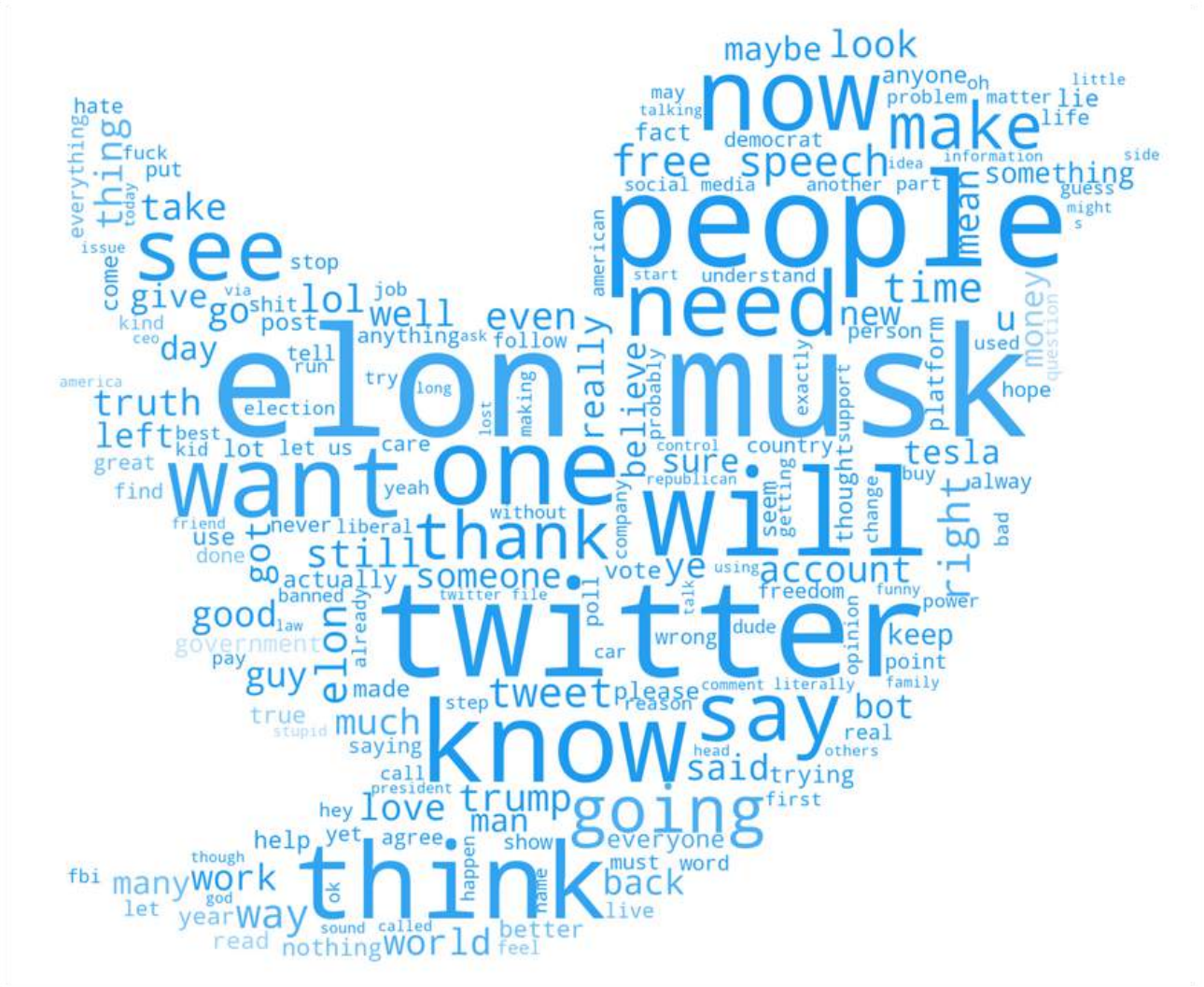
Unsupervised Sentiment Analysis

500 000 Tweets about Elon Musk 🐦



Clément Delteil

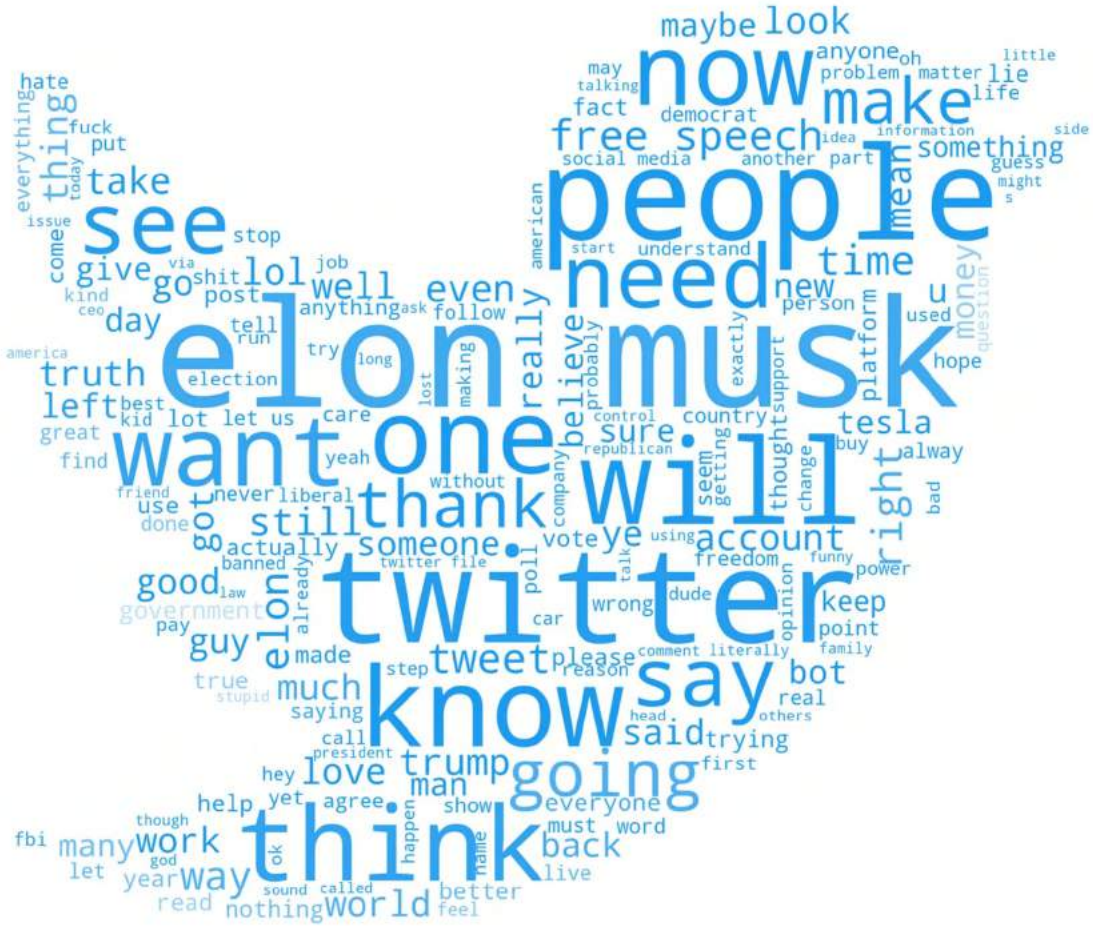
clementdelteil

Unsupervised Sentiment Analysis With Real-World Data: 500,000 Tweets on Elon Musk

Guided walkthrough in a real-world Natural Language Processing project.

Clément Delteil 🌱

Published in Towards AI · 15 min read · Feb 12



WordCloud of the most frequent words used in the dataset — Image by author

Most **Natural Language Processing** and **Unsupervised Sentiment Analysis** projects that can be found on the internet use near-perfect datasets. The textual data are all focused on a particular topic, which makes the analysis much simpler.

I wanted to challenge myself to create a complete project: from data collection to cleaning, visualization, and finally, sentimental analysis.

To do this, I needed a source of almost infinite data that could evolve over time. I immediately thought of Twitter and the more than divisive person that is Elon Musk. So I collected 500,000 tweets about him (503,986 to be exact) over 30 days between November and December 2022.

In this story, I will present you all the steps I went through to clean, visualize, and analyze this homemade data set. We will be able to discover the most frequently used words to talk about Elon Musk, the most mentioned accounts, and the overall emotions associated with his person.

Before we dive in, If you want to know how I collected this dataset, I suggest you read my article on the subject.

. . .

First look at the data

I'm not going to lie to you and pretend I'm discovering the data for the first time 😉. But it's good to remember that having a first look at your data before anything else allows you to foresee the steps to follow in the project and see the potential problems...

You will find the source code in a Google Colab notebook given at the end of this article so that you can reuse the code for your own projects. In the meantime, for the sake of clarity, I will skip the installation steps to make the reading more straightforward.

```
import pandas as pd

df = pd.read_json('data_503986.json')
df.sample(n=10)
```

	id	text
93913	1599893786863497216	Elon Musk says assassination risk 'quite significant,' vows to not sign autographs 'ever again' https://t.co/EfBtLBhYkU via @americanwire_
136659	1600443615741808641	@elonmusk When will the Pfizer files be released?
173191	1600933119309160448	@elonmusk @SawyerMerritt Buy back shares Elon.. the ones you sold on Twitter would at least help Tesla now
324598	1603384268146745345	@elonmusk ew a hyundai
386632	1604275139524304896	@Andrea614th @elonmusk @MrAndyNgo The sick of it is that Trump actually OK'd an undisclosed amount of money to be allotted to the deep state. We are talking a tremendous amount of unaccounted cash flow for whatever the deep state wanted. I remember seeing this... I can't find the documentation any more.
223030	1601572328319504391	@elonmusk @ZacksJerryRig Would you support any of those 3 candidates again knowing what you know now?
311103	1602857050484408322	@elonmusk The crime is happening and Tesla while you Tweet silly remarks and waste your time on Twitter. https://t.co/cNXoGh8zUJ
62566	1599279742611771393	. @ElonMusk is Elon Musk-pilled. (The Verge) #SocialMedia #Business https://t.co/WNTIwOjdlU https://t.co/EgOftDtOOk
450576	1605125991218790400	@cryptoworld202 Check out $GEMINI \nwith real utilities and delivering devs, I have a comfy hold for this one and ready my bags for staking very soon for long term vision\n@GEMINIERC\n\n@JakeGagain \n@chirocrypto \n@elonmusk \n@cz_binance \n@CoinMarketCap \n@iambroots \n@rovercrc
473334	1605368615435644928	@elonmusk @jus71in This is what an AI bot generate for input 'salmon' https://t.co/NF4crmkqwN

Screenshot of a data sample — Image by author

As you can see, when collecting the data, I chose only to keep the text of the tweet without any other information for anonymity reasons. This first sample allows us to identify several important details.

- Many tweets contain mentions that start with the '@' character
- Some tweets include emojis.
- Some tweets contain a link to the quoted tweet at the end of the text.

If you look in detail at the content of the tweets, we can already see that they are very emotionally intense.

Let's now proceed to the first cleaning step to visualize the most frequent words in the corpus of tweets.

. . .

Most Frequent Words

Preprocessing

After dropping the id column, I identified some special characters and unique features of our dataset. As a result, I defined the following pre-processing function.

```
def pre_process(text):
    # Remove links
    text = re.sub('http://\S+|https://\S+', '', text)
    text = re.sub('http[s]?://\S+', '', text)
    text = re.sub(r"http\S+", "", text)

    # Convert HTML references
    text = re.sub('&amp', 'and', text)
    text = re.sub('&lt', '<', text)
    text = re.sub('&gt', '>', text)
    text = re.sub('\xao', ' ', text)

    # Remove new line characters
    text = re.sub('[\r\n]+', ' ', text)

    # Remove mentions
    text = re.sub(r'@\w+', '', text)

    # Remove hashtags
    text = re.sub(r'#\w+', '', text)

    # Remove multiple space characters
    text = re.sub('\s+',' ', text)

    # Convert to lowercase
    text = text.lower()
    return text
```

We can apply it to all the rows of our dataset.

```
df['processed_text'] = df['text'].apply(pre_process)
```

text	processed_text

Since Elon Musk purchased Twitter, about 90% of my feed is conservative pundits offering mindless conspiracies about elections, COVID, Ukraine, Democrats or Jews.	since elon musk purchased twitter, about 90% of my feed is conservative pundits offering mindless conspiracies about elections, covid, ukraine, democrats or jews.
This is why your wife left you https://t.co/C63hnJeEIG	this is why your wife left you
@JHester1531 @C4Dbeginner @elonmusk That would be nice thing to do.	that would be nice thing to do.
@elonmusk That is all to frequently true	that is all to frequently true
I can not believe this picture of elon musk at the world cup is real https://t.co/H1ZTY9XhNh	i can not believe this picture of elon musk at the world cup is real

Processed sample — Image by author

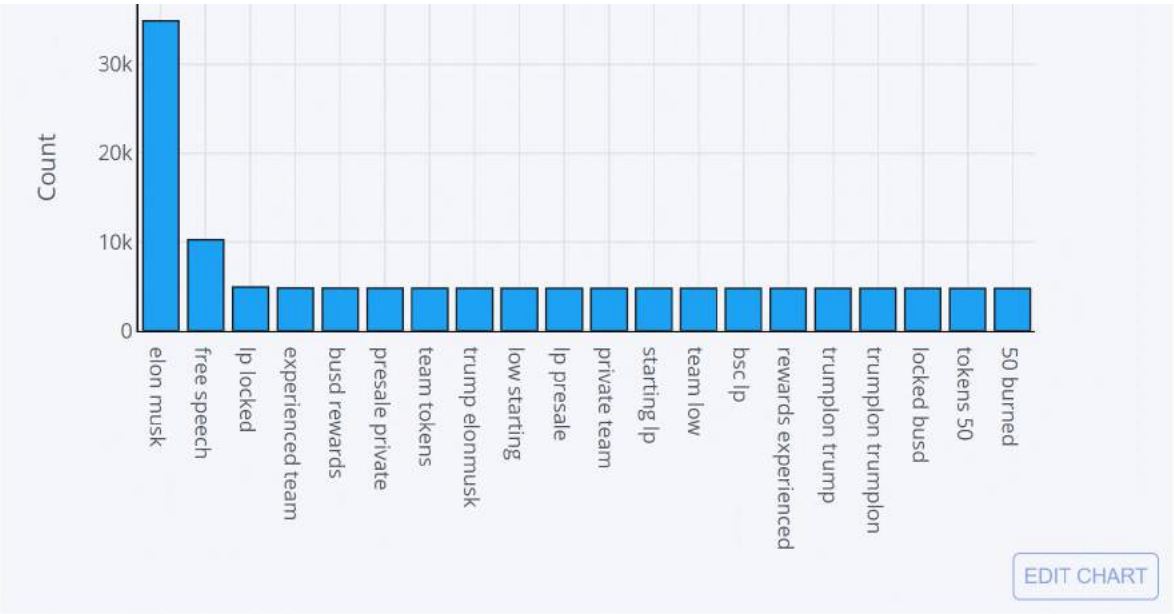
The data is already a little clearer. But we still have to deal with spam. Take a look at the second tweet displayed. Am I the only one who thinks this looks like spam? Let's have a first look at the most used words to see if they are the majority in our dataset.

N-Grams

For this, we will display the 20 most present bi-grams and tri-grams in our dataset. We will remove the empty words to get the most accurate results possible. We are going to use the Plotly library as well as some code snippets from Susan Li that allowed me to make the following graphics dynamic and more pleasing to see.

```
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.ite
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

common_words = get_top_n_bigram(df['processed_text'], 20)

df1 = pd.DataFrame(common_words, columns = ['TweetText' , 'count'])
df1.groupby('TweetText').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    title='Top 20 bigrams in Tweet before removing spams')
```

The 20 most frequent bi-grams in the dataset before removing spams



Visualization of the most present bi-grams with spams— Graphic by author

As you can see on the graph above, from the second bi-gram onwards, everything else is spam. I manually searched the dataset to confirm my hypothesis and this is indeed the case. Many spammers use Elon Musk as a hashtag or mention him to appear in front of more accounts.

Several solutions are available to us, we could remove duplicate lines in the dataset, use a machine learning model to detect spam, or manually find the tweets and remove it via a list of terms to ban.

I chose the third option because it was the easiest and fastest of the three to implement.

```
to_drop = ["LP LOCKED",
  "This guy accumulated over $100K",
  "accumulated 1 ETH",
  "help me sell a nickname",
  "As A Big **** *** To The SEC",
  "Wanna be TOP G",
  "#walv",
  "#NFTProject",
  "#1000xgem",
  "$GALI",
  "NFT",
  "What the Soul of USA is",
  "#BUSD",
  "$FXMS",
  "#fxms",
  "#Floki",
  "#FLOKIXMAS",
  "#memecoin",
  "#lowcapgem",
  "#frogxmas",
```

```
    "Xmas token",
    "crypto space",
    "Busd Rewards",
    "TRUMPLON",
    "NO PRESALE",
    "#MIKOTO",
    "$HATI",
    "$SKOLL",
    "#ebaydeals",
    "CHRISTMAS RABBIT",
    "@cz_binance",
    "NFT Airdrop",
    "#NFT"]
```

We drop the lines that contain any of the words in our list.

```
df = df[~df['text'].str.contains('|'.join(to_drop))]
```

We went from 503,986 lines to 487,047, so we eliminated 16,939 spams.

I also noticed that there were problems with the contractions of some words. **Don't know** became **don know** instead of **do not know.**

To fix this problem I decided to use the Python **contractions** library which allows me to solve this problem.

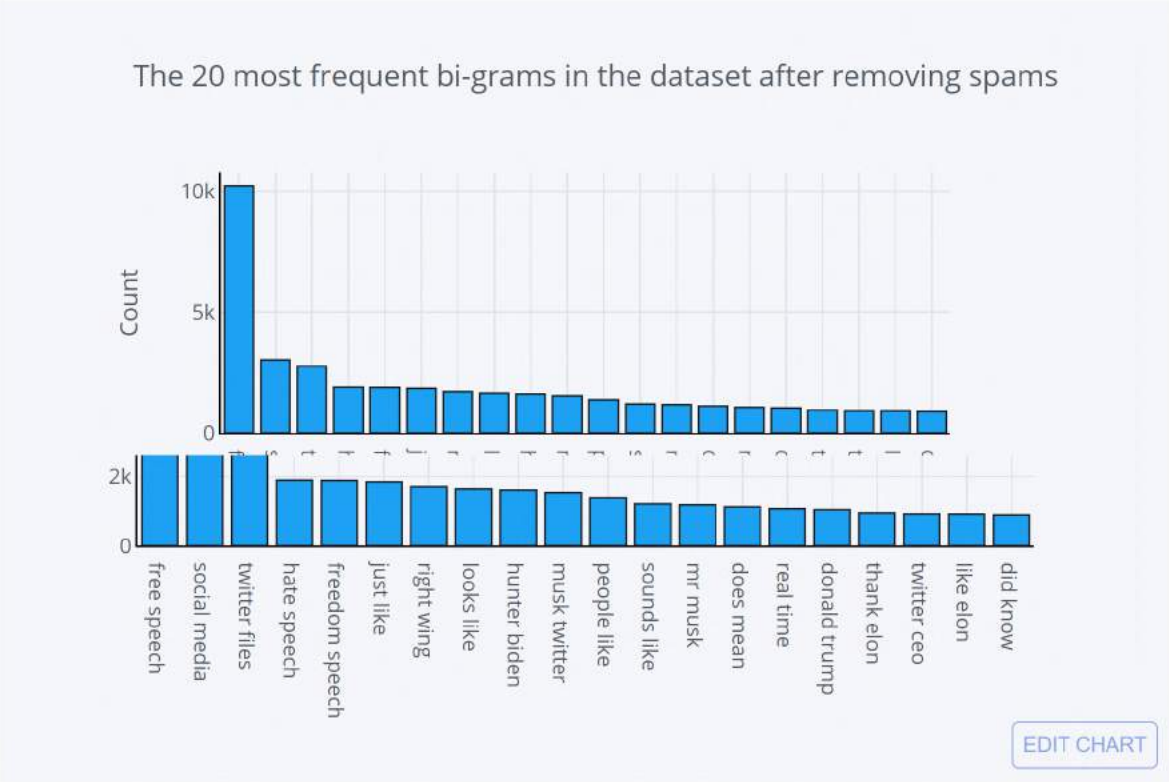
```
def expand_contractions(text):
  try:
    return contractions.fix(text)
  except:
    return text
```

	text	processed_text	expanded_text
244270	@elonmusk Would be nice to have a single "block and report fin-spam" links as opposed to the three to five pages to report.	would be nice to have a single "block and report fin-spam" links as opposed to the three to five pages to report.	@elonmusk Would be nice to have a single "block and report fin-spam" links as opposed to the three to five pages to report.
257721	@DawnMieras @RepMTG @elonmusk So, these other "scientists" that disagreed with Fauci- they're what? Not "scientists" because they thought differently?\nAre YOU a scientist? \n"Follow the science. Except when it's basic biology. Then, you can make up what you want."	so, these other "scientists" that disagreed with fauci- they are what? not "scientists" because they thought differently? are you a scientist? "follow the science. except when it is basic biology. then, you can make up what you want."	@DawnMieras @RepMTG @elonmusk So, these other "scientists" that disagreed with Fauci- they are what? Not "scientists" because they thought differently?\nAre YOU a scientist? \n"Follow the science. Except when it is basic biology. Then, you can make up what you want."
407108	@elonmusk @stillgray 100% If people are allowed to link to better products then users will start leaving for those better products!	100% if people are allowed to link to better products then users will start leaving for those better products!	@elonmusk @stillgray 100% If people are allowed to link to better products then users will start leaving for those better products!

Processed and expanded sample — Image by author

We can now properly analyze our first graph.

Note: *I removed the elon musk bi-gram because its presence has no real interest, we already know that the tweets are about him.*



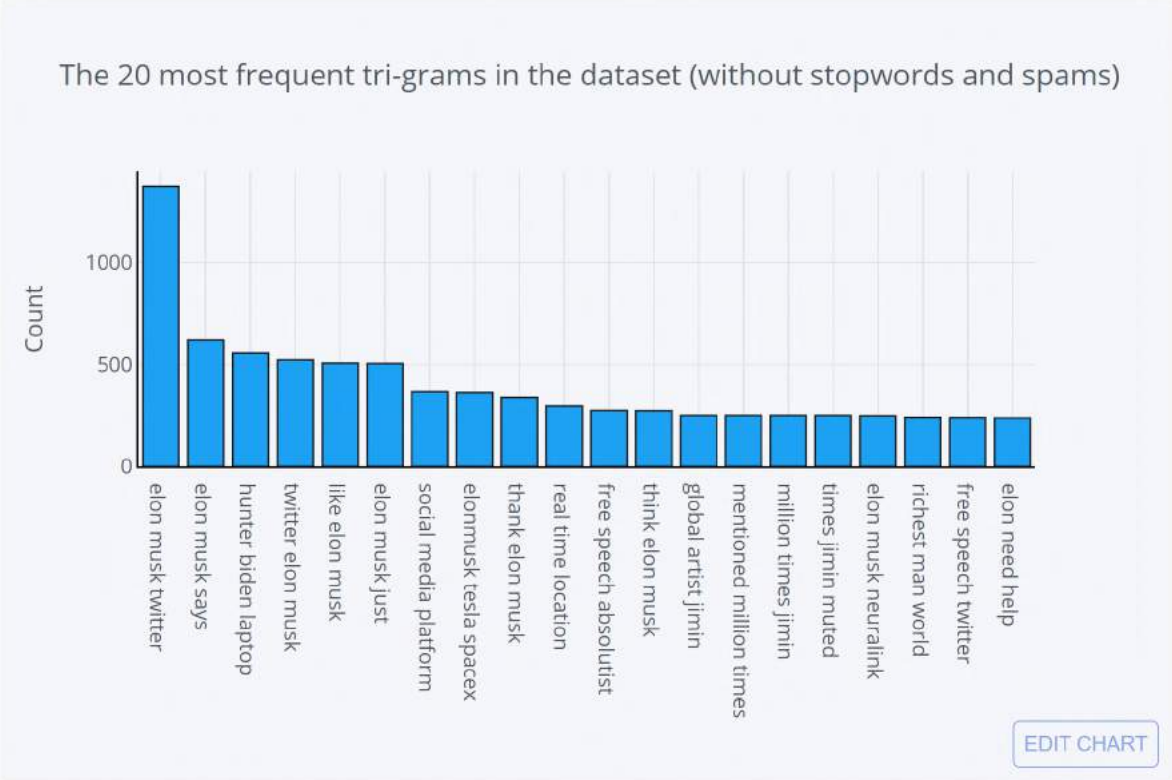
Visualization of the most present bi-grams without spams — Graphic by author

For those following what's been happening on Twitter since the takeover, you should find the various narrative arcs that have existed over the past few months.

Now let's see if the tri-grams return the same subjects. The code is more or less similar. We only change one parameter in the CountVectorizer function.

```
def get_top_n_trigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(3, 3), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.ite
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

common_words = get_top_n_trigram(df['processed_text'], 20)
```

```
for word, freq in common_words:
    print(word, freq)

df6 = pd.DataFrame(common_words, columns = ['TweetText' , 'count'])
df6.groupby('TweetText').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar',
    yTitle='Count',
    linecolor='black',
    title='The 20 most frequent tri-grams in the dataset (without stopwords and
```



Visualization of the most present tri-grams without spams — Graphic by author

Unlike the bi-grams, which had global themes, here you can find much more precise themes. Namely :

- Hunter Biden's laptop controversy.
- SpaceX, Elon Musk's company.
- Real-time sharing of Elon Musk's position by a journalist.
- Elon Musk's stated ambition to open up full freedom of expression.
- The accidental banishment of the singer Jimin from the group BTS.

We will analyze the feelings associated with these topics later in this article. In the meantime, let's move on to another type of visualization: the word cloud.

WordClouds

With the word clouds, we can visualize two things. The so-called uni-grams i.e. the single words but also the mentions. It can be interesting to see which personalities are mentioned the most in the tweets about Elon Musk. Let's have a look at it together!

For uni-grams, we will follow almost the same steps as for N-Grams. The difference is that this time we will **lemmatize** the words to reduce the similar words that would have the same meaning.

For those not familiar with this concept :

- *"I ate an apple"* will be lemmatized into *"I eat an apple"*
- *"running, ran, runs"* will be lemmatized into *"run, run, run"*

This will allow similar words to be grouped together.

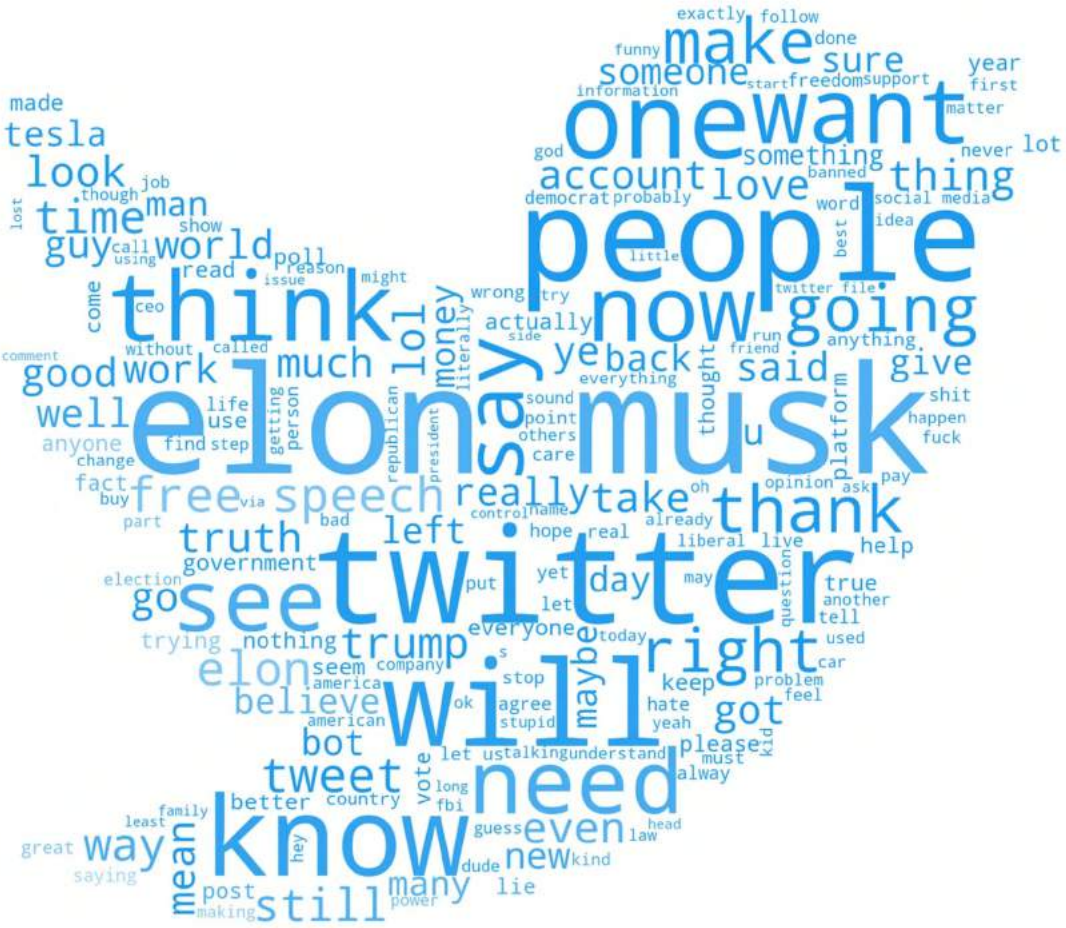
By reusing the `get_top_n_words` function used above, we have :

```
# Needed imports for WordCloud, Lemmatization, and Mask Image
from PIL import Image
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

# Initialize Lemmatize
wordnet_lem = WordNetLemmatizer()

# Lemmatize processed text and join everything in a list
df['text_lem'] = df['processed_text'].apply(wordnet_lem.lemmatize)
all_words_lem = ' '.join([word for word in df['text_string_lem']])
```

Now, all we have to do is generate a word cloud. To do this, I decided to use the Twitter logo as a **mask** to present the results more attractively.

Indeed, I doubt that some people would have clicked on this article if I had put the classic word cloud given by the library as a thumbnail 😉. Please tell me in the comments if that's the case for you. I am curious to know what made you want to read this article!

```
# Generate a word cloud image
mask = np.array(Image.open("twitter.jpg"))
stopwords = set(STOPWORDS)

wordcloud_twitter = WordCloud(height=2000,
                              width=2000,
                              background_color="white",
                              mode="RGBA",
                              stopwords=stopwords,
                              mask=mask).generate(all_words_lem)

# Create coloring from the image
image_colors = ImageColorGenerator(mask)
plt.figure(figsize=[20,20])
plt.axis('off')
plt.tight_layout(pad=0)
plt.imshow(wordcloud_twitter.recolor(color_func=image_colors), interpolation="bi

# Store visualization to file
plt.savefig("twitter_logo.png", format="png")

plt.show()
```



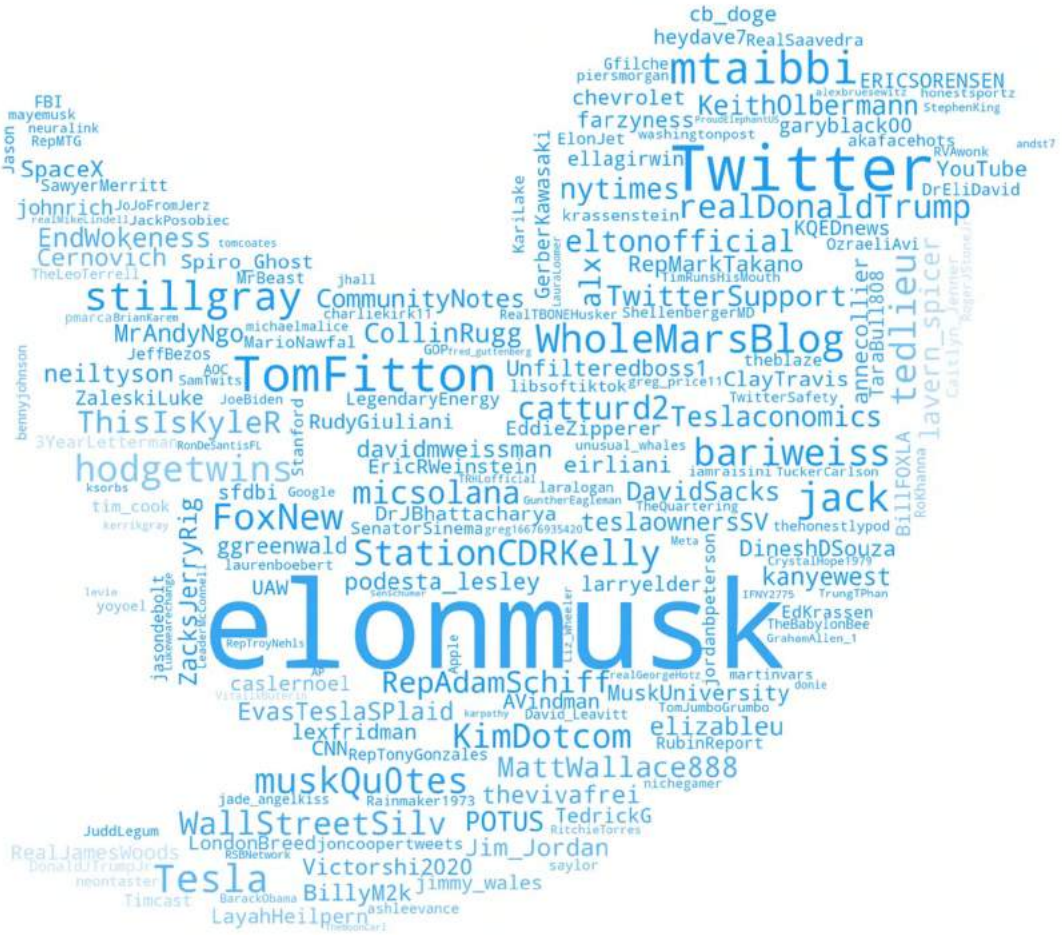
Word cloud of unigrams — Image by author

Nothing surprising about this image, but it is a practical way to attract the

attention of your readers or audience when presenting your data. You can use these types of visualizations to attract and then present other graphs that are less attractive at first glance but give more information about the data. Like I just did with you 😉.

For mentions, we will follow the same process, I will simply show you the code to specifically extract mentions from tweets.

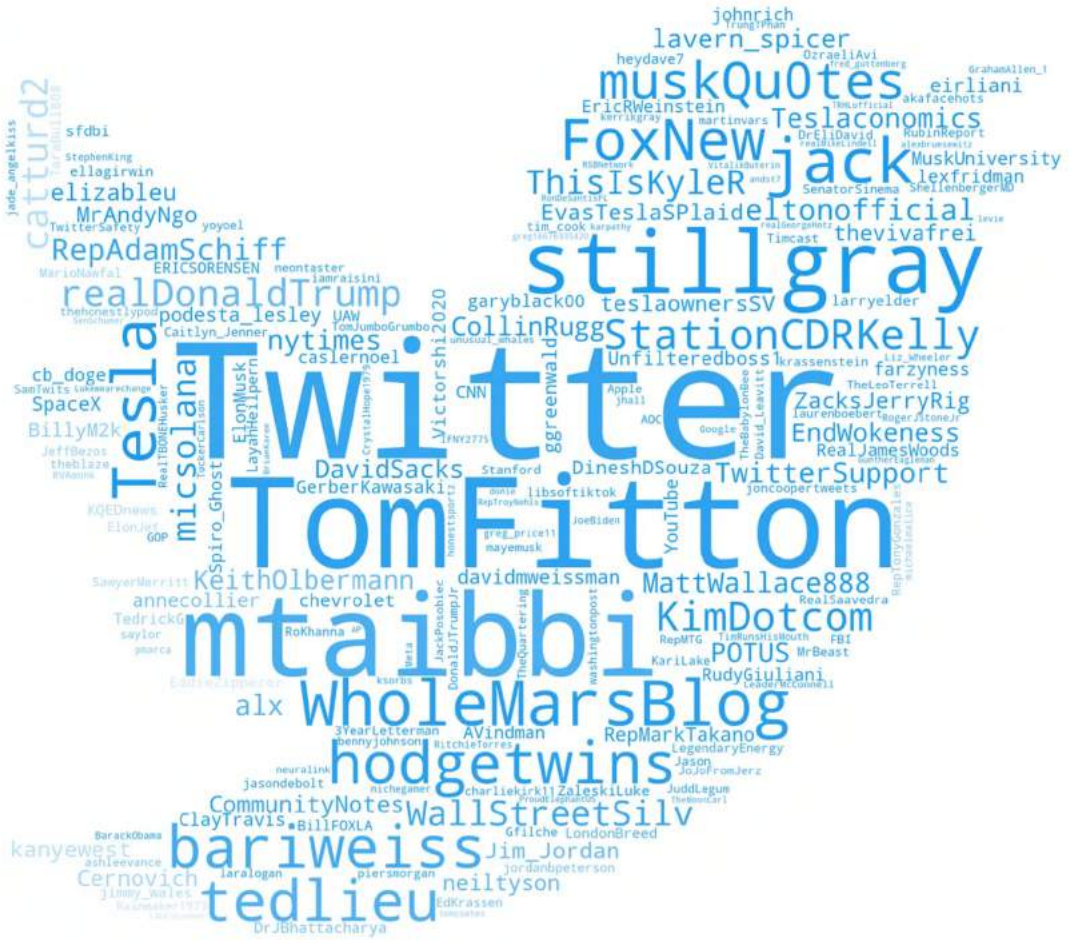
```
# Retrieves all occurrences of @+text
df['mentions'] = df['text'].str.findall(r'@\w+')
# Removes the @ in front
df['mentions'] = [list(map(lambda x: x[1:], mentions)) for mentions in df['menti
# Converts the list of words in each row to a string
df['mentions_string'] = df['mentions'].apply(lambda x: ' '.join(x))
# Concatenates all strings in one string
all_mentions = ' '.join([word for word in df['mentions_string']])
```



Word cloud of the most mentioned accounts — Image by author

Since a tweet needed to talk about Elon Musk to be part of the dataset, we logically find Elon Musk in the middle of this visualization. It is quite

embarrassing because he is so present that it is difficult to realize the importance of some other accounts. Let's see what it looks like when we remove it from the word cloud.



Word cloud of the most mentioned accounts without Elon Musk— Image by author

And voila! The size of the accounts is now a bit more balanced. We have a better idea of the most mentioned personalities in the tweets of our dataset.

. . .

Emojis

We will now look at the most used emojis in our dataset. Emojis are special characters, and not all of them can be detected natively. So, to detect them correctly, I used the **emot** library.

```
import emot

# Define a function to extract emoticons
def extract_emoticons(text):
```

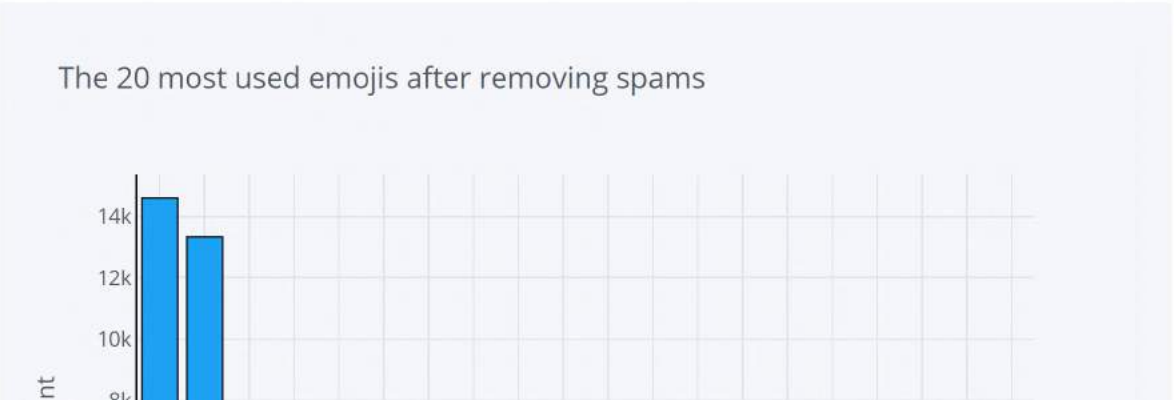
```
    res = emot_obj.emoji(text)
    return res['value']
```

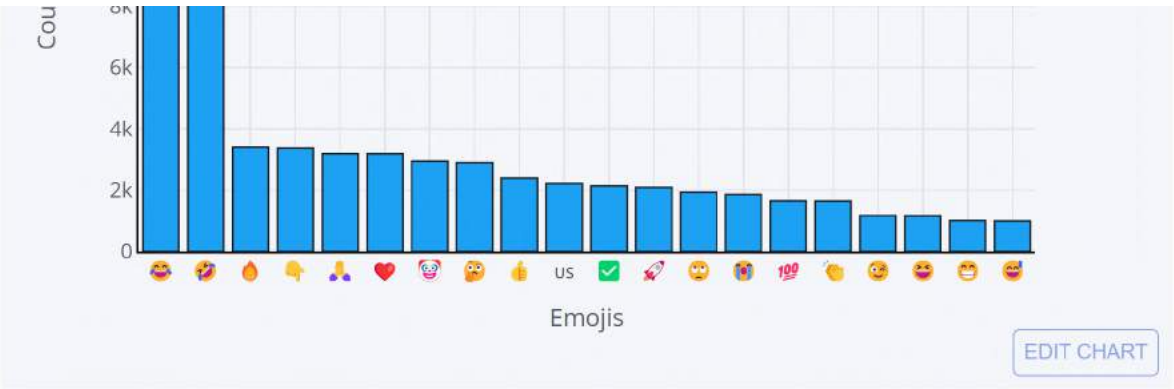
This code snippet will simply return a list of emojis contained in the input tweet. Let's apply it to our text column and merge the results in a brand new DataFrame.

```
# Apply the function to each row of the 'text' column
df['emoticons'] = df['text'].apply(extract_emoticons)
# Count the emojis in each list
df['emoticons'].apply(lambda x: collections.Counter(x))
# Combine the counts
combined_counts = sum(df['emoticons'].apply(lambda x: collections.Counter(x)), c
# Transform it into a dict
emoji_dict = dict(combined_counts)
# Sort it in a descending order
sorted_emoji_dict = dict(sorted(emoji_dict.items(), key=lambda x: x[1], reverse=
# Keep the top 20
d = {k: v for i, (k, v) in enumerate(sorted_emoji_dict.items()) if i < 20}
# Convert the dict to a DataFrame for Plotly
df = pd.DataFrame(list(d.items()), columns=['Emojis', 'Count'])
```

We now have a DataFrame with a column dedicated to emojis and another one containing the number of appearances of each of them in the whole dataset. Let's visualize it!

```
df.groupby('Emojis').sum()['Count'].sort_values(ascending=False).iplot(
  kind='bar',
  xTitle='Emojis',
  yTitle='Count',
  linecolor='black',
  title='The 20 most used emojis after removing spams')
```





Visualization of the most used emojis in the dataset — Graphic by author

We find an overwhelming majority of laughing emojis. There is little to learn from these results except for the clown emoji in 7th place which is not generally associated with positive emotions but more with irony.

. . .

Unsupervised Sentiment Analysis

When we talk about unsupervised sentimental analysis there are two main approaches.

We can either use a clustering algorithm such as KNN or DBSCAN which will identify clusters without any particular indication. It will then be up to us to identify the feelings contained in each cluster because the model is not "aware" of this concept.

Another method, much easier to implement, is the lexical approach. According to this method, in a sentence, each word carries an emotional weight. Thus, the sentimental value of a sentence or a text is the combination of the sentimental value of each word.

In this article, we will focus on the second method. If you are interested I can explore the first approach in another article.

Two models or lexicons are very well known, they are Vader and TextBlob. We will compare their predictions and visualize these results.

Vader

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()

df['vader_polarity'] = df['processed_text'].map(
    lambda text: sid.polarity_scores(text)['compound'])
```

As you can see, in just a few lines we can analyze the emotions of almost 500,000 tweets.

TextBlob

```
from textblob import TextBlob

df['blob_polarity'] = df['processed_text'].map(
    lambda text: TextBlob(text).sentiment.polarity)
```

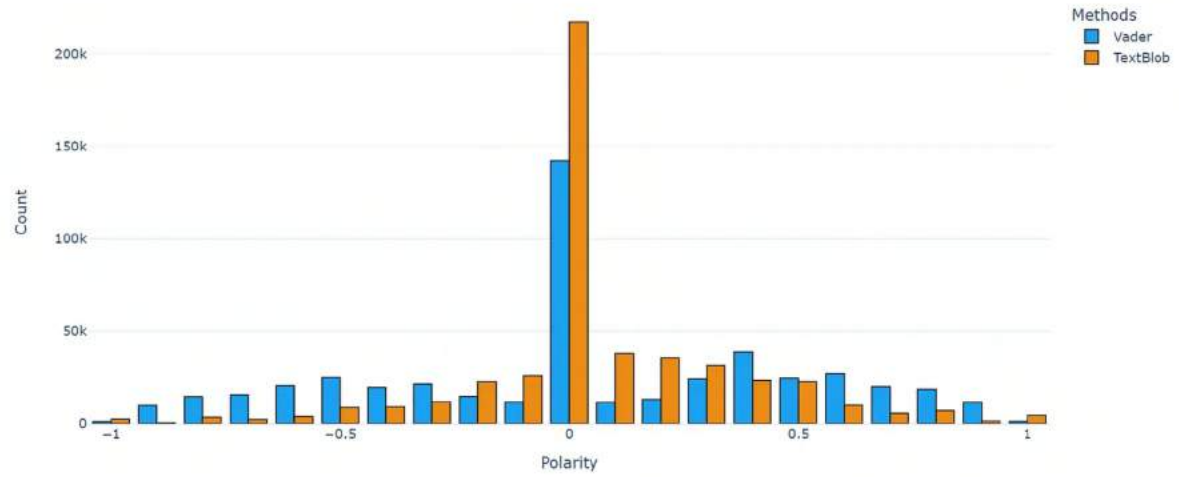
Comparison

Now that we have two new columns in our table, let's see if the two libraries more or less agree on the emotions in our dataset. To do this, we will visualize the distribution of polarities depending on the library.

```
polarity_df = df[['vader_polarity', 'blob_polarity']]
polarity_df = polarity_df.rename(columns={'vader_polarity': 'Vader',
                                          'blob_polarity': 'TextBlob'})

polarity_df.iplot(
    kind='hist',
    bins=40,
    xTitle='Polarity',
    linecolor='black',
    yTitle='Count',
    title='Comparison of the distributions of sentimental polarities',
    colors = ['#1DA1F2', '#EB8C17'],
    barmode="group")
```

Sadly due to the limitations of the free version of Plotly, I'm unable to embed it directly in this article. However, you can check the interactive version of the graph on my website.

Comparison of the distributions of sentimental polarities



Comparison of the distributions of sentimental polarities — Image by author

In any case, this graph allows us to see that although Vader and TextBlob use a similar method, the results are not exactly the same.

TextBlob considers 217,000 tweets to be neutral, while Vader considers 142,000. If we deviate from the 0.0 polarity, we realize that Vader stretches much more toward the extremes than TextBlob.

Despite the Bell Curve appearance, there is a **slight shift on the positive side** of the polarity.

Let's verify these visual intuitions by looking at the statistical data of the polarities.

	Vader	TextBlob
count	487047.000000	487047.000000
mean	0.040615	0.067221
std	0.456807	0.289077
min	-0.999600	-1.000000
25%	-0.296000	0.000000
50%	0.000000	0.000000

75%	0.401900	0.200000
max	0.998200	1.000000

Statistical description of the dataset — Image by author

Our visual observations are well confirmed by the statistics. The average polarity is slightly positive.

Putting aside one's personal opinion of Elon Musk, it's interesting to see the opinion is slightly positive about him. It's a good way to remember that our opinion, whether positive or negative, may not be shared by everyone and that we must also be aware of our own biases.

Let's now look at the sentimental polarities by topic.

Topic Analysis

Let's look at the emotional tendency of the most found bi-grams in the dataset. Before that, we remove stop words and punctuation to make it easier to find topics in the string.

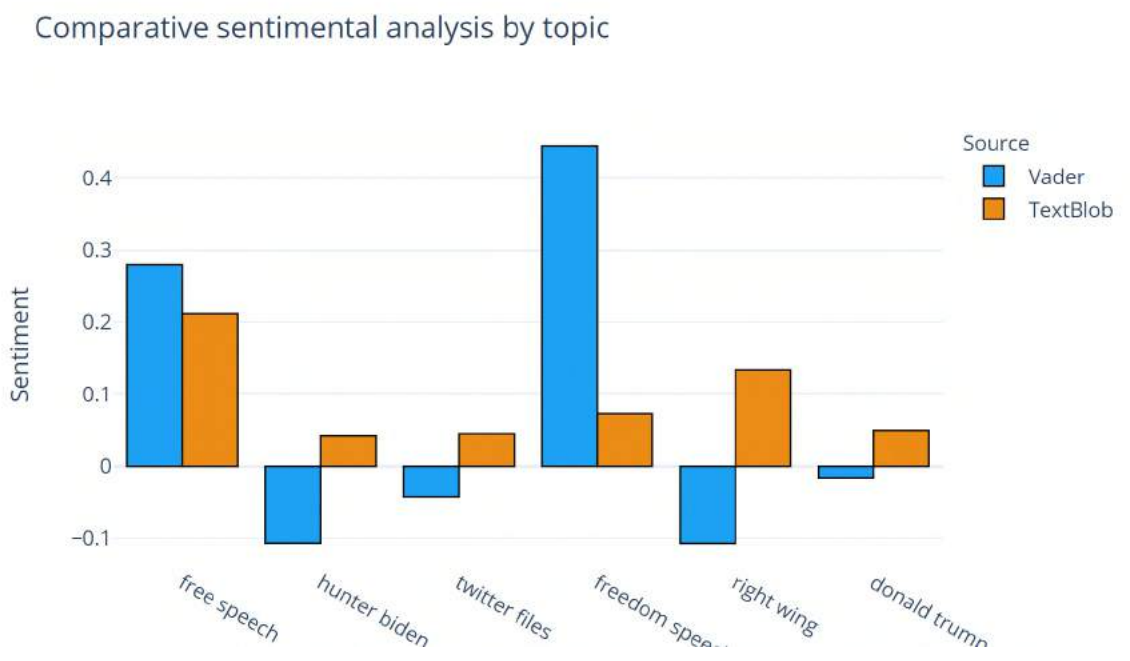
```
stop_words = nltk.corpus.stopwords.words('english')

def remove_stop_words(text):
    text = text.translate(str.maketrans('', '', string.punctuation))
    return ' '.join([word for word in text.split() if word.lower() not in stop_w

df['stop_text'] = df['processed_text'].apply(lambda x: remove_stop_words(x))
```

Now that our data is ready, let's visualize the sentiments per topic!

```
# We define a list of topics
topics = ['free speech',
          'hunter biden',
          'twitter files',
          'freedom speech',
          'right wing',
          'donald trump']

vader_sentiments = df['vader_polarity'].tolist()
textblob_sentiments = df['blob_polarity'].tolist()
text = df['stop_text'].tolist()
```

```
# We create a new column Topic
df['Topic'] = ""
for topic in topics:
    df.loc[df['stop_text'].str.contains(topic), 'Topic'] = topic

# We create a new DataFrame with columns topic / sentiment / source
data = []
for topic in topics:
    topic_rows = df[df['Topic'] == topic]
    # Average sentiment per topic
    vader_sentiments = topic_rows['vader_polarity'].sum() / topic_rows.shape[0]
    textblob_sentiments = topic_rows['blob_polarity'].sum() / topic_rows.shape[0
    # Append data
    data.append({'Topic': topic, 'Sentiment': vader_sentiments, 'Source': 'Vader
    data.append({'Topic': topic, 'Sentiment': textblob_sentiments, 'Source': 'Te

df_new = pd.DataFrame(data)

# Plot the sentiment for each topic
fig = px.bar(df_new,
             x='Topic',
             y='Sentiment',
             color='Source',
             barmode='group',
             color_discrete_sequence = ['#1DA1F2', '#EB8C17'],
             title='Comparative sentimental analysis by topic',
             template='plotly_white')

fig.update_traces(marker_line_width=1,
                  marker_line_color="black")

fig.show()
```

In order for the bar graphs to be roughly equivalent we will visualize the average polarity per library on each of the subjects.



Visualization of the sentimental polarities according to the subject and the source — Graphic by author

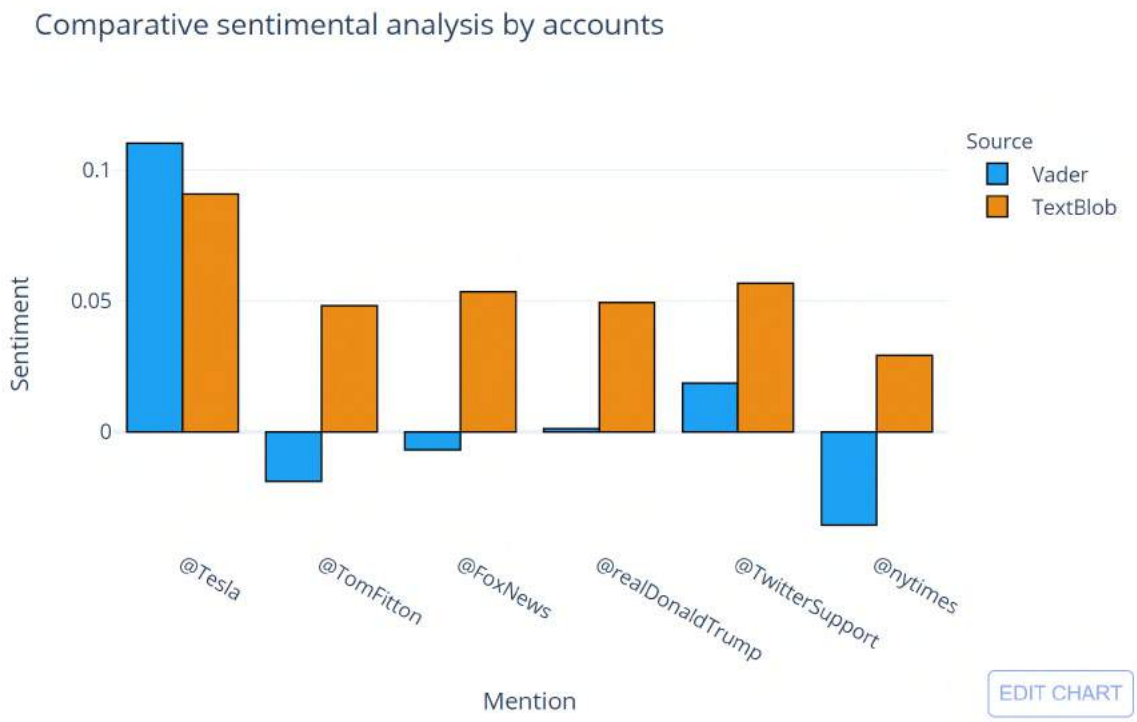
This graph allows us to see that some topics are unanimous while others are more divided. We can also see that there is an important difference on some subjects depending on the library.

Personalities

We can also look at the average emotional polarity of tweets that mentioned particular accounts. I have selected some of the most mentioned accounts.

```
usernames = ['@Tesla',
             '@TomFitton',
             '@FoxNews',
             '@realDonaldTrump' ,
             '@TwitterSupport',
             '@nytimes']
```

The rest of the code is the same so let's look at the results!



Visualization of the sentimental polarities according to the account and the source — Graphic by author

This is very interesting, the Tesla account seems to be mentioned on average

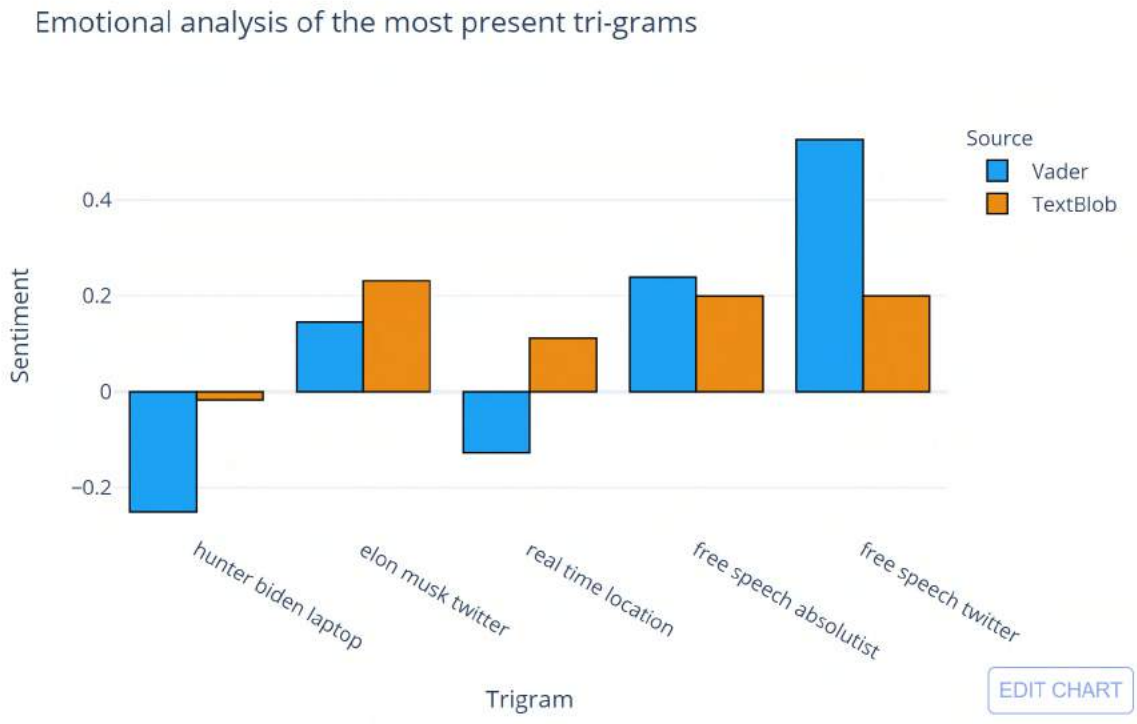
positively. Maybe customers wanting to show off their new purchase. While other accounts like the New York Times account seems to have a rather complex sentimental polarity as it differs between the two libraries.

You can also notice the difference between the sentimental averages in this graph and the first one above. It seems that it is easier to get a sentimental polarity further away from 0 when looking at a bi-gram rather than an account mention.

Following this remark, let's visualize our data one last time by analyzing tri-grams.

Tri-grams

```
tri_grams = ['hunter biden laptop',
             'elonmusk tesla spacex',
             'real time location',
             'free speech absolutist',
             'free speech twitter']
```



Visualization of the sentimental polarities according to the trigrams and the source — Graphic by author

As we explore and visualize our data set, we refine its characteristics and

find the right elements to visualize. This graph above is a great example. This time, we find very strong sentimental polarities on certain subjects.

We can see that the controversy around Hunter Biden's laptop has made a big impact. Vader gives an average polarity of **-0.25** for this topic. We also notice that other subjects are unanimous like the tweets about E*lon Musk and Twitter* as well as the tweets about *freedom of speech*. We reach a polarity peak of **0.53** for Vader on the subject of *freedom of expression on Twitter*. We also notice that on these subjects, the two libraries produce similar results.

Conclusion

In this article, we were able to explore an important part of Data Science. From a manually collected dataset, we explored the corpus of tweets to get a first idea of the topics present, the personalities mentioned as well as the emojis. To do so, we used many text processing techniques used in natural language processing.

This allowed us, during the unsupervised sentimental analysis, **to know what to visualize and why**. As a result, we were able to present useful comparisons of sentimental polarities in an attractive way.

Also, we could notice that the results were not always the same between the two libraries. This gave us a glimpse of the **limitations of these techniques**. Other techniques such as clustering or the use of a pre-trained model might have given more accurate results.

In any case, with this article, I hope to have conveyed to you the interest that these data can have. Business Intelligence services use similar tools to track trends or get an idea of public opinion on a topic or a new product.

If you have a business or topic that interests you, **go for it**! Collect a lot of data and offer your analysis. You might attract someone from that company to offer you a job, who knows 😉.

. . .

Want to connect?

- 📖 Follow me on Medium

- 💌 Subscribe to get an email whenever I publish
- ☕ Fancy a coffee in Paris to discuss AI ?
- 🤓 Connect with me on LinkedIn

I've also written:

Real-Time Sentiment Analysis with Docker, Kafka, and Spark Streaming

A Step-By-Step Guide to Deploying a Pre-trained Model in an ETL Process

pub.towardsai.net

Large-Scale Sentiment Analysis with PySpark

Comparative study of classification algorithms and feature extraction functions implemented in PySpark on 1,600,000...

pub.towardsai.net

Reducing Bias in LLMs: Fine-Tuning for Fairer Language Models

A Critical Examination of a Suggested Research Solution to Reduce Bias in LLMs

generativeai.pub

References

- You will find the whole code and links to the visualizations in this Colab notebook.
- This dataset has been acquired through an Elevated access to Twitter's API that allows Commercial Use. You can download it on Kaggle.
- Susan Li's article on dynamic visualizations with Plotly that inspired some of the code snippets.

Sentiment Analysis | NLP | Twitter | Data Visualization | Data Science