GOVERNMENT COLLEGE OF ENGINEERING, SALEM-11

(An Autonomous Institution Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IBM- NAALAYATHIRAN

(EXPERIENCE BASED PRACTICAL LEARNING)

PROJECT REPORT

THIRD YEAR – V SEMESTER

NAME	**SIVARANJANI CK**
REGISTER NUMBER	**61772121042**
NM_ID	**au61772121042**
DOMAIN	**ARTIFICIAL INTELLIGENCE**
PROJECT TITLE	**SENTIMENT ANALYSIS FOR MARKETING**

PROJECT TITTLE: SENTIMENT ANALYSIS FOR MARKETING

Problem Statement:

This type of project can show you what it's like to work as an NLP specialist. For this project, you want to find out how customers evaluate competitor products, what they like and dislike. It's a great business case. Learning what customers like about competing products can be a great way to improve your own product, so this is something that many companies are actively trying to do. Employ different NLP methods to get a deeper understanding of customer feedback and opinion.

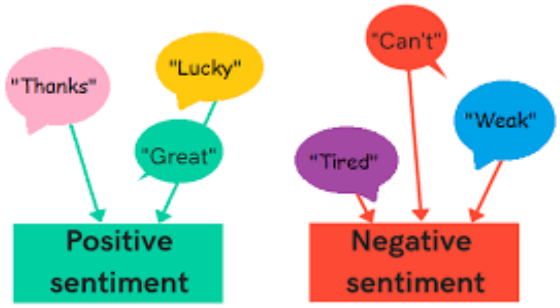
Introduction:

- Emotions are essential, not only in personal life but in business as well. The problem is to perform sentiment analysis, it is a marketing tool that helps to examine the way how the customers and target audience interact and feel about the product or brand, that provides the necessary results to evaluate and improve their product, business, marketing, and

communication strategy and also compare with the competitors to enhance performance.

- It is used to analyse and understanding the raw text (emotions, opinions, attitudes expressed by customer) to extract the necessary results using natural language processing (NLP), machine learning, and other data analytics techniques.

- Its primary goal is to measures the number of online interactions that the customers have with a brand or product, like comments and shares and posts. Using this, we can label individual interactions as positive, negative or neutral i.e., strength and weakness of the product.



- Sentiment analysis is the automated process of tagging data according to their sentiment, such as positive, negative and neutral. Sentiment analysis allows companies to analyse data at scale, detect insights and automate processes.

- Based on these labels, the sentiment analysis algorithm suggests the company to improve their product according to their customer's satisfaction.

Design Thinking:

1. Data Collection:

i. Identify and gather datasets containing customer-generated data from various sources like social media, customer reviews, surveys, and online forums, or any other relevant text data.

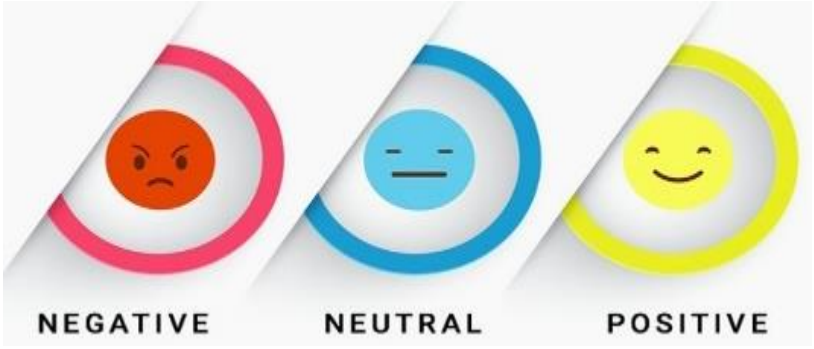
ii. Dataset Link: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment.

2. Data Preprocessing:

i. Clean and preprocess the text data by removing special characters, duplicates and stop words by using Python libraries such as NLTK, etc.
ii. Tokenize the text into individual words or phrases for analysis.

3. Labelling Data:

i. Label data with sentiment labels (**positive, negative, neutral**) to create a labelled dataset for training and testing.



4. Feature Extraction:

i. Convert the text data into numerical features by using the techniques **TF-IDF** (Term Frequency-Inverse Document Frequency) or pre-trained word embeddings (e.g., Word2Vec, etc.).

5. Model Selection:

a. Choosing a sentiment analysis model:
   i. **Random Forest** classifier
   ii. Support Vector Machines (**SVM**)
   iii. Long Short-Term Memory networks (**LSTM**) for analysis.

6. Training, validation and Deployment:

i. Train the chosen model on the labelled dataset.
ii. Split the dataset into training, validation, and test sets to evaluate the model's performance.
iii. And deploying the trained model by using the web development frameworks (e.g., Flask) to create a web application.

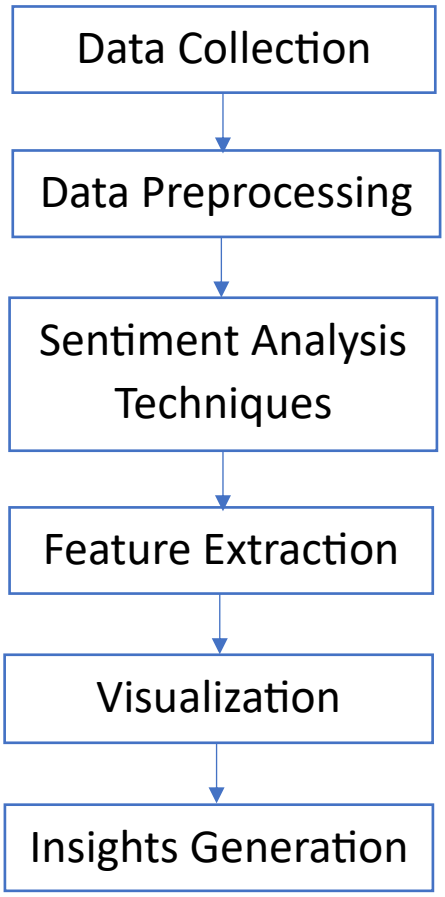
7. Visualization:

i. Create a user-friendly interface to visualize the results using charts or graphs for easy understanding.

8. Actionable Insights:

i. Gather feedback by testing the model with dataset and integrate them to the marketing systems.
ii. Translate the sentiment analysis results into actionable insights for marketing strategies.
iii. Continuously assess the performance and identify the areas for improvement based on customer feedback.

Phases of Development:



Dataset:

Dataset Link: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment.

The "**Twitter US Airline Sentiment**" dataset on Kaggle typically contains Twitter data related to the sentiment of tweets about various U.S. airlines. This dataset is commonly used for sentiment analysis, text classification tasks and natural language processing tasks

The goal is to build machine learning models to classify the sentiment of tweets. Analysts and data scientists can use this data to understand how passengers perceive different airlines on social media and to develop strategies for improving customer satisfaction.

This data originally came from Crowd flower's Data for Everyone library.

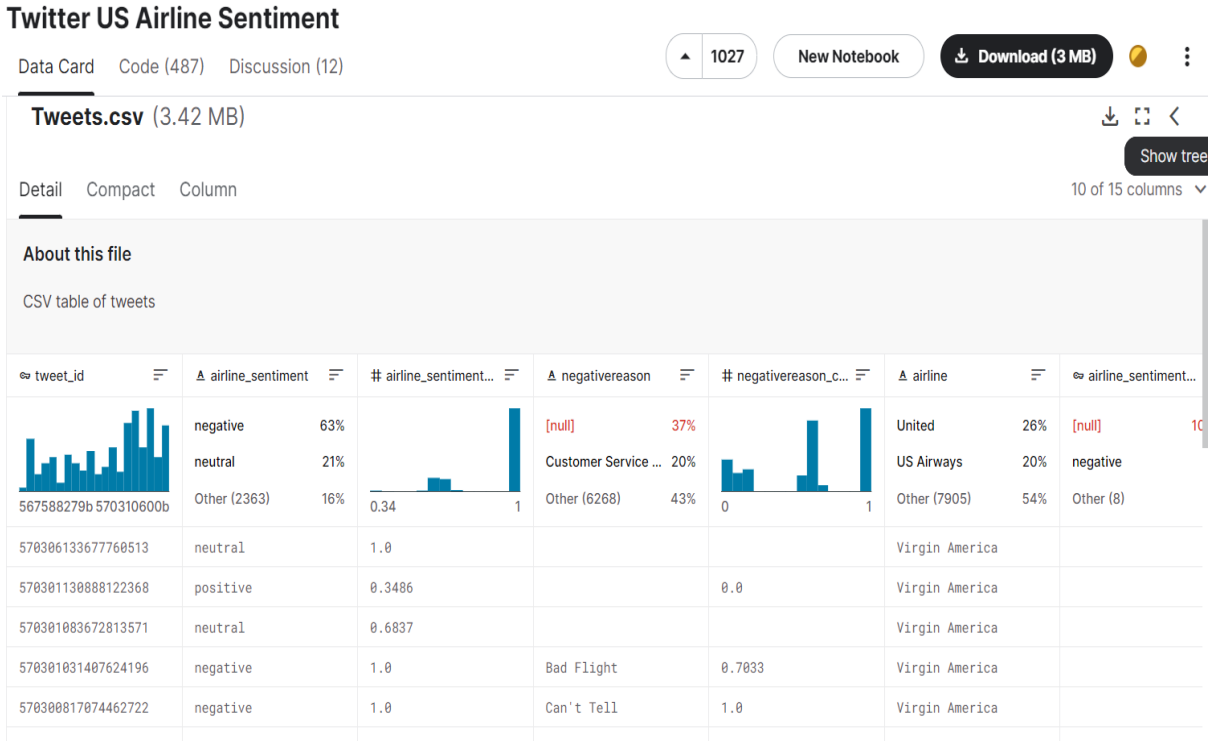
As the original source says,

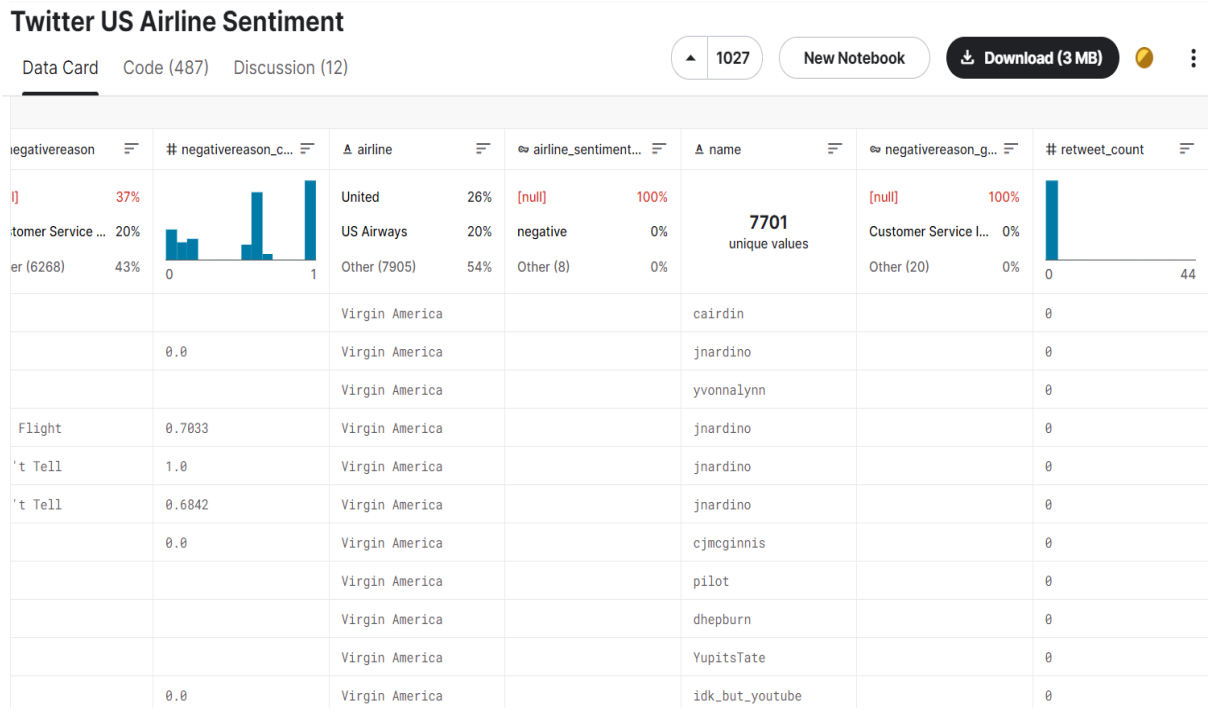
A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

It contains the sentiment of the tweets in this set was positive, neutral, or negative for six US airlines:

- **Tweets:** The dataset will contain a collection of tweets posted by Twitter users. These tweets may cover a range of topics related to different U.S. airlines.
- **Sentiment Labels:** Each tweet is usually labelled with a sentiment category, such as "positive," "neutral," or "negative." These labels indicate the sentiment expressed in the tweet.
- **Airline Information:** Information about the specific U.S. airline mentioned in each tweet. This information could include the airline's name or Twitter handle.

- **Additional Features:** The dataset include other features like the date and time of the tweet, the user's Twitter handle, the number of retweets, the number of likes, and more.
- **Dataset Size:** The dataset contains fifteen thousand of tweets.





Importing Libraries:

Let us start by importing the libraries by installing the necessary libraries with the help of command

! pip install library_name

Importing Necessary Installed Libraries:

```
[28]: #importing installed packages (pip install package_name)
#to run current cell - (ctrl+enter), to run & move to next cell - (shift+enter)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import nltk
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
pd.set_option("display.max_colwidth", 200)
```

Data Importing:

- Twitter have become a part of our everyday schedule. Many NLP techniques can be used on the text data available from Twitter.
- **Dataset Link:** https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment
- I have downloaded the csv file from the above link and saved it as **Tweets.csv**
- to import data in Python, typically used libraries like Pandas, which provide functions for reading various data formats.
- Then I have imported this file by using the **Pandas** and **DataFrame** and its function **pd.read_csv()**
- **df = pd.read_csv('Tweets.csv'):** This line reads the data from a CSV file ('Tweets.csv') and stores it in a pandas DataFrame called 'df.'

Data Import (Data Collection):

```
[29]: # link: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment
      # downloading and importing the dataset (tweets.csv)
      df = pd.read_csv('Tweets.csv')
      df.head(5)
```

[29]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negative
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	

- **df.head(5):** This line displays the first 5 rows of the DataFrame, providing a quick overview of the data.
- **df.tail(3):** This line displays the last 3 rows of the DataFrame, showing the end of the dataset.

```
[4]: df.tail(3)
```

[4]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	neg
14637	569587242672398336	neutral	1.0000	NaN	NaN	American	NaN	sanyabun	
14638	569587188687634433	negative	1.0000	Customer Service Issue	0.6659	American	NaN	SraJackson	
14639	569587140490866689	neutral	0.6771	NaN	0.0000	American	NaN	daviddtwu	

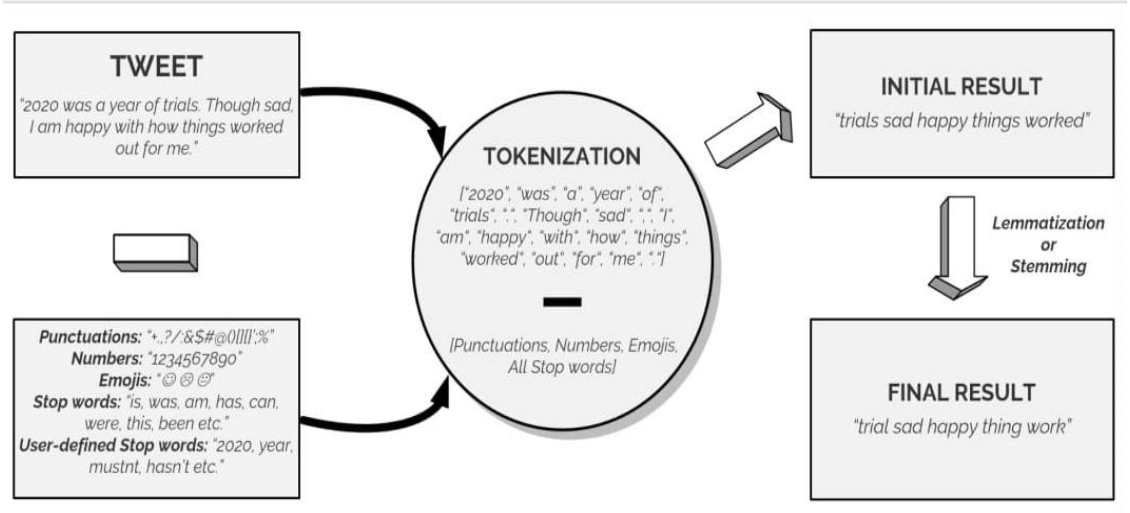
- **df.shape:** This line returns a tuple representing the dimensions of the DataFrame, with the first value indicating the number of rows and the second value indicating the number of columns.

- **df.info():** This line provides essential information about the DataFrame, including data types and memory usage, which can be helpful for data exploration and debugging.

```
[5]: # to find no. of rows and columns in dataset
     df.shape

[5]: (14640, 15)

[6]: #datatype info
     df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   tweet_id                      14640 non-null  int64
 1   airline_sentiment             14640 non-null  object
 2   airline_sentiment_confidence  14640 non-null  float64
 3   negativereason                9178 non-null   object
 4   negativereason_confidence     10522 non-null  float64
 5   airline                       14640 non-null  object
 6   airline_sentiment_gold        40 non-null     object
 7   name                          14640 non-null  object
 8   negativereason_gold           32 non-null     object
 9   retweet_count                 14640 non-null  int64
 10  text                          14640 non-null  object
 11  tweet_coord                   1019 non-null   object
 12  tweet_created                 14640 non-null  object
 13  tweet_location                9907 non-null   object
 14  user_timezone                 9820 non-null   object
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
```

Data Cleaning:

- Data cleaning is an essential step in the data preprocessing process to clean data in Python.
- we can use the Pandas library, which provides a wide range of functions for handling and cleaning data.



- **df.isnull().sum():** This line is used to check for missing values in the DataFrame 'df'. It executes the `**isnull()**` function, which returns `True` for missing (NaN) values and `False` for non-missing values. Then, `**sum()**` is applied to count the number of missing values in each column.

Data Cleaning (Data Preprocessing):

```
[7]: #check for missing values
     df.isnull().sum()
```

```
[7]: tweet_id                          0
     airline_sentiment                 0
     airline_sentiment_confidence      0
     negativereason                 5462
     negativereason_confidence      4118
     airline                           0
     airline_sentiment_gold        14600
     name                              0
     negativereason_gold           14608
     retweet_count                     0
     text                              0
     tweet_coord                   13621
     tweet_created                     0
     tweet_location                 4733
     user_timezone                  4820
     dtype: int64
```

- **df = df.drop([...], axis=1):** This code is used to remove specific columns from the DataFrame 'df'. The `**drop()**` method takes a list of column names to be dropped and the `axis=1` argument to indicate that columns, not rows, should be removed and we can **drop the unnecessary columns** from the table.

- **df.columns = [..., ...]:** Here, the column names in the DataFrame 'df' are being renamed. The two new column names specified in the list are assigned to the DataFrame's columns, providing more descriptive names.

```
[8]: # dropping the unnecessary columns from the table
     df = df.drop(['tweet_id', 'airline_sentiment_confidence', 'negativereason', 'negativereason_confidence', 'airline', '
     df.head()
```

[8]:

	airline_sentiment	text
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials to the experience... tacky.
2	neutral	@VirginAmerica I didn't today... Must mean I need to take another trip!
3	negative	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse
4	negative	@VirginAmerica and it's a really big bad thing about it

```
[9]: # renaming the columns in the table
     df.columns = ['sentiment_label', 'tweets']
     df.head()
```

[9]:

	sentiment_label	tweets
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials to the experience... tacky.
2	neutral	@VirginAmerica I didn't today... Must mean I need to take another trip!

- **Mapper Function:** A Python function called 'mapper' is defined. It takes a sentiment label as an argument and returns a numerical value based on the label. The function maps 'positive' to 1, 'neutral' to 0, and any other value to -1.

[10]:
```
# assigning value to the labels by creating a mapper() function (-1 -> negative , 0 -> neutral , 1 -> positive)
def mapper(sentiment_label):
    if sentiment_label == 'positive':
        return 1
    elif sentiment_label == 'neutral':
        return 0
    else:
        return -1
df['value_label'] = df['sentiment_label'].map(mapper)
df = df.reindex(columns = ['sentiment_label', 'value_label', 'tweets'])
df.head()
```

[10]:

	sentiment_label	value_label	tweets
0	neutral	0	@VirginAmerica What @dhepburn said.
1	positive	1	@VirginAmerica plus you've added commercials to the experience... tacky.

- **Stopwords Download:** The code downloads a set of common English stop words using the NLTK library and stores them in a set named 'stop_words'. These stop words are typically filtered out from text data during text processing.
- **stop words** present which need to be removed. Stop words refer to the connecting words like ‘the,’ ‘and’ ‘was,’ which do not provide any specific meaning, which will not help our analysis.
- Hence, we remove these and clean the data. NLTK is a python package used commonly for NLP tasks. Using this package, we can quickly get all the stopwords in English.

[11]:
```
#downloading and displaying stopwords for english language
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
print(stop_words)
```

```
{'needn', 't', "mustn't", 'them', 'wouldn', 'her', 'did', 'been', 'themselves', 'to', 'up', 'there', 'didn', "does
n't", 'once', 'some', 'theirs', 'during', 'my', 'were', "didn't", 'weren', 'all', 'further', 'through', 'can', 's',
'herself', 'now', 'me', 'these', 'down', 'haven', 'more', "don't", 'but', 'our', 'd', 've', 'a', 'over', "weren't",
'nor', 'had', 'ain', 'both', 'other', 'those', 'she', 'its', 'it', 'for', 'i', "you'd", "hasn't", 'where', 'their',
'have', 'does', 'the', "isn't", 'myself', 'same', 'yourselves', 'll', 'doesn', 'when', 'couldn', 'aren', "couldn't",
"shouldn't", 'in', 'him', 'why', "you've", 'which', "aren't", 'any', 'do', "wouldn't", 'as', 'very', 'here', "sha
n't", 'am', 'ma', 'at', 'an', "it's", 'doing', 'again', 'they', 'will', 'few', 'yourself', 'below', 'his', 'should',
'be', 'out', 'has', "mightn't", 'ours', 'no', 'hasn', 'wasn', 'whom', 'being', "needn't", 'himself', 'isn', 'should
n', 'into', 'own', "you're", 'if', 'by', 'hadn', 'how', "should've", 'just', 'after', "wasn't", 'too', 'above', 'her
s', 'mustn', 'because', 'who', 'between', 'then', 'about', 'itself', 'he', "that'll", 'not', 'o', 'won', 'than', 'o
r', 'you', 'from', 'm', 'don', 're', "hadn't", "haven't", 'on', 'of', 'and', 'yours', 'such', 'mightn', 'against', 'o
ff', 'is', 'was', 'only', 'under', 'this', 'shan', 'while', 'ourselves', 'what', 'that', 'each', 'y', "she's", 'wit
h', 'your', 'having', 'most', 'until', 'so', 'we', 'before', "you'll", "won't", 'are'}
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\sivar\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

- **Text Cleaning Functions:** Two text cleaning functions are defined:

    I. **clean_the_tweet(text):** This function tokenizes the text and keeps only alphabetic characters, converting them to lowercase. It returns a cleaned string with non-alphabetic characters removed.

    II. **text_process(msg):** This function removes punctuation and common English stop words from a text message. It returns a string with cleaned and processed words.

- **Data Filtering:** Rows in the DataFrame with 'neutral' sentiment labels are removed, effectively filtering out neutral sentiment tweets. A new column **'cleaned_tweet**' is created to store the preprocessed text.
- **Label Transformation:** In this step, the **'sentiment_label'** column is updated to contain binary labels. Any tweet labelled as 'positive' is assigned the value 1, and any other label is assigned the value 0.

```
[20]: # Function to preprocess the text
def preprocess_text(text):
    # Remove punctuations and numbers
    text = re.sub('[^a-zA-Z]', ' ', text)
    # Remove @mentions
    text = re.sub('@[A-Za-z0-9]+', ' ', text)
    # Remove '#' symbols
    text = re.sub(r'#', ' ', text)
     # Remove RT symbols
    text = re.sub(r'RT[\s]+', ' ', text)
    # Single character removal
    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)
    # Removing multiple spaces
    text = re.sub(r'\s+', ' ', text)
    # Converting to Lowercase
    text = text.lower()
    #removing urls
    text = re.sub(r'http\s+|https://\s+', ' ', text)
    return text
    # Apply the preprocessing to the 'text' column
df['clean_tweets'] = df['tweets'].apply(preprocess_text)
df.head()
```

- **Removing duplicates:** from the dataset

[22]:

	sentiment_label	value_label	tweets	clean_tweets
0	neutral	0	@VirginAmerica What @dhepburn said.	said
1	positive	1	@VirginAmerica plus you've added commercials to the experience... tacky.	plus added commercials experience tacky
2	neutral	0	@VirginAmerica I didn't today... Must mean I need to take another trip!	today must mean need take another trip
3	negative	-1	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse	really aggressive blast obnoxious entertainment guests faces amp little recourse
4	negative	-1	@VirginAmerica and it's a really big bad thing about it	really big bad thing

```
[23]: # removing duplicates
df.duplicated()
```

```
[23]: 0        False
1        False
2        False
3        False
4        False
         ...
14635    False
14636    False
14637    False
14638    False
14639    False
Length: 14640, dtype: bool
```

- **Tokenization:** The text in the 'cleaned_tweet' column is tokenized, which means it is split into lists of words or tokens. Each list represents a sentence from the original text.
- **Stemming:** Stemming is a text normalization technique that reduces words to their root forms. We need to convert these into the root word for easier modelling. We can use the **Snowball stemme**r and **Porter Stemmer** from **NLTK** is used to apply stemming to the tokenized words.

[24]:
```
# tokenization - splitting the given sentences into list of tokens, indexed or vectorized
tokenized_tweet = df['clean_tweets'].apply(lambda x: x.split())
tokenized_tweet.head()
```

[24]:
```
0                                                                                [said]
1                                                [plus, added, commercials, experience, tacky]
2                                                [today, must, mean, need, take, another, trip]
3    [really, aggressive, blast, obnoxious, entertainment, guests, faces, amp, little, recourse]
4                                                                  [really, big, bad, thing]
Name: clean_tweets, dtype: object
```

[25]:
```
# stemming - to reduce the words into root words
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(word) for word in x])
tokenized_tweet.head()
```

[25]:
```
0                                                                    [said]
1                                              [plu, ad, commerci, experi, tacki]
2                                    [today, must, mean, need, take, anoth, trip]
3    [realli, aggress, blast, obnoxi, entertain, guest, face, amp, littl, recours]
4                                                  [realli, big, bad, thing]
Name: clean_tweets, dtype: object
```

- **Combining Tokens:** The individual tokenized words are combined back into a single string for each sentence in the **'cleaned_tweet'** column. This is done to prepare the text for further analysis.
- **Final Text Preparation:** The DataFrame 'df' is updated to include the preprocessed and cleaned text in the 'cleaned_tweet' column. This step is crucial for sentiment analysis or any other text-based analysis you might want to perform.

[26]:
```
#combine words into single sentence
for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = " ".join(tokenized_tweet[i])
df['clean_tweets'] = tokenized_tweet
df.head()
```

[26]:

	sentiment_label	value_label	tweets	clean_tweets
0	neutral	0	@VirginAmerica What @dhepburn said.	said
1	positive	1	@VirginAmerica plus you've added commercials to the experience... tacky.	plu ad commerci experi tacki
2	neutral	0	@VirginAmerica I didn't today... Must mean I need to take another trip!	today must mean need take anoth trip
3	negative	-1	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse	realli aggress blast obnoxi entertain guest face amp littl recours
4	negative	-1	@VirginAmerica and it's a really big bad thing about it	realli big bad thing

- In summary, the code performs various data preprocessing and text cleaning tasks, including removing missing values, dropping columns, renaming columns, mapping sentiment labels to numeric values, removing stopwords, tokenizing text, stemming words, and preparing the text for analysis.

Data Analysis:

- **print("\nSummary Statics:"):** This line simply prints the message "Summary Statics:" to the console, indicating that a summary of statistics is about to be presented.
- **df.describe():** The `**describe()**` method generates summary statistics of the numeric columns in the DataFrame 'df'. These statistics typically include **count, mean, standard deviation, minimum, and maximum** values for each numeric column. It provides an overview of the central tendencies and dispersion of the data.

Data Analysis:

```
[30]: # description about the dataset (summary statistics)
print("\nSummary Statics:")
df.describe()
```

```
Summary Statics:
```

[30]:

	tweet_id	airline_sentiment_confidence	negativereason_confidence	retweet_count
count	1.464000e+04	14640.000000	10522.000000	14640.000000
mean	5.692184e+17	0.900169	0.638298	0.082650
std	7.791112e+14	0.162830	0.330440	0.745778
min	5.675883e+17	0.335000	0.000000	0.000000
25%	5.685592e+17	0.692300	0.360600	0.000000
50%	5.694779e+17	1.000000	0.670600	0.000000
75%	5.698905e+17	1.000000	1.000000	0.000000
max	5.703106e+17	1.000000	1.000000	44.000000

- **df.columns:** This line prints the column names in the DataFrame 'df'. It is used to display the names of all the columns in the DataFrame.

```
[4]: # to display all columns
df.columns
```

```
[4]: Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline',
       'airline_sentiment_gold', 'name', 'negativereason_gold',
       'retweet_count', 'text', 'tweet_coord', 'tweet_created',
       'tweet_location', 'user_timezone'],
      dtype='object')
```

- **print("\nDistribution of Tweet Location:"):** This line prints the message "Distribution of Tweet Location:" to the console, indicating that information about the distribution of tweet locations is going to be presented.
- **counts = df['tweet_location'].value_counts().sort_index():** Here, the code calculates the distribution of tweet locations. It counts the occurrences of each **unique value** in the 'tweet_location' column and **sorts** the results by index (the unique location names). The counts are stored in the 'counts' variable.

- **print(counts):** This line prints the 'counts' variable, which contains the distribution of tweet locations. It shows how many tweets are associated with each unique location.

```
[34]: #distribution of tweet location
print("\nDistribution of Tweet Location:")
counts = df['tweet_location'].value_counts().sort_index()
print(counts)
```

```
Distribution of Tweet Location:
tweet_location
  || san antonio, texas||    1
 Bronx, NY / Destin, Fl      1
 California 92705            1
 D(MD)V ✈️ NYC ✈️ Germany    1
 DC | Jersey City            1
                            ..
✡ Los Angeles ✡              1
✨                            1
❤                            2
サマセット、ニュージャージー州              1
명동서식 37.56638,126.984994     1
Name: count, Length: 3081, dtype: int64
```

- **print("\nDistribution of Negative Reason"):** This line prints the message "Distribution of Negative Reason" to the console, indicating that information about the distribution of negative reasons for tweets is going to be presented.
- **counts = df['negativereason'].value_counts():** Similar to the previous step, this code calculates the distribution of negative reasons. It counts the occurrences of each unique value in the 'negativereason' column. The counts are stored in the 'counts' variable.

```
[35]: #distribution of negativereason
print("\nDistribution of Negative Reason")
counts = df['negativereason'].value_counts()
print(counts)
```

```
Distribution of Negative Reason
negativereason
Customer Service Issue         2910
Late Flight                    1665
Can't Tell                     1190
Cancelled Flight                847
Lost Luggage                    724
Bad Flight                      580
Flight Booking Problems         529
Flight Attendant Complaints     481
longlines                       178
Damaged Luggage                  74
Name: count, dtype: int64
```

- **print(counts):** This line prints the 'counts' variable, which contains the distribution of negative reasons. It shows how many tweets are associated with each unique negative reason.

```
[37]: #unique values identification
      df['airline_sentiment'].unique()

[37]: array(['neutral', 'positive', 'negative'], dtype=object)
```

- **print("\nDistribution of Sentiments:"):** This line prints the message "Distribution of Sentiments:" to the console, indicating that information about the distribution of sentiment labels is going to be presented.
- **df['airline_sentiment'].value_counts():** This code counts the occurrences of each unique value in the 'airline_sentiment' column, which represents sentiment labels (positive, negative, neutral) for the tweets. It shows the count of each sentiment label.

```
[38]: # label counts
      print("\nDistribution of Sentiments:")
      df['airline_sentiment'].value_counts()

      Distribution of Sentiments:
[38]: airline_sentiment
      negative    9178
      neutral     3099
      positive    2363
      Name: count, dtype: int64
```

- In summary, this code provides a summary of statistics for numeric columns, displays the column names, and presents the distribution of tweet locations, negative reasons, and sentiment labels in the DataFrame. It is useful for understanding the characteristics of the dataset.

Data Visualization:

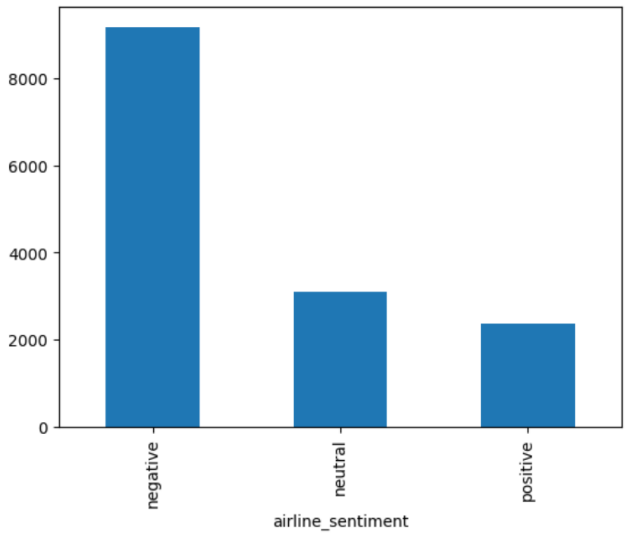
- While working with data, it can be difficult to understand the data when it's in the tabular form.
- To understand what exactly our data conveys, and to better clean it and select suitable models for it, we need to visualize it or represent it in pictorial form.
- This helps expose patterns, correlations, and trends that cannot be obtained when data is in a table or CSV file.
- The process of finding trends and correlations in our data by representing it pictorially is called Data Visualization.
- To perform data visualization in python, we can use various python data visualization modules such as Matplotlib, Seaborn, Plotly, etc.

Bar Plot Visualization

- The code extracts the **'airline_sentiment'** column from a DataFrame (df), counts the occurrences of each unique sentiment category i.e., Positive, Negative and Neutral.
- and creates a bar plot to visualize the **distribution of sentiments.**

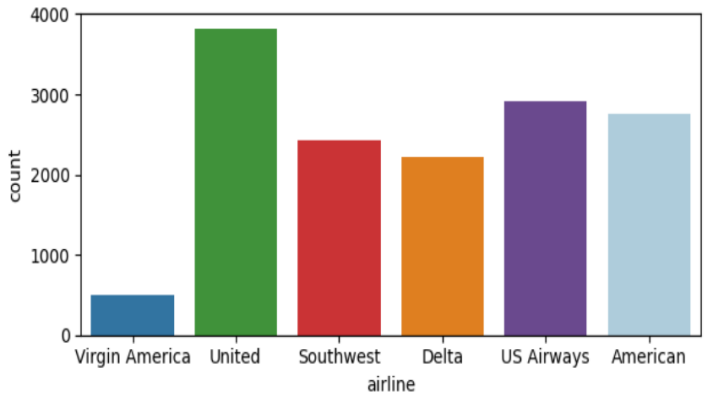
```
[36]: #plot the airline_sentiment counts
      df['airline_sentiment'].value_counts().plot(kind='bar')
```

```
[36]: <Axes: xlabel='airline_sentiment'>
```



- Here, this sets the figure size, uses **Seaborn (sns)** to create a **count plot**, and specifies the data source (df) and the variable to count (x='airline').
- It also customizes the color palette for the bars.
- Finally, it displays the count plot.
- This code is likely used to visualize the distribution of data within the 'airline' column of the DataFrame df using a bar plot.

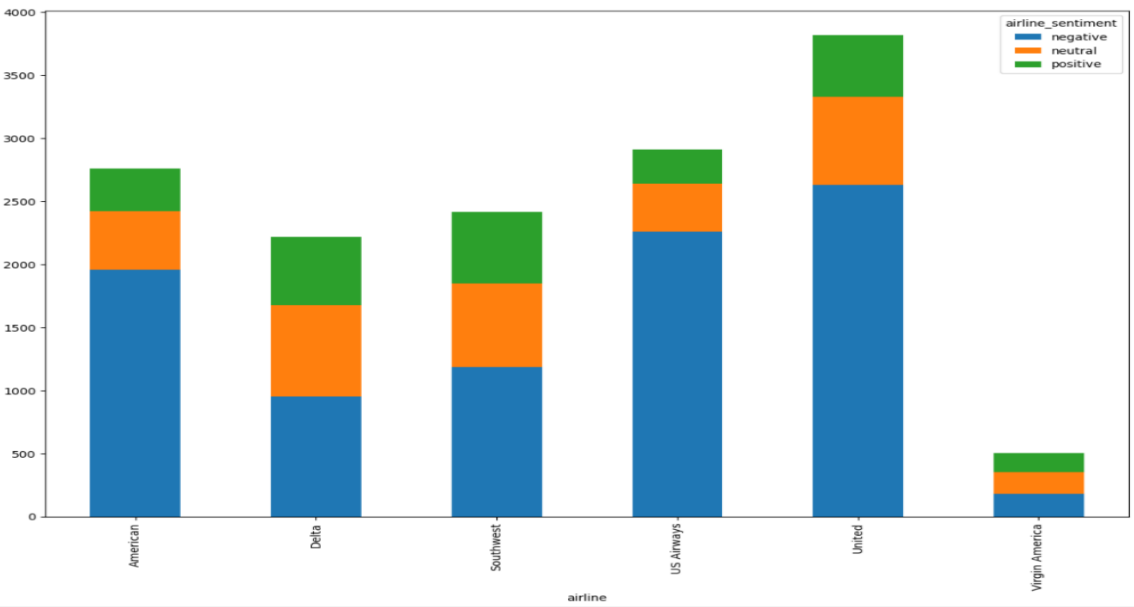
[39]:
```
# Checking the distribution of airlines over several countries
plt.figure(figsize=(7,3))
sns.countplot(data=df,x='airline', palette=['#1f78b4', '#33a02c', '#e31a1c', '#ff7f00', '#6a3d9a', '#a6cee3'])
plt.show()
```



- This groups the data in the DataFrame (df) by 'airline' and 'airline_sentiment', counts the size of each group, and then unstacks the results to create a **stacked bar plot.**
- The resulting plot visualizes the distribution of sentiment categories within each airline and is displayed with a specified figure size.
- This code is used for visualizing how sentiment categories are **distributed across different airlines.**

[40]:
```
# grouping the distribution of postivie, negative, neutral sentiment_label for each country
figure_2 = df.groupby(['airline', 'airline_sentiment']).size()
figure_2.unstack().plot(kind='bar', stacked=True, figsize=(15,10))
```

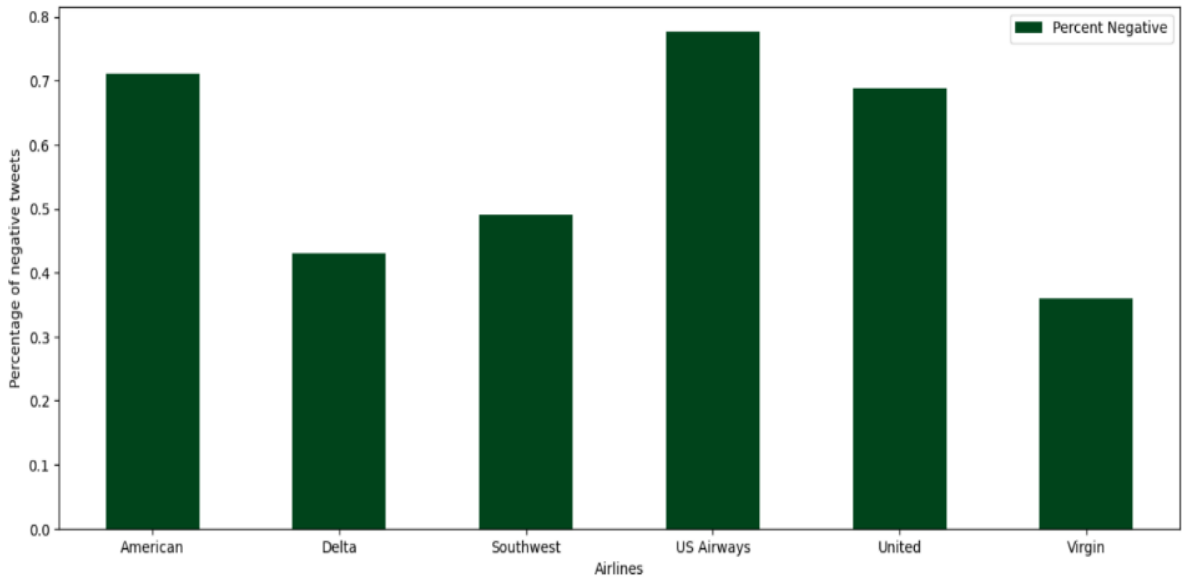
[40]: <Axes: xlabel='airline'>



- Th calculates the **percentage of negative tweets for different airlines** and displays the results in a **bar chart**.
- It groups the tweets by airline, counts negative tweets, and then shows the percentage of negative tweets for each airline.

```
[10]: #distribution of negative tweets over airlines
neg_tweets = df.groupby(['airline','airline_sentiment']).count().iloc[:,0]
total_tweets = df.groupby(['airline'])['airline_sentiment'].count()

my_dict = {'American':neg_tweets[0] / total_tweets[0],'Delta':neg_tweets[3] / total_tweets[1],'Southwest': neg_tweets[6] / total_tweets[2],
'US Airways': neg_tweets[9] / total_tweets[3],'United': neg_tweets[12] / total_tweets[4],'Virgin': neg_tweets[15] / total_tweets[5]}
perc = pd.DataFrame.from_dict(my_dict, orient = 'index')
perc.columns = ['Percent Negative']
print(perc)
ax = perc.plot(kind = 'bar', rot=0, colormap = 'Greens_r', figsize = (15,6))
ax.set_xlabel('Airlines')
ax.set_ylabel('Percentage of negative tweets')
plt.show()
```

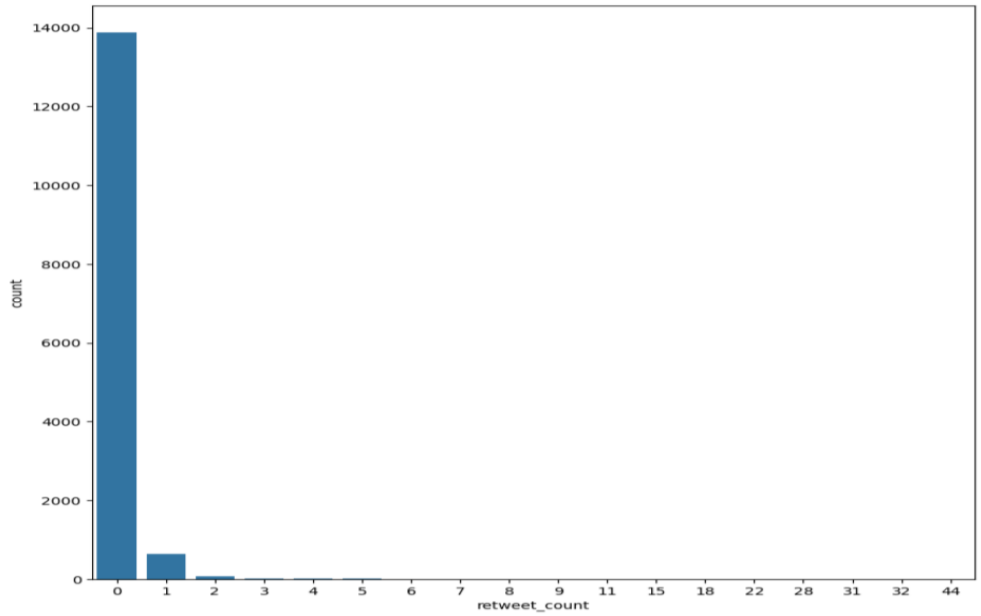
```
            Percent Negative
American            0.710402
Delta               0.429793
Southwest           0.490083
US Airways          0.776862
United              0.688906
Virgin              0.359127
```



- This will use **Seaborn (sns)** to create a **count plot**, and specifies that it should count the occurrences of values in the 'retweet_count' column from the DataFrame (df).
- The resulting plot shows the **distribution of retweet counts**

```
[7]: # distribution of retweet count across several countries
plt.figure(figsize=(10,10))
sns.countplot(x="retweet_count", data=df)
```

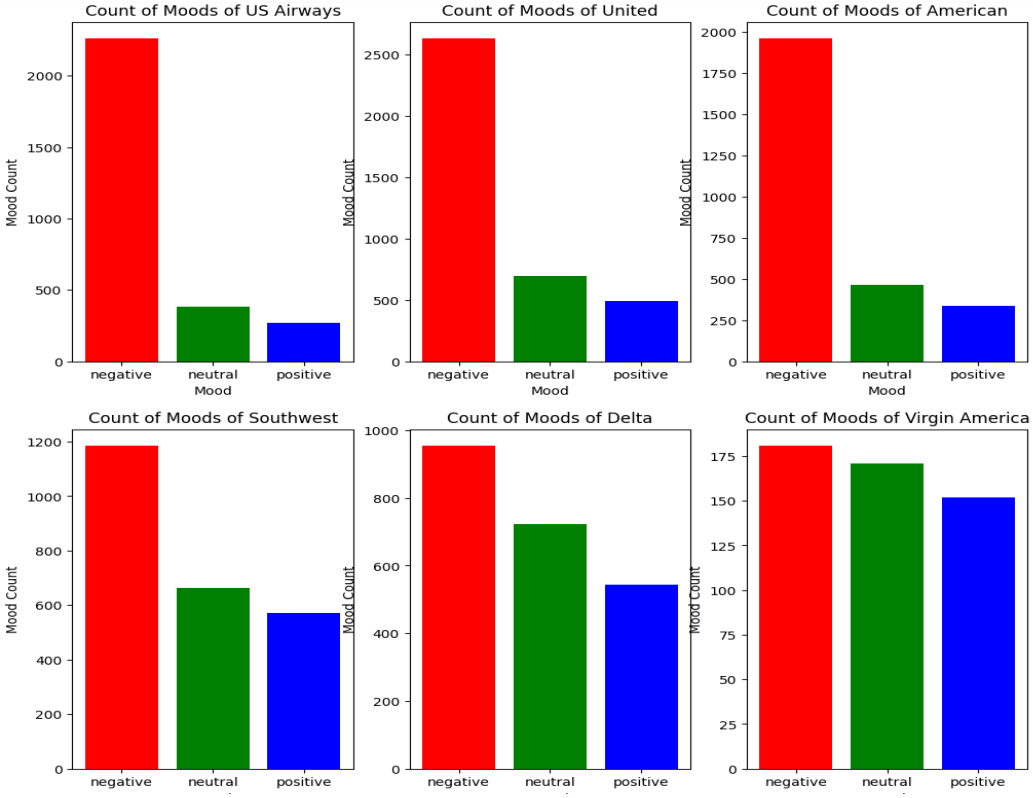
[7]: <Axes: xlabel='retweet_count', ylabel='count'>



- This code first prints the **total number of tweets for each airline** and their respective sentiment counts in descending order.
- Then, it creates a **figure with six subplots (2 rows and 3 columns)** to visualize the **distribution of sentiment counts for six specific airline**s (US Airways, United, American, Southwest, Delta, Virgin America).
- showing counts of negative, neutral, and positive tweets.

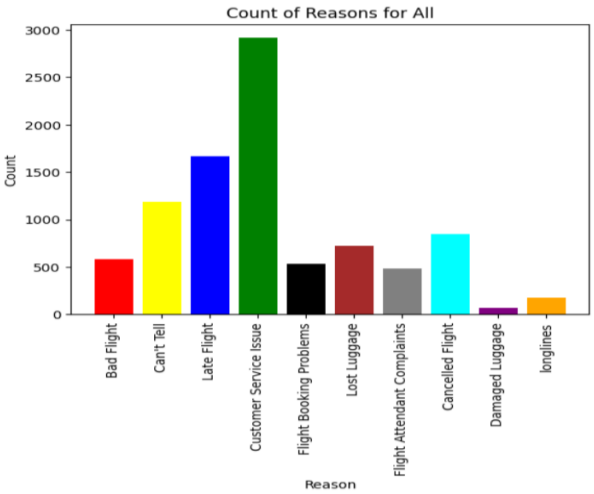
[9]:
```
#count of moods of several countries
print("Total number of tweets for each airline \n ",df.groupby('airline')['airline_sentiment'].count().sort_values(ascending=False))
airlines= ['US Airways','United','American','Southwest','Delta','Virgin America']
plt.figure(1,figsize=(12, 12))
for i in airlines:
    indices= airlines.index(i)
    plt.subplot(2,3,indices+1)
    new_df=df[df['airline']==i]
    count=new_df['airline_sentiment'].value_counts()
    Index = [1,2,3]
    plt.bar(Index,count, color=['red', 'green', 'blue'])
    plt.xticks(Index,['negative','neutral','positive'])
    plt.ylabel('Mood Count')
    plt.xlabel('Mood')
    plt.title('Count of Moods of '+i)
```

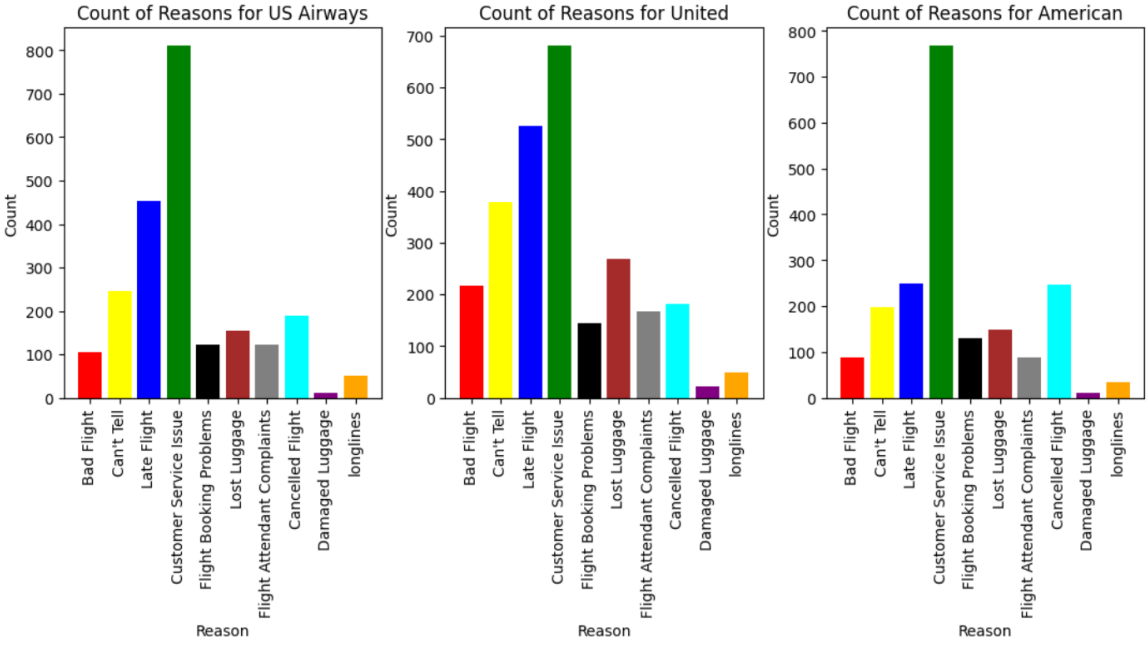
```
Total number of tweets for each airline
  airline
United            3822
US Airways        2913
American          2759
Southwest         2420
Delta             2222
Virgin America     504
Name: airline_sentiment, dtype: int64
```

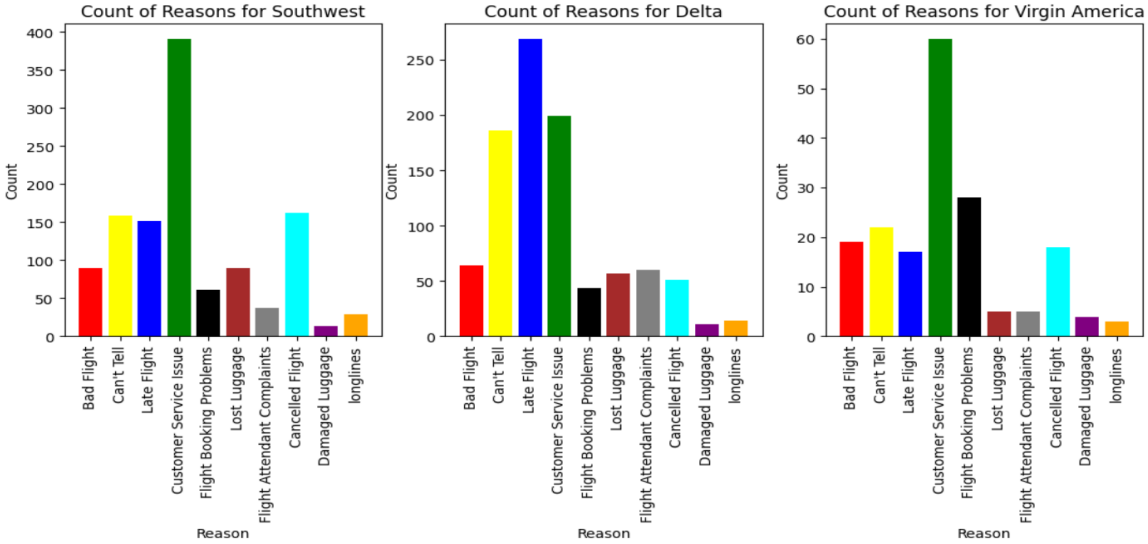


- It finds the number of unique values in the 'negativereason' column.
- It defines a function NR_Count that calculates the count of negative reasons for a given airline.
- It creates a function “plot_reason” to generate **bar plots** showing the **count of negative reasons for a specific airline.**
- It calls the “plot_reason” function for 'All' airlines and then creates a **figure with six subplots (2 rows and 3 columns)** to visualize the count of negative reasons for six specific airlines.

```
[1]: #count of reasons for all countries
df['negativereason'].nunique()

NR_Count=dict(df['negativereason'].value_counts(sort=False))
def NR_Count(Airline):
    if Airline=='All':
        a=df
    else:
        a=df[df['airline']==Airline]
    count=dict(a['negativereason'].value_counts())
    Unique_reason=list(df['negativereason'].unique())
    Unique_reason=[x for x in Unique_reason if str(x) != 'nan']
    Reason_frame=pd.DataFrame({'Reasons':Unique_reason})
    Reason_frame['count']=Reason_frame['Reasons'].apply(lambda x: count[x])
    return Reason_frame
def plot_reason(Airline):

    a=NR_Count(Airline)
    count=a['count']
    Index = range(1,(len(a)+1))
    plt.bar(Index,count, color=['red','yellow','blue','green','black','brown','gray','cyan','purple','orange'])
    plt.xticks(Index,a['Reasons'],rotation=90)
    plt.ylabel('Count')
    plt.xlabel('Reason')
    plt.title('Count of Reasons for '+Airline)

plot_reason('All')
plt.figure(2,figsize=(13, 13))
for i in airlines:
    indices= airlines.index(i)
    plt.subplot(2,3,indices+1)
    plt.subplots_adjust(hspace=0.9)
    plot_reason(i)
```

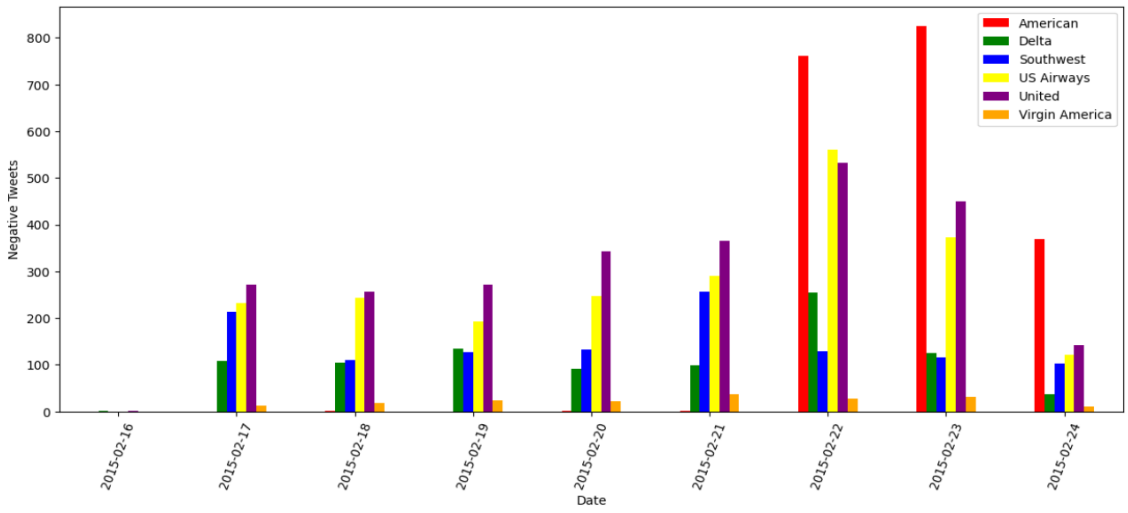

Count of Reasons for All
Count
Reason
Bad Flight
Can't Tell
Late Flight
Customer Service Issue
Flight Booking Problems
Lost Luggage
Flight Attendant Complaints
Cancelled Flight
Damaged Luggage
longlines


Count of Reasons for US Airways
Count of Reasons for United
Count of Reasons for American
Count
Reason
Bad Flight
Can't Tell
Late Flight
Customer Service Issue
Flight Booking Problems
Lost Luggage
Flight Attendant Complaints
Cancelled Flight
Damaged Luggage
longlines


Count of Reasons for Southwest
Count of Reasons for Delta
Count of Reasons for Virgin America
Count
Reason
Bad Flight
Can't Tell
Late Flight
Customer Service Issue
Flight Booking Problems
Lost Luggage
Flight Attendant Complaints
Cancelled Flight
Damaged Luggage
longlines

- It Converts the 'tweet_created' column to a Pandas datetime format.
- Extracts only the date (no time) using **“dt.date”.**
- Groups the “data” by date, airline, airline sentiment columns and count the occurrences.
- Filters for negative sentiment tweets, then Group and plot the **negative tweets by date and airline** using a **bar plot.**

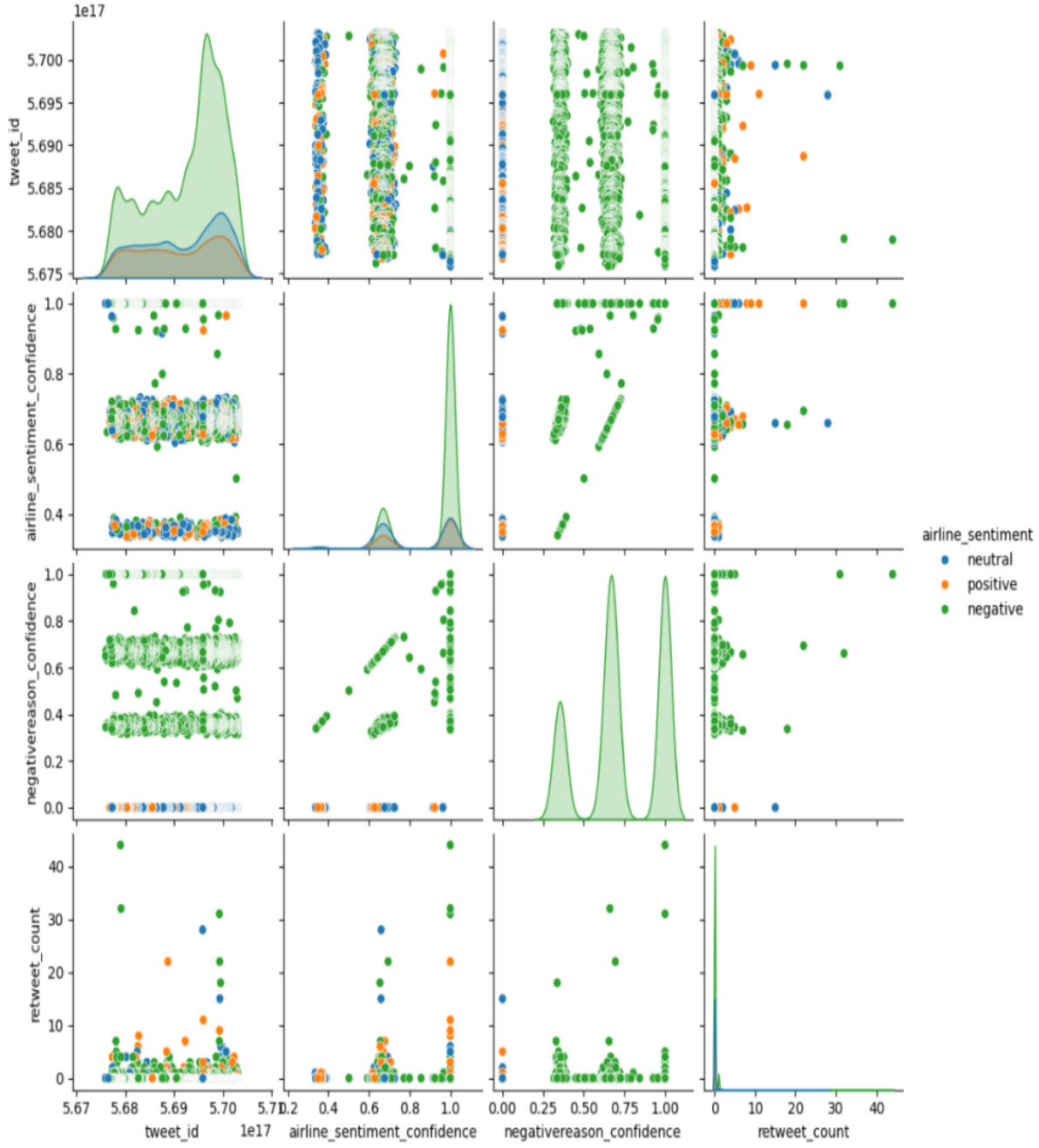
```
[12]: #grouping the date column and visualized the distribution of negative tweets over grouped date
date = df.reset_index()
#convert the Date column to pandas datetime
date.tweet_created = pd.to_datetime(date.tweet_created)
#Reduce the dates in the date column to only the date and no time stamp using the 'dt.date' method
date.tweet_created = date.tweet_created.dt.date
date.tweet_created.head()
df = date
day_df = df.groupby(['tweet_created','airline','airline_sentiment']).size()
# day_df = day_df.reset_index()
day_df
day_df = day_df.loc(axis=0)[:,:,'negative']

#groupby and plot data
ax2 = day_df.groupby(['tweet_created','airline']).sum().unstack().plot(kind = 'bar', color=['red', 'green', 'blue','yellow','purple','orange'], figsize = (15,6), rot
labels = ['American','Delta','Southwest','US Airways','United','Virgin America']
ax2.legend(labels = labels)
ax2.set_xlabel('Date')
ax2.set_ylabel('Negative Tweets')
plt.show()
```



Pair Plot Visualization

- This code uses Seaborn to create a **pairplot** of the DataFrame (df), where different pairs of variables are plotted against each other.
- It allows to visualize **relationships and distributions between variables while differentiating sentiments.**

```
[8]: # visualization of all columns
sns.pairplot(df,hue='airline_sentiment')
```



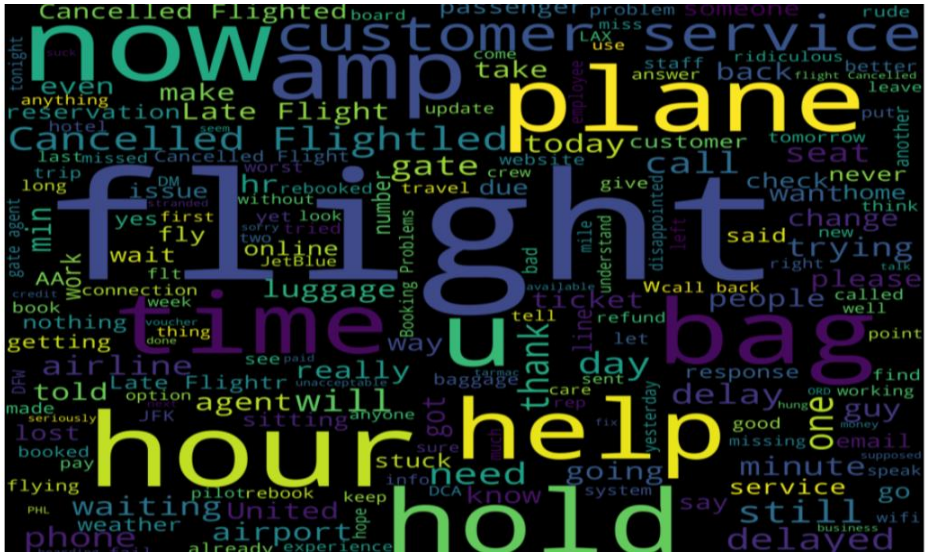
Word Cloud Visualization

- It extracts the text from these positive sentiment tweets and joins them into a single long string.
- This code creates a **word cloud** and visualizing the **most common words in positive sentiment tweets.**
- Then filtering out URLs, Twitter handles, and 'RT' (retweets), and customizing the appearance of the word cloud. where word size represents its frequency.

[13]:
```
#visualize the frequent words for +ve
from wordcloud import WordCloud,STOPWORDS
new_df=df[df['airline_sentiment']=='positive']
words = ' '.join(new_df['text'])
cleaned_word = " ".join([word for word in words.split()
                         if 'http' not in word
                             and not word.startswith('@')
                             and word != 'RT'
                         ])
wordcloud = WordCloud(stopwords=STOPWORDS,
                      background_color='black',
                      width=3000,
                      height=2500
                     ).generate(cleaned_word)
plt.figure(1,figsize=(12, 12))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

- It extracts the text from these negative sentiment tweets and joins them into a single long string.
- This code creates a **word cloud** and visualizing the **most common words in positive sentiment tweets.**
- Then filtering out URLs, Twitter handles, and 'RT' (retweets), and customizing the appearance of the word cloud. where word size represents its frequency.

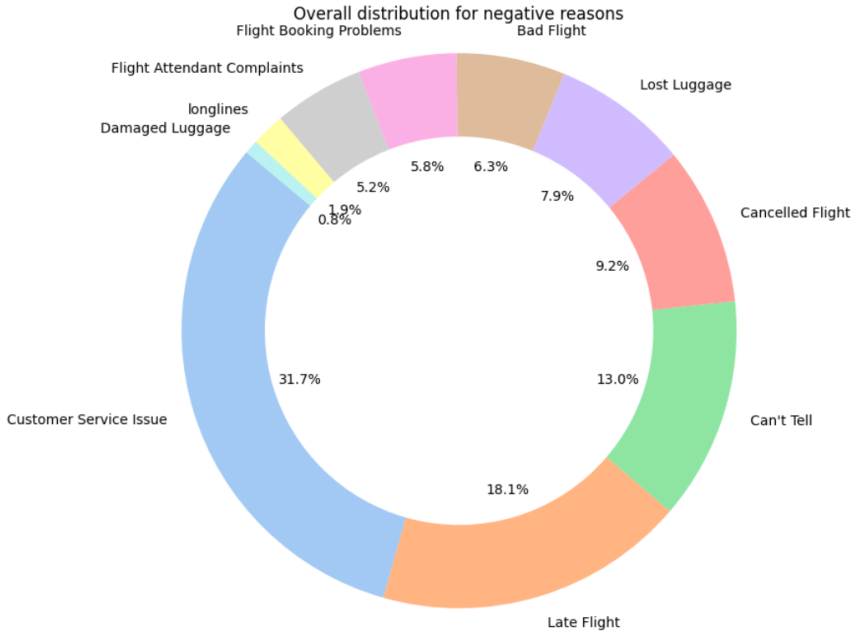
[14]:
```
#visualize the frequent words for -ve
new_df=df[df['airline_sentiment']=='negative']
words = ' '.join(new_df['text'])
cleaned_word = " ".join([word for word in words.split()
                         if 'http' not in word
                             and not word.startswith('@')
                             and word != 'RT'
                         ])
wordcloud = WordCloud(stopwords=STOPWORDS,
                      background_color='black',
                      width=3000,
                      height=2500
                     ).generate(cleaned_word)
plt.figure(1,figsize=(12, 12))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```



Pie Chart Visualization

- This code visualizes the **distribution of negative reasons in a dataset** using a **donut-like pie chart**.
- It calculates the counts of each negative reason, selects pastel colours, and creates the chart with percentage labels and a title.
- The result is a circular chart showing the distribution of negative reasons.

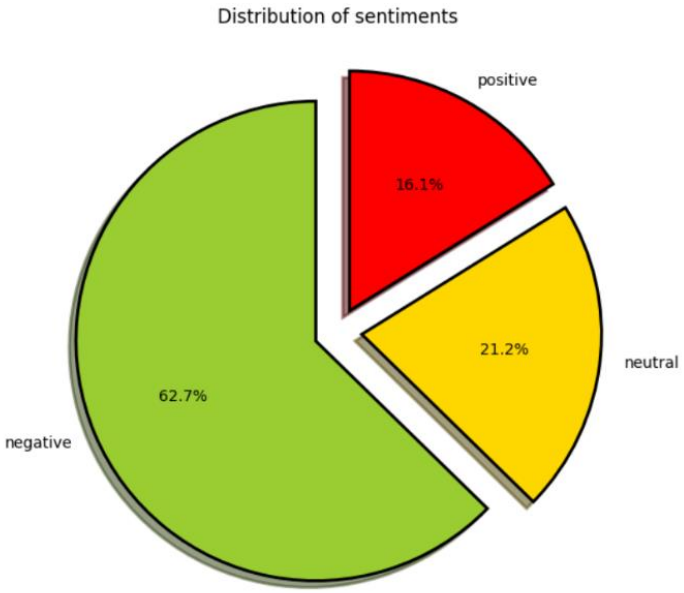
```
[15]: # Calculate the value counts for each negative reason
value_counts = df['negativereason'].value_counts()

# Create a donut-like pie chart using matplotlib and seaborn
plt.figure(figsize=(8, 8))
labels = value_counts.index
values = value_counts.values
colors = sns.color_palette('pastel')[0:len(labels)]  # Use pastel colors for the chart
plt.pie(values, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140, wedgeprops=dict(width=0.3))
plt.title('Overall distribution for negative reasons')
plt.axis('equal')  # Equal aspect ratio ensures the pie chart is drawn as a circle.
plt.show()
```



- This code creates a **pie chart** to visualize the **distribution of sentiments in the dataset.**
- It sets the figure size, colours, and various visual properties.
- The pie chart has exploded sections, percentage labels, and a title, making it easier to see the proportions of different sentiment categories.

```
[16]: # pie chart visualization of airline_sentiment
fig= plt.figure(figsize=(7,7))
col=("yellowgreen", "gold", "red")
wp={'linewidth':2, 'edgecolor':"black"}
tags=df['airline_sentiment'].value_counts()
explode= (0.1,0.1,0.1)
tags.plot(kind='pie', autopct='%1.1f%%', shadow=True, colors=col, startangle=90, wedgeprops=wp,explode=explode,label='')
plt.title('Distribution of sentiments')
```

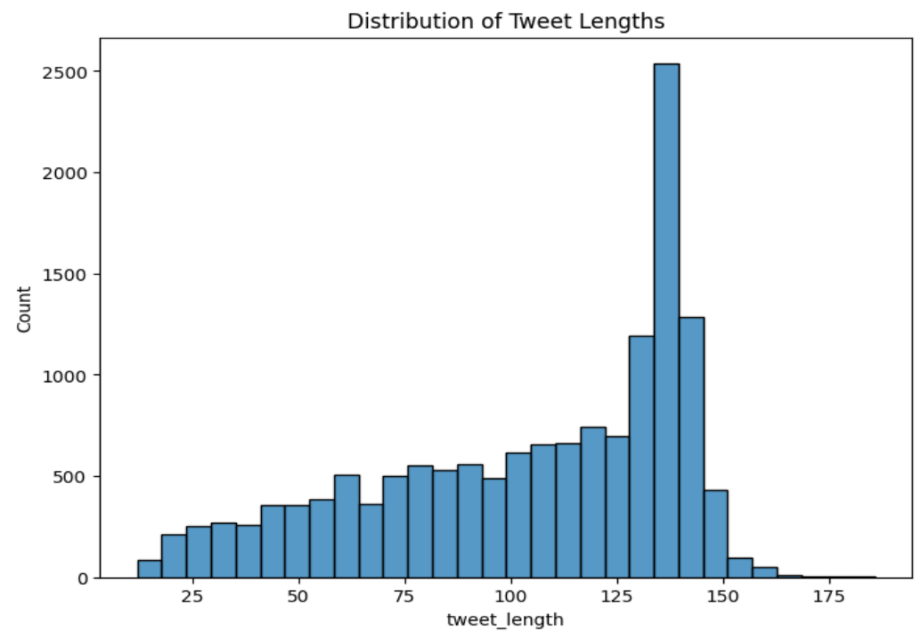
```
[16]: Text(0.5, 1.0, 'Distribution of sentiments')
```



Histogram Visualization

- This code calculates the length of tweets in a DataFrame(df) and then creates a **histogram** to show the **distribution of tweet lengths**.
- It uses **Seaborn's `histplot`** to display the distribution and adds a title for clarity.

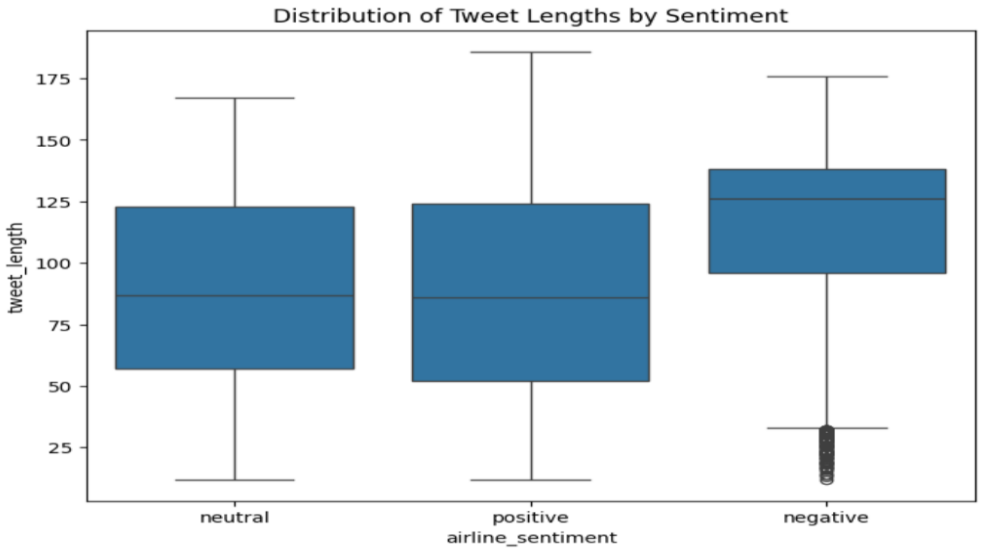
```
[17]: # Creating  column 'tweet_length'
df['tweet_length'] = df['text'].apply(len)
# Histogram of tweet lengths
plt.figure(figsize=(8,6))
sns.histplot(df['tweet_length'], bins=30)
plt.title('Distribution of Tweet Lengths')
plt.show()
```



Box Plot Visualization

- It creates a **boxplot** to visualize the **distribution of tweet lengths** by sentiment.
- It uses **Seaborn t**o display how tweet lengths vary for different sentiment categories.
- The boxplot provides insights into the distribution and spread of tweet lengths for each sentiment group.

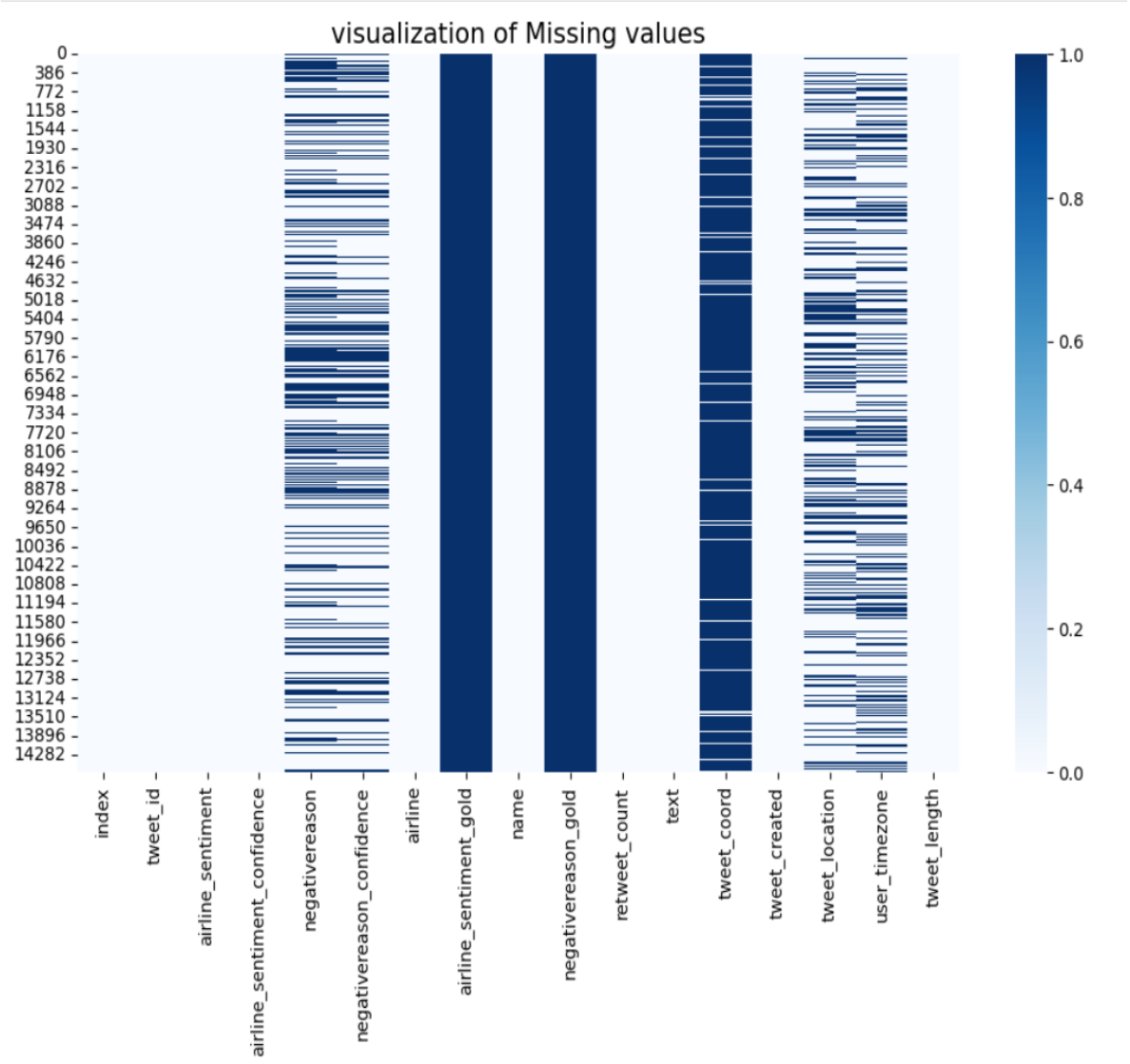
[18]:
```
# Creating  column 'tweet_length'
df['tweet_length'] = df['text'].apply(len)

# Boxplot of tweet lengths
plt.figure(figsize=(8,6))
sns.boxplot(x='airline_sentiment', y='tweet_length', data=df)
plt.title('Distribution of Tweet Lengths by Sentiment')
plt.show()
```



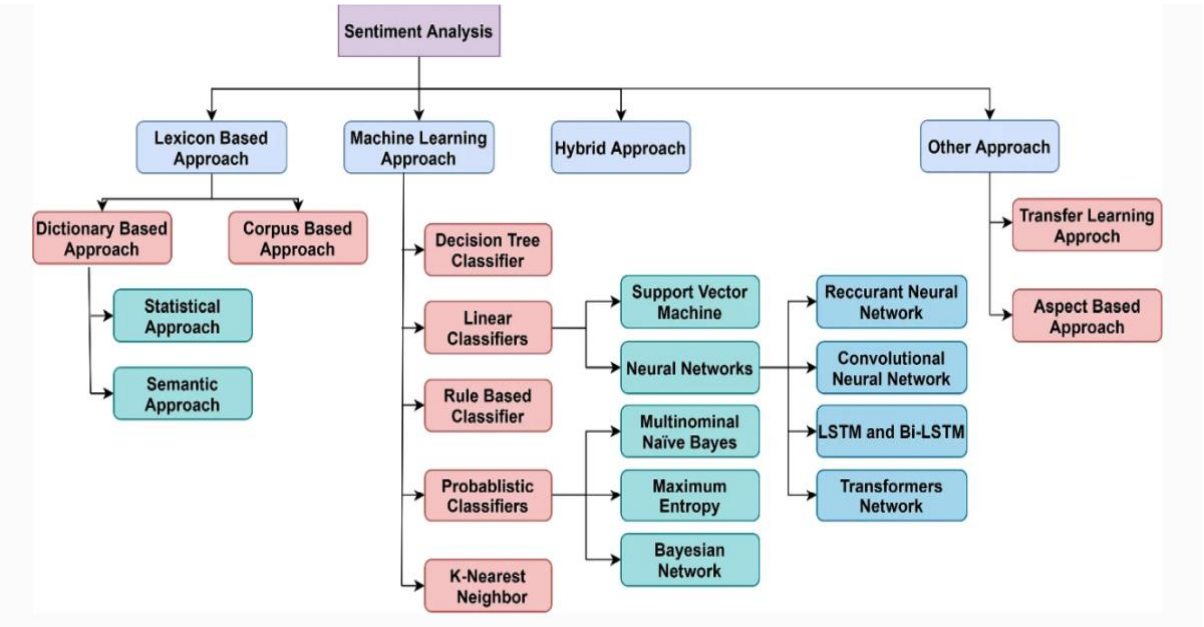
Heat Map Visualization

- This code generates a **heatmap** using **Seaborn** to visualize missing values (NaN or null values) in the DataFrame.
- The **blue** colour represents **missing values**, while the **other areas** show **non-missing data**.
- This visualization helps to **identify the presence and patterns of missing values in the dataset.**

```
[19]: #Visualization of missing value of all columns using heatmap
      plt.figure(figsize=(12,7))
      sns.heatmap(df.isnull(), cmap = "Blues")
      plt.title("visualization of Missing values", fontsize = 15)
      plt.show()
```



Model Selection:



Models Used:

We are going to **compare** the **accuracy of** these **models** for the given dataset

1. Random Forest
2. LSTM – Long Short-Term Memory
3. SVM – Support Vector Machine

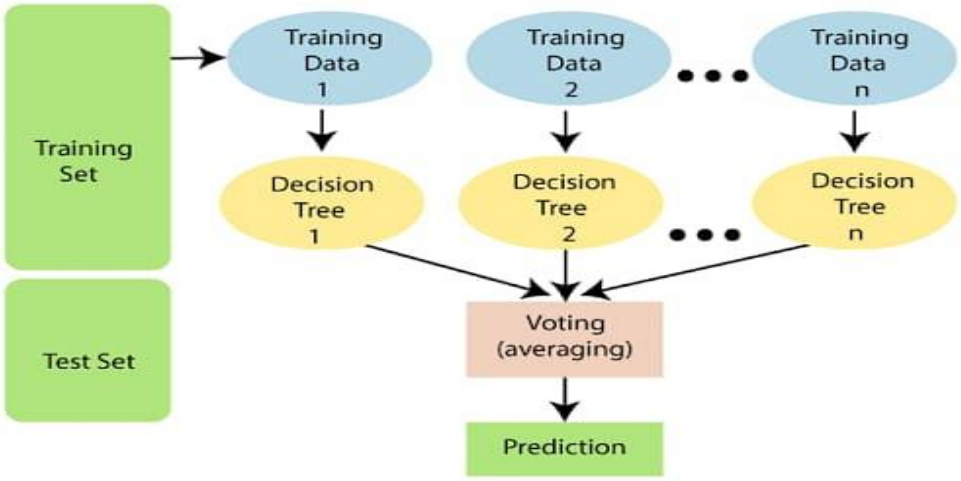
1.Random Forest:

Description:

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.
- It can be used for both Classification and Regression problems in ML.
- It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

- Random Forest is known for its high accuracy, robustness, and resistance to overfitting, making it a popular choice in various machine learning applications. It's often used in tasks like classification, regression, and feature selection.

Architecture:



- **Decision Trees:** These are the fundamental building blocks, created using a random subset of the training data. This randomness helps to reduce overfitting.
- **Bagging:** it uses a technique called bagging (Bootstrap Aggregating) to create multiple decision trees.
- **Feature Randomness:** This further diversifies the trees and improves generalization.
- **Voting (Classification) or Averaging (Regression):** In classification tasks, it combines the results of individual trees by taking a majority vote. In regression tasks, it averages the predictions from each tree.

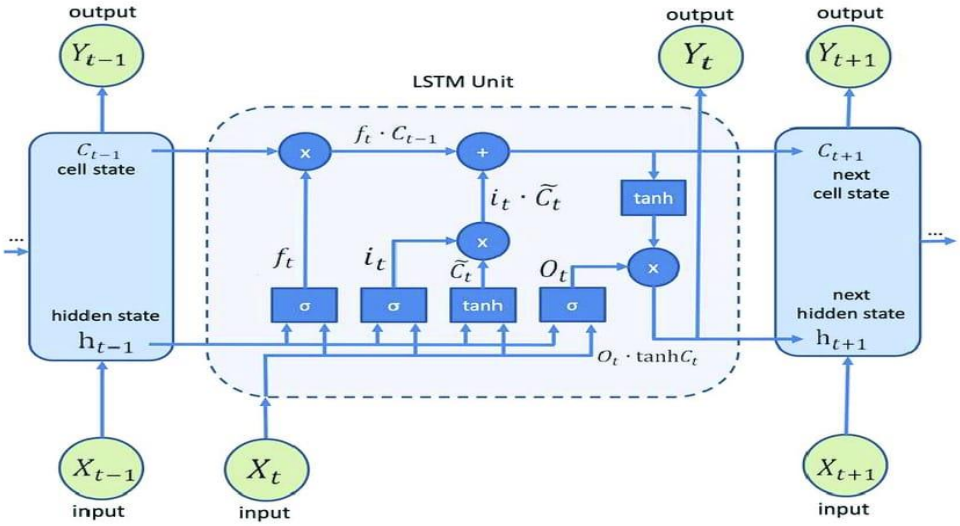
2.Long Short-Term Memory (LSTM):

Description:

- it is a type of recurrent neural network (RNN) architecture designed to capture and model sequential data.

- LSTMs are known for their ability to handle long-range dependencies in data, making them well-suited for tasks like speech recognition, machine translation, and sentiment analysis.
- They have largely overcome the vanishing gradient problem that plagued traditional RNNs, making them a powerful choice for many sequence-related tasks.

Architecture:



- **Input (Xt):** The current input data point.
- **Previous hidden state (ht-1):** Information from the previous time step.
- **Previous cell state (ct-1):** The cell state from the previous time step.
- **Forget gate:** Decides what information to forget.
- **Input gate:** Decides what new information to add.
- **Output gate:** Decides what information to send to the output.

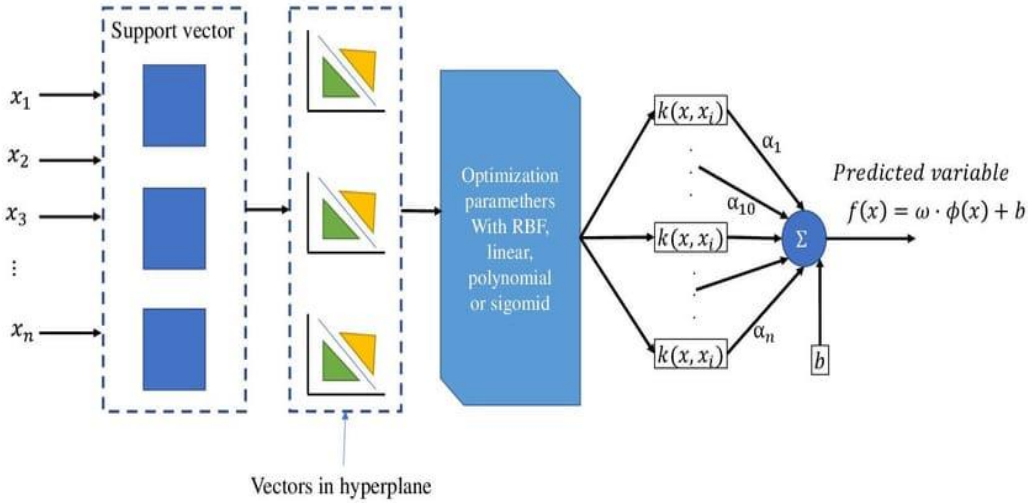
3.Support Vector Machine (SVM):

Description:

- it is a supervised machine learning algorithm used for classification and regression tasks.
- Its primary goal is to find a hyperplane

- that best separates data into distinct classes in the case of classification,
- to find a regression line that best fits the data in the case of regression.
- SVMs are known for their ability to handle high-dimensional data, work well with small to medium-sized datasets, and produce robust models.

Architecture:



- **Objective:** The main objective of SVM is to find a hyperplane that maximizes the margin between two classes, making it an ideal decision boundary.
- **Hyperplane:** In a binary classification problem, the hyperplane is the line that separates the data into two classes. For multi-class problems, SVM uses multiple hyperplanes.
- **Support Vectors:** These are the data points that are closest to the hyperplane and have the most influence on the placement and orientation of the hyperplane.
- The equation for the hyperplane is typically represented as

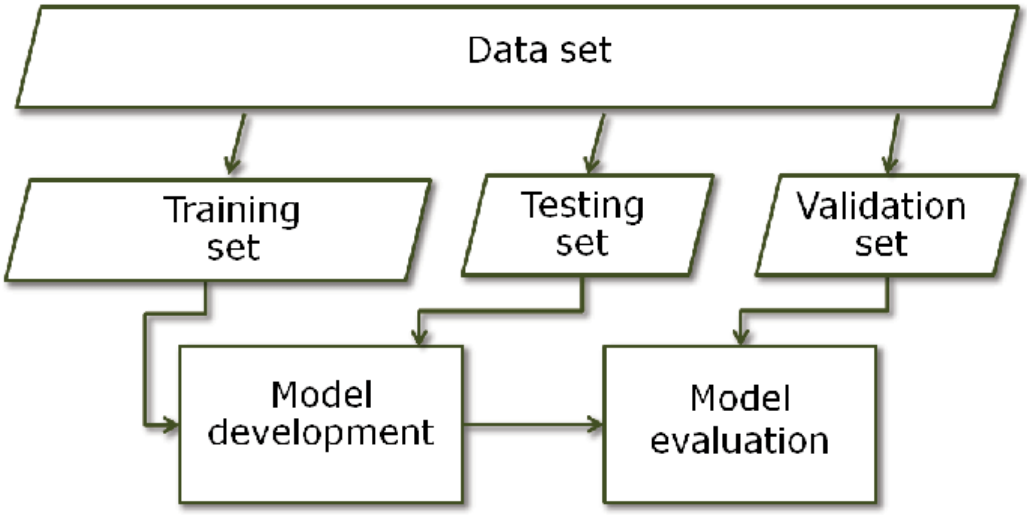
  w · x + b = 0, where w is the weight vector, x is the input data vector, and b is the bias.

- **Kernel Trick:** SVM can handle non-linear data by using a kernel function. This function transforms the original data into a higher-dimensional space

- **Classification:** To classify new data points, SVM uses the sign of the equation

  w · x + b. If it's positive, the point belongs to one class, and if it's negative, it belongs to the other class.

Model Development and Evaluation:



Importing Libraries:

Model Development And Evaluation:

[20]:
```
# importing necessary libraries
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from matplotlib import pyplot
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.metrics import f1_score
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from lime import lime_tabular
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.layers import LSTM, Bidirectional
from tensorflow.keras.layers import Dense, Dropout
import pickle
from keras.preprocessing.text import Tokenizer
import itertools
import tensorflow as tf
```

- The code imports various machine learning and deep learning libraries for text classification and natural language processing (NLP) tasks.
- It covers functionalities like data splitting, model selection, text vectorization, model evaluation metrics, and deep learning components.
- **sklearn** is imported for machine learning-related tasks, including model selection, metrics, and classifiers.
- **matplotlib** is imported for data visualization.
- **tensorflow.keras** is used for deep learning, including building neural network models.
- **pickle** is used for object serialization.
- **lime** is a library for explaining the predictions of machine learning models
- The code imports specific functions and classes from the libraries such as **train_test_split** and various metrics like **accuracy, precision, recall, and F1 score.**
- It imports different machine learning models like **Support Vector Machines (SVM)**, and ensemble methods like **Random Forest.**
- It imports NLP-specific components like text vectorization (**CountVectorizer, TfidfVectorizer**), tokenization (**Tokenizer**), and **embedding layers.**
- It imports deep learning layers and models, **including LSTM (Long Short-Term Memory)** and Bidirectional layers, which are commonly used for sequence data and NLP tasks.

Dataset Used:

Dataset Link: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment**.**

The goal is to build machine learning models to classify the sentiment of tweets. Analysts and data scientists can use this data to understand how passengers perceive different airlines on social media and to develop strategies for improving customer satisfaction.

The dataset showing the **original tweets** and the **cleaned tweets** along with the **values assigned** for each **label**.

[121…] df

[121…]

	sentiment_label	tweets	value_label	cleaned_tweet
0	1	@VirginAmerica plus you've added commercials to the experience... tacky.	2	ad commerci experi tacki
1	0	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse	0	realli aggress blast obnoxi entertain guest face amp littl recours
2	0	@VirginAmerica and it's a really big bad thing about it	0	realli big bad thing
3	0	@VirginAmerica seriously would pay $30 a flight for seats that didn't have this playing.\nit's really the only bad thing about flying VA	0	would pay flight seat play realli bad thing fli va
4	1	@VirginAmerica yes, nearly every time I fly VX this “ear worm” won’t go away :)	2	nearli everi time fli vx ear worm go away
...	...	...	...	...
11536	0	@AmericanAir my flight was Cancelled Flightled, leaving tomorrow morning. Auto rebooked for a Tuesday night flight but need to arrive Monday.	0	flight cancel flightl leav tomorrow morn auto rebook tuesday night flight need arriv monday
11537	0	@AmericanAir right on cue with the delays 👌	0	cue delay
11538	1	@AmericanAir thank you we got on a different flight to Chicago.	2	got differ flight chicago
11539	0	@AmericanAir leaving over 20 minutes Late Flight. No warnings or communication until we were 15 minutes Late Flight. That's called shitty customer svc	0	minut late flight warn commun minut late flight call shitti custom svc
11540	0	@AmericanAir you have my money, you change my flight, and don't answer your phones! Any other suggestions so I can make my commitment??	0	money chang flight answer phone suggest make commit

11541 rows × 4 columns

Model Development:

1. Random Forest:

1. Random Forest:

[160…]
```python
# Splitting the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['tweets'], df['sentiment_label'], test_size=0.2, random_state=42)

# Feature Extraction
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=2500, min_df=7, max_df=0.8)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()

# Model Training
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
classifier.fit(X_train, y_train)
```

[160…]
```
RandomForestClassifier
RandomForestClassifier(n_estimators=1000, random_state=0)
```

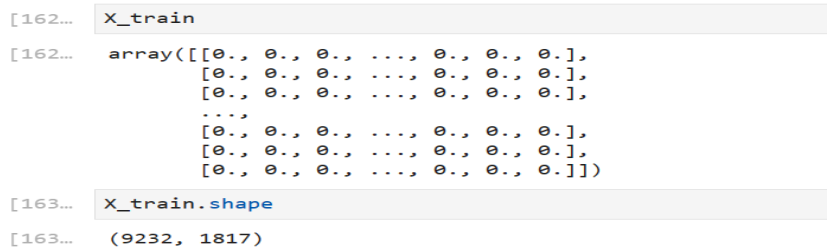
This code segment is responsible for text classification using a Random Forest Classifier

- **Data Splitting:**

    i. It uses the **`train_test_split`** function from **`sklearn`** to split the data into training and testing sets.
    ii. **`X_train`** and **`y_train`** will contain the features (tweets) and labels (sentiment labels) for training, respectively.
    iii. **`X_test`** and **`y_test`** will contain the features and labels for testing.
    iv. The data is split into training and testing sets with **an 80% training and 20% testing ratio.**
    v. **`random_state=42`** is used to ensure reproducibility.

- **Feature Extraction (TF-IDF Vectorization):**

    i. It uses the **`TfidfVectorizer`** from **`sklearn.feature_extraction.text`** to convert the text data (tweets) into numerical feature vectors.
    ii. **`max_features=2500`** specifies the maximum number of features to consider, which is limited to the top 2500 most important terms.
    iii. **`min_df=7`** specifies that a term must appear in at least 7 documents to be considered.
    iv. **`max_df=0.8`** specifies that a term should not appear in more than 80% of the documents to be considered.
    v. It transforms the text data into **TF-IDF vectors** using the **`fit_transform`** method for the training set (`X_train`) and `transform` for the testing set (`X_test`).
    vi. `X_train` and `X_test` are now represented **as arrays of TF-IDF features.**



vii. **X_train** is a variable that holds the transformed feature vectors for the training data.

viii. **X_train.shape** is a Python attribute that provides the dimensions (shape) of the X_train array.

- **Model Training (Random Forest Classifier):**

i. It uses the **`RandomForestClassifier`** from **`sklearn.ensemble`.**

ii. **`n_estimators=1000`** sets the number of decision trees in the random forest to 1000.

iii. **`random_state=0`** is used for reproducibility.

iv. The model is trained using the **TF-IDF** transformed training data **(`X_train`)** and their corresponding labels **(`y_train`).**

- **Classification Report:**

```
[167…
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
# model evaluation
def evaluate_model(y_test, y_pred):
    print('Classification Report:')
    print(classification_report(y_test, y_pred))

y_pred = classifier.predict(X_test)
evaluate_model(y_test, y_pred)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      1862
           1       0.83      0.59      0.69       447

    accuracy                           0.90      2309
   macro avg       0.87      0.78      0.81      2309
weighted avg       0.89      0.90      0.89      2309
```

i. It defines a function called **'evaluate_model'** that takes the true labels **'y_test'** and the predicted labels **'y_pred'** as inputs and prints a classification report, which includes various classification metrics such **as precision, recall, F1-score**, and support for each class.

ii. It uses the trained classifier **(Random Forest Classifier)** to predict labels for the testing data **(X_test)** and stores the predictions in **(y_pred).**

iii. Finally, it calls the **'evaluate_model'** function with**' y_test'** and **'y_pred'** to generate and display a classification report, providing insights into the model's performance on the test data.

- **Confusion Matrix:**

```
[168…  print('Confusion Matrix:')
       print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[1809   53]
 [ 184  263]]
```

i. **confusion_matrix(y_test, y_pred)** computes the confusion matrix by comparing the true labels **(y_test)** with the predicted labels **(y_pred)**.

ii. The confusion matrix provides a breakdown of correct and incorrect predictions, showing the number of **true positives, true negatives, false positives, and false negatives** for each class in the classification problem.

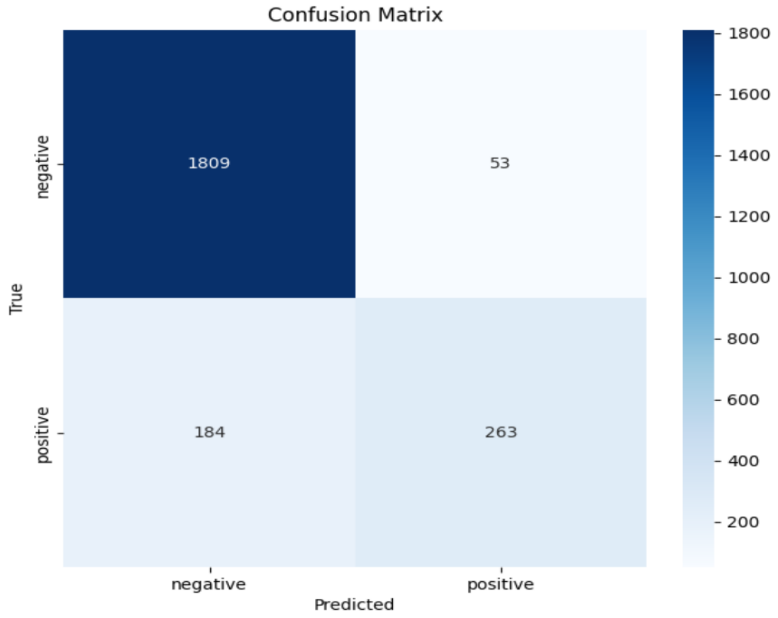
iii. It helps assess the model's performance, especially in terms of how well it correctly identifies different classes.

iv. Plotting confusion matrix:

It utilizes the **matplotlib** library to create a figure and axis for plotting, specifies the size, and sets the colormap to 'Blues' for the heatmap.

The **seaborn** library is used to plot a heatmap of the confusion matrix with annotations, displaying the true positive, true negative, false positive, and false negative values.

```
[170…
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    df_cm = pd.DataFrame(cm, index=['negative', 'positive'], columns=['negative', 'positive'])
    plt.figure(figsize=(7.5, 6.5))
    sns.heatmap(df_cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Make sure y_test and y_pred are correctly defined
plot_confusion_matrix(y_test, y_pred)
```



- **Accuracy:**

```
[169…
print('Accuracy Score:')
print(accuracy_score(y_test, y_pred))
```

```
Accuracy Score:
0.8973581637072325
```

i. **accuracy_score(y_test, y_pred)** calculates the accuracy by comparing the true labels (y_test) with the predicted labels (y_pred).

ii. The accuracy score is a single value that represents the proportion of correct predictions made by the model.

2. LSTM – Long Short-Term Memory:

2. Long Short-Term Memory (LSTM):

[21]:
```
# feature extraction
corpus = [df['cleaned_tweet'][i] for i in range( len(df))]

voc_size=5000

onehot_=[one_hot(words,voc_size)for words in corpus]

max_sent_length=max([len(i) for i in corpus])

embedded_docs=pad_sequences(onehot_,padding='pre',maxlen=max_sent_length)

embedding_vector_features=40
# defining the model
model=Sequential()
model.add(Embedding(voc_size,embedding_vector_features,input_length=max_sent_length))
model.add(Dropout(0.3))
model.add(LSTM(100))
model.add(Dropout(0.3))
model.add(Dense(1,activation='sigmoid'))
#compile the model by using binary_crossentropy for loss function , adam optimizer, accuracy metrics for monitoring
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

X_final=np.array(embedded_docs)
y_final=np.array(df['sentiment_label'])
X_final.shape,y_final.shape
#print model summary
model.summary()
```

This code explains text data preprocessing and deep learning model setup for sentiment analysis.

- **Feature Extraction:**

    i. It processes the text data from the **'cleaned_tweet'** column and stores it in the **`corpus`** list.
    ii. Defines a vocabulary size **(`voc_size`)** of 5000.
    iii. Utilizes one-hot encoding **(`one_hot`)** to convert each word in the corpus into a unique numerical representation.
    iv. Determines the maximum sentence length in the corpus.
    v. Pads and sequences the one-hot encoded vectors to make them uniform in length, with padding added to the beginning.

- **Embedding Layer Setup:**

    i. Specifies the number of **embedding vector features as 40**.
    ii. Defines a **Sequential model** for building a **neural network**.
    iii. Adds an **embedding layer** that transforms **one-hot encoded inputs into dense vectors**.
    iv. Introduces **dropout layers** to prevent overfitting.
    v. Adds an **LSTM layer** with 100 units for sequence processing.
    vi. Appends a **dense output layer** with a **sigmoid** activation function for binary classification.

- **Model Compilation:**

    i. Compiles the model using **binary cross-entropy** as the loss function, the **Adam optimizer**, and **accuracy** as a monitoring metric.

- **Data Preparation:**

    i. Converts the padded and sequenced embedded documents into **NumPy arrays.**
    ii. Prepares the **target labels**.

- **Model Summary:**

    i. Prints a **summary** of the neural network model architecture.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 113, 40)           200000

 dropout (Dropout)           (None, 113, 40)           0

 lstm (LSTM)                 (None, 100)               56400

 dropout_1 (Dropout)         (None, 100)               0

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 256501 (1001.96 KB)
Trainable params: 256501 (1001.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Data Splitting:**

    i. Splits the preprocessed data **(`X_final` and `y_final`)** into training and testing sets using the **`train_test_split`** function.
    ii. 67% of the data is used for training **(`X_train` and `y_train`),** and 33% is used for testing **(`X_test` and `y_test`).**
    iii. It sets a random seed **(`random_state=42`)** for reproducibility.

- **Model Training:**

    i. Trains the previously defined LSTM-based deep learning model **(`model`)** on the training data **(`X_train`, `y_train`)**.
    ii. Validates the model on the testing data **(`X_test`, `y_test`)**.
    iii. Specifies training parameters, including the number of training **epochs (10)** and **batch size (64)**.

```
[22]: # Splitting the data into training and testing sets
      X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.33, random_state=42)
      lstm=model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64)
```

```
Epoch 1/10
121/121 [==============================] - 14s 97ms/step - loss: 0.4667 - accuracy: 0.8048 - val_loss: 0.3371 - val_accurac
y: 0.8480
Epoch 2/10
121/121 [==============================] - 11s 95ms/step - loss: 0.2721 - accuracy: 0.8924 - val_loss: 0.2780 - val_accurac
y: 0.8868
Epoch 3/10
121/121 [==============================] - 12s 101ms/step - loss: 0.2028 - accuracy: 0.9189 - val_loss: 0.2801 - val_accurac
y: 0.8876
Epoch 4/10
121/121 [==============================] - 11s 93ms/step - loss: 0.1738 - accuracy: 0.9335 - val_loss: 0.2955 - val_accurac
y: 0.8868
Epoch 5/10
121/121 [==============================] - 11s 93ms/step - loss: 0.1534 - accuracy: 0.9426 - val_loss: 0.3214 - val_accurac
y: 0.8719
Epoch 6/10
121/121 [==============================] - 12s 100ms/step - loss: 0.1350 - accuracy: 0.9518 - val_loss: 0.3319 - val_accurac
y: 0.8816
Epoch 7/10
121/121 [==============================] - 12s 99ms/step - loss: 0.1191 - accuracy: 0.9554 - val_loss: 0.3531 - val_accurac
y: 0.8729
Epoch 8/10
121/121 [==============================] - 11s 93ms/step - loss: 0.1054 - accuracy: 0.9625 - val_loss: 0.3833 - val_accurac
y: 0.8761
Epoch 9/10
121/121 [==============================] - 11s 94ms/step - loss: 0.0977 - accuracy: 0.9651 - val_loss: 0.4416 - val_accurac
y: 0.8672
Epoch 10/10
121/121 [==============================] - 12s 96ms/step - loss: 0.0869 - accuracy: 0.9684 - val_loss: 0.4358 - val_accurac
y: 0.8719
```

- **Model Prediction:**

    i. **`model.predict(X_test)`** and **`model.predict(X_train)`** use the trained model to make predictions on the test and training data, respectively.
    ii. **`np.argmax`** is applied to get the indices of the highest predicted values along axis 1 (across the classes), effectively converting the model's output probabilities into class labels.
    iii. **`y_test_pred`** stores the predicted class labels for the test data.
    iv. **`y_train_pred`** stores the predicted class labels for the training data.

```
[23]: y_test_pred=np.argmax(model.predict(X_test),axis=1)
      y_train_pred=np.argmax(model.predict(X_train),axis=1)
```

```
120/120 [==============================] - 2s 15ms/step
242/242 [==============================] - 4s 15ms/step
```

- **Classification Report:**

    i. This prints a classification report that provides various classification metrics for model evaluation.
    ii. It compares the true class labels **(`y_test`)** with the predicted class labels **(`y_pred_classes`)** and presents metrics like **precision, recall, F1-score**, and support for each class.
    iii. This report gives insights into the model's performance in classifying test data.

```
[26]: print('Classification Report:')
      print(classification_report(y_test,y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      3037
           1       0.00      0.00      0.00       772

    accuracy                           0.80      3809
   macro avg       0.40      0.50      0.44      3809
weighted avg       0.64      0.80      0.71      3809
```

- **Confusion Matrix:**

[27]:
```
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_classes))
```

```
Confusion Matrix:
[[3037    0]
 [ 772    0]]
```

i. This prints a **confusion matrix**, which is a table that summarizes the model's classification performance.

ii. It compares the true class labels **(`y_test`)** with the predicted class labels **(`y_pred_classes`)**.

iii. The matrix shows the number of **true positives, true negatives, false positives, and false negatives**, providing insights into the model's accuracy and error patterns.

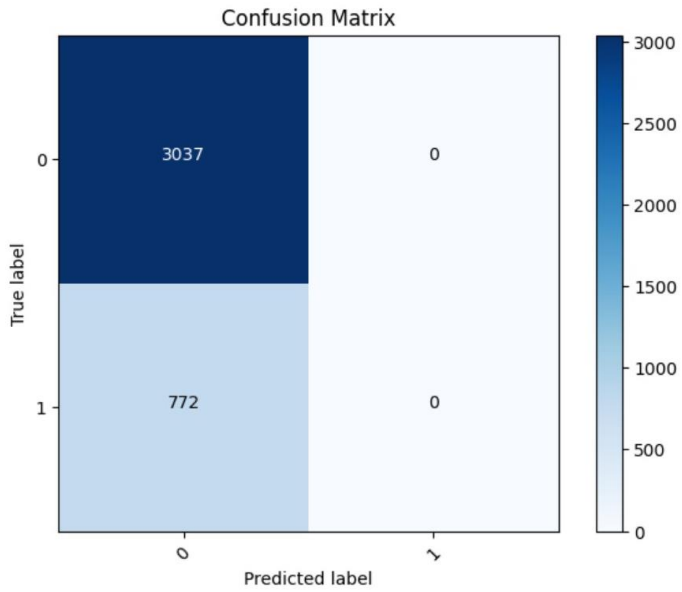
iv. **Plotting confusion matrix:**

This defines a function `**plot_confusion_matrix**` to create and display a confusion matrix plot of a model's predictions on the test data.

`confusion_mtx` calculates the confusion matrix between the true and predicted labels.

[25]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion Matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred,axis = 1)
confusion_mtx = confusion_matrix(y_test, y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes = range(2))
```

```
120/120 [==============================] - 2s 15ms/step
```



- **Accuracy:**

```
[28]: #model evalution
score = model.evaluate(X_test, y_test, verbose=False)
print("Accuracy Score:")
print(score[1])
```

```
Accuracy Score:
0.8718823790550232
```

`model.evaluate(X_test, y_test, verbose=False)'** accuracy and **`score[1]` extracts and prints the accuracy score, which represents the proportion of correct predictions made by the model on the test data.

3. Support Vector Machine (SVM):

This code segment is responsible for text classification using SVM model

- **Data Splitting:**

3. Support Vector Machine (SVM):

```
[175…] # Splitting the data into training and testing sets
x = df.cleaned_tweet
y = df.sentiment_label
x_train, x_test, Y_train, Y_test = train_test_split(x, y, random_state=42)
print(len(x_train), len(y_train))
print(len(x_test), len(y_test))
```

```
8655 9232
2886 2309
```

i. `x` and `y` represent the features (tweets) and labels (sentiment labels) from a DataFrame.
ii. **`train_test_split`** is used to randomly divide the data into training and testing sets.
iii. The resulting sets are **`x_train`**, **`Y_train`** for training and **`x_test`**, **`Y_test`** for testing.
iv. The **`print`** statements display the lengths of the training and testing sets to confirm the data split.

- **Feature Extraction:**

```
[176...
from sklearn.feature_extraction.text import CountVectorizer
# feature extraction
# instantiate the vectorizer
vect = CountVectorizer()
vect.fit(x_train)
```

```
[176...
▾ CountVectorizer
CountVectorizer()
```

i. It imports **`CountVectorizer`** from the **`sklearn.feature_extraction.text`** module, which is used to convert text data into numerical feature vectors.
ii. It creates an instance of **`CountVectorizer`** called **`vect`.**
iii. **`vect.fit(x_train)`** fits the vectorizer to the training text data **(`x_train`)**.
iv. This step prepares the vectorizer to transform text into numerical features based on the vocabulary observed in the training data.

- **Data Preparation:**

```
[177...
# Use the trained to create a document-term matrix from train and test sets
x_train_dtm = vect.transform(x_train)
x_test_dtm = vect.transform(x_test)
```

```
[178...
vect_tunned = CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=0.1, max_df=0.7, max_features=100)
vect_tunned
```

```
[178...
▾ CountVectorizer
CountVectorizer(max_df=0.7, max_features=100, min_df=0.1, ngram_range=(1, 2),
                stop_words='english')
```

i. This code uses the trained **`CountVectorizer`** to convert the text data into a **document-term matrix (DTM)** for both the training and testing sets.
ii. **`x_train_dtm`** and **`x_test_dtm`** store the DTM representations of the text data from the training and testing sets, respectively.
iii. **`vect.transform(x_train)`** and **`vect.transform(x_test)`** apply the vectorizer to transform the text data into numerical feature vectors based on the learned vocabulary from the training data.
iv. It sets the minimum document frequency to 10% of the documents **(`min_df=0.1`)**, meaning words that appear in less than 10% of the documents will be excluded.
v. It sets the maximum document frequency to 70% of the documents **(`max_df=0.7`)**, meaning words that appear in more than 70% of the documents will be excluded.
vi. It limits the number of features to the top 100 most important words or word combinations **(`max_features=100`)**.
vii. **`vect_tuned`** is configured to perform text feature extraction based on these settings.

- **Model Training:**

[179…

```
#training SVM model with linear kernel
#Support Vector Classification-wrapper around SVM
from sklearn.svm import SVC
model = SVC(kernel='linear', random_state = 10)
model.fit(x_train_dtm, Y_train)
#predicting output for test data
# model evaluation
pred = model.predict(x_test_dtm)
```

i. This code trains a **Support Vector Machine (SVM) model** for text classification with a **linear kernel**
ii. It imports the **`SVC`** **(**Support Vector Classification) class from **`sklearn.svm`.**
iii. A linear kernel **(`kernel='linear'`)** is chosen for the SVM.

iv. The model is initialized with a **random state** for reproducibility.
v. The training data in DTM format **(`x_train_dtm`)** and their corresponding labels **(`Y_train`)** are used to train the SVM model.
vi. The model is used to predict the labels for the test data **(`x_test_dtm`)** and store the predictions in the **`pred`** variable.

- **Classification Report:**

```
[180…] print("Classification Report:")
       print(classification_report(Y_test,pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      2323
           1       0.70      0.70      0.70       563

    accuracy                           0.88      2886
   macro avg       0.81      0.81      0.81      2886
weighted avg       0.88      0.88      0.88      2886
```

i. This code prints a **classification report** for evaluating the SVM model's performance:
ii. **`classification_report(Y_test, pred)`** compares the true labels **(`Y_test`)** with the predicted labels **(`pred`)** and generates a report.
iii. The report includes metrics like **precision, recall, F1-score**, and support for each class.
iv. It provides insights into the model's classification performance on the test data.

- **Confusion Matrix:**

```
[181…] #building confusion matrix
       print("Confusion Matrix:")
       cm = confusion_matrix(Y_test, pred)
       cm
```

```
Confusion Matrix:
[181… array([[2154,  169],
            [ 171,  392]], dtype=int64)
```

i. This code prints a **confusion matrix**, which is a table used to evaluate the model's classification performance:

ii. **`confusion_matrix(Y_test, pred)`** calculates the confusion matrix by comparing the true labels **(`Y_test`)** with the predicted labels **(`pred`)**.

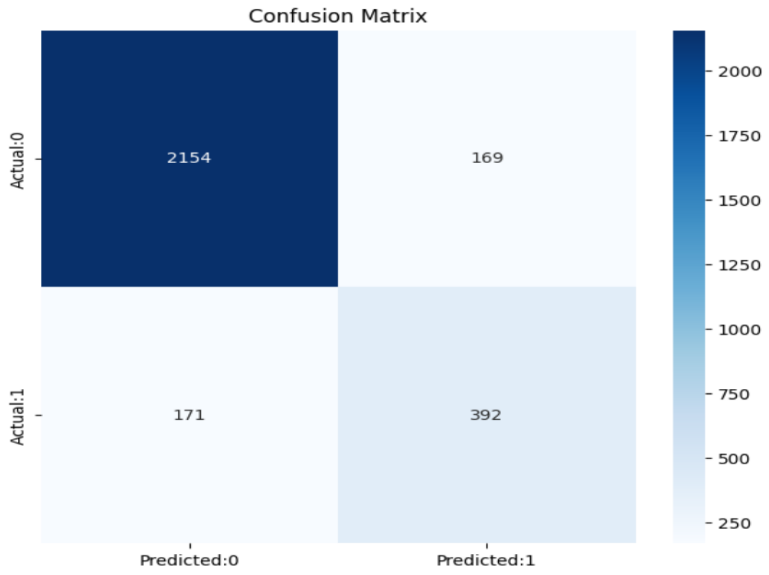
iii. The matrix shows the number of **true positives, true negatives, false positives, and false negatives** for each class, providing insights into the model's accuracy and error patterns on the test data.

iv. Plotting confusion matrix:

`conf_matrix` converts the confusion matrix (`cm`) into a DataFrame for easier manipulation.

`sns.heatmap` creates a heatmap plot of the confusion matrix with annotations for visualizing the SVM model's performance.

```
[183…
#defining the size of the canvas
plt.rcParams['figure.figsize'] = [7.5,6.5]
#confusion matrix to DataFrame
conf_matrix = pd.DataFrame(data = cm,columns = ['Predicted:0','Predicted:1',], index = ['Actual:0','Actual:1',])
#plotting the confusion matrix
sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues')
plt.title ('Confusion Matrix', fontsize=12)
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10)
plt.show()
```



- **Accuracy:**

```
[182…  #accuracy score
       print("Accuracy Score:")
       accuracy_score(Y_test,pred)

       Accuracy Score:
[182…  0.8821898821898821
```

This code calculates and prints the **accuracy score,** which represents the proportion of correct predictions made by the **SVM model** on the test data.

Output Prediction:

- This code allows a user to input text and then uses the model to make a sentiment prediction:
- It defines a function **`transform_text`** to preprocess the user's input text:

    i. It converts the text to lowercase.
    ii. Tokenizes the text into words.
    iii. Removes special characters.
    iv. Eliminates stop words and punctuation.
    v. Performs stemming to reduce words to their root forms.

- The user is prompted to input text.
- The input text is preprocessed using the **`transform_text`** function.
- The preprocessed text is vectorized using the same **`CountVectorizer` (`vect`)** used during model training.
- The model is then used to **predict the sentiment** (positive, neutral, or negative) of the **user's input**.
- Depending on the **prediction probability**, it **prints 'Positive,' 'Neutral,' or 'Negative'** as the sentiment label.
- In summary, the code preprocesses and classifies user-input text into sentiment categories based on a pre-trained model.

Output Prediction :

[218…

```
# Use the model on user input
from nltk.stem.porter import PorterStemmer
ps=PorterStemmer()
def transform_text(text):
    # Convert the message to lower case
    text=text.lower()

    # Tokenize the message
    text=nltk.word_tokenize(text)

    # Remove the special characters in the message
    y=[]
    for i in text:
        if i.isalnum():
            y.append(i)
    text=y[:]
    y.clear()

    # Remove the stop words and punctuations in the message
    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    text=y[:]
    y.clear()

    # Stemming
    for i in text:
        y.append(ps.stem(i))

    return " ".join(y)

user_text = input('Input the text: ')
preprocessed_text=transform_text(user_text)
vectorized_text=vect.transform([preprocessed_text]).toarray()

pred = model.predict(vectorized_text)
if pred > 0.8:
    print('Positive')
elif pred > 0.5:
    print('Neutral')
else:
    print('Negative')
```

Positive Label:

Input the text: @VirginAmerica it was amazing, and arrived an hour early. You're too good to me.
Positive

Negative Label:

Input the text: @VirginAmerica I can't check in or add a bag. Your website isn't working. I've tried both desktop and mob
ile http://t.co/AvyqdMpi1Y
Negative

Comparison of Models:

MODEL	ACCURACY
RANDOM FOREST	**0.8973581637072325**
LSTM – LONG SHORT-TERM MEMORY	**0.8718823790550232**
SVM – SUPPORT VECTOR MACHINE	**0.8821898821898821**

In summary, the table shows the **comparative accuracy** of different machine learning models on a sentiment analysis for marketing, with the **Random Forest Model** having the **highest accuracy**, **followed** by **Support Vector Machine (SVM) Model** and **Long Short-Term Memory (LSTM) Model.**