Importing the required libraries

In [1]:

```
import pandas as pd
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import time
from nltk.corpus import stopwords
nltk.download('stopwords')
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import precision_recall_fscore_support
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/payalrashinkar/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/payalrashinkar/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Brief: Downloading all the required packages

1. Data Preparation

Read Data

In [2]:

```
df = pd.read_csv('https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/
amazon-reviews-pds/tsv/amazon_reviews_us_Office_Products_v1_00.tsv.gz', sep='\t', on_bad_
lines='skip', low_memory=False)
```

Keep Reviews and Ratings

In [3]:

```
df_copy = df.copy()
df_copy = df_copy[['star_rating', 'review_body']].rename(columns={'star_rating':'ratings
', 'review_body':'reviews'})
df_copy.head()
```

Out[3]:

	ratings	reviews
0	5	Great product.
1	5	What's to say about this commodity item except...
2	5	Haven't used yet, but I am sure I will like it.
3	1	Although this was labeled as &#34;new&#34; the...
4	4	Gorgeous colors and easy to use

```
ratings                                     reviews
```

Brief: copying the dataframe df to df_copy and only have two columns in df_copy i.e. ratings and reviews after renaming it from start_rating and review_body respectively.

Rating count

In [4]:

```
df_copy['ratings'] = pd.to_numeric(df_copy['ratings'], errors='coerce')
df_copy_filtered = df_copy[df_copy['ratings'].notna()]
grouped_df_copy = df_copy_filtered.groupby('ratings').describe()
print(grouped_df_copy)
```

```
         reviews
           count   unique           top  freq
ratings
1.0       306967   301547  Did not work    90
2.0       138381   136977            Ok    56
3.0       193680   188441            ok   562
4.0       418348   395810          good  1119
5.0      1582704  1408137         Great  4196
```

Brief: 'Ratings' column is a string column with mixed numeric and non-numeric values, hence filtering the DataFrame to keep only rows where 'ratings' is not NaN and then proceed with the groupby and describe which outputs us with number of review count for each of the ratings as shown above.

Creating labels, 1 for positive and 0 for negative reviews and label count

In [5]:

```
df_copy['sentiment'] = df_copy['ratings'].map(lambda x: 1 if x > 3 else (0 if x <= 2 els
e None))

df_filtered = df_copy[df_copy['ratings'] != 3]

num_positive_reviews = df_filtered['sentiment'].value_counts().get(1, 0)
num_negative_reviews = df_filtered['sentiment'].value_counts().get(0, 0)
num_neutral_reviews = df_filtered['sentiment'].isna().sum()

print(f"Number of Positive Reviews: {num_positive_reviews}")
print(f"Number of Negative Reviews: {num_negative_reviews}")
print(f"Number of Neutral Reviews (Discarded): {num_neutral_reviews}")
```

```
Number of Positive Reviews: 2001183
Number of Negative Reviews: 445363
Number of Neutral Reviews (Discarded): 17
```

Brief: Creating label named sentiment, with discarding ratings with value 3 and assigning sentiment value to 0 if it is <= 2, and to 1 if it is >3 and printing the output.

Downsize the dataframe to 100,000

In [6]:

```
df_p = df_copy[df_copy['sentiment']==1].sample(n=100000, random_state=1)
df_n = df_copy[df_copy['sentiment']==0].sample(n=100000, random_state=1)
print(df_p.head())
print(df_n.head())
df1 = pd.concat([df_p, df_n]).sample(frac=1).reset_index(drop=True)
```

```
print(df1.head())
```

```
         ratings                                            reviews  sentiment
1083678      4.0  Good adhesion, they do not jam, and are sold a...        1.0
1931690      5.0  Good quality items, shipment notifications cam...        1.0
1255785      5.0  Very nice pen. Reminds me of my school days. I...        1.0
306664       4.0  I got an Epson printer that uses these cartrid...        1.0
774628       5.0                                  Works great thanks        1.0
         ratings                                            reviews  sentiment
138258       1.0  I did not open these for a few months and they...        0.0
566017       2.0  The product did not work, even after following...        0.0
1975980      1.0  I found these cartridges to be useless. First,...        0.0
471648       1.0  Not thrilled with this product.  It is very lo...        0.0
564033       1.0  The Plan's &#34;Confirmation and Contract&#34;...        0.0
   ratings                                            reviews  sentiment
0      1.0  Three words:  Bad color scans.  I have gone th...        0.0
1      2.0  They were great at first, but now the volume g...        0.0
2      1.0  I hung this calendar for one month at my place...        0.0
3      1.0  This item Its way too expensive.  I think amaz...        0.0
4      1.0  Bought this to power a Raspberry Pi but I'm ex...        0.0
```

Brief: Downsize the dataframe with 100,000 positive reviews and 100,000 negative ones and concatinate both into one dataframe df1.

2. Data Cleaning

In [7]:

```
X, y = df1['reviews'].fillna('').tolist(), df1['sentiment'].tolist()
```

Average_length_before_cleaning

In [8]:

```
# Step 1: Calculate average length before cleaning
average_length_before_cleaning = df1['reviews'].str.len().mean()
print(f"Average Length of Reviews Before Cleaning: {average_length_before_cleaning:.2f} c
haracters")
```

```
Average Length of Reviews Before Cleaning: 318.18 characters
```

In [9]:

```
# convert reviews to lower case
X = list(map(lambda x: str(x).lower(), X))
# remove HTML and URLs from reviews
X = list(map(lambda x: re.sub('<.*>', '', x), X))
X = list(map(lambda x: re.sub(r'https?://\S+', '', x), X))
# remove non-alphabetical characters
X = list(map(lambda x: re.sub('[^a-z ]', '', x), X))
# remove extra spaces
X = list(map(lambda x: re.sub(' +', ' ', x), X))
```

Brief: Performing the data cleaning process as mentioned in the comments above.

In [10]:

```
# expand contractions
contractions = {
"ain't": "am not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he would",
"he'd've": "he would have",
"he'll": "he will",
"he'll've": "he will have",
"he's": "he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how is",
"I'd": "I would",
"I'd've": "I would have",
"I'll": "I will",
"I'll've": "I will have",
"I'm": "I am",
"I've": "I have",
"isn't": "is not",
"it'd": "it would",
"it'd've": "it would have",
"it'll": "it will",
"it'll've": "it will have",
"it's": "it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she would",
"she'd've": "she would have",
"she'll": "she will",
"she'll've": "she will have",
"she's": "she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so is",
"that'd": "that would",
"that'd've": "that would have",
"that's": "that is",
"there'd": "there would",
"there'd've": "there would have",
"there's": "there is",
"they'd": "they would",
"they'd've": "they would have",
"they'll": "they will",
"they'll've": "they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what'll've": "what will have",
"what're": "what are",
"what's": "what is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where did",
"where's": "where is",
"where've": "where have",
"who'll": "who will",
"who'll've": "who will have",
"who's": "who is",
"who've": "who have",
"why's": "why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you would",
"you'd've": "you would have",
"you'll": "you will",
"you'll've": "you will have",
"you're": "you are",
"you've": "you have"
}

def func_contraction(s):
    for word in s.split(' '):
        if word in contractions.keys():
            s = re.sub(word, contractions[word], s)
    return s

X = list(map(func_contraction, X))
```

Brief: The above program defines a function func_contraction that replaces contractions (abbreviated forms of words or phrases) in a given string s with their expanded forms using a dictionary contractions. The function is then applied to each element in a list X using map, effectively expanding all contractions in the strings contained in X.

Average_length_after_cleaning

In [11]:

```
average_length_after_cleaning = pd.Series(X).str.len().mean()
print(f"Average Length of Reviews After Cleaning: {average_length_after_cleaning:.2f} cha
racters")
```

```
Average Length of Reviews After Cleaning: 255.98 characters
```

3. Preprocessing

In [12]:

```
# Sample reviews before preprocessing
sample_reviews_before = df1['reviews'].head(3).tolist()
print("Sample Reviews Before Preprocessing:\n")
for review in sample_reviews_before:
    print(review)
print()
```

```
Sample Reviews Before Preprocessing:

Three words:  Bad color scans.  I have gone through customer service purgatory with speci
al registry cleaners to install \\"new\\" drivers and still the color scans have red stre
aks and faded, blurry colors.  Basically, the unit is DOA for any color scanning.   <br /
> <br />Works great as a b&w document scanning device.  Did I mention that I am into colo
r photography? <br /> <br />Bottom line:  I'm getting a new scanner.
They were great at first, but now the volume goes up and down. You have a hard time heari
ng anyone and more than one person had told me that I sound really quite on their end too
. Also, I thought that the buttons and the display lit up because of the way they look..
They don't so very hard to read at night.
I hung this calendar for one month at my place of employment, even though the January pho
to was out of focus.  When I flipped to the February photo, I took the calendar down from
my wall.  It is a disgrace to have such poor quality pictures of Michelle Obama on displa
y.  I will not be ordering from this company again.
```

Remove the stop words

In [13]:

```
# remove stop words
stopWords =set(stopwords.words('english'))
def remvstopWords(s):
    wordlist = s.split(' ')
    newlist = []
    for word in wordlist:
        if word not in stopWords:
            newlist.append(word)
    s = ' '.join(newlist)
    return s

X = list(map(remvstopWords, X))
```

Brief: The code defines a function rmstopWords to remove English stopwords from a given string s. It then applies this function to each element in the list X using map, resulting in a new list X where each string has had its stopwords removed.

In [14]:

```
# perform lemmatization
wl = WordNetLemmatizer()
X = list(map(lambda x: ' '.join(map(wl.lemmatize, x.split(' '))), X))
```

Brief: The code applies the WordNet lemmatizer to each word in every string of the list X, replacing each word with its base or dictionary form (lemmatization), and then reconstructs each string.

In [15]:

```
# Sample reviews after preprocessing
sample_reviews_after = X[:3]
print("Sample Reviews After Preprocessing:")
for review in sample_reviews_after:
    print(review)
print()
```

```
Sample Reviews After Preprocessing:
three word bad color scan gone customer service purgatory special registry cleaner instal
```

```
l new driver still color scan red streak faded blurry color basically unit doa color scan
ning bottom line im getting new scanner
great first volume go hard time hearing anyone one person told sound really quite end als
o thought button display lit way look dont hard read night
hung calendar one month place employment even though january photo focus flipped february
photo took calendar wall disgrace poor quality picture michelle obama display ordering co
mpany
```

Calculate average length before and after preprocessing with stopwords and lemmatization

In [16]:

```
# Calculate average length before preprocessing
average_length_before_preprocessing = pd.Series(sample_reviews_before).str.len().mean()
print(f"Average Length of Reviews Before Preprocessing: {average_length_before_preprocess
ing:.2f} characters")

# Calculate average length after preprocessing
average_length_after_preprocessing = pd.Series(sample_reviews_after).str.len().mean()
print(f"Average Length of Reviews After Preprocessing: {average_length_after_preprocessin
g:.2f} characters")
```

```
Average Length of Reviews Before Preprocessing: 350.33 characters
Average Length of Reviews After Preprocessing: 182.67 characters
```

Dataset split into training and testing data set

In [17]:

```
# Divide the downsized dataset into 80% training and 20% testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

4. TF-IDF Feature Extraction

In [18]:

```
vectorizer = TfidfVectorizer()
tfidf = vectorizer.fit_transform(X_train)
tfidf_test = vectorizer.transform(X_test)
```

Brief: The code initializes a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer and then fits and transforms the training data X_train into a TF-IDF matrix. It also transforms the test data X_test into a TF-IDF matrix using the same vectorizer.

5. Perceptron

In [19]:

```
perceptron = Perceptron(random_state=1)
perceptron.fit(tfidf, y_train)
y_train_predict, y_test_predict = perceptron.predict(tfidf), perceptron.predict(tfidf_te
st)

# report accuracy, precision, recall, and f1-score on both the training and testing split
train_stats = precision_recall_fscore_support(y_train, y_train_predict, average='binary'
)
precision_train, recall_train, fscore_train = train_stats[0], train_stats[1], train_stat
s[2]

test_stats = precision_recall_fscore_support(y_test, y_test_predict, average='binary')
```

```
precision_test, recall_test, fscore_test = test_stats[0], test_stats[1], test_stats[2]

print('Perceptron, accuracy of training dataset: {:2.1%}'.format(perceptron.score(tfidf,
y_train)))
print('Perceptron, precision of training dataset: {:2.1%}'.format(precision_train))
print('Perceptron, recall of training dataset: {:2.1%}'.format(recall_train))
print('Perceptron, fscore of training dataset: {:2.1%}\n'.format(fscore_train))

print('Perceptron, accuracy of testing dataset: {:2.1%}'.format(perceptron.score(tfidf_te
st, y_test)))
print('Perceptron, precision of testing dataset: {:2.1%}'.format(precision_test))
print('Perceptron, recall of testing dataset: {:2.1%}'.format(recall_test))
print('Perceptron, fscore of testing dataset: {:2.1%}'.format(fscore_test))
```

```
Perceptron, accuracy of training dataset: 91.2%
Perceptron, precision of training dataset: 88.4%
Perceptron, recall of training dataset: 94.8%
Perceptron, fscore of training dataset: 91.5%

Perceptron, accuracy of testing dataset: 84.4%
Perceptron, precision of testing dataset: 81.4%
Perceptron, recall of testing dataset: 89.1%
Perceptron, fscore of testing dataset: 85.0%
```

Brief: The code trains a Perceptron model on the TF-IDF transformed training data (tfidf) and evaluates its performance on both training and testing datasets (tfidf and tfidf_test). It reports the accuracy, precision, recall, and F1-score for both datasets, providing a detailed assessment of the model's classification performance.

6. SVM

In [20]:

```
svm = LinearSVC(random_state=1)
svm.fit(tfidf, y_train)

y_train_predict, y_test_predict = svm.predict(tfidf), svm.predict(tfidf_test)

# report accuracy, precision, recall, and f1-score on both the training and testing split
train_stats = precision_recall_fscore_support(y_train, y_train_predict, average='binary'
)
precision_train, recall_train, fscore_train = train_stats[0], train_stats[1], train_stat
s[2]

test_stats = precision_recall_fscore_support(y_test, y_test_predict, average='binary')
precision_test, recall_test, fscore_test = test_stats[0], test_stats[1], test_stats[2]

print('SVM, accuracy of training dataset: {:2.1%}'.format(svm.score(tfidf, y_train)))
print('SVM, precision of training dataset: {:2.1%}'.format(precision_train))
print('SVM, recall of training dataset: {:2.1%}'.format(recall_train))
print('SVM, fscore of training dataset: {:2.1%}\n'.format(fscore_train))

print('SVM, accuracy of testing dataset: {:2.1%}'.format(svm.score(tfidf_test, y_test)))
print('SVM, precision of testing dataset: {:2.1%}'.format(precision_test))
print('SVM, recall of testing dataset: {:2.1%}'.format(recall_test))
print('SVM, fscore of testing dataset: {:2.1%}'.format(fscore_test))
```

```
/Users/payalrashinkar/anaconda3/lib/python3.11/site-packages/sklearn/svm/_classes.py:32:
FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Se
t the value of `dual` explicitly to suppress the warning.
  warnings.warn(
```

```
SVM, accuracy of training dataset: 93.6%
SVM, precision of training dataset: 93.9%
SVM, recall of training dataset: 93.4%
SVM, fscore of training dataset: 93.6%

SVM, accuracy of testing dataset: 88.7%
SVM, precision of testing dataset: 88.6%
```

```
SVM, recall of testing dataset: 88.8%
SVM, fscore of testing dataset: 88.7%
```

Brief: The code trains a Support Vector Machine (SVM) with a linear kernel on the TF-IDF transformed training data and then evaluates its performance on both the training and testing datasets. It calculates and prints the accuracy, precision, recall, and F1-score for both datasets, providing a comprehensive evaluation of the SVM model's effectiveness in classification tasks.

Logistic Regression

In [21]:

```
logistic = LogisticRegression(random_state=1, max_iter=200)
logistic.fit(tfidf, y_train)

y_train_predict, y_test_predict = logistic.predict(tfidf), logistic.predict(tfidf_test)

# report accuracy, precision, recall, and f1-score on both the training and testing split
train_stats = precision_recall_fscore_support(y_train, y_train_predict, average='binary'
)
precision_train, recall_train, fscore_train = train_stats[0], train_stats[1], train_stat
s[2]

test_stats = precision_recall_fscore_support(y_test, y_test_predict, average='binary')
precision_test, recall_test, fscore_test = test_stats[0], test_stats[1], test_stats[2]

print('Logistic Regression, accuracy of training dataset: {:2.1%}'.format(logistic.score(
tfidf, y_train)))
print('Logistic Regression, precision of training dataset: {:2.1%}'.format(precision_trai
n))
print('Logistic Regression, recall of training dataset: {:2.1%}'.format(recall_train))
print('Logistic Regression, fscore of training dataset: {:2.1%}\n'.format(fscore_train))

print('Logistic Regression, accuracy of testing dataset: {:2.1%}'.format(logistic.score(t
fidf_test, y_test)))
print('Logistic Regression, precision of testing dataset: {:2.1%}'.format(precision_test)
)
print('Logistic Regression, recall of testing dataset: {:2.1%}'.format(recall_test))
print('Logistic Regression, fscore of testing dataset: {:2.1%}'.format(fscore_test))
```

```
Logistic Regression, accuracy of training dataset: 90.8%
Logistic Regression, precision of training dataset: 91.2%
Logistic Regression, recall of training dataset: 90.3%
Logistic Regression, fscore of training dataset: 90.8%

Logistic Regression, accuracy of testing dataset: 89.1%
Logistic Regression, precision of testing dataset: 89.3%
Logistic Regression, recall of testing dataset: 88.7%
Logistic Regression, fscore of testing dataset: 89.0%
```

Brief: The code trains a Logistic Regression model on the TF-IDF transformed training data and evaluates its performance on both training and testing datasets. It calculates and reports the accuracy, precision, recall, and F1-score for both datasets, thereby assessing the model's classification accuracy and effectiveness in distinguishing between classes.

7. Multinomial Naive Bayes

In [22]:

```
multiNB = MultinomialNB()
multiNB.fit(tfidf, y_train)
y_train_predict, y_test_predict = multiNB.predict(tfidf), multiNB.predict(tfidf_test)
```

```
# report accuracy, precision, recall, and f1-score on both the training and testing split
train_stats = precision_recall_fscore_support(y_train, y_train_predict, average='binary'
)
precision_train, recall_train, fscore_train = train_stats[0], train_stats[1], train_stat
s[2]

test_stats = precision_recall_fscore_support(y_test, y_test_predict, average='binary')
precision_test, recall_test, fscore_test = test_stats[0], test_stats[1], test_stats[2]

print('Multi Naive Bayes, accuracy of training dataset: {:2.1%}'.format(multiNB.score(tfi
df, y_train)))
print('Multi Naive Bayes, precision of training dataset: {:2.1%}'.format(precision_train)
)
print('Multi Naive Bayes, recall of training dataset: {:2.1%}'.format(recall_train))
print('Multi Naive Bayes, fscore of training dataset: {:2.1%}\n'.format(fscore_train))

print('Multi Naive Bayes, accuracy of testing dataset: {:2.1%}'.format(multiNB.score(tfid
f_test, y_test)))
print('Multi Naive Bayes, precision of testing dataset: {:2.1%}'.format(precision_test))
print('Multi Naive Bayes, recall of testing dataset: {:2.1%}'.format(recall_test))
print('Multi Naive Bayes, fscore of testing dataset: {:2.1%}'.format(fscore_test))
```

```
Multi Naive Bayes, accuracy of training dataset: 88.2%
Multi Naive Bayes, precision of training dataset: 89.7%
Multi Naive Bayes, recall of training dataset: 86.4%
Multi Naive Bayes, fscore of training dataset: 88.0%

Multi Naive Bayes, accuracy of testing dataset: 85.9%
Multi Naive Bayes, precision of testing dataset: 87.2%
Multi Naive Bayes, recall of testing dataset: 84.0%
Multi Naive Bayes, fscore of testing dataset: 85.6%
```

Brief: The code trains a Multinomial Naive Bayes model on the TF-IDF transformed training data and evaluates its performance on both the training and testing datasets. It calculates and reports the model's accuracy, precision, recall, and F1-score for both datasets, providing insights into the effectiveness and reliability of the model in classifying the data.