Configuration Manual

MSc Research Project
M.Sc. in Data Analytics

Anne Guilcher
Student ID: x16132068

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Anne Guilcher		
Student ID:	x16132068		
Programme:	Master of Science - Data Analytics	**Year:**	2022
Module:	M.Sc. Research Project		
Lecturer:	Vladimir Milosavljevic		
Submission Due Date:	15/08/2022		
Project Title:	Configuration Manual for the following research paper: "Using artificial intelligence techniques to analyse social media content on COVID-19 children vaccination programs"		
Word Count:	2079 **Page Count:** 67		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Anne Guilcher

Date: 14/08/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anne Guilcher
x16132068

1 Introduction

This configuration manual is a support document for the following research paper: "Using artificial intelligence techniques to analyse social media content on COVID-19 children vaccination programs". This document gives an overview of the computational environment used to implement this project as well as highlight key parts of the code and some of the graphs and outputs generated while researching, developing and comparing various artificial intelligence techniques to analyse sentiment attached to tweets related to the vaccination of children against the COVID-19 virus.

2 Specifications

Specifications and requirements to develop and run files developed for this research project are listed in the following sub-sections.

2.1 Hardware specifications

The hardware specifications of the computer used to implement this research project can be seen in table 1 and figure 1.

Hardware	Configuration
System	ASUSTek COMPUTER INC.
Processor	Intel® Core™ i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Cores, 8 Logical Processors
Operating System	Microsoft Windows 10 Home (64 bit)
Installed Physical Memory (RAM)	16 GB
Total Physical Memory	15.7 GB
Available Physical Memory	4.79 GB
Total Virtual Memory	46.7 GB
Available Virtual Memory	30.8 GB
Hard Drive	952 GB
Graphic Card	Intel® Iris® Plus Graphics

Table 1 – Hardware Specifications

OS Name	Microsoft Windows 10 Home
Version	10.0.19044 Build 19044
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	LAPTOP-T35D2GUB
System Manufacturer	ASUSTeK COMPUTER INC.
System Model	ZenBook UX393JA_UX393JA
System Type	x64-based PC
System SKU	
Processor	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	American Megatrends Inc. UX393JA.304, 28/01/2021
SMBIOS Version	3.2
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	ASUSTeK COMPUTER INC.
BaseBoard Product	UX393JA
BaseBoard Version	1.0
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United Kingdom
Hardware Abstraction Layer	Version = "10.0.19041.1806"
Username	LAPTOP-T35D2GUB\aggui
Time Zone	GMT Summer Time
Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	15.7 GB
Available Physical Memory	4.79 GB
Total Virtual Memory	46.7 GB
Available Virtual Memory	30.8 GB
Page File Space	31.0 GB

Fig. 1 – Device specifications

2.2 Software required

The list of software used while developing the application can be seen in table 2.

Software	Details
IDE	PyCharm 2020.2.3, Jupyter Notebook 6.0.3, Google Colab Pro
Language	Python 3.8.10
SPSS	SPSS 27

Table 2 – Software used

2.3 Creation of an academic research account on Twitter developer portal

This project required the creation of an academic research account on Twitter to scrape over one million tweets from this social media platform, see figure 2.

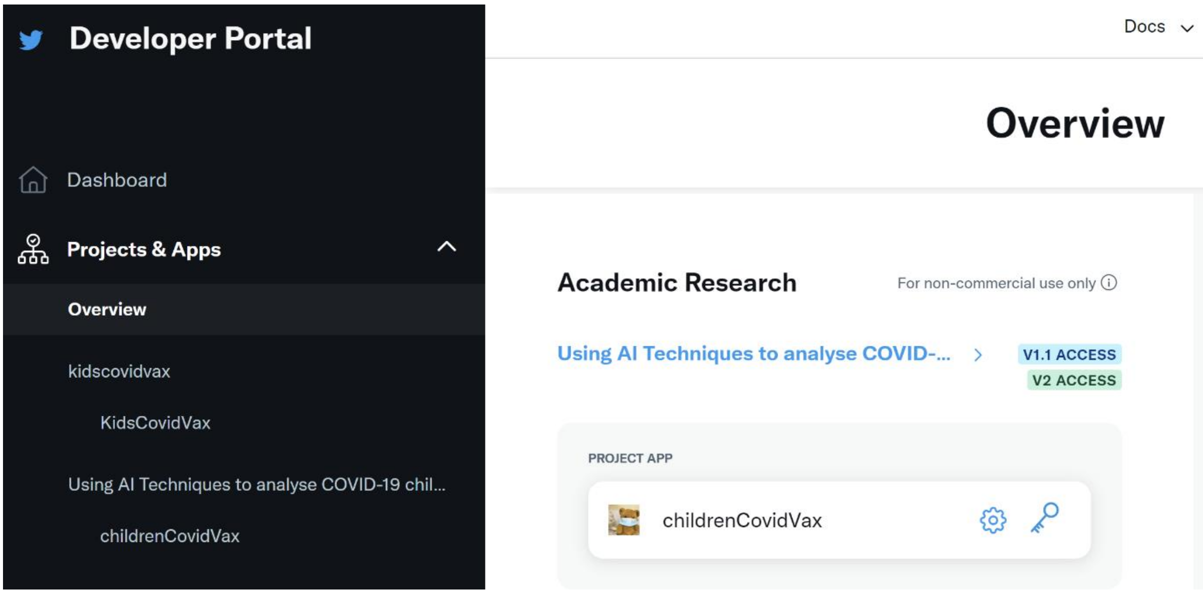


Fig. 2 – Academic Research Account -Twitter

2.4 Python libraries

The list of libraries used in this project can be seen in table 3.

Name	Description
Twarc	Scrape tweets
numpy	
panda	Data structure and analysis
Pyplot (matplotlib)	
contractions	To expand words contractions
spacy	NLP library
string	
pandas	
vaderSentiment	Sentiment analysis
stopwords	
time	
seaborn	
nltk	Natural language processing
sklearn	
torch	Deep learning
collections	Tokenization
gensim	Tokenization
transformers	For models such as BERT, DistilBERT
random	Seed for reproducibility
yellowbrick	Visualisation
pickle	Store python objects
xgboost	Machine learning
re	Regular expressions
imblearn	Resampling
glob2	

math	
mathplotlib	Data analysis and numerical plotting
os	
numpy	Scientific computing
wordcloud	
wordninja	Natural language processing
spellchecker	
random	
plotly	
keras	
termcolor	
gensim	Topic and vector space modelling, document indexing and similarity retrieval
pyLDAvis	Interactive topic model visualisation

Table 3 – Libraries

2.5 GitHub repository

The tweets and covid metrics data sets were extensively manipulated to analyse and predict sentiment attached to the vaccination of children against the COVID-19 virus. The various csv files can be found in GitHub (Guilcher, 2022).

3 Python Files

3.1 TweetsScraper.py file

This file was used to scrape over a million tweets. Tweets were extracted a month at a time, from February 2020 to June 2022. Monthly extracts were saved in a JSON format. Some code extract from this Python file can be seen in figure 3.

```python
from twarc import Twarc2, expansions
import datetime
import json

# Bearer token
client = Twarc2(
    bearer_token="[redacted]


def main():
    # Start time in UTC for the time period
    start_time = datetime.datetime(2020, 2, 1, 0, 0, 0, 0, datetime.timezone.utc)

    # End time in UTC for the time period
    end_time = datetime.datetime(2022, 6, 30, 0, 0, 0, 0, datetime.timezone.utc)

    # Query
    query = "(corona OR coronavirus OR covid-19 OR covid)(kids OR child OR children OR kid)(vaccine OR vaccination OR vax) lang:en

    # search_all method calls the full-archive search endpoint to get Tweets based on the query, start and end times
    search_results = client.search_all(query=query, start_time=start_time, end_time=end_time, max_results=100)

    # Twarc returns all Tweets for the criteria set above; we page through the results
    for page in search_results:
        # The Twitter API v2 returns the Tweet information and the user, media etc.  separately
        # We use expansions.flatten to get all the information in a single JSON
        result = expansions.flatten(page)
        for tweet in result:
            # Print the full Tweet object JSON to the console
            # print(json.dumps(tweet))
            with open("covid_vax_results.jsonl", "a") as f:
                f.write(json.dumps(tweet) + "\n")
            print("Wrote a page of results...")


if __name__ == "__main__":
    main()
```

Fig. 3 – Code extract - TweetsScraper.py

3.2 JsonToCsvTransformer.py file

JSON files were transformed into csv files using this Python file. This file was run for each of the monthly extract. An extract from this Python file can be seen in figure 4.

```python
data_repo = "C:\\Users\\aggui\\Desktop\\Msc\\scrappedTweets\\2022\\eng-06-2022\\"
jsonFileName = "covid_vax_results.jsonl"
csvFileName = "covid_vax_results.csv"

import pandas as pd

df = pd.read_json(data_repo+jsonFileName, lines = True)
print(df.head())

from twarc_csv import CSVConverter
print("Converting to CSV...")
with open(data_repo+jsonFileName, "r", encoding="utf-8") as infile:
    with open(data_repo+csvFileName, "w", encoding="utf-8", newline='') as outfile:
        converter = CSVConverter(infile, outfile)
        converter.process()

print("Finished.")
```

Fig. 4 – Code extract - JsonToCsvTransformer.py

3.3 parseCaseData.py file

This Python file was written to manipulate the csv file downloaded from 'Our World in Data' website (Our World in Data, 2022). Data was filtered as the analysis focused on worldwide data, not per country. Irrelevant columns were dropped. This resulted in the creation of a new csv file which was used while carrying out a regression analysis (see section 3.17). A code extract from this Python file can be seen in figure 5.

```
import pandas as pd

data_root = "C:\\Users\\aggui\\Desktop\\Msc\\data\\"
fileName = "owid-covid-data.csv"
fileNameFilteredData = "owid-covid-data-filtered.csv"
fileNameFilteredMergedData = "owid-covid-data-filtered-merged.csv"

def filterWorldData():
    df = pd.read_csv(data_root+fileName)

    filter = df['iso_code'] == 'OWID_WRL'
    #print(filter) # notice the boolean list based on filter criteria

    df2 = df[filter]  # next we use that boolean list to filter data

    nan_value = float("NaN")
    df2.replace(0, nan_value, inplace=True)
    df2.replace("", nan_value, inplace=True)
    df2.dropna(how='all', axis=1, inplace=True)
    print(df2)
    df2['created_at'] = pd.to_datetime(df2['date'])
    print(df2['created_at'])
    df2 = df2.sort_values(by='created_at', ascending=False)
    print(df2.columns)
    print(df2.shape)
    print(df2.info())
    #uncomment this out to create a new csv file
    df2.to_csv(data_root+fileNameFilteredData)
def retrieveWorldData():
    return pd.read_csv(data_root+fileNameFilteredData)

with open("./variables.txt", "r") as f:
    variables = f.readlines()

var_list = [d.split('=')[1].split('\n')[0] for d in variables]
data_repo = var_list[0]
csvCleanedDataset = "cleaned_tweets_dataset.csv"
df_tweets = pd.read_csv(data_repo+csvCleanedDataset)

print(df_tweets.columns)
print(df_tweets.shape)
print(df_tweets.info())
world_data = retrieveWorldData()
df_tweets['created_at'] = pd.to_datetime(df_tweets['created_at'])
df_tweets = df_tweets.merge(world_data, on='created_at', how='left')

print(df_tweets.columns)
print(df_tweets.shape)
print(df_tweets.info())


df_tweets = df_tweets.drop(['Unnamed: 0'], axis=1)
df_tweets = df_tweets.drop(['lang'], axis=1)

df_tweets.to_csv(data_root+fileNameFilteredMergedData)
```

Fig. 5 – Code extract - parseCaseData.py

3.4 filesConcatenator.py file

This Python file was used to concatenate the monthly csv files into a global one for the period February 2020 to June 2022. An extract from this Python file can be seen in figure 6.

```
import glob2
import os
import pandas as pd

#os.chdir("C:\\Users\\aggui\\Desktop\\Msc\\scrappedTweets\\concat\\merged\\2020\\")
os.chdir("C:\\Users\\aggui\\Desktop\\Msc\\scrappedTweets\\concat\\merged\\all\\tweets-world-covid-data-dropped-f
extension = 'csv'
all_filenames = [i for i in glob2.glob('*.{}'.format(extension))]
#combine all files in the list
combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames ])
#export to csv
combined_csv.to_csv("combined_csv.csv", index=False,header=True, encoding='utf-8-sig')
```

Fig. 6 – Code extract - filesConcatenator.py

3.5 KidsVaxUtilities.py file

This Python file holds multiple utility functions such as the ones shown in figure 7.

```
def modelPreparation(df):
    text_counts = countVector(df)
    le = preprocessing.LabelEncoder()
    df['sentiment'] = le.fit_transform(df['sentiment'])
    # display(df)
    x_train, x_test, y_train, y_test = train_test_split(text_counts, df['sentiment'], test_size=0.20, random_sta

    # Naive Bayes Classification
    print("In modelPreparation method, Naive Bayes:")
    cnb = ComplementNB()
    cnb.fit(x_train, y_train)
    print("Train accuracy {:.2f}%".format(cnb.score(x_train, y_train) * 100))
    print("Test accuracy {:.2f}%".format(cnb.score(x_test, y_test) * 100))
    y_pred = cnb.predict(x_test)
    cf_matrix = confusionMatrix(y_test,y_pred)
    print(cf_matrix)

    ## Display the visualization of the Confusion Matrix.
    plt.show()
    print("Accuracy Score:")
    print(accuracy_score(y_test, y_pred))
    print("Confusion Matrix:")
    plot_confusion_matrix(cnb, x_test, y_test)
    #ConfusionMatrixDisplay.from_predictions(x_test, y_test)
    plt.show()
    print("Classification Report:")
    #classes = ["Positive", "Negative"]
    classes = ["Negative", "Neutral","Positive"]
```

```python
def tweetSource(df):
    source_df = df['source'].value_counts().to_frame().reset_index().rename(
        columns={'index': 'source', 'source': 'count'})[:15]
    display(source_df.head(10))
    fig = go.Figure(go.Bar(
        x=source_df['source'], y=source_df['count'],
        marker={'color': source_df['count'],
                'colorscale': 'blues'},
        text=source_df['count'],
        textposition="outside",
    ))
    fig.update_layout(title_text='Top Sources ', xaxis_title="Sources", yaxis_title="Count ",
                      template="plotly_dark", title_x=0.5)
    fig.show()

    df['source'] = df['source'].fillna('NA')
    df = df['source'].value_counts().to_frame()

    total = df['source'].sum()
    df['percentage'] = 100 * df['source'] / total
    field = 'source'
    percent_limit = 0.5
    otherdata = df[df['percentage'] < percent_limit]
    others = otherdata['percentage'].sum()
    maindata = df[df['percentage'] >= percent_limit]

    df = maindata
    other_label = "Others(<" + str(percent_limit) + "% each)"
    df.loc[other_label] = pd.Series({field: otherdata['source'].sum()})
```

```
data = pd.read_csv(data_repo+csvCleanedDataset)

print(data['sentiment'].value_counts())
print(data['sentiment_score'].value_counts())
# plot sentiment counts
fig = plt.figure(figsize=(10, 6))
data['sentiment'].value_counts().sort_index().plot.bar()
plt.xlabel('Sentiment Label', fontsize=18)
plt.ylabel('Tweet Count', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Sentiment Distribution in data")
plt.show()
plt.tight_layout()

# Get an extract of the dataframe
df = data.sample(frac = 0.2, random_state = 0)
print(round(df.describe(),2))
print(df.info())
print("Sentiment value count in df")
print(df['sentiment'].value_counts())
print(df['sentiment_score'].value_counts())
# plot sentiment counts
fig = plt.figure(figsize=(10, 6))
df['sentiment'].value_counts().sort_index().plot.bar()
plt.xlabel('Sentiment Label', fontsize=18)
plt.ylabel('Tweet Count', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Sentiment Distribution in df")
plt.show()
```

Fig. 7 – Code extract - KidsVaxUtilities.py

3.6 balancedDatasetCreator.py file

This Python file was written to clean and transform the tweets csv file. This was a crucial step prior to carrying out the sentiment analysis. Contractions and slang words were transformed. Mentions, hashtags, retweets, hyperlinks, punctuation, stop words, single characters, emojis were removed. Words tokens were created from tweets text-content and added into a new column. The date (created_at column) was transformed as the analysis focused on the date, not the time the tweet was posted at. Negative, neutral, positive and compound scores were computed and added into new columns. Polarity and subjectivity were calculated as well and added into new columns. The dataset was resampled. Multiple csv files were created with this transformed data to facilitate the sentiment analysis. Figure 8 is an amalgamation of multiple extracts from this Python file.

```
# Cleaning and transformation of tweets text-content necessary for sentiment analysis:
# Transform contractions and slang
# Remove mentions, hashtags, retweets, hyperlinks, punctuation, stop words, single characters, emojis
# Creation of words tokens and addition of new words columns in dataset
# Transform creation tweets date
# Addition of relevant columns necessary for sentiment analysis: Negative_Score, Neutral_Score,
# Positive_Score, Compound_Score
# Addition of polarity and subjectivity columns to dataset
# Resample dataset
# Creation of multiple csv files with transformed data prior to sentiment analysis

import numpy as np
import matplotlib.pyplot as plt
import re
from textblob import TextBlob
import contractions
import nltk
from sklearn.utils import resample

from pandas.io.formats import string
from KidsVaxUtilities import handleNull, deleteColumns, sentimentAnalyzer
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import pandas as pd
pd.options.mode.chained_assignment = None

nltk.download('wordnet')
nltk.download('stopwords')
from nltk.corpus import stopwords

with open("./variables.txt", "r") as f:
```

```
with open("./variables.txt", "r") as f:
    variables = f.readlines()

var_list = [d.split('=')[1].split('\n')[0] for d in variables]
data_repo = var_list[0]
# data_repo = "C:\\Users\\aggui\\Desktop\\Msc\\scrappedTweets\\2020\\eng-02-2020\\"
csvFileName = "covid_vax_results.csv"
csvFileWithoutEmptyRowsName = "covid_vax_results_without_empty_row.csv"
csvFileWithoutEmptyRowsWithSentimentName = "covid_vax_results_with_Sentiment.csv"
csvBalancedFileName = "covid_vax_results_balanced_sentiment.csv"
csvBalancedCleanedPolaritySubjectivityFileName = "dataset_cleaned_with_polarity_subjectivity.csv"
csvExtract = "cleaned_extract.csv"
csvBalancedExtract = "csvBalancedExtract.csv"
csvCleanedDataset = "cleaned_tweets_dataset.csv"

vader = SentimentIntensityAnalyzer()

def load_dict_contractions():
    return {
        "cant":"can not",
        "dont":"do not",
        "wont":"will not",
        "ain't":"is not",
        "amn't":"am not",
        "aren't":"are not",
        "can't":"cannot",
        "'cause":"because",
        "couldn't":"could not",
        "couldn't've":"could not have",
        "could've":"could have",
```

with open("./variables.txt", "r...

```python
def load_dict_contractions():
    return {
        "cant":"can not",
        "dont":"do not",
        "wont":"will not",
        "ain't":"is not",
        "amn't":"am not",
        "aren't":"are not",
        "can't":"cannot",
        "'cause":"because",
        "couldn't":"could not",
        "couldn't've":"could not have",
        "could've":"could have",
        "daren't":"dare not",
        "daresn't":"dare not",
        "dasn't":"dare not",
        "didn't":"did not",
        "doesn't":"does not",
        "don't":"do not",
        "e'er":"ever",
        "em":"them",
        "everyone's":"everyone is",
        "finna":"fixing to",
        "gimme":"give me",
        "gonna":"going to",
        "gon't":"go not",
        "gotta":"got to",
        "hadn't":"had not",
        "hasn't":"has not",
        "haven't":"have not",
```

```python
def normalization(text):
    text = text.str.lower()
    # Number
    text = re.sub(r'@[A-Za-z0-9]+', '', text)
    text = re.sub(r'#[A-Za-z0-9]+', '', text)
    text = re.sub(r'RT[\s]+', '', text)
    text = re.sub(r'https?:\/\/\S+', '', text)  # remove hyperlink
    text = ''.join([i for i in text if not i.isdigit()])

    for punct in string.punctuation:
        text = text.replace(punct, " ")

    # Contractions
    CONTRACTIONS = load_dict_contractions()
    text = text.replace("’", "'")
    words = text.split()
    reformed = [CONTRACTIONS[word] if word in CONTRACTIONS else word for word in words]
    text = " ".join(reformed)

    # Remove stop words
    text = " ".join([word for word in text.split() if not word in stop_words])
    # Remove single characters (etc: i/I)
    text = ' '.join([w for w in text.split() if len(w) > 1 and w != 'a' and w != 'i'])
    return text
def remove_empty_rows_dataset(df):
    df.dropna(axis=0, how='all',inplace=True)
    #uncomment to create covid_vax_results_without_empty_row
    df.to_csv(data_repo+csvFileWithoutEmptyRowsName, index=False)
```

```python
def make_resample(_df, column):
  # defining Sentiments:
  sentiments = ['negative', 'neutral', 'positive']
  _df = _df.assign(sentiment_score=_df.sentiment.apply(lambda x: sentiments.index(x)))
  print(_df.info())
  print(_df.tail())
  print(_df.head())
  dfs_r = {}
  dfs_c = {}
  bigger = 0
  ignore = ""
  for c in _df[column].unique():
    dfs_c[c] = _df[_df[column] == c]
    if dfs_c[c].shape[0] > bigger:
      bigger = dfs_c[c].shape[0]
      ignore = c

  for c in dfs_c:
    if c == ignore:
      continue
    dfs_r[c] = resample(dfs_c[c],
                        replace=True,
                        n_samples=bigger - dfs_c[c].shape[0],
                        random_state=0)
  return pd.concat([dfs_r[c] for c in dfs_r] + [_df])
def balance_dataset(df):
    print("In balance_dataset")
    from sklearn.utils import resample
    # Transform into binary classification
    # df['balance'] = [1 if b=='Positive' else 0 for b in df.Sentiment]
    print("Sentiment value count")
    print(df['Sentiment'].value_counts())
    # Separate majority and minority classes
    df_majority = df[df.Sentiment == '1'] #positive
    df_minority = df[df.Sentiment == '0'] #negative
    print(df_majority.shape[0])
    print(df_minority.shape[0])
    difference = df_majority.shape[0]- df_minority.shape[0]
    print("difference: ", difference)
    print(df.shape[0])
    print(df.shape[1])
    # Upsample minority class
    df_minority_upsampled = resample(df_minority,
                                     replace=True,   # sample with replacement
                                     n_samples=df_majority.shape[0],   # to match majority class
                                     random_state=123)  # reproducible results

    # Combine majority class with upsampled minority class
    df_upsampled = pd.concat([df_majority, df_minority_upsampled])
    # Display new class counts
    print("Sentiment upsampled value count")
    print(df_upsampled.Sentiment.value_counts())
    #uncomment to create csv
    df_upsampled.to_csv(data_repo + csvBalancedFileName)
    return df_upsampled
```

```python
def vader_scores(feedbacktext, category):
    return vader.polarity_scores(feedbacktext).get(category)


def analyzeSentiment(fileName):
    tweetDF = pd.read_csv(fileName,low_memory=False)

    tweetDF = tweetDF.assign(Sentiment=tweetDF.text.apply(lambda text: sentimentAnalyzer(text)))
    print("In analyzeSentiment - Created Sentiment column")

    tweetDF["Negative_Score"] = tweetDF.apply(lambda row: vader_scores(tweetDF["text"][row.name], "neg"), axis=1)
    tweetDF["Neutral_Score"] = tweetDF.apply(lambda row: vader_scores(tweetDF["text"][row.name], "neu"), axis=1)
    tweetDF["Positive_Score"] = tweetDF.apply(lambda row: vader_scores(tweetDF["text"][row.name], "pos"), axis=1)
    tweetDF["Compound_Score"] = tweetDF.apply(lambda row: vader_scores(tweetDF["text"][row.name], "compound"), axis=1)

    tweetDF1 = handleNull(tweetDF)
    tweetDF1 = deleteColumns(tweetDF1)
    print("In analyzeSentiment - handled nulls")
    #uncomment to create csv
    tweetDF1.to_csv(data_repo+csvFileWithoutEmptyRowsWithSentimentName)
    return tweetDF1

def generateCleanDataset():
    data = pd.read_csv(data_repo + csvBalancedCleanedPolaritySubjectivityFileName, delimiter=',',low_memory=False)
    # we do not care about the exact time of each tweet, we just want the date
    data['created_at'] = pd.to_datetime(data['created_at'], errors='coerce')
    data['created_at'] = pd.to_datetime(data['created_at']).dt.date
    data['words'] = data.text.apply(lambda x: re.findall(r'\w+', x))
    data.to_csv(data_repo + csvCleanedDataset)


def generateExtract():
    # generate an extract from full cleaned_tweets_dataset
    # tweet_date_created    tweet_id    tweet_text  language    sentiment   sentiment_score
    data = pd.read_csv(data_repo + csvCleanedDataset, delimiter=',',low_memory=False)
    dfTweets = pd.DataFrame(data, columns=['created_at', 'id', 'text', 'words','sentiment', 'Negative_Score',
                                           'Positive_Score','Neutral_Score', 'Compound_Score'])
    print(dfTweets.info())
    dfTweets.to_csv(data_repo + csvExtract)


# Use regular expressions to strip each tweet of mentions, hashtags, retweet information and links
def clean_tweet_text(text):
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'#', '', text)
    text = re.sub(r'RT[\s]+', '', text)
    text = re.sub(r'https?:\/\/\S+', '', text)
    text = text.lower()
    return text

# Correct common slang and abbreviations
def clean_slang(text):
    text = re.sub(r"\babt?\b", "about", text)
    text = re.sub(r"\bcomfy\b", "comfortable", text)
    text = re.sub(r"\brll?y\b", "really", text)
    text = re.sub(r"\bso{2,}\b", "so", text)
    text = re.sub(r"\bmed\b", "medium", text)
    text = re.sub(r"\bxx?s\b", "extra small", text)
    text = re.sub(r"\bxx?l\b", "extra large", text)
    text = re.sub(r"\bfab\b", "fabulous", text)
    text = re.sub(r"\bblk\b", "black", text)
    text = re.sub(r"\bpromo\b", "promotion", text)
    text = re.sub(r"\btts\b", "true to size", text)
    text = re.sub(r"\blbs?\b", "pounds", text)
    text = re.sub(r"\brn\b", "right now", text)
    text = re.sub(r"\bwanna\b", "want to", text)
    text = re.sub(r"\besp\b", "especially", text)
    text = re.sub(r"\bgonn[ao]\b", "going to", text)
    text = re.sub(r"\btho\b", "though", text)
    text = re.sub(r"altho ", "although ", text)
    text = re.sub(r"prolly", "probably", text)
    text = re.sub(r"asap", "as soon as possible", text)
    text = re.sub(r"\bbc|b/c\b", "because", text)
    text = re.sub(r"\bavail\b", "available", text)
    text = re.sub(r"\bdiff\b", "different", text)
    text = re.sub(r"\bnxt|enxt\b", "next", text)
    text = re.sub(r" w/ ", " with ", text)
    text = re.sub(r"\bdidn ", "didn't ", text)
    text = re.sub(r" dnt ", " don't ", text)
    text = re.sub(r"\bsnd\b", "send", text)
```

```python
def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # Emoticons
                           u"\U0001F300-\U0001F5FF"  # Symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # Transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # Flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)

def cleanDataSet(df):
    print(df)
    print(df.columns)
    print(df.shape)
    print(df.info())
    # df = df.drop(columns=['Unnamed: 0'], inplace=True)
    # df = df.drop(columns=['id'], inplace=True)
    df = df.drop_duplicates('text')
    print(df.shape)
    df['text'].transform(clean_tweet_text)
    print(df.head())

    df['text'].transform(clean_slang)
    print(df.head())

    # remove emojis in the 'text' column
    df['text'].transform(remove_emoji)
    print(df.head())

    # Checking the maximum length of tweet
    length = df['text'].apply(lambda x: len(str(x).split())).max()
    print("Checking the maximum length of tweet: ", length)
    # expand contractions
    df['text'].transform(expand_contraction)

    df.to_csv("Raw_Dataset_(Cleaned).csv")
    #print(df.sample(5))
    return df
```
Fig. 8 – Code extract - balancedDatasetCreator.py

3.7 mergeDataset.py file

This Python file was used to merge data scraped from Twitter with the data extracted from 'Our World in Data' website (Our World in Data, 2022). Figure 9 is an amalgamation of multiple extracts from this Python file.

```python
data_root = "C:\\Users\\aggui\\Desktop\\Msc\\data\\"
with open("./variables.txt", "r") as f:
    variables = f.readlines()

var_list = [d.split('=')[1].split('\n')[0] for d in variables]
data_repo = var_list[0]
fileNameWorldFilteredData = "owid-covid-data-filtered.csv"
tweetsFileName = "dataset_cleaned_with_polarity_subjectivity.csv"
fileNameFilteredMergedData = "tweets-world-covid-data-filtered-cleaned-merged.csv"
fileNameFilteredMergedDroppedColsData = "tweets-world-covid-data-dropped-cols-merged.csv"
fileNameDroppedFurtherCasesColsData = "tweets-world-covid-data-dropped-further-cases-cols.csv"
csvCleanedTweetsCols = "tweets-world-covid-data-cleaned-tweets-cols.csv"
df_tweets = pd.read_csv(data_repo+tweetsFileName, delimiter=',',low_memory=False)

world_data = pd.read_csv(data_root+fileNameWorldFilteredData)
world_data['created_at'] = pd.to_datetime(world_data['created_at'], errors='coerce')
world_data['created_at'] = pd.to_datetime(world_data['created_at']).dt.date
world_data = world_data.sort_values(by='created_at', ascending=False)

# we do not care about the exact time of each tweet, we just want the date
df_tweets['created_at'] = pd.to_datetime(df_tweets['created_at'], errors='coerce')
df_tweets['created_at'] = pd.to_datetime(df_tweets['created_at']).dt.date

df_tweets = df_tweets.merge(world_data, on='created_at', how='left')

df_tweets.to_csv(data_repo+fileNameFilteredMergedData)

def deleteColumns(df):
    print(df.columns)
    print(df.shape)
    print(df.info())
    df.drop(columns=['Unnamed: 0_x'], inplace=True)
    df.drop(columns=['possibly_sensitive'], inplace=True)
    df.drop(columns=['author.created_at'], inplace=True)
    df.drop(columns=['author.pinned_tweet_id'], inplace=True)
    df.drop(columns=['author.protected'], inplace=True)
    df.drop(columns=['author.withheld.scope'], inplace=True)
    df.drop(columns=['author.withheld.copyright'], inplace=True)
    df.drop(columns=['author.withheld.country_codes'], inplace=True)
    df.drop(columns=['geo.coordinates.coordinates'], inplace=True)
    df.drop(columns=['geo.coordinates.type'], inplace=True)
    df.drop(columns=['geo.country'], inplace=True)
    df.drop(columns=['geo.country_code'], inplace=True)
    df.drop(columns=['geo.full_name'], inplace=True)
    df.drop(columns=['geo.geo.bbox'], inplace=True)
    df.drop(columns=['geo.geo.type'], inplace=True)
    df.drop(columns=['geo.id'], inplace=True)
    df.drop(columns=['geo.name'], inplace=True)
    df.drop(columns=['geo.place_id'], inplace=True)
    df.drop(columns=['geo.place_type'], inplace=True)
    df.drop(columns=['Unnamed: 73'], inplace=True)
    df.drop(columns=['Unnamed: 0_y'], inplace=True)
    df.drop(columns=['iso_code'], inplace=True)
    df.drop(columns=['location'], inplace=True)
    df.drop(columns=['new_cases_smoothed'], inplace=True)
    df.drop(columns=['new_deaths_smoothed'], inplace=True)
    df.drop(columns=['new_cases_smoothed_per_million'], inplace=True)
    df.drop(columns=['new_deaths_smoothed_per_million'], inplace=True)
```

```
    df.drop(columns=['new_people_vaccinated_smoothed'], inplace=True)
    df.drop(columns=['new_people_vaccinated_smoothed_per_hundred'], inplace=True)
    df.drop(columns=['population'], inplace=True)
    df.drop(columns=['population_density'], inplace=True)
    df.drop(columns=['median_age'], inplace=True)
    df.drop(columns=['aged_65_older'], inplace=True)
    df.drop(columns=['aged_70_older'], inplace=True)
    df.drop(columns=['gdp_per_capita'], inplace=True)
    df.drop(columns=['extreme_poverty'], inplace=True)
    df.drop(columns=['cardiovasc_death_rate'], inplace=True)
    df.drop(columns=['diabetes_prevalence'], inplace=True)
    df.drop(columns=['female_smokers'], inplace=True)
    df.drop(columns=['male_smokers'], inplace=True)
    df.drop(columns=['handwashing_facilities'], inplace=True)
    df.drop(columns=['hospital_beds_per_thousand'], inplace=True)
    df.drop(columns=['life_expectancy'], inplace=True)
    df.drop(columns=['human_development_index'], inplace=True)
    return df
```

Fig. 9 – Code extract - mergeDataset.py

3.8 tweetsLocationAnalyser.py file

This Python file was written to analyse locations attached to tweets. Figure 10 is an extract from this Python file and figure 11 a chart output while running the code. The results of this analysis were one of the factors which drove the decision to focus this research on worldwide data.

```
def tweetLocation(df):
    location_df = df['author.location'].value_counts().to_frame().reset_index().rename(
        columns={'index': 'author.location', 'author.location': 'count'})[:15]
    display(location_df.head(10))
    fig = go.Figure(go.Bar(
        x=location_df['author.location'], y=location_df['count'],
        marker={'color': location_df['count'],
                'colorscale': 'blues'},
        text=location_df['count'],
        textposition="outside",
    ))
    fig.update_layout(title_text='Top Sources ', xaxis_title="Sources", yaxis_title="Count ",
                      template="plotly_dark", title_x=0.5)
    #fig.savefig('tweet_location.png', dpi=300)
    fig.show()
    df['author.location'] = df['author.location'].fillna('NA')
    df = df['author.location'].value_counts().to_frame()
    total = df['author.location'].sum()
    print("total: ",total)
    df['percentage'] = 100 * df['author.location'] / total
    field = 'author.location'
    percent_limit = 0.5
    otherdata = df[df['percentage'] < percent_limit]
    others = otherdata['percentage'].sum()
```

```
    others = otherdata['percentage'].sum()
    maindata = df[df['percentage'] >= percent_limit]
    df = maindata
    other_label = "Others(<" + str(percent_limit) + "% each)"
    df.loc[other_label] = pd.Series({field: otherdata['author.location'].sum()})

    labels = df.index.tolist()
    datavals = df[field].tolist()
    trace = go.Pie(labels=labels, values=datavals)
    layout = go.Layout(
        title='Number of tweets per location',
        height=1000,
        width=1000
    )

    fig = go.Figure(data=[trace], layout=layout)
    #fig.savefig('tweets_per_loc.png', dpi=300)
    iplot(fig)

df = pd.read_csv(data_repo+fileName, low_memory=False)
print(df.info())
print(df.describe())
tweetLocation(df)
```

Fig. 10 – Code extract - tweetsLocationAnalyser.py

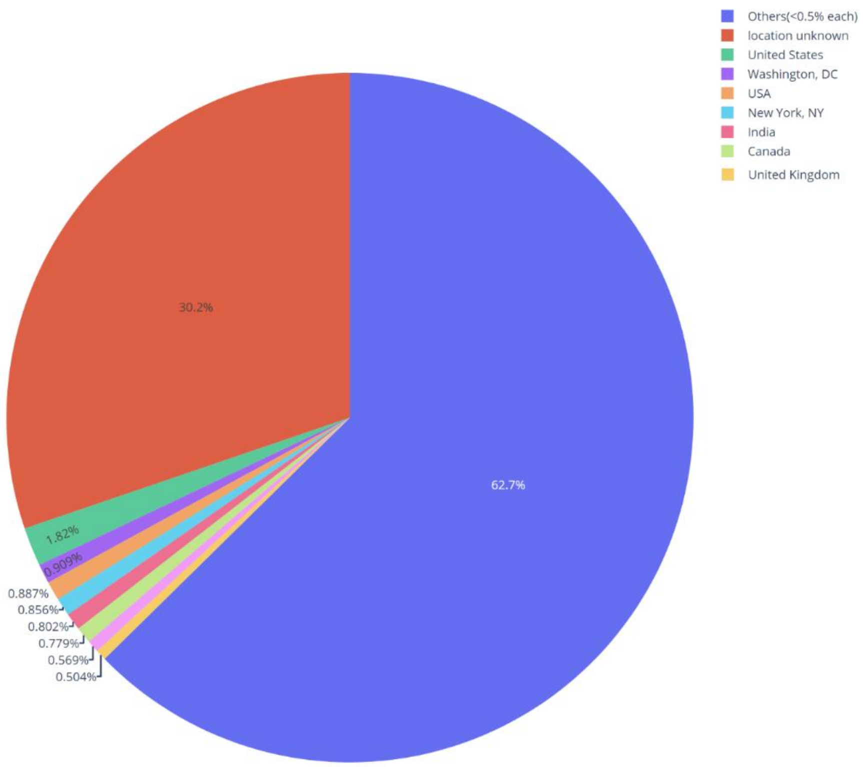


Fig. 11 – Tweets per location

3.9 covidVaccineEDAWithAllYears.py file

This Python file was written to run an exploratory data analysis on the tweets dataset. Figure 12 is an extract from this Python file and figures 13, 14, 15 and 16 are snapshots of outputs displayed while running the code. Sizes of the initial and random extract dataframes have been highlighted.

```
count_by_month = df['created_at'].groupby(df.created_at.dt.to_period("M")).agg('count')
ax = count_by_month.plot(kind='bar')
ax.set_xlabel("Months")
ax.set_ylabel("Tweet Count")
ax.set_title("Tweet Count by Month")
plt.tight_layout()
#plt.savefig("./tweet_count_by_month.png", facecolor='w')
plt.show()

count_by_hashtag = df['hashtag_count'].value_counts().sort_index()
ax = count_by_hashtag.plot(kind='bar')
ax.set_xlabel("Count")
ax.set_ylabel("Tweet Count")
ax.set_title("Tweet Count by Number of Hashtag")
plt.tight_layout()
#plt.savefig("./tweet_count_by_number_of_hashtag.png", facecolor='w')
plt.show()
```

Fig. 12 – Code extract - covidVaccineEDA.py

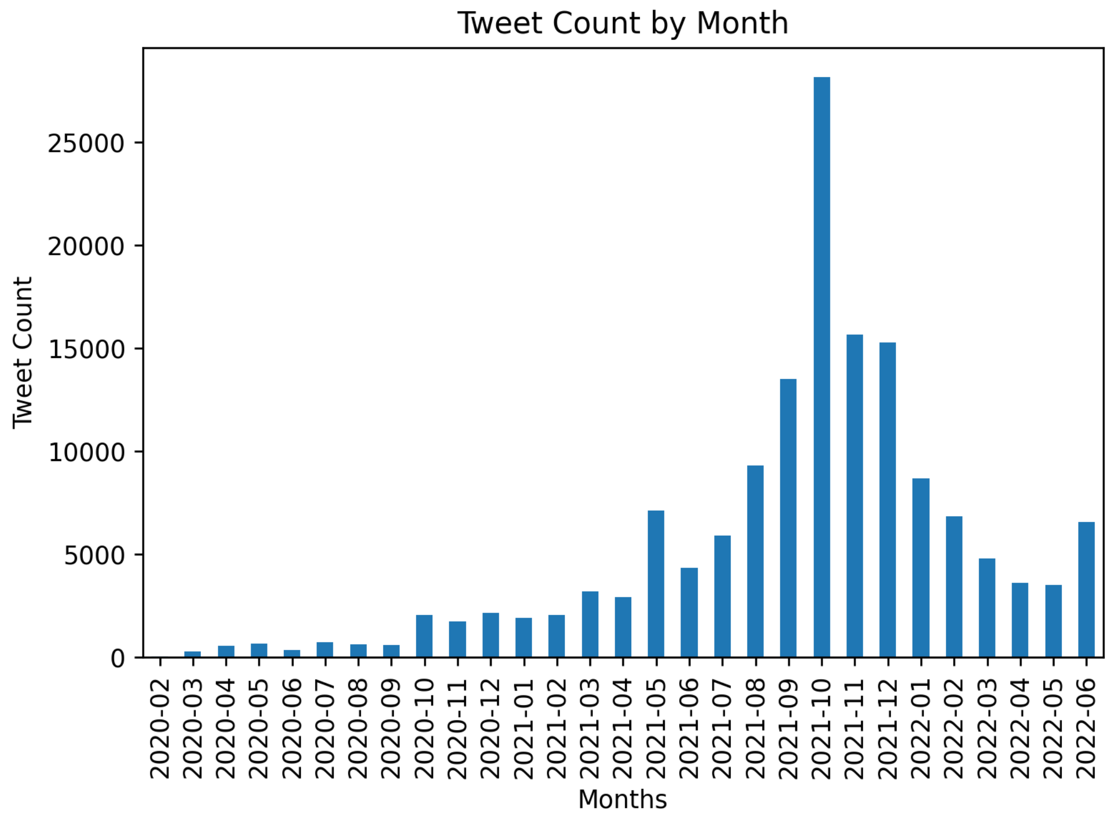


Fig. 13 – Tweets count per month

```
Number of Rows in entire DataFrame: 1019661
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153096 entries, 0 to 153095
Data columns (total 48 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Unnamed: 0                            153096 non-null  int64
 1   id                                    153096 non-null  int64
 2   conversation_id                       153096 non-null  int64
 3   author_id                             153096 non-null  int64
 4   created_at                            153096 non-null  object
 5   text                                  153096 non-null  object
 6   source                                153096 non-null  object
 7   public_metrics.like_count             153096 non-null  int64
 8   public_metrics.quote_count            153096 non-null  int64
 9   public_metrics.reply_count            153096 non-null  int64
 10  public_metrics.retweet_count          153096 non-null  int64
 11  entities.annotations                  73575 non-null   object
 12  entities.hashtags                     24667 non-null   object
 13  entities.mentions                     71076 non-null   object
 14  entities.urls                         101106 non-null  object
 15  context_annotations                   145307 non-null  object
 16  author.id                             153096 non-null  int64
 17  author.username                       153096 non-null  object
 18  author.name                           153088 non-null  object
 19  author.description                    132431 non-null  object
 20  author.entities.url.urls              63384 non-null   object
 21  author.location                       153096 non-null  object
 22  author.public_metrics.followers_count 153096 non-null  int64
```

Fig. 14 – covidVaccineEDA – size of entire tweets dataframe - output 1

```
RangeIndex: 153096 entries, 0 to 153095
Data columns (total 48 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Unnamed: 0                            153096 non-null  int64
 1   id                                    153096 non-null  int64
 2   conversation_id                       153096 non-null  int64
 3   author_id                             153096 non-null  int64
 4   created_at                            153096 non-null  object
 5   text                                  153096 non-null  object
 6   source                                153096 non-null  object
 7   public_metrics.like_count             153096 non-null  int64
 8   public_metrics.quote_count            153096 non-null  int64
 9   public_metrics.reply_count            153096 non-null  int64
```

Fig. 15 – covidVaccineEDA.py – size of tweets random extract- output 2

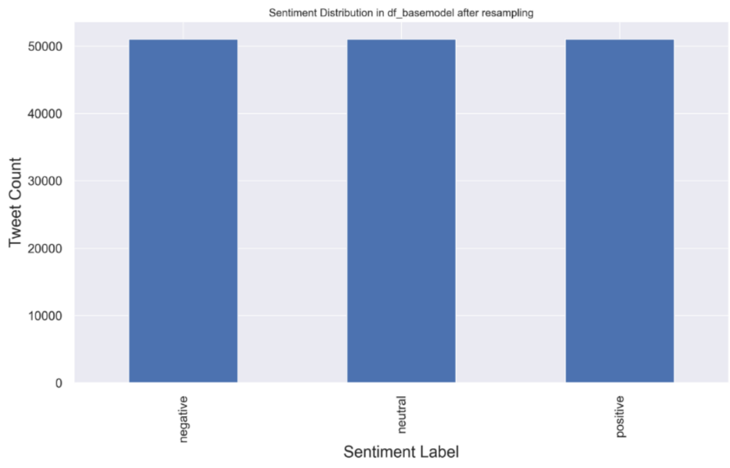
```
Name: author.verified, dtype: int64
0    132383
1     20713
Name: author.verified, dtype: int64
        Unnamed: 0                   id  ...  sentiment_score  author.created_at
0              280  1224010765977473026  ...                1         2020-02-02
1              280  1224010765977473026  ...                1         2020-02-02
2              280  1224010765977473026  ...                1         2020-02-02
3              231  1225158756897640448  ...                1         2020-02-05
4              271  1225295605301309440  ...                1         2020-02-06
...            ...                  ...  ...              ...                ...
153091       34866  1542255951755059201  ...                1         2022-06-29
153092         281  1542205871513632769  ...                0         2022-06-29
153093         701  1542042741584326656  ...                1         2022-06-29
153094       43641  1542289169619230721  ...                0         2022-06-29
153095         841  1541962223806910465  ...                0         2022-06-29

[153096 rows x 49 columns]
        Unnamed: 0                   id  ...  author.created_at  hashtag_count
0              280  1224010765977473026  ...         2020-02-02              0
1              280  1224010765977473026  ...         2020-02-02              0
2              280  1224010765977473026  ...         2020-02-02              0
3              231  1225158756897640448  ...         2020-02-05              6
4              271  1225295605301309440  ...         2020-02-06              6
...            ...                  ...  ...                ...            ...
153091       34866  1542255951755059201  ...         2022-06-29              0
153092         281  1542205871513632769  ...         2022-06-29              0
153093         701  1542042741584326656  ...         2022-06-29             12
153094       43641  1542289169619230721  ...         2022-06-29              0
```

Fig. 16 – covidVaccineEDA.py – output 3

3.10 covidKidsVaxModelsAllYears.py file

Multiple classical models have been implemented in this Python file. After resampling (fig. 17), an analysis of tweets - volume (fig. 18), objectivity/subjectivity (fig. 19 & 20) and N-gram analysis (fig. 21, 22, 23, 24) over the period was carried out.




```
neutral     514248
negative    514248
positive    514248
```

Fig. 17 – Sentiment distribution after resampling

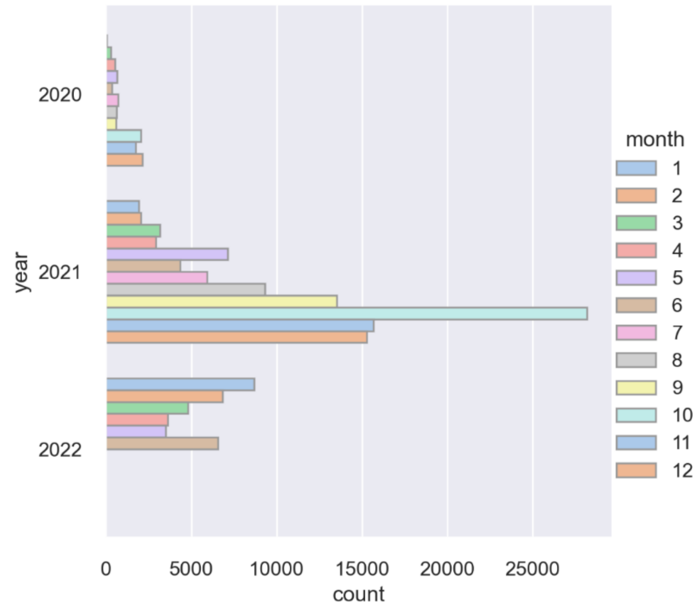


Fig. 18 – Tweets volume

sentiment			negative	neutral	positive
year	month	sentiment2			
2020	2	objective	5	15	6
		subjective	4	0	11
	3	objective	36	75	50
		subjective	49	10	55
	4	objective	110	163	85
		subjective	70	17	97
	5	objective	111	181	124
		subjective	95	40	102
	6	objective	56	118	64
		subjective	50	11	58
	7	objective	163	152	126
		subjective	128	20	136
	8	objective	128	107	118
		subjective	125	22	110
	9	objective	120	177	106
		subjective	66	18	98
	10	objective	310	833	351
		subjective	275	52	223
	11	objective	189	974	188
		subjective	173	32	181
	12	objective	338	706	368
		subjective	352	57	330

2021	1	objective	346	566	292
		subjective	369	65	277
	2	objective	344	629	464
		subjective	257	64	287
	3	objective	478	983	703
		subjective	514	87	413
	4	objective	497	859	525
		subjective	477	74	490
	5	objective	1123	2469	1480
		subjective	810	199	1036
	6	objective	714	1307	769
		subjective	658	171	735
	7	objective	1151	1346	1034
		subjective	1161	245	975
	8	objective	1665	2243	1653
		subjective	1830	376	1544
	9	objective	1910	3151	2093
		subjective	3839	420	2095
	10	objective	4315	9517	4712
		subjective	4351	885	4398
	11	objective	2322	5209	2769
		subjective	2442	532	2394
	12	objective	3139	4181	2666
		subjective	2465	524	2300
2022	1	objective	1485	2259	1488
		subjective	1553	296	1601
	2	objective	1240	2073	1269
		subjective	1074	230	942
	3	objective	754	1310	913
		subjective	747	154	922
	4	objective	648	1172	721
		subjective	436	104	545
	5	objective	529	1013	637
		subjective	503	160	666
	6	objective	978	2099	1266
		subjective	955	280	971

Fig. 19 – Tweets objectivity/subjectivity output

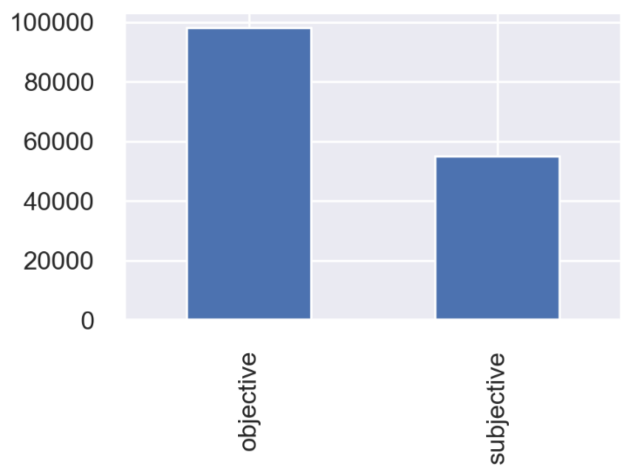


Fig. 20 – Tweets objectivity/subjectivity

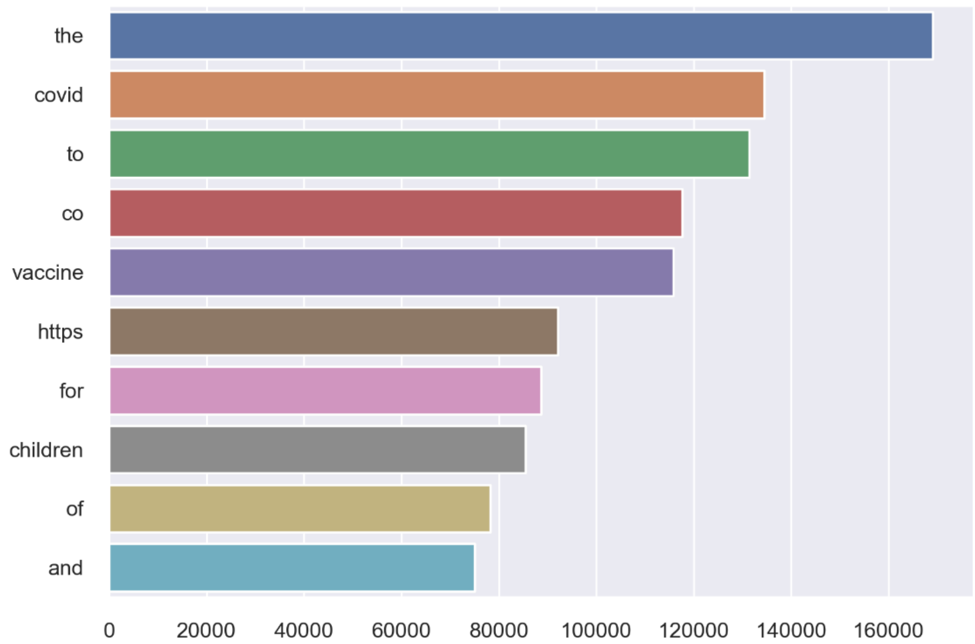


Fig. 21 – N-gram analysis -1

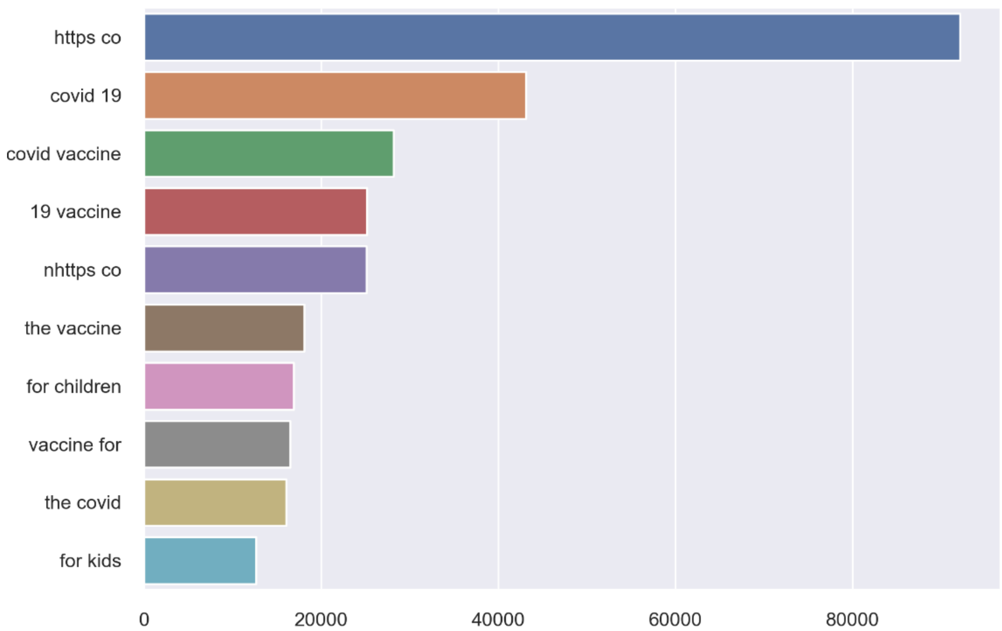


Fig. 22 – N-gram analysis -2

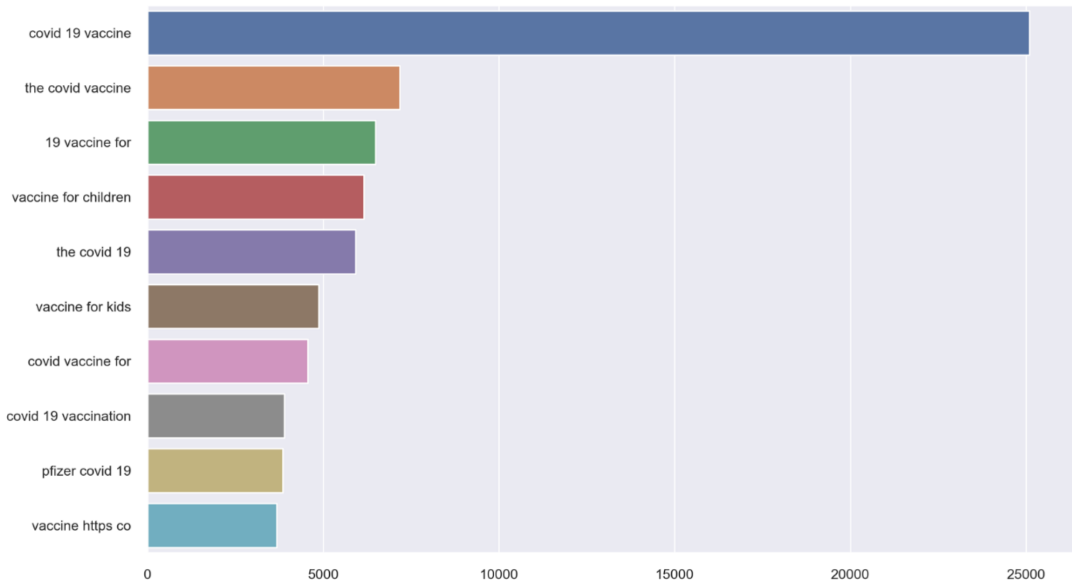


Fig. 23 – N-gram analysis -3

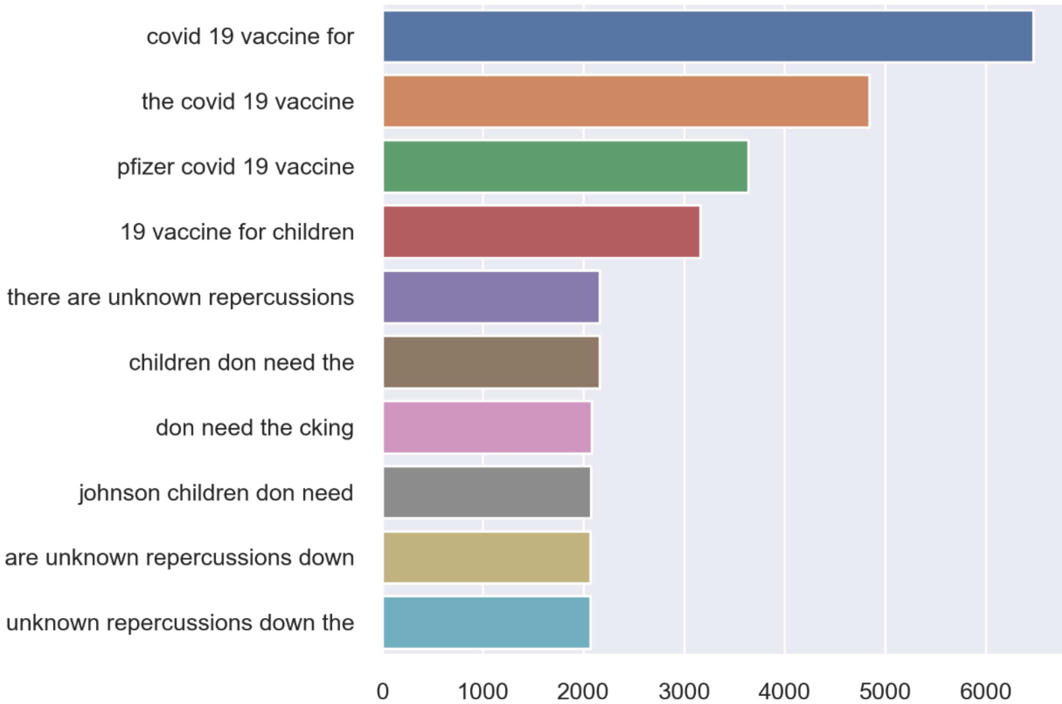


Fig. 24 – N-gram analysis -4

The number of tweets per year in the sample can be seen in fig. 25 and the average number of words in tweets in fig. 26.

2021	109383
2022	33993
2020	9720

Fig. 25 – Number of tweets per year

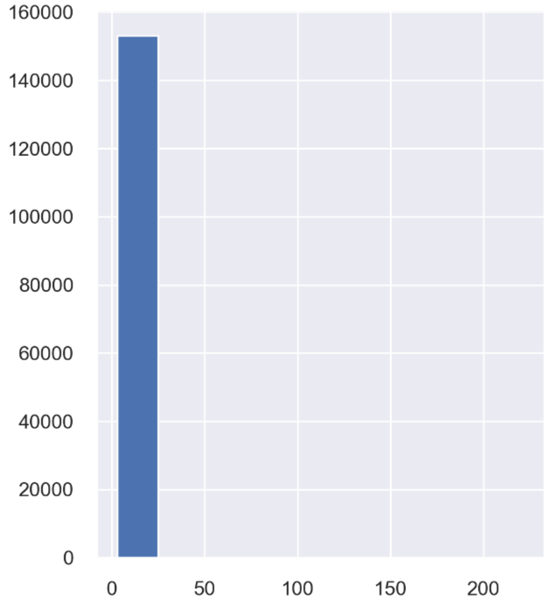


Fig. 26 - Number of words in tweets

Figure 27 is an amalgamation of multiple snapshots of code in this Python file where tweets from official news outlets are analysed and classical models are implemented (logistic regression, random forest, extreme gradient boost, K-Neighbours, SVM, Decision Tree).

```
df_nytimestweets = df_basemodel[df_basemodel['author.username'] == 'nytimes']
print(df_nytimestweets['author.username'].value_counts())
df_cnnbreaktweets = df_basemodel[df_basemodel['author.username'] == 'cnnbrk']
print("CNN breaking", df_cnnbreaktweets['author.username'].value_counts())
df_cnntweets = df_basemodel[df_basemodel['author.username'] == 'cnn']
print("CNN breaking", df_cnntweets['author.username'].value_counts())
df_usnewstweets = df_basemodel[df_basemodel['author.username'] == 'usnews']
print("US News ", df_usnewstweets['author.username'].value_counts())
df_timetweets = df_basemodel[df_basemodel['author.username'] == 'time']
print("Time ",df_timetweets['author.username'].value_counts())
df_breakingnewstweets = df_basemodel[df_basemodel['author.username'] == 'breakingnews']
print("breakingnews ",df_breakingnewstweets['author.username'].value_counts())
df_bbcbreakingtweets = df_basemodel[df_basemodel['author.username'] == 'bbcbreaking']
print("bbcbreaking: ", df_bbcbreakingtweets['author.username'].value_counts())
df_whitehousetweets = df_basemodel[df_basemodel['author.username'] == 'whitehouse']
print("The White House: ", df_whitehousetweets['author.username'].value_counts())
df_newsweektweets = df_basemodel[df_basemodel['author.username'] == 'newsweek']
print("Newsweek: ", df_newsweektweets['author.username'].value_counts())
df_huffingtonposttweets = df_basemodel[df_basemodel['author.username'] == 'huffingtonpost']
print("huffingtonpost: ", df_huffingtonposttweets['author.username'].value_counts())
df_newscientisttweets = df_basemodel[df_basemodel['author.username'] == 'newscientist']
print("New Scientist: ", df_newscientisttweets['author.username'].value_counts())
df_theeconomisttweets = df_basemodel[df_basemodel['author.username'] == 'theeconomist']
print("The Economist: ", df_theeconomisttweets['author.username'].value_counts())
df_reuterstweets = df_basemodel[df_basemodel['author.username'] == 'reuters']
print("Reuters: ", df_reuterstweets['author.username'].value_counts())
df_washingtonposttweets = df_basemodel[df_basemodel['author.username'] == 'washingtonpost']
print("Washington Post: ", df_washingtonposttweets['author.username'].value_counts())
df_politicotweets = df_basemodel[df_basemodel['author.username'] == 'politico']
print("Politico: ", df_politicotweets['author.username'].value_counts())
```

```python
def _get_top_ngram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(n, n)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx])
                  for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:10]

for word, count in most[:40]:
    if (word not in stopwords):
        x.append(word)
        y.append(count)
    top_n_bigrams = _get_top_ngram(tweet, n)[:10]
    x, y = map(list, zip(*top_n_bigrams))
    sns.barplot(x=y, y=x)
```

```python
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized_output = []

for listy in processed_tweet:
    lemmed = ' '.join([lemmatizer.lemmatize(w) for w in listy])
    lemmatized_output.append(lemmed)

Data = {'text':lemmatized_output, 'sentiment_score':target}
tweet_lemmantized = pd.DataFrame(Data)
train_tweets = df_basemodel.sample(frac = 0.75)
test_tweets = df_basemodel.drop(train_tweets.index)
print(train_tweets.shape)
print(test_tweets.shape)

from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits = 1, test_size = .25, random_state = 3)
sss.get_n_splits(tweet_lemmantized.text, tweet_lemmantized.sentiment_score)
for train_ind, test_ind in sss.split(tweet_lemmantized.text, tweet_lemmantized.sentiment_score):
    pass
print(f'Train_ind shape: {train_ind.shape}\nTest_ind shape: {test_ind.shape}')

from sklearn.feature_extraction.text import TfidfVectorizer #
vectorizer = TfidfVectorizer(analyzer = 'word', max_features = 50)
X_train = vectorizer.fit_transform(tweet_lemmantized.text[train_ind].reset_index(drop = True))
y_train = tweet_lemmantized.sentiment_score[train_ind].reset_index(drop = True)
X_test = vectorizer.transform(tweet_lemmantized.text[test_ind].reset_index(drop = True))
y_test = tweet_lemmantized.sentiment_score[test_ind].reset_index(drop = True)

model_perf = {} # dictionary for storing performance
```

```
# data modeling
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
# grid searching for optimal c values
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classification_report
params = {'C': [.001, .005, .01, .05, .1, .5, 1, 5]}
model = LogisticRegression(max_iter = 1000, random_state = 4)
clf = GridSearchCV(model, param_grid = params, n_jobs = -1)
clf.fit(X_train, y_train)
print(clf.cv_results_)
print(X_train.todense())

m1 = 'Logistic Regression'
lr = LogisticRegression()
model = lr.fit(X_train, y_train)
lr_predict = lr.predict(X_test)
lr_predict1 = lr.predict(X_train)
lr_conf_matrix = confusion_matrix(y_test, lr_predict)
lr_conf_matrix1 = confusion_matrix(y_train, lr_predict1)
lr_acc_score = accuracy_score(y_test, lr_predict)
lr_acc_score1 = accuracy_score(y_train, lr_predict1)
print("Confusion Matrix")
print(lr_conf_matrix)
print(lr_conf_matrix1)
print("\n")
print("Accuracy of Logistic Regression Test:",lr_acc_score*100,'\n')
print("Accuracy of Logistic Regression Train:",lr_acc_score1*100,'\n')
print(classification_report(y_test,lr_predict))
print(classification_report(y_train,lr_predict1))

m1 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=50)
rf.fit(X_train,y_train)
rf_predicted = rf.predict(X_test)
rf_predicted1 = rf.predict(X_train)
rf_conf_matrix = confusion_matrix(y_test, rf_predicted)
rf_conf_matrix1 = confusion_matrix(y_train, rf_predicted1)
rf_acc_score = accuracy_score(y_test, rf_predicted)
rf_acc_score1 = accuracy_score(y_train, rf_predicted1)
print("Confusion Matrix")
print(rf_conf_matrix)
print(rf_conf_matrix1)
print("\n")
print("Accuracy of Random Forest Train:",rf_acc_score*100,'\n')
print("Accuracy of Random Forest Test:",rf_acc_score1*100,'\n')
print(classification_report(y_test,rf_predicted))
print(classification_report(y_train,rf_predicted1))

m2 = 'Extreme Gradient Boost'
xgb = XGBClassifier(learning_rate=0.01, n_estimators=25, max_depth=15,gamma=0.6, subsample=0.52,colsample_bytree=0.6,seed=2
                    reg_lambda=2, booster='gbtree', colsample_bylevel=0.6, colsample_bynode=0.5)
xgb.fit(X_train, y_train)
xgb_predicted = xgb.predict(X_test)
```

```
xgb_predicted1 = xgb.predict(X_train)
xgb_conf_matrix = confusion_matrix(y_test, xgb_predicted)
xgb_conf_matrix1 = confusion_matrix(y_train, xgb_predicted1)
xgb_acc_score = accuracy_score(y_test, xgb_predicted)
xgb_acc_score1 = accuracy_score(y_train, xgb_predicted1)
print("Confusion Matrix")
print(xgb_conf_matrix)
print(xgb_conf_matrix1)
print("\n")
print("Accuracy of Extreme Gradient Boost Test:",xgb_acc_score*100,'\n')
print("Accuracy of Extreme Gradient Boost Train:",xgb_acc_score1*100,'\n')
print(classification_report(y_test,xgb_predicted))
print(classification_report(y_train,xgb_predicted1))

m3 = 'K-NeighborsClassifier'
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
knn_predicted = knn.predict(X_test)
knn_predicted1 = knn.predict(X_train)
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
knn_conf_matrix1 = confusion_matrix(y_train, knn_predicted1)
knn_acc_score = accuracy_score(y_test, knn_predicted)
knn_acc_score1 = accuracy_score(y_train, knn_predicted1)
print("Confusion Matrix")
print(knn_conf_matrix)
print(knn_conf_matrix1)
print("\n")
print("Accuracy of K-NeighborsClassifier Test:",knn_acc_score*100,'\n')
print("Accuracy of K-NeighborsClassifier Train:",knn_acc_score1*100,'\n')
print(classification_report(y_test,knn_predicted))
print(classification_report(y_train,knn_predicted1))
```

```
m4 = 'DecisionTreeClassifier'
dt = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 50)
dt.fit(X_train, y_train)
dt_predicted = dt.predict(X_test)
dt_predicted1 = dt.predict(X_train)
dt_conf_matrix = confusion_matrix(y_test, dt_predicted)
dt_conf_matrix1 = confusion_matrix(y_train, dt_predicted1)
dt_acc_score = accuracy_score(y_test, dt_predicted)
dt_acc_score1 = accuracy_score(y_train, dt_predicted1)
print("Confusion Matrix")
print(dt_conf_matrix)
print(dt_conf_matrix1)
print("\n")
print("Accuracy of DecisionTreeClassifier Test:",dt_acc_score*100,'\n')
print("Accuracy of DecisionTreeClassifier Train:",dt_acc_score1*100,'\n')
print(classification_report(y_test,dt_predicted))
print(classification_report(y_train,dt_predicted1))

m5 = 'Support Vector Classifier'
svc =  SVC(kernel='rbf', C=2)
svc.fit(X_train, y_train)
svc_predicted = svc.predict(X_test)
svc_predicted1 = svc.predict(X_train)
svc_conf_matrix = confusion_matrix(y_test, svc_predicted)
svc_conf_matrix1 = confusion_matrix(y_train, svc_predicted1)
svc_acc_score = accuracy_score(y_test, svc_predicted)
svc_acc_score1 = accuracy_score(y_train, svc_predicted1)
print("Confusion Matrix")
print(svc_conf_matrix)
print(svc_conf_matrix1)
print("\n")
```

```
print("\n")
print("Accuracy of Support Vector Classifier Test:",svc_acc_score*100,'\n')
print("Accuracy of Support Vector Classifier Train:",svc_acc_score1*100,'\n')
print(classification_report(y_test,svc_predicted))
print(classification_report(y_train,svc_predicted1))

model_ev = pd.DataFrame({'Model': ['Logistic Regression','Random Forest','Extreme Gradient Boost',
                    'K-Nearest Neighbour','Decision Tree','Support Vector Machine'], 'Test_Accuracy': [lr_acc_score*100,
                   rf_acc_score*100,xgb_acc_score*100,knn_acc_score*100,dt_acc_score*100,svc_acc_score*100], 'Train_Accurac
                   rf_acc_score1*100,xgb_acc_score1*100,knn_acc_score1*100,dt_acc_score1*100,svc_acc_score1*100]})
print(model_ev)

model_ev.plot.bar()
plt.show()
```

```python
def modelPreparation(df):
    text_counts = countVector(df)
    le = preprocessing.LabelEncoder()
    df['sentiment'] = le.fit_transform(df['sentiment'])
    # display(df)
    x_train, x_test, y_train, y_test = train_test_split(text_counts, df['sentiment'], test_size=0.20, random_state=0)
    # Naive Bayes Classification
    print("In modelPreparation method, Naive Bayes:")
    cnb = ComplementNB()
    cnb.fit(x_train, y_train)
    print("Train accuracy {:.2f}%".format(cnb.score(x_train, y_train) * 100))
    print("Test accuracy {:.2f}%".format(cnb.score(x_test, y_test) * 100))
    y_pred = cnb.predict(x_test)
    cf_matrix = confusionMatrix(y_test,y_pred)
    print(cf_matrix)
    ## Display the visualization of the Confusion Matrix.
    plt.show()
    print("Accuracy Score:")
    print(accuracy_score(y_test, y_pred))
    print("Confusion Matrix:")
    plot_confusion_matrix(cnb, x_test, y_test)
    #ConfusionMatrixDisplay.from_predictions(x_test, y_test)
    plt.show()
    print("Classification Report:")
    #classes = ["Positive", "Negative"]
    classes = ["Negative", "Neutral","Positive"]
    visualizer = ClassificationReport(cnb, classes=classes, support=True)
    visualizer.fit(x_train, y_train)
    visualizer.score(x_test, y_test)
    visualizer.show()
    print("\n")
    # Random Forest Classification
    print("Random Forest Classification:")
    clf = RandomForestClassifier(max_depth=2, random_state=0)
    clf.fit(x_train, y_train)
    print("Train accuracy {:.2f}%".format(clf.score(x_train, y_train) * 100))
    print("Test accuracy {:.2f}%".format(clf.score(x_test, y_test) * 100))
    y_pred = clf.predict(x_test)
    print("Accuracy Score:")
    print(accuracy_score(y_test, y_pred))
    print("Confusion Matrix:")
    plot_confusion_matrix(clf, x_test, y_test)
    plt.show()
    print("Classification Report:")
    classes = ["Negative", "Neutral","Positive"]
    visualizer = ClassificationReport(clf, classes=classes, support=True)
    visualizer.fit(x_train, y_train)
    visualizer.score(x_test, y_test)
    visualizer.show()
    print("\n")

    # SVM
    print("SVM:")
    svmclf = svm.SVC()
    svmclf.fit(x_train, y_train)
    print("Train accuracy {:.2f}%".format(svmclf.score(x_train, y_train) * 100))
    print("Test accuracy {:.2f}%".format(svmclf.score(x_test, y_test) * 100))
    y_pred = svmclf.predict(x_test)
```

```
confusionMatrix(y_test,y_pred)
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))
print("Confusion Matrix:")
plot_confusion_matrix(svmclf, x_test, y_test)
plt.show()
print("Classification Report:")
classes = ["Negative", "Neutral","Positive"]
visualizer = ClassificationReport(svmclf, classes=classes, support=True)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()

# Training Logistics Regression model
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(solver='lbfgs',max_iter=400) #100
LR_model.fit(x_train, y_train)
y_predict_lr = LR_model.predict(x_test)
print("accuracy for LogisticRegression:")
print(accuracy_score(y_test, y_predict_lr))
# Use score method to get accuracy of model
score = LR_model.score(x_test, y_test)
print(score)
cm = metrics.confusion_matrix(y_test, y_predict_lr)
print(cm)
plt.figure(figsize=(9, 9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size=15);
plt.show()
#DecisionTreeRegressor
print("Decision Tree")
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(x_train, y_train)
score = regressor.score(x_train, y_train)
print("R-squared:", score)
# DecisionTreeRegressor(random_state=0)
y_pred = regressor.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print("MSE: ", mse)
print("RMSE: ", mse ** (1 / 2.0))

x_ax = range(len(y_test))
plt.plot(x_ax, y_test, linewidth=1, label="original")
plt.plot(x_ax, y_pred, linewidth=1.1, label="predicted")
plt.title("y-test and y-predicted data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```
Fig. 27 – Code snapshots – CovidKidsVaxModelsAllYears.py

Outputs from models implemented in this file can be seen in the next screenshots (fig. 28, 29, 30, 31, 32, 33). Figure 34 is an overview and comparison of the models results.

```
[[7103 3274 2381]
 [2378 8705 1675]
 [4269 4572 3917]]
[[21494  9547  7233]
 [ 7049 26343  4882]
 [13034 13729 11511]]
 Accuracy of Logistic Regression: 51.5362909546951

 Accuracy of Logistic Regression: 51.68695894514988

              precision    recall  f1-score   support

           0       0.52      0.56      0.54     12758
           1       0.53      0.68      0.59     12758
           2       0.49      0.31      0.38     12758

    accuracy                           0.52     38274
   macro avg       0.51      0.52      0.50     38274
weighted avg       0.51      0.52      0.50     38274

              precision    recall  f1-score   support

           0       0.52      0.56      0.54     38274
           1       0.53      0.69      0.60     38274
           2       0.49      0.30      0.37     38274

    accuracy                           0.52    114822
   macro avg       0.51      0.52      0.50    114822
weighted avg       0.51      0.52      0.50    114822
```

Fig. 28 – Logistic Regression results & Confusion Matrix

```
[[ 9772  1619  1367]
 [ 1278 10022  1458]
 [ 2914  2893  6951]]
[[33339  3925  1010]
 [ 2363 34670  1241]
 [ 2996  5144 30134]]
```

```
Accuracy of Random Forest Test: 85.47403807632683

              precision    recall  f1-score   support

           0       0.70      0.77      0.73     12758
           1       0.69      0.79      0.73     12758
           2       0.71      0.54      0.62     12758

    accuracy                           0.70     38274
   macro avg       0.70      0.70      0.69     38274
weighted avg       0.70      0.70      0.69     38274

              precision    recall  f1-score   support

           0       0.86      0.87      0.87     38274
           1       0.79      0.91      0.85     38274
           2       0.93      0.79      0.85     38274

    accuracy                           0.85    114822
   macro avg       0.86      0.85      0.85    114822
weighted avg       0.86      0.85      0.85    114822
```
Fig. 29 – Random Forest results & Confusion Matrix

```
[[8158 2920 1680]
 [2233 9052 1473]
 [4196 3930 4632]]
[[25872  8475  3927]
 [ 6223 28576  3475]
 [11137 11067 16070]]
```

```
 Accuracy of Extreme Gradient Boost Test: 57.067460939541206

 Accuracy of Extreme Gradient Boost Train: 61.41505983173956

               precision    recall  f1-score   support

           0       0.56      0.64      0.60     12758
           1       0.57      0.71      0.63     12758
           2       0.59      0.36      0.45     12758

    accuracy                           0.57     38274
   macro avg       0.57      0.57      0.56     38274
weighted avg       0.57      0.57      0.56     38274
```


```
 Accuracy of Extreme Gradient Boost Test: 57.067460939541206

 Accuracy of Extreme Gradient Boost Train: 61.41505983173956

               precision    recall  f1-score   support

           0       0.56      0.64      0.60     12758
           1       0.57      0.71      0.63     12758
           2       0.59      0.36      0.45     12758

    accuracy                           0.57     38274
   macro avg       0.57      0.57      0.56     38274
weighted avg       0.57      0.57      0.56     38274
```


```
               precision    recall  f1-score   support

           0       0.60      0.68      0.63     38274
           1       0.59      0.75      0.66     38274
           2       0.68      0.42      0.52     38274

    accuracy                           0.61    114822
   macro avg       0.63      0.61      0.61    114822
weighted avg       0.63      0.61      0.61    114822
```

Fig. 30 – Extreme Gradient Boost results & Confusion Matrix

```
[[8175 2567 2016]
 [2403 8737 1618]
 [4231 3957 4570]]
[[27290  6337  4647]
 [ 6222 28369  3683]
 [10860 10560 16854]]
 Accuracy of K-NeighborsClassifier Test: 56.12687464074829

 Accuracy of K-NeighborsClassifier Train: 63.15253174478759

              precision    recall  f1-score   support

           0       0.55      0.64      0.59     12758
           1       0.57      0.68      0.62     12758
           2       0.56      0.36      0.44     12758

    accuracy                           0.56     38274
   macro avg       0.56      0.56      0.55     38274
weighted avg       0.56      0.56      0.55     38274

              precision    recall  f1-score   support

           0       0.62      0.71      0.66     38274
           1       0.63      0.74      0.68     38274
           2       0.67      0.44      0.53     38274

    accuracy                           0.63    114822
   macro avg       0.64      0.63      0.62    114822
weighted avg       0.64      0.63      0.62    114822
```
Fig. 31 – K-Neighbours results & Confusion Matrix

```
[[9945 1681 1132]
 [1709 9880 1169]
 [3570 3257 5931]]
[[33419  3813  1042]
 [ 3596 33685   993]
 [ 5414  5516 27344]]
 Accuracy of DecisionTreeClassifier Test: 67.29372419919528

 Accuracy of DecisionTreeClassifier Train: 82.2560136559196

              precision    recall  f1-score   support

           0       0.65      0.78      0.71     12758
           1       0.67      0.77      0.72     12758
           2       0.72      0.46      0.57     12758

    accuracy                           0.67     38274
   macro avg       0.68      0.67      0.66     38274
weighted avg       0.68      0.67      0.66     38274

              precision    recall  f1-score   support

           0       0.79      0.87      0.83     38274
           1       0.78      0.88      0.83     38274
           2       0.93      0.71      0.81     38274

    accuracy                           0.82    114822
   macro avg       0.83      0.82      0.82    114822
weighted avg       0.83      0.82      0.82    114822
```

Fig. 32 – Decision Tree results & Confusion Matrix

```
[[8288 2430 2040]
 [2236 8861 1661]
 [3973 3511 5274]]
[[27205  6613  4456]
 [ 5691 28618  3965]
 [ 9879  9365 19030]]
```

```
Accuracy of Support Vector Classifier Test: 58.585462716204205

Accuracy of Support Vector Classifier Train: 65.19046872550557

              precision    recall  f1-score   support

           0       0.57      0.65      0.61     12758
           1       0.60      0.69      0.64     12758
           2       0.59      0.41      0.49     12758

    accuracy                           0.59     38274
   macro avg       0.59      0.59      0.58     38274
weighted avg       0.59      0.59      0.58     38274

              precision    recall  f1-score   support

           0       0.64      0.71      0.67     38274
           1       0.64      0.75      0.69     38274
           2       0.69      0.50      0.58     38274

    accuracy                           0.65    114822
   macro avg       0.66      0.65      0.65    114822
weighted avg       0.66      0.65      0.65    114822
```

Fig. 33 – SVM results & Confusion Matrix

```
                     Model  Test_Accuracy  Train_Accuracy1
0      Logistic Regression      51.536291        51.686959
1            Random Forest      69.877724        85.474038
2  Extreme Gradient Boost      57.067461        61.415060
3      K-Nearest Neighbour      56.126875        63.152532
4            Decision Tree      67.293724        82.256014
```

Fig. 34 – Overall models results & comparison

3.11 TweetsSentimentAnalysisAllYears.py file

A sentiment analysis was implemented in this file. Polarity, subjectivity and objectivity are further analysed with multiple graphs produced. Multiple word clouds are generated as well (Dua, 2021). Furthermore, sentiment attached to specific vaccines has been explored as well.

Figure 35 is an amalgamation of some code extract from this Python file.

```
def vader_scores(feedbacktext, category):
    return vader.polarity_scores(feedbacktext).get(category)

def launchSentimentAnalysis(df):
    print(df)
    print(df.columns)
    print(df.shape)
    print(df.info())
    # We only care about the date, not the time
    df['created_at'] = pd.to_datetime(df['created_at']).dt.date
    # Which device are people tweeting about the vaccine from?
    df['source'].value_counts().head(n=5).plot.bar()
    plt.title("The 5 most common sources")
    plt.show()
    df['author.verified'].value_counts().head(n=10).plot.bar()
    plt.title("Verified authors")
    plt.show()
    print(df[df['author.verified'] == True].head())
    # What are the top 10 most retweeted tweets
    pd.set_option('display.max_colwidth', 400)
    print(df.sort_values(by='public_metrics.retweet_count', ascending=False)[
              ['text', 'created_at', 'author.name', 'author.location', 'entities.hashtags', 'public_metrics.like_count',
               'public_metrics.retweet_count']].head(n=10))
    print(df.sort_values(by=['created_at', 'public_metrics.like_count'], ascending=[True, False])[['text', 'created_at', 'author.name', 'author.l

    fig = plt.figure(figsize=(10, 6))
    df['polarity'].hist()
    plt.xlabel('Polarity Score', fontsize=18)
    fig = plt.figure(figsize=(10, 6))
    df['polarity'].hist()
    plt.xlabel('Polarity Score', fontsize=18)
    plt.ylabel('Frequency', fontsize=18)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    # fig.savefig("./figures/polarity_hist.png")
    plt.title("Polarity")
    plt.show()

    fig = plt.figure(figsize=(10, 6))
    df['subjectivity'].hist()
    plt.xlabel('Subjectivity Score', fontsize=18)
    plt.ylabel('Frequency', fontsize=18)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    # fig.savefig("./figures/subjectivity_hist.png")
    plt.title("Subjectivity")
    plt.show()

    # Inspect the most negatively charged tweets
    print("Most negatively charged tweets")
    print(
        df.sort_values(by='polarity', ascending=True)[['text', 'polarity', 'subjectivity']].reset_index(drop=True).head(
            n=10))
    print("Most positively charged tweets")
    # inspect the most positively charged tweets
    print(df.sort_values(by='polarity', ascending=False)[['text', 'polarity', 'subjectivity']].reset_index(
        drop=True).head(n=10))
```

```
print("Most positively charged tweets")
# inspect the most positively charged tweets
print(df.sort_values(by='polarity', ascending=False)[['text', 'polarity', 'subjectivity']].reset_index(
    drop=True).head(n=10))
print("Most subjective tweets")
# inspect the most subjective tweets (NOTE: subjectivity scale ranges from 0 to 1)
print(df.sort_values(by='subjectivity', ascending=True)[['text', 'polarity', 'subjectivity']].reset_index(
    drop=True).head(n=10))
print("Most objective tweets")
# inspect the most objective tweets
print(df.sort_values(by='subjectivity', ascending=False)[['text', 'polarity', 'subjectivity']].reset_index(
    drop=True).head(n=10))

# Inspect how many tweets there were with respect to time
timeline = df.groupby(['created_at']).count().reset_index()
timeline['count'] = timeline['text']
timeline = timeline[['created_at', 'count']]
fig = px.bar(timeline, x='created_at', y='count', labels={'created_at': 'Date', 'count': 'Tweet Count'})
fig.show()
# fig.write_image("./figures/tweet_freq_over_time.png")

# #Convert data to 3 classes (negative, neutral, and positive) to visualize it
criteria = [df['polarity'].between(-1, -0.01), df['polarity'].between(-0.01, 0.01), df['polarity'].between(0.01, 1)]
values = ['negative', 'neutral', 'positive']
df['sentiment'] = np.select(criteria, values, 0)

 # Plot sentiment counts
 fig = plt.figure(figsize=(10, 6))
 df['sentiment'].value_counts().sort_index().plot.bar()
 plt.xlabel('Sentiment Label', fontsize=18)
 plt.ylabel('Tweet Count', fontsize=18)
 plt.xticks(fontsize=14)
 plt.yticks(fontsize=14)
 plt.title("Sentiment Distribution")
 plt.show()
 plt.tight_layout()
 # fig.savefig("./figures/sentiment_value_counts", bbox_inches='tight');

 # Plot sentiment counts
 fig = plt.figure(figsize=(10, 6))
 df['Sentiment'].value_counts().sort_index().plot.bar()
 plt.xlabel('Sentiment Label', fontsize=18)
 plt.ylabel('Tweet Count', fontsize=18)
 plt.xticks(fontsize=14)
 plt.yticks(fontsize=14)
 plt.title("Sentiment Distribution (positive/negative)")
 plt.show()
 plt.tight_layout()

 print("funnel")
 print(df.info())
 print(df.columns)
 print(df.shape)
```

```python
# Plot tweets over time, color-coded by average polarity score
fig = px.bar(timeline, x='created_at', y='count', color='polarity', title="Tweets on vaccine by average polarity score")
fig.show()

# Plot tweets over time, color-coded by average subjectivity score
fig = px.bar(timeline, x='created_at', y='count', color='subjectivity', title="Tweets on vaccine by average subjectivity score")
fig.show()

pfizy_df, pfizy_timeline = filter_by_vaccy(df, ['pfizer', 'biontech'])
print("Pfizer vaccine")
print(pfizy_df.shape)
fig = px.bar(pfizy_timeline, x='created_at', y='count', color='polarity',title="Tweets on Pfizer vaccine by average polarity score")
fig.show()

moderna_df, moderna_timeline = filter_by_vaccy(df, ['moderna'])
print("Moderna vaccine")
print(moderna_df.shape)
fig = px.bar(moderna_timeline, x='created_at', y='count', color='polarity',title="Tweets on moderna vaccine by average polarity score")
fig.show()

astra_df, astra_timeline = filter_by_vaccy(df, ['astrazeneca'])
print("AstraZeneca vaccine")
print(astra_df.sort_values(by='polarity', ascending=True).reset_index(drop=True).head(n=20))

covaxin_df, covaxin_timeline = filter_by_vaccy(df, ['covaxin'])
print(covaxin_df.sort_values(by='polarity', ascending=True).reset_index(drop=True).head(n=20))
 # Convert string to a list of words
 wordcloud_df = df
 wordcloud_df['words'] = wordcloud_df.text.apply(lambda x: re.findall(r'\w+', x))
 get_smart_clouds(wordcloud_df).savefig("sentiment_wordclouds.png", bbox_inches="tight")

 if (len(pfizy_df) > 0):
     wordcloud_df = pfizy_df
     wordcloud_df['words'] = wordcloud_df.text.apply(lambda x: re.findall(r'\w+', x))
     get_smart_clouds(wordcloud_df).savefig("pfizy_sentiment_wordclouds.png", bbox_inches="tight")
 if (len(moderna_df) > 0):
     wordcloud_df = moderna_df
     wordcloud_df['words'] = wordcloud_df.text.apply(lambda x: re.findall(r'\w+', x))
     get_smart_clouds(wordcloud_df).savefig("moderna_sentiment_wordclouds.png", bbox_inches="tight")
 if (len(covaxin_df) > 0):
     wordcloud_df = covaxin_df
     wordcloud_df['words'] = wordcloud_df.text.apply(lambda x: re.findall(r'\w+', x))
     get_smart_clouds(wordcloud_df).savefig("covaxin_sentiment_wordclouds.png", bbox_inches="tight")
 if (len(johnson_df) > 0):
     print(johnson_df)
     wordcloud_df = johnson_df
     wordcloud_df['words'] = wordcloud_df.text.apply(lambda x: re.findall(r'\w+', x))
     get_smart_clouds(wordcloud_df).savefig("johnson_sentiment_wordclouds.png", bbox_inches="tight")
```

```
def generate_word_clouds(neg_doc, neu_doc, pos_doc):
    # Display the generated image:
    fig, axes = plt.subplots(1, 3, figsize=(20, 10))
    print("------------ neg_doc: ", neg_doc)
    if(len(neg_doc)>0):
        wordcloud_neg = WordCloud(max_font_size=50, max_words=100, background_color="white").generate(" ".join(neg_doc))
        print("------------ wordcloud_neg: ",wordcloud_neg)
        axes[0].imshow(wordcloud_neg.recolor(color_func=red_color_func, random_state=3), interpolation='bilinear')
        axes[0].set_title("Negative Words")
        axes[0].axis("off")
    if (len(neu_doc) > 0):
        wordcloud_neu = WordCloud(max_font_size=50, max_words=100, background_color="white").generate(" ".join(neu_doc))
        axes[1].imshow(wordcloud_neu.recolor(color_func=yellow_color_func, random_state=3), interpolation='bilinear')
        axes[1].set_title("Neutral Words")
        axes[1].axis("off")
    if (len(pos_doc) > 0):
        wordcloud_pos = WordCloud(max_font_size=50, max_words=100, background_color="white").generate(" ".join(pos_doc))
        axes[2].imshow(wordcloud_pos.recolor(color_func=green_color_func, random_state=3), interpolation='bilinear')
        axes[2].set_title("Positive Words")
        axes[2].axis("off")

    plt.tight_layout()
    return fig
```

Fig. 35 –Code snapshots - TweetsSentimentAnalysisAllYears.py

Figures 36 to 46 depict multiple graphs generated while running the code.

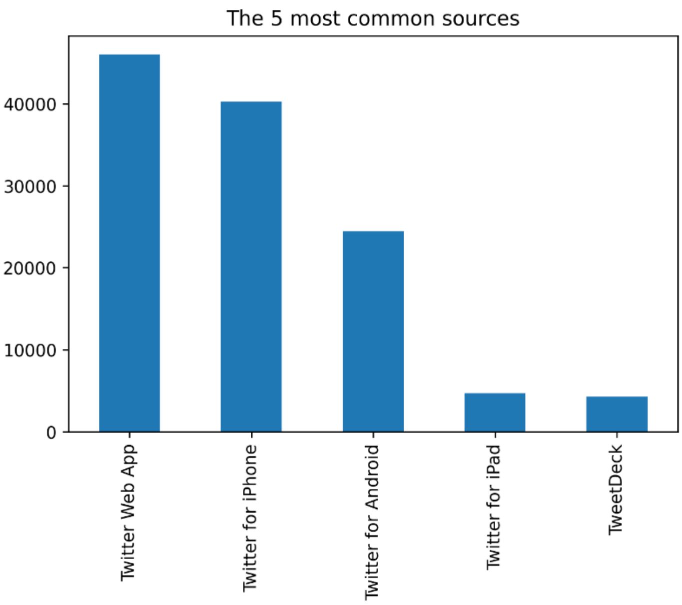


Fig. 36 – Tweets most common sources

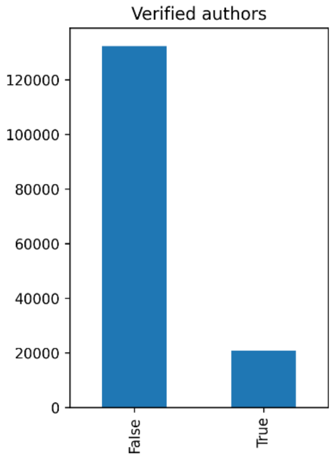


Fig. 37 – Verified authors

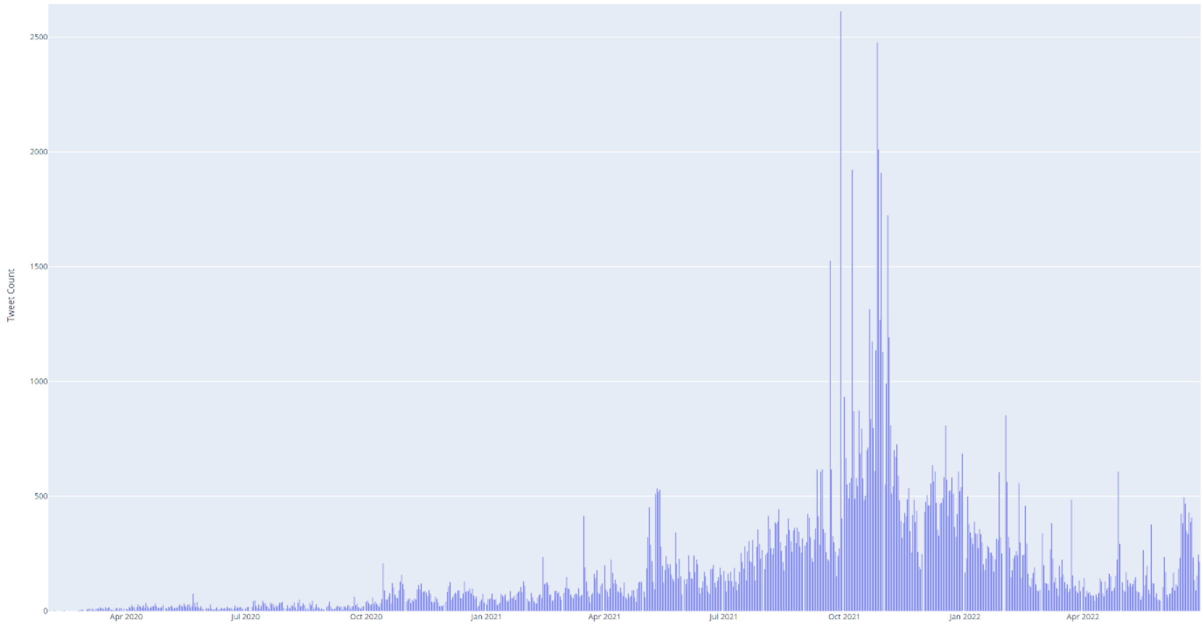


Fig. 38 – Tweets volume

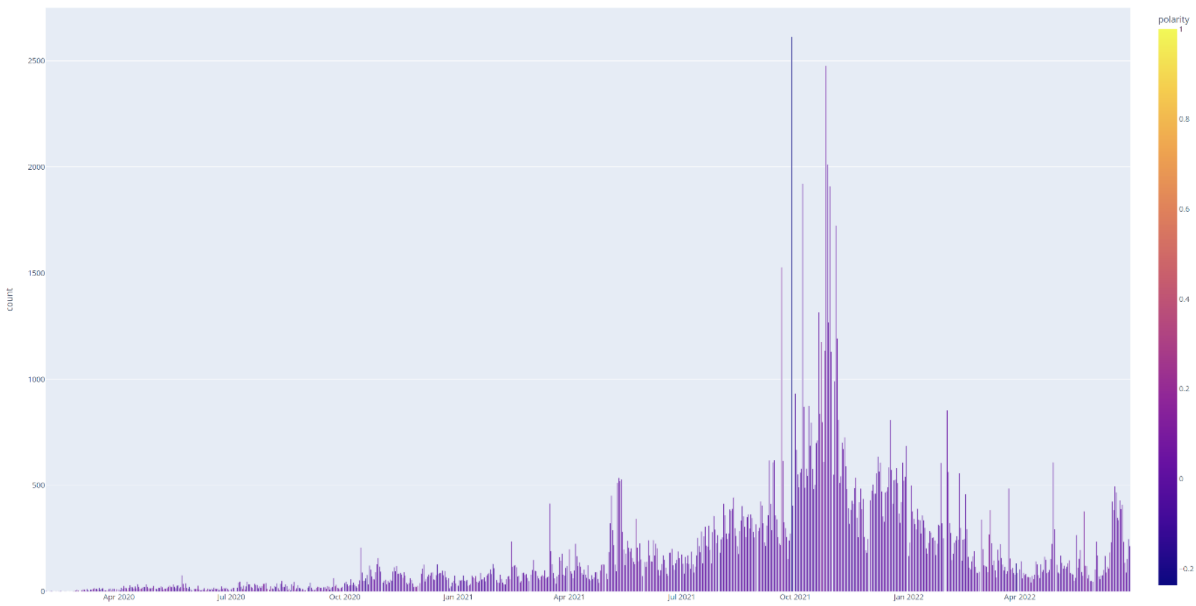


Fig. 39 – Tweets count by polarity

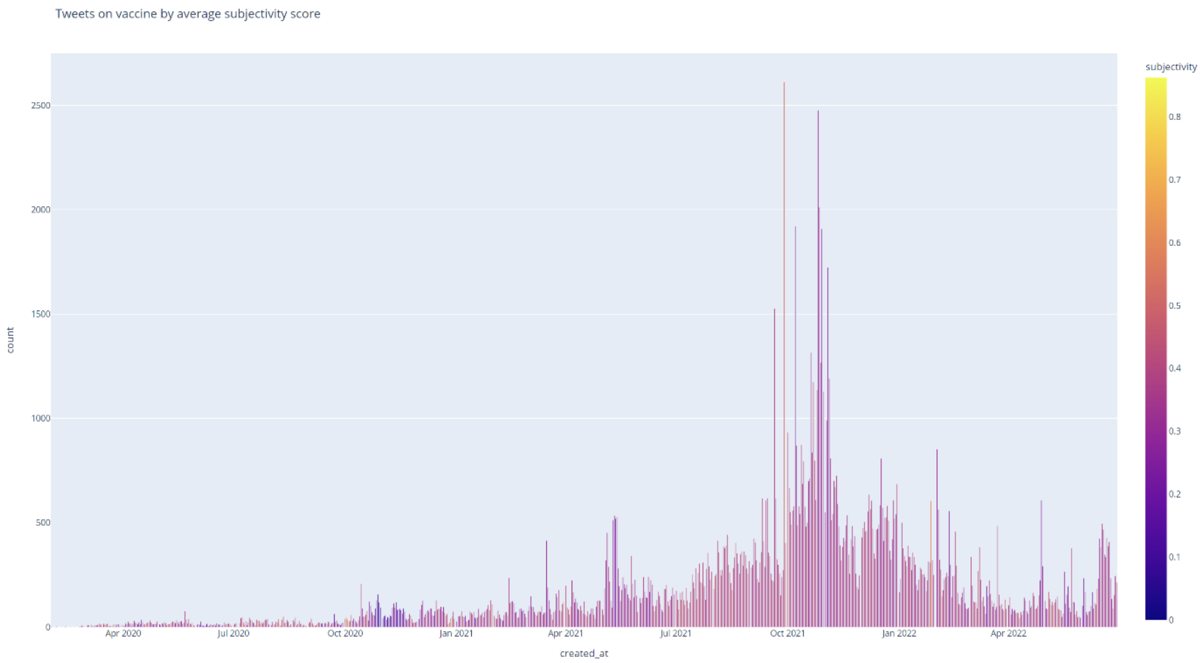


Fig. 40 – Tweets count by subjectivity

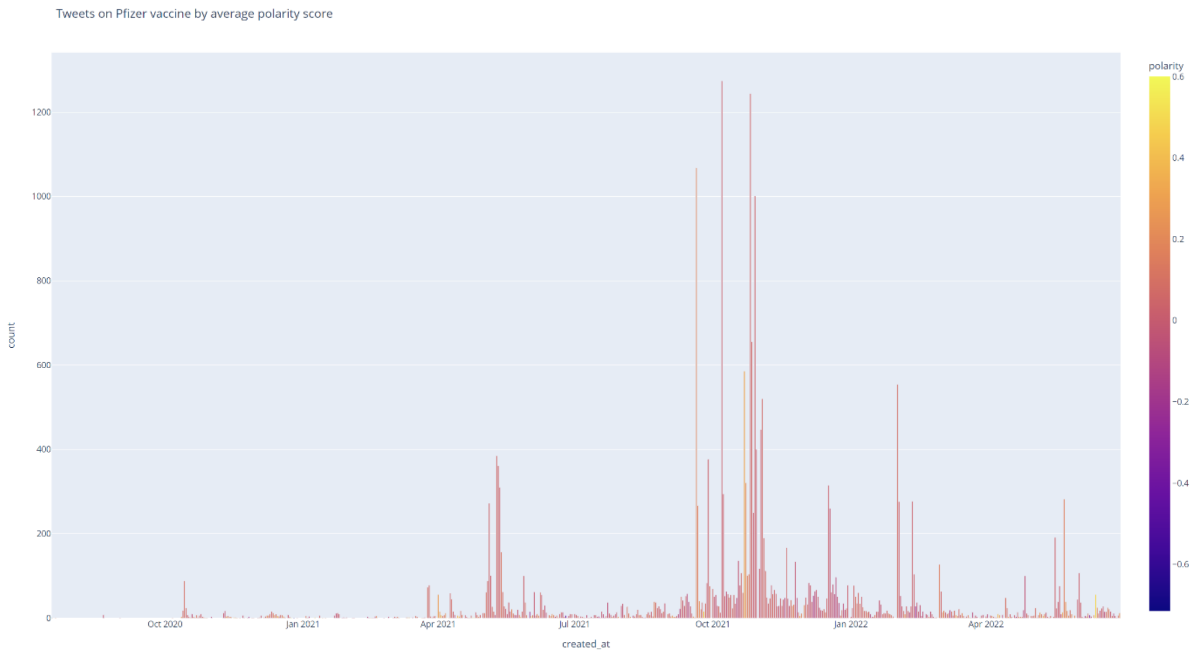


Fig. 41 – Tweets on Pfizer count by polarity

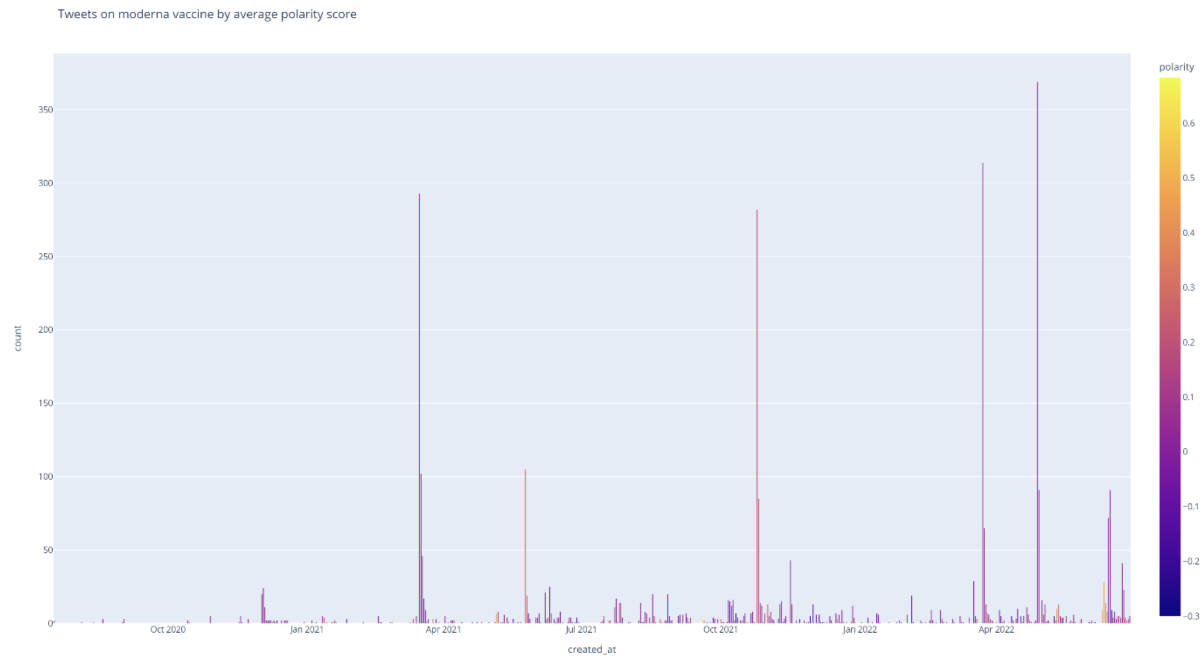


Fig. 42 – Tweets on Moderna count by polarity

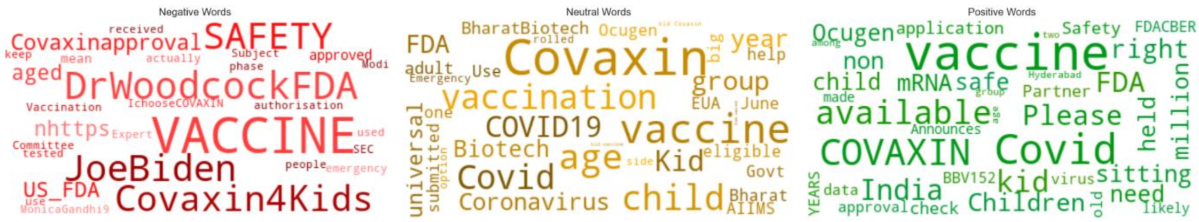


Fig. 43 - Sentiment word cloud - Covaxin

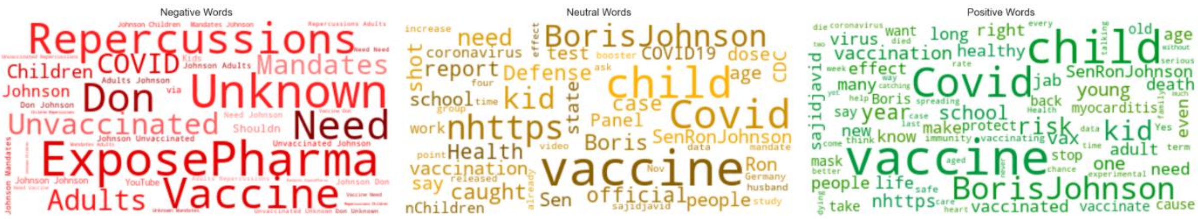


Fig. 44 - Sentiment word cloud – Johnson

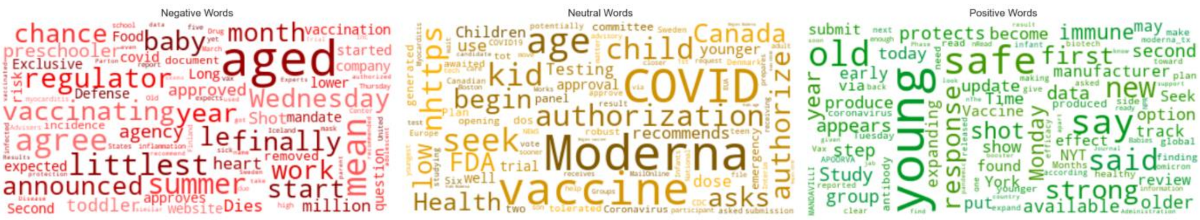


Fig. 45 - Sentiment word cloud – Moderna

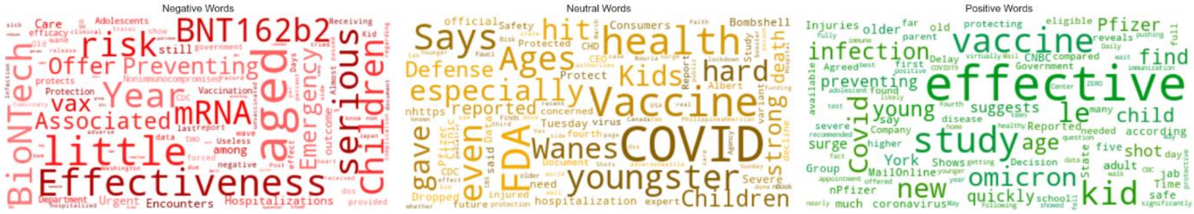


Fig. 46 - Sentiment word cloud – Pfizer

3.12 TweetsSentimentPredictionsAllYears.py file

Multiple deep learning models have been implemented in this file (SimpleRNN, single LSTM (Justin, 2020), Bidirectional LSTM, 1D Convolutional).

Figure 47 is an amalgamation of some code extract from this Python file.

```
def generateModels():
    #SimpleRNN model
    model0 = Sequential()
    model0.add(layers.Embedding(max_words, 15))
    model0.add(layers.SimpleRNN(15))
    model0.add(layers.Dense(3,activation='softmax'))
    model0.compile(optimizer='rmsprop',loss='categorical_crossentropy', metrics=['accuracy'])
    #Implementing model checkpoins to save the best metric and do not lose it on training.
    checkpoint0 = ModelCheckpoint("best_model0.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto', period=1,
                                  save_weights_only=False)
    history0 = model0.fit(X_train, y_train, epochs=5,validation_data=(X_test, y_test),callbacks=[checkpoint0])
    print(history0.params)
    print(history0.history.keys())

    accuracy = history0.history['accuracy']
    val_accuracy = history0.history['val_accuracy']
    loss = history0.history['loss']
    val_loss = history0.history['val_loss']
    print("----------------------------")
    print("SimpleRNN model:")
    print("accuracy: ",accuracy)
    print("val_accuracy: ", val_accuracy)
    print("loss: ", loss)
    print("val_loss: ", val_loss)
    score = model0.evaluate(X_test, y_test)
    print("Test Loss: %.2f%%" % (score[0] * 100))
    print("Test Accuracy: %.2f%%" % (score[1] * 100))
    print("----------------------------")
```

```
#Single LSTM layer model
print("Single LSTM layer model")
model1 = Sequential()
model1.add(layers.Embedding(max_words, 20))
model1.add(layers.LSTM(15,dropout=0.5))
model1.add(layers.Dense(3,activation='softmax'))
model1.compile(optimizer='rmsprop',loss='categorical_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint1 = ModelCheckpoint("best_model1.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto',
                              period=1,save_weights_only=False)
history1 = model1.fit(X_train, y_train, epochs=5,validation_data=(X_test, y_test),callbacks=[checkpoint1])
accuracy = history1.history['accuracy']
val_accuracy = history1.history['val_accuracy']
loss = history1.history['loss']
val_loss = history1.history['val_loss']
print("----------------------------")
print("Single LSTM layer model:")
print("accuracy: ",accuracy)
print("val_accuracy: ", val_accuracy)
print("loss: ", loss)
print("val_loss: ", val_loss)
score = model1.evaluate(X_test, y_test)
print("Test Loss: %.2f%%" % (score[0] * 100))
print("Test Accuracy: %.2f%%" % (score[1] * 100))
print("----------------------------")
```

```
#Bidirectional LSTM model
model2 = Sequential()
model2.add(layers.Embedding(max_words, 40, input_length=max_len))
model2.add(layers.Bidirectional(layers.LSTM(20,dropout=0.6)))
model2.add(layers.Dense(3,activation='softmax'))
model2.compile(optimizer='rmsprop',loss='categorical_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint2 = ModelCheckpoint("best_model2.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto',
                              period=1,save_weights_only=False)
history2 = model2.fit(X_train, y_train, epochs=5,validation_data=(X_test, y_test),callbacks=[checkpoint2])
accuracy = history2.history['accuracy']
val_accuracy = history2.history['val_accuracy']
loss = history2.history['loss']
val_loss = history2.history['val_loss']
print("----------------------------")
print("Bidirectional LSTM model:")
print("accuracy: ", accuracy)
print("val_accuracy: ", val_accuracy)
print("loss: ", loss)
print("val_loss: ", val_loss)
score = model2.evaluate(X_test, y_test)
print("Test Loss: %.2f%%" % (score[0] * 100))
print("Test Accuracy: %.2f%%" % (score[1] * 100))
import matplotlib.pyplot as plt
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
#1D Convolutional model
print("1D Convolutional model")
from keras import regularizers
model3 = Sequential()
model3.add(layers.Embedding(max_words, 40, input_length=max_len))
model3.add(layers.Conv1D(20, 6, activation='relu',kernel_regularizer=regularizers.l1_l2(l1=2e-3, l2=2e-3),
                         bias_regularizer=regularizers.l2(2e-3)))
model3.add(layers.MaxPooling1D(5))
model3.add(layers.Conv1D(20, 6, activation='relu',kernel_regularizer=regularizers.l1_l2(l1=2e-3, l2=2e-3),
                         bias_regularizer=regularizers.l2(2e-3)))
model3.add(layers.GlobalMaxPooling1D())
model3.add(layers.Dense(3,activation='softmax'))
model3.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['acc'])
checkpoint3 = ModelCheckpoint("best_model3.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto',
                              period=1,save_weights_only=False)
history3 = model3.fit(X_train, y_train, epochs=5,validation_data=(X_test, y_test),callbacks=[checkpoint3])
accuracy = history3.history['acc']
val_accuracy = history3.history['val_acc']
loss = history3.history['loss']
val_loss = history3.history['val_loss']
print("----------------------------")
print("1D Convolutional model model:")
print("accuracy: ", accuracy)
print("val_accuracy: ", val_accuracy)
print("loss: ", loss)
print("val_loss: ", val_loss)
score = model3.evaluate(X_test, y_test)
print("Test Loss: %.2f%%" % (score[0] * 100))
print("Test Accuracy: %.2f%%" % (score[1] * 100))
print("----------------------------")
```

Fig. 47 – Code extract -TweetsSentimentPredictionsAllYears.py

Figure 48 depicts multiple snapshots of code output. Figure 49 is a representation of the best performing model in this particular set of models (Bi-directional LSTM).

```
   Unnamed: 0  created_at  ...  Compound_Score  sentiment_score
0         168  2020-02-12  ...          0.1526                1
1         176  2020-02-09  ...          0.0000                1
2         233  2020-02-19  ...         -0.9313                1
3         240  2020-02-25  ...         -0.2960                1
4         240  2020-02-25  ...         -0.2960                1
5          57  2020-02-27  ...          0.0000                1
6         317  2020-02-10  ...          0.4588                1
7         101  2020-02-26  ...          0.6124                1
8         129  2020-02-25  ...          0.3400                1
9         333  2020-02-28  ...         -0.7906                1
10        266  2020-02-04  ...         -0.5267                1
11        337  2020-02-08  ...         -0.8368                1
12        337  2020-02-08  ...         -0.8368                1
13         83  2020-02-26  ...          0.0000                1
14        209  2020-02-25  ...         -0.5106                1
1542744
['neutral' 'negative' 'positive']
                                                text sentiment
0  A kid asked me why I trust snopes when I don't...   neutral
1  I’m telling my kids that the vaccine for the c...   neutral
2  @ABC7Chicago Do people realize that the flu ki...   neutral
3  @neiltyson They would refuse the vaccine and t...   neutral
4  @neiltyson They would refuse the vaccine and t...   neutral
0
```

```
Epoch 1/85
36159/36159 [==============================] - ETA: 0s - loss: 0.3723 - accuracy: 0.8724
Epoch 1: val_accuracy improved from -inf to 0.90576, saving model to best_model0.hdf5
36159/36159 [==============================] - 1294s 36ms/step - loss: 0.3723 - accuracy: 0.8724 - val_loss: 0.2914 - val_accuracy: 0.9058
Epoch 2/85
36158/36159 [============================>.] - ETA: 0s - loss: 0.3068 - accuracy: 0.9009
Epoch 2: val_accuracy improved from 0.90576 to 0.91353, saving model to best_model0.hdf5
Epoch 3/85
36158/36159 [============================>.] - ETA: 0s - loss: 0.2854 - accuracy: 0.9116
Epoch 3: val_accuracy improved from 0.91353 to 0.91691, saving model to best_model0.hdf5
36159/36159 [==============================] - 1882s 52ms/step - loss: 0.2854 - accuracy: 0.9116 - val_loss: 0.2735 - val_accuracy: 0.9169
Epoch 4/85
36158/36159 [============================>.] - ETA: 0s - loss: 0.2953 - accuracy: 0.9073
Epoch 4: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 1539s 43ms/step - loss: 0.2953 - accuracy: 0.9073 - val_loss: 0.2719 - val_accuracy: 0.9164
Epoch 5/85
36159/36159 [==============================] - ETA: 0s - loss: 0.2846 - accuracy: 0.9121
Epoch 5: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 1724s 48ms/step - loss: 0.2846 - accuracy: 0.9121 - val_loss: 0.2821 - val_accuracy: 0.9151
Epoch 6/85
36159/36159 [==============================] - ETA: 0s - loss: 0.3099 - accuracy: 0.9040
Epoch 6: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 1374s 38ms/step - loss: 0.3099 - accuracy: 0.9040 - val_loss: 0.2998 - val_accuracy: 0.9044
Epoch 7/85
36157/36159 [============================>.] - ETA: 0s - loss: 0.3007 - accuracy: 0.9075
Epoch 7: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 1436s 40ms/step - loss: 0.3007 - accuracy: 0.9075 - val_loss: 0.2849 - val_accuracy: 0.9120
Epoch 8/85
36159/36159 [==============================] - ETA: 0s - loss: 0.2991 - accuracy: 0.9066
Epoch 8: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 1694s 47ms/step - loss: 0.2991 - accuracy: 0.9066 - val_loss: 0.2803 - val_accuracy: 0.9145
Epoch 9/85
36159/36159 [==============================] - ETA: 0s - loss: 0.2797 - accuracy: 0.9145
Epoch 9: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 1560s 43ms/step - loss: 0.2797 - accuracy: 0.9145 - val_loss: 0.2817 - val_accuracy: 0.9130
Epoch 10/85
36159/36159 [==============================] - ETA: 0s - loss: 0.2807 - accuracy: 0.9140
Epoch 10: val_accuracy did not improve from 0.91691
36159/36159 [==============================] - 2355s 65ms/step - loss: 0.2807 - accuracy: 0.9140 - val_loss: 0.2873 - val_accuracy: 0.9093
```

Fig. 48 – Outputs – extract

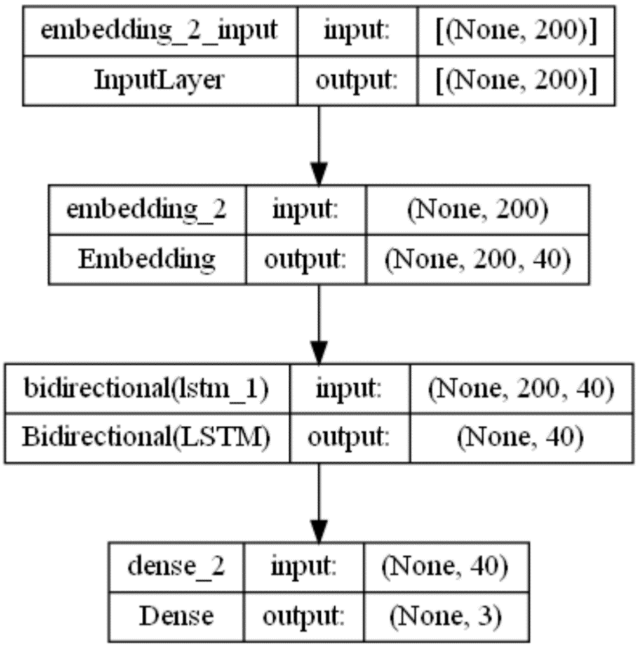


Fig. 49 – Best performing model – Bi-directional LSTM

Bidirectional LSTM model constantly outperformed the others, with best accuracy at 71% and epoch 85 as shown in figures 50 to 54.

```
3/3 - 1s - loss: 0.8646 - accuracy: 0.6706 - 660ms/epoch - 220ms/step
Best model accuracy:  0.6705882549285889
3/3 [==============================] - 1s 13ms/step
calculate the AUC for model
0.7382221545334552
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 200, 40)           200000

 bidirectional (Bidirectiona  (None, 40)               9760
 l)
```

Fig. 50 - Bidirectional LSTM - Epoch 70

```
Test Accuracy: 62.35%
-----------------------------
3/3 - 1s - loss: 0.9061 - accuracy: 0.6471 - 656ms/epoch - 219ms/step
Best model accuracy:  0.6470588445663452
3/3 [==============================] - 1s 13ms/step
calculate the AUC for model
0.7291857041323992
Model: "sequential_2"
```

Fig. 51 - Bidirectional LSTM - Epoch 200

```
Best model accuracy:  0.7058823704719543
3/3 [==============================] - 1s 15ms/step
calculate the AUC for model
0.7533167495854064
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 200, 40)           200000

 bidirectional (Bidirectiona  (None, 40)               9760
 l)
```

Fig. 52 - Bidirectional LSTM - Epoch 80

```
Best model accuracy:  0.6941176652908325
3/3 [==============================] - 1s 15ms/step
calculate the AUC for model
0.7670237249128506
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 200, 40)           200000

 bidirectional (Bidirectiona  (None, 40)               9760
 l)
```

Fig. 53 - Bidirectional LSTM - Epoch 90

```
Best model accuracy:  0.7058823704719543
3/3 [==============================] - 1s 12ms/step
calculate the AUC for model
0.7173711939170362
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 200, 40)           200000

 bidirectional (Bidirectiona  (None, 40)               9760
 l)
```

Fig. 54 - Bidirectional LSTM - Epoch 85

3.13 LDATopicsExtraction.py file

Topics were extracted and graphed in this Python file. Figure 55 is an amalgamation of some snapshots from the code while figure 56 is a graph generated with this file. The NCR lexicon was used to associate words with emotions (figure 57).

```
# LDA topics
def get_topics(edited, n_topics, n_words):
    eds = edited.values
    vec = TfidfVectorizer(use_idf=True, smooth_idf=True)
    document_term_matrix = vec.fit_transform(eds)

    model = LatentDirichletAllocation(n_components=n_topics)
    topic_matrix = model.fit_transform(document_term_matrix)

    keys = get_keys(topic_matrix)
    categories, counts = keys_to_counts(keys)
    top_n_words = get_top_n_words(n_words, n_topics, keys, document_term_matrix, vec)

    topics = ['Topic {}: \n'.format(i + 1) + top_n_words[i] for i in categories]
    data = []
    for i, topic in enumerate(topics):
        tmp = []
        tmp.append(topic)
        tmp.append(counts[i])
        data.append(tmp)
    df_topics = pd.DataFrame(data, columns=['Topics', 'Count'])
    return df_topics
```

```
# Aggregating negative and positive emotions
df_emo['neg_emotions'] = df_emo['Sadness'] + df_emo['Fear'] + df_emo['Disgust'] + df_emo['Anger']
df_emo['pos_emotions'] = df_emo['Joy'] + df_emo['Anticipation'] + df_emo['Trust'] + df_emo['Surprise']
df_emo['total_neg_emotions'] = df_emo['neg_emotions'].apply(lambda x: x > 0)
df_emo['total_pos_emotions'] = df_emo['pos_emotions'].apply(lambda x: x > 0)

props = df_emo['total_neg_emotions'].value_counts(normalize=True).unstack()
print(props)

df1 = df_emo[emotions].apply(lambda x:( x.sum()/x.count())*100)
print(df1.head())

df_ =df1.T

print(df_.reset_index())

fig, ax = plt.subplots(1, 1, figsize=(10, 6))
ax.set_title(label='Percentage of emotion-related words in tweets\n', fontweight='bold', size=18)
df_.plot(
    x="index", y="emotions", kind="bar", ax=ax
)


plt.xlabel("Emotions",fontsize = 16)
plt.ylabel("Percentage of emotion-related words",fontsize = 16)
plt.xticks(rotation=45,fontsize=14)
plt.tight_layout()
plt.savefig('images/Percentage_emotions.png')
```

Fig. 55 – Code snapshots – LDATopicsExtraction.py

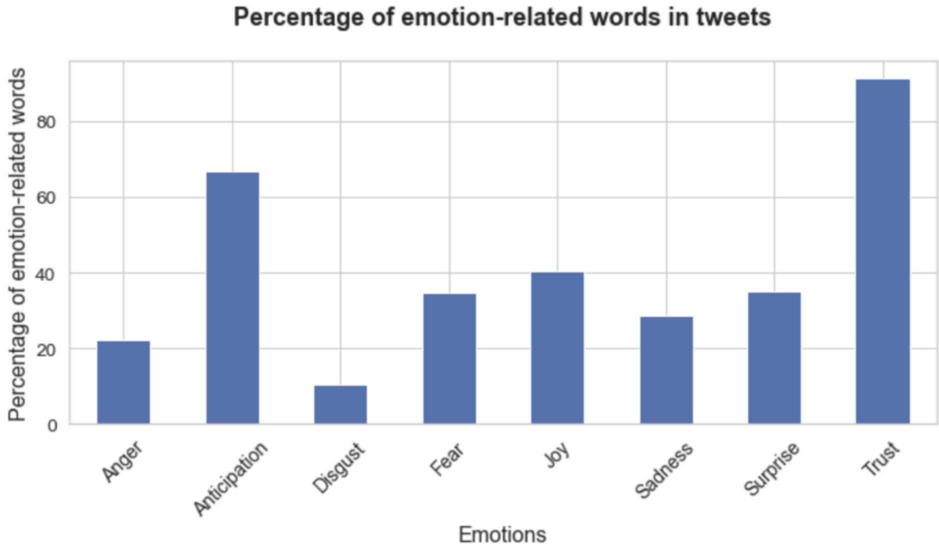


Fig. 56 – Emotion-related words

```
English;Positive;Negative;Anger;Anticipation;Disgust;Fear;Joy;Sadness;Surprise;Trust
aback;0;0;0;0;0;0;0;0;0;0
abacus;0;0;0;0;0;0;0;0;0;1
abandon;0;1;0;0;0;1;0;1;0;0
abandoned;0;1;1;0;0;1;0;1;0;0
abandonment;0;1;1;0;0;1;0;1;1;0
abate;0;0;0;0;0;0;0;0;0;0
abatement;0;0;0;0;0;0;0;0;0;0
abba;1;0;0;0;0;0;0;0;0;0
abbot;0;0;0;0;0;0;0;0;0;1
abbreviate;0;0;0;0;0;0;0;0;0;0
abbreviation;0;0;0;0;0;0;0;0;0;0
abdomen;0;0;0;0;0;0;0;0;0;0
abdominal;0;0;0;0;0;0;0;0;0;0
abduction;0;1;0;0;0;1;0;1;1;0
aberrant;0;1;0;0;0;0;0;0;0;0
aberration;0;1;0;0;1;0;0;0;0;0
abeyance;0;0;0;0;0;0;0;0;0;0
abhor;0;1;1;0;1;1;0;0;0;0
abhorrent;0;1;1;0;1;1;0;0;0;0
abide;0;0;0;0;0;0;0;0;0;0
ability;1;0;0;0;0;0;0;0;0;0
```
Fig. 57 – NCR-lexicon.csv snapshot

3.14 TopicModellingLDA.py file

A dynamic graph was generated in this file to display topics. Figure 58 is an amalgamation of some snapshots from the code while figures 59 to 67 depict graphs generated with this file.

```python
# compute the coherence scores for each number of topics
for i in range(2, 11):
    # create lda model with i topics
    lda = LdaModel(corpus=bow, num_topics=i, id2word=dictionary, random_state=42)
    # obtain the coherence score
    coherence_model = CoherenceModel(model=lda, texts=doc_list, dictionary=dictionary, coherence='c_v')
    coherence_score = np.round(coherence_model.get_coherence(), 2)
    if coherence_score > best_score:
        best_num = i
        best_score = coherence_score

print(f'The coherence score is highest ({best_score}) with {best_num} topics.')

# build the lda model
lda_model = gensim.models.ldamodel.LdaModel(corpus=bow,
                                            id2word=dictionary,
                                            num_topics=9,
                                            random_state=42)

# show the words most strongly associated with each topic
for topic in lda_model.print_topics():
    print(topic)

# obtain topic distributions for each document
topic_dist = lda_model[bow]
```

```python
import pyLDAvis
import pyLDAvis.gensim_models as gensim_models

# visualize LDA model results
pyLDAvis.enable_notebook()
gensim_models.prepare(lda_model, dictionary=dictionary, corpus=bow)
```

Fig. 58 – Code snapshots – TopicModelingLDA.py

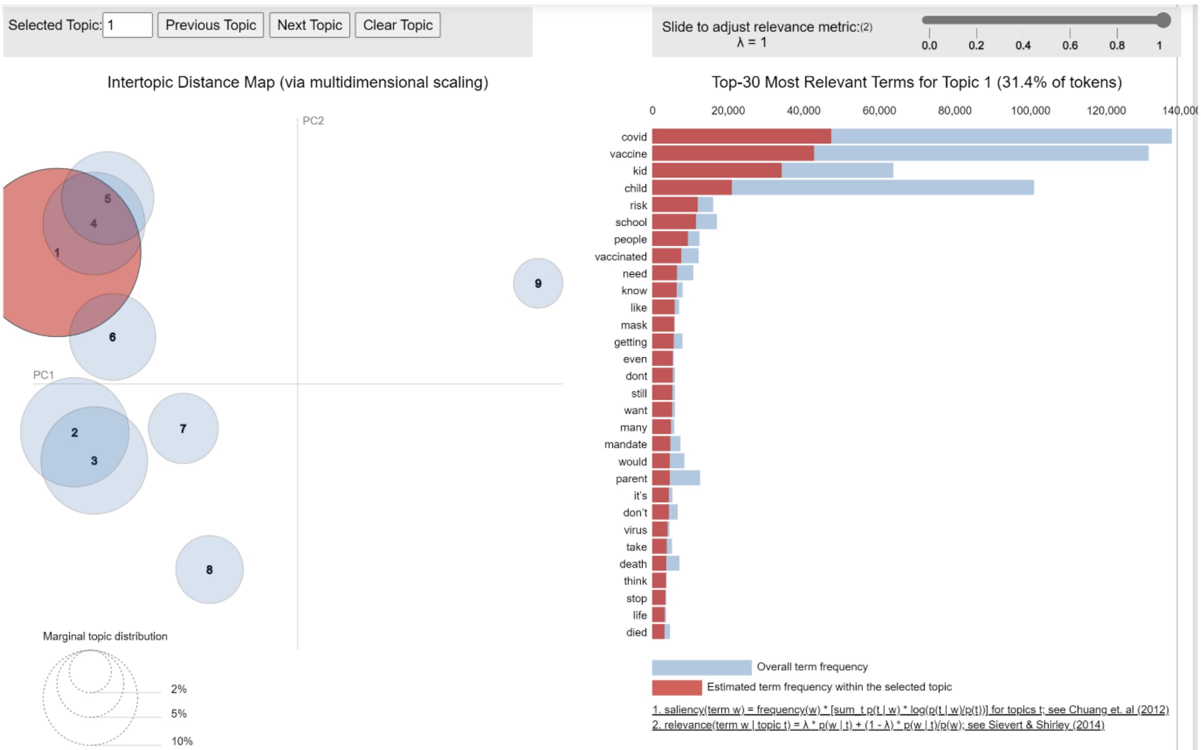


Fig. 59 – Topic 1

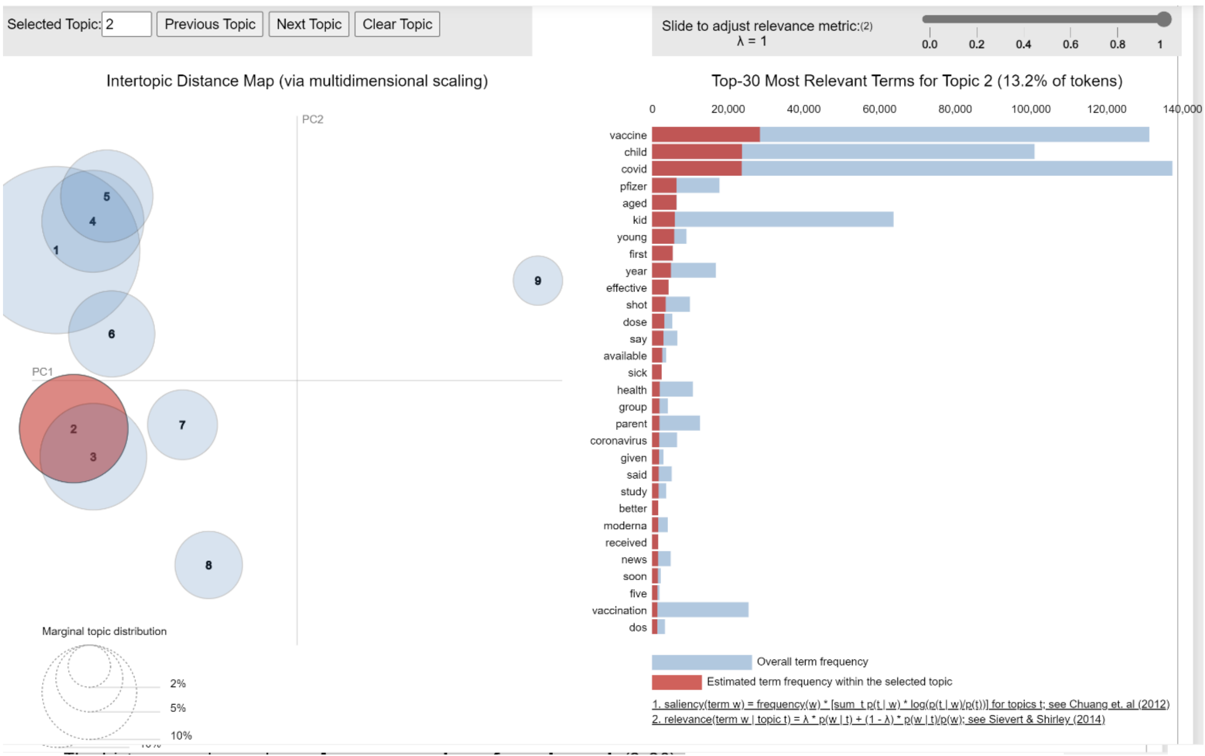


Fig. 60 – Topic 2

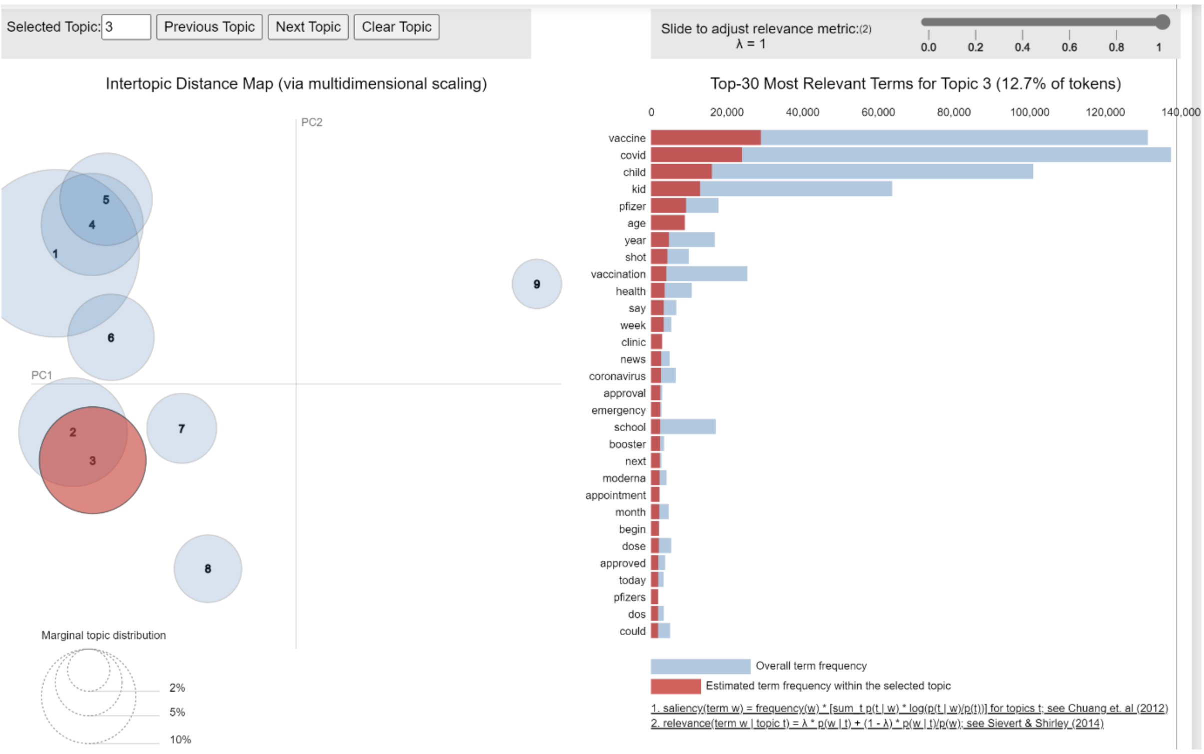


Fig. 61 – Topic 3

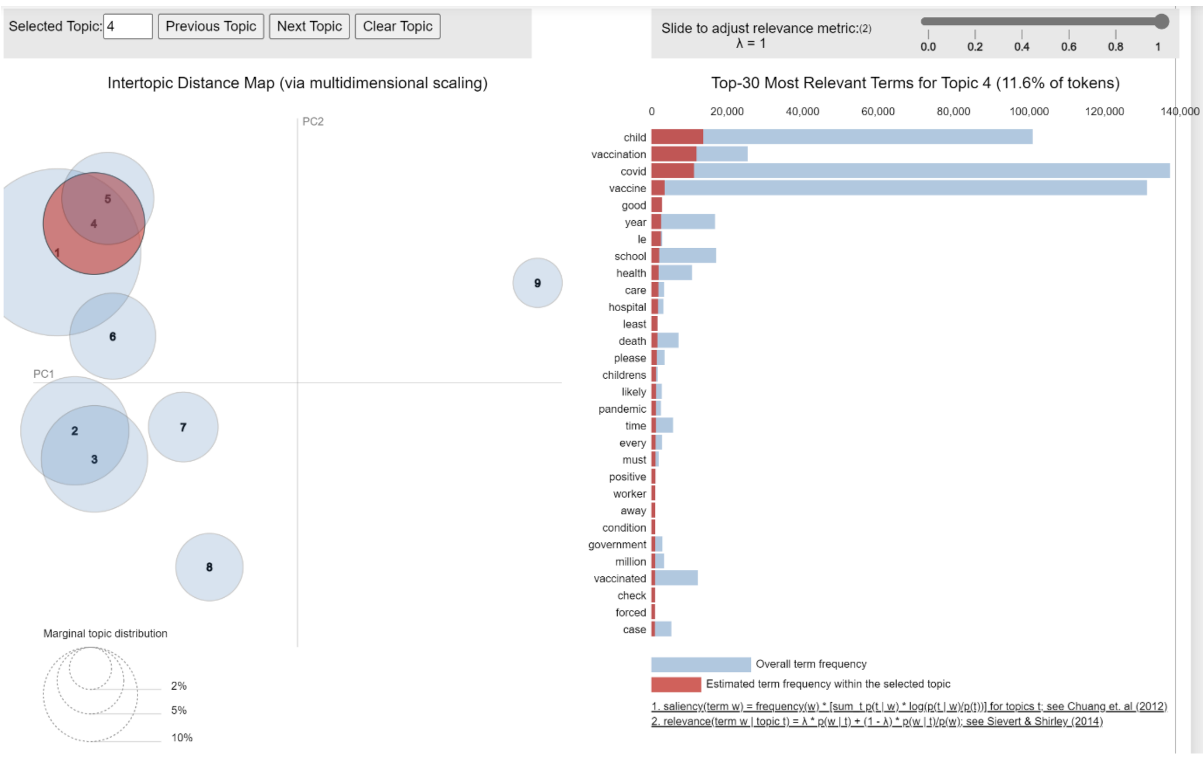


Fig. 62 – Topic 4

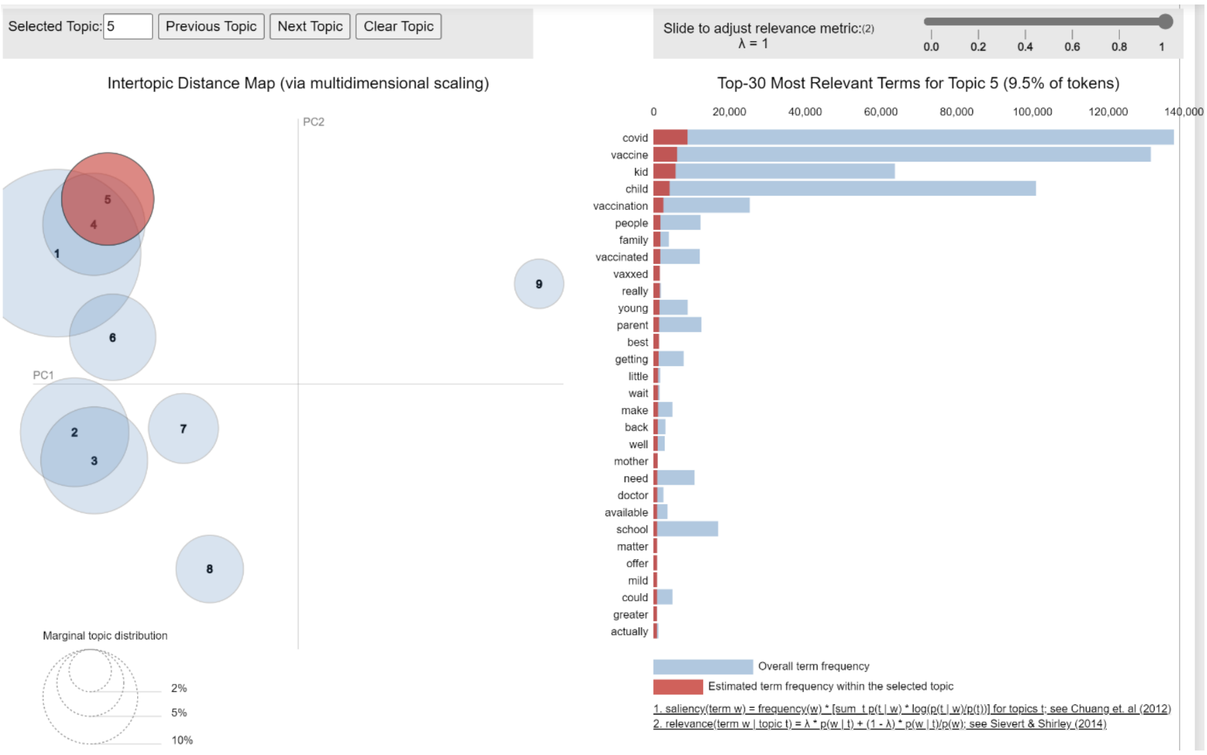


Fig. 63 – Topic 5

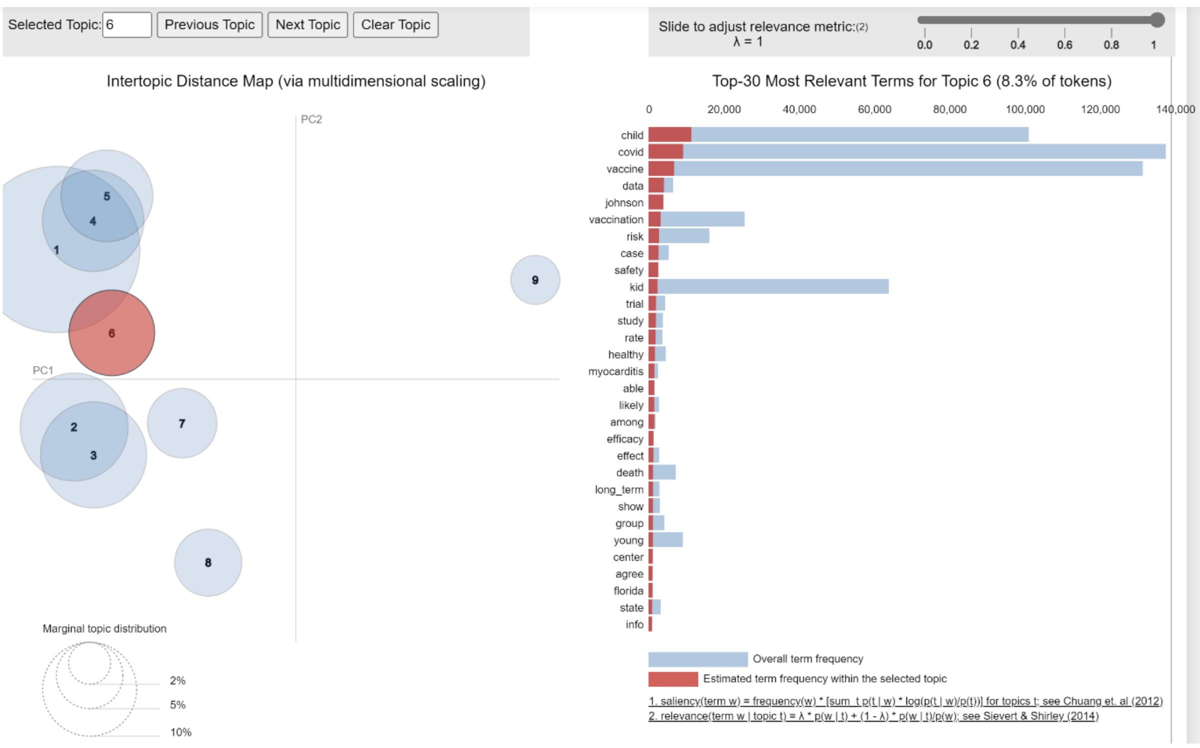


Fig. 64 – Topic 6

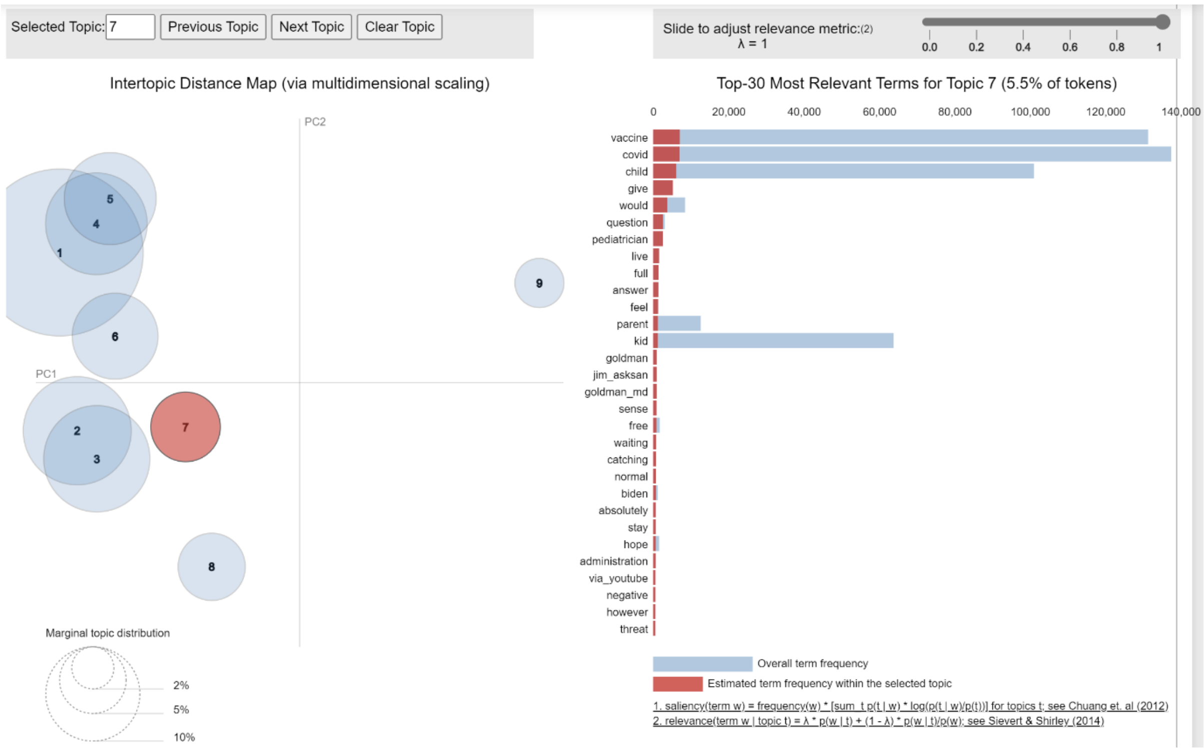


Fig. 65 – Topic 7

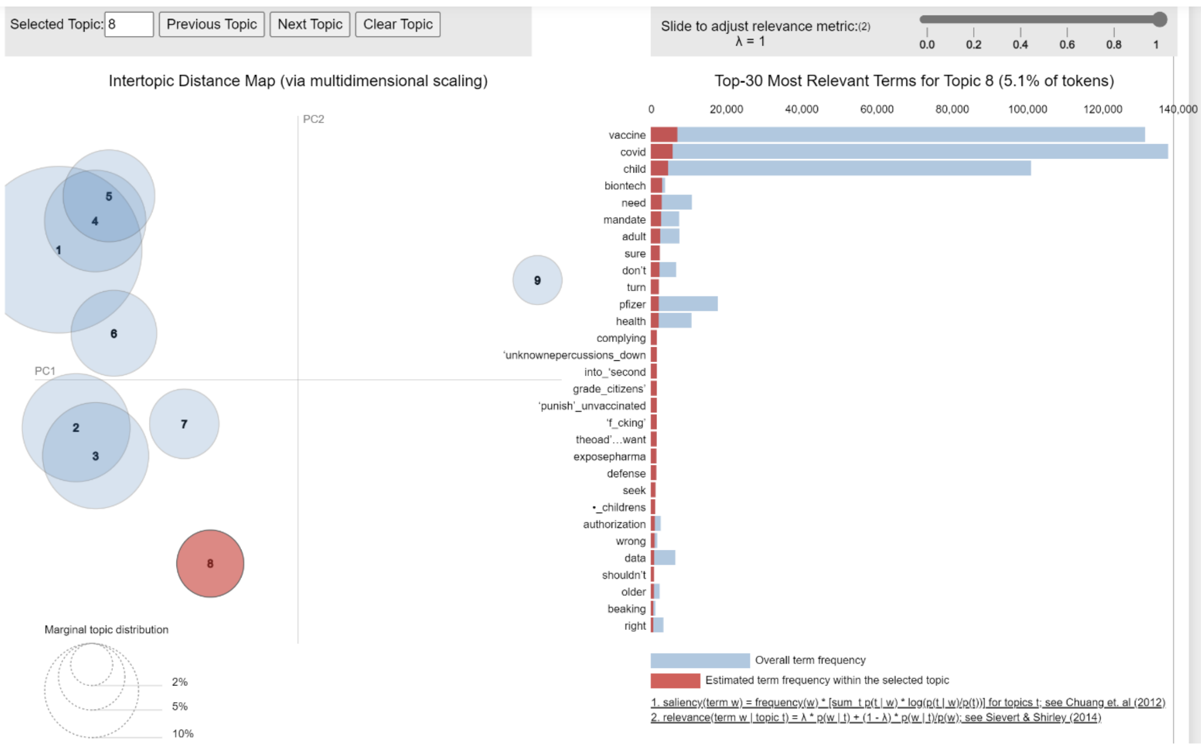


Fig. 66 – Topic 8

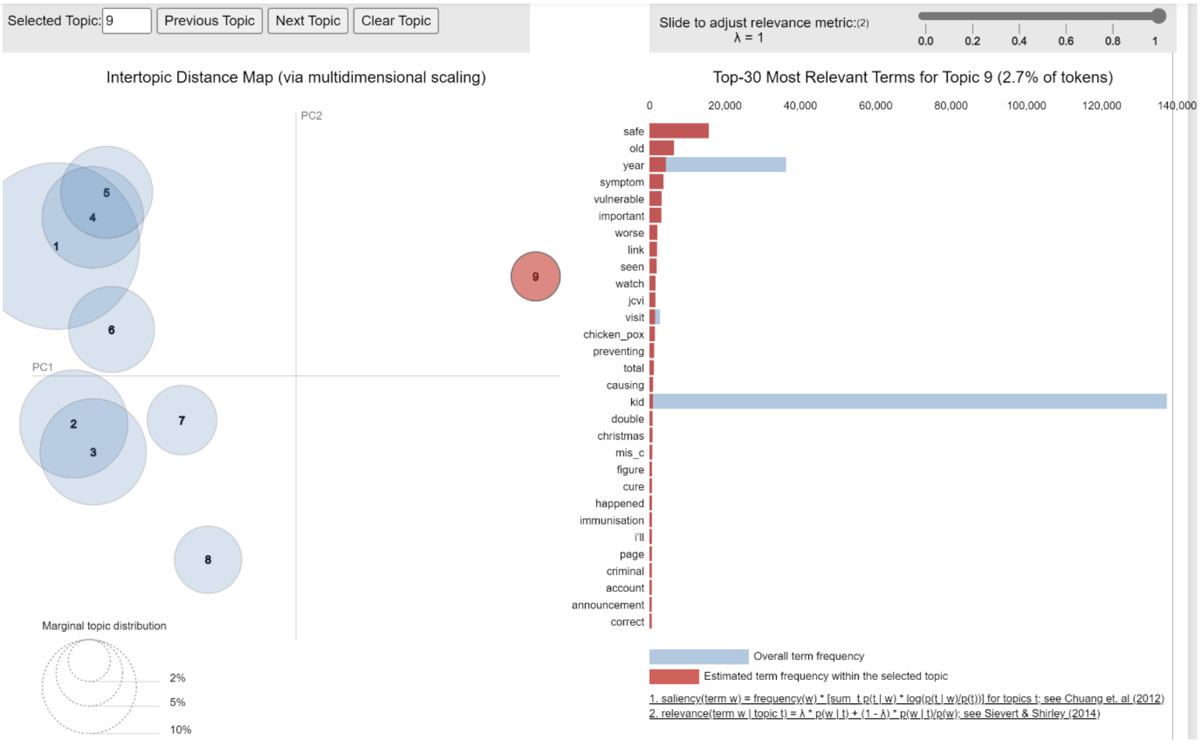


Fig. 67 – Topic 9

3.15 BERT.py file

BERT model was implemented with max_len set to 128, batch_size 32, hidden size 768, hidden size classifier 50, number of labels 5 and BERT model instantiation: bert-base-uncased, (Classify Text with BERT, 2022). Figure 68 is an amalgamation of code extracts from this file.

```python
#BERT
X = df['text_clean'].values
y = df['sentiment_score'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=seed_value)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.1, stratify=y_train,
                                                      random_state=seed_value)

ros = RandomOverSampler()
X_train_os, y_train_os = ros.fit_resample(np.array(X_train).reshape(-1,1),np.array(y_train).reshape(-1,1))

X_train_os = X_train_os.flatten()
y_train_os = y_train_os.flatten()

(unique, counts) = np.unique(y_train_os, return_counts=True)
print(np.asarray((unique, counts)).T)

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```python
def bert_tokenizer(data):
    input_ids = []
    attention_masks = []
    for sent in data:
        encoded_sent = tokenizer.encode_plus(
            text=sent,
            add_special_tokens=True,        # Add [CLS] and [SEP] special tokens
            max_length=MAX_LEN,             # Choose max length to truncate/pad
            pad_to_max_length=True,         # Pad sentence to max length
            return_attention_mask=True      # Return attention mask
            )
        input_ids.append(encoded_sent.get('input_ids'))
        attention_masks.append(encoded_sent.get('attention_mask'))
    # Convert lists to tensors
    input_ids = torch.tensor(input_ids)
    attention_masks = torch.tensor(attention_masks)
    return input_ids, attention_masks

# Tokenize train tweets
encoded_tweets = [tokenizer.encode(sent, add_special_tokens=True) for sent in X_train]

# Find the longest tokenized tweet
max_len = max([len(sent) for sent in encoded_tweets])
print('Max length: ', max_len)

MAX_LEN = 128
```

```python
MAX_LEN = 128

train_inputs, train_masks = bert_tokenizer(X_train_os)
val_inputs, val_masks = bert_tokenizer(X_valid)
test_inputs, test_masks = bert_tokenizer(X_test)

# Convert target columns to pytorch tensors format
train_labels = torch.from_numpy(y_train_os)
val_labels = torch.from_numpy(y_valid)
test_labels = torch.from_numpy(y_test)

batch_size = 32

# Create the DataLoader training set
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)

# Create the DataLoader validation set
val_data = TensorDataset(val_inputs, val_masks, val_labels)
val_sampler = SequentialSampler(val_data)
val_dataloader = DataLoader(val_data, sampler=val_sampler, batch_size=batch_size)

# Create the DataLoader test set
test_data = TensorDataset(test_inputs, test_masks, test_labels)
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=batch_size)
```

```python
class Bert_Classifier(nn.Module):
    def __init__(self, freeze_bert=False):
        super(Bert_Classifier, self).__init__()
        # Specify hidden size of BERT, hidden size of the classifier, and number of labels
        n_input = 768
        n_hidden = 50
        n_output = 5
        # Instantiate BERT model
        self.bert = BertModel.from_pretrained('bert-base-uncased')

        # Add dense layers to perform the classification
        self.classifier = nn.Sequential(
            nn.Linear(n_input, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_output)
        )
        # Add possibility to freeze the BERT model
        # to avoid fine tuning BERT params (usually leads to worse results)
        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False
    def forward(self, input_ids, attention_mask):
        # Feed input data to BERT
        outputs = self.bert(input_ids=input_ids,
                            attention_mask=attention_mask)
        # Extract the last hidden state of the token `[CLS]` for classification task
        last_hidden_state_cls = outputs[0][:, 0, :]
        # Feed input to classifier to compute logits
        logits = self.classifier(last_hidden_state_cls)
        return logits
```

```
def bert_train(model, train_dataloader, val_dataloader=None, epochs=4, evaluation=False):
    print("Start training...\n")
    for epoch_i in range(epochs):
        print("-" * 10)
        print("Epoch : {}".format(epoch_i + 1))
        print("-" * 10)
        print("-" * 38)
        print(f"{'BATCH NO.':^7} | {'TRAIN LOSS':^12} | {'ELAPSED (s)':^9}")
        print("-" * 38)

        # Measure the elapsed time of each epoch
        t0_epoch, t0_batch = time.time(), time.time()
        # Reset tracking variables at the beginning of each epoch
        total_loss, batch_loss, batch_counts = 0, 0, 0
        # TRAINING
        # Put the model into the training mode
        model.train()

        for step, batch in enumerate(train_dataloader):
            batch_counts += 1
            b_input_ids, b_attn_mask, b_labels = tuple(t.to(device) for t in batch)
            # Zero out any previously calculated gradients
            model.zero_grad()
            # Perform a forward pass and get logits.
            logits = model(b_input_ids, b_attn_mask)
```

```
bert_preds = bert_predict(bert_classifier, test_dataloader)

print('Classification Report for BERT :\n', classification_report(y_test, bert_preds, target_names=sentiments))
```

```
vocabulary, tokenized_column = Tokenize(df["text_clean"], max_len)
print(df["text_clean"].iloc[10])
print(tokenized_column[10])

keys = []
values = []
for key, value in vocabulary[:20]:
    keys.append(key)
    values.append(value)

plt.figure(figsize=(15, 5))
ax = sns.barplot(keys, values, palette='mako')
plt.title('Top 20 most common words', size=25)
ax.bar_label(ax.containers[0])
plt.ylabel("Words count")
plt.show()
```
Fig. 68 – Code extract - BERT.py

The BERT model was ran with epoch set to 2 and produced an accuracy of 90.3% as depicted in figure 69.

```
----------
Epoch : 1
----------
--------------------------------------
BATCH NO. |  TRAIN LOSS  | ELAPSED (s)
--------------------------------------
   100    |   1.049314   |  6185.60
   200    |   0.775140   |  1767.06
   300    |   0.645697   |  1667.41
   400    |   0.558958   |  5717.11
   500    |   0.510986   |  5260.15
   600    |   0.461327   | 20842.00
   700    |   0.430039   |  2141.64
   800    |   0.411655   |  4689.99
   853    |   0.401362   |  1418.17
-------------------------------------------------------------
AVG TRAIN LOSS |  VAL LOSS  | VAL ACCURACY (%) | ELAPSED (s)
-------------------------------------------------------------
   0.593247    |  0.373901  |       87.30      | 50658.52
-------------------------------------------------------------

Epoch : 2
----------
--------------------------------------
BATCH NO. |  TRAIN LOSS  | ELAPSED (s)
--------------------------------------
   100    |   0.296424   |  2361.84
   200    |   0.312465   |  2449.52
   300    |   0.295645   | 25136.60
   400    |   0.283081   | 15453.88
   500    |   0.270717   |  1782.32
   600    |   0.274128   |  2081.96
   700    |   0.247663   | 10401.61
   800    |   0.242964   | 16030.65
   853    |   0.248143   |  1128.01
-------------------------------------------------------------
AVG TRAIN LOSS |  VAL LOSS  | VAL ACCURACY (%) | ELAPSED (s)
-------------------------------------------------------------
   0.276062    |  0.303337  |       90.30      | 78621.52
-------------------------------------------------------------
```

Fig. 69 - BERT model with epoch set 2

Further analysis was carried out in this file as depicted in figure 70.

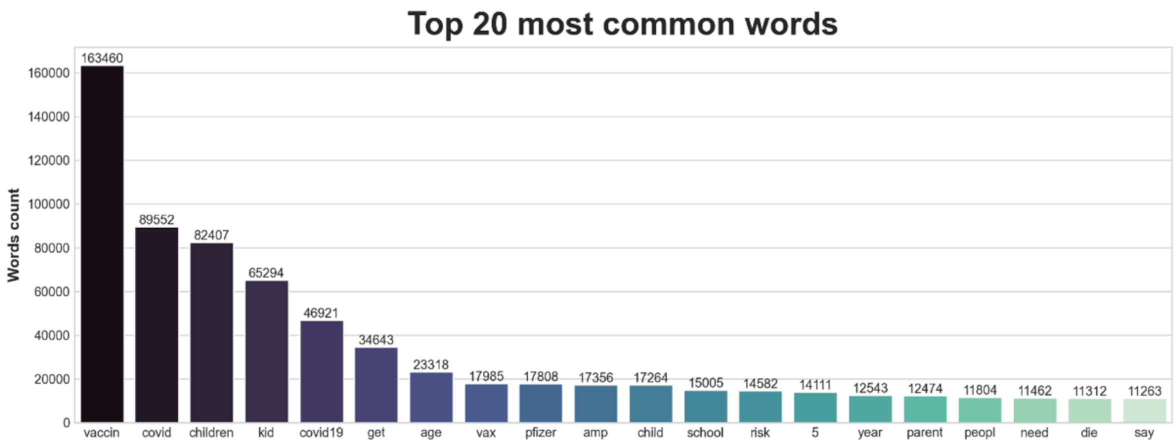


Fig. 70 – Top 20 most common words

3.16 DistilBERT.py file

A DistilBertModel (Ghisleni, 2022) was implemented using distilbert-base-uncased. Some code extract can be seen in figure 71.

```
class Bert_Classifier(nn.Module):
    def __init__(self, freeze_bert=False):
        super(Bert_Classifier, self).__init__()
        # Specify hidden size of BERT, hidden size of the classifier, and number of labels
        n_input = 768
        n_hidden = 50
        n_output = 5
        # Instantiate BERT model
        self.bert = DistilBertModel.from_pretrained('distilbert-base-uncased')

        # Add dense layers to perform the classification
        self.classifier = nn.Sequential(
            nn.Linear(n_input, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_output)
        )
        # Add possibility to freeze the BERT model
        # to avoid fine tuning BERT params (usually leads to worse results)
        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False

    def forward(self, input_ids, attention_mask):
        # Feed input data to BERT
        outputs = self.bert(input_ids=input_ids,
                            attention_mask=attention_mask)
```

```
def initialize_model(epochs=4):
    # Instantiate DistilBert Classifier
    bert_classifier = Bert_Classifier(freeze_bert=False)

    bert_classifier.to(device)

    # Set up optimizer
    optimizer = AdamW(bert_classifier.parameters(),
                      lr=5e-5,  # learning rate, set to default value
                      eps=1e-8  # decay, set to default value
                      )

    # Set up learning rate scheduler

    # Calculate total number of training steps
    total_steps = len(train_dataloader) * epochs

    # Define the scheduler
    scheduler = get_linear_schedule_with_warmup(optimizer,
                                                num_warmup_steps=0,  # Default value
                                                num_training_steps=total_steps)
    return bert_classifier, optimizer, scheduler

bert_preds = bert_predict(bert_classifier, test_dataloader)

print('Classification Report for DistilBERT :\n', classification_report(y_test, bert_preds,
                                                                          target_names=sentiments))

conf_matrix(y_test, bert_preds,' DistilBERT Sentiment Analysis\nConfusion Matrix', sentiments)
```

Fig. 71 – Code extract - DistilBERT.py

DistilBERT was ran with epoch set to 2 and produced an accuracy of 89.84 % as shown in figure 72.

```
Start training...

----------
Epoch : 1
----------
------------------------------------
BATCH NO. |  TRAIN LOSS  | ELAPSED (s)
------------------------------------
   100    |   1.040781   |  1000.09
   200    |   0.744533   |  2611.44
   300    |   0.616437   | 30163.82
   400    |   0.529280   | 11405.11
   500    |   0.505124   |  1087.66
   600    |   0.461116   |  2244.63
   700    |   0.448365   |  1437.95
   800    |   0.428590   |  1010.52
   853    |   0.384778   |  565.01
-------------------------------------------------------------
AVG TRAIN LOSS |  VAL LOSS  | VAL ACCURACY (%) | ELAPSED (s)
-------------------------------------------------------------
   0.584141    |  0.375960  |       86.18      | 51951.44
-------------------------------------------------------------
```

```
----------
Epoch : 2
----------
--------------------------------------
BATCH NO. |  TRAIN LOSS  | ELAPSED (s)
--------------------------------------
|  100    |   0.288237   | 10988.90
   200    |   0.307454   |  1661.79
   300    |   0.287987   |  1294.69
   400    |   0.272937   |  3530.13
   500    |   0.254745   |  1284.65
   600    |   0.285865   | 12101.48
   700    |   0.262227   |  5761.45
   800    |   0.259320   | 28261.52
   853    |   0.270599   |  2173.93
-------------------------------------------------------------
AVG TRAIN LOSS |  VAL LOSS  | VAL ACCURACY (%) | ELAPSED (s)
-------------------------------------------------------------
   0.276940    |  0.314839  |       89.84      | 68293.63
-------------------------------------------------------------
```

Fig. 72 – DistilBERT ran with epoch set to 2

3.17 CovidKidsVaxWithCasesInfoModelsAllYears.py file

This Python file combines a sentiment analysis and prediction with the number of cases, total boosters and new vaccinations. SPSS was used to investigate correlations. The overall analysis reveals very weak correlations as shown in fig. 73.

Correlations

		sentiment_score	total_cases	new_cases	total_deaths	new_deaths	reproduction_rate	total_vaccinations	people_vaccinated	people_fully_vaccinated	total_boosters	new_vaccinations
sentiment_score	Pearson Correlation	1	.016**	.008**	.003	-.007**	.000	.006*	-.001	.005*	.024**	-.016**
	Sig. (2-tailed)		.000	.003	.214	.006	.870	.019	.807	.044	.000	.000
	N	153096	152482	152482	152482	152482	152023	144681	144681	143984	142875	143104
total_cases	Pearson Correlation	.016**	1	.389**	.883**	-.521**	-.096**	.907**	.813**	.861**	.959**	-.279**
	Sig. (2-tailed)	.000		.000	.000	.000	.000	.000	.000	.000	.000	.000
	N	152482	152482	152482	152482	152482	152023	144681	144681	143984	142875	143104
new_cases	Pearson Correlation	.008**	.389**	1	.350**	.180**	.223**	.401**	.343**	.391**	.401**	.035**
	Sig. (2-tailed)	.003	.000		.000	.000	.000	.000	.000	.000	.000	.000
	N	152482	152482	152482	152482	152482	152023	144681	144681	143984	142875	143104
total_deaths	Pearson Correlation	.003	.883**	.350**	1	-.394**	-.028**	.980**	.974**	.963**	.736**	.097**
	Sig. (2-tailed)	.214	.000	.000		.000	.000	.000	.000	.000	.000	.000
	N	152482	152482	152482	152482	152482	152023	144681	144681	143984	142875	143104
new_deaths	Pearson Correlation	-.007**	-.521**	.180**	-.394**	1	-.199**	-.659**	-.637**	-.634**	-.577**	.231**
	Sig. (2-tailed)	.006	.000	.000	.000		.000	.000	.000	.000	.000	.000
	N	152482	152482	152482	152482	152482	152023	144681	144681	143984	142875	143104
reproduction_rate	Pearson Correlation	.000	-.096**	.223**	-.028**	-.199**	1	.143**	.157**	.155**	-.011**	.285**
	Sig. (2-tailed)	.870	.000	.000	.000	.000		.000	.000	.000	.000	.000
	N	152023	152023	152023	152023	152023	152023	144222	144222	143525	142416	143104
total_vaccinations	Pearson Correlation	.006*	.907**	.401**	.980**	-.659**	.143**	1	.978**	.988**	.781**	-.017**
	Sig. (2-tailed)	.019	.000	.000	.000	.000	.000		.000	.000	.000	.000
	N	144681	144681	144681	144681	144681	144222	144681	144681	143984	142875	142996
people_vaccinated	Pearson Correlation	-.001	.813**	.343**	.974**	-.637**	.157**	.978**	1	.981**	.637**	.101**
	Sig. (2-tailed)	.807	.000	.000	.000	.000	.000	.000		.000	.000	.000
	N	144681	144681	144681	144681	144681	144222	144681	144681	143984	142875	142996
people_fully_vaccinated	Pearson Correlation	.005*	.861**	.391**	.963**	-.634**	.155**	.988**	.981**	1	.727**	-.053**
	Sig. (2-tailed)	.044	.000	.000	.000	.000	.000	.000	.000		.000	.000
	N	143984	143984	143984	143984	143984	143525	143984	143984	143984	142875	142299
total_boosters	Pearson Correlation	.024**	.959**	.401**	.736**	-.577**	-.011**	.781**	.637**	.727**	1	-.494**
	Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000	.000	.000		.000
	N	142875	142875	142875	142875	142875	142416	142875	142875	142875	142875	141190
new_vaccinations	Pearson Correlation	-.016**	-.279**	.035**	.097**	.231**	.285**	-.017**	.101**	-.053**	-.494**	1
	Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000	.000	.000	.000	
	N	143104	143104	143104	143104	143104	143104	142996	142996	142299	141190	143104

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Fig. 73– Correlation in SPSS

The following models were implemented with the target set to sentiment_score and predictors to
new_vaccinations, total_boosters, total_cases: logistic regression, random forest, extreme gradient boost, K-Neighbors classifier, decision tree, support vector machine and Naïve Bayes. The model with the highest accuracy is Random Forest with 41.63% as shown in figure 74.

	Model	Test_Accuracy	Train_Accuracy1
0	Logistic Regression	33.236447	33.194258
1	Random Forest	41.629654	42.729188
2	Extreme Gradient Boost	41.345526	42.239296
3	K-Nearest Neighbour	38.354017	39.168490
4	Decision Tree	41.577400	42.778177
5	Support Vector Machine	35.447420	35.737614

Fig.74 – Models results with predictors set to new_vaccinations, total_boosters, total_cases

Another set of tests were run with the target being set to sentiment_score and predictors being total_cases and total_deaths. The model with the highest accuracy was Extreme Gradient Boost with 41.74% as shown in figure 75.

	Model	Test_Accuracy	Train_Accuracy1
0	Logistic Regression	34.265186	33.986250
1	Random Forest	41.629654	42.729188
2	Extreme Gradient Boost	41.747224	42.609981
3	K-Nearest Neighbour	37.864141	38.534080
4	Decision Tree	41.577400	42.778177
5	Support Vector Machine	35.499673	35.250171

Fig. 75– Models results with predictors set to total_cases and total_deaths

Another set of tests were run with the target being set to sentiment_score and predictor set to reproduction_rate. The model with the highest accuracy was Extreme Gradient Boost with 36.73% (figure 76).

	Model	Test_Accuracy	Train_Accuracy1
0	Logistic Regression	33.892880	34.190372
1	Random Forest	36.685173	37.093798
2	Extreme Gradient Boost	36.727629	37.077468
3	K-Nearest Neighbour	34.784455	34.889284
4	Decision Tree	36.708034	37.097064
5	Support Vector Machine	34.595036	34.782325

Fig. 76 – Models results with predictors set to reproduction _rate

Some code snippets from this file can be seen in figure 77.

```python
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Random Forest','Extreme Gradient Boost',
                    'K-Nearest Neighbour','Decision Tree','Support Vector Machine'], 'Test_Accuracy': [lr_acc_s
                   rf_acc_score*100,xgb_acc_score*100,knn_acc_score*100,dt_acc_score*100,svc_acc_score*100], 'T
                   rf_acc_score1*100,xgb_acc_score1*100,knn_acc_score1*100,dt_acc_score1*100,svc_acc_score1*100
print(model_ev)

model_ev.plot.bar()

m2 = 'Naive Bayes'
nb = GaussianNB()
nb.fit(X_train,y_train)
nbpred = nb.predict(X_test)
nbpred1 = nb.predict(X_train)
nb_conf_matrix = confusion_matrix(y_test, nbpred)
nb_conf_matrix1 = confusion_matrix(y_train, nbpred1)
nb_acc_score = accuracy_score(y_test, nbpred)
nb_acc_score1 = accuracy_score(y_train, nbpred1)
print("confusion matrix")
print(nb_conf_matrix)
print(nb_conf_matrix1)
print("\n")
print("Accuracy of Naive Bayes model:",nb_acc_score*100,'\n')
```

```python
subset = ['sentiment_score','total_cases','total_boosters','new_vaccinations']
tweets_df_r2 = df_basemodel.loc[:, subset]
tweets_df_r2.head()
tweets_df_r2=tweets_df_r2.fillna(0)
print(tweets_df_r2.sentiment_score.value_counts())
data = X = tweets_df_r2.iloc[:,1:]
target = tweets_df_r2.sentiment_score

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size = 0.2, random_state = 0)
sc = StandardScaler()
sc.fit(X_train, X_test)
```

```python
#Run model with following predictors:
df_basemodel_copy.drop(columns=['new_cases'], inplace=True)
df_basemodel_copy.drop(columns=['total_deaths'], inplace=True)
df_basemodel_copy.drop(columns=['new_deaths'], inplace=True)
df_basemodel_copy.drop(columns=['reproduction_rate'], inplace=True)
df_basemodel_copy.drop(columns=['total_vaccinations'], inplace=True)
df_basemodel_copy.drop(columns=['people_vaccinated'], inplace=True)
df_basemodel_copy.drop(columns=['people_fully_vaccinated'], inplace=True)
df_basemodel_copy.drop(columns=['total_boosters'], inplace=True)
df_basemodel_copy.drop(columns=['new_vaccinations'], inplace=True)
modelPreparation(df_basemodel_copy)
```

Fig. 77 – Code extract – CovidKidsVaxWithCasesInfoModelsAllYears.py

References

Classify Text with BERT. (2022). Retrieved from TensorFlow: https://www.tensorflow.org/text/tutorials/classify_text_with_bert

Dua, S. (2021). *Sentiment Analysis*. Retrieved from github: https://github.com/sejaldua/covid19-vaccy-tweets-NLP/blob/main/workbook.ipynb

Ghisleni, G. (2022). *DistilBERT sentiment analysis*. Retrieved from Machine Learning: https://gabrieleghisleni.github.io/DeepLearning-Lab/SentimentAnalysis-DistilBERT/

Justin, L. (2020). *How to do Sentiment Analysis with Deep Learning (LSTM Keras)*. Retrieved from Just into Data: https://www.justintodata.com/sentiment-analysis-with-deep-learning-lstm-keras-python/