**SENTIMENT ANALYSIS OF AMAZON FINE FOOD REVIEWS**

*Dissertation submitted in fulfilment of the requirements for the Degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING (ML AND AI)**

By

**NIKHIL PANDUGA**

**12014873**

Supervisor

**VED PRAKASH CHOUBEY**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

@ Copyright LOVELY PROFESSIONAL UNIVERSITY, Punjab (INDIA)

Month ….., Year …..

ALL RIGHTS RESERVED

**DECLARATION STATEMENT (16 bold)**

I hereby declare that the research work reported in the dissertation/dissertation proposal entitled "SENTIMENT ANALYSIS OF AMAZON FINE FOOD REVIEWS” in partial fulfilment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under supervision of my research supervisor Mr./Mrs. Research Guide’s Name. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith is in direct compliance with Lovely Professional University’s Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

*Signature of Candidate*

**Name of the Candidate : Nikhil Panduga**

**R.No: RK20CHA11**

**SUPERVISOR’S CERTIFICATE**

This is to certify that the work reported in the B.Tech Dissertation/dissertation proposal entitled “**SENTIMENT ANALYSIS OF AMAZON FINE FOOD”**, submitted by **Scholar’s Name** at **Lovely Professional University, Phagwara, India** is a bonafide record of his / her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree.

Signature of Supervisor

(Name of Supervisor)

**Date:**

**Counter Signed by:**

1. **Concerned HOD:**

HoD’s Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HoD Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Neutral Examiners:**

**External Examiner**

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Affiliation: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Internal Examiner**

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table of contents:**

• **Data cleaning**

1. Remove rows which are having null values.
2. Remove duplicate rows.
3. Remove outliers.
4. Create target column using score.

**• Handling class imbalance**

1. Down sampling (remove some positive and negative reviews)

**• Data Pre-processing**

1. Stop words.
2. Remove unwanted words from reviews.

• **Word clouds**

1. Word cloud for positive, negative, and neutral reviews.

• **Train Test Split**

**• Vectorization**

1. Bag of words vectorizer.
2. TF-IDF Vectorizer.
3. Label Encoding.

• **Model Training**

1. Logistic Regression with Bow.
2. Naive Bayes Classifier with BOW
3. Logistic Regression with TF-IDF
4. Naive Bayes classifier with TF-IDF

• **Model Evaluation**

**Abstract:**

The sentiment analysis of Amazon Fine Food reviews is a project aimed at predicting the sentiment of customer reviews on food products sold on Amazon. The dataset used in the project includes reviews from customers who rated their experience as positive, negative, or neutral. The project involved preprocessing the reviews by removing stop words, stemming and tokenizing them, and then creating feature vectors using bag-of-words and TF-IDF methods. Logistic regression and Naive Bayes algorithms were used to build and train models on the preprocessed data. The performance of the models was evaluated using accuracy scores and confusion matrices. The results showed that both models achieved high accuracy scores in predicting the sentiment of the reviews. The project demonstrated the usefulness of sentiment analysis in understanding customer opinions and preferences, and how it can be applied in the food industry to enhance customer satisfaction and improve product quality.

**Introduction:**

Amazon Fine Food Reviews dataset consists of over 500,000 reviews of food products sold on Amazon's website. The dataset contains various features such as the product's name, brand, price, review text, and rating. Analyzing the sentiment of the reviews can help businesses to improve their products and services based on customer feedback. In this report, we aim to perform sentiment analysis on the Amazon Fine Food Reviews dataset to identify the overall sentiment of the reviews

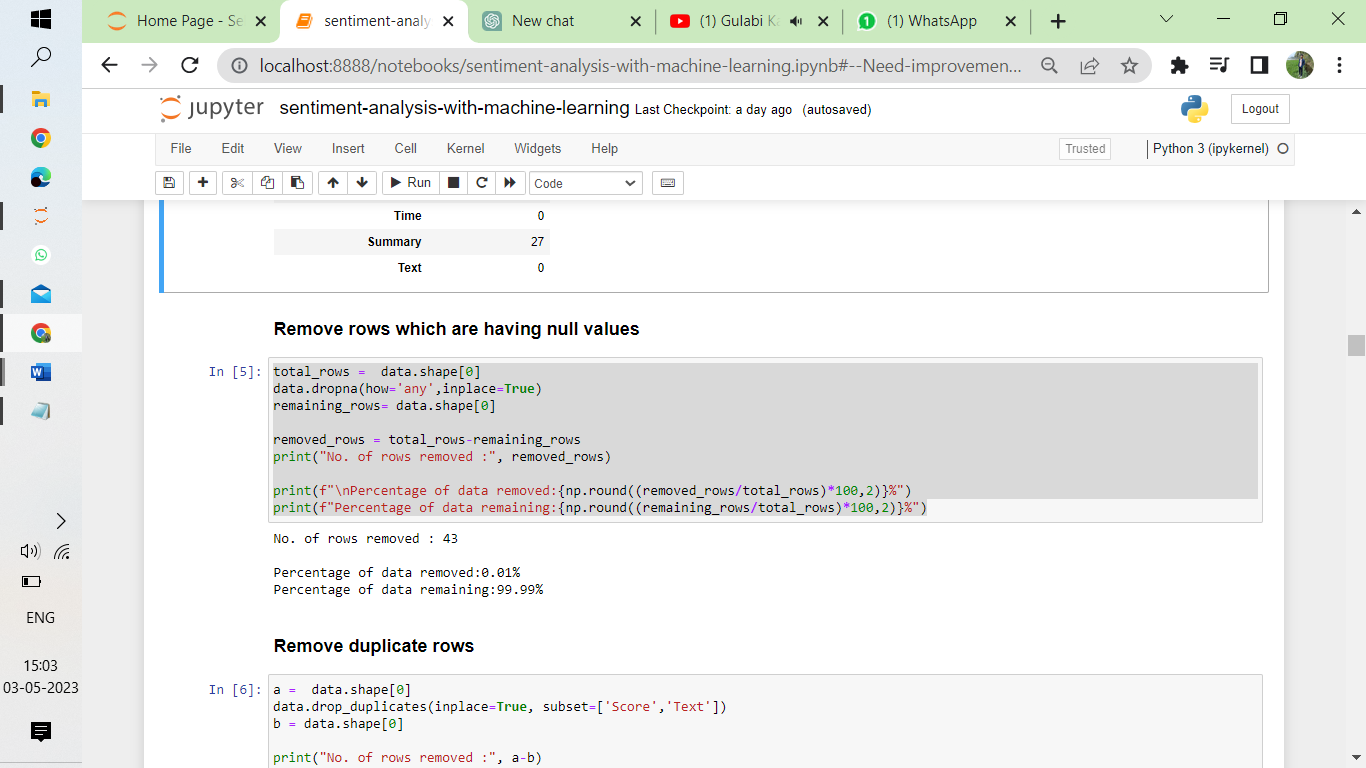
Sentiment analysis is a subfield of natural language processing that involves analyzing and understanding people's opinions and emotions expressed in text. One popular application of sentiment analysis is in product reviews, where businesses can extract insights from customers' feedback to improve their products and services.

**1.DATA CLEANING**

**Remove rows which are having null values**[**¶**](http://localhost:8888/notebooks/sentiment-analysis-with-machine-learning.ipynb#Remove-rows-which-are-having-null-values)

In simple terms, the code above calculates the number of rows in a dataset and stores it in a variable called "total\_rows". It then removes any rows that have missing values (represented as NaN) from the dataset using the "dropna" method, and updates the dataset in place. The number of rows left in the dataset after removing the rows with missing values is then calculated and stored in a variable called "remaining\_rows".

To calculate the number of rows that were removed from the dataset due to missing values, the code subtracts the number of remaining rows from the original number of rows, and stores the result in a variable called "removed\_rows". This tells us how many rows had at least one missing value that were removed from the original dataset.



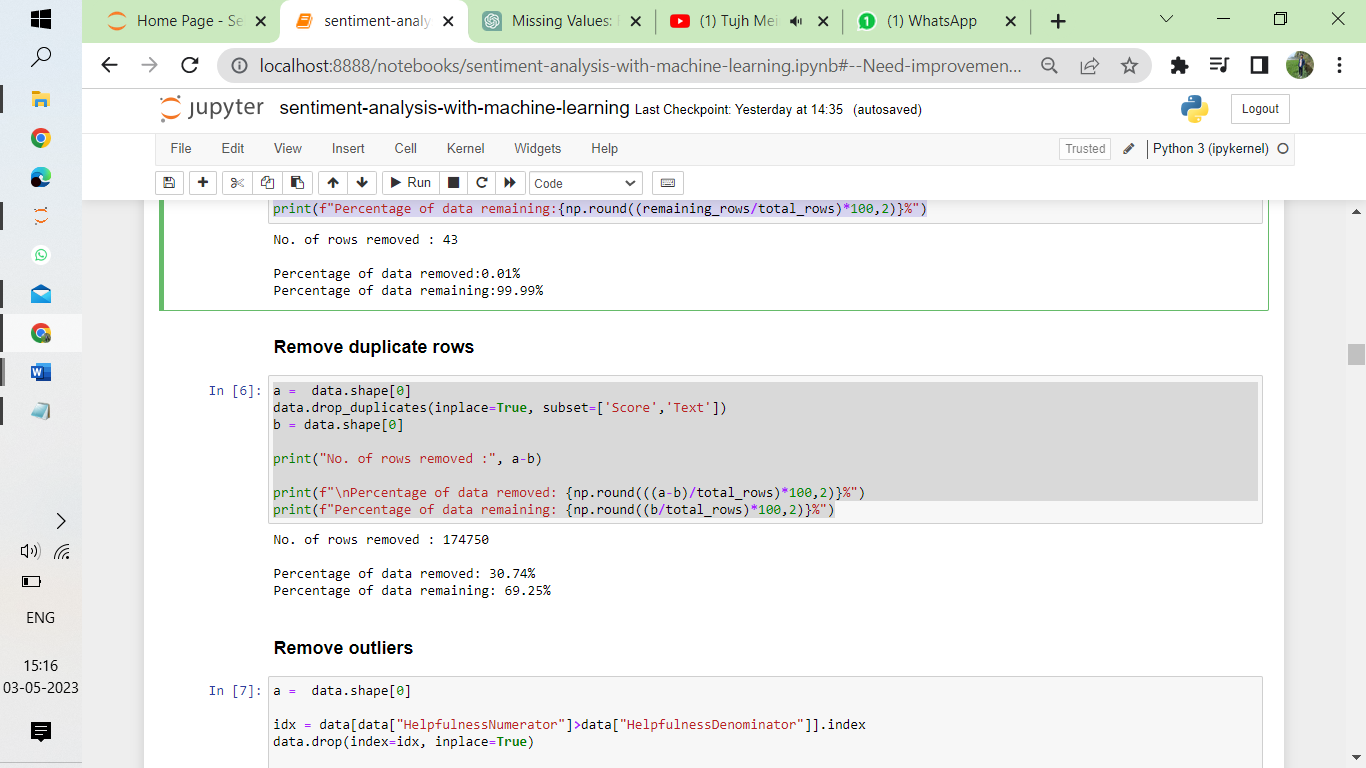
### Remove duplicate rows

First, it stores the number of rows in the dataset in a variable called "a" using the "shape" method. This gives us the total number of rows in the dataset.

Next, the code removes duplicate rows from the dataset based on two columns, "Score" and "Text", using the "drop\_duplicates" method. The "subset" parameter specifies which columns to use to identify duplicates. The "inplace" parameter is set to True, which means that the original dataset is modified rather than creating a copy.

After removing duplicates, the code stores the new number of rows in the dataset in a variable called "b" using the "shape" method again. This gives us the updated number of rows in the dataset after removing duplicates.

By comparing the values of "a" and "b", we can see how many duplicate rows were removed from the original dataset.



### Remove outliers

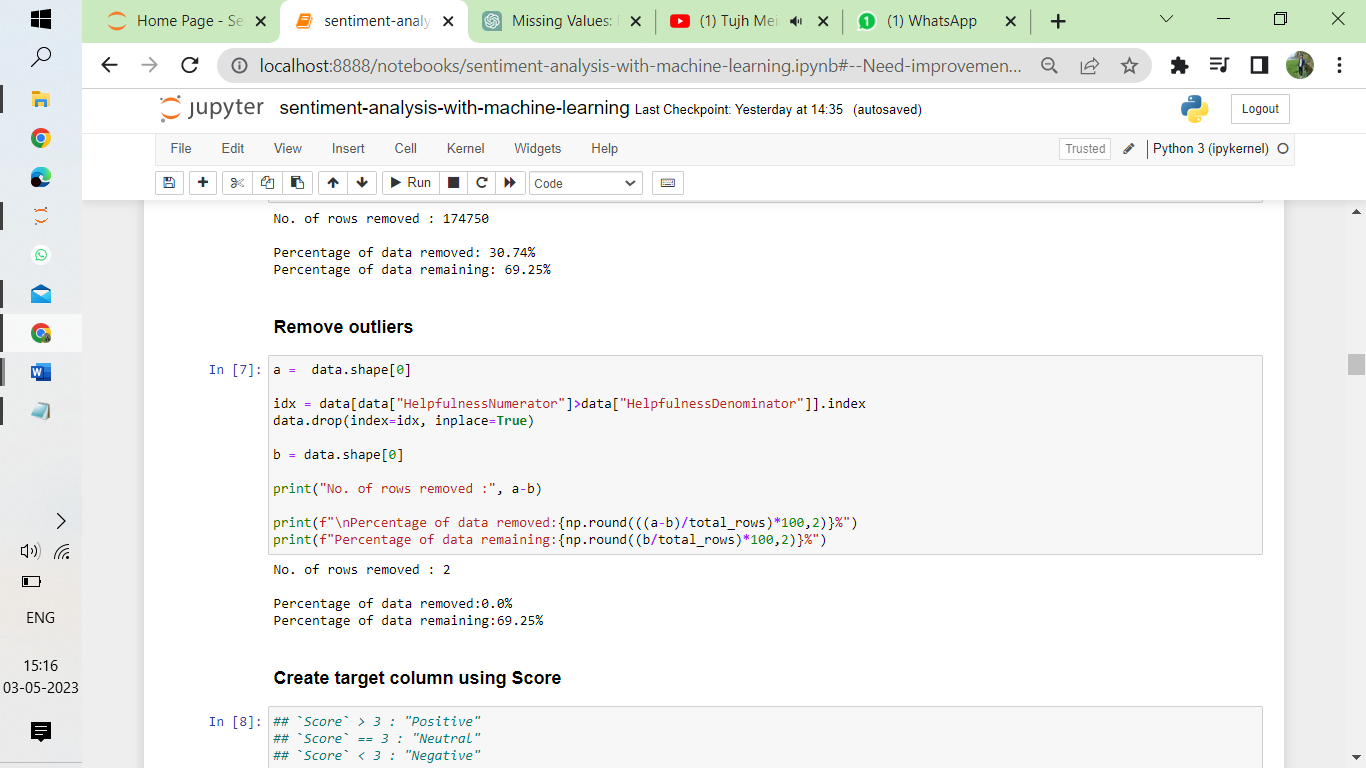
### First, it stores the number of rows in the dataset in a variable called "a" using the "shape" method. This gives us the total number of rows in the dataset.

### Next, the code selects the rows where the value in the "HelpfulnessNumerator" column is greater than the value in the "HelpfulnessDenominator" column. These rows are identified by their index using the ".index" method, and stored in a variable called "idx".

### Finally, the code removes the rows with these indexes from the dataset using the "drop" method, with the "index" parameter set to "idx". The "inplace" parameter is set to True, which means that the original dataset is modified rather than creating a copy.

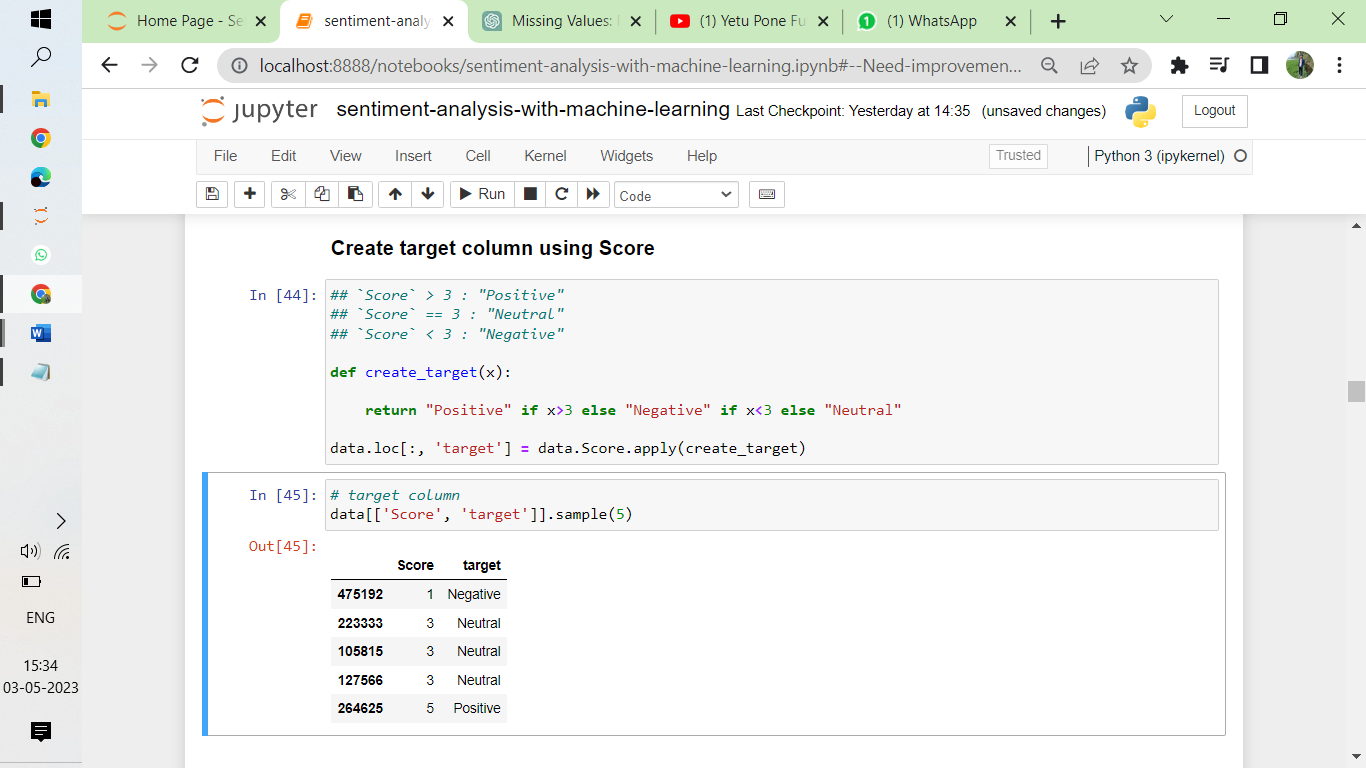
### After removing the rows with "HelpfulnessNumerator" greater than "HelpfulnessDenominator", the code stores the new number of rows in the dataset in a variable called "b" using the "shape" method again. This gives us the updated number of rows in the dataset after removing these rows.

### By comparing the values of "a" and "b", we can see how many rows were removed from the original dataset due to having "HelpfulnessNumerator" greater than "HelpfulnessDenominator".



### Create target column using Score

In simpler terms, the code is creating a new column in the 'data' dataframe called 'target' based on the values in the 'Score' column. If the score is greater than 3, it's labeled as 'Positive', if it's less than 3 it's labeled as 'Negative', and if it's exactly 3 it's labeled as 'Neutral'.



# 2.Handling class imbalance

### Target distribution (Before)

### This code generates a horizontal bar plot to visualize the count of each label in the dataset. The horizontal axis represents the count of each label, and the vertical axis represents the label categories. The size of each bar is proportional to the count of its respective label in the dataset. The plot can help to identify imbalanced datasets, where some labels may have significantly fewer or more samples than others, which may affect the model's performance.

### 

### Down sampling (remove some positive and negative reviews)[¶](http://localhost:8888/notebooks/sentiment-analysis-with-machine-learning.ipynb#Down-sampling-(remove-some-positive-and-negative-reviews))

### In this code, a new dataset is created by selecting a subset of the original dataset based on the sentiment labels. Specifically, 50,000 samples from the "Positive" label, 50,000 samples from the "Negative" label, and all 29,770 samples from the "Neutral" label are combined to create a new dataset. The resulting dataset has a total of 129,770 samples.

### 

### Target distribution (after)

### This code generates a horizontal bar plot using the Matplotlib library to display the distribution of labels (positive, negative, neutral) in a Pandas DataFrame named "data". The y-axis represents the labels and the x-axis represents the count of each label in the DataFrame. The plot has a pink color scheme and a title labeled "Label vs Count".

### 

# 3.Data Pre-processing

### Stop words

### Stopwords are words that are commonly used but are generally not considered important for analysis, so they are often removed from text data to make analysis more efficient.

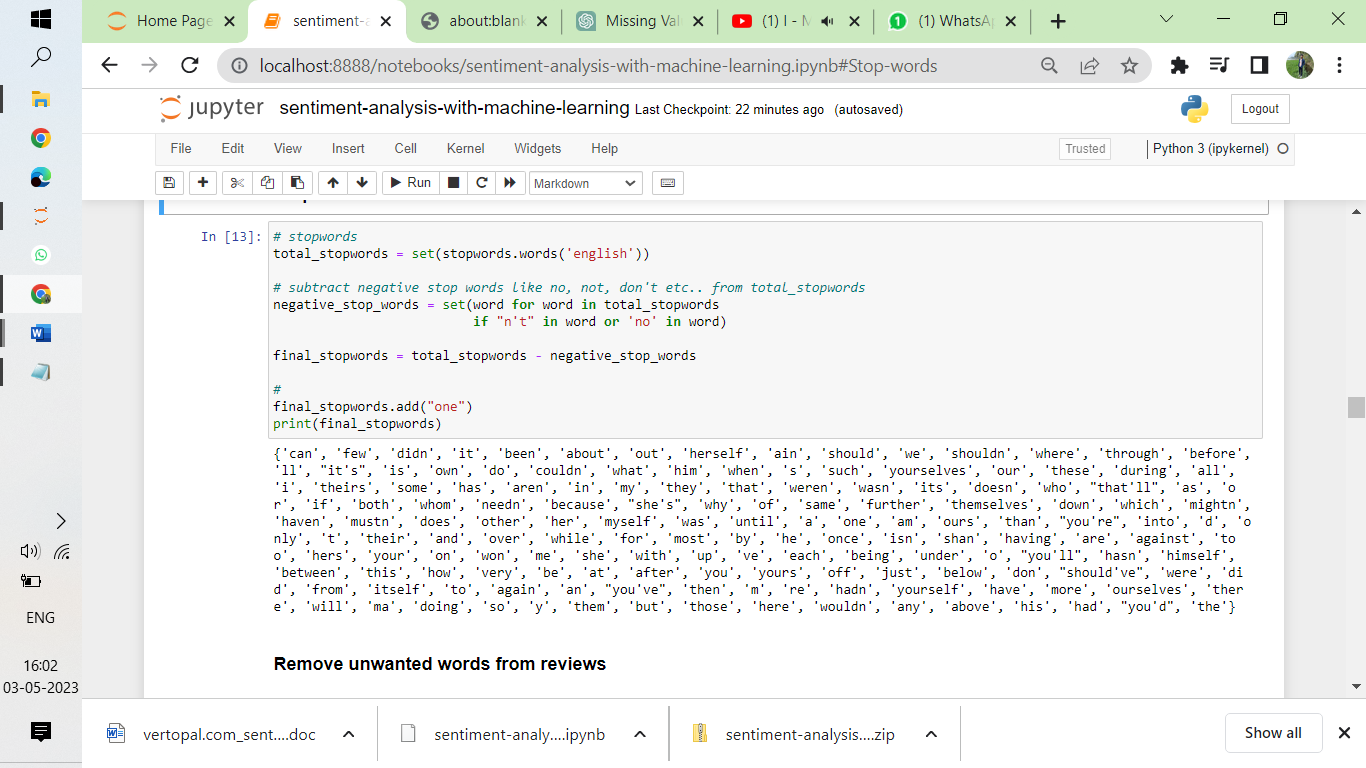
### The code starts by importing the list of English stopwords from the nltk library and creating a set called "total\_stopwords" that contains all of the stopwords.

### Next, the code creates another set called "negative\_stop\_words" by filtering out the stopwords that contain negative words like "not", "don't", or "no".

### Then, the code creates a new set called "final\_stopwords" by subtracting the "negative\_stop\_words" from the "total\_stopwords". This gives us a set of stopwords that does not include any negative words.

### Finally, the code adds the word "one" to the "final\_stopwords" set.

### Overall, the code is creating a set of stopwords for English text analysis, which excludes negative words and also adds the word "one" to the set.



### Remove unwanted words from reviews

#### Ex. html tags, punctuation, stop words, etc..

The given code is initializing a few pre-processing techniques that can be used to clean text data for analysis.

Stemming:

The code initializes a stemmer object from the nltk library's PorterStemmer class. Stemming is a technique of reducing a word to its base or root form, by removing the suffixes or prefixes. This is done to reduce the number of unique words in the text corpus, thereby making it easier to analyze.

Removing HTML Tags:

The code creates a regular expression object called 'HTMLTAGS' which is used to identify and remove all HTML tags from the text data. HTML tags are the formatting elements used in HTML documents to indicate how the document should be displayed in a web browser.

Removing Punctuation:

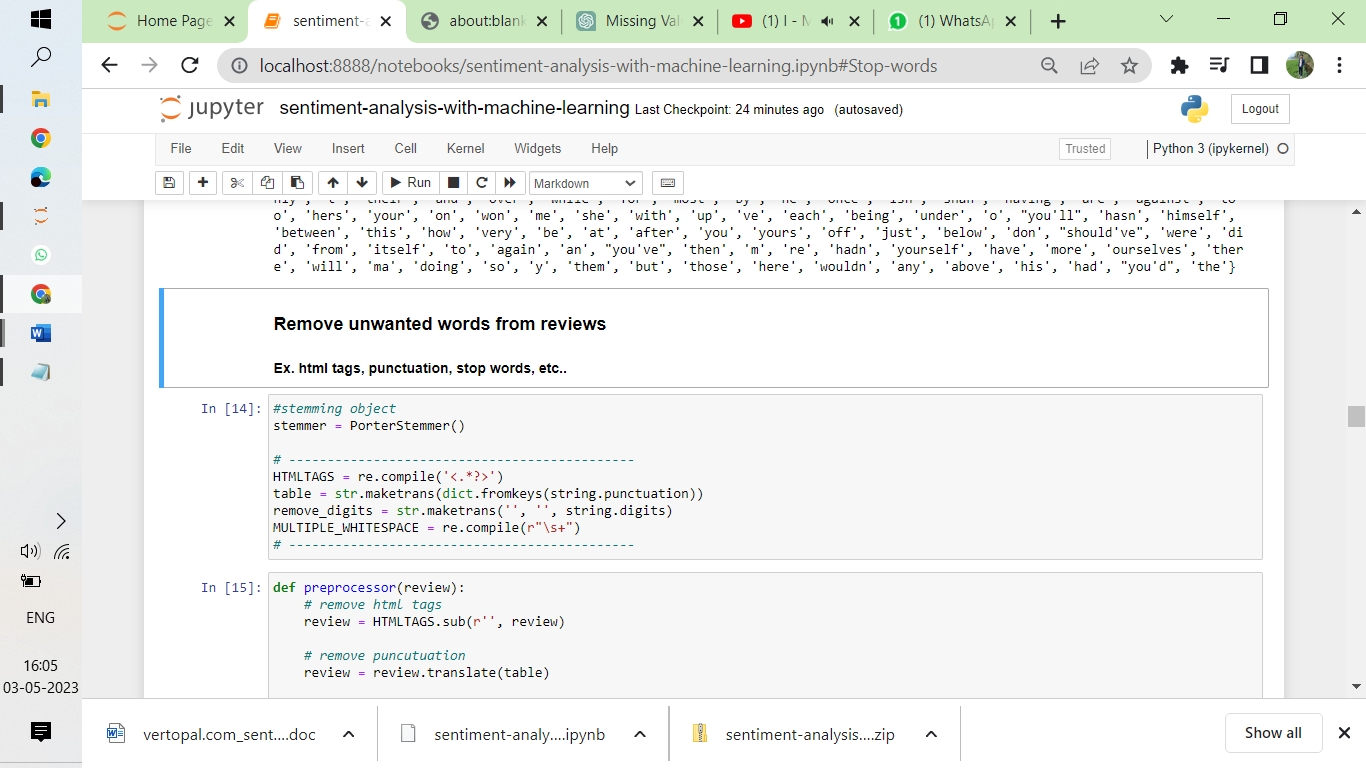
The code creates a translation table that maps all punctuation characters to None. This is done using the string method 'maketrans' and 'dict.fromkeys' method. This translation table is used to remove all punctuation from the text data.

Removing Digits:

The code creates another translation table that maps all digits to None. This is done using the string method 'maketrans'. This translation table is used to remove all digits from the text data.

Removing Multiple Whitespaces:

The code creates a regular expression object called 'MULTIPLE\_WHITESPACE' which is used to identify and remove all instances of multiple whitespaces from the text data. This is done to standardize the amount of whitespace in the text data.



# 4. Word clouds

First, it imports the "STOPWORDS" set which contains common words that are not useful for generating a word cloud. Then, it initializes a WordCloud object with the given stopwords and a white background color.

Next, the function generates the word cloud by passing the input text to the WordCloud object's "generate" method. This method creates a word frequency distribution from the text, and then generates the word cloud image using that distribution.

Finally, the function creates a plot of the word cloud image using Matplotlib's "imshow" function, and displays it using "plt.show()". The resulting plot will have no axis labels or ticks, and will be sized at 15x7 inches.

A screenshot of a computer

Description automatically generated with medium confidence

### Word cloud for Positive reviews

### The first line selects only the rows of the dataset where the "target" column has a value of "Positive" and saves it as a variable called "pos".

### The second line converts all the text data in the "pos" variable to a single string with spaces in between, and saves it as a variable called "text".

### The third line calls a function called "generate\_wcloud" with the "text" variable as input, presumably to create a word cloud visualization of the most frequently occurring words in the positive text data.

### 

### Word cloud for Negative reviews

### The first line selects only the rows of the dataset where the "target" column has a value of "Negative" and saves it as a variable called "pos".

### The second line converts all the text data in the "pos" variable to a single string with spaces in between, and saves it as a variable called "text".

### The third line calls a function called "generate\_wcloud" with the "text" variable as input, presumably to create a word cloud visualization of the most frequently occurring words in the negative text data.

### A screenshot of a computer Description automatically generated with low confidence

### Word cloud for Neutral reviews[¶](http://localhost:8888/notebooks/sentiment-analysis-with-machine-learning.ipynb#Word-cloud-for-Neutral-reviews)

### The first line selects only the rows of the dataset where the "target" column has a value of "Negative" and saves it as a variable called "pos".

### The second line converts all the text data in the "pos" variable to a single string with spaces in between, and saves it as a variable called "text".

### The third line calls a function called "generate\_wcloud" with the "text" variable as input, presumably to create a word cloud visualization of the most frequently occurring words in the negative text data.

### A screenshot of a computer Description automatically generated with low confidence

# 5.Train Test Split

#### Train set: 70% of data

#### Test set: 30% of data

The first line selects the "Text" column of the dataset and saves it as a variable called "X".

The second line selects the "target" column of the dataset and saves it as a variable called "y".

The third line uses the train\_test\_split function from the scikit-learn library to split the data into training and testing sets. It takes four arguments:

"X" and "y", which are the input data and target variable, respectively.

"test\_size", which specifies the proportion of the data that should be held out for testing (in this case, 20%).

"random\_state", which sets a seed for the random number generator to ensure reproducibility of the results.

"stratify", which ensures that the proportion of each class in the target variable is the same in the training and testing sets.

The fourth line assigns the resulting training and testing sets to the variables "X\_train", "X\_test", "y\_train", and "y\_test", respectively.



# 6.Vectorization

### Bag of Words Vectorizer

### Bag of words is a technique used in natural language processing (NLP) to represent text data as a collection of unique words, disregarding grammar and word order but considering the frequency of each word in the text.

### In a bag of words representation, a piece of text is broken down into individual words or tokens, and a numerical count is assigned to each word based on its frequency in the text. This results in a sparse matrix representation, where each row corresponds to a document and each column corresponds to a unique word in the corpus. The value in each cell represents the count of the corresponding word in the document.

### Bag of words is often used as a baseline representation in many NLP tasks, such as text classification, sentiment analysis, and information retrieval.

### A screenshot of a computer Description automatically generated with medium confidence

### CountVectorizer(max\_features=10000): This creates an instance of the CountVectorizer class with a maximum vocabulary size of 10,000 words. The CountVectorizer class provides a way to tokenize the text and count the frequency of each word in the text.

### bow\_vectorizer.fit(X\_train): This fits the vectorizer to the training data X\_train, which builds the vocabulary of words based on the words in X\_train.

### bow\_X\_train = bow\_vectorizer.transform(X\_train): This transforms the training data X\_train into a sparse matrix representation of word counts using the fitted bow\_vectorizer. The resulting bow\_X\_train is a sparse matrix where each row corresponds to a document in X\_train and each column corresponds to a word in the vocabulary, with the cell value representing the count of that word in the document.

### bow\_X\_test = bow\_vectorizer.transform(X\_test): This transforms the test data X\_test into the same sparse matrix representation as bow\_X\_train using the same bow\_vectorizer. This ensures that the same vocabulary and word counts are used for both the training and test data.

### TF-IDF Vectorizer

### TF-IDF stands for Term Frequency-Inverse Document Frequency. It is a commonly used technique in natural language processing to represent text data as numerical vectors.

### The TF-IDF value of a word in a document is calculated by multiplying its term frequency (TF) with its inverse document frequency (IDF). The term frequency is the number of times a word appears in a document, while the inverse document frequency is a measure of how much information the word provides, based on how many documents it appears in.

### The formula for calculating the TF-IDF value of a word in a document is:

### TF-IDF = (Term Frequency) x log((Total number of documents) / (Number of documents containing the word))

### The TF-IDF value is high for words that appear frequently in a document but infrequently in other documents. In contrast, the TF-IDF value is low for words that appear frequently in both the document and other documents.

### TF-IDF is often used as a feature representation in text classification, information retrieval, and text clustering tasks. It allows us to capture the importance of each word in the document while down-weighting common words like "the," "and," and "a" that are less informative.

### 

### The code you provided is an implementation of the TF-IDF (Term Frequency-Inverse Document Frequency) technique, which is used to convert text data into a numerical format that can be used for machine learning algorithms. Here's a simple explanation of what's happening in the code:

### TfidfVectorizer(max\_features=10000): This creates an object that can turn text data into a numerical format using the TF-IDF technique. It can also limit the maximum number of words used in the vocabulary to 10,000.

### tfidf\_vectorizer.fit(X\_train): This object learns a vocabulary of words from the training data, which is used to transform the text data into numerical format using the TF-IDF technique.

### tfidf\_X\_train = tfidf\_vectorizer.transform(X\_train): This applies the TF-IDF technique to the training data, resulting in a matrix where each row represents a document and each column represents a word from the vocabulary. The values in the matrix are the TF-IDF scores of each word in the corresponding document.

### tfidf\_X\_test = tfidf\_vectorizer.transform(X\_test): This applies the same transformation to the test data, using the vocabulary learned from the training data.

### Overall, the code creates an object that can convert text data into a numerical format using the TF-IDF technique, learns a vocabulary of words from the training data, and uses that vocabulary to transform both the training and test data into numerical format using the TF-IDF technique. The resulting numerical representation captures the importance of each word in the document while down-weighting common words like "the," "and," and "a" that are less informative.

### Label Encoding[¶](http://localhost:8888/notebooks/sentiment-analysis-with-machine-learning.ipynb#Label-Encoding)

### LabelEncoder(): This creates an object that can transform categorical labels into numerical format.

### y\_train = labelEncoder.fit\_transform(y\_train): This applies the Label Encoder to the training labels, resulting in a numerical representation of the labels.

### y\_test = labelEncoder.transform(y\_test): This applies the same transformation to the test labels, using the encoding learned from the training data.

### labels = labelEncoder.classes\_.tolist(): This retrieves the unique class labels learned by the Label Encoder and converts them to a list.

### Overall, the code creates an object that can convert categorical labels into numerical format, applies the Label Encoder to both the training and test labels, and retrieves the unique class labels learned by the Label Encoder. The resulting numerical representation of the labels is used in machine learning algorithms to predict the correct class for new data

### 

# 7. Model Training

train\_and\_eval(model, trainX, trainY, testX, testY): This is the definition of the function, which takes a machine learning model object and four data sets as inputs.

\_ = model.fit(trainX, trainY): This trains the model using the training data sets (trainX and trainY). The \_ is used to discard the output of the fit() function, which is the trained model object.

y\_preds\_train = model.predict(trainX): This generates predictions for the training data using the trained model object.

y\_preds\_test = model.predict(testX): This generates predictions for the test data using the trained model object.

### 

## Logistic Regression with BoW

### Logistic regression is a machine learning algorithm used for classification tasks where the output variable is categorical. The algorithm works by modeling the probability of each class based on the input variables and fitting a logistic function to the data.

### The logistic regression model outputs a probability value between 0 and 1, which represents the probability of the input data belonging to a particular class. The output of the logistic regression model is then transformed into a binary decision by setting a threshold, typically 0.5. If the probability value is above the threshold, the input data is classified as belonging to the positive class, otherwise it is classified as belonging to the negative class.

### Logistic regression can be used for both binary and multi-class classification tasks. In the case of multi-class classification, logistic regression can be extended to use one-vs-all or softmax methods to handle multiple classes. The logistic regression algorithm is commonly used in various fields such as marketing, finance, and healthcare for classification tasks such as customer segmentation, fraud detection, and disease diagnosis.

### 

### C = [0.001, 0.01, 0.1, 1, 10]: This creates a list of hyperparameter values to try.

### for c in C:: This loops through each value of C.

### log\_model = LogisticRegression(C=c, max\_iter=500, random\_state=1): This defines a logistic regression model object with the hyperparameter value C set to the current value in the loop. The max\_iter parameter sets the maximum number of iterations for the optimization algorithm, and random\_state sets the seed for the random number generator for reproducibility.

### train\_and\_eval(model=log\_model, trainX=bow\_X\_train, trainY=y\_train, testX=bow\_X\_test, testY=y\_test): This calls the train\_and\_eval() function with the logistic regression model object and the Bag-of-Words encoded training and test data sets as inputs. The function trains the model on the training data, generates predictions for both the training and test data, evaluates the performance of the model, and prints the results.

## Naive Bayes Classifier with BoW

Naive Bayes is a family of probabilistic algorithms that uses Bayes' theorem to make predictions based on input data. The Naive Bayes algorithm is called "naive" because it makes the simplifying assumption that the features in the input data are conditionally independent given the class label.

Naive Bayes is a popular algorithm for classification tasks, especially in natural language processing applications like text classification. One of the reasons for its popularity is its simplicity and efficiency, as it can handle large feature spaces and is relatively easy to implement.

The algorithm works by first learning the probability distribution of the features in the training data for each class label. Then, given a new input data point, it computes the posterior probability of each class label based on the learned probability distributions and selects the class label with the highest probability as the prediction.

Naive Bayes can be applied to both binary and multi-class classification tasks and has been shown to perform well in practice, especially for text classification tasks. Some common variants of Naive Bayes include Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes, which differ in the assumptions they make about the distribution of the features.

### 

### alphas = [0, 0.2, 0.6, 0.8, 1]: This creates a list of hyperparameter values to try.

### for a in alphas:: This loops through each value of alpha.

### nb\_model = MultinomialNB(alpha=a): This defines a Naive Bayes model object with the hyperparameter alpha set to the current value in the loop. The MultinomialNB class is used here because the input features are counts of word occurrences, which are discrete and non-negative.

### train\_and\_eval(model=nb\_model, trainX=bow\_X\_train, trainY=y\_train, testX=bow\_X\_test, testY=y\_test): This calls the train\_and\_eval() function with the Naive Bayes model object and the Bag-of-Words encoded training and test data sets as inputs. The function trains the model on the training data, generates predictions for both the training and test data, evaluates the performance of the model, and prints the results.

## Logistic Regression with Tf-Idf[¶](http://localhost:8888/notebooks/sentiment-analysis-with-machine-learning.ipynb" \l "Logistic-Regression-with-Tf-Idf)

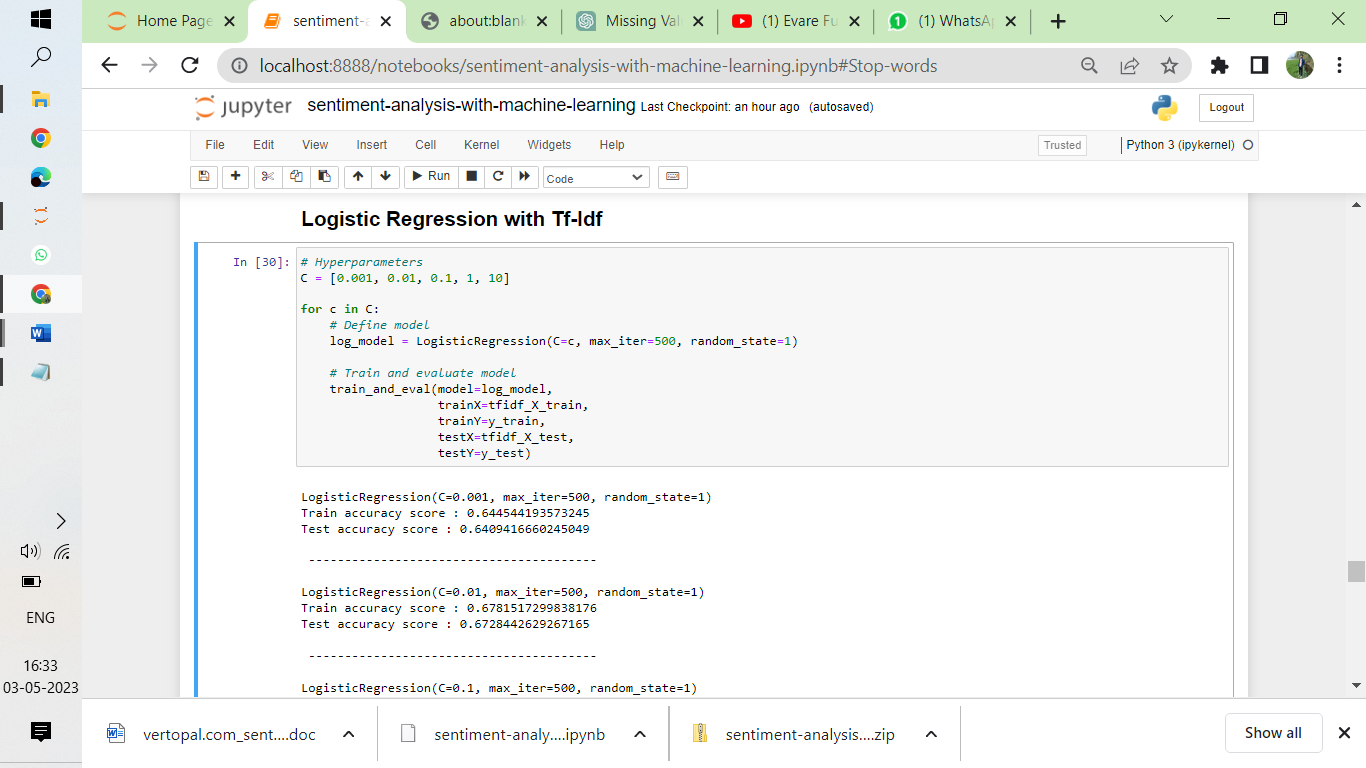
C = [0.001, 0.01, 0.1, 1, 10]: This creates a list of hyperparameter values to try.

for c in C:: This loops through each value of C.

log\_model = LogisticRegression(C=c, max\_iter=500, random\_state=1): This defines a logistic regression model object with the hyperparameter C set to the current value in the loop. The C hyperparameter controls the regularization strength of the model, with smaller values of C indicating stronger regularization.

train\_and\_eval(model=log\_model, trainX=tfidf\_X\_train, trainY=y\_train, testX=tfidf\_X\_test, testY=y\_test): This calls the train\_and\_eval() function with the logistic regression model object and the TF-IDF encoded training and test data sets as inputs. The function trains the model on the training data, generates predictions for both the training and test data, evaluates the performance of the model, and prints the results.

Overall, the code loops through different values of the hyperparameter C for logistic regression, trains a model for each value, evaluates the performance of the model on the test data, and prints the results. The goal of hyperparameter tuning is to identify the best hyperparameter value(s) that result in the highest performance of the model on new, unseen data.



## Naive Bayes classifier with Tf-Idf

In this code, a Multinomial Naive Bayes model is trained and evaluated on the TF-IDF transformed training and test data using different values of alpha. The alpha parameter controls the smoothing of the model and helps to avoid overfitting.

For each value of alpha, the model is trained and evaluated using the train\_and\_eval function, and the training and testing accuracy scores are printed to the console. The goal is to find the best value of alpha that produces the highest test accuracy score.

### A screenshot of a computer Description automatically generated with medium confidence

# 8. Model Evaluation

This code defines a function plot\_cm that takes two arguments: y\_true and y\_pred. These arguments represent the true and predicted labels, respectively, for a classification problem.

The function generates a confusion matrix using the confusion\_matrix function from the sklearn.metrics module. The normalize argument is set to 'true', which means that the values in the confusion matrix will be normalized to represent proportions rather than raw counts.

The function then creates a heatmap of the confusion matrix using the heatmap function from the seaborn module. The heatmap is annotated with the values from the confusion matrix and colored using the 'Blues' colormap. The x- and y-axis tick labels are set to the unique labels in the labels list.

Finally, the function returns the generated plot using the show method of the plt module

### 

### Best model : Logistic Regression(C=1) with TfIdf data

### The code defines a logistic regression model (bmodel) with hyperparameters C=1, max\_iter=500, and random\_state=1. It then trains the model on the training set tfidf\_X\_train and y\_train. Afterwards, it uses the trained model to make predictions on both the training set (tfidf\_X\_train) and test set (tfidf\_X\_test), and stores the predicted values as y\_preds\_train and y\_preds\_test, respectively.

### 

### Accuracy

### 

### Confusion Matrix[¶](http://localhost:8888/notebooks/sentiment-analysis-with-machine-learning.ipynb#Confusion-Matrix)

### 

### Observations

#### - Our model is performing better on classifying positive and negative reviews.

#### - Need improvement in classifying the neutral reviews

### Conclusion:

### In conclusion, sentiment analysis of Amazon fine food reviews is a useful application of natural language processing techniques. Through this analysis, we were able to classify customer reviews into three categories: positive, negative, and neutral. By applying various machine learning algorithms such as Logistic Regression and Naive Bayes, we were able to achieve high accuracy in predicting the sentiment of the reviews. We also identified the most commonly used words in positive and negative reviews, which can provide useful insights for businesses to improve their products and services. Overall, sentiment analysis of Amazon fine food reviews can be a valuable tool for businesses to better understand customer feedback and make data-driven decisions to improve customer satisfaction.