Assignment: 2 **Date: 15/04/23**
Reg. No.: 21BAI1533 **Course: Artificial Intelligence**
Name: Zeel Mehta **Course Code: BCSE306L**

AI - RESEARCH PROJECT

Problem Statement:

Using Machine Learning's Natural Language Processing to filter out good, bad and helpful reviews of Amazon products, and conclude the overall opinions of users about the service or product using graphs and charts.

Abstract:

Surfing through the an e-commerce application or website and not concluding on what to buy because you don't know concretely if the product is good or bad is a problem everyone faces in online shopping. But we do have the user review system to simplify this process for us right? Yes we do but it may or may not be much of a help. Since there are hundreds of reviews to go through, the task maybe tiring and often lead to indecisiveness which may lead to the person not buying the product. If there are first few negative comments but the overall majority is positive, this would be huge a loss to the companies selling the product as well as the customer who would miss out on a great product. Customers often judge by the initial comments because they do not have the time to go through all the reviews.

With this project we aim to simplify and make is visually appealing for the customer to trust the reviews. This is done by using Natural Language Processing in Machine Learning to filter the reviews into categories of good and bad and make charts for user to clearly see the majority opinion and buy the product. With this method even a glance at the charts would be enough for customer to buy the product. The charts will be purely based on experiences of previous users and hence new customers tend to trust such reviews more.

For the customers that are unable to read, visually appealing charts would provide a great help in making their minds to make use of the service or buy the product.

Even for companies, instead of going through thousands of reviews individually, they can make use of these simple charts and filtered data to arrive upon a conclusion about their products' opinion and make improvement and changes according to the audience.

Proposed System:

A simple Natural Language Processing Artificial Intelligence model is proposed to implement this project. A dataset from Kaggle is used to run, train and test our model upon (https://www.kaggle.com/code/mdanasmondol/amazon-food-reviews-sentiment-analysis-with-nltk/input).

Data Exploration:

The dataset is a CSV file with 10 columns and 568,000 rows. The columns contain the:

Product ID : Unique identifier for the product.
User ID : Unique identifier for the user.
Profile Name : Profile name of the user.
HelpfulnessNumerator : Number of users who found the review helpful.
HelpfulnessDenominator : Number of users who indicated whether they found the review helpful or not.
Score : Rating between 1 and 5.
Time : Timestamp for the review.
Summary : Brief summary of the review.
Text : Text of the review.

Each observation in this dataset is a review of a particular business by a particular user.

Data Visualisation:

Initially, a simple data exploration and analysis is done in order to understand the given dataset. Python's inbuilt data visualisation libraries like matplotlib and seaborn are used to visualise the exploratory data. A box plot, histogram and bar charts are used here to classify the given data.

The 'Score' column is used to classify the data numerically. Mean, correlations and a heat map is generated to understand the data's relations with respect to each other.

Text Pre-Processing:

Further, we move on to text processing, which is cleaning of the text (reviews) we have. The main issue with the data is that it is all in text format (strings) and it is needed to classify it. We will need some sort of numerical feature vector in order to perform the classification task. The simplest is the the bag-of-words approach, where each unique word in a text will be represented by one number.

Vectorisation:

Moving on to Vectorisation, Once the data is cleansed and classified, next step is to prepare the data to train and test on our model. A concept of vectorisation is used where each of those “tokens” are converted into a vector the SciKit Learn's algorithm models can work with.

Each vector will have as many dimensions as there are unique words in the Summary corpus. SciKit Learn's **CountVectorizer** is used and this model will convert a collection of text documents to a matrix of token counts. Imagine it as a 2-Dimensional matrix. Where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents, in this case a column per text message. Since there are so many messages, we can expect a lot of zero counts for the presence of that word in that document.

Train Test Split:

A few methods are run on the cleansed and classified data and the sparse matrix is found. Next the data is split into training and testing data. Initially Multinomial Naives-Bayes classifier from sklearn’s naive-bayes library is used for training and testing purposes. Moving on another method called the TF-IDF transformation is also used which makes use of another feature called pipeline to make predictions.

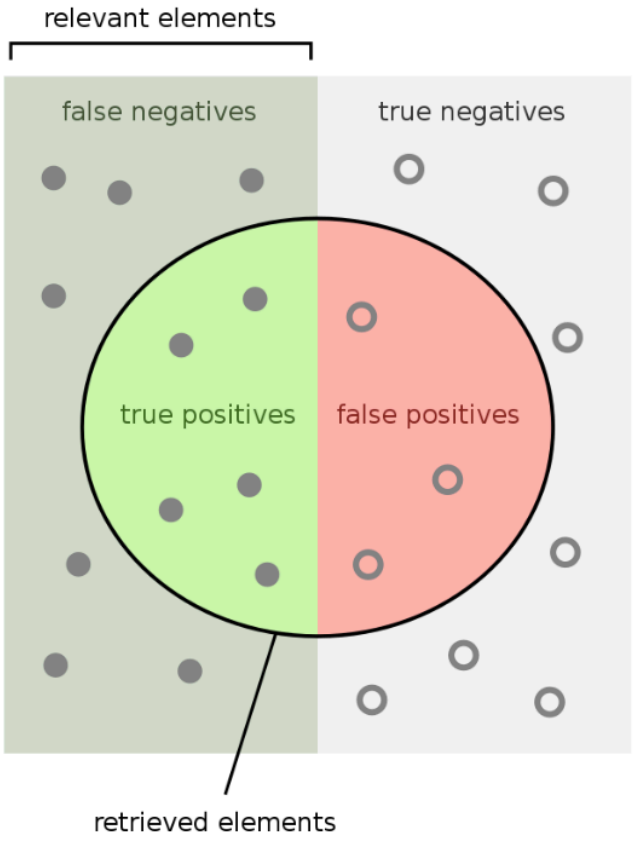
Model Evaluation:

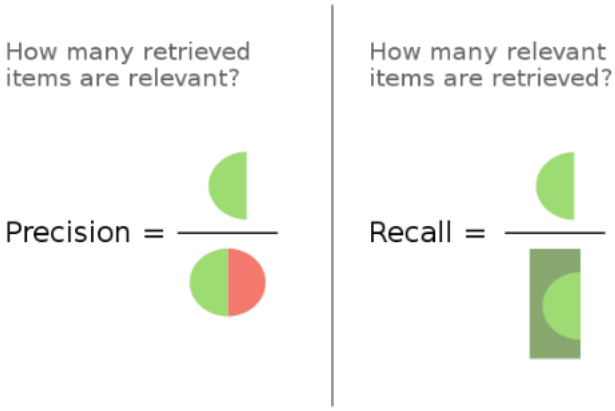
To determine how the model is performing, SciKit Learn's built-in classification report is used, which returns precision, recall, f1-score, and a column for support i.e. meaning how many cases supported that classification.

There are quite a few possible metrics for evaluating model performance. Which one is the most important depends on the task and the business effects of decisions based off of the model.

In the initial "evaluation",we evaluated accuracy on the same data we used for training. You should never actually evaluate on the same dataset you train on!Such evaluation tells us nothing about the true predictive power of our model. If we simply remembered each example during training, the accuracy on training data would trivially be 100%, even though we wouldn't be able to classify any new messages.

A proper way is to split the data into a training/test set, where the model only ever sees the training data during its model fitting and parameter tuning. The test data is never used in any way. This is then our final evaluation on test data is representative of true predictive performance.





Creating a Data Pipeline:

We run model again and then predict off the test set. SciKit Learn's pipeline capabilities are used to store a pipeline of workflow. This will allow to set up all the transformations that we will do to the data for future use. Pipelining gives the classification report for the true testing set.

Comparing and contrasting between the two train test split evaluation methods gives an insight into the actual precision, recall and other parameters to confirm data's true prediction.

Module Description:

Natural Language Processing:

Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.Together, these technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment.There’s a good chance you’ve interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences.

To implement this project, multiple python modules and each are described below:

Pandas:

Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way towards this goal.

Main Features:

1. Easy handling of missing data
2. Size mutability
3. Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
4. Intuitive merging and joining data sets
5. Flexible reshaping and pivoting of data sets
6. Hierarchical labelling of axes
7. Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging

Matplotlib:

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

Seaborn:

Seaborn is an amazing visualisation library for statistical graphics plotting in Python. It provides beautiful default styles and colour palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas.
Seaborn aims to make visualisation the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for same variables for better understanding of dataset.

Scikit Learn:

Scikit-learn is an open source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem. Key concepts and features include.

1. Algorithmic decision-making methods, including.

   - Classification: identifying and categorising data based on patterns.
   - Regression: predicting or projecting data values based on the average mean of existing and planned data.
   - Clustering: automatic grouping of similar data into datasets.

2. Algorithms that support predictive analysis ranging from simple linear regression to neural network pattern recognition.

3. Interoperability with NumPy, pandas, and matplotlib libraries.

ML is a technology that enables computers to learn from input data and to build/train a predictive model without explicit programming. ML is a subset of Artificial Intelligence (AI). It's versatile and integrates well with other Python libraries, such as matplotlib for plotting, numpy for array vectorization, and pandas for dataframes.

- **Count Vectoriser:**

In order to use textual data for predictive modelling, the text must be parsed to remove certain words – this process is called tokenisation. These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms. This process is called feature extraction (or vectorisation).

Scikit-learn's CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

- **Train Test Split:**

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced

- **Confusion Matrix:**

A confusion matrix is a matrix that summarizes the performance of a machine learning model on a set of test data. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance. The matrix displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) produced by the model on the test data.

For binary classification, the matrix will be of a 2X2 table, For multi-class classification, the matrix shape will be equal to the number of classes i.e for n classes it will be nXn.

- **Classification Matrix:**

Classification report or classification matrix gives us the value of parameters like precision, recall, f1-score, support.

The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label a negative sample as positive.

The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights recall more than precision by a factor of beta. beta == 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in y_true.

- **TFID Transformer:**

TF-IDF stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

- **Pipeline:**

Pipeline can be used to chain multiple estimators into one. This is useful as there is often a fixed sequence of steps in processing the data, for example feature selection, normalization and classification. Pipeline serves multiple purposes here:

Convenience and encapsulation
You only have to call fit and predict once on your data to fit a whole sequence of estimators.

Joint parameter selection
You can grid search over parameters of all estimators in the pipeline at once.

Safety
Pipelines help avoid leaking statistics from your test data into the trained model in cross-validation, by ensuring that the same samples are used to train the transformers and predictors.

All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).

NLTK:

NLTK, or Natural Language Toolkit, is a Python package that you can use for NLP.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.” Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more.

Implementation and Results:

In [1]:
```
import nltk
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import string
%matplotlib inline
from sklearn.metrics import accuracy_score
```

In [2]:
```
y=pd.read_csv('Reviews.csv')
```

In [3]:
```
y.head()
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	A
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	

In [4]:
```
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      568454 non-null  int64
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568438 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64
 5   HelpfulnessDenominator  568454 non-null  int64
 6   Score                   568454 non-null  int64
 7   Time                    568454 non-null  int64
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

In [5]:
```
y.describe()
```

Out[5]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	568454.000000	568454.000000	568454.00000	568454.000000	5.684540e+05
mean	284227.500000	1.743817	2.22881	4.183199	1.296257e+09
std	164098.679298	7.636513	8.28974	1.310436	4.804331e+07
min	1.000000	0.000000	0.00000	1.000000	9.393408e+08
25%	142114.250000	0.000000	0.00000	4.000000	1.271290e+09
50%	284227.500000	0.000000	1.00000	5.000000	1.311120e+09

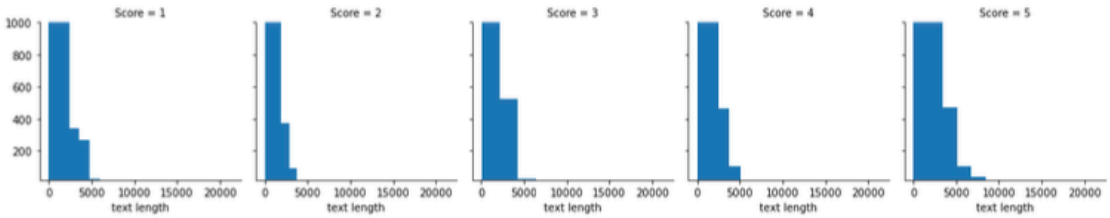
50%	284227.500000	0.000000	1.00000	5.000000	1.311120e+09
75%	426340.750000	2.000000	2.00000	5.000000	1.332720e+09
max	568454.000000	866.000000	923.00000	5.000000	1.351210e+09

In [6]:
```
y['text length']=y['Text'].apply(len)
```

In [7]:
```
import seaborn as sns
```

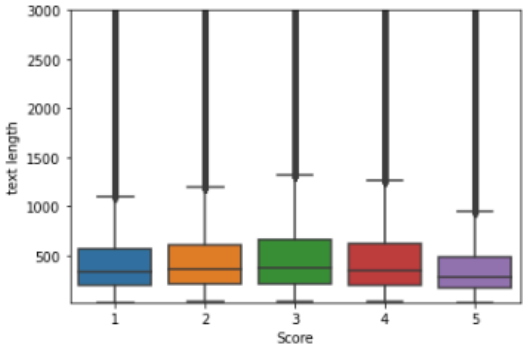
In [8]:
```
f=sns.FacetGrid(y,col='Score')
f.map(plt.hist,'text length')
f.set(ylim=(20,1000))
```

Out[8]:
```
<seaborn.axisgrid.FacetGrid at 0x16672fd00>
```



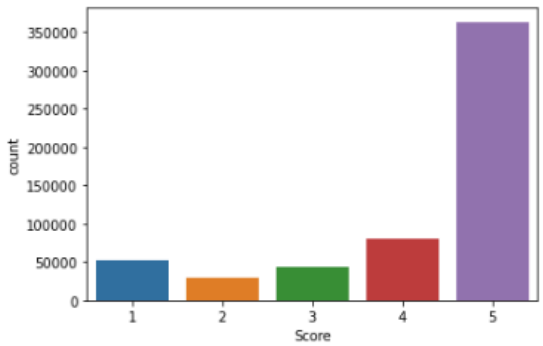
In [9]:
```
ax=sns.boxplot(x='Score',y='text length',data=y)
ax.set(ylim=(20,3000))
```

Out[9]:
```
[(20.0, 3000.0)]
```



In [10…
```
sns.countplot(x='Score',data=y)
```

Out[10]:
```
<AxesSubplot:xlabel='Score', ylabel='count'>
```



In [11…
```
s=y.groupby('Score').mean()
s
```

Out[11]:

Score	Id	HelpfulnessNumerator	HelpfulnessDenominator	Time	text length
1	282280.390258	2.735211	4.869825	1.303159e+09	478.082230
2	280778.019853	1.859014	3.102724	1.301131e+09	490.655010
3	279645.617495	1.700962	2.466393	1.300126e+09	520.319817

4	281713.258608	1.390292	1.666084	1.296722e+09	496.603261
5	285887.043393	1.675228	1.874108	1.294306e+09	402.447368

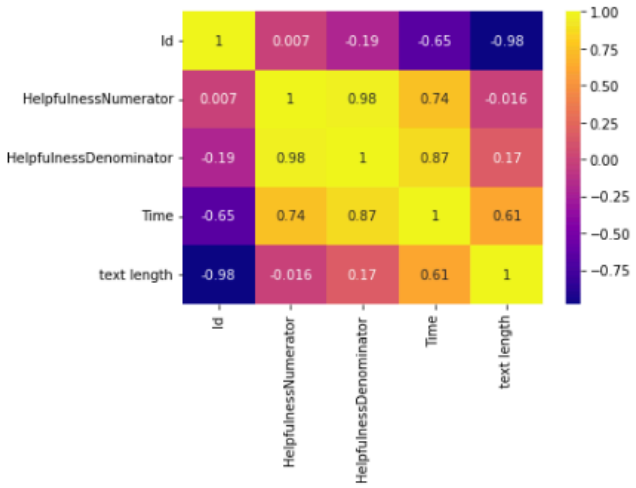
In [12]:
```
s.corr()
```

Out[12]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Time	text length
Id	1.000000	0.007005	-0.187289	-0.648444	-0.980093
HelpfulnessNumerator	0.007005	1.000000	0.975554	0.743499	-0.015590
HelpfulnessDenominator	-0.187289	0.975554	1.000000	0.865762	0.168137
Time	-0.648444	0.743499	0.865762	1.000000	0.610611
text length	-0.980093	-0.015590	0.168137	0.610611	1.000000

In [13]:
```
sns.heatmap(s.corr(),cmap='plasma',annot=True)
```

Out[13]:
```
<AxesSubplot:>
```



In [14]:
```
aclass=y[(y.Score==1) | (y.Score==5)]
```

In [15]:
```
X=aclass['Summary']
y=aclass['Score']
```

In [16]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer()
```

In [17]:
```
X=cv.fit_transform(X)
```

In [18]:
```
from sklearn.model_selection import train_test_split
```

In [19]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3,random_state=101)
```

In [20]:
```
from sklearn.naive_bayes import MultinomialNB
nb=MultinomialNB()
```

In [21]:
```
nb.fit(X_train,y_train)
```

Out[21]:
```
MultinomialNB()
```

In [22]:
```
pred=nb.predict(X_test)
```

In [23]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [24]:
```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[ 11209   4644]
 [  3135 105629]]
              precision    recall  f1-score   support
```

```
In [24…
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[ 11209   4644]
 [  3135 105629]]
              precision    recall  f1-score   support

           1       0.78      0.71      0.74     15853
           5       0.96      0.97      0.96    108764

    accuracy                           0.94    124617
   macro avg       0.87      0.84      0.85    124617
weighted avg       0.94      0.94      0.94    124617
```

```
In [25…
from sklearn.feature_extraction.text import TfidfTransformer
```

```
In [26…
from sklearn.pipeline import Pipeline
```

```
In [27…
pipe=Pipeline([
    ('bow',CountVectorizer()),
    ('tfidf',TfidfTransformer()),
    ('nb',MultinomialNB())
])
```

```
In [29…
X = aclass['Summary']
y = aclass['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_state=101)
```

```
In [30…
pipe.fit(X_train,y_train)
```

```
Out[30]:
Pipeline(steps=[('bow', CountVectorizer()), ('tfidf', TfidfTransformer()),
                ('nb', MultinomialNB())])
```

```
In [31…
prediction=pipe.predict(X_test)
```

```
In [32…
print(confusion_matrix(y_test,prediction))
print(classification_report(y_test,prediction))
```

```
[[  8176   7677]
 [   711 108053]]
              precision    recall  f1-score   support

           1       0.92      0.52      0.66     15853
           5       0.93      0.99      0.96    108764

    accuracy                           0.93    124617
   macro avg       0.93      0.75      0.81    124617
weighted avg       0.93      0.93      0.92    124617
```

```
In [ ]:
```

Future Scope and Conclusion:

To expand this project further, we plan to deploy it applications and websites that require similar jobs and easier visual data analysis. It can be used as an embedded feature in existing major applications like shopping apps and websites , or analytical websites etc. It can also have graphical representation for better understanding the data distribution among the given premises. We can also include the sentiment analysis and recommender system feature to direct customers to their targeted product.

This product is very useful for a quick analysis of data and adding more features might help connecting right audience to right service.