INFOSYS SPRINGBOARD INTERNSHIP REPORT

PROJECT NAME: AI STYLIST

MENTOR NAME: ANIL KUMAR SHAW

TEAM MEMBERS:

1. RAKUL NATRAJ
2. DHARANIDHRAN S
3. DEEKSHA R
4. SAINATH DOBBALI
5. SOORAJ P

INSTITUTION: INFOSYS SPRINGBOARD

DATE:

CONTENTS:

INTRODUCTION:

The AI Stylist is an intelligent fashion recommendation system designed to enhance the shopping experience by offering personalized and relevant product suggestions. By analyzing user preferences, product attributes, and visual features, the AI Stylist helps users discover clothing and accessories that align with their style and needs.

This system aims to revolutionize the fashion industry by providing smart and efficient solutions for product discovery. Whether it’s recommending similar items, identifying trending styles, or matching outfits, the AI Stylist serves as a virtual assistant, enabling users to make informed fashion choices with ease and confidence.

ABSTRACT:

The "AI Stylist" project is an AI-powered fashion recommendation system designed to enhance the shopping experience by providing personalized and relevant fashion suggestions. Leveraging advanced artificial intelligence techniques, the system analyzes various factors such as product names, brands, and image similarities to recommend fashion products that align with user preferences. By integrating state-of-the-art machine learning, natural language processing (NLP), and computer vision methods, the AI Stylist revolutionizes how users discover and engage with fashion items.

The system explores multiple recommendation approaches, including content-based filtering using CountVectorizer and TF-IDF, semantic similarity with Word2Vec embeddings, brand-based recommendations, and a hybrid approach combining TF-IDF and Word2Vec. Additionally, an Artificial Neural Network (ANN) model is employed for image-based recommendations to analyze visual features and suggest products with similar attributes. These methods are applied to a dataset of fashion products to evaluate their effectiveness in providing tailored suggestions. The hybrid recommendation strategy demonstrates the system's capability to address diverse user needs while ensuring relevance and accuracy.

To further enhance its functionality, the project leverages deep learning and pre-trained models such as ResNet50, VGG16, MobileNet, and DenseNet121 for visual feature extraction. These embeddings, representing style and attributes, are used alongside similarity metrics like cosine similarity to recommend visually appealing fashion items. The results highlight the potential of AI in transforming the fashion industry by delivering intelligent, efficient, and personalized shopping tools, offering users an engaging and seamless fashion discovery experience.

METHODOLOGY

The development of the AI Stylist involves a systematic approach that integrates advanced machine learning, natural language processing (NLP), and computer vision techniques to build a robust fashion recommendation system. The methodology consists of the following key steps.

DATA CLEANING:

Handling Missing values:

1. Missing values are handled by taking median for the numeric columns and assigning the 'None' values for the non-numeric columns.

Removing Duplicates:

1. Duplicate entries are identified based on columns like 'product_name', 'product_url', 'asin', 'meta_keywords', and 'medium'.
2. The first occurrence is retained, and the rest are removed.

Outlier detection and removal:

1. Outliers in numerical features like 'sales_price' and 'rating' are detected using the Interquartile Range (IQR) method.
2. These outliers are removed to improve data quality.

Text cleaning:

1. Special characters and non-English characters are removed from product names.
2. Text normalization techniques such as lemmatization and stemming are applied

DATA VISULAIZATION:

1. **Missing Values Heatmap:** Visualize the distribution of missing values in the dataset using heatmaps.
2. **Scatter Plots:** Explore the relationship between 'sales_price' and 'rating' through scatter plots.
3. **Histograms:** Examine the distribution of 'sales_price' with histograms.
4. **Pie Charts:** Display the distribution of 'delivery_type' (or relevant categories).
5. **Bar Charts:** Showcase the top 10 brands by count.
6. **Box Plots:** Visualize the distribution of 'rating' and identify potential outlier

FEATURE ENGINEERING:

Text cleaning and Normalization:

1. All text is converted to lowercase to avoid inconsistencies caused by different letter cases (e.g., "Shirt" and "shirt" are treated the same).
2. Common words like "and," "the," and "is" are removed as they do not contribute significant meaning to the context.

Word Embeddings:

1. Generate word embeddings for product names using Word2Vec to capture semantic relationships.

Image Embeddings:

2. Use pre-trained models such as ResNet50, VGG16, MobileNet, and DenseNet121 to extract image embeddings.

MODEL TRAINING:

Content-Based Filtering:

3. **Bagofwords and TF-IDF:** Build content-based recommendation models using Bagofwords and TF-IDF to compute product name similarity.
4. **Word2Vec Similarity:** Use Word2Vec embeddings to calculate cosine similarity between products based on semantic meaning.
5. **Brand-Based Recommendations:** Generate recommendations based on the brand of a given product and similarity.
6. **Hybrid Approach:** Combine TF-IDF and Word2Vec similarities to create a hybrid model for enhanced recommendations.
7. **Pre-Trained model:** Train an ANN model on image features extracted using MobileNetV2 to provide image-based recommendations.

Collaborative Filtering:

1. **Cosine Similarity:** Cosine similarity is used to measure the similarity between two image feature vectors. Cosine similarity calculates the cosine of the angle between two vectors.
2. **VGG16:** A pre-trained VGG16 model, trained on a large dataset is utilized for feature extraction. This helps to create a recommendation system.
3. **MobileNet:** A pre-trained MobileNet model is used for feature extraction, leveraging the knowledge learned from a large dataset to improve the efficiency and performance of the similarity search.
4. **DenseNet:** A pre-trained DenseNet model is used to extract image features, capturing complex patterns and representations for similarity comparison.

MODEL EVALUTION:

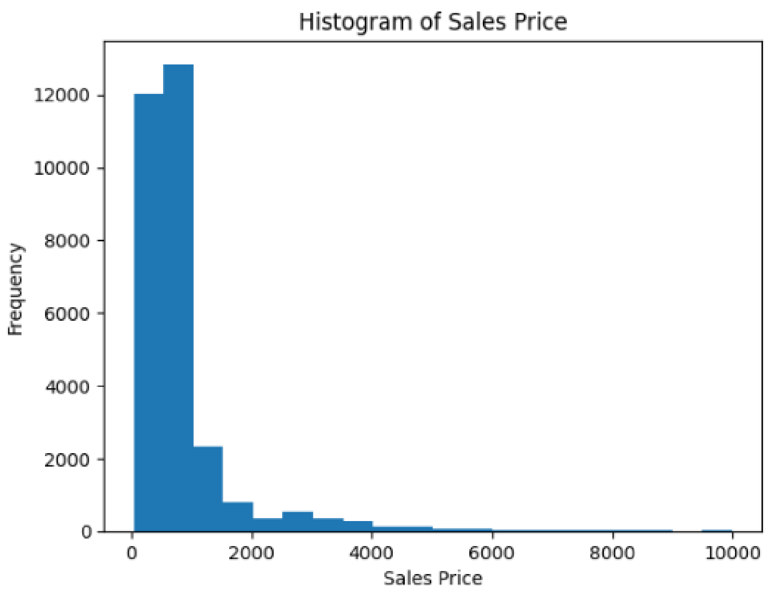
1. The AI Stylist recommendation model is evaluated based on the **similarity scores** between the recommended products and the input product. This ensures that the recommendations are relevant and aligned with user preferences.
2. The similarity score is computed using **cosine similarity** between the embeddings of the input product and the recommended products.
3. High similarity scores indicate that the recommended products closely match the input product in terms of visual or textual attributes.

DATA VISUALIZATION-CONTENT BASED FILTERING:

Data visualization played a key role in analyzing and presenting insights from the datasets used in the AI Stylist project. Various visualizations were employed to better understand patterns and relationships between product attributes and user preferences.

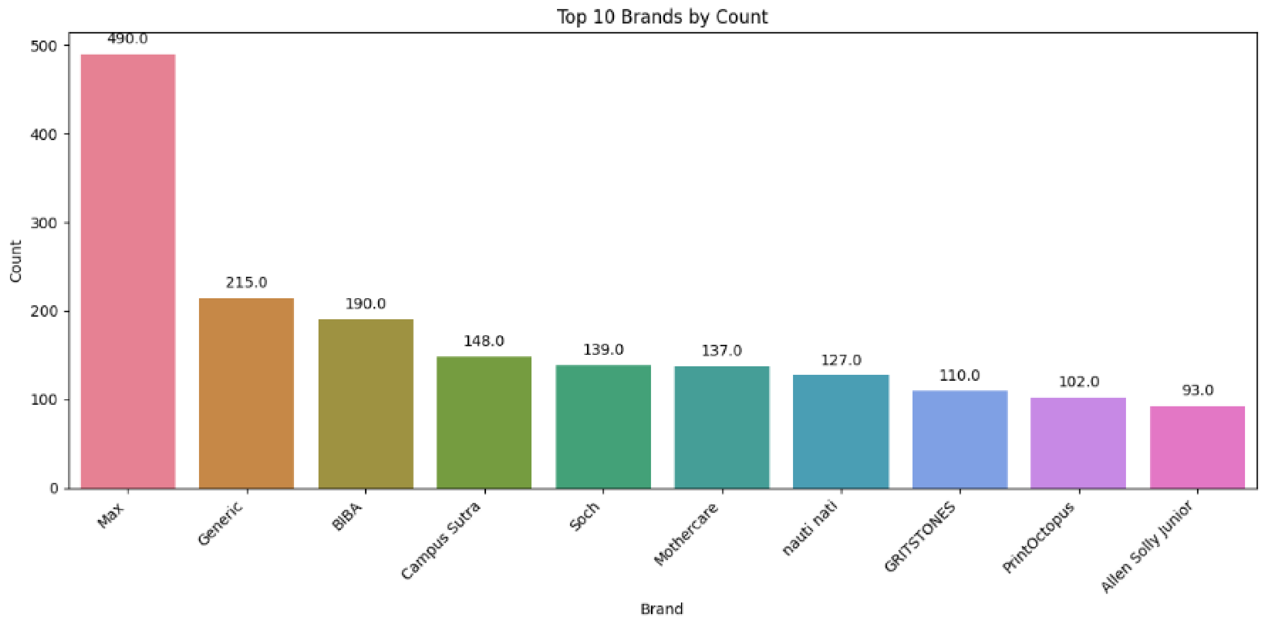
Histogram:

- Histogram is used for Sales price frequency.



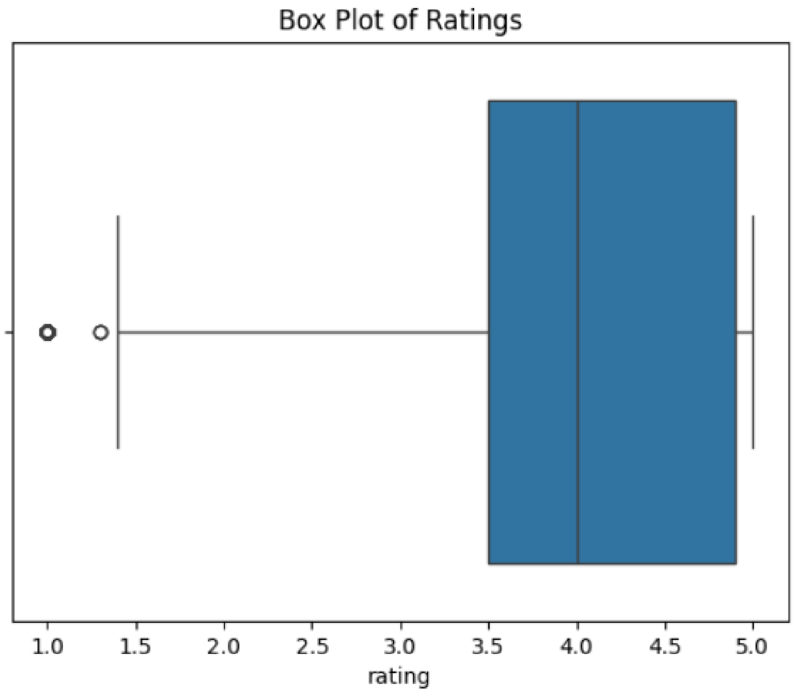
Bar plot:

1. Bar plot for visualizing the Top 10 brand by counts.



Box plot:

2. Box plot for visualizing the rating in the dataset



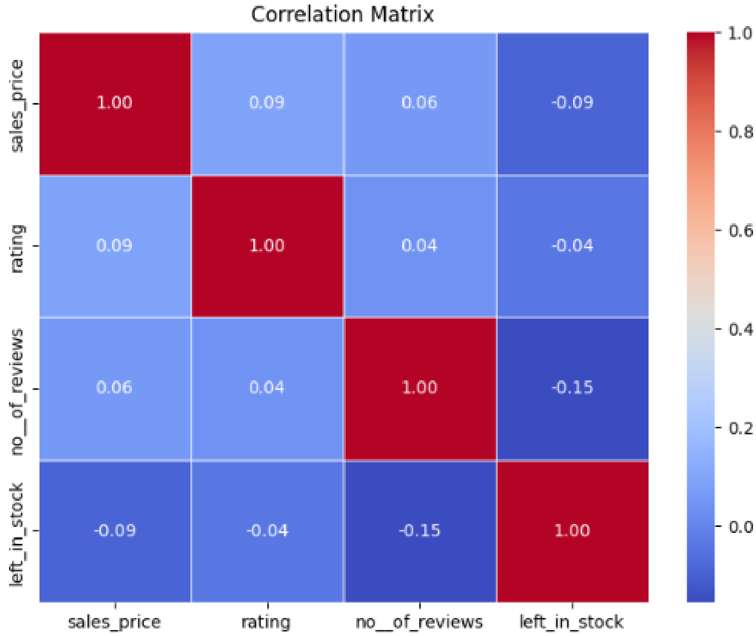
Scatter plot:

3. Scatter plot for understanding the outlier in the sales price and removing those outliers using the IQR.
4. Before Removing outliers:

1. After Removing outliers

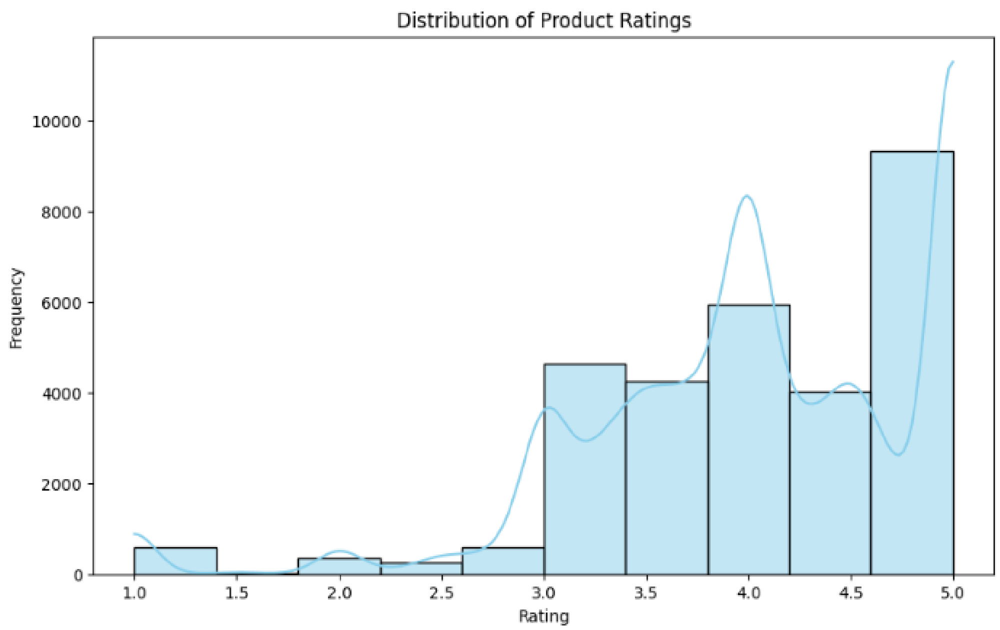
Correlation Matrix:

2. The correlation matrix is computed to analyze the relationships between numerical features.
3. A heatmap is used to visually represent these correlations, highlighting the strength and direction of their associations.



Histogram for Rating:

1. The histogram visualizes the distribution of product ratings, showing the frequency of different rating values.



DATA CLEANING:

Handling missing values:

2. Missing values are handled by taking median for the numeric columns such as sales price and rating.
3. “None” was assigned to the missing values in the non-numeric columns such as brand etc.

Removing stop words:

1. Stop words are common words that typically do not carry significant meaning in text analysis, such as "the," "a," "is," and "and." Removing stop words is a preprocessing step often used in natural language processing tasks to reduce noise and improve the efficiency of analysis.
2. NLTK library contains some predefined stop words and this library was used to remove the stop words in our dataset.

```
nltk.download('stopwords')
def clean_product_name(text):
    text = re.sub(r"[^a-zA-Z0-9 ]", "", text)
    words = text.lower().split()
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]
    cleaned_text = " ".join(words)

    return cleaned_text


data['cleaned_product_name'] = data['product_name'].apply(clean_product_name)
print(data['cleaned_product_name'].head())
```

Removing Special characters:

1. Special characters include symbols like @, #, $, %, &, *, !, and punctuation marks.

2. Special character removal is an essential preprocessing step to clean and standardize text data for analysis and model training.
3. A regular expression (regex) pattern such as [^a-zA-Z0-9\s] is used to identify characters that are not alphanumeric or whitespace.

```
import re

def remove_special_characters(text):

    cleaned_text = re.sub(r"[^a-zA-Z0-9 ]", "", text)
    return cleaned_text

data['product_name'] = data['product_name'].apply(remove_special_characters)


print(data['product_name'].head())
```

Removing Non-English words:

1. Removing non-English words is a critical preprocessing step in the AI Stylist project to ensure the textual data is consistent and relevant. This step helps standardize product names and improve the effectiveness of models relying on text-based features.
2. A regular expression (regex) pattern such as "[^\x00-\x7F]+" is used to identify non-english character and remove those characters.

```
import re

def remove_non_english_chars(text):
    pattern = r"[^\x00-\x7F]+"
    cleaned_text = re.sub(pattern, "", text)
    return cleaned_text


data['product_name'] = data['product_name'].apply(remove_non_english_chars)
```

Converting short to long form:

1. Expanding short forms (abbreviations or acronyms) to their full forms is a vital text preprocessing step. In the AI Stylist project, this ensures that product descriptions are clear and consistent, enabling better semantic understanding and similarity computation
2. Example:
   1) Blk -> Black
   2) Jr -> Junior

```
def reshape_to_long_format(data, id_vars=['asin', 'product_name']):
    data_columns = data.columns.tolist()
    value_vars = [col for col in data_columns if col not in id_vars]
    data_long = pd.melt(data,
                        id_vars=id_vars,
                        value_vars=value_vars,
                        var_name='attribute',
                        value_name='value')
    return data_long

data_long = reshape_to_long_format(data)
print(data_long.head())
```

Stemming and Lemmatization:

1. Stemming and lemmatization are essential preprocessing steps in text processing. In the AI Stylist project, they ensure that product descriptions are normalized by reducing words to their base or root forms, improving consistency and enabling better similarity calculations.
2. Stemming: Reduces words to their root form by removing prefixes or suffixes, often without considering the word's meaning. Stemming is rule-based and faster but can produce non-standard words
3. Example: "running, runner, runs" -> "run, run, run"

4. Lemmatization: Converts words to their base or dictionary form, considering the word's meaning and part of speech. Lemmatization is more accurate but computationally intensive.
5. Example: "running, better, studies" -> "run, good, study"

```
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.tokenize import word_tokenize


nltk.download('punkt')
nltk.download('wordnet')
nltk.download('punkt_tab')

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

def lemmatize_text(text):
    tokens = word_tokenize(text)
    lemmas = [lemmatizer.lemmatize(token) for token in tokens]
    return " ".join(lemmas)

def stem_text(text):
    tokens = word_tokenize(text)
    stems = [stemmer.stem(token) for token in tokens]
    return " ".join(stems)


data['lemmatized_product_name'] = data['product_name'].apply(lemmatize_text)
data['stemmed_product_name'] = data['product_name'].apply(stem_text)


print(data[['product_name', 'lemmatized_product_name', 'stemmed_product_name']].head())
```

CONTENT BASED FILTERING:

MODEL TRAINING:

1. Content-based filtering is a technique used in recommendation systems to suggest items based on the features of the items themselves, rather than user interactions or behaviors. This method analyzes product features such as product names, descriptions, brands, categories, and images to find similarities between items and recommend products that are most similar to a given input.
2. In the AI Stylist project, content-based filtering is used to recommend similar fashion items based on product descriptions, brand, and other relevant features.
3. Recommendation system was done by using Bag of words , TF-IDF , Word2vec , Pre trained models etc.

Bag of words:

4. The Bag of Words (BoW) is a popular text representation technique used in natural language processing (NLP) and recommendation systems. It transforms text data into a format that machine learning models can easily understand, making it easier to compare and analyze textual information. In the context of the AI Stylist project, BoW is used to represent product descriptions, names, and other textual features as numerical data for content-based filtering and other models.
5. The first step in the Bag of Words model is to tokenize the input text, which means breaking down the text into individual words (tokens). For example, a product description like "Men’s Casual Shirt" would be split into the tokens: ["Men’s", "Casual", "Shirt"]. These tokens are then used to create a vocabulary of unique words across the entire dataset.

6. The Bag of Words approach assigns each word in the vocabulary a position in the vector and then counts the frequency of each word in the document.
7. The Bag of Words (BoW) model represents text as a numerical vector based on the frequency of words in a document. The formula can be described mathematically as:

$$BoW = [f(w_1), f(w_2), f(w_3), \ldots, f(w_n)]$$

8. Once products are represented as vectors using BoW, various similarity metrics can be applied. The most commonly used similarity measure is **cosine similarity**, A measure of the cosine of the angle between two vectors. A higher cosine similarity score indicates more similarity between the documents.

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Where:

- $\mathbf{A} \cdot \mathbf{B}$ = Dot product of vectors $\mathbf{A}$ and $\mathbf{B}$, calculated as:

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i \cdot B_i$$

- $\|\mathbf{A}\|$ = Magnitude (Euclidean norm) of vector $\mathbf{A}$, given by:

$$\|\mathbf{A}\| = \sqrt{\sum_{i=1}^{n} A_i^2}$$

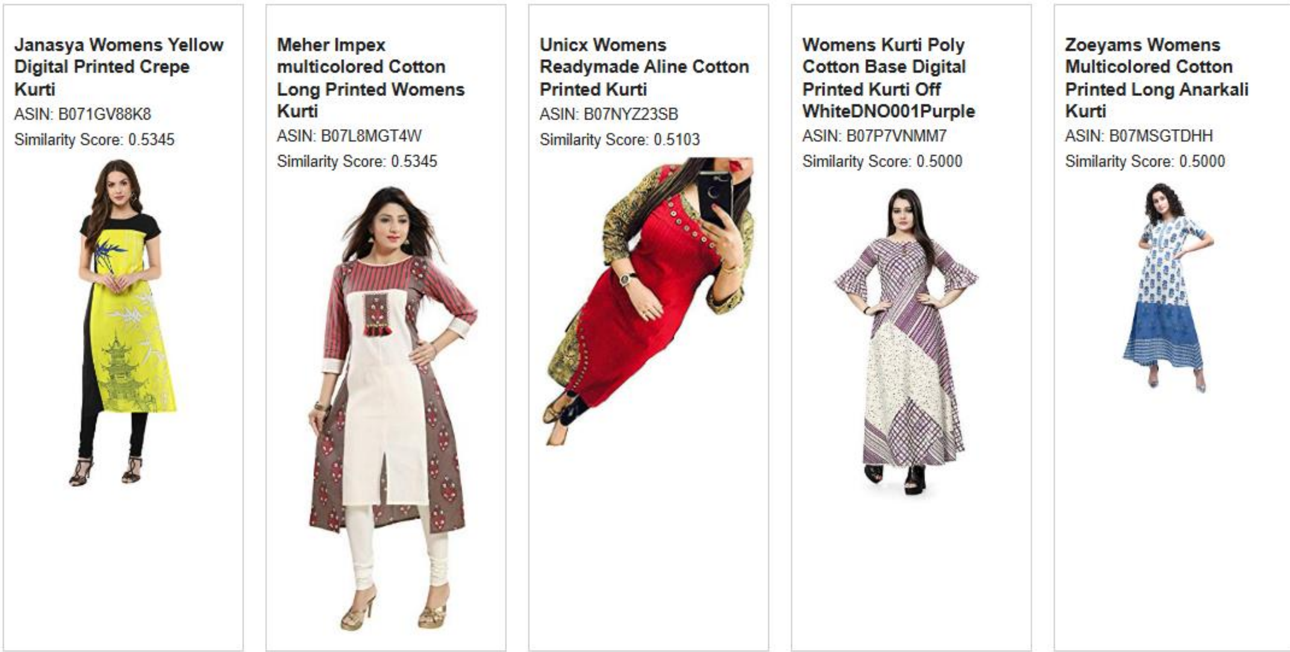
- $\|\mathbf{B}\|$ = Magnitude of vector $\mathbf{B}$, calculated similarly.

Code snippets:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import pairwise_distances
import pandas as pd
from IPython.display import display, HTML,Image

def recommend_products_count_vectorizer(product_id, num_products, data):
    product_names = data['product_name'].tolist()
    product_names = [
        name for name in product_names
        if isinstance(name, str) and name.strip() and not pd.isnull(name)
    ]
    vectorizer = CountVectorizer()
    bow_matrix = vectorizer.fit_transform(product_names)
    distance_matrix = pairwise_distances(bow_matrix, metric='cosine')
    distance_df = pd.DataFrame(
        distance_matrix, index=product_names, columns=product_names
    )
    target_product_name = data.loc[data['asin'] == product_id, 'product_name'].iloc[0]
    if target_product_name in distance_df.index:
        distances = distance_df.loc[target_product_name]

        sorted_distances = distances.sort_values()

        recommended_indices = sorted_distances.index[1:num_products + 1]
        similarity_scores = 1 - sorted_distances.iloc[1:num_products + 1].values

        recommended_products = data.loc[data['product_name'].isin(recommended_indices), 'asin'].tolist()
        results = list(zip(recommended_products, similarity_scores))
    else:
        results = []

    return results
```

```python
def display_recommended_products(recommended_products, data):
    html_content = "<div style='display: flex; flex-wrap: wrap;'>"
    for asin, similarity_score in recommended_products:
        product_info = data[data['asin'] == asin].iloc[0]
        image_url = product_info['medium']


        product_html = f"""
            <div style='border: 1px solid #ccc; padding: 10px; margin: 10px; width: 200px;'>
                <h3>{product_info['product_name']}</h3>
                <p>ASIN: {product_info['asin']}</p>
                <p>Similarity Score: {similarity_score:.4f}</p>
                """

        if image_url:
            product_html += f"<img src='{image_url}' style='max-width: 100%; height: auto;'/>"
        else:
            product_html += "<p>Image not available</p>"

        product_html += "</div>"
        html_content += product_html

    html_content += "</div>"
    display(HTML(html_content))


product_id = 'B01BKBMTW4'
recommended_products = recommend_products_count_vectorizer(product_id, 5, data)
display_recommended_products(recommended_products, data)
```

Output:



TF-IDF:

1. TF-IDF (Term Frequency-Inverse Document Frequency) is a text representation technique that evaluates the importance of words in a document relative to a collection of documents (corpus). It is widely used in natural language processing and recommendation systems to represent textual features. Unlike Bag of Words, TF-IDF considers the frequency of words in a document and their rarity across the corpus, giving higher importance to unique and meaningful words.
2. **Term Frequency (TF)**: Measures how often a term appears in a specific document, normalized by the total number of terms in the document.

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

3. **Inverse Document Frequency (IDF)**: Measures how unique or rare a term is across the corpus. Rare terms receive higher weights.

$$IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term } t}\right)$$

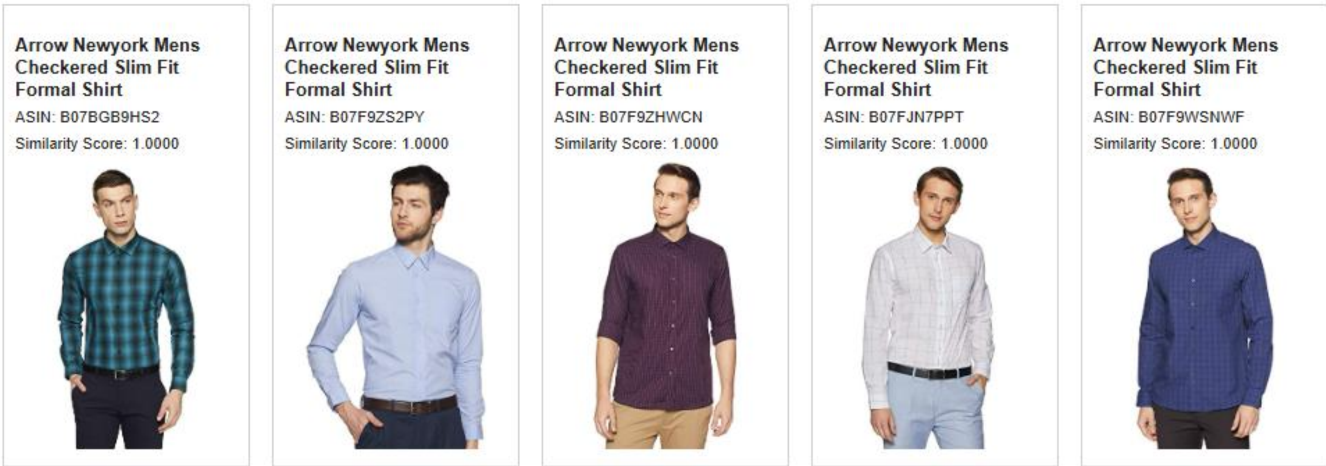
4. **TF-IDF Score**: The final score for each term is the product of TF and IDF

$$TFIDF(t,d) = TF(t,d) \times IDF(t)$$

Code snippets:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from IPython.display import display, HTML,Image

def recommend_products_tfidf(product_id, num_products, data):
  vectorizer = TfidfVectorizer()
  tfidf_vectors = vectorizer.fit_transform(data['product_name'])
  if product_id in data['asin'].values:
    product_index = data.index[data['asin'] == product_id].tolist()[0]
    cosine_similarities = cosine_similarity(tfidf_vectors[product_index], tfidf_vectors).flatten()
    related_product_indices = cosine_similarities.argsort()[:-num_products-2:-1][1:]
    recommended_products = data.iloc[related_product_indices]['asin'].tolist()
    similarity_scores = cosine_similarities[related_product_indices]

    results = list(zip(recommended_products, similarity_scores))
    return results
  else:
    print(f"Product ID {product_id} not found in the data.")
    return []


def display_recommended_products(recommended_products, data):
    html_content = "<div style='display: flex; flex-wrap: wrap;'>"
    for asin, similarity_score in recommended_products:
        product_info = data[data['asin'] == asin].iloc[0]
        image_url = product_info['medium']


        product_html = f"""
            <div style='border: 1px solid #ccc; padding: 10px; margin: 10px; width: 200px;'>
                <h3>{product_info['product_name']}</h3>
                <p>ASIN: {product_info['asin']}</p>
                <p>Similarity Score: {similarity_score:.4f}</p>
                """

        if image_url:
            product_html += f"<img src='{image_url}' style='max-width: 100%; height: auto;'/>"
        else:
            product_html += "<p>Image not available</p>"

        product_html += "</div>"
        html_content += product_html

    html_content += "</div>"
    display(HTML(html_content))

product_id = 'B07F9ZS2PY'
recommended_products = recommend_products_tfidf(product_id, 5, data)
display_recommended_products(recommended_products, data)
```

Output:



Word2vec:

5. Word2Vec is a neural network-based model used for generating vector representations of words, capturing their semantic meaning and relationships. Unlike traditional text representation methods like Bag of Words and TF-IDF, Word2Vec learns the context of words from a large corpus and embeds them into a continuous vector space. These embeddings allow for semantic similarity between words to be measured, making Word2Vec a powerful tool for recommendation systems.
6. In the **AI Stylist** project, Word2Vec is used to process product names and descriptions, converting them into vector representations. These embeddings are then leveraged to calculate the similarity between products using metrics like cosine similarity. For example, if two products have names with similar semantic meanings (e.g., "denim jeans" and "blue jeans"), their embeddings will be closer, allowing the recommendation system to identify them as related items.

7. Word2Vec generates dense, continuous vectors for words, where semantically similar words are closer in the vector space. The embeddings are learned using one of two approaches:
8. **Skip-Gram Model**: Predicts the context words given a target word.

$$P(w_{context}|w_{target}) = \frac{\exp(\mathbf{v}_{context} \cdot \mathbf{v}_{target})}{\sum_{w \in V} \exp(\mathbf{v}_w \cdot \mathbf{v}_{target})}$$

Where:

- $\mathbf{v}_{target}$ = Vector representation of the target word.
- $\mathbf{v}_{context}$ = Vector representation of a context word.
- V = Vocabulary size.

9. **Continuous Bag of Words (CBOW) Model**: Predicts a target word based on its surrounding context words.

$$P(w_{target}|w_{context}) = \frac{\exp(\mathbf{v}_{context} \cdot \mathbf{v}_{target})}{\sum_{w \in V} \exp(\mathbf{v}_w \cdot \mathbf{v}_{context})}$$

10. The main advantage of Word2Vec in the project is its ability to understand the context of words, enabling the system to provide recommendations that are not only syntactically similar but also semantically meaningful. This enhances the accuracy and relevance of product suggestions, significantly improving the user experience in the fashion recommendation system.

Code snippet:

```
import nltk
import numpy as np
import pandas as pd
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from sklearn.metrics.pairwise import cosine_similarity
from IPython.display import display, HTML, Image
import re

nltk.download('punkt')


def remove_non_english_chars(text):
    pattern = r"[^\x00-\x7F]+"
    cleaned_text = re.sub(pattern, "", text)
    return cleaned_text

def recommend_products_word2vec(product_id, num_products, data):
    data['product_name_cleaned'] = data['product_name'].apply(remove_non_english_chars)
    all_product_names = data['product_name_cleaned'].tolist()
    tokenized_product_names = [word_tokenize(name) for name in all_product_names if isinstance(name, str)]
    model = Word2Vec(tokenized_product_names, vector_size=1000, window=5, min_count=1, workers=4)
    product_name = data.loc[data['asin'] == product_id, 'product_name_cleaned'].iloc[0]
    product_embedding = get_product_embedding(product_name, model)


    similarities = []

    for index, row in data.iterrows():
        if row['asin'] != product_id:
            other_product_name = row['product_name_cleaned']
            other_product_embedding = get_product_embedding(other_product_name, model)
            if other_product_embedding is not None:
                similarity = cosine_similarity(product_embedding.reshape(1, -1), other_product_embedding.reshape(1, -1))[0][0]
                similarities.append((row['asin'], similarity))


    sorted_similarities = sorted(similarities, key=lambda item: item[1], reverse=True)
    recommended_products = sorted_similarities[:num_products]

    return recommended_products

def get_product_embedding(product_name, model):
    try:
        tokens = word_tokenize(product_name)
        embeddings = [model.wv[token] for token in tokens if token in model.wv]
        if embeddings:
            return np.mean(embeddings, axis=0)
        else:
            return None
    except TypeError:
        return None
```

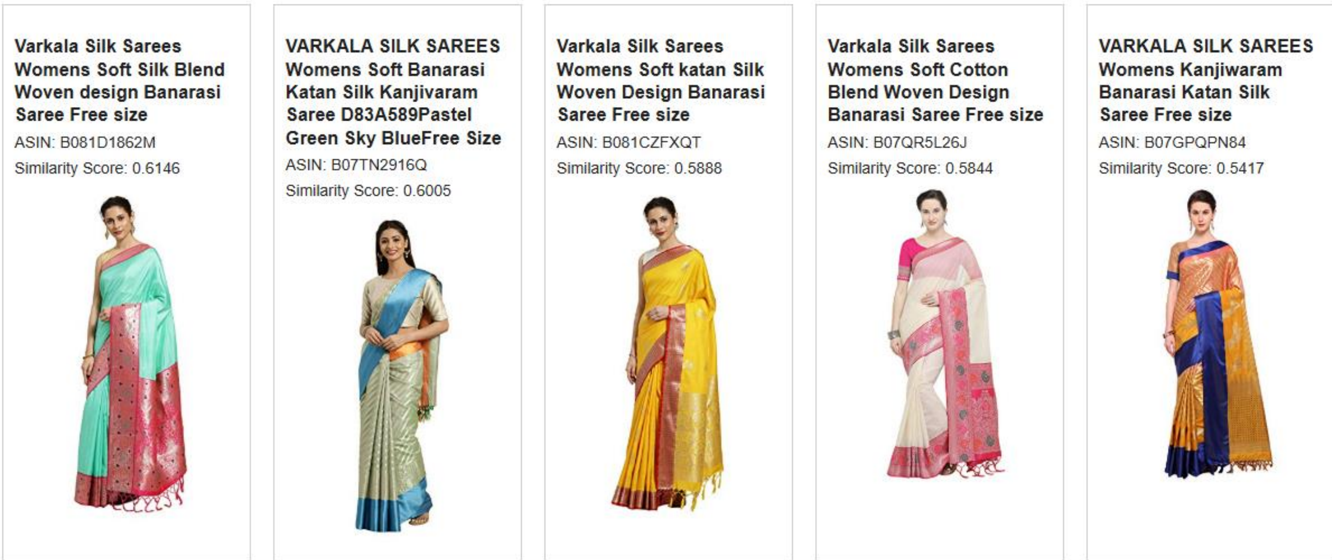
```python
def display_recommended_products(recommended_products, data):
    html_content = "<div style='display: flex; flex-wrap: wrap;'>"
    for asin, similarity_score in recommended_products:
        product_info = data[data['asin'] == asin].iloc[0]
        image_url = product_info['medium']


        product_html = f"""
            <div style='border: 1px solid #ccc; padding: 10px; margin: 10px; width: 200px;'>
                <h3>{product_info['product_name']}</h3>
                <p>ASIN: {product_info['asin']}</p>
                <p>Similarity Score: {similarity_score:.4f}</p>
                """

        if image_url:
            product_html += f"<img src='{image_url}' style='max-width: 100%; height: auto;'/>"
        else:
            product_html += "<p>Image not available</p>"

        product_html += "</div>"
        html_content += product_html

    html_content += "</div>"
    display(HTML(html_content))

product_id = 'B07HNQK9JX'
recommended_products = recommend_products_word2vec(product_id, 5, data)
display_recommended_products(recommended_products, data)
```

Output:

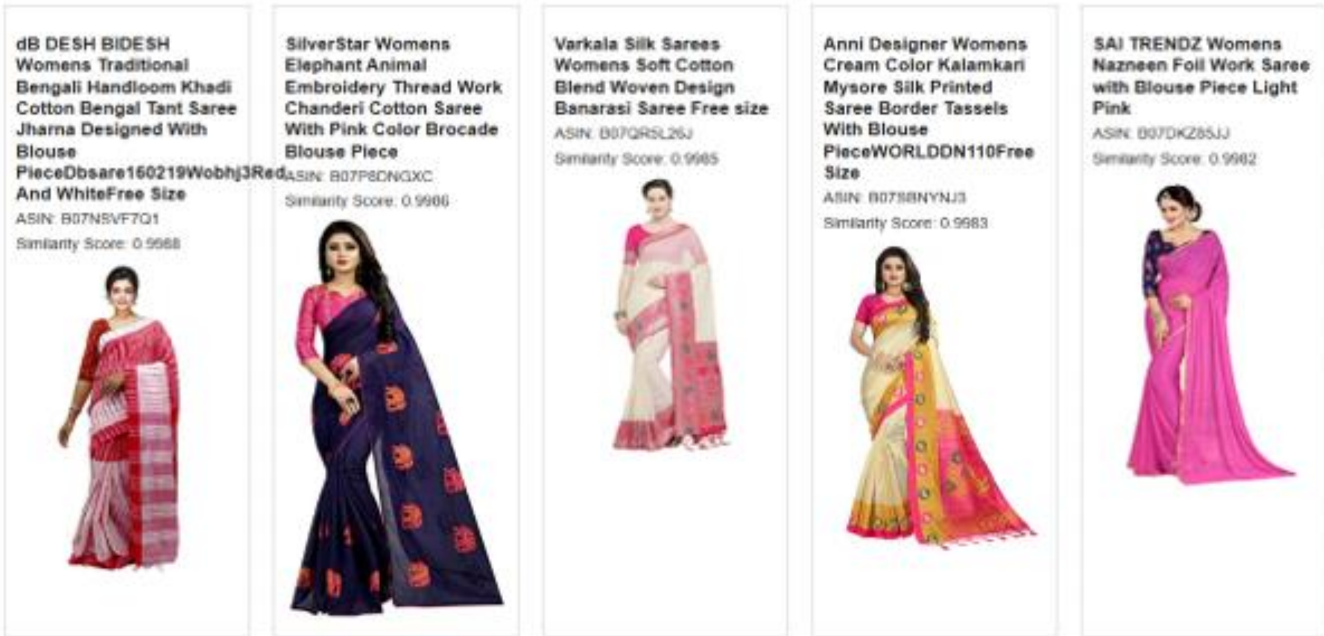
TF-IDF vs Word2vec:

11. TF-IDF and Word2Vec are widely used techniques for text representation, each serving different purposes. **TF-IDF** focuses on word frequency and importance, generating sparse, high-dimensional vectors. It is effective for tasks requiring precise word matching but lacks the ability to capture deeper semantic relationships. This approach works well in scenarios where exact matches or word importance relative to the dataset are critical.
12. **Word2Vec**, on the other hand, creates dense, low-dimensional embeddings that capture the semantic and contextual meaning of words. It uses training models like Skip-Gram and CBOW to learn relationships between words based on their usage in context. This makes Word2Vec effective for identifying synonyms and nuanced word meanings.
13. Combining TF-IDF's statistical approach with Word2Vec's semantic depth enhances the system's ability to offer precise and meaningful fashion suggestions.
1. TF-IDF focuses on word frequencies and inverse document frequencies within the dataset, while Word2Vec creates embeddings that capture semantic relationships between words.
1. In one instance, using the same product ID was give as the input. As per the similarity score between the two recommendations, the Word2Vec approach appeared to yield more relevant recommendations compared to TF-IDF.

Output of TF-IDF:



Output of Word2vec:



TF-IDF and Word2Vec Integration:

The AI Stylist project uses **TF-IDF** and **Word2Vec** to recommend fashion products based on similarities in product names. TF-IDF measures word importance within the product catalog, while Word2Vec generates word embeddings that capture semantic meaning. By combining both techniques, the model effectively suggests products with both lexical and semantic similarity.

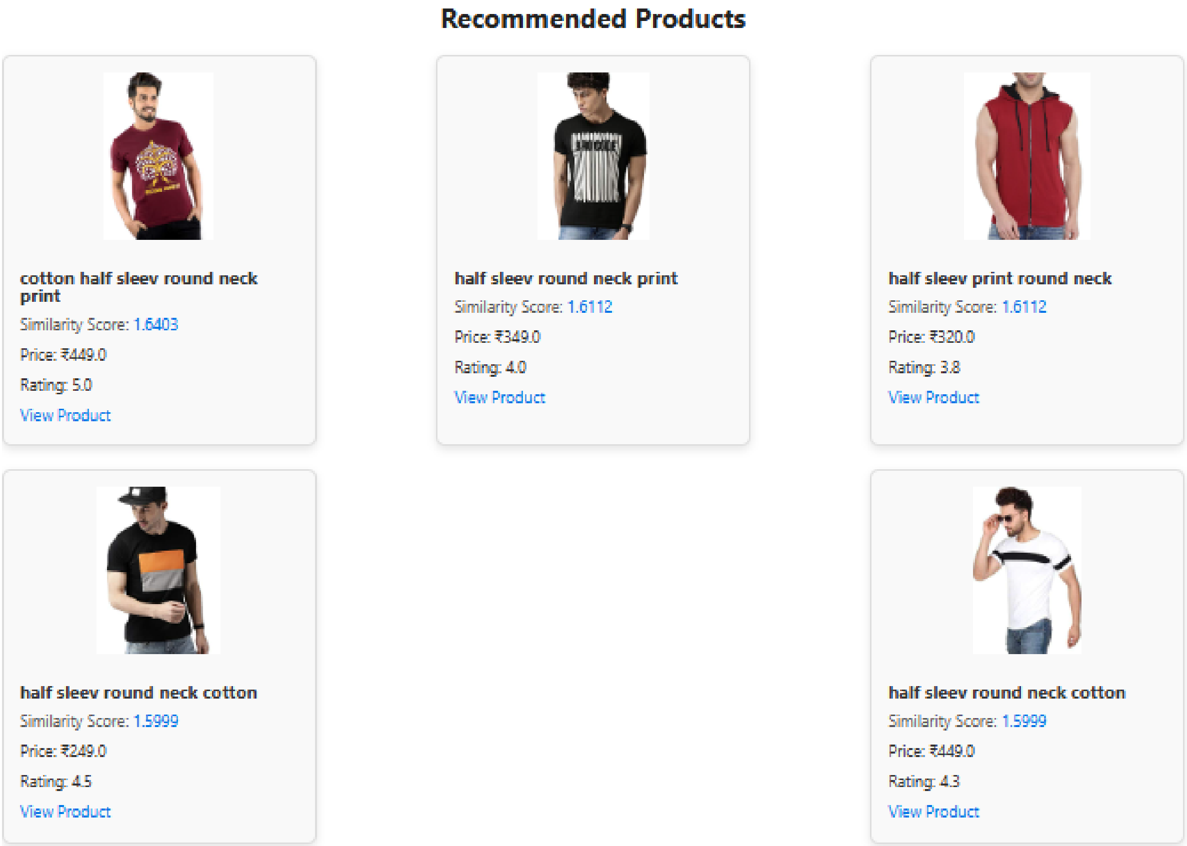
Model Functionality

1. Preprocessing: Product names are tokenized into words for analysis.
2. Word2Vec Model: A Word2Vec model is trained on product names to create word embeddings.
3. Similarity Calculation: Cosine similarity measures the similarity between products using both TF-IDF and Word2Vec.
4. Recommendations: Similarity scores are combined to recommend the top 5 products.
5. Display: Recommended products are shown with details like name, price, and rating.

CODE SNIPPET:

```
# Step 5: Function to get recommendations based on both TF-IDF and Word2Vec
def get_recommendations(asin_value, data):
    # Preprocess product names and train the Word2Vec model
    tokenized_data = preprocess_product_names(data)
    word2vec_model = train_word2vec_model(tokenized_data)

    # Step 5.1: Calculate TF-IDF similarities
    vectorizer = TfidfVectorizer(stop_words='english')
    X = vectorizer.fit_transform(data['product_name'])
    tfidf_cosine_sim = cosine_similarity(X, X, dense_output=False)

    # Step 5.2: Calculate Word2Vec similarities
    product_vectors = [get_product_vector(name, word2vec_model) for name in data['product_name']]
    product_idx = data[data['asin'] == asin_value].index[0]
    target_product_vector = get_product_vector(data.iloc[product_idx]['product_name'], word2vec_model)
    word2vec_cosine_sim = cosine_similarity([target_product_vector], product_vectors).flatten()

    # Step 5.3: Combine TF-IDF and Word2Vec similarity scores
    combined_similarities = tfidf_cosine_sim[product_idx].toarray().flatten() + word2vec_cosine_sim

    # Step 5.4: Sort products based on the combined similarity scores (excluding the product itself)
    sorted_indices = combined_similarities.argsort()[-6:-1][::-1]  # Top 5 similar products
    recommended_products = data.iloc[sorted_indices]
```

Output:



Brand-Based Recommendation Model

This model recommends products from the same brand using Word2Vec embeddings based on the brand name. By leveraging the semantic similarity between the target product's brand and other products in the dataset, this model suggests similar items within the same brand.

Model Functionality:

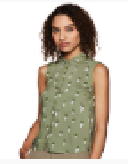
1. Preprocessing: Tokenizes brand names into individual words.
2. Word2Vec Embeddings: Trains a Word2Vec model on the tokenized brand names to capture semantic similarities.
3. Brand Similarity: For a given product, it identifies other products from the same brand and computes similarity scores based on Word2Vec embeddings.

CODE SNIPPET:

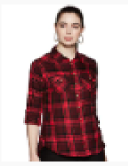
```python
# Step 5: Function to get recommendations strictly from the same brand
def get_brand_based_recommendations(asin_value, data):
    # Preprocess brand names and train the Word2Vec model for brands
    tokenized_data = preprocess_brand_names(data)
    brand_word2vec_model = train_brand_word2vec_model(tokenized_data)

    # Step 5.1: Identify the brand of the target product
    target_product = data[data['asin'] == asin_value]
    if target_product.empty:
        print(f"No product found with ASIN: {asin_value}")
        return

    target_brand = target_product.iloc[0]['brand']

    # Step 5.2: Filter data for products from the same brand
    same_brand_products = data[data['brand'] == target_brand].reset_index(drop=True)

    if len(same_brand_products) < 2:
        print(f"Not enough products found for brand: {target_brand}")
        return

    # Step 5.3: Calculate Word2Vec similarities for filtered products
    brand_vectors = [get_brand_vector(brand, brand_word2vec_model) for brand in same_brand_products['brand']]
    target_brand_vector = get_brand_vector(target_brand, brand_word2vec_model)
    word2vec_cosine_sim = cosine_similarity([target_brand_vector], brand_vectors).flatten()

    # Step 5.4: Sort products based on Word2Vec similarity scores (excluding the product itself)
    target_index = same_brand_products[same_brand_products['asin'] == asin_value].index[0]
    word2vec_cosine_sim[target_index] = -1  # Exclude the target product itself
    sorted_indices = word2vec_cosine_sim.argsort()[-4:][::-1]  # Top 4 recommendations

    recommended_products = same_brand_products.iloc[sorted_indices]
```

OUTPUT:

Products from Brand: Max



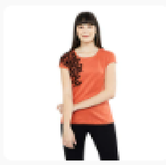
Max Women's Regular Fit Shirt
Brand: Max
Similarity Score: 0.9096
Price: ₹399.0
Rating: 4.3
View Product



Max Women's Regular Fit Shirt
Brand: Max
Similarity Score: 0.9096
Price: ₹599.0
Rating: 4.5
View Product

Max Women's Regular Fit Top
Brand: Max
Similarity Score: 0.8523
Price: ₹279.0
Rating: 4.3
View Product

Max Women's Regular fit Top
Brand: Max
Similarity Score: 0.8523
Price: ₹249.0
Rating: 4.0
View Product



Max Women's Regular fit Top
Brand: Max
Similarity Score: 0.8523
Price: ₹299.0
Rating: 3.2
View Product

Image-Based Recommendation Model

This model recommends products based on image similarity, using deep learning to extract features from product images and compute similarity scores. By leveraging the VGG19 model pre-trained on ImageNet, the model extracts meaningful features from product images to generate recommendations similar to the input image.

What the model does:

4. Feature Extraction: Uses a modified VGG19 model to extract features from product images.
5. Cosine Similarity: Computes cosine similarity between the feature vectors of images to find similar products.
6. Recommendations: Returns the top 5 similar products based on image similarity, excluding the input product itself.

CODE SNIPPET:

```
d# Load the pre-trained VGG19 model for feature extraction
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
feature_extractor = Sequential([
    base_model,
    Flatten(),
    Dense(4096, activation='relu'),
    Dense(1000, activation='relu')
])


def extract_features(image_url):
    """
    Extracts features from an image URL using the modified VGG19 model.
    """
    try:
        response = requests.get(image_url)
        if response.status_code == 200:
            img = Image.open(BytesIO(response.content)).convert('RGB')
            img = img.resize((224, 224))  # Resize to match VGG19 input size
            img_array = img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
            img_array = preprocess_input(img_array)  # Preprocess for VGG19
            features = feature_extractor.predict(img_array)
            return features.flatten()
        else:
            print(f"Failed to fetch image from URL: {image_url}")
            return np.zeros((1000,))
    except Exception as e:
        print(f"Error processing image URL {image_url}: {e}")
        return np.zeros((1000,))
```

```python
def preprocess_data(data, n_samples=100):
    """
    Preprocess the dataset by reducing it to n_samples images and splitting it into training and testing sets.
    """
    data = data.head(n_samples).copy()  # Limit to the first n_samples products

    # Extract features for all product images
    data['image_features'] = data['medium'].apply(extract_features)
    features_matrix = np.vstack(data['image_features'].to_numpy())  # Create a matrix of features

    # One-hot encode the ASIN values
    encoder = OneHotEncoder(sparse_output=False)
    encoded_labels = encoder.fit_transform(data[['asin']])

    # Split data into training and testing sets
    train_data, test_data, train_labels, test_labels = train_test_split(
        features_matrix, encoded_labels, test_size=0.2, random_state=42
    )

    return train_data, test_data, train_labels, test_labels, data, encoder
```

```python
def recommend_based_on_asin(asin_value, data, top_n=5):
    """
    Recommend products based on the image of a product identified by its ASIN.
    """
    target_product = data[data['asin'] == asin_value]
    if target_product.empty:
        print(f"No product found with ASIN: {asin_value}")
        return

    target_image_url = target_product.iloc[0]['medium']
    target_features = extract_features(target_image_url)

    data['similarity_score'] = data['image_features'].apply(
        lambda x: cosine_similarity([target_features], [x]).flatten()[0]
    )

    recommended_products = (
        data[data['asin'] != asin_value]
        .sort_values(by='similarity_score', ascending=False)
        .head(top_n)
    )
```

Explanation:

7. **extract_features(image_url)**: Extracts the feature vector from a product image by passing it through a pre-trained VGG19 model.

8. **preprocess_data(data, n_samples)**: Prepares the dataset by extracting image features for a specified number of samples.

9. **recommend_based_on_asin(asin_value, data)**: Recommends the top N similar products to a given product based on its image's feature vector and cosine similarity.

OUTPUT:

```
Epoch 1/10: Training...
Loss: 0.5780, Accuracy: 98.93%
Epoch 2/10: Training...
Loss: 0.9434, Accuracy: 99.55%
Epoch 3/10: Training...
Loss: 0.0267, Accuracy: 89.00%
Epoch 4/10: Training...
Loss: 0.9639, Accuracy: 84.94%
Epoch 5/10: Training...
Loss: 0.7217, Accuracy: 86.69%
Epoch 6/10: Training...
Loss: 0.0024, Accuracy: 80.34%
Epoch 7/10: Training...
Loss: 0.1662, Accuracy: 85.17%
Epoch 8/10: Training...
Loss: 0.9639, Accuracy: 82.57%
Epoch 9/10: Training...
Loss: 0.9882, Accuracy: 83.66%
Epoch 10/10: Training...
Loss: 0.4075, Accuracy: 94.35%
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 584ms/step
```

Recommended Products Similar to ASIN: B07QCYNW3C

stop fashion knee long w style pack

Brand: 1 Stop Fashion

Similarity Score: 0.6134

View Product

stop fashion crepe straight pack

Brand: 1 Stop Fashion

Similarity Score: 0.6111

View Product

stop fashion crepe straight pack

Brand: 1 Stop Fashion

Similarity Score: 0.6065

View Product

stop fashion crepe straight pack

Brand: 1 Stop Fashion

Similarity Score: 0.6031

View Product

stop fashion crepe straight pack

Brand: 1 Stop Fashion

Similarity Score: 0.5960

View Product

Recommendation Based on Product Name and Brand Name

This function recommends products based on a combination of the product name and brand name. It uses TF-IDF (Term Frequency-Inverse Document Frequency) to compute the similarity between the input product and other products from the same brand, then sorts the recommendations based on product ratings and similarity score.

Key Components:

10. TF-IDF Vectorization: Converts product names into vector representations to capture textual features.
11. Cosine Similarity: Measures the similarity between the input product name and other products' names.
12. Sorting: Recommends products with the highest ratings first, breaking ties with similarity scores.

CODE SNIPPET:

```
#recommendation based on combination of product name and brand name
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
from IPython.display import display, HTML

# Assuming the 'filtered_data' DataFrame has already been created and preprocessed

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english')

# Fit and transform the product names
product_name_vectors = vectorizer.fit_transform(filtered_data['product_name'])

# Function to recommend products from the same brand based on similarity and rating, with images and brand name
def recommend_products_by_brand(product_name, brand, num_recommendations=5):
    """
    Recommends similar products from the same brand, sorted by their ratings (highest first), and includes images and brand name.
    """
    # Filter products by the specified brand
    same_brand_products = filtered_data[filtered_data['brand'] == brand].reset_index(drop=True)

    if same_brand_products.empty:
        print(f"No products found for brand: {brand}")
        return

    # Transform the input product name into a vector
    input_vector = vectorizer.transform([product_name])

    # Compute cosine similarity between the input and all other products in the same brand
    same_brand_vectors = vectorizer.transform(same_brand_products['product_name'])
    similarity_scores = cosine_similarity(input_vector, same_brand_vectors).flatten()

    # Add similarity scores to the brand-specific products
    same_brand_products['similarity_score'] = similarity_scores

    # Sort products by rating (highest first), then by similarity score (highest first)
    recommendations = same_brand_products.sort_values(by=['rating', 'similarity_score'], ascending=[False, False]).head(num_recommendations)
```

OUTPUT:

```
# Example usage:
product_name_input = "jogger"  # Replace with the input product name
brand_input = "max"  # Replace with the desired brand
recommend_products_by_brand(product_name_input, brand_input, num_recommendations=5)
```

Top 5 Similar Products from max

max jogger

max

Price: ₹369.0

Rating: 5.0

View Product

max jogger

max

Price: ₹369.0

Rating: 5.0

View Product

max regular fit jogger

max

Price: ₹369.0

Rating: 5.0

View Product

max relax fit jogger

max

Price: ₹369.0

Rating: 5.0

View Product

max cargo jogger

max

Price: ₹248.0

Rating: 5.0

View Product

Collaborative Filtering for Fashion Product Recommendation

In the context of our AI Stylist project, collaborative filtering forms a cornerstone for creating personalized product recommendations. This section documents the preprocessing steps and dataset exploration, which are prerequisites for implementing collaborative filtering. The goal is to analyze user preferences and item similarities for generating accurate recommendations.

Dataset Preprocessing and Exploration

The dataset used is extracted from a ZIP file stored in Google Drive. It contains metadata about fashion products, such as articleType, gender, season, and images of the products. Below are the steps and code snippets that were executed for dataset preparation and initial exploration.

Dataset Loading and Inspection

The dataset (styles.csv) was read into a DataFrame, skipping any problematic lines. Key statistics, unique values, and null value distributions were inspected.

```
[ ] import pandas as pd

    # Update with the actual dataset file name
    dataset_path = 'extracted_dataset/styles.csv'
    df = pd.read_csv(dataset_path, on_bad_lines='skip')
    # Preview the data
    print(df.head())
```

```
      id gender masterCategory subCategory  articleType baseColour  season  \
0  15970    Men        Apparel     Topwear       Shirts  Navy Blue    Fall
1  39386    Men        Apparel  Bottomwear        Jeans       Blue  Summer
2  59263  Women    Accessories     Watches      Watches     Silver  Winter
3  21379    Men        Apparel  Bottomwear  Track Pants      Black    Fall
4  53759    Men        Apparel     Topwear      Tshirts       Grey  Summer

     year   usage                             productDisplayName
0  2011.0  Casual               Turtle Check Men Navy Blue Shirt
1  2012.0  Casual             Peter England Men Party Blue Jeans
2  2016.0  Casual                       Titan Women Silver Watch
3  2011.0  Casual  Manchester United Men Solid Black Track Pants
4  2012.0  Casual                          Puma Men Grey T-shirt
```

```
[ ] print("Column Names:")
    print(df.columns)
```

```
Column Names:
Index(['id', 'gender', 'masterCategory', 'subCategory', 'articleType',
       'baseColour', 'season', 'year', 'usage', 'productDisplayName'],
      dtype='object')
```

```python
# prompt: diplay all the columns with data types

df.dtypes
```

	0
id	int64
gender	object
masterCategory	object
subCategory	object
articleType	object
baseColour	object
season	object
year	float64
usage	object
productDisplayName	object

dtype: object

```python
print("\nNull Values in Each Column:")
print(df.isnull().sum())
```

```
Null Values in Each Column:
id                      0
gender                  0
masterCategory          0
subCategory             0
articleType             0
baseColour             15
season                 21
year                    1
usage                 317
productDisplayName      7
dtype: int64
```

```python
print("\nUnique Values in Each Column:")
print(df.nunique())
```

```
Unique Values in Each Column:
id                    44424
gender                    5
masterCategory            7
subCategory              45
articleType             143
baseColour               46
season                    4
year                     13
usage                     8
productDisplayName    31121
dtype: int64
```

```python
# Shape of the dataset
print("\nShape of the dataset (rows, columns):")
print(df.shape)
```

```
Shape of the dataset (rows, columns):
(44424, 10)
```

```python
# Data types and non-null count
print("\nDataset Info:")
print(df.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44424 entries, 0 to 44423
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  44424 non-null  int64
 1   gender              44424 non-null  object
 2   masterCategory      44424 non-null  object
 3   subCategory         44424 non-null  object
 4   articleType         44424 non-null  object
 5   baseColour          44409 non-null  object
 6   season              44403 non-null  object
 7   year                44423 non-null  float64
 8   usage               44107 non-null  object
 9   productDisplayName  44417 non-null  object
dtypes: float64(1), int64(1), object(8)
memory usage: 3.4+ MB
None
```

```python
# Summary statistics
print("\nSummary Statistics:")
print(df.describe(include='all'))
```

```
Summary Statistics:
                  id gender masterCategory subCategory articleType baseColour  \
count   44424.000000  44424          44424       44424       44424      44409
unique           NaN      5              7          45         143         46
top              NaN    Men        Apparel     Topwear     Tshirts      Black
freq             NaN  22147          21397       15402        7067       9728
mean    29696.334301    NaN            NaN         NaN         NaN        NaN
std     17049.490518    NaN            NaN         NaN         NaN        NaN
min      1163.000000    NaN            NaN         NaN         NaN        NaN
25%     14768.750000    NaN            NaN         NaN         NaN        NaN
50%     28618.500000    NaN            NaN         NaN         NaN        NaN
75%     44683.250000    NaN            NaN         NaN         NaN        NaN
max     60000.000000    NaN            NaN         NaN         NaN        NaN

        season          year  usage              productDisplayName
count    44403  44423.000000  44107                           44417
unique       4           NaN      8                           31121
top     Summer           NaN Casual  Lucera Women Silver Earrings
freq     21472           NaN  34406                              82
mean       NaN   2012.806497    NaN                             NaN
std        NaN      2.126480    NaN                             NaN
min        NaN   2007.000000    NaN                             NaN
25%        NaN   2011.000000    NaN                             NaN
50%        NaN   2012.000000    NaN                             NaN
75%        NaN   2015.000000    NaN                             NaN
max        NaN   2019.000000    NaN                             NaN
```

Data Overview

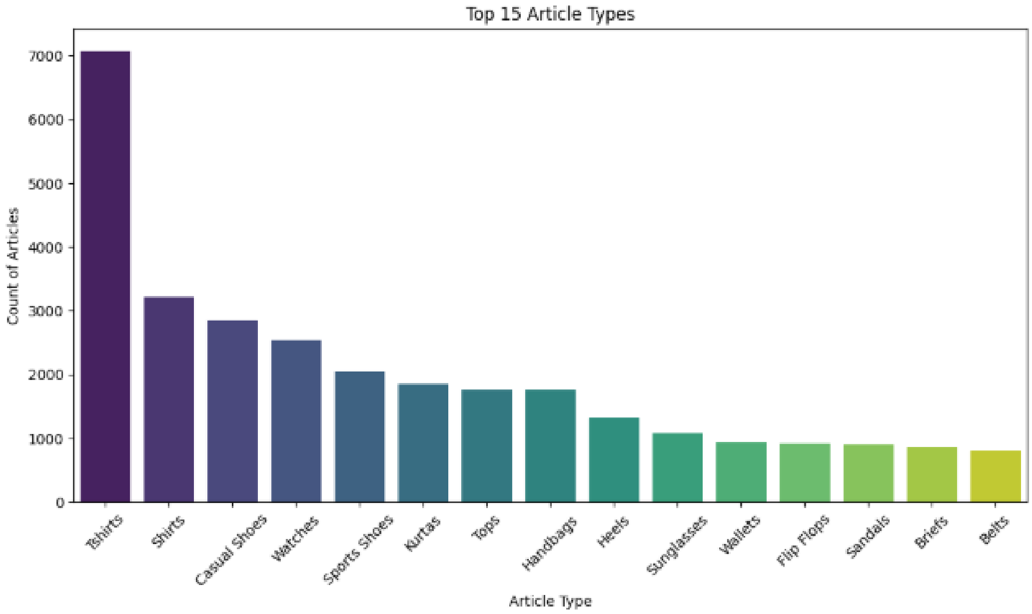
The dataset contains 44,424 rows and 10 columns. The most frequent articleType is "Tshirts," and the majority of products are targeted at the "Men" category. There are missing values in columns like baseColour, season, and usage.

```python
# Count articles by `articleType`
top_article_type = df['articleType'].value_counts().head(15)

# Display the result
print(top_article_type)
```

```
articleType
Tshirts         7067
Shirts          3217
Casual Shoes    2845
Watches         2542
Sports Shoes    2036
Kurtas          1844
Tops            1762
Handbags        1759
Heels           1323
Sunglasses      1073
Wallets          936
Flip Flops       914
Sandals          897
Briefs           849
Belts            813
Name: count, dtype: int64
```

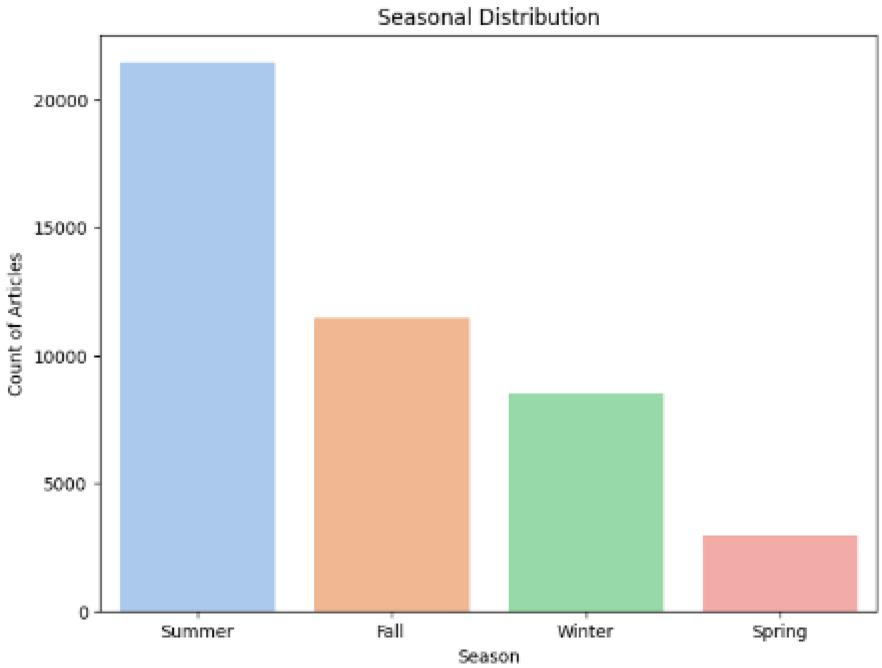
Top 15 Article Types

We visualized the most common article types to understand the distribution of products.



Seasonal Distribution

The dataset was analyzed to understand product availability across seasons.



Master Category Distribution

To explore the main product categories, a bar chart was generated.

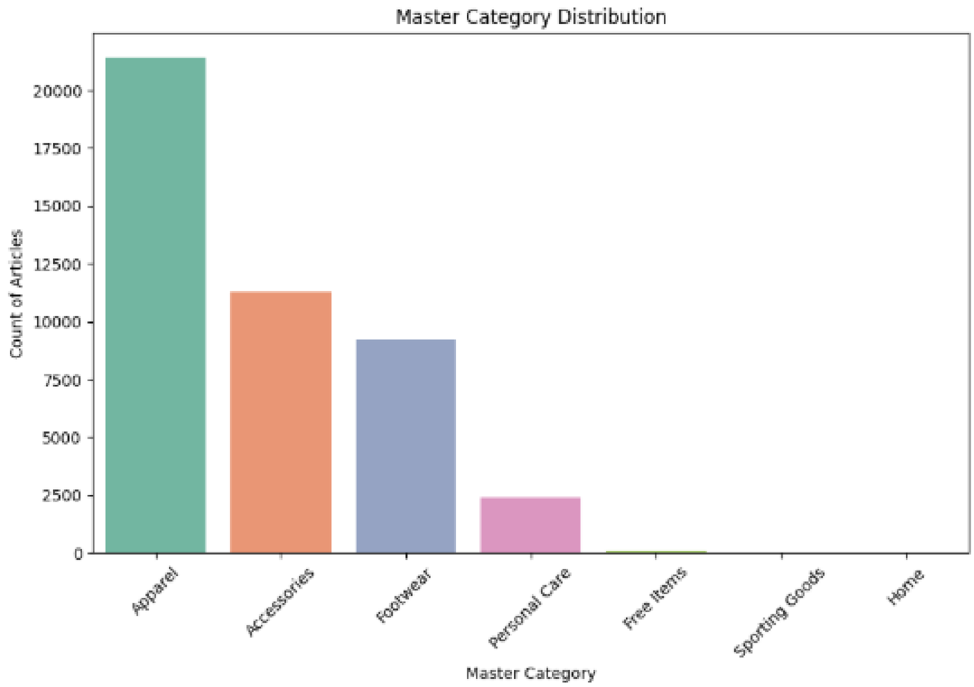


Image Verification and Display

Images corresponding to product IDs were verified for existence. Additionally, sample images were visualized with metadata.

```
[ ]  from PIL import Image
     import matplotlib.pyplot as plt

     # Load and display a sample image with metadata
     sample_row = df[df['exists']].iloc[0]  # Get a valid row
     image_path = sample_row['imagePath']

     # Display image
     image = Image.open(image_path)
     plt.imshow(image)
     plt.axis('off')
     plt.title(f"Article Type: {sample_row['articleType']}\nGender: {sample_row['gender']}")
     plt.show()
```

Image Analysis and Collaborative Filtering using VGG16

In this section, we focus on the image analysis pipeline and the use of a VGG16 model to generate product recommendations based on image similarity.

Image Path Verification

The first step involves verifying the existence of images in the dataset. We create an imagePath column in the DataFrame by combining the product IDs with the .jpg extension. We then check whether the image exists at the given path using the os.path.exists() function. The following output shows that 44,419 images were found, and only 5 images were missing.

```python
import os

# Verify image paths
df['imagePath'] = df['id'].apply(lambda x: f"extracted_dataset/images/{x}.jpg")
df['exists'] = df['imagePath'].apply(os.path.exists)

# Count missing images
missing_count = df['exists'].value_counts()
print(f"Images found: {missing_count[True]}, Missing: {missing_count[False]}")
```

```
Images found: 44419, Missing: 5
```

Image Visualization

To inspect the dataset visually, we read and display the first five images from the dataset. The images are displayed using OpenCV and Matplotlib.

```python
import cv2
import matplotlib.pyplot as plt
import os

# Path to images
image_folder = 'extracted_dataset/images'

# List some images
image_files = os.listdir(image_folder)[:5]  # Display the first 5 images

# Display images
for image_file in image_files:
    img_path = os.path.join(image_folder, image_file)
    img = cv2.imread(img_path)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(image_file)
    plt.show()
```

Average Color Calculation

Next, we calculate the average color of each image. The image is read, and the mean pixel values across all channels (Red, Green, Blue) are computed. This allows us to observe the overall color distribution of the images in the dataset.

```
import numpy as np

# Calculate average color
colors = ['Blue', 'Green', 'Red']
average_colors = []

for image_file in image_files:
    img_path = os.path.join(image_folder, image_file)
    img = cv2.imread(img_path)
    avg_color_per_channel = np.mean(img, axis=(0, 1))  # Mean across rows and columns
    average_colors.append(avg_color_per_channel)

# Plot average colors
average_colors = np.array(average_colors)
for i, color in enumerate(colors):
    plt.plot(average_colors[:, i], label=color)

plt.title("Average Color Distribution")
plt.legend()
plt.show()
```

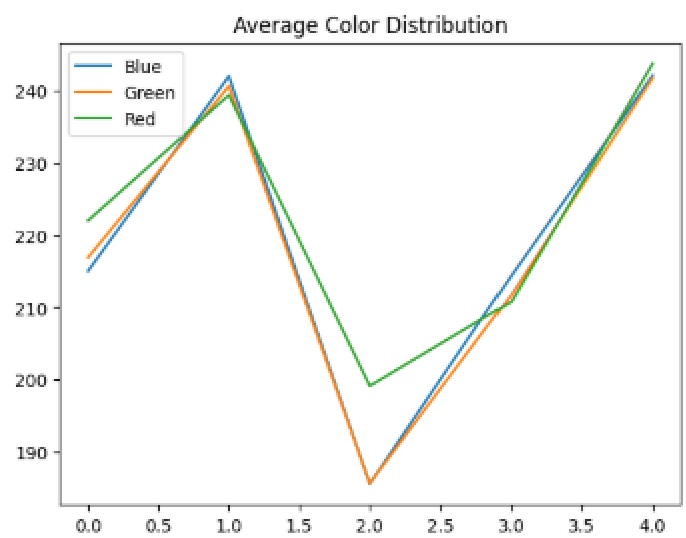
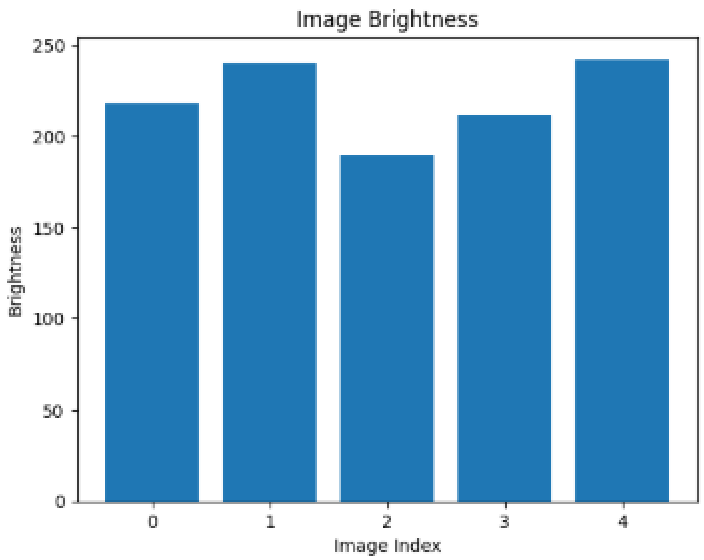


Image Brightness Calculation

The brightness of each image is calculated as the mean pixel value in grayscale. A bar chart is plotted to visualize the brightness of the images.

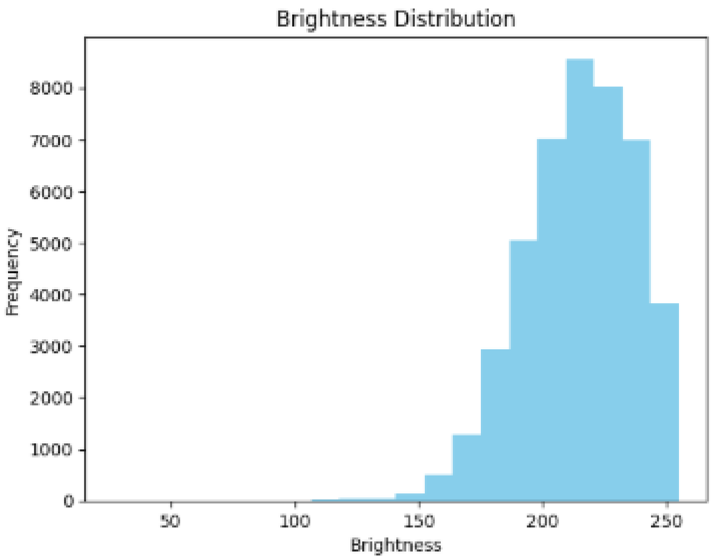
```
[ ] # Calculate brightness as the mean pixel value
    brightness_values = []
    for image_file in image_files:
        img_path = os.path.join(image_folder, image_file)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        brightness = np.mean(img)
        brightness_values.append(brightness)

    # Plot brightness values
    plt.bar(range(len(brightness_values)), brightness_values)
    plt.title("Image Brightness")
    plt.xlabel("Image Index")
    plt.ylabel("Brightness")
    plt.show()
```



Full Dataset Brightness Histogram

To analyze the full dataset, we calculate the brightness for all images and plot a histogram of the brightness values. This provides an overview of the overall brightness distribution in the dataset.

```
[ ] # Full dataset analysis
    brightness_values_full = []
    for image_file in os.listdir(image_folder):
        img_path = os.path.join(image_folder, image_file)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        brightness = np.mean(img)
        brightness_values_full.append(brightness)

    # Plot histogram of brightness
    plt.hist(brightness_values_full, bins=20, color='skyblue')
    plt.title("Brightness Distribution")
    plt.xlabel("Brightness")
    plt.ylabel("Frequency")
    plt.show()
```



Collaborative Filtering Using VGG16 for Visual Recommendation

In this section of the report, we explore the process of collaborative filtering using the VGG16 model to generate visual product recommendations. Collaborative filtering is a widely used recommendation technique, where similar items are recommended based on the preferences of other users or item similarities. In this case, we focus on content-based filtering, leveraging visual features extracted from images to recommend similar products.

1. VGG16 Model for Feature Extraction

The VGG16 model, pre-trained on ImageNet, is used to extract feature embeddings from product images. The model is designed to recognize visual features such as textures, shapes, and patterns from images. We modify the VGG16 architecture by removing the top classification layer and using the output of the penultimate layer (i.e., the fully connected layer) as the feature embeddings.

```
import numpy as np
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
```

```
# Load pre-trained VGG16 model without the top classification layer
vgg_model = VGG16(weights='imagenet', include_top=False, pooling='avg')
```

In this code, the VGG16 model is initialized with pre-trained weights from ImageNet, excluding the top classification layer (include_top=False). The pooling='avg' option applies average pooling on the output feature map to generate a fixed-size embedding vector for each image.

2. Image Preprocessing

VGG16 requires images to be resized to 224x224 pixels and preprocessed before being passed through the model. The preprocessing includes scaling pixel values to a range that the model was originally trained on.

```
# Set dimensions for VGG model input
img_width, img_height = 224, 224  # VGG16 expects images of size 224x224
```

Each image is resized and then preprocessed using the preprocess_input function to ensure that the image is compatible with VGG16's input requirements.

```
# Function to predict and extract embeddings using the model
def model_predict(model, img_name):
    try:
        # Load and preprocess the image
        img = image.load_img(img_path(img_name), target_size=(img_width, img_height))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)  # Add batch dimension
        x = preprocess_input(x)        # Preprocess for VGG
        # Generate embeddings
        return model.predict(x).reshape(-1)  # Flatten the embeddings
    except Exception as e:
        print(f"Error processing {img_name}: {e}")
        return None
```

The function model_predict loads an image, processes it, and extracts the embeddings using the VGG16 model.

3. Embedding Extraction

Once the embeddings are extracted for each image, they are stored in a DataFrame. Each row contains an image's file path and the corresponding embedding, which is a high-dimensional vector representing the image's visual features.

```
# Apply the updated function
df_subset['embedding'] = df_subset['imagePath'].apply(lambda img_name: model_predict(vgg_model, img_name))

# Filter out invalid rows (with None embeddings)
```

After embeddings are extracted, the embeddings for all images are saved to a file for later use, and invalid rows (where the embedding extraction failed) are removed.

```
# Filter out invalid rows (with None embeddings)
df_subset = df_subset[df_subset['embedding'].notnull()]

# Save results to a file
df_subset.to_pickle('embeddings_5000.pkl')
```

4. Cosine Similarity for Recommendations

The core idea behind collaborative filtering in this context is to recommend products based on their visual similarity. Cosine similarity is used to measure the similarity between the embeddings of two images. Higher cosine similarity means that the images are visually similar.

```
def generate_recommendations(product_id, num_recommendations, df):
    """
    Generate recommendations based on cosine similarity.

    Args:
    - product_id: ID of the product to generate recommendations for.
    - num_recommendations: Number of similar products to recommend.
    - df: DataFrame containing product embeddings.

    Returns:
    - DataFrame with recommended product IDs and similarity scores.
    """
    # Get the embedding for the given product ID
    query_embedding = df.loc[df['id'] == product_id, 'embedding'].values[0]

    # Compute cosine similarity between query embedding and all other embeddings
    df['similarity'] = df['embedding'].apply(lambda x: cosine_similarity([query_embedding], [x])[0][0])

    # Sort by similarity (highest first) and exclude the query product
    recommendations = df[df['id'] != product_id].sort_values(by='similarity', ascending=False)

    # Return top N recommendations
    return recommendations[['id', 'imagePath', 'similarity']].head(num_recommendations)
```

The function generate_recommendations takes a product ID and calculates the cosine similarity between the query product's embedding and all other products in the dataset. It then sorts the products by similarity and returns the top N most similar products.

5. Displaying Recommendations

Once recommendations are generated, the results are displayed visually to allow easy inspection of the most similar products. The query image (the product for which recommendations are generated) is shown alongside the recommended images, with their corresponding similarity scores.

```
[ ] import matplotlib.pyplot as plt
    from matplotlib.image import imread
    import os

    def display_recommendations(product_id, num_recommendations, df):
        # Generate recommendations
        recommendations = generate_recommendations(product_id, num_recommendations, df)

        # Get query image path
        query_image_path = df.loc[df['id'] == product_id, 'imagePath'].values[0]

        # Plot query image
        plt.figure(figsize=(15, 5))
        plt.subplot(1, num_recommendations + 1, 1)
        if os.path.exists(query_image_path):
            plt.imshow(imread(query_image_path))
            plt.axis('off')
            plt.title("Query Product")
        else:
            print(f"Query image not found: {query_image_path}")
            return

        # Plot recommended images
        for i, row in enumerate(recommendations.itertuples(), start=2):
            recommended_image_path = row.imagePath
            plt.subplot(1, num_recommendations + 1, i)
            if os.path.exists(recommended_image_path):
                plt.imshow(imread(recommended_image_path))
                plt.axis('off')
                plt.title(f"Sim: {row.similarity:.2f}")
            else:
                print(f"Recommended image not found: {recommended_image_path}")
                plt.text(0.5, 0.5, "Image Not Found", ha='center', va='center')

        plt.show()
```

This function first displays the query product's image, followed by images of the recommended products with their similarity scores.

OUTPUT:

```
[ ] product_id = 15970  # Replace with a valid product ID from your dataset
    num_recommendations = 5

    display_recommendations(product_id, num_recommendations, df_subset)
```


```
[ ] import numpy as np
```

Collaborative Filtering with MobileNet for Product Recommendations

The provided code outlines an approach for generating product recommendations based on image embeddings using MobileNet, a lightweight neural network model pre-trained on ImageNet. The methodology is based on calculating cosine similarity between embeddings of different products to recommend similar items. Below is an explanation of the key steps involved:

1)Loading the MobileNet Model

```
# Step 1: Load the MobileNet model
print("Loading MobileNet model...")
mobilenet_model = MobileNet(weights='imagenet', include_top=False, pooling='avg')
```

In this step, the MobileNet model is loaded. The model is pre-trained on ImageNet, which means it has already learned to identify various features in images. By excluding the top classification layers (include_top=False), we use the model as a feature extractor, generating embeddings of images instead of classifying them.

2)Preparing Image Data

```
# Step 2: Load dataset and generate image paths
# Assuming 'df' already contains the column 'id'
# Create 'imagePath' column if not already present
df['imagePath'] = 'extracted_dataset/images/' + df['id'].astype(str) + '.jpg'
```

The dataset (df) contains product information, including product IDs. Here, we construct the file path for the images corresponding to each product, assuming that the images are stored in the folder 'extracted_dataset/images/'.

3. Generating Embeddings

```
print("Generating embeddings for 1000 images...")
df_subset['mobilenet_embedding'] = df_subset['imagePath'].apply(
    lambda img_path: mobilenet_predict(mobilenet_model, img_path)
)
```

This step involves extracting image embeddings using MobileNet. Each product's image is processed to generate a fixed-length vector (embedding) that represents the image in the feature space. The mobilenet_predict function loads and preprocesses each image, passing it through the MobileNet model to extract the features.

4.Handling Missing Embeddings

```
df_subset = df_subset[df_subset['mobilenet_embedding'].notnull()]
```

Finally, an example is shown where a product with ID 39386 is used as a query to generate and display the top 5 recommendations. The display_recommendations function will visually output the query product and its most similar products based on cosine similarity.

5.Saving Embeddings

```
output_file = 'embeddings_mobilenet.pkl'
df_subset.to_pickle(output_file)
```

After generating embeddings for the product images, they are saved to a pickle file ('embeddings_mobilenet.pkl') for future use. This step helps avoid recalculating embeddings each time.

6.Recommendation Function: Cosine Similarity

```
# Step 2: Recommendation function
def generate_recommendations(product_id, num_recommendations, df):
    """
    Generate recommendations based on cosine similarity.

    Args:
        product_id (int): ID of the product to generate recommendations for.
        num_recommendations (int): Number of recommendations.
        df (pd.DataFrame): DataFrame with MobileNet embeddings.

    Returns:
        pd.DataFrame: Top N recommended products.
    """
    # Get the embedding for the query product
    query_embedding = df.loc[df['id'] == product_id, 'mobilenet_embedding'].values[0]

    # Compute cosine similarity between query embedding and all other embeddings
    df['similarity'] = df['mobilenet_embedding'].apply(lambda x: cosine_similarity([query_embedding], [x])[0][0])

    # Sort by similarity (exclude the query product itself)
    recommendations = df[df['id'] != product_id].sort_values(by='similarity', ascending=False)

    # Return top N recommendations
    return recommendations[['id', 'imagePath', 'similarity']].head(num_recommendations)
```

The generate_recommendations function calculates the cosine similarity between the embedding of the query product and all other products in the dataset. Cosine similarity measures the cosine of the angle between two vectors, ranging from -1 (completely dissimilar) to 1 (identical). Based on the similarity scores, it returns the top N products that are most similar to the query product.

7.Displaying the Recommendations

```python
def display_recommendations(product_id, num_recommendations, df):
    """
    Display the query product and its recommended products using matplotlib.

    Args:
        product_id (int): ID of the query product.
        num_recommendations (int): Number of recommendations.
        df (pd.DataFrame): DataFrame with embeddings and image paths.
    """
    # Generate recommendations
    recommendations = generate_recommendations(product_id, num_recommendations, df)

    # Query image path
    query_image_path = df.loc[df['id'] == product_id, 'imagePath'].values[0]

    # Plot the query image
    plt.figure(figsize=(15, 5))
    plt.subplot(1, num_recommendations + 1, 1)
    if os.path.exists(query_image_path):
        plt.imshow(imread(query_image_path))
        plt.axis('off')
        plt.title("Query Product")
    else:
        print(f"Query image not found: {query_image_path}")
        return
```

The display_recommendations function uses matplotlib to visually present the recommended products. It displays the query product alongside the top recommendations, showing images with their similarity scores. This provides a clear, intuitive visualization of the recommendations.

8. Example Usage

```python
# Step 4: Example usage
product_id = 39386  # Replace with a valid product ID
num_recommendations = 5

print(f"Generating recommendations for product ID: {product_id}")
display_recommendations(product_id, num_recommendations, df)
```

```
Loading embeddings from 'embeddings_mobilenet.pkl'...
Generating recommendations for product ID: 39386
```

Finally, an example is shown where a product with ID 39386 is used as a query to generate and display the top 5 recommendations. The display_recommendations function will visually output the query product and its most similar products based on cosine similarity.

References

- **Books and Research Papers**

- **Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013).** ***Efficient Estimation of Word Representations in Vector Space.*** **https://arxiv.org/abs/1301.3781**
  - **Describes the foundational concepts of Word2Vec.**
- **Salton, G., Wong, A., & Yang, C. S. (1975).** ***A Vector Space Model for Automatic Indexing.*** **Communications of the ACM, 18(11), 613–620.**
  - **Introduces the vector space model and TF-IDF concepts.**

- **Python Libraries and Documentation**

- **Scikit-learn Documentation:**
  - **https://scikit-learn.org/stable/**
  - **Comprehensive guide for using TF-IDF and cosine similarity.**
- **Gensim Library:**
  - **https://radimrehurek.com/gensim/**
  - **Official documentation for Word2Vec and related NLP models.**
- **Pandas Documentation:**
  - **https://pandas.pydata.org/**
  - **For efficient data manipulation and analysis.**
- **NumPy Documentation:**
  - **https://numpy.org/**
  - **For numerical operations and array manipulations.**

Tools and External Resources

1. **Requests Library Documentation:**
   1. **https://docs.python-requests.org/**
   2. **Used for validating image URLs.**
2. **Amazon ASIN Details:**
   1. **https://www.amazon.com/**
   2. **Provides product data reference for ASIN values.**
3. **Chart Styling Inspiration:**
   1. **HTML/CSS reference from W3Schools: https://www.w3schools.com/**

Datasets

1. ***Fashion Products Dataset*:**

    1. **Your local dataset fashion_products_data.ldjson, cleaned and preprocessed for recommendation tasks.**
2. **Kaggle: Fashion Datasets**
    1. **https://www.kaggle.com/**
    2. **Useful for benchmarking results or exploring similar datasets.**

Tutorials and Articles

3. ***Introduction to Recommender Systems*: Towards Data Science.**
    1. **https://towardsdatascience.com/**
4. ***Hybrid Recommendation Systems*: Analytics Vidhya.**
    1. **https://www.analyticsvidhya.com/**