**IBM HACKCHALLENGE 2020**

**SENTIMENTAL ANALYSIS OF COVID-19 TWEETS – VISUALISATION DASHBOARD**

**Challenge Title** : IBM Hack Challenge 2020

**Project ID** : SPS\_PRO\_302

**Project Title**  : Sentimental Analysis of Covid-19 Tweets

**Duration**  : 32 Days

**Application ID** : SPS\_CH\_APL\_20200001593

**Team Name : Error\_404**

**Team Members :** Mayank Mohan

Pranjal Mehrotra

Adarsh Bisht

Gaurav Joshi

CONTENT

**1. Introduction  
 1.1 Overview**

**1.2 Purpose**

**2. Literature Survey  
 2.1 Existing Problem**

**2.2 Proposed Solution**

**3. Theoretical Analysis  
 3.1 Block Diagram**

**3.2 Software Designing**

**4. Functions  
5. Experimental Analysis  
6. Flow Chart  
7. Results  
8. Output of Sentiment Analysis & Dashboard**

**9. Advantages and Disadvantages**

**10. Bibliography**

**11. Source code**    

**1. INTRODUCTION:**

The main of the project is to do the sentiment analysis on the tweets about the COVID-19 in the Twitter.The API is used to connect the communication between the Node-red application and the Twitter.The Twitter Node that is found in the Node-red application is used as the source to give the tweets about the COVID-19 .The Important node that is used to give the sentiment score is the sentiment node that is found in the Node-red application.The tweets are given as the input to the Sentiment node the sentiment node separates the tweets as POSITIVE,NEUTRAL AND NEGATIVE.If the value is positive then there was positive thing going on the twitter, if the sentiment score is negative then there was negative action going on the twitter .If the value is neutral then there was a neutral action going in the twitter.It will be very helpful to known about the public opinion.

**1.1 OVERVIEW:**

The overview of the IBM HACK CHALLENGE 2020 Problem for COVID-19 sentiment analysis is to do the sentiment analysis for the tweets that were tweeted by the public and the social media about the corona virus .

**1.2 PURPOSE:**

* The usefulness is that it creates the awareness of the COVID-19 among the public.
* The website contains the sentiment of the covid-19 based on the tweets in the twitter by the people.
* The sentiment gives the information among the people about the seriousness of the COVID-19 among the public.
* The website also gives the sentiment data about the COVID-19 .
* The website also gives the tips and Ayurvedic medicine tips to protect against the COVID-19.
* The website also gives the information about the wearing of the mask and the hand-washing.
* This also helps the government to extract the data about the COVID-19 among the public.

**2. LITERATURE SURVEY:**

**2.1 EXISTING PROBLEM:**

The Existing problem is that the news some times obtained is not understood by the people whether it is a Negative or it is the positive.

**2.2 PROPOSED SOLUTION:**

This is will help to divide the news into three sentiments namely:

1.POSITIVE(sentiment value>=1)  
2. NEGATIVE(sentiment value<0)

3. NEUTRAL(sentiment value=0)

Example:

Let us take the existing situation about the COVID-19  
People do not known the current situation is positive or negative about the corona .

The idea is made in the form of the website where it gives the exact current situation about the covid-19.  
if(sentiment\_value==0):

then output is NEUTRAL

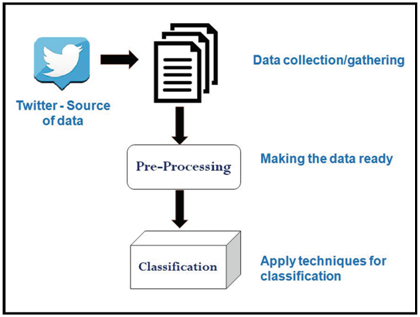
else if(sentiment\_value>0):

then output POSITIVE

else:

the output NEGATIVE

**3. THEORITICAL ANALYSIS:**

**3.1 BLOCK DIAGRAM:**

**3.2 SOFTWARE DESIGNING:**

The Software is designed with the help of services provided by the IBM CLOUD SERVICES.

**IBM Cloud** provides a full-stack, public **cloud** platform with a variety of offerings in the catalog, including compute, **storage**, and networking options, end-to-end developer solutions for app development, testing and deployment, security management **services**, traditional and open-source databases, and **cloud**-native .

* **Project Requirements**: Python, IBM Cloud, IBM Watson,
* **Functional Requirements**: IBM cloud
* **Technical Requirements**: Machine Learning, Deep Learning, NLP, WATSON Services, Python, Watson Dashboard, Twitter API
* **Software Requirements**: Python,Watson Studio

**4. FUNCTIONS:**

The Function of the COVID-19 Sentiment is that it spread the awareness among the public and the statistical data can derived using the COVID-19 Sentiment Analysis in the Simple Way.

It Consists of the Website + Application. I Have Built the Application Using the MIT APP

**5. EXPERIMENTAL ANALYSIS:**

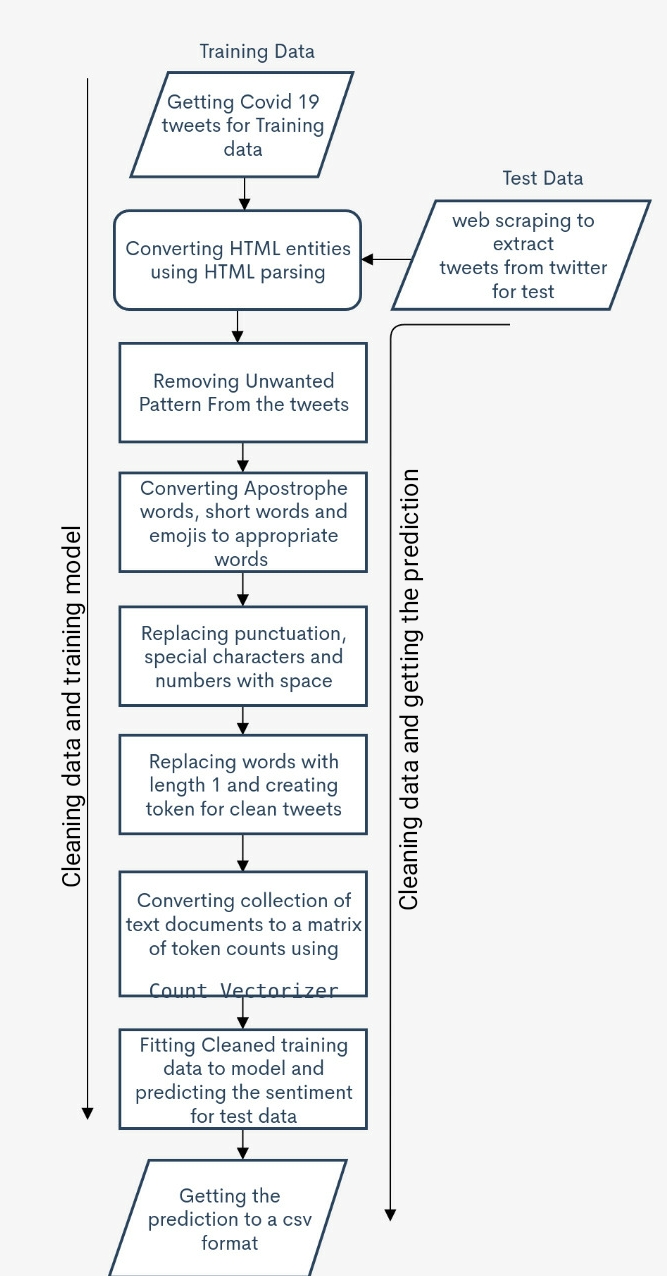
I have done Various Experiment Analysis on this Covid-19 Sentiment Analysis. OBSERVED that is gives:

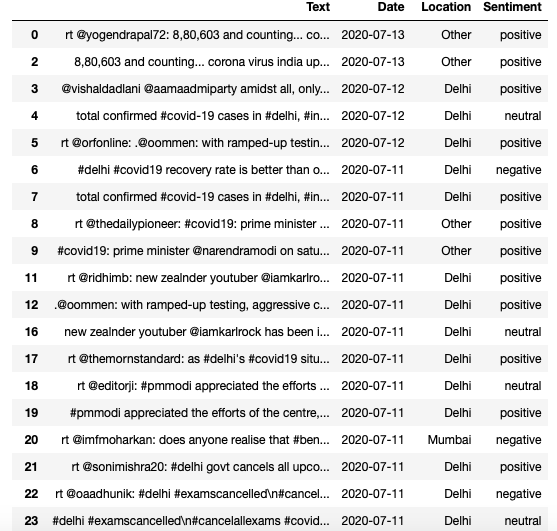
POSITIVE: >1

NEGATIVE: <1

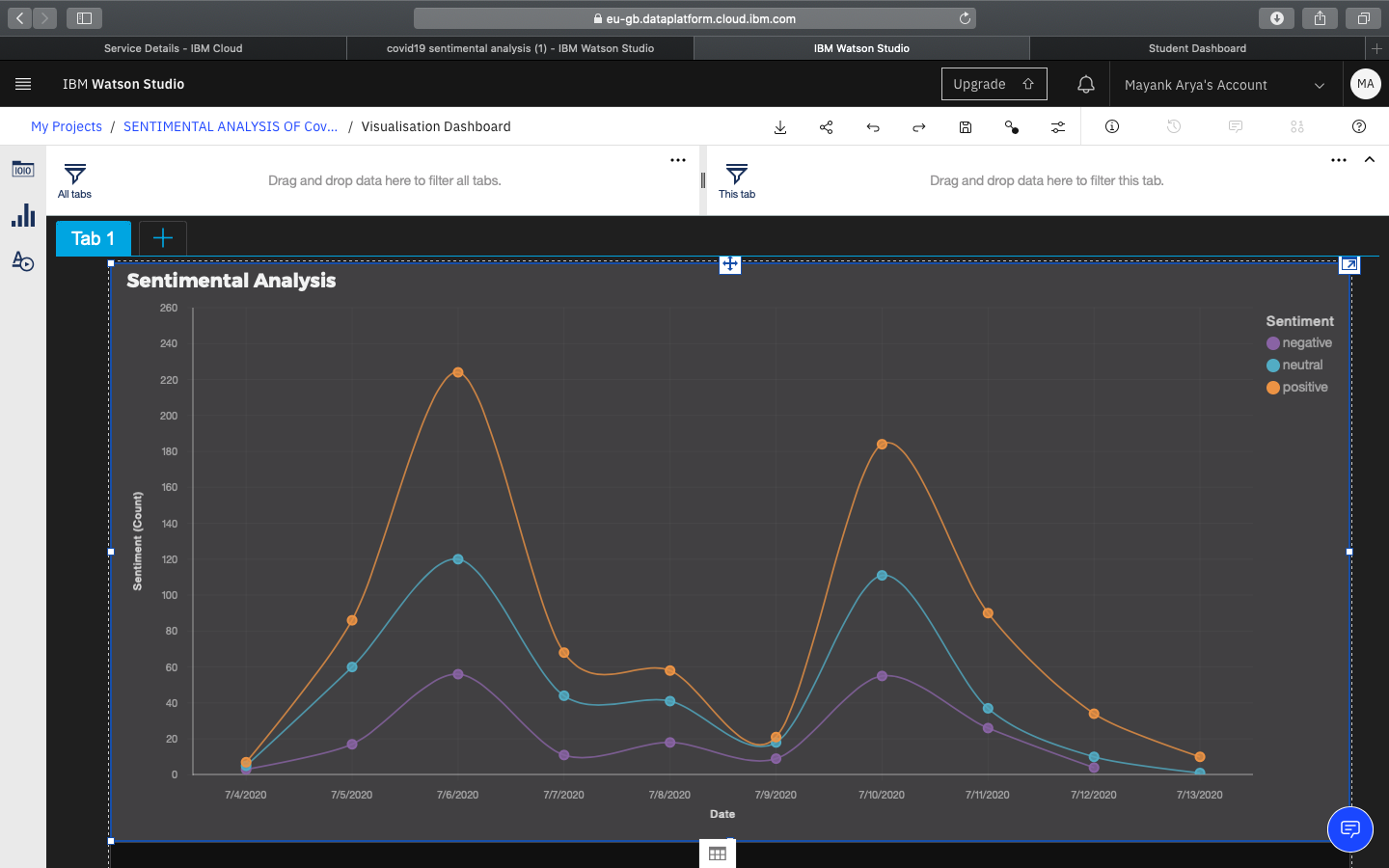
NEUTRAL: =0

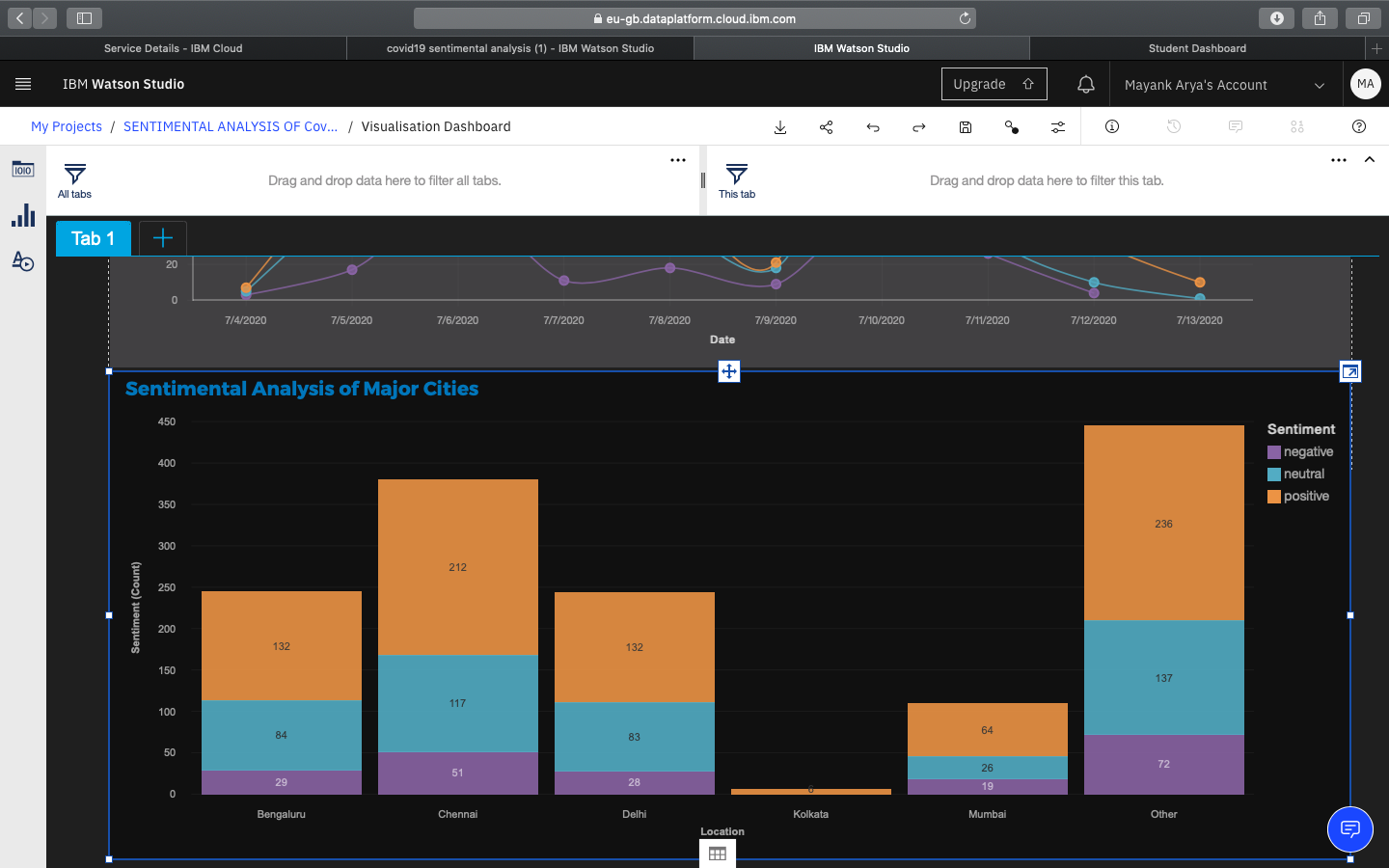
**6. FLOW CHART:**

**7. RESULTS:**

****

**8. OUTPUT OF SENTIMENTAL ANALYSIS & DASHBOARD:**

****



**9. ADVANTAGES :**

* Helps to make awareness among the public Gives the sentiment values
* Simple to handle
* Simple to get any statistical data
* Easy for Government to analysis the people view over their policies.

**DISADVANTAGES:**

Some times the dashboard lost s its connection due to the server problem.

**10. BIBLOGRAPHY:**

* SmartBridge Bootcamps
* Google
* Twitter
* Mentor support

**11. Source Code:**

# # Importing Required Libraries

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** warnings

**import** re

**import** types

**import** pandas **as** pd

**from** botocore.client **import** Config

**import** ibm\_boto3

**def** \_\_iter\_\_(self): **return** 0

# @hidden\_cell

# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.

# You might want to remove those credentials before you share the notebook.

client\_ebbb4a33aa644527a29c121fe811d861 = ibm\_boto3.client(service\_name='s3',

ibm\_api\_key\_id='CATeNJuWTcb5iuplHUzr71UNMP3TOofUtoYLNXei6xUh',

ibm\_auth\_endpoint="https://iam.cloud.ibm.com/oidc/token",

config=Config(signature\_version='oauth'),

endpoint\_url='https://s3.eu-geo.objectstorage.service.networklayer.com')

body = client\_ebbb4a33aa644527a29c121fe811d861.get\_object(Bucket='sentimentalanalysisofcovid19tweet-donotdelete-pr-ye1lk5o4jj2gzb',Key='Data.csv')['Body']

# add missing \_\_iter\_\_ method, so pandas accepts body as file-like object

**if** **not** hasattr(body, "\_\_iter\_\_"): body.\_\_iter\_\_ = types.MethodType( \_\_iter\_\_, body )

data= pd.read\_csv(body)

data.head()

data=data.drop(columns='retweet\_count')

data.shape

# #### checking for all null values

data.isnull().sum()

# #### Dropping null value

data=data.dropna()

data.isnull().sum()

# #### Changing all the tweets into lowercase¶

#

data['clean\_tweet'] = data['full\_text'].apply(**lambda** x: x.lower())

data.head(5)

# #### Removing words whom length is 1 And Replacing Numbers (integers) with space¶

data['clean\_tweet'] = data['clean\_tweet'].apply(**lambda** x: ' '.join([w **for** w **in** x.split() **if** len(w)>1]))

data['clean\_tweet'] = data['clean\_tweet'].apply(**lambda** x: re.sub(r'[^a-zA-Z]',' ',x))

data.head()

**import** nltk

nltk.download("punkt")

# #### Creating token for the clean tweets

data['tweet\_token'] = data['clean\_tweet'].apply(**lambda** x:nltk.word\_tokenize(x))

data.head()

**from** nltk.corpus **import** stopwords

nltk.download('stopwords')

# #### Importing stop words from NLTK corpus for english language

stop\_words = set(stopwords.words('english'))

data['tweet\_token\_filtered'] = data['tweet\_token'].apply(**lambda** x: [word **for** word **in** x **if** **not** word **in** stop\_words])

data.head()

nltk.download('wordnet')

# #### Lemmatization - Lemmatization is the process of converting a word to its base form.¶

#

# Importing library for lemmatizing

**from** nltk.stem.wordnet **import** WordNetLemmatizer

lemmatizing = WordNetLemmatizer()

# Created one more columns tweet\_lemmatized it shows tweets' lemmatized version

data['tweet\_lemmatized'] = data['tweet\_token\_filtered'].apply(**lambda** x: ' '.join([lemmatizing.lemmatize(i) **for** i **in** x]))

data['tweet\_lemmatized'].head(10)

get\_ipython().system('pip install wordcloud')

# #### Will see the most commonly used words in the column i.e. " "tweet\_lematized"

#visualizing all the words in column "tweet\_lemmatized" in our data using the wordcloud plot.

all\_words = ' '.join([text **for** text **in** data['tweet\_lemmatized']])

**from** wordcloud **import** WordCloud

wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(all\_words)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis('off')

plt.title("Most Common words in column Tweet Lemmatized")

plt.show()

# #### Most common words in Positive tweets

positive\_words =' '.join([text **for** text **in** data['tweet\_lemmatized'][data['Sentiment'] == "positive"]])

wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(positive\_words)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis('off')

plt.title("positive Tweet Lemmatized")

plt.show()

# #### Most common words in neutral tweets

neutral\_words =' '.join([text **for** text **in** data['tweet\_lemmatized'][data['Sentiment'] == "neutral"]])

wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(neutral\_words)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis('off')

plt.title("neutral Tweet Lemmatized")

plt.show()

# #### Most common words in negative tweets

negative\_words =' '.join([text **for** text **in** data['tweet\_lemmatized'][data['Sentiment'] == "negative"]])

wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(negative\_words)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis('off')

plt.title("negative Tweet Lemmatized")

plt.show()

# #### Bag-of-Words Features

## Importing library

**from** sklearn.feature\_extraction.text **import** CountVectorizer

bow\_vectorizer = CountVectorizer(max\_df=0.90, min\_df=2, max\_features=1000, stop\_words='english')

# Bag-Of-Words feature matrix - For columns "combine\_df['tweet\_lemmatized']"

bow\_lemm = bow\_vectorizer.fit\_transform(data['tweet\_lemmatized']).toarray()

bow\_lemm

# #### mapping the Sentiment column (Dependent column) with 1, 0 and -1

x=bow\_lemm

y=data['Sentiment']

y.replace(['neutral','positive','negative'],[0,1,-1],inplace=**True**)

y.head()

# #### Splitting data into train and test

**from** sklearn.model\_selection **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 42)

# #### Scaling the training and testing data

**from** sklearn.preprocessing **import** StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.transform(x\_test)

# #### Using Decision Tree Classifier for training and testing

**from** sklearn.tree **import** DecisionTreeClassifier

model = DecisionTreeClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("test Accuracy :", model.score(x\_test, y\_test))

y\_pred=pd.Series(y\_pred)

y\_pred.replace([0,1,-1],['neutral','positive','negative'],inplace=**True**)

y\_pred.head()

# ### Scrapping twitter data with the help of Tweepy Library and saving it in a csv file

get\_ipython().system('pip install tweepy')

**import** tweepy

**import** pandas **as** pd

consumer\_key = "rm2bLDjA2BzljoA0GomL5o6W7"

consumer\_secret = "xiFBG4VKWPuQts1v3uqAesllpDp36y44YkFnzBtezSbSYW9dBV"

access\_token = "935519854064418816-sOBxmFMaDygAx3FQXRBjH0drpZ2OXpB"

access\_token\_secret = "GbOTefzapdet9vpmR3H9OBRuJNJNs1cI4Adh5HrkIYPJz"

# Creating the authentication object

auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

# Setting your access token and secret

auth.set\_access\_token(access\_token, access\_token\_secret)

# Creating the API object while passing in auth information

api = tweepy.API(auth)

number\_of\_items=500

searchword="#COVID and #DELHI "

tweets=tweepy.Cursor(api.search,q=searchword,lang='en').items(number\_of\_items)

names\_text=[[tweet.user.screen\_name,tweet.text.lower(),tweet.created\_at,tweet.user.location.lower()]**for** tweet **in** tweets]

df=pd.DataFrame(data=names\_text,columns=['Name','Text',"Date","Location"])

searchword="#COVID and #Chennai"

tweets=tweepy.Cursor(api.search,q=searchword,lang='en').items(number\_of\_items)

names\_text=[[tweet.user.screen\_name,tweet.text.lower(),tweet.created\_at,tweet.user.location.lower()]**for** tweet **in** tweets]

df1=pd.DataFrame(data=names\_text,columns=['Name','Text',"Date","Location"])

searchword="#COVID and #Mumbai"

tweets=tweepy.Cursor(api.search,q=searchword,lang='en').items(number\_of\_items)

names\_text=[[tweet.user.screen\_name,tweet.text.lower(),tweet.created\_at,tweet.user.location.lower()]**for** tweet **in** tweets]

df2=pd.DataFrame(data=names\_text,columns=['Name','Text',"Date","Location"])

searchword="#COVID and #Bengaluru"

tweets=tweepy.Cursor(api.search,q=searchword,lang='en').items(number\_of\_items)

names\_text=[[tweet.user.screen\_name,tweet.text.lower(),tweet.created\_at,tweet.user.location.lower()]**for** tweet **in** tweets]

df3=pd.DataFrame(data=names\_text,columns=['Name','Text',"Date","Location"])

searchword="#COVID and #Kolkata"

tweets=tweepy.Cursor(api.search,q=searchword,lang='en').items(number\_of\_items)

names\_text=[[tweet.user.screen\_name,tweet.text.lower(),tweet.created\_at,tweet.user.location.lower()]**for** tweet **in** tweets]

df4=pd.DataFrame(data=names\_text,columns=['Name','Text',"Date","Location"])

frames=[df,df1,df2,df3,df4]

result = pd.concat(frames)

print(result)

**for** i **in** range(len(result)) :

**if** "delhi" **in** result.iloc[i, 3] **or** "delhi" **in** result.iloc[i, 1]:

result.iloc[i, 3]="Delhi"

**elif** "mumbai" **in** result.iloc[i, 3] **or** "mumbai" **in** result.iloc[i, 1]:

result.iloc[i, 3]="Mumbai"

**elif** "chennai" **in** result.iloc[i, 3] **or** "chennai" **in** result.iloc[i, 1]:

result.iloc[i, 3]="Chennai"

**elif** "bengaluru" **in** result.iloc[i, 3] **or** "bengaluru" **in** result.iloc[i, 1]:

result.iloc[i, 3]="Bengaluru"

**elif** "kolkata" **in** result.iloc[i, 3] **or** "kolkata" **in** result.iloc[i, 1]:

result.iloc[i, 3] = "Kolkata"

**else**:

result.iloc[i, 3] = "Other"

result.to\_csv('tweets.csv', index=**True**)

df=result

# ### Viewing the scapped tweets csv file

df["Location"].value\_counts()

df.head()

df.shape

# ### Converting HTML entities and saving clean tweets to new clean tweet column

**import** html

**from** html.parser **import** HTMLParser

html\_parser = HTMLParser()

df['clean\_tweet'] = df['Text'].apply(**lambda** x: html.unescape(x))

# ### Removing twitter handles from all tweets in clean tweet column

**def** remove\_pattern(input\_txt, pattern):

r = re.findall(pattern, input\_txt)

**for** i **in** r:

input\_txt = re.sub(i, '', input\_txt)

**return** input\_txt

df['clean\_tweet'] = np.vectorize(remove\_pattern)(df['clean\_tweet'], "@[\w]\*")

# ### converting tweets to lowercase

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: x.lower())

# ### Apostrophe lookup in tweets

apostrophe\_dict = {

"ain't": "am not / are not",

"aren't": "are not / am not",

"can't": "cannot",

"can't've": "cannot have",

"'cause": "because",

"could've": "could have",

"couldn't": "could not",

"couldn't've": "could not have",

"didn't": "did not",

"doesn't": "does not",

"don't": "do not",

"hadn't": "had not",

"hadn't've": "had not have",

"hasn't": "has not",

"haven't": "have not",

"he'd": "he had / he would",

"he'd've": "he would have",

"he'll": "he shall / he will",

"he'll've": "he shall have / he will have",

"he's": "he has / he is",

"how'd": "how did",

"how'd'y": "how do you",

"how'll": "how will",

"how's": "how has / how is",

"i'd": "I had / I would",

"i'd've": "I would have",

"i'll": "I shall / I will",

"i'll've": "I shall have / I will have",

"i'm": "I am",

"i've": "I have",

"isn't": "is not",

"it'd": "it had / it would",

"it'd've": "it would have",

"it'll": "it shall / it will",

"it'll've": "it shall have / it will have",

"it's": "it has / it is",

"let's": "let us",

"ma'am": "madam",

"mayn't": "may not",

"might've": "might have",

"mightn't": "might not",

"mightn't've": "might not have",

"must've": "must have",

"mustn't": "must not",

"mustn't've": "must not have",

"needn't": "need not",

"needn't've": "need not have",

"o'clock": "of the clock",

"oughtn't": "ought not",

"oughtn't've": "ought not have",

"shan't": "shall not",

"sha'n't": "shall not",

"shan't've": "shall not have",

"she'd": "she had / she would",

"she'd've": "she would have",

"she'll": "she shall / she will",

"she'll've": "she shall have / she will have",

"she's": "she has / she is",

"should've": "should have",

"shouldn't": "should not",

"shouldn't've": "should not have",

"so've": "so have",

"so's": "so as / so is",

"that'd": "that would / that had",

"that'd've": "that would have",

"that's": "that has / that is",

"there'd": "there had / there would",

"there'd've": "there would have",

"there's": "there has / there is",

"they'd": "they had / they would",

"they'd've": "they would have",

"they'll": "they shall / they will",

"they'll've": "they shall have / they will have",

"they're": "they are",

"they've": "they have",

"to've": "to have",

"wasn't": "was not",

"we'd": "we had / we would",

"we'd've": "we would have",

"we'll": "we will",

"we'll've": "we will have",

"we're": "we are",

"we've": "we have",

"weren't": "were not",

"what'll": "what shall / what will",

"what'll've": "what shall have / what will have",

"what're": "what are",

"what's": "what has / what is",

"what've": "what have",

"when's": "when has / when is",

"when've": "when have",

"where'd": "where did",

"where's": "where has / where is",

"where've": "where have",

"who'll": "who shall / who will",

"who'll've": "who shall have / who will have",

"who's": "who has / who is",

"who've": "who have",

"why's": "why has / why is",

"why've": "why have",

"will've": "will have",

"won't": "will not",

"won't've": "will not have",

"would've": "would have",

"wouldn't": "would not",

"wouldn't've": "would not have",

"y'all": "you all",

"y'all'd": "you all would",

"y'all'd've": "you all would have",

"y'all're": "you all are",

"y'all've": "you all have",

"you'd": "you had / you would",

"you'd've": "you would have",

"you'll": "you shall / you will",

"you'll've": "you shall have / you will have",

"you're": "you are",

"you've": "you have"

}

**def** lookup\_dict(text, dictionary):

**for** word **in** text.split():

**if** word.lower() **in** dictionary:

**if** word.lower() **in** text.split():

text = text.replace(word, dictionary[word.lower()])

**return** text

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: lookup\_dict(x,apostrophe\_dict))

# ### Short words lookup

short\_word\_dict = {

"121": "one to one",

"a/s/l": "age, sex, location",

"adn": "any day now",

"afaik": "as far as I know",

"afk": "away from keyboard",

"aight": "alright",

"alol": "actually laughing out loud",

"b4": "before",

"b4n": "bye for now",

"bak": "back at the keyboard",

"bf": "boyfriend",

"bff": "best friends forever",

"bfn": "bye for now",

"bg": "big grin",

"bta": "but then again",

"btw": "by the way",

"cid": "crying in disgrace",

"cnp": "continued in my next post",

"cp": "chat post",

"cu": "see you",

"cul": "see you later",

"cul8r": "see you later",

"cya": "bye",

"cyo": "see you online",

"dbau": "doing business as usual",

"fud": "fear, uncertainty, and doubt",

"fwiw": "for what it's worth",

"fyi": "for your information",

"g": "grin",

"g2g": "got to go",

"ga": "go ahead",

"gal": "get a life",

"gf": "girlfriend",

"gfn": "gone for now",

"gmbo": "giggling my butt off",

"gmta": "great minds think alike",

"h8": "hate",

"hagn": "have a good night",

"hdop": "help delete online predators",

"hhis": "hanging head in shame",

"iac": "in any case",

"ianal": "I am not a lawyer",

"ic": "I see",

"idk": "I don't know",

"imao": "in my arrogant opinion",

"imnsho": "in my not so humble opinion",

"imo": "in my opinion",

"iow": "in other words",

"ipn": "I’m posting naked",

"irl": "in real life",

"jk": "just kidding",

"l8r": "later",

"ld": "later, dude",

"ldr": "long distance relationship",

"llta": "lots and lots of thunderous applause",

"lmao": "laugh my ass off",

"lmirl": "let's meet in real life",

"lol": "laugh out loud",

"ltr": "longterm relationship",

"lulab": "love you like a brother",

"lulas": "love you like a sister",

"luv": "love",

"m/f": "male or female",

"m8": "mate",

"milf": "mother I would like to fuck",

"oll": "online love",

"omg": "oh my god",

"otoh": "on the other hand",

"pir": "parent in room",

"ppl": "people",

"r": "are",

"rofl": "roll on the floor laughing",

"rpg": "role playing games",

"ru": "are you",

"shid": "slaps head in disgust",

"somy": "sick of me yet",

"sot": "short of time",

"thanx": "thanks",

"thx": "thanks",

"ttyl": "talk to you later",

"u": "you",

"ur": "you are",

"uw": "you’re welcome",

"wb": "welcome back",

"wfm": "works for me",

"wibni": "wouldn't it be nice if",

"wtf": "what the fuck",

"wtg": "way to go",

"wtgp": "want to go private",

"ym": "young man",

"gr8": "great"

}

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: lookup\_dict(x,short\_word\_dict))

# ### Emoticon Lookup

emoticon\_dict = {

":)": "happy",

":‑)": "happy",

":-]": "happy",

":-3": "happy",

":->": "happy",

"8-)": "happy",

":-}": "happy",

":o)": "happy",

":c)": "happy",

":^)": "happy",

"=]": "happy",

"=)": "happy",

"<3": "happy",

":-(": "sad",

":(": "sad",

":c": "sad",

":<": "sad",

":[": "sad",

">:[": "sad",

":{": "sad",

">:(": "sad",

":-c": "sad",

":-< ": "sad",

":-[": "sad",

":-||": "sad"

}

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: lookup\_dict(x,emoticon\_dict))

# ### Replacing punctuations with spaces

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: re.sub(r'[^\w\s]',' ',x))

# ### Replacing special characters with spaces

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: re.sub(r'[^a-zA-Z0-9]',' ',x))

# ### Replacing numbers with spaces

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: re.sub(r'[^a-zA-Z]',' ',x))

# ### Removing words with length 1

df['clean\_tweet'] = df['clean\_tweet'].apply(**lambda** x: ' '.join([w **for** w **in** x.split() **if** len(w)>1]))

# ### Tokenizing the clean\_tweet column and removing stop words from new tweet\_token column

**from** nltk **import** word\_tokenize

df['tweet\_token'] = df['clean\_tweet'].apply(**lambda** x: word\_tokenize(x))

df['tweet\_token\_filtered'] = df['tweet\_token'].apply(**lambda** x: [word **for** word **in** x **if** **not** word **in** stop\_words])

# ### Lemmatization - Lemmatization is the process of converting a word to its base form.

df['tweet\_lemmatized'] = df['tweet\_token\_filtered'].apply(**lambda** x: ' '.join([lemmatizing.lemmatize(i) **for** i **in** x]))

df.head()

# ### Extracting features from lemmatied tweets with the help of Bag of words Feature

bow\_vectorizer = CountVectorizer(max\_df=0.90, min\_df=2, max\_features=1000, stop\_words='english')

bow\_main= bow\_vectorizer.fit\_transform(data['tweet\_lemmatized']).toarray()

bow\_main

# ### Prediction of sentiments with the help of our trained model

bow\_main = sc.fit\_transform(bow\_main)

main\_pred=model.predict(bow\_main)

main\_pred=pd.Series(main\_pred)

main\_pred.replace([0,1,-1],['neutral','positive','negative'],inplace=**True**)

main\_pred.head()

final=pd.concat([df["Text"],df["Date"],df["Location"]],axis=1)

final["Sentiment"]=main\_pred

final.head(15)

final['Date'] = final['Date'].dt.date

final.drop\_duplicates(subset='Text', keep='first', inplace=**False**)

final.drop(['Text'], axis=1)

final

**from** project\_lib **import** Project

project = Project(sc,"b1b16b09-e8e8-4dad-9443-7329983c8856","p-bb1cb98712cd7b785cb10f7bad5b2fda9de95b90")

project.save\_data(file\_name = "final.csv",data = final.to\_csv(index=**False**))

final