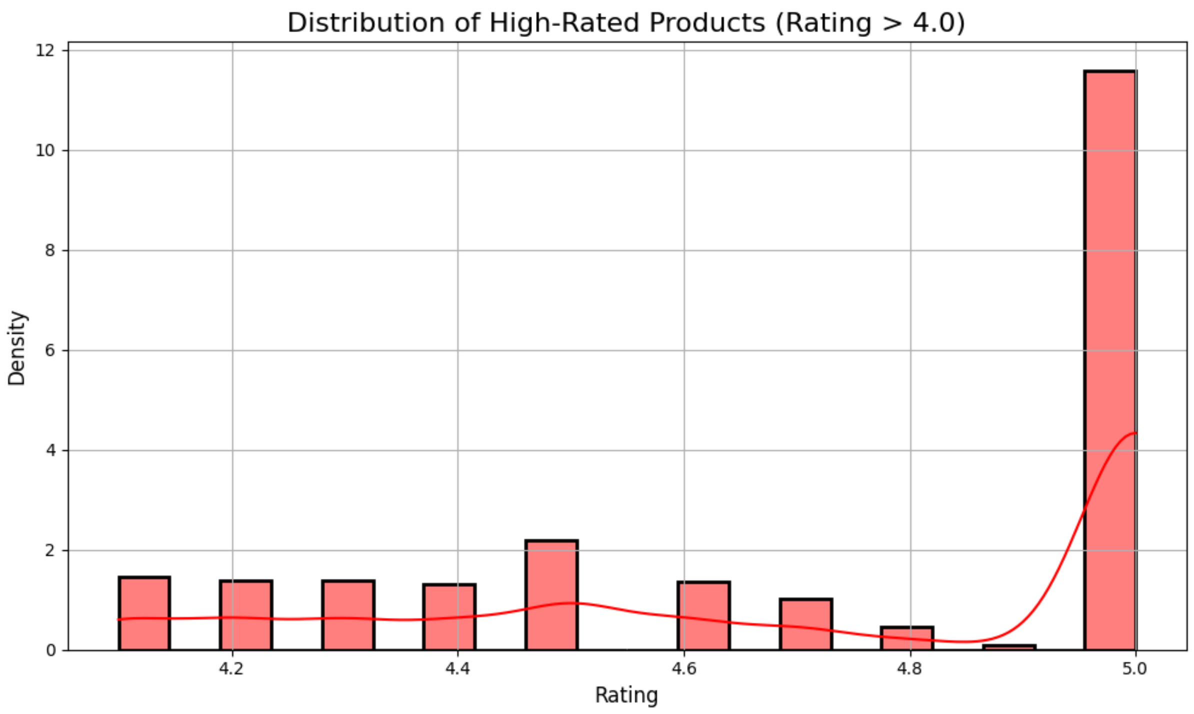
1. **Prime Products (Y)**:
    - 45% of the products in the dataset are **Amazon Prime eligible**.
    - These products likely benefit from faster shipping and other Prime benefits.
2. **Non-Prime Products (N)**:
    - 55% of the products are **not eligible for Amazon Prime**.
    - These might have longer shipped times or lack the benefits provided to Prime customers.
3. **Distribution**:
    - The chart shows that a slight majority of products (55%) are not available with Prime benefits, suggesting a relatively balanced distribution.

Distribution of Ratings for High-Rated Products

The distribution of ratings for high-rated products focuses on analysing products that have received high user ratings, typically 4.0 or above, on a 5-point scale**.**

Purpose distribution of ratings for high-rated products

Understand User Preferences, Assess Product Quality Consistency, Support Marketing Strategies.

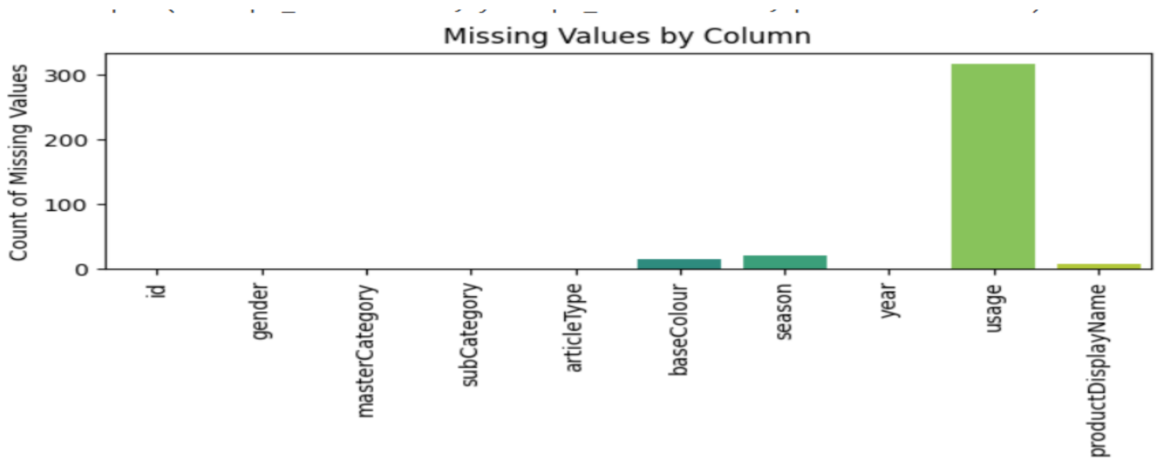


Visualization for a second Dataset to use collaborative filtering

Code:

```
# Missing Values Visualization
plt.subplot(2, 2, 1)
sns.barplot(x=missing_values.index, y=missing_values.values, palette="viridis")
plt.xticks(rotation=90)
plt.title("Missing Values by Column")
plt.ylabel("Count of Missing Values")
plt.xlabel("Columns")
```

Output:



Analysis:

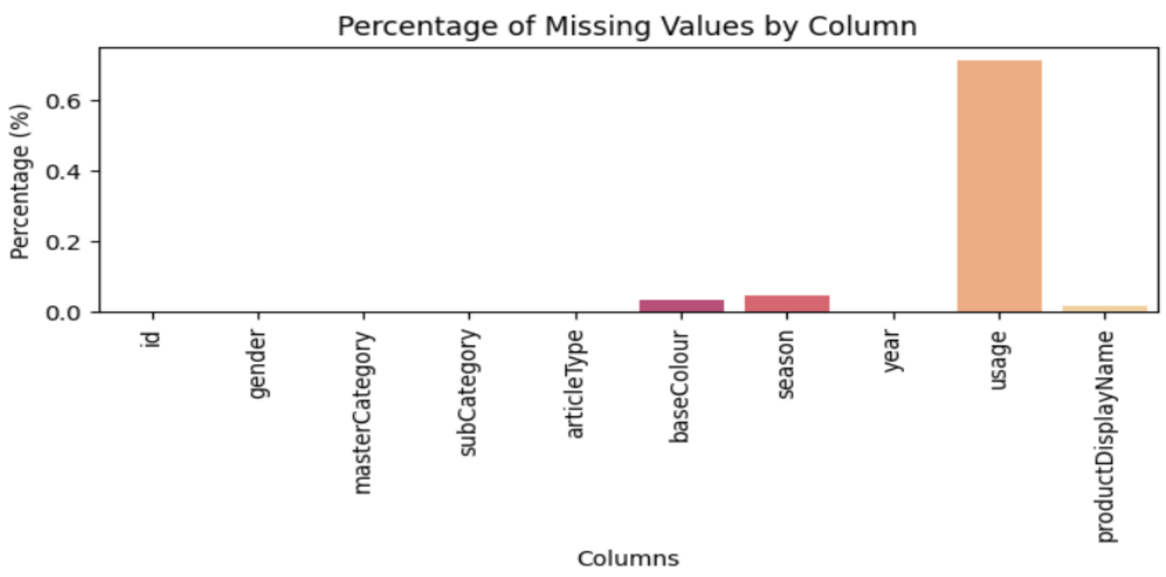
This visualization is a bar chart showing the count of missing values across different columns in a dataset. Let me analyse the key insights:

1. Most columns have few or no missing values, which is positive for data quality
2. However, there's a significant spike in missing values for the "usage" column, with approximately 300 missing entries

Code:

```
# Missing Percentage Visualization
plt.subplot(2, 2, 2)
sns.barplot(x=missing_percentage.index, y=missing_percentage.values, palette="magma")
plt.xticks(rotation=90)
plt.title("Percentage of Missing Values by Column")
plt.ylabel("Percentage (%)")
plt.xlabel("Columns")
```

Output:



Analysis:

1. The "usage" column has the highest percentage of missing values, at approximately 70% (0.7 or 70% on the y-axis)

2.Most other columns show extremely low or zero percentages of missing values:

- base Colour and season have very minimal missing values (less than 5%)
- All other columns (id, gender, master Category, Subcategory, article Type, product DisplayName, etc.) appear to have 0% missing values

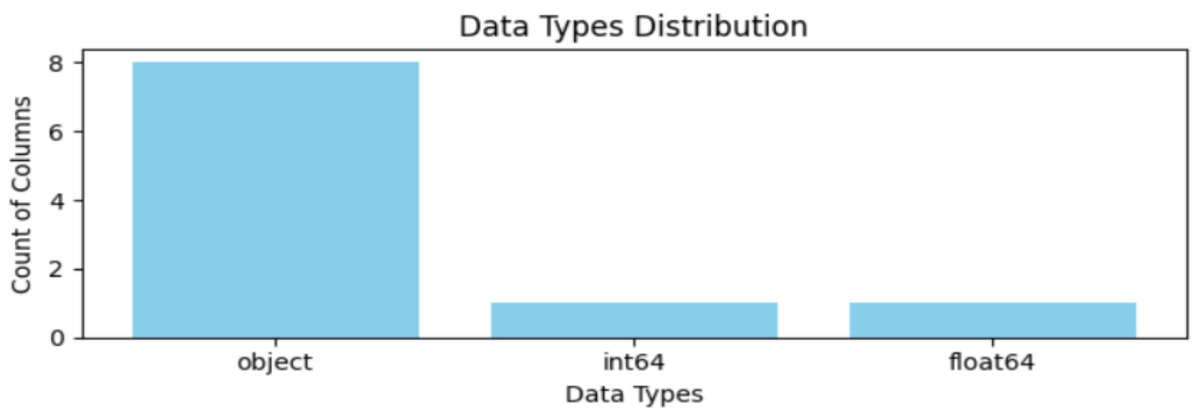
3. This percentage view gives us a better perspective on the severity of the missing data issue:

- While we saw ~300 missing values for "usage" in the previous chart, now we can see this represents about 70% of all entries
- This is a critically high percentage of missing data that could significantly impact any analysis using this column

Code:

```
# Data Types Distribution
plt.subplot(2, 2, 3)
plt.bar(data_types.index.astype(str), data_types.values, color="skyblue")
plt.title("Data Types Distribution")
plt.ylabel("Count of Columns")
plt.xlabel("Data Types")
```

Output:



Analysis:

1. There are three different data types present:

- object (string/categorical data)
- int64 (64-bit integers)
- float64 (64-bit floating-point numbers)

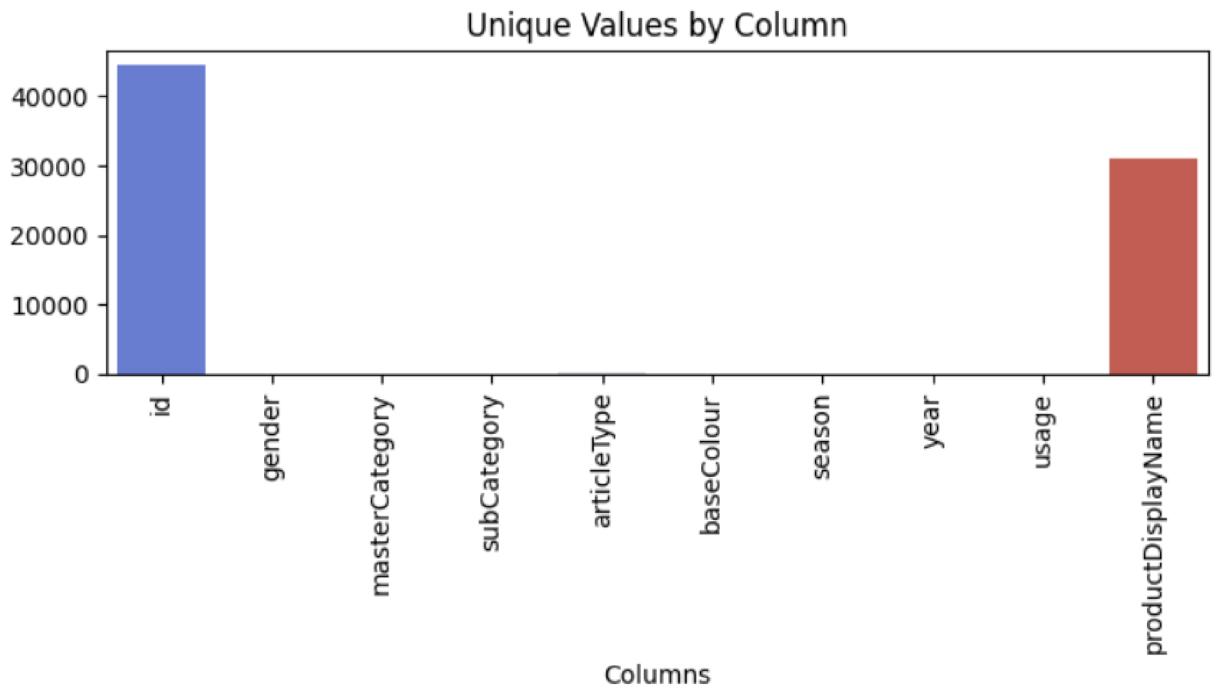
2. Distribution of columns by data type:

- object: Approximately 8 columns
- int64: 1 column
- float64: 1 column

Code:

```python
# Unique Values Distribution
plt.subplot(2, 2, 4)
sns.barplot(x=unique_values.index, y=unique_values.values, palette="coolwarm")
plt.xticks(rotation=90)
plt.title("Unique Values by Column")
plt.ylabel("Number of Unique Values")
plt.xlabel("Columns")
```

Output:



Analysis:

1.Two columns stand out with very high numbers of unique values:

- 'id': approximately 40,000+ unique values
- 'Product DisplayName': approximately 30,000+ unique values

2. All other columns have very few unique values in comparison, appearing almost flat at the bottom of the chart. This suggests:

- These columns are likely categorical with a limited set of possible values
- Fields like gender, master Category, Subcategory, article Type, base Colour, and season probably contain a small set of predefined categories

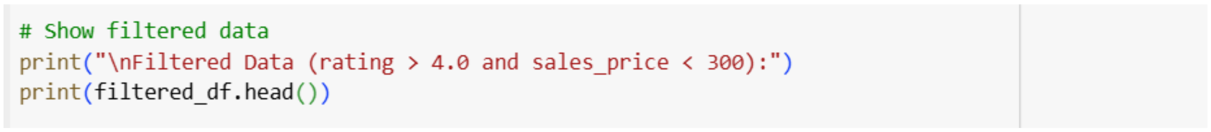
Basic Exploratory Data Analysis (EDA) for AI STYLIST

Understand the Dataset

Inspect Data Structure: Use functions like head(), info(), and describe() to examine the dataset's size, structure, and types of variables.

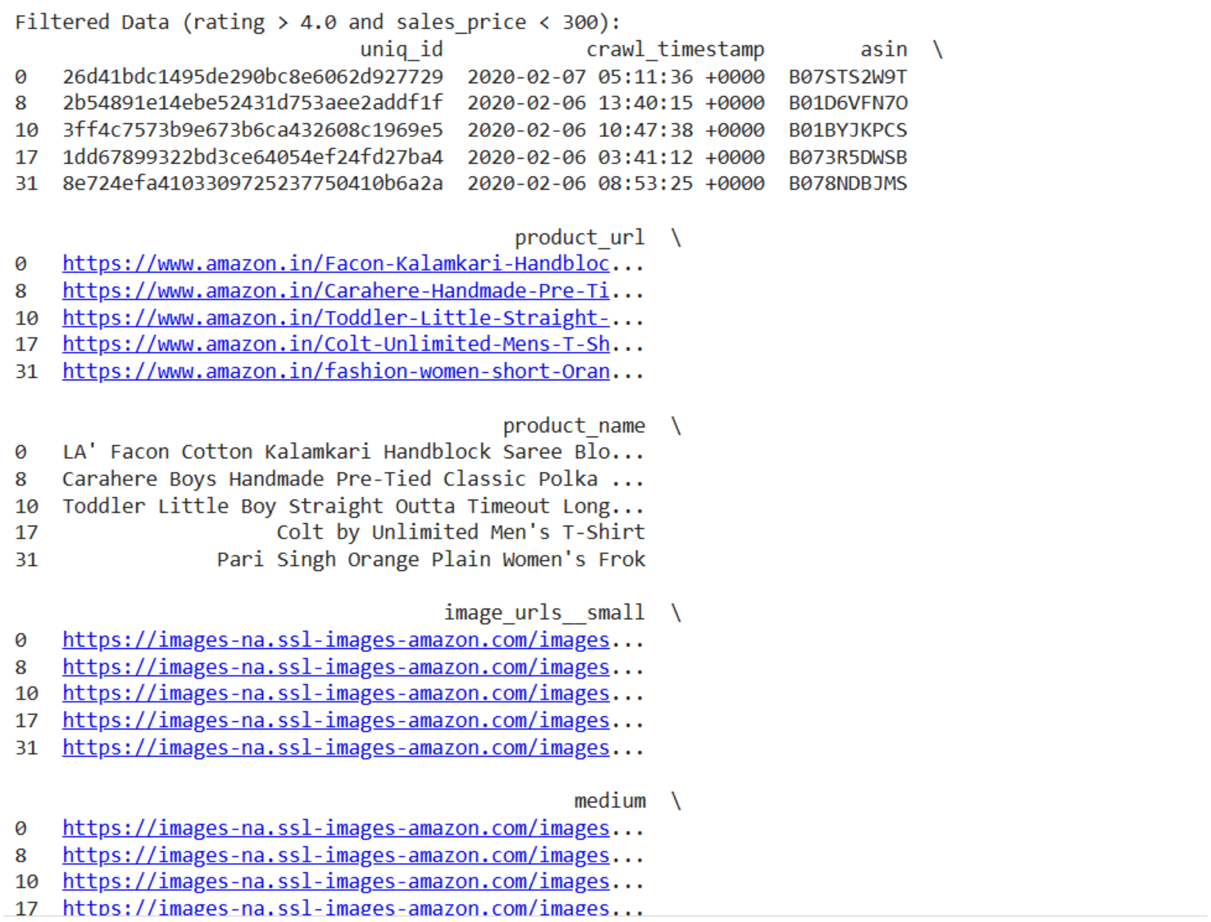
Display the head of the first Dataset

Code:

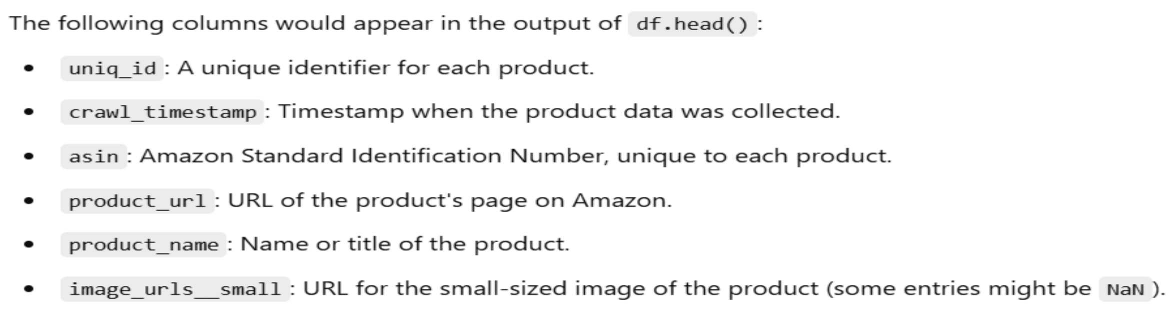


```
# Show filtered data
print("\nFiltered Data (rating > 4.0 and sales_price < 300):")
print(filtered_df.head())
```

Output:



```
Filtered Data (rating > 4.0 and sales_price < 300):
                            uniq_id            crawl_timestamp        asin  \
0   26d41bdc1495de290bc8e6062d927729  2020-02-07 05:11:36 +0000  B07STS2W9T
8   2b54891e14ebe52431d753aee2addf1f  2020-02-06 13:40:15 +0000  B01D6VFN7O
10  3ff4c7573b9e673b6ca432608c1969e5  2020-02-06 10:47:38 +0000  B01BYJKPCS
17  1dd67899322bd3ce64054ef24fd27ba4  2020-02-06 03:41:12 +0000  B073R5DWSB
31  8e724efa4103309725237750410b6a2a  2020-02-06 08:53:25 +0000  B078NDBJMS

                                          product_url  \
0   https://www.amazon.in/Facon-Kalamkari-Handbloc...
8   https://www.amazon.in/Carahere-Handmade-Pre-Ti...
10  https://www.amazon.in/Toddler-Little-Straight-...
17  https://www.amazon.in/Colt-Unlimited-Mens-T-Sh...
31  https://www.amazon.in/fashion-women-short-Oran...

                                         product_name  \
0   LA' Facon Cotton Kalamkari Handblock Saree Blo...
8   Carahere Boys Handmade Pre-Tied Classic Polka ...
10  Toddler Little Boy Straight Outta Timeout Long...
17                      Colt by Unlimited Men's T-Shirt
31                 Pari Singh Orange Plain Women's Frok

                                    image_urls__small  \
0   https://images-na.ssl-images-amazon.com/images...
8   https://images-na.ssl-images-amazon.com/images...
10  https://images-na.ssl-images-amazon.com/images...
17  https://images-na.ssl-images-amazon.com/images...
31  https://images-na.ssl-images-amazon.com/images...

                                               medium  \
0   https://images-na.ssl-images-amazon.com/images...
8   https://images-na.ssl-images-amazon.com/images...
10  https://images-na.ssl-images-amazon.com/images...
17  https://images-na.ssl-images-amazon.com/images...
```


The following columns would appear in the output of `df.head()`:

- `uniq_id`: A unique identifier for each product.
- `crawl_timestamp`: Timestamp when the product data was collected.
- `asin`: Amazon Standard Identification Number, unique to each product.
- `product_url`: URL of the product's page on Amazon.
- `product_name`: Name or title of the product.
- `image_urls__small`: URL for the small-sized image of the product (some entries might be `NaN`).

Handle Missing Data

Handling missing data ensures the dataset is clean, reliable, and ready for analysis or modelling. By choosing appropriate strategies based on the dataset's characteristics, we can minimize data bias, preserve information, and maintain the integrity of subsequent analyses.

Code:

```python
# Load the CSV file into a DataFrame
df = pd.read_csv('fashion_products_data.csv', low_memory=False)

# Display the missing values before cleaning (for debugging)
missing_before = df.isnull().sum().sum()
print(f"Missing values before cleaning: {missing_before}")

# Identify string columns and fill missing values with 'Unknown'
string_cols = df.select_dtypes(include=['object']).columns
df[string_cols] = df[string_cols].fillna('Unknown')  # For string columns

# Identify numerical columns and fill missing values with 0
numerical_cols = df.select_dtypes(include=['number']).columns
df[numerical_cols] = df[numerical_cols].fillna(0)    # For numerical columns

# Handle complex JSON-like columns
for col in df.select_dtypes(include=['object']):
    try:
        # Try parsing as JSON
        sampleValue = df[col].dropna().iloc[0]
        json.loads(sampleValue)  # Check if it's JSON
        # Assign {} or [] to fill missing JSON-like values
        df[col] = df[col].fillna('{}' if isinstance(sampleValue, dict) else '[]')
    except:
        continue

# Display missing values after cleaning
missing_after = df.isnull().sum().sum()
print(f"Missing values after cleaning: {missing_after}")

# Save cleaned DataFrame
df.to_csv('cleaned_fashion_products_data.csv', index=False)
```

```
Missing values before cleaning: 316610
Missing values after cleaning: 0
```

Code:

```python
# Calculate and display missing values with percentages
missing_values = data.isnull().sum()
total_rows = data.shape[0]
missing_percentage = (missing_values / total_rows) * 100

# Print missing values and their percentages
print("Missing Values Report:")
for column, value in missing_values.items():
    percentage = missing_percentage[column]
    print(f"{column}: {value} missing values ({percentage:.2f}%)")
```

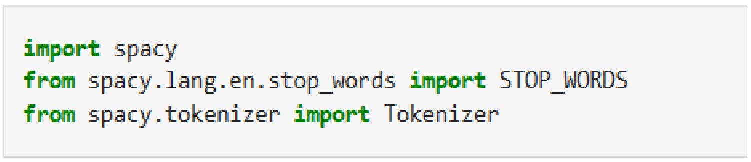
```
Missing Values Report:
id: 0 missing values (0.00%)
gender: 0 missing values (0.00%)
masterCategory: 0 missing values (0.00%)
subCategory: 0 missing values (0.00%)
articleType: 0 missing values (0.00%)
baseColour: 15 missing values (0.03%)
season: 21 missing values (0.05%)
year: 1 missing values (0.00%)
usage: 317 missing values (0.71%)
productDisplayName: 7 missing values (0.02%)
```

Preprocessing Steps

Install SpaCy and Required Model for Text Preprocessing

By installing SpaCy and using its models, you can streamline text preprocessing, enabling better insights and improved AI model performance.

Code:

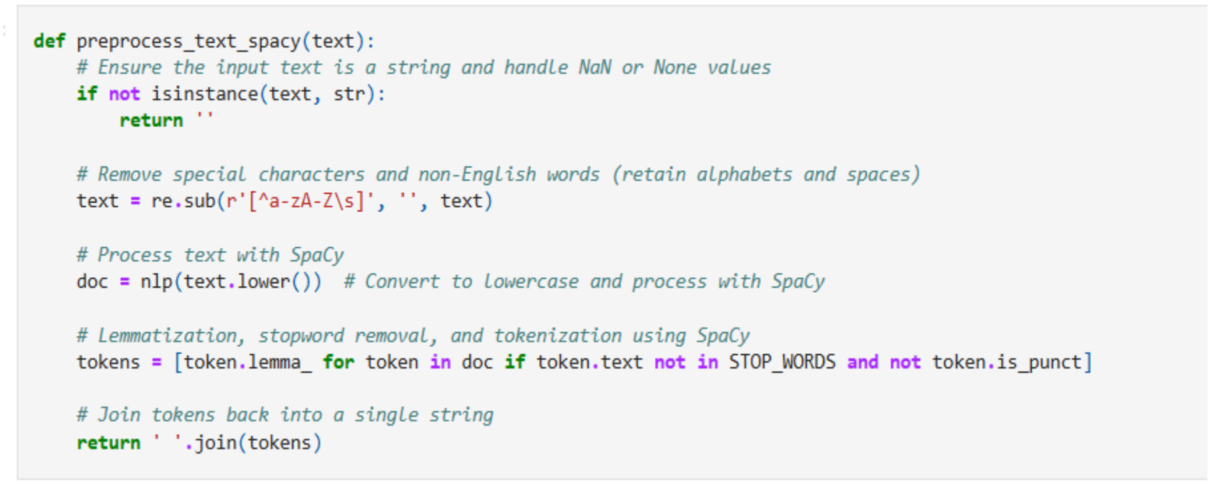


```python
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from spacy.tokenizer import Tokenizer
```

Load SpaCy Language Model

Define Text Preprocessing Function Using SpaCy

Code:



```python
def preprocess_text_spacy(text):
    # Ensure the input text is a string and handle NaN or None values
    if not isinstance(text, str):
        return ''

    # Remove special characters and non-English words (retain alphabets and spaces)
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Process text with SpaCy
    doc = nlp(text.lower())  # Convert to lowercase and process with SpaCy

    # Lemmatization, stopword removal, and tokenization using SpaCy
    tokens = [token.lemma_ for token in doc if token.text not in STOP_WORDS and not token.is_punct]

    # Join tokens back into a single string
    return ' '.join(tokens)
```

Check Available Columns and Apply Preprocessing to Relevant Column

Code:

```
# Check available columns
print("Available Columns in the DataFrame:")
print(df.columns)

if 'product_name' in df.columns:
    # Apply the text preprocessing function to the 'product_name' column
    df['product_name'] = df['product_name'].apply(preprocess_text_spacy)
    print("Preprocessing applied to 'product_name' column.")
else:
    print("'product_name' column not found. Please check the available columns.")
    print(df.columns)
```

Output:

```
Available Columns in the DataFrame:
Index(['uniq_id', 'crawl_timestamp', 'asin', 'product_url', 'product_name',
       'image_urls__small', 'medium', 'large', 'browsenode', 'brand',
       'sales_price', 'weight', 'rating', 'sales_rank_in_parent_category',
       'sales_rank_in_child_category', 'delivery_type', 'meta_keywords',
       'amazon_prime__y_or_n', 'parent___child_category__all',
       'best_seller_tag__y_or_n', 'other_items_customers_buy',
       'product_details__k_v_pairs', 'discount_percentage', 'colour',
       'no__of_reviews', 'seller_name', 'seller_id', 'left_in_stock',
       'no__of_offers', 'no__of_sellers', 'technical_details__k_v_pairs',
       'formats___editions', 'name_of_author_for_books'],
      dtype='object')
Preprocessing applied to 'product_name' column.
```

Verify Preprocessing Results

Code:

```
# Show the first few rows of the DataFrame after applying preprocessing
print(df[['product_name']].head())
```

```
                                        product_name
0  la facon cotton kalamkari handblock saree blou...
1       sf jeans pantaloon men plain slim fit tshirt
2  lovista cotton gota patti tassel traditional p...
3              people men print regular fit tshirt
4  monte carlo grey solid cotton blend polo colla...
```

EDA for a Collaborative Filtering Dataset

Display the head ()

Code:

```
import pandas as pd # Import the pandas library and assign it to the alias 'pd'
data=pd.read_csv('styles.csv',on_bad_lines='skip')
data.head()
```

Output:

	id	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName
0	15970	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt
1	39386	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans
2	59263	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch
3	21379	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants
4	53759	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt

Unique Values:

Code:

```
for column in data.columns:
  print(f"Unique Vlues in column {column}: {data[column].unique()}")
  print(data['articleType'].unique())
```

Output:

```
Unique Vlues in column id: [15970 39386 59263 ... 18842 46694 51623]
['Shirts' 'Jeans' 'Watches' 'Track Pants' 'Tshirts' 'Socks' 'Casual Shoes'
 'Belts' 'Flip Flops' 'Handbags' 'Tops' 'Bra' 'Sandals' 'Shoe Accessories'
 'Sweatshirts' 'Deodorant' 'Formal Shoes' 'Bracelet' 'Lipstick' 'Flats'
 'Kurtas' 'Waistcoat' 'Sports Shoes' 'Shorts' 'Briefs' 'Sarees'
 'Perfume and Body Mist' 'Heels' 'Sunglasses' 'Innerwear Vests' 'Pendant'
 'Nail Polish' 'Laptop Bag' 'Scarves' 'Rain Jacket' 'Dresses'
 'Night suits' 'Skirts' 'Wallets' 'Blazers' 'Ring' 'Kurta Sets' 'Clutches'
 'Shrug' 'Backpacks' 'Caps' 'Trousers' 'Earrings' 'Camisoles' 'Boxers'
 'Jewellery Set' 'Dupatta' 'Capris' 'Lip Gloss' 'Bath Robe' 'Mufflers'
 'Tunics' 'Jackets' 'Trunk' 'Lounge Pants' 'Face Wash and Cleanser'
 'Necklace and Chains' 'Duffel Bag' 'Sports Sandals'
 'Foundation and Primer' 'Sweaters' 'Free Gifts' 'Trolley Bag'
```

- **Column "id" Unique Values**: [15970, 39386, 59263, ..., 18842, 46694, 51623]
  - Represents unique identifiers (e.g., product IDs or transaction IDs) in the column. Each ID is unique to a specific data entry, ensuring no duplication. **Column "Category" Unique Values**:

['Shirts', 'Jeans', 'Watches', ..., 'Duffel Bag', 'Sports Sandals']

  - Lists the distinct categories of fashion items in the dataset. It provides insights into the types of products available and their variety.

Output Analysis:

1. **Unique IDs**:
   - Helps verify data integrity by checking for duplicates or missing entries.
   - Ensures each record represents a single item or transaction.
2. **Categories**:
   - Indicates the diversity of product types in the dataset.

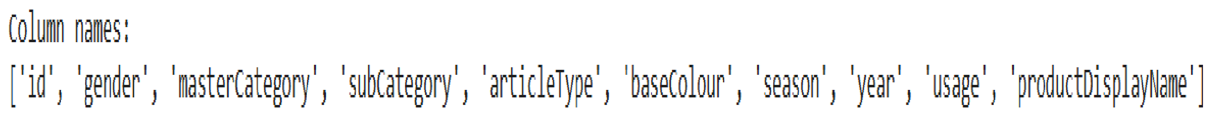
    - Useful for visualizations, grouping, and filtering during analysis.
    - Provides input for advanced techniques like one-hot encoding or clustering.

Display the Column Names:

Code:

```
print(f"Column names:\n{data.columns.tolist()}\n")
```

Output:



```
Column names:
['id', 'gender', 'masterCategory', 'subCategory', 'articleType', 'baseColour', 'season', 'year', 'usage', 'productDisplayName']
```

The **column names** represent the attributes or features of the dataset, with each column corresponding to a specific aspect of the data.

- **id**: A unique identifier for each record (e.g., product ID).
- **gender**: The target demographic (e.g., Men, Women, Unisex, Boys, Girls) the product is designed for.
- **master Category**: A high-level category grouping the product (e.g., Apparel, Footwear, Accessories).
- **Subcategory**: A more detailed category under master Category (e.g., Tops, Shoes, Jewellery).
- **article Type**: The specific type of product (e.g., Shirts, Sandals, Earrings).
- **base Colour**: The primary colour of the product (e.g., Black, Blue, Red).
- **season**: The intended seasonal usage of the product (e.g., Summer, Winter).
- **year**: The year the product was introduced or last updated in the dataset.
- **usage**: The functional purpose of the product (e.g., Casual, Sports, Formal).
- **Product DisplayName**: The descriptive name of the product for display purposes.

Display the Datatypes:

Code:

```
print(f"Data types:\n{data.dtypes}\n")
```

Output:

```
Data types:
id                     int64
gender                object
masterCategory        object
subCategory           object
articleType           object
baseColour            object
season                object
year                 float64
usage                 object
productDisplayName    object
dtype: object
```

Display the Missing Values:

The missing values report function analyses the dataset to identify and quantify the number of missing or null values in each column. It also calculates the percentage of missing values relative to the total number of rows, providing a comprehensive understanding of data completeness. This step is essential for data quality assessment during preprocessing.

Code:

```
# Calculate and display missing values with percentages
missing_values = data.isnull().sum()
total_rows = data.shape[0]
missing_percentage = (missing_values / total_rows) * 100

# Print missing values and their percentages
print("Missing Values Report:")
for column, value in missing_values.items():
    percentage = missing_percentage[column]
    print(f"{column}: {value} missing values ({percentage:.2f}%)")
```

Output:

```
Missing Values Report:
id: 0 missing values (0.00%)
gender: 0 missing values (0.00%)
masterCategory: 0 missing values (0.00%)
subCategory: 0 missing values (0.00%)
articleType: 0 missing values (0.00%)
baseColour: 15 missing values (0.03%)
season: 21 missing values (0.05%)
year: 1 missing values (0.00%)
usage: 317 missing values (0.71%)
productDisplayName: 7 missing values (0.02%)
```

1. **Identification**:
   - The code checks each column in the dataset for missing (null) values using methods like isnull() or isna() in libraries such as pandas.

2. **Quantification**:
    - It counts the number of missing values in each column.
    - Calculates the percentage of missing data relative to the total dataset size.
3. **Output Interpretation**:
    - For each column, it outputs:
        - The name of the column.
        - The count of missing values.
        - The percentage of missing data.
4. **Purpose**:
    - To highlight which columns, require attention for data cleaning.
    - To help decide the appropriate handling strategy for missing values (e.g., removal, imputation).

Output Analysis:

Key Observations:

1. **Columns with No Missing Values**:
    - Columns such as id, gender, master Category, and Subcategory have **0 missing values**, indicating they are complete and ready for analysis.
2. **Columns with Minor Missing Values**:
    - base Colour: 15 missing values (0.03%).
    - season: 21 missing values (0.05%).
    - Product DisplayName: 7 missing values (0.02%).
    - The low percentage suggests minimal impact, and imputation with mode or removal of rows may suffice.
3. **Columns with Noticeable Missing Values**:
    - usage: 317 missing values (0.71%).
    - Although under 1%, this is relatively higher and may require a more thoughtful handling approach, such as imputing based on related features.
4. **Critical Column**:
    - year: 1 missing value (0.00%).
    - Although the count is very small, if this column is crucial for analysis, care must be taken to fill the missing value meaningfully.

Removing Stopwords and Special Characters

The objective of this step is to preprocess the textual data within the product_name column by eliminating unnecessary words (commonly known as stopwords) and special characters. This helps in improving the quality of the data and ensures that only meaningful words remain for further analysis or processing. In the context of product names, this cleaning process is essential to focus on the key features of the product rather than filler words or extraneous symbols that might be irrelevant for analysis or machine learning tasks.

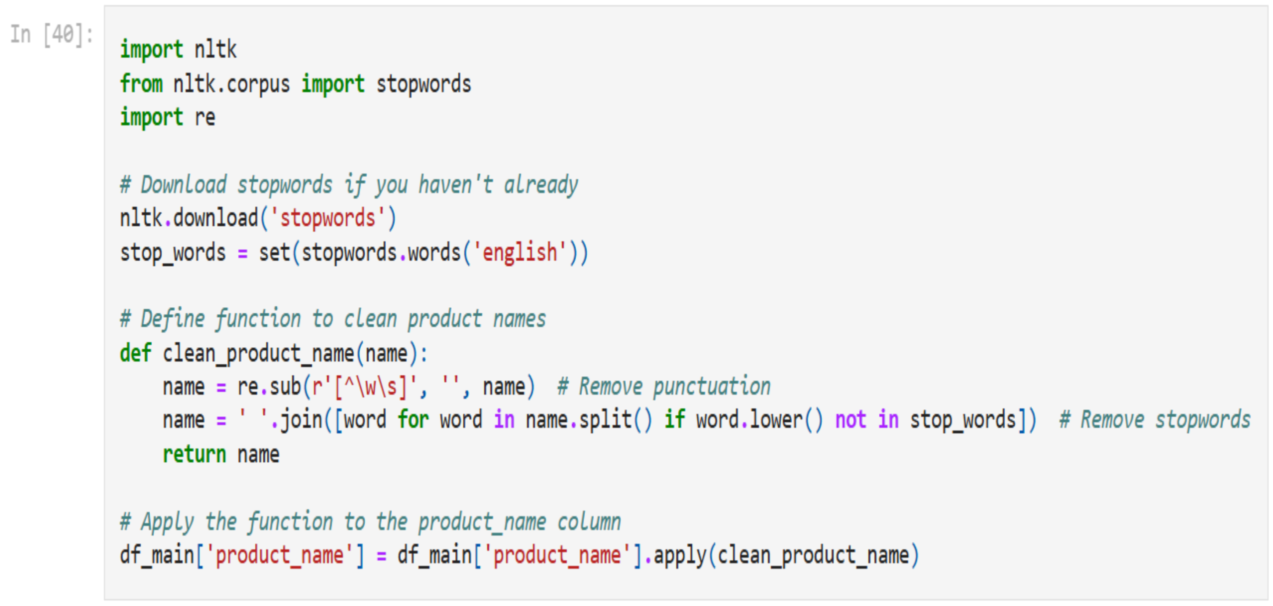
The cleaning process involves two key actions:

1. **Removing Special Characters**: Special characters like punctuation marks (e.g., !, #, %) are removed from the product names. These characters do not contribute to understanding the product name and are irrelevant for most text analysis tasks.

2. **Removing Stopwords**: Common stopwords, such as "and", "the", "is", are filtered out. These words are frequently used but provide little to no value in understanding the key features of the product. Removing them helps to focus on more meaningful and unique terms in the product names.

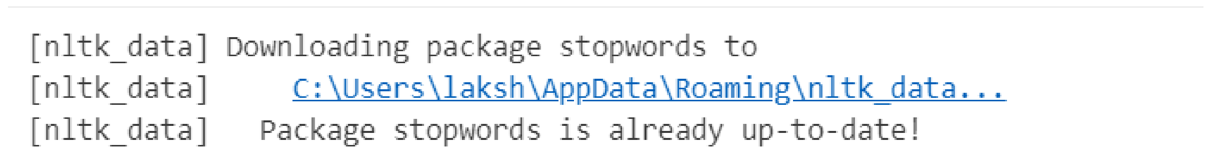
3. The cleaned product names are processed through a function called clean_product_name, which performs both actions on each product name. This ensures that the data is ready for further analysis, such as feature extraction or model building.

Code :



```
In [40]:
import nltk
from nltk.corpus import stopwords
import re

# Download stopwords if you haven't already
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Define function to clean product names
def clean_product_name(name):
    name = re.sub(r'[^\w\s]', '', name)  # Remove punctuation
    name = ' '.join([word for word in name.split() if word.lower() not in stop_words])  # Remove stopwords
    return name

# Apply the function to the product_name column
df_main['product_name'] = df_main['product_name'].apply(clean_product_name)
```

Output:



```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\laksh\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Converting Short Forms to Their Corresponding Long Forms in product_name

This step standardizes the product names in the dataset by converting abbreviations or short forms into their corresponding long forms. The primary goal of this task is to enhance the clarity and consistency of the text data. Abbreviations like "TShirt" or "Mens" often appear in the product_name column, which can make the data ambiguous or inconsistent. By replacing these with their full forms, the text becomes more descriptive and uniform, facilitating better analysis.

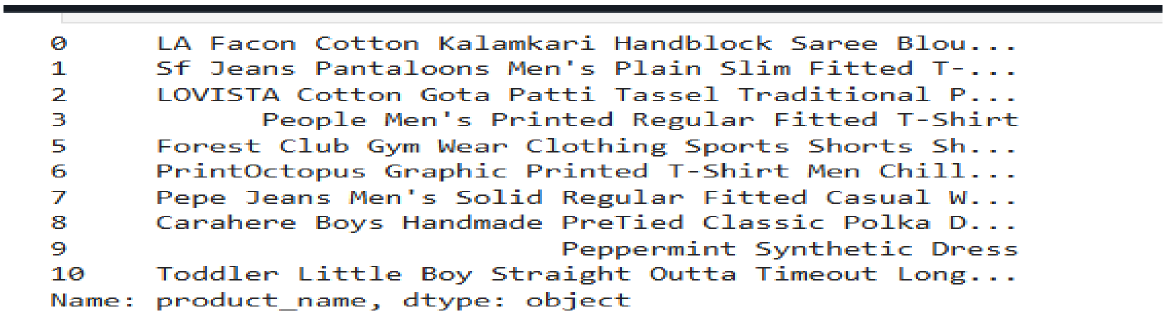
The method involves the following key steps:

1. **Define Abbreviation Mappings**
   A dictionary is created with abbreviations as keys and their corresponding long forms as values. Examples include:
   - "TShirt" to "T-Shirt"
   - "Mens" to "Men's"
   - "S/S" to "Short Sleeve"
2. **Process Product Names**
   A function named replace_abbreviations is written to process each product name. The function uses regular expressions to identify abbreviations and replace them with their long forms in a case-insensitive manner.
3. **Update Dataset**
   The processed product names are then updated in the product_name column of the dataset.

Code:

```python
import re

# Dictionary of common short forms and their corresponding long forms
abbreviations = {
    "TShirt": "T-Shirt",
    "Unisex": "Unisex Clothing",
    "Mens": "Men's",
    "Womens": "Women's",
    "Gym Wear": "Gym Wear Clothing",
    "Casual": "Casual Wear",
    "Fit": "Fitted",
    "Reg": "Regular",
    "Polo": "Polo Shirt",
    "Shirt": "Shirt",
    "S/S": "Short Sleeve",
    "L/S": "Long Sleeve",
    "Belt": "Belt Accessory"
}

# Function to replace abbreviations with long forms
def replace_abbreviations(product_name):
    for short, long in abbreviations.items():
        # Using regex to replace words only if they match exactly (case-insensitive)
        product_name = re.sub(rf'\b{short}\b', long, product_name, flags=re.IGNORECASE)
    return product_name

# Apply the function to the 'product_name' column
df_main['product_name'] = df_main['product_name'].apply(replace_abbreviations)

# Display the updated product names to confirm changes
print(df_main['product_name'].head(10))
```

OUTPUT:



```
0      LA Facon Cotton Kalamkari Handblock Saree Blou...
1      Sf Jeans Pantaloons Men's Plain Slim Fitted T-...
2      LOVISTA Cotton Gota Patti Tassel Traditional P...
3            People Men's Printed Regular Fitted T-Shirt
5      Forest Club Gym Wear Clothing Sports Shorts Sh...
6      PrintOctopus Graphic Printed T-Shirt Men Chill...
7      Pepe Jeans Men's Solid Regular Fitted Casual W...
8      Carahere Boys Handmade PreTied Classic Polka D...
9                        Peppermint Synthetic Dress
10     Toddler Little Boy Straight Outta Timeout Long...
Name: product_name, dtype: object
```

Stemming Words in the product_name Column

To standardize the product names in the dataset and simplify the text data, stemming is applied to the product_name column. This step reduces words to their root forms, enhancing consistency and reducing variations caused by inflected or derived word forms.

Explanation

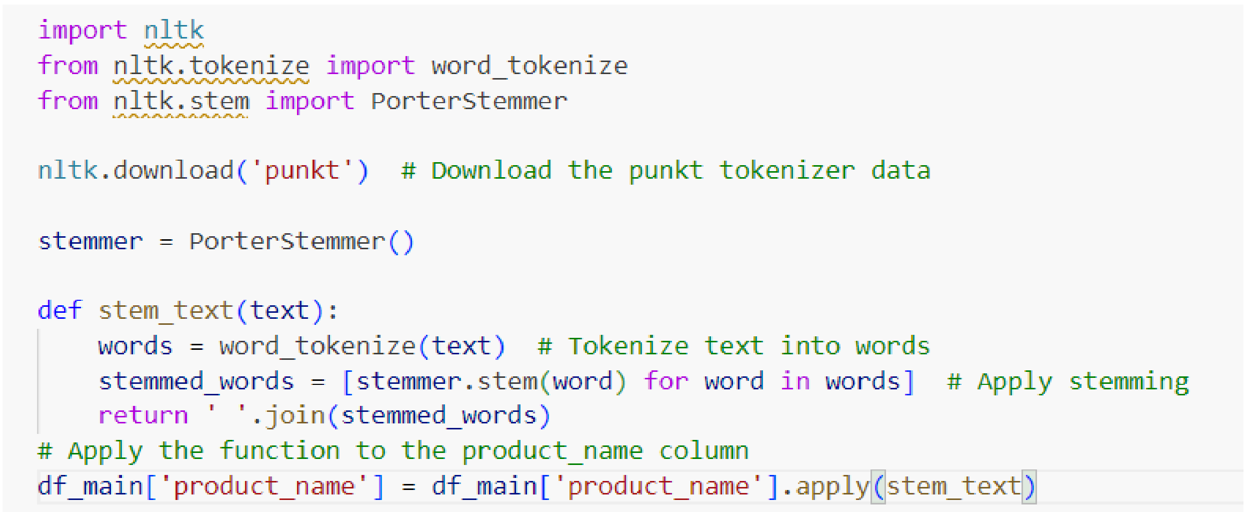
Stemming is a text preprocessing technique that reduces words to their base or root form. For instance, words like "running," "runner," and "ran" are reduced to their root form, "run." By applying stemming, the dataset becomes less variable and easier to analyze, particularly in text-based machine learning models or natural language processing tasks.

The NLTK library is used to tokenize the text into individual words and apply the stemming process to each word. The processed product names are then reassembled and updated in the product_name column.

Methodology

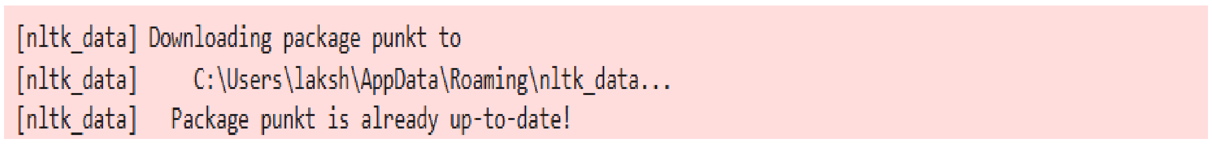
1. **Stemmer Initialization**
   The PorterStemmer from NLTK, a widely used algorithm, is employed for reducing words to their root forms.
2. **Text Tokenization**
   Product names are split into individual words using the word_tokenize function.
3. **Apply Stemming**
   Each word is processed to its root form (e.g., "Casual" to "casual," "Running" to "run").
4. **Dataset Update**
   Stemmed words are reassembled and stored back in the product_name column.

Code:



```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

nltk.download('punkt')  # Download the punkt tokenizer data

stemmer = PorterStemmer()

def stem_text(text):
    words = word_tokenize(text)  # Tokenize text into words
    stemmed_words = [stemmer.stem(word) for word in words]  # Apply stemming
    return ' '.join(stemmed_words)
# Apply the function to the product_name column
df_main['product_name'] = df_main['product_name'].apply(stem_text)
```

Output:



```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\laksh\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Content-Based Filtering

Content-based filtering is a machine learning technique widely used in recommendation systems that focuses on analyzing the attributes of items and matching them with user preferences or historical interactions. Unlike collaborative filtering, which relies on data from multiple users, content-based filtering generates recommendations based solely on the user's own interactions and the features of the items they have shown interest in. This method relies on attributes such as descriptions, categories, or metadata of the items, which are compared to the user's preferences to provide personalized suggestions. The key advantage of this approach is that it does not require information about other users, making it particularly robust for new users, thus solving the cold-start problem. Furthermore, since recommendations are derived directly from the item features, content-based filtering tends to be more explainable, allowing users to understand the rationale behind the suggestions.

Models Used in our AI Stylist Project

In our AI Stylist project, several models were employed to implement content-based filtering for personalized fashion recommendations. The models used include Bag-of-words, TF-IDF, Word2Vec, and VGG16.

- **Bag-of-Words**: Converts text into numerical features based on word frequency to identify item similarities.

- **TF-IDF**: Highlights unique words by reducing the impact of common ones, enhancing product description analysis.
- **Word2Vec**: Captures semantic word relationships, enabling recommendations based on contextual similarity.
- **VGG16**: Extracts visual features from images for recommending visually similar products.

By using these models in combination, the AI Stylist project is able to offer a comprehensive content-based filtering approach that leverages both textual and visual information to generate personalized, accurate fashion recommendations.

Bag-of-Words Model

Overview:

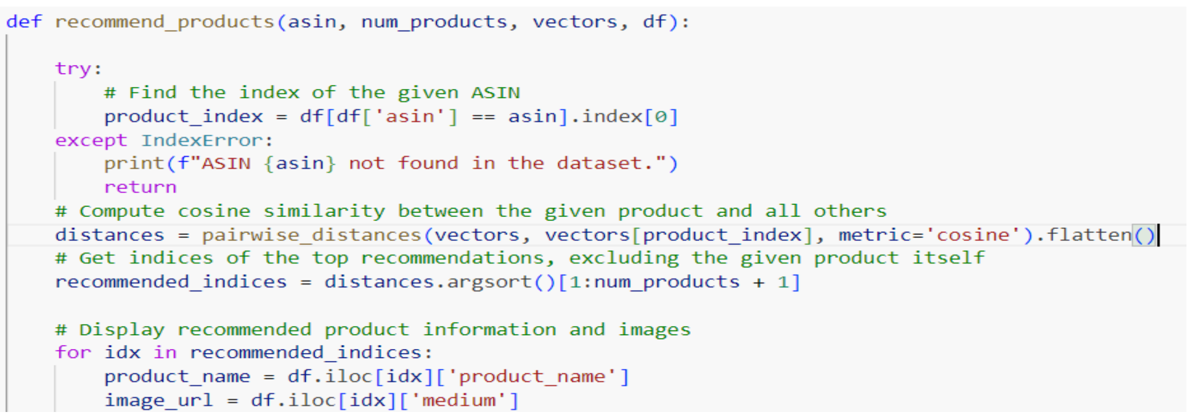
In our recommendation system, the Bag-of-Words (BoW) model plays a crucial role in identifying and suggesting similar products based on their textual descriptions. The implementation uses CountVectorizer to transform the product descriptions into numerical representations by counting the occurrences of each word. These vectors serve as a foundation for comparing products.

To recommend products, the system calculates cosine similarity using pairwise_distances. This metric measures the textual closeness between the selected product and others in the dataset. The product for which recommendations are required is identified using its ASIN, and the system retrieves the most similar products based on their BoW vectors.

Functionality of the Bag-of-Words Model

1. The CountVectorizer converts product descriptions into numerical vectors by counting word occurrences.
2. Cosine similarity is calculated using pairwise_distances to find products similar to the selected one.
3. The model retrieves and ranks the most similar products based on their textual attributes.
4. Product names and images are displayed after validating and accessing their URLs.
5. Robust error handling ensures smooth execution, managing missing data and invalid URLs effectively.

Code:



```python
def recommend_products(asin, num_products, vectors, df):

    try:
        # Find the index of the given ASIN
        product_index = df[df['asin'] == asin].index[0]
    except IndexError:
        print(f"ASIN {asin} not found in the dataset.")
        return
    # Compute cosine similarity between the given product and all others
    distances = pairwise_distances(vectors, vectors[product_index], metric='cosine').flatten()
    # Get indices of the top recommendations, excluding the given product itself
    recommended_indices = distances.argsort()[1:num_products + 1]

    # Display recommended product information and images
    for idx in recommended_indices:
        product_name = df.iloc[idx]['product_name']
        image_url = df.iloc[idx]['medium']
        try:
            # Skip empty or invalid URLs
            if not image_url or not image_url.startswith(('http://', 'https://')):
                print(f"No valid image URL for: {product_name}")
                continue
            # Check URL accessibility
            response = requests.get(image_url, stream=True)
            if response.status_code == 200 and 'image' in response.headers.get('Content-Type', ''):
                print(f"Recommended Product: {product_name}")
                display(Image(image_url))
            else:
                print(f"Image not available for: {product_name}")
        except Exception as e:
            print(f"Error displaying image for {product_name}: {e}")

# Example usage:
# Assuming 'df_main' is your DataFrame and 'product_name' is the text data
vectorizer = CountVectorizer()
bow_vectors = vectorizer.fit_transform(df_main['product_name'])

# Call the recommendation function using 'asin' as the identifier
recommend_products(asin='B07K6PHHHM', num_products=5, vectors=bow_vectors, df=df_main)
```

OUTPUT:

```
Recommended Product: winza design chiffon sare blous piec
```

```
Recommended Product: winza design chiffon sare blous piec
```

```
Recommended Product: winza design cotton sare blous piec
```

```
Recommended Product: winza design cotton sare blous piec
```

```
Recommended Product: winza design georgett sare blous piec
```

Overview of TF-IDF Model

In our recommendation system, the TF-IDF (Term Frequency-Inverse Document Frequency) model is used to enhance the quality of product suggestions by considering both the frequency of words within a product description and their rarity across the entire dataset. The TfidfVectorizer transforms product descriptions into weighted vectors, which highlight important words for comparison.
For recommending products, the system calculates the cosine similarity between the product's TF-IDF vector and the vectors of all other products in the dataset. The product with the closest similarity is then ranked, and the most relevant recommendations are displayed.

Formula:

TF-IDF (t, d) = TF (t, d) × IDF (t)

Where:

- TF (t, d) = count of t in d / number of words in d
- DF(t) = occurrence of t in documents
- DF(t) = N(t)
- Where
- DF(t) = Document frequency of a term t
- N(t) = Number of documents containing the term t
- IDF (t) = N/ DF(t) = N/N(t)
- IDF (t) = log(N/ DF(t))

How it works:

1. **TF-IDF Transformation:** The TfidfVectorizer converts product descriptions into weighted numerical vectors, reflecting the importance of terms in the context of the entire dataset.
2. **Cosine Similarity:** The cosine similarity between TF-IDF vectors measures the textual similarity between products.
3. **Ranking:** Products are ranked based on similarity scores, and the most similar ones are selected.
4. **Display Recommendations:** Product details such as name, URL, and image are shown to the user.
5. **Error Handling:** Missing product data and image issues are managed efficiently

Example:

For products with descriptions:

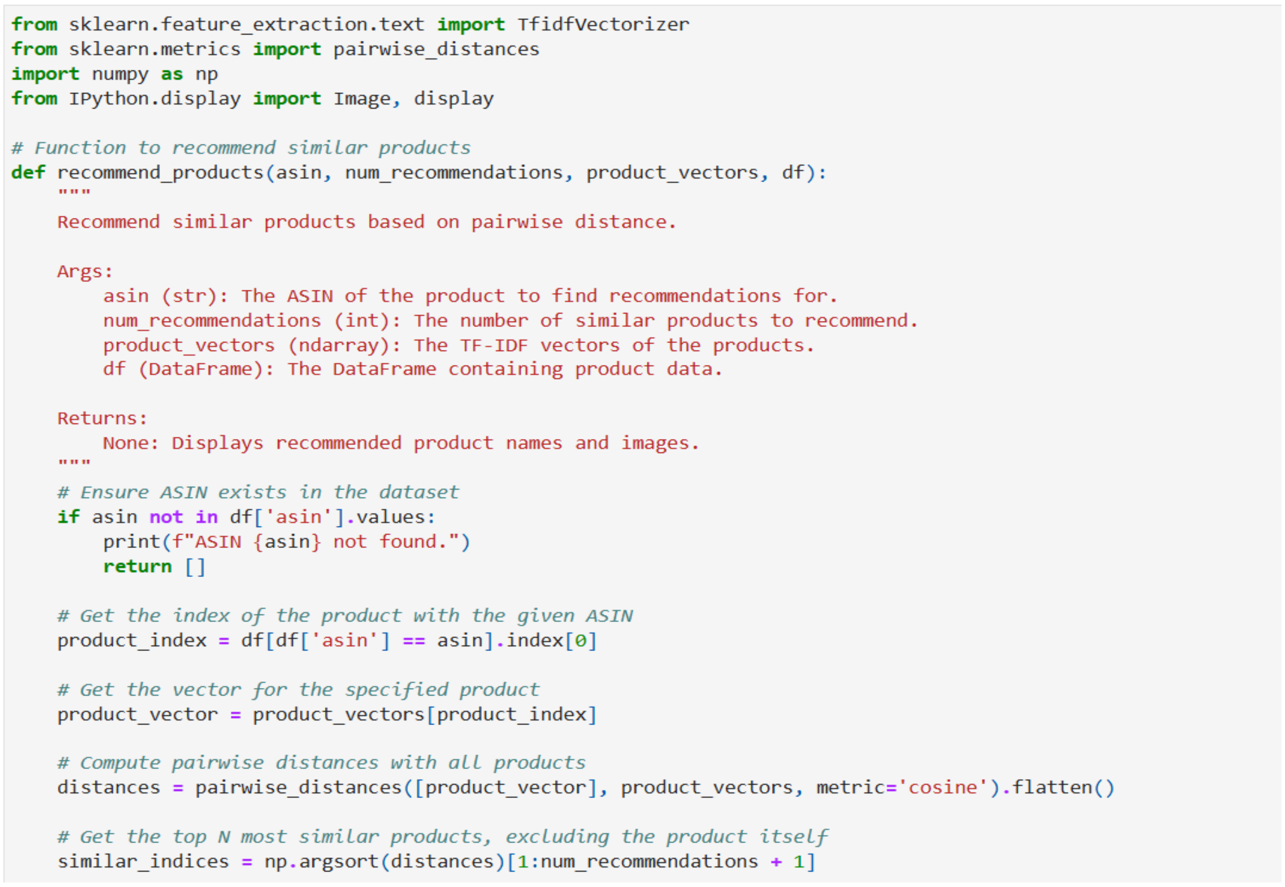
1.Product A: "Black leather watch"

2.Product B: "Blue leather watch"

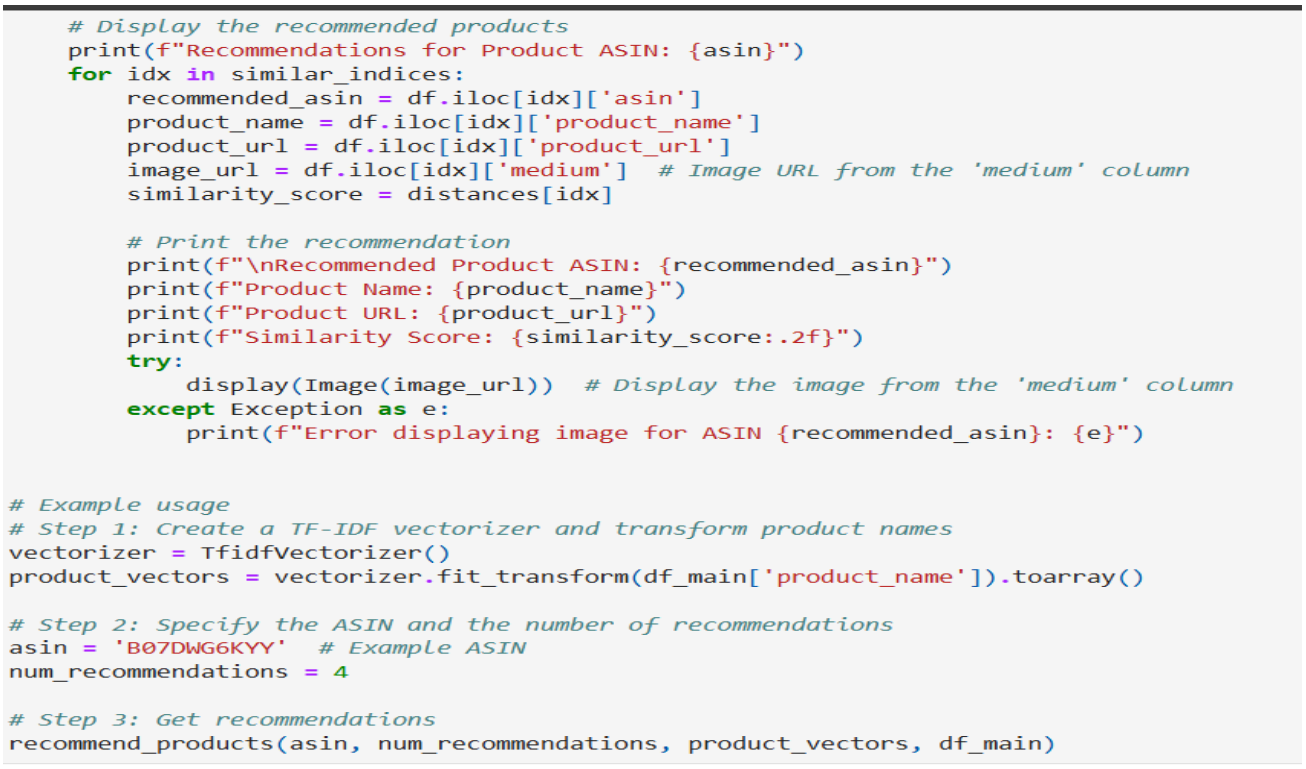
3.Product C: "Leather strap black watch"

- TF-IDF vectors are computed for each description.
- Cosine similarity is calculated, and similar products (e.g., Product A and Product C) are recommended based on their vector similarity.

- **Code:**



```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import pairwise_distances
import numpy as np
from IPython.display import Image, display

# Function to recommend similar products
def recommend_products(asin, num_recommendations, product_vectors, df):
    """
    Recommend similar products based on pairwise distance.

    Args:
        asin (str): The ASIN of the product to find recommendations for.
        num_recommendations (int): The number of similar products to recommend.
        product_vectors (ndarray): The TF-IDF vectors of the products.
        df (DataFrame): The DataFrame containing product data.

    Returns:
        None: Displays recommended product names and images.
    """
    # Ensure ASIN exists in the dataset
    if asin not in df['asin'].values:
        print(f"ASIN {asin} not found.")
        return []

    # Get the index of the product with the given ASIN
    product_index = df[df['asin'] == asin].index[0]

    # Get the vector for the specified product
    product_vector = product_vectors[product_index]

    # Compute pairwise distances with all products
    distances = pairwise_distances([product_vector], product_vectors, metric='cosine').flatten()

    # Get the top N most similar products, excluding the product itself
    similar_indices = np.argsort(distances)[1:num_recommendations + 1]
```



```python
    # Display the recommended products
    print(f"Recommendations for Product ASIN: {asin}")
    for idx in similar_indices:
        recommended_asin = df.iloc[idx]['asin']
        product_name = df.iloc[idx]['product_name']
        product_url = df.iloc[idx]['product_url']
        image_url = df.iloc[idx]['medium']  # Image URL from the 'medium' column
        similarity_score = distances[idx]

        # Print the recommendation
        print(f"\nRecommended Product ASIN: {recommended_asin}")
        print(f"Product Name: {product_name}")
        print(f"Product URL: {product_url}")
        print(f"Similarity Score: {similarity_score:.2f}")
        try:
            display(Image(image_url))  # Display the image from the 'medium' column
        except Exception as e:
            print(f"Error displaying image for ASIN {recommended_asin}: {e}")


# Example usage
# Step 1: Create a TF-IDF vectorizer and transform product names
vectorizer = TfidfVectorizer()
product_vectors = vectorizer.fit_transform(df_main['product_name']).toarray()

# Step 2: Specify the ASIN and the number of recommendations
asin = 'B07DWG6KYY'  # Example ASIN
num_recommendations = 4

# Step 3: Get recommendations
recommend_products(asin, num_recommendations, product_vectors, df_main)
```

OUTPUT:

```
Recommendations for Product ASIN: B07DWG6KYY

Recommended Product ASIN: B083QQK18K
Product Name: jevi print women 's cotton straight kurta
Product URL: https://www.amazon.in/Jevi-Prints-Womens-Straight-Kurti_K-1312_XXL_Beige_Xx-Large/dp/B07L5MDN7N/
Similarity Score: 0.29
```

```
Recommended Product ASIN: B083QGM3CS
Product Name: jevi print women 's cotton straight kurta
Product URL: https://www.amazon.in/Jevi-Prints-Straight-KURTI_S-1717_40_Pink-Grey_40/dp/B07L9D4KZ1/
Similarity Score: 0.29
```

```
Recommended Product ASIN: B083QMR6BG
Product Name: jevi print women 's cotton print straight kurta prj614
Product URL: https://www.amazon.in/Jevi-Prints-Womens-Straight-PRJ-614_XXL/dp/B07F1RLLVT/
Similarity Score: 0.46
```

```
Recommended Product ASIN: B075V9DCH1
Product Name: jevi print women 's cotton dress materi
Product URL: https://www.amazon.in/Jevi-Prints-Material-Saheli-1216-1505-1530_Free-Size_Multi-Coloured/dp/B075V9DCH1/
Similarity Score: 0.49
```

Overview of Word2Vec Model

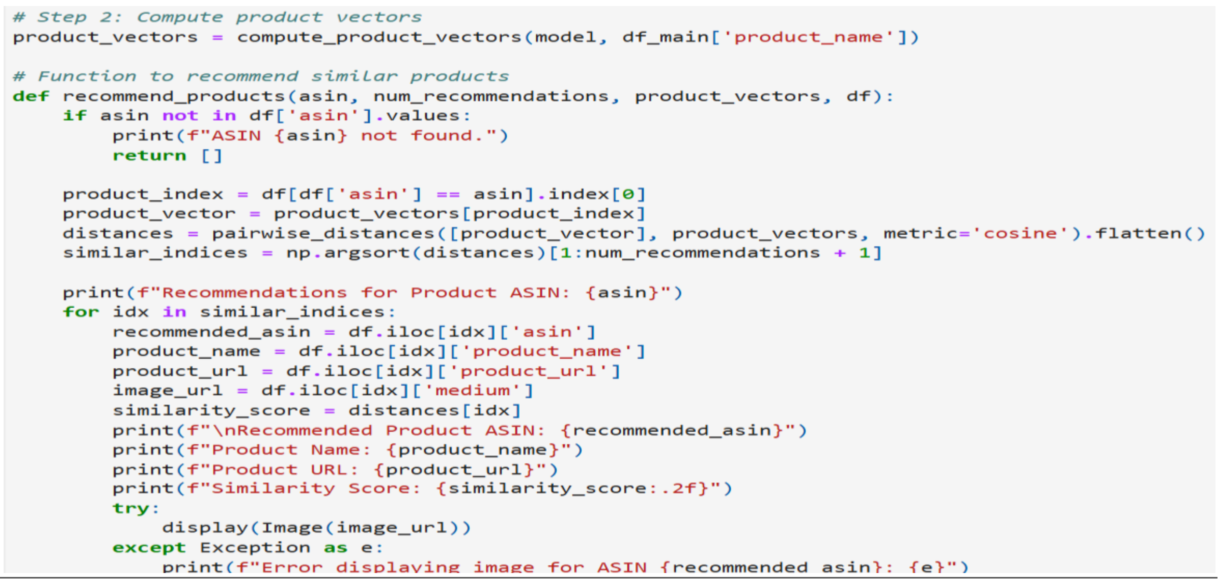
In our recommendation system, the Word2Vec model is used to compute product vectors based on the semantic meaning of words in product descriptions. The model transforms words into dense vectors that capture the context and relationships between words. For each product description, we generate a vector by averaging the Word2Vec vectors of the individual words. The system then computes the cosine similarity between the vectors of products to recommend similar items.
The advantage of Word2Vec over simpler models like Bag-of-Words or TF-IDF is that it captures the semantic relationships between words, which enhances the recommendation accuracy based on the meaning of the product names.

Functionality of the Word2Vec Model

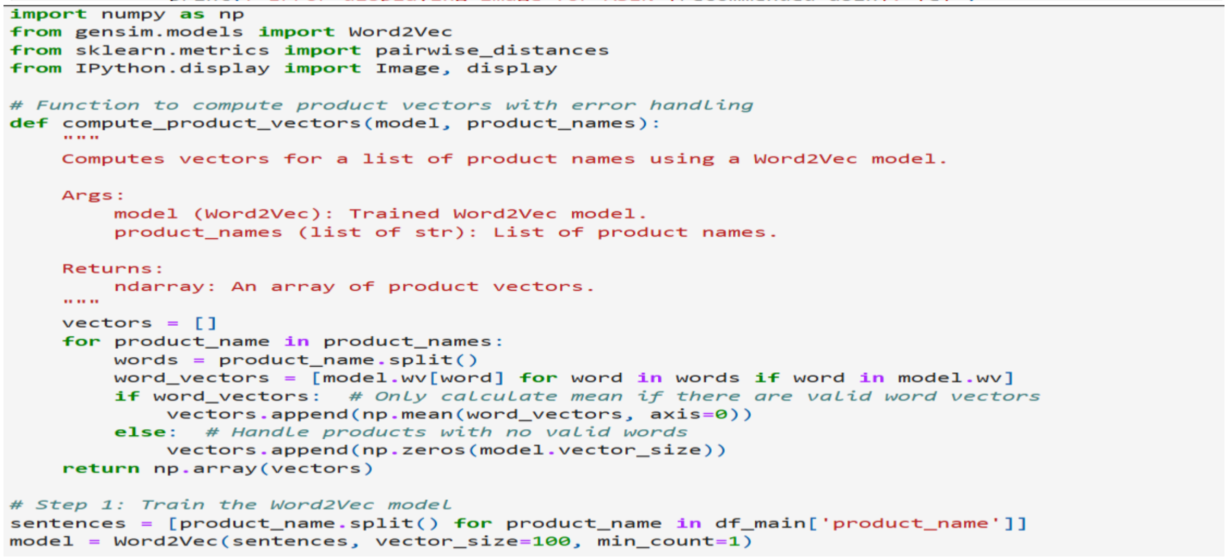
1. **Word Vector Creation**: The Word2Vec model is trained on the product descriptions to learn dense vector representations of words.

2. **Product Vector Computation**: For each product, its vector is calculated as the average of the Word2Vec vectors of the words in its name.
3. **Cosine Similarity**: The system calculates cosine similarity between the product's vector and all other product vectors to measure the semantic closeness.
4. **Product Recommendations**: The most similar products are identified and displayed, including their names, URLs, and images.
5. **Error Handling:** The system ensures robust error management, including handling cases where no valid word vectors are found for a product name.

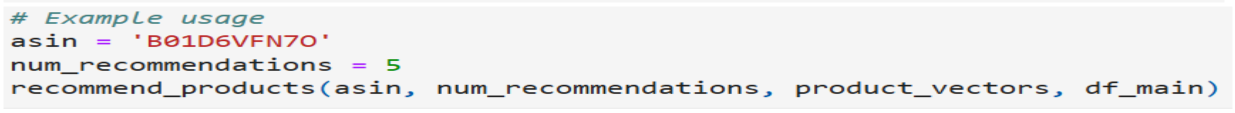
Code:



```python
# Step 2: Compute product vectors
product_vectors = compute_product_vectors(model, df_main['product_name'])

# Function to recommend similar products
def recommend_products(asin, num_recommendations, product_vectors, df):
    if asin not in df['asin'].values:
        print(f"ASIN {asin} not found.")
        return []

    product_index = df[df['asin'] == asin].index[0]
    product_vector = product_vectors[product_index]
    distances = pairwise_distances([product_vector], product_vectors, metric='cosine').flatten()
    similar_indices = np.argsort(distances)[1:num_recommendations + 1]

    print(f"Recommendations for Product ASIN: {asin}")
    for idx in similar_indices:
        recommended_asin = df.iloc[idx]['asin']
        product_name = df.iloc[idx]['product_name']
        product_url = df.iloc[idx]['product_url']
        image_url = df.iloc[idx]['medium']
        similarity_score = distances[idx]
        print(f"\nRecommended Product ASIN: {recommended_asin}")
        print(f"Product Name: {product_name}")
        print(f"Product URL: {product_url}")
        print(f"Similarity Score: {similarity_score:.2f}")
        try:
            display(Image(image_url))
        except Exception as e:
            print(f"Error displaying image for ASIN {recommended_asin}: {e}")
```

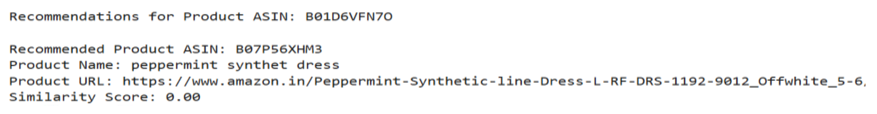


```python
import numpy as np
from gensim.models import Word2Vec
from sklearn.metrics import pairwise_distances
from IPython.display import Image, display

# Function to compute product vectors with error handling
def compute_product_vectors(model, product_names):
    """
    Computes vectors for a list of product names using a Word2Vec model.

    Args:
        model (Word2Vec): Trained Word2Vec model.
        product_names (list of str): List of product names.

    Returns:
        ndarray: An array of product vectors.
    """
    vectors = []
    for product_name in product_names:
        words = product_name.split()
        word_vectors = [model.wv[word] for word in words if word in model.wv]
        if word_vectors:  # Only calculate mean if there are valid word vectors
            vectors.append(np.mean(word_vectors, axis=0))
        else:  # Handle products with no valid words
            vectors.append(np.zeros(model.vector_size))
    return np.array(vectors)

# Step 1: Train the Word2Vec model
sentences = [product_name.split() for product_name in df_main['product_name']]
model = Word2Vec(sentences, vector_size=100, min_count=1)
```



```python
# Example usage
asin = 'B01D6VFN7O'
num_recommendations = 5
recommend_products(asin, num_recommendations, product_vectors, df_main)
```

OUTPUT:



```
Recommendations for Product ASIN: B01D6VFN7O

Recommended Product ASIN: B07P56XHM3
Product Name: peppermint synthet dress
Product URL: https://www.amazon.in/Peppermint-Synthetic-line-Dress-L-RF-DRS-1192-9012_Offwhite_5-6,
Similarity Score: 0.00
```

```
Recommended Product ASIN: B07P6DH68C
Product Name: peppermint synthet dress
Product URL: https://www.amazon.in/Peppermint-Synthetic-line-L-RF-DRS-1192-8967_Navy-Blue_8-9/dp/B07NBPY7DF/
Similarity Score: 0.00
```

```
Recommended Product ASIN: B07P5FC9ZV
Product Name: peppermint synthet dress
Product URL: https://www.amazon.in/Peppermint-Synthetic-line-Dress-L-RF-DRS-1192-9157_Offwhite_10-11,
Similarity Score: 0.00
```

```
Recommended Product ASIN: B07P43X53C
Product Name: peppermint synthet dress
Product URL: https://www.amazon.in/Peppermint-Synthetic-line-Dress-L-RF-DRS-1192-9098_Offwhite_4-5/dp/B0
Similarity Score: 0.00
```

Overview of Hybrid Recommendation Model Using Weighted Word2Vec and TF-IDF

In this recommendation system, we utilize a hybrid approach combining Word2Vec and TF-IDF to compute product vectors and enhance recommendation accuracy. The Word2Vec model captures the semantic meaning of words by representing them as dense vectors, while TF-IDF highlights the importance of words based on their frequency in a product description relative to the entire dataset. The system computes weighted product vectors by combining these two techniques, then calculates the cosine similarity between product vectors to recommend similar items.

This hybrid approach improves upon simpler models like Bag-of-Words or pure TF-IDF by incorporating both semantic context and term importance, leading to more accurate and meaningful recommendations.

Functionality of the Hybrid Model

1. **Word Vector Creation**: The Word2Vec model is trained on product descriptions to learn dense vector representations of words that capture their semantic meaning.

2. **TF-IDF Calculation**: The TF-IDF vectorizer is used to calculate the importance of each word in the product descriptions, giving more weight to significant terms.

3. **Weighted Product Vector Computation**: For each product, its vector is calculated by averaging the Word2Vec vectors of the individual words, weighted by their respective TF-IDF scores.

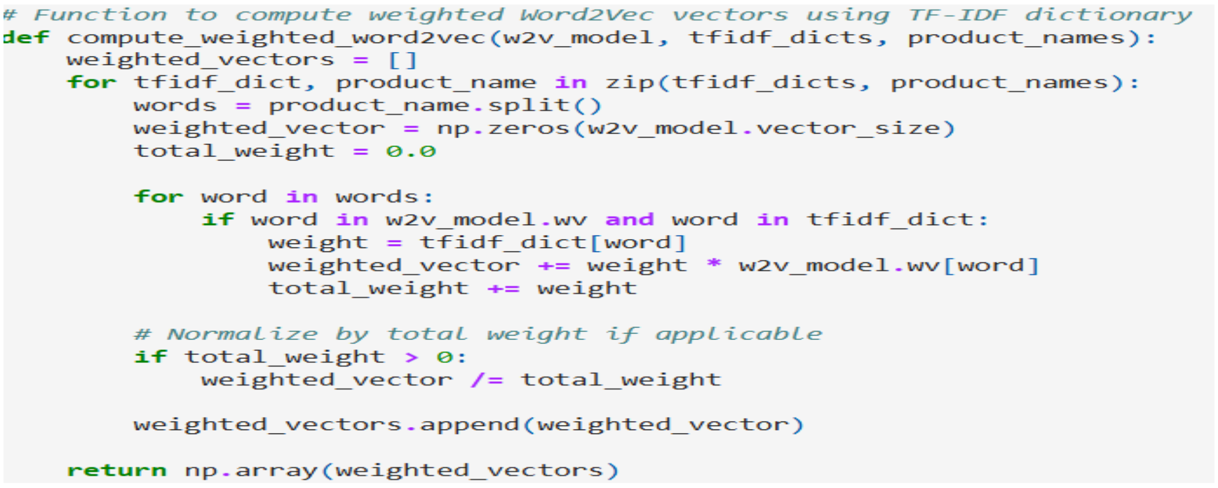
4. **Cosine Similarity Calculation**: The system calculates the cosine similarity between the weighted product vectors to measure the closeness between products.

5. **Product Recommendations**: Based on the cosine similarity, the system identifies the most similar products and displays relevant details, such as product names, URLs, and images.

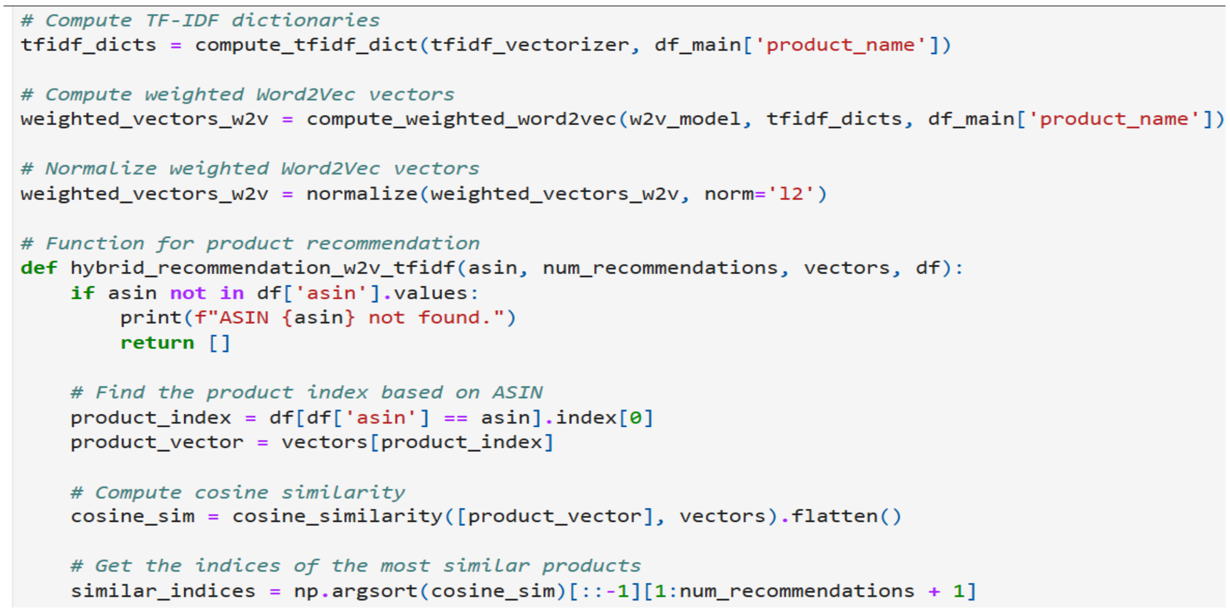
6. **Error Handling**: The system includes robust error management to handle cases where no valid word vectors are found or other issues arise during the recommendation process.

This hybrid model significantly enhances recommendation accuracy by combining the strengths of both Word2Vec and TF-IDF, enabling the system to recommend products that are both semantically similar and contextually relevant based on their descriptions.

Code:

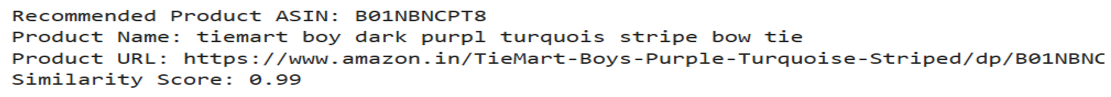


```python
# Function to compute weighted Word2Vec vectors using TF-IDF dictionary
def compute_weighted_word2vec(w2v_model, tfidf_dicts, product_names):
    weighted_vectors = []
    for tfidf_dict, product_name in zip(tfidf_dicts, product_names):
        words = product_name.split()
        weighted_vector = np.zeros(w2v_model.vector_size)
        total_weight = 0.0

        for word in words:
            if word in w2v_model.wv and word in tfidf_dict:
                weight = tfidf_dict[word]
                weighted_vector += weight * w2v_model.wv[word]
                total_weight += weight

        # Normalize by total weight if applicable
        if total_weight > 0:
            weighted_vector /= total_weight

        weighted_vectors.append(weighted_vector)

    return np.array(weighted_vectors)
```

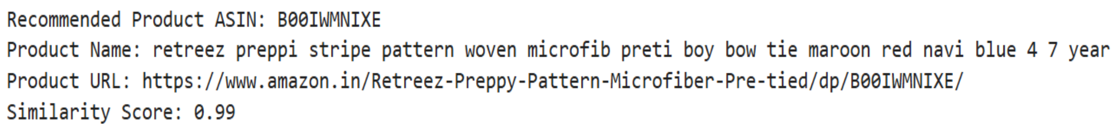


```python
# Compute TF-IDF dictionaries
tfidf_dicts = compute_tfidf_dict(tfidf_vectorizer, df_main['product_name'])

# Compute weighted Word2Vec vectors
weighted_vectors_w2v = compute_weighted_word2vec(w2v_model, tfidf_dicts, df_main['product_name'])

# Normalize weighted Word2Vec vectors
weighted_vectors_w2v = normalize(weighted_vectors_w2v, norm='l2')

# Function for product recommendation
def hybrid_recommendation_w2v_tfidf(asin, num_recommendations, vectors, df):
    if asin not in df['asin'].values:
        print(f"ASIN {asin} not found.")
        return []

    # Find the product index based on ASIN
    product_index = df[df['asin'] == asin].index[0]
    product_vector = vectors[product_index]

    # Compute cosine similarity
    cosine_sim = cosine_similarity([product_vector], vectors).flatten()

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

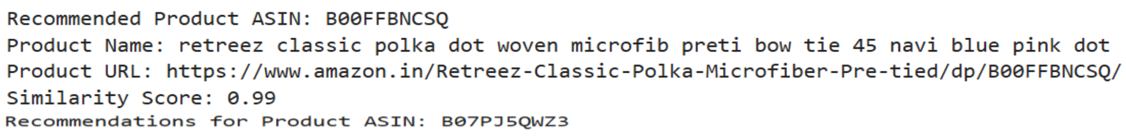
OUTPUT:



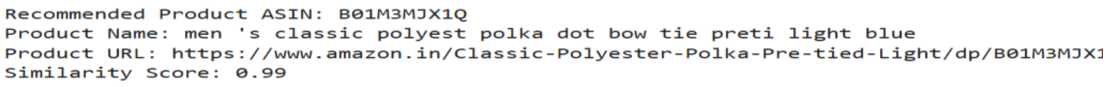
```
Recommended Product ASIN: B01NBNCPT8
Product Name: tiemart boy dark purpl turquois stripe bow tie
Product URL: https://www.amazon.in/TieMart-Boys-Purple-Turquoise-Striped/dp/B01NBNC
Similarity Score: 0.99
```



```
Recommended Product ASIN: B00IWMNIXE
Product Name: retreez preppi stripe pattern woven microfib preti boy bow tie maroon red navi blue 4 7 year
Product URL: https://www.amazon.in/Retreez-Preppy-Pattern-Microfiber-Pre-tied/dp/B00IWMNIXE/
Similarity Score: 0.99
```



```
Recommended Product ASIN: B00FFBNCSQ
Product Name: retreez classic polka dot woven microfib preti bow tie 45 navi blue pink dot
Product URL: https://www.amazon.in/Retreez-Classic-Polka-Microfiber-Pre-tied/dp/B00FFBNCSQ/
Similarity Score: 0.99
Recommendations for Product ASIN: B07PJ5QWZ3
```



```
Recommended Product ASIN: B01M3MJX1Q
Product Name: men 's classic polyest polka dot bow tie preti light blue
Product URL: https://www.amazon.in/Classic-Polyester-Polka-Pre-tied-Light/dp/B01M3MJX1
Similarity Score: 0.99
```

Brand based Product Recommendation Using Word2Vec

Overview:

In this product recommendation system, we integrate Word2Vec with brand prioritization to recommend products that are semantically similar and from the same brand. Word2Vec generates dense vector representations of product names, capturing the contextual meaning of words. To refine the recommendations, we introduce a brand-weighting mechanism that prioritizes products from the same brand as the queried product, ensuring brand consistency in the suggestions. The model computes the cosine similarity between the product's Word2Vec vector and the vectors of other products in the same cluster, adjusting the similarity scores by the brand weight.

This approach enhances traditional Word2Vec recommendations by adding a layer of brand prioritization, leading to more personalized and relevant suggestions for users who are looking for similar products from a particular brand.

Key Functionality

1. **Word Vector Creation:** Word2Vec trains on product names to create semantic vectors.
2. **Cluster Labeling:** Products are grouped into clusters for contextual similarity.
3. **Brand Prioritization:** Same-brand products are prioritized within the same cluster.
4. **Weighted Product Vector Computation:** TF-IDF-weighted Word2Vec vectors are calculated for each product.
5. **Cosine Similarity Calculation:** Similarity between the query and other product vectors is computed.

6. **Brand-Weighted Similarity:** Scores are adjusted by multiplying with a brand weight, prioritizing same-brand products.
7. **Recommendations:** Top-ranked products, with names, brands, similarity scores, and images, are presented.
8. **Error Handling:** The system ensures robustness in cases of missing vectors or unmatched brands.

This model provides more relevant product recommendations by combining semantic similarities with brand preferences, making it suitable for users who are looking for similar products within specific brands or categories**.**

Code:

```python
# Function to train and save the Word2Vec model
def train_word2vec_model(texts, vector_size=100, window=5, min_count=1):
    tokenized_texts = [text.split() for text in texts]
    model = gensim.models.Word2Vec(tokenized_texts, vector_size=vector_size, window=window, min_count=min_count)
    model.save("word2vec.model")
    return model
```

```python
# Function to recommend products
def recommend_products_with_brand_weight_w2v(asin, num_recommendations, df, cluster_labels, brand_weight=1.5, w2v_model=None):
    if asin not in df['asin'].values:
        print(f"ASIN {asin} not found.")
        return []

    product_index = df[df['asin'] == asin].index[0]
    selected_brand = df.iloc[product_index]['brand']
    selected_cluster = cluster_labels[product_index]
    query_name = df.iloc[product_index]['product_name']

    # Get Word2Vec vector for the query product
    query_vector = np.mean([w2v_model.wv[word] for word in query_name.split() if word in w2v_model.wv], axis=0)

    if query_vector is None or np.isnan(query_vector).any():
        print("Query vector could not be computed.")
        return []

    # Filter products in the same cluster
    same_cluster_indices = np.where(cluster_labels == selected_cluster)[0]
    same_cluster = df.iloc[same_cluster_indices].copy()

    # Filter products by brand
    same_brand_products = same_cluster[same_cluster['brand'] == selected_brand].copy()
```

Output:

```
Recommendations for ASIN: B01D6VFN70 (Brand: Peppermint)
```

```
Product Name: peppermint synthet dress
Brand: Peppermint
Similarity Score: 1.50
```

```
Product Name: peppermint synthet dress
Brand: Peppermint
Similarity Score: 1.50
```

```
Product Name: peppermint girl alin maxi dress
Brand: Peppermint
Similarity Score: 1.44
```

```
Product Name: peppermint synthet dress
Brand: Peppermint
Similarity Score: 1.50
```

Image based product recommendation

VGG16 model Overview:

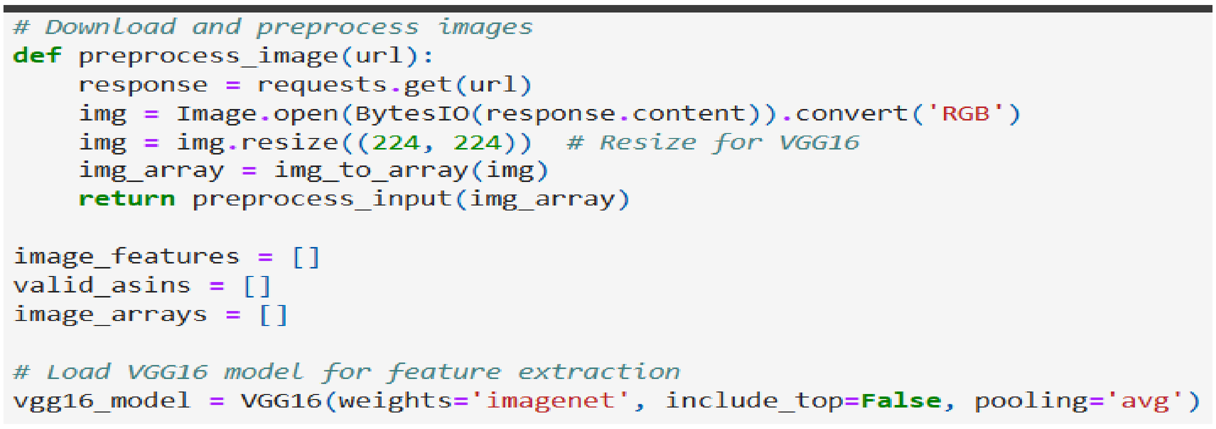
The recommendation system utilizes a deep learning model based on VGG16, a well-established pre-trained Convolutional Neural Network (CNN), for feature extraction from product images. The VGG16 model, initially trained on large datasets like ImageNet, is capable of recognizing complex visual patterns and structures. In this system, we employ the VGG16 model in its convolutional layers, which are designed to extract rich visual features from images

Key Functionality:

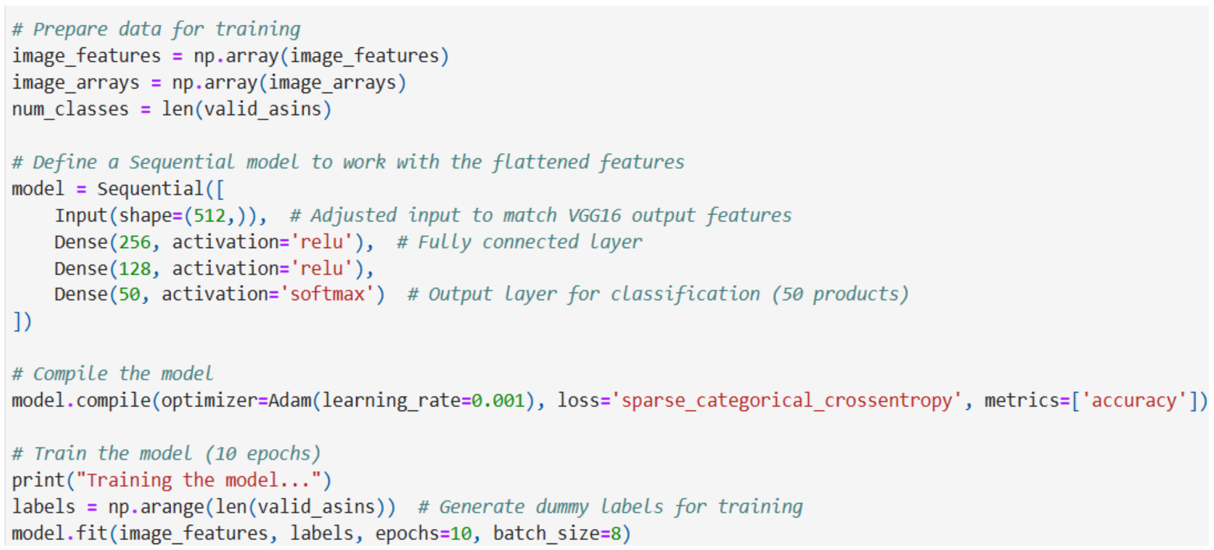
1. **Image Preprocessing**: Product images are resized and preprocessed to be input into the VGG16 model.
2. **Feature Extraction**: VGG16 extracts features from product images, providing a dense representation of the visual content.
3. **Similarity Calculation**: Cosine similarity between the queried product's features and others is computed to identify visually similar products.
4. **Recommendation Generation**: Based on similarity scores, top-ranked products are recommended.
5. **Display Results**: Recommendations are shown in an HTML format, featuring images, product names, and similarity scores.

Code:

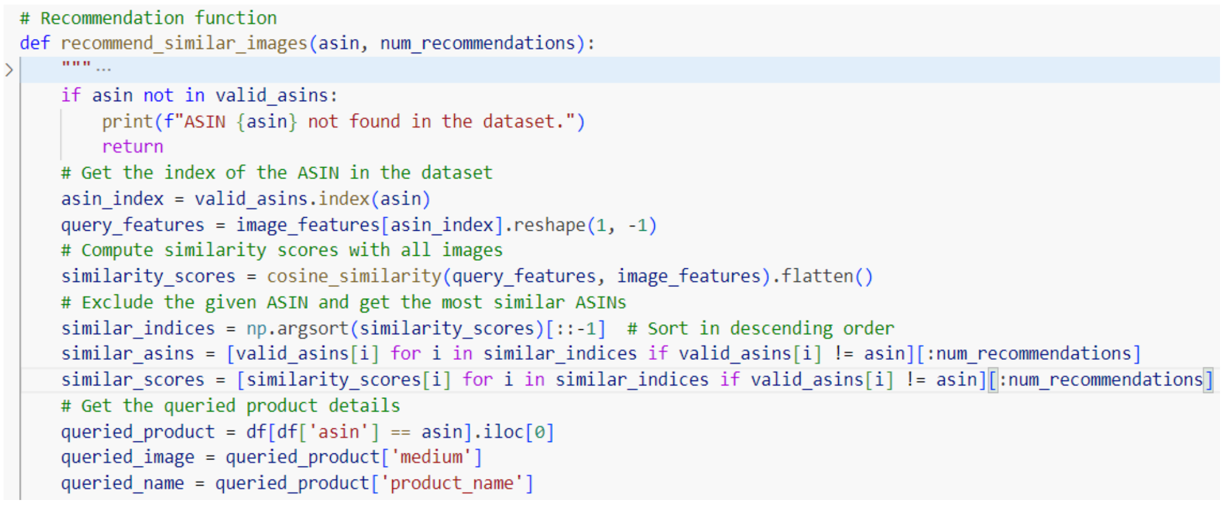
```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.optimizers import Adam
from sklearn.metrics.pairwise import cosine_similarity
from IPython.display import display, HTML
import requests
from io import BytesIO
```



```python
# Download and preprocess images
def preprocess_image(url):
    response = requests.get(url)
    img = Image.open(BytesIO(response.content)).convert('RGB')
    img = img.resize((224, 224))  # Resize for VGG16
    img_array = img_to_array(img)
    return preprocess_input(img_array)

image_features = []
valid_asins = []
image_arrays = []

# Load VGG16 model for feature extraction
vgg16_model = VGG16(weights='imagenet', include_top=False, pooling='avg')
```



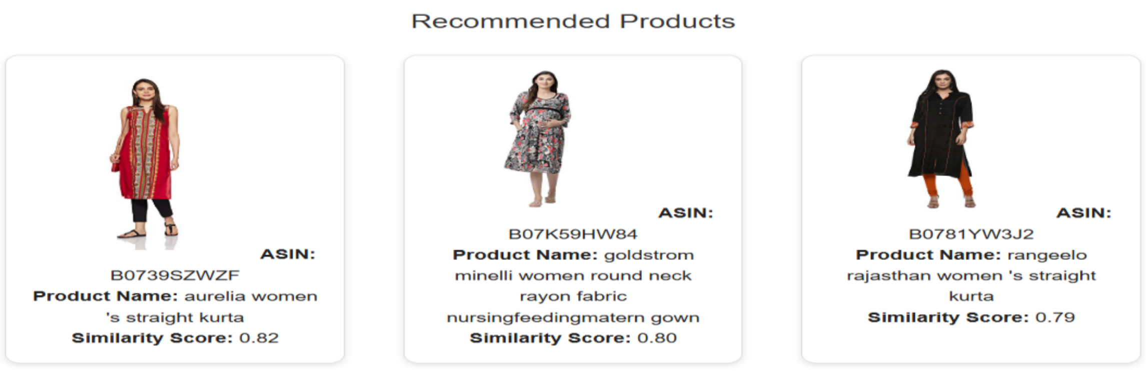
```python
# Prepare data for training
image_features = np.array(image_features)
image_arrays = np.array(image_arrays)
num_classes = len(valid_asins)

# Define a Sequential model to work with the flattened features
model = Sequential([
    Input(shape=(512,)),  # Adjusted input to match VGG16 output features
    Dense(256, activation='relu'),  # Fully connected layer
    Dense(128, activation='relu'),
    Dense(50, activation='softmax')  # Output layer for classification (50 products)
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model (10 epochs)
print("Training the model...")
labels = np.arange(len(valid_asins))  # Generate dummy labels for training
model.fit(image_features, labels, epochs=10, batch_size=8)
```



```python
# Recommendation function
def recommend_similar_images(asin, num_recommendations):
    """...
    if asin not in valid_asins:
        print(f"ASIN {asin} not found in the dataset.")
        return
    # Get the index of the ASIN in the dataset
    asin_index = valid_asins.index(asin)
    query_features = image_features[asin_index].reshape(1, -1)
    # Compute similarity scores with all images
    similarity_scores = cosine_similarity(query_features, image_features).flatten()
    # Exclude the given ASIN and get the most similar ASINs
    similar_indices = np.argsort(similarity_scores)[::-1]  # Sort in descending order
    similar_asins = [valid_asins[i] for i in similar_indices if valid_asins[i] != asin][:num_recommendations]
    similar_scores = [similarity_scores[i] for i in similar_indices if valid_asins[i] != asin][:num_recommendations]
    # Get the queried product details
    queried_product = df[df['asin'] == asin].iloc[0]
    queried_image = queried_product['medium']
    queried_name = queried_product['product_name']
```

Output:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 272ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
Training the model...
Epoch 1/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.0000e+00 - loss: 7.5256
Epoch 2/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2912 - loss: 3.4210
Epoch 3/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7296 - loss: 1.6608
Epoch 4/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7484 - loss: 1.0881
Epoch 5/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9269 - loss: 0.5692
```



Recommended Products

ASIN: B0739SZWZF
Product Name: aurelia women 's straight kurta
Similarity Score: 0.82

ASIN: B07K59HW84
Product Name: goldstrom minelli women round neck rayon fabric nursingfeedingmatern gown
Similarity Score: 0.80

ASIN: B0781YW3J2
Product Name: rangeelo rajasthan women 's straight kurta
Similarity Score: 0.79

Product and Brand-based Recommendation System

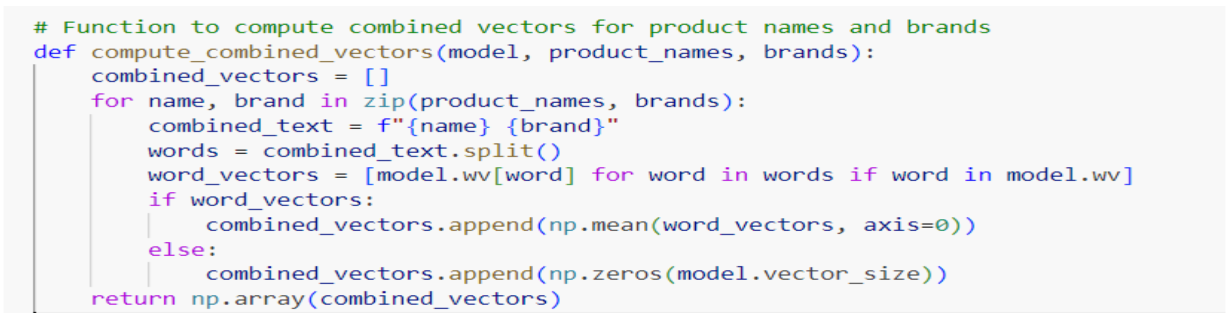
Overview:

The recommendation system utilizes a Word2Vec-based approach to suggest products based on both product names and brands. Word2Vec is a popular technique in Natural Language Processing (NLP) that converts words into dense vector representations. In this system, Word2Vec is used to compute vectors for product names and brands, and these vectors are then combined to calculate the similarity between products. This system leverages the word embeddings to recommend similar products based on their textual descriptions.

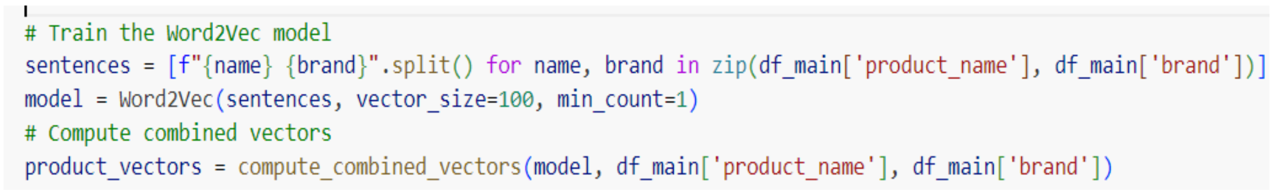
Key Functionality:

1. **Text Preprocessing**: The product names and brands are concatenated and tokenized into words for Word2Vec model input.
2. **Feature Extraction**: Word2Vec generates vector representations of words, which are then averaged to form a single vector for each product based on its name and brand.
3. **Similarity Calculation**: Cosine distance is computed between the vector of the queried product and others in the dataset. The smaller the cosine distance, the more similar the products are.
4. **Recommendation Generation**: Based on similarity scores, the top-ranked products are recommended, considering both textual similarity and user ratings.
5. **Display Results**: Recommendations are displayed with product details

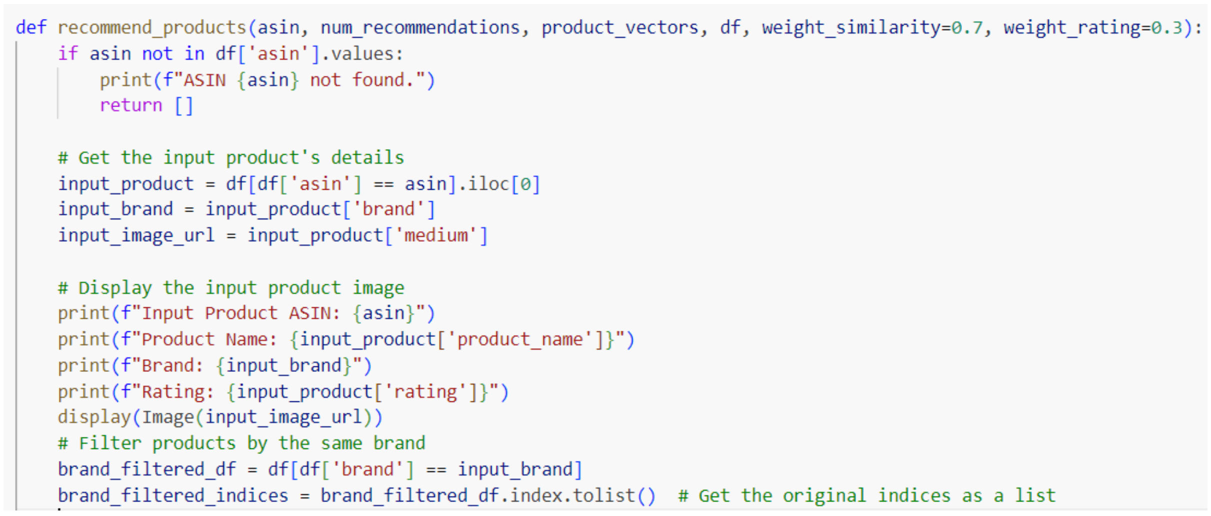
Code:



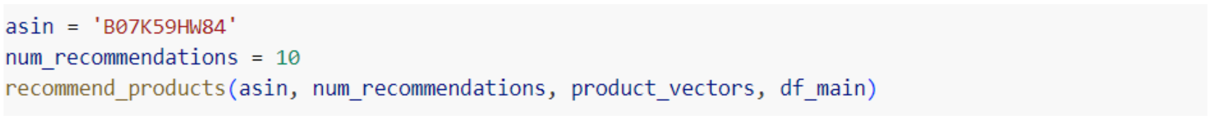
```python
# Function to compute combined vectors for product names and brands
def compute_combined_vectors(model, product_names, brands):
    combined_vectors = []
    for name, brand in zip(product_names, brands):
        combined_text = f"{name} {brand}"
        words = combined_text.split()
        word_vectors = [model.wv[word] for word in words if word in model.wv]
        if word_vectors:
            combined_vectors.append(np.mean(word_vectors, axis=0))
        else:
            combined_vectors.append(np.zeros(model.vector_size))
    return np.array(combined_vectors)
```



```python
# Train the Word2Vec model
sentences = [f"{name} {brand}".split() for name, brand in zip(df_main['product_name'], df_main['brand'])]
model = Word2Vec(sentences, vector_size=100, min_count=1)
# Compute combined vectors
product_vectors = compute_combined_vectors(model, df_main['product_name'], df_main['brand'])
```



```python
def recommend_products(asin, num_recommendations, product_vectors, df, weight_similarity=0.7, weight_rating=0.3):
    if asin not in df['asin'].values:
        print(f"ASIN {asin} not found.")
        return []

    # Get the input product's details
    input_product = df[df['asin'] == asin].iloc[0]
    input_brand = input_product['brand']
    input_image_url = input_product['medium']

    # Display the input product image
    print(f"Input Product ASIN: {asin}")
    print(f"Product Name: {input_product['product_name']}")
    print(f"Brand: {input_brand}")
    print(f"Rating: {input_product['rating']}")
    display(Image(input_image_url))
    # Filter products by the same brand
    brand_filtered_df = df[df['brand'] == input_brand]
    brand_filtered_indices = brand_filtered_df.index.tolist()  # Get the original indices as a list
```



```python
asin = 'B07K59HW84'
num_recommendations = 10
recommend_products(asin, num_recommendations, product_vectors, df_main)
```

Output:

```
Recommendations for Product ASIN: B075T5GYKN

Recommended Product ASIN: B07ML6872L
Product Name: texco stripe maxi dress women
Brand: TEXCO
Rating: 5.0
Product URL: https://www.amazon.in/TEXCO-Black-W
hite-Striped-Dress/dp/B07MKNVBWF/
Combined Score: 0.98
```

```
Recommended Product ASIN: B07NWLHX5L
Product Name: texco revers women shirt dress
Brand: TEXCO
Rating: 5.0
Product URL: https://www.amazon.in/TEXCO-Burgund
y-Reversible-Women-Shirt/dp/B07NWMVV8Y/
Combined Score: 0.97
```

```
Recommended Product ASIN: B01II08Y38
Product Name: texco women 's cotton dress belt a
ccessori blue white xs
Brand: TEXCO
Rating: 5.0
Product URL: https://www.amazon.in/TEXCO-Womens-
Cotton-Dress-White/dp/B01II08Y38/
Combined Score: 0.94
```

```
Recommended Product ASIN: B075T5GYKN
Product Name: texco pink side draw string crop t
op
Brand: TEXCO
Rating: 3.0
Product URL: https://www.amazon.in/TEXCO-Mustard
-Yellow-Strings-Women/dp/B075T4JV75/
Combined Score: 0.88
```

```
Recommended Product ASIN: B01N0L6YIY
Product Name: texco cotton polyst fleec winter h
ood multicolor jacket
Brand: TEXCO
Rating: 4.1
Product URL: https://www.amazon.in/TEXCO-Cotton-
Polyster-Fleece-Multicolor/dp/B017IHWO1U/
Combined Score: 0.86
```

```
Recommended Product ASIN: B01JCQXCV6
Product Name: texco digon stripe neck detail max
i dress
Brand: TEXCO
Rating: 5.0
Product URL: https://www.amazon.in/TEXCO-Fashion
-Digonal-Striped-Detailing/dp/B01JCQXUPE/
Combined Score: 0.84
```

```
Recommended Product ASIN: B01N63J8XG
Product Name: texco red stud embelish turtl neck
full sleev women sweatshirt
Brand: TEXCO
Rating: 4.6
Product URL: https://www.amazon.in/TEXCO-Embelis
hed-Turtle-Sleeve-Sweatshirt/dp/B01N74VI8N/
Combined Score: 0.83
```

```
Recommended Product ASIN: B012FIDJCK
Product Name: texco poli creap long dress women
's
Brand: TEXCO
Rating: 5.0
Product URL: https://www.amazon.in/TEXCO-Poly-Cr
eap-Women-Dress/dp/B012FIDJCK/
Combined Score: 0.77
```

Comparison: TF-IDF vs Bag of Words vs Word2Vec

Aspect	Bag of Words (BoW)	TF-IDF (Term Frequency - Inverse Document Frequency)	Word2Vec
Definition	A text representation method where each word in the corpus is represented as a unique token in a fixed-length vector.	TF-IDF is a numerical statistic used to reflect the importance of a word in a document relative to its frequency in the	A deep learning-based model that learns word representations by mapping words to high-dimensional vectors based on context.
Formula	BoW Representation: Vector = [count(word1), count(word2), ..., count(wordN)]	TF = (Frequency of word in document) / (Total words in document) IDF = log(Total number of documents / Number of documents containing the word) TF-IDF = TF * IDF	Word2Vec uses either CBOW (Continuous Bag of Words) or Skip-gram method to predict surrounding words or context. Formula: words or context. `log(P(w_t
Dimensionality	Fixed and equal to the size of the vocabulary (length of vocabulary).	Dynamic depending on the size of the document corpus.	Variable, typically much smaller due to dense vector representation (e.g., 100–300 dimensions).
Representation Type	Sparse Vector	Sparse Vector (though often optimized for sparsity)	Dense Vector (low-dimensional, typically 100-300 dimensions)
Example	Text: "The cat sat on the mat" BoW Vector: [1, 1, 1, 1, 1] (words: "The", "cat", "sat", "on", "mat")	Text: "The cat sat on the mat" TF-IDF Vector: [0.2, 0.2, 0.4,0.4, 0.5] (weights for each word based on term frequency and inverse document frequency)	Text: "The cat sat on the mat" Word2Vec Vector: [0.12, 0.58, -0.32, 0.45, ...] (contextual word representation for each word in the sentence
Contextualization	No context is captured; all words are treated equally and independently.	Still no true context beyond frequency; however, important words are emphasized.	Captures context of words based on surrounding words (semantic relationships).

Use in AI Stylist Project	In an AI stylist project, BoW can be used for basic text classification tasks (e.g., categorizing product descriptions or user feedback).	TF-IDF can be useful for identifying important keywords in text data like product name, meta data , and user ratings.	Word2Vec is ideal for capturing semantic relationships, such as suggesting similar fashion items or identifying user preferences.
Advantages	- Simple and easy to implement. - Straightforward for text classification.	- Considers word importance in documents. - Reduces the impact of common but unimportant words.	-Captures semantic meaning. - Allows for finding similar items based on word context (e.g., similar products or styles).
Disadvantages	- Does not capture word meaning or context. - Produces large, sparse vectors.	- Still ignores semantic meaning. - Complex and slower for large corpora.	- Requires more data and computational power. - Harder to implement and fine-tune.

Collaborative Filtering

Collaborative filtering is based on the idea that similar people (based on the data) generally tend to like similar things. It predicts which item a user will like based on the item preferences of other similar users.

Collaborative filtering uses a user-item matrix to generate recommendations. This matrix contains the values that indicate a user's preference towards a given item. These values can represent either explicit feedback (direct user ratings) or implicit feedback (indirect user behaviour such as listening, purchasing, watching).

- **Explicit Feedback:** The amount of data that is collected from the users when they choose to do so. Many of the times, users choose not to provide data for the user. So, this data is scarce and sometimes costs money. For example, ratings from the user.
- **Implicit Feedback:** In implicit feedback, we track user behaviour to predict their preference.

Models Used in Our AI Stylist Project

In the AI Stylist project, collaborative filtering was implemented using state-of-the-art deep learning models to analyze both textual and visual attributes of the dataset. The dataset includes features such as id, gender, masterCategory, subCategory, articleType, baseColour, season, year, usage, productDisplayName, and imagePath. The models employed include VGG16, ResNet50, and DenseNet for image-based analysis.

- **VGG16**: Extracts visual features from product images, enabling identification of visually similar items.
- **ResNet50:** Utilizes residual connections to capture deeper and more intricate patterns in images, ensuring high accuracy in visual similarity detection.
- **DenseNet**: Connects each layer to every other layer, improving feature propagation and reusability, making it highly effective for extracting detailed visual information.

By combining these models, the collaborative filtering approach in the AI Stylist project provides personalized recommendations by identifying patterns in user behaviour and leveraging both item attributes and visual content. This ensures an engaging and user-centric experience tailored to individual preferences.

Overview of VGG16 Embedding-Based Model

The **VGG16 model** is a pre-trained convolutional neural network designed for image feature extraction. It processes images through multiple convolutional and pooling layers to generate dense, high-dimensional vector embeddings that capture essential visual features like textures, edges, and shapes.

By removing its top classification layers, VGG16 is used as a feature extractor in tasks like image similarity and recommendation systems. For an input image, its embedding is compared to those of other images using **cosine similarity** to identify visually similar items.

What the Model Does

1. **Feature Extraction**:
   The VGG16 model processes each image to create a dense vector (embedding) that represents the image's visual features.

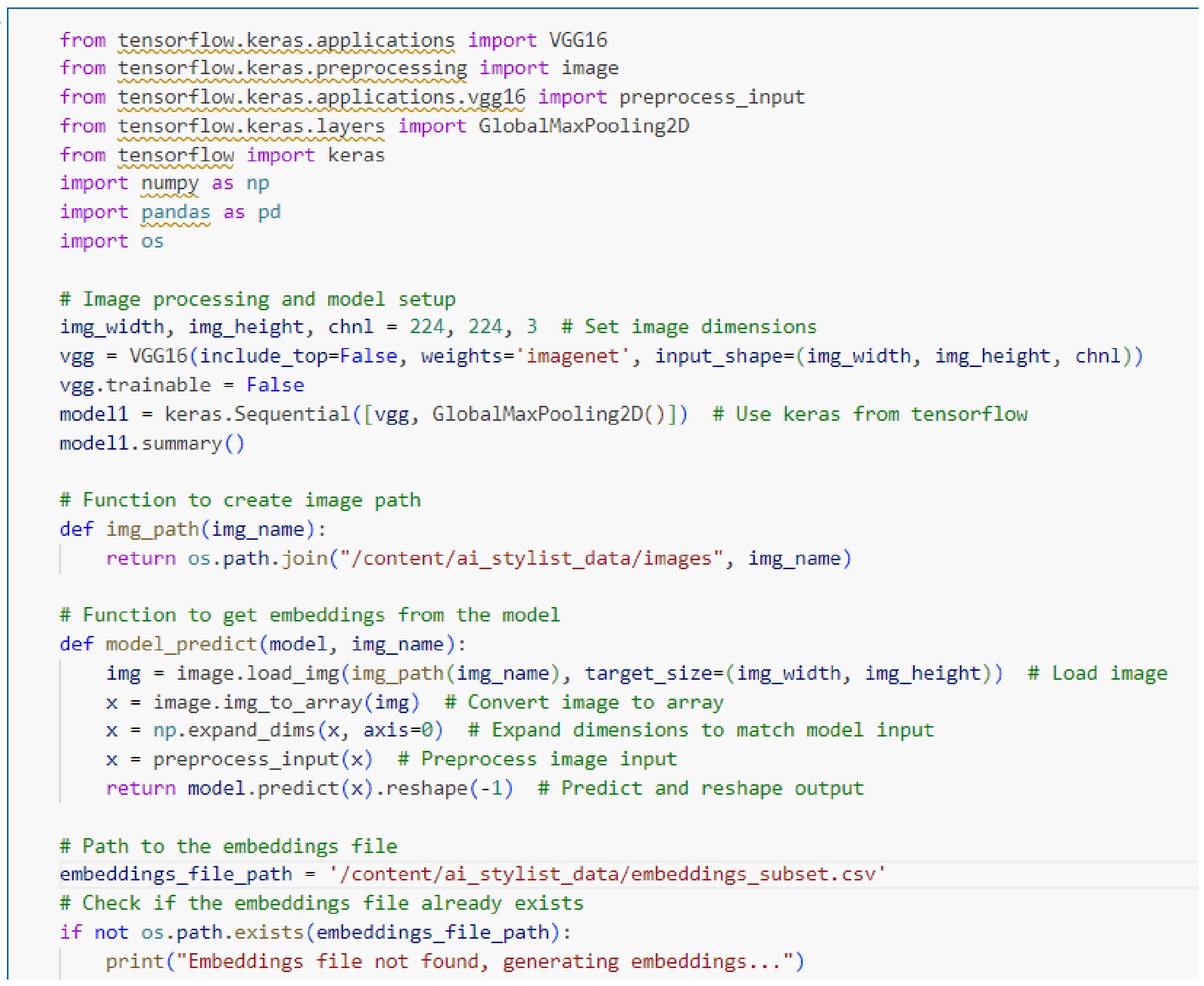
2. **Similarity Calculation:**
   The embeddings are compared using cosine similarity, allowing the system to measure how visually similar two images are.

Example

- Input: An image of a red dress.
- Processing: The image is resized to 224x224, preprocessed, and passed through the VGG16 model to generate its embedding.
- Output: The system compares this embedding to embeddings of other images (e.g., blue dresses, shoes) and identifies the most similar items (e.g., another red dress or similar clothing).

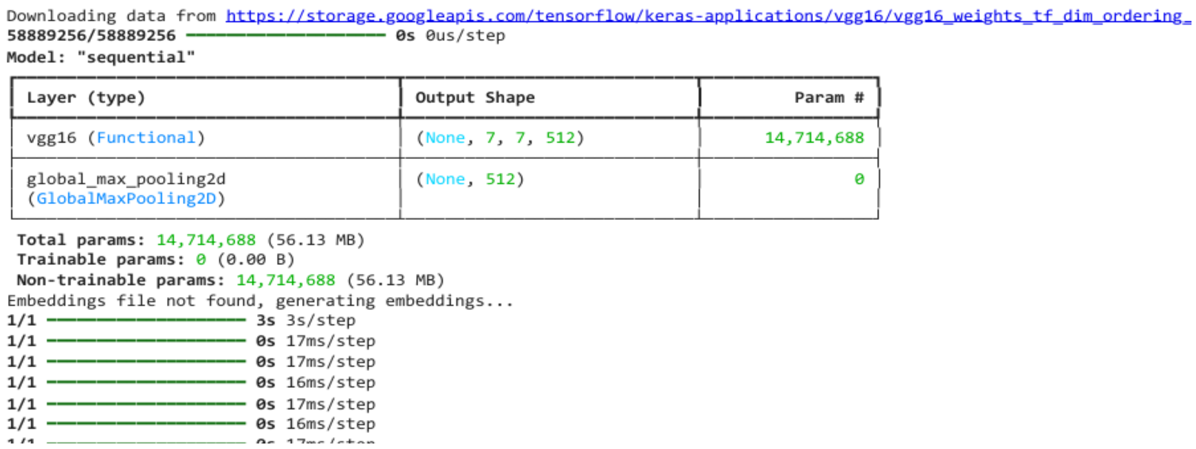
This approach is widely used in applications like fashion recommendation systems and visual search engines.

Code:

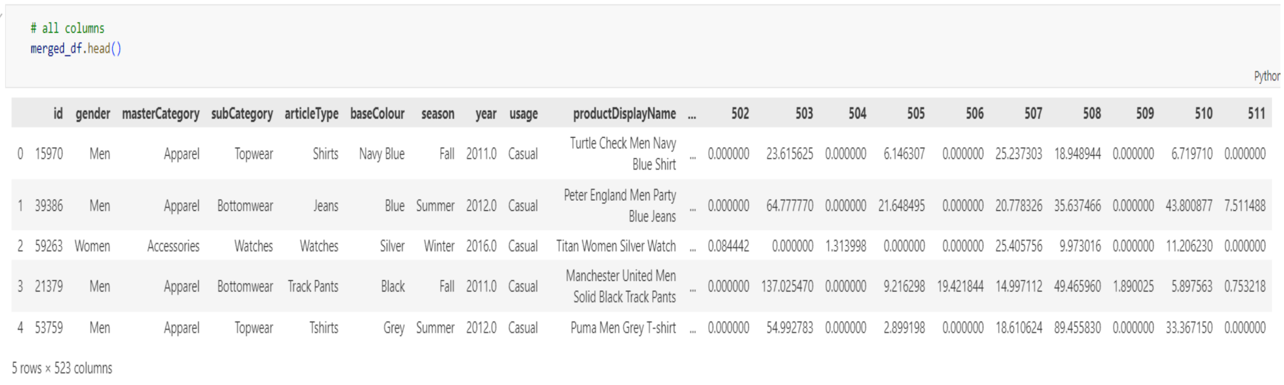


```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.layers import GlobalMaxPooling2D
from tensorflow import keras
import numpy as np
import pandas as pd
import os

# Image processing and model setup
img_width, img_height, chnl = 224, 224, 3  # Set image dimensions
vgg = VGG16(include_top=False, weights='imagenet', input_shape=(img_width, img_height, chnl))
vgg.trainable = False
model1 = keras.Sequential([vgg, GlobalMaxPooling2D()])  # Use keras from tensorflow
model1.summary()

# Function to create image path
def img_path(img_name):
    return os.path.join("/content/ai_stylist_data/images", img_name)

# Function to get embeddings from the model
def model_predict(model, img_name):
    img = image.load_img(img_path(img_name), target_size=(img_width, img_height))  # Load image
    x = image.img_to_array(img)  # Convert image to array
    x = np.expand_dims(x, axis=0)  # Expand dimensions to match model input
    x = preprocess_input(x)  # Preprocess image input
    return model.predict(x).reshape(-1)  # Predict and reshape output

# Path to the embeddings file
embeddings_file_path = '/content/ai_stylist_data/embeddings_subset.csv'
# Check if the embeddings file already exists
if not os.path.exists(embeddings_file_path):
    print("Embeddings file not found, generating embeddings...")
    # Generate embeddings for the subset of images
    df_embedding_subset = df_copy_subset['imagePath'].apply(lambda x: model_predict(model1, x))  # Apply model to each image
    # Convert the embeddings into a DataFrame
    df_embedding_subset = df_embedding_subset.apply(pd.Series)
    # Add embeddings to the copied subset DataFrame
    df_copy_subset = pd.concat([df_copy_subset, df_embedding_subset], axis=1)
   # Save the resulting DataFrame with embeddings for the first 2,000 images
    df_copy_subset.to_csv(embeddings_file_path, index=False)
  # Display the first few rows of the resulting DataFrame
    print(df_copy_subset.head())
else:
    print("Embeddings file already exists, skipping the embedding generation.")
```

Output:



```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃          Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━┩
│ vgg16 (Functional)                   │ (None, 7, 7, 512)           │       14,714,688 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ global_max_pooling2d                 │ (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)
Embeddings file not found, generating embeddings...
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
```



```
# all columns
merged_df.head()
```

Python

	id	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName	...	502	503	504	505	506	507	508	509	510	511
0	15970	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt	...	0.000000	23.615625	0.000000	6.146307	0.000000	25.237303	18.948944	0.000000	6.719710	0.000000
1	39386	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans	...	0.000000	64.777770	0.000000	21.648495	0.000000	20.778326	35.637466	0.000000	43.800877	7.511488
2	59263	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch	...	0.084442	0.000000	1.313998	0.000000	0.000000	25.405756	9.973016	0.000000	11.206230	0.000000
3	21379	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants	...	0.000000	137.025470	0.000000	9.216298	19.421844	14.997112	49.465960	1.890025	5.897563	0.753218
4	53759	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt	...	0.000000	54.992783	0.000000	2.899198	0.000000	18.610624	89.455830	0.000000	33.367150	0.000000

5 rows × 523 columns

Recommendation System Using VGG16 Embeddings

Overview:

The recommendation system leverages the embeddings generated by the VGG16 model to suggest visually similar products for a given input image. By computing the cosine similarity between the input image's embedding and precomputed embeddings of other products, the system ranks and displays the most similar items.

What the Recommendation System Does

1. **Input Image Processing:**
   The input image is resized, preprocessed, and passed through VGG16 to extract visual features as a dense vector embedding.

2. **Similarity Computation:**
   Cosine similarity is used to compare the input embedding with a precomputed dataset of product embeddings. The most similar items are ranked and selected.

3. **Recommendations Display:**
   The input image and the top-N recommended products are displayed with their names and similarity scores for a user-friendly experience**.**

Example

For an input image of a black watch:

- The system extracts its embedding and computes similarity with other products in the dataset.
- Recommendations include:
  - Skagen Men Black Watch (Score: 1.00)
  - Maxima Men Black Watch (Score: 0.82)
  - Maxima Men Black Watch (Score: 0.82)
  - Fastrack Men Navy Blue Dial Watch (Score: 0.81)
  - Esprit Men Black Watch (Score: 0.81)

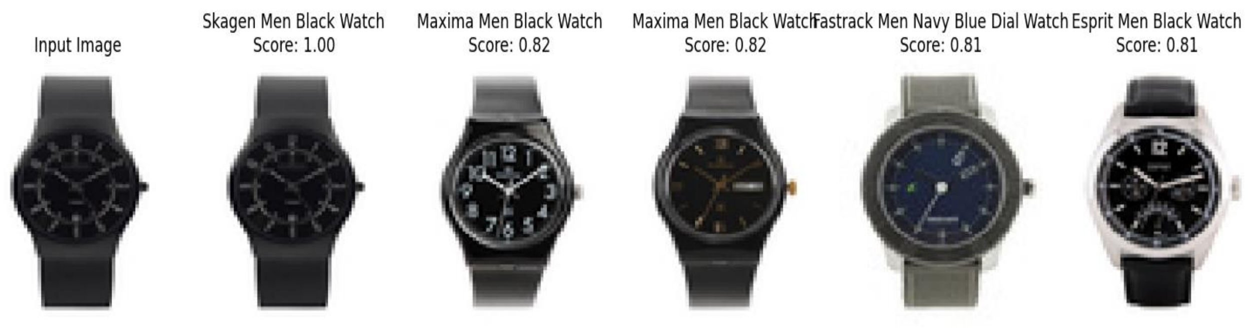
Code:

```python
# Function to recommend products based on embeddings
def recommend_products(input_img_name, top_n=5):
    # Get input image embedding
    input_embedding = model_predict(model1, input_img_name).reshape(1, -1)

    # Compute cosine similarity
    similarities = cosine_similarity(input_embedding, embeddings).flatten()

    # Get top N recommendations (excluding the input image itself if it's in the dataset)
    recommended_indices = np.argsort(similarities)[::-1][:top_n]
    recommended_scores = similarities[recommended_indices]

    # Display recommendations
    input_img_path = img_path(input_img_name)
    display_recommendations(input_img_path, recommended_indices, recommended_scores)

# Example usage
input_image_name = '30039.jpg'  # Replace with your image filename
recommend_products(input_image_name, top_n=5)
```

```python
# Extract embeddings as a numpy array
embeddings = df.iloc[:, -512:].values  # Assuming embeddings are the last 512 columns
# Function to display images with product names and similarity scores
def display_recommendations(input_img_path, recommended_indices, scores):
    fig, axes = plt.subplots(1, len(recommended_indices) + 1, figsize=(15, 5))

    # Display input image
    img = image.load_img(input_img_path, target_size=(224, 224))
    axes[0].imshow(img)
    axes[0].set_title("Input Image")
    axes[0].axis('off')
    # Display recommended images
    for i, idx in enumerate(recommended_indices):
        rec_img_path = img_path(df.iloc[idx]['imagePath'])  # Get image path
        rec_img = image.load_img(rec_img_path, target_size=(224, 224))
        axes[i + 1].imshow(rec_img)
        axes[i + 1].set_title(f"{df.iloc[idx]['productDisplayName']}\nScore: {scores[i]:.2f}")
        axes[i + 1].axis('off')

    plt.tight_layout()
    plt.show()
```

```python
# Example usage
input_image_name = '30039.jpg'  # Replace with your image filename
recommend_products(input_image_name, top_n=5)
```

Output:



ResNet50 Embedding-Based Model

Overview:

The ResNet50 model is a pre-trained convolutional neural network renowned for its deep residual learning framework. It is designed to extract robust and high-dimensional embeddings that capture intricate visual features such as patterns, textures, and shapes.

By utilizing ResNet50 without its top classification layers, it serves as a powerful feature extractor for applications like image similarity and recommendation systems. Each image's embedding is compared with others to identify visually similar items.

What the Model Does

1. **Data Preparation**:
    - Valid image paths are filtered and combined with their full file paths.
    - A subset of 5000 images is selected for processing.
2. **Feature Extraction**:
    - ResNet50, pre-trained on ImageNet, processes each image to create dense vector embeddings representing its unique visual features.
    - Images are resized, preprocessed, and passed through the model to generate embeddings.
3. **Embedding Storage:**
    - Rows with invalid or missing embeddings are removed.
    - The final embeddings are saved to a CSV file for efficient future use.

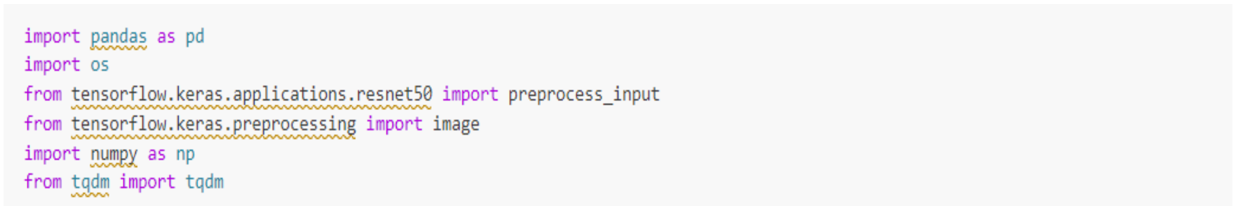
Example Workflow

- **Input:** An image of a red handbag.
- **Processing:** The image is resized, preprocessed, and passed through the ResNet50 model to generate its embedding. Cosine similarity is used to compare it with other embeddings.

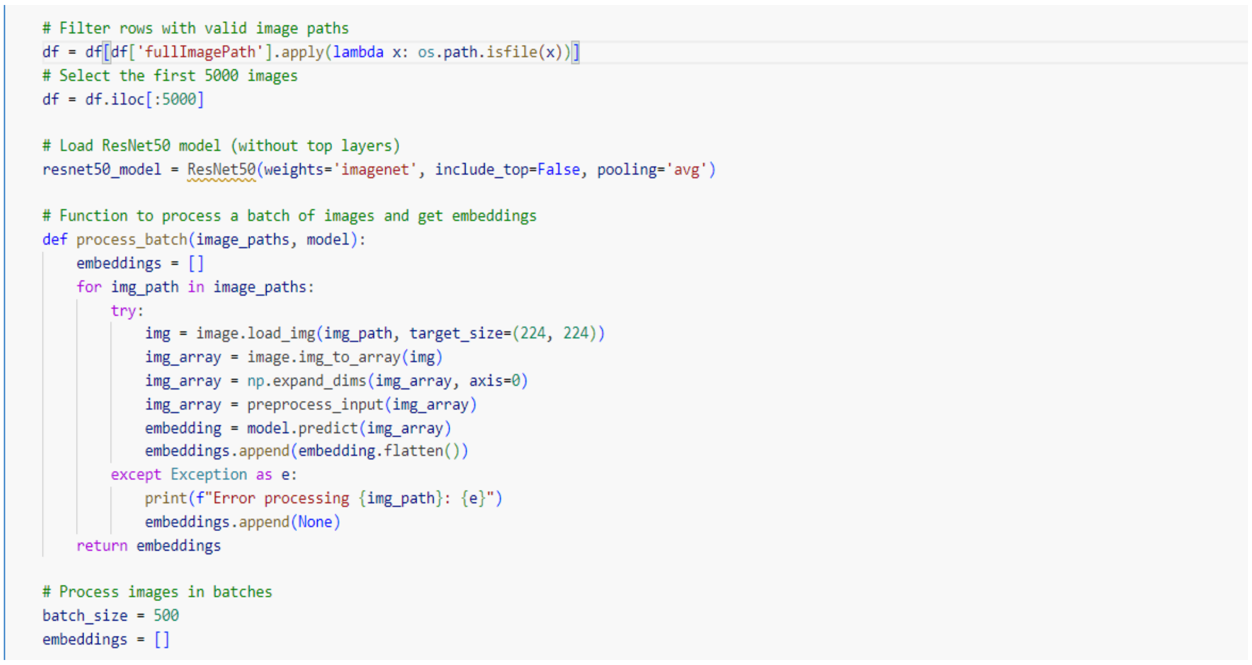
- **Output:** Similar handbags are identified, such as a Red Leather Tote (Score: 1.00) and a Maroon Shoulder Bag (Score: 0.88).

This approach enables scalable and efficient extraction of image features, making it ideal for building recommendation systems and visual search engines.

Code:

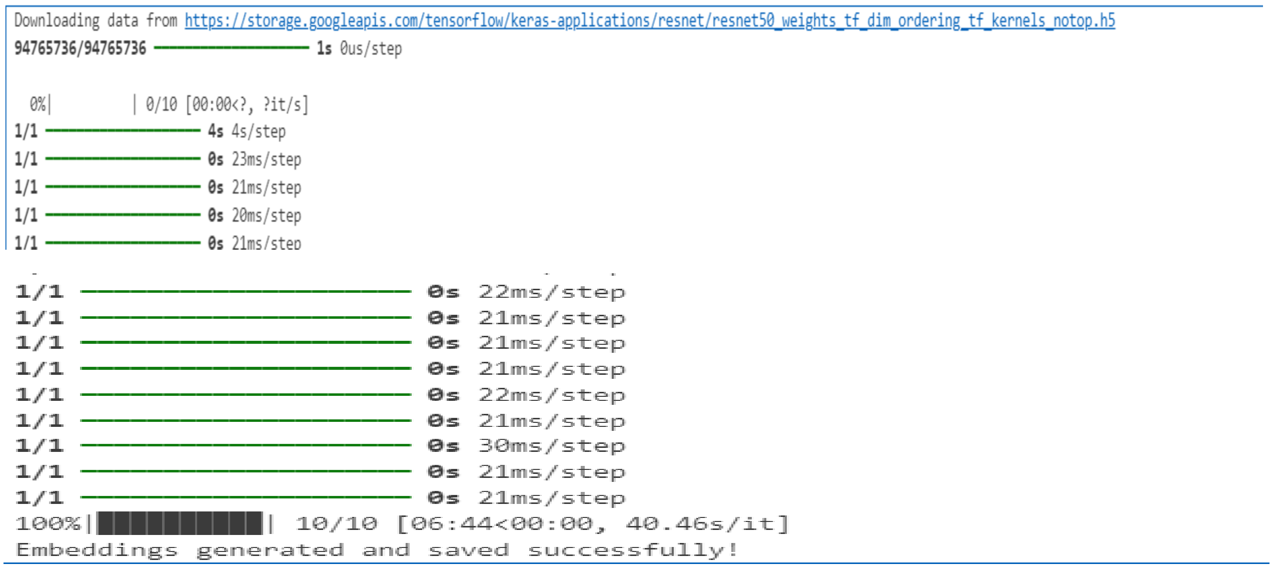


```
import pandas as pd
import os
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
import numpy as np
from tqdm import tqdm
```



```
# Filter rows with valid image paths
df = df[df['fullImagePath'].apply(lambda x: os.path.isfile(x))]
# Select the first 5000 images
df = df.iloc[:5000]

# Load ResNet50 model (without top layers)
resnet50_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')

# Function to process a batch of images and get embeddings
def process_batch(image_paths, model):
    embeddings = []
    for img_path in image_paths:
        try:
            img = image.load_img(img_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0)
            img_array = preprocess_input(img_array)
            embedding = model.predict(img_array)
            embeddings.append(embedding.flatten())
        except Exception as e:
            print(f"Error processing {img_path}: {e}")
            embeddings.append(None)
    return embeddings

# Process images in batches
batch_size = 500
embeddings = []
```

Output:



```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step

  0%|          | 0/10 [00:00<?, ?it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
100%|██████████| 10/10 [06:44<00:00, 40.46s/it]
Embeddings generated and saved successfully!
```

Recommendation System Using ResNet50 Embeddings

Overview:

This recommendation system uses embeddings generated by the ResNet50 model to suggest visually similar products to a given input image. The system calculates the cosine similarity between the input image's embedding and the precomputed embeddings of other products in the dataset. The most similar items are ranked and displayed based on similarity scores.

How the Recommendation System Works:

1. **Input Image Processing:**
    - The input image is resized to a standard size (e.g., 224x224 pixels), preprocessed to match the input requirements of ResNet50, and passed through the model to extract visual features as a dense vector embedding.
2. **Similarity Computation:**
    - Cosine similarity is used to measure the similarity between the input image's embedding and the embeddings of other products in the dataset. This step helps to quantify how similar the products are in terms of visual features.
3. **Recommendations Display:**
    - The system displays the input image along with the top-N most similar products, including their names and similarity scores, for a more user-friendly experience.

Example:

For an input image of a Saree, the recommendation system works as follows:

- The system extracts the embedding of the Saree image and computes the cosine similarity between the query image's embedding and precomputed embeddings of other sarees in the dataset.
- **The top recommendations may include:**
    - Saree 1 **(Score: 1.00)**
    - Saree 2 **(Score: 0.92)**
    - Saree 3 **(Score: 0.91)**
    - Saree 4 **(Score: 0.89)**
    - Saree 5 **(Score: 0.88)**

Code:

```python
# Example recommendation function
def recommend_similar_images(image_id, top_n=5):
    # Find the embedding of the query image
    query_embedding = df[df['id'] == image_id]['embeddings'].values[0]

    # Compute cosine similarity
    all_embeddings = np.stack(df['embeddings'].values)
    similarity_scores = cosine_similarity([query_embedding], all_embeddings).flatten()
    # Get top N similar images
    top_indices = similarity_scores.argsort()[-top_n-1:][::-1][1:]  # Exclude the query image itself
    similar_images = df.iloc[top_indices]

    # Display the images
    plt.figure(figsize=(15, 5))
    for i, (_, row) in enumerate(similar_images.iterrows()):
        img = image.load_img(row['fullImagePath'], target_size=(224, 224))
        plt.subplot(1, top_n, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"Similarity: {similarity_scores[top_indices[i]]:.2f}")
    plt.show()
```

```python
# Test the recommendation function
test_image_id = 57958    # Replace with an actual ID from the styles.csv
recommend_similar_images(test_image_id, top_n=5)
```

Output: