In [1]:
```python
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
import re
from bs4 import BeautifulSoup
from sklearn.preprocessing import StandardScaler
import gzip
import requests
from io import BytesIO, TextIOWrapper
import random
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\91912\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91912\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\91912\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\91912\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:
```python
#! pip install bs4
#! pip install scikit-learn
#! pip install contractions
#Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Beauty_v1_00.tsv.gz
```

Read Data

In [3]:
```python
# Storing URL of the dataset in url
url = "https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Office_Products_v1_00.tsv.gz"

# Here we will fetch the dataset
response = requests.get(url)
response.raise_for_status()  #  Had to use this to ensure that the request was successful

# Now we need to decompress the content to use it for further tasks
with gzip.GzipFile(fileobj=BytesIO(response.content)) as gz:
    amazon_review_data = pd.read_csv(
        TextIOWrapper(gz, encoding='utf-8'),  # we set this explicitly to UTF-8 encoding because it was giving error for some values which was not in the proper format
        sep='\t',
        on_bad_lines='skip',  # Here we will skip problematic lines
        low_memory=False      # Lastly we do this to improve performance for large datasets
    )

#Finally we will print few lines to make sure that the data has been successfully loaded
amazon_review_data.head()
```

Out[3]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_votes	total_votes	vine	verified_purchase	review_headline	revie
0	US	43081963	R18RVCKGH1SSI9	B001BM2MAC	307809868	Scotch Cushion Wrap 7961, 12 Inches x 100 Feet	Office Products	5	0.0	0.0	N	Y	Five Stars	Great p
1	US	10951564	R3L4L6LW1PUOFY	B00DZYEXPQ	75004341	Dust-Off Compressed Gas Duster, Pack of 4	Office Products	5	0.0	1.0	N	Y	Phffffffft, Phffffffft. Lots of air, and it's C...	What's ab com item e
2	US	21143145	R2J8AWXWTDX2TF	B00RTMUHDW	529689027	Amram Tagger Standard Tag Attaching Tagging Gu...	Office Products	5	0.0	0.0	N	Y	but I am sure I will like it.	Haven yet, b sure I wi
3	US	52782374	R1PR37BR7G3M6A	B00D7H8XB6	868449945	AmazonBasics 12-Sheet High-Security Micro-Cut ...	Office Products	1	2.0	3.0	N	Y	and the shredder was dirty and the bin was par...	Althou was lab "ne
4	US	24045652	R3BDDDZMZBZDPU	B001XCWP34	33521401	Derwent Colored Pencils, Inktense Ink Pencils,...	Office Products	4	0.0	0.0	N	Y	Four Stars	Go colors ar

Keep Reviews and Ratings

In [4]:
```python
from sklearn.model_selection import train_test_split

# A small function to assign star rating to binary labels
def assign_label(star_rating):
    if star_rating > 3:
        return 1
    elif star_rating <= 2:
        return 0
    else:
        return None

# we will filter relevant columns and will keep only Reviews and Ratings columns and drop the rest of the columns
amazon_review_data = amazon_review_data[['review_body', 'star_rating']].dropna()

# We need to make sure that star_rating is numeric for further parts in code
amazon_review_data['star_rating'] = pd.to_numeric(amazon_review_data['star_rating'], errors='coerce')

# Print statistics for the three classes: positive, negative, and neutral reviews
positive_count = (amazon_review_data['star_rating'] > 3).sum()
negative_count = (amazon_review_data['star_rating'] <= 2).sum()
neutral_count = (amazon_review_data['star_rating'] == 3).sum()
```

```python
# Print the counts for each class in the requested format
print(f"Positive reviews: {positive_count}, Negative reviews: {negative_count}, Neutral reviews: {neutral_count}")

# Now we map ratings to binary labels
amazon_review_data['label'] = amazon_review_data['star_rating'].apply(assign_label)

# As sid in the assignment, we need to drop neutral reviews
amazon_review_data = amazon_review_data.dropna(subset=['label'])

# We will downsample the dataset to 100,000 positive and negative reviews according to the assignment
# Also we will be using random state = 42 such that we can get the consistent results
# Assign 1 for positive reviews and 0 for negative reviews
positive_reviews = amazon_review_data[amazon_review_data['label'] == 1].sample(100000, random_state=42)
negative_reviews = amazon_review_data[amazon_review_data['label'] == 0].sample(100000, random_state=42)

# We will now combine the downsampled amazon_review_data
balanced_data = pd.concat([positive_reviews, negative_reviews])

# Finally the main task where we will split into training and testing datasets
train_data, test_data = train_test_split(balanced_data, test_size=0.2, random_state=42)

# Lastly, we print dataset statistics as asked
print("Training size:", len(train_data))
print("Testing size:", len(test_data))
print("Class distribution in training:", train_data['label'].value_counts())
```

```
Positive reviews: 2001052, Negative reviews: 445348, Neutral reviews: 193680
Training size: 160000
Testing size: 40000
Class distribution in training: label
1.0    80007
0.0    79993
Name: count, dtype: int64
```

Data Cleaning

- convert the all reviews into the lower case.
- remove the HTML and URLs from the reviews
- remove non-alphabetical characters
- remove extra spaces
- perform contractions on the reviews, e.g., won't → will not. Include as many contractions in English that you can think

In [5]:
```python
contractions_dict = {
    "can't": "cannot", "won't": "will not", "i'm": "i am",
    "you're": "you are", "he's": "he is", "she's": "she is",
    "it's": "it is", "we're": "we are", "they're": "they are",
    "isn't": "is not", "aren't": "are not", "wasn't": "was not",
    "weren't": "were not", "don't": "do not", "doesn't": "does not",
    "didn't": "did not", "haven't": "have not", "hasn't": "has not",
    "hadn't": "had not", "wouldn't": "would not", "shouldn't": "should not",
    "couldn't": "could not", "mightn't": "might not", "mustn't": "must not",
    "let's": "let us", "that's": "that is", "what's": "what is",
    "who's": "who is", "there's": "there is", "here's": "here is",
    "how's": "how is", "where's": "where is", "why's": "why is",
    "when's": "when is", "weren't": "were not", "could've": "could have",
    "should've": "should have", "would've": "would have", "might've": "might have",
    "must've": "must have", "we've": "we have", "you've": "you have",
    "they've": "they have", "who've": "who have", "i've": "i have",
    "hasn't": "has not", "you'll": "you will", "he'll": "he will",
    "she'll": "she will", "it'll": "it will", "we'll": "we will",
    "they'll": "they will", "i'll": "i will", "that'll": "that will",
    "there'll": "there will", "who'll": "who will", "what'll": "what will",
    "won't": "will not", "shan't": "shall not", "who'd": "who would",
    "it'd": "it would", "we'd": "we would", "they'd": "they would",
    "you'd": "you would", "she'd": "she would", "he'd": "he would",
    "i'd": "i would", "they're": "they are", "we're": "we are",
    "you're": "you are", "i'm": "i am", "he's": "he is",
    "she's": "she is", "it's": "it is", "ain't": "is not",
    "y'all": "you all", "gonna": "going to", "wanna": "want to",
    "gotta": "got to", "lemme": "let me", "gimme": "give me",
    "dunno": "do not know", "outta": "out of", "sorta": "sort of",
    "kinda": "kind of", "oughta": "ought to", "coulda": "could have",
    "woulda": "would have", "shoulda": "should have", "how'd": "how did",
    "why'd": "why did", "where'd": "where did", "when'd": "when did",
    "y'know": "you know", "c'mon": "come on", "how're": "how are",
    "what're": "what are", "who're": "who are", "where're": "where are",
    "when're": "when are", "why're": "why are", "there're": "there are",
    "that'd": "that would", "this'll": "this will", "it'll've": "it will have",
    "we'll've": "we will have", "who'll've": "who will have",
    "it'd've": "it would have", "nothin'": "nothing", "somethin'": "something",
    "everythin'": "everything", "givin'": "giving", "movin'": "moving",
    "y'all've": "you all have", "y'all'd": "you all would",
    "ain'tcha": "are not you", "didn'tcha": "did not you",
    "ya'll": "you all", "ain'tcha": "are not you", "mightn't've": "might not have",
    "mustn't've": "must not have", "shouldn't've": "should not have",
    "you'd've": "you would have", "there'd've": "there would have",
    "who'd've": "who would have", "what'd've": "what would have"
}
```

In [6]:
```python
# Custom Function to expand contractions based on whatever is needed
def expand_contractions(text):
    words = text.split()
    expanded_words = []
    for word in words:
        # We will check if the word is in the contractions dictionary
        if word in contractions_dict:
            # Here we replace the word with its expanded form
            expanded_words.append(contractions_dict[word])
        else:
            # Keep the word as is
            expanded_words.append(word)
    return " ".join(expanded_words)
```

In [7]:
```python
# Updated cleaning function
def clean_text(text):
    text = text.lower()  # We will convert text to lowercase
    text = re.sub(r'https?://\S+|www\.\S+', '', text)  # We will Remove URLs
    text = re.sub(r'<.*?>', '', text)  # We will remove HTML tags
    text = re.sub(r'[^a-z\s]', '', text)  # We will remove non-alphabetical characters
    text = re.sub(r'\s+', ' ', text).strip()  # We will remove extra spaces
    text = expand_contractions(text)  # We will expand contractions manually
```

```python
    return text

# Apply cleaning to training and testing datasets
train_data['cleaned_review'] = train_data['review_body'].apply(clean_text)
test_data['cleaned_review'] = test_data['review_body'].apply(clean_text)

# Print average length before and after cleaning
print("Average length before cleaning:", train_data['review_body'].str.len().mean())
print("Average length after cleaning:", train_data['cleaned_review'].str.len().mean())
```

```
Average length before cleaning: 317.4268625
Average length after cleaning: 299.67749375
```

In [8]:
```python
#Helper Cell for debugging
train_data.head()
train_data['cleaned_review'].iloc[0]
```

Out[8]:
```
'do disappointed that the chocolate was melted and stuck to the plastic impossible to remove perhaps shipping'
```

In [9]:
```python
#Helper Cell for debugging and checking if the cleaning is happening properly or not
sample_review = train_data['review_body'].iloc[0]
cleaned_review_1 = clean_text(sample_review)
print("Original Review:", sample_review)
print("Cleaned Review (Function):", cleaned_review_1)
print("Cleaned Review (Loaded):", train_data['cleaned_review'].iloc[0])
```

```
Original Review: Do disappointed that the chocolate was melted and stuck to the plastic. Impossible to remove. Perhaps shipping?
Cleaned Review (Function): do disappointed that the chocolate was melted and stuck to the plastic impossible to remove perhaps shipping
Cleaned Review (Loaded): do disappointed that the chocolate was melted and stuck to the plastic impossible to remove perhaps shipping
```

Pre-processing

Remove the Stop Words

In [10]:
```python
from nltk.corpus import stopwords # type: ignore

# We will set and define stop words
stop_words = set(stopwords.words('english'))

# This function is to remove stop words
def remove_stop_words(text):
    tokens = word_tokenize(text)  # Here we will tokenize the text
    tokens = [word for word in tokens if word not in stop_words]  # Here we will filter out stop words
    return tokens
```

Perform Lemmatization

In [11]:
```python
from nltk.stem import WordNetLemmatizer # type: ignore
# We will initialize lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to perform lemmatization
def lemmatize_tokens(tokens):
    lemmatized = [lemmatizer.lemmatize(word) for word in tokens]  # Here we will lemmatize each token
    return ' '.join(lemmatized)  # Here we will join tokens back into a string
```

Print three sample reviews before and after data cleaning + preprocessing

In [12]:
```python
# We need to ensure that all values in 'cleaned_review' are strings and replace NaN values with an empty string
train_data['cleaned_review'] = train_data['cleaned_review'].fillna("").astype(str)
test_data['cleaned_review'] = test_data['cleaned_review'].fillna("").astype(str)

# This is the preprocessing function where we will call the other functions(like the remove stop word function and lemmatize token function) for preprocessing our data
def preprocess_text(text):
    tokens = remove_stop_words(text)  # Remove stop words
    processed_text = lemmatize_tokens(tokens)  # Perform lemmatization
    return processed_text

# Apply preprocessing to the dataset
train_data['preprocessed_review'] = train_data['cleaned_review'].apply(preprocess_text)
test_data['preprocessed_review'] = test_data['cleaned_review'].apply(preprocess_text)

# Print three random samples before and after preprocessing
sample_indices = random.sample(range(len(train_data)), 3)  # Select 3 random indices
for i in sample_indices:
    print(f"Sample {i + 1}:")
    print("Before preprocessing:", train_data['cleaned_review'].iloc[i])
    print("After preprocessing:", train_data['preprocessed_review'].iloc[i])
    print("-" * 50)
```

```
Sample 52190:
Before preprocessing: this was detected as a cartridge that was not accepted by the lexmark printer it was made for had to get another that had a chip that the printer would accept was with
out a printer for weeks
After preprocessing: detected cartridge accepted lexmark printer made get another chip printer would accept without printer week
--------------------------------------------------
Sample 143446:
Before preprocessing: the color cartridges work just fine for me so far but the black cartridges do not work at all the quality is terrible and looks blurredwhen it prints also greatly redu
ced the quality of the color cartridgethe color was fine until i used the compatible black cartridge then the color all but stopped working ill keep these to use if im in a bind but i wont
reorder
After preprocessing: color cartridge work fine far black cartridge work quality terrible look blurredwhen print also greatly reduced quality color cartridgethe color fine used compatible bl
ack cartridge color stopped working ill keep use im bind wont reorder
--------------------------------------------------
Sample 103578:
Before preprocessing: i havent used it much but so far it works great only one time did i have to reset it to my wireless router
After preprocessing: havent used much far work great one time reset wireless router
--------------------------------------------------
```

In [13]:
```python
# Handle NaN values in 'preprocessed_review'
train_data['preprocessed_review'] = train_data['preprocessed_review'].fillna("")
test_data['preprocessed_review'] = test_data['preprocessed_review'].fillna("")

# Printing the average length after preprocessing the data
print("Average length before preprocessing:", train_data['cleaned_review'].str.len().mean())
print("Average length after preprocessing:", train_data['preprocessed_review'].str.len().mean())
```

```
Average length before preprocessing: 299.67749375
Average length after preprocessing: 190.75154375
```

TF-IDF Feature Extraction

In [14]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

# We will Extract TF-IDF features
tfidf = TfidfVectorizer(max_features=5000)
X_train = tfidf.fit_transform(train_data['preprocessed_review']).toarray()
X_test = tfidf.transform(test_data['preprocessed_review']).toarray()

y_train = train_data['label']
y_test = test_data['label']
```

Perceptron

In [15]:
```python
from sklearn.linear_model import Perceptron

# Training Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, y_train)

# Predictions being made here
y_train_pred = perceptron.predict(X_train)
y_test_pred = perceptron.predict(X_test)

# Printing Metrics
print("Perceptron Metrics:")
print("Accuracy Score Train:",accuracy_score(y_train, y_train_pred))
print("Precision Score Train:",precision_score(y_train, y_train_pred))
print("Recall Score Train:",recall_score(y_train, y_train_pred))
print("F1 Score Train:",f1_score(y_train, y_train_pred))
print("Accuracy Score Test:",accuracy_score(y_test, y_test_pred))
print("Precision Score Test:",precision_score(y_test, y_test_pred))
print("Recall Score Test:",recall_score(y_test, y_test_pred))
print("F1 Score Test:",f1_score(y_test, y_test_pred))
```

```
Perceptron Metrics:
Accuracy Score Train: 0.8316375
Precision Score Train: 0.7595949713838478
Recall Score Train: 0.970440086492432
F1 Score Train: 0.8521693319138194
Accuracy Score Test: 0.821125
Precision Score Test: 0.7494365430947385
Recall Score Test: 0.9646376231681089
F1 Score Test: 0.8435278938045356
```

SVM

In [16]:
```python
from sklearn.svm import LinearSVC

# Training LinearSVC
linear_svc = LinearSVC()
linear_svc.fit(X_train, y_train)

# Predictions being made here
y_train_pred = linear_svc.predict(X_train)
y_test_pred = linear_svc.predict(X_test)

# Printing Metrics
print("LinearSVM Metrics:")
print("Accuracy Score Train:",accuracy_score(y_train, y_train_pred))
print("Precision Score Train:",precision_score(y_train, y_train_pred))
print("Recall Score Train:",recall_score(y_train, y_train_pred))
print("F1 Score Train:",f1_score(y_train, y_train_pred))
print("Accuracy Score Test:",accuracy_score(y_test, y_test_pred))
print("Precision Score Test:",precision_score(y_test, y_test_pred))
print("Recall Score Test:",recall_score(y_test, y_test_pred))
print("F1 Score Test:",f1_score(y_test, y_test_pred))
```

```
LinearSVM Metrics:
Accuracy Score Train: 0.9070875
Precision Score Train: 0.9089562171188931
Recall Score Train: 0.9048208281775345
F1 Score Train: 0.9068838083307235
Accuracy Score Test: 0.893175
Precision Score Test: 0.8927643413951629
Recall Score Test: 0.8936127644675637
F1 Score Test: 0.893188351456068
```

Logistic Regression

In [17]:
```python
from sklearn.linear_model import LogisticRegression

# Training Logistic Regression
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)

# Predictions being made here
y_train_pred = logistic_regression.predict(X_train)
y_test_pred = logistic_regression.predict(X_test)

# Printing Metrics
print("Logistic Regression Metrics:")
print("Accuracy Score Train:",accuracy_score(y_train, y_train_pred))
print("Precision Score Train:",precision_score(y_train, y_train_pred))
print("Recall Score Train:",recall_score(y_train, y_train_pred))
print("F1 Score Train:",f1_score(y_train, y_train_pred))
print("Accuracy Score Test:",accuracy_score(y_test, y_test_pred))
print("Precision Score Test:",precision_score(y_test, y_test_pred))
print("Recall Score Test:",recall_score(y_test, y_test_pred))
print("F1 Score Test:",f1_score(y_test, y_test_pred))
```

```
Logistic Regression Metrics:
Accuracy Score Train: 0.902725
Precision Score Train: 0.9053579740593038
Recall Score Train: 0.8994962940742685
F1 Score Train: 0.9024176154887897
Accuracy Score Test: 0.893825
Precision Score Test: 0.8944784046497645
Recall Score Test: 0.8929125193817836
F1 Score Test: 0.8936947761007233
```

Naive Bayes

In [18]:
```
from sklearn.naive_bayes import MultinomialNB

# Training Naive Bayes
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# Predictions being made here
y_train_pred = naive_bayes.predict(X_train)
y_test_pred = naive_bayes.predict(X_test)

# Printing Metrics
print("Naive Bayes Metrics:")
print("Accuracy Score Train:",accuracy_score(y_train, y_train_pred))
print("Precision Score Train:",precision_score(y_train, y_train_pred))
print("Recall Score Train:",recall_score(y_train, y_train_pred))
print("F1 Score Train:",f1_score(y_train, y_train_pred))
print("Accuracy Score Test:",accuracy_score(y_test, y_test_pred))
print("Precision Score Test:",precision_score(y_test, y_test_pred))
print("Recall Score Test:",recall_score(y_test, y_test_pred))
print("F1 Score Test:",f1_score(y_test, y_test_pred))
```

```
Naive Bayes Metrics:
Accuracy Score Train: 0.8668125
Precision Score Train: 0.8666866574209429
Recall Score Train: 0.8670116364818078
F1 Score Train: 0.8668491164929645
Accuracy Score Test: 0.861775
Precision Score Test: 0.8601952385695787
Recall Score Test: 0.8638523483219127
F1 Score Test: 0.8620199146514936
Recall Score Test: 0.8638523483219127
F1 Score Test: 0.8620199146514936
```

Perceptron model:

Training Accuracy: ~83%

Testing Accuracy: ~82%

This modal has a high recall (97% on both sets) indicates the model is good at identifying positive samples.

It's Precision (75%) is lower which suggests that there are some false positives.

It has balanced F1 scores (~85%) highlighting reasonable overall performance.

Linear SVM:

Training Accuracy: 90.5%

Testing Accuracy: 89.0%

This model has excellent balance between precision (90%) and recall (89%) on both sets.

It also has higher F1 scores (~89%) which suggests a robust and well-generalized model.

Logistic Regression:

Training Accuracy: 90.2%

Testing Accuracy: 89.3%

This model's performance is nearly identical to Linear SVM, with strong precision (~89%) and balanced recall.

It has a slightly lower F1 score compared to Linear SVM but still competitive when compared to all 4.

Naive Bayes:

Training Accuracy: 86.6%

Testing Accuracy: 86.0%

This model has a decent performance with balanced precision and recall (~86%).

It has a lower accuracy compared to SVM and Logistic Regression but still reasonable for simpler models.

Overall Efficiency

Best Model in this case is Linear SVM for its high accuracy, precision, recall, and well-balanced performance.

Naive Bayes is simpler and less computationally expensive but slightly less accurate.

Perceptron model is adequate but less reliable due to false positives.

Logistic Regression is excellent as a close competitor to Linear SVM.