CHAPTER 1: IMPORTING DATA FROM TWITTER API

Theory

Text mining is the application of natural language processing techniques and analytical methods to text data to derive relevant information. Text mining is getting a lot of attention these years, due to the exponential increase in digital text data from web pages, Google's projects as google books and google n-gram, and social media services such as Twitter. Twitter data constitutes a rich source that can be used for capturing information about any topic imaginable. This data can be used in different use cases such as finding trends related to a specific keyword, measuring brand sentiment, and gathering feedback about new products and services.

What is the Twitter Sentiment Analysis?

Sentiment Analysis is a technique used in text mining. It may, therefore, be described as a text mining technique for analyzing the underlying sentiment of a text message, i.e., a tweet. Twitter sentiment or opinion expressed through it may be positive, negative, or neutral. However, no algorithm can give you 100% accuracy or prediction on sentiment analysis.

As a part of NLP, algorithms like SVM, Naïve Bayes are used in predicting the polarity of the sentence. Sentiment analysis of Twitter data may also depend upon sentence level and document level. Methods like positive and negative words to find on the sentence are however inappropriate because the flavor of the text block depends a lot on the context. This may be done by looking at the POS (Parts of Speech) Tagging.

Why Twitter Sentiment Analysis?

Sentiment Analysis Dataset Twitter has several applications:

Business: Companies use Twitter Sentiment Analysis to develop their business strategies, to assess customer's feelings towards products or brands, how people respond to their campaigns or product launches and why consumers are not buying certain products.

Politics: In politics Sentiment Analysis Dataset Twitter is used to keep track of political views, to detect consistency and inconsistency between statements and actions at the government level. Sentiment Analysis Dataset Twitter is also used for analyzing election results.

Public Actions: Twitter Sentiment Analysis also is used for monitoring and analyzing social phenomena, predicting potentially dangerous situations, and determining the general mood of the blogosphere.

Getting Data from Twitter Streaming API

API stands for an Application Programming Interface. It is a tool that makes interaction with computer programs and web services easy. Many web services provide APIs to developers to interact with their services and to access data in a programmatic way.

Get access to the Twitter API

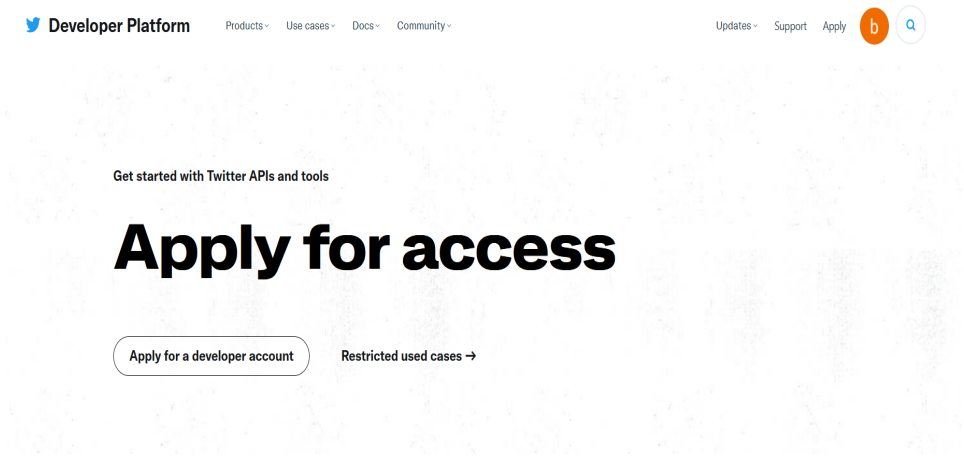
To access Twitter Streaming API, we need to get pieces of information from Twitter: API Key, API secret, Access token, and Access token secret.

Follow the steps below to get all the 4 elements:

1. Apply for a developer account with Twitter

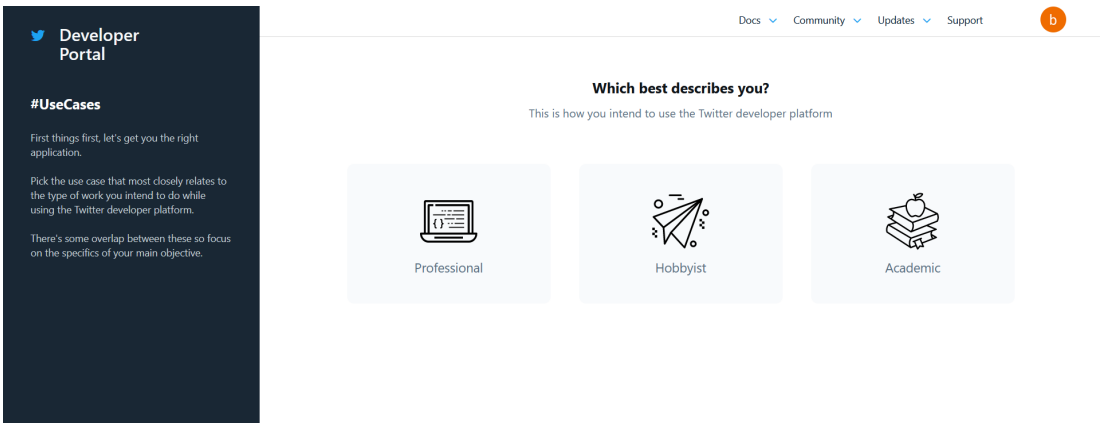
To apply for a developer account with Twitter –

- Create a Twitter account if you do not already have one.
- Go to https://developer.twitter.com/en/apply-for-access and follow the below step-by-step guide to accessing the Twitter developer account. Once you click the above link, you will be directed to the Twitter developer web page. You need to log in with your Twitter credentials.

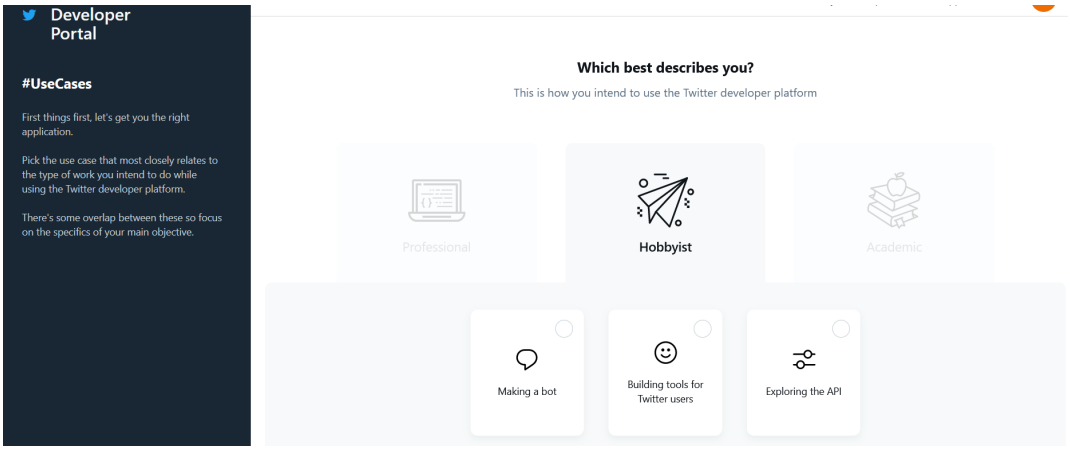


Above is the screenshot of the Twitter Developer web page.

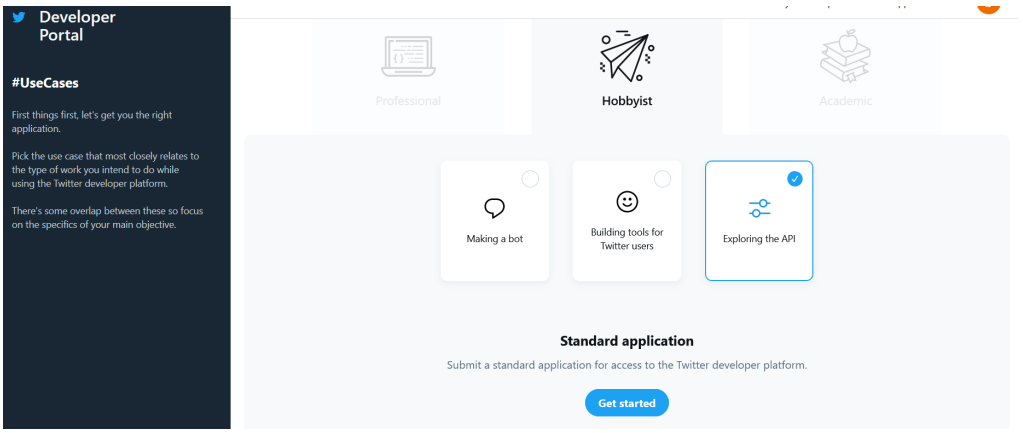
We need to apply for a developer account by clicking on the “Apply for a developer account” button seen on the web page.



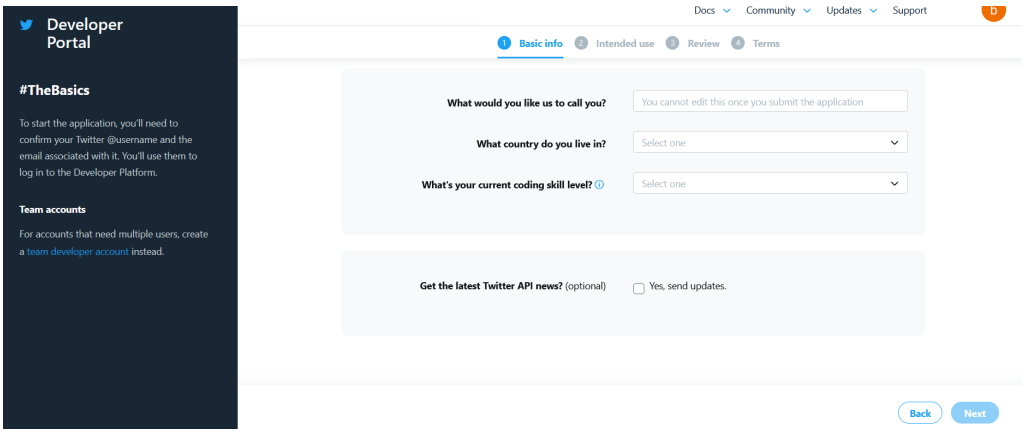
We need to choose any of the above options depending upon your requirement. To demonstrate my purpose I will select **Hobbyist**. Once you select your option, next we need to select the reason for requesting and Twitter API access.



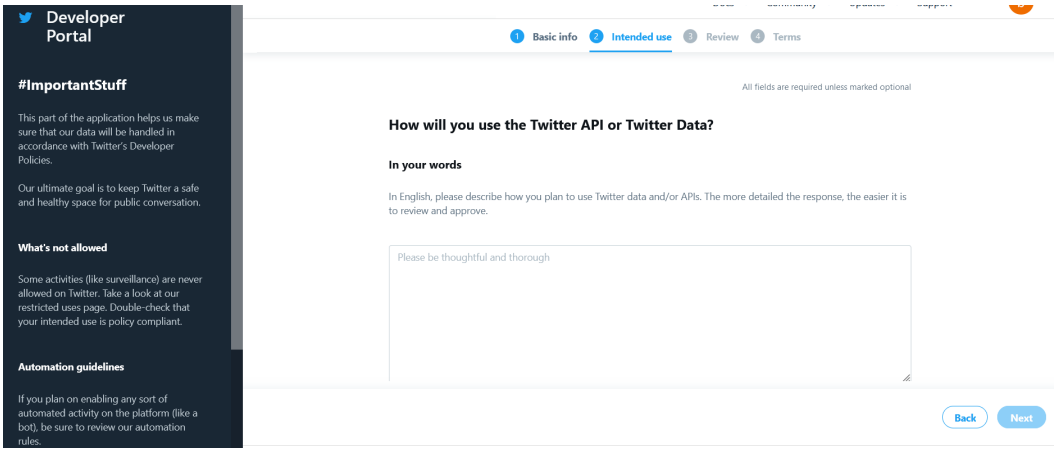
Select **Exploring API** and click on **Get started**.



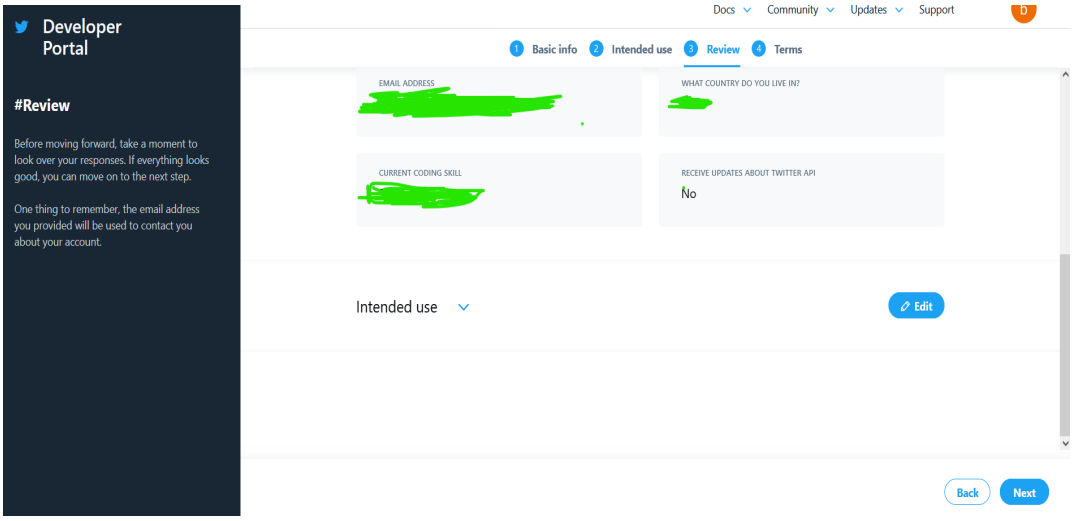
Then you will be navigated to the next page.



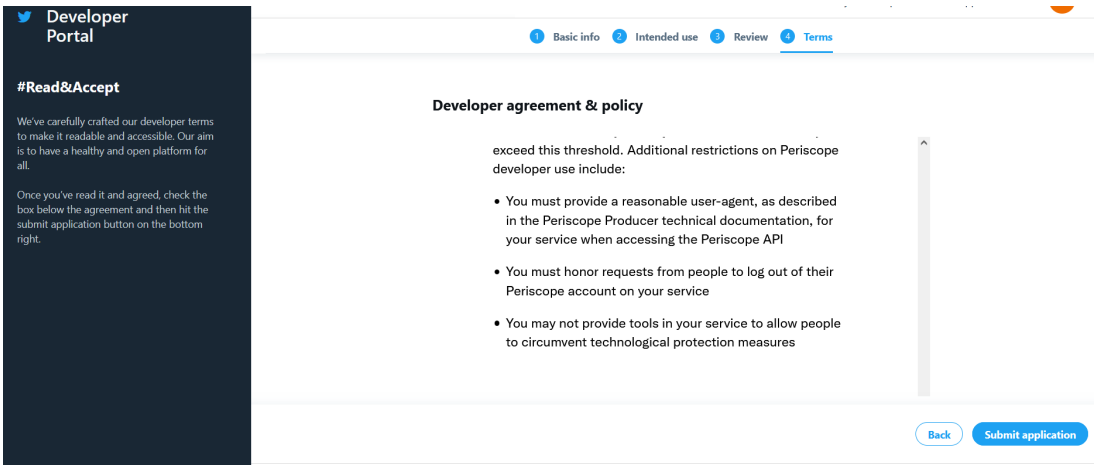
Fill in all the **Basic info** and click **Next**.



Fill in all the **Intended use** fields and click **Next**.

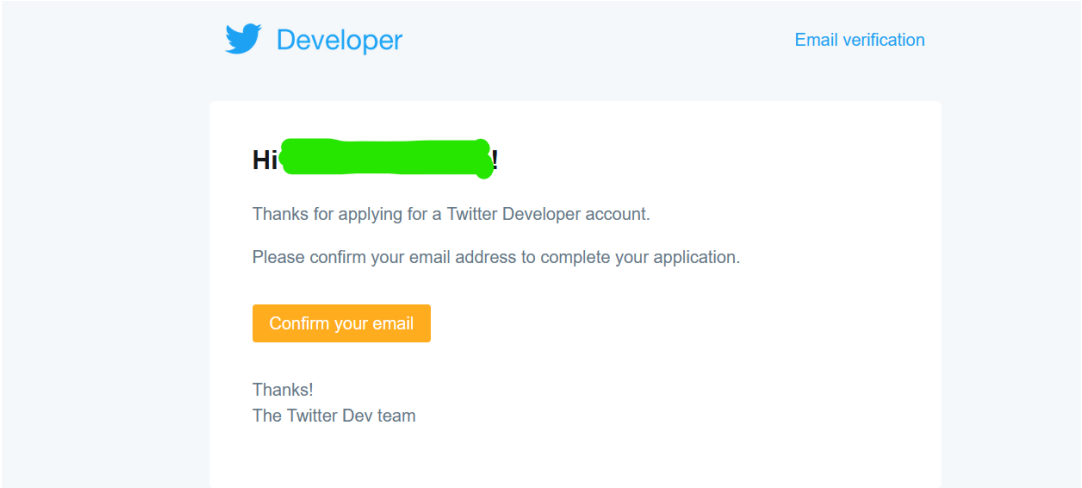


Review all the information filled in and then click **Next**.



Agree on the Developer agreement & policy and then submit your application.

Once you submit your application, You will receive an email from Twitter Developer for **Email Verification**.

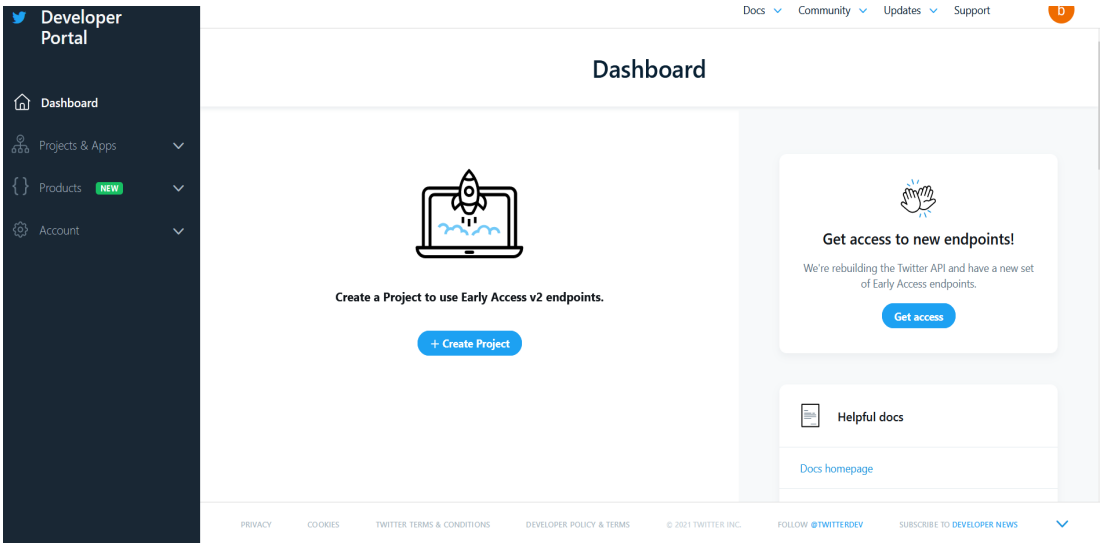


Once you confirm your email, your application will be reviewed and provided access for Twitter Developer Account.

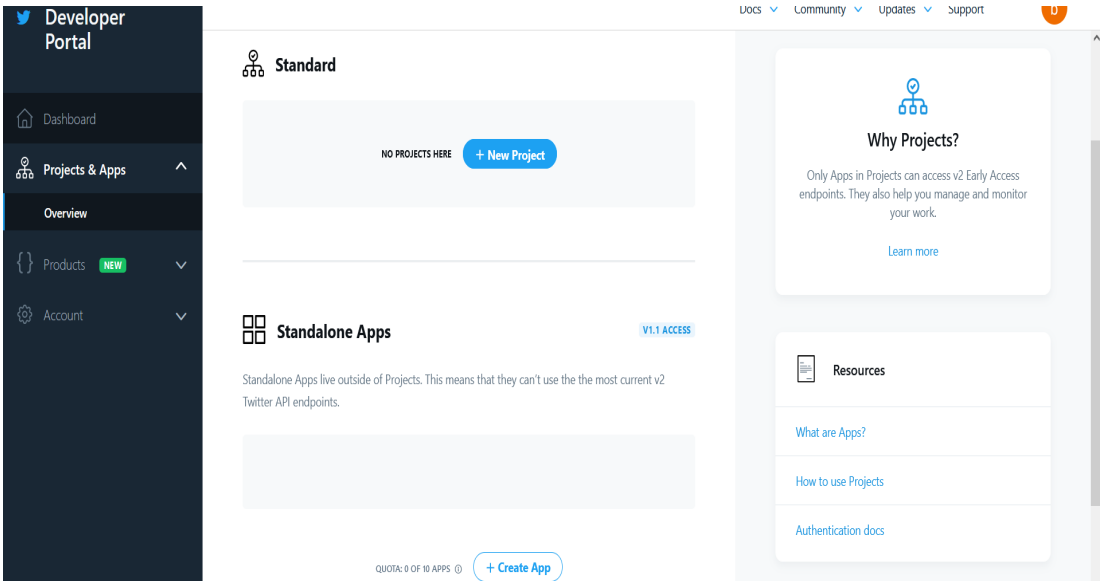
2. Get Your Twitter API Key and Access Token

Generally, if Twitter doesn't find anything off with your application, You'd be able to access your developer account immediately after completing your application process. Now, to get your API Key and Access Token follow the steps –

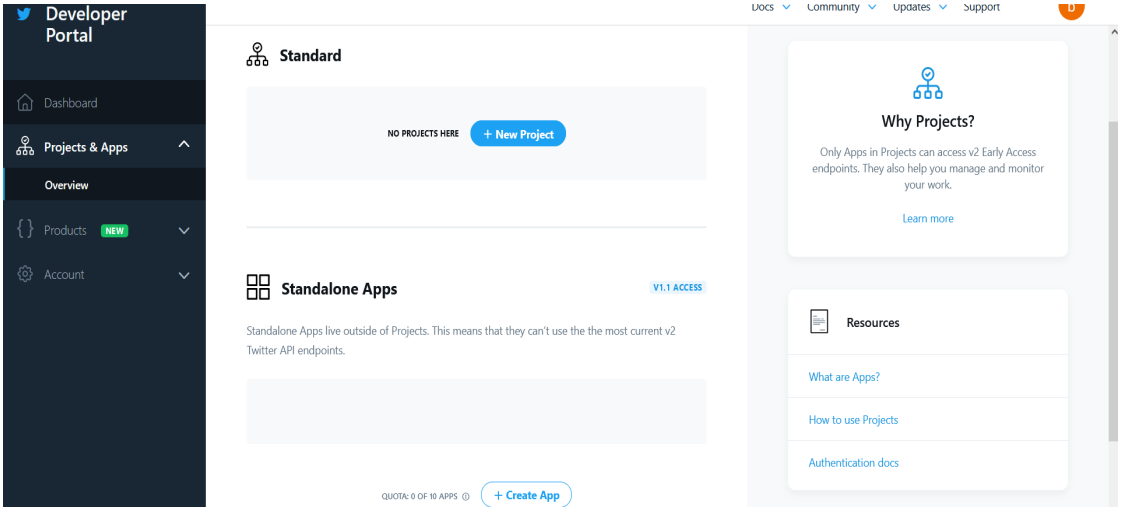
- Log in to the Twitter Developer account and you will see the below Dashboard screen once you get access to the Developer account.



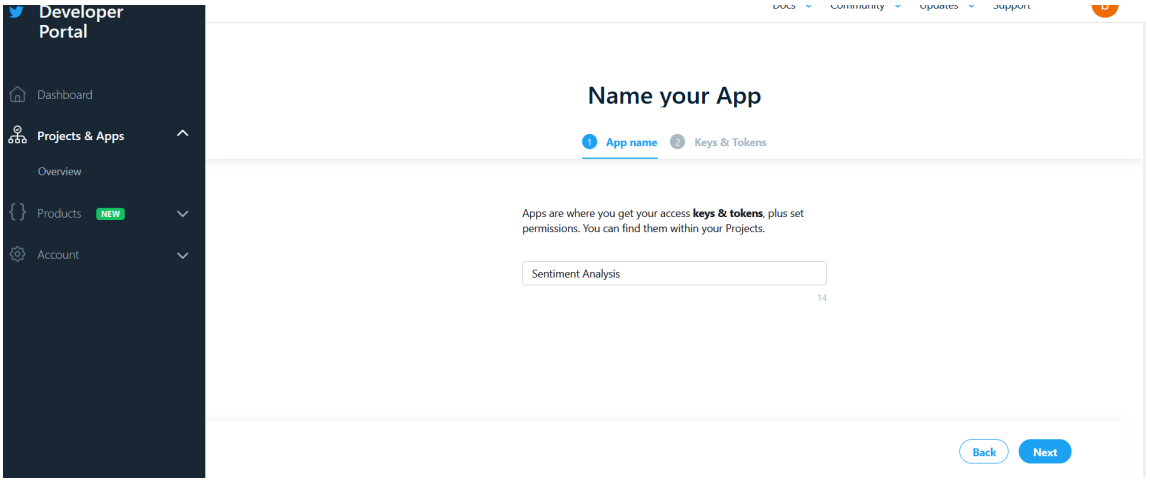
- Now let's proceed and generate API Key and API secret key



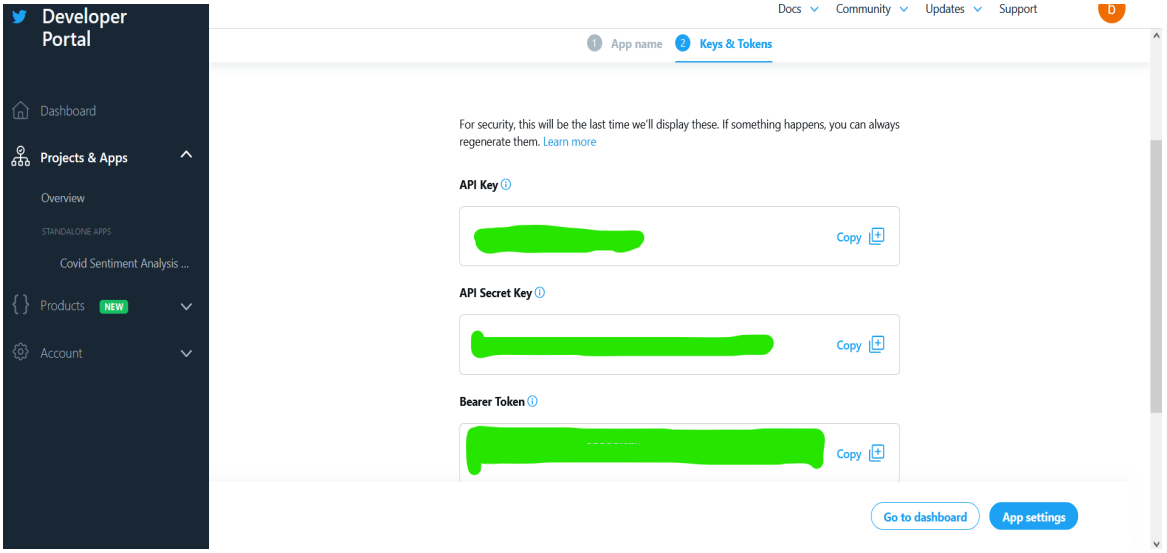
- Click on **Create App**



- **Name your App** and click **Next**



- You will be able to get your **API Key, API Secret Key**.



Case Study – Data Gathering from Twitter for Sentiment Analysis

Project Description

The project aims to retrieve a dataset for analysis and to discover public conversations on Twitter surrounding the Covid-19 pandemic. As strong concerns and emotions are expressed in the publicly available tweets.

Fetch Data from Twitter API in Python

There are several ways to access data from the Twitter API in python, for this tutorial, we will be using the **tweepy** Python library which makes it easy to connect to and fetch data from the Twitter API. In this tutorial, we'll be fetching the tweets with a specific hashtag (#covid19) from the API.

Tweepy is an excellently supported tool for accessing the Twitter API. There are a couple of different ways to install Tweepy. The easiest way is using pip

Installing Tweepy using PIP

```
!pip install tweepy
```

Once you are done with installing the Tweepy package. You need to import it to use it.

```
# import tweepy
import tweepy as tw
```

Authenticate with your credentials

We need to add our own authentication formation which we have generated above in the Twitter Developer site.

```
# your Twitter API key and API secret
my_api_key = "[redacted]"
my_api_secret = "[redacted]"
```

Now it's time to create our API object

```
# authenticate
auth = tw.OAuthHandler(my_api_key, my_api_secret)
api = tw.API(auth, wait_on_rate_limit=True)
```

This will be the basis of every application we build, so make sure you don't delete it.

Search Twitter for Tweets

Now you are ready to search Twitter for tweets! Start by finding recent tweets that use the **#covid19** hashtag. You will use the "**.Cursor"** method to get an object containing tweets containing the hashtag **#covid19**.

To create this query. You will define the search term – in this case, **#covid19**

```
search_query = "#covid19 -filter:retweets"
```

A retweet is when someone shares someone else's tweet. It is like sharing on Facebook. Sometimes you may want to remove retweets as they contain duplicate content that might skew your analysis if you are only looking at word frequency. Other times, you may want to keep retweets.

Collect the Tweets

```
# get tweets from the API
tweets = tw.Cursor(api.search,
                   q=search_query,
                   lang="en",
                   since="2020-09-16").items(100)
```

Above you use "**.Cursor()"** to search Twitter for tweets containing the search term #covid19. You can restrict the number of tweets returned by specifying a number in the "**.item()"** method. **.item(100)** will return 100 of the most recent tweets.

```
print(tweets)
```

```
<tweepy.cursor.ItemIterator object at 0x00000252AC65EC40>
```

.Cursor() returns an object that you can iterate or loop over to access the data collected. Each item in the iterator has various attributes that you can access to get information about each tweet including.

1. The text of the tweet
2. Who sent the tweet?
3. The date the tweet was sent

And more. The code below loops through the object and prints the text associated with each tweet.

```
# store the API responses in a list
tweets_copy = []
for tweet in tweets:
    tweets_copy.append(tweet)

print("Total Tweets fetched:", len(tweets_copy))
```

```
Total Tweets fetched: 100
```

Who is Tweeting About covid19?

You can access a wealth of information associated with each tweet. Below is an example of accessing the users who are sending the tweets related to **#covid19** and their locations. Note that user locations are manually entered into Twitter by the user. Thus, you will see a lot of variation in the format of this value.

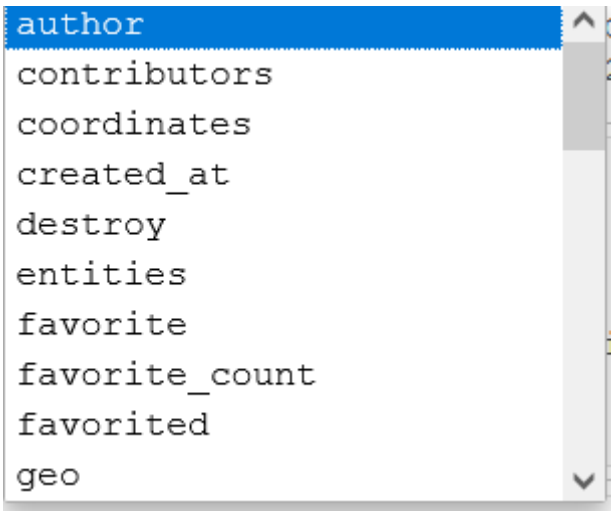
- **"tweet.user.screen_name"** provides the user's Twitter handle associated with each tweet.
- **"tweet.user.location"** provides the user's provided location.

```
# get tweets from the API
tweets = tw.Cursor(api.search,
              q=search_query,
              lang="en",
              since="2020-09-16").items(5)
user_locs = [[tweet.user.screen_name, tweet.user.location] for tweet in tweets]

user_locs
```

```
[['AHFarmServices', 'Littlehampton (SA)'],
 ['njoyflyfishing', 'Los Angeles, CA'],
 ['ironorehopper', 'Padova'],
 ['CityTshwane', 'South Africa- Gauteng'],
 ['therandomswill', '']]
```

You can experiment with other items available within a tweet by typing **a tweet.** and using the tab button to see all the available attributes stored.



Create the dataset

Once you have a list of items that you wish to work with, you can create a pandas data frame that contains the data.

```
# get tweets from the API
tweets = tw.Cursor(api.search,
              q=search_query,
              lang="en",
              since="2020-09-16").items(1000)
user_locs = [[tweet.user.name, tweet.user.screen_name, tweet.user.location,
              tweet.created_at, tweet.user.description] for tweet in tweets]
```

```
import pandas as pd
tweet_text = pd.DataFrame(data=user_locs, columns=['UserName','ScreenName',
                                                   'location','TweetAt', 'OriginalTweet'])
tweet_text.head()
```

	UserName	ScreenName	location	TweetAt	OriginalTweet
0	Sight Magazine	sightmagazine	Australia	2021-08-15 07:42:15	Covering global news and issues from a Christi...
1	Pune Vaccine Updater	PuneUpdater		2021-08-15 07:42:08	Follow us to get update as soon 18 plus vaccin...
2	COVID Watch Kansas	CovidWatchKS	Kansas	2021-08-15 07:42:07	Helping to keep #Kansans informed of the lates...
3	Nathan Joyner	njoyflyfishing	Los Angeles, CA	2021-08-15 07:41:57	Global Venture Captial and Private Equity/Busi...
4	Hulhumalé Hospital	HMH_mv	Maldives	2021-08-15 07:41:33	Official Twitter account of Hulhumalé Hospital...

We have the data set ready for analysis. Let's save the data set into our local drive.

```
tweet_text.to_csv('twitter.csv')
```

SUMMARY

We have started with how tweets can be used for analyzing sentiment, then moved on to see the step-by-step process for getting Twitter API access. Once we have granted access to the Twitter developer account, we used the Authentication credentials to retrieve the tweets using the tweepy python library. Finally, passing the search query we have created a pandas data frame, data frame is quite useful for further analysis.

Assignment

1. Follow the step-by-step procedure and generate the Twitter API Key and API secret key.
2. Create a dataset for analyzing the sentiment on Race Bias(#Racism).
3. Try to experiment with attributes like “Verified”, “Source”, “hashtag”, etc.

CHAPTER 2: Twitter Data Preprocessing

Theory

In the previous chapter we saw, how to get Twitter API access and how to import Twitter data into our local desktop. In this chapter, we will be doing data preprocessing on the Twitter data dataset.

Approach To Analyze Various Sentiments

Before we proceed further, one should know what is mean by Sentiment Analysis. Sentiment Analysis is the process of computationally identifying and categorizing opinions expressed in a piece of text, especially to determine whether the writer's attitude towards a particular topic is Positive, Negative, or Neutral.

Following is the Standard Operating Procedure to tackle the Sentiment Analysis kind of Project. We will be going through this procedure to predict what we are supposed to predict!

1. **Exploratory Data Analysis.**
2. **Data Preprocessing.**
3. **Vectorization.**
4. **Classification Models.**
5. **Evaluation.**
6. **Conclusion.**

Let's Guess some tweets

I will read the tweet, and can you tell me the sentiment of that whether it is Positive, Negative, or Neutral. So, the first tweet is **"Still shocked by the number of #Toronto supermarket employees working without some sort of mask. We all know by now; employees can be asymptomatic while spreading #coronavirus".** What's your guess? Yeah, you are correct. This is a Negative tweet because it contains negative words like "shocked".

Data Summary

The original dataset has 4 columns and 6318 rows. To analyze various sentiments, we require just two columns Original Tweet and Sentiment. The tweets are available in the Original Tweets columns, and we need Sentiment labels, that we will be generating in going ahead.

```
#load the data
data = pd.read_csv("twitter.csv", )

data.head()
```

	Unnamed: 0	UserId	location	TweetAt	OriginalTweet
0	0	996111615899635714	Nottinghamshire, England	2021-08-19 09:26:02	Native of East Hull. \nFormer: police research...
1	1	1341762551328157696	England - Singapore - Penang	2021-08-19 09:23:21	Musings on colonial #Malaya & #Singapore - the...
2	2	1123248342333575169	Sri Lanka	2021-08-19 09:20:10	Presenting The Interesting With Colour & Style
3	3	3991108098	Chennai, India	2021-08-19 09:19:44	DT Next is the English daily from Daily Thanth...
4	4	1244954351199817730	Ibadan	2021-08-19 09:13:26	This is the official handle of the Oyo State C...

Exploratory Data Analysis

The columns such as “UserId”, “Location” and “TweetAt” do not give any meaningful insights for our analysis. Hence, we are not using these features for model building.

```
data.info()
```

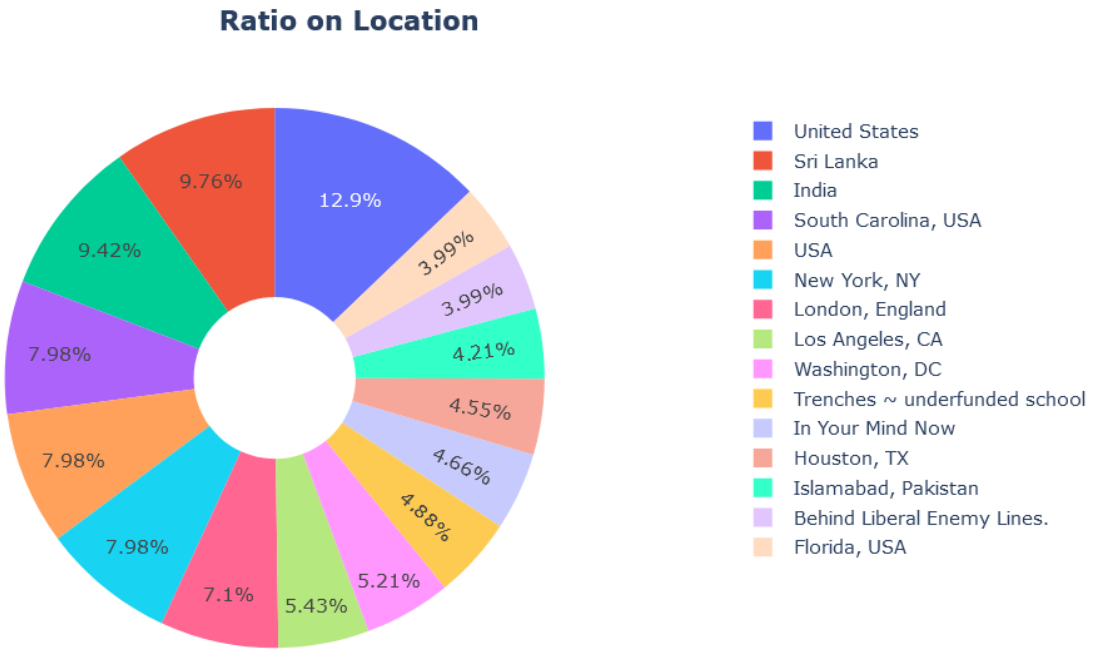
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6319 entries, 0 to 6318
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Unnamed: 0     6319 non-null   int64 
 1   UserId         6319 non-null   int64 
 2   location       5096 non-null   object
 3   TweetAt        6319 non-null   object
 4   OriginalTweet  6012 non-null   object
dtypes: int64(2), object(3)
memory usage: 247.0+ KB
```

There are some null values in the location and Original Tweet, but we don’t need to deal with them.

Let’s see from which location we are getting the maximum number of tweets.

```
data_viz = {
   "values": loc_analysis['count'][:15],
   "labels": loc_analysis.index[:15],
   "domain": {"column": 0},
   "name": "Location Name",
   "hoverinfo":"label+percent+name",
   "hole": .3,
   "type": "pie"
}
layout = go.Layout(title="<b>Ratio on Location</b>", legend=dict(x=0.9, y=1, orientation="v"))

data_viz = [data_viz]
fig = go.Figure(data = data_viz, layout = layout,)
fig.update_layout(title_x=0.5)
fig.show()
```



The maximum tweets came from United States (12.9%) location as evident from the above figure.

Let's see the top 10 countries contributing the highest number of tweets.

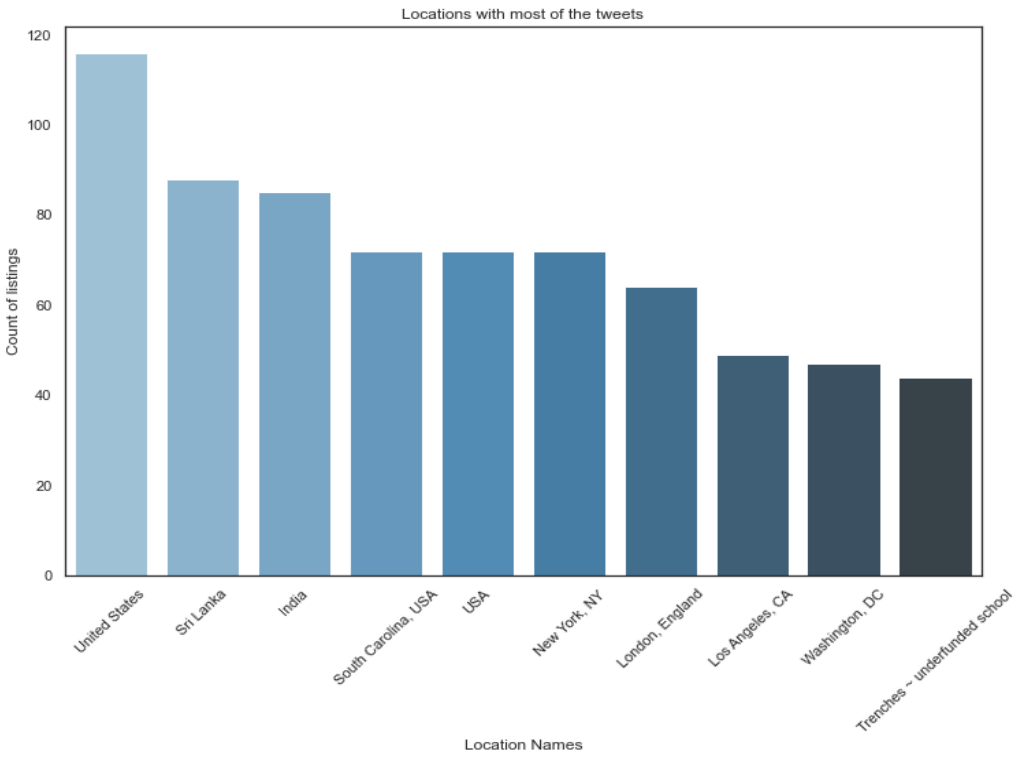
```
Top_Location_Of_tweet = data['location'].value_counts().head(10)

sns.set(rc={'figure.figsize':(12,8)})
sns.set_style('white')
```

```
Top_Location_Of_tweet.head(10)
```

```
United States                    116
Sri Lanka                         88
India                             85
South Carolina, USA               72
USA                               72
New York, NY                      72
London, England                   64
Los Angeles, CA                   49
Washington, DC                    47
Trenches ~ underfunded school     44
Name: location, dtype: int64
```

Following are the top 10 countries contributing the highest number of tweets.



We must drop null values from our dataset to clean our dataset.

```
data.dropna(inplace=True)
```

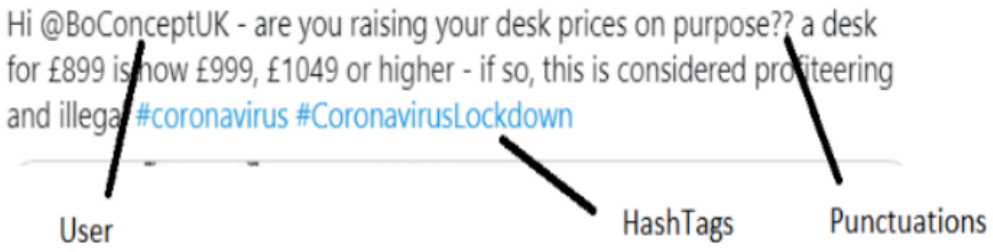
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4956 entries, 0 to 6317
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Unnamed: 0      4956 non-null   int64 
 1   UserId          4956 non-null   int64 
 2   location        4956 non-null   object
 3   TweetAt         4956 non-null   object
 4   OriginalTweet   4956 non-null   object
dtypes: int64(2), object(3)
memory usage: 232.3+ KB
```

You can see there are no null values in any of the columns. Now we can continue with the data cleaning part.

Data Cleaning

The preprocessing of the text data is an essential step as it makes the raw text ready for mining. The objective of this step is to clean those that are less relevant to find the sentiment of tweets such as **punctuation, special characters, numbers, tweet handle links,** and which don't carry much weightage in context to the text.




```
def cleaner(tweet):
    tweet = re.sub("@[A-Za-z0-9]+","",tweet) #Remove @ sign
    tweet = re.sub(r"(?:\@|http?\://|https?\://|www)\S+", "", tweet) #Remove http links
    tweet = " ".join(tweet.split())
    tweet = ''.join(c for c in tweet if c not in emoji.UNICODE_EMOJI) #Remove Emojis
    tweet = tweet.replace("#", "").replace("_", " ") #Remove hashtag sign but keep the text
    tweet = " ".join(w for w in nltk.wordpunct_tokenize(tweet) \
         if w.lower() in words or not w.isalpha())
    return tweet


data['Tweets'] = data['OriginalTweet'].apply(cleaner)
```

We have defined a function “Cleaner” to clean the text using the “regular expression” library.

The first line will remove all the “@” signs, the second line will remove “HTTP/HTTPS” links, and then we will split all the tweets and join them with a space. The next line of the function will remove the emojis followed by “#” signs. finally, the function removes all the punctuations and converts the alphabets into the lower case.

Once all the cleaning is done, we are saving the cleaned text to a new variable called “Tweets”

	OriginalTweet	Tweets
	Native of East Hull. \nFormer: police research...	Native of East Hull . Former : police research...
	Musings on colonial #Malaya & #Singapore - the...	on colonial & - the which formed part of the -...
	Presenting The Interesting With Colour & Style	The Interesting With Colour & Style
	DT Next is the English daily from Daily Thanth...	Next is the daily from Daily group . With a mi...
	This is the official handle of the Oyo State C...	This is the official handle of the State Covid...

We can see “\n” and “#” were removed and the cleaned text is saved in the variable “Tweets”.

Let’s create a new data frame containing only “Tweets” and we need to add a new column for sentiment labels.

```
df = pd.DataFrame()
df['Tweets'] = data['Tweets']
df.head()
```

	Tweets
0	Native of East Hull . Former : police research...
1	on colonial & - the which formed part of the -...
2	The Interesting With Colour & Style
3	Next is the daily from Daily group . With a mi...
4	This is the official handle of the State Covid...

Target Labels

To build a sentiment analysis model using machine learning or NLP, we need to have a labeled variable to train our model. To create a target variable, we are using a python package called Textblob.

Textblob

It is a simple python library that offers API access to different NLP tasks such as sentiment analysis, spelling correction, etc. Which uses Rule-based sentiment analysis. Rule-based sentiment analysis is one of the very basic approaches to calculate text sentiments. It only requires minimal pre-work, and the idea is quite simple, this method does not use any machine learning to figure out the text sentiment. For example, we can figure out the sentiment of a sentence by counting the number of times the user has used the word “sad” in his/her tweet.

Textblob sentiment analyzer returns two properties for a given input sentence:

- **Polarity** is a float that lies between [-1,1], -1 indicates negative sentiment and +1 indicates positive sentiments.
- **Subjectivity** is also a float that lies in the range of [0,1]. Subject sentences generally refer to opinion, emotion, or judgment.

```
from textblob import TextBlob

#creating a function to get the subjectivity
def getSubjectivity(text):
    return TextBlob(text).sentiment.subjectivity

#creating a function to get the polarity
def getPolarity(text):
    return TextBlob(text).sentiment.polarity
```

Imported a TextBlob function from the textblob library. Defined two functions to get the polarity and subjectivity of the tweets and save them to the respective columns in the data frame.

```
#Create two columns

df['Subjectivity'] = df['Tweets'].apply(getSubjectivity)
df['Polarity'] = df['Tweets'].apply(getPolarity)
```

```
df.head()
```

	Tweets	Subjectivity	Polarity
0	Native of East Hull . Former : police research...	0.0	0.0
1	on colonial & - the which formed part of the -...	0.0	0.0
2	The Interesting With Colour & Style	0.5	0.5
3	Next is the daily from Daily group . With a mi...	0.0	0.0
4	This is the official handle of the State Covid...	0.0	0.0

As we discussed, Subjectivity and Polarity have given a float value for the tweets based on this we can our desired target label variable as follows.

```
#Create a function to compute the negative, neutral and positive analysis
def getAnalysis(score):
    if score < 0:
        return 'Negative'
    elif score == 0:
        return 'Neutral'
    else:
        return 'Positive'


df['Analysis'] = df['Polarity'].apply(getAnalysis)

df.head()
```

	Tweets	Subjectivity	Polarity	Analysis
0	Native of East Hull . Former : police research...	0.0	0.0	Neutral
1	on colonial & - the which formed part of the -...	0.0	0.0	Neutral
2	The Interesting With Colour & Style	0.5	0.5	Positive
3	Next is the daily from Daily group . With a mi...	0.0	0.0	Neutral
4	This is the official handle of the State Covid...	0.0	0.0	Neutral

Let's check the stats of the "Analysis" variables.

```
df['Analysis'].value_counts()
```

```
Neutral     2226
Positive    2122
Negative     608
Name: Analysis, dtype: int64
```

Let's filter out the Positive, Negative, and Neutral tweets and check the words used on those respective tweets.

```
Neutral_df = df[df['Analysis']=='Neutral']
Neutral_df.head()
```

	Tweets	Subjectivity	Polarity	Analysis
0	Native of East Hull . Former : police research...	0.0	0.0	Neutral
1	on colonial & - the which formed part of the -...	0.0	0.0	Neutral
3	Next is the daily from Daily group . With a mi...	0.0	0.0	Neutral
4	This is the official handle of the State Covid...	0.0	0.0	Neutral
8	you news and on current .	0.4	0.0	Neutral

We can visually analyze the words which are used in Neutral sentiment by using a word cloud.

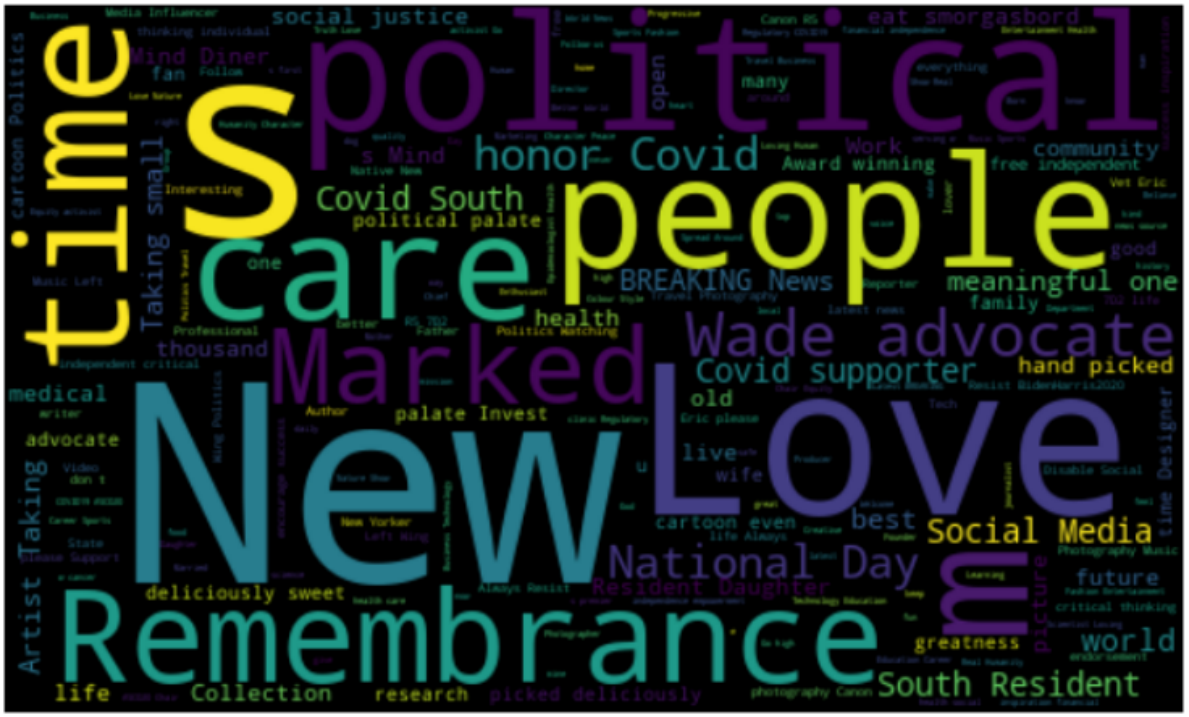
```
allWords_neutral = ' '.join(twts for twts in Neutral_df['Tweets'])
wordCloud_neutral = WordCloud(width=500, height=300, random_state=21,
                      max_font_size=119).generate(allWords_neutral)
plt.imshow(wordCloud_neutral, interpolation='bilinear')
plt.axis('off')
plt.show()
```

You see the words like News, Health, Choice, and Public were used mostly in the Neutral Tweets.

Similarly, let's check the Positive and Negative Word cloud.

```
Positive_df=df[df['Analysis']=='Positive']
Positive_df.head()
```

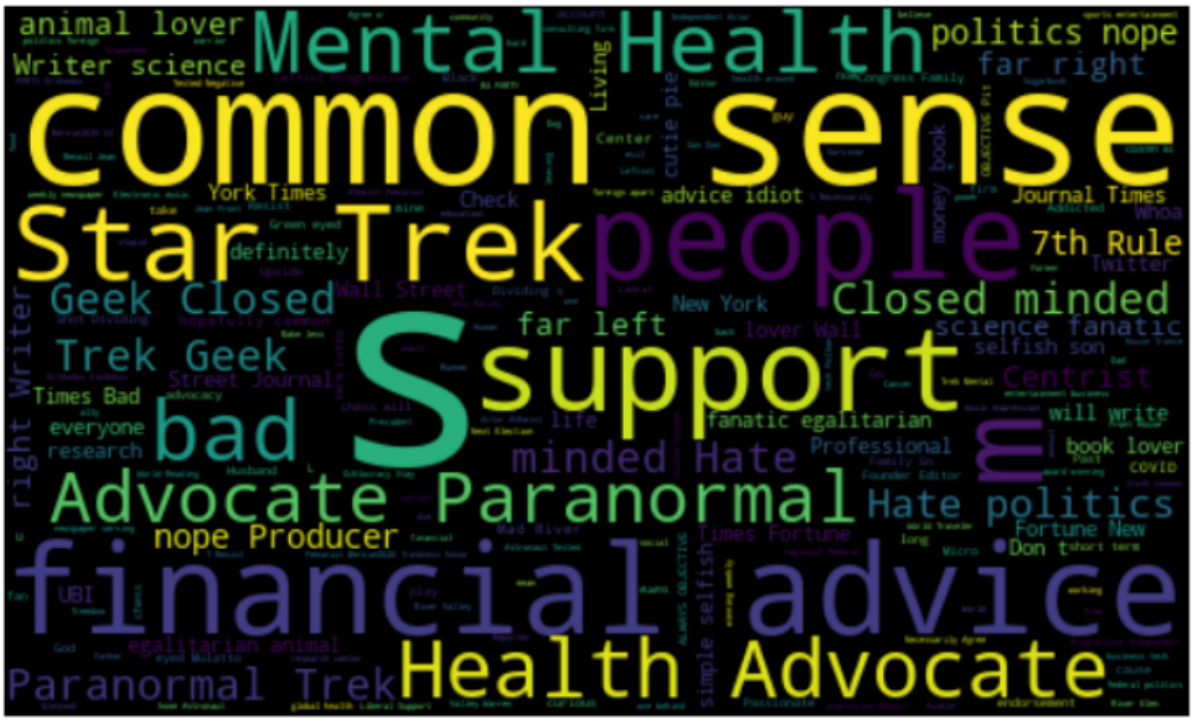
```
allWords_positive = ' '.join(twts for twts in Positive_df['Tweets'])
wordCloud_positive = WordCloud(width=500, height=300, random_state=21,
                    max_font_size=119).generate(allWords_positive)
plt.imshow(wordCloud_positive, interpolation='bilinear')
plt.axis('off')
plt.show()
```



The most used words in positive tweets were Love, New, Care, People, and so on.

```
Negative_df = df[df['Analysis']=='Negative']
Negative_df.head()
```

```
allWords_negative = ' '.join(twts for twts in Negative_df['Tweets'])
wordCloud_negative = WordCloud(width=500, height=300, random_state=21,
                    max_font_size=119).generate(allWords_negative)
plt.imshow(wordCloud_negative, interpolation='bilinear')
plt.axis('off')
plt.show()
```



The most frequently used words in Negative Tweets were Common Sense, Mental Health, Financial, Health Advocate, and so on.

Let's save the cleaned data to a local drive for our future use.

```
df[['Tweets', 'Analysis']].to_csv('Cleaned_tweets.csv')
```

We have done with our preprocessing steps, In the next chapter, we will be using the NLTK library to create vectorization, steaming and much more to make our text ready for building a sentiment analysis model.

SUMMARY

We started with importing the Twitter dataset which we have created in our previous chapter. Once imported the data, did some exploratory data analysis followed by data cleaning to remove special characters like “@/#”, “HTTP/HTTPS” links, and so on. After cleaning the data, performed Rule-based sentiment analysis to label the data. Now are good to go with text vectorization, lemmatization, and build the sentiment analysis model.

ASSIGNMENT

1. Import the Twitter dataset with at least 5000 tweets and create a pandas data frame by filtering the tweets related to “Cuba” or “Miami” or “California” location.
2. Perform exploratory data analysis and draw some meaningful insights.
3. Perform Data Cleaning
4. Create a target variable using the Textblob package.
5. Analyze the most repeated words from positive, negative, and neutral tweets.

CHAPTER 3: Text Preprocessing using NLTK

Theory

In the previous chapter, we saw data exploration on the Twitter data dataset. In this chapter, we will be doing text preprocessing like Tokenization, Stemming, Lemmatization, Stop Words, POS tagging, and Named Entity Recognition.

What is Natural Language Processing?

Natural Language Processing (NLP) is a process of manipulating or understanding the text or speech by any software or machine. An analogy is that humans interact and understand each other's views and respond with the appropriate answer. In NLP, this interaction, understanding, and response are made by a computer instead of a human.

What is NLTK?

NLTK (Natural Language Toolkit) is a suite that contains libraries and programs for statistical language processing. It is one of the most powerful NLP libraries, which contains packages to make machines understand human language and reply to it with an appropriate response.

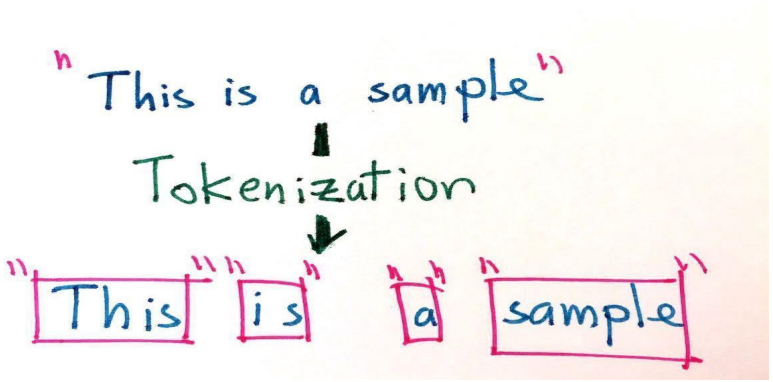
Techniques of Text Preprocessing using NLTK library

Following are some of the most common text preprocessing steps.

Tokenization

Tokenization is the process of breaking down a given paragraph of text into a list of sentences or words. When a paragraph is broken down into a list of sentences, it is called sentence tokenization. Similarly, if the sentences are further broken down into a list of words, it is known as Word tokenization.

Example:



Below is a given paragraph, let's see how tokenization works on it:

```
['Abraham Lincoln was an American lawyer and statesman who served as the /16th president of the United States from 18
61 until his assassination in 1865.', "Lincoln led the nation through the American Civil War, the country's greatest
moral, cultural, constitutional, and political crisis.", 'He succeeded in preserving the Union, abolishing slavery, b
olstering the federal government, and modernizing the U.S. economy.']
```

Word Tokenizer

```
['Abraham', 'Lincoln', 'was', 'an', 'American', 'lawyer', 'and', 'statesman', 'who', 'served', 'as', 'the', '/16th',
'president', 'of', 'the', 'United', 'States', 'from', '1861', 'until', 'his', 'assassination', 'in', '1865', '.', 'Li
ncoln', 'led', 'the', 'nation', 'through', 'the', 'American', 'Civil', 'War', ',', 'the', 'country', "'s", 'greatest
', 'moral', ',', 'cultural', ',', 'constitutional', ',', 'and', 'political', 'crisis', '.', 'He', 'succeeded', 'in',
'preserving', 'the', 'Union', ',', 'abolishing', 'slavery', ',', 'bolstering', 'the', 'federal', 'government', ',', '
and', 'modernizing', 'the', 'U.S.', 'economy', '.']
```

Stemming and Lemmatization

Many words that are used in a sentence are not always used in their base form but are used as per the rules of grammar.

Example:

- Running Run (base word)
- Runs Run (base word)
- Ran Run (base word)

Although the underlying meaning is the same, the form of the base word changes to preserve the correct grammatical meaning.

Stemming and lemmatization are basically used to bring such words to their basic form so that the words with the same base are treated as the same words rather than treated differently.

The only difference in Stemming and Lemmatization is the way in which they change the word's base form.

- **Stemming**

Stemming means mapping a group of words to the same stem by removing prefixes or suffixes without giving any value to the “grammatical meaning” of the stem formed after the process.

Example:
Computation Comput
Computer Comput
Hobbies Hobbi

We can see that steaming tries to bring the word back to its base word, but the base word may or may not have the correct grammatical meaning.

- **Lemmatization**
  Lemmatization also does the same thing as stemming and tries to bring a word to its base form, but unlike stemming it does keep in account the actual meaning of the base word i.e., the base word belongs to any specific language. The “base word” is known as “Lemma”.

  Example:
  Finally Final
  Final Final
  Finalized Final

Stop Words

Stop words are words that are very common in occurrence such as ‘a’, ‘an’, ‘the’, ‘at’ etc. We ignore such words during the preprocessing part since they do not give any important information and would just take additional space. We can make our custom list of stop words as well if we want.

['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"]

Parts Of Speech Tagging (POS Tagging) and Chunking

As the name suggests, it is a method of tagging individual words based on their parts of speech.

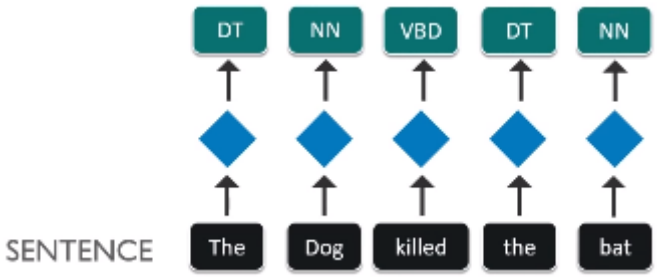
Part-of-speech tagging also called grammatical tagging or word-category disambiguation is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

There are 9 parts of speech in grammars, but in NLP there are more than 9 POS tags based on a different set of rules, such as:

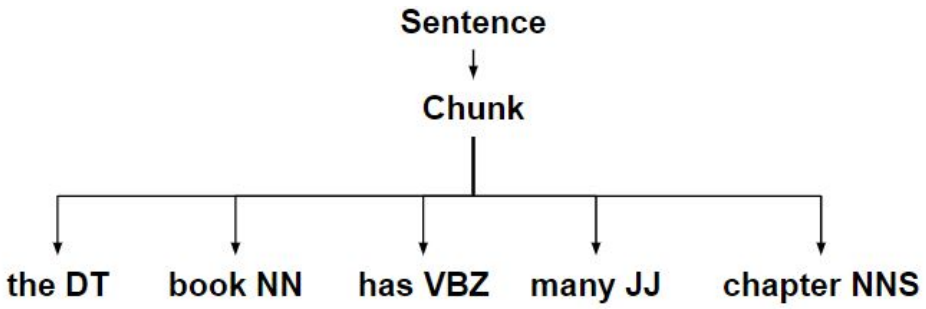
Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBR	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WP$	Possessive whpronoun
WRB	Whadverb

Example:



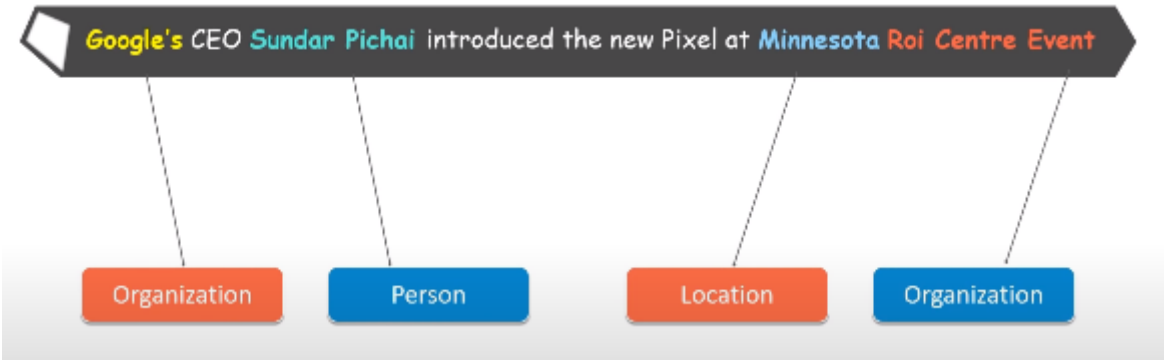
Chunking can be used to make data more structured by giving a specific set of rules. Chunking is also known as a shallow parser.



Named Entity Recognition (NER)

In chucking, we read that we can set rules to keep different POS tags under one single user-defined tag. One such form of chunking in NLP is known as Named Entity Recognition.

Example:



Parts Of Speech Tagging (POS Tagging) and Chunking

As the name suggests, it is a method of tagging individual words based on their parts of speech.

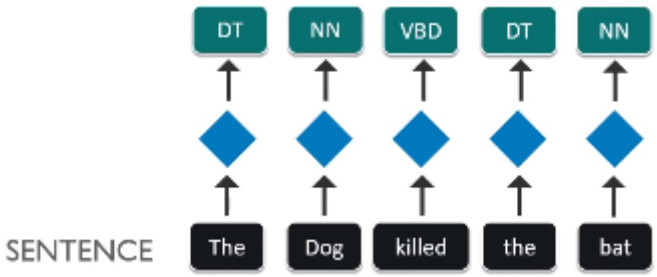
Part-of-speech tagging also called grammatical tagging or word-category disambiguation is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

There are 9 parts of speech in grammars, but in NLP there are more than 9 POS tags based on a different set of rules, such as:

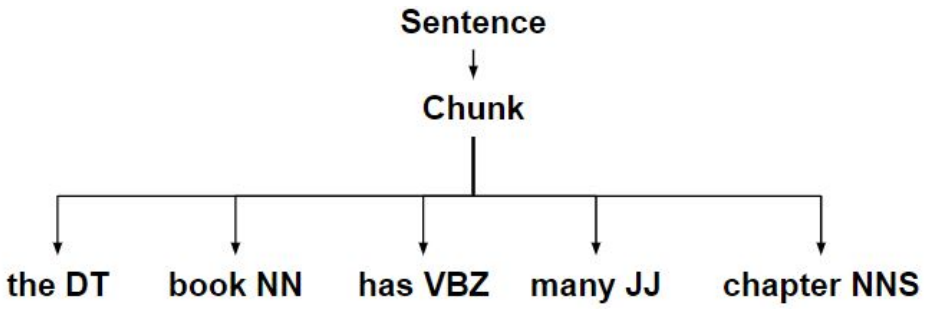
Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBR	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WP$	Possessive whpronoun
WRB	Whadverb

Example:



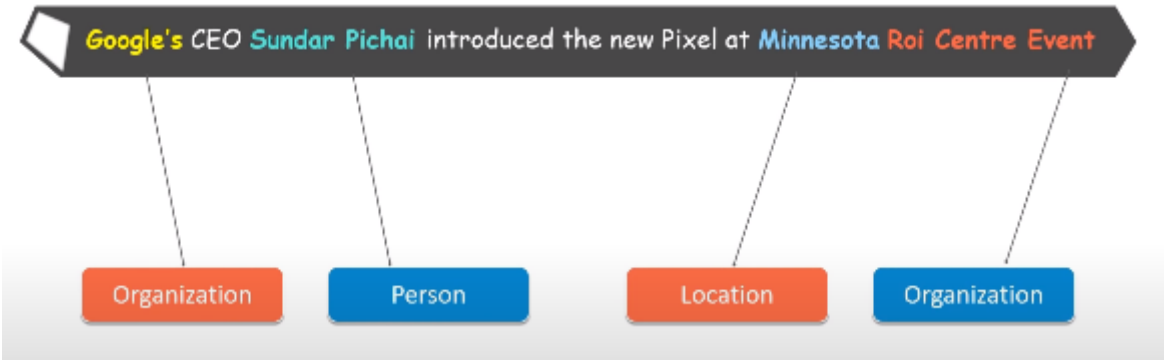
Chunking can be used to make data more structured by giving a specific set of rules. Chunking is also known as a shallow parser.



Named Entity Recognition (NER)

In chucking, we read that we can set rules to keep different POS tags under one single user-defined tag. One such form of chunking in NLP is known as Named Entity Recognition.

Example:



Understanding Word N-grams

N-gram is probably the easiest concept to understand in the whole text preprocessing space. An N-gram means a sequence of N words. So, for example, “Sentiment Analysis” is a 2-gram (a bigram), “Twitter Sentiment Analysis” is a 3-gram (trigram).

When,

- n = 1, we call it a “unigram”
- n = 2, it is called a “bigram”
- n = 3, it is called a “trigram”.

Let’s understand this with an example:

Text1 = “I went to have a cup of coffee but I ended up having lunch with her.”

Unigram

[I, went, to, have, a, cup, of, coffee, but, I, ended, up, having, lunch, with, her]

Bigram

[I went], [went to],[to have],[have a],[a cup],[cup f],[of coffee],[coffee but],[but I],[I ended],[ended up], [up having],[having lunch],[lunch with],[with her]

Tri-gram

[I went to], [went to have], [to have a], [have a cup],[a cup of], [cup of coffee],[of coffee but],[coffee but I],[but I ended],[I ended up],[ended up having],[up having lunch],[having lunch with],[lunch with her].

Note: We can clearly see that the BOW model is nothing but an n-gram model when n = 1.

Skip-grams

Skip grams are a type of n-grams where the words are not necessarily in the same order as are in the given text. i.e., some words can be skipped.

Example:

Text2 = “I don’t understand, what is the problem here?”

1-skip 2-grams (we have to make 2-gram while skipping 1 word)

[I understand, don’t what, understand is, what the, is the problem, here].

AIM

The aim of the following lab exercises is to perform various exercises by writing python code so that we can get hands-on Text Preprocessing

The labs for this chapter include the following exercises.

- Installing NLTK package
- Tokenization
- Stemming
- Lemmatization
- Stop Words
- POS tagging
- Name Entity Recognition
- Applying the above techniques on our Tweets dataset.

NLTK installation

Step 1: Open the anaconda command prompt and run the below code

```
(base) C:\Users\pc>pip install nltk
```

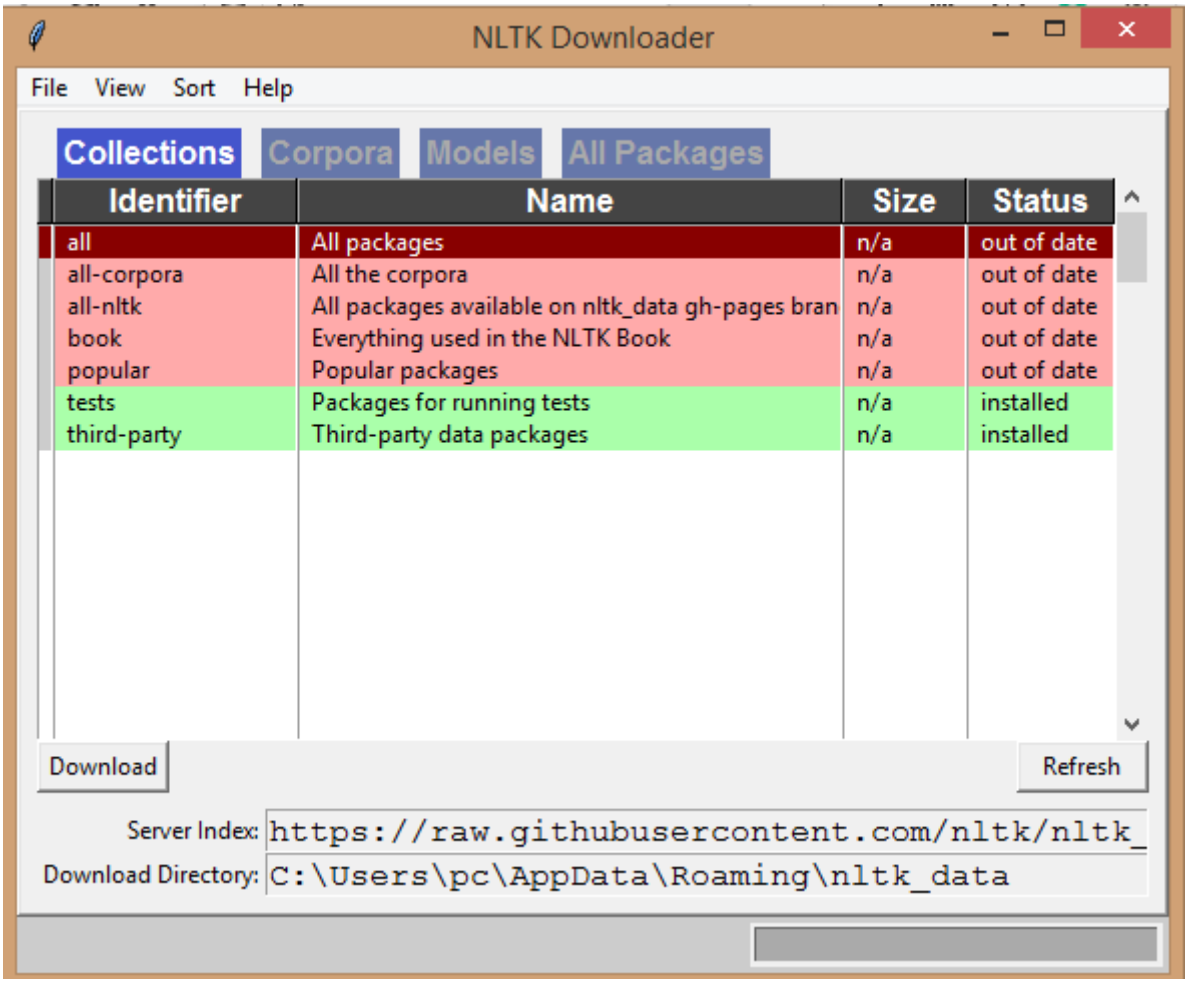
Step2: Open the jupyter notebook and import the nltk package

```
import nltk
```

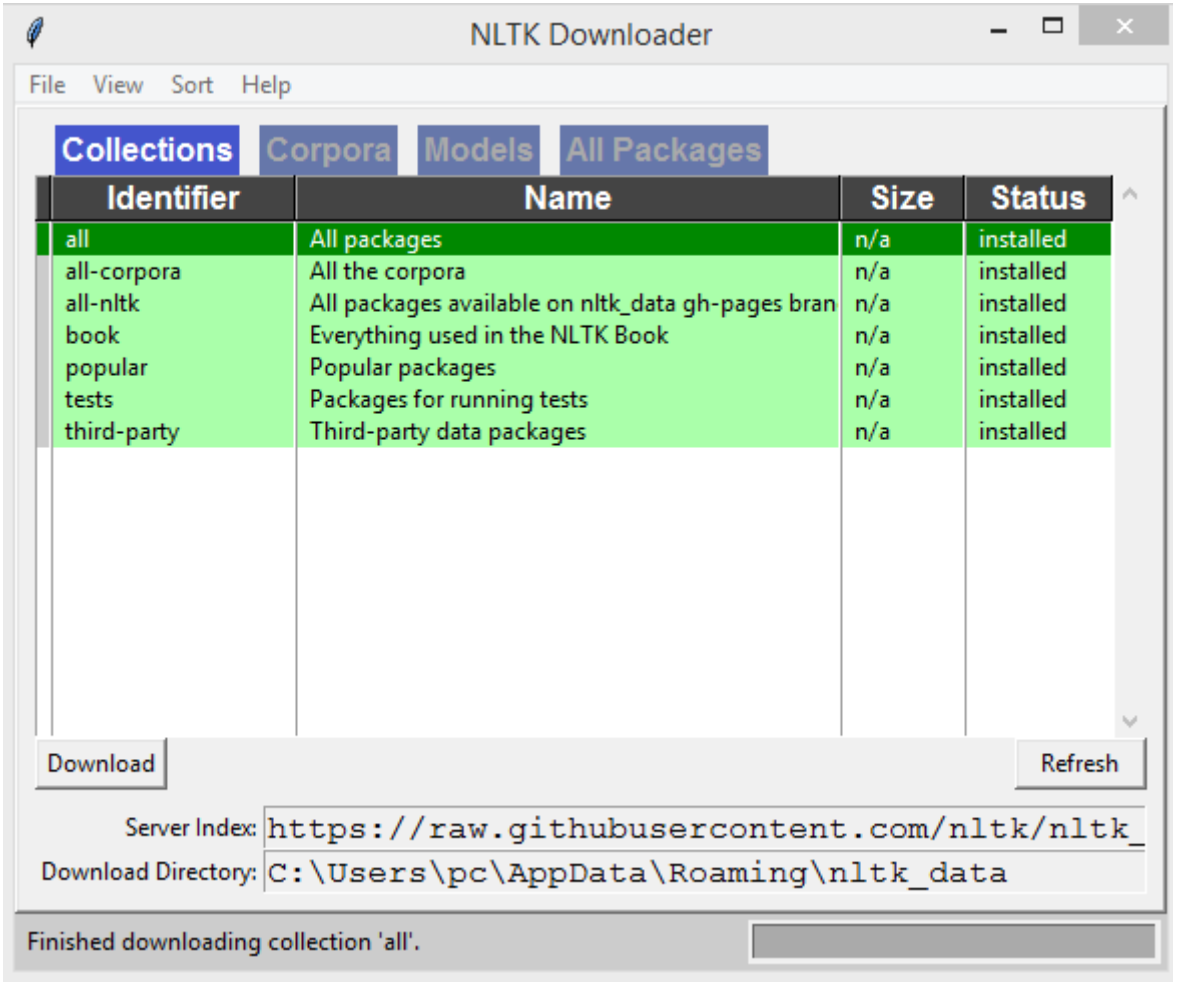
Step3: After successfully importing the nltk package, we need to download all the nltk package functions using the following command

```
nltk.download()
```

Step4: When you run the above command, the nltk downloader window pops-up



Step5: From the above window we need to select "All Packages" and click the download button. Once the download completes we can see the status as "installed" for all the packages.



Tokenization

- **Sentence Tokenization**

Step1: Assign the text data to the variable “data”

```
data = '''Abraham Lincoln was an American lawyer and statesman who
served as the 16th president of the United States from 1861 until his
assassination in 1865. Lincoln led the nation through the American
Civil War, the country's greatest moral, cultural, constitutional,
and political crisis. He succeeded in preserving the Union,
abolishing slavery, bolstering the federal government,
and modernizing the U.S. economy.'''
```

Step2: Perform the sentence tokenization using the “sent_tokenization” function.

```
print(nltk.sent_tokenize(data))
```

```
['Abraham Lincoln was an American lawyer and statesman who served as t
he 16th president of the United States from 1861 until his assassinati
on in 1865.', "Lincoln led the nation through the American Civil War,
the country's greatest moral, cultural, constitutional, and political
crisis.", 'He succeeded in preserving the Union, abolishing slavery, b
olstering the federal government, and modernizing the U.S. economy.']
```

The entire sentence was treated as a single token.

- **Word Tokenization**

Perform the word tokenization using the “word_tokenize” function.

```
print(nltk.word_tokenize(data))
```

```
['Abraham', 'Lincoln', 'was', 'an', 'American', 'lawyer', 'and', 'stat
esman', 'who', 'served', 'as', 'the', '16th', 'president', 'of', 'the
', 'United', 'States', 'from', '1861', 'until', 'his', 'assassination
', 'in', '1865', '.', 'Lincoln', 'led', 'the', 'nation', 'through', 't
he', 'American', 'Civil', 'War', ',', 'the', 'country', "'s", 'greates
t', 'moral', ',', 'cultural', ',', 'constitutional', ',', 'and', 'poli
tical', 'crisis', '.', 'He', 'succeeded', 'in', 'preserving', 'the', '
Union', ',', 'abolishing', 'slavery', ',', 'bolstering', 'the', 'feder
al', 'government', ',', 'and', 'modernizing', 'the', 'U.S.', 'economy
', '.']
```

Stemming

Step1: Importing the packages

```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer, SnowballStemmer
```

Step2: Performing the stemming operation

```
lancaster = LancasterStemmer()
porter = PorterStemmer()
Snowball = SnowballStemmer("english")
print('Porter stemmer')
print(porter.stem("hobby"))
print(porter.stem("hobbies"))
print(porter.stem("computer"))
print(porter.stem("computation"))
print("**************************")
print('lancaster stemmer')
print(lancaster.stem("hobby"))
print(lancaster.stem("hobbies"))
print(lancaster.stem("computer"))
print(porter.stem("computation"))
print("**************************")
print('Snowball stemmer')
print(Snowball.stem("hobby"))
print(Snowball.stem("hobbies"))
print(Snowball.stem("computer"))
print(Snowball.stem("computation"))
```

```
Porter stemmer
hobbi
hobbi
comput
comput
**************************
lancaster stemmer
hobby
hobby
comput
comput
**************************
Snowball stemmer
hobbi
hobbi
comput
comput
```

Example2:

```
sent = "I was going to the office on my bike when\
i saw a car passing by hit the tree."
token = list(nltk.word_tokenize(sent))
for stemmer in (Snowball, lancaster, porter):
    stemm = [stemmer.stem(t) for t in token]
    print(" ".join(stemm))
```

```
i was go to the offic on my bike wheni saw a car pass by hit the tree .
i was going to the off on my bik when saw a car pass by hit the tre .
I wa go to the offic on my bike wheni saw a car pass by hit the tree .
```

Example3:

```
paragraph = """I have three visions for India. In 3000 years of our history, people from all over
               the world have come and invaded us, captured our lands, conquered our minds.
               From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,
               the French, the Dutch, all of them came and looted us, took over what was ours.
               Yet we have not done this to any other nation. We have not conquered anyone.
               We have not grabbed their land, their culture,
               their history and tried to enforce our way of life on them.
               Why? Because we respect the freedom of others.That is why my
               first vision is that of freedom. I believe that India got its first vision of
               this in 1857, when we started the War of Independence. It is this freedom that
               we must protect and nurture and build on. If we are not free, no one will respect us.
               My second vision for India's development. For fifty years we have been a developing nation.
               It is time we see ourselves as a developed nation. We are among the top 5 nations of the world
               in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling.
               Our achievements are being globally recognised today. Yet we lack the self-confidence to
               see ourselves as a developed nation, self-reliant and self-assured. Isn't this incorrect?
               I have a third vision. India must stand up to the world. Because I believe that unless India
               stands up to the world, no one will respect us. Only strength respects strength. We must be
               strong not only as a military power but also as an economic power. Both must go hand-in-hand.
               My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of
               space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material.
               I was lucky to have worked with all three of them closely and consider this the great opportunity of my life.
               I see four milestones in my career"""
```

```
sentences = nltk.sent_tokenize(paragraph)
stemmer = PorterStemmer()


# Stemming
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [stemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)

sentences
```

```
['I three vision india .',
 'In 3000 year histori , peopl world come invad us , captur land , conquer mind .',
 'from alexand onward , greek , turk , mogul , portugues , british , french , dutch , came loot us , took .',
 'yet done nation .',
 'We conquer anyon .',
 'We grab land , cultur , histori tri enforc way life .',
 'whi ?',
 'becaus respect freedom others.that first vision freedom .',
 'I believ india got first vision 1857 , start war independ .',
 'It freedom must protect nurtur build .',
 'If free , one respect us .',
 'My second vision india ’ develop .',
 'for fifti year develop nation .',
 'It time see develop nation .',
 'We among top 5 nation world term gdp .',
 'We 10 percent growth rate area .',
 'our poverti level fall .',
 'our achiev global recognis today .',
 'yet lack self-confid see develop nation , self-reli self-assur .',
 'isn ’ incorrect ?',
 'I third vision .',
 'india must stand world .',
 'becaus I believ unless india stand world , one respect us .',
 'onli strength respect strength .',
 'We must strong militari power also econom power .',
 'both must go hand-in-hand .',
 'My good fortun work three great mind .',
 'dr. vikram sarabhai dept .',
 'space , professor satish dhawan , succeed dr. brahm prakash , father nuclear materi .',
 'I lucki work three close consid great opportun life .',
 'I see four mileston career']
```

The Lancaster algorithm is faster than porter’s, but it is more complex. Porter Stemmer is the oldest algorithm and was most popular to use.

Snowball stemmer, also known as porter2, is the updated version of the Porter Stemmer and is currently the most popular stemming algorithm.

Snowball stemmer is available in multiple languages as well.

Lemmatization

Step1: Import the package

```
from nltk.stem import WordNetLemmatizer
```

Step2: Performing the lemmatization operation using the “WordNetLemmatizer” function.

```
lemma = WordNetLemmatizer()

print(lemma.lemmatize('running'))
print(lemma.lemmatize('runs'))
print(lemma.lemmatize('ran'))
```

```
running
run
ran
```

Here, we can see the lemma has changed for the words with the same base. This is because; we haven't given any context to the Lemmatizer.

Example 2: Let's perform Lemmatization on the paragraph which we created above

```
sentences = nltk.sent_tokenize(paragraph)
lemmatizer = WordNetLemmatizer()

# Lemmatization
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)

sentences
```

```
['I three vision India .',
 'In 3000 year history , people world come invaded u , captured land , conquered mind .',
 'From Alexander onwards , Greeks , Turks , Moguls , Portuguese , British , French , Dutch , came looted u , took .',
 'Yet done nation .',
 'We conquered anyone .',
 'We grabbed land , culture , history tried enforce way life .',
 'Why ?',
 'Because respect freedom others.That first vision freedom .',
 'I believe India got first vision 1857 , started War Independence .',
 'It freedom must protect nurture build .',
 'If free , one respect u .',
 'My second vision India ' development .',
 'For fifty year developing nation .',
 'It time see developed nation .',
 'We among top 5 nation world term GDP .',
 'We 10 percent growth rate area .',
 'Our poverty level falling .',
 'Our achievement globally recognised today .',
 'Yet lack self-confidence see developed nation , self-reliant self-assured .',
 'Isn ' incorrect ?',
 'I third vision .',
 'India must stand world .',
 'Because I believe unless India stand world , one respect u .',
 'Only strength respect strength .',
 'We must strong military power also economic power .',
 'Both must go hand-in-hand .',
 'My good fortune worked three great mind .',
 'Dr. Vikram Sarabhai Dept .',
 'space , Professor Satish Dhawan , succeeded Dr. Brahm Prakash , father nuclear material .',
 'I lucky worked three closely consider great opportunity life .',
 'I see four milestone career']
```

We have excluded the stop words before performing the lemmatization.

Below you can see the list of stop words in English.

Stop Words

Step1: Importing the package

```
from nltk.corpus import stopwords
```

Step2: Check the stop words in the English language.

```
stop_words = stopwords.words('english')
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',
```

POS tagging

```
data =' We will see an example of POS tagging.'

pos = nltk.pos_tag(nltk.word_tokenize(data))

pos
```

```
[('We', 'PRP'),
 ('will', 'MD'),
 ('see', 'VB'),
 ('an', 'DT'),
 ('example', 'NN'),
 ('of', 'IN'),
 ('POS', 'NNP'),
 ('tagging', 'NN'),
 ('.', '.')]
```

Example2:

Step1: Importing the necessary packages

```
# We also have punctuations which we can ignore from our set of words just like stopwords.

import string
import nltk
import string
from nltk.corpus import stopwords

punct =string.punctuation
punct
```

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

Step2: Let's clean the data before we perform the POS tagging

```
# Let's word tokenize the given sample after we remove the stopwords and
stop_words = stopwords.words('english')
punct =string.punctuation

data = '''Abraham Lincoln was an American lawyer and statesman who
served as the 16th president of the United States from 1861 until his
assassination in 1865. Lincoln led the nation through the American
Civil War, the country's greatest moral, cultural, constitutional,
and political crisis. He succeeded in preserving the Union, abolishing
slavery, bolstering the federal government, and modernizing the
U.S. economy.'''
clean_data =[]

for word in nltk.word_tokenize(data):
    if word not in punct:
        if word not in stop_words:
            clean_data.append(word)

print(clean_data)
```

Below is the sample of cleaned data

```
['Abraham', 'Lincoln', 'American', 'lawyer', 'statesman', 'served', '1
6th', 'president', 'United', 'States', '1861', 'assassination', '1865
', 'Lincoln', 'led', 'nation', 'American', 'Civil', 'War', 'country',
"'s", 'greatest', 'moral', 'cultural', 'constitutional', 'political',
'crisis', 'He', 'succeeded', 'preserving', 'Union', 'abolishing', 'sla
very', 'bolstering', 'federal', 'government', 'modernizing', 'U.S.', '
economy']
```

Great! Our data looks so much cleaner now after removing stop words and punctuation. I Hope, this clears up why we should remove stop words and punctuation before processing data.

Step3: Let's see POS tagging for cleaned data

```
nltk.pos_tag(clean_data)
```

```
[('Abraham', 'NNP'),
 ('Lincoln', 'NNP'),
 ('American', 'NNP'),
 ('lawyer', 'NN'),
 ('statesman', 'NN'),
 ('served', 'VBD'),
 ('16th', 'CD'),
 ('president', 'NN'),
 ('United', 'NNP'),
 ('States', 'NNPS'),
 ('1861', 'CD'),
 ('assassination', 'NN'),
 ('1865', 'CD'),
 ('Lincoln', 'NNP'),
 ('led', 'VBD'),
 ('nation', 'NN'),
 ('American', 'NNP'),
 ('Civil', 'NNP'),
 ('War', 'NNP'),
 ('country', 'NN'),
 ("'s", 'POS'),
```

Name Entity Recognition

In NER, we try to group entities like people, places, countries, things, etc. together

```
pos_tag = nltk.pos_tag(clean_data)
namedEntity = nltk.ne_chunk(pos_tag)
print(namedEntity)
```

```
(S
  (PERSON Abraham/NNP)
  (PERSON Lincoln/NNP American/NNP)
  lawyer/NN
  statesman/NN
  served/VBD
  16th/CD
  president/NN
  (GPE United/NNP States/NNPS)
  1861/CD
  assassination/NN
  1865/CD
  (PERSON Lincoln/NNP)
  led/VBD
  nation/NN
  (ORGANIZATION American/NNP)
  Civil/NNP
  War/NNP
  country/NN
  's/POS
  greatest/JJS
  moral/JJ
  cultural/JJ
  constitutional/JJ
  political/JJ
  crisis/NN
  He/PRP
```

Commonly used Types of Named Entities

```
[('European', 'JJ'),
 ('authorities', 'NNS'),
 ('fined', 'VBD'),
 ('Google', 'NNP'),
 ('a', 'DT'),
 ('record', 'NN'),
 ('$', '$'),
 ('5.1', 'CD'),
 ('billion', 'CD'),
 ('on', 'IN'),
 ('Wednesday', 'NNP'),
 ('for', 'IN'),
 ('abusing', 'VBG'),
 ('its', 'PRP$'),
 ('power', 'NN'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('mobile', 'JJ'),
 ('phone', 'NN'),
 ('market', 'NN'),
 ('and', 'CC'),
 ('ordered', 'VBD'),
 ('the', 'DT'),
 ('company', 'NN'),
 ('to', 'TO'),
 ('alter', 'VB'),
 ('its', 'PRP$'),
 ('practices', 'NNS')]
```

With the help of NET, we can select any category from a given word document or sentence. Suppose we need all the names mentioned in a document, we can use NER and select the words with the tag “Person”.

Applying NLTK text preprocessing technique on the Twitter dataset.

Importing the “cleaned_tweets” dataset which we save in our previous chapter.

```
import pandas as pd
tweets = pd.read_csv("Cleaned_tweets.csv")
tweets.head()
```

	Unnamed: 0	Tweets	Analysis
0	0	Native of East Hull . Former : police research...	Neutral
1	1	on colonial & - the which formed part of the -...	Neutral
2	2	The Interesting With Colour & Style	Positive
3	3	Next is the daily from Daily group . With a mi...	Neutral
4	4	This is the official handle of the State Covid...	Neutral

Let's check the length of the dataset and any null values in the dataset.

```
len(tweets)
```

```
4956
```

```
tweets.isnull().sum()
```

```
Unnamed: 0      0
Tweets         24
Analysis        0
dtype: int64
```

There are 24 null values in our "Tweets" column. We must drop the null values before we proceed further.

```
tweets.dropna(inplace=True)
```

```
tweets.isnull().sum()
```

```
Unnamed: 0      0
Tweets          0
Analysis        0
dtype: int64
```

We have dropped all the missing values, now we can go ahead with performing the NLTK text preprocessing techniques.

Converting our Tweets into a list to performing text preprocessing

```
tweet = df['Tweets'].to_list()
```

```
tweet[0:5]
```

```
['Native of East Hull . Former : police researcher , financial adviser
, logistics manager , microbiologist , viola player socialist',
 'on colonial & - the which formed part of the - post & independence n
ot in chronological order',
 'The Interesting With Colour & Style',
 'Next is the daily from Daily group . With a micro - focus on & , Nex
t incisive coverage on politics , sports , & .',
 'This is the official handle of the State Covid19 Emergency Operation
.']
```

Let's start with removing all the characters other than the string and convert the string into lower case.

```
corpus = []
for i in range(0, len(tweet)):
    review = re.sub(r'[^a-zA-Z]', ' ', tweet[i])
    review = review.lower()
    corpus1.append(review)
```

We created an empty list called "corpus" to save our output. The first line in the "for loop" replaces all the characters which are not strings with blank space, next we convert the string in the lower case and finally append the results to the corpus.

```
print(corpus1[:5])
```

```
['native of east hull   former   police researcher   financial adviser
logistics manager   microbiologist   viola player socialist', 'on colo
nial     the which formed part of the   post   independence not in chr
onological order', 'the interesting with colour   style', 'next is the
daily from daily group   with a micro   focus on     next incisive cov
erage on politics   sports      ', 'this is the official handle of the
state covid   emergency operation  ']
```

Let's join the list of strings into a single string

```
corpus = " ".join(output)
```

```
corpus[:1000]
```

```
'native of east hull former police researcher financial adviser logist
ics manager microbiologist viola player socialist on colonial the whic
h formed part of the post independence not in chronological order the
interesting with colour style next is the daily from daily group with
a micro focus on next incisive coverage on politics sports this is the
official handle of the state covid emergency operation health politics
social history world citizen eu my own miller drinking chicken eating
dress so fly water and firm in consulting treatment plant and collecti
on and construction you news and on current latest news from stock and
global real time information and nifty bank deputy news editor at news
author of life in the clock tower valley by in march order at logistic
s in cargo freight clearing overseas cargo pickup and car importation
we respond to within call the narrative on the continent investigative
and narrative journalism the heart of our strategy a platform for thou
ght leadership a'
```

We can go ahead with word tokenization now.

```
words = nltk.word_tokenize(corpus)

print(words[:20])
```

```
['native', 'of', 'east', 'hull', 'former', 'police', 'researcher', 'fi
nancial', 'adviser', 'logistics', 'manager', 'microbiologist', 'viola
', 'player', 'socialist', 'on', 'colonial', 'the', 'which', 'formed']
```

Can you see words like “of/on/the/which” in our corpus? Those words are called stop words which don’t have any importance in our analysis. We can go ahead and remove those words from our corpus.

```
clean_data =[]

for word in nltk.word_tokenize(corpus):
    if word not in punct:
        if word not in stop_words:
            clean_data.append(word)

print(clean_data[:20])
```

```
['native', 'east', 'hull', 'former', 'police', 'researcher', 'financia
l', 'adviser', 'logistics', 'manager', 'microbiologist', 'viola', 'pla
yer', 'socialist', 'colonial', 'formed', 'part', 'post', 'independence
', 'chronological']
```

We tokenized the sentence into words then removed all the stop words along with punctuations if any.

Let’s move on to POS tagging for each tokenized word.

```
pos_tags = nltk.pos_tag(clean_data)

print(pos_tags[:10])
```

```
[('native', 'JJ'), ('east', 'NN'), ('hull', 'NN'), ('former', 'JJ'),
('police', 'NN'), ('researcher', 'NN'), ('financial', 'JJ'), ('adviser
', 'NN'), ('logistics', 'NNS'), ('manager', 'NN')]
```

You can see the tags are pretty accurate. Now let's do stemming.

```
from nltk.stem.porter import PorterStemmer
pe = PorterStemmer()

stem = []
for word in clean_data:
    stemmer = pe.stem(word)
    stem.append(stemmer)

stem[:5]
```

```
['nativ', 'east', 'hull', 'former', 'polic']
```

After performing "PosterStemming", words like native and police are mapped to "nativ" and "polic' by removing the suffixes.

Let's perform lemmatization and see how the words are mapped.

```
from nltk.stem import WordNetLemmatizer

lemma = WordNetLemmatizer()

lem=[]
for word in clean_data:
    lemmatize = lemma.lemmatize(word)
    lem.append(lemmatize)

lem[:5]
```

```
['native', 'east', 'hull', 'former', 'police']
```

Perfect! The words are mapped to a meaning word after lemmatization unlike stemming.

So, we have seen the different techniques of text preprocessing using the NLTK library.

SUMMARY

Natural Language Processing (NLP) refers to the AI method of communicating with intelligent systems using a natural language such as English. We studied text preprocessing techniques like tokenization, stemming, lemmatization to break the sentence and normalize the words into base form to perform sentiment analysis.

ASSIGNMENT

1. Import the dataset and perform data cleaning tasks like removing special characters, converting the words into lower case.
2. Remove stop words, create word tokens, and perform stemming technique
3. Create lemmatization mapping and generate POS tagging.

CHAPTER 4: Model Building

Theory

In the previous chapter, we saw text preprocessing like Tokenization, Stemming, Lemmatization, Stop Words, POS tagging, and Named Entity Recognition. In this chapter, we will be vectorizing the words and building a machine learning model to classify the sentiment of the tweets.

Word Vectorization (World Embedding)

Word Vectorization is the process of mapping words to a set of real numbers or vectors. This is done to process the given words using machine learning techniques and extract relevant information from them such that it can be used in further predicted words. Vectorization is done by comparing a given word to the corpus (collection) of the available words. There are many different methods used for vectorizing a given set of words. Let's see the most popular word vectorization techniques.

I'll take a popular example to explain Word Vectorization in this chapter.

We all love watching movies. I tend to always look at the reviews of a movie before I commit to watching it. I know a lot of you do the same! So, I'll use this example here.

Here's a sample of reviews about a particular horror movie:

- Review 1: This movie is very scary and long
- Review 2: This movie is not scary and is slow
- Review 3: This movie is spooky and good

You can see that there are some contrasting reviews about the movie as well as the length and pace of the movie. Imagine looking at a thousand reviews like these. Clearly, there is a lot of interesting insights we can draw from them and build upon them to gauge how well the movie performed.

However, as we saw above, we cannot simply give these sentences to a machine learning model and ask it to tell us whether a review was positive or negative. We need to perform certain text preprocessing steps.

Bag of Words (BOW)

Bag of Words is the technique of pre-processing the text by converting it into a number/vector format, which keeps a count of the total occurrences of most frequently used words in the document.

Bag-of-Words and TF-IDF are two examples of how to do this. Let's understand them in detail.

BOW model is used in NLP to represent the given text/sentence/document as a collection (bag) of words without giving any importance to grammar or the occurrence order of the words. It keeps an account of the frequency of the words in the text document, which can be used as features in many models.

Let's recall the three types of movie reviews we saw earlier:

- Review 1: This movie is very scary and long
- Review 2: This movie is not scary and is slow
- Review 3: This movie is spooky and good

We will first build a vocabulary from all the unique words in the above three reviews. The vocabulary consists of these 11 words: 'This', 'movie', 'is', 'very', 'scary', 'and', 'long', 'not', 'slow', 'spooky', 'good'.

We can now take each of these words and make their occurrence in the three movie reviews above with 1s and 0s. This will give us 3 vectors for 3 reviews:

	1 This	2 Movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	1	1	0	1	1	0	0	8
Review 3	1	1	1	0	0	1	0	0	0	1	1	6

Vector of Review 1: [1 1 1 1 1 1 1 0 0 0 0]

Vector of Review 2: [1 1 2 0 1 1 0 1 1 0 0]

Vector of Review 3: [1 1 1 0 0 1 0 0 0 1 1]

And that's the core idea behind a Bag of Words (BOW) model.

Drawbacks of using a Bag-of-words (BOW) model

In the above example, we can have vectors of length 11. However, we start facing issues when we come across new sentences:

1. If the new sentences contain new words, then our vocabulary size would increase, and thereby, the length of the vectors would increase too.
2. Additionally, the vectors would also contain many 0s, thereby resulting in a sparse matrix (which is what we would like to avoid)
3. We are retaining no information on the grammar of the sentences or on the ordering of the words in the text.
4. Sematic information is not stored in the BOW approach.

Term Frequency – Inverse Document Frequency (TF – IDF)

As discussed in the BOW model, the BOW model doesn't keep the importance of words in a given text, only the frequency of words matters. TF-IDF makes sure they have semantic meaning in each of the sentences.

Term Frequency (TF)

Let's first understand Term Frequent (TF). It is a measure of how frequently a term, t, appears in a document, d:

$$tf_{t,d} = \frac{n_{t,d}}{Number\ of\ terms\ in\ the\ document}$$

Here, in the numerator, n is the number of times the term "t" appears in the document "d". Thus, each document and term would have its own TF value.

We will again use the same vocabulary we had built in the Bag-of-Words model to show how to calculate the TF for Review #2:

Review 2: This movie is not scary and is slow

Here,

- Vocabulary: 'This', 'movie', 'is', 'very', 'scary', 'and', 'long', 'not', 'slow', 'spooky', 'good'
- Number of words in Review 2 = 8
- TF for the word 'this' = (number of times 'this appears in review 2)/(number of terms in review 2) = 1/8

Similarly,

Word	TF
movie	1/8
is	14
very	0/8
scary	1/8
and	1/8
long	0/8
not	1/8
slow	1/8
spooky	0/8
good	0/8

We can calculate the term frequencies for all the terms and all the reviews in this manner:

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

Inverse Documents Frequency (IDF)

IDF is a measure of how important a term is. We need the IDF value because computing just the TF alone is not sufficient to understand the importance of words:

$$idf_t = \log \frac{number\ of\ documents}{number\ of\ documents\ with\ term\ 't'}$$

We can calculate the IDF values for all the words in Review 2:

IDF ('this') = log number of documents/number of documents containing the word 'this')

$$\log (3/3) = \log (1)$$

$$\log (1) = 0$$

Similarly,

- IDF('movie') = log(3/3) = 0
- IDF ('is') = log(3/3) = 0
- IDF ('not') = log(3/1) = log(3) = 0.48
- IDF('scary') = log(3/2) = 0.18
- IDF('and') = log(3/3) = 0
- IDF('slow') = log(3/1) = 0.48

We can calculate the IDF values for each word like this. Thus, the IDF values for the entire vocabulary would be:

Term	Review 1	Review 2	Review 3	IDF
This	1	1	1	0.00
movie	1	1	1	0.00
is	1	2	1	0.00
very	1	0	0	0.48
scary	1	1	0	0.18
and	1	1	1	0.00
long	1	0	0	0.48
not	0	1	0	0.48
slow	0	1	0	0.48
spooky	0	0	1	0.48
good	0	0	1	0.48

Hence, we see that words like 'is', 'this', 'and', etc., are reduced to 0 and have little importance; while words like 'scary', 'long', 'good', etc. are words with importance and thus have a higher value.

We can now compute the TF-IDF score for each word in the corpus. Words with a higher score are more important, and those with a lower score are less important:

$$(tf\_idf)_{t,d} = tf_{t,d} * idf_t$$

We can now calculate the TF-IDF score for every word in Review 2:

TF-IDF ('this', Review 2) = TF ('this', Review 2) * IDF ('this')

1/8*0 = 0

Similarly,

- TF-IDF('movie', Review 2) = 1/8 * 0 = 0
- TF-IDF('is, Review 2) = 1/4 * 0 = 0
- TF-IDF('not', Review 2) = 1/8 * 0.48 = 0.06
- TF-IDF('scary, Review 2) = 1/8 * 0.18 = 0
- TF-IDF('and, Review 2) = 1/8 * 0 = 0.023
- TF-IDF('slow', Review 2) = 1/8 * 0.48 = 0.06

Similarly, we can calculate the TF-IDF scores for all the words with respect to all the reviews:

Term	Review 1	Review 2	Review 3	IDF	TF-IDF (Review 1)	TF-IDF (Review 2)	TF-IDF (Review 3)
This	1	1	1	0.00	0.000	0.000	0.000
movie	1	1	1	0.00	0.000	0.000	0.000
is	1	2	1	0.00	0.000	0.000	0.000
very	1	0	0	0.48	0.068	0.000	0.000
scary	1	1	0	0.18	0.025	0.022	0.000
and	1	1	1	0.00	0.000	0.000	0.000
long	1	0	0	0.48	0.068	0.000	0.000
not	0	1	0	0.48	0.000	0.060	0.000
slow	0	1	0	0.48	0.000	0.060	0.000
spooky	0	0	1	0.48	0.000	0.000	0.080
good	0	0	1	0.48	0.000	0.000	0.080

We have now obtained the TF-IDF scores for our vocabulary. TF-IDF also gives larger values for less frequent words and is high when both IDF and TF values are high i.e., the word is rare in all the documents combined but frequent in a single document.

Bag of Words

Step1: Importing the required packages

```
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
```

Step2: Assign the text data to a variable

```
paragraph =  """I have three visions for India. In 3000 years of our history, people from all over
              the world have come and invaded us, captured our lands, conquered our minds.
              From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,
              the French, the Dutch, all of them came and looted us, took over what was ours.
              Yet we have not done this to any other nation. We have not conquered anyone.
              We have not grabbed their land, their culture,
              their history and tried to enforce our way of life on them.
              Why? Because we respect the freedom of others.That is why my
              first vision is that of freedom. I believe that India got its first vision of
              this in 1857, when we started the War of Independence. It is this freedom that
              we must protect and nurture and build on. If we are not free, no one will respect us.
              My second vision for India's development. For fifty years we have been a developing nation.
              It is time we see ourselves as a developed nation. We are among the top 5 nations of the world
              in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling.
              Our achievements are being globally recognised today. Yet we lack the self-confidence to
              see ourselves as a developed nation, self-reliant and self-assured. Isn't this incorrect?
              I have a third vision. India must stand up to the world. Because I believe that unless India
              stands up to the world, no one will respect us. Only strength respects strength. We must be
              strong not only as a military power but also as an economic power. Both must go hand-in-hand.
              My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of
              space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material.
              I was lucky to have worked with all three of them closely and consider this the great opportunity of my life.
              I see four milestones in my career"""
```

Step3: Clean the text

```
ps = PorterStemmer()
wordnet=WordNetLemmatizer()
sentences = nltk.sent_tokenize(paragraph)
corpus = []
for i in range(len(sentences)):
    review = re.sub('[^a-zA-Z]', ' ', sentences[i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

We check the corpus to see how the cleaning as performed

```
corpus[:4]
```

```
['three vision india',
 'year histori peopl world come invad us captur land conquer mind',
 'alexand onward greek turk mogul portugues british french dutch came loot us took',
 'yet done nation']
```

We can compare the corpus with the original sentence

```
sentences[:4]
```

```
['I have three visions for India.',
 'In 3000 years of our history, people from all over \n                the world have come and i
 'From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,\n
 'Yet we have not done this to any other nation.']
```

The first sentence "I have three visions for India" got converted into "three vision India" after data cleaning.

In the second sentence “In 3000 years of our history, people from all over” which got converted to “year histori peopl world come invad us captur land conquer mind”.

You can see “history” is converted to “histori”, the stemming is not giving a meaningful word. What we can do is instead of doing stemming, we can do lemmatization.

```
ps = PorterStemmer()
wordnet=WordNetLemmatizer()
sentences = nltk.sent_tokenize(paragraph)
corpus = []
for i in range(len(sentences)):
    review = re.sub('[^a-zA-Z]', ' ', sentences[i])
    review = review.lower()
    review = review.split()
    review = [wordnet.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

Now let’s compare a sentence with a corpus

```
sentences[:4]
```

```
['I have three visions for India.',
 'In 3000 years of our history, people from all over \n                the world have come and i
 'From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,\n
 'Yet we have not done this to any other nation.']
```

```
corpus[:4]
```

```
['three vision india',
 'year history people world come invaded u captured land conquered mind',
 'alexander onwards greek turk mogul portuguese british french dutch came looted u took',
 'yet done nation']
```

Here we can see the “history” is converted to “history”, “years” to “year”. So, lemmatization makes sure that the intermediate representation will definitely have meaning.

Step4: Creating the Bag of Words model

```
cv = CountVectorizer(max_features = 1500)
X = cv.fit_transform(corpus).toarray()
```

We are using the ''CountVectorizer '' function from the “Sklearn“ library. We will be responsible for creating a vector matrix.

We created a “cv” object and fitted a bag of words using the “fit_transfrom” method.

We are going to apply “fit_transform” to our new corpus.

If we print the “X”, we can see the numerical representation of the text

```
X
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 1, 0],
       [0, 1, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

```
X.shape
```

```
(31, 114)
```

We can see the shape of an array is 31X114, which means we have 31 sentences and 114 words.

Each row is a sentence, and each column is a word. If there is a value 1, then it represents that the word is present once in that sentence. If there is a value of three then it represents that word is present thrice in that sentence.

TI-IDF

Step1: Importing the required packages

```
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

Step2: We will be using the same text data which we used for the bag of words.

```
paragraph =  """I have three visions for India. In 3000 years of our history, people from all over
              the world have come and invaded us, captured our lands, conquered our minds.
              From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,
              the French, the Dutch, all of them came and looted us, took over what was ours.
              Yet we have not done this to any other nation. We have not conquered anyone.
              We have not grabbed their land, their culture,
              their history and tried to enforce our way of life on them.
              Why? Because we respect the freedom of others.That is why my
              first vision is that of freedom. I believe that India got its first vision of
              this in 1857, when we started the War of Independence. It is this freedom that
              we must protect and nurture and build on. If we are not free, no one will respect us.
              My second vision for India's development. For fifty years we have been a developing nation.
              It is time we see ourselves as a developed nation. We are among the top 5 nations of the world
              in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling.
              Our achievements are being globally recognised today. Yet we lack the self-confidence to
              see ourselves as a developed nation, self-reliant and self-assured. Isn't this incorrect?
              I have a third vision. India must stand up to the world. Because I believe that unless India
              stands up to the world, no one will respect us. Only strength respects strength. We must be
              strong not only as a military power but also as an economic power. Both must go hand-in-hand.
              My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of
              space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material.
              I was lucky to have worked with all three of them closely and consider this the great opportunity of my life.
              I see four milestones in my career"""
```

Step3: Data cleaning

```
ps = PorterStemmer()
wordnet=WordNetLemmatizer()
sentences = nltk.sent_tokenize(paragraph)
corpus = []
for i in range(len(sentences)):
    review = re.sub('[^a-zA-Z]', ' ', sentences[i])
    review = review.lower()
    review = review.split()
    review = [wordnet.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

Step4: Creating the TF-IDF model

```
cv = TfidfVectorizer()
X = cv.fit_transform(corpus).toarray()
```

```
X
```

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.25883507, 0.30512561,
        0.        ],
       [0.        , 0.28867513, 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

Let's move to our Twitter sentiment analysis problem and build a machine learning model to predict the sentiment of the tweet.

Import Necessary Dependencies

We start with importing necessary packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import re
import nltk
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
# %matplotlib inline
```

Read and Load the Dataset

Importing the dataset which we have cleaned in our previous chapter.

```
df = pd.read_csv("D:\Sentiment Analysis\Code\Cleaned_tweets.csv")
df.head()
```

	Unnamed: 0	Tweets	Analysis
0	0	Native of East Hull . Former : police research...	Neutral
1	1	on colonial & - the which formed part of the -...	Neutral
2	2	The Interesting With Colour & Style	Positive
3	3	Next is the daily from Daily group . With a mi...	Neutral
4	4	This is the official handle of the State Covid...	Neutral

Let’s drop the first column in our dataset which is a duplicate of the index column.

```
df = df.iloc[:,1:]
df.head()
```

	Tweets	Analysis
0	Native of East Hull . Former : police research...	Neutral
1	on colonial & - the which formed part of the -...	Neutral
2	The Interesting With Colour & Style	Positive
3	Next is the daily from Daily group . With a mi...	Neutral
4	This is the official handle of the State Covid...	Neutral

Let's see the length of the data frame.

```
len(df)
```

```
4956
```

Do we have any missing values in the dataset? Let's check for missing values.

```
df.isnull().sum()
```

```
Tweets      24
Analysis     0
dtype: int64
```

There are few missing values in the "Tweets" variable, we should drop all the missing values to go ahead with our analysis.

```
df.dropna(inplace=True)
```

```
df.isnull().sum()
```

```
Tweets      0
Analysis    0
dtype: int64
```

We can see all the missing values are dropped.

Data Preprocessing

In the above-given problem statement before training the model, we have to perform various pre-processing steps on the dataset that mainly dealt with removing stopwords, removing emojis, converting text documents into lowercase for better generalization.

Subsequently, the punctuations were cleaned and removed thereby reducing the unnecessary noise from the dataset. We are also going to remove the repeating characters from the words along with removing the URLs as they do not have any significant importance.

At last, we will then perform word Tokenization, Lemmatization, and word cloud visualization

Text lowering

```
df['Tweets'] = df['Tweets'].str.lower()
df['Tweets'].head()
```

```
0    native of east hull . former : police research...
1    on colonial & - the which formed part of the -...
2                  the interesting with colour & style
3    next is the daily from daily group . with a mi...
4    this is the official handle of the state covid...
Name: Tweets, dtype: object
```

Removing the stopwords

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

df['Tweets'] = df['Tweets'].apply(lambda text: cleaning_stopwords(text))
df['Tweets'].head()
```

```
0    native east hull . former : police researcher ...
1    colonial & - formed part - post & independence...
2                          interesting colour & style
3    next daily daily group . micro - focus & , nex...
4    official handle state covid19 emergency operat...
Name: Tweets, dtype: object
```

Removing Emoji's

As we can see there are few emoji in the text. We must remove those for our analysis.

```
df['Tweets'].tail()
```

```
4951    green - eyed 🇯🇲🇮🇪 mulatto l . . 😎 leftist . p...
4952    • efficiency ⚡ strategy • 📋 ~ 2008 • r & • r ...
4953    passionate : body - mind health , food medicin...
4954    ' super important , ' definitely want read eve...
4955                                                news
Name: Tweets, dtype: object
```

```
import re
def remove_emojis(data):
    emoj = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002500-\U00002BEF"  # chinese char
        u"\U00002702-\U000027B0"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U00010000-\U0010ffff"
        u"\u2640-\u2642"
        u"\u2600-\u2B55"
        u"\u200d"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\ufe0f"  # dingbats
        u"\u3030"
                      "]+", re.UNICODE)
    return re.sub(emoj, '', data)
df['Tweets'] = df['Tweets'].apply(lambda x: remove_emojis(x))
df['Tweets'].tail()
```

```
4951    green - eyed   mulatto l . .  leftist . progre...
4952    • efficiency  strategy •  ~ 2008 • r & • r ~ 2...
4953    passionate : body - mind health , food medicin...
4954    ' super important , ' definitely want read eve...
4955                                                 news
Name: Tweets, dtype: object
```

Removing the special characters

```
import re
def nospecial(text):
    text = re.sub("[^a-zA-Z]+", " ",text)
    return text

df['Tweets'] = df['Tweets'].apply(lambda x: nospecial(x))
```

```
df['Tweets'].tail()
```

```
4951    green eyed mulatto l leftist progressive indep...
4952     efficiency strategy r r event advice born ppm...
4953    passionate body mind health food medicine poli...
4954     super important definitely want read everythi...
4955                                                news
Name: Tweets, dtype: object
```

Removing the Punctuations

```
import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
df['Tweets']= df['Tweets'].apply(lambda x: cleaning_punctuations(x))
df['Tweets'].head()
```

```
0    native east hull former police researcher fina...
1    colonial formed part post independence chronol...
2                              interesting colour style
3    next daily daily group micro focus next incisi...
4     official handle state covid emergency operation
Name: Tweets, dtype: object
```

Cleaning the repeated characters

```
import re
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
df['Tweets'] = df['Tweets'].apply(lambda x: cleaning_repeating_char(x))
df['Tweets'].head()
```

```
0    native east hull former police researcher fina...
1    colonial formed part post independence chronol...
2                              interesting colour style
3    next daily daily group micro focus next incisi...
4     official handle state covid emergency operation
Name: Tweets, dtype: object
```

Removing the URL's

```
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
df['Tweets'] = df['Tweets'].apply(lambda x: cleaning_URLs(x))
df['Tweets'].head()
```

```
0    native east hull former police researcher fina...
1    colonial formed part post independence chronol...
2                              interesting colour style
3    next daily daily group micro focus next incisi...
4     official handle state covid emergency operation
Name: Tweets, dtype: object
```

Data Visualization

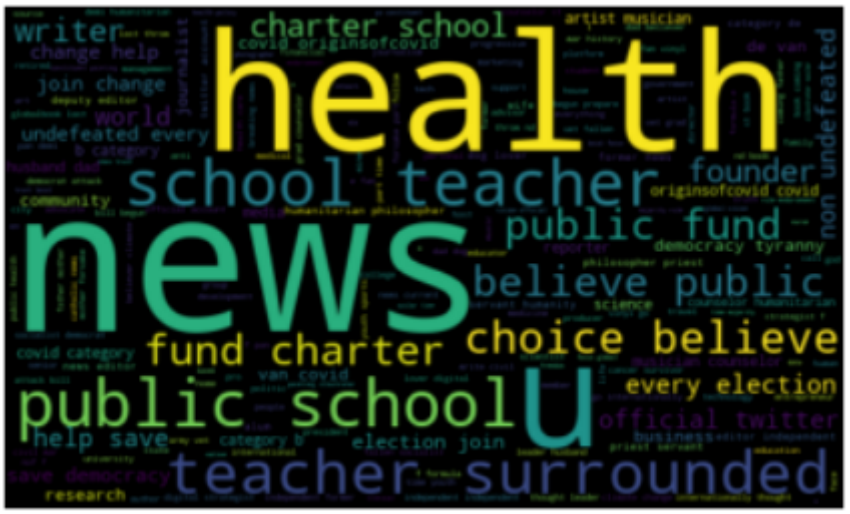
We can create a word cloud to visualize the words used in sentiment.

```
data_pos = df[df['Analysis'] == 'Positive']
data_neg = df[df['Analysis'] == 'Negative']
data_nut = df[df['Analysis'] == 'Neutral']
```

Data has been filtered based on the sentiment type. Let's visualize the type of words used mostly in each sentiment.

```
: from wordcloud import WordCloud

allWords_neutral = ' '.join(twts for twts in data_nut['Tweets'])
wordCloud_neutral = WordCloud(width=500, height=300, random_state=21,
                      max_font_size=119).generate(allWords_neutral)
plt.imshow(wordCloud_neutral, interpolation='bilinear')
plt.axis('off')
plt.show()
```



The words like Health, News, School, Teacher, and so on are mostly used in neutral sentiment tweets.

```
allWords_neg = ' '.join(twts for twts in data_neg['Tweets'])
wordCloud_neg = WordCloud(width=500, height=300, random_state=21,
                    max_font_size=119).generate(allWords_neg)
plt.imshow(wordCloud_neg, interpolation='bilinear')
plt.axis('off')
plt.show()
```

The words like Common Sense, Advocate, Paranormal, Mental health were most frequently used in negative sentiment tweets.

```
allWords_pos = ' '.join(twts for twts in data_pos['Tweets'])
wordCloud_pos = WordCloud(width=500, height=300, random_state=21,
                    max_font_size=119).generate(allWords_pos)
plt.imshow(wordCloud_pos, interpolation='bilinear')
plt.axis('off')
plt.show()
```

The words like Love, Wade, Covid Supporter were most frequently used in positive sentiment tweets.

Word Vectorization

We are using the TF-IDF model to create a vectorization of text and build a machine learning model using these vectors.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

```
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
final_vectorized_data=vectoriser.fit_transform(df['Tweets'])
final_vectorized_data
```

```
<4932x20934 sparse matrix of type '<class 'numpy.float64'>'
        with 73108 stored elements in Compressed Sparse Row format>
```

Train and Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(final_vectorized_data, df['Analysis'],
                                                    test_size=0.2, random_state=69)
```

We split the data into train and test. We use 80% of our data to train and 20% of our data to test our model.

```
print("X_train_shape : ",X_train.shape)
print("X_test_shape : ",X_test.shape)
print("y_train_shape : ",y_train.shape)
print("y_test_shape : ",y_test.shape)
```

```
X_train_shape :  (3945, 20934)
X_test_shape :  (987, 20934)
y_train_shape :  (3945,)
y_test_shape :  (987,)
```

We are using Naive Bayes Classifier to train our Twitter sentiment analysis model.

Naïve Bayes Classifier

Naïve Bayes classifier is a family of simple "probabilistic classifiers" based on applying Bayes theorem with strong (naïve) independence assumptions between the features.

Bayes's Theorem

In probability theory and statistics, "Bayes's Theorem" describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Mathematically, it can be written as

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Where A and B are events and P(B)≠0

- P(A/B) is a conditional probability: the likelihood of event A occurring given that B is true.
- P(B/A) is also conditional probability: the likelihood of event B occurring given that A is true.

- P(A) and P(B) are the probabilities of observing A and B respectively; they are known as the marginal probability.

```
from sklearn.naive_bayes import MultinomialNB  # Naive Bayes Classifier

model_naive = MultinomialNB().fit(X_train, y_train)
```

```
predicted_naive = model_naive.predict(X_test)
```

We have fitted our Naïve Bayes model for our training data. Let's make the prediction on the test data and check the performance of the model.

```
from sklearn import metrics
print("Confusion Metrics\n",metrics.confusion_matrix(y_test,predicted_naive), end="\n\n\n")

print("Classification Report\n",metrics.classification_report(y_test,predicted_naive), end="\n\n\n")

print("Accuracy Score:", metrics.accuracy_score(y_test,predicted_naive))
```

```
Confusion Metrics
 [[ 36  43  38]
 [  0 388  65]
 [  0  62 355]]


Classification Report
              precision    recall  f1-score   support

    Negative       1.00      0.31      0.47       117
     Neutral       0.79      0.86      0.82       453
    Positive       0.78      0.85      0.81       417

    accuracy                           0.79       987
   macro avg       0.85      0.67      0.70       987
weighted avg       0.81      0.79      0.78       987



Accuracy Score: 0.7892603850050659
```

As we can see, the model performed well in classifying the Positive and Neutral sentiments and could not perform well in classifying Negative sentiment tweets.

This could be because of class imbalance; we can overcome this by increasing the Negative labeled data in our training set.

Let's try KNN Classifier and check how the model performs on this.

```
from sklearn.neighbors import KNeighborsClassifier
Knn_clf = KNeighborsClassifier()

Knn_clf.fit(X_train, y_train)
```

```
KNN_predictions = Knn_clf.predict(X_test)
```

```
print("Confusion Metrics\n",metrics.confusion_matrix(y_test,KNN_predictions), end="\n\n\n")

print("Classification Report\n",metrics.classification_report(y_test,KNN_predictions), end="\n\n\n")

print("Accuracy Score:", metrics.accuracy_score(y_test,KNN_predictions))
```

```
Confusion Metrics
 [[ 34  83   0]
 [  0 453   0]
 [  0 258 159]]


Classification Report
              precision    recall  f1-score   support

    Negative       1.00      0.29      0.45       117
     Neutral       0.57      1.00      0.73       453
    Positive       1.00      0.38      0.55       417

    accuracy                           0.65       987
   macro avg       0.86      0.56      0.58       987
weighted avg       0.80      0.65      0.62       987


Accuracy Score: 0.6545086119554204
```

KNN classifier could not perform well on our dataset.

SUMMARY

We have started with word vectorization and different types of word vectorization techniques like a bag of words and TF-IDF. Then, we have imported the data set to perform Twitter sentiment analysis. Once data is imported, we performed all the data cleaning steps and visualized the cleaned text. Now we have the clean text, and we went ahead with the model building step. We have used a Naïve Bayes classifier and KNN classifier models and tested the models on the test set and evaluated the performance of the models.

ASSIGNMENT

1. Import the dataset and perform data cleaning tasks like removing special characters, converting the words into lower case.
2. Split the data set into train and test sets.
3. Fit the Naïve Bayes Classifier model and evaluate the performance of the model.

REFERENCE

- https://www.nltk.org/book/

- https://en.wikipedia.org/wiki/Bag-of-words_model
- https://en.wikipedia.org/wiki/Tf%E2%80%93idf
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier