# Dataset table

This dataset is having the data of 1K+ Amazon Product's Ratings and Reviews as per their details listed on the official website of Amazon

| Field Name | Description |
|---|---|
| product_id | Product ID |
| product_name | Name of the Product |
| category | Category of the Product |
| discounted_price | Discounted Price of the Product |
| actual_price | Actual Price of the Product |
| discount_percentage | Percentage of Discount for the Product |
| rating | Rating of the Product |
| rating_count | Number of people who voted for the Amazon rating |
| about_product | Description about the Product |
| user_id | ID of the user who wrote review for the Product |
| user_name | Name of the user who wrote review for the Product |
| review_id | ID of the user review |
| review_title | Short review |
| review_content | Long review |
| img_link | Image Link of the Product |
| product_link | Official Website Link of the Product |

In [148]:

```
import pandas as pd

df = pd.read_csv("/kaggle/input/amazon-sales-dataset/amazon.csv")

df.head()
```

Out[148]:

| | product_id | product_name | category | discounted_price | actual_price | discount_percentag |
|---|---|---|---|---|---|---|
| 0 | B07JW9H4J1 | Wayona Nylon Braided USB to Lightning Fast Cha... | Computers&Accessories\|Accessories&Peripherals\|... | ₹399 | ₹1,099 | 64% |
| 1 | B098NS6PVG | Ambrane Unbreakable 60W / 3A Fast Charging 1.5... | Computers&Accessories\|Accessories&Peripherals\|... | ₹199 | ₹349 | 43% |
| 2 | B096MSW6CT | Sounce Fast Phone Charging Cable & Data Sync U... | Computers&Accessories\|Accessories&Peripherals\|... | ₹199 | ₹1,899 | 90% |
| 3 | B08HDJ86NZ | boAt Deuce USB 300 2 in 1 Type-C & Micro USB S... | Computers&Accessories\|Accessories&Peripherals\|... | ₹329 | ₹699 | 53% |
| 4 | B08CF3B7N1 | Portronics Konnect L 1.2M Fast Charging 3A 8 P... | Computers&Accessories\|Accessories&Peripherals\|... | ₹154 | ₹399 | 61% |

In [149]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1465 entries, 0 to 1464
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   product_id           1465 non-null   object
 1   product_name         1465 non-null   object
 2   category             1465 non-null   object
 3   discounted_price     1465 non-null   object
 4   actual_price         1465 non-null   object
 5   discount_percentage  1465 non-null   object
 6   rating               1465 non-null   object
 7   rating_count         1463 non-null   object
 8   about_product        1465 non-null   object
 9   user_id              1465 non-null   object
 10  user_name            1465 non-null   object
 11  review_id            1465 non-null   object
 12  review_title         1465 non-null   object
 13  review_content       1465 non-null   object
 14  img_link             1465 non-null   object
 15  product_link         1465 non-null   object
dtypes: object(16)
memory usage: 183.2+ KB
```

## Check for nan values

In [150]:

```
df.isna().sum()
```

Out[150]:

```
product_id             0
product_name           0
category               0
discounted_price       0
actual_price           0
discount_percentage    0
rating                 0
rating_count           2
about_product          0
user_id                0
user_name              0
review_id              0
review_title           0
review_content         0
img_link               0
product_link           0
dtype: int64
```

In [151]:

```
df.dropna(inplace=True)
df = df[df.rating != '|'] # Remove elements with | value in the rating column
```

## Remove duplicates

In [152]:

```
print(reviews.shape)
df = df.drop_duplicates()
print(reviews.shape)
```

```
(1462,)
(1462,)
```

## Preprocessing

### 1. Convert feature into appropriate formats

Looking through the dataset, there is a number of characteristics are not in the proper format (now formatted as 'object'), including discounted_price, actual_price, discount_percentage, rating, and rating_count. In order to enable a more fruitful examination, we have to to translate these into integer numbers.

In [153]:

```
# Convert 'discounted_price' and 'actual_price' by removing currency symbol and converting to float
df['discounted_price'] = df['discounted_price'].astype(str).str.replace('₹', '').str.replace(',', '').astype(float)
df['actual_price'] = df['actual_price'].astype(str).str.replace('₹', '').str.replace(',', '').astype(float)

# Convert 'discount_percentage' by removing '%' and converting to float
df['discount_percentage'] = df['discount_percentage'].astype(str).str.replace('%', '').astype(float)

# Convert 'rating' to float
df['rating'] = pd.to_numeric(df['rating'].astype(str).str.replace('|', ''), errors='coerce')

# Convert 'rating_count' by removing commas and converting to int
df['rating_count'] = df['rating_count'].astype(str).str.replace(',', '').astype(int)
```

In [154]:

```
df.head()
```

Out[154]:

| | product_id | product_name | category | discounted_price | actual_price | discount_percentag |
|---|---|---|---|---|---|---|
| 0 | B07JW9H4J1 | Wayona Nylon Braided USB to Lightning Fast Cha... | Computers&Accessories\|Accessories&Peripherals\|... | 399.0 | 1099.0 | 64. |
| 1 | B098NS6PVG | Ambrane Unbreakable 60W / 3A Fast Charging 1.5... | Computers&Accessories\|Accessories&Peripherals\|... | 199.0 | 349.0 | 43. |
| 2 | B096MSW6CT | Sounce Fast Phone Charging Cable & Data Sync U... | Computers&Accessories\|Accessories&Peripherals\|... | 199.0 | 1899.0 | 90. |
| 3 | B08HDJ86NZ | boAt Deuce USB 300 2 in 1 Type-C & Micro USB S... | Computers&Accessories\|Accessories&Peripherals\|... | 329.0 | 699.0 | 53. |
| 4 | B08CF3B7N1 | Portronics Konnect L 1.2M Fast Charging 3A 8 P... | Computers&Accessories\|Accessories&Peripherals\|... | 154.0 | 399.0 | 61. |

### 2. Clean and preprocess the text.

- Lowercase.
- Removing punctuation and special characters.
- Eliminating stopwords.
- Lemmatization (reducing words to their base).

In [155]:

```
import re

import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

# Cleaning and preprocessing text without lemmatization
def clean_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation and special characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    # Split text into words and rejoin without stopwords
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

# Assuming df is your DataFrame and it has been previously loaded
# Apply the clean_text function to the DataFrame columns
df['product_name'] = df['product_name'].apply(clean_text)
df['about_product'] = df['about_product'].apply(clean_text)
df['review_content'] = df['review_content'].apply(clean_text)
df['category'] = df['category'].apply(clean_text)
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

As you can see, our dataset's "category" columns are arranged the same ways as categories are on Amazon. For example, going through Computers & Accessories -> Accessories -> Cables & Accessories -> Cables -> USB Cables is one way to buy a cable. We intend to reduce the number of 'category' columns and concentrate on the broadest category level in order to streamline our data.

In [156]:

```
#Extracting the top-level category
df['category'] = df['category'].apply(lambda x: x.split('|')[0] if pd.notnull(x) else x)
```

In [157]:

```
df.head()
```

Out[157]:

| | product_id | product_name | category | discounted_price | actual_price | discount_percentage |
|---|---|---|---|---|---|---|
| 0 | B07JW9H4J1 | wayona nylon braided usb lightning fast chargi... | computersaccessoriesaccessoriesperipheralscabl... | 399.0 | 1099.0 | 64.0 |
| 1 | B098NS6PVG | ambrane unbreakable 60w 3a fast charging 15m b... | computersaccessoriesaccessoriesperipheralscabl... | 199.0 | 349.0 | 43.0 |
| 2 | B096MSW6CT | sounce fast phone charging cable data sync usb... | computersaccessoriesaccessoriesperipheralscabl... | 199.0 | 1899.0 | 90.0 |
| 3 | B08HDJ86NZ | boat deuce usb 300 2 1 typec micro usb stress ... | computersaccessoriesaccessoriesperipheralscabl... | 329.0 | 699.0 | 53.0 |
| 4 | B08CF3B7N1 | portronics konnect l 12m fast charging 3a 8 pi... | computersaccessoriesaccessoriesperipheralscabl... | 154.0 | 399.0 | 61.0 |

## Let's perform some data analysis.

### Exploratory Data Analysis

#### 1. Marketplace Product Performance and Pricing Analysis

In [158]:

```
unique_products_count = df['product_id'].nunique()
average_price = df['actual_price'].mean()
best_selling_product = df.loc[df['rating_count'].idxmax()]
least_selling_product = df.loc[df['rating_count'].idxmin()]
top_rated_product = df.loc[df['rating'].idxmax()]
lowest_rated_product = df.loc[df['rating'].idxmin()]
most_expensive_product = df.loc[df['actual_price'].idxmax()]
cheapest_product = df.loc[df['actual_price'].idxmin()]
highest_discount_product = df.loc[df['discount_percentage'].idxmax()]
avg_rating_count = df.groupby('product_id')['rating_count'].mean().mean()

df_anl = pd.DataFrame({
    'Question': [
        'Number of Unique Products',
        'Average Price',
        'Best-selling Product',
        'Least-selling Product',
        'Top-rated Product',
        'Lowest-rated Product',
        'Most Expensive Product',
        'Cheapest Product',
        'Highest Discount Product',
        'Average Rating Count for Each Product'
    ],
    'Answer': [
        unique_products_count,
        average_price,
        best_selling_product['product_name'],
        least_selling_product['product_name'],
        top_rated_product['product_name'],
        lowest_rated_product['product_name'],
        most_expensive_product['product_name'],
        cheapest_product['product_name'],
        highest_discount_product['product_name'],
        avg_rating_count
    ],
    'Actual Price': [
        None,
        None,
        best_selling_product['actual_price'],
        least_selling_product['actual_price'],
        top_rated_product['actual_price'],
        lowest_rated_product['actual_price'],
        most_expensive_product['actual_price'],
        cheapest_product['actual_price'],
        highest_discount_product['actual_price'],
        None
    ]
})

df_anl
```

Out[158]:

| | Question | Answer | Actual Price |
|---|---|---|---|
| 0 | Number of Unique Products | 1348 | NaN |
| 1 | Average Price | 5453.087743 | NaN |
| 2 | Best-selling Product | amazonbasics flexible premium hdmi cable black... | 700.0 |
| 3 | Least-selling Product | khaitan orfin fan heater home kitchenk0 2215 | 2495.0 |
| 4 | Top-rated Product | syncwire ltg usb cable fast charging compatibl... | 1999.0 |
| 5 | Lowest-rated Product | khaitan orfin fan heater home kitchenk0 2215 | 2495.0 |
| 6 | Most Expensive Product | sony bravia 164 cm 65 inches 4k ultra hd smart... | 139900.0 |
| 7 | Cheapest Product | ecosmos 5v 12w portable flexible usb led light... | 39.0 |
| 8 | Highest Discount Product | rts 2 pack mini usb c type c adapter plug type... | 4999.0 |
| 9 | Average Rating Count for Each Product | 17656.855341 | NaN |

#### 2. Find most popular items

In [159]:

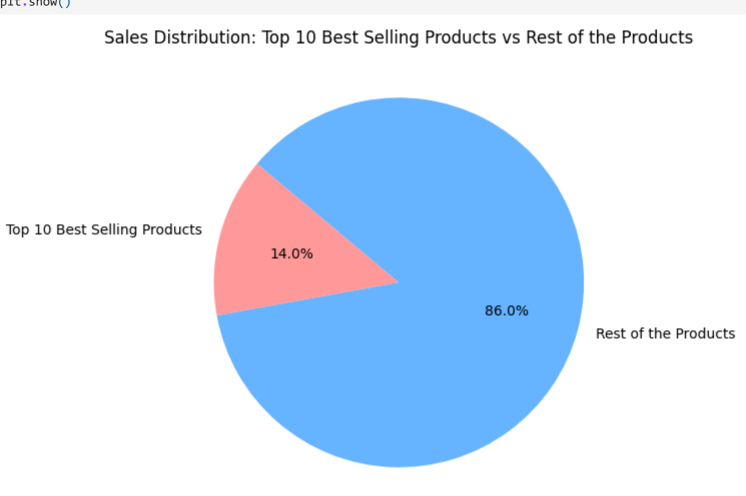
```
# Sorting the data by rating_count in descending order
top_selling_products = df.sort_values(by='rating_count', ascending=False).head(10)

# Selecting relevant columns for display
top_selling_products = top_selling_products[['product_name', 'rating', 'rating_count']]
top_selling_products.reset_index(drop=True, inplace=True)
top_selling_products
```

Out[159]:

| | product_name | rating | rating_count |
|---|---|---|---|
| 0 | amazonbasics flexible premium hdmi cable black... | 4.4 | 426973 |
| 1 | amazon basics highspeed hdmi cable 6 feet 2pac... | 4.4 | 426973 |
| 2 | amazon basics highspeed hdmi cable 6 feet supp... | 4.4 | 426973 |
| 3 | amazonbasics flexible premium hdmi cable black... | 4.4 | 426972 |
| 4 | boat bassheads 100 ear wired earphones micfuri... | 4.1 | 363713 |
| 5 | boat bassheads 100 ear wired earphones mictaff... | 4.1 | 363713 |
| 6 | boat bassheads 100 inear wired headphones mic ... | 4.1 | 363711 |
| 7 | redmi 9 activ carbon black 4gb ram 64gb storag... | 4.1 | 313836 |
| 8 | redmi 9a sport coral green 2gb ram 32gb storag... | 4.1 | 313836 |
| 9 | redmi 9a sport carbon black 2gb ram 32gb stora... | 4.1 | 313832 |

In [160]:

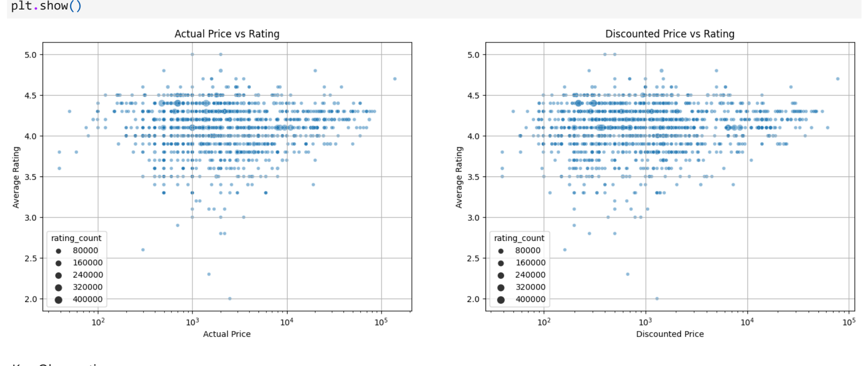
```
import matplotlib.pyplot as plt

top_10_total_ratings = top_selling_products['rating_count'].sum()
total_ratings_all_products = df['rating_count'].sum()
ratings_rest_of_products = total_ratings_all_products - top_10_total_ratings
pie_data = [top_10_total_ratings, ratings_rest_of_products]
labels = ['Top 10 Best Selling Products', 'Rest of the Products']

# Creating the pie chart
plt.figure(figsize=(10, 6))
plt.pie(pie_data, labels=labels, autopct='%1.1f%%', startangle=140, colors=['#ff9999','#66b3ff'])
plt.title('Sales Distribution: Top 10 Best Selling Products vs Rest of the Products')
plt.show()
```




#### 3. How does pricing (both actual and discounted) affects product ratings and popularity?

In [161]:

```
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(18, 6))

#Scatter plot for actual price vs rating
sns.scatterplot(ax=axes[0], data=df, x='actual_price', y='rating', size='rating_count', alpha=0.5)
axes[0].set_title('Actual Price vs Rating')
axes[0].set_xlabel('Actual Price')
axes[0].set_ylabel('Average Rating')
axes[0].set_xscale('log')
axes[0].grid(True)

#Scatter plot for discounted price vs rating
sns.scatterplot(ax=axes[1], data=df, x='discounted_price', y='rating', size='rating_count', alpha=0.5)
axes[1].set_title('Discounted Price vs Rating')
axes[1].set_xlabel('Discounted Price')
axes[1].set_ylabel('Average Rating')
axes[1].set_xscale('log')
axes[1].grid(True)

plt.show()
```




Key Observations:

- Price and Rating Correlation: Ratings exhibit a diverse range across various price categories. Products with higher prices generally garner a moderate to high number of ratings, implying consumer engagement across the entire spectrum of prices.
- Price Discounts and Rating Distribution: Similar to the actual price, discounted prices display a widespread distribution of ratings. This suggests that discounted prices do not exert a significant influence on average ratings, indicating that consumers prioritize rating based on product quality and satisfaction rather than mere price considerations.

In the context of a recommendation system, these insights imply that popularity is not exclusive to either high or low-priced products, and the presence of discounts does not necessarily result in higher ratings.

#### 4. Top categories based on the number of products

In [162]:

```
from collections import Counter

categories = df['category'].str.split('|').explode()
category_counts = Counter(categories)
category_df = pd.DataFrame(category_counts.items(), columns=['Category', 'Count']).sort_values(by='Count', ascending=Fa

#Display the top categories
top_categories = category_df.head(10)
top_categories.reset_index(drop=True, inplace=True)
top_categories
```

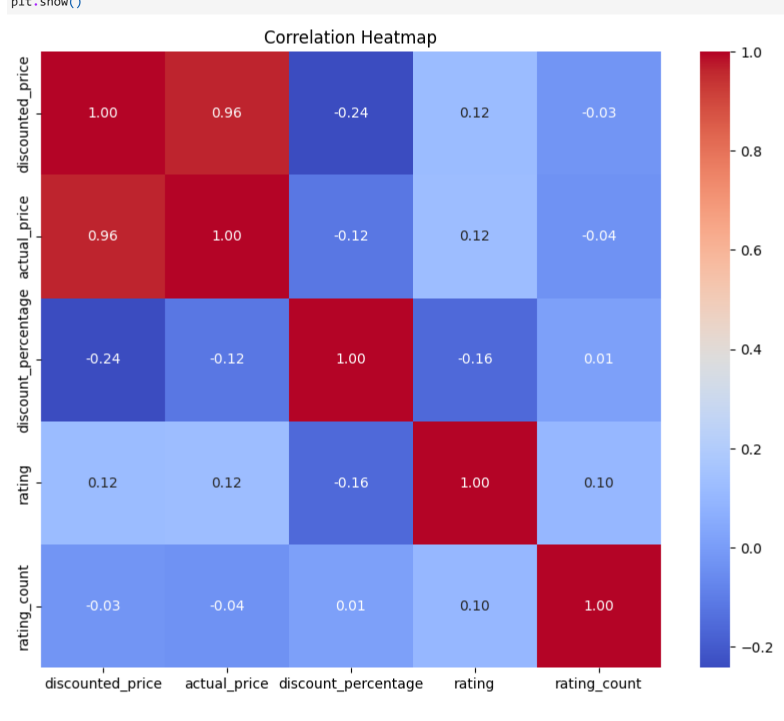
Out[162]:

| | Category | Count |
|---|---|---|
| 0 | computersaccessoriesaccessoriesperipheralscabl... | 231 |
| 1 | electronicswearabletechnologysmartwatches | 76 |
| 2 | electronicsmobilesaccessoriessmartphonesbasicm... | 68 |
| 3 | electronicshometheatertvvideotelevisionssmartt... | 63 |
| 4 | electronicsheadphonesearbudsaccessoriesheadpho... | 52 |
| 5 | electronicshometheatertvvideoaccessoriesremote... | 49 |
| 6 | homekitchenkitchenhomeappliancessmallkitchenap... | 27 |
| 7 | electronicshometheatertvvideoaccessoriescables... | 24 |
| 8 | homekitchenkitchenhomeappliancesvacuumcleaning... | 24 |
| 9 | computersaccessoriesaccessoriesperipheralskeyb... | 24 |

#### 5. Correlation heat map for the numerical columns.

In [163]:

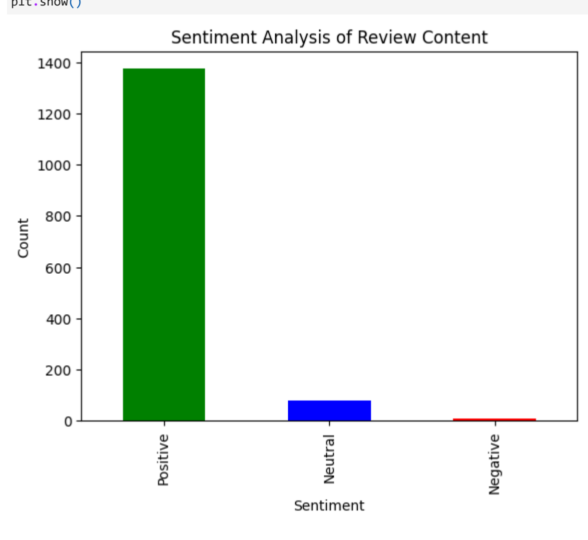
```
# Select only non-'object' (numerical) columns
numerical_df = df.select_dtypes(exclude=['object'])

# Calculate the correlation matrix
corr_matrix = numerical_df.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```




#### 6. Sentiment Analysis

In [164]:

```
from textblob import TextBlob

# Sentiment analysis
def sentiment_analysis(text):
    analysis = TextBlob(text)
    # Get the polarity score
    score = analysis.sentiment.polarity
    # Threshold for positive and negative sentiments
    if analysis.sentiment.polarity > 0.1:
        sentiment = 'Positive'
    elif analysis.sentiment.polarity < -0.1:
        sentiment = 'Negative'
    else:
        sentiment = 'Neutral'

    return sentiment, score


# Applying sentiment analysis to the review content
reviews = df['review_content']
# Creating two lists for sentiments and scores
reviews_sentiments = []
reviews_scores = []

# Applying the modified function to each review
for review in reviews:
    sentiment, score = sentiment_analysis(review)
    reviews_sentiments.append(sentiment)
    reviews_scores.append(score)

# Converting lists to Pandas Series
df['reviews_sentiments'] = pd.Series(reviews_sentiments)
df['reviews_scores'] = pd.Series(reviews_scores)

# Counting the occurrences of each sentiment
sentiment_counts = df['reviews_sentiments'].value_counts()

sentiment_counts.plot(kind='bar', color=['green', 'blue', 'red'], title='Sentiment Analysis of Review Content')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```




## Feature Engineering

### Drop irelevant columns

In [165]:

```
drop_col = ['discounted_price', 'actual_price', 'discount_percentage', 'review_id', 'review_title',
            'user_name', 'img_link', 'product_link']
df = df.drop(columns=drop_col)
```

### Implementing TF-IDF (Term Frequency and Inverse Document Frequency)

The primary objective is to assess the significance of a word within a document. Typically, this involves:

- Term Frequency (TF): gauges how often a term appears in a document. It is computed by dividing the number of occurrences of a term in a document by the total number of terms in that document.
- Inverse Document Frequency (IDF): evaluates the importance of the term across the corpus. It is calculated by taking the logarithm of the total number of documents divided by the number of documents containing the term.

To effectively apply TF-IDF to this dataset, let's consider consolidating text from product_name, category, and about_product to create a comprehensive representation for each product.

In [166]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

df['combined_text'] = df['product_name'] + ' ' + df['category'] + ' ' + df['about_product'] +' '+ df['review_content']
# Fill null with empty string to avoid issues
df['combined_text'] = df['combined_text'].fillna('')

# Instantiate TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_df=0.95, min_df=2, ngram_range=(1, 1))

# Fit and transform
tfidf_matrix = vectorizer.fit_transform(df['combined_text'])
```

### Label-Encoding the User, Product and Sentiment columns

In [167]:

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

# Fitting the encoder and transforming the 'Sentiment' column
df['encoded_sentiment'] = label_encoder.fit_transform(df['reviews_sentiments'])
```

# Recommendation systems

## Preprocessing

Before continuing with the experiments, we need to create the user-item rating matrix

In [168]:

```
# Create a product-user matrix with overall product ratings
product_user_matrix = df.pivot_table(index='user_id', columns='product_id', values='rating')

# Fill missing values with the average rating
product_user_matrix = product_user_matrix.fillna(0)

# Display the product-user matrix
product_user_matrix.head()
```

Out[168]:

| |
|---|
| AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDV |
| AE23RS3W7GZO7LHYKJU6KSKVM4MQ,AEQUNEY6GQOTEGUMS6KRUEYNXJSQ,AGYPIE5BICV44WEEEPJVEFQOCJSQ,AFR7CEQKWZE5 |
| AE27UOZENYSWCQVQRRUQIV2ZM7VA,AGMYSLV6NNOAYES25JDTJPCZY47A,AFHS33MWRQGSS64EETZJGCBWXXXA,AHYXZVXUY3Q |
| AE2JTMRKTUOIVIZWS2WDGTMNTU4Q,AF4QXCB32VC2DVE7O3DGFNQVFFNQ,AGAFYHMPFGVPR3MOS4QAZLAWPW3A,AGNNWLEF6V57T |

5 rows × 1348 columns

## Experiments

### Experiment 1: GPT-like approach

The GPT-like method for recommendation generation is a simplified simulation designed to resemble the way a language model, like GPT, might handle text-based user queries to recommend items. This method uses a rule-based simulation with Python and user-item ratings data, rather than an actual GPT model or natural language processing.

In [169]:

```
# Data Preparation: Converting the matrix to a textual format
user_preferences = {}
for user in product_user_matrix.index:
    liked_items = product_user_matrix.columns[(product_user_matrix.loc[user] > 3)] # considering ratings above 3 as 'li
    preference_text = f"{user} likes " + ", ".join(liked_items)
    user_preferences[user] = preference_text

# Example of user preference text for AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O46THXE
example_user_preference = user_preferences["AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O
example_user_preference
```

Out[169]:

```
'AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O46THXEUY3SA,AFPMBSBIEX45OQ6UCQWPDG55GWLQ,AG
WJU3WUQBDQYPSYAJSR3AKBLCOA,AEOVUNFCIFV223O536GVW5JHZKOA likes B07ZR4S1G4'
```

In [170]:

```
def recommend_items(user_query, df):
    # Parsing the user query to extract liked items
    liked_items_in_query = user_query.replace("I liked ", "").split(", ")

    # Check if the liked items are in the DataFrame columns
    valid_liked_items = [item for item in liked_items_in_query if item in df.columns]

    # Identifying users with similar preferences (simpler approach)
    similar_users = df.index[df[valid_liked_items].mean(axis=1) > 3]

    # Finding recommended items based on similar users' preferences
    recommended_items = df.columns[df.loc[similar_users].mean() > 3]
    recommended_items = set(recommended_items) - set(valid_liked_items)

    return list(recommended_items)
```

In [171]:

```
# Test the function with the user query
user_query = "I liked B07ZR4S1G4. What other items would you recommend?"
recommendation_list = recommend_items(user_query, product_user_matrix)
print(recommendation_list)
```

```
[]
```

### Experiment 2: Singular Value Decomposition (SVD)

A popular matrix factorization method in recommendation systems for predicting missing items in user-item interaction matrices is called singular value decomposition, or SVD. Recommendation algorithm optimization for a wide range of applications is greatly aided by SVD, which reveals hidden features that improve personalized suggestion accuracy by dissecting the original matrix into essential components.

In [172]:

```
from sklearn.preprocessing import LabelEncoder

svd_df = df.copy()
svd_df = svd_df[["user_id", "product_id", "rating"]]

product_encoder = LabelEncoder()
user_encoder = LabelEncoder()

svd_df.product_id = product_encoder.fit_transform(knn_df.product_id)
svd_df.user_id = user_encoder.fit_transform(knn_df.user_id)
```

In [173]:
```python
from sklearn.model_selection import train_test_split
from scipy.linalg import svd
from sklearn.metrics import mean_squared_error
from math import sqrt

# Convert the DataFrame to a matrix
ratings_matrix = product_user_matrix.values
user_ids = product_user_matrix.index.tolist()
item_ids = product_user_matrix.columns.tolist()

# Perform SVD
U, sigma, VT = svd(ratings_matrix)

# Using a reduced number of singular values to reconstruct the matrix
k = min(len(user_ids), len(item_ids), 10)  # Ensure k doesn't exceed dimensions of U and VT
sigma_reduced = np.diag(sigma[:k])
U_reduced = U[:, :k]
VT_reduced = VT[:k, :]

# Reconstruct the Matrix with reduced singular values
ratings_matrix_reconstructed = np.dot(U_reduced, np.dot(sigma_reduced, VT_reduced))
df_ratings_reconstructed = pd.DataFrame(ratings_matrix_reconstructed, index=user_ids, columns=item_ids)

# Generate Recommendations using SVD
def recommend_items_svd(user_id, original_df, reconstructed_df, num_recommendations=5):
    # Identify items not rated by the user (assuming unrated items are zeros)
    unrated_items = original_df.columns[original_df.loc[user_id] == 0]

    # Predict ratings for unrated items
    predicted_ratings = reconstructed_df.loc[user_id, unrated_items]

    # Recommend items with the highest predicted ratings
    recommended_items = predicted_ratings.nlargest(num_recommendations).index.tolist()

    return recommended_items

recommendations_svd = recommend_items_svd('AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O4
print(recommendations_svd)
```

```
['B002PD61Y4', 'B002SZEOLG', 'B003B00484', 'B003L62T7W', 'B004IO5BMQ']
```

## Experiment 3: K Nearest Neighbors (KNN)

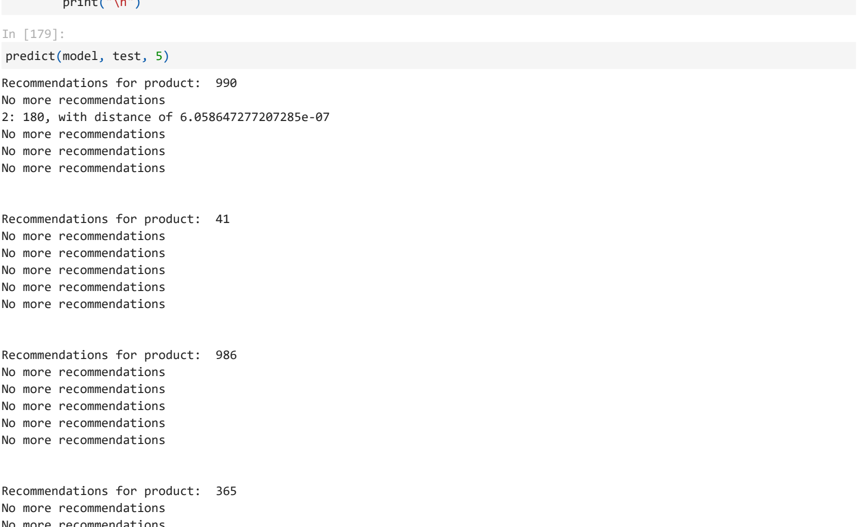
K-Nearest Neighbors (KNN) is used in recommendation systems to identify and recommend items based on their similarity to items that a user has interacted with or shown interest in. The approach involves measuring the similarity between items or users using a distance metric.

In [174]:
```python
from sklearn.preprocessing import LabelEncoder

knn_df = df.copy()
knn_df = knn_df[["user_id", "product_id", "rating"]]

product_encoder = LabelEncoder()
user_encoder = LabelEncoder()

knn_df.product_id = product_encoder.fit_transform(knn_df.product_id)
knn_df.user_id = user_encoder.fit_transform(knn_df.user_id)
```

In [175]:
```python
from sklearn.model_selection import train_test_split

train, test = train_test_split(knn_df, test_size=0.05, random_state=42)
```

In [176]:
```python
from sklearn.neighbors import NearestNeighbors

model = NearestNeighbors(metric='cosine', algorithm='brute')
model.fit(train)
distances, indices = model.kneighbors(train)
```

In [177]:
```python
plt.figure(figsize=(15, 8))
plt.scatter(x=range(0, len(indices)), y=indices[:, 0])
plt.scatter(x=range(0, len(indices)), y=indices[:, 1])
plt.scatter(x=range(0, len(indices)), y=indices[:, 2])
plt.scatter(x=range(0, len(indices)), y=indices[:, 3])
plt.scatter(x=range(0, len(indices)), y=indices[:, 4])
plt.show()
```




In [178]:
```python
def predict(model, data, n_recommendations):
    distances, indices = model.kneighbors(data)
    for i in range(len(data)):
        print("Recommendations for product: ", data.iloc[i, 0])
        for j in range(1, n_recommendations + 1):
            try:
                print(f"{j}: {data.iloc[indices[i, j], 0]}, with distance of {distances[i, j]}")
            except:
                print("No more recommendations")
                continue
        print("\n")
```

In [179]:
```python
predict(model, test, 5)
```

```
Recommendations for product:  990
No more recommendations
2: 180, with distance of 6.058647277207285e-07
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  41
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  986
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  365
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  711
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  248
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  223
No more recommendations
2: 483, with distance of 2.7603186316116357e-05
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  483
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  50
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1120
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  229
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  838
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  292
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1030
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  868
1: 350, with distance of 3.0795621530632644e-07
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  180
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  144
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1111
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  596
1: 785, with distance of 6.930184447972998e-06
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  579
No more recommendations
2: 777, with distance of 1.1918648087139161e-05
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  509
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  183
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  465
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  498
1: 67, with distance of 1.5937955825107153e-06
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  991
No more recommendations
No more recommendations
3: 854, with distance of 9.199039214813354e-07
No more recommendations
No more recommendations


Recommendations for product:  496
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  364
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  642
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1165
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  627
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  803
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  168
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1182
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  847
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  306
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  286
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  667
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  172
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  841
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  113
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  350
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  842
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  518
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  67
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  287
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1153
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  773
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  350
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  632
No more recommendations
2: 1030, with distance of 1.8714256766472204e-07
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  460
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  76
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  777
No more recommendations
No more recommendations
3: 614, with distance of 2.8671154691517486e-06
No more recommendations
No more recommendations


Recommendations for product:  957
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  479
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  562
No more recommendations
2: 50, with distance of 2.2501256997653485e-06
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  990
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  288
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  762
No more recommendations
No more recommendations
3: 614, with distance of 2.1065360045202652e-06
No more recommendations
No more recommendations


Recommendations for product:  366
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  106
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  360
No more recommendations
No more recommendations
No more recommendations
4: 496, with distance of 2.369410574776243e-05
No more recommendations


Recommendations for product:  830
1: 366, with distance of 2.721547447404049e-06
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  278
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1178
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  298
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  614
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  1137
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  854
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  890
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  785
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  34
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  870
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  965
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations


Recommendations for product:  973
No more recommendations
No more recommendations
No more recommendations
No more recommendations
No more recommendations
```

## Experiment 4: Hybrid Model (TF-IDF + Count Vectorizer + KNN)

- TF-IDF and Count Vectorizer provide representations of item features by capturing the importance of words in textual data.
- KNN excels in capturing user-item interactions and similarities between items or users.
- By integrating TF-IDF, Count Vectorizer, and KNN, the system can make personalized recommendations based on both content similarity and collaborative filtering, incorporating the benefits of term frequency and item counts.

In [180]:
```python
from sklearn.preprocessing import LabelEncoder

hybrid_df = df.copy()
```

**Create a weighted matrix.**

- High rating products with many rating counts need to have a higher priority compared to high ratings and low rating counts. But low ratings and high ratings counts should have a lower priority than low ratings and low rating counts. Using log(rating_count + 1) for calculating weighted scores in a recommendation system serves two key purposes:
  1. **Diminishing Returns:** As the rating count increases, its impact on the weighted score grows, but at a decreasing rate.
  2. **Handling Zero Counts:** Adding 1 inside the logarithm ensures that the function can handle cases where rating_count is zero.
- Before we calculated a sentinent analysis score. We now use this score to multiply the product rating * log(rating_count + 1) to get a more accurate weight of each user rating.

In [181]:
```python
import numpy as np

def create_weighted_interaction_matrix(df):

    # Calculate weighted score
    hybrid_df['weighted_score'] = (hybrid_df['rating'] * np.log(hybrid_df['rating_count'] + 1)) * hybrid_df["reviews_sc
    # Pivot to create a matrix
    interaction_matrix = hybrid_df.pivot_table(index='user_id', columns='product_id', values='weighted_score')
    # Fill missing values with zeros
    interaction_matrix = interaction_matrix.fillna(0)
    return interaction_matrix
```

In [182]:
```python
hybrid_df = hybrid_df[["product_id", "rating", "rating_count", "user_id", "reviews_scores", "product_name", "combined_t
weighted_interaction_matrix = create_weighted_interaction_matrix(hybrid_df)
weighted_interaction_matrix.head()
```

Out[182]:

| | |
|---|---|
| | **AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDV** |
| **AE23RS3W7GZO7LHYKJU6KSKVM4MQ,AEQUNEY6GQOTEGUMS6KRUEYNXJSQ,AGYPIE5BICV44WEEEPJVEFQOCJSQ,AFR7CEQKWZE5** | |
| **AE27UOZENYSWCQVQRRUQIV2ZM7VA,AGMYSLV6NNOAYES25JDTJPCZY47A,AFHS33MWRQGSS64EETZJGCBWXXXA,AHYXZVXUY3Q** | |
| **AE2JTMRKTUOIVIZWS2WDGTMNTU4Q,AF4QXCB32VC2DVE7O3DGFNQVFFNQ,AGAFYHMPFGVPR3MOS4QAZLAWPW3A,AGNNWLEF6V57T** | |

5 rows × 1345 columns

In [183]:
```python
text_hybrid_df = hybrid_df[["product_id", "product_name", "combined_text"]]
text_hybrid_df.head()
```

Out[183]:

| | product_id | product_name | combined_text |
|---|---|---|---|
| 0 | B07JW9H4J1 | wayona nylon braided usb lightning fast chargi... | wayona nylon braided usb lightning fast chargi... |
| 1 | B098NS6PVG | ambrane unbreakable 60w 3a fast charging 15m b... | ambrane unbreakable 60w 3a fast charging 15m b... |
| 2 | B096MSW6CT | sounce fast phone charging cable data sync usb... | sounce fast phone charging cable data sync usb... |
| 3 | B08HDJ86NZ | boat deuce usb 300 2 1 typec micro usb stress ... | boat deuce usb 300 2 1 typec micro usb stress ... |
| 4 | B08CF3B7N1 | portronics konnect l 12m fast charging 3a 8 pi... | portronics konnect l 12m fast charging 3a 8 pi... |

**TF-IDF**

In [184]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

def tfidf_vectorization(text_hybrid_df):
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(text_hybrid_df['combined_text'])
    return tfidf_matrix
```

**Train the KNN model**

In [185]:
```python
from sklearn.feature_extraction.text import CountVectorizer

def count_vectorization(text_hybrid_df):
    count_vectorizer = CountVectorizer()
    count_matrix = count_vectorizer.fit_transform(text_hybrid_df['combined_text'])
    return count_matrix
```

**K-Nearest Neighbors**

In [186]:
```python
from sklearn.neighbors import NearestNeighbors

def knn_model(feature_matrix, n_neighbors=5):
    knn = NearestNeighbors(n_neighbors=n_neighbors, algorithm='brute', metric='cosine')
    knn.fit(feature_matrix)
    return knn
```

**Hybrid recommendation**

In [187]:
```python
tfidf_matrix = tfidf_vectorization(text_hybrid_df)
count_matrix = count_vectorization(text_hybrid_df)
knn_tfidf = knn_model(tfidf_matrix)
knn_count = knn_model(weighted_interaction_matrix)
```

### Combine Scores

In [188]:

```python
import numpy as np

def recommend_products(product_id, user_id, knn_tfidf, knn_count, text_hybrid_df, weighted_interaction_matrix, top_n=5)
    # Step 1: Find similar products using TF-IDF
    distances, indices = knn_tfidf.kneighbors(tfidf_matrix[product_id], n_neighbors=top_n)
    similar_product_ids = text_hybrid_df.iloc[indices.flatten()]['product_id']

    # Step 2: Filter/prioritize using user interactions
    user_interactions = weighted_interaction_matrix.iloc[user_id]
    interaction_scores = user_interactions[similar_product_ids].values

    # Combine scores
    combined_scores = distances.flatten() - interaction_scores
    recommended_product_indices = np.argsort(combined_scores)[:top_n]

    # Get recommended product IDs
    recommended_products = np.unique(similar_product_ids.iloc[recommended_product_indices].values)

    return recommended_products
```

In [189]:

```python
# Data Preparation: Converting the matrix to a textual format
user_preferences = {}
for user in product_user_matrix.index:
    liked_items = product_user_matrix.columns[(product_user_matrix.loc[user] > 3)] # considering ratings above 3 as 'li
    preference_text = f"{user} likes " + ", ".join(liked_items)
    user_preferences[user] = preference_text

# Example of user preference text for AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O46THXE
example_user_preference = user_preferences["AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O
example_user_preference
```

Out[189]:

```
'AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O46THXEUY3SA,AFPMBSBIEX45OQ6UCQWPDG55GWLQ,AG
WJU3WUQBDQYPSYAJSR3AKBLCOA,AEOVUNFCIFV223O536GVW5JHZKOA likes B07ZR4S1G4'
```

In [198]:

```python
product_id = 9
user_id = 0
recommended_products = recommend_products(product_id, user_id, knn_tfidf, knn_count, text_hybrid_df, weighted_interacti

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

```
Recommended Products: ['B082LZGK39' 'B098NS6PVG']
```

## Experiment 5: Neural Network (NN)

This model architecture focuses on analyzing product-user interactions with a rating system. The architecture is structured as follows:

1. **Input Layers:** Distinct input layers for users and products, handling two streams of data.
2. **Embedding Layers:** Each input type is passed through an embedding layer to obtain 50-dimensional vectors.
3. **Flattening:** The higher-dimensional outputs from embedding layers are flattened into vectors.
4. **Dot Product Layer:** A dot product of the user and product vectors is calculated, merging the data into a single scalar value.
5. **Model Compilation:** The network is compiled with the Adam optimizer and binary crossentropy as the loss function, optimizing it for binary classification tasks.

The objective is to understand how closely a user's preferences align with a product's characteristics. The embeddings capture the nuances of users and products, and the dot product measures their compatibility. The mean squared error function is chosen to minimize prediction errors.

In [191]:

```python
weighted_interaction_df = weighted_interaction_matrix.reset_index()
weighted_interaction_df = weighted_interaction_df.melt(id_vars='user_id', var_name='product_id', value_name='weighted_s
weighted_interaction_df.head()
```

Out[191]:

| | user_id | product_id | weighted_score |
|---|---|---|---|
| 0 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B002PD61Y4 | 0.0 |
| 1 | AE23RS3W7GZO7LHYKJU6KSKVM4MQ,AEQUNEY6GQOTEGUMS... | B002PD61Y4 | 0.0 |
| 2 | AE242TR3GQ6TYC6W4SJ5UYYKBTYQ | B002PD61Y4 | 0.0 |
| 3 | AE27UOZENYSWCQVQRRUQIV2ZM7VA,AGMYSLV6NNOAYES25... | B002PD61Y4 | 0.0 |
| 4 | AE2JTMRKTUOIVIZWS2WDGTMNTU4Q,AF4QXCB32VC2DVE7O... | B002PD61Y4 | 0.0 |

In [192]:

```python
# Convert user_id and movie_id to categorical codes
weighted_interaction_df['user_id'] = weighted_interaction_df['user_id'].astype('category')
weighted_interaction_df['product_id'] = weighted_interaction_df['product_id'].astype('category')

user_ids = weighted_interaction_df['user_id'].cat.codes.values
product_ids = weighted_interaction_df['product_id'].cat.codes.values

# Split the dataset
X = np.stack([user_ids, product_ids], axis=1)
y = weighted_interaction_df['weighted_score'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model parameters
n_users = weighted_interaction_df['user_id'].nunique()
n_products = weighted_interaction_df['product_id'].nunique()
n_factors = 50
```

In [193]:

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Embedding, Flatten, Input, Dot
from tensorflow.keras.optimizers import Adam

# Building the model
user_input = Input(shape=(1,), name='user_input') # User Input
product_input = Input(shape=(1,), name='product_input') # Product Input
user_embedding = Embedding(output_dim=n_factors, input_dim=n_users, input_length=1, name='user_embedding')(user_input)
product_embedding = Embedding(output_dim=n_factors, input_dim=n_products, input_length=1, name='product_embedding')(pro
user_vector = Flatten(name='flatten_users')(user_embedding) # Flatten Users
product_vector = Flatten(name='flatten_products')(product_embedding) # Flatten Products
dot_product = Dot(axes=1, name='dot_product')([user_vector, product_vector]) # Dot Product
model = Model(inputs=[user_input, product_input], outputs=dot_product) # Model creation
model.compile(optimizer=Adam(0.001), loss='mean_squared_error') # Compile the model
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 user_input (InputLayer)     [(None, 1)]                  0         []

 product_input (InputLayer)  [(None, 1)]                  0         []

 user_embedding (Embedding)  (None, 1, 50)                59400     ['user_input[0][0]']

 product_embedding (Embeddi  (None, 1, 50)                67250     ['product_input[0][0]']
 ng)

 flatten_users (Flatten)     (None, 50)                   0         ['user_embedding[0][0]']

 flatten_products (Flatten)  (None, 50)                   0         ['product_embedding[0][0]']

 dot_product (Dot)           (None, 1)                    0         ['flatten_users[0][0]',
                                                                     'flatten_products[0][0]']

==================================================================================================
Total params: 126650 (494.73 KB)
Trainable params: 126650 (494.73 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

In [194]:

```python
# Train the model
model.fit([X_train[:, 0], X_train[:, 1]], y_train, epochs=5, validation_data=([X_test[:, 0], X_test[:, 1]], y_test))
```

```
Epoch 1/5
39947/39947 [==============================] - 130s 3ms/step - loss: 0.1063 - val_loss: 0.1246
Epoch 2/5
39947/39947 [==============================] - 126s 3ms/step - loss: 0.1063 - val_loss: 0.1246
Epoch 3/5
39947/39947 [==============================] - 126s 3ms/step - loss: 0.1063 - val_loss: 0.1246
Epoch 4/5
39947/39947 [==============================] - 126s 3ms/step - loss: 0.1063 - val_loss: 0.1246
Epoch 5/5
39947/39947 [==============================] - 125s 3ms/step - loss: 0.1063 - val_loss: 0.1246
```

Out[194]:

```
<keras.src.callbacks.History at 0x794f5c78a710>
```

In [195]:

```python
import random
user_id_to_predict = "AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZAHJK6NGYHFMA,AGYURQ3476BNT4D2O46THXEUY3SA,AFPMBSBIEX
product_ids_to_predict = weighted_interaction_df['product_id'].tolist()
product_ids_to_predict = random.sample(product_ids_to_predict, 10000)
product_input_array = np.array(product_ids_to_predict)

# Since our model expects input in the form of category codes,
# we need to convert these IDs to their corresponding codes
user_id_code = np.argwhere(weighted_interaction_df['user_id'].cat.categories == user_id_to_predict)[0, 0]

predictions = []

# Iterate over each product using tqdm for the progress bar
for product in tqdm(product_input_array, desc="Predicting ratings"):

    # Get the encoded product ID
    product_id_code = np.argwhere(weighted_interaction_df['product_id'].cat.categories == product)[0, 0]

    # Making the prediction
    predicted_rating = model.predict([np.array([user_id_code]), np.array([product_id_code])], verbose=0)

    predictions.append(predicted_rating)

prediction_results = pd.DataFrame({
    'user_id': [user_id_to_predict]*10000,
    'product_id': product_ids_to_predict,
    'prediction': predictions
})

prediction_results
```

```
Predicting ratings:   0%|          | 0/10000 [00:00<?, ?it/s]
```

Out[195]:

| | user_id | product_id | prediction |
|---|---|---|---|
| 0 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09PNKXSKF | [[0.0014418687]] |
| 1 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B08LT9BMPP | [[0.0022619823]] |
| 2 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B0B2CWRDB1 | [[0.004258927]] |
| 3 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09LHZSMRR | [[-0.000953355]] |
| 4 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B0B1YVCJ2Y | [[0.0035950392]] |
| ... | ... | ... | ... |
| 9995 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B08497Z1MQ | [[0.0013286453]] |
| 9996 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09VCHLSJF | [[-0.011740241]] |
| 9997 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B07NPBG1B4 | [[0.0034995624]] |
| 9998 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B00GZLB57U | [[-0.004505522]] |
| 9999 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09XB1R2F3 | [[0.00037350203]] |

10000 rows × 3 columns

The key components are the embedding layers for users and products, and their interaction through a dot product. The output of the model, which is the dot product of the user and product embeddings, can be interpreted as follows:

1. **Positive Values:** A positive value from the dot product indicates a positive interaction between the user and the product. In the context of a recommendation system, a positive value typically suggests that the user is likely to have a positive preference or affinity for the product. Higher positive values indicate a stronger predicted preference.

2. **Negative Values:** Conversely, a negative value from the dot product suggests a negative interaction. This implies that the user is less likely or less inclined to prefer or like the product. More negative values indicate a stronger negative preference or dislike.

The essence of this model is to learn the latent factors (hidden features) of both users and products through the embedding layers. When the embeddings of a user and a product are similar, their dot product tends to be higher (positive), indicating compatibility or a higher chance of the user liking the product. If they are dissimilar, the dot product becomes negative, indicating incompatibility.

In [196]:

```python
prediction_results = prediction_results[prediction_results['prediction'] > 0].drop_duplicates(subset=["product_id"]).so
prediction_results.head(10)
```

Out[196]:

| | user_id | product_id | prediction |
|---|---|---|---|
| 640 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B07ZR4S1G4 | [[0.05453977]] |
| 3550 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B0814ZY6FP | [[0.02731574]] |
| 82 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B008FWZGSG | [[0.024092546]] |
| 138 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B00LVMTA2A | [[0.023302758]] |
| 360 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09127FZCK | [[0.022759398]] |
| 45 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B07PR1CL3S | [[0.021975866]] |
| 1271 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B07GVGTSLN | [[0.0200186]] |
| 1869 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09NL7LBWT | [[0.018961199]] |
| 1067 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B09NS5TKPN | [[0.01731132]] |
| 2068 | AE22Y3KIS7SE6LI3HE2VS6WWPU4Q,AHWEYO2IJ5I5GDWZA... | B08HD7JQHX | [[0.01715759]] |

## References


1. Luong Vuong Nguyen, Quoc-Trinh Vo and Tri-Hai Nguyen: Adaptive KNN-Based Extended Collaborative Filtering Recommendation Services. Big Data Cogn. Comput. 2023. [MDPI Paper]
2. Cooper315315: Job-and-Talent-Recommendation-System, 2021. [GitHub]
3. H. Zarzour, Z. A. Al-Sharif and Y. Jararweh, "RecDNNing: a recommender system using deep neural network with user and item embeddings," 2019 10th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2019, pp. 99-103, doi: 10.1109/IACS.2019.8809156. [IEEE Paper]
4. Adrian Tam: Using Singular Value Decomposition to Build a Recommender System, 2021 [Machine Learning Mastery]