

```

1 import pandas as pd
2 import numpy as np
3 dataset = pd.read_excel("../Data/Womens Clothing Reviews Data New.xlsx")
4 dataset

```

	Product_ID	Category	SubCategory1	SubCategory2	Location	Channel	Customer_Age	Rating	Recom
0	767	Initmates	Intimate	Intimates	Mumbai	Mobile	33	4	
1	1080	General	Dresses	Dresses	Bangalore	Mobile	34	5	
2	1077	General	Dresses	Dresses	Gurgaon	Mobile	60	3	
3	1049	General Petite	Bottoms	Pants	Chennai	Web	50	5	
4	847	General	Tops	Blouses	Bangalore	Web	47	5	

```

1 import nltk
2 from nltk.sentiment.vader import SentimentIntensityAnalyzer
3
4 # Download necessary resources for SentimentIntensityAnalyzer
5 nltk.download('vader_lexicon')

```

```

[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\pholl\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
True

```

```
1 dataset.columns
```

```

Index(['Product_ID', 'Category', 'SubCategory1', 'SubCategory2', 'Location',
      'Channel', 'Customer_Age', 'Rating', 'Recommend_Flag', 'Merged_Review'],
      dtype='object')

```

```

1 from nltk.sentiment import SentimentIntensityAnalyzer
2 import pandas as pd
3
4 # Assuming you have loaded your dataset into the variable 'dataset'
5
6 # Initialize the SentimentIntensityAnalyzer
7 analyzer = SentimentIntensityAnalyzer()
8
9 # Function to calculate sentiment scores using SentimentIntensityAnalyzer
10 def get_sentiment_score(row):
11     sentiment_score = analyzer.polarity_scores(row['Merged_Review'])
12     return sentiment_score['compound']
13
14 # Apply the sentiment analysis function to each row of the DataFrame
15 dataset['sentiment_score'] = dataset.apply(get_sentiment_score, axis=1)
16
17 # Function to get sentiment category based on the compound score
18 def get_sentiment_category(compound_score):
19     if compound_score > 0.1:
20         return 'Positive'
21     elif compound_score < -0.1:
22         return 'Negative'
23     else:
24         return 'Neutral'
25
26 # Apply the sentiment analysis function to each row of the DataFrame
27 dataset['sentiment_category'] = dataset['sentiment_score'].apply(get_sentiment_category)
28
29 # Display the DataFrame with the added 'sentiment_category' column
30 print(dataset[['Merged_Review', 'sentiment_category']])

```

```

      Merged_Review sentiment_category
0      Absolutely wonderful - silky and sexy and com... Positive
1      Love this dress! it's sooo pretty. i happen... Positive
2      Some major design flaws I had such high hopes ... Positive
3      My favorite buy! I love, love, love this jumps... Positive
4      Flattering shirt This shirt is very flattering... Positive
...      ...
22637 Great dress for many occasions I was very happ... Positive
22638 Wish it was made of cotton It reminds me of ma... Positive
22639 Cute, but see through This fit well, but the t... Positive
22640 Very cute dress, perfect for summer parties an... Positive
22641 Please make more like this one! This dress in ... Positive

```

[22642 rows x 2 columns]

```
1 dataset.head()
```

	Product_ID	Category	SubCategory1	SubCategory2	Location	Channel	Customer_Age	Rating	Recommend
0	767	Initmates	Intimate	Intimates	Mumbai	Mobile	33	4	
1	1080	General	Dresses	Dresses	Bangalore	Mobile	34	5	
2	1077	General	Dresses	Dresses	Gurgaon	Mobile	60	3	

```

1 # Calculate the total positive, negative, and neutral sentiments using value_counts()
2 sentiment_counts = dataset['sentiment_category'].value_counts()
3
4 # Display the total positive, negative, and neutral sentiments
5 print(sentiment_counts)

```

```

Positive    21089
Negative     1208
Neutral       345
Name: sentiment_category, dtype: int64

```

```
1 dataset.columns
```

```

Index(['Product_ID', 'Category', 'SubCategory1', 'SubCategory2', 'Location',
      'Channel', 'Customer_Age', 'Rating', 'Recommend_Flag', 'Merged_Review',
      'sentiment_score', 'sentiment_category'],
      dtype='object')

1 # import plotly.express as px
2
3 # # Assuming sentiment_counts is a DataFrame containing the sentiment counts
4 # # Create a bar plot using Plotly Express to visualize the distribution of sentiments
5 # fig_bar = px.bar(sentiment_counts, x='sentiment_category', y='count', color='sentiment_category',
6 #                  labels={'sentiment_category': 'Sentiment', 'count': 'Count'},
7 #                  title='Sentiment Distribution in the Dataset')
8
9 # # Show the bar plot
10 # fig_bar.show()
11
12 # # Create a pie chart using Plotly Express to visualize the distribution of sentiments
13 # fig_pie = px.pie(sentiment_counts, values='count', names='sentiment_category',
14 #                  title='Sentiment Distribution in the Dataset')
15
16 # # Show the pie chart
17 # fig_pie.show()
18

1 # import matplotlib.pyplot as plt
2 # import matplotlib.gridspec as gridspec
3 # import seaborn as sns
4 # import plotly.express as px
5 # # Calculate the total positive, negative, and neutral sentiments using value_counts()
6 # sentiment_counts = dataset['sentiment_category'].value_counts().reset_index()
7
8 # # Create a bar plot using Plotly Express to visualize the distribution of sentiments
9 # fig_bar = px.bar(sentiment_counts, x='index', y='sentiment_category', color='index',
10 #                  labels={'index': 'Sentiment', 'sentiment_category': 'Count'},
11 #                  title='Sentiment Distribution in the Dataset')
12
13 # # Create a pie chart using Plotly Express to visualize the distribution of sentiments
14 # fig_pie = px.pie(sentiment_counts, values='sentiment_category', names='index',
15 #                  title='Sentiment Distribution in the Dataset')
16
17 # # Display both the bar plot and pie chart side by side
18 # fig_bar.show()
19 # fig_pie.show()

1 dataset.Rating.value_counts()

5    12541
4     4908
3     2823
2     1549
1       821
Name: Rating, dtype: int64

1 pd.crosstab(dataset.sentiment_category, dataset.Rating)


```

	Rating	1	2	3	4	5
sentiment_category						
Negative		286	330	348	147	97
Neutral		58	87	112	53	35
Positive		477	1132	2363	4708	12409

```

1 """ Divide the data into three groups on the basis of sentiments like positive, negative and neutral """
2 dataset_neg = dataset[(dataset.sentiment_category=='Negative')]
3 dataset_pos = dataset[(dataset.sentiment_category=='Positive')]
4 dataset_neu = dataset[(dataset.sentiment_category=='Neutral')]

1 dataset.columns

Index(['Product_ID', 'Category', 'SubCategory1', 'SubCategory2', 'Location',
      'Channel', 'Customer_Age', 'Rating', 'Recommend_Flag', 'Merged_Review',

```

```

'sentiment_score', 'sentiment_category'],
dtype='object')

1 """ plit the data into train & Test where y variable is Rating """
2 from sklearn.model_selection import train_test_split
3 ## X-variable is Merged_Review and y-variable is Rating
4 # define X and y
5 X = dataset.Merged_Review
6 y = dataset.Rating
7
8 # split the new DataFrame into training and testing sets
9 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
10 print(X_train.shape)
11 print(X_test.shape)
12 print(y_train.shape)
13 print(y_test.shape)

(16981,)
(5661,)
(16981,)
(5661,)

1 """ Split the data into train & Test for positive sentiments and negative sentiments """
2 # create a new DataFrame that only contains the 5 Rating and 1-Rating reviews
3 # define X and y
4 X2 = dataset_pos.Merged_Review
5 y2 = dataset_pos.Rating
6
7 # split the new DataFrame into training and testing sets
8 X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, random_state=1)
9 print(X2_train.shape)
10 print(X2_test.shape)
11 print(y2_train.shape)
12 print(y2_test.shape)

(15816,)
(5273,)
(15816,)
(5273,)

1 # define X and y
2 X1 = dataset_neg.Merged_Review
3 y1 = dataset_neg.Rating
4
5 # split the new DataFrame into training and testing sets
6 X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, random_state=1)
7 print(X1_train.shape)
8 print(X1_test.shape)
9 print(y1_train.shape)
10 print(y1_test.shape)

(906,)
(302,)
(906,)
(302,)

1 dataset_pos.shape

(21089, 12)

1 dataset_neg.shape

(1208, 12)

1 """ Creating user defined functions for clean the text and pre-process the data """
2 import re
3 from nltk.tokenize import word_tokenize
4 def clean_text(Merged_Review):
5     Merged_Review = Merged_Review.lower()
6     Merged_Review = Merged_Review.strip()
7     Merged_Review = re.sub(r' +', ' ', Merged_Review)
8     Merged_Review = re.sub(r"[-()\"#/@;:{}'`+=~|.|?*%&€!_><|, '0-9]", "", Merged_Review)
9     Merged_Review = Merged_Review.replace('wat', 'what').replace('txts', 'texts').replace('vry', 'very').replace('gud', 'good').replace('n
10     return Merged_Review

```

```

1 import nltk
2 nltk.download('stopwords')
3
4 sw = list(set(nltk.corpus.stopwords.words('english')))
5 print(sw)

['will', 'wasn', 'more', 'them', 'only', 'because', 'in', 'where', 'our', 'any', "it's", 'no', 'off', "isn't", 'so', 'at', "weren't", 'c
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\pholl\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!

```

```

1 import string
2 import nltk
3 from nltk.corpus import stopwords
4 from nltk.tokenize import word_tokenize
5 from nltk.stem import WordNetLemmatizer
6
7 # Download NLTK resources
8 nltk.download('punkt')
9 nltk.download('wordnet')
10 nltk.download('stopwords')
11
12 # Create a set of English stopwords
13 stop = set(stopwords.words('english'))
14
15 # Initialize the WordNetLemmatizer
16 lemmatizer = WordNetLemmatizer()
17
18 def pre_process(Merged_Review):
19     Merged_Review = Merged_Review.str.replace('/', '') # Replacing the / with none
20     Merged_Review = Merged_Review.apply(lambda x: " ".join(word for word in word_tokenize(x) if word not in stop)) # Removing stop words
21     Merged_Review = Merged_Review.apply(lambda x: " ".join(lemmatizer.lemmatize(word) for word in word_tokenize(x))) # Lemmatization
22     return Merged_Review

```

```

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

```

```

1 X_train = X_train.apply(lambda x: clean_text(x))
2 X_test = X_test.apply(lambda x: clean_text(x))

```

```

1 X_train=pre_process(X_train)
2 X_test=pre_process(X_test)

```

Clean the text and pre-process the data for positive sentiments

```

1 X2_train = X2_train.apply(lambda x: clean_text(x))
2 X2_test = X2_test.apply(lambda x: clean_text(x))

```

```

1 X2_train=pre_process(X2_train)
2 X2_test=pre_process(X2_test)

```

```

1 X2_train

```

```

16673    gorgeous design poor fit suit absolutely gorgeo...
6499     knockoff dress gorgeous flattering oood qualit...
8615     cozy soft shirt love shirt subdued color casua...
17057    classy tried store ended size almost could gon...
8238     dont try arent prepared buy love jacket great ...
...
11754             love love shirt fit great cute must fall
18557    im thrilled fabric little heavier fit really w...
5567     cute run large top run large loose fitting mid...
13057    wanted love like feel stretch softness dress c...
253     pretty denim jacket perfect jacket shirt tee d...
Name: Merged_Review, Length: 15816, dtype: object

```

Clean the text and pre-process the data for negative sentiments

```

1 X1_train = X1_train.apply(lambda x: clean_text(x))
2 X1_test = X1_test.apply(lambda x: clean_text(x))

1 X1_train=pre_process(X1_train)
2 X1_test=pre_process(X1_test)

1 X1_train

13594    doesnt look like photo arrived week disappoint...
13390    didnt work im shorter side spectrum usually we...
3903     product hole pocket area soft comfortable jack...
18013    returning thought dress would perfect th birth...
15187    returned immediately first product arrived sma...
      ...
13868    odd fit dress odd fit loose top pointy dart ti...
17329    fabric comfortable pretty blouse polyester fab...
20594    underwhelmed stalked blouse price dropped poun...
4369     white background love top everything previous ...
19826    oh dear elastic waistband ruined skirt meit ma...
Name: Merged_Review, Length: 906, dtype: object

```

Vectorization (Count, Tfidf) for positive sentiments

```

1 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, HashingVectorizer, TfidfTransformer
2 #Train
3 count_vect2 = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}',
4                               ngram_range=(1, 1 ),
5                               min_df=5,
6                               encoding='latin-1' ,
7                               max_features=800)
8 xtrain2_count = count_vect2.fit_transform(X2_train)

1 xtrain2_count

<15816x800 sparse matrix of type '<class 'numpy.int64'>'
  with 363350 stored elements in Compressed Sparse Row format>

```

Vectorization (Count, Tfidf) for negative sentiments

```

1 #Train
2 count_vect1 = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}',
3                               ngram_range=(1, 1 ),
4                               min_df=5,
5                               encoding='latin-1' ,
6                               max_features=800)
7 xtrain1_count = count_vect1.fit_transform(X1_train)

1 xtrain1_count

<906x800 sparse matrix of type '<class 'numpy.int64'>'
  with 19773 stored elements in Compressed Sparse Row format>

```

View the document term metrics for positive sentiments

```

1 dtm=xtrain2_count.toarray()

1 feature_names = count_vect2.get_feature_names_out()
2 feature_names

array(['able', 'absolutely', 'across', 'actually', 'add', 'added',
      'addition', 'adorable', 'adore', 'ag', 'ago', 'agree', 'airy',
      'aline', 'almost', 'along', 'already', 'also', 'although',
      'always', 'amazing', 'amount', 'anighthing', 'ankle', 'another',
      'anyone', 'anyway', 'appears', 'appropriate', 'area', 'arent',
      'arm', 'armhole', 'around', 'arrived', 'athletic', 'attractive',
      'available', 'average', 'away', 'awesome', 'awkward', 'b', 'baby',
      'back', 'bad', 'bag', 'baggy', 'band', 'barely', 'based', 'basic',

```

'bc', 'beach', 'beading', 'beautiful', 'beautifully', 'beauty',
'believe', 'belly', 'belt', 'best', 'better', 'big', 'bigger',
'billyow', 'bit', 'black', 'blazer', 'blouse', 'blue', 'bodice',
'body', 'boot', 'booty', 'bottom', 'bought', 'boxy', 'bra',
'brand', 'breezy', 'bright', 'broad', 'brown', 'build', 'bulky',
'bust', 'busty', 'butt', 'button', 'buy', 'buying', 'c', 'came',
'cami', 'camisole', 'cant', 'cardigan', 'care', 'case', 'casual',
'caught', 'chance', 'cheap', 'chest', 'chested', 'chic', 'classic',
'classy', 'clean', 'clingy', 'close', 'closet', 'clothes',
'clothing', 'coat', 'cold', 'collar', 'color', 'colored',
'combination', 'come', 'comfort', 'comfortable', 'comfy',
'complaint', 'completely', 'compliment', 'considering', 'cool',
'cooler', 'coral', 'cotton', 'could', 'couldnt', 'couple', 'cover',
'coverage', 'cozy', 'cream', 'crop', 'cropped', 'cuff', 'cup',
'curve', 'curvy', 'cut', 'cute', 'cuter', 'dark', 'darker',
'daughter', 'day', 'dd', 'deal', 'decided', 'deep', 'definitely',
'delicate', 'denim', 'depending', 'description', 'design',
'despite', 'detail', 'detailing', 'didnt', 'different',
'difficult', 'dinner', 'disappointed', 'doesnt', 'done', 'dont',
'dot', 'drape', 'dream', 'dress', 'dressed', 'dressing', 'dressy',
'dry', 'due', 'easily', 'easy', 'edge', 'either', 'elastic',
'elegant', 'else', 'embroidery', 'end', 'ended', 'enough',
'especially', 'etc', 'even', 'evening', 'event', 'ever', 'every',
'everyday', 'everyone', 'everything', 'everywhere', 'exactly',
'excellent', 'except', 'exchange', 'excited', 'expect', 'expected',
'expecting', 'expensive', 'extra', 'extremely', 'eye', 'fabric',
'fabulous', 'fact', 'fairly', 'fall', 'fan', 'fantastic', 'far',
'favorite', 'feel', 'feeling', 'fell', 'felt', 'feminine',
'figure', 'finally', 'find', 'finding', 'fine', 'first', 'fit',
'fitted', 'fitting', 'flare', 'flat', 'flattering', 'flaw',
'floral', 'flow', 'flower', 'flowing', 'flowy', 'foot',
'forgiving', 'form', 'forward', 'found', 'frame', 'friend',
'front', 'frumpy', 'full', 'fun', 'gave', 'generally', 'get',
'getting', 'girl', 'give', 'glad', 'glove', 'go', 'going', 'gold',
'gone', 'good', 'gorgeous', 'got', 'gotten', 'gray', 'great',
'green', 'grey', 'guess', 'half', 'hand', 'hang', 'hanging',
'happy', 'hard', 'havent', 'heavier', 'heavy', 'heel', 'height',
'help', 'hem', 'hide', 'high', 'higher', 'highly', 'hip', 'hit',
'hold', 'hole', 'holiday', 'home', 'hope', 'hoped', 'hoping',
'hot', 'hour', 'hourglass', 'house', 'however', 'hug', 'huge',
'hung', 'husband', 'id', 'idea', 'ill', 'im', 'imagine',
'immediately', 'inch', 'incredibly', 'inside', 'instead',
'interesting', 'isnt', 'issue', 'itchy', 'item', 'ive', 'ivory',
'jacket', 'jean', 'jumpsuit', 'justice', 'keep', 'keeper',
'keeping', 'kept', 'kind', 'knee', 'knew', 'knit', 'know', 'l',
'lace', 'lady', 'large', 'larger', 'last', 'lay', 'layer',
'layered', 'layering', 'lb', 'le', 'least', 'leather', 'left',
'leg', 'legging', 'length', 'let', 'light', 'lighter',
'lightweight', 'like', 'liked', 'line', 'lined', 'linen', 'lining',
'little', 'live', 'local', 'long', 'longer', 'look', 'looked'

```
1 dtm1=pd.DataFrame(dtm, columns = count_vect2.get_feature_names_out())
2 dtm1
```

	able	absolutely	across	actually	add	added	addition	adorable	adore	ag	...	xd	xl	xx	y
0	0	1	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	1	0	0	0	0	0	0	0	...	0	0	0	
...	
15811	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
15812	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
15813	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
15814	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
15815	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

15816 rows × 800 columns

```
1 dtm1.apply(sum)

able      296
absolutely 638
across    243
```

```

actually      475
add           442
...
yes           172
yet           482
youre        282
zip           109
zipper       278
Length: 800, dtype: int64

```

View the document term metrics for negative sentiments

```

1 dtm4=xtrain1_count.toarray()
2 print(count_vect1.get_feature_names_out())

```

```

['able' 'absolutely' 'across' 'actually' 'add' 'added' 'adorable' 'afraid'
'ago' 'agree' 'ala' 'aline' 'almost' 'along' 'already' 'also' 'although'
'always' 'amazing' 'amount' 'anighthing' 'ankle' 'another' 'anyone'
'anyway' 'apart' 'appears' 'area' 'arent' 'arm' 'armhole' 'armpit'
'around' 'arrived' 'assumed' 'attached' 'available' 'average' 'avoid'
'away' 'awesome' 'awful' 'awkward' 'b' 'back' 'backside' 'bad' 'bag'
'baggy' 'band' 'barely' 'based' 'basically' 'bead' 'beautiful' 'behind'
'belt' 'best' 'better' 'beware' 'big' 'bigger' 'bit' 'black' 'blah'
'blazer' 'blouse' 'blue' 'bodice' 'body' 'boob' 'boot' 'boring' 'bother'
'bottom' 'bought' 'box' 'boxy' 'bra' 'brand' 'breast' 'bright' 'broad'
'broke' 'brown' 'built' 'bulk' 'bulky' 'bummer' 'bust' 'busty' 'butt'
'button' 'buy' 'buying' 'c' 'calf' 'came' 'cami' 'camisole' 'cant'
'cardigan' 'care' 'case' 'casual' 'center' 'cheap' 'chest' 'cheded'
'cling' 'close' 'closely' 'clothes' 'clothing' 'coat' 'cold' 'collar'
'color' 'colored' 'come' 'comfortable' 'comfy' 'coming' 'compared'
'complaint' 'completely' 'considered' 'considering' 'construction' 'cool'
'correctly' 'cost' 'cotton' 'could' 'couldnt' 'couple' 'coverage'
'covered' 'cozy' 'crazy' 'cream' 'cropped' 'crotch' 'cuff' 'cup' 'curve'
'curvy' 'customer' 'cut' 'cute' 'damaged' 'dark' 'darker' 'day' 'dd'
'deal' 'decided' 'deep' 'definitely' 'delicate' 'denim' 'depending'
'description' 'design' 'designer' 'despite' 'detail' 'didnt' 'different'
'difficult' 'dinner' 'disappointed' 'disappointing' 'disappointment'
'disaster' 'doesnt' 'dont' 'dot' 'drape' 'draped' 'drapey' 'dress'
'dressy' 'dry' 'due' 'dull' 'easily' 'edge' 'effect' 'either' 'elastic'
'else' 'elsewhere' 'embroidery' 'empire' 'end' 'ended' 'enormous'
'enough' 'entire' 'especially' 'even' 'ever' 'every' 'everyone'
'everything' 'exactly' 'except' 'exchanged' 'excited' 'execution'
'expected' 'expecting' 'expensive' 'exposed' 'extra' 'extremely' 'fabric'
'fact' 'fall' 'fan' 'far' 'fat' 'favorite' 'feel' 'feeling' 'fell' 'felt'
'feminine' 'figure' 'finally' 'find' 'fine' 'first' 'fit' 'fitted'
'fitting' 'flare' 'flat' 'flatter' 'flattering' 'flaw' 'flimsy' 'flow'
'flowy' 'fold' 'forward' 'found' 'frame' 'front' 'frumpy' 'full' 'fuller'
'fun' 'garment' 'gathered' 'get' 'getting' 'girl' 'give' 'given' 'go'
'going' 'gone' 'good' 'gorgeous' 'got' 'gotten' 'gray' 'great' 'green'
'grey' 'guess' 'half' 'hang' 'hanging' 'happened' 'happy' 'hard' 'hate'
'hated' 'head' 'heavier' 'heavy' 'height' 'held' 'help' 'helped' 'hem'
'hide' 'high' 'higher' 'hip' 'hit' 'hold' 'hole' 'home' 'hook' 'hope'
'hoping' 'horrible' 'horribly' 'hot' 'hour' 'hourglass' 'however' 'huge'
'hung' 'husband' 'id' 'idea' 'ill' 'im' 'image' 'imagine' 'immediately'
'impossible' 'impressed' 'inch' 'incredibly' 'initially' 'inside'
'instead' 'isnt' 'issue' 'itchy' 'item' 'ive' 'ivory' 'jacket' 'jean'
'jumpsuit' 'keep' 'keeping' 'kept' 'kind' 'knee' 'knew' 'knit' 'know' 'l'
'label' 'lace' 'lack' 'large' 'larger' 'last' 'lately' 'later' 'lay'
'layer' 'lb' 'le' 'least' 'leave' 'leaving' 'left' 'leg' 'legging'
'length' 'level' 'life' 'light' 'lightweight' 'like' 'liked' 'line'
'lined' 'lining' 'listened' 'literally' 'little' 'live' 'local' 'long'
'longer' 'look' 'looked' 'looking' 'loose' 'lose' 'lost' 'lot' 'love'
'loved' 'lovely' 'low' 'lower' 'made' 'maeve' 'mail' 'make' 'making'
'many' 'mark' 'match' 'material' 'maternity' 'may' 'maybe' 'mean'
'medium' 'meet' 'meh' 'mentioned' 'mess' 'messy' 'mid' 'middle' 'might'
'mind' 'mine' 'minute' 'misleading' 'miss' 'missing' 'mistake' 'model'
'mom' 'money' 'month' 'move' 'much' 'must' 'narrow' 'navy' 'near' 'neck'
'neckline' 'need' 'needed' 'negative' 'neither' 'never' 'new' 'next'
'nice' 'nicely' 'nope' 'normal' 'normally' 'nothing' 'notice' 'noticed'
'nude' 'odd' 'oddly' 'oh' 'ok' 'okay' 'old' 'one' 'online' 'open'
'opened' 'opening' 'opinion' 'orange' 'order' 'ordered' 'ordering'
'others' 'otherwise' 'outer' 'outside' 'overall' 'oversized' 'p'
'package' 'pair' 'pajama' 'pant' 'part' 'pas' 'past' 'pattern' 'people'

```

```

1 dtm5=pd.DataFrame(dtm4, columns = count_vect1.get_feature_names_out())
2 dtm5

```


	able	absolutely	across	actually	add	added	adorable	afraid	ago	agree	...	xx	year	yellow
0	0	0	0	0	0	1	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0
...
901	0	0	0	0	0	0	0	0	0	0	...	0	0	0
902	0	0	0	0	0	0	0	0	0	0	...	0	0	0
903	0	0	0	0	0	0	0	0	0	0	...	0	0	0
904	0	0	0	0	0	0	0	0	0	0	...	0	0	1

```
1 dtm5.apply(sum)

able      18
absolutely 20
across    14
actually  21
add        8
..
youd       5
youre     10
zip       22
zipped     8
zipper    34
Length: 800, dtype: int64
```

Word frequencies for positive sentiments

```
1 word_freq = pd.DataFrame(dtm1.apply(sum).head(40), columns=['freq'])
2 word_freq.sort_values('freq', ascending=False, inplace=True)

1 word_freq
```

also	1100
arm	1011
around	813
absolutely	638
adorable	580
another	543
almost	506
area	491
amazing	479
actually	475
add	442
always	423
although	302
able	296
anighthing	277
already	266
agree	251
across	243
ankle	236
away	234
arrived	220

```
1 word_freq_dictionary = dict(dtm1.apply(sum))
2 word_freq
```

	freq
also	1788
arm	1011
around	813
absolutely	638
adorable	580
another	543
almost	506
area	491
amazing	479
actually	475
add	442
always	423
although	302
able	296
anighthing	277
already	266

```

1 # import pandas as pd
2 # import plotly.express as px
3
4 # # Assuming word_freq is a DataFrame containing the word frequencies
5 # # Sort the DataFrame by frequency in descending order
6 # word_freq.sort_values('freq', ascending=False, inplace=True)
7
8 # # Reset the index (and drop the existing 'level_0' column)
9 # word_freq.reset_index(drop=True, inplace=True)
10
11 # # Create the bar graph using Plotly Express
12 # fig = px.bar(word_freq, x='index', y='freq', color='index', color_discrete_sequence=px.colors.qualitative.Pastel1)
13
14 # # Customize the layout
15 # fig.update_layout(
16 #     title='Word Frequency',
17 #     xaxis_title='Words',
18 #     yaxis_title='Frequency',
19 #     showlegend=False # Hide the legend
20 # )
21
22 # # Show the plot
23 # fig.show()
24

```

Word frequencies for negative sentiments

```

1 word_freq1 = pd.DataFrame(dtm5.apply(sum).head(40), columns=['freq'])
2 word_freq1.sort_values('freq', ascending=False, inplace=True)

```

```

1 word_freq_dictionary1 = dict(dtm5.apply(sum))

```

```

1 word_freq_dictionary1

```

```

{'able': 18,
 'absolutely': 20,
 'across': 14,
 'actually': 21,
 'add': 8,
 'added': 11,
 'adorable': 20,
 'afraid': 5,
 'ago': 6,
 'agree': 9,
 'ala': 6,
 'aline': 7,

```

```

'almost': 33,
'along': 11,
'already': 11,
'also': 146,
'although': 14,
'always': 23,
'amazing': 9,
'amount': 5,
'anighthing': 17,
'ankle': 12,
'another': 41,
'anyone': 12,
'anyway': 6,
'apart': 6,
'appears': 17,
'area': 53,
'arent': 7,
'arm': 79,
'armhole': 19,
'armpit': 12,
'around': 58,
'arrived': 29,
'assumed': 5,
'attached': 7,
'available': 12,
'average': 9,
'avoid': 6,
'away': 18,
'awesome': 6,
'awful': 38,
'awkward': 40,
'b': 18,
'back': 207,
'backside': 6,
'bad': 70,
'bag': 16,
'baggy': 16,
'band': 20,
'barely': 11,
'based': 9,
'basically': 8,
'bead': 7,
'beautiful': 71,
'behind': 7,
'belt': 14,
'best': 12

```

For making word_clouds for postive sentiments

```

1 from wordcloud import WordCloud, STOPWORDS
2 import matplotlib.pyplot as plt
3 import random
4
5 # Create a custom color map
6 def random_color_func(word=None, font_size=None, position=None, orientation=None, font_path=None, random_state=None):
7     h = int(360.0 * random.random())
8     s = int(100.0 * random.random())
9     l = int(50.0 + 20.0 * random.random())
10    return "hsl({}, {}, {})".format(h, s, l)
11
12 # Calculate the desired aspect ratio (width:height)
13 aspect_ratio = 1 # Since the desired figsize is 12x12, the aspect ratio is 1 (square)
14
15 # Calculate the width and height based on the aspect ratio and desired figsize
16 figsize = (12, 12) # Desired figsize
17 width = int(figsize[0] * 100) # Convert to inches and then to pixels
18 height = int(width / aspect_ratio)
19
20 # Create a WordCloud object with custom parameters
21 wordcloud = WordCloud(
22     background_color='white',
23     stopwords=STOPWORDS,
24     color_func=random_color_func,
25     width=width,
26     height=height,
27     max_words=50,
28     random_state=42
29 )
30
31 # Generate the word cloud from word_freq_dictionary

```



```

1 from wordcloud import WordCloud, STOPWORDS
2 import matplotlib.pyplot as plt
3 import random
4
5 # Create a custom color map
6 def random_color_func(word=None, font_size=None, position=None, orientation=None, font_path=None, random_state=None):
7     h = int(360.0 * random.random())
8     s = int(100.0 * random.random())
9     l = int(50.0 + 20.0 * random.random())
10    return "hsl({}, {}, {})".format(h, s, l)
11
12 # Calculate the desired aspect ratio (width:height)
13 aspect_ratio = 1 # Since the desired figsize is 12x12, the aspect ratio is 1 (square)
14
15 # Calculate the width and height based on the aspect ratio and desired figsize

```

```

16 figsize = (12, 12) # Desired figsize
17 width = int(figsize[0] * 100) # Convert to inches and then to pixels
18 height = int(width / aspect_ratio)
19
20 # Create a WordCloud object with custom parameters
21 wordcloud1 = WordCloud(
22     background_color='white',
23     stopwords=STOPWORDS,
24     color_func=random_color_func,
25     width=width,
26     height=height,
27     max_words=50,
28     random_state=42
29 )
30
31 # Generate the word cloud from word_freq_dictionary
32 wordcloud1 = wordcloud1.generate_from_frequencies(word_freq_dictionary1)
33
34 # Plot the word cloud
35 plt.figure(figsize=figsize)
36 plt.imshow(wordcloud1, interpolation='bilinear')
37 plt.axis('off')
38 plt.show()

```

```
1 dataset['age_group'] = pd.cut(x= dataset.Customer_Age,bins=[20, 29, 39, 49,59 ,69,79,89 ,99])

1 dataset.head()
```

	Product_ID	Category	SubCategory1	SubCategory2	Location	Channel	Customer_Age	Rating	Recommend
0	767	Initmates	Intimate	Intimates	Mumbai	Mobile	33	4	
1	1080	General	Dresses	Dresses	Bangalore	Mobile	34	5	
2	1077	General	Dresses	Dresses	Gurgaon	Mobile	60	3	

```
1 dataset.groupby(['Location', 'age_group', 'Category', 'SubCategory1', 'SubCategory2', "sentiment_category" ]).agg({'sentiment_category': 'count' })
```

						sentiment_category
Location	age_group	Category	SubCategory1	SubCategory2	sentiment_category	
Bangalore	(20, 29]	General	Bottoms	Blouses	Negative	0
					Neutral	0
					Positive	0
			Casual bottoms		Negative	0
					Neutral	0
...
Mumbai	(89, 99]	Initmates	Trend	Swim	Neutral	0
					Positive	0
				Trend	Negative	0
					Neutral	0
					Positive	0

34560 rows × 1 columns

```
1 """
2 d. Perform predictive analytics to understand the drivers of customers who are
3 recommending the products.
4
5 Regression model
6 Vectorization (count, tfidf) for both train & test
7 """

' \nd. Perform predictive analytics to understand the drivers of customers who are \nrecommending the products. \n\nRegression
model\nVectorization (count, tfidf) for both train & test\n'
```

```
1 dataset.head()
```

	Product_ID	Category	SubCategory1	SubCategory2	Location	Channel	Customer_Age	Rating	Recommend
0	767	Initmates	Intimate	Intimates	Mumbai	Mobile	33	4	
1	1080	General	Dresses	Dresses	Bangalore	Mobile	34	5	
2	1077	General	Dresses	Dresses	Gurgaon	Mobile	60	3	

```

1 """ Split the data in X and Y ( Recommend_Flag) """

    ' Split the data in X and Y ( Recommend_Flag) '

1 ## X-variable is Review_text and y-variable is Rating
2 # define X and y
3 X4 = dataset.Merged_Review
4 y4 = dataset.Recommend_Flag
5
6 # split the new DataFrame into training and testing sets
7 X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y4, random_state=1)
8 print(X4_train.shape)
9 print(X4_test.shape)
10 print(y4_train.shape)
11 print(y4_test.shape)

    (16981,)
    (5661,)
    (16981,)
    (5661,)

1 # Making a model using X_train (Merged_Review) data

1 X4_train = X4_train.apply(lambda x: clean_text(x))
2 X4_test = X4_test.apply(lambda x: clean_text(x))

1 # Train
2 count_vect = CountVectorizer(analyzer='word',
3                             token_pattern=r'\w{1,}',
4                             ngram_range=(1, 1 ),
5                             min_df=5,
6                             lowercase=True,
7                             encoding='latin-1',
8                             max_features=1000)
9 X_train_count4 = count_vect.fit_transform(X4_train)
10
11 tfidf_vect = TfidfVectorizer(analyzer='word',
12                             token_pattern=r'\w{1,}',
13                             ngram_range=(1, 1 ),
14                             min_df=5,
15                             encoding='latin-1',
16                             lowercase=True,
17                             max_features=1000)
18 X_train_tfidf4 = tfidf_vect.fit_transform(X4_train)
19
20 # Test
21 X_test_count4 = count_vect.transform(X4_test)
22 X_test_tfidf4 = tfidf_vect.transform(X4_test)
23
24 dtm_count = pd.DataFrame(X_train_count4.toarray(), columns=count_vect.get_feature_names_out())
25 dtm_tfidf = pd.DataFrame(X_train_tfidf4.toarray(), columns=tfidf_vect.get_feature_names_out())
26

1 """
2 Adding Features to a Document-Term Matrix
3
4 Dummy Creation
5 """

    '\nAdding Features to a Document-Term Matrix\n\nDummy Creation\n'

1 dataset.head()

```



```
Product_ID  Category  SubCategory1  SubCategory2  Location  Channel  Customer_Age  Rating  Recommend

0          767  Initmates      Intimate      Intimates   Mumbai   Mobile              33      4

1 # An utility function to create dummy variable
2 def create_dummies(dataset, colname):
3     col_dummies = pd.get_dummies(dataset[colname], prefix = colname, drop_first = True)
4     dataset = pd.concat([dataset, col_dummies], axis = 1)
5     dataset.drop(colname, axis = 1, inplace = True )
6     return dataset

1 catagory_variables = dataset[['Category', 'SubCategory1', 'SubCategory2', 'sentiment_category','Location', 'Channel']]
2
3 # for c_feature in categorical_features
4 for c_feature in ['Category', 'SubCategory1', 'SubCategory2', 'sentiment_category','Location', 'Channel']:
5     catagory_variables[c_feature] = catagory_variables[c_feature].astype('category')
6     catagory_variables = create_dummies(catagory_variables, c_feature)
7
8
9 catagory_variables.head()

C:\Users\leaps\AppData\Local\Temp\ipykernel_7364\3716304331.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.
```

	Category_General Petite	Category_Initmates	SubCategory1_Dresses	SubCategory1_Intimate	SubCategory1_Jac
0	0	1	0	1	
1	0	0	1	0	
2	0	0	1	0	
3	1	0	0	0	
4	0	0	0	0	

5 rows × 32 columns

```
1 dtm_count

a  all  also  am  an  and  are  as  at  back  ...  was  wear  well  when  which  will  with  work
0   3   0   0  0  1   4   0  0  0   0  ...  1   1   1   0   2   0   0   0
1   2   0   1  0  1   2   1  0  0   1  ...  1   0   0   1   2   0   0   0
2   1   0   0  0  0   6   0  0  0   0  ...  0   0   0   1   0   0   0   0
3   3   0   0  1  0   3   0  0  0   0  ...  0   1   0   1   0   0   1   0
4   6   0   1  1  0   3   1  0  1   0  ...  0   1   0   0   1   0   2   0
...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
16976 4   0   1  0  0   3   1  0  0   0  ...  1   1   0   0   0   0   2   0
16977 5   0   0  0  1   3   1  0  0   1  ...  0   1   1   0   1   1   1   0
16978 1   0   0  0  0   0   0  0  0   0  ...  0   0   0   0   0   0   0   0
16979 6   0   1  1  0   2   1  0  0   0  ...  1   0   0   0   0   0   0   0
16980 2   0   0  0  0   2   1  0  0   0  ...  0   0   0   0   0   0   0   1

16981 rows × 100 columns
```

```
1 dataset.head()
```

	Product_ID	Category	SubCategory1	SubCategory2	Location	Channel	Customer_Age	Rating	Recommend
0	767	Initmates	Intimate	Intimates	Mumbai	Mobile	33	4	
1	1080	General	Dresses	Dresses	Bangalore	Mobile	34	5	

```
1 dataset_new =dataset.loc[:, [ 'Merged_Review', 'Customer_Age', 'Rating', 'sentiment_score','Recommend_Flag']]
```

	Merged_Review	Customer_Age	Rating	sentiment_score	Recommend_Flag
0	Absolutely wonderful - silky and sexy and com...	33	4	0.8932	1
1	Love this dress! it's sooo pretty. i happen...	34	5	0.9729	1
2	Some major design flaws I had such high hopes ...	60	3	0.9427	0
3	My favorite buy! I love, love, love this jumps...	50	5	0.7182	1
4	Flattering shirt This shirt is very flattering...	47	5	0.9436	1

```
1 data = pd.concat([ dataset_new,catagory_variables], axis =1)
```

```
1 data.columns
```

```
Index(['Merged_Review', 'Customer_Age', 'Rating', 'sentiment_score',  
      'Recommend_Flag', 'Category_General Petite', 'Category_Initmates',  
      'SubCategory1_Dresses', 'SubCategory1_Intimate', 'SubCategory1_Jackets',  
      'SubCategory1_Tops', 'SubCategory1_Trend',  
      'SubCategory2_Casual bottoms', 'SubCategory2_Chemises',  
      'SubCategory2_Dresses', 'SubCategory2_Fine gauge',  
      'SubCategory2_Intimates', 'SubCategory2_Jackets', 'SubCategory2_Jeans',  
      'SubCategory2_Knits', 'SubCategory2_Layering', 'SubCategory2_Legwear',  
      'SubCategory2_Lounge', 'SubCategory2_Outerwear', 'SubCategory2_Pants',  
      'SubCategory2_Short', 'SubCategory2_Skirts', 'SubCategory2_Sleep',  
      'SubCategory2_Sweaters', 'SubCategory2_Swim', 'SubCategory2_Trend',  
      'sentiment_category_Neutral', 'sentiment_category_Positive',  
      'Location_Chennai', 'Location_Gurgaon', 'Location_Mumbai',  
      'Channel_Web'],  
      dtype='object')
```

```
1 data.head()
```

	Merged_Review	Customer_Age	Rating	sentiment_score	Recommend_Flag	Category_General Petite	Category_Ini
0	Absolutely wonderful - silky and sexy and com...	33	4	0.8932	1	0	
1	Love this dress! it's sooo pretty. i happen...	34	5	0.9729	1	0	
2	Some major design flaws I had such high hopes ...	60	3	0.9427	0	0	
3	My favorite buy! I love, love, love this jumps...	50	5	0.7182	1	1	
4	Flattering shirt This shirt is very flattering...	47	5	0.9436	1	0	

5 rows × 37 columns

```
1 #define X and y  
2 feature_cols = ['Merged_Review', 'Customer_Age', 'Rating', 'sentiment_score',
```

```

3     'Recommend_Flag', 'Category_General Petite', 'Category_Initmates',
4     'SubCategory1_Dresses', 'SubCategory1_Intimate', 'SubCategory1_Jackets',
5     'SubCategory1_Tops', 'SubCategory1_Trend',
6     'SubCategory2_Casual bottoms', 'SubCategory2_Chemises',
7     'SubCategory2_Dresses', 'SubCategory2_Fine gauge',
8     'SubCategory2_Intimates', 'SubCategory2_Jackets', 'SubCategory2_Jeans',
9     'SubCategory2_Knits', 'SubCategory2_Layering', 'SubCategory2_Legwear',
10    'SubCategory2_Lounge', 'SubCategory2_Outerwear', 'SubCategory2_Pants',
11    'SubCategory2_Short', 'SubCategory2_Skirts', 'SubCategory2_Sleep',
12    'SubCategory2_Sweaters', 'SubCategory2_Swim', 'SubCategory2_Trend',
13    'sentiment_category_Neutral', 'sentiment_category_Positive',
14    'Location_Chennai', 'Location_Gurgaon', 'Location_Mumbai',
15    'Channel_Web']
16 X = data[feature_cols]
17 y = data.Recommend_Flag
18
19 #split into training and testing sets
20 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

1 X_train.Merged_Review

10853    Cute and comfy The tank top fits well. i usual...
7464     Itchy and odd fit I was so bummed when this ca...
17084    Basic everyday joggers I just purchased these ...
18045    I love the fit and length of this sweater! Thi...
19034    Great fit, perfect for summer I bought this cu...
      ...
10955    Love these pants I bought these about a month ...
17289    Black is see-through up close I'm usually a sm...
5192     Runs small Should have ordered the xl instead ...
12172    Great Love these-although i'm exchanging mine ...
235     Recommend but not for me Love pilcro, love the...
Name: Merged_Review, Length: 16981, dtype: object

1 # use TfidfVectorizer with Merged_Review column only
2 vect = TfidfVectorizer(lowercase=True, stop_words='english', max_features=100, min_df=5, ngram_range=(1, 2))
3 X_train_dtm = vect.fit_transform(X_train.Merged_Review)
4 X_test_dtm = vect.transform(X_test.Merged_Review)
5 print(X_train_dtm.shape)
6 print(X_test_dtm.shape)
7
8 # shape of other four feature columns
9 X_train.drop('Merged_Review', axis=1).shape

(16981, 100)
(5661, 100)
(16981, 36)

1 print(vect.get_feature_names_out())

['_x000d_' 'beautiful' 'better' 'big' 'bit' 'black' 'blue' 'body' 'bought'
'casual' 'color' 'colors' 'comfortable' 'comfy' 'cut' 'cute' 'definitely'
'design' 'did' 'didn' 'does' 'don' 'dress' 'fabric' 'fall' 'feel' 'fit'
'fits' 'flattering' 'going' 'good' 'gorgeous' 'got' 'great' 'high'
'jacket' 'jeans' 'just' 'large' 'lbs' 'length' 'like' 'little' 'long'
'look' 'looked' 'looking' 'looks' 'loose' 'love' 'loved' 'lovely'
'material' 'medium' 'model' 'nice' 'online' 'ordered' 'pants' 'perfect'
'perfectly' 'person' 'petite' 'piece' 'pretty' 'price' 'purchased'
'quality' 'really' 'retailer' 'right' 'runs' 'sale' 'shirt' 'short'
'size' 'skirt' 'sleeves' 'small' 'soft' 'store' 'style' 'summer' 'super'
'sweater' 'think' 'tight' 'tried' 'true' 'true size' 'try' 'usually'
'waist' 'wanted' 'way' 'wear' 'wearing' 'white' 'work' 'xs']

1 # use CountVectorizer with Merged_Review column only
2 vect = CountVectorizer(lowercase=True, stop_words='english', max_features=100, min_df=5, ngram_range=(1, 2))
3 X_train_dtm = vect.fit_transform(X_train.Merged_Review)
4 X_test_dtm = vect.transform(X_test.Merged_Review)
5 print(X_train_dtm.shape)
6 print(X_test_dtm.shape)
7
8 # shape of other four feature columns
9 X_train.drop('Merged_Review', axis=1).shape

(16981, 100)
(5661, 100)
(16981, 36)

```

```

1 print(vect.get_feature_names_out())

['_x000d_' 'beautiful' 'better' 'big' 'bit' 'black' 'blue' 'body' 'bought'
 'casual' 'color' 'colors' 'comfortable' 'comfy' 'cut' 'cute' 'definitely'
 'design' 'did' 'didn't' 'does' 'don' 'dress' 'fabric' 'fall' 'feel' 'fit'
 'fits' 'flattering' 'going' 'good' 'gorgeous' 'got' 'great' 'high'
 'jacket' 'jeans' 'just' 'large' 'lbs' 'length' 'like' 'little' 'long'
 'look' 'looked' 'looking' 'looks' 'loose' 'love' 'loved' 'lovely'
 'material' 'medium' 'model' 'nice' 'online' 'ordered' 'pants' 'perfect'
 'perfectly' 'person' 'petite' 'piece' 'pretty' 'price' 'purchased'
 'quality' 'really' 'retailer' 'right' 'runs' 'sale' 'shirt' 'short'
 'size' 'skirt' 'sleeves' 'small' 'soft' 'store' 'style' 'summer' 'super'
 'sweater' 'think' 'tight' 'tried' 'true' 'true size' 'try' 'usually'
 'waist' 'wanted' 'way' 'wear' 'wearing' 'white' 'work' 'xs']

1 from scipy import sparse
2 # cast other feature columns to float and convert to a sparse matrix
3 extra = sparse.csr_matrix(X_train.drop('Merged_Review', axis=1).astype(float))
4 extra.shape
5
6 # combine sparse matrices
7 X_train_dtm_extra = sparse.hstack((X_train_dtm, extra))
8 X_train_dtm_extra.shape
9
10 # repeat for testing set
11 extra = sparse.csr_matrix(X_test.drop('Merged_Review', axis=1).astype(float))
12 X_test_dtm_extra = sparse.hstack((X_test_dtm, extra))
13 X_test_dtm_extra.shape

(5661, 136)

1 from sklearn.linear_model import LogisticRegression
2 from sklearn import metrics
3 # use logistic regression with text column only
4 logreg = LogisticRegression(C=1e9)
5 logreg.fit(X_train_dtm, y_train)
6 y_pred_class = logreg.predict(X_test_dtm)
7 print(metrics.accuracy_score(y_test, y_pred_class))

0.8572690337396219

1 print(dir(logreg))

['C', '__annotations__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__']

1 # Finding the score for validation
2
3 from sklearn.metrics import precision_recall_fscore_support as score
4 from sklearn.metrics import roc_auc_score, accuracy_score
5
6 tr_pred=logreg.predict(X_train_dtm)
7 y_pred = logreg.predict(X_test_dtm)
8
9
10 trprecision,trrecall,trfscore,trsupport=score(y_train,tr_pred)
11 tracc=accuracy_score(y_train,tr_pred)
12 precision,recall,fscore,support=score(y_test,y_pred)
13 acc=accuracy_score(y_test,y_pred)

1 # For Training
2
3 print('Precision : ',trprecision)
4 print('\nRecall : ',trrecall)
5 print('\nF-Score : ',trfscore)
6 print('\nAccuracy : ',tracc)

Precision : [0.67402096 0.8771097 ]

Recall : [0.39598185 0.95746671]

F-Score : [0.49887732 0.91552833]

Accuracy : 0.8554266533184147

```

```

1 # For Testing
2
3 print('Precision : ',precision)
4 print('\nRecall : ',recall)
5 print('\nF-Score : ',fscore)
6 print('\nAccuracy : ',acc)

Precision : [0.67814114 0.87775591]

Recall : [0.38817734 0.95975032]

F-Score : [0.49373434 0.91692371]

Accuracy : 0.8572690337396219

```

```

1 #Saving model
2 import pickle
3 Pkl_Filename = "Pickle_LR_Model.pkl"
4
5 with open(Pkl_Filename, 'wb') as file:
6     pickle.dump(logreg, file)

1 # Load the Model back from file
2 with open(Pkl_Filename, 'rb') as file:
3     Pickled_LR_Model = pickle.load(file)
4
5 Pickled_LR_Model

```

```

▼ LogisticRegression
LogisticRegression(C=1000000000.0)

```

```

1 # Use the Reloaded Model to
2 # Calculate the accuracy score and predict target values
3
4 # Calculate the Score
5 score = Pickled_LR_Model.score(X_test_dtm, y_test)
6 # Print the Score
7 print("Test score: {0:.2f} %".format(100 * score))
8
9 # Predict the Labels using the reloaded Model
10 Ypredict = Pickled_LR_Model.predict(X_test_dtm)
11 Ypredict

Test score: 85.73 %
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)

```

```
1 print(vect.get_feature_names_out())
```

```

['_x000d_' 'beautiful' 'better' 'big' 'bit' 'black' 'blue' 'body' 'bought'
'casual' 'color' 'colors' 'comfortable' 'comfy' 'cut' 'cute' 'definitely'
'design' 'did' 'didn' 'does' 'don' 'dress' 'fabric' 'fall' 'feel' 'fit'
'fits' 'flattering' 'going' 'good' 'gorgeous' 'got' 'great' 'high'
'jacket' 'jeans' 'just' 'large' 'lbs' 'length' 'like' 'little' 'long'
'look' 'looked' 'looking' 'looks' 'loose' 'love' 'loved' 'lovely'
'material' 'medium' 'model' 'nice' 'online' 'ordered' 'pants' 'perfect'
'perfectly' 'person' 'petite' 'piece' 'pretty' 'price' 'purchased'
'quality' 'really' 'retailer' 'right' 'runs' 'sale' 'shirt' 'short'
'size' 'skirt' 'sleeves' 'small' 'soft' 'store' 'style' 'summer' 'super'
'sweater' 'think' 'tight' 'tried' 'true' 'true size' 'try' 'usually'
'waist' 'wanted' 'way' 'wear' 'wearing' 'white' 'work' 'xs']

```

```

1 """
2     So, Above we made one logistic model where Y variable is recommend_flag and X variable
3     is review_text. The model gives accuracy of 85.73% in train and test. The difference between
4     train and test accuracy is less so, we can say the model is good to use. The main key
5     drivers who are responsible for recommending the product are as above.
6 """

```

```

' \n     So, Above we made one logistic model where Y variable is recommend_flag and X variable \n     is review_text. The model gives
accuracy of 85.73% in train and test. The difference between \n     train and test accuracy is less so, we can say the model is good to
use. The main key \n     drivers who are responsible for recommending the product are as above.\n '

```

```

1 #define X and y
2 feature_cols = ['Merged_Review', 'Customer_Age', 'Rating', 'sentiment_score',

```

```

3     'Recommend_Flag', 'Category_General Petite', 'Category_Initmates',
4     'SubCategory1_Dresses', 'SubCategory1_Intimate', 'SubCategory1_Jackets',
5     'SubCategory1_Tops', 'SubCategory1_Trend',
6     'SubCategory2_Casual bottoms', 'SubCategory2_Chemises',
7     'SubCategory2_Dresses', 'SubCategory2_Fine gauge',
8     'SubCategory2_Intimates', 'SubCategory2_Jackets', 'SubCategory2_Jeans',
9     'SubCategory2_Knits', 'SubCategory2_Layering', 'SubCategory2_Legwear',
10    'SubCategory2_Lounge', 'SubCategory2_Outerwear', 'SubCategory2_Pants',
11    'SubCategory2_Short', 'SubCategory2_Skirts', 'SubCategory2_Sleep',
12    'SubCategory2_Sweaters', 'SubCategory2_Swim', 'SubCategory2_Trend',
13    'sentiment_category_Neutral', 'sentiment_category_Positive',
14    'Location_Chennai', 'Location_Gurgaon', 'Location_Mumbai',
15    'Channel_Web']
16 X = data[feature_cols]
17 y = data.Rating
18
19 #split into training and testing sets
20 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

1 # use TfidfVectorizer with Merged_Review column only
2 vect = TfidfVectorizer(lowercase=True, stop_words='english', max_features=100, min_df=5, ngram_range=(1, 2))
3 X_train_dtm = vect.fit_transform(X_train.Merged_Review)
4 X_test_dtm = vect.transform(X_test.Merged_Review)
5 print(X_train_dtm.shape)
6 print(X_test_dtm.shape)
7
8 # shape of other four feature columns
9 X_train.drop('Merged_Review', axis=1).shape

(16981, 100)
(5661, 100)
(16981, 36)

1 print(vect.get_feature_names_out())

['_x000d' 'beautiful' 'better' 'big' 'bit' 'black' 'blue' 'body' 'bought'
'casual' 'color' 'colors' 'comfortable' 'comfy' 'cut' 'cute' 'definitely'
'design' 'did' 'didn' 'does' 'don' 'dress' 'fabric' 'fall' 'feel' 'fit'
'fits' 'flattering' 'going' 'good' 'gorgeous' 'got' 'great' 'high'
'jacket' 'jeans' 'just' 'large' 'lbs' 'length' 'like' 'little' 'long'
'look' 'looked' 'looking' 'looks' 'loose' 'love' 'loved' 'lovely'
'material' 'medium' 'model' 'nice' 'online' 'ordered' 'pants' 'perfect'
'perfectly' 'person' 'petite' 'piece' 'pretty' 'price' 'purchased'
'quality' 'really' 'retailer' 'right' 'runs' 'sale' 'shirt' 'short'
'size' 'skirt' 'sleeves' 'small' 'soft' 'store' 'style' 'summer' 'super'
'sweater' 'think' 'tight' 'tried' 'true' 'true size' 'try' 'usually'
'waist' 'wanted' 'way' 'wear' 'wearing' 'white' 'work' 'xs']

1 # use logistic regression with text column only
2 logreg2 = LogisticRegression(C=1e9)
3 logreg2.fit(X_train_dtm, y_train)
4 y_pred_class = logreg2.predict(X_test_dtm)
5 print(metrics.accuracy_score(y_test, y_pred_class))

0.5991874227168345
c:\Users\pholl\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

1 #Using KNN model
2 from sklearn.neighbors import KNeighborsClassifier
3
4 model3=KNeighborsClassifier(n_neighbors=5,n_jobs=-1)
5 model3.fit(X_train_dtm,y_train)

▼ KNeighborsClassifier
KNeighborsClassifier(n_jobs=-1)

```

```

1 # Finding the score for validation
2
3 from sklearn.metrics import precision_recall_fscore_support as score
4 from sklearn.metrics import roc_auc_score, accuracy_score
5
6 tr_pred=logreg2.predict(X_train_dtm)
7 y_pred = logreg2.predict(X_test_dtm)
8
9
10 trprecision, trrecall, trfscore, trsupport=score(y_train, tr_pred)
11 tracc=accuracy_score(y_train, tr_pred)
12 precision, recall, fscore, support=score(y_test, y_pred)
13 acc=accuracy_score(y_test, y_pred)

Precision : [0.3649635 0.32454361 0.38529718 0.43855816 0.68327316]

Recall : [0.16207455 0.13793103 0.34776471 0.2179696 0.90654604]

F-Score : [0.22446689 0.19358742 0.36557012 0.2912058 0.77923147]

Accuracy : 0.6076791708379954

1 # For Testing
2
3 print('Precision : ', trprecision)
4 print('\nRecall : ', trrecall)
5 print('\nF-Score : ', trfscore)
6 print('\nAccuracy : ', tracc)

Precision : [0.34883721 0.3030303 0.33552632 0.41826923 0.68142652]

Recall : [0.14705882 0.1285347 0.29226361 0.21323529 0.90495868]

F-Score : [0.20689655 0.18050542 0.31240429 0.28246753 0.77744402]

Accuracy : 0.5991874227168345

1 # Above model is useful where X variable is Merged_Review and Y variable is Rating.

1 #Create user defined function for train Classification the models

1 def train_model(classifier, feature_vector_train, label, feature_vector_valid, valid_y, is_neural_net=False):
2     # fit the training dataset on the classifier
3     classifier.fit(feature_vector_train, label)
4
5     # predict the labels on validation dataset
6     predictions = classifier.predict(feature_vector_valid)
7
8     if is_neural_net:
9         predictions = predictions.argmax(axis=-1)
10
11     return metrics.accuracy_score(predictions, valid_y)

1 #Naive Bayes (With only review_text in X-vribles)
2 #Naive Bayes on Count Vectors and TF-IDF
3 from sklearn import naive_bayes
4 accuracy_L1 = train_model(naive_bayes.MultinomialNB(), X_train_dtm, y_train, X_test_dtm, y_test)
5 print("NB for L1, TFIDF Vectors: ", accuracy_L1)

NB for L1, TFIDF Vectors: 0.5594417947359124

1 #Logistic Regression
2 #Logistic Regression on Count Vectors and TF-IDF
3 accuracy_L1 = train_model(LogisticRegression(), X_train_dtm, y_train, X_test_dtm, y_test)
4 print("LR for L1, tfidf Vectors: ", accuracy_L1)

```

```
LR for L1, tfidf Vectors: 0.600247306129659
c:\Users\pholl\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
```

```
1 #Linear SVC
2 #Linear SVC on Count Vectors and TF-IDF
3 from sklearn import svm
4
5 accuracy_L1 = train_model(svm.LinearSVC(), X_train_dtm, y_train, X_test_dtm, y_test)
6 print("SVC for L1, Count Vectors: ", accuracy_L1)
```

```
c:\Users\pholl\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning:

The default value of `dual` will change from `True` to `auto` in 1.5. Set the value of `dual` explicitly to suppress the warning.

SVC for L1, Count Vectors: 0.5968910086557145
```

```
1 #Hence from above logistic regression gives best result.save the model as pickle object.
```

```
1 print(vect.get_feature_names_out())

['_x000d_', 'beautiful', 'better', 'big', 'bit', 'black', 'blue', 'body', 'bought',
'casual', 'color', 'colors', 'comfortable', 'comfy', 'cut', 'cute', 'definitely',
'design', 'did', 'didn', 'does', 'don', 'dress', 'fabric', 'fall', 'feel', 'fit',
'fits', 'flattering', 'going', 'good', 'gorgeous', 'got', 'great', 'high',
'jacket', 'jeans', 'just', 'large', 'lbs', 'length', 'like', 'little', 'long',
'look', 'looked', 'looking', 'looks', 'loose', 'love', 'loved', 'lovely',
'material', 'medium', 'model', 'nice', 'online', 'ordered', 'pants', 'perfect',
'perfectly', 'person', 'petite', 'piece', 'pretty', 'price', 'purchased',
'quality', 'really', 'retailer', 'right', 'runs', 'sale', 'shirt', 'short',
'size', 'skirt', 'sleeves', 'small', 'soft', 'store', 'style', 'summer', 'super',
'sweater', 'think', 'tight', 'tried', 'true', 'true size', 'try', 'usually',
'waist', 'wanted', 'way', 'wear', 'wearing', 'white', 'work', 'xs']
```

e. Create topics and understand themes behind the topics by performing TOPIC MINING

Topic Modeling

Topic Modeling using gensim

```
1 # Importing Gensim
2 import gensim
3 from gensim import corpora

1 X_train_tokens = [doc.split() for doc in X_train]
2 X_train_tokens

[['Merged_Review'],
 ['Customer_Age'],
 ['Rating'],
 ['sentiment_score'],
 ['Recommend_Flag'],
 ['Category_General', 'Petite'],
 ['Category_Initmates'],
 ['SubCategory1_Dresses'],
 ['SubCategory1_Intimate'],
 ['SubCategory1_Jackets'],
 ['SubCategory1_Tops'],
 ['SubCategory1_Trend'],
 ['SubCategory2_Casual', 'bottoms'],
 ['SubCategory2_Chemises'],
 ['SubCategory2_Dresses'],
 ['SubCategory2_Fine', 'gauge'],
 ['SubCategory2_Intimates'],
 ['SubCategory2_Jackets'],
 ['SubCategory2_Jeans'],
 ['SubCategory2_Knits'],
```



```
[ 'SubCategory2_Layering'],
[ 'SubCategory2_Legwear'],
[ 'SubCategory2_Lounge'],
[ 'SubCategory2_Outerwear'],
[ 'SubCategory2_Pants'],
[ 'SubCategory2_Shorts'],
[ 'SubCategory2_Skirts'],
[ 'SubCategory2_Sleep'],
[ 'SubCategory2_Sweaters'],
[ 'SubCategory2_Swim'],
[ 'SubCategory2_Trend'],
[ 'sentiment_category_Neutral'],
[ 'sentiment_category_Positive'],
[ 'Location_Chennai'],
[ 'Location_Gurgaon'],
[ 'Location_Mumbai'],
[ 'Channel_Web']]
```

```
1 X_train.head()
```

	Merged_Review	Customer_Age	Rating	sentiment_score	Recommend_Flag	Category_General Petite	Category
10853	Cute and comfy The tank top fits well. i usual...	32	4	0.7803	1	0	
7464	Itchy and odd fit I was so bummed when this ca...	62	2	0.9042	0	0	
17084	Basic everyday joggers I just purchased these ...	53	4	0.8810	1	0	
18045	I love the fit and length of this sweater! Thi...	35	5	0.9027	1	1	
19034	Great fit, perfect for summer I bought this cu...	57	5	0.9819	1	1	

5 rows × 37 columns

```
1 # Creating a Gensim dictionary from tokenized documents
2 dictionary = corpora.Dictionary(X_train_tokens)
3 print(dictionary)
```

Dictionary<40 unique tokens: ['Merged_Review', 'Customer_Age', 'Rating', 'sentiment_score', 'Recommend_Flag']...>

```
1 # Creating a Document-Term Matrix (DTM) using Gensim's doc2bow function
2 doc_term_matrix = [dictionary.doc2bow(doc) for doc in X_train_tokens]
```

```
1 # Creating the object for LDA model using gensim library
2 Lda = gensim.models.ldamodel.LdaModel
```

```
1 # Running and Trainign LDA model on the document term matrix.
2 ldamodel = Lda(doc_term_matrix, num_topics=5, id2word = dictionary,passes=1)
```

```
1 print(ldamodel.print_topics(num_topics=5, num_words=20))
```

[(0, '0.064*Category_General" + 0.064*SubCategory2_Casual" + 0.064*SubCategory2_Fine" + 0.064*gauge" + 0.064*Petite" + 0.064*botte

```
1 topics = ldamodel.show_topics(formatted=False, num_words=20)
2
3 for t in range(len(topics)):
4     print("\nTopic {}, top {} words:".format(t+1,30))
5     print(" ".join([w[0] for w in topics[t][1]]))
```

Topic 1, top 30 words: