**Sentiment Analysis for Marketing: Loading and Preprocessing the dataset (Part 1)**

Author: **ANEESH SHAJI**

Reg no.: 961621104019

****

**Introduction: Sentiment Analysis using Machine Learning**

Sentiment analysis is the process of identifying and extracting opinions and emotions from text. It is a powerful tool that can be used for a variety of purposes, including marketing. By understanding how customers feel about their brand, products, and services, businesses can tailor their marketing efforts to better meet customer needs and wants.

AI can be used to improve the accuracy and efficiency of sentiment analysis. For example, AI can be used to fine-tune pre-trained sentiment analysis models, such as BERT and RoBERTa as done in previous phase. This can help the models to better understand the context of customer reviews and social media posts, and to produce more accurate sentiment predictions.

**Dataset:**

Dataset link: [*https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment*](https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment)

The Twitter Airline Sentiment dataset contains the following columns:

* airline: The name of the airline.
* text: The text of the tweet.
* sentiment: The sentiment of the tweet (positive, negative, or neutral).



In this Phase, to begin building our project on sentiment analysis for marketing using AI, we need to load and preprocess the dataset.

**Loading the Dataset**

The dataset that we will use is the Twitter Airline Sentiment dataset from Kaggle. This dataset contains over 14,000 tweets that have been labeled with the sentiment (positive, negative, or neutral).

To load the dataset, we can use the following Python code:

import pandas as pd

# Load the dataset from the CSV file

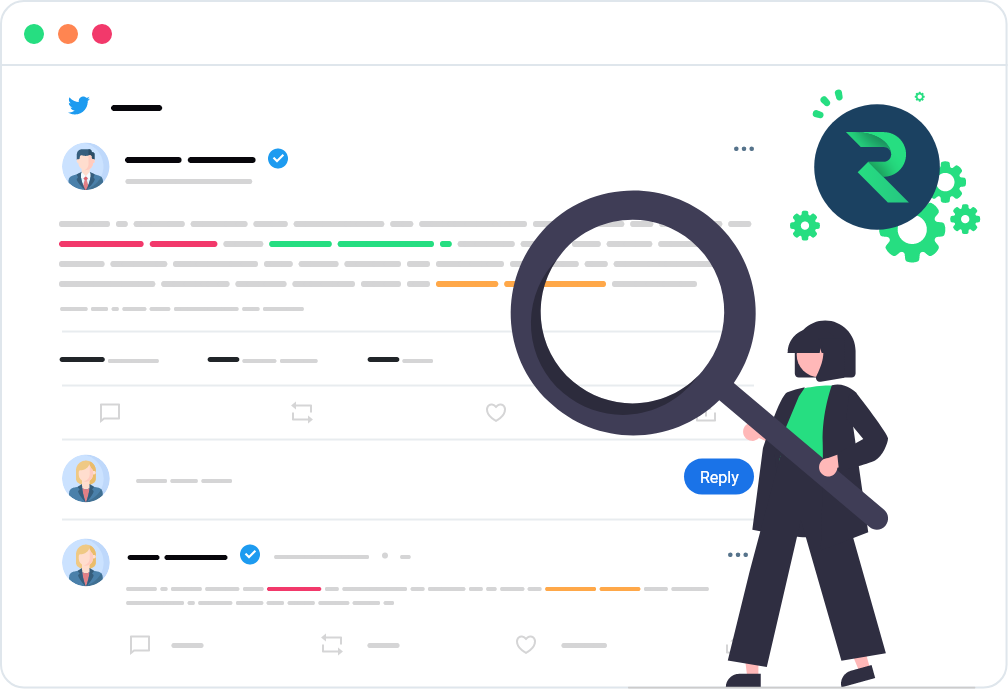
df = pd.read\_csv('airline\_sentiment.csv')

# Print the first few rows of the dataset

print(df.head())

*This will print the first few rows of the dataset to the console. You can then use this data to train your sentiment analysis model.*

**Preprocessing the Dataset**



Data preprocessing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, greater is the reliance on the produced results. **Incomplete, noisy, and inconsistent data** are the properties of large real-world datasets. Data preprocessing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise and resolving inconsistencies.

* **Incomplete data** can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Relevant data may not be recorded due to a misunderstanding, or because of equipment malfunctions.
* There are many possible reasons for **noisy data** (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Errors in data transmission can also occur. Incorrect data may also result from inconsistencies in naming conventions or data codes used, or inconsistent formats for input fields, such as date.

There are a number of data preprocessing techniques available such as:

1. **Data cleaning** can be applied to filling in missing values, remove noise, resolving inconsistencies, identifying and removing outliers in the data.
2. **Data integration** merges data from multiple sources into a coherent data store, such as a data warehouse.
3. **Data transformations**, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements.
4. **Data reduction** can reduce the data size by eliminating redundant features, or clustering, for instance.

Let us import them now as well as a few other libraries we will be using.

import nltk

from nltk.corpus import twitter\_samples

import matplotlib.pyplot as plt

import random

Now let's download the dataset.

import kaggle

# Download the dataset

kaggle.datasets.download('crowdflower/twitter-airline-sentiment')

# Extract the dataset from the ZIP file

with zipfile.ZipFile('twitter-airline-sentiment.zip', 'r') as zip\_ref:

zip\_ref.extractall()

Now let us load the positive and negative examples separately and then we will have a detalied look at them.

all\_positive\_tweets = twitter\_samples.strings('positive\_tweets.json')

all\_negative\_tweets = twitter\_samples.strings('negative\_tweets.json')

print('Number of positive tweets: ', len(all\_positive\_tweets))

print('Number of negative tweets: ', len(all\_negative\_tweets))

print('**\n**The type of all\_positive\_tweets is: ', type(all\_positive\_tweets))

print('The type of a tweet entry is: ', type(all\_negative\_tweets[0]))

Number of positive tweets: 5000

Number of negative tweets: 5000

The type of all\_positive\_tweets is: <class 'list'>

The type of a tweet entry is: <class 'str'>

We observe that the tweet dataset are two list of strings. The individual s trings contains tweeter handles, punctuations, emoticons, urls etc. We need to do some good preprocessing to work with them. Just before that, let's check their comparative lengths.

total\_positive\_words = []

for sentence **in** all\_positive\_tweets:

total\_positive\_words.append(sentence.count(' '))

total\_negative\_words = []

for sentence **in** all\_negative\_tweets:

total\_negative\_words.append(sentence.count(' '))

import plotly.graph\_objects as go

import numpy as np

x0 = np.array(total\_positive\_words)

x1 = np.array(total\_negative\_words)

fig = go.Figure()

fig.add\_trace(go.Histogram(x=x1, name = 'Negative'))

fig.add\_trace(go.Histogram(x=x0, name = 'Positive'))

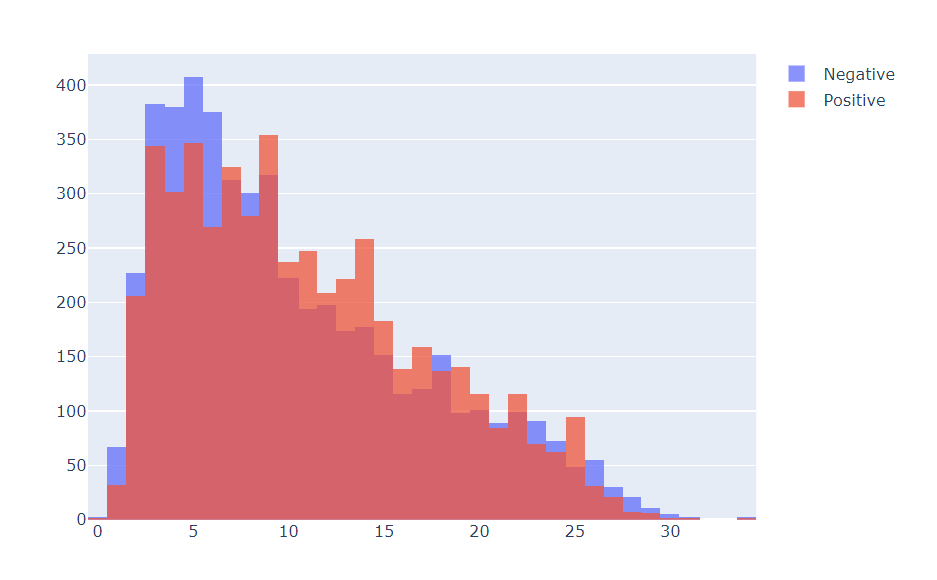
*# Overlay both histograms*

fig.update\_layout(barmode='overlay')

*# Reduce opacity to see both histograms*

fig.update\_traces(opacity=0.75)

fig.show()



Data preprocessing is one of the critical steps in any machine learning project. It includes cleaning and formatting the data before feeding into a machine learning algorithm. For NLP, the preprocessing steps are comprised of the following tasks:

* Tokenizing the string
* Lowercasing
* Removing stop words and punctuation
* Stemming

Let us select any sample from the data given and we will successively work our way up to the higher level of processing and download the stopwords for the NLTK library.

tweet = all\_positive\_tweets[1455]

print(tweet)

nltk.download('stopwords')

import re

import string

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.tokenize import TweetTokenizer

### **Remove hyperlinks, Twitter marks and styles**

Since we have a Twitter dataset, we'd like to remove some substrings commonly used on the platform like the hashtag, retweet marks, and hyperlinks. We'll use the [re](https://docs.python.org/3/library/re.html) library to perform regular expression operations on our tweet. We'll define our search pattern and use the sub() method to remove matches by substituting with an empty character (i.e. '')

print('Original Tweet: ')

print(tweet)

*# it will remove the old style retweet text "RT"*

tweet2 = re.sub(r'^RT[\s]+', '', tweet)

*# it will remove hyperlinks*

tweet2 = re.sub(r'https?:\/\/.\*[\r\n]\*', '', tweet2)

*# it will remove hashtags. We have to be careful here not to remove*

*# the whole hashtag because text of hashtags contains huge information.*

*# only removing the hash # sign from the word*

tweet2 = re.sub(r'#', '', tweet2)

*# it will remove single numeric terms in the tweet.*

tweet2 = re.sub(r'[0-9]', '', tweet2)

print('**\n**After removing old style tweet, hyperlinks and # sign')

print(tweet2)

Original Tweet:

Stats for the day have arrived. 1 new follower and NO unfollowers :) via http://t.co/0s8GQYOeus.

After removing old style tweet, hyperlinks and # sign

Stats for the day have arrived. new follower and NO unfollowers :) via

### **Tokenize the string**

To tokenize means to split the strings into individual words without blanks or tabs. In this same step, we will also convert each word in the string to lower case. The [tokenize](https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual) module from NLTK allows us to do these easily:

print('Before Tokenizing: ')

print(tweet2)

*# instantiate the tokenizer class*

tokenizer = TweetTokenizer(preserve\_case=False,

strip\_handles=True,

reduce\_len=True)

*# tokenize the tweets*

tweet\_tokens = tokenizer.tokenize(tweet2)

print('**\n**Tokenized string:')

print(tweet\_tokens)

Before Tokenizing:

Stats for the day have arrived. new follower and NO unfollowers :) via

Tokenized string:

['stats', 'for', 'the', 'day', 'have', 'arrived', '.', 'new', 'follower', 'and', 'no', 'unfollowers', ':)', 'via']

### **Remove stop words and punctuations**

The next step is to remove stop words and punctuation. Stop words are words that don't add significant meaning to the text. You'll see the list provided by NLTK when you run the cells below.

*#Import the english stop words list from NLTK*

stopwords\_english = stopwords.words('english')

print('Stop words**\n**')

print(stopwords\_english)

print('**\n**Punctuation**\n**')

print(string.punctuation)

Stop words

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

Punctuation

!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~

### **Observation regarding Stop words and Punctuations**

* We can see that the stop words list above contains some words that could be important in some contexts. These could be words like i, not, between, because, won, against. You might need to customize the stop words list for some applications. For our exercise, we will use the entire list.
* For the punctuation, we saw earlier that certain groupings like ':)' and '...' should be retained when dealing with tweets because they are used to express emotions. In other contexts, like medical analysis, these should also be removed.

print('Before tokenization')

print(tweet\_tokens)

tweets\_clean = []

for word **in** tweet\_tokens: *# Go through every word in your tokens list*

if (word **not** **in** stopwords\_english **and** *# remove stopwords*

word **not** **in** string.punctuation): *# remove punctuation*

tweets\_clean.append(word)

print('**\n\n**After removing stop words and punctuation:')

print(tweets\_clean)

Before tokenization

['stats', 'for', 'the', 'day', 'have', 'arrived', '.', 'new', 'follower', 'and', 'no', 'unfollowers', ':)', 'via']

After removing stop words and punctuation:

['stats', 'day', 'arrived', 'new', 'follower', 'unfollowers', ':)', 'via']

### **Stemming**

Stemming is the process of converting a word to its most general form, or stem. This helps in reducing the size of our vocabulary.

NLTK has different modules for stemming and we will be using the [PorterStemmer](https://www.nltk.org/api/nltk.stem.html" \l "module-nltk.stem.porter) module which uses the [Porter Stemming Algorithm](https://tartarus.org/martin/PorterStemmer/). Let's see how we can use it in the cell below.

*# Instantiate stemming class*

stemmer = PorterStemmer()

*# Create an empty list to store the stems*

tweets\_stem = []

for word **in** tweets\_clean:

stem\_word = stemmer.stem(word) *# stemming word*

tweets\_stem.append(stem\_word) *# append to the list*

print('Words after stemming: ')

print(tweets\_stem)

Words after stemming:

['stat', 'day', 'arriv', 'new', 'follow', 'unfollow', ':)', 'via']

**Conclusion**

To conclude, in this phase of our project on sentiment analysis for marketing using AI, we have loaded and preprocessed the Twitter Airline Sentiment dataset. We have loaded the dataset using the Python Pandas library and preprocessed it by removing stop words, stemming the words, and converting the text data to a numerical representation using word embedding.

In the next phase, we will split the preprocessed data into training and test sets. We will then use the training set to train our sentiment analysis model and the test set to evaluate the model's performance.