SENTIMENT ANALYSIS ON COVID-19 VACCINATION USING TWITTER DATA

Developed by:

Name: Anand Vadavelli, Red-ID:- 823389639

Name:- Mayur Bagwe, Red-ID:- 825903748

Overview:

For this project, we used tweepy API to get all the vaccination tweets realted to Covid-19 by maintianing a list of Covid-19 keywords. We decided to do the sentimental analysis on this data set using pandas and pyspark dataframe. We identified the most postive, negative and neutral tweets based on polarity scores. We also performed a topic model analysis on these tweets

Please **DOWNLOAD** the dataset for this project using this **link**

You can also download the project from our **Github Link**

Learning:

SENTIMENTAL ANALYSIS

With this project we had a great learning curve towards pyspark dataframes and understood how it is different from pandas dataframe. We got to know how to use sql with spark dataframes. We had an opportunity to learn how spark streaming can be done with a twitter developer account.This project made us explore the word cloud and plotting of it based on different interpolations. We got a chance to learn how to perform pre-processing techniques like removing twitter handles, Punctuations, links, Numbers and Tokenization, Stemming. We got an opportunity to use Vader for Sentimental Analysis and to calculate different accuracy scores like pos, neu, neg and compound. We also learned about SentimentIntensityAnalyzer and implemented in this project.

TOPIC MODEL ANALYSIS

From our data analysis we figured out most of the tweets are neutral.Then we also analyzed these tweets using Topic Model Analysis, there we learnt a lot many things like Gensim Phrase model and also about Natural Language Tool Kit(NLTK), how to tokenize the sentences and clean them. Then we also came across how to create the bigram and trigram models and concept of lemmatization also. Found that LDA topic model can be used to print the topics that are most prevalent or dominant.We found the importance of pyLDAvis for the visualization of important topics and keywords and clearly observed that when executing it and seeing the results. Also gained knowledge in how to calculate the perplexity and coherence score which is used to measure if the model chosen is good or not.

TWITTER API USING TWEEPY

We tried different techniques to get real time twitter data. The one that stood up for us in readily available tweepy library. Tweepy provides an api to search for the tweets using any of the keywork you are interested in. Each api call returns 100 tweets. We prepared a list of Coivd-19 related keywords that can used along with vaccination. Then we used search_tweets api to query the recent tweets on those topics. We were able to read the tweets along with the date posted, hashtags and also learnt that spark was efficient in processing unrestricted limit of data.

Issues Faced:

1.Initally setting up spark on local boxes did have issues. we learned how to fix those issues
2.Though the syntax of spark dataframe is a bit similar to pandas we had challenges on finding out the syntax for spark dataframe.
3.Faced problem in extracting more than 100 tweets through tweepy api's.
4.With twitter steaming we were able to read the tweets continously however we couldn't get the tweets to jupyter notebook
5.Using spacy became a problem for us therefore we had to run anaconda and install spacy realted packages as admin

Prerequisities:

Before running this notebook, please install the following libraries:

```
pip install tweepy
pip install matplotlib
pip install seaborn
pip install pandas
pip install wordcloud
pip install nltk
pip install punkt
pip install vaderSentiment
pip install gensim
pip install textblob
pip install pyspark
pip install findspark
```

```
            u'([\u2600-\u27BF])|([\uD83C][\uDF00-\uDFFF])|([\uD83D][\uDC00-\uDE4F])|([\uD83D][\uDE80-\uDEFF])')
    return emojis_pattern


def get_hashtags_pattern():
    return re.compile(r'#\w*')


def get_single_letter_words_pattern():
    return re.compile(r'(?<![\w\-])\w(?![\w\-])')


def get_blank_spaces_pattern():
    return re.compile(r'\s{2,}|\t')


def get_twitter_reserved_words_pattern():
    return re.compile(r'(RT|rt|FAV|fav|VIA|via)')


def get_mentions_pattern():
    return re.compile(r'@\w*')


def is_year(text):
    if (len(text) == 3 or len(text) == 4) and (MIN_YEAR < len(text) < MAX_YEAR):
        return True
    else:
        return False
```

```
class TwitterPreprocessor:

    def __init__(self, text: str):
        self.text = text

    def fully_preprocess(self):
        return self \
            .remove_urls() \
            .remove_mentions() \
            .remove_hashtags() \
            .remove_twitter_reserved_words() \
            .remove_punctuation() \
            .remove_single_letter_words() \
            .remove_blank_spaces() \
            .remove_stopwords() \
            .remove_numbers()

    def remove_urls(self):
        self.text = re.sub(pattern=get_url_patern(), repl='', string=self.text)
        return self

    def remove_punctuation(self):
        self.text = self.text.translate(str.maketrans('', '', string.punctuation))
        return self

    def remove_mentions(self):
        self.text = re.sub(pattern=get_mentions_pattern(), repl='', string=self.text)
        return self

    def remove_hashtags(self):
        self.text = re.sub(pattern=get_hashtags_pattern(), repl='', string=self.text)
        return self
```

```
    def remove_twitter_reserved_words(self):
        self.text = re.sub(pattern=get_twitter_reserved_words_pattern(), repl='', string=self.text)
        return self

    def remove_single_letter_words(self):
        self.text = re.sub(pattern=get_single_letter_words_pattern(), repl='', string=self.text)
        return self

    def remove_blank_spaces(self):
        self.text = re.sub(pattern=get_blank_spaces_pattern(), repl=' ', string=self.text)
        return self

    def remove_stopwords(self, extra_stopwords=None):
        if extra_stopwords is None:
            extra_stopwords = []
        text = nltk.word_tokenize(self.text)
        stop_words = set(stopwords.words('english'))

        new_sentence = []
        for w in text:
            if w not in stop_words and w not in extra_stopwords:
                new_sentence.append(w)
        self.text = ' '.join(new_sentence)
        return self

    def remove_numbers(self, preserve_years=False):
        text_list = self.text.split(' ')
        for text in text_list:
```

```
            if text.isnumeric():
                if preserve_years:
                    if not is_year(text):
                        text_list.remove(text)
                else:
                    text_list.remove(text)

        self.text = ' '.join(text_list)
        return self

    def lowercase(self):
        self.text = self.text.lower()
        return self
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\avadavelli\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\avadavelli\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Now lets process the data cleaning on tweets column and append it to a list

In [15]:
```
%%time
tweets = [str(row.tweet) for row in vaccination_tweets.collect()]
clean_tweets = []
for tweet in tweets:
    c = TwitterPreprocessor((tweet))
    c.fully_preprocess()
    c = c.text
    clean_tweets.append(c)
```

```
Wall time: 14.6 s
```

Lets create a pandas dataframe for the processed list and convert into spark dataframe

In [16]:
```
%%time
sqlContext = SQLContext(sc)
cleaned_tweets_pandas = pd.DataFrame(clean_tweets, columns={'clean_tweets'})
cleaned_tweets = spark.createDataFrame(cleaned_tweets_pandas)
```

```
Wall time: 395 ms
```

Lets look at the processed tweets spark dataframe

In [17]:
```
cleaned_tweets.head(10)
```

Out[17]:
```
[Row(clean_tweets='The key issue transmission drive impact illness amp pressure NHS Severity amp clinical spectrum il…'),
 Row(clean_tweets='Look happened Canada vaccination passes made mandatory use public transpo'),
 Row(clean_tweets='Let clear ’ emphasized past Japan ’ success strategy exemplifying wide use mas…'),
 Row(clean_tweets='medical practitioners scientists call end mass vaccination …'),
 Row(clean_tweets='The omicron variant Texas Now time get vaccine booster ’ already Vis…'),
 Row(clean_tweets='🔘ATM Says NO To Mandatory Vaccination🔘 The African Transformation Movement says rejects mandatory vaccinatio
n The …'),
 Row(clean_tweets='NEW Germany Father shot wife children employer found forged wifes vac…'),
 Row(clean_tweets='Riddle highest rate today …'),
 Row(clean_tweets='BRAZIL VACCINATION IS AN INDIVIDUAL CHOICE The President reiterates implement vaxx obligatio…'),
 Row(clean_tweets='Canada wonderful example world Higher Education Higher Vaccination Thank Neighbor♡💔')]
```

Lets us merge the cleaned tweets dataframe with the orginial dataframe

In [18]:
```
%%time
vaccination_tweets = vaccination_tweets.withColumn("id", monotonically_increasing_id())
cleaned_tweets = cleaned_tweets.withColumn("id", monotonically_increasing_id())
vaccination_tweets = cleaned_tweets.join(vaccination_tweets, "id", "outer").drop("id")
```

```
Wall time: 71.7 ms
```

Let us look at the combined dataframe

In [19]:
```
vaccination_tweets.show()
```

```
+--------------------+----------+--------------------+--------------------+
|        clean_tweets|      date|               tweet|            hashtags|
+--------------------+----------+--------------------+--------------------+
|FREE Skills Sessi...|2021-12-07|RT @BasketballCB:...|                  []|
|The Austrian Bish...|2021-12-07|RT @ATschugguel: ...|                  []|
|“ Weve got long w...|2021-12-07|RT @TreWardTV: “W...|                  []|
|Stop knows experi...|2021-12-07|#StoptheShot! Sto...|['#StoptheShot', ...|
|Preliminary resul...|2021-12-07|Preliminary resul...|                  []|
|Optimistic commen...|2021-12-07|Optimistic commen...|                  []|
|Agreed hear mains...|2021-12-07|@BellaWallerstei ...|                  []|
|They bad outbreak...|2021-12-07|@slurpmynutts @Ja...|                  []|
|The ability flu2 ...|2021-12-07|@InfoPEI The abil...|                  []|
|It ’ side effect ...|2021-12-07|@AriCohn @Popehat...|                  []|
|Just got 3rd Covi...|2021-12-07|Just got my 3rd C...|                  []|
|got Moderna boost...|2021-12-07|@POTUS I got the ...|                  []|
|                   …|2021-12-07|.@POTUS @BorisJoh...|                  []|
|Mark Meadows trai...|2021-12-07|Mark Meadows is a...|                  []|
|Please Sir provid...|2021-12-07|Please Sir @Imran...|                  []|
|ZyCoVD used seven...|2021-12-03|ZyCoV-D to be use...|                  []|
|People wearing fa...|2021-12-02|People wearing fa...|                  []|
|Are they… fliing Tae|2021-12-07|Are they… flirtin...|                  []|
|Omg😭😭😭would ’ ...|2021-12-07|Omg😭😭😭would’ve...|                  []|
|Bring bow burning...|2021-12-07|Bring me my bow o...|                  []|
```

```
+------------------+----------+------------------+--------------------+
only showing top 20 rows
```

Data Analysis

We will now look at the words that are commonly used in tweets. For this we will use word cloud visualization

In word cloud, the words that are frequently used will appear is bigger font

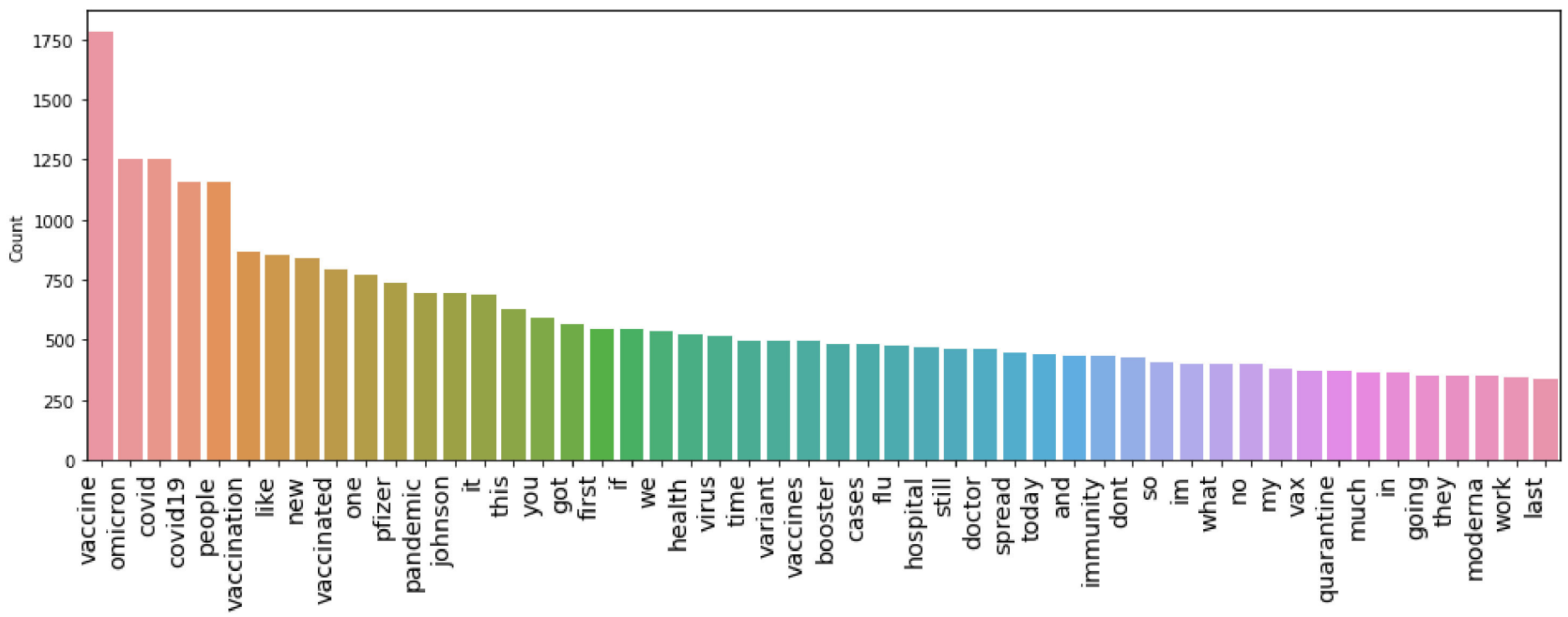
For this we will first get all words in the tweets as a list

In [20]:
```
%%time
tweets_row_list = vaccination_tweets.select('clean_tweets').collect()
tweets_string_list = [ ele.__getattr__('clean_tweets') for ele in tweets_row_list]
all_tweet_words = ' '.join(tweets_string_list)
```

```
Wall time: 6.45 s
```

Plot the top 10 keywords use with vaccination

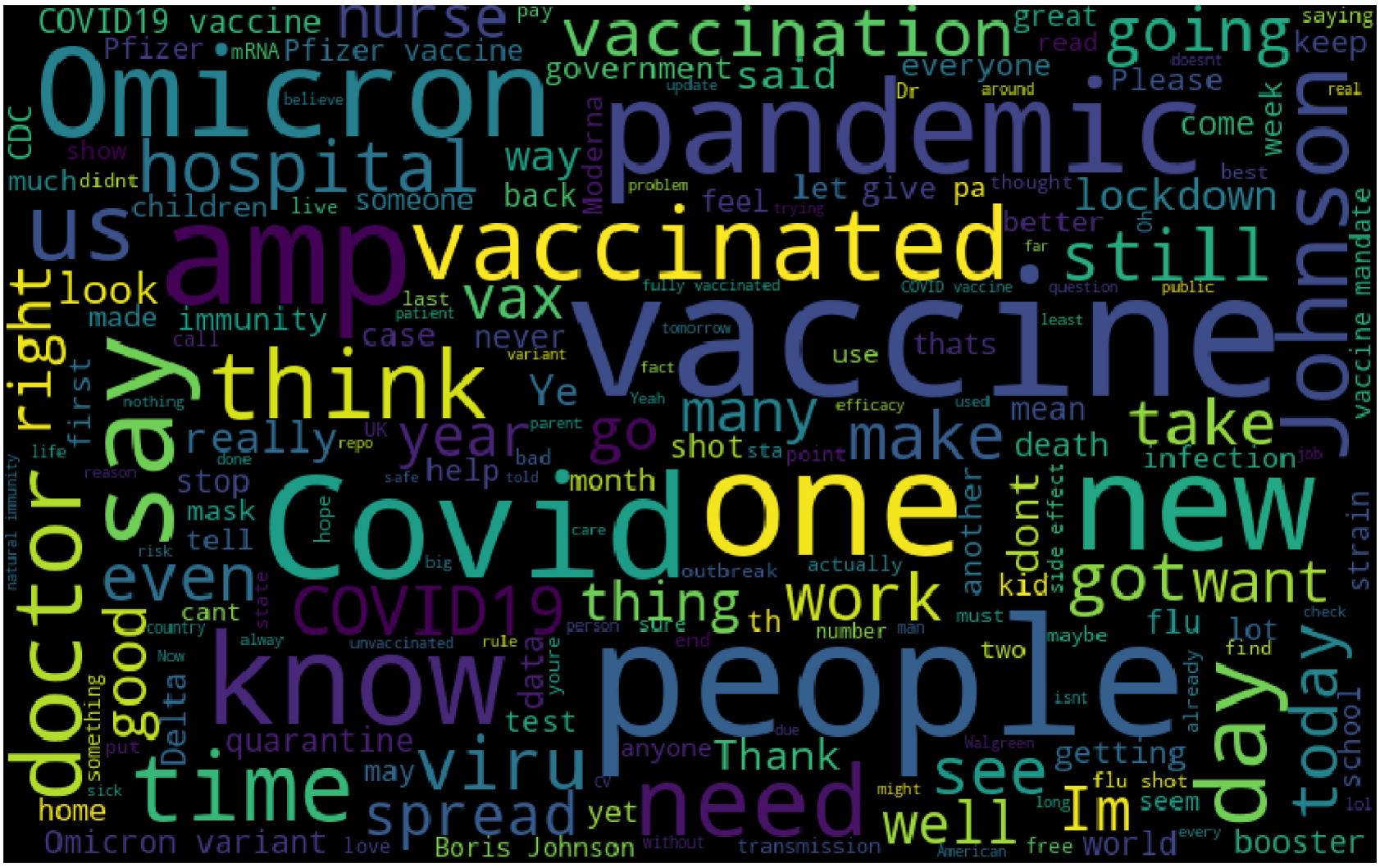
In [21]:
```
%%time
all_tweet_words
counts = dict()
words = all_tweet_words.split()
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use', 'not', 'would', 'say', 'could', '_',\
                   'be', 'know', 'good', 'go', 'get', 'do', 'done', 'try', 'many', 'some', 'nice',\
                   'thank', 'think', 'see', 'rather', 'easy', 'easily', 'lot', 'lack', 'make', 'want',\
                   'seem', 'run', 'need', 'even', 'right', 'line', 'even', 'also', 'may', 'take', 'come',\
                   '’', '…', 'The', '“', 'amp', '”', 'it', 'this', 'you', 'if', 'us'])
words = [x for x in words if x not in stop_words]
words = [x.lower() for x in words]

for word in words:
    if word in counts:
        counts[word] += 1
    else:
        counts[word] = 1
# NLTK Stop words
counts = sorted(counts.items(), key=lambda x: x[1], reverse=True)
counts
x_counts = [x[0] for x in counts]
y_counts = [x[1] for x in counts]

plt.figure(figsize=(16,5))
ax = sns.barplot(x= x_counts[:50], y = y_counts[:50])
ax.set(ylabel = 'Count')
plt.xticks(
    rotation=90,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
plt.show()
```




```
Wall time: 1.51 s
```

Plot the word cloud of most words used in tweets

In [22]:
```
%%time
wordcloud = WordCloud(width=800, height=500, random_state=None, max_font_size=100).generate(all_tweet_words)

plt.figure(figsize=(15, 10))
plt.imshow(wordcloud, interpolation="nearest")
plt.axis('off')
plt.show()
```




```
Wall time: 2.41 s
```

Sentiment Analysis Using VADER

Vader has the ability to handles emojis and slangs. It also considers capitalization and punctuations when giving the scores

Vader gives 4 types of values, positive, negative, neutral and compound

In [23]:
```
nltk.download("vader_lexicon")
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\avadavelli\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[23]: True

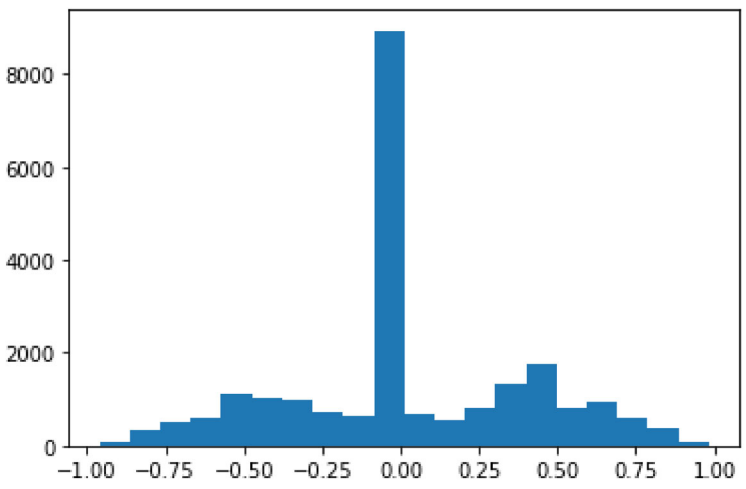
Create a sentiment intensity analyzer object

Plot a graph based on the tweets compund polarity score

In [24]:
```
%%time
analyzer = SentimentIntensityAnalyzer()
scores = [analyzer.polarity_scores(tweet)['compound'] for tweet in tweets_string_list]
plt.hist(scores, bins=20)
```

```
Wall time: 3.09 s
```

Out[24]:
```
(array([  88.,  330.,  529.,  602., 1126., 1028.,  979.,  709.,  629.,
        8936.,  683.,  555.,  813., 1335., 1779.,  826.,  948.,  590.,
         391.,   80.]),
 array([-0.9612 , -0.86397, -0.76674, -0.66951, -0.57228, -0.47505,
        -0.37782, -0.28059, -0.18336, -0.08613,  0.0111 ,  0.10833,
         0.20556,  0.30279,  0.40002,  0.49725,  0.59448,  0.69171,
         0.78894,  0.88617,  0.9834 ]),
 <BarContainer object of 20 artists>)
```



From the above graph we can see the most of our tweets and neutral

Let us calculate the pos, neu, neg and compund polarity scores on each tweet and save it

Note: We tried to save the polarity scores directly to the dataframe with the below commented logic however performing calculation on these dataframes was taking very long time. Also we couldn't figure out how with a single lamda function we could save all the polarity scores instead of calculating it individually.

So we decided to calculate polarity score to a list and then merge to the original dataframe

In [25]:
```
tweet_compound = []
tweet_pos = []
tweet_neu = []
tweet_neg = []
```

In [26]:
```
%%time
for row in vaccination_tweets.rdd.toLocalIterator():
    polarity_score = analyzer.polarity_scores(row['clean_tweets'])
    tweet_pos.append(polarity_score['pos'])
    tweet_neu.append(polarity_score['neu'])
    tweet_neg.append(polarity_score['neg'])
    tweet_compound.append(polarity_score['compound'])
```

```
Wall time: 11 s
```

Create a spark datafram for the pos, neu, neg and compund polarity scores

In [27]:
```
%%time
polarity_scores = sqlContext.createDataFrame(zip(tweet_pos, tweet_neu, tweet_neg, tweet_compound), \
                                             schema=['tweet_pos', 'tweet_neu', 'tweet_neg', 'tweet_compound'])
```

```
Wall time: 873 ms
```

Let us look at polarity values summary

In [28]:
```
polarity_scores.describe().show()
```

```
+-------+------------------+------------------+------------------+-------------------+
|summary|         tweet_pos|          tweet_neu|          tweet_neg|     tweet_compound|
+-------+------------------+------------------+------------------+-------------------+
|  count|             22956|             22956|             22956|              22956|
|   mean|0.12573575535807613| 0.759060637741766|0.10100509670674294|0.03457141488064121|
| stddev|0.17543428828255211|0.23259310969201946|0.15836365573650618|0.37863130631594655|
|    min|               0.0|               0.0|               0.0|             -0.9612|
|    max|               1.0|               1.0|               1.0|             0.9834|
+-------+------------------+------------------+------------------+-------------------+
```

Let us merge the polarity scores with the original dataframe

Though there are many ways to merge lists to spark dataframe, we preffered using spark sql to acheive this

In [29]:
```
%%time
polarity_scores = polarity_scores.withColumn("id", monotonically_increasing_id())
polarity_scores.createOrReplaceTempView("metrics")
polarity_scores_sql = spark.sql("SELECT * FROM metrics")
# polarity_scores_sql.show()
```

```
Wall time: 278 ms
```

In [30]:
```
%%time
vaccination_tweets = vaccination_tweets.withColumn("id", monotonically_increasing_id())
vaccination_tweets.createOrReplaceTempView("tweets")
vaccination_tweets_sql = spark.sql("SELECT * FROM tweets")
# vaccination_tweets_sql.show()
```

```
Wall time: 39.4 ms
```

In [31]:
```
%%time
vaccination_tweets_merged = vaccination_tweets_sql.join(polarity_scores_sql, 'id', 'inner')
```

```
Wall time: 18 ms
```

Let us look at the our new dataframe with the polarity values

In [32]:
```
%%time
vaccination_tweets_merged.show()
vaccination_tweets_merged.printSchema()
# vaccination_tweets_merged.describe().show()
```

```
+---+--------------------+----------+--------------------+--------------------+---------+---------+---------+--------------+
| id|        clean_tweets|      date|               tweet|            hashtags|tweet_pos|tweet_neu|tweet_neg|tweet_compound|
+---+--------------------+----------+--------------------+--------------------+---------+---------+---------+--------------+
| 26|Its interesting H...|2021-12-07|@davidallengreen ...|                  []|    0.231|    0.769|      0.0|        0.4019|
| 29|posttraumatic str...|2021-12-07|post-traumatic st...|                  []|    0.156|    0.391|    0.453|       -0.6486|
| 65|Inseions SARSCoV2...|2021-12-07|Insertions in the...|['#biorxiv_bioinfo']|      0.0|      1.0|      0.0|           0.0|
| 19|Bring bow burning...|2021-12-07|Bring me my bow o...|                  []|    0.184|    0.816|      0.0|        0.4019|
| 54|Want work CVS Hea...|2021-12-07|Want to work at C...|                  []|    0.091|    0.909|      0.0|        0.0772|
|  0|FREE Skills Sessi...|2021-12-07|RT @BasketballCB:...|                  []|    0.428|    0.572|      0.0|        0.7739|
| 22|This measured “ p...|2021-12-07|This was measured...|                  []|      0.0|      1.0|      0.0|           0.0|
|  7|They bad outbreak...|2021-12-07|@slurpmynutts @Ja...|                  []|      0.0|    0.551|    0.449|       -0.6908|
| 77|Of Coloradans cur...|2021-12-07|Of the Coloradans...|                  []|      0.0|      1.0|      0.0|           0.0|
| 34|This December Fro...|2021-12-07|This December 8! ...|                  []|    0.182|    0.818|      0.0|        0.4404|
| 50|Instead weak nega...|2021-12-07|'Instead, there w...|                  []|    0.117|    0.455|    0.429|       -0.7003|
| 94|What Its really g...|2021-12-07|@holder_casper Wh...|                  []|    0.285|    0.715|      0.0|        0.4927|
| 57|shocked spencer b...|2021-12-07|shocked that spen...|                  []|    0.192|    0.678|     0.13|        0.2732|
| 32|        ’ sorry hear|2021-12-07|@sabsabeni I’m so...|                  []|      0.0|    0.435|    0.565|       -0.0772|
| 43|Thanks Walgreens ...|2021-12-02|Thanks to Walgree...|['#GetVaccinated'...|     0.73|     0.27|      0.0|        0.6597|
| 84|New IPO smallcap ...|2021-12-07|@realwillmeade - ...|                  []|    0.199|    0.646|    0.155|        0.1742|
| 31|The couple take s...|2021-12-07|The couple had to...|                  []|      0.0|    0.621|    0.379|       -0.7717|
| 39|Sorry intrude … W...|2021-12-07|@ACarpenDigital @...|                  []|      0.0|    0.755|    0.245|       -0.0772|
| 98|alarmist damn Omi...|2021-12-08|@maureviv so you ...|                  []|      0.0|    0.722|    0.278|       -0.4019|
```

```
| 25|’ honestly surpri...|2021-12-07|@Foxylad35309982 ...|                  []|     0.62|     0.38|      0.0|        0.5994|
+---+--------------------+----------+--------------------+--------------------+---------+---------+---------+--------------+
only showing top 20 rows

root
 |-- id: long (nullable = false)
 |-- clean_tweets: string (nullable = true)
 |-- date: date (nullable = true)
 |-- tweet: string (nullable = true)
 |-- hashtags: string (nullable = true)
 |-- tweet_pos: double (nullable = true)
 |-- tweet_neu: double (nullable = true)
 |-- tweet_neg: double (nullable = true)
 |-- tweet_compound: double (nullable = true)

Wall time: 58.4 s
```

Lets analyze on how the hashtags contributed to the tweets sentiment

In [33]:
```
def hashtag_extract(x):
    hashtags = []
    # Loop over the words in the tweet
    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)

    return hashtags
```

Get the hastags with compound score <, >, and =0 in to seperate dataframes

In [34]:
```
%%time
vaccination_tweets_merged.createOrReplaceTempView("tweets")
neutral_hashtag = spark.sql("SELECT hashtags FROM tweets WHERE tweet_compound = 0")
negative_hashtag = spark.sql("SELECT hashtags FROM tweets WHERE tweet_compound < 0")
positive_hashtag = spark.sql("SELECT hashtags FROM tweets WHERE tweet_compound > 0")
```

```
Wall time: 125 ms
```

Hashtags with positive polarity

Calculating the number of positive tweets

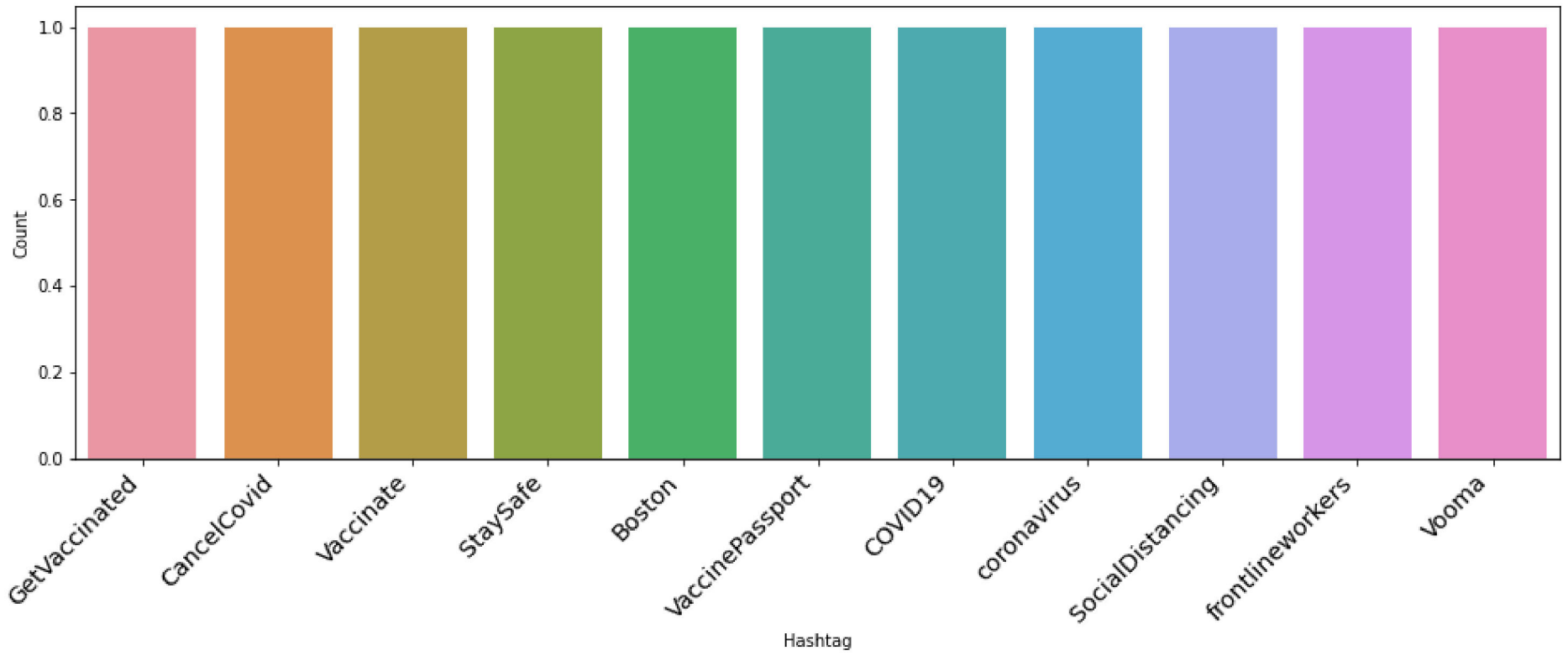
In [35]:
```
%%time
positive_hashtag_row_list = positive_hashtag.collect()
positive_hashtag_string_list = [ ele.__getattr__('hashtags') for ele in positive_hashtag_row_list]
positive_tweets=hashtag_extract(positive_hashtag_string_list)
positive_tweets = sum(positive_tweets,[])
```

```
Wall time: 1min 6s
```

Plot a heat map and a bar graph for the positive tweets

Note: We tried to plot directly using the spark dataframe however we couldn't get much help of plotting through spark dataframe therfore used pandas dataframe for this usecase

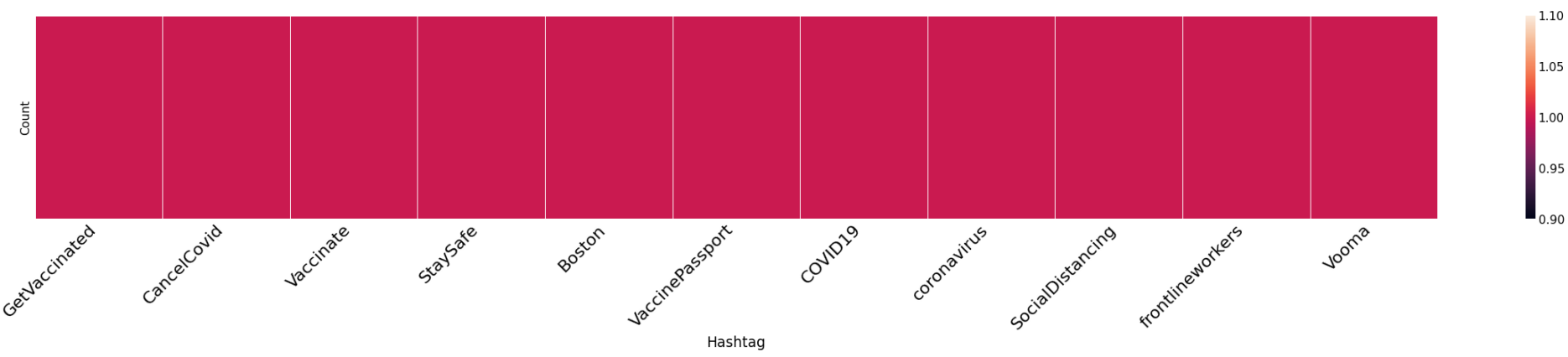
In [36]:
```
%%time
pos_tweets = nltk.FreqDist(positive_tweets)
pos_tweets_df = pd.DataFrame({'Hashtag': list(pos_tweets.keys()),
                  'Count': list(pos_tweets.values())})

# selecting top 20 most frequent hashtags
pos_tweets_df = pos_tweets_df.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=pos_tweets_df, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
plt.show()
```




```
Wall time: 287 ms
```

In [37]:

```
%%time
pos_tweets_df.set_index('Hashtag', inplace=True)
pos_tweets_df = pos_tweets_df.T
# Plot a heatmap for these hashtags
from matplotlib.pyplot import figure
figure(num=None, figsize=(30, 6), dpi=80, facecolor='w', edgecolor='k')
plt.style.use('fivethirtyeight')
ax = sns.heatmap(linewidth=0.5, data=pos_tweets_df)
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
plt.tight_layout()
plt.show()
```




```
Wall time: 391 ms
```

Hashtags with negative polarity

Calculating the number of negative tweets

In [38]:

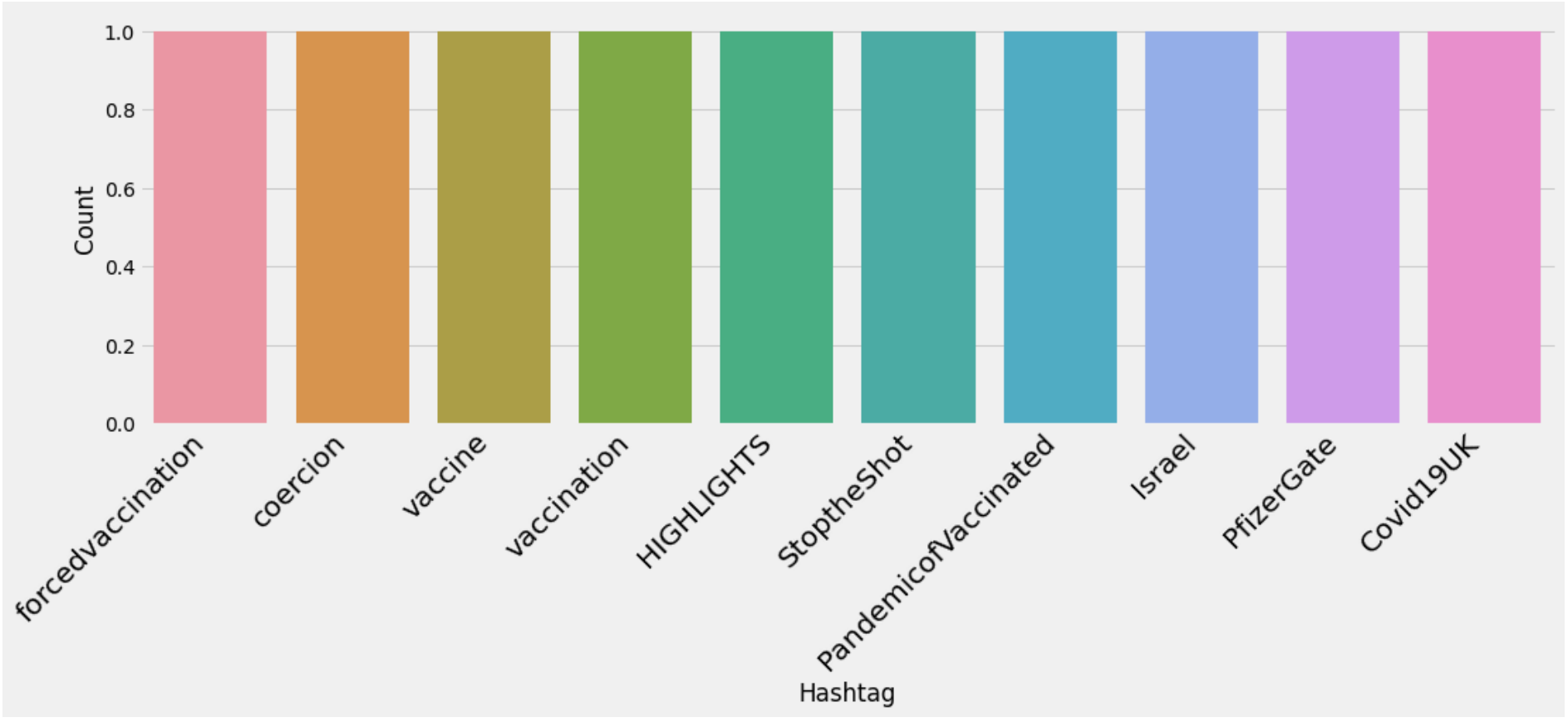
```
%%time
negative_hashtag_row_list = negative_hashtag.collect()
negative_hashtag_string_list = [ ele.__getattr__('hashtags') for ele in negative_hashtag_row_list]
negative_tweets=hashtag_extract(negative_hashtag_string_list)
negative_tweets = sum(negative_tweets,[])
```

```
Wall time: 1min 12s
```

Plot bar graph and heatmap for the negative tweets

In [39]:

```
%%time
neg_tweets = nltk.FreqDist(negative_tweets)
neg_tweets_df = pd.DataFrame({'Hashtag': list(neg_tweets.keys()),
                  'Count': list(neg_tweets.values())})

# selecting top 20 most frequent hashtags
neg_tweets_df = neg_tweets_df.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=neg_tweets_df, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
plt.show()
```



Wall time: 232 ms

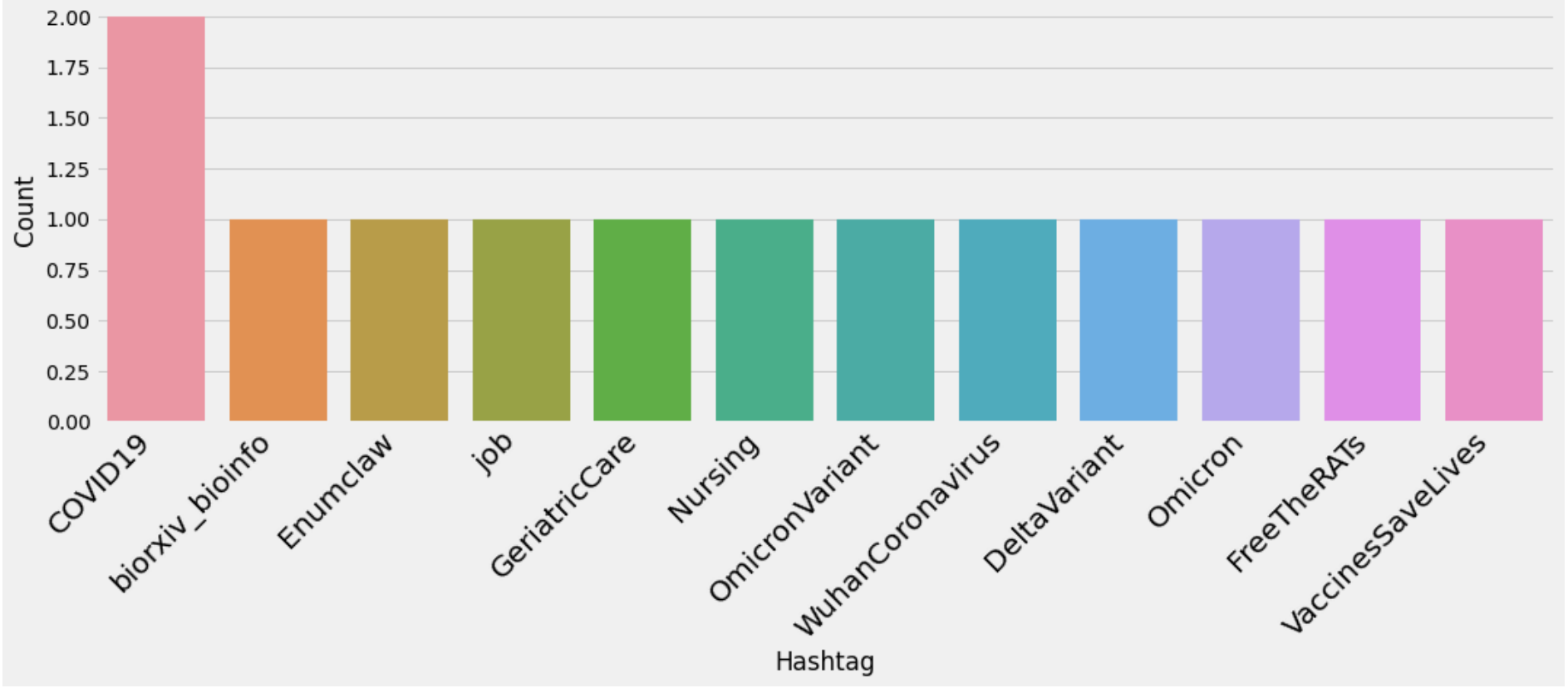
Hashtags with neutral polarity

Calculating the number of neutral tweets

In [40]:
```
%%time
neutral_hashtag_row_list = neutral_hashtag.collect()
neutral_hashtag_string_list = [ ele.__getattr__('hashtags') for ele in neutral_hashtag_row_list]
neutral_tweets=hashtag_extract(neutral_hashtag_string_list)
neutral_tweets = sum(neutral_tweets,[])
```

Wall time: 1min 10s

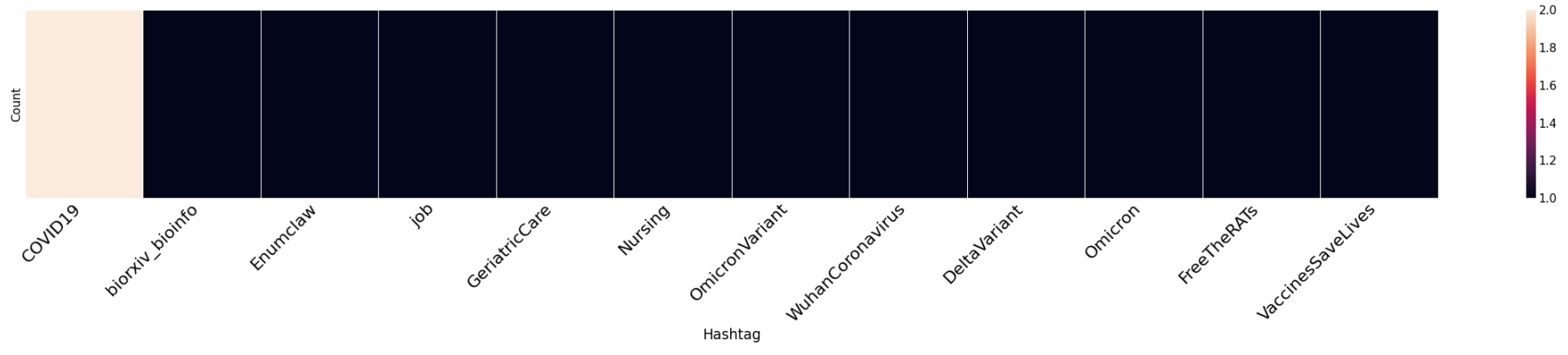
In [41]:
```
%%time
neu_tweets = nltk.FreqDist(neutral_tweets)
neu_tweets_df = pd.DataFrame({'Hashtag': list(neu_tweets.keys()),
                  'Count': list(neu_tweets.values())})

# selecting top 20 most frequent hashtags
neu_tweets_df = neu_tweets_df.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=neu_tweets_df, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
plt.show()
```



Wall time: 297 ms

In [42]:
```
%%time
neu_tweets_df.set_index('Hashtag', inplace=True)
neu_tweets_df = neu_tweets_df.T
# Plot a heatmap for these hashtags
```

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(30, 6), dpi=80, facecolor='w', edgecolor='k')
plt.style.use('fivethirtyeight')
ax = sns.heatmap(linewidth=0.5, data=neu_tweets_df)
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
plt.tight_layout()
plt.show()
```




```
Wall time: 304 ms
```

Let's perform topic model analysis

TOPIC MODEL ANALYSIS

Import all the required packages

Preparing stop words that are already downloaded and make them as stop_words

In [43]:
```
# Gensim
import gensim, spacy, logging, warnings
import gensim.corpora as corpora
from gensim.utils import lemmatize, simple_preprocess
from gensim.models import CoherenceModel

# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use', 'not', 'would', 'say', 'could', '_',\
                   'be', 'know', 'good', 'go', 'get', 'do', 'done', 'try', 'many', 'some', 'nice',\
                   'thank', 'think', 'see', 'rather', 'easy', 'easily', 'lot', 'lack', 'make', 'want',\
                   'seem', 'run', 'need', 'even', 'right', 'line', 'even', 'also', 'may', 'take', 'come'])
```

Remove new line characters,emails, single quotes and split the sentence into a list of words using gensim's simple_preprocess()

In [44]:
```
def sent_to_words(sentences):
    for sent in sentences:
        sent = re.sub('\S*@\S*\s?', '', sent)
        sent = re.sub('\s+', ' ', sent)  # remove newline chars
        sent = re.sub("\'", "", sent)  # remove single quotes
        sent = gensim.utils.simple_preprocess(str(sent), deacc=True)
        yield(sent)
```

Tokenizing the words and cleaning up the text in the tweets i.e like removing the unnecessary punctuations and characters

In [45]:
```
%%time
all_tweets = vaccination_tweets.select('clean_tweets')
all_tweets_row_list = all_tweets.collect()
all_tweets_string_list = [ ele.__getattr__('clean_tweets') for ele in all_tweets_row_list]

all_tweets_words = list(sent_to_words(all_tweets_string_list))
print(all_tweets_words[:1])
```

```
[['free', 'skills', 'sessions', 'sydney', 'academy', 'saturday', 'athletes', 'interested', 'preparing', 'provincials', 'springs
u']]
Wall time: 5.56 s
```

Creating the bigram and Trigram using Gensim's Phrases model

In [46]:
```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(all_tweets_words, min_count=5, threshold=100) # higher threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[all_tweets_words], threshold=100)
# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)
```

```
# See trigram example
print(trigram_mod[bigram_mod[all_tweets_words[0]]])
```

```
['free', 'skills', 'sessions', 'sydney', 'academy', 'saturday', 'athletes', 'interested', 'preparing', 'provincials', 'springsu']
```

Removing the stopwords and then making the bigrams ready to lemmatize and call the functions out sequentially

In [47]:
```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

Calling out the functions in order

In [48]:
```
# Remove Stop Words
data_words_nostops = remove_stopwords(all_tweets_words)
```

In [49]:
```
# Form Bigrams
data_words_bigrams = make_bigrams(all_tweets_words)
```

In [50]:
```
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
```

In [51]:
```
%%time
# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:1])
```

```
[['free', 'skill', 'session', 'sydney', 'academy', 'athlete', 'interested', 'prepare', 'provincial', 'springsu']]
Wall time: 52 s
```

Creating the dictionary and also the Corpus those are needed for Topic Modeling Analysis

In [52]:
```
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1)]]
```

Building Linear Discriminant Analysis (LDA) Model

In [53]:
```
%%time
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=20,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

```
Wall time: 1min 42s
```

We realized that building this model on our dataset took close to 20 minutes

Viewing the topics in the LDA Model

In [54]:
```
%%time
# Print the Keyword in the 10 topics
print(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0, '0.248*"case" + 0.232*"virus" + 0.057*"transmission" + 0.047*"coronavirus" + 0.035*"confirm" + 0.035*"bring" + 0.025*"current
ly" + 0.020*"unvaccinate" + 0.009*"percent" + 0.005*"hospitalize"'), (1, '0.433*"get" + 0.181*"vaccination" + 0.044*"infection" +
0.043*"do" + 0.039*"mean" + 0.030*"pm" + 0.019*"top" + 0.015*"instead" + 0.013*"negative" + 0.010*"host"'), (2, '0.105*"time" + 0.
101*"vaccinate" + 0.098*"year" + 0.064*"datum" + 0.057*"flu" + 0.035*"pharmacy" + 0.035*"rule" + 0.031*"fact" + 0.031*"second" +
0.030*"positive"'), (3, '0.179*"test" + 0.114*"give" + 0.098*"right" + 0.072*"month" + 0.049*"ever" + 0.048*"kill" + 0.044*"perso
n" + 0.034*"place" + 0.021*"expe" + 0.020*"design"'), (4, '0.188*"know" + 0.182*"take" + 0.170*"amp" + 0.083*"stop" + 0.036*"yet"
+ 0.031*"protect" + 0.027*"money" + 0.023*"blood" + 0.012*"fail" + 0.012*"clearly"'), (5, '0.420*"covid" + 0.118*"also" + 0.072*"e
ven" + 0.052*"re" + 0.049*"let" + 0.036*"dose" + 0.029*"understand" + 0.020*"drop" + 0.019*"third" + 0.016*"level"'), (6, '0.118
*"many" + 0.077*"bad" + 0.063*"issue" + 0.056*"cluster" + 0.047*"family" + 0.043*"outbreak" + 0.042*"testing" + 0.040*"county" +
0.037*"offer" + 0.036*"include"'), (7, '0.167*"need" + 0.161*"day" + 0.159*"work" + 0.134*"health" + 0.065*"job" + 0.026*"fight" +
0.022*"more" + 0.016*"cvs" + 0.014*"double" + 0.013*"sorry"'), (8, '0.441*"say" + 0.061*"hear" + 0.057*"life" + 0.053*"remember" +
0.042*"pass" + 0.029*"guy" + 0.016*"boy" + 0.000*"s" + 0.000*"today" + 0.000*"well"'), (9, '0.141*"pfizer" + 0.101*"still" + 0.084
*"want" + 0.081*"quarantine" + 0.072*"vaccinated" + 0.063*"death" + 0.058*"show" + 0.041*"back" + 0.038*"result" + 0.033*"efficac
y"'), (10, '0.099*"booster" + 0.073*"last" + 0.068*"much" + 0.068*"come" + 0.060*"government" + 0.055*"lockdown" + 0.054*"rate" +
0.046*"kid" + 0.044*"shot" + 0.037*"moderna"'), (11, '0.133*"tell" + 0.103*"call" + 0.091*"mask" + 0.051*"plan" + 0.039*"hour" +
0.037*"wear" + 0.036*"treatment" + 0.030*"clinic" + 0.026*"leave" + 0.023*"sit"'), (12, '0.135*"think" + 0.121*"good" + 0.118*"spr
ead" + 0.072*"use" + 0.067*"news" + 0.062*"great" + 0.060*"thank" + 0.058*"really" + 0.047*"long" + 0.026*"boost"'), (13, '0.216
*"omicron" + 0.155*"variant" + 0.097*"feel" + 0.053*"just" + 0.041*"question" + 0.037*"antibody" + 0.035*"community" + 0.027*"fac
e" + 0.026*"receive" + 0.018*"answer"'), (14, '0.243*"make" + 0.096*"thing" + 0.090*"ve" + 0.069*"put" + 0.037*"flu_shot" + 0.036
*"lead" + 0.030*"food" + 0.025*"daily" + 0.020*"possibly" + 0.020*"whole"'), (15, '0.365*"go" + 0.102*"care" + 0.078*"cause" + 0.0
47*"worker" + 0.037*"literally" + 0.033*"add" + 0.013*"final" + 0.000*"hospital" + 0.000*"today" + 0.000*"s"'), (16, '0.459*"vacci
ne" + 0.146*"new" + 0.054*"mandate" + 0.053*"help" + 0.045*"public" + 0.027*"low" + 0.018*"approve" + 0.014*"information" + 0.013
*"recent" + 0.009*"govt"'), (17, '0.000*"georgia" + 0.000*"danish" + 0.000*"brifing" + 0.000*"faint" + 0.000*"ltns" + 0.000*"inhal
ation" + 0.000*"golly" + 0.000*"remotely" + 0.000*"alr" + 0.000*"tribune"'), (18, '0.211*"see" + 0.206*"doctor" + 0.082*"watch" +
0.049*"free" + 0.043*"there" + 0.022*"soon" + 0.020*"realize" + 0.019*"honestly" + 0.017*"thought" + 0.012*"interested"'), (19,
'0.329*"people" + 0.127*"look" + 0.088*"way" + 0.043*"talk" + 0.037*"already" + 0.034*"side_effect" + 0.034*"old" + 0.032*"proble
m" + 0.023*"like" + 0.015*"control"')]
Wall time: 8.05 ms
```

Computing the Perplexity and also Coherence Score. Lower perplexity score indicates the model is good.

In [55]:

```
%%time
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -20.686425031377937

Coherence Score:  0.37780160795104456
Wall time: 15.6 s
```

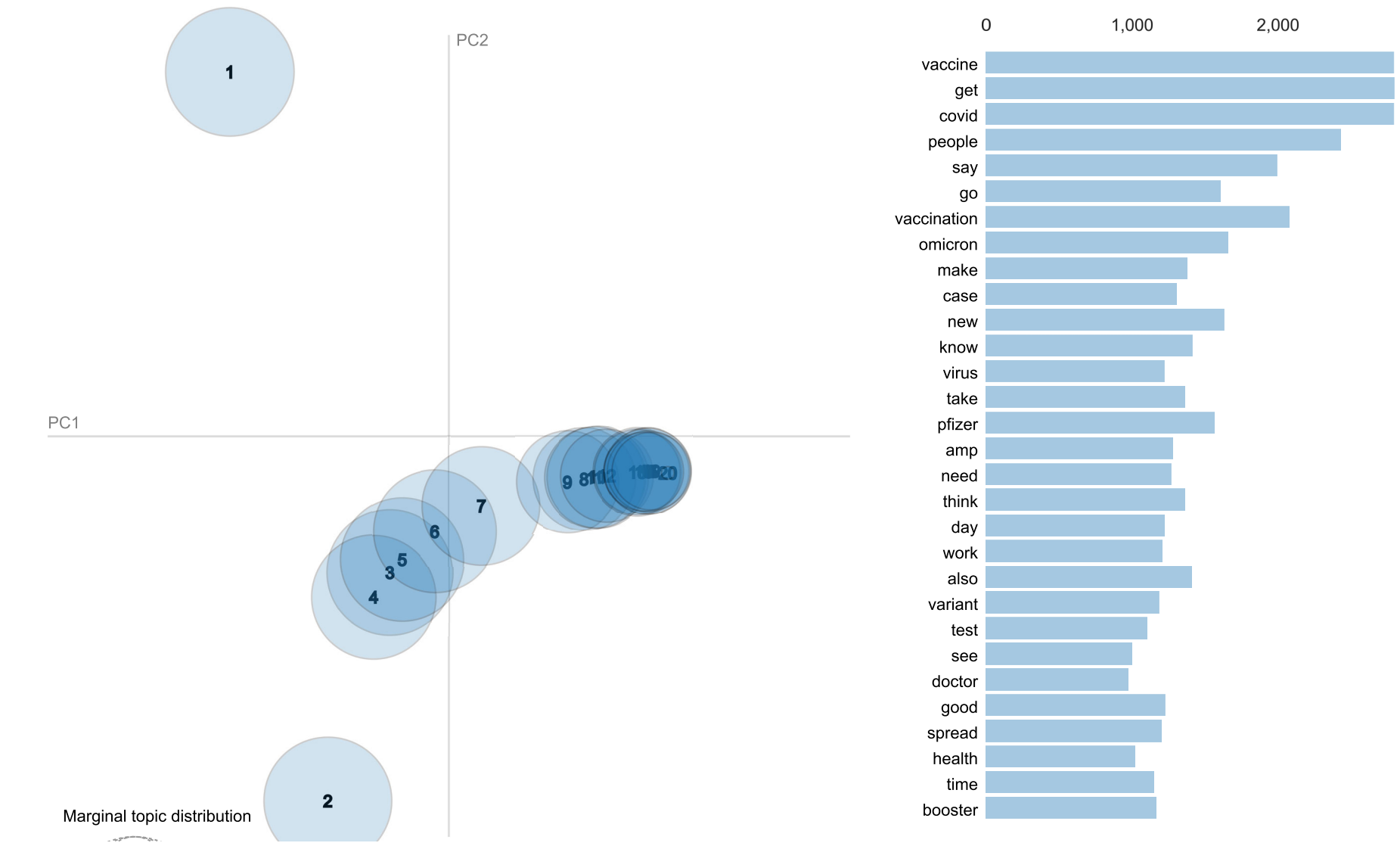
Visualizing the keywords of the topics

In [56]:

```
%%time
import pyLDAvis
import pyLDAvis.gensim_models
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
vis
```

```
C:\ProgramData\Anaconda3\envs\twitter\lib\site-packages\pyLDAvis\_prepare.py:246: FutureWarning: In a future version of pandas all
arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  default_term_info = default_term_info.sort_values(
Wall time: 10.9 s
```

Out[56]:



The above visualization is not getting displayed unless the notebook is run. Therefore we are adding the screen shot of visualization

Explanation

- **Actually there is no better tool than pyLDAvis package's interactive chart and it worked really well in visualizing the topics-keywords.**
- **Each bubble that is present on the left side plot represents a topic. The larger bubble represents that it is more prevalent topic.**
- **We can observe from the graph the model having big overlapping bubbles scattered is a good topic model.**
- **Also a model with many topics is having many overlaps and all of the small bubbles are grouped into a cluster.**
- **If you move the cursor over the bubbles then you can see the bars on the right side will get updated accordingly with the important keywords in that selected topic.**

In [ ]: