COVID ANALYZER

1.INTRODUCTION

1.1 Overview:

The Covid Analyzer is a one stop dock where people's emotions are brought into limelight by gathering their tweets and thus analysing their sentiments; taking into consideration each individuals perspective.

1.2 Purpose:

Covid 19 has created a major impact on our day to day lives. Extension of lockdown has caused a variety of mixed emotions which are released in a common hub.People feel free to emote here,which results in the boom of social media platforms such as Twitter, Instagram , Facebook etc.The goal of "The Covid Analyzer" is to understand their reactions towards this pandemic.

2.LITERATURE SURVEY

2.1 Existing Problem:

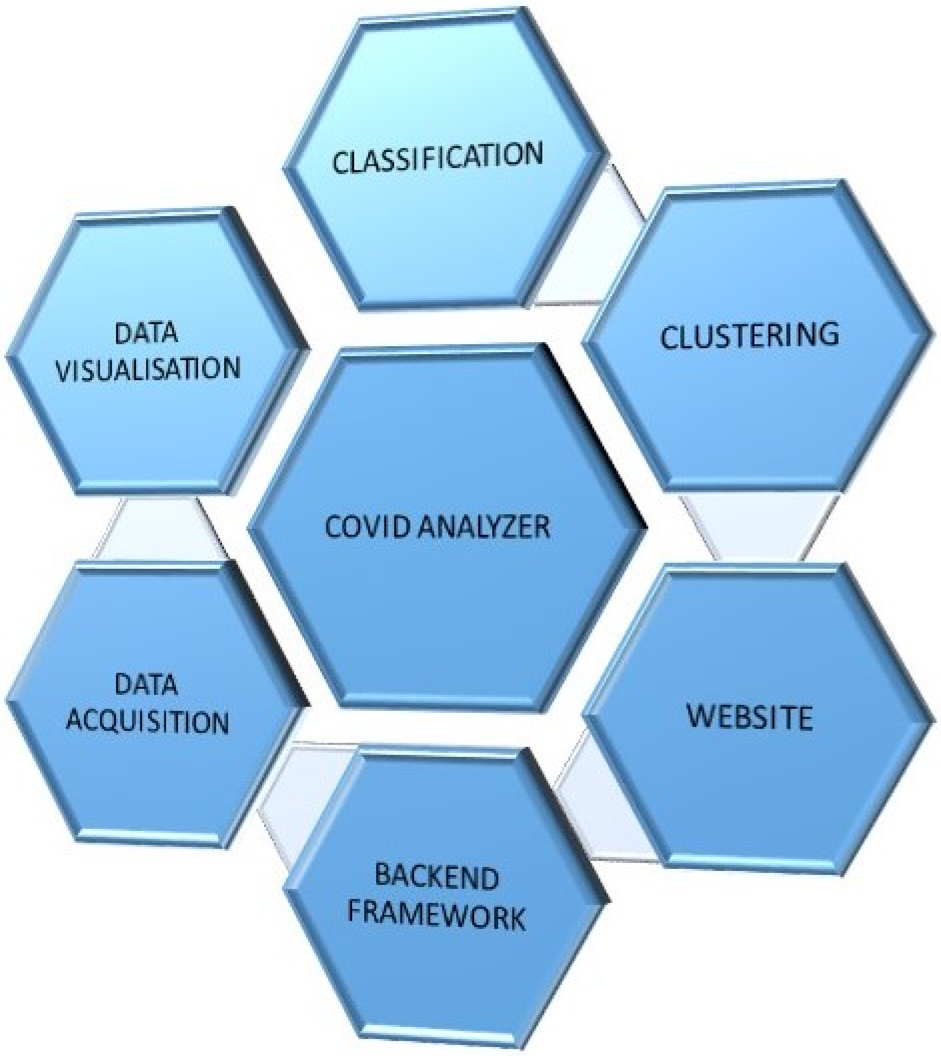
Ever since the pandemic started,people have been thorough in expressing their emotions via various media such as blogs or statuses or tweets, through which people try to share their undergoings.Need of the hour is that people would like to know the varied reactions of everyone and also share their combined sentiments.

2.2 Proposed Solution:

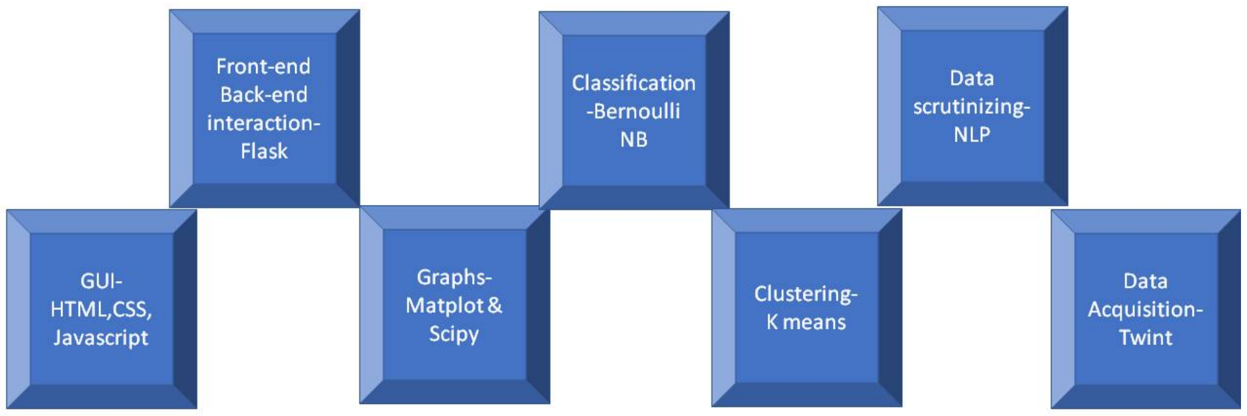
The way people express themselves during lockdown and the way they react to the Covid pandemic is very well captured by their feeds. A machine learning model which uses NLP(Natural Language Processing) to enact these sentiments in the stage, is a well reaching solution to this pandemic. This solution helps people analyse each individual's views pertaining to not only the lockdown,but also about the generalised views of Covid as such. Clustering is inbuilt to show the different emotions expressed by people with respect to a single domain. Each country's individual response is also captured and data analysis based on the language is determined. The approach towards the analysis is first supervised and later made as a hybrid terrain. The comments, likes for each post, verified accounts , followers count and even the no. of favourites of each comment is analysed and the model is trained respectively. Ultimately this model is trained to make people aware about the effects, Covid 19 has caused and the mixed reactions each individual has towards it, in different countries.

<u>3.THEORETICAL ANALYSIS</u>

<u>3.1 Block Diagram:</u>



3.2 Hardware/Software Designing:



4.EXPERIMENTAL INVESTIGATION

Steps involved in experimental investigation are as follows:

Step 1: Project Idea

With all the chaos generated due to the pandemic , having a clear cut idea about what is happening in the world during this time is vital. As a step towards achieving it , we have created a website that can effectively make people get further insights about all the happenings.

Step 2: Background Research

In order to familiarize people about the effects of the pandemic and about their responses and views, social networking sites are of great help.
By the technique of web scraping , data is obtained from these sites which helps in deriving numerous analytical conclusions.

Step 3: Hypothesis

Creating a dashboard with varied content that helps people understand each other's mindset and thoughts.

Step 4: Design of experiment

Based on the data acquired , an optimal way is chosen to group the data to convey the clear information .

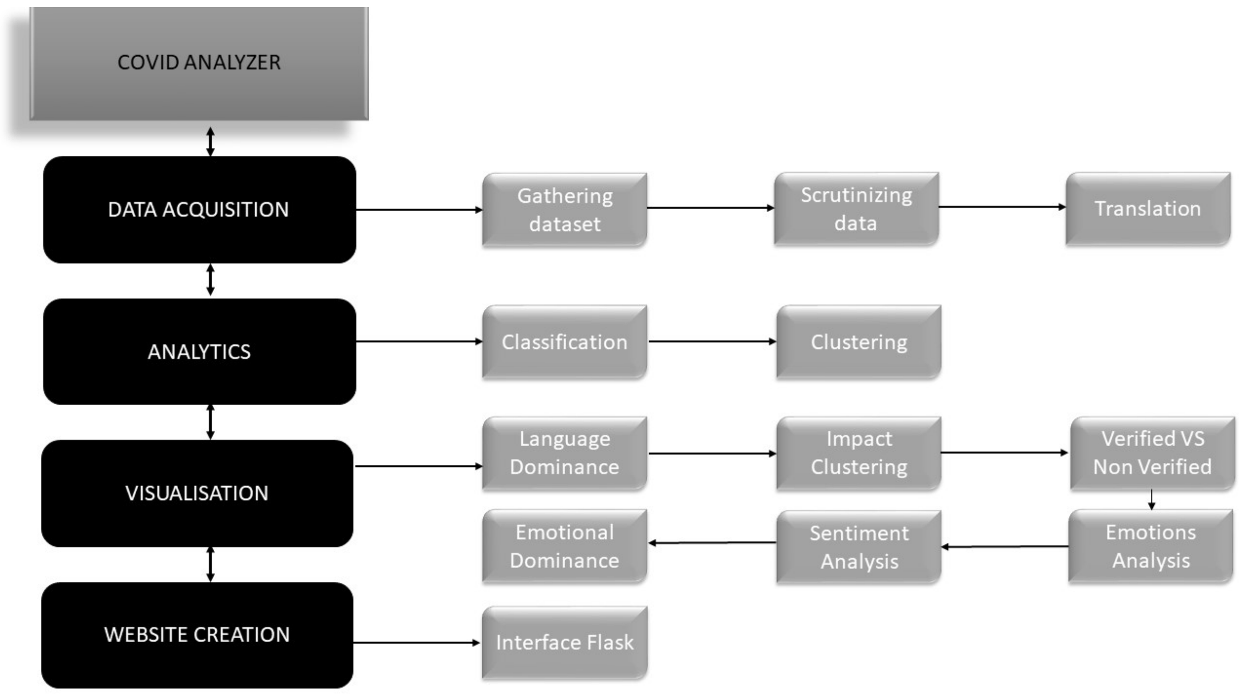
Step 5: Data Collection

Using web scraping and libraries like 'twint' , data from various backgrounds is gathered.

Step 6: Data Analysis and Conclusion

Analysis of data is made by employing various machine learning models based on classification and clustering. The data is visualized as donuts,histograms,etc using libraries like matplotlib.pyplot and scipy.interpolate.

5.FLOWCHART



6.RESULT

With "Covid Analyzer" , information about the existing pandemic can reach people in a better and an effective way.

7.ADVANTAGES AND DISADVANTAGES

By creating a dynamic website, timely information about the pandemic can be made facile. Further, the analysis of the thoughts through a website could make the information reach a wider range of people. With all the positivity depicted by the analysis , people could see through the good things that are still around which is quite necessary at this point of time.

Data acquisition is a challenge due to the restrictions imposed by sites like Twitter. A limit that is set on the amount of data that can be scraped makes it difficult to run complex programs that involves machine learning.

8.APPLICATIONS

1. Instilling a positive outlook towards the deadly virus
2. Changing the perspective of a common man towards this pandemic
3. Data Visualization

9.CONCLUSION

The notion of this project is to make people cognizant about the existing pandemic in every possible way so that they become more resilient.With "Covid Analyzer" in the market , this was made possible .

10.FUTURE SCOPE

In the positive side , a future without this website would be a happier option to all of us. With all the happenings due to the Coronavirus , a future that is free of this virus is the best that one can hope for.

11.BIBLIOGRAPHY

Flask-
https://www.youtube.com/channel/UC4JX40jDee_tINbkjycV4Sg

Matplot library-
https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

SciPyhttps://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html#:~:text=interp1d,-class%20scipy.interpolate&text=Interpolate%20a%201%2DD%20function,the%20value%20of%20new%20points.

Twint-
https://github.com/twintproject/twint

Kaggle-
https://www.kaggle.com/smid80/coronavirus-covid19-tweets

Time Series Visualizations-
https://towardsdatascience.com/introduction-to-interactive-time-series-visualizations-with-plotly-in-python-d3219eb7a7af

Implementation of Twint-
https://nealcaren.org/lessons/twint/

Favicon-
https://fontawesome.com/v4.7.0/icons/

Google Forms-
https://www.google.com/forms/about/

Unsupervised Sentiment Analysis-
https://towardsdatascience.com/unsupervised-sentiment-analysis-a38bf1906483

Preparing text for NLP-
https://medium.com/datadriveninvestor/preparing-text-for-natural-language-processing-f93b11bfb3fe

Sentiment Analyzer with text classification-
https://itnext.io/how-to-create-a-sentiment-analyzer-with-text-classification-python-ai-f3a5d10922c5

Sentiment Score-
https://www.kdnuggets.com/2018/08/emotion-sentiment-analysis-practitioners-guide-nlp-5.html

13.APPENDIX:

A.SOURCE CODE:

```python
#COVID ANALYZER

#Importing libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

#Importing Dataset

data=pd.read_csv('CORONA.CSV',engine='python')
X=data.iloc[:,4].values

#Filling missing data

def isNaN(string):
    return string != string

for i in range(463418):
    if(isNaN(X[i])):
        X[i]="coronavirus"
data['text']=X
```

```python

#Cleaning the texts

corpus = []
for i in range(0, 463418):
    rev=re.sub(r'http\S+', '',data['text'][i] )
    review = re.sub('[^a-zA-Z0-9]', ' ',rev)
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    sw=stopwords.words('english')
    sw.remove('not')
    review = [ps.stem(word) for word in review if not word in set(sw) ]
    review = ' '.join(review)
    corpus.append(review)


#Obtaining sentiment scores

nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sia=SentimentIntensityAnalyzer()
data['neg']=data['text'].apply(lambda x:sia.polarity_scores(x)['neg'])
data['neu']=data['text'].apply(lambda x:sia.polarity_scores(x)['neu'])
data['pos']=data['text'].apply(lambda x:sia.polarity_scores(x)['pos'])
data['compound']=data['text'].apply(lambda x:sia.polarity_scores(x)['compound'])

#Percentage values of sentiments

pos_review=[j for i,j in enumerate(data['text']) if data['compound'][i]>0.2]
neu_review=[j for i,j in enumerate(data['text']) if
0.2>=data['compound'][i]>=-0.2]
neg_review=[j for i,j in enumerate(data['text']) if data['compound'][i]<-0.2]
print("Percentage of positive review:{}%".format(len(pos_review)*100/len(data['text'])))
print("Percentage of neutral review:{}%".format(len(neu_review)*100/len(data['text'])))
print("Percentage of negative review:{}%".format(len(neg_review)*100/len(data['text'])))

#CLUSTERING USING LENGTH

#Finding length of tweets

length=[]
for i in range(463418):
    length.append(len(str(X[i])))
data['len']=length

#Elbow method

'''
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(Y)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
'''

#K-Means for Clustering using length

```

```python
Y=data.iloc[:,[17,16]].values
kmeans = KMeans(n_clusters =2, init = 'k-means++', random_state = 0)
y_kmeans = kmeans.fit_predict(Y)
fig,ax=plt.subplots()
ax.scatter(Y[y_kmeans == 1, 0],Y[y_kmeans == 1, 1], s = 10, c = 'red', label = 'Cluster 1')
ax.scatter(Y[y_kmeans == 0, 0],Y[y_kmeans == 0, 1], s = 10, c = 'blue', label = 'Cluster 2')
'''
plt.scatter(Y[y_kmeans == 2, 0],Y[y_kmeans == 2, 1], s = 30, c = 'green', label = 'Cluster 3')
plt.scatter(Y[y_kmeans == 3, 0],Y[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(arr[y_kmeans == 4, 0], arr[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(arr[y_kmeans == 5, 0], arr[y_kmeans == 5, 1], s = 100, c = 'black', label = 'Cluster 6')
plt.scatter(arr[y_kmeans == 6, 0], arr[y_kmeans == 6, 1], s = 100, c = 'violet', label = 'Cluster 7')
plt.scatter(arr[y_kmeans == 7, 0], arr[y_kmeans == 7, 1], s = 100, c = 'purple', label = 'Cluster 8')
plt.scatter(arr[y_kmeans == 8, 0], arr[y_kmeans == 8, 1], s = 100, c = 'grey', label = 'Cluster 9')
plt.scatter(arr[y_kmeans == 9, 0], arr[y_kmeans == 9, 1], s = 100, c = 'pink', label = 'Cluster 10')
'''
ax.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 30, c = 'yellow', label = 'Centroids')
plt.title('Clusters of people')
plt.xlabel('Length of tweet')
plt.ylabel('Sentiment score')
plt.legend()
plt.show()
fig.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/clus
terlength.png")

#CLUSTERING USING MAX FEATURES

#Obtaining bag of words model

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

#Finding number of max features

l=[]
for i in range(463418):
    s=0
    for j in range(1500):
        s=s+X[i][j]
    l.append(s)
data['len']=l

#Elbow method

'''
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
random_state = 42)
    kmeans.fit(Y)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
'''

#K-Means for Clustering using max features

```

```
kmeans = KMeans(n_clusters =2, init = 'k-means++',
  random_state = 0)
y_kmeans = kmeans.fit_predict(Y)
fig,ax=plt.subplots()
ax.scatter(Y[y_kmeans == 1, 0],Y[y_kmeans == 1, 1], s = 1,
  c = 'red', label = 'Cluster 1')
ax.scatter(Y[y_kmeans == 0, 0],Y[y_kmeans == 0, 1], s = 1,
  c = 'blue', label = 'Cluster 2')
'''
plt.scatter(Y[y_kmeans == 2, 0],Y[y_kmeans == 2, 1], s =
  1, c = 'green', label = 'Cluster 3')
plt.scatter(Y[y_kmeans == 3, 0],Y[y_kmeans == 3, 1], s =
  100, c = 'cyan', label = 'Cluster 4')
plt.scatter(arr[y_kmeans == 4, 0], arr[y_kmeans == 4, 1],
  s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(arr[y_kmeans == 5, 0], arr[y_kmeans == 5, 1],
  s = 100, c = 'black', label = 'Cluster 6')
plt.scatter(arr[y_kmeans == 6, 0], arr[y_kmeans == 6, 1],
  s = 100, c = 'violet', label = 'Cluster 7')
plt.scatter(arr[y_kmeans == 7, 0], arr[y_kmeans == 7, 1],
  s = 100, c = 'purple', label = 'Cluster 8')
plt.scatter(arr[y_kmeans == 8, 0], arr[y_kmeans == 8, 1],
  s = 100, c = 'grey', label = 'Cluster 9')
plt.scatter(arr[y_kmeans == 9, 0], arr[y_kmeans == 9, 1],
  s = 100, c = 'pink', label = 'Cluster 10')
'''
ax.scatter(kmeans.cluster_centers_[:, 0],
  kmeans.cluster_centers_[:, 1], s = 30, c = 'yellow', label
  = 'Centroids')
plt.title('Clusters of people')
plt.xlabel('Max features')
plt.ylabel('Sentiment score')
plt.legend()
plt.show()
fig.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/clus
  termax.png")
```

```

#VERIFIED AND NON-VERIFIED USING SENTIMENT SCORES

#Obtaining verified and non-verified accounts count

X=data.iloc[:,11].values
vcount=0
ncount=0
for i in range(463418):
    if(X[i]==1):
        vcount+=1
    else:
        ncount+=1

#Count of positive,negative and neutral tweets

v=[]
n=[]
for i in range(463418):
    if(X[i]==1):
        v.append(data['compound'][i])
    else:
        n.append(data['compound'][i])
vlist=[0,0,0]
nlist=[0,0,0]
for i in range(vcount):
    if(v[i]>0):
        vlist[0]+=1
    elif(v[i]==0):
        vlist[1]+=1
    else:
        vlist[2]+=1
for i in range(ncount):
    if(n[i]>0):
        nlist[0]+=1
    elif(n[i]==0):
        nlist[1]+=1
    else:
        nlist[2]+=1

#Pie Chart for Verified Tweets

labels = 'Positive','Neutral','Negative'
sizes=vlist
explode = (0.1, 0, 0)
fig1, ax1 = plt.subplots()
colors=['#008080','#CACACA','#575757']
ax1.pie(sizes, explode=explode, labels=labels,colors=colors, autopct='%1.1f%%',shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
fig1.savefig('C:/Users/pjai1/OneDrive/Desktop/IBM/images/verifiedpie.png',dpi=100, bbox_inches='tight', pad_inches=0.0)

#Pie Chart for Non-Verified Tweets

labels = 'Positive','Neutral','Negative'
sizes=nlist
explode = (0.1, 0, 0)
fig2, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels,colors=colors, autopct='%1.1f%%',shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
fig2.savefig('C:/Users/pjai1/OneDrive/Desktop/IBM/images/nonverifiedpie.png',dpi=100, bbox_inches='tight', pad_inches=0.0)

#LANGUAGE DOMINANCE

```

```
#Finding number of tweets in each language

Y=dataset.iloc[:,12].values
X=[]
Z=[]
langs=['es','zh','en','fr','nl','el','ja','th','hi','ar','pt','tr','tl','fa','et','ta','de','it','in','ru','bn','gu','kn','or','te','sl','ne','ro','lt','mr','pl','ur','ml','ko','ca','pa','vi','da','no','si','sv','cs','fi','ht','iw','eu','bg','cy','hy','am','sr','is','hu','lv','ps','sd','dv','uk','km','lo','ckb','ka']
la=[]
for i in langs:
    c=0
    for j in range(463418):
        if(Y[j]==i):
            c=c+1
    X.append(c)

#Creating a dictionary

dic={}
lan={}
for key in langs:
    for value in X:
        lan[key]=value
        X.remove(value)
        break
Z=[]
dic= sorted(lan.items(), key=lambda x: x[1], reverse=True)
for i in range(62):
    Z.append(dic[i][0])
for j in range(62):
    la.append(dic[j][1])

#Finding top five languages count
```

```python

Z=Z[:5]
la=la[:5]
fig,ax=plt.subplots()
ax.bar(Z[0],la[0],color="#000000")
ax.bar(Z[1],la[1],color="#0000CD")
ax.bar(Z[2],la[2],color="#00FFFF")
ax.bar(Z[3],la[3],color="#ADD8E6")
ax.bar(Z[4],la[4],color="#A9A9A9")
ax.legend(labels=la)
plt.xlabel('Languages')
plt.ylabel('Tweets')
plt.show()
fig.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/languages.png",dpi=100, bbox_inches='tight', pad_inches=0.0)

#VERIFIED AND NON-VERIFIED

posvalue=[0,0]
negvalue=[0,0]
neuvalue=[0,0]
for i in range(463418):
    if(data['compound'][i]>0):
        if(data['verified'][i]==1):
            posvalue[0]+=1
        else:
            posvalue[1]+=1
    elif(data['compound'][i]<0):
        if(data['verified'][i]==1):
            negvalue[0]+=1
        else:
            negvalue[1]+=1
    else:
        if(data['verified'][i]==1):
            neuvalue[0]+=1
        else:
            neuvalue[1]+=1

#Donut for Positive Sentiments

labels =['Verified','Non Verified']
sizes=posvalue
colors = ['#1515B4','#3BB9FF']
my_circle=plt.Circle( (0,0), 0.5, color='white')
patches, texts=plt.pie(sizes,colors=colors)
plt.rcParams['text.color'] = 'black'
plt.legend(patches, labels, loc="upper right")
plt.title('Positive')
p=plt.gcf()
plt.pie(sizes,colors=colors, wedgeprops = { 'linewidth':7, 'edgecolor' : 'white' },autopct='%1.1f%%',pctdistance=0.7)
p.gca().add_artist(my_circle)
p.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/posvernon.png")
plt.show()

#Donut for Negative Sentiments

labels =['Verified','Non Verified']
sizes=negvalue
colors = ['#6F4E37','#C85A17']
my_circle=plt.Circle( (0,0), 0.5, color='white')
patches, texts=plt.pie(sizes,colors=colors)
plt.rcParams['text.color'] = 'black'
plt.legend(patches, labels, loc="upper right")
plt.title('Negative')
p=plt.gcf()
plt.pie(sizes,colors=colors, wedgeprops = { 'linewidth':7, 'edgecolor' : 'white' },autopct='%1.1f%%',pctdistance=0.7)
p.gca().add_artist(my_circle)
p.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/negvernon.png")
```

```python
plt.show()

#Donut for Neutral Sentiments

labels =['Verified','Non Verified']
sizes=neuvalue
colors = ['#EC83D9','#ED1FC4']
my_circle=plt.Circle( (0,0), 0.5, color='white')
patches, texts=plt.pie(sizes,colors=colors)
plt.rcParams['text.color'] = 'black'
plt.legend(patches, labels, loc="upper right")
plt.title('Neutral')
p=plt.gcf()
plt.pie(sizes,colors=colors, wedgeprops = { 'linewidth':7, 'edgecolor' : 'white' },autopct='%1.1f%%',pctdistance=0.7)
p.gca().add_artist(my_circle)
p.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/neuvernon.png")
plt.show()

#TOTAL SENTIMENTS

value=[0,0,0]
for i in range(num):
    if(data['compound'][i]>0):
        value[0]+=1
    elif(data['compound'][i]==0):
        value[1]+=1
    else:
        value[2]+=1

#Plotting overall sentiments

labels ='Positive','Neutral','Negative'
sizes=value
fig1, ax1 = plt.subplots()
```

```
my_circle=plt.Circle( (0,0), 0.5, color='white')
plt.pie(sizes,labels=labels,labeldistance=1.2,autopct='%1.1f%%',pctdistance=0.7,wedgeprops = { 'linewidth' : 7, 'edgecolor' : 'white' })
plt.rcParams['text.color'] = 'black'
p=plt.gcf()
p.gca().add_artist(my_circle)
ax.legend(labels=['Positive','Neutral','Negative'])
p.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/posneg.png")
plt.show()

#IMPACT CLUSTERING

#Elbow Method
'''
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(Y)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
'''
#K Means Clustering for followers vs favourite count

Y=dataset.iloc[:,[8,6]].values
kmeans = KMeans(n_clusters =2, init = 'k-means++', random_state = 0)
y_kmeans = kmeans.fit_predict(Y)
fig,ax=plt.subplots()
ax.scatter(Y[y_kmeans == 0, 0],Y[y_kmeans == 0, 1], s =
30, c = 'red', label = 'Cluster 1')
ax.scatter(Y[y_kmeans == 1, 0],Y[y_kmeans == 1, 1], s = 30, c = 'blue', label = 'Cluster 2')
'''
plt.scatter(Y[y_kmeans == 2, 0],Y[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(Y[y_kmeans == 3, 0],Y[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(arr[y_kmeans == 4, 0], arr[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(arr[y_kmeans == 5, 0], arr[y_kmeans == 5, 1], s = 100, c = 'black', label = 'Cluster 6')
plt.scatter(arr[y_kmeans == 6, 0], arr[y_kmeans == 6, 1], s = 100, c = 'violet', label = 'Cluster 7')
plt.scatter(arr[y_kmeans == 7, 0], arr[y_kmeans == 7, 1], s = 100, c = 'purple', label = 'Cluster 8')
plt.scatter(arr[y_kmeans == 8, 0], arr[y_kmeans == 8, 1], s = 100, c = 'grey', label = 'Cluster 9')
plt.scatter(arr[y_kmeans == 9, 0], arr[y_kmeans == 9, 1], s = 100, c = 'pink', label = 'Cluster 10')
'''
ax.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = 'yellow', label = 'Centroids')
plt.title('Clusters of people')
plt.ylabel('No. of Followers')
plt.xlabel('Favourites count')
plt.legend()
plt.show()
fig.savefig("C:/Users/pjai1/OneDrive/Desktop/IBM/images/impactcluster.png",dpi=100, bbox_inches='tight', pad_inches=0.0)

corpus = []
for i in range(0,num):
    rev=re.sub(r'http\S+', '',data['text'][i] )
    review = re.sub('[^a-zA-Z0-9]', ' ',rev)
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    sw=stopwords.words('english')
    sw.remove('not')
    review = [ps.stem(word) for word in review if not word in set(sw) ]
    review = ' '.join(review)
    corpus.append(review)

#CLASSIFICATION

i=100000
cv = CountVectorizer(max_features = 1500)
X = cv.fit_transform(corpus).toarray()
X_train=[]
X_test=[]
y=[]
while i<num:
    y=data.iloc[:i,18].values
    data=pd.read_csv('CORONA.CSV',nrows=i+10000,engine="python")
    X_train=X[:i]
    X_test=X[i:i+10000]
    classifier = BernoulliNB()
    classifier.fit(X_train, y)
    y_pred = classifier.predict(X_test)
    y=y.tolist()
    if((num-i)>=10000):
        for k in range(10000):
            y.append(y_pred[k])
    else:
        for k in range(num-i):
            y.append(y_pred[k])
    data['labels']=pd.Series(y)
    i+=10000

data['labels']=pd.Series(y)


#LABELS VS VERIFIED AND NON-VERIFIED

Y=data.iloc[:,18].values
X=data.iloc[:,11].values
labels=['sad','happy','angry','informative']
v=[0,0,0,0]
n=[0,0,0,0]
for i in range(num):
    if(X[i]==1):
        if(Y[i]=='sad'):
            v[0]=v[0]+1
        elif(Y[i]=='happy'):
            v[1]=v[1]+1
        elif(Y[i]=='angry'):
            v[2]=v[2]+1
        else:
            v[3]=v[3]+1
    else:
        if(Y[i]=='sad'):
            n[0]=n[0]+1
        elif(Y[i]=='happy'):
            n[1]=n[1]+1
        elif(Y[i]=='angry'):
            n[2]=n[2]+1
        else:
            n[3]=n[3]+1

#Labels for verified accounts

fig,ax=plt.subplots()
ax.bar(labels[0],v[0],color="#000000")
```

```python
ax.bar(labels[1],v[1],color="#0000CD")
ax.bar(labels[2],v[2],color="#00FFFF")
ax.bar(labels[3],v[3],color="#ADD8E6")
ax.legend(labels=['sad','happy','angry','informative'])
plt.xlabel('Emotions')
plt.ylabel('No. of accounts')
plt.title('Emotions of Verified Users')
plt.show()
fig.savefig("../flask/images/labelsvsverified.png",dpi=100,
  bbox_inches='tight', pad_inches=0.0)

#Labels for non-verified accounts

fig,ax=plt.subplots()
ax.bar(labels[0],n[0],color="#000000")
ax.bar(labels[1],n[1],color="#0000CD")
ax.bar(labels[2],n[2],color="#00FFFF")
ax.bar(labels[3],n[3],color="#ADD8E6")
ax.legend(labels=['sad','happy','angry','informative'])
plt.xlabel('Emotions')
plt.ylabel('No. of accounts')
plt.title('Emotions of Non-Verified Users')
plt.show()
fig.savefig("../flask/images/labelsvsnv.png",dpi=100,
  bbox_inches='tight', pad_inches=0.0)


#LABELS VS SENTIMENTS

W=data.iloc[:,18].values

#Encoding data

labelencoder_X = LabelEncoder()
labelencoder_X.fit_transform(W)
data['new']=W
```

```
Z=data.iloc[:,[19,16]].values

#Elbow method
'''
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
 random_state = 42)
    kmeans.fit(Z)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
'''

# K Means for sentiment score vs labels
kmeans = KMeans(n_clusters = 4, init = 'k-means++',
 random_state = 42)
y_kmeans = kmeans.fit_predict(Z)
plt.scatter(Z[y_kmeans == 0, 0], Z[y_kmeans == 0,
 1],marker='*',s = 100, c = 'red', label = 'Angry')
plt.scatter(Z[y_kmeans == 1, 0], Z[y_kmeans == 1,
 1],marker='P', s = 70, c = 'blue', label = 'Sad')
plt.scatter(Z[y_kmeans == 2, 0], Z[y_kmeans == 2,
 1],marker='8', s = 30, c = 'green', label = 'Informative')
plt.scatter(Z[y_kmeans == 3, 0], Z[y_kmeans == 3,
 1],marker='2', s = 500, c = 'black',label = 'Happy' )
#plt.scatter(kmeans.cluster_centers_[:, 0],
 kmeans.cluster_centers_[:, 1], s = 30, c = 'yellow', label
 = 'Centroids')
plt.title('Sentiments VS Emotions')
plt.xlabel('Emotion')
plt.ylabel('Sentiment score')
ax = plt.subplot(111)
```

```python
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width*0.65,
  box.height])
legend_x = 1.5
legend_y = 0.5
plt.legend(loc="center right",bbox_to_anchor=(legend_x,
  legend_y))
plt.savefig("../flask/images/sentivsemo.png",dpi=100,
  bbox_inches='tight', pad_inches=0.0)

#ANALYSIS OF LABELS

#Label-SAD

Y=dataset.iloc[:,18].values
X=dataset.iloc[:,11].values
labels=['sad','happy','angry','informative']
Z=[]
for i in labels:
    c=0
    for j in range(num):
        if(Y[j]==i):
            c=c+1
    Z.append(c)
svno=[0,0]
for i in range(num):
    if(Y[i]=='sad'):
        if(X[i]==1):
            svno[0]=svno[0]+1
        else:
            svno[1]=svno[1]+1

#Pie Chart for Sad

labels = 'Verified','Non-Verified'
sizes=svno
```

```
explode = (0.1, 0)
colors=['#A74AC7','#461B7E']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode,
  labels=labels,colors=colors, autopct='%1.1f%%',shadow=True,
  startangle=180)
ax1.axis('equal')
plt.title('Sad comments')
fig1.savefig("../flask/images/Sad.png")

#Label-HAPPY

hvno=[0,0]
for i in range(num):
    if(Y[i]=='happy'):
        if(X[i]==1):
            hvno[0]=hvno[0]+1
        else:
            hvno[1]=hvno[1]+1

#Pie Chart for Happy

labels = 'Verified','Non-Verified'
sizes=hvno
explode = (0.1, 0)
colors=['#FFDB58','#8B4513']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode,colors=colors,
  labels=labels, autopct='%1.1f%%',shadow=True,
  startangle=180)
ax1.axis('equal')
plt.title('Happy comments')
plt.show()
fig1.savefig("../flask/images/Happy.png")

#Label-ANGRY
```

```python

avno=[0,0]
for i in range(num):
    if(Y[i]=='angry'):
        if(X[i]==1):
            avno[0]=avno[0]+1
        else:
            avno[1]=avno[1]+1

#Pie Chart for Angry

labels = 'Verified','Non-Verified'
sizes=avno
colors=['#F75D59','#CC0000']
explode = (0.1, 0)

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels,colors=colors, autopct='%1.1f%%',shadow=True, startangle=180)
ax1.axis('equal')
plt.title('Angry comments')
plt.show()
fig1.savefig("../flask/images/Angry.png")

#Label-INFORMATIVE

ivno=[0,0]
for i in range(num):
    if(Y[i]=='informative'):
        if(X[i]==1):
            ivno[0]=avno[0]+1
        else:
            ivno[1]=avno[1]+1

#Pie Chart for Informative
```

```

labels = 'Verified','Non-Verified'
sizes=ivno
colors=['#FFB6C1','#FF1493']
explode = (0.1, 0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode,colors=colors,
  labels=labels, autopct='%1.1f%%',shadow=True,
  startangle=180)
ax1.axis('equal')
plt.title('Informative comments')
plt.show()
fig1.savefig("../flask/images/Informative.png")

#OVERALL CHART FOR LABELS

#Pie Chart for Labels

labels = 'Angry','Happy','Sad','Informative'
sizes=Z
explode = (0, 0.1,0,0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels,
  autopct='%1.1f%%',shadow=True,startangle=180)
ax1.axis('equal')
plt.title('Tweet emotions')
plt.show()
fig1.savefig("../flask/images/Labels.png")

#Donut Chart

labels =['Angry','Happy','Sad','Informative']
sizes=Z
colors = ['#EC83D9','#A74AC7','#461B7E','#ED1FC4']
my_circle=plt.Circle( (0,0), 0.5, color='white')
patches, texts=plt.pie(sizes,colors=colors)
```

```python
plt.rcParams['text.color'] = 'black'
plt.legend(patches, labels, loc=1)
plt.title('Tweet Emotions')
p=plt.gcf()
plt.pie(sizes,colors=colors, wedgeprops = { 'linewidth':2,
  'edgecolor' : 'white' },autopct='%1.1f%%',pctdistance=0.7)
p.gca().add_artist(my_circle)
p.savefig("../flask/images/Labels1.png")
plt.show()


#Flask code
from flask import Flask ,render_template
from flask_fontawesome import FontAwesome
import stats
import titletophome
import tweetstop
import CovidAnalyzer
import sentitop
app = Flask(__name__)
fa = FontAwesome(app)


@app.route('/home')
def home():

    return
  render_template("index.html",tottweet=titletophome.tottweet
  ,engtweet=titletophome.engtweet,pos=sentitop.pnn[0],info=ti
  tletophome.info,ver=titletophome.ver,date=titletophome.date
  )
@app.route('/positive')
def positive():

    return
  render_template("index2.html",var1=stats.favb,var2=stats.fa
vbfols,var3=stats.folbfavs)


@app.route('/tweets')
def tweets():
    return render_template("index3.html",info0=tweetstop.info[0],info1=tweetstop.info[1],info2=tweetstop.info[2],sad0=tweetstop.sad[0],sad1=tweetstop.sad[1],sad2=tweetstop.sad[2],angry0=tweetstop.angry[0],angry1=tweetstop.angry[1],angry2=tweetstop.angry[2],happy0=tweetstop.happy[0],happy1=tweetstop.happy[1],happy2=tweetstop.happy[2])

@app.route('/sentiment')
def sentiment():
    return render_template("index4.html",post=sentitop.pnn[0],negt=sentitop.pnn[1],neut=sentitop.pnn[2],posp=CovidAnalyzer.posp,neup=CovidAnalyzer.neup,negp=CovidAnalyzer.negp)

@app.route('/data_visualization')
def data_visualisation():
    return render_template("ChartJs.html")
if __name__ =='__main__':
    app.run()


```