MPS Analytics

Course: ALY6040 - Data Mining

Module 6 – Final Report

Topic: Women's E-Commerce Clothing Reviews

Submitted On:

May 19, 2023

Submitted to:

Professor: Ahmadi Behzad

Submitted by:

Heejae Roh

Qihuan He

Shyamala Venkatakrishnan

Introduction

In today's digital age, online reviews have become an integral aspect of conducting business. It's commonplace for customers to check reviews as part of the shopping process, and the impact of reviews cannot be overstated. Positive reviews can significantly enhance a company or product's social credibility, leaving a favorable impression on potential customers. Analyzing reviews can enable any business to gauge the level of customer satisfaction and better comprehend customers' preferences. Therefore, reviews hold great importance for any business seeking to understand and cater to its customers' needs.

For our final project, our group has selected a Women's E-commerce Clothing Reviews dataset from Kaggle. The dataset is analyzed to determine the overall customer sentiment towards various clothing products. To gain a comprehensive understanding of the dataset attributes, Descriptive Analysis is performed to obtain summary statistics such as mean, median, min-max range, and skewness. These statistics provide an overall view of the attribute value distribution. Exploratory data analysis techniques are employed to analyze the dataset, and some of the important findings are illustrated using various graphs and charts. The reviews are analyzed for sentiment using several machine learning models and algorithms like Logistic Regression, Decision Trees, and Random Forest classifiers to classify them as positive or negative. Text mining methods like K-Means clustering, Naïve Bayes Classifier, and LDA (Latent Dirichlet Allocation) topic modeling are employed to group text reviews based on similar themes or topics. This report documents the results of these techniques, and the analysis is conducted using Python libraries.

Dataset Cleaning & EDA

The dataset on e-commerce women's clothing reviews was sourced from the Kaggle platform. Initially, it comprised 23,486 observations and 11 variables. However, after removing the 'id' column and rows with missing values, the dataset's dimensions changed to (19,662, 10). Therefore, the dataset contained 19,662 observations and 10 variables.

Description of the variables/features in the dataset.

No.	Feature	Dictionary
1.	Clothing ID	Indicates unique ID of the product
2.	Age	Age of reviewer. Min: 18, Max: 99
3.	Title	Title of the review
4.	Review Text	whole review text
5.	Rating	Indicates reviewer’s rating 1, 2, 3, 4, 5
6.	Recommended IND	Indicates if the reviewer recommends the product or not
7.	Positive Feedback Count	Indicates the number of postive feedback the review got
8.	Division Name	Name of the division product in it. General, General Petite, Intimates
9.	Department Name	Department product in it. Bottoms, Dresses, Intimate, Jackets, Tops, Trend
10.	Class Name	More specific type of product. Blouses, Causal and others which are 20 distinct types

Descriptive Analysis

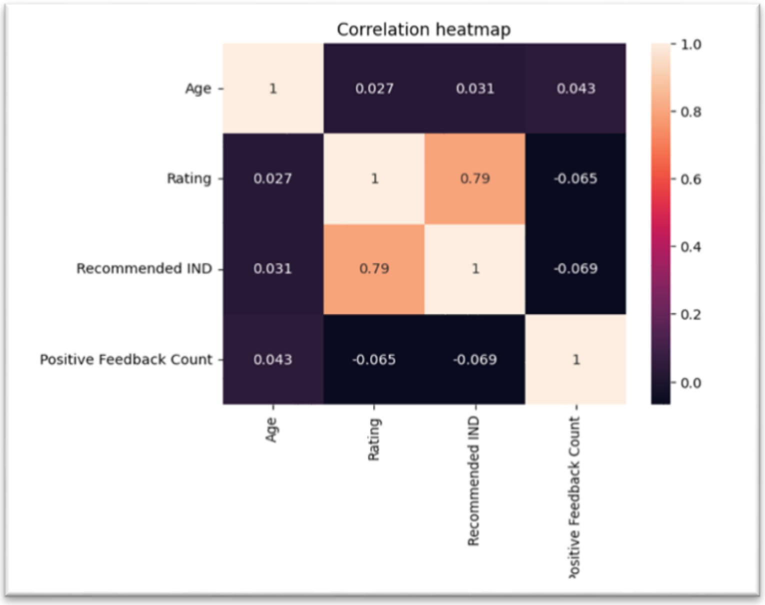
X	mean	sd	median	trimmed	mad	min	max	range	skewed	kurtosis	se
clothing_id	921	200	936	956	158	1	1205	1204	-2.10	5.35	1.4
age*	43.3	12.3	41	42.6	11.9	18	99	81	0.52	-0.14	0.1
title*	6922	3950	7022	6908	5056	1	13983	13982	0.02	-1.14	28.2
review_text*	9829	5674	9830	9829	7284	1	19656	19655	0.00	-1.20	40.4
rating	4.18	1.11	5	4.4	0.00	1	5	4	-1.28	0.71	0.0
recommended_ind	0.82	0.39	1.0	0.9	0.00	0	1	1	-1.65	0.72	0.00
positive_feedback _count	2.65	5.83	1.0	1.38	1.48	0	122	122	6.34	67.49	0.04
division_name*	1.47	0.61	1.0	1.38	0.00	1	3	2	0.94	-0.15	0.00
department_name*	3.35	1.63	3.0	3.43	2.97	1	6	5	-0.16	-1.67	0.01
class_name*	7.92	5.22	8.0	7.53	5.93	1	20	19	0.58	-0.69	0.04

1. Numeric variables in the dataset include age, rating, recommended_ind, and positive_feedback_count.
2. The age variable ranges from 18 to 99, with a median value of 41. The data is positively skewed, indicating that the majority of values are below the median. Consequently, it can be inferred that most individuals fall within their thirties.
3. The rating variable ranges from 1 to 5, with a median of 5. The data is negatively skewed, indicating that most values are above the median. This suggests that the majority of ratings are high.
4. The recommended_ind variable has a mean of 0.82, with a large proportion of values equaling 1. When viewed as a binary variable, the data is negatively skewed.
5. The positive_feedback_count variable ranges from 0 to 122, with a mean of 2.65. The data is positively skewed, indicating that most values are below the mean. This variation is due to the differing number of recommendations for each review.

Correlation Plot

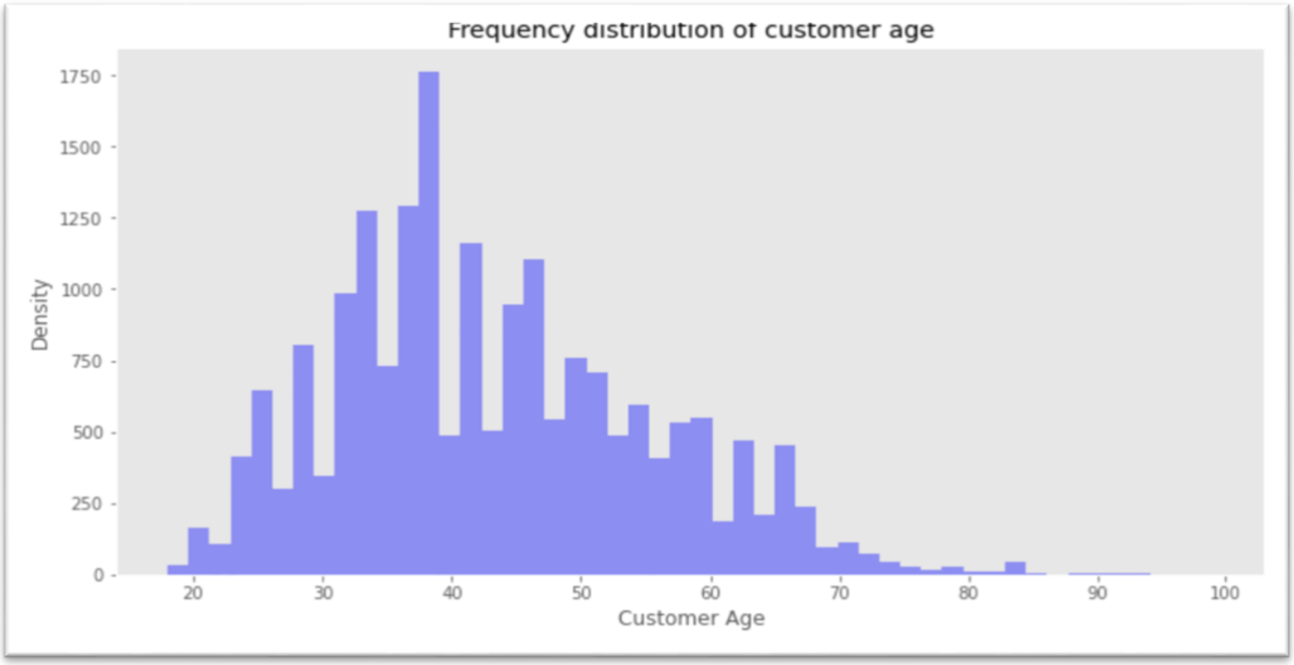
Observations:

Our analysis revealed that the rating variable exhibited the strongest correlation with the recommended indicator, while the other numerical variables had a weak correlation. Due to the high correlation between the recommended indicator and rating, we excluded the rating variable from our analysis. Instead, we concentrated solely on text analysis to identify the key factors that influence customer recommendations.



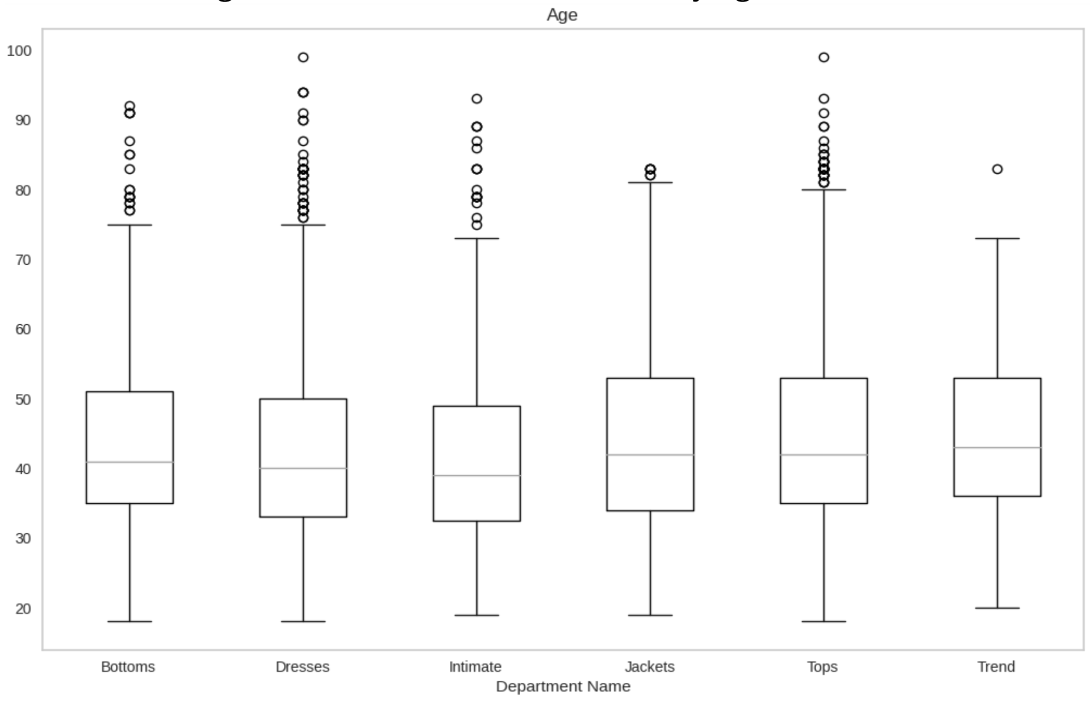
Module 1: Exploratory Data Analysis

1. What is the distribution of the age of the customers?



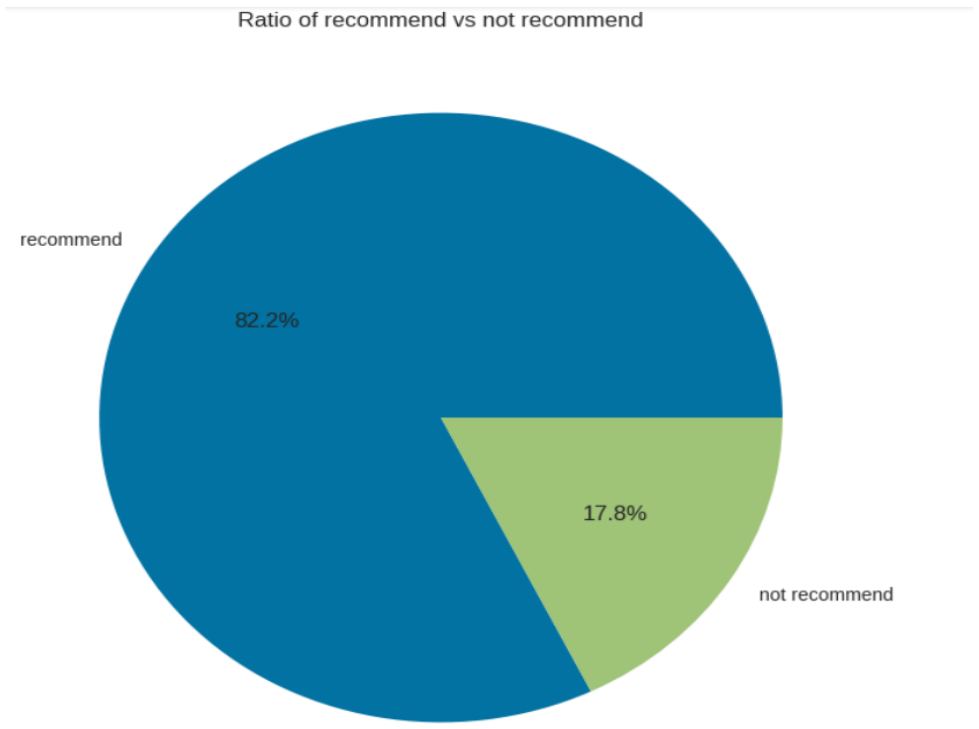
The above histogram shows that most of the customers are in the age group of 30 - 50. The customers purchasing and reviewing the clothing items are mostly middle aged and hence the designers and retailers can look to produce more clothes for the customers in the age group of 30-50.

2. What is the age distribution of the customers buying clothes from different categories?



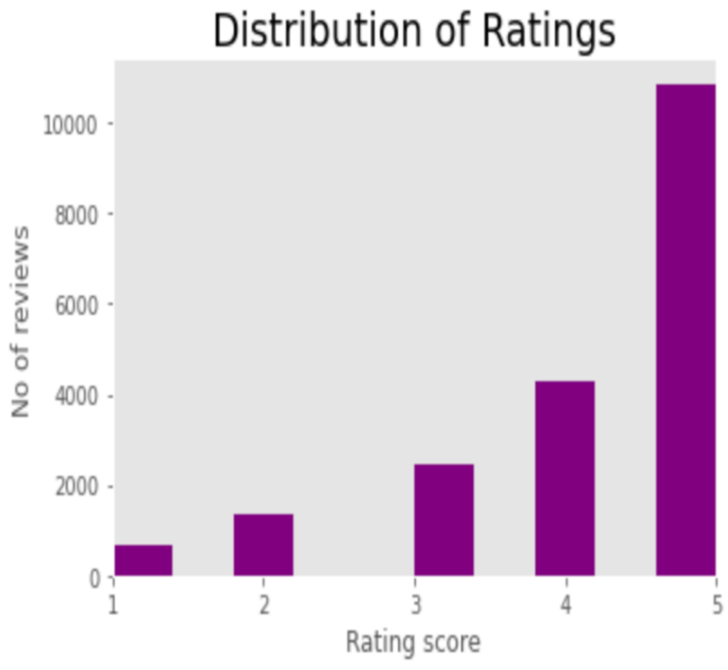
The age distribution of customers who bought intimates appears to be slightly younger compared to those who provide feedback on jackets and tops. It is worth noting that each category's age distribution includes outliers of customers aged over 75 years.

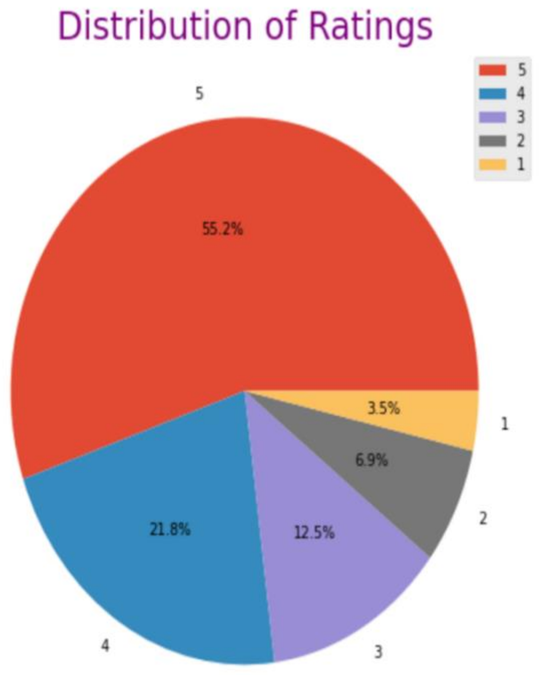
3. How was the distribution of customers who will recommend or not recommend the products?



The pie chart depicts the proportion of customers who are likely to recommend or not recommend the products. It is evident that the majority of customers, accounting for 82.2% of the total feedback, are likely to recommend the products.

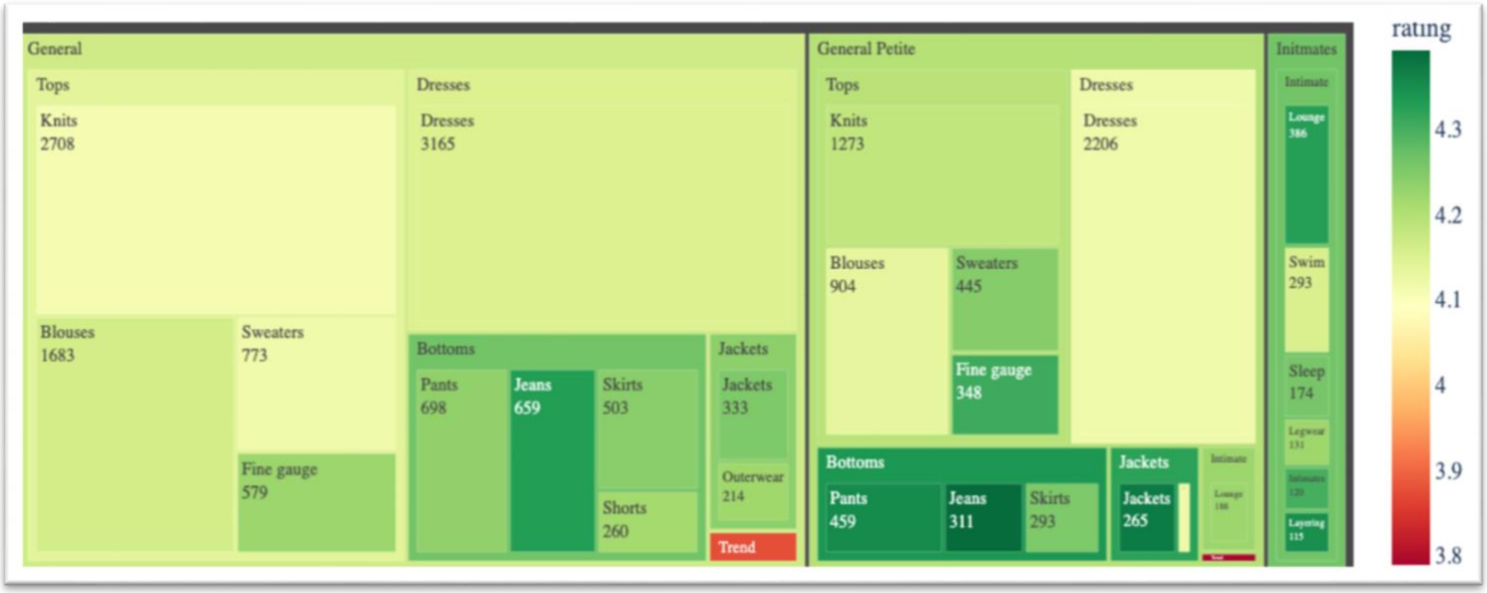
4. What is the distribution of the ratings given by the customers?





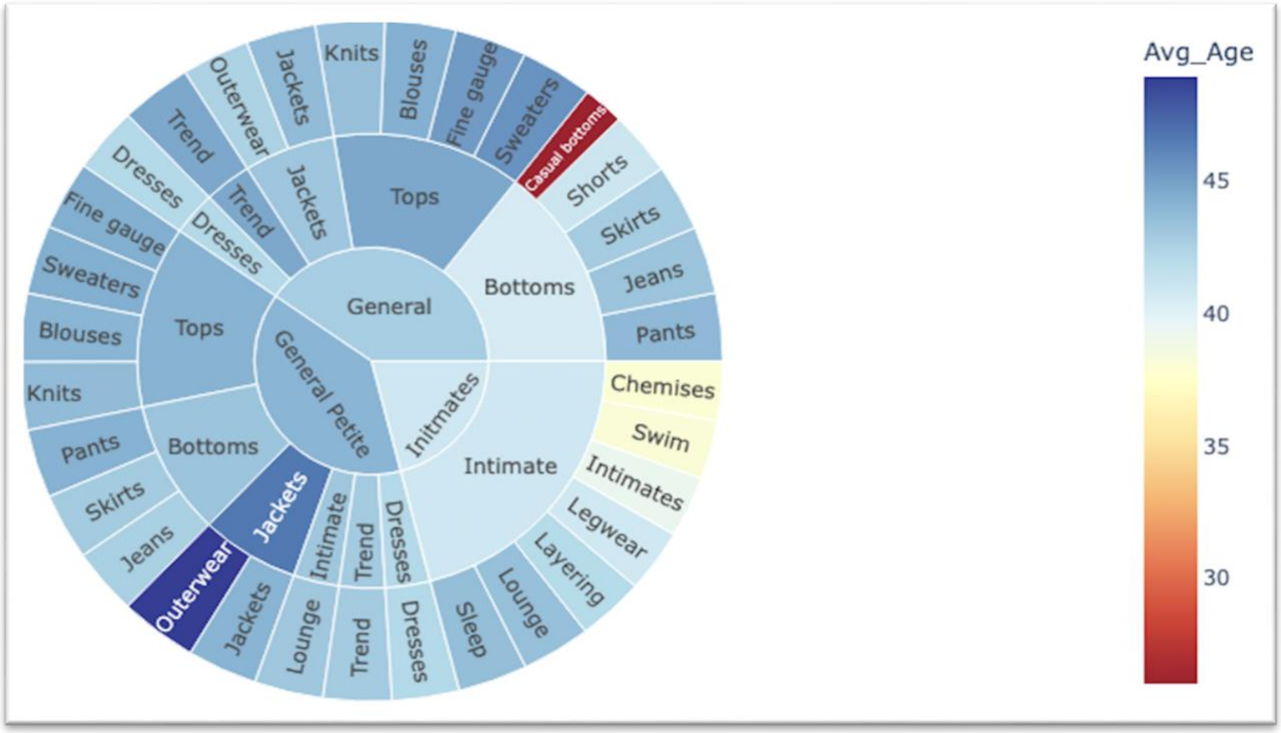
The data indicates that the majority of customers expressed satisfaction with the women's clothing products, with approximately 77% of customers giving a rating of 4 or 5. A small percentage of customers, approximately 3.5% and 6.9%, gave a rating of 1 or 2, respectively. Overall, the feedback for the products was positive.

5. What is the total number of reviews and average rating in each division, department and class of clothes?



The above treemap displays the total number of reviews available under each division, department and class. A color scale has been added to indicate the average ratings received under each class of clothes. Overall, Jeans, Jackets and Lounge class of products have received the highest average rating of 4.3 and above. Knits, Dresses, Sweaters and Blouses have received an average rating of 4 to 4.1. Trendy clothes have received the lowest average rating of 3.8.

6. What is the average age group of customers buying different divisions, departments and classes of clothes?



From the above sunburst chart, it can be observed that the age distribution of customers buying different types of clothing items indicates that customers aged above 45 are predominantly purchasing outerwear and jackets. Customers in the age range of 35 to 40 are mostly buying clothes in the categories of chemises, swimwear, and intimates. The casual bottoms category is more popular among customers below the age of 30.

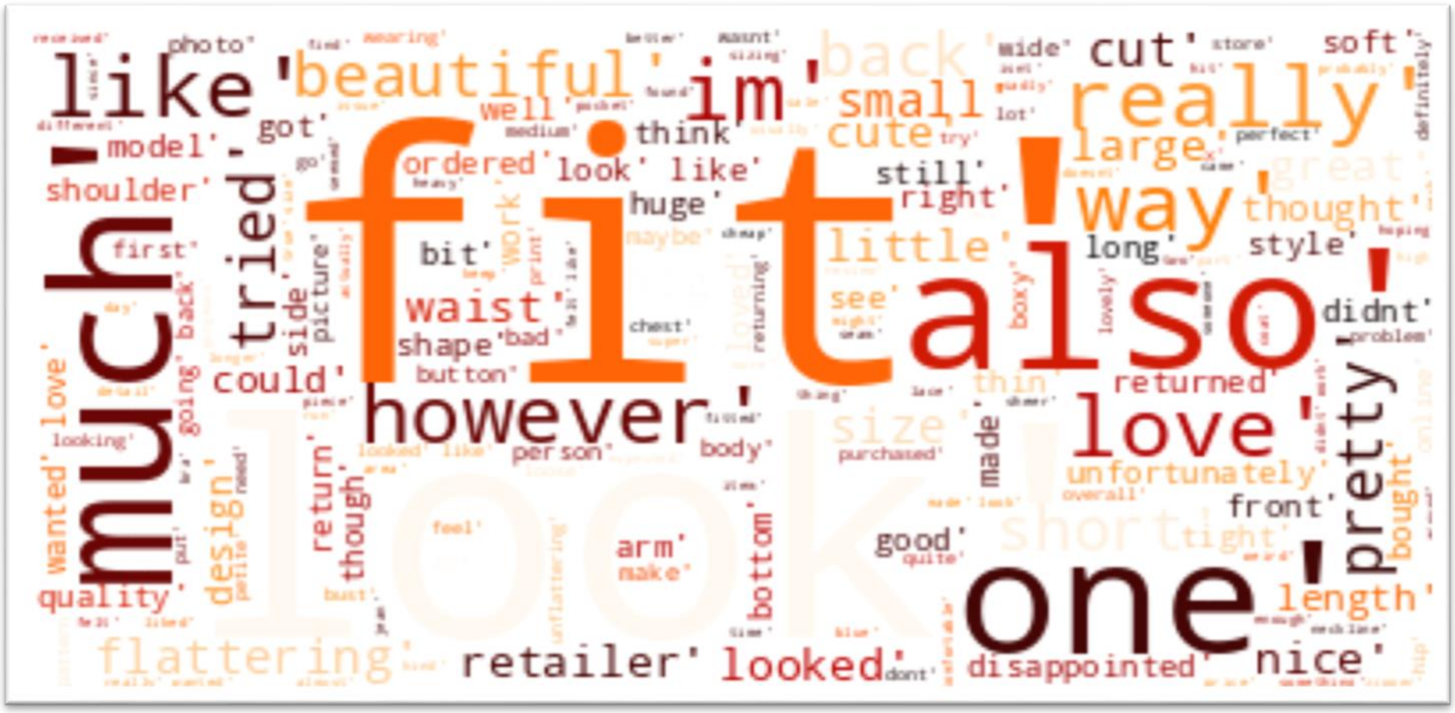
7. What are the keywords to be considered when designing women's clothing items?

The reviews given by each customer are analyzed to generate a word cloud using review_text column.

The following text pre-processing steps were applied in order to generated the word cloud:

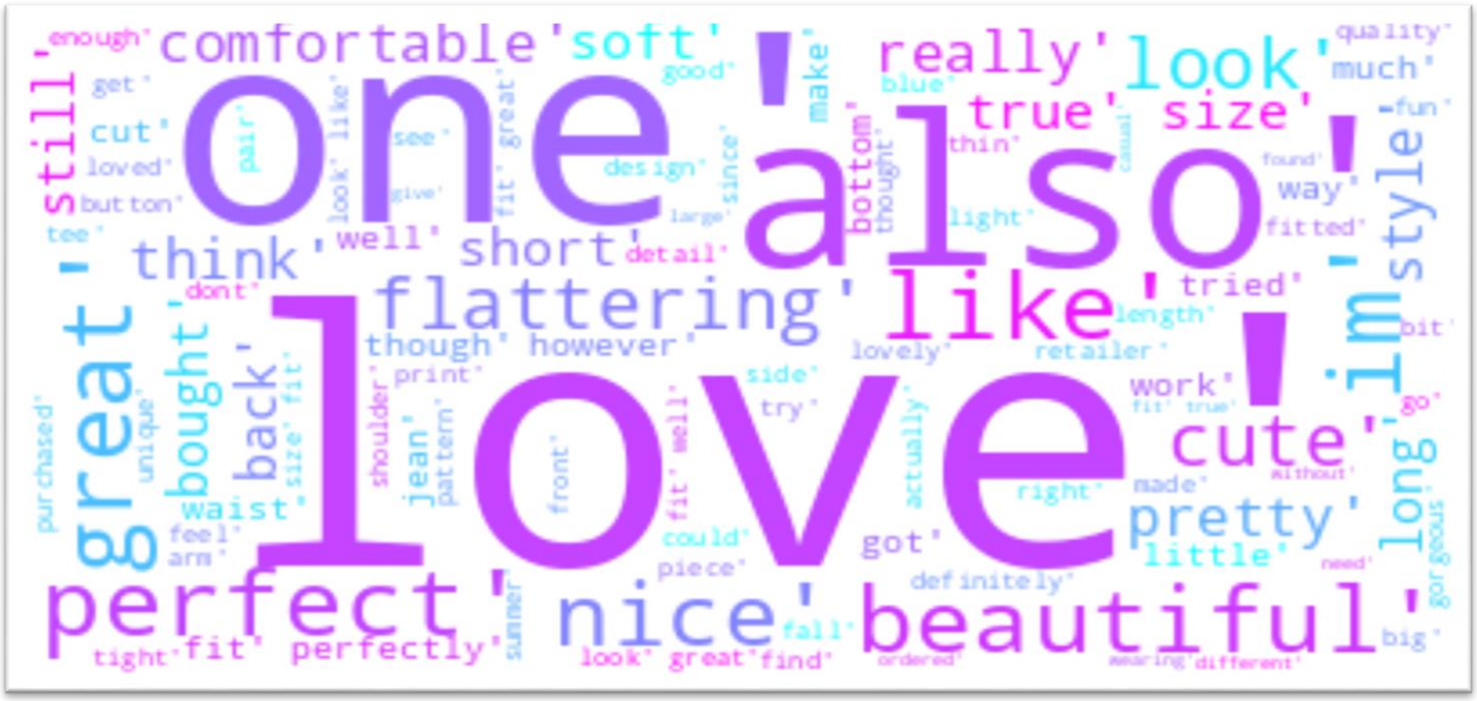
- **Removal of Punctuations:** Removed non alpha characters , other punctuations from the review text.
- **Tokenization:** Splitted the text with spaces to extract the words from each review text.
- **Removal of Stop words**: Removed stop words from the word cloud. Stop words are those words which don't add much meaning to the context. Ex: she, he, it, Iam, a, the, etc.,
- **Lemmatization:** Using WordNetLemmatizer Python library, reduced the words to their root words. Ex play, playing, plays, played are transformed to their root word play and for identifying the context of same meaning words.

Keywords of Not recommended clothes:



This is the word cloud generated from the reviews of the customers who have not recommended the products. The keywords extracted from the negative feedback of the customers include fit, small, large, disappointed, unfortunately, little.

Keywords of recommended clothes:



This is the word cloud generated from the reviews of the customers who have recommended the products. The keywords extracted from the positive feedback of the customers include perfect, love, beautiful, comfortable, soft, style, etc.,

These keywords can be helpful for the retailers to understand the terms to be considered when designing women’s clothing products and also the size of each word in these word clouds is proportional to its frequency.

Module 2: Introduction To Data Mining Techniques:

We constructed various supervised machine learning models using algorithms such as Logistic Regression, Decision Tree, and Random Forest to classify customer sentiment. To comprehend the number of true and false positives and negatives, we plotted a confusion matrix. We compared the models' performances using various metrics such as F1 score and precision-recall curves.

Text pre-processing

We converted all the review text sentences to lowercase, removed all punctuation marks (rem_puc), cleaned the regular expression in URLs or Non ascii (clean_regex), removed all stopwords (stopwords.words). Lemmatization was performed using the NLTK library.

Feature Extraction from the Reviews

We decided to use both Bag of words and Tf-Idf vectorization to extract the features from the review text. A bag-of-words is a representation of text that describes the occurrence of words within a document. Specially used in the Text Classification task. TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

Primary Methodologies

Sentiment Analysis

- **Decision Tree:**

  By utilizing the Decision Tree algorithm, we were able to predict whether or not a user recommended a product with high accuracy. Specifically, we achieved an F-1 score of 89% using Bag-of-Words and 88% using TF-IDF.

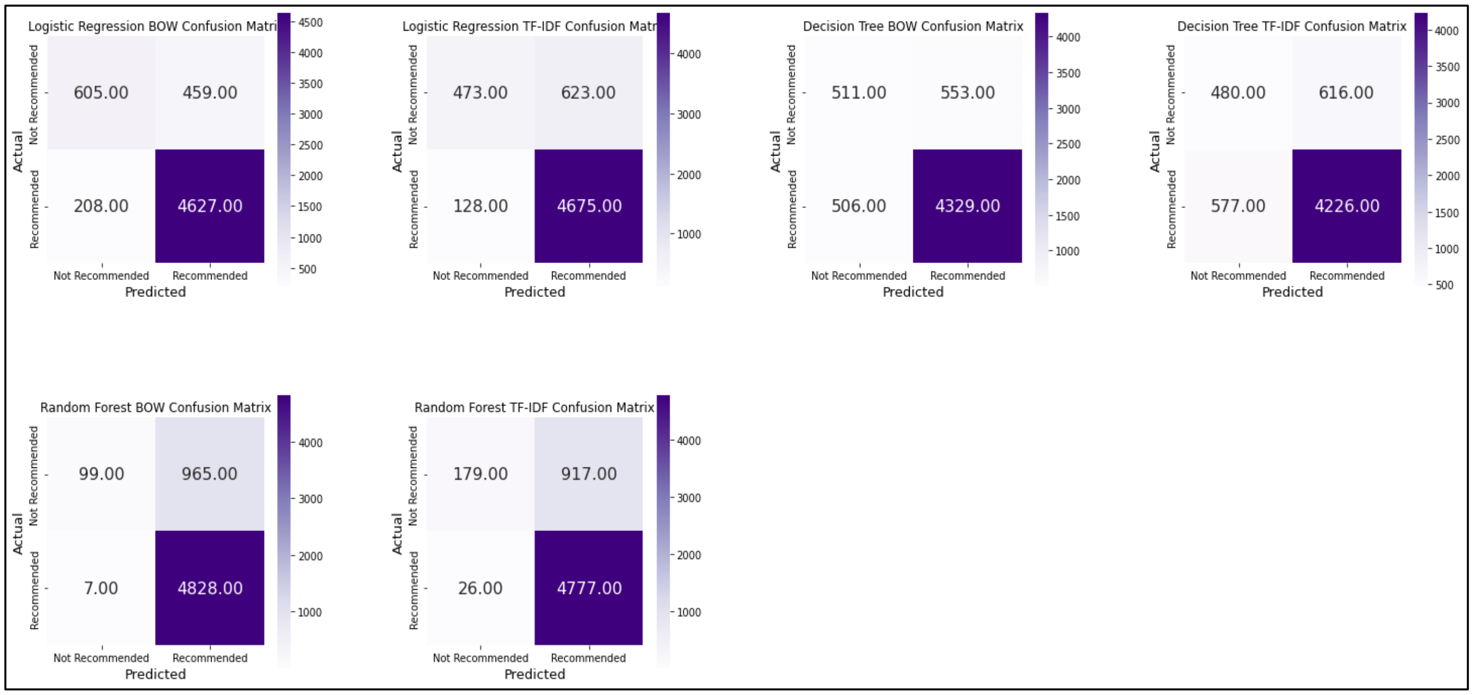
- **Random Forest:**

  Through the application of the Decision Tree algorithm, we successfully predicted whether a user recommended a product with high accuracy. Specifically, we achieved an F-1 score of 91% using both Bag-of-Words and TF-IDF vectorization methods. Notably, this percentage is higher than that obtained using the Decision Tree alone, and the performance did not vary significantly between the two vectorization methods.

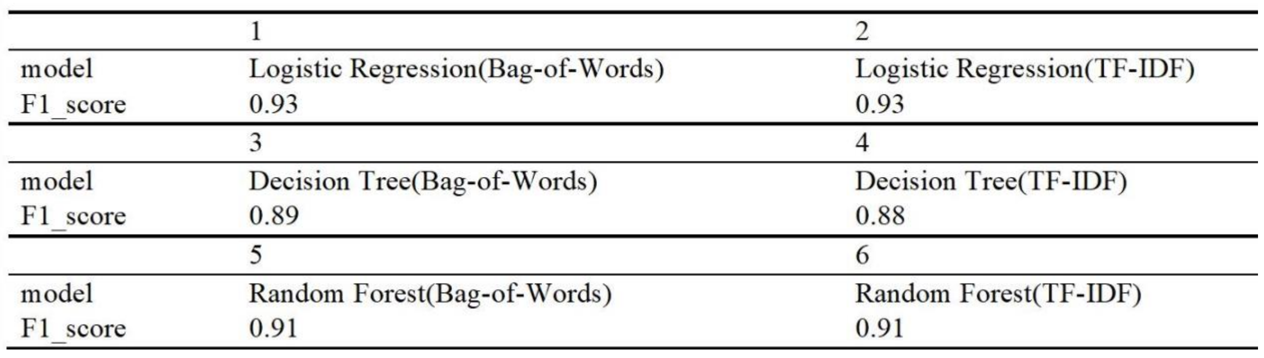
- **Logistic regression:**

  In order to compare the performance of different classification algorithms, we evaluated the Decision Tree, Random Forest, and Logistic Regression models. Interestingly, the Logistic Regression model achieved the highest F-1 score among the three, with a score of 93% using both the Bag-of-Words and TF-IDF vectorization methods. These results indicate that Logistic Regression is a promising approach for predicting product recommendations in this context.

Confusion matrix of the sentiment classifier Models:

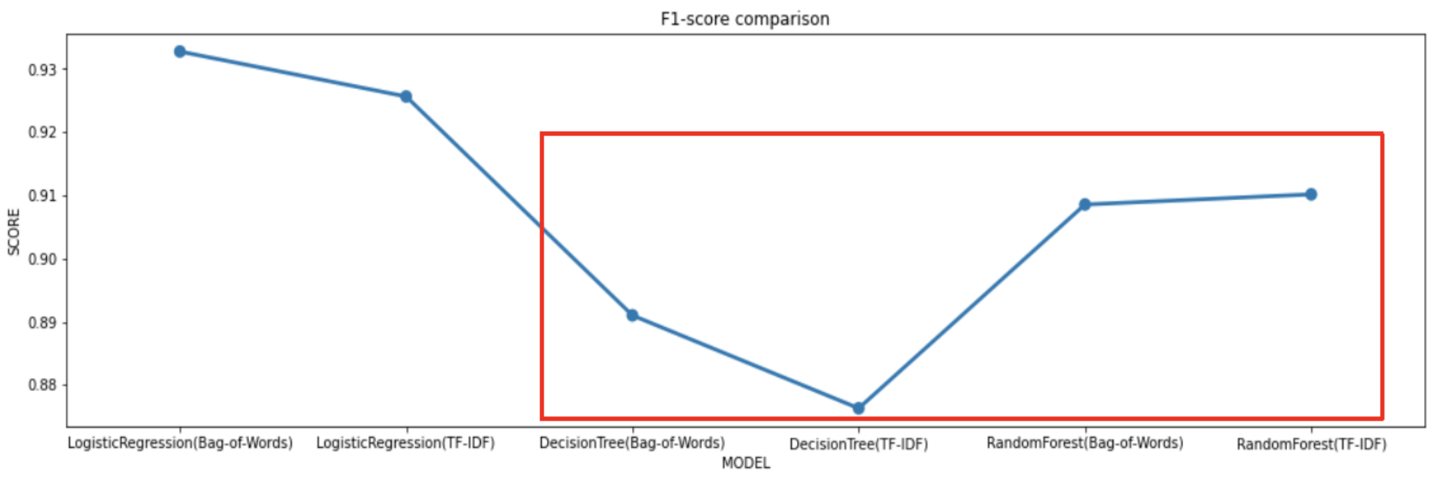


F-1 Score of each model



	1	2
model	Logistic Regression(Bag-of-Words)	Logistic Regression(TF-IDF)
F1_score	0.93	0.93
	3	4
model	Decision Tree(Bag-of-Words)	Decision Tree(TF-IDF)
F1_score	0.89	0.88
	5	6
model	Random Forest(Bag-of-Words)	Random Forest(TF-IDF)
F1_score	0.91	0.91

F-1 score comparison:



We chose to use the F1 score to compare models because the recommended indicator, our target variable, was imbalanced at 82:18, meaning that there were more positive recommendations. Using Accuracy as our evaluation metric in such a case could result in a higher number of false positives. A higher F1 score indicates better model performance.

After comparing the F1 scores, we found that the Logistic Regression classifier model slightly outperformed the Random Forest model, as it had a higher F1 score. The Decision Tree classifier model had the lowest F1 score and, therefore, had poorer model performance than the Logistic Regression and Random Forest models.

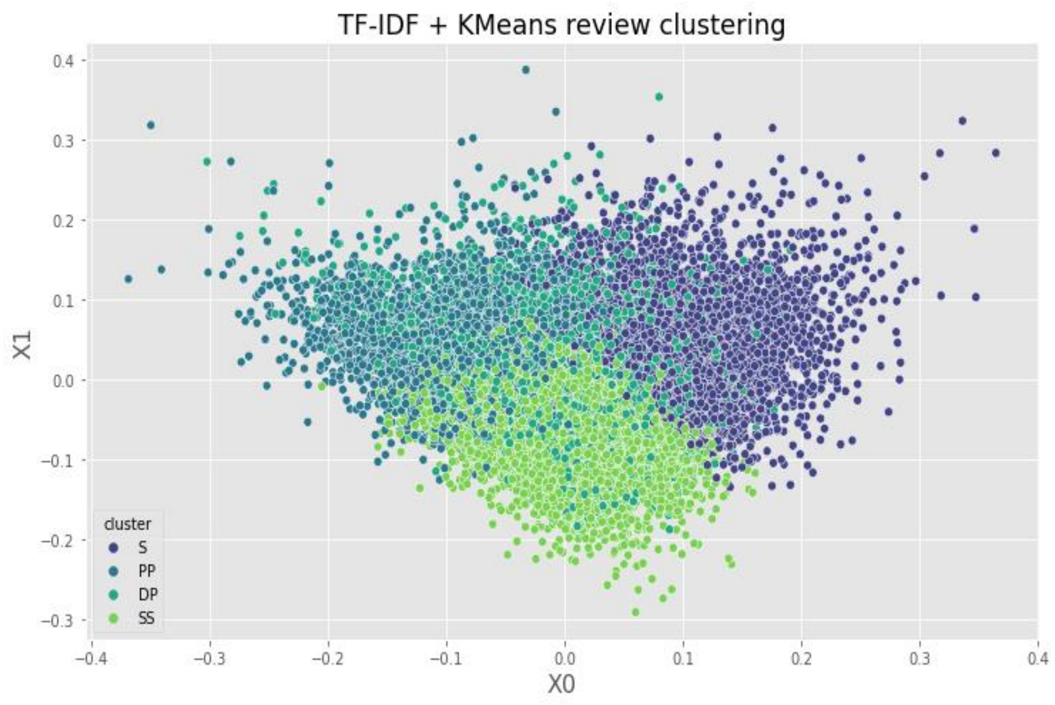
Module 3: Clusters, Association Mining, and Linear Discriminant Analysis

Text Clustering:

Text clustering is the process of categorizing text documents into groups based on their similarities in terms of topics, themes, or other characteristics. In our study, we aimed to determine the appropriate group for a given text by utilizing text clustering. Due to the shared characteristics of the text information, we employed a multi-faceted approach to achieve accurate results.

- **K-Means clustering:**

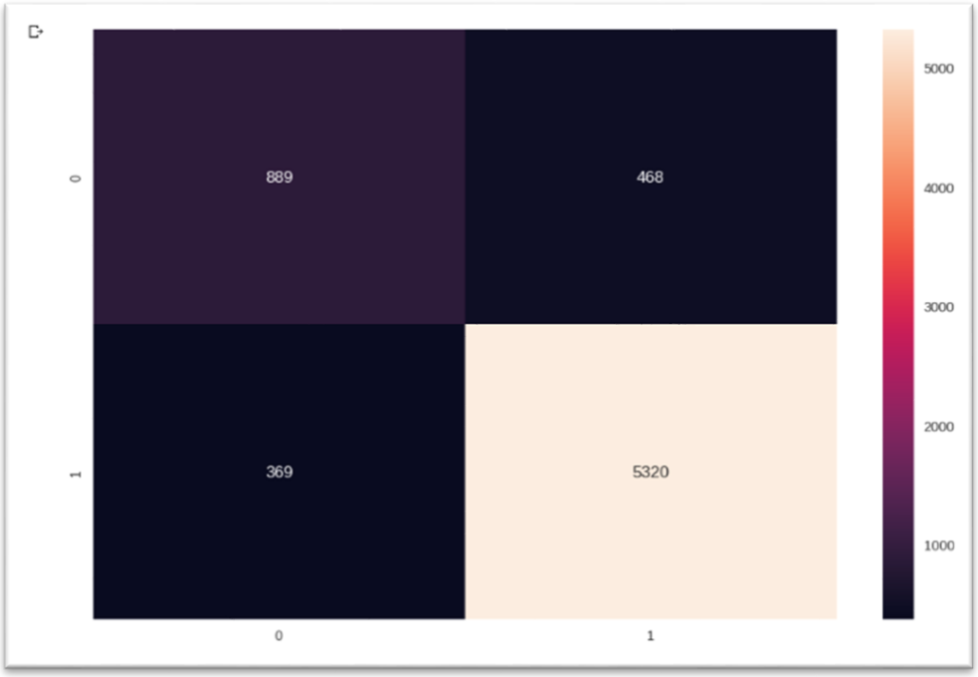
  Using K-means clustering, we identified four distinct clusters that effectively separated the data. We then examined each of the 10 keywords within each cluster and found significant overlap between them. This outcome is not unexpected, given that our text clustering analysis was conducted on a homogeneous sample of data consisting of women's clothing reviews from a single website. As such, the resulting text clusters naturally share many commonalities.



Trimmed Top Keywords		
Cluster 0 (S)	medium(1), top(2), order(1), run(1), large(1), small(1), size(2)	Size
Cluster 1 (DP)	like(2), great(2), fabric(2), size(2), dress(1)	Dress & positive
Cluster 2 (SS)	shirt(1), sweater(1), fabric(2), like(2), top(2)	Shirt, Sweater
Cluster 3 (PP)	perfect(1), comfortable(1), pant(1), jean(1), great(2)	Pants & positive

- **Gaussian Naive Bayes Classifier:**

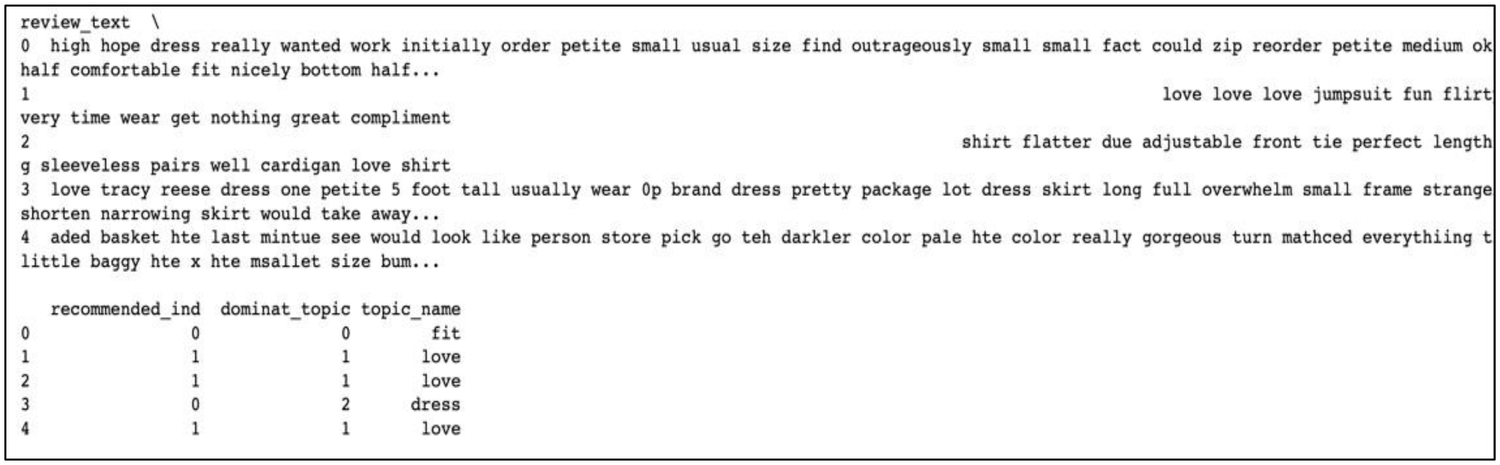
We utilized clustering based on conditional probabilities assuming a Gaussian distribution, using techniques such as TF-IDF. This approach allowed us to perform a thorough analysis of the data and apply conditional probability theory to predict new categories for the data points. Our Gaussian Naive Bayes classifier demonstrated an impressive accuracy rate of 0.88, indicating the effectiveness of this approach in uncovering meaningful patterns in the data.



- **Topic Modelling using LDA (Latent Dirichlet Allocation):**

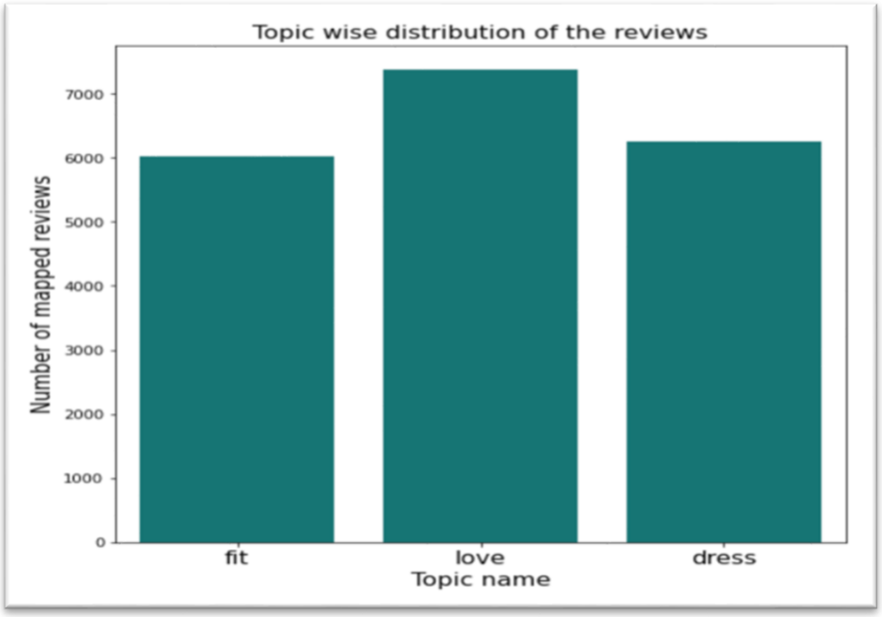
  Leveraging the advantages of LDA (Latent Dirichlet Allocation) for text clustering, we aimed to improve the accuracy and multifaceted nature of the results. LDA is a widely used technique in the industry for topic modeling, which identifies hidden patterns and generates topics for each document. It is a common approach for analyzing online reviews. We further examined the distribution of each topic to determine the number of topics.

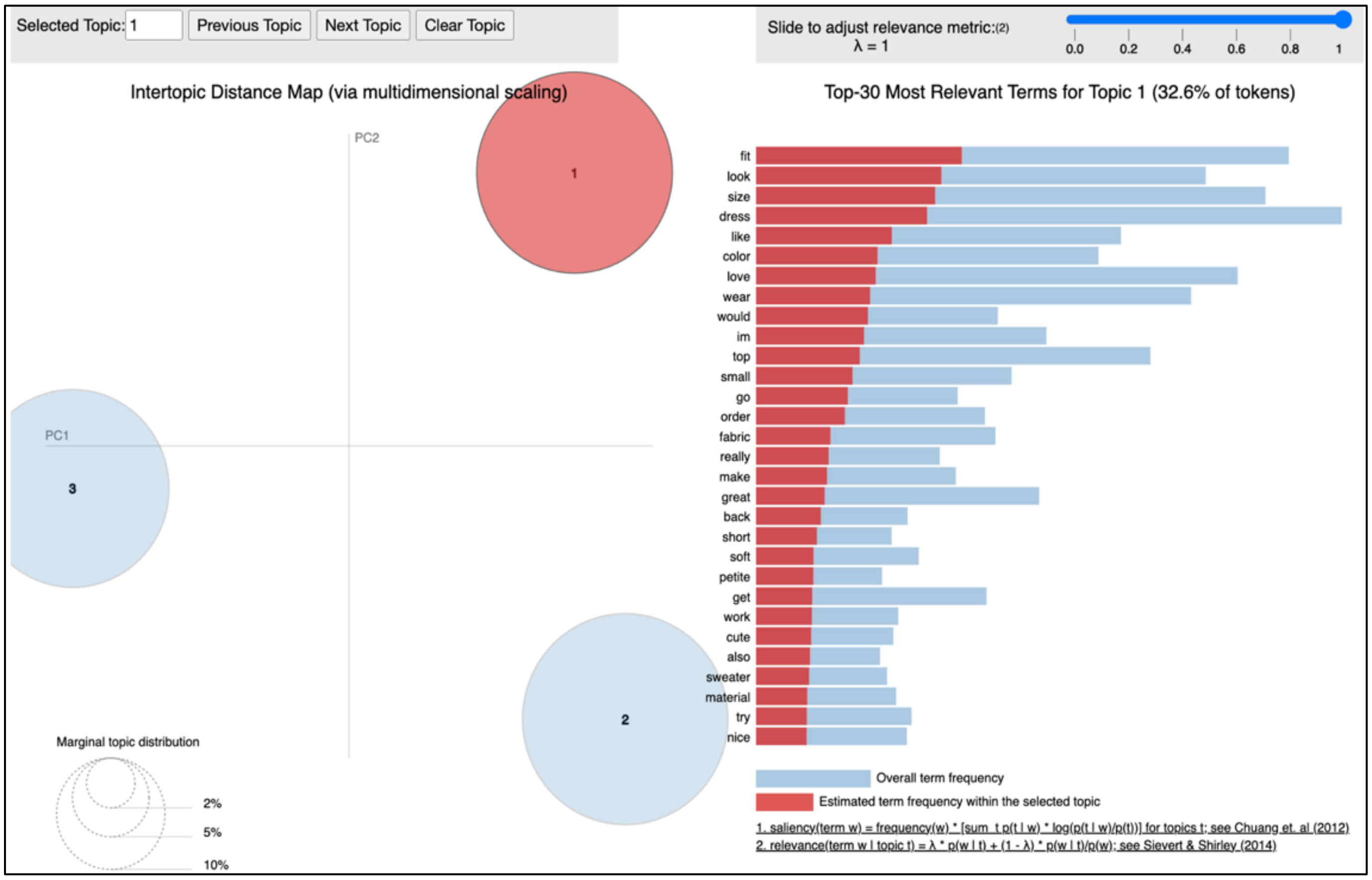
The dominant topic name for each of the reviews is displayed as below:



```
review_text  \
0  high hope dress really wanted work initially order petite small usual size find outrageously small small fact could zip reorder petite medium ok
half comfortable fit nicely bottom half...
1                                                                                                                                  love love love jumpsuit fun flirt
very time wear get nothing great compliment
2                                                                                                              shirt flatter due adjustable front tie perfect length
g sleeveless pairs well cardigan love shirt
3  love tracy reese dress one petite 5 foot tall usually wear 0p brand dress pretty package lot dress skirt long full overwhelm small frame strange
shorten narrowing skirt would take away...
4  aded basket hte last mintue see would look like person store pick go teh darkler color pale hte color really gorgeous turn mathced everythiing t
little baggy hte x hte msallet size bum...

   recommended_ind  dominat_topic topic_name
0                0              0        fit
1                1              1       love
2                1              1       love
3                0              2      dress
4                1              1       love
```





Evaluating the LDA Model performance:

● The perplexity of the above LDA model is -6.93, which indicates how well the model can predict the topic for reviews in the testing phase, given the reviews in the training phase. The lower the perplexity value, the better the model performance.

● Coherence metric computes the similarity between words within a topic and compares the similarity between words in different topics. Higher coherence scores indicate more coherent and interpretable topics, while lower scores indicate less coherent topics.

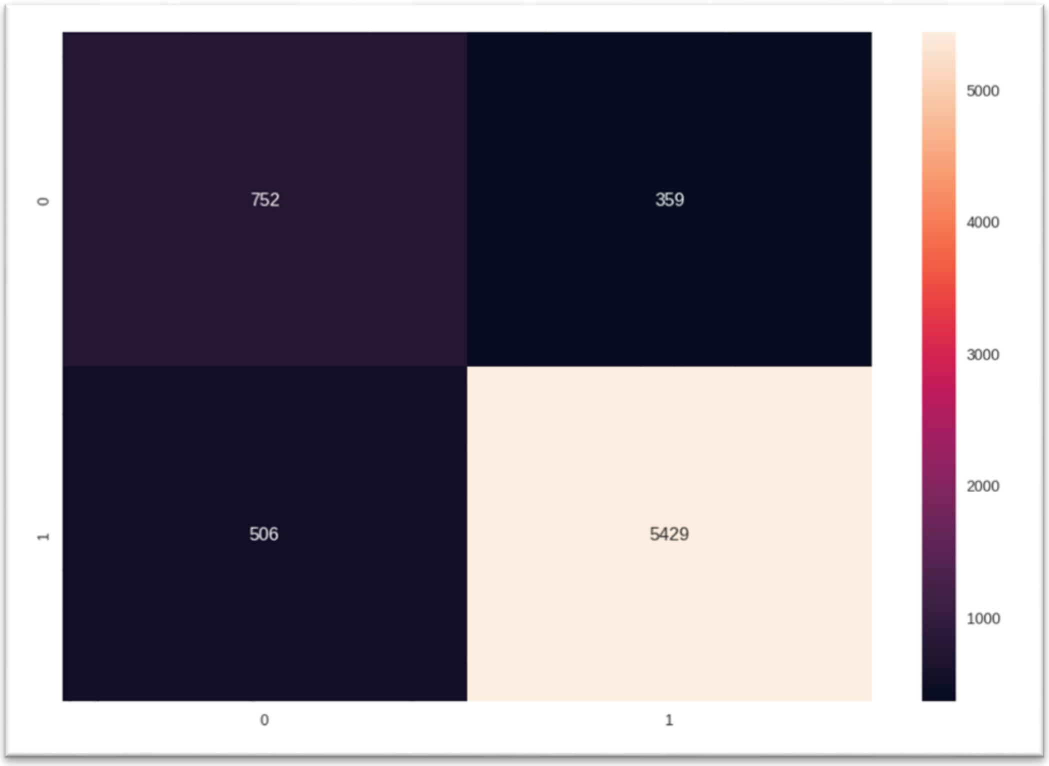
● The coherence value of the LDA model is 0.33 and is computed using the CoherenceModel Gensim module.

For our analysis, we employed K-means clustering to form four distinct clusters based on primary keywords, namely Size, Dress & positive, Shirt & Sweater, and Pants & positive. To classify the clusters, we utilized the Naïve Bayes Classifier and obtained an accuracy rate of 88%. Furthermore, we utilized the LDA model to form three clusters and analyzed the top keywords within each cluster.

Module 4 — Support Vector Machines

- **SVM (Support Vector Machines):**

  The process described involves four steps. The first step is to preprocess the feedback data by word vectorization, which transforms the raw feedback data into numerical feature vectors based on weighted frequency of words. In the second step, the preprocessed data is split into a 70% training set and a 30% testing set. In the third step, an SVM model is generated by feeding the training data set into the model. The SVM model finds the optimized hyperplane to divide the data into two subspaces based on different categories. Finally, in the fourth step, the trained model is used to predict the test data. The SVM model classifies new text data by assigning it to the class that lies on the corresponding side of the hyperplane. If a feedback data has features that lie on one side of the hyperplane which is classified as recommended, it will be predicted as recommended, and if the features lie on the other side, it can be classified as not recommended.



Evaluating the SVM Model performance:

● The accuracy score of the SVM model is 0.87. The confusion matrix above indicates that 752 data is correctly classified as un-recommend and 5429 feedback is correctly classified as recommended.

● 506 feedbacks from customers who will not recommend the products are incorrectly labeled as will recommend, while 359 feedback from customers who will recommend the products are inaccurately labeled as not recommended.

SVM can be a powerful tool for text classification and spam detection, especially when dealing with high-dimensional feature spaces and complex relationships between the features and the classes.

Conclusion

In this project, we analyzed a dataset containing reviews of women's clothing items from an e-commerce company to determine customer sentiment towards various clothing products. We conducted data cleaning and exploratory data analysis, presenting interesting insights in the form of graphs and charts. Below are some of the insights extracted from the dataset:

- The majority of the customers are in the age group of 30 - 50.
- The age distribution of customers who bought intimates appears to be slightly younger compared to those who provide feedback on jackets and tops.
- The majority of customers, accounting for 82.2% of the total feedback, are likely to recommend the products.
- The majority of customers expressed satisfaction with the women's clothing products, with approximately 77% of customers giving a rating of 4 or 5.
- The customers aged above 45 are predominantly purchasing outerwear and jackets. Customers in the age range of 35 to 40 are mostly buying clothes in the categories of chemises, swimwear, and intimates.
- From the word cloud, the keywords extracted from the positive feedback of the customers include perfect, love, beautiful, comfortable, soft, style, etc., The keywords extracted from the negative feedback of the customers include fit, small, large, disappointed, unfortunately, little.

We built several supervised ML models, including Logistic Regression, Decision Tree, and Random Forest, to classify review sentiment as positive or negative. Numerical features were extracted from the reviews using techniques such as Bag of Words and TF-IDF, and model performance was evaluated using the F1 score metric and precision-recall curves. Logistic Regression model outperformed Decision Tree and Random Forest models with a higher F1 score, while Random Forest performed better than Decision Tree model.

We also employed text clustering, topic modeling, and Naïve Bayes Classifier to group reviews based on common themes or topics. KMeans clustering was used to form four text groups, but we identified the limitation of overlapping parts. We applied Gaussian Naïve Bayes Classifier to classify text data and identify features, checking the results with a confusion matrix. LDA was used for topic modeling to uncover hidden patterns and assign dominant topics to each review, condensing reviews into several different topics. The number of clusters was determined based on inter-cluster distance, reducing clusters when they overlapped or had smaller distances. The reviews were grouped under 3 major topics like fit, love and dress.

Reference

NICAPOTATO. (2018). Women's e-commerce clothing reviews. kaggle. Retrieved from https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews?datasetId=11827&sortBy=voteCount&searchQuery=eda

MATTHEW CONNOR. (2022). NLP: Comparative RNN & DL Models with detailed EDA. kaggle. Retrieved from https://www.kaggle.com/code/azizozmen/nlp-comparative-rnn-dl-models-with-detailed-eda

Shankar297. (May 31, 2022). A Complete guide on feature extraction techniques. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2022/05/a-complete-guide-on-feature-extraction-techniques/

Purva Huilgol. (February 28, 2020). Quick introduction to bag-of-words (BoW) and TF-IDF for creating features from text. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/#:~:text=Bag%20of%20Words%20just%20creates,vectors%20are%20easy%20to%20interpret.

Doug, Steen. (September 19, 2020). Precision-recall curves. Medium. Retrieved from https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248

Andrea, D'Agostino. (November 24, 2021). Text clustering with TF-IDF in python. Medium. Retrieved from https://medium.com/mlearning-ai/text-clustering-with-tf-idf-in-python-c94cd26a31e7

Shivam5992. (August 24, 2016). Beginners guide to topic modeling in python. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/

Aravind CR. (July 26, 2020). Topic modeling using gensim-LDA in python. Medium. Retrieved from https://medium.com/analytics-vidhya/topic-modeling-using-gensim-lda-in-python-48eaa2344920

Gunjit, Bedi. (2018, November 9). A guide to Text Classification(NLP) using SVM and Naive Bayes with Python. Medium. Retrieved from https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34

sepandhaghighi. (2018, June 4). How to get accuracy, confusion matrix of binary SVM classifier equivalent to multiclass classification?. Cross Validated. Retrieved from https://stats.stackexchange.com/questions/303524/how-to-get-accuracy-confusion-matrix-of-binary-svm-classifier-equivalent-to-mul

A Dash of Data. (2020). Natural language processing (Part 5): topic modeling with latent dirichlet allocation in python. YouTube. Retrieved from https://www.youtube.com/watch?v=NYkbqzTlW3w

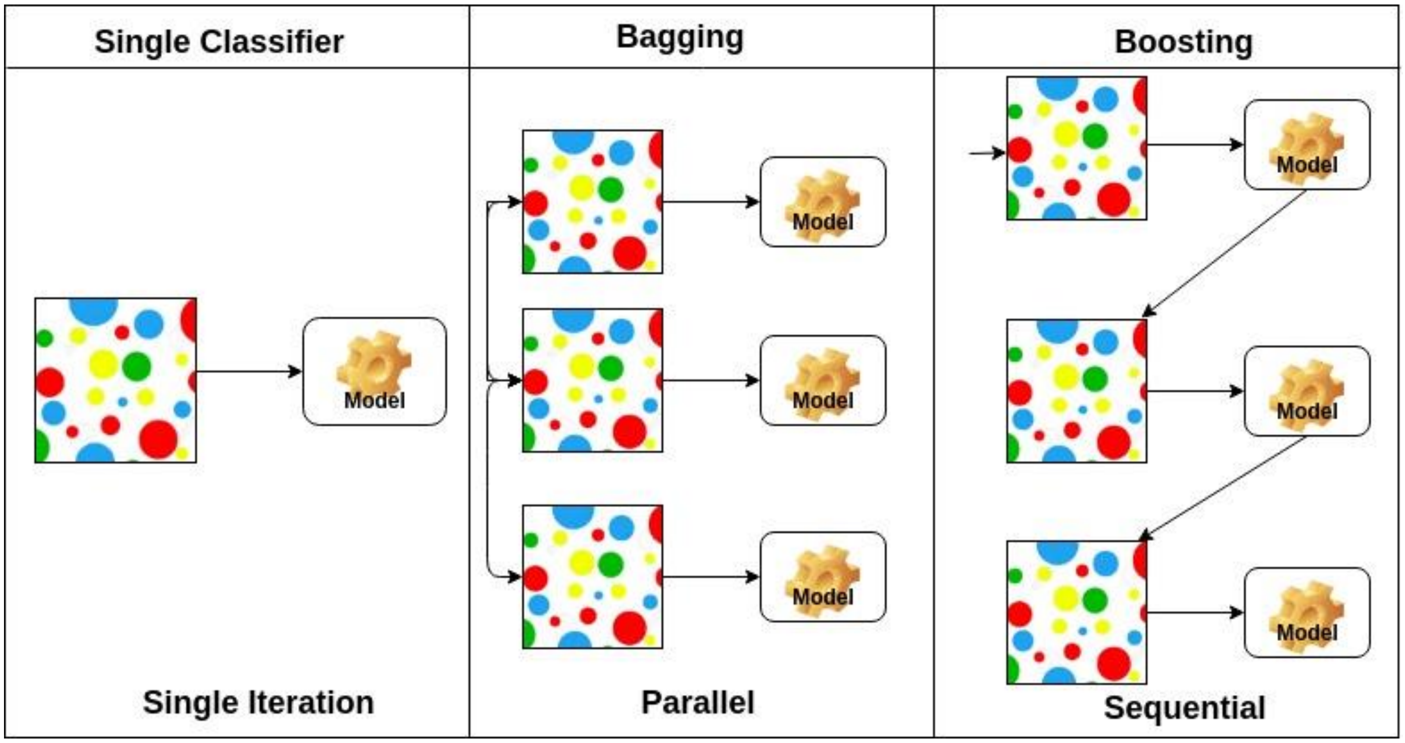
Alice, Zhao. (2018). Topic modeling. github. Retrieved from https://github.com/adashofdata/nlp-in-python-tutorial/blob/master/4-Topic-Modeling.ipynb

Module 5 progress. Appendix

- **Adaptive Boosting (AdaBoost):**

  AdaBoost is a boosting method that uses the complete training dataset to train the weak learners. It is a sequential process, where each subsequent model tries to correct the errors of the previous model. So, the succeeding models are dependent on the previous model. AdaBoost is an ensemble learning method (also known as “meta-learning”) which was initially created to increase the efficiency of binary classifiers. AdaBoost uses an iterative approach to learn from the mistakes of weak classifiers, and turn them into strong ones (Vihar Kurama, 2021).

Ensemble Machine Learning Approach:



Boosting is a sequential modeling technique that builds a family of models to create a strong learner. Each model in the sequence is fitted by giving more importance to observations that were poorly handled by previous models. This adaptive learning approach helps reduce bias in the resulting strong learner.

Boosting can also help reduce variance, but its main focus is on reducing bias. It achieves this by using base models with low variance but high bias, such as shallow decision trees with limited depth. These weak learners are computationally less expensive to fit compared to complex models.

Unlike bagging, where models can be trained in parallel, boosting requires sequential fitting of models, making it computationally expensive to fit multiple complex models in a sequence. Therefore, boosting is a suitable approach when using weak learners with low variance and high bias to create a powerful ensemble model while keeping computational costs manageable.

Steps of AdaBoosting (datacamp, 2018):

● Initially, Adaboost selects a training subset randomly.

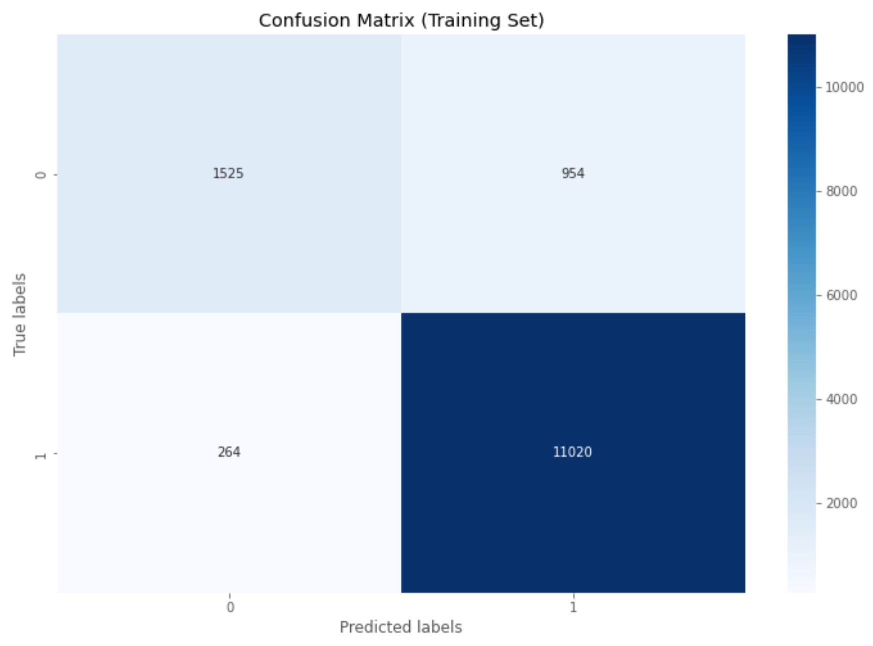
● It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.

● It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.

● Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.

● This process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators.

● To classify, perform a "vote" across all of the learning algorithms you built.



Evaluating the AdaBoosting performance:

● The F1-score of the AdaBoosting is 0.94. The confusion matrix above indicates that 1,525 data is correctly classified as un-recommend and 11,020 feedback is correctly classified as recommended.

● 954 feedbacks from customers who will not recommend the products are incorrectly labeled as will recommend, while 264 feedback from customers who will recommend the products are inaccurately labeled as not recommended.

● Advantage of AdaBoost in Python is its ability to improve the performance of weak learners and create a strong ensemble model.

● In this scenario, AdaBoost demonstrated superior performance compared to Logistic Regression, Decision Tree, and Random Forest models. However, it should be noted that the observed differences were relatively small, with a maximum margin of 0.05. Despite not being statistically significant, these minor variations in performance still highlight the potential advantage of AdaBoost over the aforementioned models.

Appendix (Python):

```
# In[2]:


import nltk
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re
import plotly.express as px
import plotly.graph_objects as go


from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

nltk.download("punkt")
nltk.download('stopwords')
nltk.download('wordnet')

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from collections import Counter
from wordcloud import WordCloud




nltk.download("punkt")
nltk.download('stopwords')

nltk.download('wordnet')

# Importing plotly and cufflinks in offline mode
import plotly.express as px
import plotly.offline


# Ignore Warnings
import warnings
```

```python
warnings.filterwarnings("ignore")
warnings.warn("this will not show")

# Figure&Display options
get_ipython().run_line_magic('matplotlib', 'inline')
fig, ax = plt.subplots()
# fig.set_size_inches(10, 6)
plt.rcParams["figure.figsize"] = (12, 8)  # the size of A4 paper use (11.7, 8.27)
pd.set_option('max_colwidth', 200)
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 200)
pd.set_option('display.float_format', lambda x: '%.2f' % x)


import ipywidgets
from ipywidgets import interact


# In[3]:


df1 = pd.read_csv('/Users/Raghavan/Downloads/Womens Clothing E-Commerce Reviews.csv')
df = df1.copy()
df.head()


# In[4]:


def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1,
keys=['Missing_Number', 'Missing_Percent'])
    return missing_values[missing_values['Missing_Number']>0]

###############################################################################

def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('-'*79, 'red', attrs=['bold']),
          colored("\nInfo:\n", attrs=['bold']), sep='')
    print(df.info(), '\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", attrs=['bold']), df.nunique(),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values:\n", attrs=['bold']), missing_values(df),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", attrs=['bold']), list(df.columns),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    df.columns= df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')

    print(colored("Columns after rename:", attrs=['bold']), list(df.columns),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')


def multicolinearity_control(df):
    feature =[]
    collinear=[]
    for col in df.corr().columns:
        for i in df.corr().index:
            if (abs(df.corr()[col][i])> .9 and abs(df.corr()[col][i]) < 1):
                    feature.append(col)
                    collinear.append(i)
                    print(colored(f"Multicolinearity alert in between:{col} - {i}",
                                  "red", attrs=['bold']), df.shape,'\n',
                                  colored('-'*79, 'red', attrs=['bold']), sep='')

def duplicate_values(df):
    print(colored("Duplicate check...", attrs=['bold']), sep='')
    duplicate_values = df.duplicated(subset=None, keep='first').sum()
    if duplicate_values > 0:
        df.drop_duplicates(keep='first', inplace=True)
        print(duplicate_values, colored("Duplicates were dropped!"),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')
    else:
        print(colored("There are no duplicates"),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')

def drop_columns(df, drop_columns):
    if drop_columns !=[]:
        df.drop(drop_columns, axis=1, inplace=True)
        print(drop_columns, 'were dropped')
    else:
        print(colored('We will now check the missing values and if necessary will drop
realted columns!', attrs=['bold']),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')

def drop_null(df, limit):
    print('Shape:', df.shape)
    for i in df.isnull().sum().index:
        if (df.isnull().sum()[i]/df.shape[0]*100)>limit:
            print(df.isnull().sum()[i], 'percent of', i ,'null and were dropped')
            df.drop(i, axis=1, inplace=True)
            print('new shape:', df.shape)
    print('New shape after missing value control:', df.shape)

###############################################################################
```

```
# To view summary information about the column

def first_look(col):
    print("column name    : ", col)
    print("--------------------------------")
    print("per_of_nulls   : ", "%", round(df[col].isnull().sum()/df.shape[0]*100, 2))
    print("num_of_nulls   : ", df[col].isnull().sum())
    print("num_of_uniques : ", df[col].nunique())
    print(df[col].value_counts(dropna = False))

################################################################################




# In[5]:


import colorama
from colorama import Fore, Style  # maakes strings colored
from termcolor import colored


# In[6]:


first_looking(df)


# In[12]:


df.info()


# In[8]:


df = df.dropna()


# In[9]:


df.head(2)


# In[11]:


df.info()
```

```python
# In[184]:


plt.style.use('ggplot')
plt.figure(figsize=(12,6))
sns.distplot(df['age'],kde = False,color="blue")
plt.title("Frequency distribution of customer age")
plt.xlabel("Customer Age")
plt.ylabel("Density")
plt.grid(False)

#distribution of age
sns.histplot(x=df['Age'])
plt.title("Distribution of Age")

#The ratio of recommend vs not-recommend
total = df1['Recommended IND'].value_counts().values.sum()

def fmt(x):
    return '{:.1f}%\n{:.0f}'.format(x, total*x/100)

plt.pie(df1['Recommended IND'].value_counts().values,labels=df1['Recommended
IND'].value_counts().index, autopct=fmt)
plt.title("Ratio of recommend vs not recommend")

#age distribution of different department
df1.boxplot(column='Age',by='Department Name')

sns.countplot(data=df1, x="Rating", hue="Recommended IND")
plt.title('Count of number of ratings')
plt.xlabel('Rating')
plt.ylabel('No. of rating')

#correlation heatmap
sns.heatmap(num_cols.corr(), annot=True)
plt.title('Correlation heatmap')
plt.show()


# Most of the customers are in the age group of 30-50

# In[96]:


plt.hist(df['rating'],color="purple")
plt.title("Distribution of Ratings", size=20)
plt.xlabel("Rating score")
plt.ylabel("No of reviews")
plt.xlim(1,5)
plt.grid(False)
plt.xticks(np.arange(df['rating'].min(), df['rating'].max()+1, 1))


# Most of the customers were happy with the products and higher ratings of 4 and 5 were given
for the products

# In[ ]:


sns.stripplot(y="age", x="rating", hue="recommended_ind", data=df, palette="husl");


# In[100]:


plt.figure(figsize=(8, 8))

ax =df.rating.value_counts()
labels=df['rating'].value_counts().index
plt.pie(ax,labels=labels,autopct='%1.1f%%')
plt.title("Distribution of Ratings",fontsize=25,color='purple')
plt.legend(loc='upper right')
plt.show()


# Most of the customers were happy with the products and around 75% of the customers have
given a rating of 4 and 5

# In[117]:


fig = px.treemap(df,
                 path=['division_name','department_name','class_name'],
                 title = 'Number of Clothes by division, department and class of products',
                 color_continuous_scale="rdylbu",
                 color="rating",
                 width=1200, height=600)

fig.update_traces(textinfo='label+value', textfont_size=13,
                  marker=dict(line=dict(color='white', width=0.2)))
fig.update_layout(font = dict(size=17, family = 'Franklin Gothic'))
fig.show()


# In[120]:


recommended = df[df['recommended_ind']==1]
```

```
not_recommended = df[df['recommended_ind']==0]


# In[121]:


fig = plt.figure(figsize=(14, 14))
ax1 = plt.subplot2grid((2, 2), (0, 0))
ax1 = sns.countplot(recommended['division_name'], color = "green", alpha = 0.8, label = 
"Recommended",
                   order = recommended['division_name'].value_counts().index)
ax1 = sns.countplot(not_recommended['division_name'], color = "red", alpha = 0.8, label = 
"Not Recommended",
                   order = recommended['division_name'].value_counts().index)
ax1 = plt.title("Recommended Items in each Division")

ax1 = plt.legend()

ax2 = plt.subplot2grid((2, 2), (0, 1))
ax2 = sns.countplot(recommended['department_name'], color="yellow", alpha = 0.8, label = 
"Recommended",
                   order = recommended['department_name'].value_counts().index)
ax2 = sns.countplot(not_recommended['department_name'], color="purple", alpha = 0.8, label = 
"Not Recommended",
                   order = recommended['department_name'].value_counts().index)
ax2 = plt.title("Recommended Items in each Department")
ax2 = plt.legend()

ax3 = plt.subplot2grid((2, 2), (1, 0), colspan=2)
ax3 = plt.xticks(rotation=45)
ax3 = sns.countplot(recommended['class_name'], color="cyan", alpha = 0.8, label = 
"Recommended",
                    order = recommended['class_name'].value_counts().index)
ax3 = sns.countplot(not_recommended['class_name'], color="blue", alpha = 0.8, label = "Not 
Recommended",
                    order = recommended['class_name'].value_counts().index)
ax3 = plt.title("Recommended Items in each Class")
ax3 = plt.legend()


# In[163]:


# from raw value to percentage
plt.figure(figsize=(12,6))
import matplotlib.patches as mpatches
total = df.groupby('division_name')['clothing_id'].count().reset_index()
recommend = 
df[df.recommended_ind==1].groupby('division_name')['clothing_id'].count().reset_index()
recommend['clothing_id'] = [i / j * 100 for i,j in zip(recommend['clothing_id'],
total['clothing_id'])]
total['clothing_id'] = [i / j * 100 for i,j in zip(total['clothing_id'],
total['clothing_id'])]

# bar chart 1 -> top bars (group of 'smoker=No')
bar1 = sns.barplot(x="division_name",  y="clothing_id", data=total, color='red')

# bar chart 2 -> bottom bars (group of 'smoker=Yes')
bar2 = sns.barplot(x="division_name", y="clothing_id", data=recommend, color='green')

# add legend
top_bar = mpatches.Patch(color='red', label='Not Recommended')
bottom_bar = mpatches.Patch(color='green', label='Recommended')
plt.legend(handles=[top_bar, bottom_bar],loc='right')

# show the graph
plt.title("Percentage of Recommended clothes in each division")
plt.xlabel("Division name")
plt.ylabel("Percentage of Reviews")
plt.show()


# In[164]:


# from raw value to percentage
plt.figure(figsize=(12,6))
import matplotlib.patches as mpatches
total = df.groupby('department_name')['clothing_id'].count().reset_index()
recommend =
df[df.recommended_ind==1].groupby('department_name')['clothing_id'].count().reset_index()
recommend['clothing_id'] = [i / j * 100 for i,j in zip(recommend['clothing_id'],
total['clothing_id'])]
total['clothing_id'] = [i / j * 100 for i,j in zip(total['clothing_id'],
total['clothing_id'])]

# bar chart 1 -> top bars (group of 'smoker=No')
bar1 = sns.barplot(x="department_name",  y="clothing_id", data=total, color='purple')

# bar chart 2 -> bottom bars (group of 'smoker=Yes')
bar2 = sns.barplot(x="department_name", y="clothing_id", data=recommend, color='yellow')

# add legend
top_bar = mpatches.Patch(color='purple', label='Not Recommended')
bottom_bar = mpatches.Patch(color='yellow', label='Recommended')
plt.legend(handles=[top_bar, bottom_bar],loc='right')

# show the graph
plt.title("Percentage of Recommended clothes in each department")
plt.xlabel("Department name")
plt.ylabel("Percentage of Reviews")
plt.show()


# In[160]:



# from raw value to percentage
plt.figure(figsize=(22,6))
import matplotlib.patches as mpatches
total = df.groupby('class_name')['clothing_id'].count().reset_index()
recommend =
df[df.recommended_ind==1].groupby('class_name')['clothing_id'].count().reset_index()
recommend['clothing_id'] = [i / j * 100 for i,j in zip(recommend['clothing_id'],
total['clothing_id'])]
total['clothing_id'] = [i / j * 100 for i,j in zip(total['clothing_id'],
total['clothing_id'])]

# bar chart 1 -> top bars (group of 'smoker=No')
bar1 = sns.barplot(x="class_name",  y="clothing_id", data=total, color='blue')

# bar chart 2 -> bottom bars (group of 'smoker=Yes')
bar2 = sns.barplot(x="class_name", y="clothing_id", data=recommend, color='cyan')

# add legend
top_bar = mpatches.Patch(color='blue', label='Not Recommended')
bottom_bar = mpatches.Patch(color='cyan', label='Recommended')
plt.legend(handles=[top_bar, bottom_bar],loc='right')

# show the graph
plt.title("Percentage of Recommended clothes in each class")
plt.xlabel("Class name")
plt.ylabel("Percentage of Reviews")
plt.show()


# In[30]:



get_ipython().system('pip install sqldf')
import sqldf as sql


# In[129]:



query = """
SELECT division_name, count(*) as count
FROM df
group by division_name
```

```
order by count desc;
"""


# In[130]:


recommended_df= sql.run(query)


# In[131]:


recommended_df


# In[127]:


total = df.groupby('division_name')['clothing_id'].count().reset_index()


# In[128]:


total


# In[87]:


query_1 = """
SELECT division_name, department_name,class_name,avg(rating) as Avg_Rating
FROM df
group by division_name, department_name,class_name
order by Avg_Rating desc;
"""


# In[88]:


sunburst_df = sql.run(query_1)


# In[89]:


sunburst_df
```

```python
# In[90]:


fig = px.sunburst(sunburst_df, path=['division_name', 'department_name','class_name'],
                  values='Avg_Rating',color='Avg_Rating')
fig.show()


# In[166]:


query_2 = """
SELECT division_name, department_name,class_name,avg(age) as Avg_Age
FROM df
group by division_name, department_name,class_name
order by Avg_Age desc;
"""


# In[167]:


age_df = sql.run(query_2)


# In[168]:


age_df


# In[181]:


fig_1 = px.sunburst(age_df, path=['division_name', 'department_name','class_name'],
                  values='Avg_Age',color='Avg_Age',
                   color_continuous_scale='rdylbu')
fig_1.show()


# In[9]:


from matplotlib.patches import Patch


plt.xticks(rotation=45)
plt.xlabel('Clothing ID')
plt.ylabel('No of reviews')
plt.title("Top 20 most reviewed clothes")
colours = {"Dresses": "blue", "Bottoms": "green","Tops": "orange","Intimate":
"yellow","Jackets": "purple",
          "Trend": "red"}
df['clothing_id'].value_counts()[:20].plot(kind='bar',color=df['department_name'].replace(col
ours)).legend(
    [
        Patch(facecolor=colours['Dresses']),
        Patch(facecolor=colours['Tops']),
        Patch(facecolor=colours['Bottoms']),
        Patch(facecolor=colours['Intimate']),
        Patch(facecolor=colours['Jackets']),
        Patch(facecolor=colours['Trend'])
    ], ["Dresses", "Tops", "Bottoms", "Intimate","Jackets","Trend"]
)


# In[89]:


df['department_name'].unique()


#

# In[191]:


get_ipython().system('pip install pywaffle')
from pywaffle import Waffle


# In[197]:


# Prepare Data
df_waffle = df.groupby('class_name').size().reset_index(name='counts')
n_categories = df_waffle.shape[0]
colors = [plt.cm.inferno_r(i/float(n_categories)) for i in range(n_categories)]

# Draw Plot and Decorate
fig = plt.figure(
    FigureClass=Waffle,
    plots={
        '111': {
            'values': df_waffle['counts'],
            'labels': ["{0} ({1})".format(n[0], n[1]) for n in df_waffle[['class_name',
'counts']].itertuples()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), 'fontsize': 12},
            'title': {'label': '# Clothes by Class', 'loc': 'center', 'fontsize':18}
        },
    },
    rows=20,
    colors=colors,
    figsize=(16, 9)
)


# In[306]:


df['division_name'].unique()


# In[196]:


n_categories


# In[201]:


# Prepare Data
fig = plt.figure(figsize=(14, 9))
df_waffle = df.groupby('division_name').size().reset_index(name='counts')
n_categories = df_waffle.shape[0]
colors = [plt.cm.inferno_r(i/float(n_categories)) for i in range(n_categories)]

# Draw Plot and Decorate
fig = plt.figure(
    FigureClass=Waffle,
    plots={
        '111': {
            'values': df_waffle['counts'],
            'labels': ["{0} ({1})".format(n[0], n[1]) for n in df_waffle[['division_name',
'counts']].itertuples()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), 'fontsize': 12},
            'title': {'label': '# Clothes by Division', 'loc': 'center', 'fontsize':18}
        },
    },
    rows=3,
    colors=colors,
    figsize=(16, 9)
)


# In[13]:
```

```python
get_ipython().system('pip3 install catboost')


# In[14]:


get_ipython().system('pip install textblob')


# In[15]:


from textblob import *
df['ReviewText_Polarity'] = df['review_text'].map(lambda text:
TextBlob(text).sentiment.polarity)


# In[16]:


df.head()


# In[17]:


px.histogram(df, x = 'ReviewText_Polarity', color_discrete_sequence=['purple'])


# In[19]:


df['Sentiment'] = ''
df.loc[df['ReviewText_Polarity']>=0,'Sentiment']='Positive'
df.loc[df['ReviewText_Polarity']<0,'Sentiment']='Negative'


# In[20]:


fig7 = px.histogram(df,
            x="Sentiment",
            y="recommended_ind",
            color="recommended_ind",
            barmode="group",
            histnorm='percent',
            histfunc = "count",
            color_discrete_sequence=["orange", "blue"],
            title=f"Recommendation-Sentiment Relation")
```

```python
fig7.update_layout(yaxis_title="Percentage")
fig7.update_xaxes(type='category')
fig7.update_layout(yaxis={"ticksuffix":"%"})
fig7.show()


# ## Text Pre-processing steps

# In[21]:


df["review_text"] = df["review_text"].str.lower()


# In[22]:


def punctuation_removal(punc):
    rem_punc = [i for i in punc if i not in string.punctuation]
    after_punc = ''.join(rem_punc)
    return after_punc


# In[23]:


import string
df['review_text'] = df['review_text'].apply(punctuation_removal)
df.head()


# In[24]:


url_regex = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'

# write a regular expression to identify non-ascii characters in text
non_ascii_regex = r'[^\x00-\x7F]+'


# In[25]:


def clean_regex(text_variable):

    # use library re to replace urls by token - urlplaceholder
    text_variable = re.sub(url_regex, 'urlplaceholder', text_variable)

    # use library re to replace non ascii characters by a space
    text_variable = re.sub(non_ascii_regex, ' ', text_variable)
    return text_variable


# In[26]:


df["review_text"] = np.vectorize(clean_regex)(df["review_text"])


# In[27]:


# Remove stopwords

stop = stopwords.words('english')
df['review_text'] = df['review_text'].apply(lambda x: ' '.join([word for word in x.split() if
word not in (stop)]))


# In[28]:


df.head()


# In[29]:


lemmatizer = WordNetLemmatizer()

# function to convert nltk tag to wordnet tag

def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_sentence(sentence):
    #tokenize the sentence and find the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)


# In[32]:


# import WordNetLemmatizer from nltk library
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
df['review_text'] = df['review_text'].apply(lemmatize_sentence)


# In[33]:




from sklearn.feature_extraction.text import CountVectorizer


# In[34]:


df_modeldata = df[['review_text', 'recommended_ind']]


# In[35]:


df_modeldata.head()


# In[36]:


df_modeldata.reset_index(drop=True, inplace=True)
df_modeldata.index


# In[193]:
```

```python
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(ngram_range=(1, 2), max_features = 10000, stop_words='english')

bow = vectorizer.fit_transform(df_modeldata['review_text'])

df_bow = pd.DataFrame(bow.todense())

df_bow


# In[194]:


from sklearn.feature_extraction.text import TfidfVectorizer

tfidf=TfidfVectorizer(ngram_range=(1, 2),max_features = 10000,stop_words='english')

tfidf_matrix=tfidf.fit_transform(df_modeldata['review_text'])

df_tfidf = pd.DataFrame(tfidf_matrix.todense())

df_tfidf


# In[195]:


train_bow = bow

train_bow.todense()


# In[197]:


train_bow.shape


# In[196]:


train_tfidf_matrix = tfidf_matrix

train_tfidf_matrix.todense()
```

```
# ## LDA Topic Modelling
#

# In[139]:


get_ipython().system('pip install pyLDAvis')


# In[140]:


import gensim #the library for Topic modelling
from gensim.models.ldamulticore import LdaMulticore
from gensim import corpora, models
import pyLDAvis.gensim #LDA visualization library


# In[354]:


#create dictionary

text_clean = [text.split() for text in df_modeldata['review_text']]

dictionary = corpora.Dictionary(text_clean)
#Total number of non-zeroes in the BOW matrix (sum of the number of unique words per document
over the entire corpus).
print(dictionary.num_nnz)


# In[238]:


#create document term matrix
doc_term_matrix = [dictionary.doc2bow(text) for text in text_clean]
print(len(doc_term_matrix))


# In[144]:


lda = gensim.models.ldamulticore.LdaMulticore


# In[223]:


num_topics=3
get_ipython().run_line_magic('time', 'ldamodel =
lda(doc_term_matrix,num_topics=num_topics,id2word=dictionary,minimum_probability=0)')


# In[224]:


ldamodel.print_topics(num_topics=num_topics)


# In[225]:


lda_display = pyLDAvis.gensim.prepare(ldamodel, doc_term_matrix, dictionary,
sort_topics=False, mds='mmds')
pyLDAvis.display(lda_display)


# In[355]:


# Assigns the topics to the documents in corpus
lda_corpus = ldamodel[doc_term_matrix]
print(len(lda_corpus))


# In[227]:


from gensim.models import CoherenceModel
# Compute Perplexity
print('\nPerplexity: ', ldamodel.log_perplexity(doc_term_matrix))
# a measure of how good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=ldamodel, texts=text_clean, dictionary=dictionary,
coherence='c_v')

coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)


# In[228]:


import warnings
warnings.simplefilter('ignore')
from itertools import chain


# In[356]:
```

```
#Threshold - Average of the topic probabilities in the reviews

scores = list(chain(*[[score for topic_id,score in topic]                       for topic in
[doc for doc in lda_corpus]]))

threshold = sum(scores)/len(scores)
print(threshold)


# In[324]:


cluster1 = [j for i,j in zip(lda_corpus,df_modeldata.index) if i[0][1] > threshold]
cluster2 = [j for i,j in zip(lda_corpus,df_modeldata.index) if i[1][1] > threshold]
cluster3 = [j for i,j in zip(lda_corpus,df_modeldata.index) if i[2][1] > threshold]
#cluster4 = [j for i,j in zip(lda_corpus,df.index) if i[3][1] > threshold]
#cluster5 = [j for i,j in zip(lda_corpus,df.index) if i[4][1] > threshold]

print(len(cluster1))
print(len(cluster2))
print(len(cluster3))
#print(len(cluster4))
#print(len(cluster5))


# In[325]:


df_modeldata.iloc[cluster1]


# In[326]:


df_modeldata.iloc[cluster2]


# In[327]:


df_modeldata.iloc[cluster3]


# In[349]:



# lda_model is the trained LDA model
```

```python
# corpus is the document-term matrix in the form of a list of lists
# id2word is the mapping between words and their integer ids

doc_topics = ldamodel.get_document_topics(doc_term_matrix, minimum_probability=0.0)

# Convert the document-topic distribution to a numpy array
doc_topic_matrix = np.array([np.array(doc_topic)[:,1] for doc_topic in doc_topics])

# Get the dominant topic for each document
dominant_topic = np.argmax(doc_topic_matrix, axis=1)

# Map the topic index to the corresponding topic label
topic_labels = [ldamodel.show_topic(topic)[0][0] for topic in dominant_topic]

# Create a pandas dataframe to display the dominant topic and topic label for each document
df = pd.DataFrame({'Dominant Topic': dominant_topic, 'Topic Label': topic_labels})

print(df.head())

df_modeldata['dominat_topic'] = df['Dominant Topic']

df_modeldata['topic_name'] = df['Topic Label']


print(df_modeldata.head())


# In[357]:


print(df_modeldata['topic_name'].unique())


# In[358]:


fig = plt.figure(figsize=(8, 8))

fig = sns.countplot(df_modeldata['topic_name'], color = "teal")

plt.xlabel("Topic name",fontsize="15")
plt.ylabel("Number of mapped reviews",fontsize="15")
plt.title("Topic wise distribution of the reviews",fontsize="15")
plt.xticks(fontsize="15")
plt.show(fig)


# # Splitting the data into training and validation set

# # Bag-of-Words Features
# In[199]:


x_train_bow, x_valid_bow, y_train_bow, y_valid_bow =
train_test_split(train_bow,df_modeldata.recommended_ind.values,
                                                                        test_size=0.3,
                                                                        random_state=2)


# # TF-IDF features

# In[200]:


x_train_tfidf, x_valid_tfidf, y_train_tfidf, y_valid_tfidf =
train_test_split(train_tfidf_matrix,df_modeldata.recommended_ind.values,
                                                                                test_size=0.3,

random_state=17)


# In[96]:


from sklearn.model_selection import train_test_split
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc_curve,
accuracy_score, recall_score, classification_report, f1_score, average_precision_score,
precision_recall_fscore_support, roc_auc_score
from catboost import CatBoostClassifier
from sklearn.linear_model import LogisticRegression


# ###  Bag-of-Words Features
# ### Fitting the Logistic Regression Model.

# In[163]:


Log_Reg = LogisticRegression(random_state=0,solver='lbfgs')
```

```
Log_Reg.fit(x_train_bow,y_train_bow)
prediction_bow = Log_Reg.predict(x_valid_bow)

prediction_bow


# In[164]:


from sklearn import metrics


# converting the results to integer type
prediction_int = prediction_bow.astype(np.int)
prediction_int

# calculating f1 score
log_bow = f1_score(y_valid_bow, prediction_int)

log_bow


# ## Hyper parameter optimization

# In[124]:






# In[125]:






# In[126]:






# In[128]:






# In[ ]:
```

```python
# In[ ]:




# ### TF-IDF Features
# ### Fitting the Logistic Regression Model.

# In[165]:


Log_Reg.fit(x_train_tfidf,y_train_tfidf)

prediction_tfidf = Log_Reg.predict(x_valid_tfidf)

prediction_tfidf


# ### Calculating the F1 Score

# In[168]:




prediction_int = prediction_tfidf.astype(np.int)
prediction_int

# calculating f1 score
log_tfidf = f1_score(y_valid_tfidf, prediction_int)

log_tfidf


# # Decision Trees

# In[167]:


dct = DecisionTreeClassifier(criterion='entropy', random_state=1)


# ### Bag-of-Words Features
# ### Fitting the Decision Tree model.
# In[169]:


dct.fit(x_train_bow,y_train_bow)

dct_bow = dct.predict(x_valid_bow)

dct_bow


# ### Calculating the F1 Score

# In[170]:




# converting the results to integer type
dct_int_bow=dct_bow.astype(np.int)

# calculating f1 score
dct_score_bow=f1_score(y_valid_bow,dct_int_bow)

dct_score_bow


# # TF-IDF Features
# ### Fitting the Decision Tree model

# In[171]:


dct.fit(x_train_tfidf,y_train_tfidf)

dct_tfidf = dct.predict(x_valid_tfidf)

dct_tfidf


# ### Calculating the F1 Score

# In[172]:




# converting the results to integer type
dct_int_tfidf=dct_tfidf.astype(np.int)

# calculating f1 score
```

```
dct_score_tfidf=f1_score(y_valid_tfidf,dct_int_tfidf)

dct_score_tfidf


# # Random Forest

# In[220]:


rf =  RandomForestClassifier()


# ### Bag-of-Words Features
# ### Fitting the Random Forest model.

# In[256]:


rf.fit(x_train_bow,y_train_bow)

rf_bow = rf.predict(x_valid_bow)

rf_bow


# ### Calculating the F1 Score

# In[257]:




# converting the results to integer type
rf_int_bow=rf_bow.astype(np.int)

# calculating f1 score
rf_score_bow=f1_score(y_valid_bow,rf_int_bow)

rf_score_bow


# In[260]:


rf =  RandomForestClassifier()
rf.get_params().keys()


# In[261]:
```

```python
param_grid = {
    'n_estimators': [25, 50, 100, 150, 200, 400, 600, 1000, 1200, 1400],
    'max_features': ['sqrt', 'auto', 'log2', None],
    'max_depth': [3, 6, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    'max_leaf_nodes': [3, 6, 9],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10]
}


# In[179]:




# In[262]:


from sklearn.model_selection import RandomizedSearchCV

random_search_rf = RandomizedSearchCV(estimator = rf, param_distributions = param_grid,
n_iter = 50,
                                      cv = 3, verbose=2, random_state=42, n_jobs = -1)


# In[263]:


from datetime import datetime

start_time = timer(None) # timing starts from this point for "start_time" variable
random_search_rf.fit(x_train_bow,y_train_bow)
timer(start_time) # timing ends here for "start_time" variable


# In[264]:


random_search_rf.best_estimator_


# In[265]:


rf_hyper_tuning =  RandomForestClassifier(max_depth=20, max_features=None, max_leaf_nodes=9,
                       min_samples_leaf=4, min_samples_split=10,
                       n_estimators=25)
```

```
# In[266]:


rf_hyper_tuning.fit(x_train_bow,y_train_bow)

rf_bow_tuning = rf_hyper_tuning.predict(x_valid_bow)

rf_bow_tuning


# In[268]:




# converting the results to integer type
rf_int_bow_tuning=rf_bow_tuning.astype(np.int)

# calculating f1 score
rf_score_bow_tuning=f1_score(y_valid_bow,rf_int_bow_tuning)

rf_score_bow_tuning


# ### TF-IDF Features
# ### Fitting the Random Forest model

# In[176]:


rf.fit(x_train_tfidf,y_train_tfidf)

rf_tfidf = rf.predict(x_valid_tfidf)

rf_tfidf


# ### Calculating the F1 Score

# In[177]:




# converting the results to integer type
rf_int_tfidf=rf_tfidf.astype(np.int)

# calculating f1 score
rf_score_tfidf=f1_score(y_valid_tfidf,rf_int_tfidf)
```

```
rf_score_tfidf


# In[269]:


rf_hyper_tuning.fit(x_train_tfidf,y_train_tfidf)

rf_tfidf_tuning = rf_hyper_tuning.predict(x_valid_tfidf)

rf_tfidf_tuning


# In[270]:



# converting the results to integer type
rf_tfidf_tuning_int=rf_tfidf_tuning.astype(np.int)

# calculating f1 score
rf_score_tfidf_tuning=f1_score(y_valid_tfidf,rf_tfidf_tuning_int)

rf_score_tfidf_tuning


# # Boosting Models

# In[71]:


from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier


# In[72]:


import xgboost as XGB


# In[140]:


gradient_booster = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
                  max_depth=3, loss = "exponential", random_state=606)

xg_boost = XGB.XGBClassifier()

ada_boost = AdaBoostClassifier(n_estimators = 600, learning_rate = 0.2)
```

```
# ## Gradient Booster

# In[112]:


gradient_booster.fit(x_train_bow,y_train_bow)

gradient_booster_bow = gradient_booster.predict(x_valid_bow)

gradient_booster_bow


# In[113]:


# converting the results to integer type
gradient_booster_int_bow=gradient_booster_bow.astype(np.int)

# calculating f1 score
gradient_booster_score_bow=f1_score(y_valid_bow,gradient_booster_int_bow)

gradient_booster_score_bow


# In[114]:


gradient_booster.fit(x_train_tfidf,y_train_tfidf)

gradient_booster_tfidf = gradient_booster.predict(x_valid_tfidf)

gradient_booster_tfidf


# In[115]:


# converting the results to integer type
gradient_booster_tfidf_int=gradient_booster_tfidf.astype(np.int)

# calculating f1 score
gradient_booster_tfidf_score=f1_score(y_valid_tfidf,gradient_booster_tfidf_int)

gradient_booster_tfidf_score


# ## XGBoost
# In[141]:


xg_boost.fit(x_train_bow,y_train_bow)

xg_boost_bow = xg_boost.predict(x_valid_bow)

xg_boost_bow


# In[142]:


# converting the results to integer type
xg_boost_int_bow=xg_boost_bow.astype(np.int)

# calculating f1 score
xg_boost_score_bow=f1_score(y_valid_bow,xg_boost_int_bow)

xg_boost_score_bow


# In[138]:


## Hyperparameter optimization using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV


# In[137]:


def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin,
round(tsec, 2)))


# In[139]:


## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}


# In[143]:


random_search=RandomizedSearchCV(xg_boost,param_distributions=params,n_iter=5,scoring='roc_au
c',n_jobs=-1,cv=5,verbose=3)


# In[144]:


from datetime import datetime
# Here we go
start_time = timer(None) # timing starts from this point for "start_time" variable
random_search.fit(x_train_bow,y_train_bow)
timer(start_time) # timing ends here for "start_time" variable


# In[145]:


random_search.best_estimator_


# In[153]:


xg_boost = XGB.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.4, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=12,
              min_child_weight=5, monotone_constraints='()',
              n_estimators=100, n_jobs=1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)


# In[154]:


xg_boost.fit(x_train_bow,y_train_bow)

xg_boost_bow = xg_boost.predict(x_valid_bow)
```

```
xg_boost_bow


# In[155]:


# converting the results to integer type
xg_boost_int_bow=xg_boost_bow.astype(np.int)

# calculating f1 score
xg_boost_score_bow=f1_score(y_valid_bow,xg_boost_int_bow)

xg_boost_score_bow


# In[118]:


xg_boost.fit(x_train_tfidf,y_train_tfidf)

xg_boost_tfidf = xg_boost.predict(x_valid_tfidf)

xg_boost_tfidf


# In[119]:


# converting the results to integer type
xg_boost_tfidf_int =xg_boost_tfidf.astype(np.int)

# calculating f1 score
xg_boost_tfidf_score=f1_score(y_valid_tfidf,xg_boost_tfidf_int)

xg_boost_tfidf_score


# ## Ada Boost

# In[120]:


ada_boost.fit(x_train_bow,y_train_bow)

ada_boost_bow = ada_boost.predict(x_valid_bow)

ada_boost_bow

# In[83]:
```

```
# converting the results to integer type
ada_boost_bow_int=ada_boost_bow.astype(np.int)

# calculating f1 score
ada_boost_bow_score=f1_score(y_valid_bow,ada_boost_bow_int)

ada_boost_bow_score

# In[121]:

ada_boost.fit(x_train_tfidf,y_train_tfidf)

ada_boost_tfidf = ada_boost.predict(x_valid_tfidf)

ada_boost_tfidf

# In[85]:

# converting the results to integer type
ada_boost_tfidf_int =ada_boost_tfidf.astype(np.int)

# calculating f1 score
ada_boost_tfidf_score=f1_score(y_valid_tfidf,ada_boost_tfidf_int)

ada_boost_tfidf_score


# ### Plotting F1 scores

# In[178]:

Algo_1 = ['LogisticRegression(Bag-of-Words)','LogisticRegression(TF-IDF)',
          'DecisionTree(Bag-of-Words)','DecisionTree(TF-IDF)',
          'RandomForest(Bag-of-Words)','RandomForest(TF-IDF)']

score_1 = [log_bow,log_tfidf,dct_score_bow,dct_score_tfidf,rf_score_bow,rf_score_tfidf]

compare_1 = pd.DataFrame({'Model':Algo_1,'F1_Score':score_1},index=[i for i in range(1,7)])

compare_1.T

# In[123]:


plt.figure(figsize=(18,5))


sns.pointplot(x='Model',y='F1_Score',data=compare_1)
```

```python
plt.title('F1-score comparison')
plt.xlabel('MODEL')
plt.ylabel('SCORE')

plt.show()


# ## Confusion Matrix for the models

# In[194]:


# compute the confusion matrix
#Logistic Regression
#Bag of words
a = 3  # number of rows
b = 4  # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(25, 18))



cm = {
    'Logistic Regression BOW Confusion Matrix' :
confusion_matrix(y_valid_bow,prediction_bow),
    'Logistic Regression TF-IDF Confusion Matrix' :
confusion_matrix(y_valid_tfidf,prediction_tfidf),
    'Decision Tree BOW Confusion Matrix' : confusion_matrix(y_valid_bow,dct_bow),
    'Decision Tree TF-IDF Confusion Matrix' : confusion_matrix(y_valid_tfidf,dct_tfidf),
    'Random Forest BOW Confusion Matrix' : confusion_matrix(y_valid_bow,rf_bow),
    'Random Forest TF-IDF Confusion Matrix' : confusion_matrix(y_valid_tfidf,rf_tfidf)
}

for name, cm in cm.items():

    plt.subplot(a, b, c)
    #plt.title(name)
    fig.subplots_adjust(left=None, bottom=None, right= None, top=None, wspace=0.4, hspace=
0.4)
    #Plot the confusion matrix.
    sns.heatmap(cm,
                annot=True,
                xticklabels=['Not Recommended', 'Recommended'],
                yticklabels=['Not Recommended', 'Recommended'],
                fmt='.2f', square=True, annot_kws={"size": 16}, cmap =
'Purples').set_title(name, fontsize = 20)
    plt.ylabel('Actual',fontsize=13)
    plt.xlabel('Predicted',fontsize=13)
    plt.title(name)
    c = c + 1

plt.show()


# In[201]:



from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt


y_test_pred_log_reg = Log_Reg.predict_proba(x_valid_bow)[:,1]

#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_valid_bow, y_test_pred_log_reg)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Logistic Regression BOW Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()

# In[202]:

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt

y_test_pred_dct_reg = dct.predict_proba(x_valid_bow)[:,1]

#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_valid_bow, y_test_pred_dct_reg)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Decision Tree BOW Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
```

```python
#display plot
plt.show()

# In[203]:
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt


y_test_pred_rf_reg = rf.predict_proba(x_valid_bow)[:,1]

#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_valid_bow, y_test_pred_rf_reg)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Random Forest BOW Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()

#SVM classifier
from sklearn import svm
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(X_trainds, y_train)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(X_testds)

# Use accuracy_score function to get the accuracy
from sklearn import metrics
from sklearn.metrics import f1_score, roc_auc_score,balanced_accuracy_score,
classification_report, confusion_matrix
print("SVM Accuracy Score -> ",metrics.accuracy_score(predictions_SVM, y_test))

# calculating f1 score of SVM
f1_svm = f1_score(y_test, predictions_SVM)
f1_svm

confusion_svm= confusion_matrix(predictions_SVM,y_test)
confusion_svm

from sklearn.metrics import precision_recall_curve

import matplotlib.pyplot as plt
```

```
#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_test,predictions_SVM)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='blue',label="SVM Classifier")

#add axis labels to plot
ax.set_title('SVM Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()

#SVM classifier
sns.heatmap(confusion_nb,annot=True,fmt='.0f')
plt.show()

#naive bayes classifier
from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()
nb_model.fit(X_trainds, y_train)

y_pred_nb = nb_model.predict(X_testds)

from sklearn import metrics
from sklearn.metrics import f1_score, roc_auc_score,balanced_accuracy_score,
classification_report, confusion_matrix
print("Naive Bayes accuracy score",metrics.accuracy_score(y_pred_nb,y_test))

# calculating f1 score of naive bayes
f1_nb = f1_score(y_test, y_pred_nb)
f1_nb

confusion_nb= confusion_matrix(y_pred_nb,y_test)
confusion_nb

from sklearn.metrics import precision_recall_curve

import matplotlib.pyplot as plt
#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_nb)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple',label="Naive Bayes Classifier")

#add axis labels to plot
```

```
ax.set_title('Naive Bayes Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()

#Decision tree classifier
sns.heatmap(confusion_nb,annot=True,fmt='.0f')
plt.show()
```

Appendix (R):

```
setwd("C:\\Users\\14083\\Desktop")
wc <- read.csv("Womens Clothing E-Commerce Reviews.csv", stringsAsFactors = T,
            header=T)
install.packages("DataExplorer")
library(DataExplorer)
library("dplyr")
library("tidyr")
library(psych)

describe(wc)
str(wc)
wc2 <- wc
wc2[wc2 == "" | wc2 == " "] <- NA

colSums(is.na(wc2))
plot_missing(wc2)
# check the NA value in rows
subset(wc2,is.na(Department.Name))

wc2 <- na.omit(wc2)
str(wc2)

# data numeric
wc.num <- subset(wc2,select = c("Rating", "Age","Positive.Feedback.Count","Recommended.IND"))
wc.num

pairs(wc.num)
```