In [1]:
```
import pandas as pd
import numpy as np

import nltk
nltk.download('wordnet')
nltk.download('stopwords')

import re
import contractions
from bs4 import BeautifulSoup

import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
```

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

In [2]:
```
! pip install bs4 # in case you don't have it installed
! pip install contractions
# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Jewelry_v1_00.tsv.gz
```

```
Requirement already satisfied: bs4 in /Users/davisyusuf/opt/anaconda3/lib/python3.9/site-packages (0.
0.1)
Requirement already satisfied: beautifulsoup4 in /Users/davisyusuf/opt/anaconda3/lib/python3.9/site-pa
ckages (from bs4) (4.10.0)
Requirement already satisfied: soupsieve>1.2 in /Users/davisyusuf/opt/anaconda3/lib/python3.9/site-pac
kages (from beautifulsoup4->bs4) (2.2.1)
Requirement already satisfied: contractions in /Users/davisyusuf/opt/anaconda3/lib/python3.9/site-pack
ages (0.1.72)
Requirement already satisfied: textsearch>=0.0.21 in /Users/davisyusuf/opt/anaconda3/lib/python3.9/sit
e-packages (from contractions) (0.0.21)
Requirement already satisfied: anyascii in /Users/davisyusuf/opt/anaconda3/lib/python3.9/site-packages
(from textsearch>=0.0.21->contractions) (0.3.1)
Requirement already satisfied: pyahocorasick in /Users/davisyusuf/opt/anaconda3/lib/python3.9/site-pac
kages (from textsearch>=0.0.21->contractions) (1.4.4)
```

Read Data

In [3]:
```
#Reading the review and rating data using pandas read_table function
col = ['review_body', 'star_rating']
data = pd.read_table('amazon_reviews_us_Jewelry_v1_00.tsv', usecols = col)
```

```
/Users/davisyusuf/opt/anaconda3/lib/python3.9/site-packages/IPython/core/interactiveshell.py:3444: Dty
peWarning: Columns (7) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Keep Reviews and Ratings

In [4]:
```
 # This step is completed in the above function as a parameter.
```

We select 20000 reviews randomly from each rating class.

In [5]:
```
#removing all Nan values in the rating
data['star_rating'] = data['star_rating'].fillna(0)

#Dropping all the rating that is not 1-5 rating (5 classes)
data.drop(data[(data['star_rating'] != 1) & (data['star_rating'] != 2) & (data['star_rating'] != 3) & (c
types = data['star_rating'].unique()

#Grouping the data by the ratings 1-5
new_data = data.groupby('star_rating')

#split each class as a dataframe
group_1 = new_data.get_group(1)
group_2 = new_data.get_group(2)
group_3 = new_data.get_group(3)
group_4 = new_data.get_group(4)
group_5 = new_data.get_group(5)

#randomize 20000 reviews from each class
group_1 = group_1.sample(n=20000)
group_2 = group_2.sample(n=20000)
group_3 = group_3.sample(n=20000)
group_4 = group_4.sample(n=20000)
group_5 = group_5.sample(n=20000)

#combine all the data then randomize again
reduced_data = group_1.append(group_2)
reduced_data = reduced_data.append(group_3)
reduced_data = reduced_data.append(group_4)
reduced_data = reduced_data.append(group_5)
reduced_data = reduced_data.sample(n=100000)
```

Data Cleaning

In [6]:
```
# Calculatig the average before data cleaning
average_before = reduced_data['review_body'].str.len()
print('The Average Length of the Reviews Before Cleaning: ')
print(average_before.mean())

# Making all characters lower-case
reduced_data['review_body'] = reduced_data['review_body'].str.lower()

# Removing all extra white spaces
reduced_data['review_body'] = reduced_data['review_body'].str.strip()

# Removing all the HTML code using Regex, this will remove all the tag that open with < and close with >
reduced_data['review_body'] = reduced_data['review_body'].str.replace('<[^<]+?>', '')

# Removing all URL links using Regex, this will remove all links that start with http: and/or www.
reduced_data['review_body'] = reduced_data['review_body'].str.replace('http\S+|www.\S+', '')

# Removing all non-alphabetical (not a-z or A-Z) characters and replacing them with space
reduced_data['review_body'] = reduced_data['review_body'].str.replace('[^a-zA-Z\s]', '')

# Casting all review data as a string to make sure the data has no errors
reduced_data['review_body'] = reduced_data['review_body'].astype('str')

# Using the contractions library, we apply the "contraction.fix" function to every review in our databas
reduced_data["review_body"] = reduced_data['review_body'].apply(lambda x: contractions.fix(x))

#Calculatig the average after data cleaning
average_after = reduced_data['review_body'].str.len()
print('The Average Length of the Reviews After Cleaning: ')
print(average_after.mean())
```

```
The Average Length of the Reviews Before Cleaning:
188.21263275796548

/var/folders/x3/tycjclwd73q0jqyyk8y7g0pw0000gn/T/ipykernel_76615/2525289905.py:13: FutureWarning: The
default value of regex will change from True to False in a future version.
  reduced_data['review_body'] = reduced_data['review_body'].str.replace('<[^<]+?>', '')
/var/folders/x3/tycjclwd73q0jqyyk8y7g0pw0000gn/T/ipykernel_76615/2525289905.py:16: FutureWarning: The
default value of regex will change from True to False in a future version.
  reduced_data['review_body'] = reduced_data['review_body'].str.replace('http\S+|www.\S+', '')
/var/folders/x3/tycjclwd73q0jqyyk8y7g0pw0000gn/T/ipykernel_76615/2525289905.py:19: FutureWarning: The
```

```
default value of regex will change from True to False in a future version.
  reduced_data['review_body'] = reduced_data['review_body'].str.replace('[^a-zA-Z\s]', '')

The Average Length of the Reviews After Cleaning:
181.73777
```

Pre-processing

remove the stop words

In [7]:
```
# Calculating the average characters of the review before pre-processing
average_before = reduced_data['review_body'].str.len()
print('The Average Length of the Reviews Before Pre-Processing: ')
print(average_before.mean())

# Using the NLTK stopwords library, we get the English stopwords
from nltk.corpus import stopwords
stopw = stopwords.words('english')
# For every one of the reviews in the dataset, we split the string into individual words, we then verify
# if not, we can concatenate it back to the review. In this process, we remove all the stopwords.
reduced_data['review_body'] = reduced_data['review_body'].apply(lambda x: ' '.join([i for i in x.split()
```

```
The Average Length of the Reviews Before Pre-Processing:
181.73777
```

perform lemmatization

In [8]:
```
# Using the NLTK library, we get the Lemmatizer
from nltk.stem import WordNetLemmatizer

# For every one of the reviews in the dataset, we split the string into individual words, we then apply
reduced_data['review_body'] = reduced_data['review_body'].apply(lambda x: ' '.join([WordNetLemmatizer().
```

In [9]:
```
# Calculating the average characters of the review after pre-processing
average_after = reduced_data['review_body'].str.len()
print('The Average Length of the Reviews After Pre-Processing: ')
print(average_after.mean())
```

```
The Average Length of the Reviews After Pre-Processing:
107.94387
```

TF-IDF Feature Extraction

In [10]:
```
# Using the sklearn feature extraction library we create a TF-IDF Vectorizer and extract features
feature_vec = TfidfVectorizer()
features = feature_vec.fit_transform(reduced_data['review_body'])

# We create a numpy array to store all the labels for later use
labels = reduced_data['star_rating'].values
```

In [11]:
```
# We create a vector to store the names/words of each feature that we got
names = feature_vec.get_feature_names()

# We use the function train_test_split to split all the features and labels into training and testing co
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, train_size
```

In [12]:
```
# We cast all the labels as integers because some of the review are floats (like 1.0) instead of integer
train_int_labels = train_labels.astype(int)
test_int_labels = test_labels.astype(int)
```

Perceptron

In [13]:
```
# We create a perceptron model instance
perceptron_model = Perceptron()

# We train the model using the training features and labels
perceptron_model.fit(train_features, train_int_labels)
```

Out[13]:
```
Perceptron()
```

In [14]:
```
# We get the accuarcy score using test features and labels
print("The Accuracy Score for Perceptron is: ")
Perc_acc = perceptron_model.score(test_features, test_int_labels)
print(Perc_acc)
```

```
The Accuracy Score for Perceptron is:
0.40935
```

In [15]:
```
# We predict the ouput labels by using the test data
p_test_pred = perceptron_model.predict(test_features)

# We obtain the precision, recall and F1 scores using the sklearn metrics library
precision_mark_p = precision_score(test_int_labels, p_test_pred, average=None)
recall_mark_p = recall_score(test_int_labels, p_test_pred, average=None)
f1_mark_p = f1_score(test_int_labels, p_test_pred, average=None)
```

In [16]:

```
# We create arrays to store all the score values
prec_arr = []
recall_arr = []
f1_arr = []
avg_arr = []

# The average of precision, recall and F1 scores are calculated by summing them individually and divide
avg_arr = [sum(precision_mark_p)/5, sum(recall_mark_p)/5, sum(f1_mark_p)/5]

# Converting the float scores into strings
for x in precision_mark_p:
    prec_arr.append(str(x))

for x in recall_mark_p:
    recall_arr.append(str(x))

for x in f1_mark_p:
    f1_arr.append(str(x))

# Organizing the string outputs into 5 classes and the average
c_one = [prec_arr[0], recall_arr[0], f1_arr[0]]
c_two = [prec_arr[1], recall_arr[1], f1_arr[1]]
c_three = [prec_arr[2], recall_arr[2], f1_arr[2]]
c_four = [prec_arr[3], recall_arr[3], f1_arr[3]]
c_five = [prec_arr[4], recall_arr[4], f1_arr[4]]
c_avg = ' '.join(str(x) for x in avg_arr)

# Printing the scores and averages from the Perceptron Model
print("In the Perceptron Model:" )
c_one = ', '.join(c_one)
print("The Scores Class 1 are: " + c_one)
c_two = ', '.join(c_two)
print("The Scores Class 2 are: " + c_two)
c_three = ', '.join(c_three)
print("The Scores Class 3 are: " + c_three)
c_four = ', '.join(c_four)
print("The Scores Class 4 are: " + c_four)
c_five = ', '.join(c_five)
print("The Scores Class 5 are: " + c_five)

print("The Averages of the Scores: " + c_avg)
print("* Scores are in the order of Precision, Recall, F1")
```

```
In the Perceptron Model:
The Scores Class 1 are: 0.5001191327138432, 0.5161052372756332, 0.5079864472410455
The Scores Class 2 are: 0.3305576583396417, 0.3278278278278278, 0.32918708380449807
The Scores Class 3 are: 0.30474967907573813, 0.29809141135107986, 0.3013837755490669
The Scores Class 4 are: 0.36067588325652844, 0.2953459119496855, 0.3247579529737206
The Scores Class 5 are: 0.5153518123667378, 0.607286432160804, 0.5575547866205306
The Averages of the Scores: 0.40229083315049785 0.40893136411300607 0.4041740092377723
* Scores are in the order of Precision, Recall, F1
```

SVM

In [17]:

```python
# We create a linear SVM model instance
svm_linear_model = svm.LinearSVC()

# We train the model using the training features and labels
svm_linear_model.fit(train_features, train_int_labels)
```

Out[17]:

```
LinearSVC()
```

In [18]:

```python
# We get the accuarcy score using test features and labels
print("The Accuracy Score for SVM is: ")
svm_acc = svm_linear_model.score(test_features, test_int_labels)
print(svm_acc)
```

```
The Accuracy Score for SVM is:
0.4884
```

In [19]:

```python
# We predict the ouput labels by using the test data
svm_test_pred = svm_linear_model.predict(test_features)

# We obtain the precision, recall and F1 scores using the sklearn metrics library
precision_mark_svm = precision_score(test_int_labels, svm_test_pred, average=None)
recall_mark_svm = recall_score(test_int_labels, svm_test_pred, average=None)
f1_mark_svm = f1_score(test_int_labels, svm_test_pred, average=None)
```

In [20]:
```
# We create arrays to store all the score values
prec_arr_svm = []
recall_arr_svm = []
f1_arr_svm = []
avg_arr_svm = []

# The average of precision, recall and F1 scores are calculated by summing them individually and divide
avg_arr_svm = [sum(precision_mark_svm)/5, sum(recall_mark_svm)/5, sum(f1_mark_svm)/5]

# Converting the float scores into strings
for x in precision_mark_svm:
    prec_arr_svm.append(str(x))

for x in recall_mark_svm:
    recall_arr_svm.append(str(x))

for x in f1_mark_svm:
    f1_arr_svm.append(str(x))

# Organizing the string outputs into 5 classes and the average
c_one_svm = [prec_arr_svm[0], recall_arr_svm[0], f1_arr_svm[0]]
c_two_svm = [prec_arr_svm[1], recall_arr_svm[1], f1_arr_svm[1]]
c_three_svm = [prec_arr_svm[2], recall_arr_svm[2], f1_arr_svm[2]]
c_four_svm = [prec_arr_svm[3], recall_arr_svm[3], f1_arr_svm[3]]
c_five_svm = [prec_arr_svm[4], recall_arr_svm[4], f1_arr_svm[4]]
c_avg_svm = ' '.join(str(x) for x in avg_arr_svm)

# Printing the scores and averages from the SVM Model
print("In the SVM Model:")
c_one_svm = ', '.join(c_one_svm)
print("The Scores Class 1 are: " + c_one_svm)
c_two_svm = ', '.join(c_two_svm)
print("The Scores Class 2 are: " + c_two_svm)
c_three_svm = ', '.join(c_three_svm)
print("The Scores Class 3 are: " + c_three_svm)
c_four_svm = ', '.join(c_four_svm)
print("The Scores Class 4 are: " + c_four_svm)
c_five_svm = ', '.join(c_five_svm)
print("The Scores Class 5 are: " + c_five_svm)

print("The Averages of the Scores: " + c_avg_svm)
print("* Scores are in the order of Precision, Recall, F1")
```

```
In the SVM Model:
The Scores Class 1 are: 0.5536237955592794, 0.6498647651831817, 0.5978961655921277
The Scores Class 2 are: 0.38361581920903953, 0.33983983983983984, 0.3604033970276008
The Scores Class 3 are: 0.40151745068285283, 0.332245102963335, 0.36361137831523976
The Scores Class 4 are: 0.4407188841201717, 0.41333333333333333, 0.4265870440088277
The Scores Class 5 are: 0.600686253484881, 0.7037688442211055, 0.6481545759574222
The Averages of the Scores: 0.47603244061124494 0.4878103771081591 0.4793305121802437
* Scores are in the order of Precision, Recall, F1
```

Logistic Regression

In [21]:

```
# We create a Logistic Regression model instance
LR_model = LogisticRegression(max_iter=1000)

# We train the model using the training features and labels
LR_model.fit(train_features, train_int_labels)
```

Out[21]:

```
LogisticRegression(max_iter=1000)
```

In [22]:

```
# We get the accuarcy score using test features and labels
print("The Accuracy Score for Logistic Regression is: ")
LR_acc = LR_model.score(test_features, test_int_labels)
print(LR_acc)
```

```
The Accuracy Score for Logistic Regression is:
0.51395
```

In [23]:

```
# We predict the ouput labels by using the test data
LR_test_pred = LR_model.predict(test_features)

# We obtain the precision, recall and F1 scores using the sklearn metrics library
precision_mark_LR = precision_score(test_int_labels, LR_test_pred, average=None)
recall_mark_LR = recall_score(test_int_labels, LR_test_pred, average=None)
f1_mark_LR = f1_score(test_int_labels, LR_test_pred, average=None)
```

In [24]:
```
# We create arrays to store all the score values
prec_arr_LR = []
recall_arr_LR = []
f1_arr_LR = []
avg_arr_LR = []

# The average of precision, recall and F1 scores are calculated by summing them individually and divide
avg_arr_LR = [sum(precision_mark_LR)/5, sum(recall_mark_LR)/5, sum(f1_mark_LR)/5]

# Converting the float scores into strings
for x in precision_mark_LR:
    prec_arr_LR.append(str(x))

for x in recall_mark_LR:
    recall_arr_LR.append(str(x))

for x in f1_mark_LR:
    f1_arr_LR.append(str(x))

# Organizing the string outputs into 5 classes and the average
c_one_LR = [prec_arr_LR[0], recall_arr_LR[0], f1_arr_LR[0]]
c_two_LR = [prec_arr_LR[1], recall_arr_LR[1], f1_arr_LR[1]]
c_three_LR = [prec_arr_LR[2], recall_arr_LR[2], f1_arr_LR[2]]
c_four_LR = [prec_arr_LR[3], recall_arr_LR[3], f1_arr_LR[3]]
c_five_LR = [prec_arr_LR[4], recall_arr_LR[4], f1_arr_LR[4]]
c_avg_LR = ' '.join(str(x) for x in avg_arr_LR)

# Printing the scores and averages from the Logisitic Regression Model
print("In the Logistic Regression Model:")
c_one_LR = ', '.join(c_one_LR)
print("The Scores Class 1 are: " + c_one_LR)
c_two_LR = ', '.join(c_two_LR)
print("The Scores Class 2 are: " + c_two_LR)
c_three_LR = ', '.join(c_three_LR)
print("The Scores Class 3 are: " + c_three_LR)
c_four_LR = ', '.join(c_four_LR)
print("The Scores Class 4 are: " + c_four_LR)
c_five_LR = ', '.join(c_five_LR)
print("The Scores Class 5 are: " + c_five_LR)

print("The Averages of the Scores: " + c_avg_LR)
print("* Scores are in the order of Precision, Recall, F1")
```

```
In the Logistic Regression Model:
The Scores Class 1 are: 0.5871373959973015, 0.6419965576592083, 0.6133427296217994
The Scores Class 2 are: 0.4120308756986958, 0.38738738738738737, 0.3993292918870115
The Scores Class 3 are: 0.42903663500678424, 0.3970366649924661, 0.4124168514412416
The Scores Class 4 are: 0.46682590233545646, 0.44251572327044025, 0.4543458607774764
The Scores Class 5 are: 0.640110522680175, 0.6984924623115578, 0.6680283551603988
The Averages of the Scores: 0.5070282663436826 0.5134857591242119 0.5094926177775856
* Scores are in the order of Precision, Recall, F1
```

Naive Bayes

In [25]:

```
# We create a Naive Bayes model instance
bayes_model = MultinomialNB()

# We train the model using the training features and labels
bayes_model.fit(train_features, train_int_labels)
```

Out[25]:

```
MultinomialNB()
```

In [26]:

```
# We get the accuarcy score using test features and labels
print("The Accuracy Score for Naive Bayes is: ")
bayes_acc = bayes_model.score(test_features, test_int_labels)
print(bayes_acc)
```

```
The Accuracy Score for Naive Bayes is:
0.50015
```

In [27]:

```
# We predict the ouput labels by using the test data
bayes_test_pred = bayes_model.predict(test_features)

# We obtain the precision, recall and F1 scores using the sklearn metrics library
precision_mark_bayes = precision_score(test_int_labels, bayes_test_pred, average=None)
recall_mark_bayes = recall_score(test_int_labels, bayes_test_pred, average=None)
f1_mark_bayes = f1_score(test_int_labels, bayes_test_pred, average=None)
```

In [28]:

```
# We create arrays to store all the score values
prec_arr_bayes = []
recall_arr_bayes = []
f1_arr_bayes = []
avg_arr_bayes = []

# The average of precision, recall and F1 scores are calculated by summing them individually and divide
avg_arr_bayes = [sum(precision_mark_bayes)/5, sum(recall_mark_bayes)/5, sum(f1_mark_bayes)/5]

# Converting the float scores into strings
for x in precision_mark_bayes:
    prec_arr_bayes.append(str(x))

for x in recall_mark_bayes:
    recall_arr_bayes.append(str(x))

for x in f1_mark_bayes:
    f1_arr_bayes.append(str(x))

# Organizing the string outputs into 5 classes and the average
c_one_bayes = [prec_arr_bayes[0], recall_arr_bayes[0], f1_arr_bayes[0]]
c_two_bayes = [prec_arr_bayes[1], recall_arr_bayes[1], f1_arr_bayes[1]]
c_three_bayes = [prec_arr_bayes[2], recall_arr_bayes[2], f1_arr_bayes[2]]
c_four_bayes = [prec_arr_bayes[3], recall_arr_bayes[3], f1_arr_bayes[3]]
c_five_bayes = [prec_arr_bayes[4], recall_arr_bayes[4], f1_arr_bayes[4]]
c_avg_bayes = ' '.join(str(x) for x in avg_arr_bayes)

# Printing the scores and averages from the Multinomial Navie Bayes Model
print("In the Multinomial Naive Bayes Model:")
c_one_bayes = ', '.join(c_one_bayes)
print("The Scores Class 1 are: " + c_one_bayes)
c_two_bayes = ', '.join(c_two_bayes)
print("The Scores Class 2 are: " + c_two_bayes)
c_three_bayes = ', '.join(c_three_bayes)
print("The Scores Class 3 are: " + c_three_bayes)
c_four_bayes = ', '.join(c_four_bayes)
print("The Scores Class 4 are: " + c_four_bayes)
c_five_bayes = ', '.join(c_five_bayes)
print("The Scores Class 5 are: " + c_five_bayes)

print("The Averages of the Scores: " + c_avg_bayes)
print("* Scores are in the order of Precision, Recall, F1")
```

```
In the Multinomial Naive Bayes Model:
The Scores Class 1 are: 0.6068548387096774, 0.5920826161790017, 0.5993777224642189
The Scores Class 2 are: 0.3952141057934509, 0.39264264264264265, 0.39392417775546074
The Scores Class 3 are: 0.4054848188050931, 0.41587142139628325, 0.41061244730969504
The Scores Class 4 are: 0.4400715563506261, 0.4332075471698113, 0.4366125760649087
The Scores Class 5 are: 0.6514145141451414, 0.6653266331658292, 0.6582970789310131
The Averages of the Scores: 0.49980796676079775 0.4998261721107136 0.4997648005050593
* Scores are in the order of Precision, Recall, F1
```