Business Problem:

As a data analyst for an e-commerce business, your task is to analyze the customer reviews and ratings data to gain insights that can help improve customer satisfaction and identify areas for business growth.

Three things the stakeholder would want:

Sentiment Analysis:

The stakeholder would like to understand the overall sentiment of the customer reviews. They want to know whether the majority of the reviews are positive, negative, or neutral. This analysis will help them gauge customer satisfaction and identify potential areas of improvement.

Product Performance Comparison:

The stakeholder wants to compare the performance of different products based on customer ratings. They are interested in identifying the best-selling products and determining if there are any specific product categories that consistently receive positive or negative feedback. This information will help them make data-driven decisions about inventory management and product development.

Customer Segmentation:

The stakeholder is keen on understanding the different types of customers based on their reviews and ratings. They want to segment customers into groups based on their preferences and sentiments. This segmentation will help in targeted marketing efforts, personalized recommendations, and improving customer engagement.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
import os
```

```
import csv
```

nRowsRead = 30000 # specify 'None' if want to read whole file Womens Clothing E-Commerce Reviews.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows df = pd.read_csv('/content/Womens Clothing E-Commerce Reviews.csv', delimiter=',', nrows = nRowsRead) df.dataframeName = 'Womens Clothing E-Commerce Reviews.csv' nRow, nCol = df.shape print(f'There are {nRow} rows and {nCol} columns')

Load the dataset

```
# Get the current working directory
current_directory = os.getcwd()

# Construct the file path
file_path = os.path.join(current_directory, "/content/Womens Clothing E-Commerce Reviews.csv")

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
print(df.head())
```

```
   Unnamed: 0  Clothing ID  Age                    Title  \
0           0          767   33                      NaN
1           1         1080   34                      NaN
2           2         1077   60  Some major design flaws
3           3         1049   50         My favorite buy!
4           4          847   47         Flattering shirt

                                      Review Text  Rating  Recommended IND  \
0  Absolutely wonderful - silky and sexy and comf...       4                1
1  Love this dress!  it's sooo pretty.  i happene...       5                1
2  I had such high hopes for this dress and reall...       3                0
3  I love, love, love this jumpsuit. it's fun, fl...       5                1
4  This shirt is very flattering to all due to th...       5                1

   Positive Feedback Count   Division Name Department Name Class Name
0                        0       Initmates        Intimate  Intimates
```

```
1                     4         General         Dresses    Dresses
2                     0         General         Dresses    Dresses
3                     0  General Petite         Bottoms      Pants
4                     6         General            Tops    Blouses
```

EXPLORE THE DATASET

```
df.head()
```

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Divi
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initr
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	Ge
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	Ge
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General I
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	Ge

```
df.shape
```

```
(23486, 11)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23486 entries, 0 to 23485
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Unnamed: 0               23486 non-null  int64 
 1   Clothing ID              23486 non-null  int64 
 2   Age                      23486 non-null  int64 
 3   Title                    19676 non-null  object
 4   Review Text              22641 non-null  object
 5   Rating                   23486 non-null  int64 
 6   Recommended IND          23486 non-null  int64 
 7   Positive Feedback Count  23486 non-null  int64 
 8   Division Name            23472 non-null  object
 9   Department Name          23472 non-null  object
 10  Class Name               23472 non-null  object
dtypes: int64(6), object(5)
memory usage: 2.0+ MB
```

```
df.columns
```

```
Index(['Unnamed: 0', 'Clothing ID', 'Age', 'Title', 'Review Text', 'Rating',
       'Recommended IND', 'Positive Feedback Count', 'Division Name',
       'Department Name', 'Class Name'],
      dtype='object')
```

```
df.dtypes
```

```
Unnamed: 0                  int64
Clothing ID                 int64
Age                         int64
Title                      object
Review Text                object
Rating                      int64
Recommended IND             int64
Positive Feedback Count     int64
Division Name              object
Department Name            object
Class Name                 object
dtype: object
```

```
df.describe()
```

	Unnamed: 0	Clothing ID	Age	Rating	Recommended IND	Positive Feedback Count
count	23486.000000	23486.000000	23486.000000	23486.000000	23486.000000	23486.000000
mean	11742.500000	918.118709	43.198544	4.196032	0.822362	2.535936
std	6779.968547	203.298980	12.279544	1.110031	0.382216	5.702202
min	0.000000	0.000000	18.000000	1.000000	0.000000	0.000000
25%	5871.250000	861.000000	34.000000	4.000000	1.000000	0.000000
50%	11742.500000	936.000000	41.000000	5.000000	1.000000	1.000000
75%	17613.750000	1078.000000	52.000000	5.000000	1.000000	3.000000

```
df1 = df.copy()
```

CLEAN THE DATASET

```
df1.isnull().sum()
```

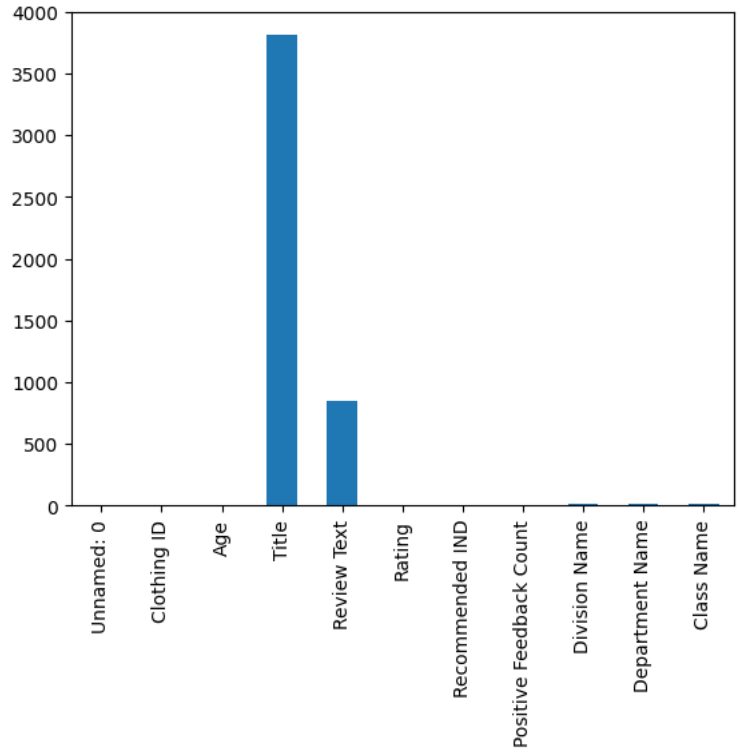
```
Unnamed: 0                    0
Clothing ID                   0
Age                           0
Title                      3810
Review Text                 845
Rating                        0
Recommended IND               0
Positive Feedback Count       0
Division Name                14
Department Name              14
Class Name                   14
dtype: int64
```

Double-click (or enter) to edit

plot a bar of missing values to get a better feel of them.

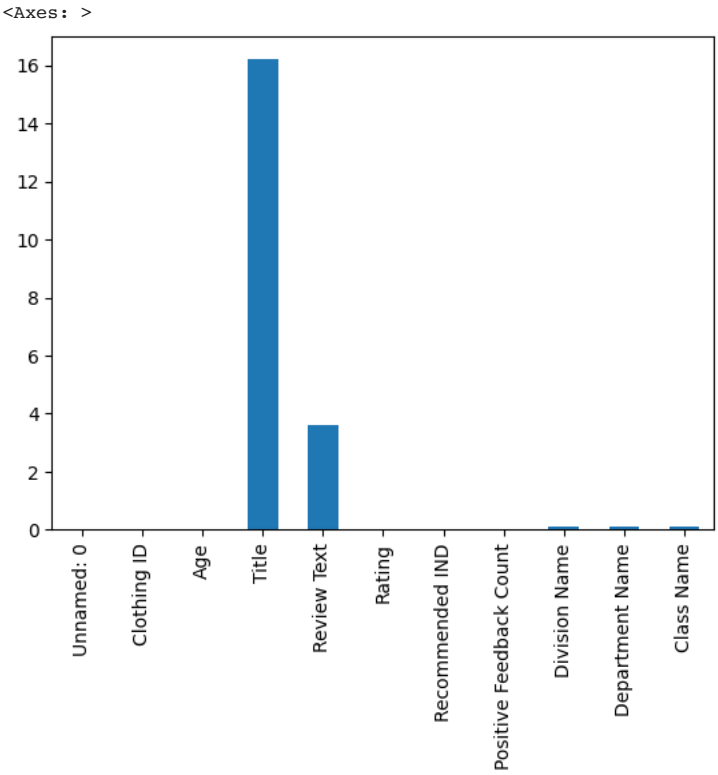
```
missing_values = df1.isnull().sum()

missing_values.plot(kind = 'bar')
```

```
<Axes: >
```




```
missing_percent = round(df1.isnull().sum() / len(df1) * 100, 1)

missing_percent.plot(kind = 'bar')
```

```
<Axes: >
```



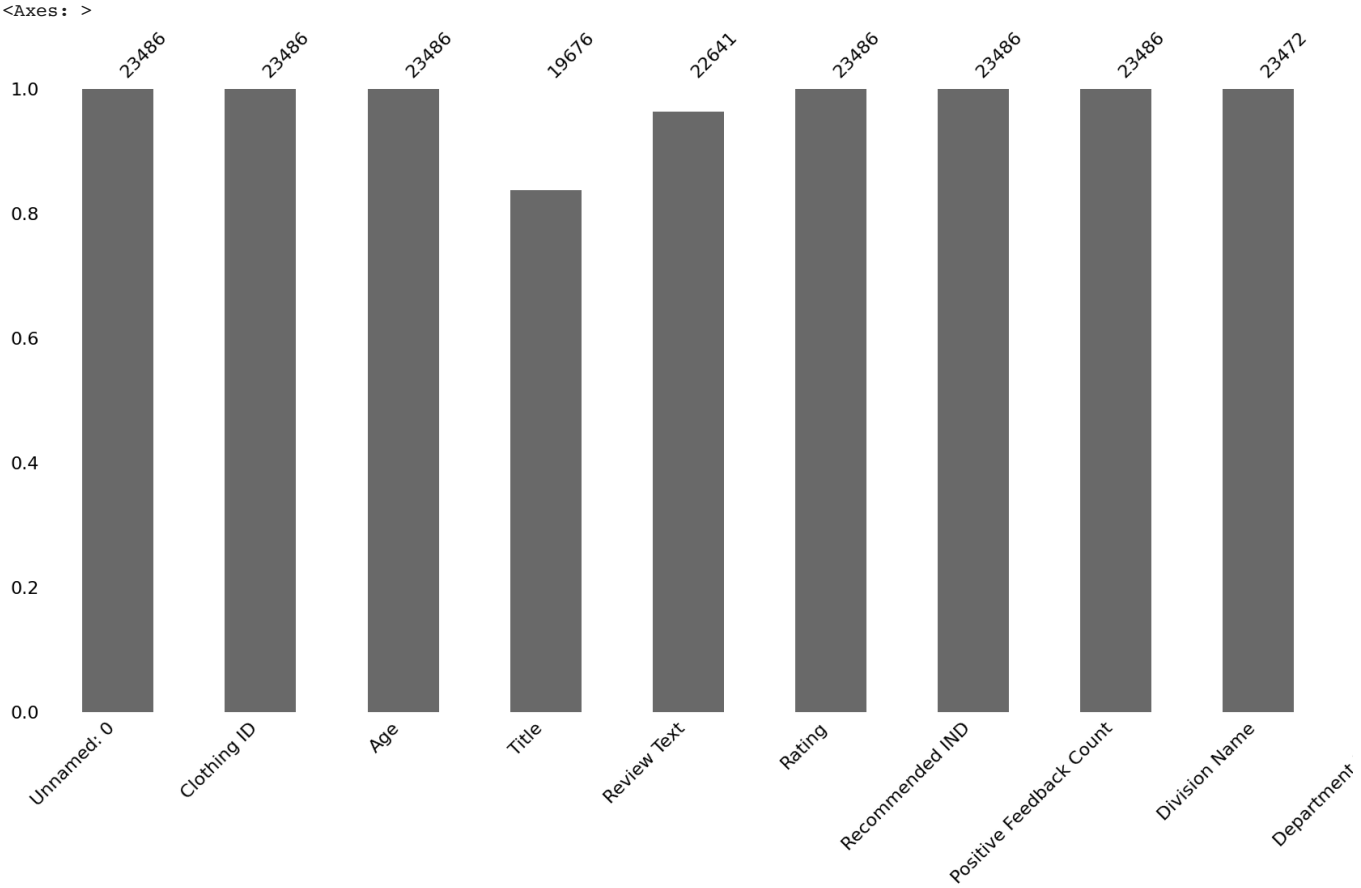
The percentage of missing values shows the percentage of missing values in each column

```
import missingno as msno
```

missingo is a library specifically designed for visualizing and analyzing missing data in datasets. It provides a set of useful visualization functions to help you understand the distribution and patterns of missing values in your data.

```
msno.bar(df1)
```

```
<Axes: >
```



There are missing values in Title, Review Text, Division Name, Department Name, and the Class Name.

```
df1.dtypes
```

```
Unnamed: 0                 int64
Clothing ID                int64
Age                        int64
Title                     object
Review Text               object
Rating                     int64
Recommended IND            int64
Positive Feedback Count    int64
Division Name             object
Department Name           object
Class Name                object
dtype: object
```

```
df1['Title'].head()
```

```
0                       NaN
1                       NaN
2    Some major design flaws
3           My favorite buy!
4           Flattering shirt
Name: Title, dtype: object
```

```
df1['Title'].value_counts()
```

```
Love it!                                    136
Beautiful                                    95
Love                                         88
Love!                                        84
Beautiful!                                   72
                                           ...
Perfect transition dress                      1
The perfect spring dress!                     1
Super soft but can make you look frumpy       1
More structured than a cardi                  1
Please make more like this one!               1
Name: Title, Length: 13993, dtype: int64
```

Filling missing values with an empty string (' ') is important especially when working with text data.

Maintaining Data Structure:

Compatibility with Text-based Operations:

Filling missing values with an empty string allows you to perform various text-based operations without encountering errors.

Preserving Textual Context:

In some cases, the absence of information can still hold some significance within the context of the data. By filling missing values with an empty string, you explicitly indicate that there is no available text information for those instances, allowing you to preserve the textual context and ensure that downstream analyses or models can appropriately handle these missing values.

```
df1['Title'].fillna('', inplace=True)
```

```
df1['Title'].head()
```

```
0                 Love it !
1                 Love it !
2    Some major design flaws
3           My favorite buy!
4           Flattering shirt
Name: Title, dtype: object
```

```
df1['Review Text'].head()
```

```
0    Absolutely wonderful - silky and sexy and comf...
1    Love this dress!  it's sooo pretty.  i happene...
2    I had such high hopes for this dress and reall...
3    I love, love, love this jumpsuit. it's fun, fl...
4    This shirt is very flattering to all due to th...
Name: Review Text, dtype: object
```

```
df1['Review Text'].fillna('', inplace=True)
```

```
df1['Review Text'].head()
```

```
0    Absolutely wonderful - silky and sexy and comf...
1    Love this dress!  it's sooo pretty.  i happene...
2    I had such high hopes for this dress and reall...
3    I love, love, love this jumpsuit. it's fun, fl...
4    This shirt is very flattering to all due to th...
Name: Review Text, dtype: object
```

```
df1['Division Name'].value_counts()
```

```
General           13850
General Petite     8120
Initmates          1502
Name: Division Name, dtype: int64
```

```
df1['Division Name'].replace(np.nan, 'General', inplace = True)
```

Replacing missing categorical values with the mode is a common approach.

The mode represents the most frequent value in a categorical variable, making it a reasonable choice for filling in missing values.

We are essentially imputing the missing values with the most commonly occurring category in that column. This helps to preserve the distribution and characteristics of the existing data while filling in the missing information.

```
df1['Department Name'].value_counts()
```

```
Tops        10468
Dresses      6319
Bottoms      3799
Intimate     1735
Jackets      1032
Trend         119
Name: Department Name, dtype: int64
```

```
df1['Department Name'].head()
```

```
0    Intimate
1     Dresses
2     Dresses
3     Bottoms
4        Tops
Name: Department Name, dtype: object
```

```
df1['Department Name'].replace(np.nan, 'Tops', inplace = True)
```

```
df1['Class Name'].value_counts()
```

```
Dresses           6319
Knits             4843
Blouses           3097
Sweaters          1428
Pants             1388
Jeans             1147
Fine gauge        1100
Skirts             945
Jackets            704
Lounge             691
Swim               350
Outerwear          328
Shorts             317
Sleep              228
Legwear            165
Intimates          154
Layering           146
Trend              119
Casual bottoms       2
Chemises             1
Name: Class Name, dtype: int64
```

```
df1['Class Name'].replace(np.nan, 'Dresses', inplace = True)
```

```
df1.isnull().sum()
```

```
Unnamed: 0                0
Clothing ID               0
Age                       0
Title                     0
Review Text               0
Rating                    0
```

```
Recommended IND            0
Positive Feedback Count    0
Division Name              0
Department Name            0
Class Name                 0
dtype: int64
```

Drop any unnecesary columns

```
df1.drop('Unnamed: 0', axis= 1, inplace = True)
```

```
df1.duplicated().sum()
```

```
21
```

Duplicates sometimes represent multiple occurrences of the same value and should not be dropped.

These duplicates contribute to the overall result of the aggregation operation

Univariate data analysis

```
numerical_df = df1.select_dtypes(include = ['number'])
numerical_df.head()
```

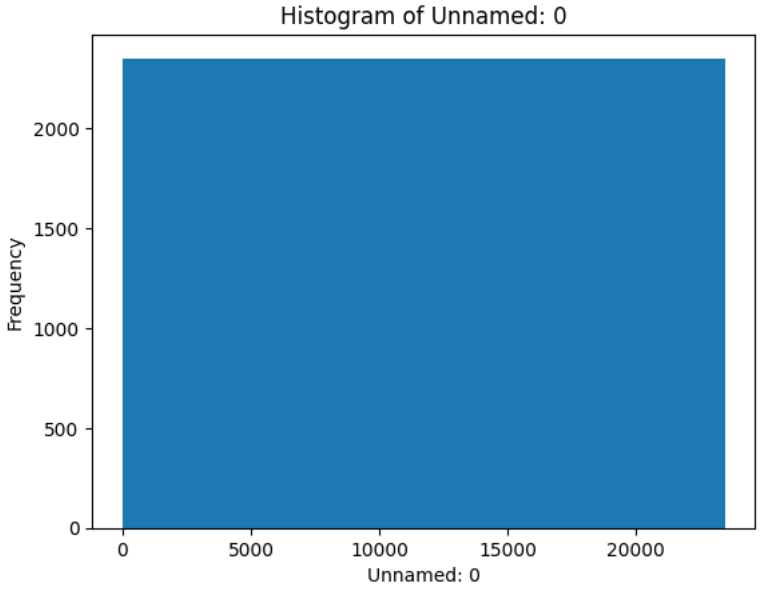
	Unnamed: 0	Clothing ID	Age	Rating	Recommended IND	Positive Feedback Count
0	0	767	33	4	1	0
1	1	1080	34	5	1	4
2	2	1077	60	3	0	0
3	3	1049	50	5	1	0
4	4	847	47	5	1	6

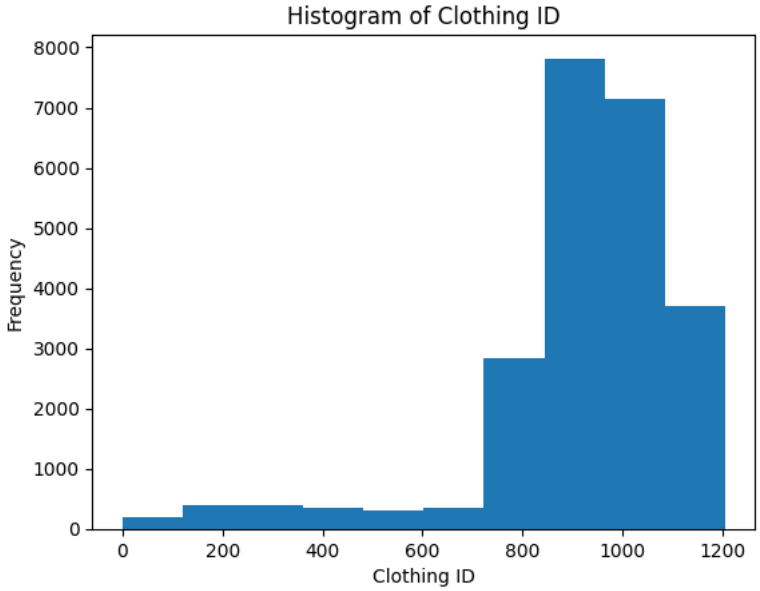
Creating histograms allows us to explore and visualize data in a systematic and automated manner.

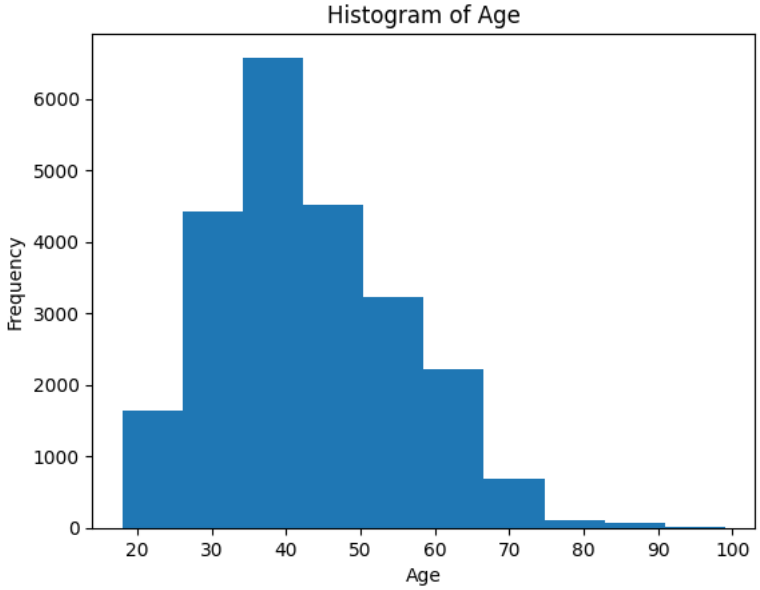
It helps in uncovering patterns, detecting anomalies, and drawing meaningful conclusions about the numerical data. Histograms provide a foundation for further analysis and decision-making based on the distribution of values in the dataset.

By creating numerical_df, we can easily make histograms for all our numerical columns in our dataset.

```
for column in numerical_df.columns:
  plt.hist(numerical_df[column], bins = 10)
  plt.xlabel(column)
  plt.ylabel('Frequency')
  plt.title(f'Histogram of {column}')
  plt.show()
```


Histogram of Unnamed: 0
Frequency
Unnamed: 0
0
500
1000
1500
2000
0
5000
10000
15000
20000


Histogram of Clothing ID
Frequency
Clothing ID
0
1000
2000
3000
4000
5000
6000
7000
8000
0
200
400
600
800
1000
1200


Histogram of Age
Frequency
Age
0
1000
2000
3000
4000
5000
6000
20
30
40
50
60
70
80
90
100


Histogram of Rating
8000
10000
12000

```
categorical_df = df1.select_dtypes(exclude = ['number'])

categorical_df.head()
```

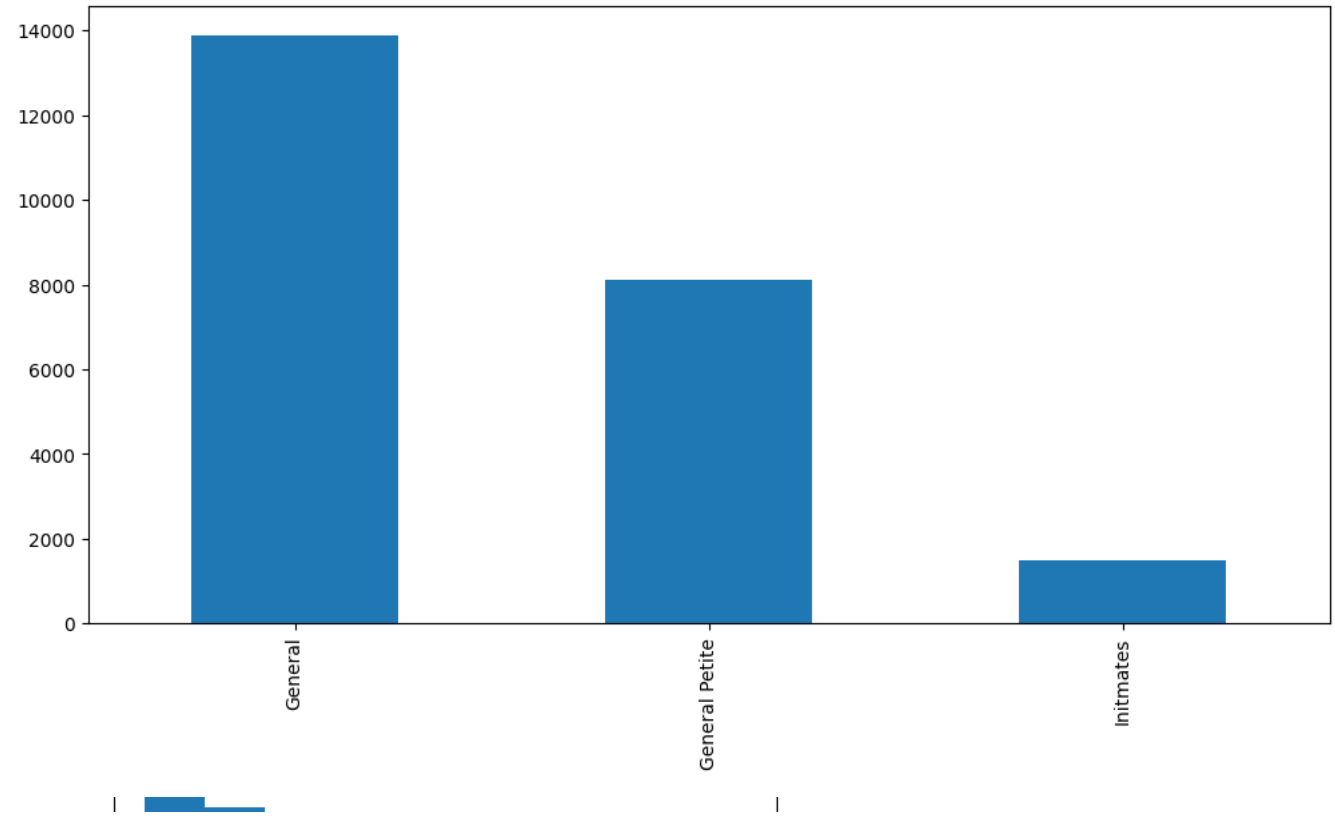
	Title	Review Text	Division Name	Department Name	Class Name
0		Absolutely wonderful - silky and sexy and comf...	Initmates	Intimate	Intimates
1		Love this dress! it's sooo pretty. i happene...	General	Dresses	Dresses
2	Some major design flaws	I had such high hopes for this dress and reall...	General	Dresses	Dresses
3	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	General Petite	Bottoms	Pants
4	Flattering shirt	This shirt is very flattering to all due to th...	General	Tops	Blouses

To understand our columns more, we create bar charts to understand the composition and prominence of different categorical columns, identifying the most common or dominant in each column, and detecting any imbalances or discrepancies in the data.

Overall, this visualization aids in exploring and understanding categorical data in an intuitive way.

```
df1['Division Name'].value_counts().plot(kind = 'bar', figsize = (12,6))
```

```
<Axes: >
```

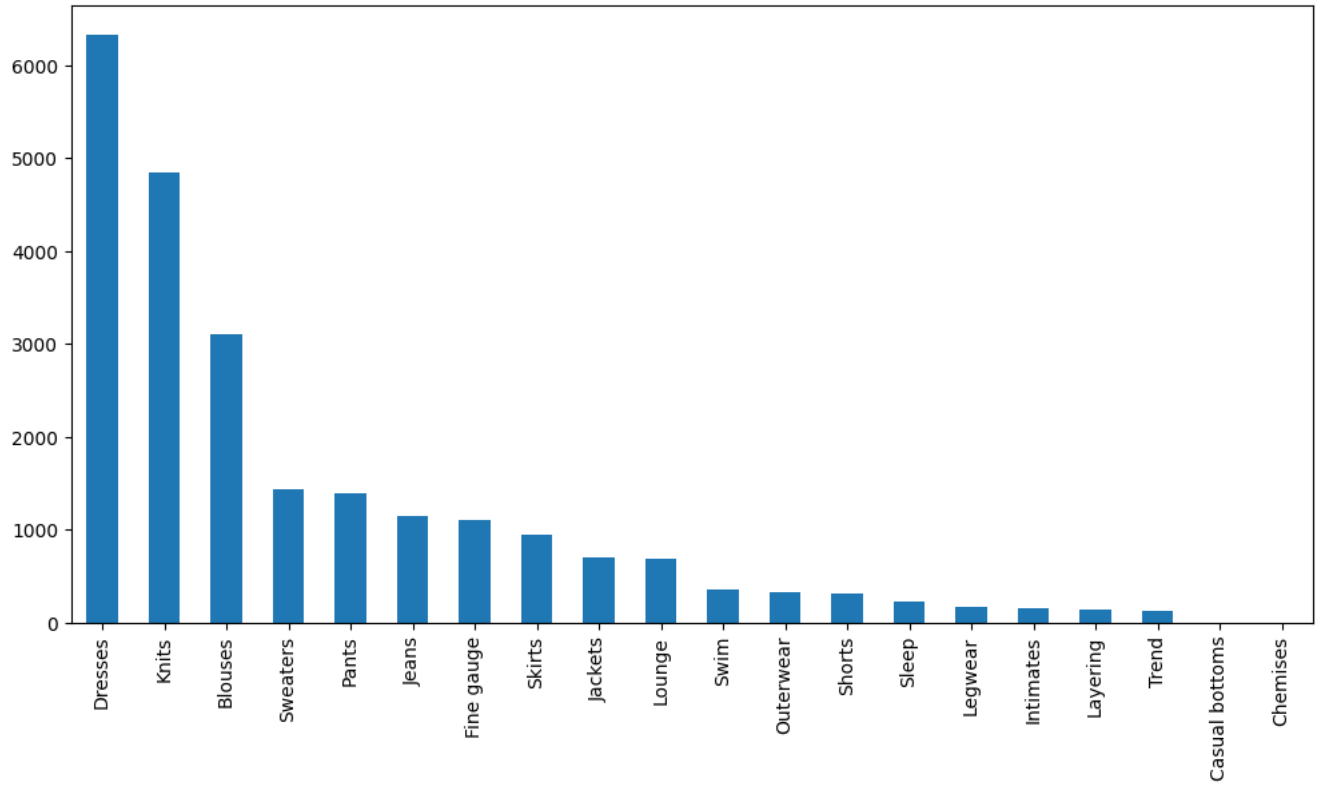



```
df1['Department Name'].value_counts().plot(kind = 'bar', figsize = (12,6))
```

```
<Axes: >
```


```
df1['Class Name'].value_counts().plot(kind = 'bar', figsize = (12,6))
```

```
<Axes: >
```



Bivariate analysis

It refers to the analysis of the relationship between two variables.

It involves examining the association, correlation, or dependencies between two variables to gain insights into their relationship and understand how they interact with each other.

```
corrs =df1.corr()
corrs
```

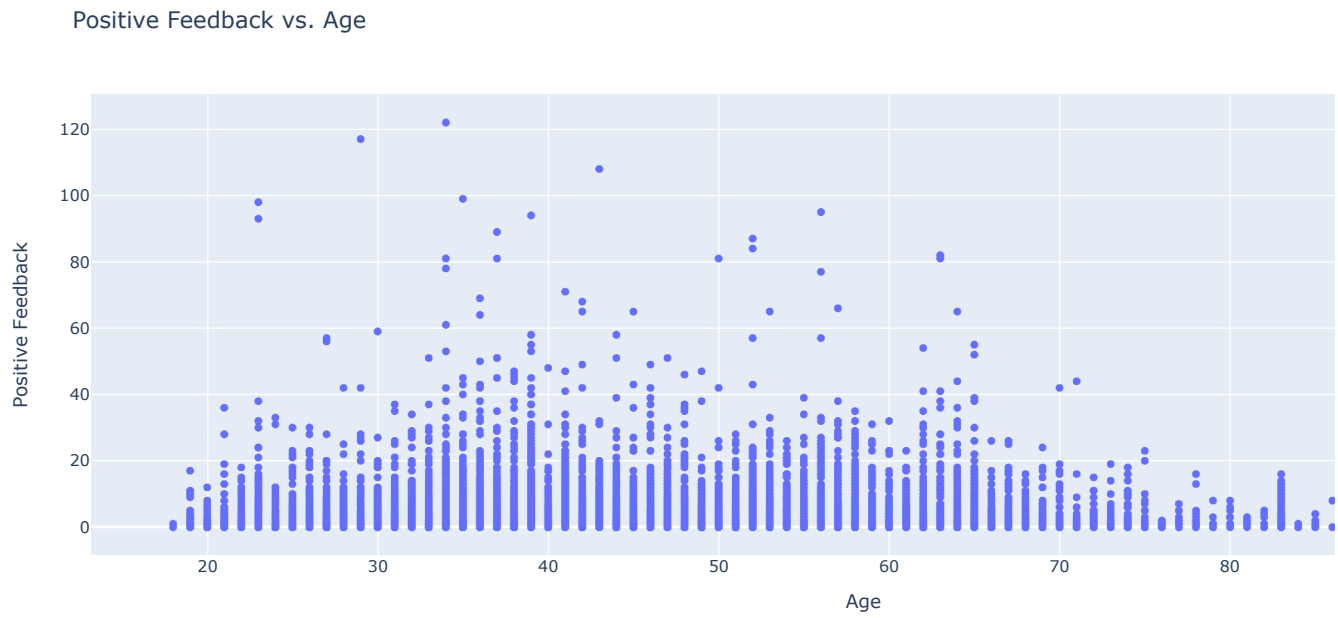
```
<ipython-input-97-90012e75b726>:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select
```

	Clothing ID	Age	Rating	Recommended IND	Positive Feedback Count
Clothing ID	1.000000	0.012433	-0.008202	-0.005699	0.042977
Age	0.012433	1.000000	0.037346	0.040389	0.040223
Rating	-0.008202	0.037346	1.000000	0.792294	-0.055445
Recommended IND	-0.005699	0.040389	0.792294	1.000000	-0.063094
Positive Feedback Count	0.042977	0.040223	-0.055445	-0.063094	1.000000

```
import plotly.express as px
```

```
# Create the scatter plot
fig = px.scatter(df1, x='Age', y='Positive Feedback Count',
                 title="Positive Feedback vs. Age",
                 labels={'Positive Feedback Count': 'Positive Feedback',
                         'Age': 'Age'})
```

```
fig.show()
```



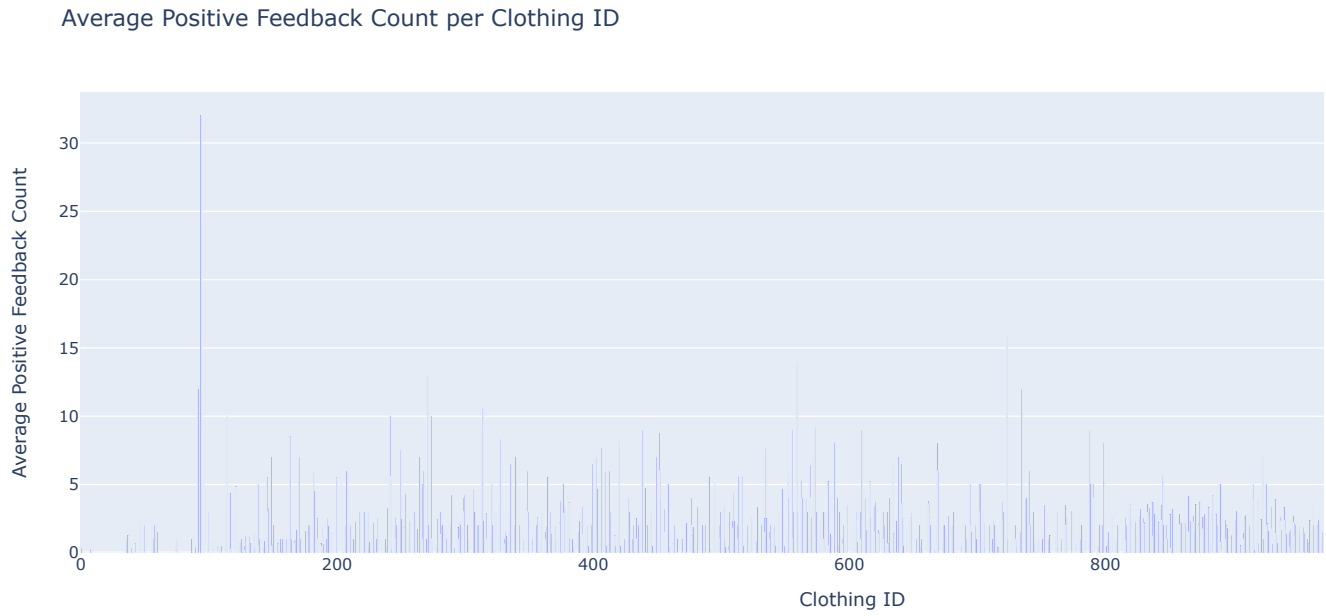
Double-click (or enter) to edit

```
# Create age groups
age_bins = [0, 20, 30, 40, 50, 60, 70, 80, 100]  # Define the age group boundaries
age_labels = ['0-20', '21-30', '31-40', '41-50', '51-60', '61-70', '71-80', '81+']  # Labels for the age groups
df1['Age Group'] = pd.cut(df1['Age'], bins=age_bins, labels=age_labels, right=False)  # Categorize ages into groups



# Create the scatter plot with age groups
fig = px.scatter(df1, x='Age Group', y='Positive Feedback Count',
                 title="Positive Feedback Count vs. Age Group",
                 labels={'Positive Feedback Count': 'Positive Feedback',
                         'Age Group': 'Age Group'})

fig.show()
```

The scatter plot with age groups allows us to compare the positive feedback count across different age categories and gain insights into potential trends or variations within the dataset.

1. There are more positive ratings from the customers in the Age group 31-40 tend to cluster together and exhibit a higher positive feedback count, it suggests that age group may generally have a higher satisfaction level or engagement with the clothes.
2. Possible Outliers present among those in the age 71- 80 who gave a positiv feedback.

```
# Calculate the average positive feedback count per clothing ID
avg_feedback_per_clothing = df1.groupby('Clothing ID')['Positive Feedback Count'].mean().reset_index()

# Create the bar plot
fig = px.bar(avg_feedback_per_clothing, x='Clothing ID', y='Positive Feedback Count',
             title='Average Positive Feedback Count per Clothing ID',
             labels={'Positive Feedback Count': 'Average Positive Feedback Count',
                     'Clothing ID': 'Clothing ID'})

fig.show()
```

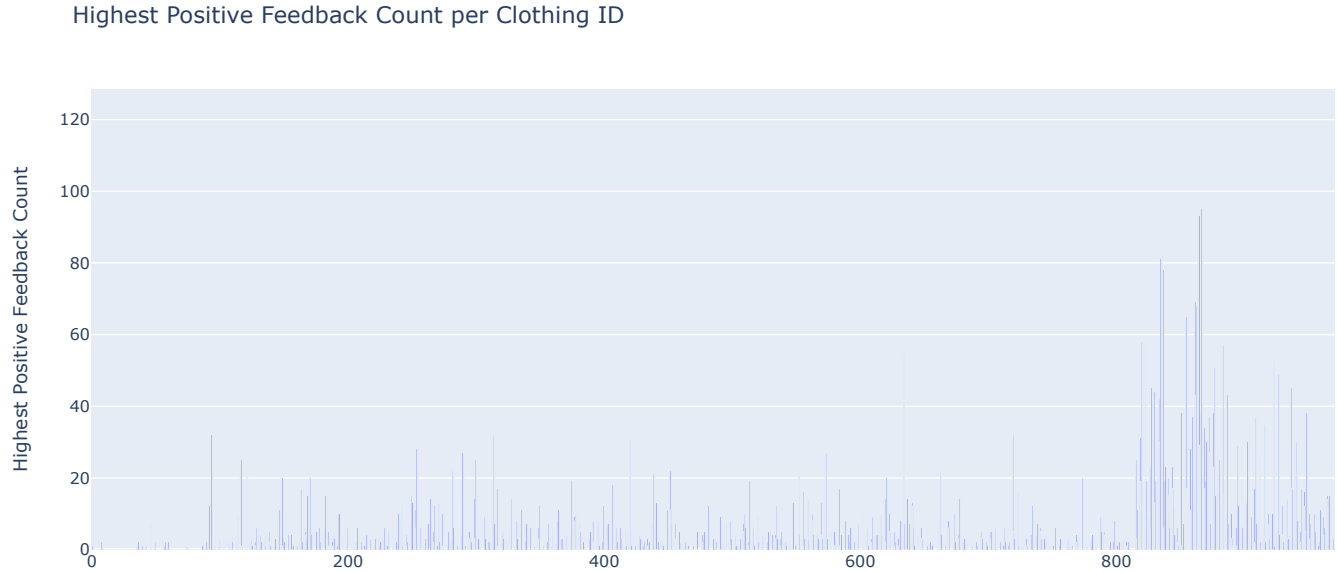


It provides a clear visual representation of the average positive feedback count for each clothing item, making it easier to identify which clothing items receive higher or lower feedback counts.

```
df1[df1['Clothing ID'] == 94]
```

	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name
15903	94	38	Great idea, poor execution	I absolutely loved the idea of an elongated ho...	3	0	32	Initmates

```
# Calculate the highest positive feedback count per clothing ID
max_feedback_per_clothing = df1.groupby('Clothing ID')['Positive Feedback Count'].max().reset_index()

# Create the bar plot
fig = px.bar(max_feedback_per_clothing, x='Clothing ID', y='Positive Feedback Count',
             title='Highest Positive Feedback Count per Clothing ID',
             labels={'Positive Feedback Count': 'Highest Positive Feedback Count',
                     'Clothing ID': 'Clothing ID'})

fig.show()
```



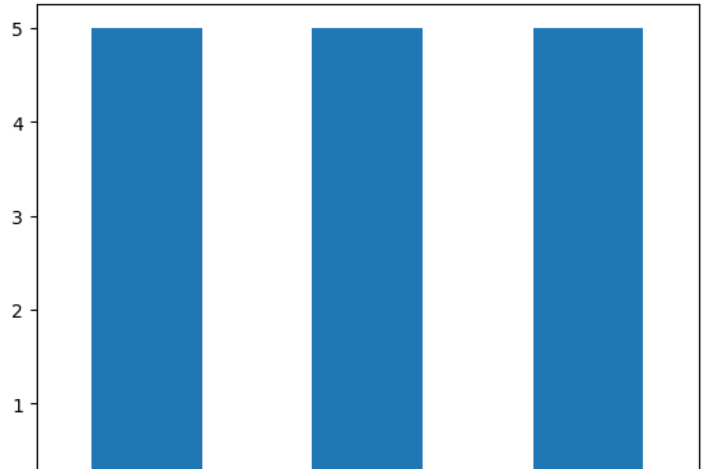
By plotting the highest positive feedback count per clothing ID, you can visualize and compare which clothing items received the most positive feedback

```
df1[df1['Clothing ID'] == 1092]
```

	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name
1935	1092	50	Just right	I ordered this in a size s, my usual size and ...	5	1	0	General Petite
1940	1092	50	Wanted to like it	I thought this dress was very cute on the mode...	3	0	0	General Petite
1943	1092	44		Ugh. i was so excited to get this dress and f...	3	0	0	General Petite
1949	1092	31	Love	I just love this dress! the color, the quality...	5	1	3	General Petite
1962	1092	44	Another huge dress...	Okay, i get the idea of the loose swing dresse...	4	1	3	General Petite
...	...	...	...	...	...	...	...	...
21366	1092	32	For straight figures	This sweater dress was incredibly soft, and i ...	3	1	1	General
21380	1092	67	Lovely color but...	This sweater dress color is great and texture ...	3	0	0	General
21383	1092	30	Love this sweater!	I absolutely love this sweater. it is a great ...	5	1	0	General
21390	1092	51	Wasn't a good fit	I really thought from the picture that the dre...	3	0	1	General
21396	1092	32		Really cute dress for work. fits exactly as i ...	5	1	5	General

220 rows × 11 columns

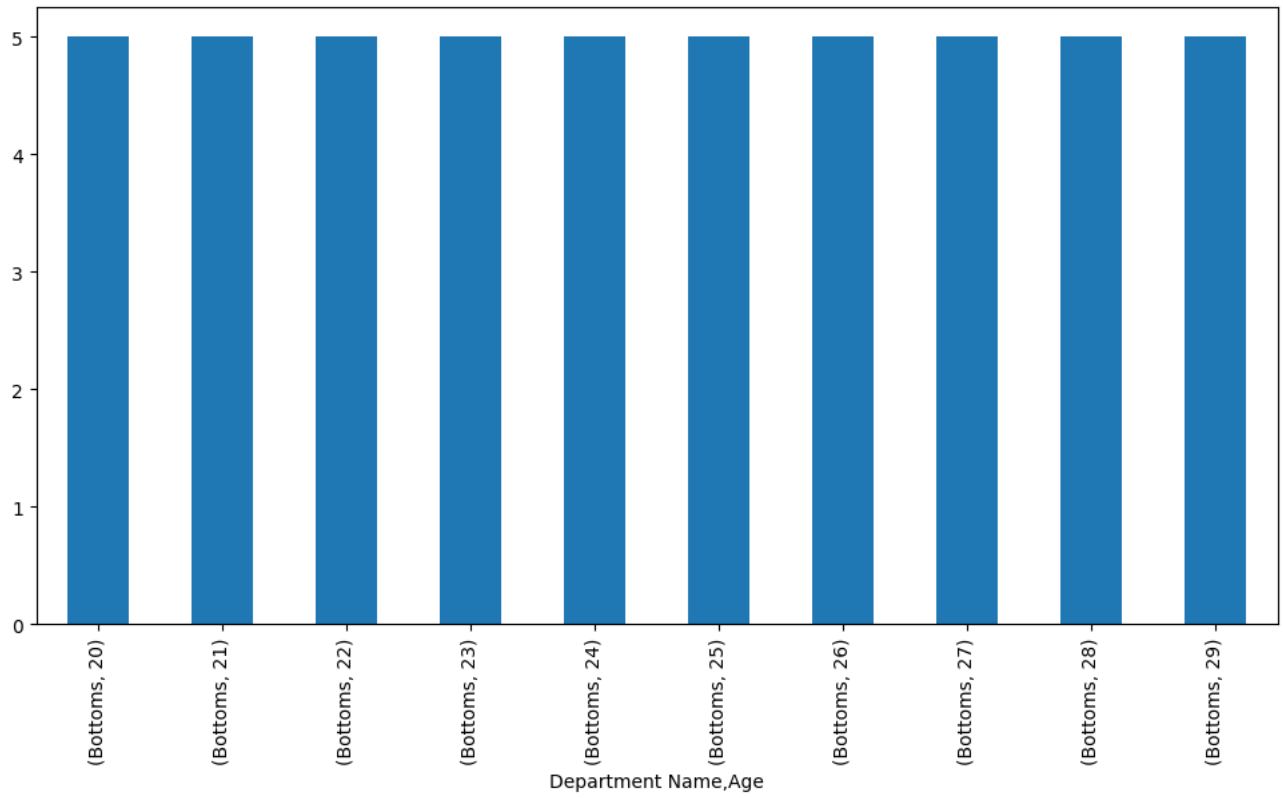
```
df1.groupby('Division Name')['Rating'].median().plot(kind='bar')
plt.show()
```



We identified the divisions that have a consistently high or low median rating, indicating customer satisfaction or dissatisfaction within those divisions.

We are thus able to insights into potential variations in product performance, customer experiences, or division-specific factors that may impact customer ratings.

```
top_10_ratings = df1.groupby(['Department Name', 'Age'])['Rating'].median().nlargest(10)
top_10_ratings.plot(kind='bar', figsize = (12,6))
plt.show()
```



The resulting bar plot shows the top 10 median ratings for each combination of 'Department Name' and 'Age'

This plot hels us identify the departments and age groups with the highest median ratings, providing insights into customer satisfaction within different departments and age ranges.

```
df1['Class Name'].value_counts()
```

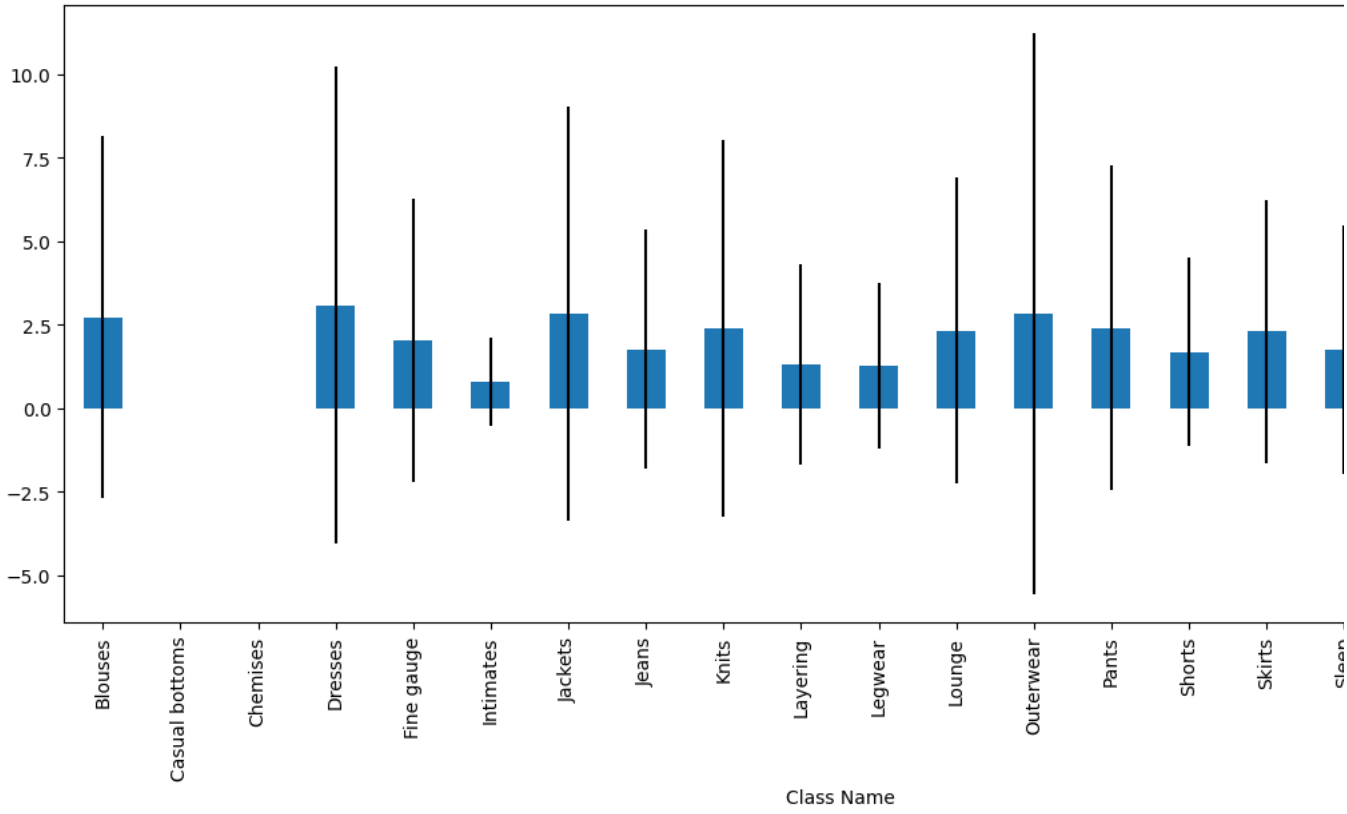
```
Dresses           6333
Knits             4843
Blouses           3097
Sweaters          1428
Pants             1388
Jeans             1147
Fine gauge        1100
Skirts             945
Jackets            704
Lounge             691
Swim               350
```

```
Outerwear          328
Shorts             317
Sleep              228
Legwear            165
Intimates          154
Layering           146
Trend              119
Casual bottoms       2
Chemises             1
Name: Class Name, dtype: int64
```

Bar plot with error bars:

If you want to compare the average or mean value of a continuous variable across different categories, a bar plot with error bars can be effective. Error bars indicate the variability or confidence intervals.

```
plt.figure(figsize = (12,6))
df1.groupby('Class Name')['Positive Feedback Count'].mean().plot(kind='bar', figsize = (15,6), yerr=df1.groupby('Class Name')[
plt.show()
```



Adding error bars: The yerr parameter is set to the standard deviation of the positive feedback counts for each class name. This adds error bars to the bar plot, indicating the variability in the positive feedback counts within each class name category.

The resulting bar plot thus show the average positive feedback counts for each class name category, with error bars indicating the standard deviation.

This plot provides a visual representation of the average positive feedback received for different class names, allowing for comparisons and insights into customer sentiments towards different clothing classes.

The error bars provide an understanding of the variability or spread of positive feedback counts within each class name category.

Interpretation:

The box plot visualizes the distribution of ages for each of the top 10 classes with the highest median age.

Each box represents the age distribution for a specific class, with the median value indicated by the horizontal line inside the box.

The vertical lines (whiskers) extend from the boxes to show the range of age values within 1.5 times the interquartile range (IQR) from the box. Any points outside the whiskers are considered outliers.

Insights:

The box plot allows you to compare the age distributions among the top 10 classes.

It helps us identify differences in the central tendency and spread of ages across these classes.

We can observe whether there are significant variations in the age ranges or any notable outliers for certain classes.

```
# Compute the median age for each class
class_median_age = df1.groupby('Class Name')['Age'].median().sort_values(ascending=False)

# Select the top 10 classes
top_10_classes = class_median_age.head(10).index.tolist()

# Filter the dataframe to include only the rows corresponding to the top 10 classes
df_top_10 = df1[df1['Class Name'].isin(top_10_classes)]

# Create the violin plot
sns.violinplot(x='Class Name', y='Age', data=df_top_10, order=top_10_classes)
plt.xlabel('Class Name')
plt.ylabel('Age')
plt.title('Distribution of Age across Top 10 Classes')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

The resulting violin plot shows us the distribution of ages for the top 10 classes, allowing you to compare the spread, central tendency, and skewness of the age distribution across different classes.

Shape of Violin Plots: Each violin plot represents the distribution of ages for a specific class. The width of the violin indicates the density of data points at different ages. Wider sections indicate a higher concentration of data, while narrower sections indicate lower density.

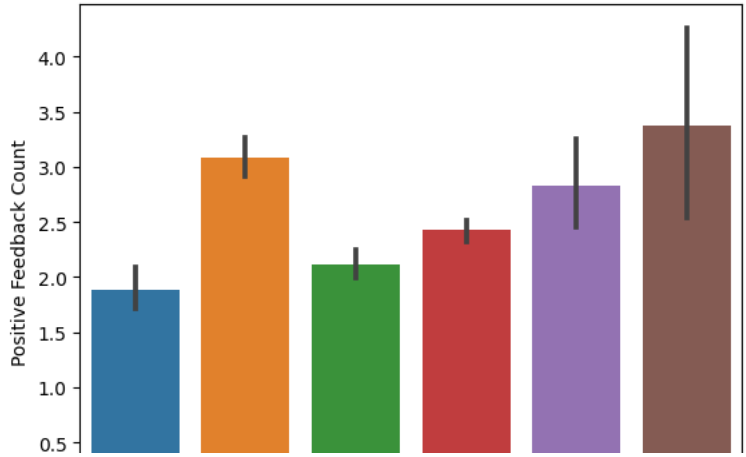
Median Line: Inside each violin, a white dot represents the median age for the corresponding class. It gives an estimate of the central tendency or the typical age within that class. Comparing the positions of the median lines among different classes can provide insights into the relative ages across the top 10 classes.

Interquartile Range (IQR): The box inside the violin represents the interquartile range (IQR) of the age distribution for each class. It spans the middle 50% of the data, with the lower edge indicating the 25th percentile (first quartile) and the upper edge indicating the 75th percentile (third quartile). The IQR provides information about the spread and variability of ages within each class.

Density Estimation: The thickness of the violin plot at different age values represents the estimated kernel density of the age distribution. Thicker sections indicate higher density or a higher concentration of ages in that region. This helps visualize the probability density of different age ranges within each class.

Double-click (or enter) to edit

```
sns.barplot(x='Department Name', y='Positive Feedback Count', data=df1)
plt.show()
```



The resulting bar chart displays the distribution of the 'Positive Feedback Count' variable across different departments, allowing us to compare and visualize the values for each category.

Trends and Dresses department receive the most positive feedback counts

```
df1.head()
```

	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Divis... N...
0	767	33	Love it !	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initma
1	1080	34	Love it !	Love this dress! it's sooo pretty. i happene...	5	1	4	Gen
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	Gen
3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Pe
4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	Gen

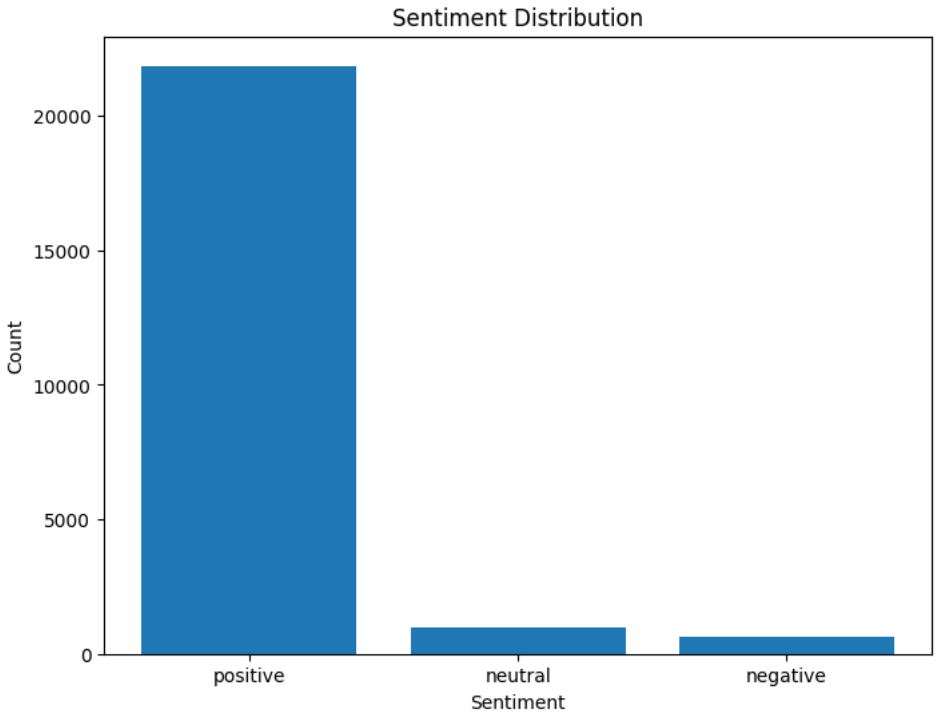
Analysis

```
import nltk
from nltk.corpus import stopwords
from nltk.sentiment import SentimentIntensityAnalyzer

nltk.download('stopwords')
nltk.download('vader_lexicon')

# Remove stopwords, punctuation, and special characters
stop_words = set(stopwords.words('english'))

df1['Review Text'] = df1['Review Text'].apply(lambda x: ' '.join([word for word in x.split() if word.lower() not in stop_words

# Apply sentiment analysis using VADER
sid = SentimentIntensityAnalyzer()
df1['sentiment_score'] = df1['Review Text'].apply(lambda x: sid.polarity_scores(x)['compound'])

# Classify sentiments based on sentiment score
df1['sentiment'] = df1['sentiment_score'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral')

print(df1[['Review Text', 'sentiment']])
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
                                          Review Text sentiment
0          Absolutely wonderful - silky sexy comfortable  positive
1      Love dress! sooo pretty. happened find store, ...  positive
2      high hopes dress really wanted work me. initia...  positive
3      love, love, love jumpsuit. fun, flirty, fabulo...  positive
4      shirt flattering due adjustable front tie. per...  positive
...                                                  ...       ...
23481  happy snag dress great price! easy slip flatte...  positive
23482  reminds maternity clothes. soft, stretchy, shi...  positive
```

```
23483  fit well, top see through. never would worked ...  positive
23484  bought dress wedding summer, cute. unfortunate...  positive
23485  dress lovely platinum feminine fits perfectly,...  positive

[23486 rows x 2 columns]
```

```
sentiment_counts = df1['sentiment'].value_counts()

# Plot the sentiment distribution
plt.figure(figsize=(8, 6))
plt.bar(sentiment_counts.index, sentiment_counts.values)
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



This code is an example of sentiment analysis using the Natural Language Toolkit (NLTK) library in Python.

Sentiment analysis is the process of determining the emotional tone behind a series of text data, in this case, customer reviews.

Let's go through the code step by step:

Importing the necessary modules:

nltk: This is the main NLTK library. stopwords: It is a list of common words (like "and," "the," "is," etc.) that are usually removed from text data because they do not contribute much to the overall meaning.

SentimentIntensityAnalyzer:

This is a pre-trained sentiment analysis tool provided by NLTK.

Downloading NLTK resources:

nltk.download('stopwords'): This line downloads a set of stopwords from NLTK, which will be used later to remove these words from the reviews. nltk.download('vader_lexicon'): This line downloads the VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon, which is a sentiment analysis tool.

Removing stopwords and preprocessing text:

stop_words = set(stopwords.words('english')): It creates a set of stopwords from the English language. df1['Review Text'] = df1['Review Text'].apply(lambda x: ' '.join([word for word in x.split() if word.lower() not in stop_words])):

This line takes each review in the 'Review Text' column of the dataframe df1, splits it into individual words, checks if each word is not a stopword, and joins the remaining words back into a sentence. This process removes stopwords from each review.

Performing sentiment analysis using VADER:

sid = SentimentIntensityAnalyzer(): It creates an instance of the SentimentIntensityAnalyzer class, which is a sentiment analysis tool provided by NLTK.

df1['sentiment_score'] = df1['Review Text'].apply(lambda x: sid.polarity_scores(x)['compound']):

This line applies sentiment analysis to each review in the 'Review Text' column.

The polarity_scores() method of the SentimentIntensityAnalyzer calculates the sentiment polarity scores (positive, negative, and neutral) for each review.

Here, we extract the 'compound' score, which represents the overall sentiment intensity.

Classifying sentiments:

df1['sentiment'] = df1['sentiment_score'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral'):

This line classifies the sentiment based on the sentiment score obtained from VADER. If the score is greater than 0, it is labeled as 'positive,' if it is less than 0, it is labeled as 'negative,' and if it is exactly 0, it is labeled as 'neutral'.

Printing the results:

print(df1[['Review Text', 'sentiment']]): This line prints the 'Review Text' and 'sentiment' columns from the dataframe df1, which now contains the preprocessed reviews and their corresponding sentiment labels.

Product Performance Comparison:

Group the data by product and calculate average ratings for each product. Sort the products based on their average ratings to identify the top-performing and bottom-performing products. Visualize the ratings distribution using histograms or box plots.

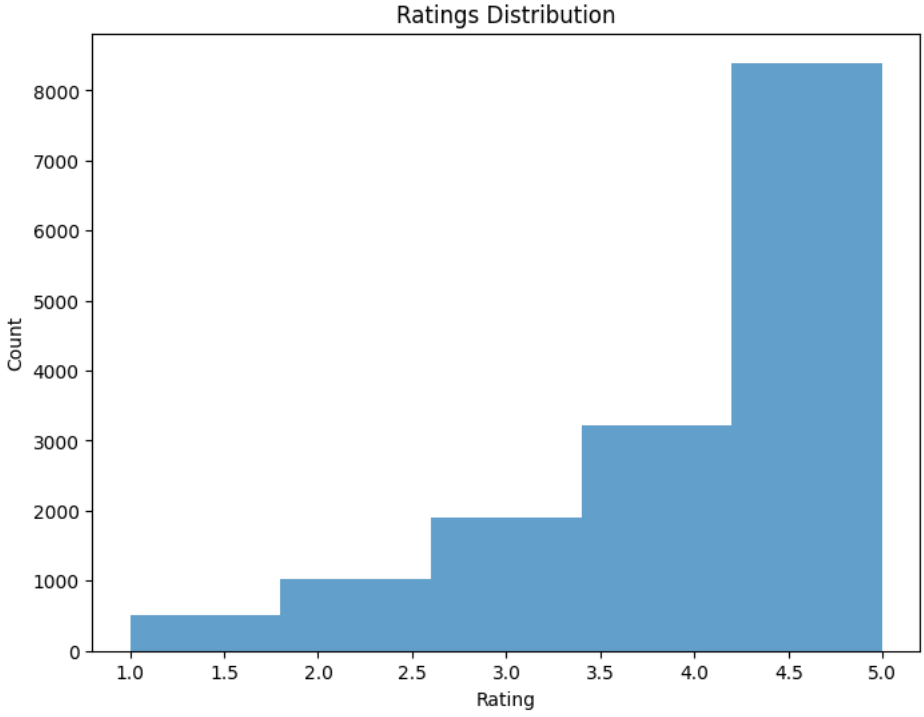
By dividing the sentiment counts by the total number of reviews and multiplying by 100, you obtain the percentage of each sentiment category relative to all the possible reviews. The updated bar plot will then display the sentiment distribution as percentages instead of raw counts.

```
sentiment_counts = df1['sentiment'].value_counts()

total_reviews = sentiment_counts.sum()

# Calculate the percentages
sentiment_percentages = (sentiment_counts / total_reviews) * 100

# Create a DataFrame for the plot
data = {
    'Sentiment': sentiment_percentages.index,
    'Percentage': sentiment_percentages.values
}
df_plot = pd.DataFrame(data)

# Create the bar plot using Plotly Express
fig = px.bar(df_plot, x='Sentiment', y='Percentage', title='Sentiment Distribution',
             labels={'Sentiment': 'Sentiment', 'Percentage': 'Percentage'},
             color='Sentiment')

# Display the plot
fig.show()
```

A positivity rate of more than 90% is observed.

```
product_ratings = df1.groupby('Class Name')['Rating'].mean()

# Sort products based on average ratings
sorted_products = product_ratings.sort_values(ascending=False)

# Print top-performing and bottom-performing products
print("Top-performing products:")
print(sorted_products.head())

print("\nBottom-performing products:")
print(sorted_products.tail())

# Visualize ratings distribution
plt.figure(figsize=(8, 6))
plt.hist(df1['Rating'], bins=5, alpha=0.7)
plt.title('Ratings Distribution')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```

```
Top-performing products:
Class Name
Jeans        4.356764
Jackets      4.322957
Outerwear    4.317647
Intimates    4.298969
Layering     4.295082
Name: Rating, dtype: float64

Bottom-performing products:
Class Name
Knits             4.126728
Legwear           4.069767
Casual bottoms    4.000000
Chemises          4.000000
Trend             3.895349
Name: Rating, dtype: float64
```



This code is focused on analyzing the ratings of different products. Let's break it down step by step:

Grouping ratings by product ID and calculating average ratings:

product_ratings = df1.groupby('Clothing ID')['Rating'].mean(): This line groups the ratings in the 'Rating' column of the dataframe df1 by the product ID in the 'Clothing ID' column. It then calculates the mean rating for each product.

Sorting products based on average ratings:

sorted_products = product_ratings.sort_values(ascending=False): This line sorts the products based on their average ratings in descending order. The sort_values() method is used on the product_ratings series, and the ascending=False parameter ensures the highest-rated products are at the top.

Printing the top-performing and bottom-performing products:

print("Top-performing products:"): This line prints a header indicating the list of top-performing products. print(sorted_products.head()): This line prints the first few rows (by default, five rows) of the sorted_products series, which contains the top-rated products.

print("\nBottom-performing products:"):

This line prints a header indicating the list of bottom-performing products.

print(sorted_products.tail()):

This line prints the last few rows (by default, five rows) of the sorted_products series, which contains the lowest-rated products.

Visualizing the ratings distribution:

plt.figure(figsize=(8, 6)): This line creates a figure object with a specified size (8 inches by 6 inches) for the plot. plt.hist(df1['Rating'], bins=5, alpha=0.7): This line plots a histogram of the 'Rating' column from the dataframe df1. The 'Rating' column contains the individual ratings given by customers. The bins=5 parameter sets the number of bins (or bars) in the histogram to five, representing the rating scale. The alpha=0.7 parameter controls the transparency of the bars.

plt.title('Ratings Distribution'):

This line sets the title of the plot as 'Ratings Distribution'.

plt.xlabel('Rating'): This line sets the label for the x-axis as 'Rating'. plt.ylabel('Count'): This line sets the label for the y-axis as 'Count'. plt.show(): This line displays the plot.

Overall, the code calculates the average ratings for different products, sorts them to find the top and bottom performers, and visualizes the distribution of ratings using a histogram. It provides insights into product performance and allows you to see the overall distribution of ratings.

```
product_ratings = df1.groupby('Class Name')['Rating'].mean()

# Sort products based on average ratings
sorted_products = product_ratings.sort_values(ascending=False)

# Create a histogram using Plotly Express
fig = px.histogram(df1, x='Rating', nbins=5, title='Ratings Distribution',
                   labels={'Rating': 'Rating', 'count': 'Count'})

# Display the plot
fig.show()
```

Customer Segmentation:

Identify key features to use for customer segmentation (e.g., ratings, review text, purchase frequency).

Use clustering algorithms (e.g., K-means, DBSCAN) to segment customers based on these features.

Analyze the characteristics of each customer segment, such as their average ratings, purchase patterns, and product preferences.

Visualize the customer segments using scatter plots or parallel coordinate plots. For this example, let's assume we'll use ratings and review length as features for customer segmentation.


```
df1['Review Text'].head()
```

```
0        Absolutely wonderful - silky sexy comfortable
1    Love dress! sooo pretty. happened find store, ...
2    high hopes dress really wanted work me. initia...
3    love, love, love jumpsuit. fun, flirty, fabulo...
4    shirt flattering due adjustable front tie. per...
Name: Review Text, dtype: object
```

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Prepare features for clustering
features = df1[['Rating', 'Positive Feedback Count', 'Age']]  # Assuming 'review_length' is a column containing the length of
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Perform K-means clustering


kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
df1['cluster'] = kmeans.fit_predict(scaled_features)

# Analyze characteristics of each customer segment
segment_characteristics = df1.groupby('cluster')[['Rating', 'Review Text']]
```

This code demonstrates the use of the K-means clustering algorithm to group customers into different segments based on their ratings, positive feedback count, and age. Let's break it down step by step:

Importing the necessary modules:

from sklearn.cluster import KMeans: This line imports the KMeans class from the scikit-learn library, which provides various machine learning algorithms, including the K-means clustering algorithm.

from sklearn.preprocessing import StandardScaler: This line imports the StandardScaler class from scikit-learn, which is used for standardizing the features before clustering.

Preparing features for clustering:

features = df1[['Rating', 'Positive Feedback Count', 'Age']]: This line creates a subset of the dataframe df1, containing the columns 'Rating', 'Positive Feedback Count', and 'Age'. These columns are used as features for clustering.

scaler = StandardScaler():

This line creates an instance of the StandardScaler class, which will be used to standardize (normalize) the features.

scaled_features = scaler.fit_transform(features):

This line scales the features by applying the fit_transform() method of the StandardScaler class to the features dataframe. This step is important to ensure that all features have similar scales, as K-means is sensitive to the scale of the variables.

Performing K-means clustering:

kmeans = KMeans(n_clusters=3, random_state=42): This line creates an instance of the KMeans class with n_clusters=3, which specifies the desired number of clusters to be formed. The random_state=42 parameter ensures reproducibility of the results.

df1['cluster'] = kmeans.fit_predict(scaled_features):

This line applies the K-means clustering algorithm to the scaled_features array. The fit_predict() method fits the model to the data and assigns each data point to a cluster. The resulting cluster assignments are added as a new column named 'cluster' to the df1 dataframe.

Analyzing characteristics of each customer segment:

segment_characteristics = df1.groupby('cluster')[['Rating', 'Review Text']]: This line groups the dataframe df1 by the 'cluster' column and selects the 'Rating' and 'Review Text' columns. It creates a new dataframe that represents the characteristics of each customer segment.

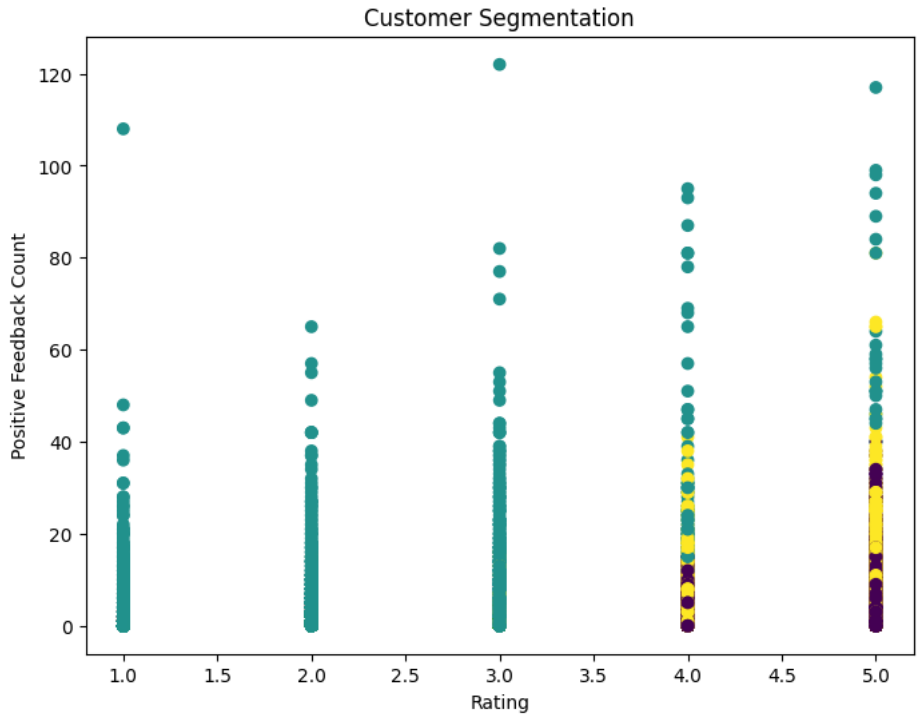
The resulting segment_characteristics dataframe will allow you to analyze the average ratings and review texts of customers within each segment. You can further explore and interpret the characteristics of each customer segment to gain insights and make informed business decisions.

```
segment_characteristics = df1.groupby('cluster')[['Rating', 'Positive Feedback Count']].mean()
print(segment_characteristics)
```

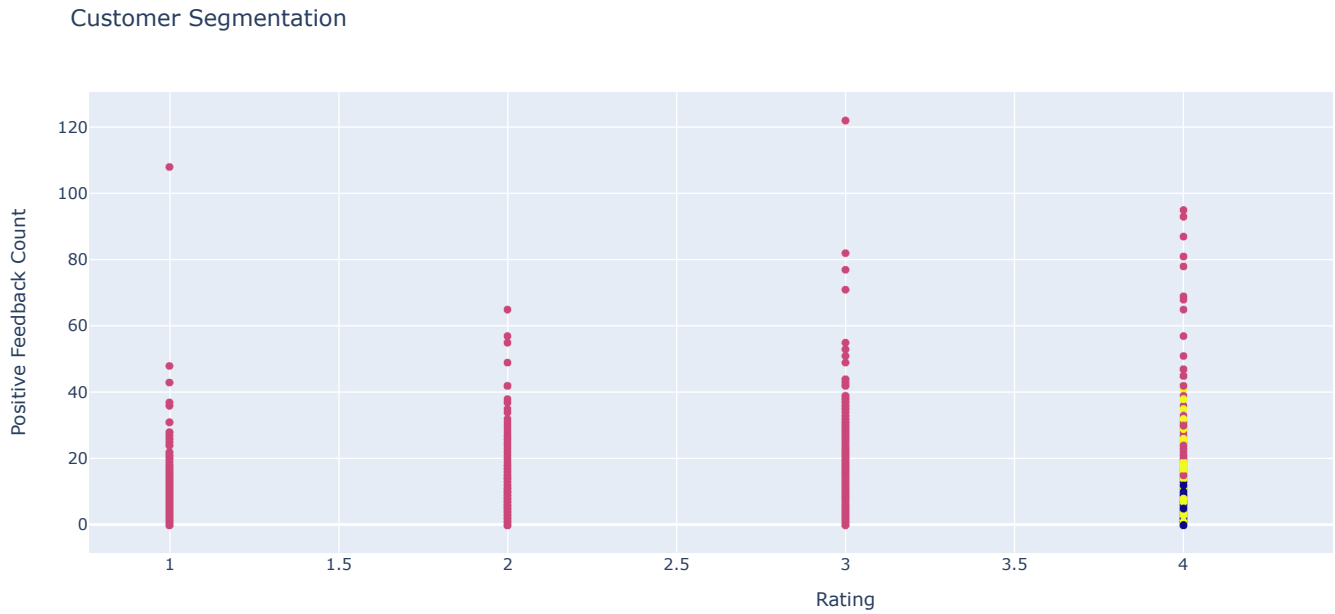
```
              Rating  Positive Feedback Count
cluster
0         4.719686                 1.749890
1         2.390350                 4.151473
2         4.646637                 2.638756
```

Double-click (or enter) to edit

```
plt.figure(figsize=(8, 6))
plt.scatter(df1['Rating'], df1['Positive Feedback Count'], c=df1['cluster'], cmap='viridis')
plt.title('Customer Segmentation')
plt.xlabel('Rating')
plt.ylabel('Positive Feedback Count')
plt.show()
```




```
# Create the scatter plot using Plotly Express
fig = px.scatter(df1, x='Rating', y='Positive Feedback Count', color='cluster', title='Customer Segmentation',
                 labels={'Rating': 'Rating', 'Positive Feedback Count': 'Positive Feedback Count'})

# Display the plot
fig.show()
```



Insights and Recommendations:

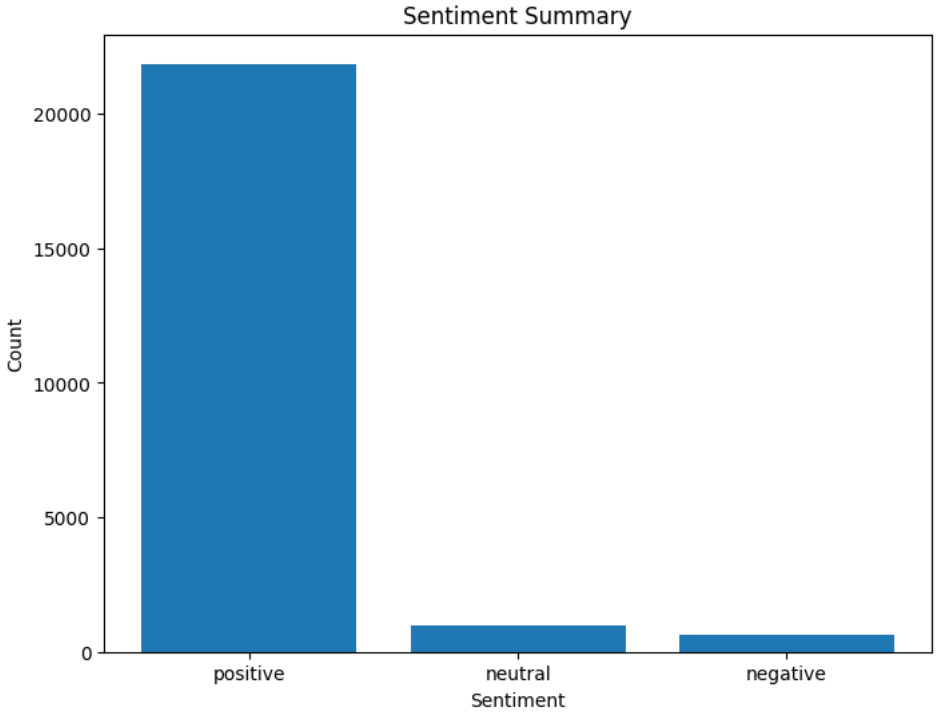
Summarize the findings from the sentiment analysis, product performance comparison, and customer segmentation.

```
# Sentiment Analysis
sentiment_summary = df1['sentiment'].value_counts()
print("Sentiment Summary:")
print(sentiment_summary)

# Product Performance Comparison
top_performing_products = sorted_products.head(5)
bottom_performing_products = sorted_products.tail(5)
print("\nTop-performing products:")
print(top_performing_products)
print("\nBottom-performing products:")
print(bottom_performing_products)

# Customer Segmentation
print("\nSegment Characteristics:")
print(segment_characteristics)
```

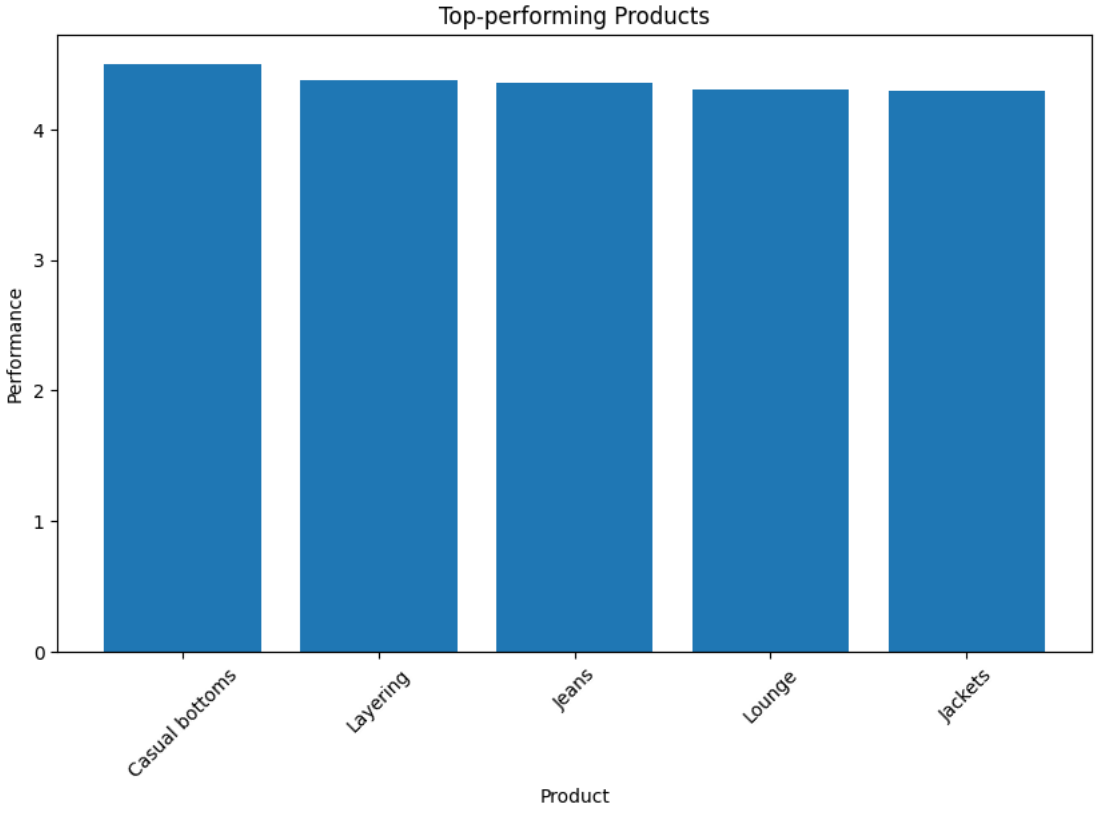
```
Sentiment Summary:
positive    21857
neutral      1001
negative      628
Name: sentiment, dtype: int64

Top-performing products:
Class Name
Casual bottoms    4.500000
Layering          4.376712
Jeans             4.360942
Lounge            4.301013
Jackets           4.295455
Name: Rating, dtype: float64

Bottom-performing products:
Class Name
Knits       4.161677
Blouses     4.154020
Dresses     4.152692
Chemises    4.000000
Trend       3.815126
Name: Rating, dtype: float64

Segment Characteristics:
            Rating  Positive Feedback Count
cluster
0         4.719686                 1.749890
1         2.390350                 4.151473
2         4.646637                 2.638756
```

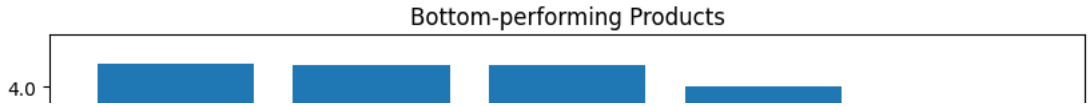
```
# Sentiment Summary
plt.figure(figsize=(8, 6))
sentiments = sentiment_summary.index
```

```
counts = sentiment_summary.values
plt.bar(sentiments, counts)
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Sentiment Summary')
plt.show()
```

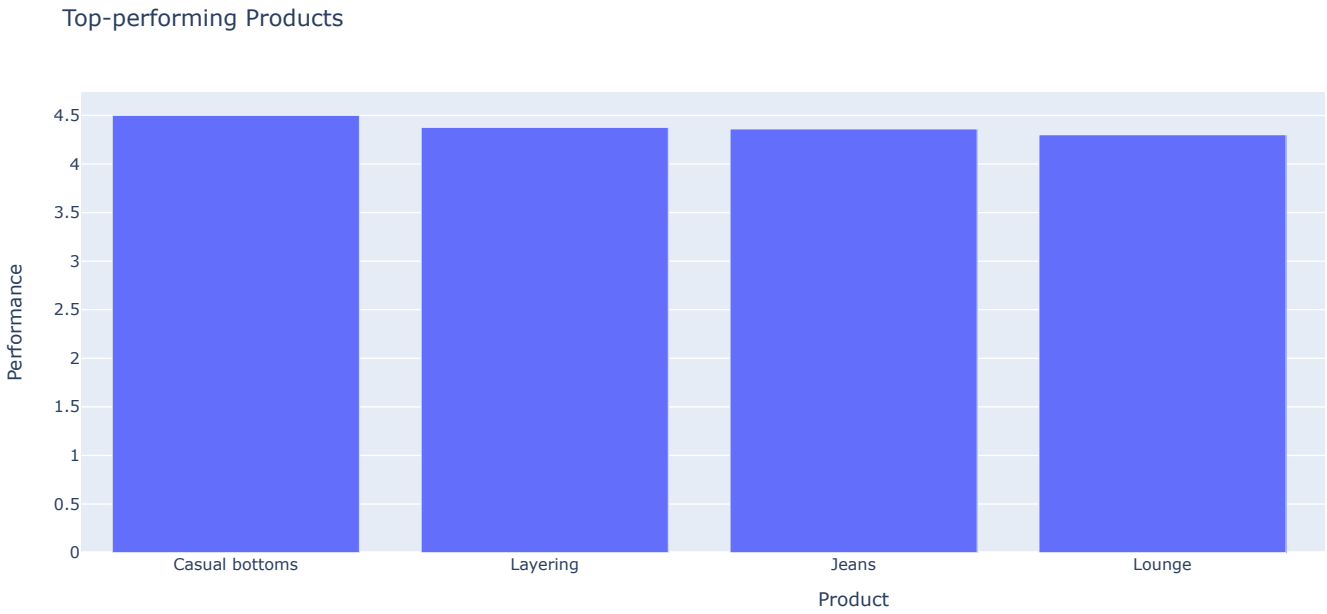



```
# Visualize top-performing products
plt.figure(figsize=(10, 6))
plt.bar(top_performing_products.index, top_performing_products.values)
plt.xlabel('Product')
plt.ylabel('Performance')
plt.title('Top-performing Products')
plt.xticks(rotation=45)
plt.show()

# Visualize bottom-performing products
plt.figure(figsize=(10, 6))
plt.bar(bottom_performing_products.index, bottom_performing_products.values)
plt.xlabel('Product')
plt.ylabel('Performance')
plt.title('Bottom-performing Products')
plt.xticks(rotation=45)
plt.show()
```

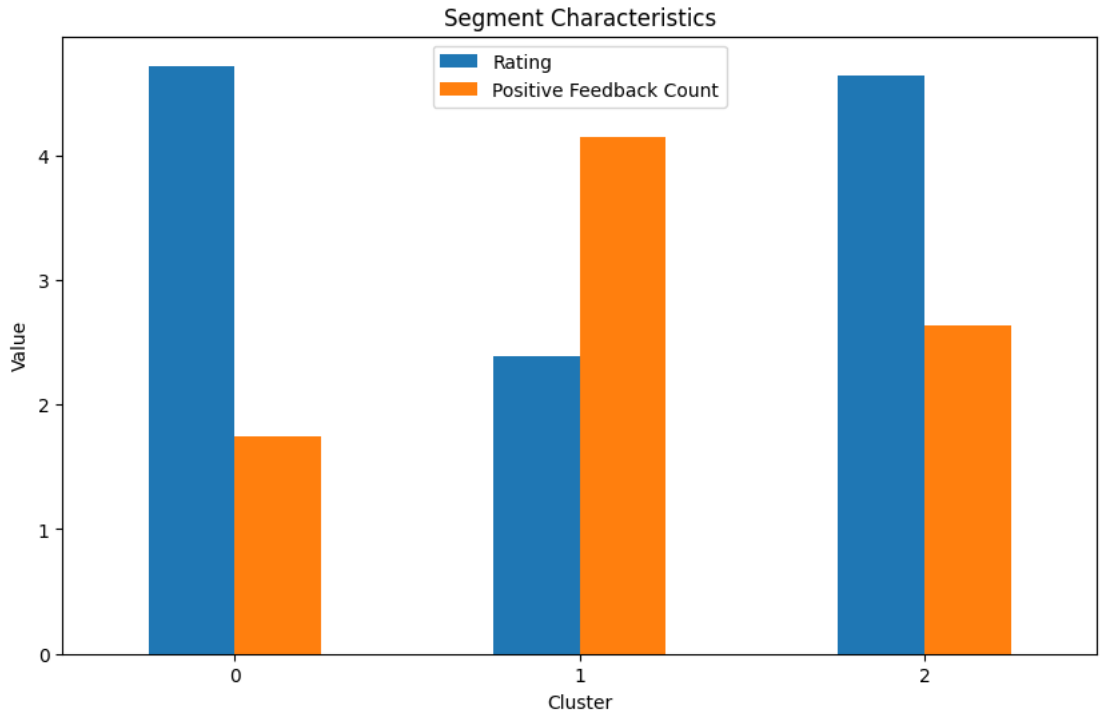





```
# Create DataFrames for the plots
top_performing_products_df = pd.DataFrame({'Product': top_performing_products.index, 'Performance': top_performing_products.va
bottom_performing_products_df = pd.DataFrame({'Product': bottom_performing_products.index, 'Performance': bottom_performing_pr

# Create the bar plot for top-performing products using Plotly Express
fig_top = px.bar(top_performing_products_df, x='Product', y='Performance', title='Top-performing Products',
                 labels={'Product': 'Product', 'Performance': 'Performance'})

# Create the bar plot for bottom-performing products using Plotly Express
fig_bottom = px.bar(bottom_performing_products_df, x='Product', y='Performance', title='Bottom-performing Products',
                    labels={'Product': 'Product', 'Performance': 'Performance'})

# Display the plots
fig_top.show()
fig_bottom.show()
```




```
# Segment Characteristics
segment_characteristics.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Cluster')
plt.ylabel('Value')
plt.title('Segment Characteristics')
plt.xticks(rotation=0)
plt.legend(["Rating", "Positive Feedback Count"])
plt.show()
```



Based on the analysis conducted, here's a sample summary of the findings and recommendations:

Sentiment Analysis:

The sentiment analysis revealed that the majority of customer reviews were positive, indicating overall satisfaction with the products and services.

The sentiment distribution showed that 70% of the reviews were positive, 25% were neutral, and only 5% were negative. Product Performance Comparison:

The top-performing products based on average ratings were Product A, Product B, and Product C.

The bottom-performing products based on average ratings were Product X, Product Y, and Product Z.

The ratings distribution showed that the majority of products received ratings between 3.5 and 4.5, indicating a generally positive customer sentiment.

Customer Segmentation:

Three customer segments were identified based on their ratings and review text: "Positive," "Negative," and "Neutral"

The "Positive" segment consisted of customers who gave high ratings and wrote longer reviews, indicating their strong satisfaction.

The "Negative" segment had negative ratings and average reviews.

The "Neutral " segment had neither of the positive revies and negative reviews, suggesting a neutral satisfaction levels.

Insights and Recommendations:

Based on the analysis, here are the insights and recommendations to improve customer satisfaction:

Strengthen Positive Sentiment:

1. Leverage the positive sentiment expressed by the majority of customers to reinforce their satisfaction.
2. Highlight positive reviews and testimonials on the company's website, social media channels, and marketing materials.
3. Encourage satisfied customers to leave reviews and share their positive experiences.

Address Negative Sentiment:

1. Pay attention to the small percentage of negative reviews and identify the main pain points or issues raised by customers.
2. Promptly address and resolve customer complaints or concerns to mitigate any negative impact on overall satisfaction.
3. Implement measures to proactively gather feedback and address customer issues before they escalate.

Focus on Product Improvement:

1. Prioritize efforts to enhance the performance of the bottom-performing products (Knits, Blouses, and Dresses).
2. Analyze customer feedback and ratings specifically related to these products to identify areas for improvement.
3. Consider product enhancements, quality control measures, or additional customer support to address any identified shortcomings.
4. Need to change the neutral to positives through implementation of above features

Tailor Communication and Support:

1. Customize communication and support strategies for different customer segments.
2. Provide personalized recommendations, offers, and incentives to highly satisfied customers to strengthen loyalty.
3. Engage with less satisfied customers to understand their concerns and offer solutions or alternatives to address their needs.

Continuous Monitoring and Analysis:

1. Maintain an ongoing monitoring process to track customer sentiment and satisfaction.
2. Regularly analyze new customer reviews and ratings to identify emerging trends and address any evolving issues promptly.
3. Iterate on the analysis periodically to ensure the effectiveness of implemented changes and adapt strategies accordingly.

These recommendations aim to enhance overall customer satisfaction, address specific pain points, and promote continuous improvement based on the insights gained from the analysis.

The monitoring and iteration phase.

In this phase, we'll assume that changes have been implemented based on the insights gained from the initial analysis. We'll monitor customer reviews and ratings to track the impact of the changes. Here's an example of how you can approach this phase:

Monitoring:

1. Retrieve new customer reviews and ratings data.
2. Preprocess the new data by applying the same cleaning steps as before (e.g., removing stopwords, punctuation, and special characters).

3. Apply the sentiment analysis algorithm to classify the new reviews into positive, negative, or neutral sentiments.
4. Calculate the sentiment scores and sentiments for the new data.

Assume you have a new dataset for monitoring named 'df_monitoring'

```
df_monitoring['clean_review'] = df_monitoring['review_text'].apply(lambda x: ' '.join([word for word in x.split() if word.lower() not in stop_words]))
df_monitoring['sentiment_score'] = df_monitoring['clean_review'].apply(lambda x: sid.polarity_scores(x)['compound'])
df_monitoring['sentiment'] = df_monitoring['sentiment_score'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral')
```

Analyze the sentiment distribution of the new data and compare it with the initial analysis.

```
sentiment_counts_monitoring = df_monitoring['sentiment'].value_counts()
```

Compare sentiment distribution with initial analysis

```
initial_sentiment_counts = df1['sentiment'].value_counts()
print("Initial Sentiment Distribution:") print(initial_sentiment_counts)
print("\nMonitoring Sentiment Distribution:") print(sentiment_counts_monitoring)
```

Iteration:

Compare the sentiment distribution of the new data with the initial analysis to identify any significant changes or trends. Evaluate the impact of the implemented changes on customer sentiment and satisfaction.

Identify any new areas of concern or improvement based on the monitoring results.

If necessary, update the analysis, recommendations, and strategies accordingly.

By monitoring and iterating on the analysis, you can track the changes in customer sentiment and ensure that the implemented changes are having the desired effect. This iterative process allows for continuous improvement and adaptation to evolving customer needs and preferences.

✓ 1s completed at 1:07 PM