Problem statement:

The world is going through COVID-19 pandemic caused due to Novel Coronavirus and people are mostly in their homes across the world due to lockdown and they are searching for various things related to Coronavirus on internet using popular search engines such as Chrome, Bing, Yahoo, DuckDuckGo etc.

Now, the problem task is to attempt to explore the intent of the people by just using the queries (searches) in the last four months Jan to April 2020 and also build a predicting model which predicts the Country of origin from where the search query was issued , as search queries made by people can really help in understanding what's going in people's mind during this pandemic and exploring the queries at global level as well as state level granularity will enable state and central authorities to take appropriate action and eventually helping out and benefitting the Citizens.

Motivation:

As a Computer Engineering Student and my deep interested in Machine learning and Deep learning including NLP tasks, I have my inner passion to help out researchers / scientists working day and night for helping us by providing some technological solution using my knowledge.
I have always believed AI can bring great revolutionary ideas in field of healthcare.

Project Objectives:

The major goals of this project are as follows:

- Dataset collection based on above problem statement
- Perform Exploratory Data Analysis of the dataset collected
- Perform sanity checks and other analysis for preprocessing
- Build and create ensembles of classifiers using supervised / unsupervised algorithms and perform data modelling
- Cross validation against chosen metrics of interest
- Finally, deployment and hosting of our trained model for real time predictions (inference)

Dataset Collection:

After searching on various platforms, I found the dataset relevant for this problem and currently only this dataset is actually available for this problem and Microsoft Bing Team has been very generous to provide the dataset on their GitHub.

Dataset Link: https://github.com/microsoft/BingCoronavirusQuerySet

Some info about dataset:

- Data range: 2020-01-01 to 2020-04-30
- All the private data has already been removed by the Bing Team.
- Only searches that were issued many times by multiple users were included.
- Dataset includes queries from all over the world that had an intent related to the Coronavirus or Covid-19.

Dataset schema:

QueriesByCountry_DateRange.tsv : A tab separated text file that contains queries with Coronavirus intent by Country.

Date	string, Date on which the query was issued.
Query	string, The actual search query issued by user(s).
IsImplicitIntent	bool, True if query did not mention covid or coronavirus or sarsncov2 (e.g, "Shelter in place"). False otherwise.
Country	string, Country from where the query was issued.
PopularityScore	int, Value between 1 and 100 inclusive. 1 indicates least popular query on the day/Country with Coronavirus intent, and 100 indicates the most popular query for the same Country on the same day.

Visualizations and EDA:

- Since the data has been divided over months, so some visualizations have been performed for individual month data and others for complete data (combined).

Quick look over the dataset for each month:

January -

shape of dataframe : (33901, 5)

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-01-01	webasto	True	Germany	1
1	2020-01-01	coronavirus	False	United States	100
2	2020-01-01	p2 masks	True	Australia	100
3	2020-01-01	china virus	True	United States	15
4	2020-01-01	p2 masks australia	True	Australia	1

February –

shape of dataframe : (132979, 5)

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-02-01	coronavirus cover up	False	United States	1
1	2020-02-01	corona virus cdc	False	United States	1
2	2020-02-01	coronavirus	False	Pakistan	100
3	2020-02-01	lysol corona virus	False	United States	1
4	2020-02-01	coronavirus ottawa	False	Canada	3

March –

shape of dataframe : (993787, 5)

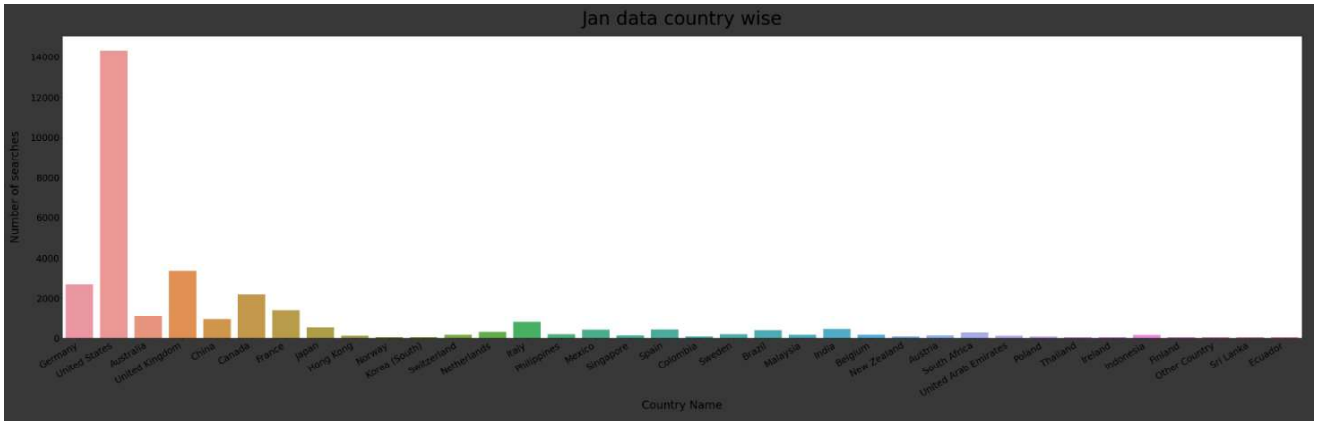
	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-03-01	gouvernement.fr coronavirus	False	France	1
1	2020-03-01	butlins	True	United Kingdom	4
2	2020-03-01	kachelmann	True	Germany	1
3	2020-03-01	berkshire coronavirus	False	United Kingdom	1
4	2020-03-01	oberstdorf	True	Germany	1

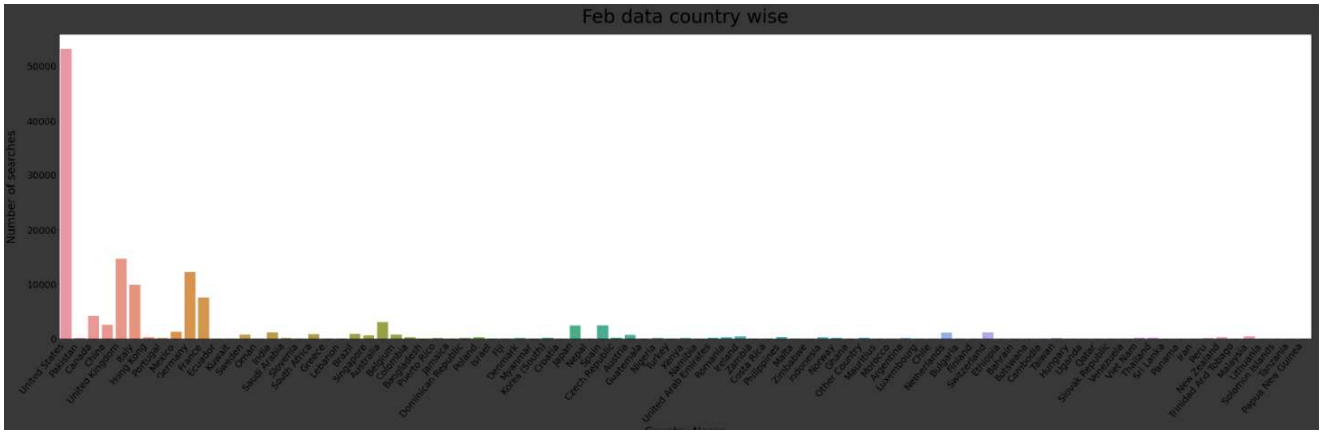
April—

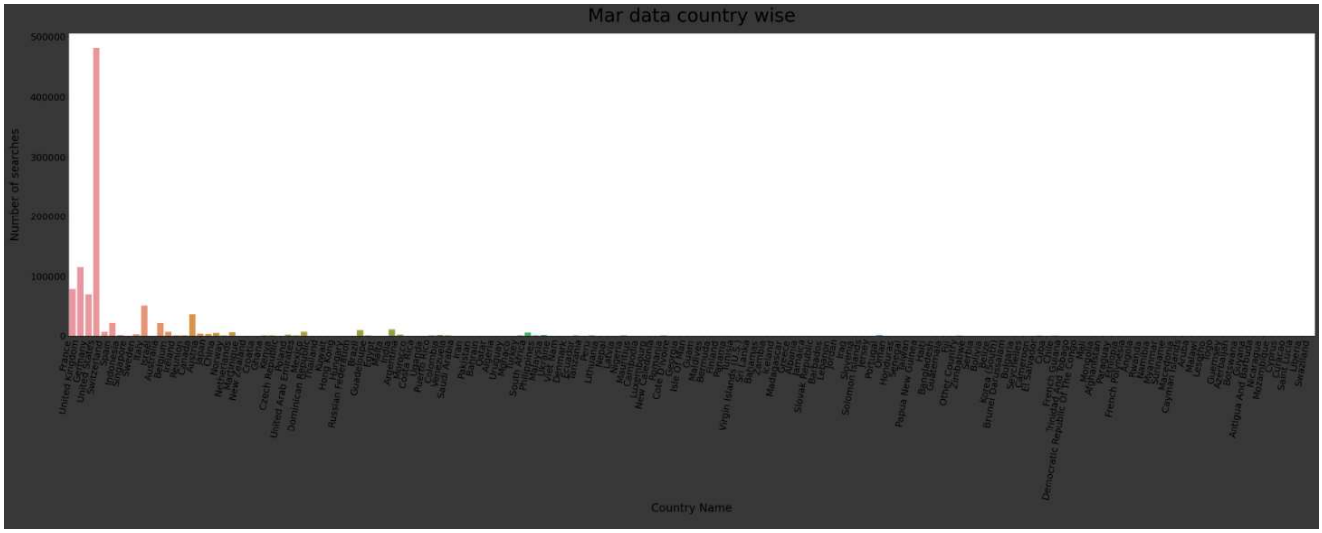
shape of dataframe : (675038, 5)

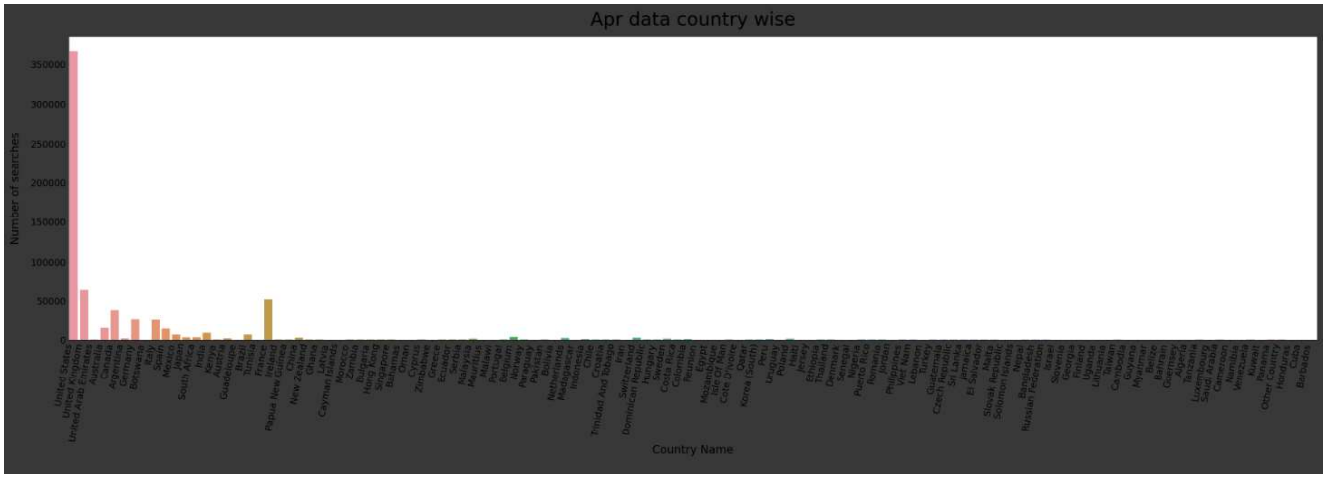
	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-04-01	is influenza a coronavirus	False	United States	1
1	2020-04-01	pennsylvania coronavirus stay at home	False	United States	1
2	2020-04-01	number of coronavirus tests by state	False	United States	1
3	2020-04-01	https://www.gov.uk/coronavirus-extremely-vulne...	False	United Kingdom	1
4	2020-04-01	cdc coronavirus update new york state	False	United States	1

How search queries were distributed over the months country wise:

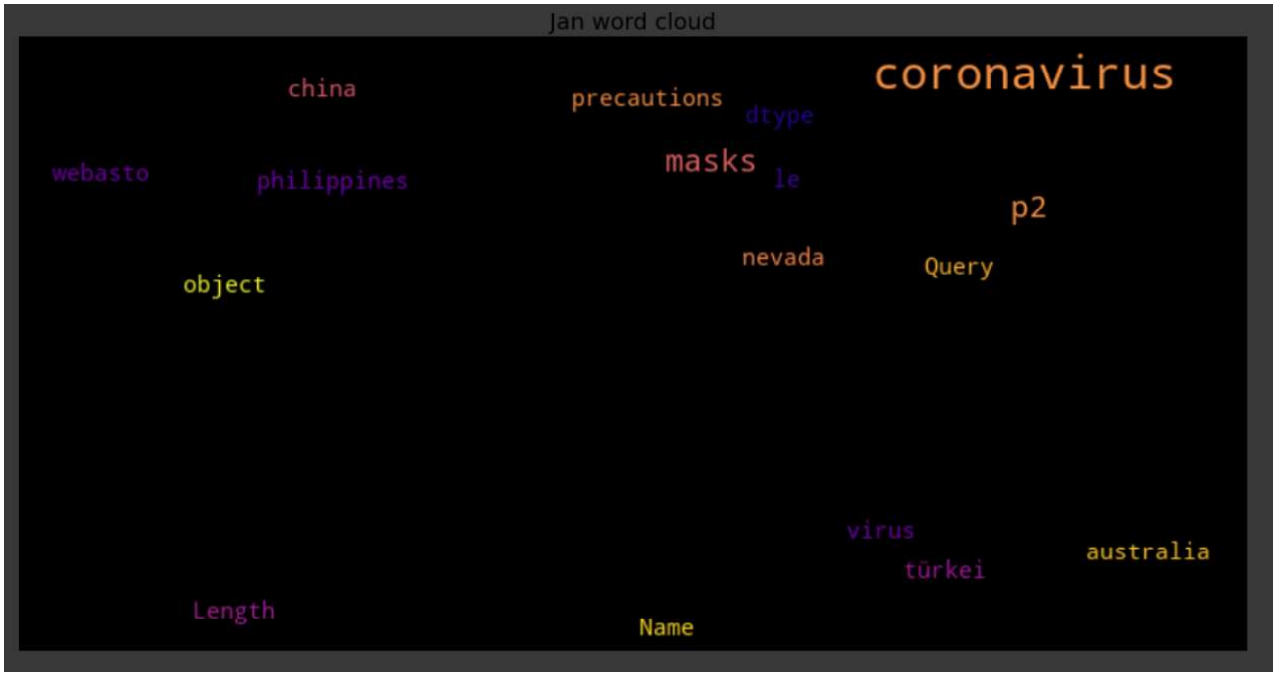


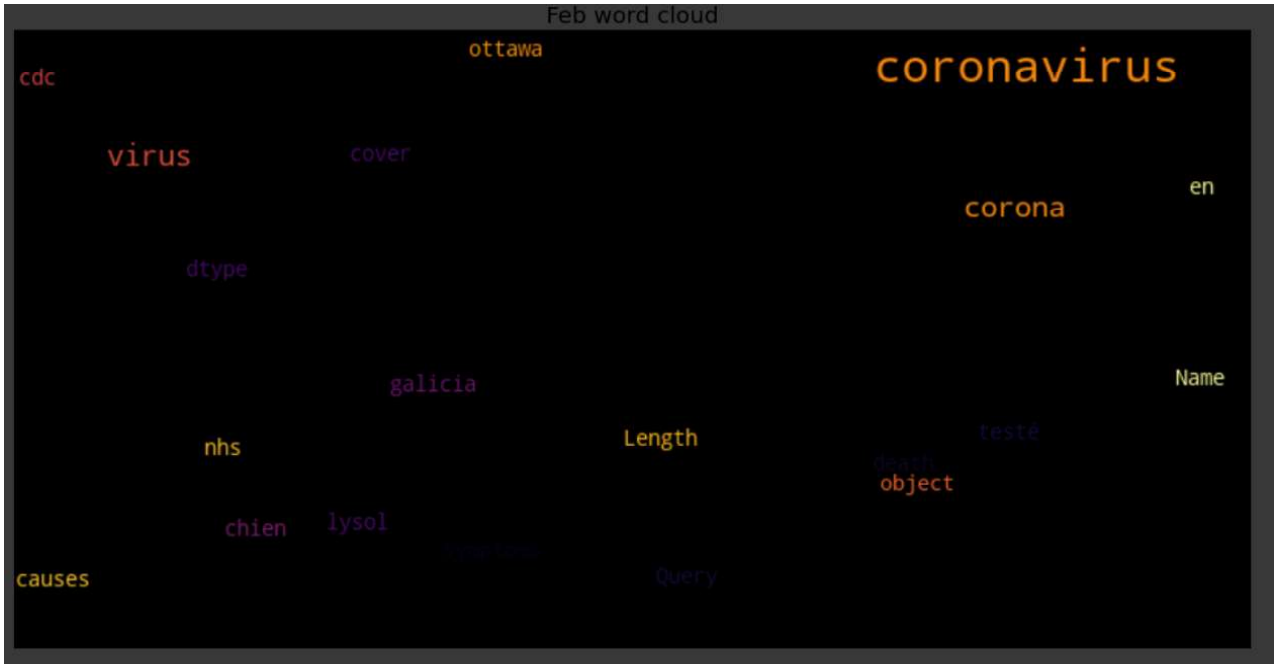


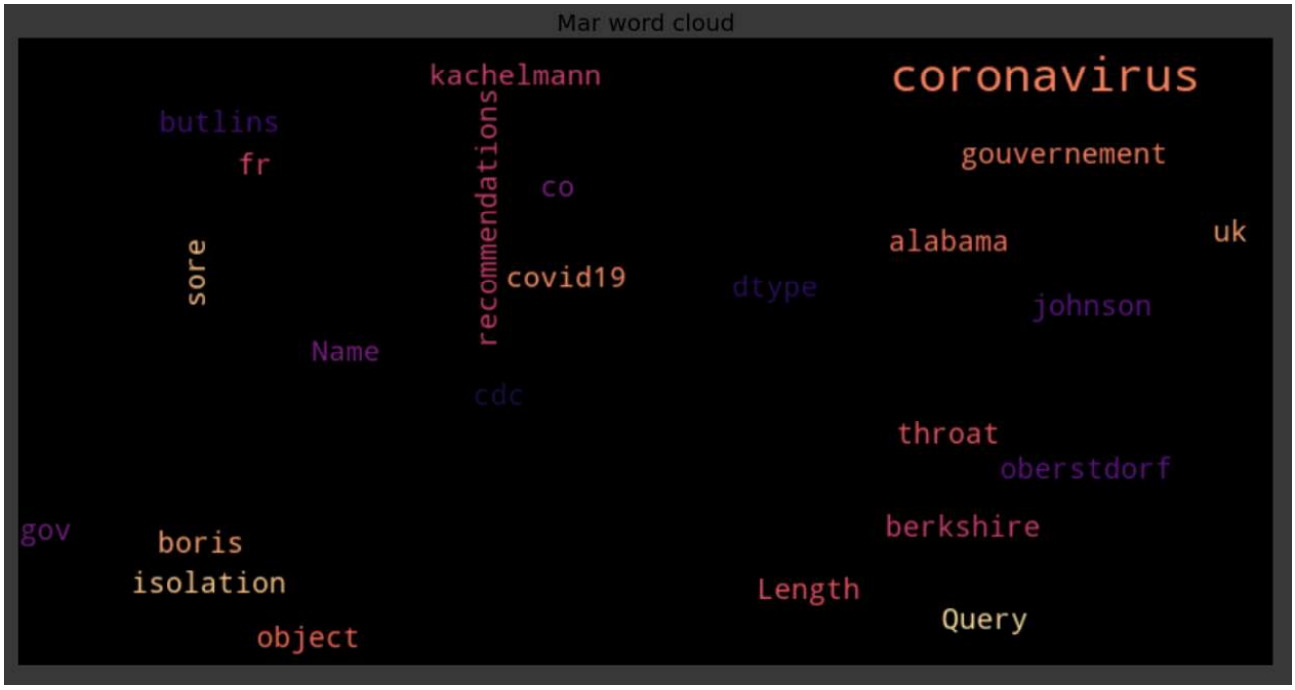


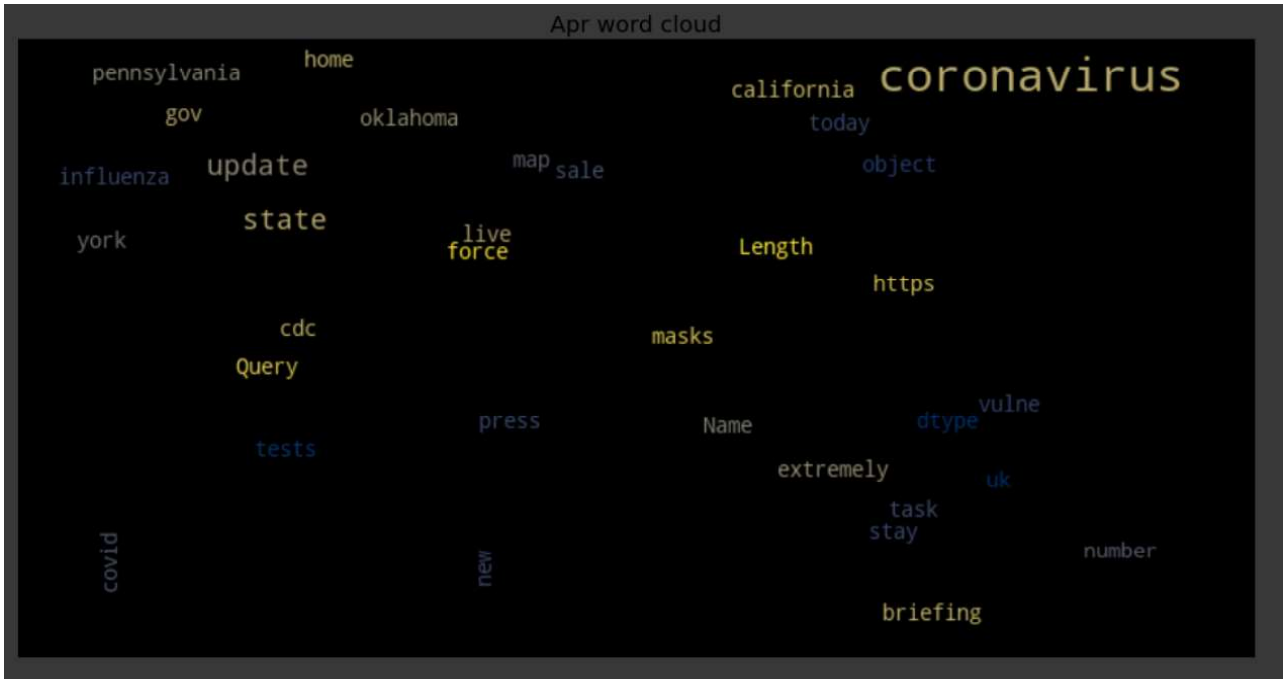


Now, we want to know people are searching what exact terms and how much?

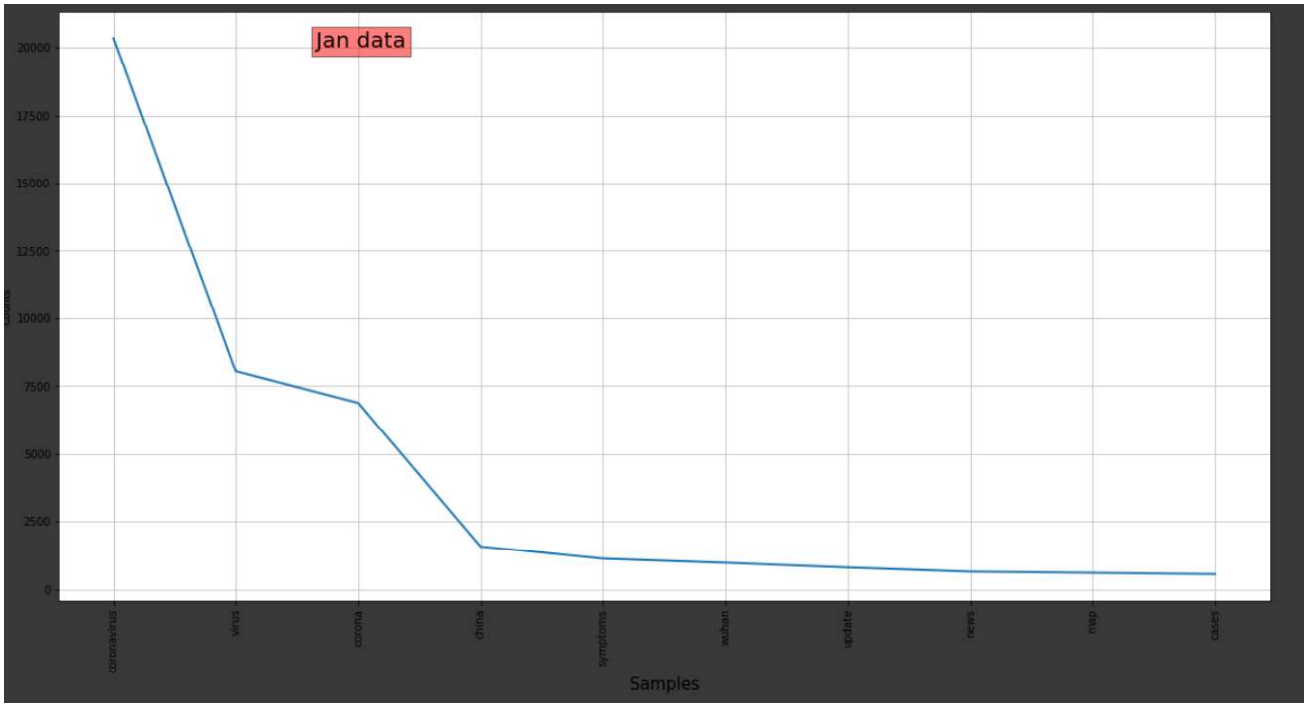


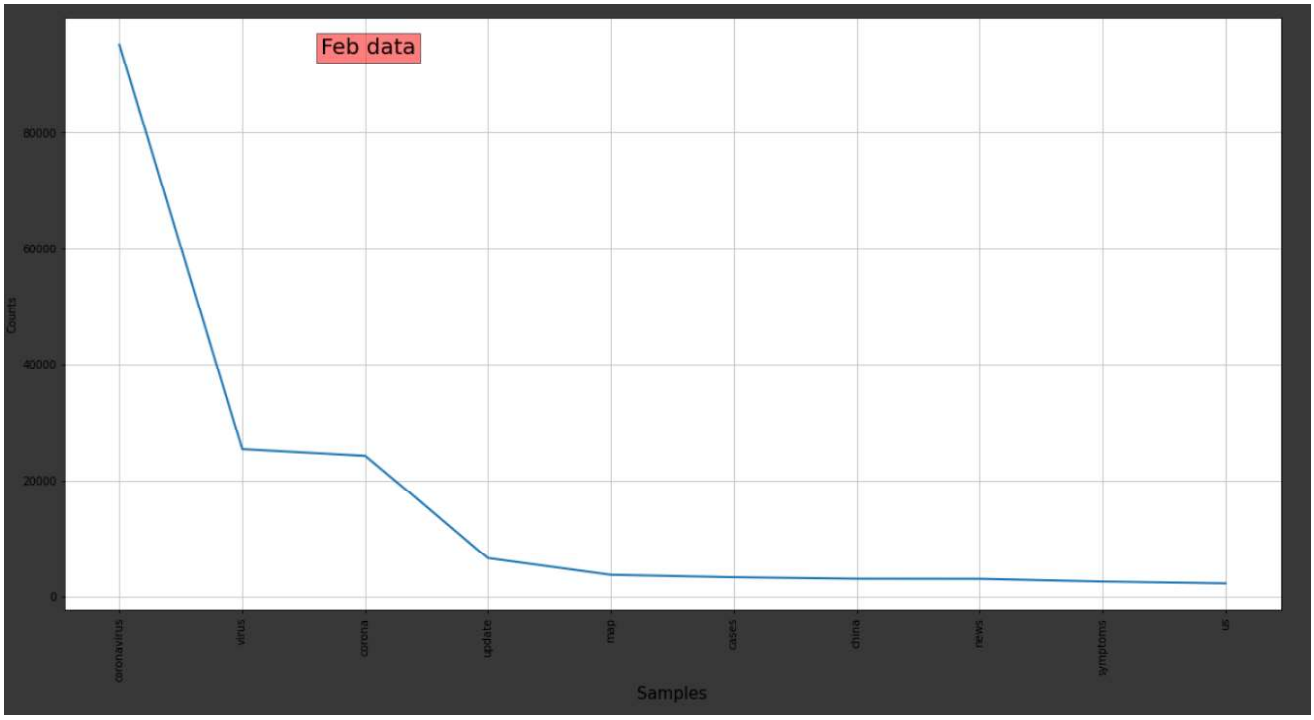

Feb word cloud
ottawa
cdc
coronavirus
virus
cover
en
corona
dtype
galicia
Name
nhs
Length
teste
object
chien
lysol
causes
Query

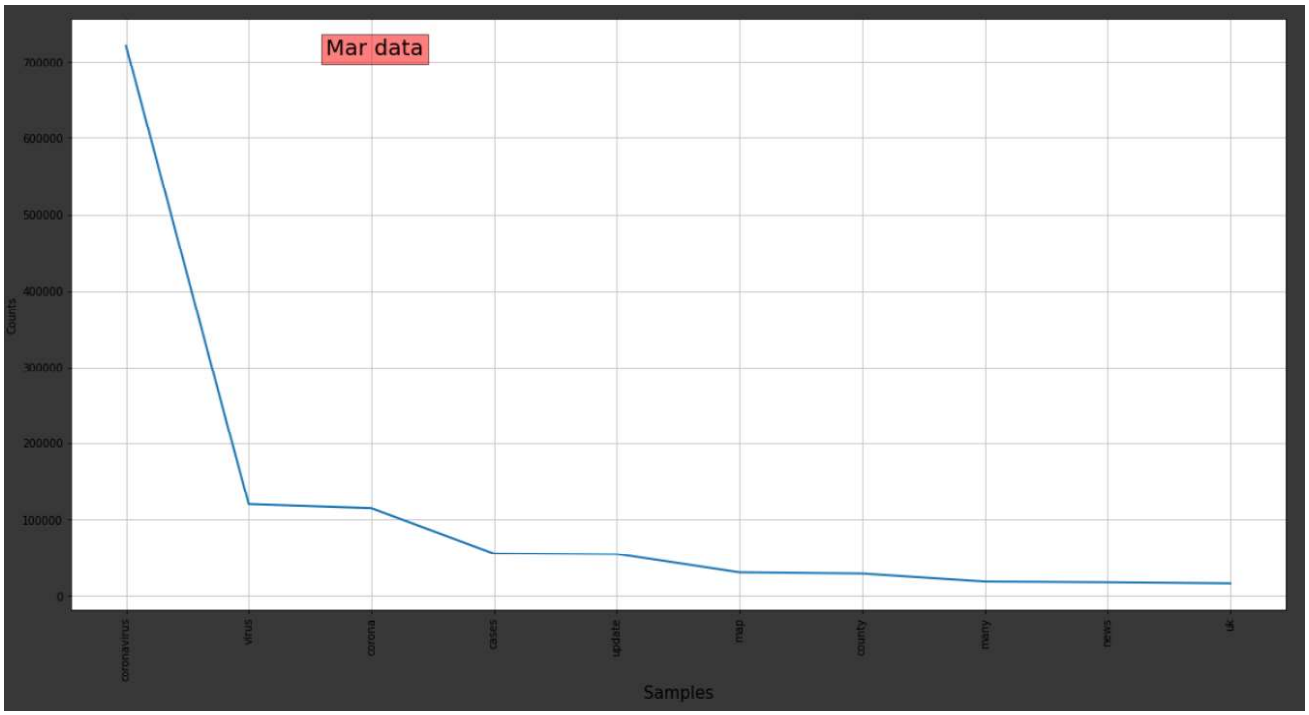

Mar word cloud
kachelmann
coronavirus
butlins
fr
recommendations
co
gouvernement
alabama
uk
sore
covid19
dtype
johnson
Name
cdc
throat
oberstdorf
gov
boris
berkshire
isolation
Length
Query
object

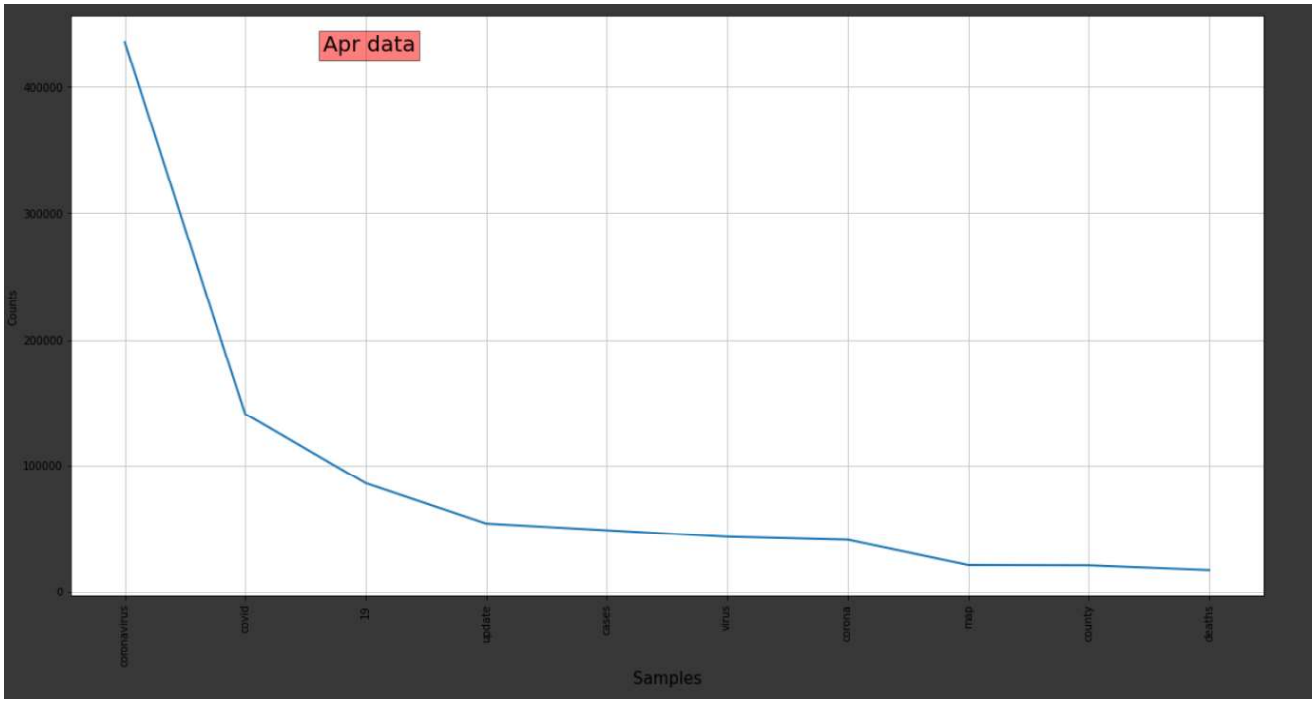


Checking top 10 keywords searched for during different months using frequency distribution graph.

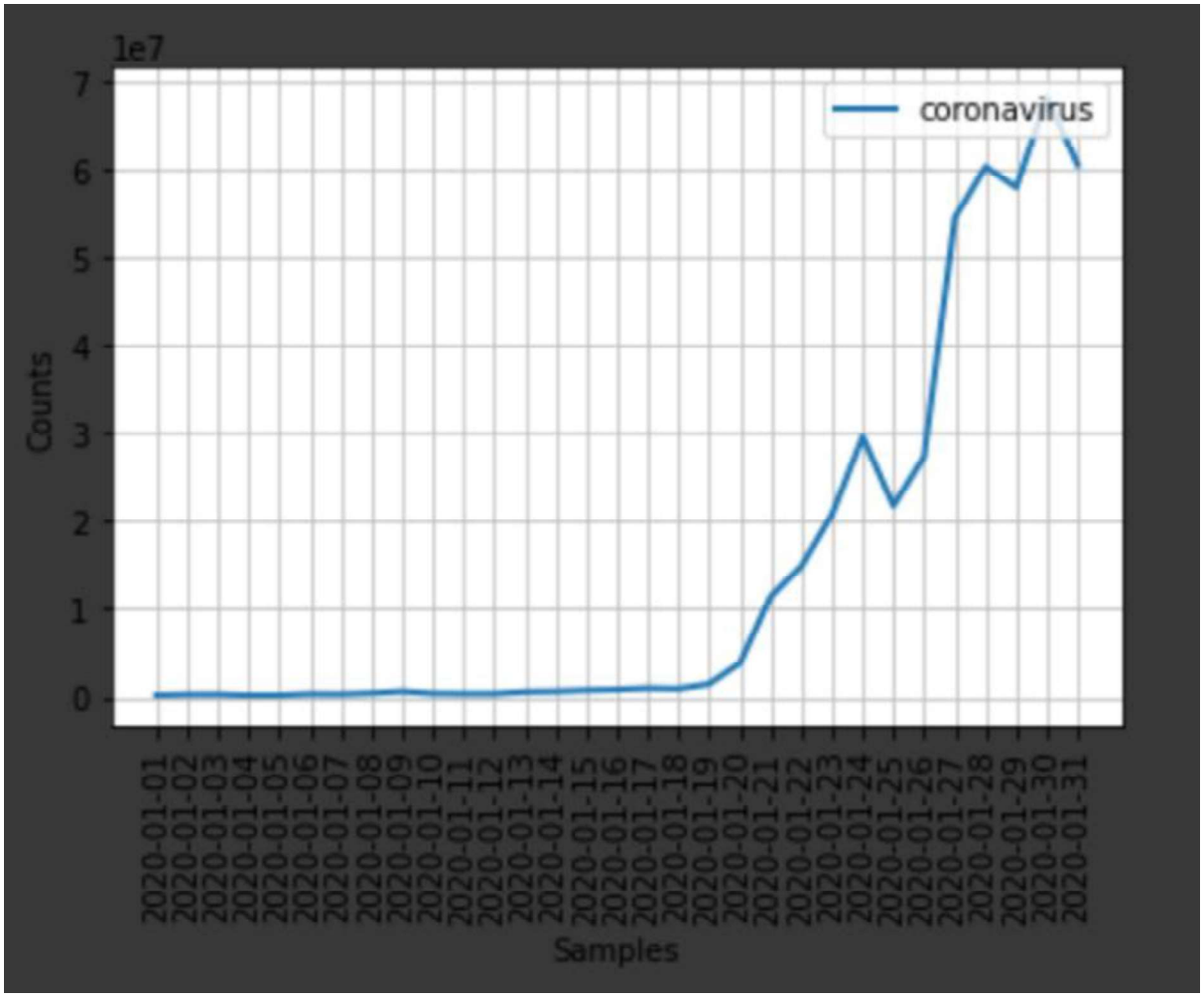

Jan data
20000
17500
15000
12500
10000
7500
5000
2500
0
coronavirus
virus
corona
china
symptoms
wuhan
update
news
map
cases
Samples


Feb data
80000
60000
40000
20000
0
Counts
coronavirus
virus
corona
update
map
cases
china
news
symptoms
us
Samples

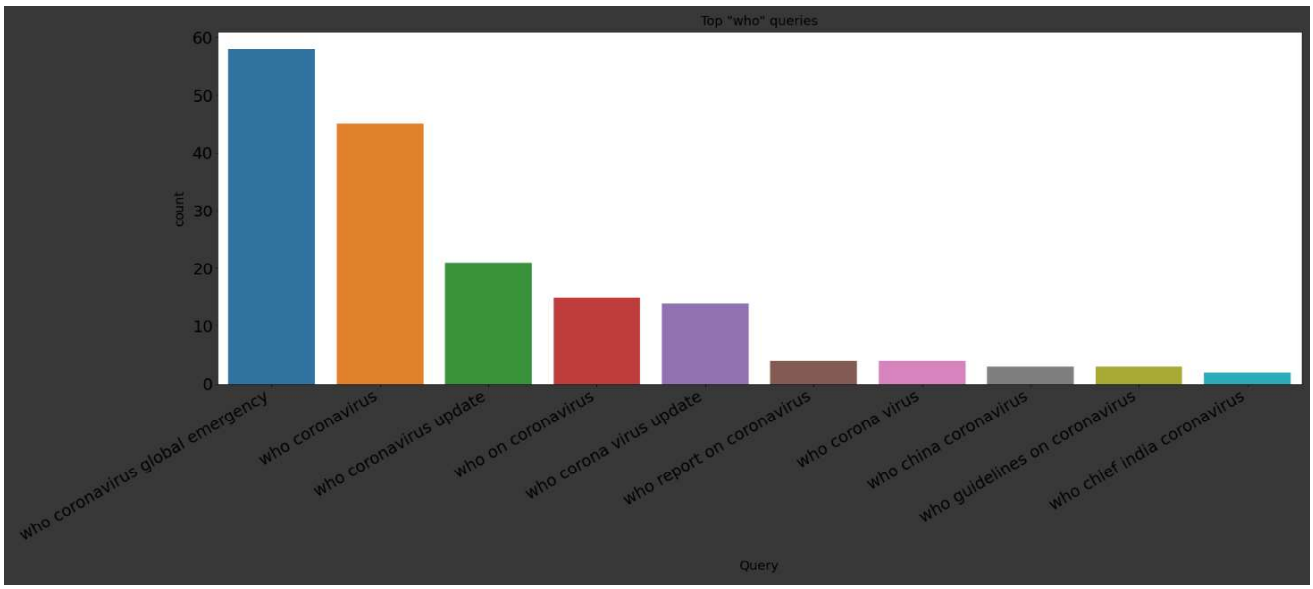

Mar data
Counts
700000
600000
500000
400000
300000
200000
100000
0
coronavirus
virus
corona
cases
update
map
county
many
news
uk
Samples

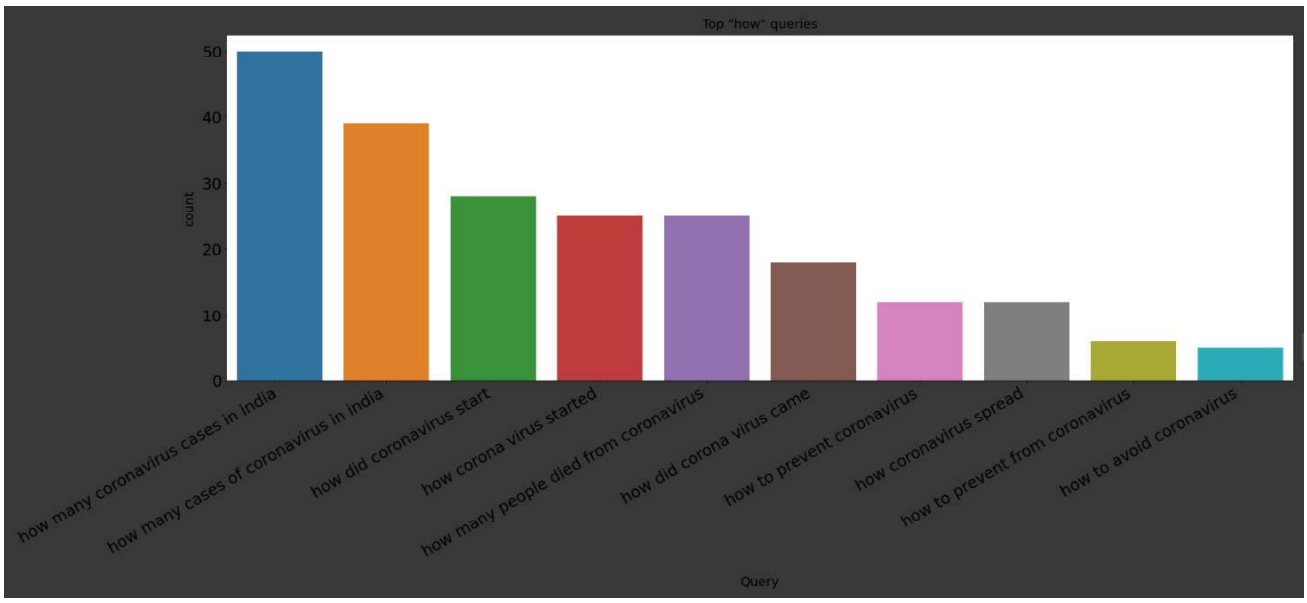

Apr data
Counts
400000
300000
200000
100000
0
coronavirus
covid
19
update
cases
virus
corona
map
county
deaths
Samples

We can also see how each of these most frequently occurring keywords varies over time to time in terms of month.

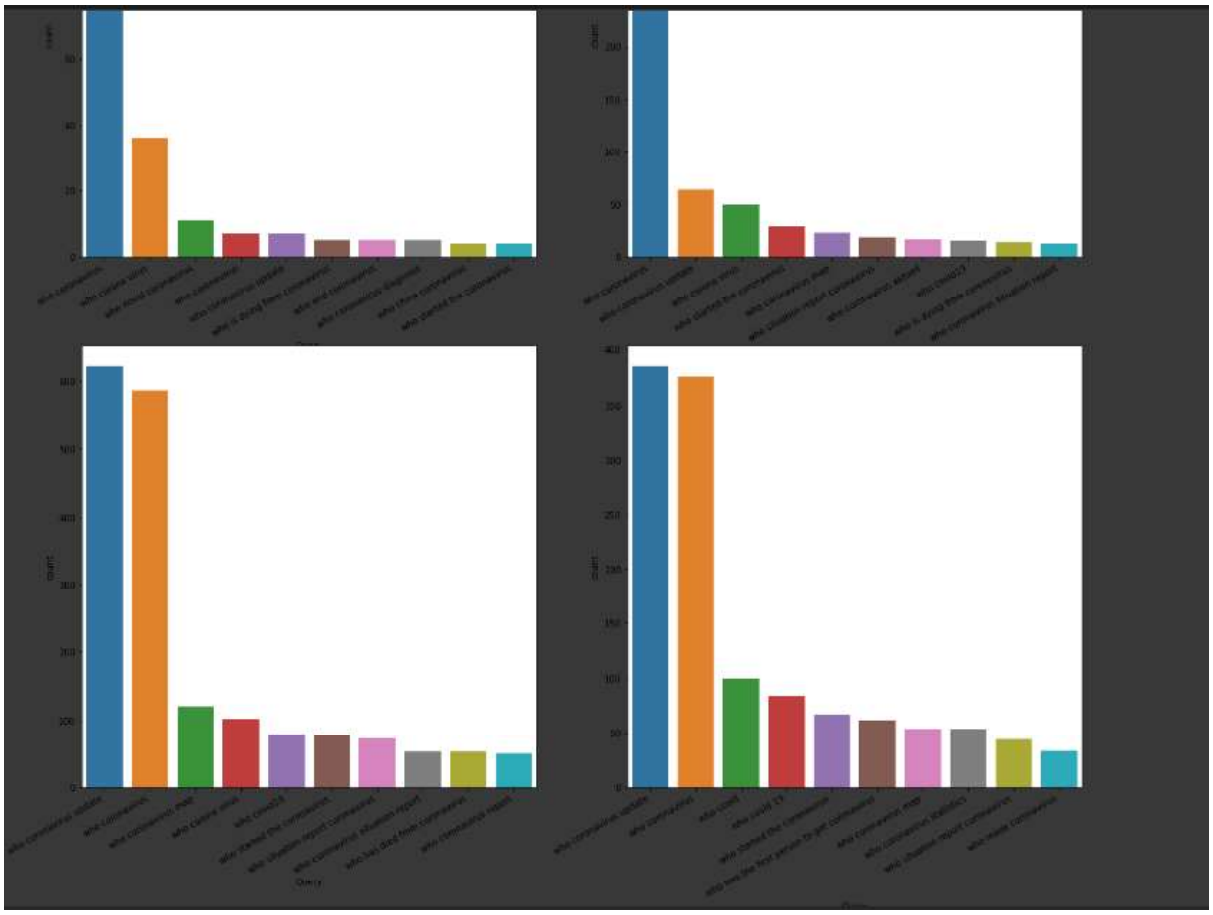


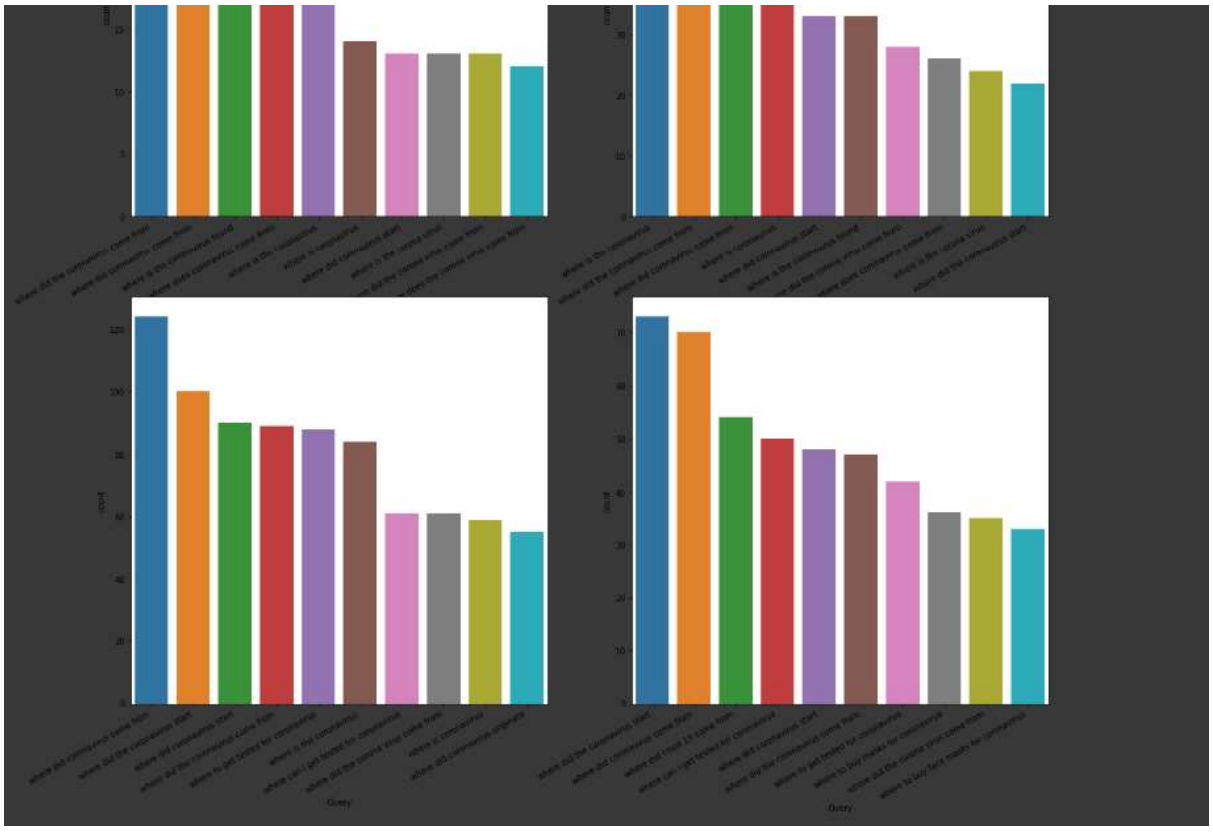
Checking top queries searched for special keywords like queries starting with "who", "where", "how" etc... to better understand intent of people searching Internet during this pandemic.

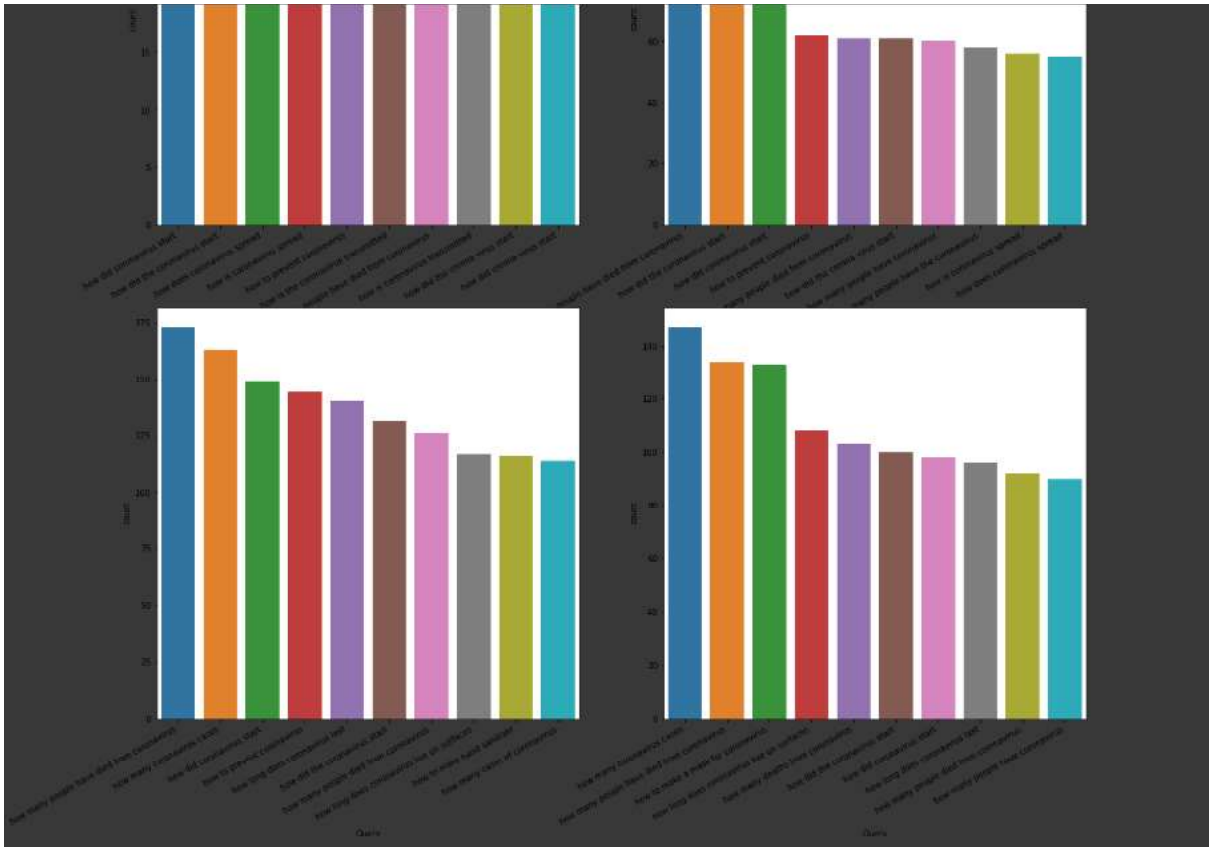


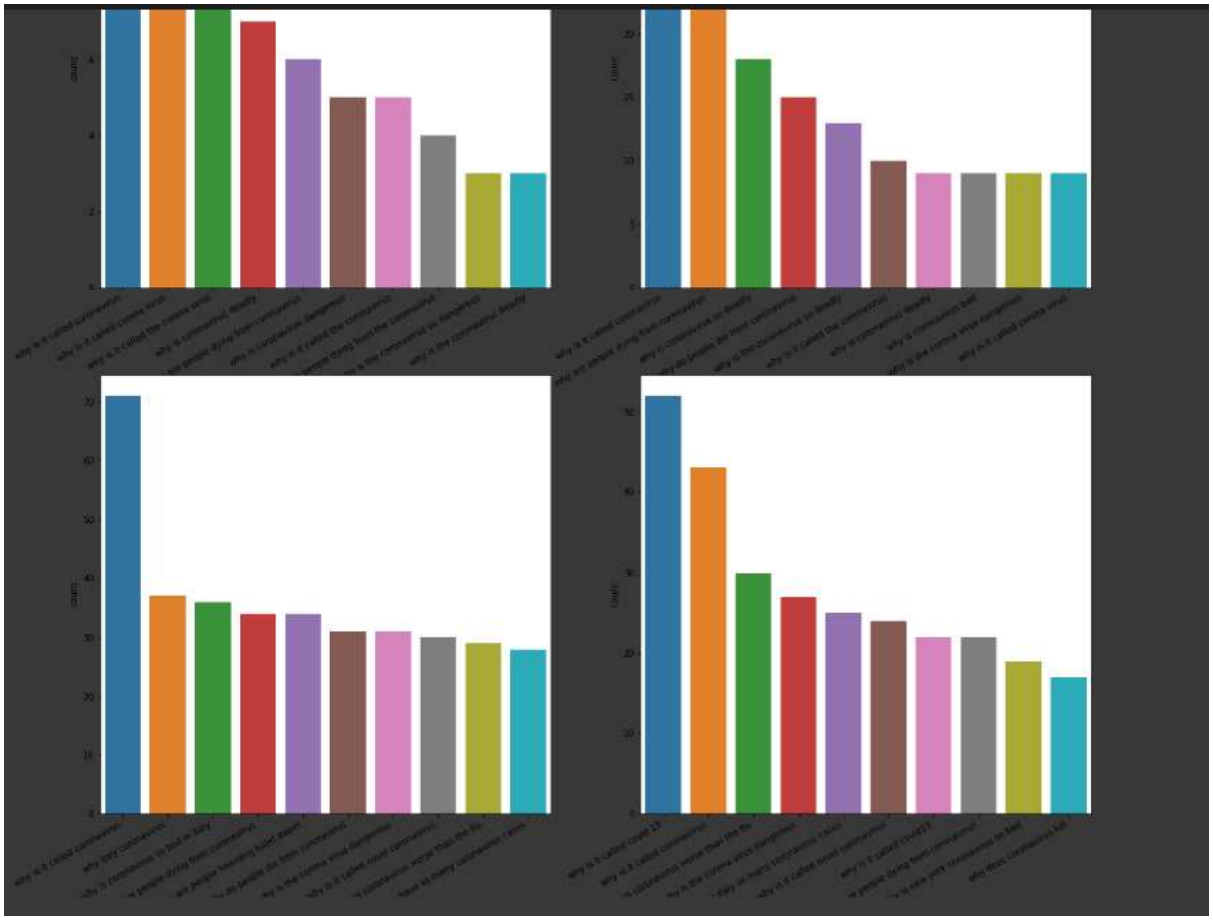


Comparison across months in one plot –



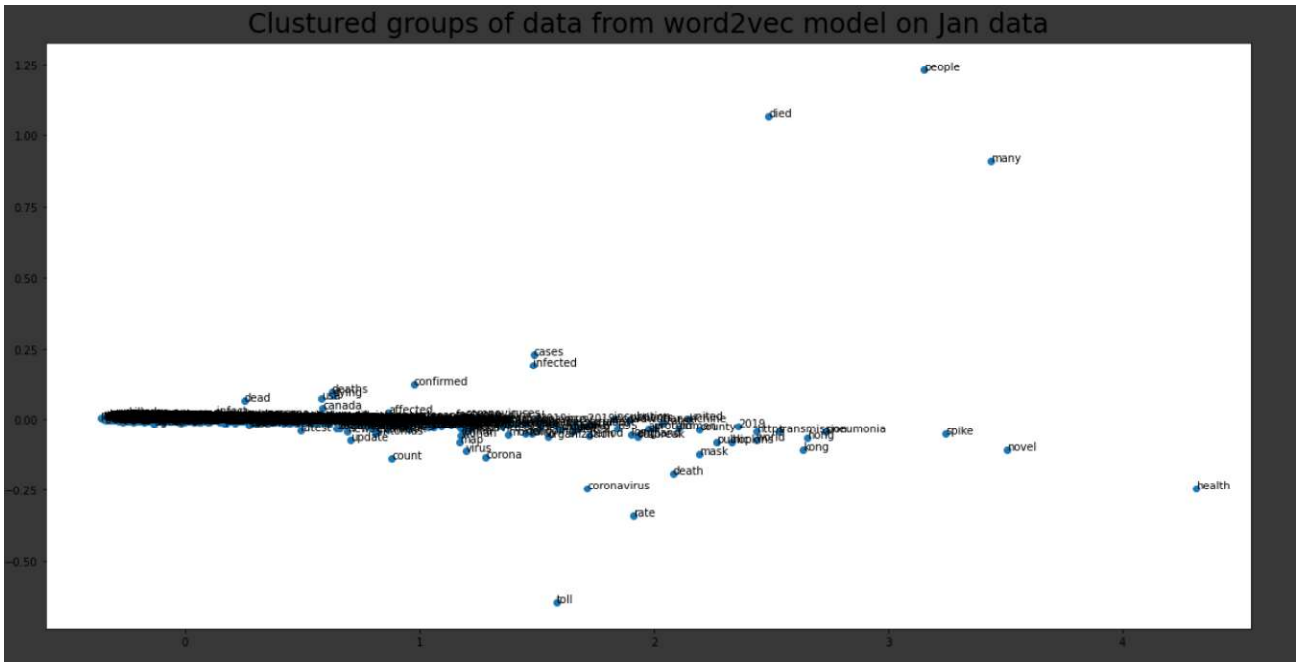


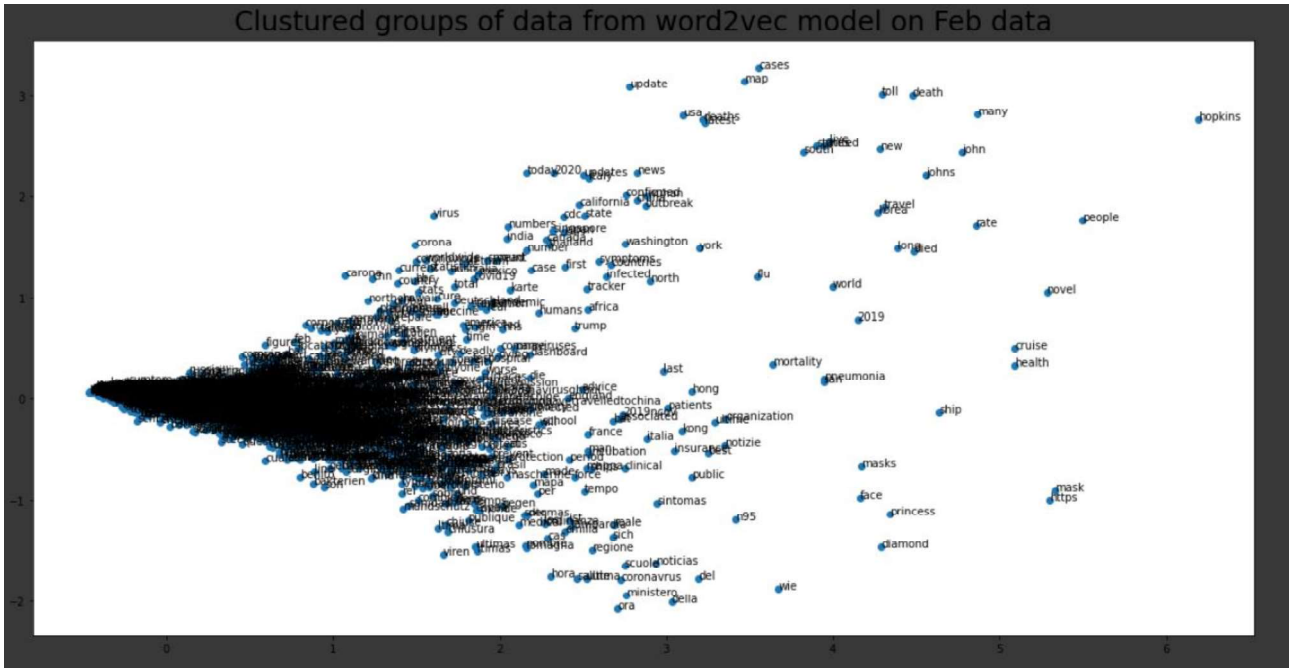


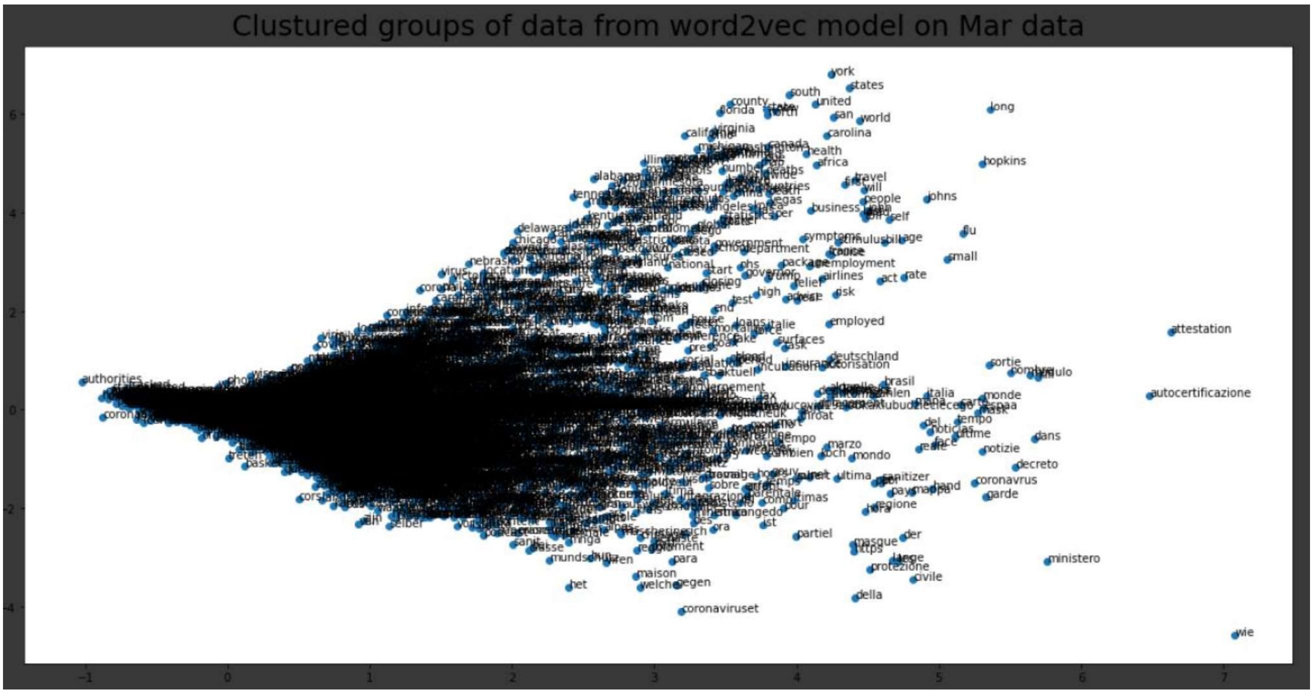


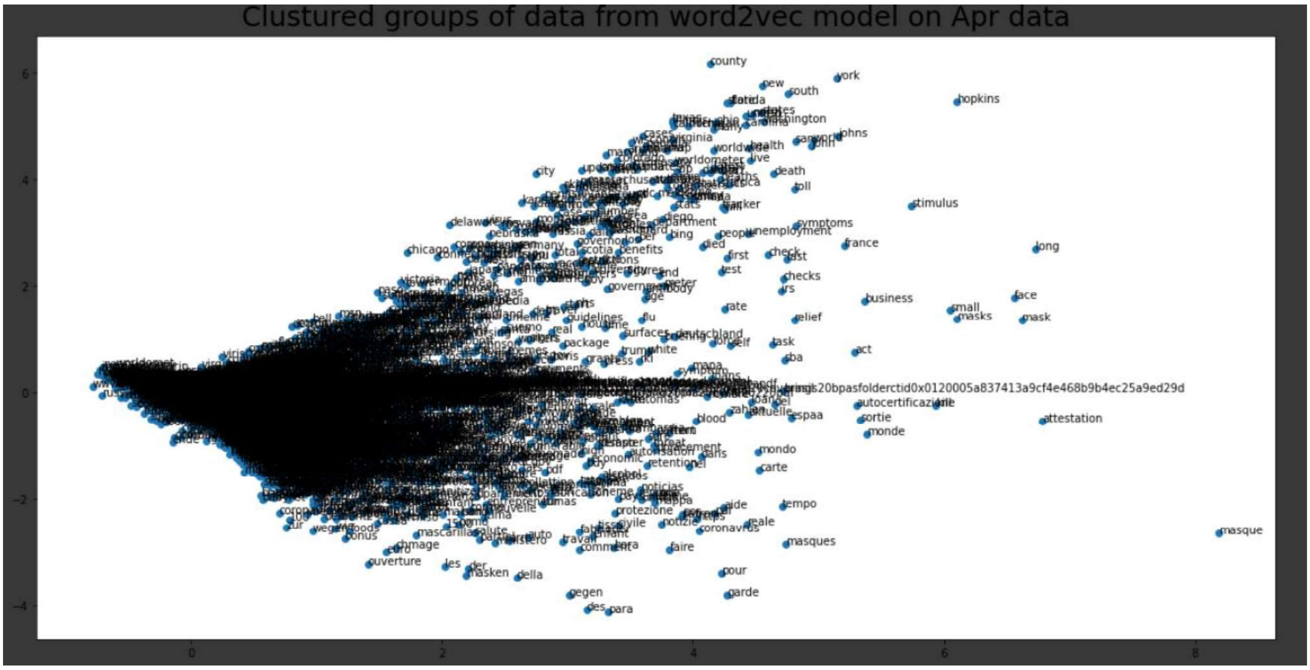
Modelling:

Unsupervised clustering using Word2Vec –



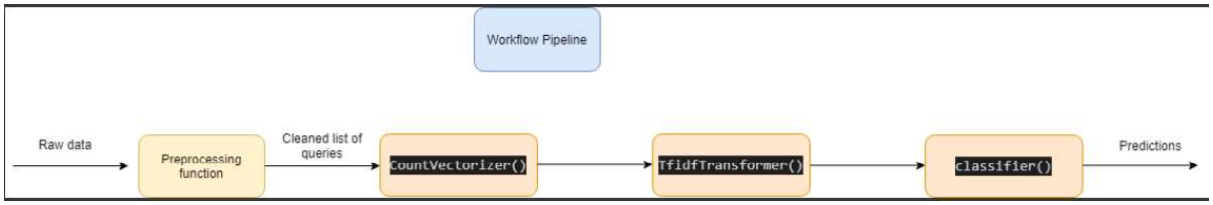


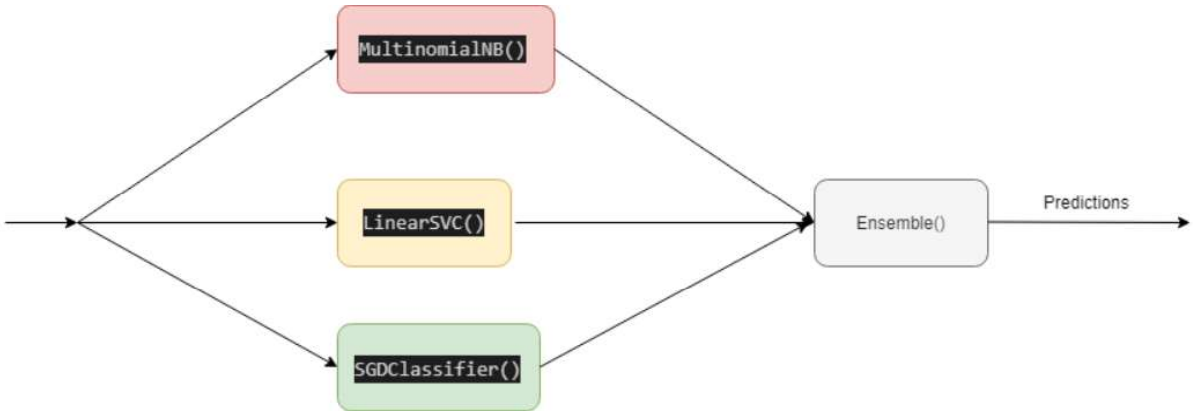

Clustured groups of data from word2vec model on Mar data


Clustured groups of data from word2vec model on Apr data

Supervised algorithms—

- It's been identified by exploring the dataset that classes (labels) are highly imbalanced and so SMOTE technique has been applied for oversampling minority classes.
- Then, series of classifiers have been tested – SGDclassifier(), MultinomialNB(), LinearSVC() etc. and finally ensemble of all models have been made and saved.
- There has been detailed analysis of all the iterative process applied in the jupyter notebook section below.
- Finally, metrics such as accuracy, precision and recall has been tested against test dataset.



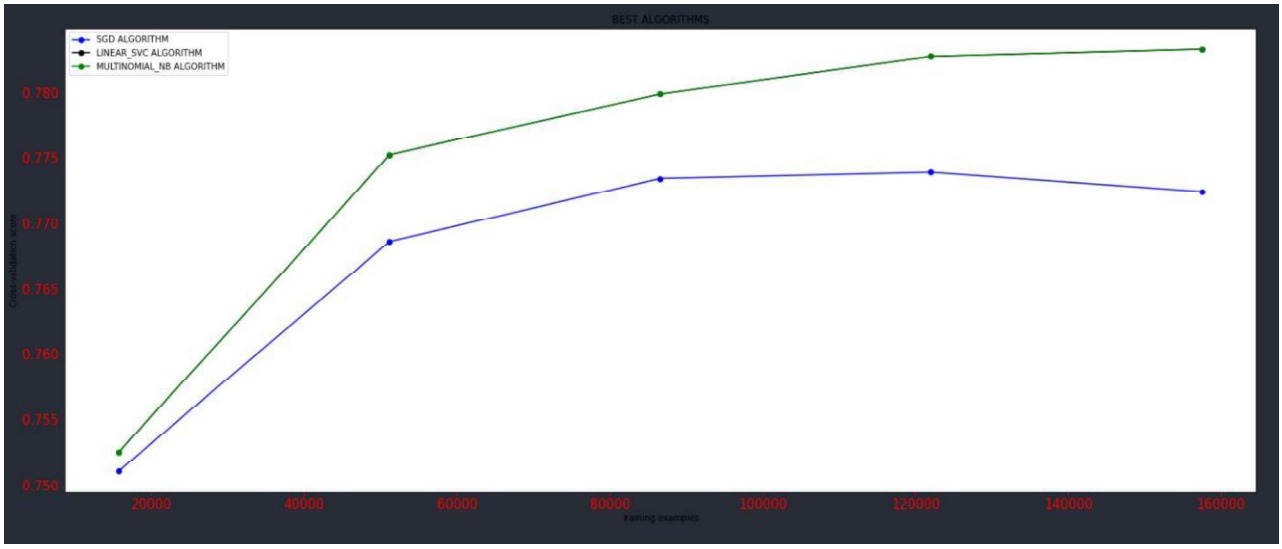


```
# predicting using each one of trained classifier, the results will be same as everyone got almost equal accuracy

print(pipe_sgd.predict(['angela merkel take on coronavirus']))
print(pipe_svc.predict(['angela merkel take on coronavirus']))
print(pipe_mnb.predict(['angela merkel take on coronavirus']))
```

```
['Germany']
['Germany']
['Germany']
```

```
print(pipe_sgd.predict(['coronavirus Updates in india']))
print(pipe_svc.predict(['coronavirus Updates in india']))
print(pipe_mnb.predict(['coronavirus Updates in india']))
```

```
['India']
['India']
['India']
```

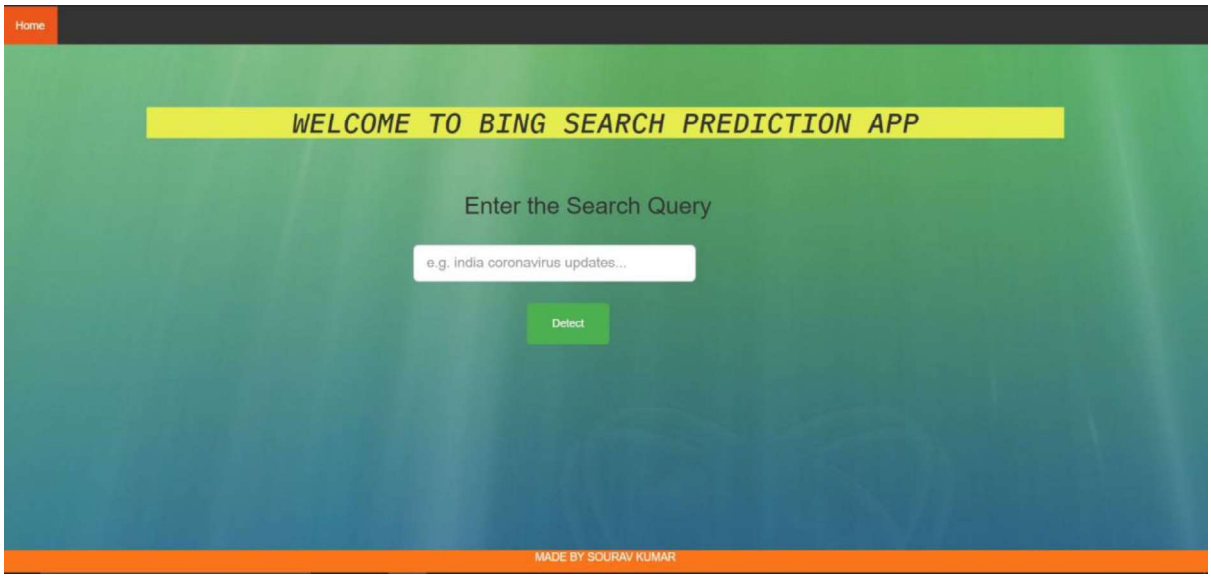


Deployment and Hosting:

- The final model which is trained and saved is then loaded and deployed with Flask backend server and hosted on pythonanywhere.com cloud service.

  The code for the same is available at my github repo: https://github.com/souravs17031999/MicrosoftBing-search-query-prediction

- Server Link (Live website) : https://souravsdlboy.pythonanywhere.com/bing_search



Complete Jupyter Notebook Code:

EXPLORING BING SEARCHES (MICROSOFT) BY PEOPLE DURING COVID-19 PANDEMIC AND MODELLING BY BUILDING CLASSIFIER FOR PREDICTING ORIGIN OF COUNTRY BASED ON USER'S QUERY AS INPUT

@ author : sourav kumar (101883068)

Dataset Link : https://github.com/microsoft/BingCoronavirusQuerySet (https://github.com/microsoft/BingCoronavirusQuerySe

Dataset provider : MicrosoftBing Team

Dataset info :

- This dataset comprises the text queries issued by the people from different parts of the world using Bing search.
- The queries which were only issued multiple times by multiple users is taken into account.
- The dataset comprises of Four months of data (JAN-APR) till present (28May 2020) but it will hopefully increase more as time passes by.
- Private data has already been removed by Bing Team.

Data Acquisition

In [2]:

```
# DATASET LINK : https://github.com/microsoft/BingCoronavirusQuerySet
!git clone https://github.com/microsoft/BingCoronavirusQuerySet
```

```
Cloning into 'BingCoronavirusQuerySet'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 82 (delta 8), reused 9 (delta 3), pack-reused 59
Unpacking objects: 100% (82/82), done.
```

Importing libraries

In [57]:

```
# import all packages
import time
import datetime
import os
import random
from tqdm import tqdm
import chardet

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns
import chardet
from wordcloud import WordCloud, STOPWORDS

import nltk
nltk.download('punkt')
nltk.download('stopwords')
import re
from string import punctuation
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import gensim
from nltk.probability import FreqDist
from nltk.corpus import stopwords
from string import punctuation
from nltk.collocations import *
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from nltk.collocations import TrigramCollocationFinder
from nltk.metrics import TrigramAssocMeasures
from gensim.models import Word2Vec

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, le
arning_curve
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import LinearRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC


from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering

from imblearn.over_sampling import ADASYN, SMOTE, RandomOverSampler


from sklearn.decomposition import PCA
import joblib
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:

```
# list of all paths
# /content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-01-01_2020-01-31.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-02-01_2020-02-29.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-03-01_2020-03-31.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-04-01_2020-04-30.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-01-01_2020-01-31.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-02-01_2020-02-29.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-03-01_2020-03-31.tsv
# /content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-04-01_2020-04-30.tsv
```

In [ ]:

```
def load_dataset(dataset_path, encoding_guess=True):
    """
    helps in guessing the encoding and returning the dataframe created from raw .csv fi
le

    Parameters:
        dataset_path (str): full path of the dataset
        encoding_guess (str): guessing the encoding using chardet module, generally use
ful when unknown encoding

    Returns:
        df (dataframe) : pandas dataframe containing loaded .csv file

    """
    # if encoding is given to be true, this will try to guess the encoding and then rea
d the file
    if encoding_guess:
        with open(dataset_path, 'rb') as f:
            result = chardet.detect(f.read())

        print(f"Detected file encoding as {result['encoding']}")

        df = pd.read_csv(dataset_path,sep="\t",header=0, encoding=result['encoding'])
    # otherwise it will use the default encoding what pandas will use to read, if it fa
ils , then try setting it to true
    else:
        df = pd.read_csv(dataset_path,sep="\t",header=0)

    return df
```

In [ ]:

```
# storing all the paths as a string
jan_country_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-01-01_2020-01-31.tsv"
feb_country_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-02-01_2020-02-29.tsv"
mar_country_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-03-01_2020-03-31.tsv"
apr_country_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByCountry_2020-04-01_2020-04-30.tsv"

jan_state_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-01-01_2020-01-31.tsv"
feb_state_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-02-01_2020-02-29.tsv"
mar_state_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-03-01_2020-03-31.tsv"
apr_state_data_path = "/content/BingCoronavirusQuerySet/data/2020/QueriesByState_2020-04-01_2020-04-30.tsv"
```

Loading Data

In [ ]:

```
# loading all country month wise data
df_jan_country = load_dataset(jan_country_data_path, encoding_guess=False) # data for january month
df_feb_country = load_dataset(feb_country_data_path, encoding_guess=False) # data for february month
df_mar_country = load_dataset(mar_country_data_path, encoding_guess=False) # data for march month
df_apr_country = load_dataset(apr_country_data_path, encoding_guess=False) # data for april month
```

Looking at data

In [ ]:

```
# checking shape of jan data
print(f"shape of dataframe : {df_jan_country.shape}")
df_jan_country.head()
```

```
shape of dataframe : (33901, 5)
```

Out[ ]:

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-01-01	webasto	True	Germany	1
1	2020-01-01	coronavirus	False	United States	100
2	2020-01-01	p2 masks	True	Australia	100
3	2020-01-01	china virus	True	United States	15
4	2020-01-01	p2 masks australia	True	Australia	1

In [ ]:

```
# checking shape of feb data
print(f"shape of dataframe : {df_feb_country.shape}")
df_feb_country.head()
```

```
shape of dataframe : (132979, 5)
```

Out[ ]:

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-02-01	coronavirus cover up	False	United States	1
1	2020-02-01	corona virus cdc	False	United States	1
2	2020-02-01	coronavirus	False	Pakistan	100
3	2020-02-01	lysol corona virus	False	United States	1
4	2020-02-01	coronavirus ottawa	False	Canada	3

In [ ]:

```
# checking shape of mar data
print(f"shape of dataframe : {df_mar_country.shape}")
df_mar_country.head()
```

```
shape of dataframe : (993787, 5)
```

Out[ ]:

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-03-01	gouvernement.fr coronavirus	False	France	1
1	2020-03-01	butlins	True	United Kingdom	4
2	2020-03-01	kachelmann	True	Germany	1
3	2020-03-01	berkshire coronavirus	False	United Kingdom	1
4	2020-03-01	oberstdorf	True	Germany	1

In [ ]:

```
# checking shape of apr data
print(f"shape of dataframe : {df_apr_country.shape}")
df_apr_country.head()
```

```
shape of dataframe : (675038, 5)
```

Out[ ]:

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-04-01	is influenza a coronavirus	False	United States	1
1	2020-04-01	pennsylvania coronavirus stay at home	False	United States	1
2	2020-04-01	number of coronavirus tests by state	False	United States	1
3	2020-04-01	https://www.gov.uk/coronavirus-extremely-vulne...	False	United Kingdom	1
4	2020-04-01	cdc coronavirus update new york state	False	United States	1

EXPLORATORY DATA ANALYSIS (EDA)

In [ ]:

```
def plot_count_search(df, interested_country_names, title = None, fig_size = (60, 15),
title_size=50, title_pad=30, x_label_size = 30, y_label_size = 30, x_label_pad=5, y_lab
el_pad = 25, rotation=30):
  """
    This function helps in plotting the countplot containing number of searches
    on y-axis and country name on x-axis

    Parameters:
        df (dataframe): dataframe name
        interested_country_names (list): list of all countries we are interested
          based on our threshold value of searches to be taken into account
        title (str) : title for the countplot
        fig_size (tuple) : tuple containing width, height values
        title_size (int) : size of title of countplot
        title_pad (int) : padding, if required values for title
        x_label_size (int) : size of font label on x-axis
        y_label_size (int) : size of font label on y-axis
        x_label_pad (int) : padding, if required for x-label
        y_label_pad (int) : padding, if required for y-label
        rotation (int/float) : rotation in degree for x-ticks

    Returns:
        None

    """
  # making the grid of 1 x 1
  fig, ax = plt.subplots(1,1, figsize=fig_size)
  # plotting countplot
  sns.countplot(df[df['Country'].isin(interested_country_names)]['Country'], ax=ax) # h
ere we have filtered the dataframe rows according to passed country names

  # controlling other params of the graph
  ax.set_title(title,fontsize=title_size, pad=title_pad)
  ax.set_xlabel("Country Name",fontsize=x_label_size, labelpad=x_label_pad)
  ax.set_ylabel("Number of searches",fontsize=y_label_size, labelpad=y_label_pad)
  ax.tick_params(axis="x", labelsize=25)
  ax.tick_params(axis="y", labelsize=25)
  # rotating the x ticks for better visuals
  N = plt.setp(ax.get_xticklabels(), rotation=rotation, horizontalalignment='right')
```

In [ ]:

```
# Tracking less relevant entries for our analysis
irrelevant_items = 0
for i in df_jan_country['Country'].value_counts():
  if i < 50:
    irrelevant_items += 1

print("Number of rows containing queries less than 100 : ", irrelevant_items)
```

```
Number of rows containing queries less than 100 :  149
```

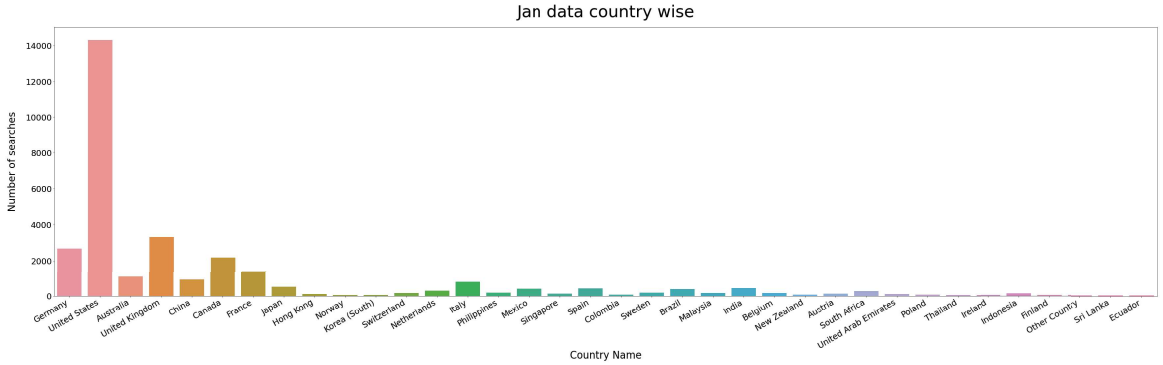
In [ ]:

```
# saving important country names for our analysis who have search queries over 50
interested_country_names_jan = [i for i, j in df_jan_country['Country'].value_counts().
to_dict().items() if j > 50]
```

How search queries are distributed across the world country wise , for each month ?

In [ ]:

```
plot_count_search(df_jan_country, interested_country_names_jan, title = "Jan data country wise")
```



In [ ]:

```
df_jan_country['Country'].value_counts().head(10)
```

Out[ ]:

```
United States     14330
United Kingdom     3330
Germany            2670
Canada             2172
France             1396
Australia          1089
China               926
Italy               794
Japan               511
India               444
Name: Country, dtype: int64
```

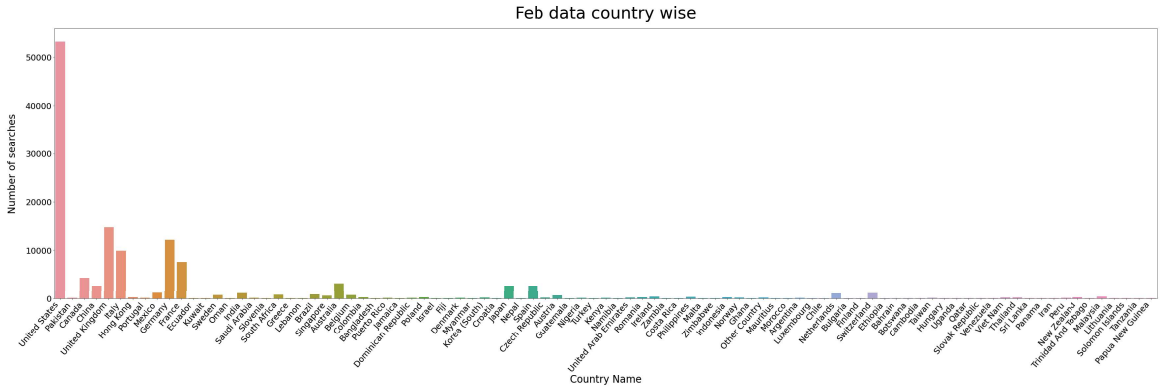
January data shows the most search queries were issued by users from US, followed by UK , India being at 10 with 444 quiries.

In [ ]:

```
interested_country_names_feb = [i for i, j in df_feb_country['Country'].value_counts().to_dict().items() if j > 50]
```

In [ ]:

```
plot_count_search(df_feb_country, interested_country_names_feb, title = "Feb data count
ry wise", rotation=50)
```



In [ ]:

```
df_feb_country['Country'].value_counts().head(15)
```

Out[ ]:

```
United States     53233
United Kingdom    14795
Germany           12237
Italy              9972
France             7641
Canada             4358
Australia          3155
China              2447
Spain              2433
Japan              2349
Mexico             1238
Switzerland        1197
India              1151
Netherlands        1087
Brazil              889
Name: Country, dtype: int64
```

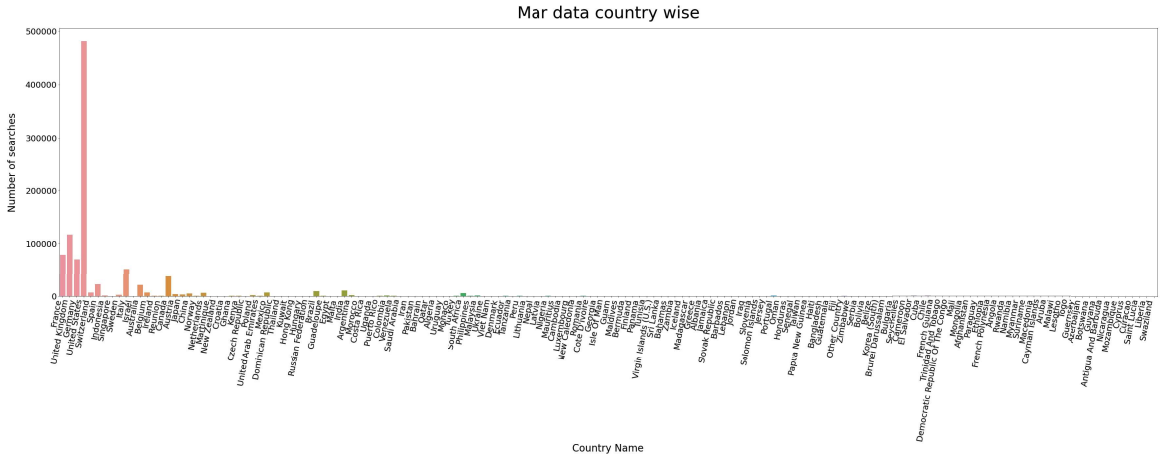
February data shows the most search queries were issued by users from US, followed by UK , India being at 10 with 1151 quiries.

In [ ]:

```
interested_country_names_mar = [i for i, j in df_mar_country['Country'].value_counts().
to_dict().items() if j > 50]
```

In [ ]:

```
plot_count_search(df_mar_country, interested_country_names_mar, title = "Mar data country wise", rotation=80)
```



In [ ]:

```
df_mar_country['Country'].value_counts().head(10)
```

Out[ ]:

```
United States     481819
United Kingdom    116569
France             78914
Germany            70046
Italy              51745
Canada             37199
Spain              22667
Australia          21032
India              10462
Brazil              9429
Name: Country, dtype: int64
```

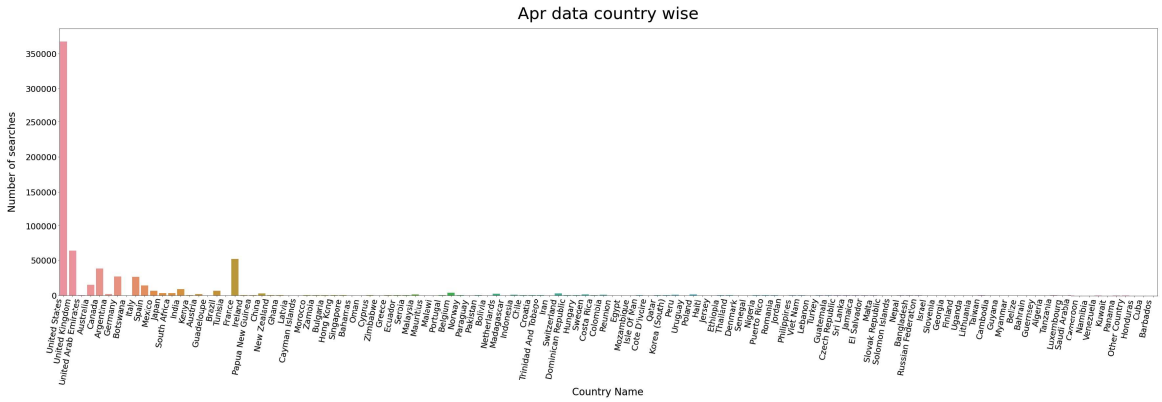
March data shows the most search queries were issued by users from US, followed by UK , India being at 10 with 10462 quiries.

In [ ]:

```
interested_country_names_apr = [i for i, j in df_apr_country['Country'].value_counts().to_dict().items() if j > 50]
```

In [ ]:

```
plot_count_search(df_apr_country, interested_country_names_apr, title = "Apr data count
ry wise", rotation=80)
```



In [ ]:

```
df_apr_country['Country'].value_counts().head(10)
```

Out[ ]:

```
United States     367453
United Kingdom     63848
France             51790
Canada             38185
Germany            26956
Italy              26279
Australia          15215
Spain              14203
India               9126
Brazil              6605
Name: Country, dtype: int64
```

APRIL data shows the most search queries were issued by users from US, followed by UK , India being at 10 with 9126 quiries.

As we can see number of search queries have increased for India as well as other countries where COVID-19 cases have increased ever since January 2020 ... till present.

Now, we want to know people are searching what exact terms and how much ?

In [ ]:

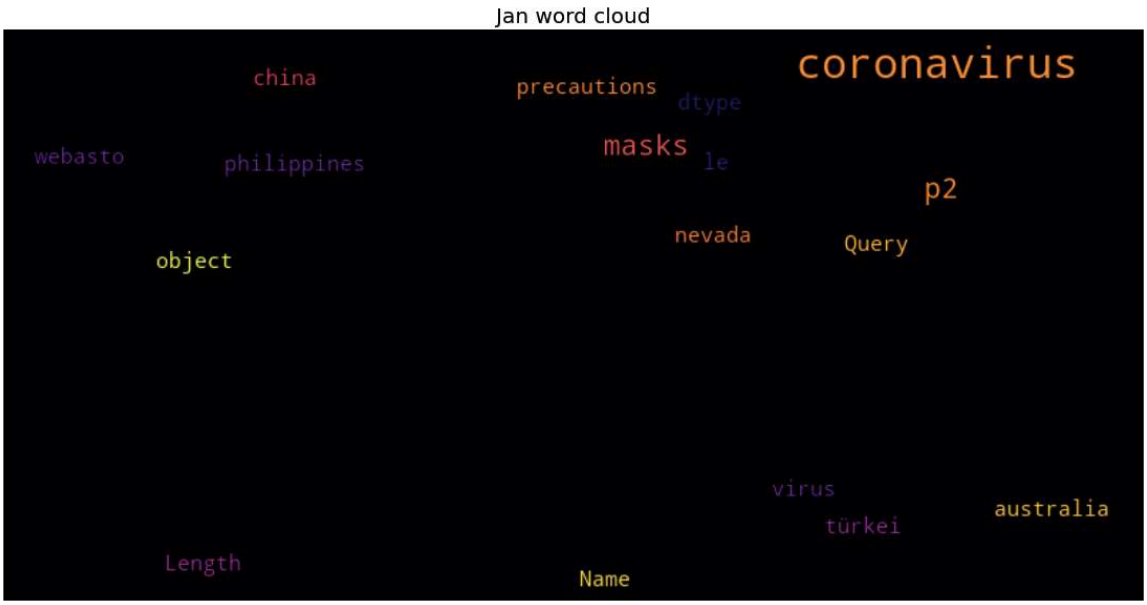
```
stopwords = set(STOPWORDS) # making a list of stopwords and using it to remove those wo
rds while counting frequencies for word cloud
```

In [ ]:

```
def plot_word_cloud(df, figsize=(20, 10), title = None, colormap='plasma'):
  """
    This function helps in creating the wordcloud for all the words encountered in the
 passed series/list of strings

    Parameters:
        df (dataframe): dataframe name
        figsize (tuple) : tuple values indicating width, height size
        colormap (str) : matplotlib colormaps , https://matplotlib.org/3.2.1/tutorials/
colors/colormaps.html

    Returns:
        None

    """
  # creating wordcloud object with the given dataframe object
  wordcloud_country_jan = WordCloud(width=800,
                            height=400,
                            background_color='black',
                            stopwords=stopwords,
                            max_words=500,
                            max_font_size=30,
                            random_state=42,
                            colormap=colormap
                           ).generate(str(df['Query']))

  # creating the grid of 1 x 1
  fig, ax = plt.subplots(1,1, figsize=figsize)
  plt.title(title, fontsize=20)
  plt.imshow(wordcloud_country_jan, interpolation='bilinear')
  plt.axis("off")
  # plotting the wordcloud
  plt.show()
```

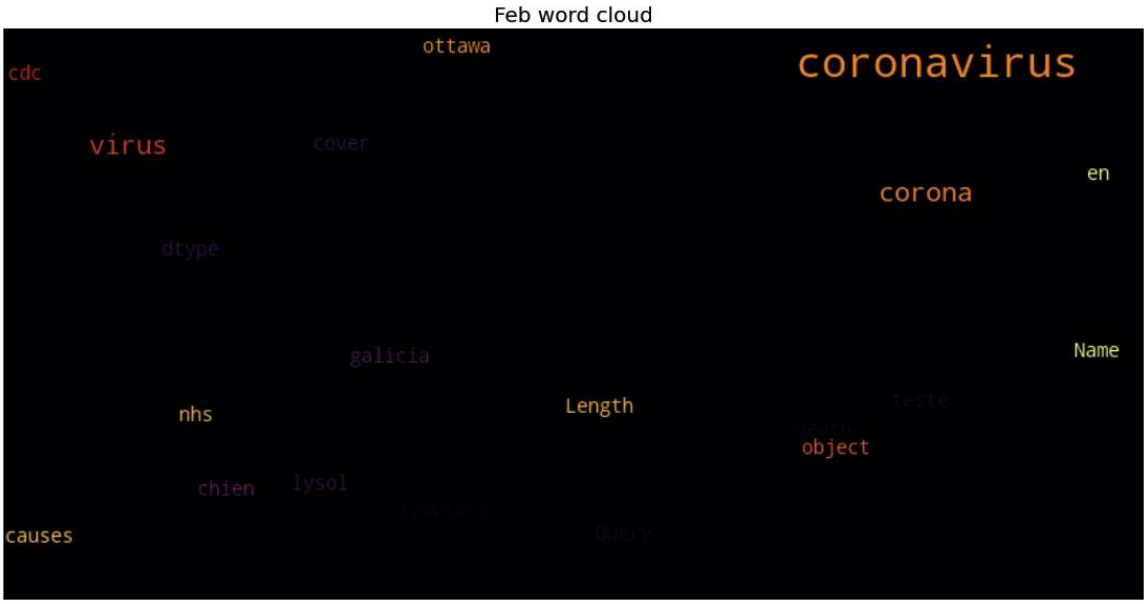
In [ ]:

```
plot_word_cloud(df_jan_country, title="Jan word cloud")
```



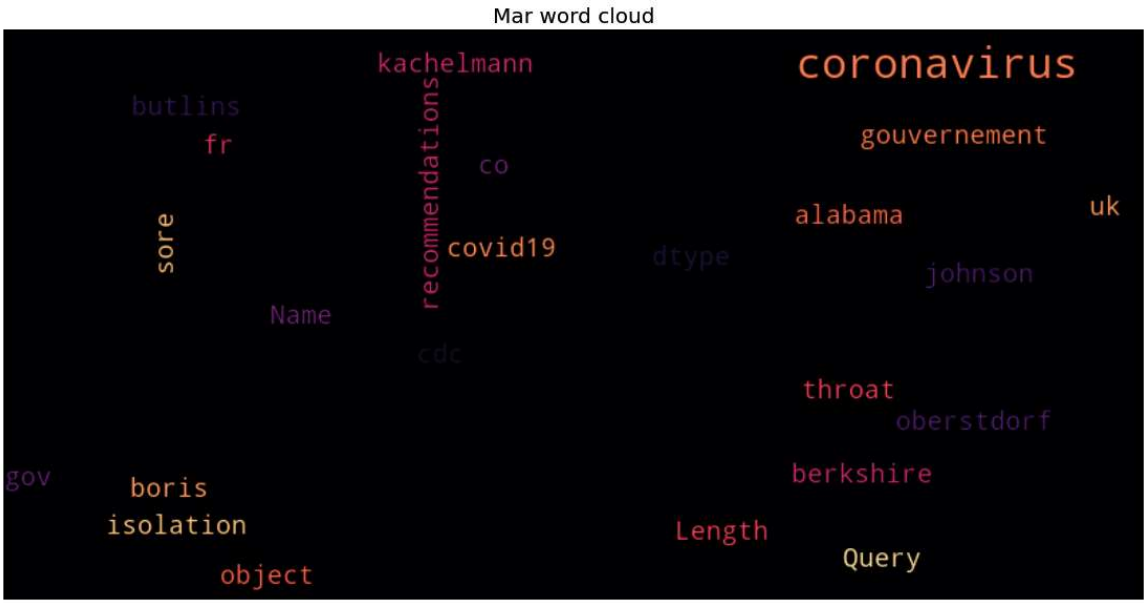
In [ ]:

```
plot_word_cloud(df_feb_country, title="Feb word cloud", colormap='inferno')
```



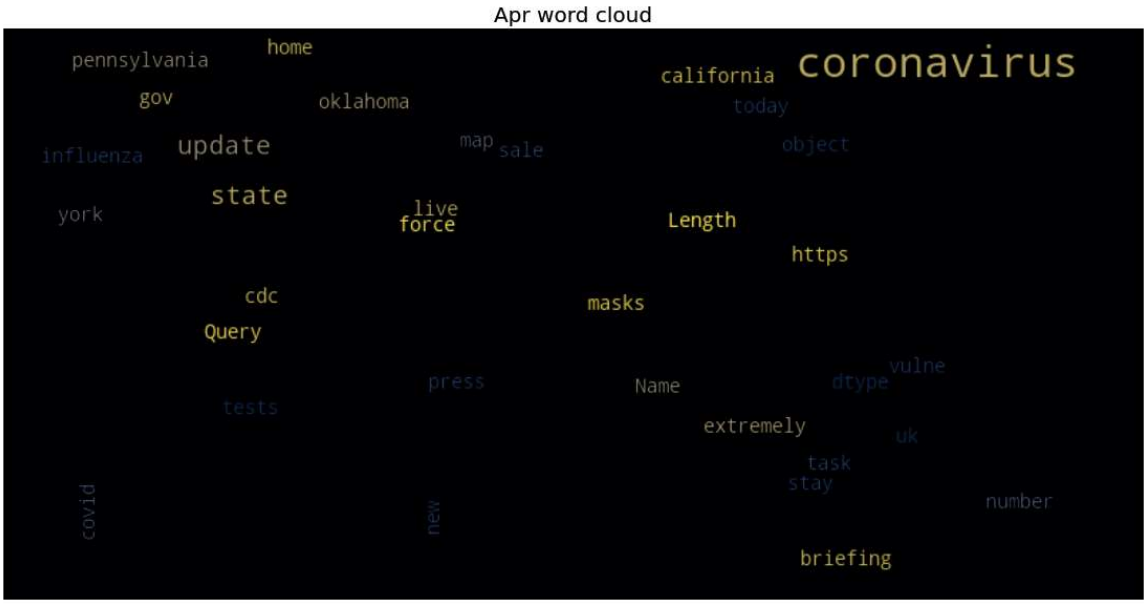
In [ ]:

```
plot_word_cloud(df_mar_country, title="Mar word cloud", colormap='magma')
```



In [ ]:

```
plot_word_cloud(df_apr_country, title="Apr word cloud", colormap='cividis')
```



From all the plots above, in all the four months we can see the queries related to coronavirus consisted mostly of the term itself "Coronavirus" and some other terms also appeared like "china", "masks", "virus", "covid19", "isolation"...

It tells a lot about intent of the people how they searched.

- Like in january people were mostly talking about "Precautions", "masks", "china" along with "Coronavirus".
- Then, in february, "virus", "cover", "causes"
- Then, in march, "sore", "recommendations", "covid19", "isolation"
- The, in april, "influenza", "tests", "press", "briefing"

Similar to above wordcloud , we can also use simple graph to check top 10 keywords searched for during different months using frequency distribution graph

In [ ]:

```
def build_query_string(df, interested_country_names):
  """
    This function helps in returning the joined string of words , all from list of quer
ies selected from df

    Parameters:
        df (dataframe): dataframe name
        interested_country_names (list) : list of only those countries name which have
 queries greater than threshold
    Returns:
        a large string made of words from all the queries

    """
  return " ".join([x for x in df[df['Country'].isin(interested_country_names)]['Query'
]])
```

In [13]:

```
print("Building string query for Jan data")
query_jan = build_query_string(df_jan_country, interested_country_names_jan)
print("Building string query for Feb data")
query_feb = build_query_string(df_feb_country, interested_country_names_feb)
print("Building string query for Mar data")
query_mar = build_query_string(df_mar_country, interested_country_names_mar)
print("Building string query for Apr data")
query_apr = build_query_string(df_apr_country, interested_country_names_apr)
```

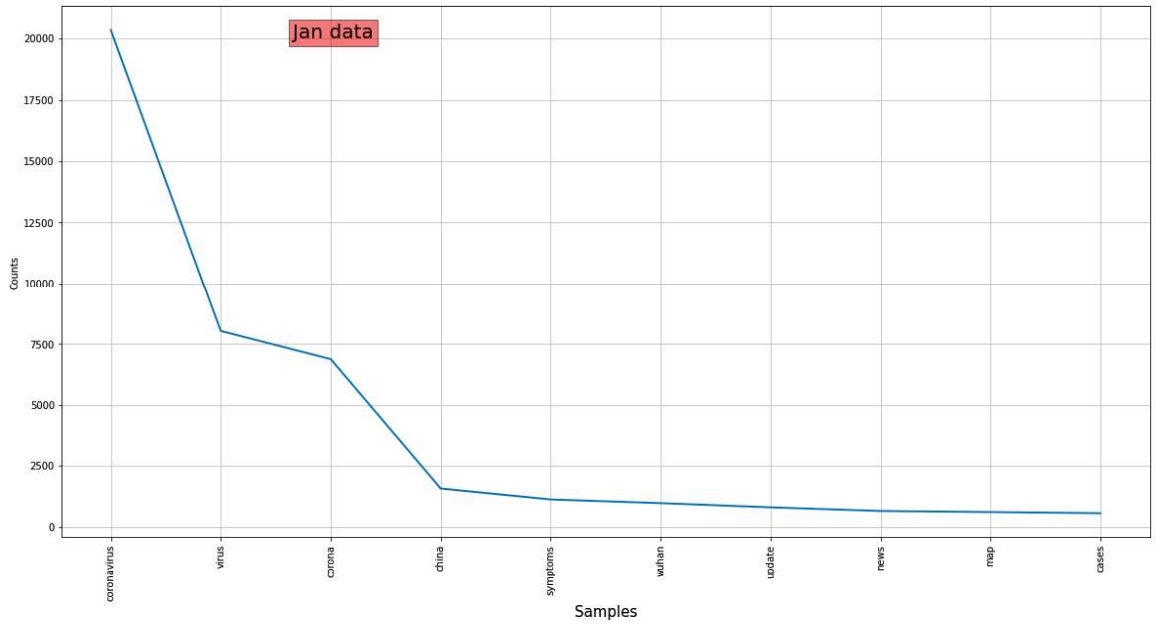
```
Building string query for Jan data
Building string query for Feb data
Building string query for Mar data
Building string query for Apr data
```

In [ ]:

```
def plot_cum_sum_fdist(query, title):
  """
    This function helps in plotting the frequency distribution for count of occurence o
f keywords

    Parameters:
        query (str) : a large string made of words from all the 'Query' from dataset
        title (str) : title for the text box for the plot
    Returns:
        None

    """
  stop_words = set(stopwords.words('english'))
  # preprocessing to clean the text
  fdist = FreqDist(word.lower() for word in word_tokenize(query) if word not in stop_wo
rds and word not in punctuation)

  # creating the grid 1 x 1
  fig, ax =plt.subplots(1,1, figsize=(20, 10))
  ax.set_xlabel("Country Name",fontsize=15)

  # setting the anchor box text
  plt.text(0.25, 0.95, s=title, horizontalalignment='center', verticalalignment='cente
r', fontsize=20, transform=ax.transAxes, bbox=dict(facecolor='red', alpha=0.5))
  # plotting the fdist
  fdist.plot(10, cumulative=False)
```

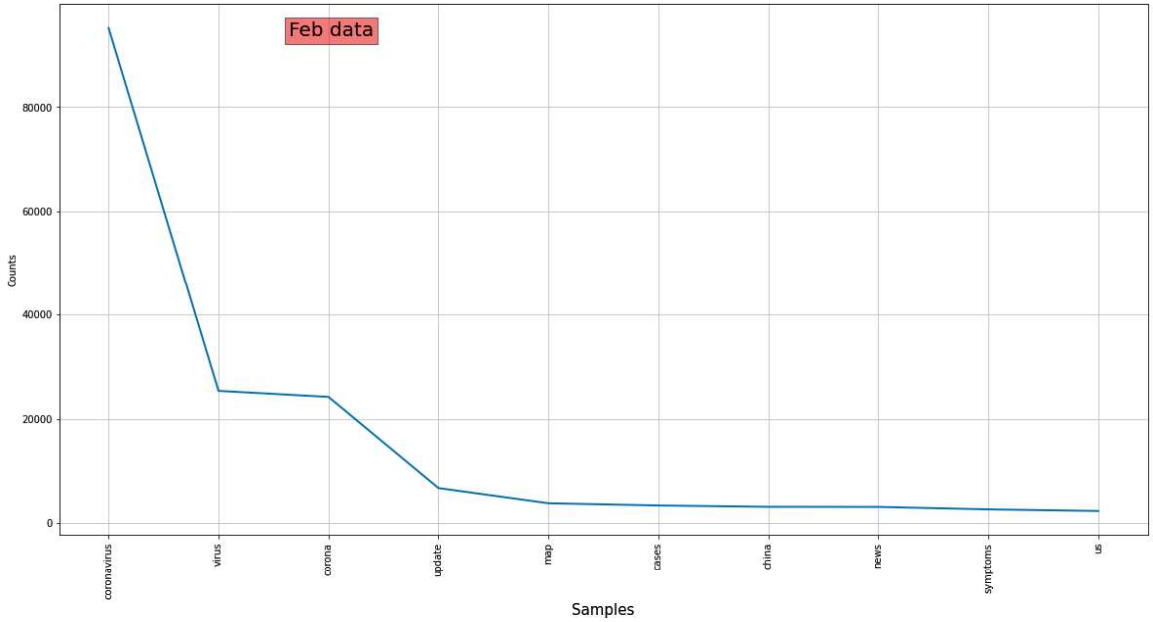
In [ ]:

```
plot_cum_sum_fdist(query_jan, title='Jan data')
```



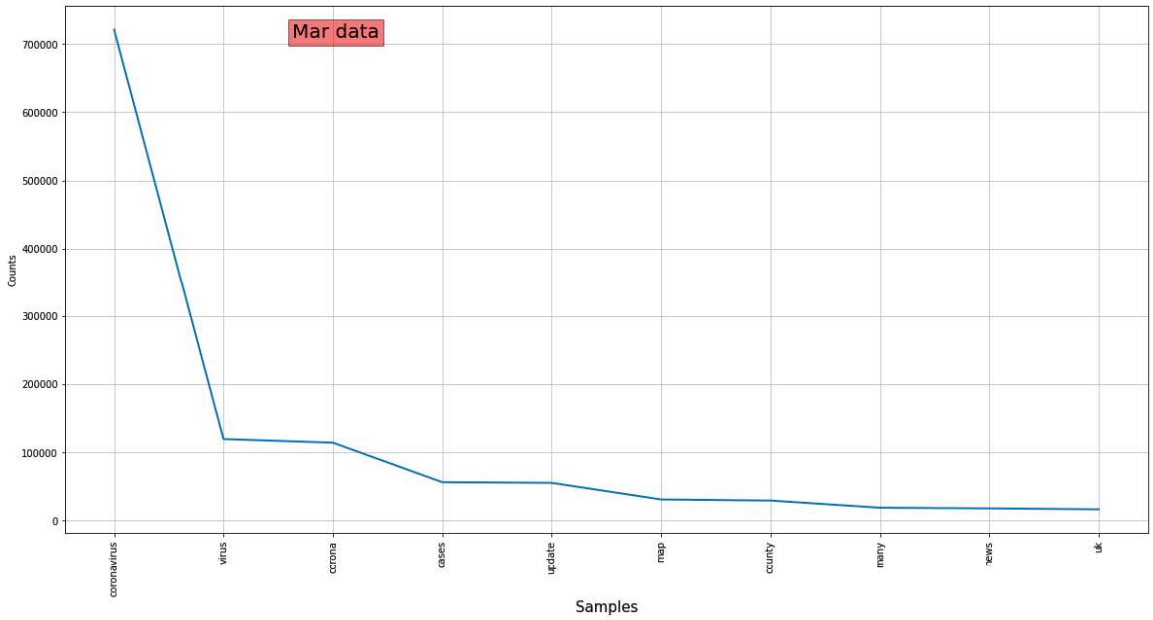
In [ ]:

```
plot_cum_sum_fdist(query_feb, title='Feb data')
```



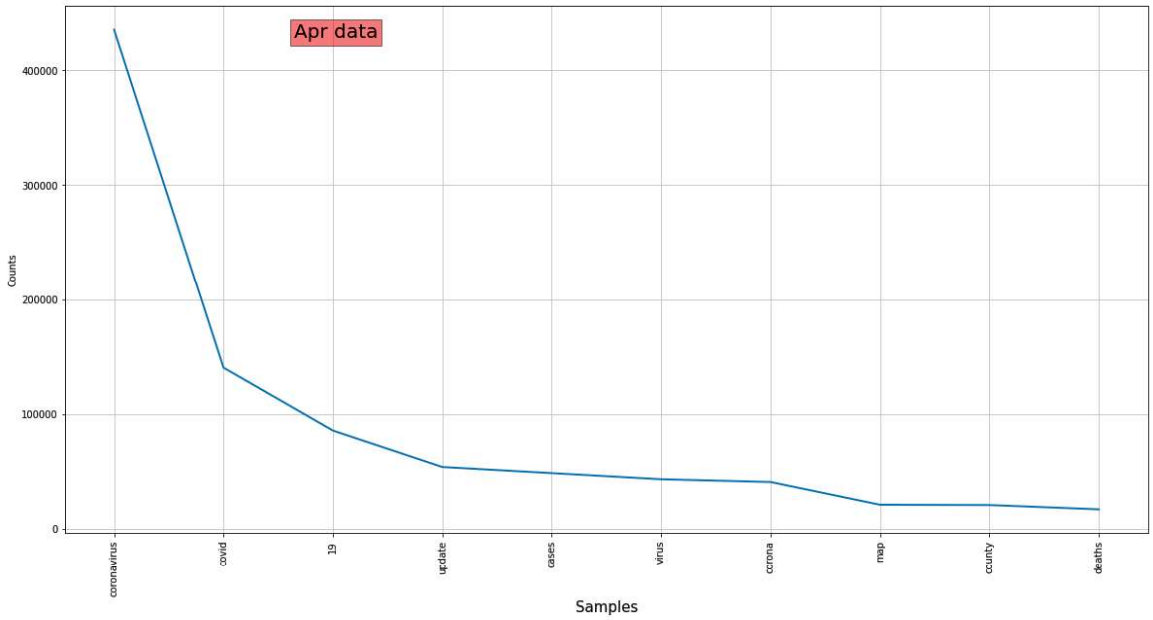
In [ ]:

```
plot_cum_sum_fdist(query_mar, title='Mar data')
```



In [ ]:

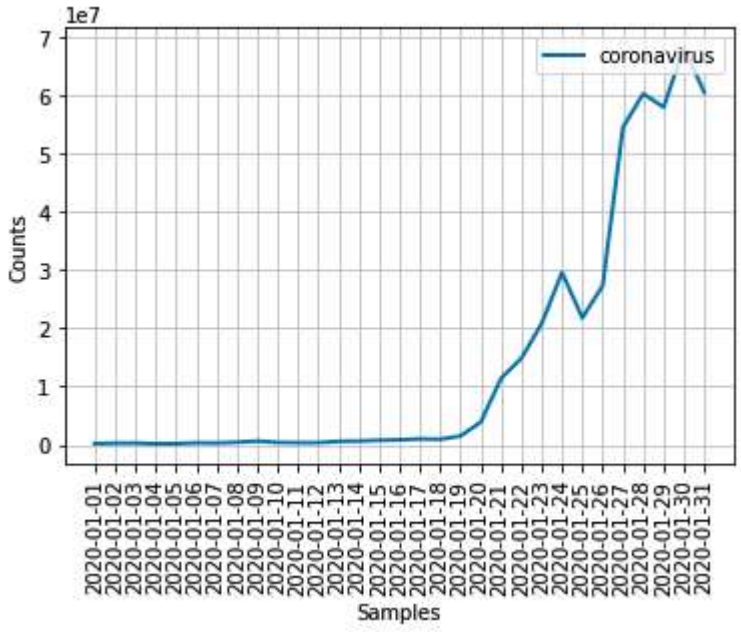
```
plot_cum_sum_fdist(query_apr, title='Apr data')
```



We can also see how each of these most frequently occuring keywords varies over time to time in terms of month.

In [ ]:

```
# plotting the frequency distribution based on the condition if the query starts with
 "coronavirus" and checking it's use over time period as shown
cfd = nltk.ConditionalFreqDist((target, fileid) for fileid in df_jan_country['Date'] fo
r w in df_jan_country['Query'] for target in ['coronavirus'] if w.lower().startswith(ta
rget))
cfd.plot()
```



The above graph is for January only and the same can be done acheived for other months (since other months have very data and is taking much time to plot)

Combining the dataset

In [ ]:

```
def combine_dataset(*args):
  """
    This function helps combining the dataset from different variables of datasets

    Parameters:
        *args (iterable) : list/tuple of all passed values (variable number of argument
s)
    Returns:
        df (dataframe) : combined Pandas DataFrame object

    """
  # combining all the above individual dataframes
  df = pd.concat([*args])
  return df
```

In [ ]:

```
combined_country_data = combine_dataset(df_jan_country[df_jan_country['Country'].isin(i
nterested_country_names_jan)],
                                        df_feb_country[df_feb_country['Country'].isin(interes
ted_country_names_feb)],
                                        df_mar_country[df_mar_country['Country'].isin(interes
ted_country_names_mar)],
                                        df_apr_country[df_apr_country['Country'].isin(interes
ted_country_names_apr)])
```

In [16]:

```
combined_country_data.shape
```

Out[16]:

```
(1829404, 5)
```

In [ ]:

```
combined_country_data.to_csv("combined_country_data.csv", index=False)
```

Collocations analysis

In [ ]:

```
def bigram_collocation_analyse(df):
  """
    This function helps in calculating probabilistic collocations using bigram model

    Parameters:
        df (dataframe) : pandas dataframe object for which bigram model has to be appli
ed
    Returns:
        list of tuples containing words occuring along with each other

    """
  helper_string = " ".join([x for x in df['Query']])
  # tokenizing the query string
  query_string = helper_string.split(" ")
  # make the list of all the stopwords
  stopset = set(stopwords.words('english'))

  # filtering words having chars of length less than 3 or if it is in stopword set
  filter_stops = lambda w: len(w) < 3 or w in stopset

  # trigram collocation
  biagram_collocation = BigramCollocationFinder.from_words(query_string)
  biagram_collocation.apply_word_filter(filter_stops)

  # returning the found trigrams based on likelihood ratio measure
  return biagram_collocation.nbest(BigramAssocMeasures.likelihood_ratio, 15)
```

In [ ]:

```
bigram_collocation_analyse(combined_country_data)
```

Out[ ]:

```
[('corona', 'virus'),
 ('coronavirus', 'coronavirus'),
 ('new', 'york'),
 ('john', 'hopkins'),
 ('death', 'toll'),
 ('coronavirus', 'virus'),
 ('corona', 'coronavirus'),
 ('united', 'states'),
 ('johns', 'hopkins'),
 ('many', 'people'),
 ('covid', 'coronavirus'),
 ('coronavirus', 'update'),
 ('coronavirus', 'cases'),
 ('many', 'cases'),
 ('small', 'business')]
```

In [ ]:

```
def trigram_collocation_analyse(df):
  """
    This function helps in calculating probabilistic collocations using trigram model

    Parameters:
        df (dataframe) : pandas dataframe object for which trigram model has to be appl
ied
    Returns:
        list of tuples containing words occuring along with each other

    """
  helper_string = " ".join([x for x in df['Query']])
  # tokenizing the query string
  query_string = helper_string.split(" ")
  # make the list of all the stopwords
  stopset = set(stopwords.words('english'))

  # filtering words having chars of length less than 3 or if it is in stopword set
  filter_stops = lambda w: len(w) < 3 or w in stopset

  # trigram collocation
  trigram_collocation = TrigramCollocationFinder.from_words(query_string)
  trigram_collocation.apply_word_filter(filter_stops)
  trigram_collocation.apply_freq_filter(3)

  # returning the found trigrams based on likelihood ratio measure
  return trigram_collocation.nbest(TrigramAssocMeasures.likelihood_ratio, 15)
```

In [ ]:

```
trigram_collocation_analyse(combined_country_data)
```

Out[ ]:

```
[('corona', 'virus', 'update'),
 ('corona', 'virus', 'coronavirus'),
 ('corona', 'virus', 'virus'),
 ('corona', 'corona', 'virus'),
 ('covid', 'corona', 'virus'),
 ('corona', 'virus', 'cases'),
 ('virus', 'corona', 'virus'),
 ('corona', 'virus', 'covid'),
 ('corona', 'virus', 'corona'),
 ('coronavirus', 'corona', 'virus'),
 ('corona', 'virus', 'county'),
 ('corona', 'virus', 'statistics'),
 ('corona', 'virus', 'live'),
 ('corona', 'virus', 'map'),
 ('corona', 'virus', 'start')]
```

In [ ]:

```
combined_country_data.head(5)
```

Out[ ]:

	Date	Query	IsImplicitIntent	Country	PopularityScore
0	2020-01-01	webasto	True	Germany	1
1	2020-01-01	coronavirus	False	United States	100
2	2020-01-01	p2 masks	True	Australia	100
3	2020-01-01	china virus	True	United States	15
4	2020-01-01	p2 masks australia	True	Australia	1

Okay, so now let's see top queries searched for special keywords like queries starting with "who", "where", "how" etc... to better understand intent of people searching Internet during this pandemic.

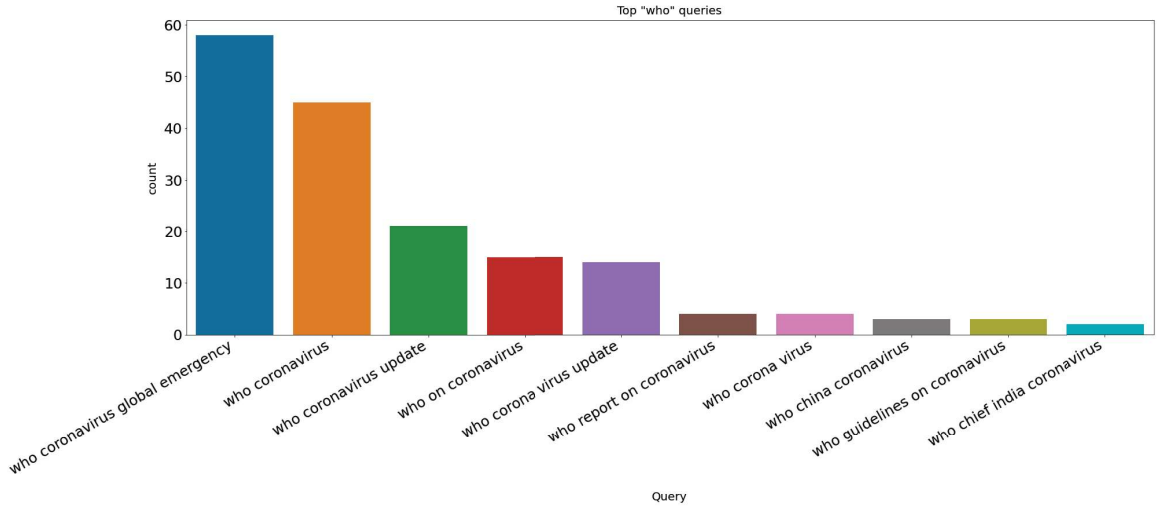
In [ ]:

```
def plot_queries(df, query, country_specific = None, rotation = 30, hue = False):
  """
    This function helps in plotting countplot of queries starting with some special que
stion keywords

    Parameters:
        df (dataframe) : pandas dataframe object
        query (str) : question keywords like "who", "where", "why" or "how" etc...
        country_specific (list) : if we want to filter our results for only specific co
untries, pass it as a list of names
        rotation (int/float) : value in degrees for rotation of x-ticks labels of plot
        hue (bool) : if hue is required by giving one additional argument - Country

    Returns:
        None

    """
  # this takes care of query being either specific to country or in general (for all co
untries)
  if country_specific != None:
    queries_df = df[(df['Query'].str.startswith(query)) & (df['Country'].isin(country_s
pecific))] # here we have filtered the dataframe rows according to query string and pas
sed country names (if passed)
  else:
    queries_df = df[df['Query'].str.startswith(query)]

  # constructing a grid of 1 x 1
  fig, ax = plt.subplots(1,1, figsize=(30, 10))
  ax.set_title(f"Top \"{query}\" queries",fontsize=20, pad=10)
  # controlling other params of grid
  ax.set_xlabel("Query",fontsize=20, labelpad=25)
  ax.set_ylabel("Number of searches",fontsize=20, labelpad=10)
  ax.tick_params(axis="x", labelsize=25)
  ax.tick_params(axis="y", labelsize=25)
  N = plt.setp(ax.get_xticklabels(), rotation=rotation, horizontalalignment='right')
  try:
    if not hue:
      sns.countplot(queries_df['Query'], ax=ax, data = queries_df, order=queries_df.Que
ry.value_counts().iloc[:10].index)
    else:
      sns.countplot(queries_df['Query'], ax=ax, hue = 'Country', data = queries_df, ord
er=queries_df.Query.value_counts().iloc[:10].index)
  except Exception as error:
    print("The queries requested did not resulted in any value to plot it on graph ! Th
is will result in an empty graph. Try to use some another query.")
  #plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Let's focus on queries issued from India.

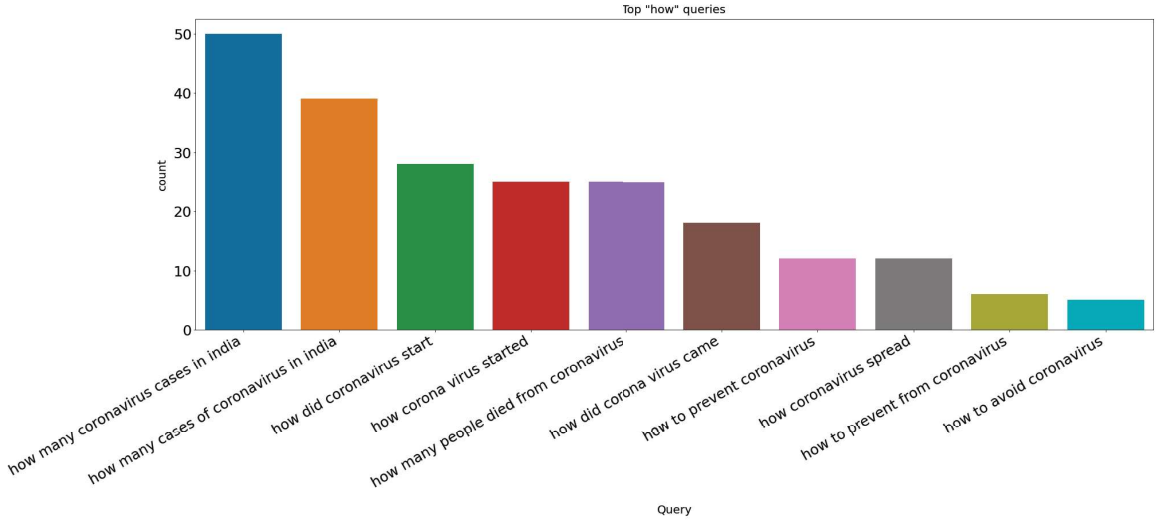
In [ ]:

```
plot_queries(combined_country_data, 'who', ['India'])
```



In [ ]:

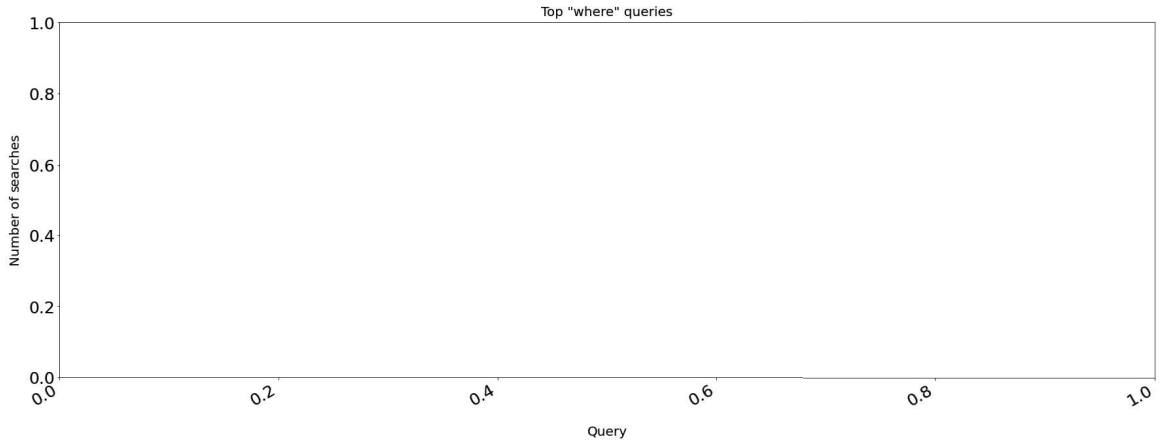
```
plot_queries(combined_country_data, 'how', ['India'])
```



In [ ]:

```
plot_queries(combined_country_data, 'where', ['India'])
```

```
The queries requested did not resulted in any value to plot it on graph !
This will result in an empty graph. Try to use some another query.
```



- In the who queries - there is i guess one anamoly because of people writing, "WHO" as in world health organisation as well as "WHO" as in questioning word. So, maybe we need to dug deeper which one is which one.
- Also, it's strange there's no query starting with where.

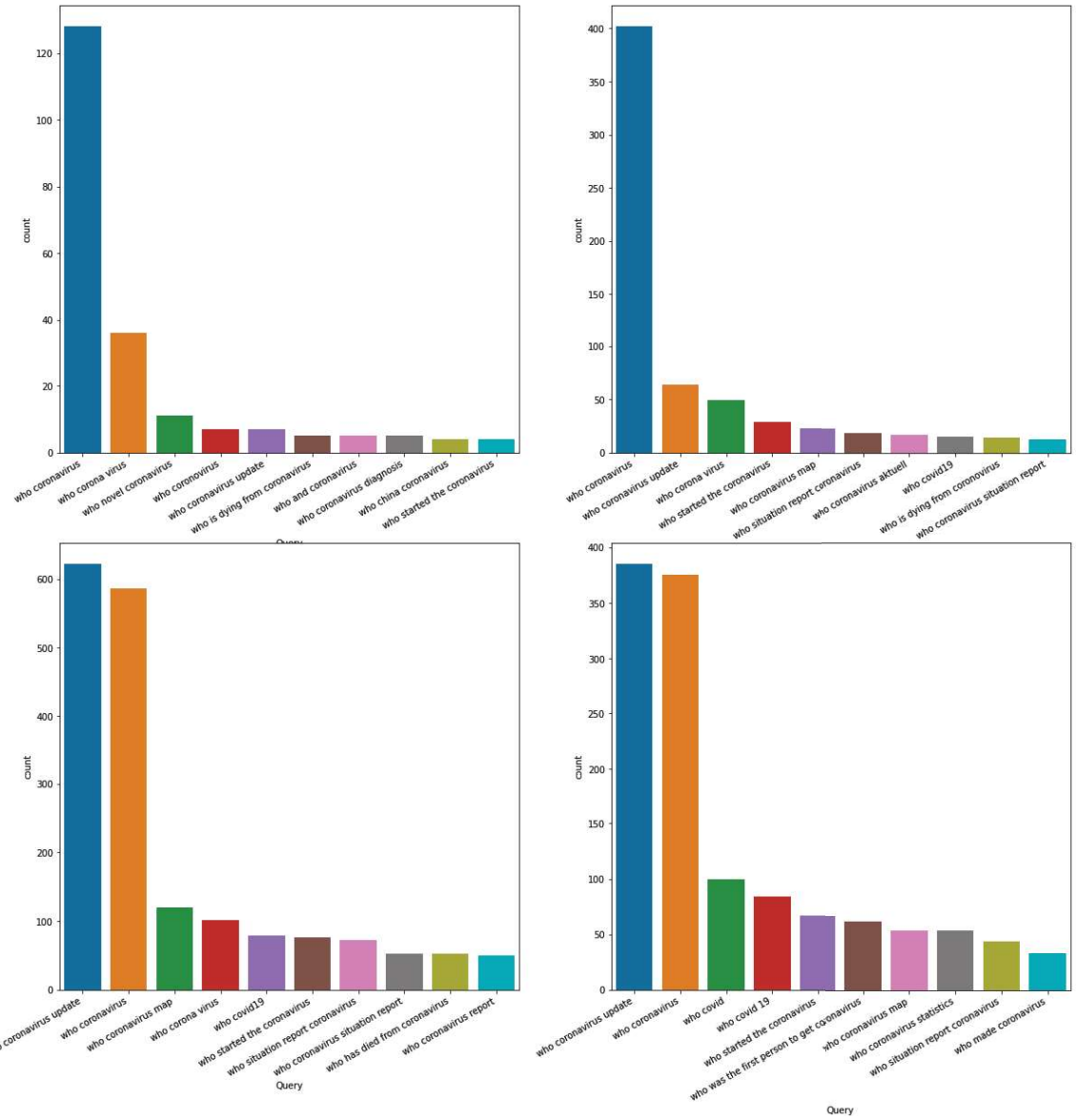
Now, let's compare how queries changed over month to month (here since there is not much data for specific India so i am going for overall countries)

In [ ]:

```
def plot_multiple_queries(query):
  """
    This function helps in plotting countplot of multiple subplots in one go

    Parameters:
        query (str) : question keywords like "who", "where", "why" or "how" etc...

    Returns:
        None

    """

  # constructing temp. dataframes to be used later , filtered based on query string
  queries_jan_df = df_jan_country[(df_jan_country['Query'].str.startswith(query))]
  queries_feb_df = df_feb_country[(df_feb_country['Query'].str.startswith(query))]
  queries_mar_df = df_mar_country[(df_mar_country['Query'].str.startswith(query))]
  queries_apr_df = df_apr_country[(df_apr_country['Query'].str.startswith(query))]

  # queries is a 2-d level nested list comprising of all the above made df for convinie
nce
  queries = [[queries_jan_df, queries_feb_df],
             [queries_mar_df, queries_apr_df]]

  # constructing subplot by making grid of 2 x 2
  fig, ax = plt.subplots(2,2, figsize=(20, 20))

  # plotting all the graphs
  for i in range(2):
    for j in range(2):
      N = plt.setp(ax[i][j].get_xticklabels(), rotation=30, horizontalalignment='right'
)
      sns.countplot(queries[i][j]['Query'], ax=ax[i][j], data = queries[i][j], order=qu
eries[i][j].Query.value_counts().iloc[:10].index)
```

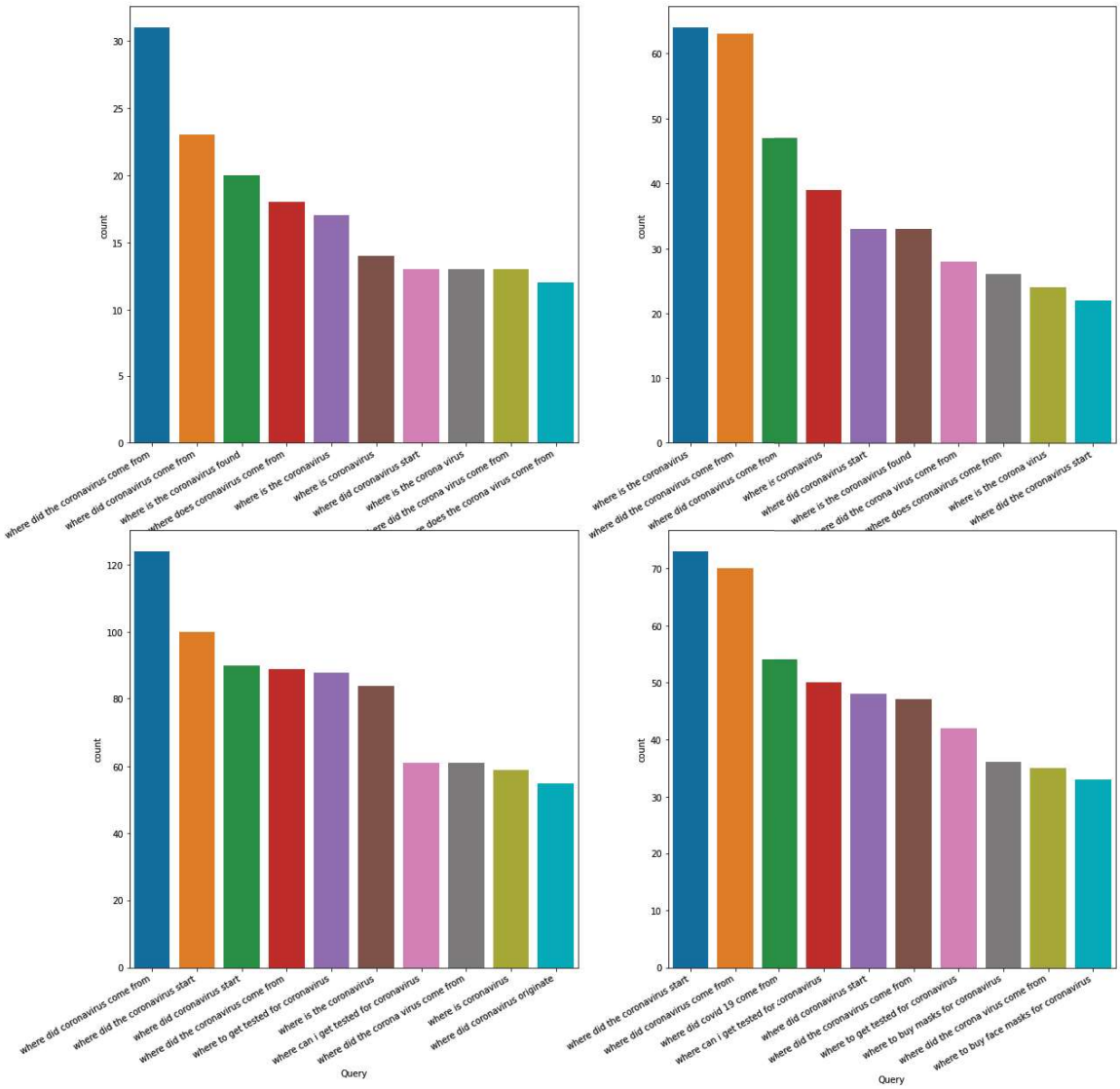
In [ ]:

```
plot_multiple_queries("who")
```



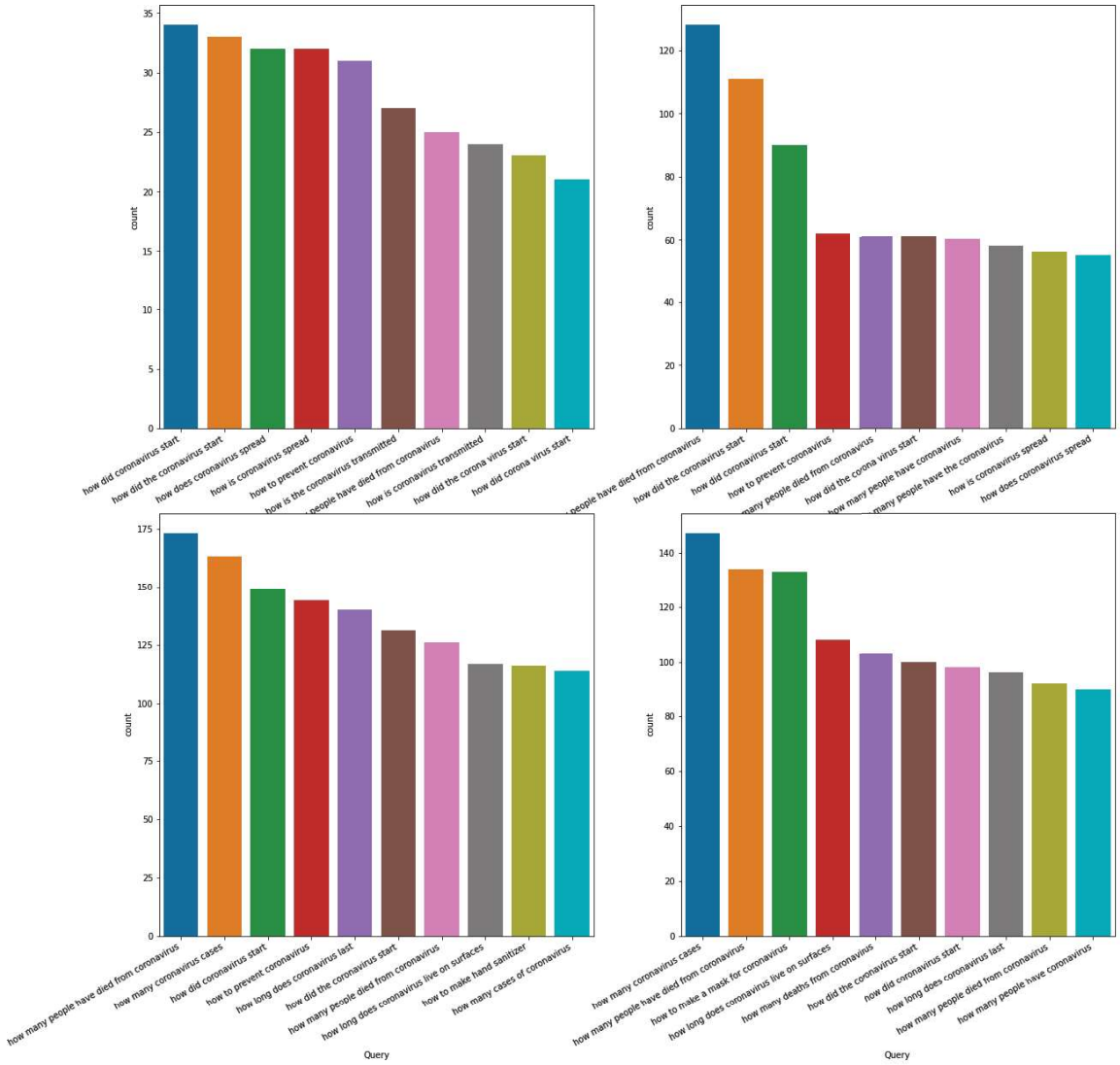
In [ ]:

```
plot_multiple_queries("where")
```



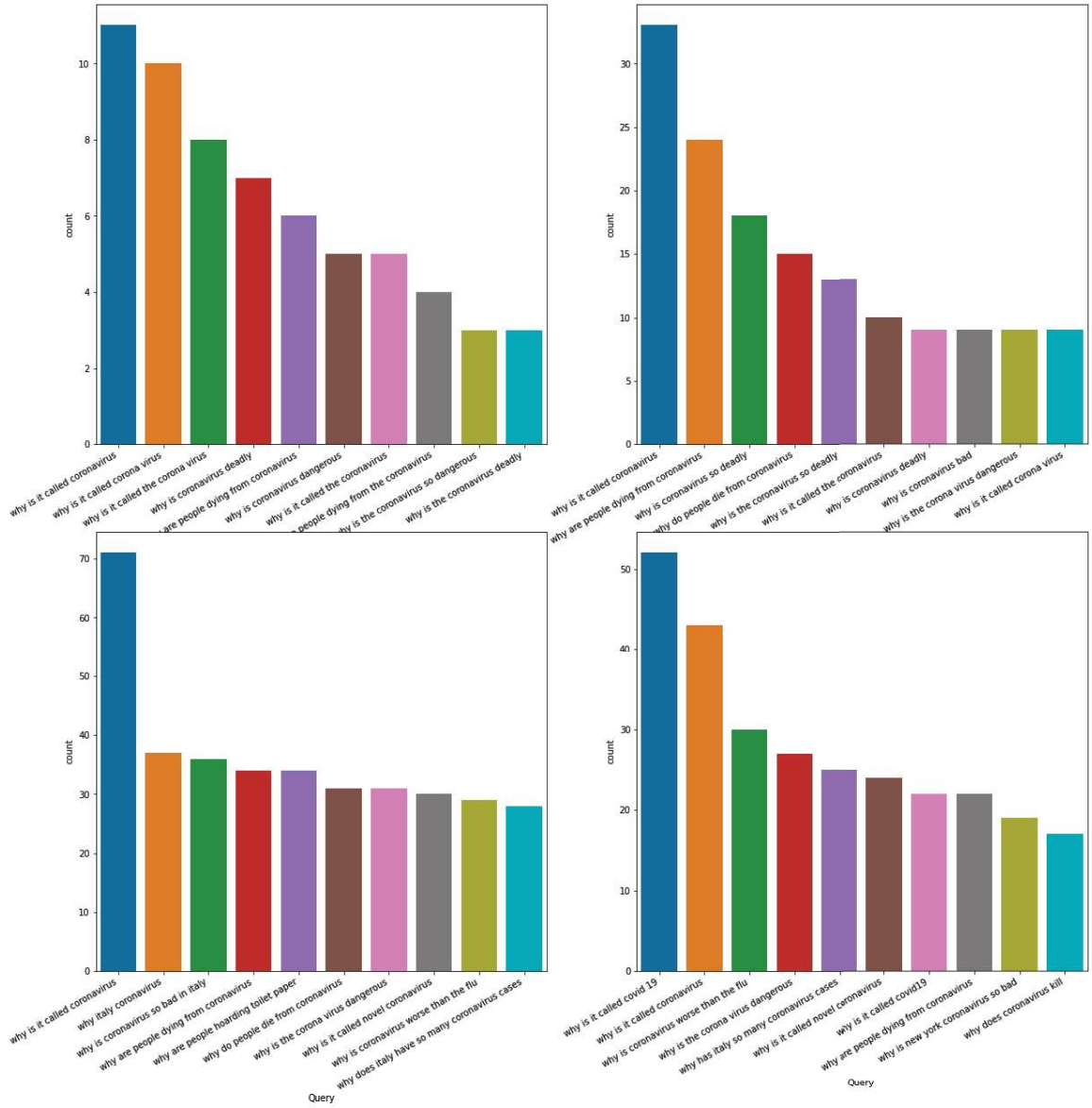
In [ ]:

```
plot_multiple_queries("how")
```



In [ ]:

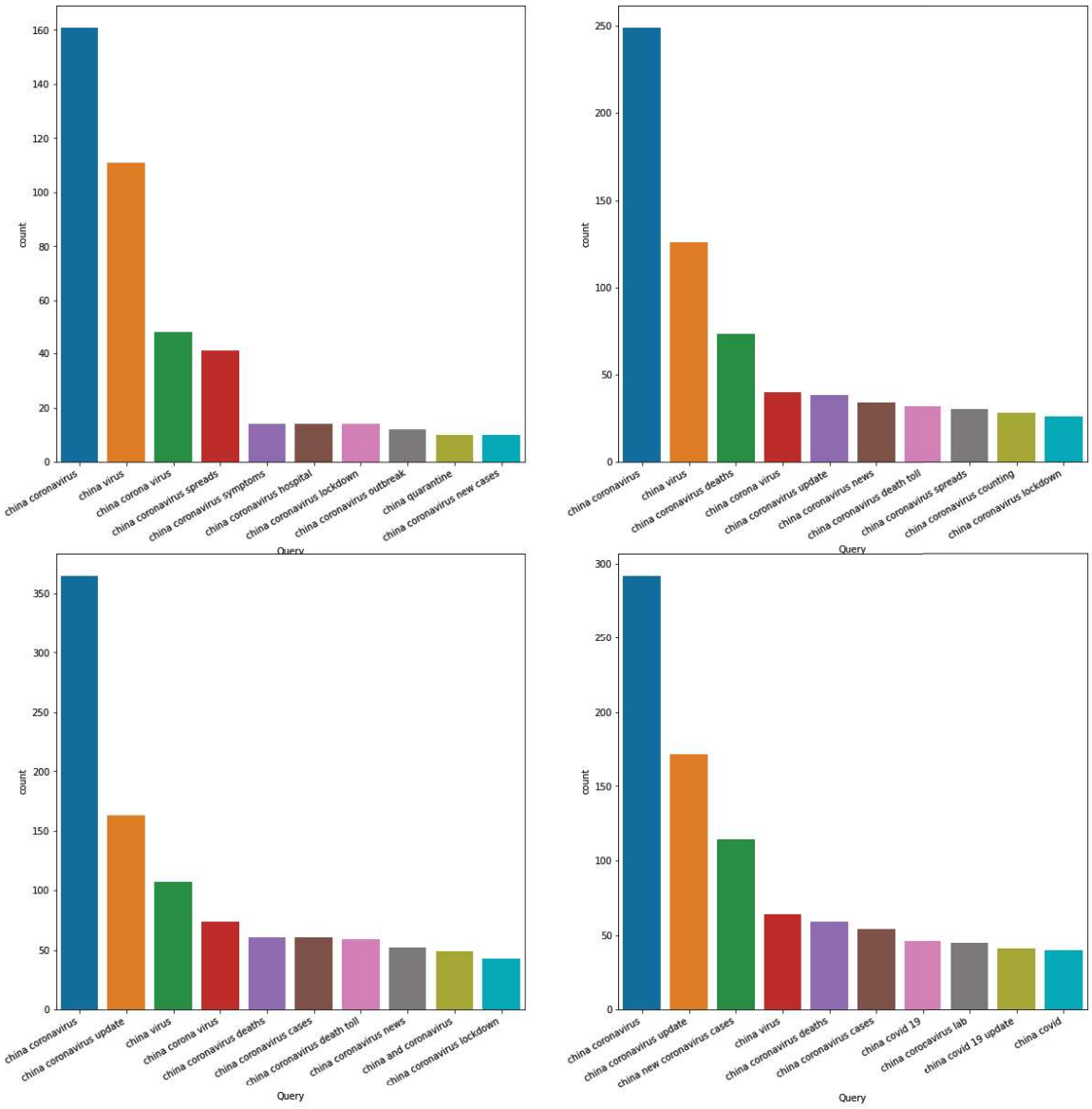
```
plot_multiple_queries("why")
```



Interesting insight :

In [ ]:

```
plot_multiple_queries("china")
```



This finishes our part for EDA.

Preprocessing and Modelling

In [ ]:

```
def clean_text(text):
    """
    helps in pre-processing the text data

    Parameters:
        text (str): a large raw text

    Returns:
        text (str) : processed string after various cleaning ops.
    """
    # converting to lower case
    text = text.lower()
    # removing unwanted chars using regular expressions
    text = re.compile('[/(){}\[\]\|@,;]').sub(' ', text)
    text = re.compile('[^0-9a-z #+_]').sub('', text)
    # removing words which are less than 3 chars and also which are stopwords
    text = ' '.join(word for word in text.split() if word not in STOPWORDS and word not
in punctuation and len(word) > 2)
    return text
```

Checking for any missing values (if any)

In [19]:

```
combined_country_data.isnull().sum()
```

Out[19]:

```
Date                0
Query               0
IsImplicitIntent    0
Country             0
PopularityScore     0
dtype: int64
```

checking if there are any queries corresponding to any country which has less than 50 entries (a threshold set by us in the beginning)

In [20]:

```
combined_country_data['Country'].value_counts()
```

Out[20]:

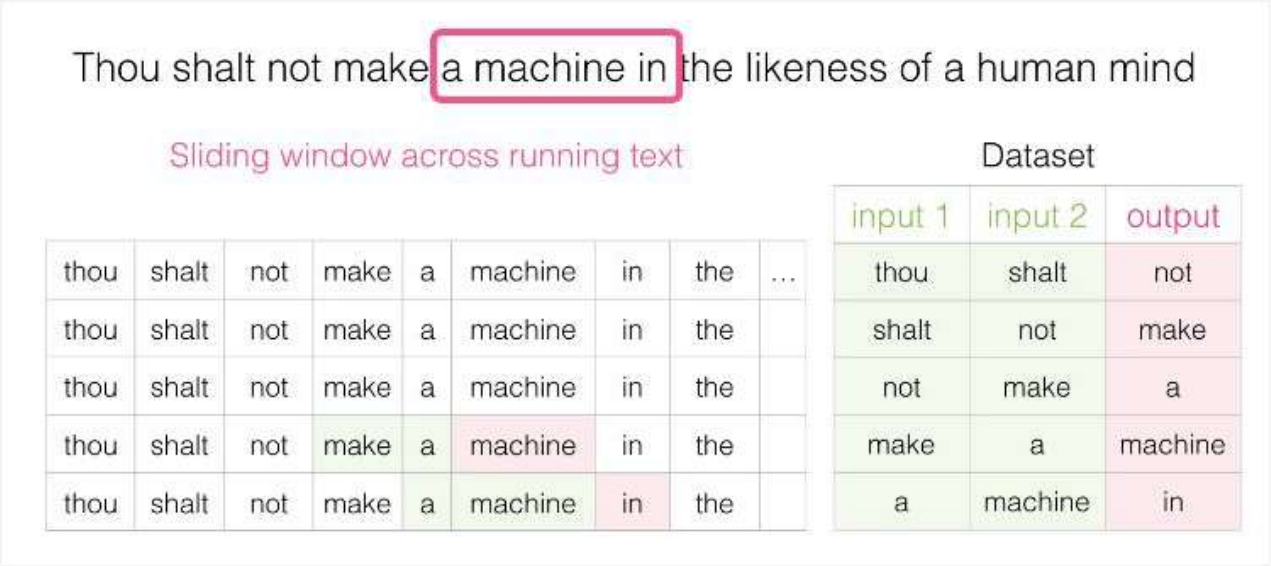
```
United States       916835
United Kingdom      198542
France              139741
Germany             111909
Italy                88790
                     ...
Monaco                  60
Togo                    57
Saint Lucia             56
Mali                    54
French Polynesia        53
Name: Country, Length: 156, dtype: int64
```

Now, we are going to create Word Embeddings - "Word2Vec" to analyse linguistic context between each terms

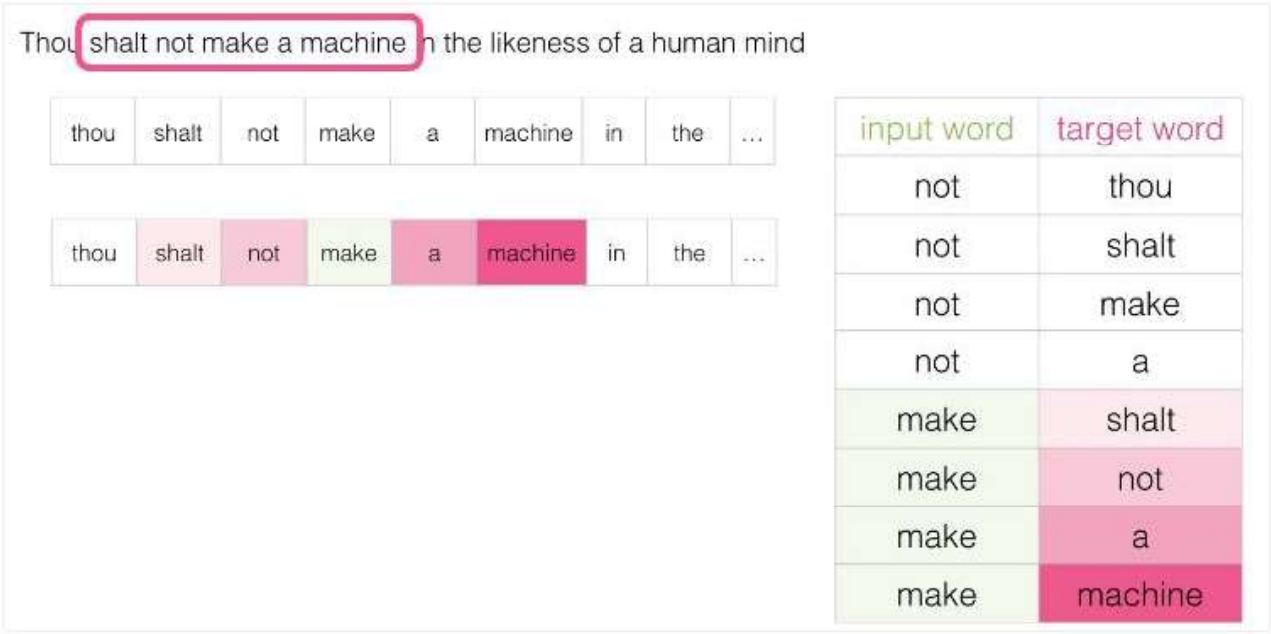
There are two approaches for using this algorithm :

- Continuos bag of words (CBOW)
- skim-gram

- CBOW IMAGE



- SKIM-GRAM IMAGE



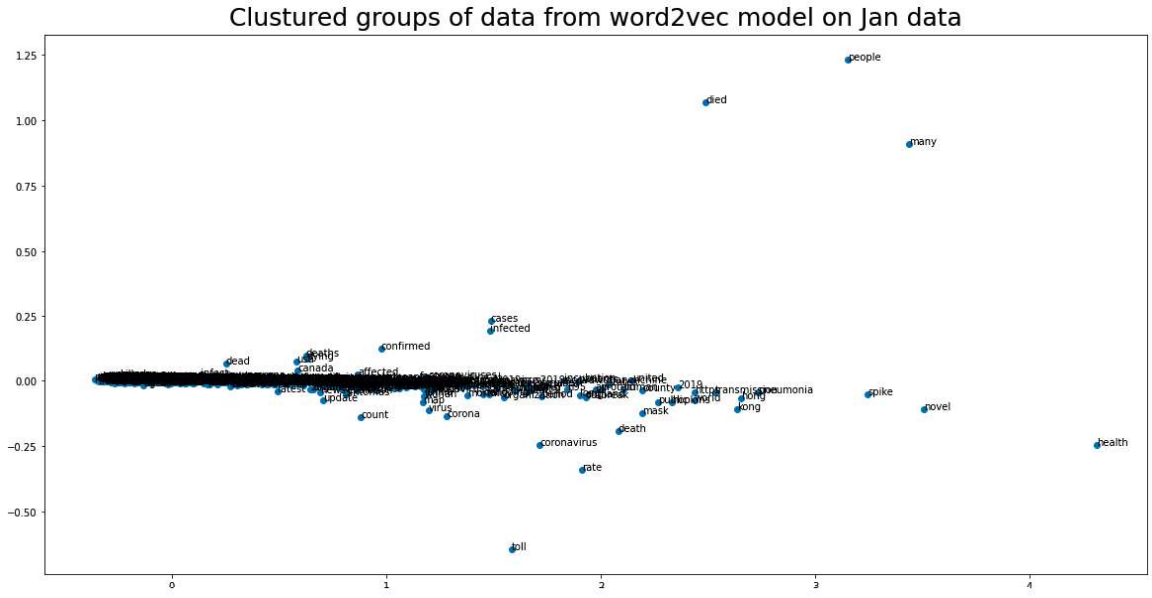
In [ ]:

```
def word_2_vec_plot(df, title=None):
  """
    helps in plotting PCA projection for clustered groups of words according to word em
bedding created using
    word2vec algorithm.

    Parameters:
        df (dataframe): Pandas dataframe object
        title (str) : title for the graph

    Returns:
        None
    """
  # preprocessing the dataframe rows
  df['Query'] = df['Query'].apply(clean_text)
  # creating 2-d nested list (matrix) of all cleaned rows
  sentences = [x.split(" ") for x in df['Query']]
  # Training word2vec model for our custom sentences
  model = Word2Vec(sentences, min_count=1)
  # Getting our vocan
  words = list(model.wv.vocab)
  # Getting our embedding
  X = model[model.wv.vocab]
  # Performing PCA projection on fetched embeddings
  pca = PCA(n_components=2)
  result = pca.fit_transform(X)
  # plotting PCA
  fig, ax = plt.subplots(1,1, figsize=(20, 10))
  ax.set_title(title,fontsize=25, pad=10)
  pyplot.scatter(result[:, 0], result[:, 1])
  # annotating our plot with actual words from the vocab
  for i, word in enumerate(words):
          pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
```

In [101]:

```
word_2_vec_plot(df_jan_country, "Clustured groups of data from word2vec model on Jan data")
```

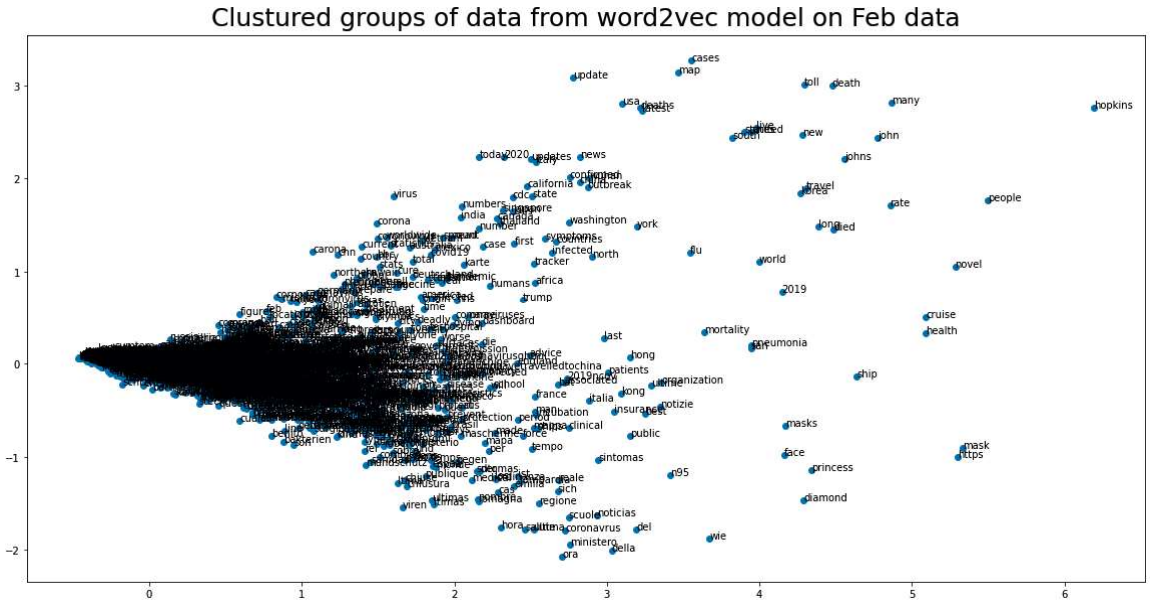
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  import sys
```



In [104]:

```
word_2_vec_plot(df_feb_country, "Clustured groups of data from word2vec model on Feb data")
```

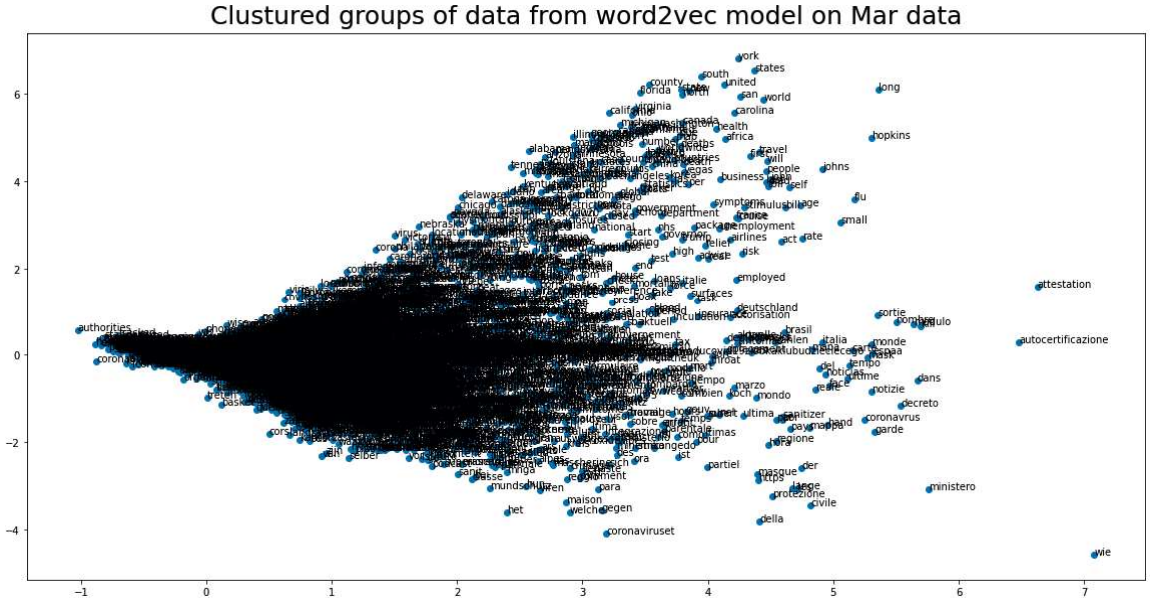
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  import sys
```



In [106]:

```
word_2_vec_plot(df_mar_country, "Clustured groups of data from word2vec model on Mar da
ta")
```

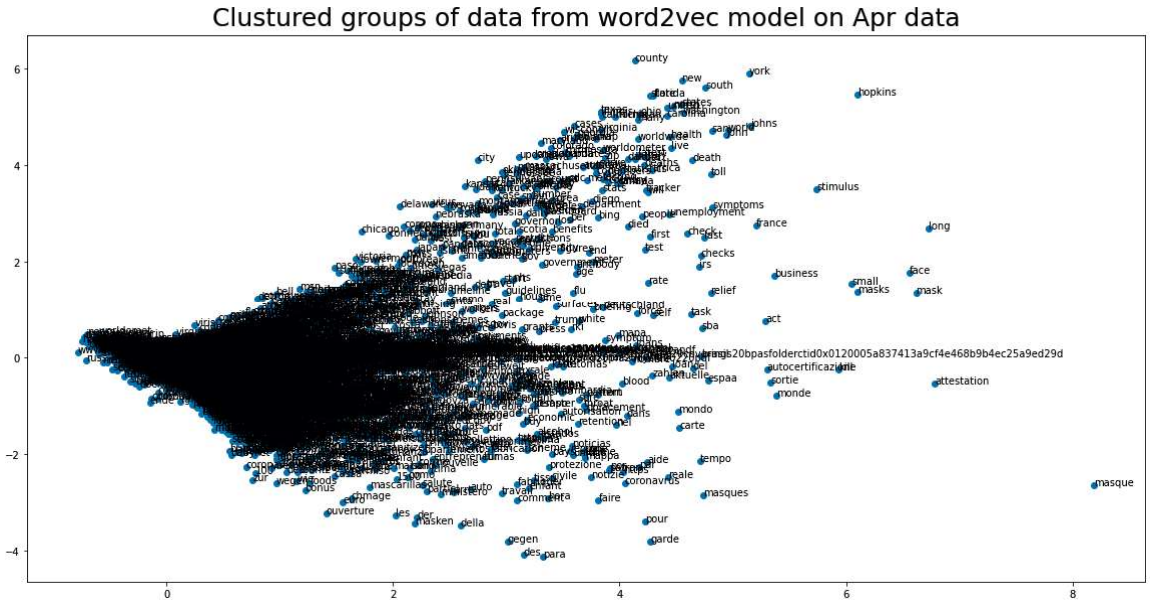
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: Deprecatio
nWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.
0, use self.wv.__getitem__() instead).
  import sys
```



In [107]:

```
word_2_vec_plot(df_apr_country, "Clustured groups of data from word2vec model on Apr da
ta")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: Deprecatio
nWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.
0, use self.wv.__getitem__() instead).
  import sys
```



Supervised algorithms for building classifiers

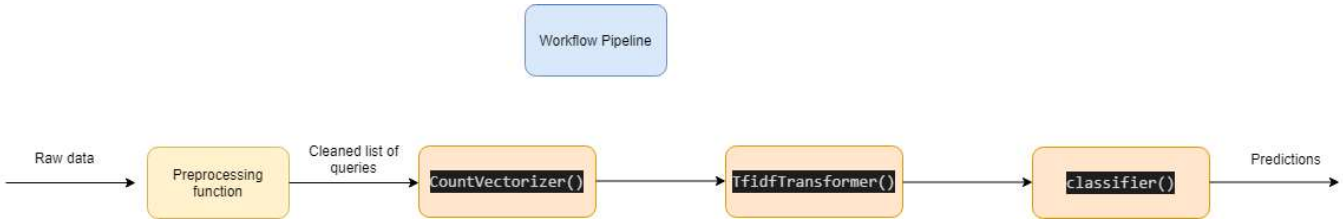
- Here I take Query as input to our model and predict the Country
- Since the classes are heavily imbalanced , we need to follow some best approaches to avoid overfitting for majority classes.

Here we are using Pipeline to make our ML workflow smooth.

Algorithm workflow :

- Convert the sentences in our query by tokenizing it and create a matrix containing count of each of word for each sentence.
- Then , we use tfidf transformer which taker this count matrix and computes tf and idf terms and create tf-idf matrix for our dataset.
- Finally , we are going to run our classifiers to check against different metrics.

Pipeline diagram :



In [ ]:

```
interested_country = [i for i, j in combined_country_data['Country'].value_counts().to_
dict().items() if j > 17000]
```

In [84]:

```
interested_country
```

Out[84]:

```
['United States',
 'United Kingdom',
 'France',
 'Germany',
 'Italy',
 'Canada',
 'Australia',
 'Spain',
 'India',
 'Brazil']
```

In [ ]:

```
dataset = combined_country_data[combined_country_data['Country'].isin(interested_countr
y)]
```

In [86]:

```
dataset['Country'].value_counts()
```

Out[86]:

```
United States     916835
United Kingdom    198542
France            139741
Germany           111909
Italy              88790
Canada             81914
Australia          40491
Spain              39733
India              21183
Brazil             17316
Name: Country, dtype: int64
```

In [ ]:

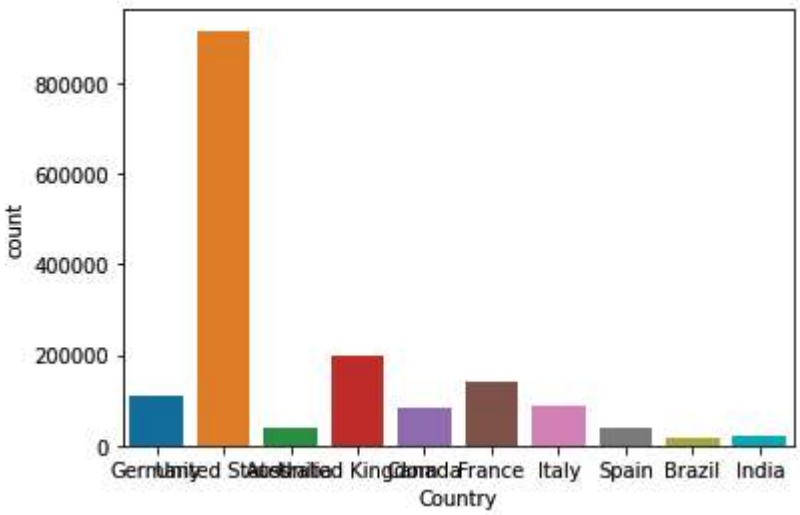
```
X = dataset['Query']
y = dataset['Country']
```

In [88]:

```
sns.countplot(y)
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4bf4685ef0>
```



As we can see there is highly imbalance in the dataset, so we will perform SMOTE (synthetic minority oversampling technique) to oversample the minority classes like india, brazil who have very less number of queries.

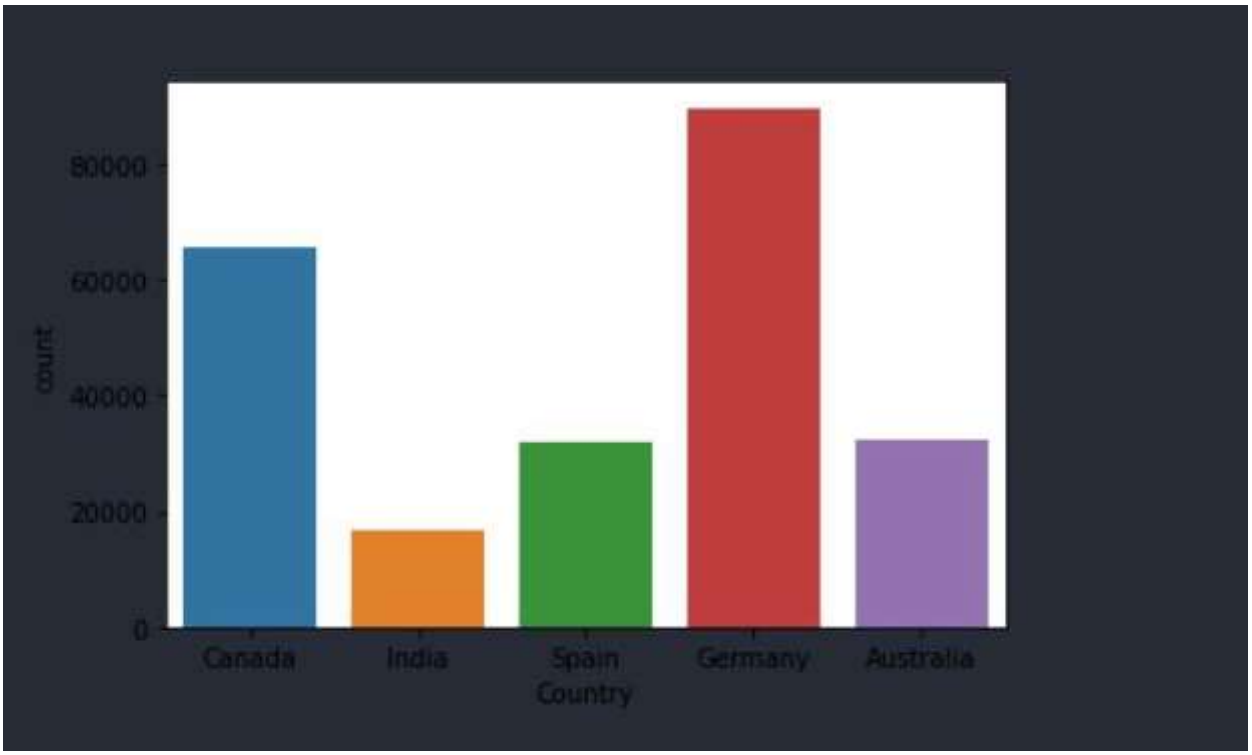
Also since this is a huge dataset with more than a million number of rows and hence I will be creating model for only a subset of classes which have less queries after perform sampling using SMOTE because model is taking much more longer time to train on original dataset, hence we will be performing training and evaluation on smaller subset of data.

For the first iteration, i have decided to include only 5 labels as of now as given below :

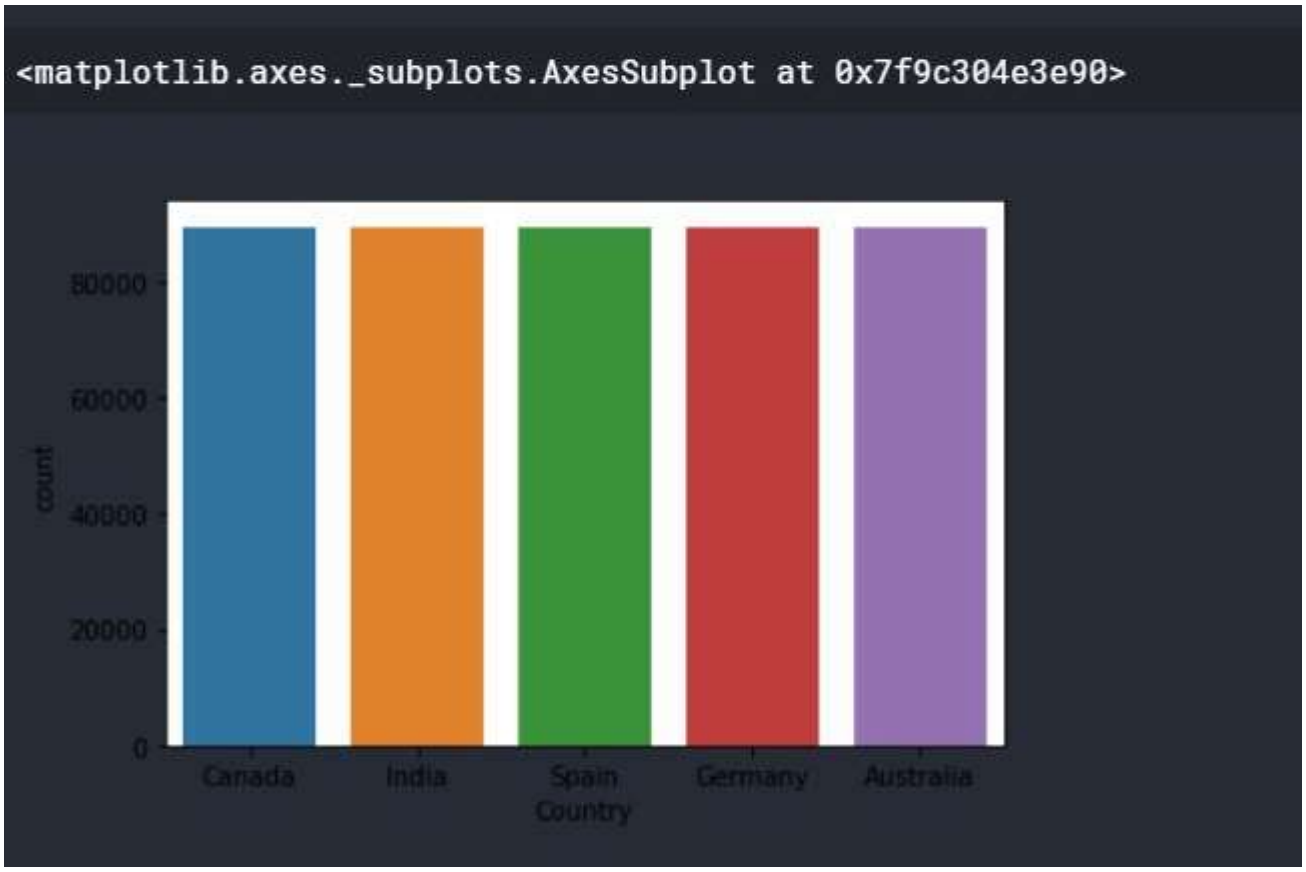
['Germany', 'Canada', 'Australia', 'Spain', 'India']

Here, i have shown how SMOTE performed really good on making all class labels as selected by me equally distributed.

Imbalanced version of data selected :



Balanced version of data selected after SMOTE:



In [21]:

```
interested_country = ['Germany', 'Canada', 'Australia', 'Spain', 'India']
dataset = combined_country_data[combined_country_data['Country'].isin(interested_countr
y)]
dataset['Country'].value_counts()
```

Out[21]:

```
Germany      111909
Canada        81914
Australia     40491
Spain         39733
India         21183
Name: Country, dtype: int64
```

In [ ]:

```
X = dataset['Query']  # fetching input to our model
y = dataset['Country'] # fetching labels for our model
# splitting the training set into 80-20 dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state =
0, stratify = y)
```

In [ ]:

```
# performing testing for checking shapes

assert X_train.shape == y_train.shape
assert X_test.shape == y_test.shape
```

In [ ]:

```
from imblearn.pipeline import Pipeline
def pipe_train(X_train, X_test, y_train, y_test, model):
    """
    helps in Training and evaluating the model with printed classification report

    Parameters:
        X_train (np.ndarray) : numpy array containing training set examples
        X_test (np.ndarray) : numpy array containing testing set examples
        y_train (np.ndarray) : numpy array containing training labels
        y_test (np.ndarray) : numpy array containing testing labels
        model (sklearn model object): scikit-learn classifier object

    Returns:
        pipeline object containing all the transformers and estimators
    """

  pipe = Pipeline([('vect', CountVectorizer()),
                   ('tfidf', TfidfTransformer()),
                   ('smt', SMOTE(random_state=777)),
                   ('clf', model)
                  ])
  print("fitting started")
  pipe.fit(X_train, y_train)
  print("Model trained succesfully !")
  print("scoring started....")
  print(accuracy_score(y_test, pipe.predict(X_test)))
  return pipe
```

Training SGDClassifier()

In [ ]:

```
sgd = SGDClassifier(verbose=5)
```

In [27]:

```
pipe_sgd = pipe_train(X_train, X_test, y_train, y_test, sgd)
```

```
fitting started

/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: Fu
tureWarning: Function safe_indexing is deprecated; safe_indexing is deprec
ated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.

-- Epoch 1
Norm: 20.57, NNZs: 7236, Bias: -0.945478, T: 447635, Avg. loss: 0.248232
Total training time: 0.13 seconds.
-- Epoch 2
Norm: 20.53, NNZs: 7737, Bias: -0.946131, T: 895270, Avg. loss: 0.236843
Total training time: 0.26 seconds.
-- Epoch 3
Norm: 20.52, NNZs: 7827, Bias: -0.947543, T: 1342905, Avg. loss: 0.236139
Total training time: 0.39 seconds.
-- Epoch 4
Norm: 20.51, NNZs: 7868, Bias: -0.946158, T: 1790540, Avg. loss: 0.235768
Total training time: 0.52 seconds.
-- Epoch 5
Norm: 20.49, NNZs: 7881, Bias: -0.946190, T: 2238175, Avg. loss: 0.235569
Total training time: 0.64 seconds.
-- Epoch 6
Norm: 20.49, NNZs: 7895, Bias: -0.947342, T: 2685810, Avg. loss: 0.235471
Total training time: 0.77 seconds.
-- Epoch 7
Norm: 20.49, NNZs: 7907, Bias: -0.946837, T: 3133445, Avg. loss: 0.235414
Total training time: 0.89 seconds.
Convergence after 7 epochs took 0.89 seconds
-- Epoch 1

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.9s remaining:
0.0s
```

```
Norm: 22.64, NNZs: 7864, Bias: -0.873047, T: 447635, Avg. loss: 0.246351
Total training time: 0.13 seconds.
-- Epoch 2
Norm: 22.70, NNZs: 8146, Bias: -0.879636, T: 895270, Avg. loss: 0.234989
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 22.69, NNZs: 8198, Bias: -0.881262, T: 1342905, Avg. loss: 0.234191
Total training time: 0.38 seconds.
-- Epoch 4
Norm: 22.71, NNZs: 8225, Bias: -0.883798, T: 1790540, Avg. loss: 0.233834
Total training time: 0.51 seconds.
-- Epoch 5
Norm: 22.72, NNZs: 8240, Bias: -0.885428, T: 2238175, Avg. loss: 0.233662
Total training time: 0.63 seconds.
-- Epoch 6
Norm: 22.71, NNZs: 8248, Bias: -0.887643, T: 2685810, Avg. loss: 0.233484
Total training time: 0.76 seconds.
-- Epoch 7
Norm: 22.72, NNZs: 8251, Bias: -0.888489, T: 3133445, Avg. loss: 0.233424
Total training time: 0.88 seconds.
Convergence after 7 epochs took 0.89 seconds
-- Epoch 1
Norm: 35.01, NNZs: 8338, Bias: -0.234415, T: 447635, Avg. loss: 0.134188
Total training time: 0.12 seconds.
-- Epoch 2


[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.8s remaining:
0.0s

Norm: 35.07, NNZs: 8361, Bias: -0.242938, T: 895270, Avg. loss: 0.124251
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 35.13, NNZs: 8363, Bias: -0.244397, T: 1342905, Avg. loss: 0.123439
Total training time: 0.38 seconds.
-- Epoch 4
Norm: 35.16, NNZs: 8363, Bias: -0.244650, T: 1790540, Avg. loss: 0.122996
Total training time: 0.51 seconds.
-- Epoch 5
Norm: 35.18, NNZs: 8363, Bias: -0.245965, T: 2238175, Avg. loss: 0.122738
Total training time: 0.64 seconds.
-- Epoch 6
Norm: 35.19, NNZs: 8363, Bias: -0.247022, T: 2685810, Avg. loss: 0.122605
Total training time: 0.76 seconds.
-- Epoch 7
Norm: 35.20, NNZs: 8363, Bias: -0.246968, T: 3133445, Avg. loss: 0.122486
Total training time: 0.89 seconds.
Convergence after 7 epochs took 0.89 seconds
-- Epoch 1

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    2.7s remaining:
0.0s
```

```
Norm: 19.52, NNZs: 5275, Bias: -0.986456, T: 447635, Avg. loss: 0.236568
Total training time: 0.13 seconds.
-- Epoch 2
Norm: 19.53, NNZs: 6709, Bias: -0.983421, T: 895270, Avg. loss: 0.227686
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 19.50, NNZs: 6867, Bias: -0.977486, T: 1342905, Avg. loss: 0.226754
Total training time: 0.37 seconds.
-- Epoch 4
Norm: 19.54, NNZs: 6940, Bias: -0.980611, T: 1790540, Avg. loss: 0.226527
Total training time: 0.50 seconds.
-- Epoch 5
Norm: 19.54, NNZs: 7020, Bias: -0.977067, T: 2238175, Avg. loss: 0.226316
Total training time: 0.63 seconds.
-- Epoch 6
Norm: 19.54, NNZs: 7065, Bias: -0.977700, T: 2685810, Avg. loss: 0.226158
Total training time: 0.75 seconds.
-- Epoch 7
Norm: 19.53, NNZs: 7099, Bias: -0.976681, T: 3133445, Avg. loss: 0.226090
Total training time: 0.88 seconds.
Convergence after 7 epochs took 0.88 seconds
-- Epoch 1
Norm: 29.04, NNZs: 8007, Bias: -0.749700, T: 447635, Avg. loss: 0.099179
Total training time: 0.12 seconds.
-- Epoch 2

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    3.6s remaining:
0.0s

Norm: 29.18, NNZs: 8165, Bias: -0.754862, T: 895270, Avg. loss: 0.092110
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 29.19, NNZs: 8189, Bias: -0.754895, T: 1342905, Avg. loss: 0.091437
Total training time: 0.37 seconds.
-- Epoch 4
Norm: 29.22, NNZs: 8199, Bias: -0.755517, T: 1790540, Avg. loss: 0.091210
Total training time: 0.49 seconds.
-- Epoch 5
Norm: 29.23, NNZs: 8204, Bias: -0.756351, T: 2238175, Avg. loss: 0.091075
Total training time: 0.62 seconds.
-- Epoch 6
Norm: 29.24, NNZs: 8208, Bias: -0.756468, T: 2685810, Avg. loss: 0.090927
Total training time: 0.74 seconds.
-- Epoch 7
Norm: 29.24, NNZs: 8209, Bias: -0.757220, T: 3133445, Avg. loss: 0.090863
Total training time: 0.87 seconds.
Convergence after 7 epochs took 0.87 seconds
Model trained succesfully !
scoring started....

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    4.5s finished

0.776208379907191
```

Training LinearSVC()

In [28]:

```
svc = LinearSVC(verbose=5)
pipe_svc = pipe_train(X_train, X_test, y_train, y_test, svc)
```

```
fitting started

/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: Fu
tureWarning: Function safe_indexing is deprecated; safe_indexing is deprec
ated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)

[LibLinear]Model trained succesfully !
scoring started....
0.7906886156555906
```

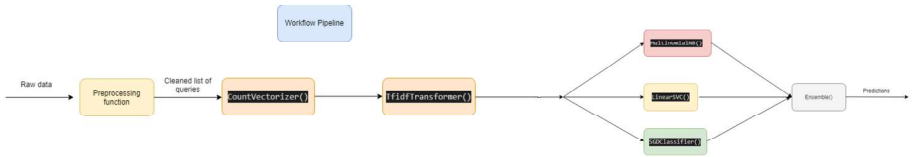
Training MultinomialNB()

In [29]:

```
mnb = MultinomialNB()
pipe_mnb = pipe_train(X_train, X_test, y_train, y_test, mnb)
```

```
fitting started

/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: Fu
tureWarning: Function safe_indexing is deprecated; safe_indexing is deprec
ated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)

Model trained succesfully !
scoring started....
0.7883175829014667
```

Creating Ensemble of classifiers by stacking all of the above classifiers



In [31]:

```
pipe_ensemble = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('smt', SMOTE(random_state=777))
                    ])

ensemble = VotingClassifier(estimators=[('sgd', pipe_sgd['clf']), ('svc', pipe_svc['cl
f']), ('mnb', pipe_mnb['clf'])], voting='soft')
pipe_ensemble.steps.append(['ensemble', ensemble])
pipe_ensemble
```

Out[31]:

```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='ut
f-8',
                                 input='content', lowercase=True, max_df=
1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=Non...
                                                 warm_start=Fa
lse)),
                                           ('svc',
                                            LinearSVC(C=1.0,
                                                      class_weight=Non
e,
                                                      dual=True,
                                                      fit_intercept=Tru
e,
                                                      intercept_scaling
=1,
                                                      loss='squared_hin
ge',
                                                      max_iter=1000,
                                                      multi_class='ov
r',
                                                      penalty='l2',
                                                      random_state=Non
e,
                                                      tol=0.0001,
                                                      verbose=5)),
                                           ('mnb',
                                            MultinomialNB(alpha=1.0,
                                                          class_prior=N
one,
                                                          fit_prior=Tru
e))],
                                  flatten_transform=True, n_jobs=None,
                                  voting='soft', weights=None)]],
         verbose=False)
```

In [32]:

```
pipe_ensemble.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: Fu
tureWarning: Function safe_indexing is deprecated; safe_indexing is deprec
ated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.

-- Epoch 1
Norm: 20.47, NNZs: 7247, Bias: -0.945934, T: 447635, Avg. loss: 0.248530
Total training time: 0.12 seconds.
-- Epoch 2
Norm: 20.52, NNZs: 7751, Bias: -0.946498, T: 895270, Avg. loss: 0.236876
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 20.50, NNZs: 7832, Bias: -0.946174, T: 1342905, Avg. loss: 0.236133
Total training time: 0.37 seconds.
-- Epoch 4
Norm: 20.47, NNZs: 7868, Bias: -0.945557, T: 1790540, Avg. loss: 0.235818
Total training time: 0.50 seconds.
-- Epoch 5
Norm: 20.48, NNZs: 7889, Bias: -0.946826, T: 2238175, Avg. loss: 0.235580
Total training time: 0.63 seconds.
-- Epoch 6
Norm: 20.48, NNZs: 7897, Bias: -0.946779, T: 2685810, Avg. loss: 0.235475
Total training time: 0.75 seconds.
-- Epoch 7
Norm: 20.49, NNZs: 7906, Bias: -0.946653, T: 3133445, Avg. loss: 0.235425
Total training time: 0.87 seconds.
Convergence after 7 epochs took 0.87 seconds
-- Epoch 1

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.9s remaining:
0.0s

Norm: 22.72, NNZs: 7872, Bias: -0.870694, T: 447635, Avg. loss: 0.246603
Total training time: 0.12 seconds.
-- Epoch 2
Norm: 22.66, NNZs: 8139, Bias: -0.879605, T: 895270, Avg. loss: 0.235158
Total training time: 0.24 seconds.
-- Epoch 3
Norm: 22.72, NNZs: 8179, Bias: -0.881625, T: 1342905, Avg. loss: 0.234222
Total training time: 0.35 seconds.
-- Epoch 4
Norm: 22.70, NNZs: 8209, Bias: -0.885348, T: 1790540, Avg. loss: 0.233764
Total training time: 0.47 seconds.
-- Epoch 5
Norm: 22.70, NNZs: 8218, Bias: -0.885496, T: 2238175, Avg. loss: 0.233619
Total training time: 0.59 seconds.
-- Epoch 6
Norm: 22.71, NNZs: 8221, Bias: -0.886015, T: 2685810, Avg. loss: 0.233481
Total training time: 0.72 seconds.
-- Epoch 7
Norm: 22.72, NNZs: 8228, Bias: -0.887521, T: 3133445, Avg. loss: 0.233412
Total training time: 0.84 seconds.
Convergence after 7 epochs took 0.84 seconds
-- Epoch 1
Norm: 35.00, NNZs: 8341, Bias: -0.237717, T: 447635, Avg. loss: 0.133919
Total training time: 0.12 seconds.
-- Epoch 2
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.7s remaining:
0.0s

Norm: 35.11, NNZs: 8361, Bias: -0.241922, T: 895270, Avg. loss: 0.124408
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 35.14, NNZs: 8361, Bias: -0.242321, T: 1342905, Avg. loss: 0.123315
Total training time: 0.37 seconds.
-- Epoch 4
Norm: 35.17, NNZs: 8363, Bias: -0.243303, T: 1790540, Avg. loss: 0.123013
Total training time: 0.50 seconds.
-- Epoch 5
Norm: 35.18, NNZs: 8363, Bias: -0.246115, T: 2238175, Avg. loss: 0.122707
Total training time: 0.62 seconds.
-- Epoch 6
Norm: 35.19, NNZs: 8363, Bias: -0.246576, T: 2685810, Avg. loss: 0.122609
Total training time: 0.74 seconds.
-- Epoch 7
Norm: 35.20, NNZs: 8363, Bias: -0.246754, T: 3133445, Avg. loss: 0.122503
Total training time: 0.87 seconds.
-- Epoch 8
Norm: 35.20, NNZs: 8363, Bias: -0.247420, T: 3581080, Avg. loss: 0.122392
Total training time: 0.99 seconds.
Convergence after 8 epochs took 0.99 seconds
-- Epoch 1
Norm: 19.67, NNZs: 5249, Bias: -0.991042, T: 447635, Avg. loss: 0.236452
Total training time: 0.12 seconds.
-- Epoch 2

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    2.7s remaining:
0.0s

Norm: 19.57, NNZs: 6675, Bias: -0.984670, T: 895270, Avg. loss: 0.227516
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 19.56, NNZs: 6862, Bias: -0.980866, T: 1342905, Avg. loss: 0.226781
Total training time: 0.38 seconds.
-- Epoch 4
Norm: 19.55, NNZs: 6953, Bias: -0.979802, T: 1790540, Avg. loss: 0.226444
Total training time: 0.50 seconds.
-- Epoch 5
Norm: 19.55, NNZs: 7010, Bias: -0.978267, T: 2238175, Avg. loss: 0.226311
Total training time: 0.63 seconds.
-- Epoch 6
Norm: 19.55, NNZs: 7045, Bias: -0.977222, T: 2685810, Avg. loss: 0.226180
Total training time: 0.76 seconds.
-- Epoch 7
Norm: 19.54, NNZs: 7088, Bias: -0.977008, T: 3133445, Avg. loss: 0.226083
Total training time: 0.89 seconds.
Convergence after 7 epochs took 0.89 seconds
-- Epoch 1

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    3.6s remaining:
0.0s
```

```
Norm: 29.08, NNZs: 8037, Bias: -0.752047, T: 447635, Avg. loss: 0.099387
Total training time: 0.13 seconds.
-- Epoch 2
Norm: 29.18, NNZs: 8181, Bias: -0.754499, T: 895270, Avg. loss: 0.092239
Total training time: 0.25 seconds.
-- Epoch 3
Norm: 29.20, NNZs: 8199, Bias: -0.753992, T: 1342905, Avg. loss: 0.091455
Total training time: 0.38 seconds.
-- Epoch 4
Norm: 29.23, NNZs: 8207, Bias: -0.755776, T: 1790540, Avg. loss: 0.091193
Total training time: 0.50 seconds.
-- Epoch 5
Norm: 29.24, NNZs: 8213, Bias: -0.755865, T: 2238175, Avg. loss: 0.091011
Total training time: 0.62 seconds.
-- Epoch 6
Norm: 29.25, NNZs: 8217, Bias: -0.756567, T: 2685810, Avg. loss: 0.090891
Total training time: 0.74 seconds.
-- Epoch 7
Norm: 29.25, NNZs: 8217, Bias: -0.757112, T: 3133445, Avg. loss: 0.090830
Total training time: 0.86 seconds.
Convergence after 7 epochs took 0.86 seconds
[LibLinear]

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    4.5s finished
```

Out[32]:

```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='ut
f-8',
                                 input='content', lowercase=True, max_df=
1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=Non...
                                                          warm_start=Fa
lse)),
                                            ('svc',
                                             LinearSVC(C=1.0,
                                                       class_weight=Non
e,
                                                       dual=True,
                                                       fit_intercept=Tru
e,
                                                       intercept_scaling
=1,
                                                       loss='squared_hin
ge',
                                                       max_iter=1000,
                                                       multi_class='ov
r',
                                                       penalty='l2',
                                                       random_state=Non
e,
                                                       tol=0.0001,
                                                       verbose=5)),
                                            ('mnb',
                                             MultinomialNB(alpha=1.0,
                                                           class_prior=N
one,
                                                           fit_prior=Tru
e))],
                                 flatten_transform=True, n_jobs=None,
                                 voting='soft', weights=None)]],
         verbose=False)
```

Okay, now let's try to predict on some random queries -

In [37]:

```
# predicting using each one of trained classifier, the results will be same as everyone got almost equal accuracy

print(pipe_sgd.predict(['angela merkel take on coronavirus']))
print(pipe_svc.predict(['angela merkel take on coronavirus']))
print(pipe_mnb.predict(['angela merkel take on coronavirus']))
```

```
['Germany']
['Germany']
['Germany']
```

In [40]:

```
print(pipe_sgd.predict(['coronavirus Updates in india']))
print(pipe_svc.predict(['coronavirus Updates in india']))
print(pipe_mnb.predict(['coronavirus Updates in india']))
```

```
['India']
['India']
['India']
```

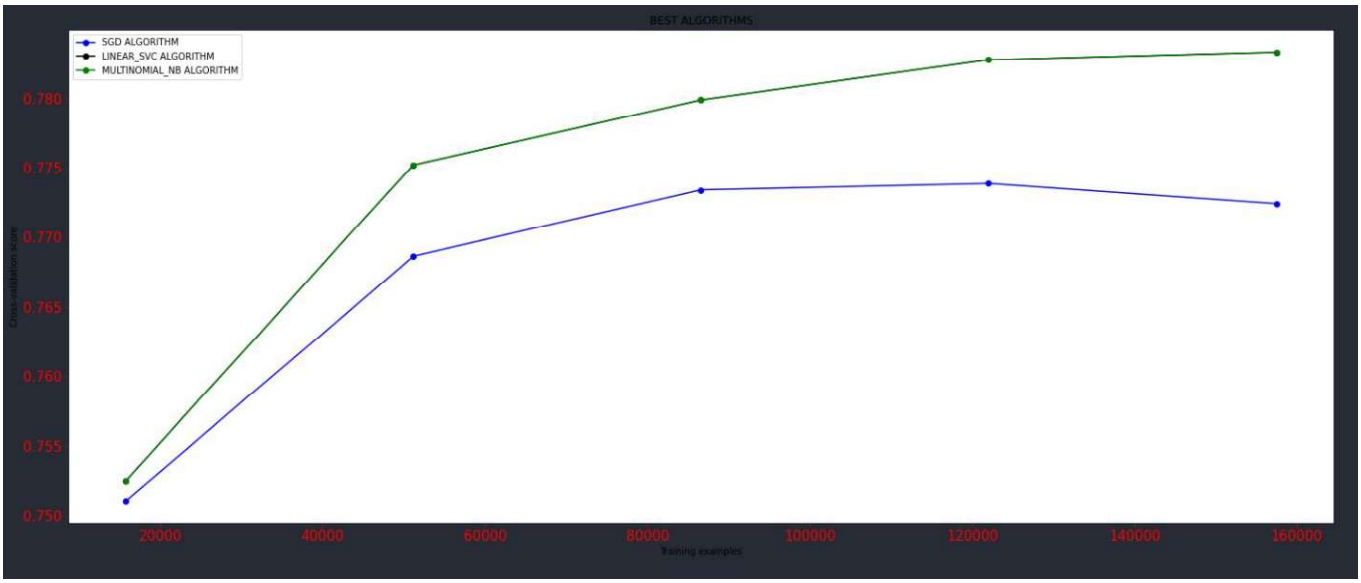
Cross validated Evaluation

In [ ]:

```
def plot_best(X_train, y_train):
    """
    plotting all best algorithms on basis of cross validation scores

    Parameters:
        X_train (dataframe): training features
        y_train (dataframe): training target values

    Returns:
        None
    """


    test_scores_mean = []

    train_sizes, train_scores, test_scores = learning_curve(pipe_sgd, X_train, y_train,
cv=5, random_state=0)
    test_scores_mean.append(np.mean(test_scores, axis=1))

    train_sizes, train_scores, test_scores = learning_curve(pipe_svc, X_train, y_train,
cv=5, random_state=0)
    test_scores_mean.append(np.mean(test_scores, axis=1))

    train_sizes, train_scores, test_scores = learning_curve(pipe_mnb, X_train, y_train,
cv=5, random_state=0)
    test_scores_mean.append(np.mean(test_scores, axis=1))

    print("Plotting final metrics cross validation scores for all algorithms : ")

    fig, ax =plt.subplots(1,1, figsize=(25, 10))
    plt.xlabel("Training examples")
    plt.ylabel("Cross-validation score")
    plt.title('BEST ALGORITHMS')
    plt.plot(train_sizes, test_scores_mean[0], 'o-', color="b", label="SGD ALGORITHM")
    plt.plot(train_sizes, test_scores_mean[2], 'o-', color="k", label="LINEAR_SVC ALGOR
ITHM")
    plt.plot(train_sizes, test_scores_mean[1], 'o-', color="g", label="MULTINOMIAL_NB A
LGORITHM")
    plt.legend(loc='best')
    plt.show()
```

In [ ]:

```
plot_best(X_train, y_train)
```



In [68]:

```
y_pred_sgd = pipe_sgd.predict(X_test)
conf_mat_sgd = confusion_matrix(y_test, y_pred_sgd) # getting confusion matrix
print(metrics.classification_report(y_test, y_pred_sgd, target_names=dataset['Country']
.unique())) # getting classification metrics such as recall, precision
```

```
              precision    recall  f1-score   support

     Germany       0.65      0.63      0.64      8098
   Australia       0.86      0.64      0.73     16383
      Canada       0.92      0.88      0.90     22382
       Spain       0.38      0.83      0.52      4237
       India       0.85      0.88      0.86      7946

    accuracy                           0.78     59046
   macro avg       0.73      0.77      0.73     59046
weighted avg       0.82      0.78      0.79     59046
```

In [71]:

```
y_pred_mnb = pipe_mnb.predict(X_test)
conf_mat_mnb = confusion_matrix(y_test, y_pred_mnb) # getting confusion matrix
print(metrics.classification_report(y_test, y_pred_mnb, target_names=dataset['Country']
.unique())) # getting classification metrics such as recall, precision
```

```
              precision    recall  f1-score   support

     Germany       0.65      0.65      0.65      8098
   Australia       0.83      0.69      0.75     16383
      Canada       0.97      0.87      0.92     22382
       Spain       0.38      0.85      0.52      4237
       India       0.88      0.88      0.88      7946

    accuracy                           0.79     59046
   macro avg       0.74      0.79      0.75     59046
weighted avg       0.84      0.79      0.80     59046
```

In [72]:

```
y_pred_svc = pipe_mnb.predict(X_test)
conf_mat_svc = confusion_matrix(y_test, y_pred_svc) # getting confusion matrix
print(metrics.classification_report(y_test, y_pred_svc, target_names=dataset['Country']
.unique())) # getting classification metrics such as recall, precision
```

```
              precision    recall  f1-score   support

     Germany       0.65      0.65      0.65      8098
   Australia       0.83      0.69      0.75     16383
      Canada       0.97      0.87      0.92     22382
       Spain       0.38      0.85      0.52      4237
       India       0.88      0.88      0.88      7946

    accuracy                           0.79     59046
   macro avg       0.74      0.79      0.75     59046
weighted avg       0.84      0.79      0.80     59046
```

Saving the trained model

In [45]:

```
joblib.dump(pipe_ensemble, 'model.pkl')
```

Out[45]:

```
['model.pkl']
```

Deployment and Hosting

- The code for the backend deployment is available at my github repo: https://github.com/souravs17031999/MicrosoftBing-search-query-prediction (https://github.com/souravs17031999/MicrosoftBing-search-query-prediction)
- Server Link (Live website) : https://souravsdlboy.pythonanywhere.com/bing_search (https://souravsdlboy.pythonanywhere.com/bing_search)

Future Scope :

1. More classifiers can be combined to form bigger ensembles
2. Pre-trained models like transformers, BERT etc. can be used
3. Dataset can be augmented using Neural Machine translation for better diversity of training examples.
4. It can be extended for more number of classes (labels) maybe upto more than 30-40 labels using pre trained models.

References:

- Relevant Course material from course Lab sessions
- http://www.nltk.org/
- https://machinelearningmastery.com/develop-word-embeddings-python-gensim/
- http://scikit-learn.org/stable/modules/clustering.html
- SMOTE PAPER: https://arxiv.org/pdf/1106.1813.pdf
- WORD2VEC PAPER : https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
- http://jalammar.github.io/illustrated-word2vec/
- https://huggingface.co/transformers/