eBay Review Classification

Team 10

Leonard Littleton, Kiran Singh, Andrew Zazueta

Libraries

In [11]:
```
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
import requests

from string import punctuation
from nltk.corpus import stopwords
import re
import emoji

import seaborn as sns
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from spacy.lang.en.stop_words import STOP_WORDS as stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.decomposition import NMF
```

Reading in Data from eBay

Custom Functions

In [3]:
```
# Generating links by imputting review page link

def link_generator(url):

    # truncating link to get base

    page = url
    page_base = page[:-1]

    # finding total review pages

    r = requests.get(page)
    soup = BeautifulSoup(r.text, 'html.parser')
    total_pages = int(str(soup).split('"totalPages":')[1].split('}')[0])
    links = []

    # generating links

    for link in range(1,total_pages+1):
        links.append(page_base+str(link))

    return links

def ebay_scrape(links):

    stars = []
    content = []
    titles = []

    for page in links:

        r = requests.get(page)
        soup = BeautifulSoup(r.text, 'html.parser')

        for star in soup.find_all('div', class_='ebay-review-section-l'):
            stars.append(star.find('span','star-rating')['aria-label'])

        for review_content in soup.find_all('p', class_ = 'review-item-content rvw-wrap-spaces'):
            content.append(review_content.text)

        for review_title in soup.find_all('h3', class_ = 'review-item-title rvw-nowrap-spaces'):
            titles.append(review_title.text)

    return stars, content, titles
```

Product 1: Texas Instruments TI-83 Plus Graphing Calculator

In [4]:
```
# initialzing df

eBay_1 = []
eBay_1 = pd.DataFrame(eBay_1)
```

In [5]:
```
# Getting page and page base to generate links

url = 'https://www.ebay.com/urw/Texas-Instruments-TI-83-Plus-Graphing-Calculator/product-reviews/54847886?co
links = link_generator(url)
```

In [6]:
```
# using ebay_scrape to collect star rating, the content of the reviews, and the review title

stars, content, titles = ebay_scrape(links)
```

In [7]:
```
# Putting list into Data Frame

eBay_1['Review Title'] = titles
#eBay['Review Content'] = content
eBay_1['Stars'] = stars
eBay_1['Item'] = 'Texas Instruments TI-83 Plus Graphing Calculator'
```

A user must include a title in their review, but content is not needed. This creates a different amount of content items than titles and stars. This results in us having to use titles as our source of text, as there is no way to know which reviews have content in their review or not.

Product 2: Apple AirPods Pro Left Airpod OEM Left Side Airpods Pro Only

In [8]:
```
# initialzing df

eBay_2 = []
eBay_2 = pd.DataFrame(eBay_2)
```

In [9]:
```
# Getting page and page base to generate links

url = 'https://www.ebay.com/urw/Apple-AirPods-Pro-Wireless-In-Ear-Headsets-White/product-reviews/10034976643
links = link_generator(url)
```

In [10]:
```
# using ebay_scrape to collect star rating, the content of the reviews, and the review title

stars, content, titles = ebay_scrape(links)
```

In [11]:
```
# Putting list into Data Frame

eBay_2['Review Title'] = titles
eBay_2['Stars'] = stars
eBay_2['Item'] = 'Apple AirPods Pro Left Airpod OEM Left Side Airpods Pro Only'
```

Product 3: Sony PS5 Console w/ Blu-Ray Disc

In [12]:
```
# initialzing df

eBay_3 = []
eBay_3 = pd.DataFrame(eBay_3)
```

In [17]:
```
# Getting page and page base to generate links

url = 'https://www.ebay.com/urw/Sony-PS5-Blu-Ray-Edition-Console-White/product-reviews/19040936896?_itm=2651
links = link_generator(url)
```

In [18]:
```
# using ebay_scrape to collect star rating, the content of the reviews, and the review title

stars, content, titles = ebay_scrape(links)
```

In [19]:
```
# Putting list into Data Frame

eBay_3['Review Title'] = titles
eBay_3['Stars'] = stars
eBay_3['Item'] = 'Sony PS5 Console w/ Blu-Ray Disc'
```

Product 4: Super Bright 90000LM LED Tactical Flashlight Zoomable With Rechargeable Battery

In [20]:
```
# initialzing df

eBay_4 = []
eBay_4 = pd.DataFrame(eBay_4)
```

In [21]:
```
# Getting page and page base to generate links

url = 'https://www.ebay.com/urw/Garberiel-AF20171221-8000-Lumens-Tactical-Flashlight-With-Rechargeable-Batte
links = link_generator(url)
```

In [22]:
```
# using ebay_scrape to collect star rating, the content of the reviews, and the review title

stars, content, titles = ebay_scrape(links)
```

In [23]:
```
# Putting list into Data Frame

eBay_4['Review Title'] = titles
eBay_4['Stars'] = stars
eBay_4['Item'] = 'Super Bright 90000LM LED Tactical Flashlight Zoomable With Rechargeable Battery'
```

Product 5: Canon PIXMA MG2520 All-In-One Inkjet Printer

In [24]:
```
# initialzing df

eBay_5 = []
eBay_5 = pd.DataFrame(eBay_5)
```

In [25]:
```
# Getting page and page base to generate links

url = 'https://www.ebay.com/urw/Canon-PIXMA-MG2520-All-In-One-Inkjet-Printer/product-reviews/182754283?pgn=1
links = link_generator(url)
```

In [26]:
```
# using ebay_scrape to collect star rating, the content of the reviews, and the review title

stars, content, titles = ebay_scrape(links)
```

In [27]:
```
# Putting list into Data Frame

eBay_5['Review Title'] = titles
eBay_5['Stars'] = stars
eBay_5['Item'] = 'Canon PIXMA MG2520 All-In-One Inkjet Printer'
```

Creating Final Data Frame

In [28]:
```
# Combining data to get one data frame

eBay = pd.concat([eBay_1, eBay_2, eBay_3, eBay_4, eBay_5], ignore_index=True)
eBay.reset_index(drop = True, inplace = True)
```

In [29]:
```
# Binning reviews for predictive modeling

is_positive = []

for stars in eBay['Stars']:
    if stars == '5 stars' or stars == '4 stars':
        is_positive.append(1)
    else:
        is_positive.append(0)

eBay['Is_Positive?'] = is_positive
```

In [30]:
```
# What the final dataframe looks like

eBay
```

Out[30]:

	Review Title	Stars	Item	Is_Positive?
0	Nice item- as we used to say: "Works fine, las...	5 stars	Texas Instruments TI-83 Plus Graphing Calculator	1
1	Cheap	3 stars	Texas Instruments TI-83 Plus Graphing Calculator	0
2	Texas Instruments TI-83 Plus Graphic Calculator	4 stars	Texas Instruments TI-83 Plus Graphing Calculator	1
3	TI-83	4 stars	Texas Instruments TI-83 Plus Graphing Calculator	1
4	Handy calculator, solid and functional	4 stars	Texas Instruments TI-83 Plus Graphing Calculator	1
...	...	...	...	...
7172	good	4 stars	Canon PIXMA MG2520 All-In-One Inkjet Printer	1
7173	ok	4 stars	Canon PIXMA MG2520 All-In-One Inkjet Printer	1
7174	A+	4 stars	Canon PIXMA MG2520 All-In-One Inkjet Printer	1
7175	Good printer but uses way to much ink	4 stars	Canon PIXMA MG2520 All-In-One Inkjet Printer	1
7176	not bad for the price. but no wifi no usb cabl...	3 stars	Canon PIXMA MG2520 All-In-One Inkjet Printer	0

7177 rows × 4 columns

In [27]:
```
# Reading DataFrame into CSV

eBay.to_csv('eBay.csv', index=False)
```

In [3]:
```
# Reading in DataFrame

eBay = pd.read_csv('eBay.csv')
```

Cleaning Data

In [4]:
```
# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison
tw_punct = punctuation - {"#"}

# Stopwords
sw = stopwords.words("english")

# Two useful regex
whitespace_pattern = re.compile(r"\s+")
hashtag_pattern = re.compile(r"^#[0-9a-zA-Z]+")

# full set of emojis
all_language_emojis = set()

for country in emoji.UNICODE_EMOJI :
    for em in emoji.UNICODE_EMOJI[country] :
        all_language_emojis.add(em)
```

In [5]:
```
#function to calculate descriptive stats

def descriptive_stats(tokens, num_tokens = 5, verbose=True) :
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical_diversity),
        and num_tokens most common tokens. Return a list with the number of tokens, number
        of unique tokens, lexical diversity, and number of characters.

    """

    # Fill in the correct values here.
    tokes = tokens.split()
    num_tokens = sum(map(len, (s.split() for s in tokes)))
    num_unique_tokens = len(set(w.lower() for w in tokes))
    lexical_diversity = num_unique_tokens / num_tokens
    num_characters = sum(list(map(len, tokes)))

    if verbose :
        print(f"There are {num_tokens} tokens in the data.")
        print(f"There are {num_unique_tokens} unique tokens in the data.")
        print(f"There are {num_characters} characters in the data.")
        print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")

        # print the five most common tokens

    return([num_tokens, num_unique_tokens,
            lexical_diversity,
            num_characters])
```

In [6]:
```
#other useful functions from previous modules
def is_emoji(s):
    return(s in all_language_emojis)

def contains_emoji(s):

    s = str(s)
    emojis = [ch for ch in s if is_emoji(ch)]

    return(len(emojis) > 0)


def remove_stop(tokens) :
    # modify this function to remove stopwords
    tokens_wo_sw = []
    for w in tokens:
        if w.lower() not in sw:
            tokens_wo_sw.append(w)
    return(tokens_wo_sw)

def remove_punctuation(text, punct_set=tw_punct):
    for ele in text:
        if ele in punct_set:
            text = text.replace(ele, "")
    return(text)

def tokenize(text) :
    """ Splitting on whitespace rather than the book's tokenize function. That
        function will drop tokens like '#hashtag' or '2A', which we need for Twitter. """

    # modify this function to return tokens
    text = text.split()
    return(text)

def prepare(text, pipeline) :
    tokens = str(text)

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

In [7]:
```
my_pipeline = [str.lower, remove_punctuation, tokenize, remove_stop]
```

In [8]:
```
clean_eBay = []

for review in range(len(eBay)):
    temp = prepare(eBay['Review Title'][review], my_pipeline)
    clean_eBay.append(temp)
```

Descriptive Statistics

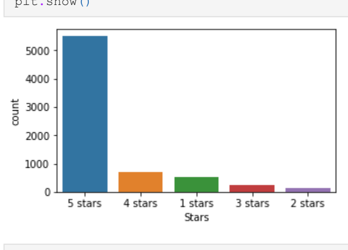
In [9]:
```
descriptive_stats(str(clean_eBay), verbose=True)
```

```
There are 16880 tokens in the data.
There are 3481 unique tokens in the data.
There are 160626 characters in the data.
The lexical diversity is 0.206 in the data.
```

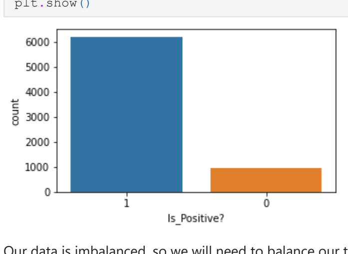
Out[9]:
```
[16880, 3481, 0.20622037914691943, 160626]
```

In [10]:
```
# Adding cleaned text to data frame and removing rows without clean data

eBay['Clean Title'] = clean_eBay
eBay = eBay[eBay['Clean Title'].str.len() != 0]
```

In [39]:
```
# Seeing Star rating distribution and Is_Positive? distribution

plt.figure(figsize=(5, 3))
sns.countplot(data=eBay, x='Stars', order = eBay['Stars'].value_counts().index)
plt.show()
```



In [40]:
```
plt.figure(figsize=(5, 3))
sns.countplot(data=eBay, x='Is_Positive?', order = eBay['Is_Positive?'].value_counts().index)
plt.show()
```



Our data is imbalanced, so we will need to balance our training set prior to modeling.

Building Models

Creating sets for model buidling

In [41]:
```
# Splitting Data into testing, validation, and training sets

X = eBay['Clean Title']
y = eBay['Is_Positive?']
train_X, rem_X, train_y, rem_y = train_test_split(X, y, train_size=0.7, random_state=42, stratify = y)
valid_X, test_X, valid_y, test_y = train_test_split(rem_X, rem_y, train_size=0.5, random_state=42, stratify
```

In [42]:
```
# Looking at the distribution, we will need to undersample the majority (1s)

train_y.value_counts()
```

Out[42]:
```
1    4344
0     654
Name: Is_Positive?, dtype: int64
```

In [43]:
```
# randomly picking samples from positive reviews

eBay_test = pd.concat([train_X, train_y], axis = 1)

positive_indices = eBay_test[eBay_test['Is_Positive?'] == 1].index
random_indices = np.random.choice(positive_indices, 637, replace=False)
positive_sample = eBay_test.loc[random_indices]
```

In [44]:
```
# Joining the data to get our new training set

new_train = pd.concat([positive_sample, eBay_test[eBay_test['Is_Positive?'] == 0]], axis = 0)
new_train['Is_Positive?'].value_counts()
```

Out[44]:
```
0    654
1    637
Name: Is_Positive?, dtype: int64
```

Altering data so it can work with our Classification models

In [45]:
```
# splitting into X and y

new_train_X = new_train['Clean Title']
new_train_y = new_train['Is_Positive?']
```

In [46]:
```
# formating data so it is no longer in list form

new_x = []

for i in new_train_X.index:
    new_x.append(' '.join(new_train_X[i]))

new_train_y.reset_index(drop=True,inplace=True)
```

In [47]:
```
x_df = []
x_df = pd.DataFrame(x_df)
x_df['Clean Title'] = pd.DataFrame(new_x)
```

In [48]:
```
# new x training set

new_train_X = x_df
new_train_X = new_train_X['Clean Title']
new_train_X
```

Out[48]:
```
0                       perfect
1                   ps5 amazing
2           fast great graphics
3       perfect like description
4                         happy
                 ...
1286           original product
1287      calculator doesnt work
1288                     unfair
1289                      fakes
1290                 严重怀疑是华强北仿冒品
Name: Clean Title, Length: 1291, dtype: object
```

In [49]:
```
# making sure we still have all our data

new_train_y
```

Out[49]:
```
0       1
1       1
2       1
3       1
4       1
       ..
1286    0
1287    0
1288    0
1289    0
1290    0
Name: Is_Positive?, Length: 1291, dtype: int64
```

In [50]:
```
# performing the same steps for our Validation Set

new_x = []
new_valid_y = valid_y

for i in valid_X.index:
    new_x.append(' '.join(valid_X[i]))

new_valid_y.reset_index(drop=True,inplace=True)

x_df = []
x_df = pd.DataFrame(x_df)
x_df['Clean Title'] = pd.DataFrame(new_x)
new_valid_X = x_df
new_valid_X = new_valid_X['Clean Title']
new_valid_X
```

Out[50]:
```
0             great purchase
1                      great
2          shouldn't purchased
3                 airpods pro
4                       good
                 ...
1066                     ps5
1067                 awesome
1068      great items around
1069        canon pixma 2520
1070                     ps5
Name: Clean Title, Length: 1071, dtype: object
```

In [51]:
```
# making sure we have the same amount of data

new_valid_y
```

Out[51]:
```
0       1
1       1
2       0
3       1
4       0
       ..
1066    1
1067    1
1068    1
1069    1
1070    1
Name: Is_Positive?, Length: 1071, dtype: int64
```

Bulding LinearSVC Model

In [52]:
```
# HyperTuning Parameters with Validation Set

tfidf = TfidfVectorizer(min_df=10, ngram_range=(1, 2), stop_words="english")
X_train_tf = tfidf.fit_transform(new_train_X)

model1 = LinearSVC(random_state=0, tol=1e-5)
model1.fit(X_train_tf, new_train_y)


X_valid_tf = tfidf.transform(new_valid_X)
Y_pred = model1.predict(X_valid_tf)
print('Accuracy Score - ', accuracy_score(valid_y, Y_pred))
print(classification_report(valid_y, Y_pred))
```

```
Accuracy Score -  0.6526610644257703
              precision    recall  f1-score   support

           0       0.25      0.86      0.39       140
           1       0.97      0.62      0.76       931

    accuracy                           0.65      1071
   macro avg       0.61      0.74      0.57      1071
weighted avg       0.87      0.65      0.71      1071
```

In [53]:
```
# performing the same steps for our test set

new_x = []
new_test_y = test_y

for i in test_X.index:
    new_x.append(' '.join(test_X[i]))

new_test_y.reset_index(drop=True,inplace=True)

x_df = []
x_df = pd.DataFrame(x_df)
x_df['Clean Title'] = pd.DataFrame(new_x)
new_test_X = x_df
new_test_X = new_test_X['Clean Title']
```

In [54]:
```
# Using test set

X_test_tf = tfidf.transform(new_test_X)
Y_pred = model1.predict(X_test_tf)
print('Accuracy Score - ', accuracy_score(test_y, Y_pred))
print(classification_report(test_y, Y_pred))
```

```
Accuracy Score -  0.6548507462686567
              precision    recall  f1-score   support

           0       0.25      0.84      0.39       141
           1       0.96      0.63      0.76       931

    accuracy                           0.65      1072
   macro avg       0.61      0.74      0.58      1072
weighted avg       0.87      0.65      0.71      1072
```

Logistic Regression

In [55]:
```
lr = LogisticRegression()
lr.fit(X_train_tf, new_train_y)
```

Out[55]:
```
▾ LogisticRegression
LogisticRegression()
```

In [56]:
```
Y_pred = lr.predict(X_valid_tf)
print('Accuracy Score - ', accuracy_score(valid_y, Y_pred))
print(classification_report(valid_y, Y_pred))
```

```
Accuracy Score -  0.6507936507936508
              precision    recall  f1-score   support

           0       0.26      0.89      0.40       140
           1       0.97      0.62      0.75       931

    accuracy                           0.65      1071
   macro avg       0.62      0.75      0.58      1071
weighted avg       0.88      0.65      0.71      1071
```

In [57]:
```
# Using test set
Y_pred = lr.predict(X_test_tf)
print('Accuracy Score - ', accuracy_score(test_y, Y_pred))
print(classification_report(test_y, Y_pred))
```

```
Accuracy Score -  0.6455223880597015
              precision    recall  f1-score   support

           0       0.25      0.86      0.39       141
           1       0.97      0.61      0.75       931

    accuracy                           0.65      1072
   macro avg       0.61      0.74      0.57      1072
weighted avg       0.87      0.65      0.70      1072
```

Naïve Bayes

In [58]:
```
# Vectorize text reviews to numbers

nb = GaussianNB()

nb.fit(X_train_tf.todense(), new_train_y)
```

```
C:\Users\lenny\anaconda3\envs\my-env-for-ads509\lib\site-packages\sklearn\utils\validation.py:727: FutureWar
ning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy arra
y with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.h
tml
  warnings.warn(
```

Out[58]:
```
▾ GaussianNB
GaussianNB()
```

In [86]:
```
Y_pred = nb.predict(X_valid_tf.todense())
print('Accuracy Score - ', accuracy_score(valid_y, Y_pred))
print(classification_report(valid_y, Y_pred))
```

```
Accuracy Score -  0.8524743230625583
              precision    recall  f1-score   support

           0       0.42      0.32      0.36       140
           1       0.90      0.93      0.92       931

    accuracy                           0.85      1071
   macro avg       0.66      0.63      0.64      1071
weighted avg       0.84      0.85      0.84      1071
```

```
C:\Users\lenny\anaconda3\envs\my-env-for-ads509\lib\site-packages\sklearn\utils\validation.py:727: FutureWar
ning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy arra
y with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.h
tml
  warnings.warn(
```

In [88]:
```
Y_pred = nb.predict(X_test_tf.todense())
print('Accuracy Score - ', accuracy_score(test_y, Y_pred))
print(classification_report(test_y, Y_pred))
```

```
Accuracy Score -  0.8526119402985075
              precision    recall  f1-score   support

           0       0.42      0.30      0.35       141
           1       0.90      0.94      0.92       931

    accuracy                           0.85      1072
   macro avg       0.66      0.62      0.63      1072
weighted avg       0.84      0.85      0.84      1072
```

```
C:\Users\lenny\anaconda3\envs\my-env-for-ads509\lib\site-packages\sklearn\utils\validation.py:727: FutureWar
ning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy arra
y with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.h
tml
  warnings.warn(
```

K- Nearest Neighbours

In [61]:
```
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X_train_tf, new_train_y)
```

Out[61]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=15)
```

In [83]:
```
Y_pred = knn.predict(X_valid_tf)
print('Accuracy Score - ', accuracy_score(valid_y, Y_pred))
print(classification_report(valid_y, Y_pred))
```

```
Accuracy Score -  0.6274509803921569
              precision    recall  f1-score   support

           0       0.24      0.88      0.38       140
           1       0.97      0.59      0.73       931

    accuracy                           0.63      1071
   macro avg       0.61      0.73      0.56      1071
weighted avg       0.88      0.63      0.69      1071
```

In [84]:
```
Y_pred = knn.predict(X_test_tf)
print('Accuracy Score - ', accuracy_score(test_y, Y_pred))
print(classification_report(test_y, Y_pred))
```

```
Accuracy Score -  0.6184701492537313
              precision    recall  f1-score   support

           0       0.24      0.84      0.37       141
           1       0.96      0.58      0.73       931

    accuracy                           0.62      1072
   macro avg       0.60      0.71      0.55      1072
weighted avg       0.87      0.62      0.68      1072
```

Random Forest

In [72]:
```
rf = RandomForestClassifier(max_depth=50, random_state=0)
rf.fit(X_train_tf, new_train_y)
```

Out[72]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=50, random_state=0)
```

In [79]:
```
Y_pred = rf.predict(X_valid_tf)
print('Accuracy Score - ', accuracy_score(valid_y, Y_pred))
print(classification_report(valid_y, Y_pred))
```

```
Accuracy Score -  0.6657329598506069
              precision    recall  f1-score   support

           0       0.27      0.89      0.41       140
           1       0.97      0.63      0.77       931

    accuracy                           0.67      1071
   macro avg       0.62      0.76      0.59      1071
weighted avg       0.88      0.67      0.72      1071
```

In [80]:
```
Y_pred = rf.predict(X_test_tf)
print('Accuracy Score - ', accuracy_score(test_y, Y_pred))
print(classification_report(test_y, Y_pred))
```

```
Accuracy Score -  0.648320895522388
              precision    recall  f1-score   support

           0       0.25      0.86      0.39       141
           1       0.97      0.62      0.75       931

    accuracy                           0.65      1072
   macro avg       0.61      0.74      0.57      1072
weighted avg       0.87      0.65      0.71      1072
```

Support Vector Machines

In [76]:
```
svm = svm.SVC()
svm.fit(X_train_tf, new_train_y)
```

Out[76]:
```
▾ SVC
SVC()
```

In [81]:
```
Y_pred = svm.predict(X_valid_tf)
print('Accuracy Score - ', accuracy_score(valid_y, Y_pred))
print(classification_report(valid_y, Y_pred))
```

```
Accuracy Score -  0.6573295985060691
              precision    recall  f1-score   support

           0       0.26      0.86      0.40       140
           1       0.97      0.63      0.76       931

    accuracy                           0.66      1071
   macro avg       0.61      0.75      0.58      1071
weighted avg       0.88      0.66      0.71      1071
```

In [82]:
```
Y_pred = svm.predict(X_test_tf)
print('Accuracy Score - ', accuracy_score(test_y, Y_pred))
print(classification_report(test_y, Y_pred))
```

```
Accuracy Score -  0.6632462686567164
              precision    recall  f1-score   support

           0       0.26      0.84      0.40       141
           1       0.96      0.64      0.77       931

    accuracy                           0.66      1072
   macro avg       0.61      0.74      0.58      1072
weighted avg       0.87      0.66      0.72      1072
```

Topic Modeling: LSA

In this section, we will try to find 5 topics that are divided among the 5 different products that were in our data set: Calculator, Apple AirPods, PS5, Flashlight, and Printor

In [91]:
```
def display_topics(model, features, no_top_words=5):
    for topic, words in enumerate(model.components_):
        total = words.sum()
        largest = words.argsort()[::-1] # invert sort order
        print("\nTopic %02d" % topic)
        for i in range(0, no_top_words):
            print("  %s (%2.2f)" % (features[largest[i]], abs(words[largest[i]]*100.0/total)))
```

In [92]:
```
# performing the same steps for all the X data

new_x = []

for i in X.index:
    new_x.append(' '.join(X[i]))

x_df = []
x_df = pd.DataFrame(x_df)
x_df['Clean Title'] = pd.DataFrame(new_x)
new_X = x_df
new_X = new_X['Clean Title']
```

In [93]:
```
tfidf_text_vectorizer = tfidf.fit_transform(new_X)
tfidf_text_vectorizer.shape
```

Out[93]:
```
(7141, 279)
```

In [104…
```
svd_para_model = TruncatedSVD(n_components = 5, random_state=42)
W_svd_para_matrix = svd_para_model.fit_transform(tfidf_text_vectorizer)
H_svd_para_matrix = svd_para_model.components_

display_topics(svd_para_model, tfidf.get_feature_names_out())
```

```
Topic 00
  great (35.47)
  product (7.69)
  great product (7.16)
  good (3.23)
  calculator (2.88)

Topic 01
  good (58.87)
  good product (6.08)
  product (5.46)
  calculator (2.05)
  good calculator (1.90)

Topic 02
  ps5 (83.03)
  awesome (4.02)
  love (2.86)
```

```
  love ps5 (1.96)

Topic 03
  awesome (55.29)
  calculator (8.29)
  product (3.95)
  ti83 (2.63)
  awesome product (2.27)

Topic 04
  calculator (39.65)
  ti83 (13.24)
  great calculator (9.20)
  plus (6.77)
  ti83 plus (6.29)
```

There are only really two topics seen here, the calculator (00, 01, 03, 04) and the PS5 (02).

Topic Modeling: LDA

In [97]:
```python
count_text_vectorizer = CountVectorizer(stop_words=stopwords, min_df=5, max_df=0.7)
count_text_vectors = count_text_vectorizer.fit_transform(new_X)
count_text_vectors.shape
```

```
C:\Users\lenny\anaconda3\envs\my-env-for-ads509\lib\site-packages\sklearn\feature_extraction\text.py:404: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['ll', 've'] not in stop_words.
  warnings.warn(
```

Out[97]:
```
(7141, 358)
```

In [98]:
```python
lda_text_model = LatentDirichletAllocation(n_components = 5, random_state=42)
W_lda_text_matrix = lda_text_model.fit_transform(count_text_vectors)
H_lda_text_matrix = lda_text_model.components_
```

In [103…
```python
# Call `display_topics` on your fitted model here
display_topics(lda_text_model, count_text_vectorizer.get_feature_names_out())
```

```
Topic 00
  ps5 (17.00)
  love (9.78)
  excellent (9.25)
  printer (5.64)
  buy (4.51)

Topic 01
  calculator (16.12)
  great (10.45)
  best (7.48)
  purchase (3.85)
  graphing (3.38)

Topic 02
  great (29.72)
  product (13.35)
  awesome (7.39)
  price (4.79)
  perfect (4.19)

Topic 03
  good (29.21)
  airpods (8.78)
  pro (4.38)
  system (3.19)
  work (2.78)

Topic 04
  ti83 (10.68)
  calculator (7.73)
  plus (7.11)
  amazing (5.23)
  nice (4.90)
```

Topic Modeling: NMF

In [102…
```python
nmf_text_model = NMF(n_components=5, random_state=42)
W_text_matrix = nmf_text_model.fit_transform(tfidf_text_vectorizer)
H_text_matrix = nmf_text_model.components_

display_topics(nmf_text_model, tfidf.get_feature_names_out())
```

```
Topic 00
  great (42.91)
  product (8.63)
  great product (8.57)
  great calculator (2.14)
  works (2.10)

Topic 01
  good (55.17)
  good product (5.67)
  product (5.38)
  good calculator (1.63)
  good value (1.55)

Topic 02
  ps5 (77.28)
  love (2.59)
  review (1.82)
  love ps5 (1.81)
  amazing (1.80)

Topic 03
  awesome (71.05)
  product (3.31)
  awesome product (2.85)
  airpods (1.83)
  console (1.63)

Topic 04
  calculator (26.92)
  ti83 (9.25)
  great calculator (5.74)
  plus (4.69)
  ti83 plus (4.35)
```