Performing Sentiment Analysis on Tweets using Python

helloml.org/performing-sentiment-analysis-on-tweets-using-python/

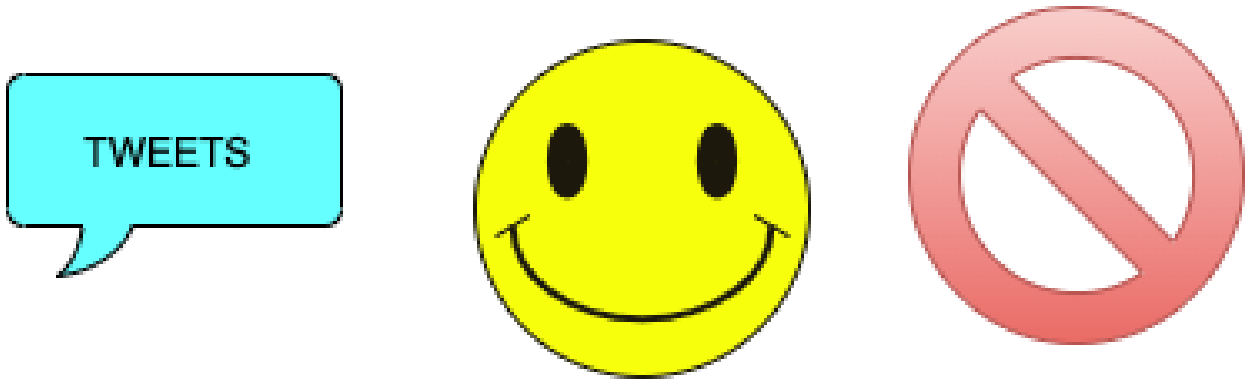
November 22, 2021

Today we are going to learn everything about text-data mining and processing (performing sentiment analysis) using standard Python libraries and a few other 3rd party modules. We will extract and analyse over 100,000 tweets posted on Twitter. Additionally, we will also learn about hosting our datasets online and then fetching them back to our working notebook without storing them locally (using Google Drive and Cloud SDK).

Introduction to Sentiment Analysis

Sentiment analysis can be considered as the use of natural language processing, text analysis and computational linguistics to identify and extract sentiment information in the source materials. Generally, sentiment analysis aims to find the attitude of a writer with respect to any relevant topic or the overall contextual polarity of a document. In today's article, we will extract data from Twitter (Tweets containing the keywords "united nations") and then perform Sentiment Analysis on the data so as to know the general public opinion of the United Nations (UN) over a period of time is.

The main task in sentiment analysis is classifying the **polarity** of a given text at the document, sentence, or feature level whether the expressed opinion in a document, a sentence or a feature is **positive**, **negative**, or **neutral**. Document-level sentiment analysis is the classification of the overall sentiments mentioned by the reviewer in the whole document text in positive, negative or neutral classes.



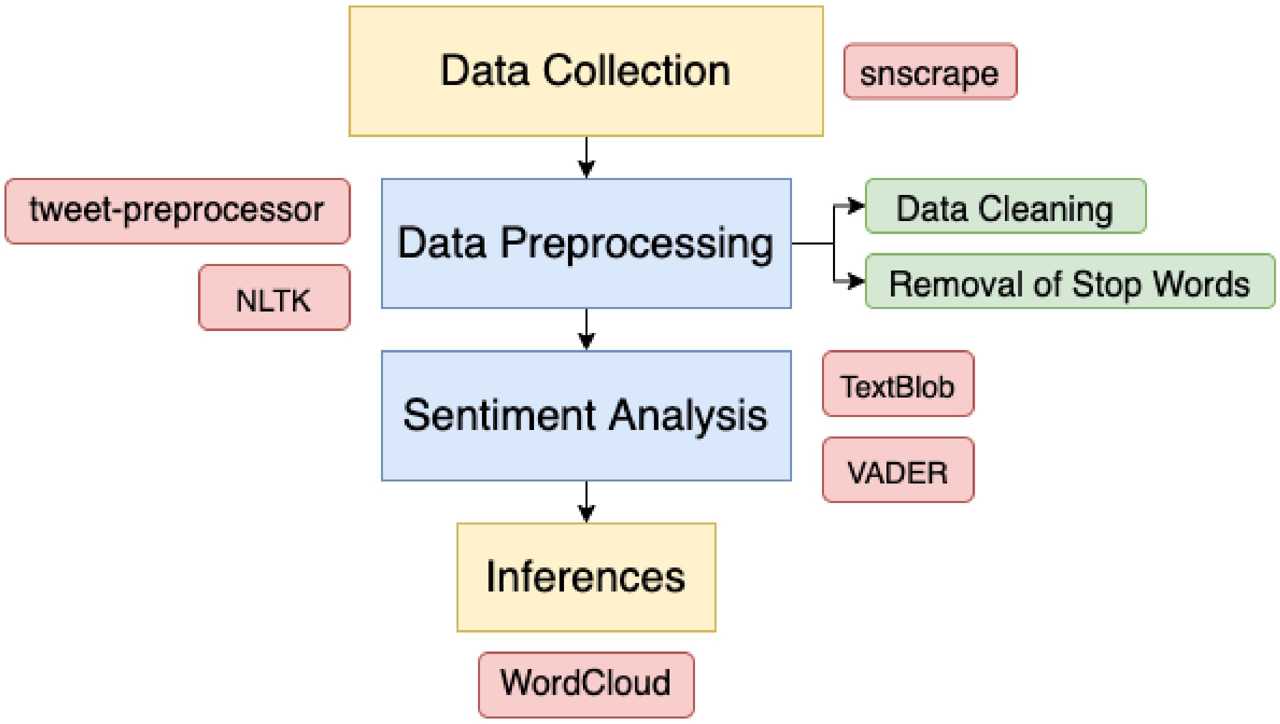
Any piece of text data (in our case, tweets) can be classified as having one of 3 sentiments: Positive, Negative (and not displayed here – Neutral).

There are mainly two approaches to extract the sentiment from given reviews and classify the result as positive, neutral or negative: **Lexicon** basedapproach and a **Machine Learning based** approach. The machine learning approach applies ML algorithms and uses **linguistic features**. The Lexicon-based Approach relies on a **sentiment lexicon**, a collection of known sentiment terms. It is divided into **dictionary-based** approaches and

corpus-based approaches which use statistical or semantic methods to find **sentiment polarity**. The **accuracy** of sentiment analysis is based on how well it agrees with human judgements. Let’s begin analysing some Twitter data!

Workflow

First, we will scrape data from Twitter. Next, we perform text preprocessing operations on our collected data. Then, we perform sentiment analysis and finally, extract inferences from our performed analyses.



Overall workflow used in our analysis

Beginning with Twitter Sentiment Analysis: Scraping Twitter Data (Using snscrape)

Sentiment analysis has shown its impact on social media websites such as Twitter, Facebook to understand user opinion and gauge a general view of public stance over certain issues/topics. Thus, Sentiment Analysis on mass twitter data can help us with graphing the public’s general stand over issues and gain a better understanding of public opinion over certain matters. However, to perform any kind of analysis/textual processing, we would require data and in this case, real-world tweets collected straight from Twitter. There are many ways to do it including using the most popular method of using the official **Twitter Data API** via **Tweepy**, but we can also use many other 3rd party modules to achieve the same and in our case, we’ll use **snscrape**, a very handy module which enables users to scrape data very easily from different social networking websites to collect and assemble our dataset.

In order to scrape Twitter data from snscrape, we’ll just search for all tweets based on keyword search terms posted on Twitter between a certain time period and then collect all the data into a **Pandas**‘ dataframe. We then export the dataset into a **.csv** file and host it on **Google Drive** so that we can access the data online from our **Colab notebook**.

We start our analysis by collecting and then assembling our United Nations Tweets' dataset. We will extract the top 100,000 tweets worldwide containing the keyword "**united nations**" (case insensitive) between 25-10-2021 and 02-11-2021.

Assuming you have **git** installed, enter the following command on Terminal/using Command Prompt. If not, then install git first to enable this installation of the developer's version of snscrape.

```
pip install git+https://github.com/JustAnotherArchivist/snscrape.git
```

Scraping Tweets via a Text Search Query

We will separate the task of data extraction and analysis into different files. First, we begin by importing the required dependencies.

```
import snscrape.modules.twitter as sntwitter
import pandas as pd
```

Next, we define an empty list. Then, we will scrape data using snscrape's sntwitter.TwitterSearchScraper() and append it to this list.

```
tweets_list = []

# Scraping data and append tweets to list
for i, tweet in enumerate(
        sntwitter.TwitterSearchScraper('united nations since:2020-07-30
until:2021-11-03').get_items()):
    if i > 100000:
        break
    tweets_list.append([tweet.date, tweet.id, tweet.content, tweet.user.username,
tweet.likeCount, tweet.user.displayname, tweet.lang])
```

Then, we create a data frame using the tweets_list list and export it as a .csv file which we can then upload to Google Drive for further use.

```
# Creating the dataframe, Export .csv file
tweets_df = pd.DataFrame(tweets_list, columns=['Datetime', 'Tweet Id', 'Text',
'Username', 'Like Count', 'Display Name', 'Language'])
tweets_df.to_csv('united_nations.csv')
```

When we run this program, it will take some time to execute scraping. It took me about 45 minutes to collect 100,000 tweets. It can vary depending on the internet connection speeds and the computational power of different machines. Once data collection is complete, we can move on to perform text preprocessing.

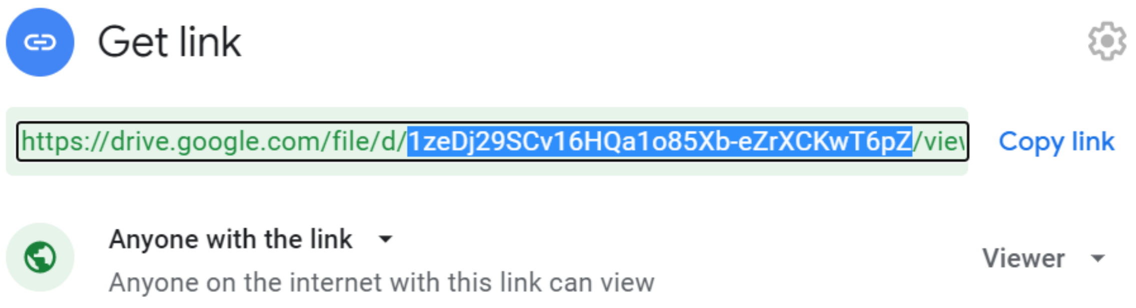
Performing Text Pre-processing

Setting up our Workspace

Before performing text processing on the collected tweets' dataset, we will need to preprocess the data. I am using Google Colab for the task but one can easily use Jupyter Notebook/Locally installed version of Python too for the task. Let's first set up the Colab notebook and import the dataset from Google Drive.

```
# Import PyDrive and associated libraries
# This only needs to be done once per notebook
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client
# This only needs to be done once per notebook
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

Once you run this cell, you will be asked to verify your Google account. Just follow the instructions shown on the screen and we're all set to fetch our .csv file from Drive. We are downloading the file from Drive using its file id. That can be fetched from right-clicking the file, fetching its URL and copying the part between "d" and "view". An example of a file id from Google Drive:


Get link
https://drive.google.com/file/d/1zeDj29SCv16HQa1o85Xb-eZrXCKwT6pZ/viev
Copy link
Anyone with the link
Anyone on the internet with this link can view
Viewer

The highlighted/selected portion represents the file id for this particular file.

After retrieving the file from the drive, we will download it locally to our Colab workspace as "tweet_data.csv".

```
# Download a file based on its file ID.

# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
file_id = '1QxkppL24m9TAT5Ix5AjJBvlK-6sfeN0V' # Check your own ID in GDrive
downloaded = drive.CreateFile({'id': file_id})

# Save file in Colab memory
downloaded.GetContentFile('tweet_data.csv')
```

Next, we download and import some necessary libraries. First, we need to install the tweet-preprocessor module. On Colab, just copy the below code. Else, a normal pip/conda install will also do the job.

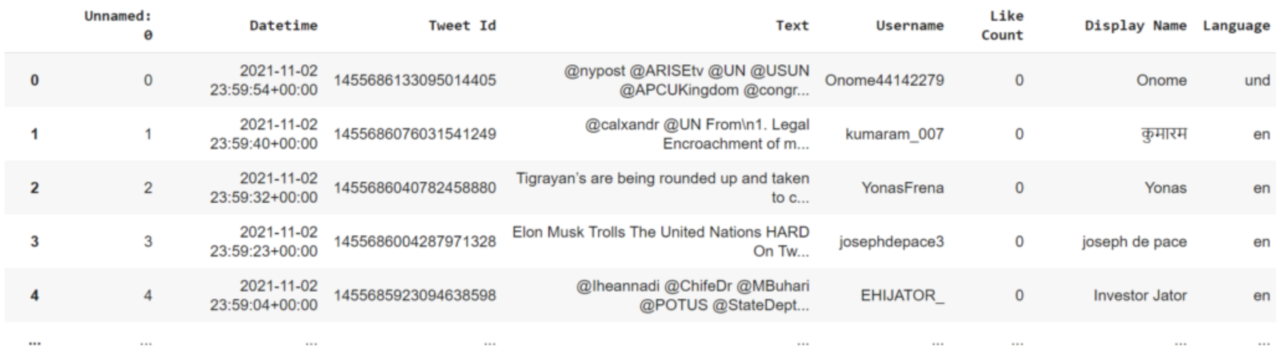
```
!pip install tweet-preprocessor
```

Importing the necessary libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import preprocessor as p
```

Reading the .csv data into a Pandas' dataframe.

```
df = pd.read_csv('tweet_data.csv',lineterminator='\n')
df
```



	Unnamed: 0	Datetime	Tweet Id	Text	Username	Like Count	Display Name	Language
0	0	2021-11-02 23:59:54+00:00	1455686133095014405	@nypost @ARISEtv @UN @USUN @APCUKingdom @congr...	Onome44142279	0	Onome	und
1	1	2021-11-02 23:59:40+00:00	1455686076031541249	@calxandr @UN From\n1. Legal Encroachment of m...	kumaram_007	0	कुमारम	en
2	2	2021-11-02 23:59:32+00:00	1455686040782458880	Tigrayan's are being rounded up and taken to c...	YonasFrena	0	Yonas	en
3	3	2021-11-02 23:59:23+00:00	1455686004287971328	Elon Musk Trolls The United Nations HARD On Tw...	josephdepace3	0	joseph de pace	en
4	4	2021-11-02 23:59:04+00:00	1455685923094638598	@Iheannadi @ChifeDr @MBuhari @POTUS @StateDept...	EHIJATOR_	0	Investor Jator	en
...	...	...	...	...	...	...	...	...

Output

Now that we have completed the setup, we can begin with our preprocessing tasks. First, we will filter out all the non-English tweets as we will restrict our analysis to English-only tweets. Along with that, we will also drop some unnecessary columns from the data frame which are of no use to our analysis.

```
df = df[df["Language"] == "en"].drop(["Tweet Id","Unnamed: 0", "Language"],
axis=1)
```

Preprocessing the Data (Using tweet-preprocessor)

First, we define a couple of functions to preprocess the tweets and also to clean the date-time data extracted from Twitter. We only require the date portion of the latter (to chart our time-series plot later on):

```
def clean_date(date_obj):
  for x in range(0, len(date_obj)):
    if date_obj[x] == " ":
      break
  date_obj = date_obj[:x]
  return date_obj

def preprocess_tweet(text):
    text = p.clean(text)
    return text
```

In the case of tweets data, it contains some twitter features such as RT tags (Retweet tags), user tags (@), hashtags (#), etc. Tweets may also contain links to external websites (URLs). These features do not affect the sentiment of the tweets and are unnecessary as well as redundant for our analysis. Tweets also contain some language-based features since on social media most of us make an informal (usage of multiple punctuations, emojis, words, unordered casings, etc.) and not a logical/structured use of language.

Our brain might comprehend this very easily but to make our models understand the meaning of such text messages we need to normalise all the messages to bring them all into a homogenous form. Thus, we need to strip the tweets off of these features and store these newly preprocessed/cleaned tweets separately and perform our analyses on this new, cleaned text data. For preprocessing operations, we are making use of the tweet-preprocessor module, a 3rd party module that cleans unnecessary features in tweets and brings them into a homogenous, “normalised” form. We are also removing unnecessary white spaces from our tweets.

```
df["Datetime"] = df["Datetime"].apply(clean_date)
df['Clean Text'] = df['Text'].apply(preprocess_tweet)
df['Clean Text']= df['Clean Text'].str.replace('[^\w\s]','')
```

Removing Stop Words

Stop words are the words that do not add any meaning to the sentence in terms of **Natural Language Processing (NLP)**. Some examples of stop words in the English language are: “I”, “me”, “my”, “myself”, “we”, “our”, “ours”, etc.

To remove the collection of stop words, we make use of the **Natural Language Toolkit (NLTK)**. NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenisation, stemming, tagging, parsing, semantic reasoning and wrappers for industrial-strength NLP libraries.

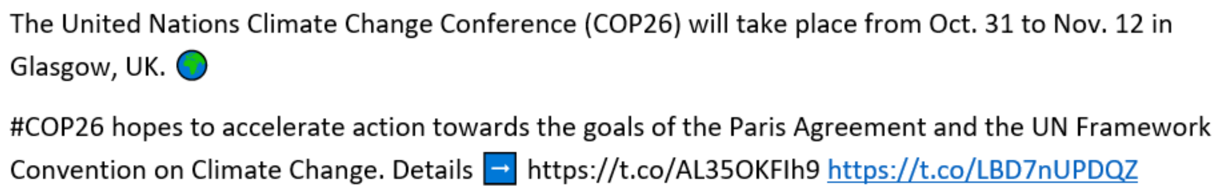
We first install and import NLTK.

```
# Removing Stop Words
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop = stopwords.words('english')
```

Then, we remove all the stop words from the cleaned tweets and finally convert all our preprocessed tweets into lower case.

```
df['Clean Text']  = df['Clean Text'].apply(lambda x: " ".join(x.lower() for x in
x.split() if x not in stop))
```

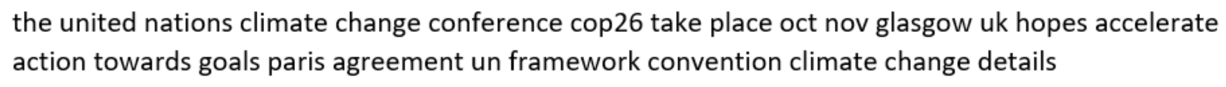
Now that our preprocessing steps are completed, let us see an example of it in action. First, a random tweet from our collected dataset:


The United Nations Climate Change Conference (COP26) will take place from Oct. 31 to Nov. 12 in Glasgow, UK. 🌍

#COP26 hopes to accelerate action towards the goals of the Paris Agreement and the UN Framework Convention on Climate Change. Details ➡️ https://t.co/AL35OKFIh9 https://t.co/LBD7nUPDQZ

Original Tweet

Next, we will see how it is transformed at the end after the completion of all our preprocessing steps.


the united nations climate change conference cop26 take place oct nov glasgow uk hopes accelerate action towards goals paris agreement un framework convention climate change details

Preprocessed/Cleaned Tweet

Now that preprocessing is done, we move on to the next step, which is actually performing sentiment analysis.

Performing Sentiment Analysis

To perform sentiment analysis, we will use two different rule-based models/sentiment classifiers, TextBlob and VADER. The reason we are using both is that these two are very popular and fast sentiment classifiers and by using two different classifiers, even though their individual results may be different, we can confirm the general trend of public sentiments (Using both classifiers, sentiments should be either leaning towards positive, negative or neutral even though individual percentage values may differ)

Sentiment Analysis using TextBlob

TextBlob provides us with an API that allows us to make use of text processing operations such as sentiment analysis, part of speech (pos) tagging, noun phrase extraction, etc. for our data analysis. TextBlob is itself based on the NLTK (Natural Language Toolkit) platform, and provides a very good general overview of the data; Hence we will use it for our analysis.

The approach that TextBlob applies to sentiment analysis differs in that it’s rule-based and therefore requires a pre-defined set of categorised words. Thus, it can be classified as a lexicon-based approach to sentiment analysis. In TextBlob, the range of polarity is between -1 to 1. Negative values near −1 indicate negative sentiment, positive values near 1 indicate positive sentiments, and 0 indicates neutral sentiment.

Sentiment Analysis using VADER (Valence Aware Dictionary for sEntiment Reasoning)

VADER, just like TextBlob is a lexicon and rule-based sentiment analysis method. VADER has been quite successful when dealing with social media texts, product reviews etc. VADER not only tells about the positivity and negativity score but also tells about how positive or negative a sentiment is. VADER analyses a piece of text to see if any of the words from the text are present and returns the metric values of the negative, neutral, positive, and compound for a given sentence.

Generally, VADER is better for predicting sentiments for social network data, but as we need to establish the trends which data shows, we will use both the models and then compare the results. TextBlob gives us a polarity score, which helps us to classify sentiments. Similarly, VADER also gives a compound score, which is used to set classification labels.

We set the TextBlob polarity score as well as the VADER compound score's neutral threshold to be 0.05. Hence, we will use the following rule to classify sentiments:

- If -0.05 <= score <= 0.05: Tweet is of **neutral** sentiment.
- If -0.05 > score: Tweet is of **negative** sentiment.
- If 0.05 < score: Tweet is of **positive** sentiment.

Installing and importing the required libraries (On Colab/Juptyer Notebooks, else use normal pip install).

```
!pip install textblob

!pip install wordcloud

from textblob import TextBlob
from wordcloud import WordCloud
nltk.download('vader_lexicon') # Download the VADER lexicon
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

VADER can be used right from nltk.sentiment.vader built-in with NLTK.

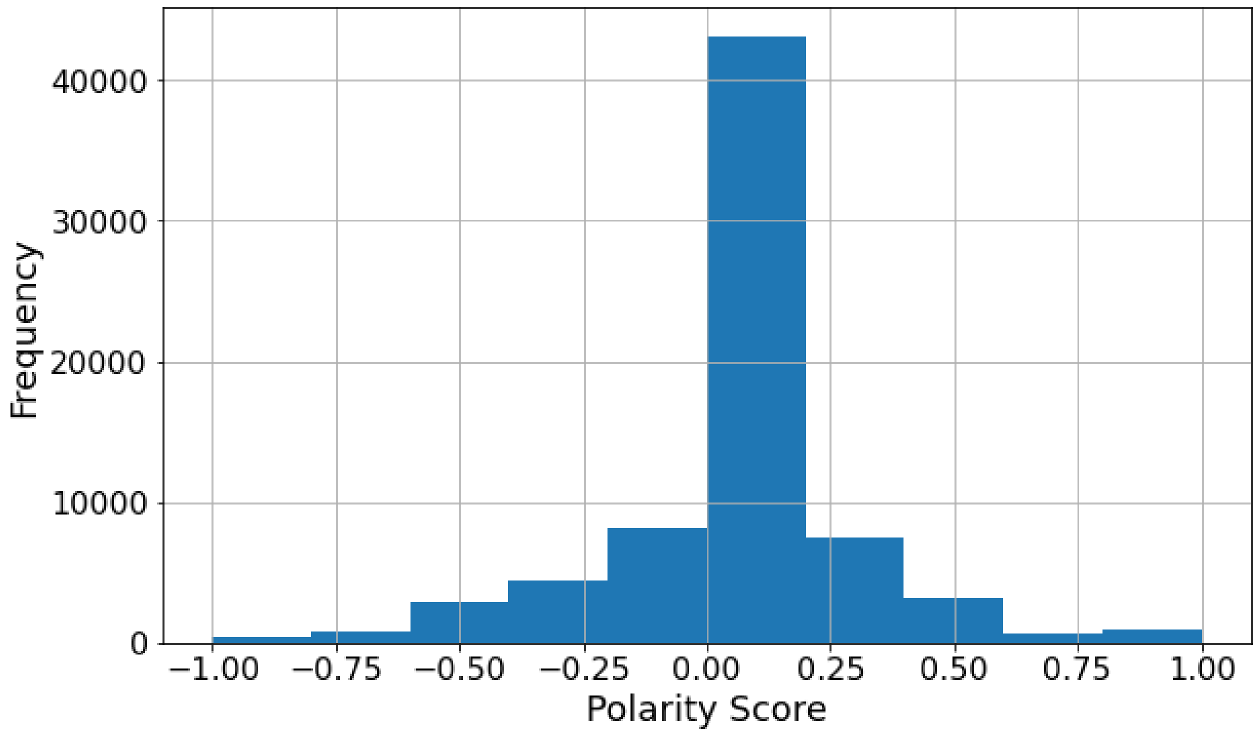
Analysing Sentiments using TextBlob

First, we will use TextBlob to assign **polarity scores** to each of our cleaned tweets and then based on the rules specified above, we will classify sentiments based on one of 3 categories: **Positive**, **Negative** or **Neutral**.

```
df['TBScore'] = df['Clean Text'].apply(lambda x: TextBlob(x).sentiment.polarity)
# Set threshold to define neutral sentiment
neutral_thresh = 0.05

# Convert polarity score into sentiment categories
df['Sentiment'] = df['TBScore'].apply(lambda c: 'Positive' if c >= neutral_thresh
else ('Negative' if c <= -(neutral_thresh) else 'Neutral'))
```

Now, that we have assigned Sentiments to our tweets, we will plot a histogram/frequency plot of polarity scores to better understand the distribution of tweets among sentiment categories.

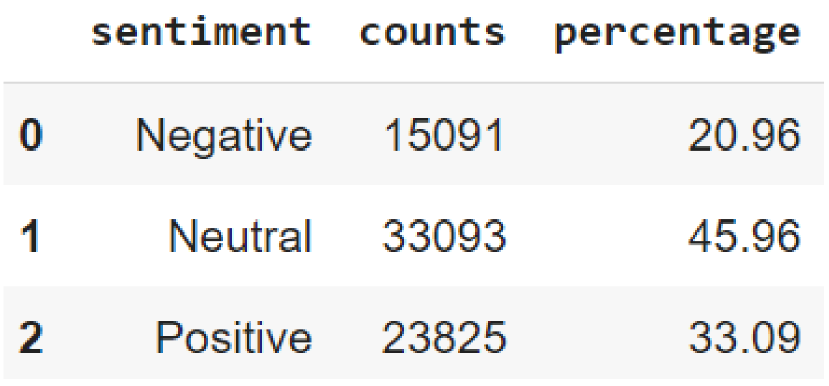
```
fig = plt.figure(figsize=(10, 6))
df['TBScore'].hist()
plt.xlabel('Polarity Score', fontsize=18)
plt.ylabel('Frequency', fontsize=18)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
```



Distribution of tweets: A majority of the tweets can be said to be neutral/slightly positive.

Next, we will look at the percentage of tweets across all sentiment categories.

```
def get_value_counts(col_name):
    count = pd.DataFrame(df[col_name].value_counts())
    percentage = pd.DataFrame(df[col_name].value_counts(normalize=True).mul(100))
    value_counts_df = pd.concat([count, percentage], axis = 1)
    value_counts_df = value_counts_df.reset_index()
    value_counts_df.columns = ['sentiment', 'counts', 'percentage']
    value_counts_df.sort_values('sentiment', inplace = True)
    value_counts_df['percentage'] = value_counts_df['percentage'].apply(lambda x:
round(x,2))
    value_counts_df = value_counts_df.reset_index(drop = True)
    return value_counts_df
tb_sentiment_df = get_value_counts('Sentiment')
tb_sentiment_df
```

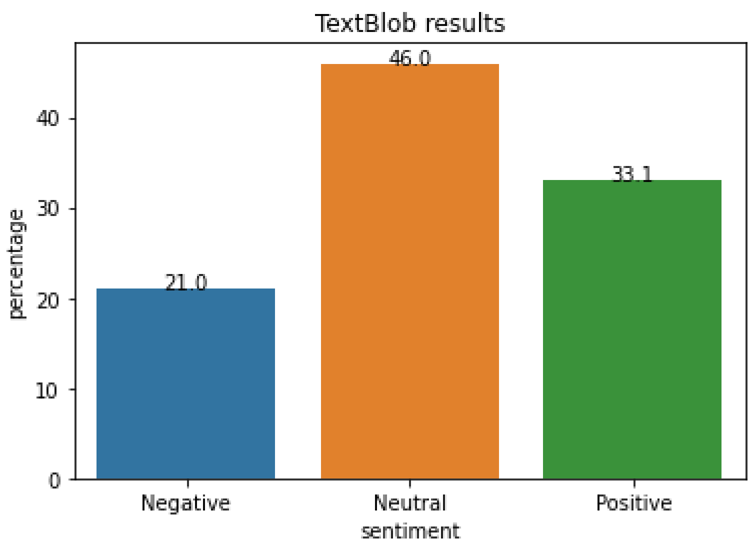


	sentiment	counts	percentage
0	Negative	15091	20.96
1	Neutral	33093	45.96
2	Positive	23825	33.09

Output: A Majority of the tweets have been classified as “neutral” by TextBlob.

We also can plot out a bar (or even a pie) chart of the resultant tb_sentiment_df from above.

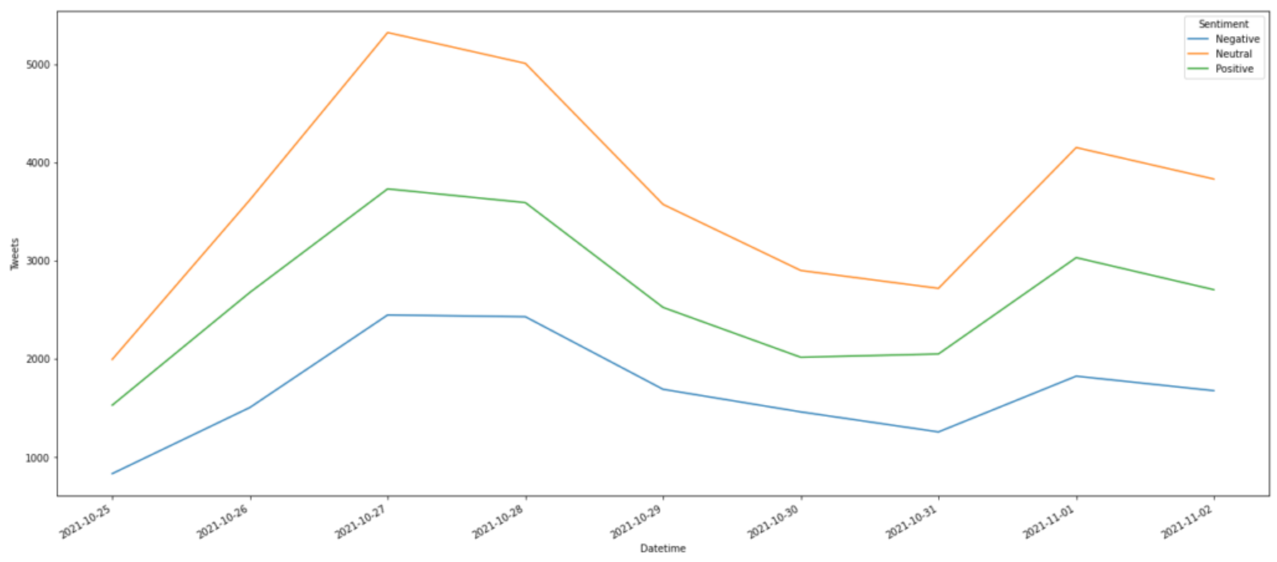
```
ax = sns.barplot(x="sentiment", y="percentage", data=tb_sentiment_df)
ax.set_title('TextBlob results')

for index, row in tb_sentiment_df.iterrows():
    ax.text(row.name,row.percentage, round(row.percentage,1), color='black',
ha="center")
```



Output

Next, we plot a time-series plot using the date data to better understand changing sentiments across all categories over a period of time.

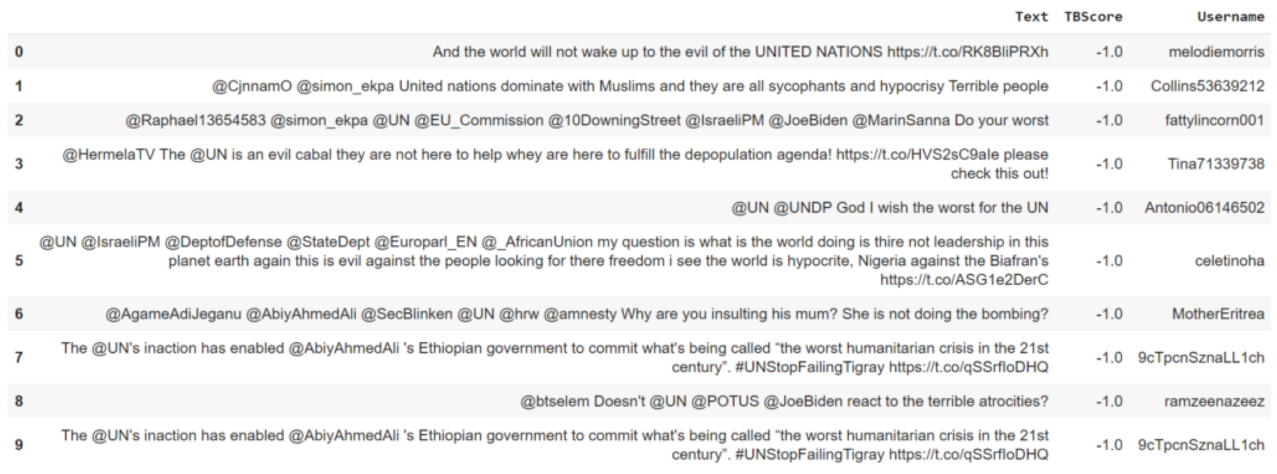
```
# Plotting timeseries plot
df["Datetime"] = pd.to_datetime(df.Datetime)
timeline = df.resample('D', on='Datetime')["Sentiment"].value_counts().unstack(1)
timeline.reset_index(inplace=True)
timeline = timeline.melt("Datetime", var_name='Sentiment',  value_name='Tweets')

plt.figure(figsize=(22,10))
ax = sns.lineplot(x="Datetime", y="Tweets", hue="Sentiment", data=timeline)
ax.figure.autofmt_xdate()
```



Graphing the continuously evolving public sentiments on Twitter for the search term “**united nations**” over a period of 1 week.

To understand the kinds of tweets classified as Negative or Positive, we can plot out the most negative and the most positive set of tweets classified by TextBlob.

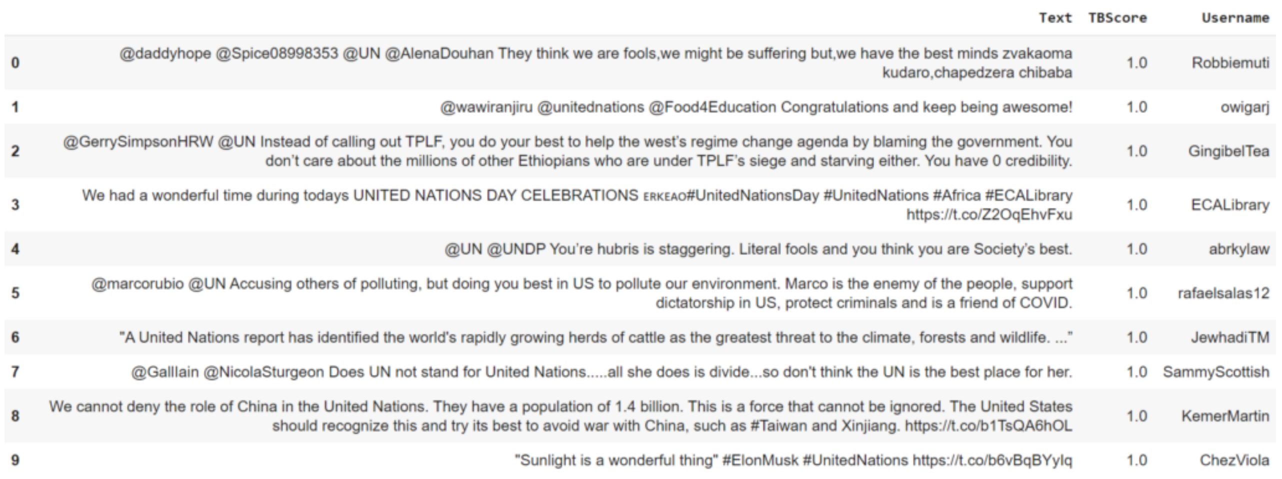
```
pd.set_option('display.max_colwidth', 400)
df.sort_values(by='TBScore', ascending=True)[['Text', 'TBScore',
'Username']].reset_index(drop=True).head(n=10)
```



	Text	TBScore	Username
0	And the world will not wake up to the evil of the UNITED NATIONS https://t.co/RK8BliPRXh	-1.0	melodiemorris
1	@CjnnamO @simon_ekpa United nations dominate with Muslims and they are all sycophants and hypocrisy Terrible people	-1.0	Collins53639212
2	@Raphael13654583 @simon_ekpa @UN @EU_Commission @10DowningStreet @IsraeliPM @JoeBiden @MarinSanna Do your worst	-1.0	fattylincorn001
3	@HermelaTV The @UN is an evil cabal they are not here to help whey are here to fulfill the depopulation agenda! https://t.co/HVS2sC9ale please check this out!	-1.0	Tina71339738
4	@UN @UNDP God I wish the worst for the UN	-1.0	Antonio06146502
5	@UN @IsraeliPM @DeptofDefense @StateDept @Europarl_EN @_AfricanUnion my question is what is the world doing is thire not leadership in this planet earth again this is evil against the people looking for there freedom i see the world is hypocrite, Nigeria against the Biafran's https://t.co/ASG1e2DerC	-1.0	celetinoha
6	@AgameAdiJeganu @AbiyAhmedAli @SecBlinken @UN @hrw @amnesty Why are you insulting his mum? She is not doing the bombing?	-1.0	MotherEritrea
7	The @UN's inaction has enabled @AbiyAhmedAli 's Ethiopian government to commit what's being called “the worst humanitarian crisis in the 21st century”. #UNStopFailingTigray https://t.co/qSSrfloDHQ	-1.0	9cTpcnSznaLL1ch
8	@btselem Doesn't @UN @POTUS @JoeBiden react to the terrible atrocities?	-1.0	ramzeenazeez
9	The @UN's inaction has enabled @AbiyAhmedAli 's Ethiopian government to commit what's being called “the worst humanitarian crisis in the 21st century”. #UNStopFailingTigray https://t.co/qSSrfloDHQ	-1.0	9cTpcnSznaLL1ch

Top 10 most negative tweets as assessed by TextBlob

```
df.sort_values(by='TBScore', ascending=False)[['Text', 'TBScore',
'Username']].reset_index(drop=True).head(n=10)
```



	Text	TBScore	Username
0	@daddyhope @Spice08998353 @UN @AlenaDouhan They think we are fools,we might be suffering but,we have the best minds zvakaoma kudaro,chapedzera chibaba	1.0	Robbiemuti
1	@wawiranjiru @unitednations @Food4Education Congratulations and keep being awesome!	1.0	owigarj
2	@GerrySimpsonHRW @UN Instead of calling out TPLF, you do your best to help the west's regime change agenda by blaming the government. You don't care about the millions of other Ethiopians who are under TPLF's siege and starving either. You have 0 credibility.	1.0	GingibelTea
3	We had a wonderful time during todays UNITED NATIONS DAY CELEBRATIONS ERKEAO#UnitedNationsDay #UnitedNations #Africa #ECALibrary https://t.co/Z2OqEhvFxu	1.0	ECALibrary
4	@UN @UNDP You're hubris is staggering. Literal fools and you think you are Society's best.	1.0	abrkylaw
5	@marcorubio @UN Accusing others of polluting, but doing you best in US to pollute our environment. Marco is the enemy of the people, support dictatorship in US, protect criminals and is a friend of COVID.	1.0	rafaelsalas12
6	"A United Nations report has identified the world's rapidly growing herds of cattle as the greatest threat to the climate, forests and wildlife. ..."	1.0	JewhadiTM
7	@Galllain @NicolaSturgeon Does UN not stand for United Nations.....all she does is divide...so don't think the UN is the best place for her.	1.0	SammyScottish
8	We cannot deny the role of China in the United Nations. They have a population of 1.4 billion. This is a force that cannot be ignored. The United States should recognize this and try its best to avoid war with China, such as #Taiwan and Xinjiang. https://t.co/b1TsQA6hOL	1.0	KemerMartin
9	"Sunlight is a wonderful thing" #ElonMusk #UnitedNations https://t.co/b6vBqBYylq	1.0	ChezViola

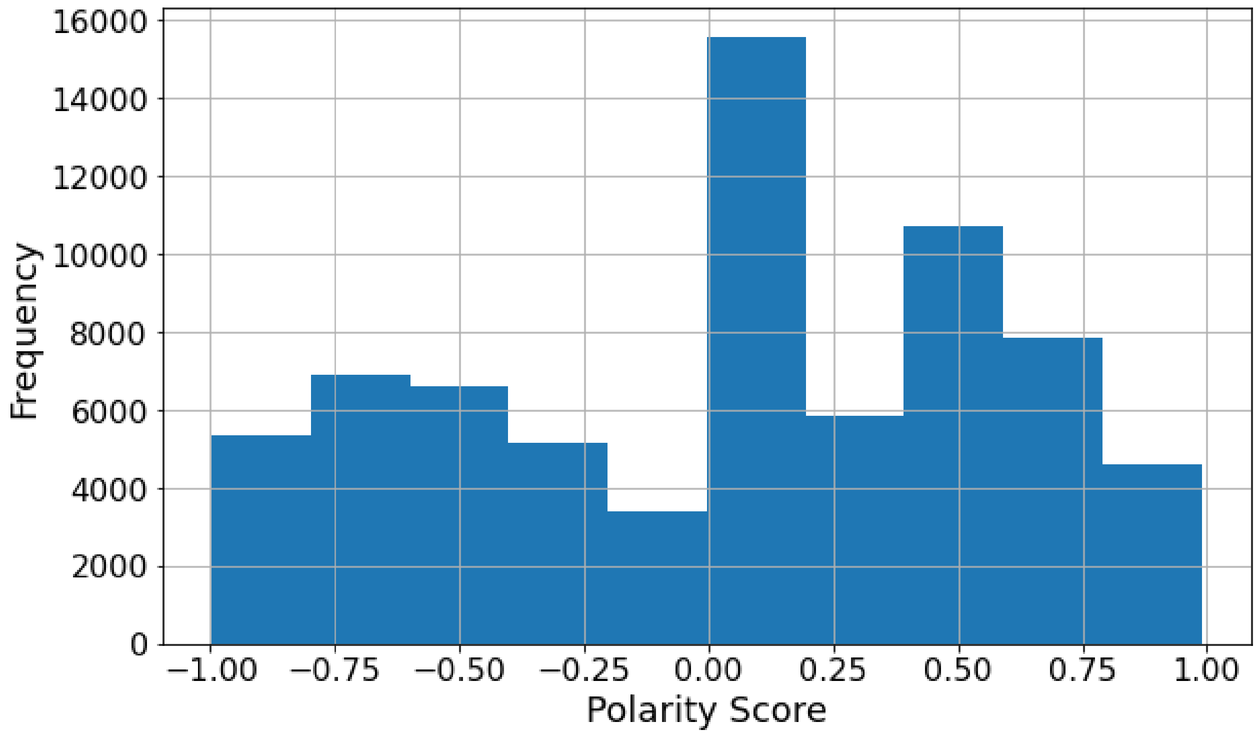
Top 10 most positive tweets as assessed by TextBlob

We can clearly observe that majority of the classified tweets are correct and can come to a conclusion that TextBlob generally does the work of classifying sentiments well so if one needs quick and easy sentiment analysis for a job, they can simply use TextBlob. Next, we will repeat our analysis using VADER to confirm whether our established trends are correct or there is a contradiction between the two models. First, we will set VADER-based polarity and compound scores to our tweets and then classify them into 3 categories: Positive, Negative or Neutral based on the same set of rules defined for TextBlob above.

```
# Performing VADER Sentiment Analysis
import nltk
nltk.download('vader_lexicon') # Download the VADER lexicon
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize sentiment intensity analyzer
sia = SentimentIntensityAnalyzer()

# Obtaining NLTK scores
df['VScore'] = df['Clean Text'].apply(lambda x: sia.polarity_scores(x))

# Obtaining NLTK compound score
df['VComp'] = df['VScore'].apply(lambda score_dict: score_dict['compound'])

# Set threshold to define neutral sentiment
neutral_thresh = 0.05

# Categorize scores into the sentiments of positive, neutral or negative
df['Sentiment'] = df['VComp'].apply(lambda c: 'Positive' if c >= neutral_thresh
else ('Negative' if c <= -(neutral_thresh) else 'Neutral'))
```

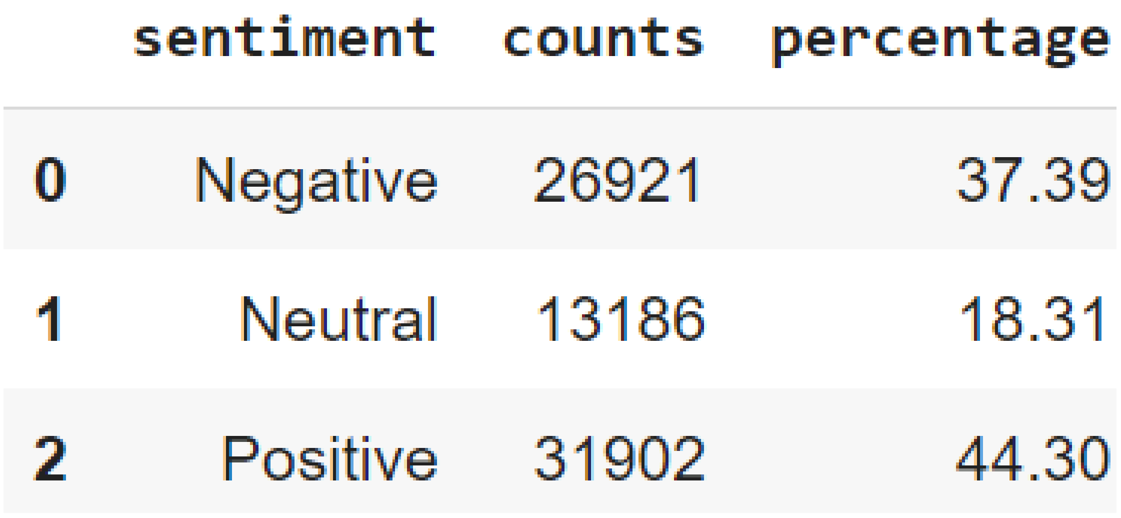
Then, we will repeat the same set of inference graphs as we did for TextBlob and then compare the results.

```
fig = plt.figure(figsize=(10, 6))
df['VComp'].hist()
plt.xlabel('Polarity Score', fontsize=18)
plt.ylabel('Frequency', fontsize=18)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
```



Output: Polarity is much better distributed across varying scores by VADER. However, the majority of tweets are still "neutral/slightly positive" as established by TextBlob.

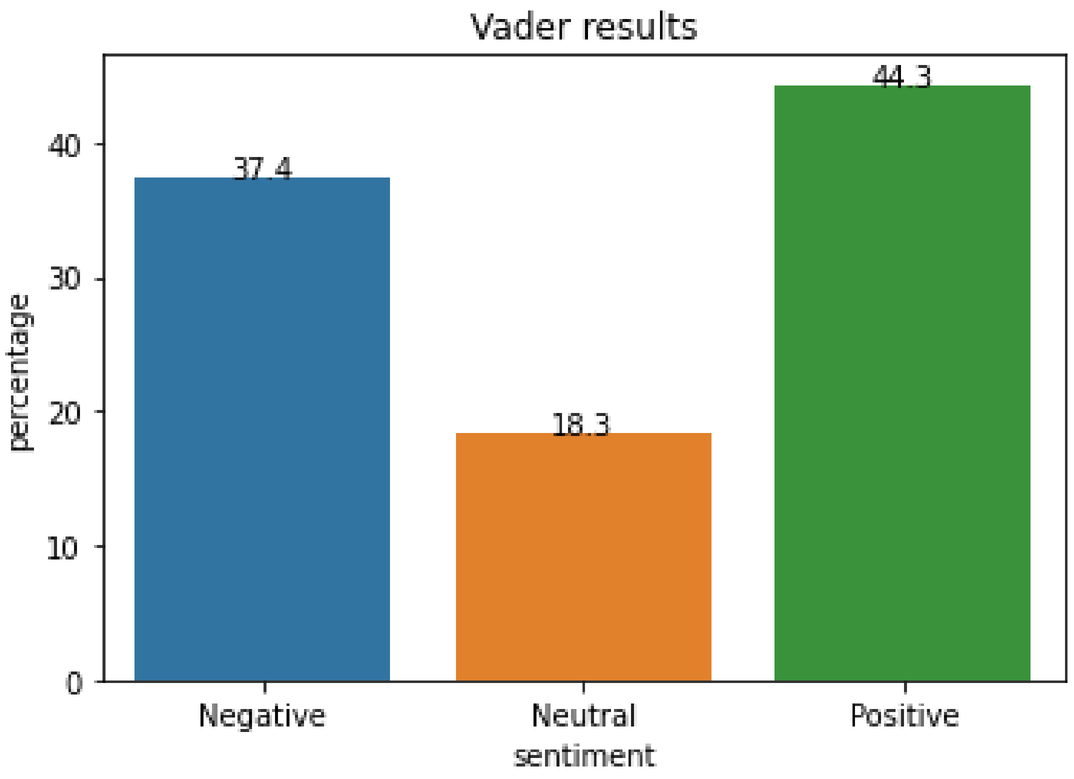
```
def get_value_counts(col_name):
    count = pd.DataFrame(df[col_name].value_counts())
    percentage = pd.DataFrame(df[col_name].value_counts(normalize=True).mul(100))
    value_counts_df = pd.concat([count, percentage], axis = 1)
    value_counts_df = value_counts_df.reset_index()
    value_counts_df.columns = ['sentiment', 'counts', 'percentage']
    value_counts_df.sort_values('sentiment', inplace = True)
    value_counts_df['percentage'] = value_counts_df['percentage'].apply(lambda x:
round(x,2))
    value_counts_df = value_counts_df.reset_index(drop = True)
    return value_counts_df
tb_sentiment_df = get_value_counts('Sentiment')
tb_sentiment_df
```



	sentiment	counts	percentage
0	Negative	26921	37.39
1	Neutral	13186	18.31
2	Positive	31902	44.30

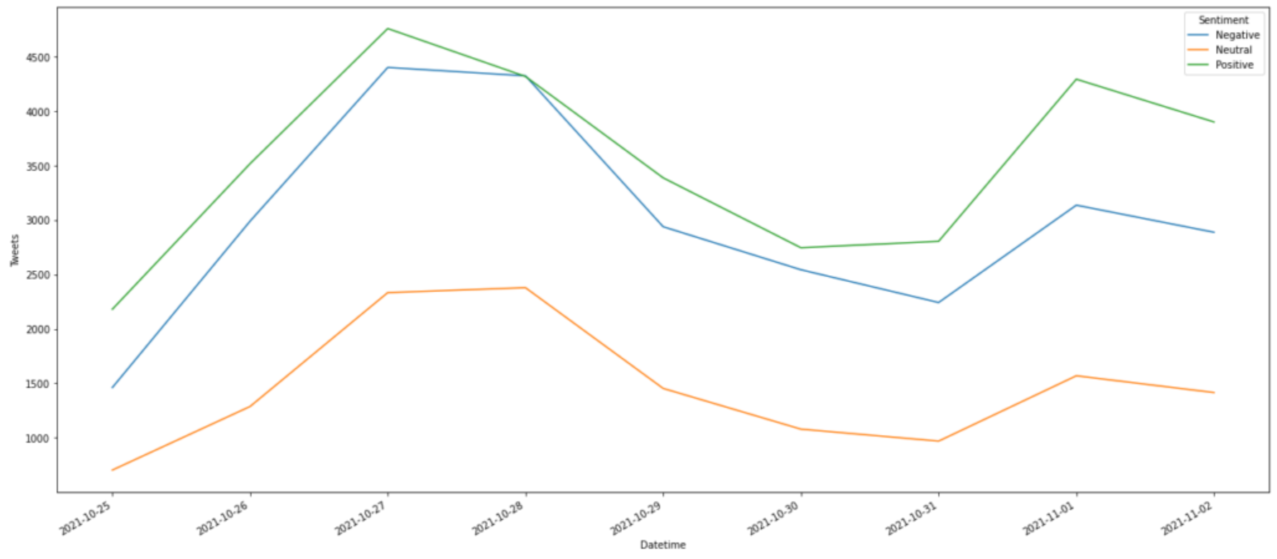
Output: Rather than classifying Tweets as Neutral, we can see that VADER generally classifies them into one of the extreme categories. This is due to the fact that the distribution of VADER classified tweets was better, well-spread across varying polarity scores as compared to TextBlob.

```
ax = sns.barplot(x="sentiment", y="percentage", data=tb_sentiment_df)
ax.set_title('Vader results')

for index, row in tb_sentiment_df.iterrows():
    ax.text(row.name,row.percentage, round(row.percentage,1), color='black',
ha="center")
```



Output

```
df["Datetime"] = pd.to_datetime(df.Datetime)
timeline = df.resample('D', on='Datetime')["Sentiment"].value_counts().unstack(1)
timeline.reset_index(inplace=True)
timeline = timeline.melt("Datetime", var_name='Sentiment',  value_name='Tweets')

plt.figure(figsize=(22,10))
ax = sns.lineplot(x="Datetime", y="Tweets", hue="Sentiment", data=timeline)
ax.figure.autofmt_xdate()
```



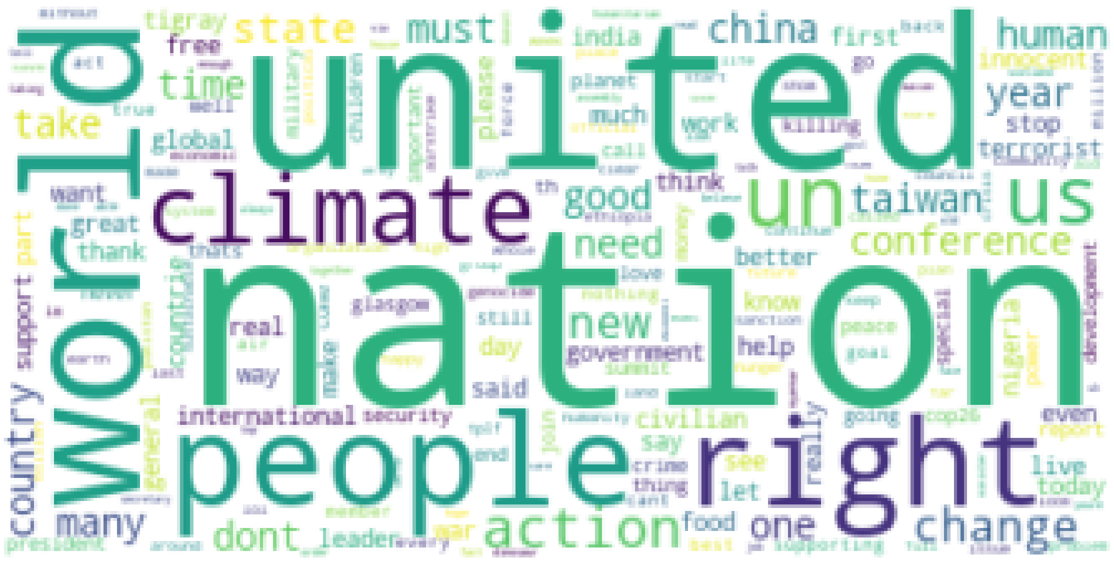
Time-Series plot of varying sentiments classified by VADER across the week for which data was collected.

Forming Word Clouds

Plotting Word clouds is a technique of visualising text data. It conveys important information about the text corpus such as the most recurring words and also their intensity of recurrence. A word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it’s mentioned within a given text and the more important it is pertaining to the text corpus.

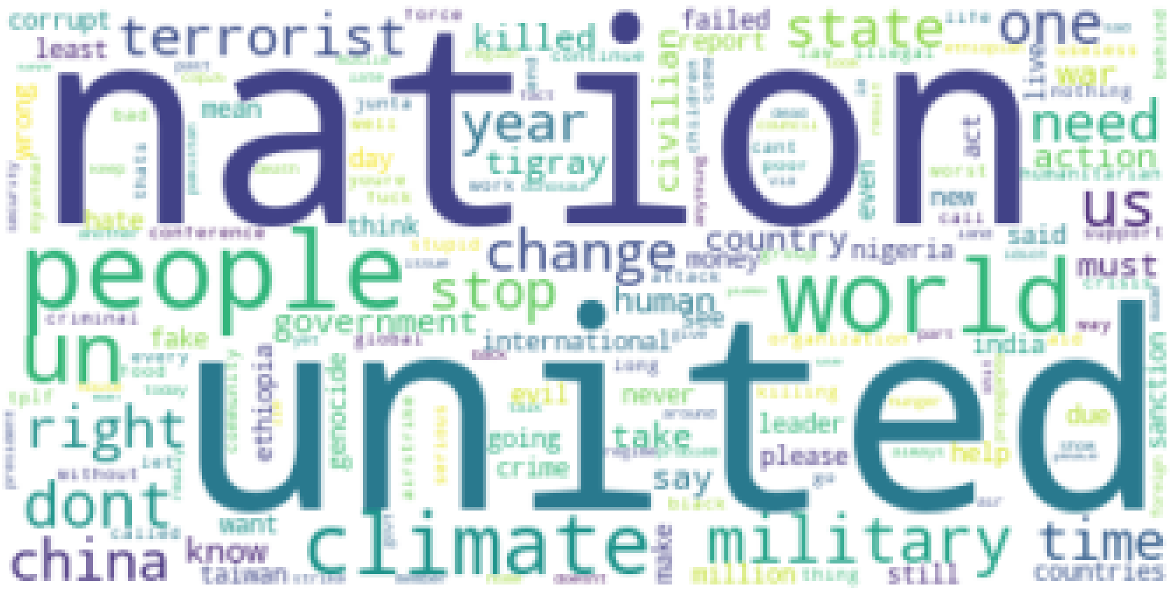
Thus, we will plot out word clouds for positive and negative tweets to better understand what kinds of words people are using in tweets that are positive/negative. We will use results from **TextBlob** and the **wordcloud** module to chart our word clouds.

```
from wordcloud import STOPWORDS
pos_tweets=df[df["Sentiment"]=="Positive"]
txt=" ".join(tweet for tweet in pos_tweets["Clean Text"])
stop_words = ["amp"] + list(STOPWORDS)
wordcloud = WordCloud(collocations = False,
                      background_color = 'white', stopwords =
stop_words).generate(txt)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```



Output – Positive tweets' word cloud

```
neg_tweets=df[df["Sentiment"]=="Negative"]
txt=" ".join(tweet.lower() for tweet in neg_tweets["Clean Text"])
wordcloud = WordCloud(collocations = False,background_color =
'white',stopwords=stop_words).generate(txt)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```



Output – Negative tweets' word cloud

As we can see, the negative word cloud has generally "negative" words (words often used in a negative light during conversations) such as terrorist, failed state, military, china (due to its oppressive regime), don't, killed, stop, attack, etc. highlighted much more than the positive word cloud. Thus, word clouds are beneficial for us to quickly visualise and extract inferences from text data.

Conclusions

By looking at the sentiment polarity scores' distribution and the time-series plots, we can confirm that majority of the tweets regarding the united nations were in a positive light. The trend lines for all 3 sentiments across the week follow similar patterns which suggest that at any given moment in time, the percentage of positive, negative and neutral tweets will largely be the same. Also, we plotted out the word clouds for both the sentiments and found out some keywords used by people in positive as well as in negative tweets.

We can also infer that VADER is more likely to assign a more extreme polarity score rather than TextBlob (which classified majority tweets as negative). Across scientific literature, VADER has been proven time and again to be better than most other lexicon-based classifiers for classifying sentiments for data extracted from social networking websites. An example is this paper which compares VADER, TextBlob and other lexicon-based models for performing sentiment analysis.

That's it for today! If you have read the article till here, you now know:

- How to **scrape** tweets from twitter based on an entered keyword/search term.
- How to use **cloud-based storage service** to work with our data rather than working with local copies.
- How to **preprocess** text (and twitter) data.
- How to perform **Sentiment Analysis** using **lexicon-based** models like **TextBlob** and **VADER**.
- How to visualize trends (via **time-series plots**) and visualize text (via **word clouds**).

Additionally, here are some of the links you may want to refer to:

- **Complete source code** for the article.
- Data used in the article: (Data extracted from Twitter) **Link**
- **This** link from Cambridge University press provides an excellent introduction to Sentiment Analysis and its applications.
- Introduction to tweet-preprocessor: **Link**
- Paper which introduced VADER: **Link**
- Comparison between VADER and TextBlob: **Link**