Twitter Financial News, (twitter_sentiment_analysis)

April 19, 2025

1 Name Lakshman Chaudhary

2 Twitter-sentiment-extaction-analysis

3 Importing Libraries

[ ]:
```
# Basic utilities
import os
import re
import string
import random
from collections import Counter

# Data handling
import numpy as np
import pandas as pd

# Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
from plotly import graph_objs as go
import plotly.express as px
import plotly.figure_factory as ff
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

# Natural Language Processing
import nltk
from nltk.corpus import stopwords
import spacy
from spacy.util import compounding
from spacy.util import minibatch

# Progress bar
from tqdm import tqdm

# Warnings filter
```

```
import warnings
warnings.filterwarnings("ignore")

# Setting for inline visualization for Jupyter Notebooks
%matplotlib inline

# Example of listing files in a specific directory - specific to Kaggle␣
 ↪environments
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/tse-spacy-model/models/model_neu/meta.json
/kaggle/input/tse-spacy-model/models/model_neu/tokenizer
/kaggle/input/tse-spacy-model/models/model_neu/vocab/key2row
/kaggle/input/tse-spacy-model/models/model_neu/vocab/strings.json
/kaggle/input/tse-spacy-model/models/model_neu/vocab/lexemes.bin
/kaggle/input/tse-spacy-model/models/model_neu/vocab/vectors
/kaggle/input/tse-spacy-model/models/model_neu/ner/cfg
/kaggle/input/tse-spacy-model/models/model_neu/ner/model
/kaggle/input/tse-spacy-model/models/model_neu/ner/moves
/kaggle/input/tse-spacy-model/models/model_pos/meta.json
/kaggle/input/tse-spacy-model/models/model_pos/tokenizer
/kaggle/input/tse-spacy-model/models/model_pos/vocab/key2row
/kaggle/input/tse-spacy-model/models/model_pos/vocab/strings.json
/kaggle/input/tse-spacy-model/models/model_pos/vocab/lexemes.bin
/kaggle/input/tse-spacy-model/models/model_pos/vocab/vectors
/kaggle/input/tse-spacy-model/models/model_pos/ner/cfg
/kaggle/input/tse-spacy-model/models/model_pos/ner/model
/kaggle/input/tse-spacy-model/models/model_pos/ner/moves
/kaggle/input/tse-spacy-model/models/model_neg/meta.json
/kaggle/input/tse-spacy-model/models/model_neg/tokenizer
/kaggle/input/tse-spacy-model/models/model_neg/vocab/key2row
/kaggle/input/tse-spacy-model/models/model_neg/vocab/strings.json
/kaggle/input/tse-spacy-model/models/model_neg/vocab/lexemes.bin
/kaggle/input/tse-spacy-model/models/model_neg/vocab/vectors
/kaggle/input/tse-spacy-model/models/model_neg/ner/cfg
/kaggle/input/tse-spacy-model/models/model_neg/ner/model
/kaggle/input/tse-spacy-model/models/model_neg/ner/moves
/kaggle/input/tse-spacy-model/models2/model_neg/meta.json
/kaggle/input/tse-spacy-model/models2/model_neg/tokenizer
/kaggle/input/tse-spacy-model/models2/model_neg/vocab/key2row
/kaggle/input/tse-spacy-model/models2/model_neg/vocab/strings.json
/kaggle/input/tse-spacy-model/models2/model_neg/vocab/lexemes.bin
/kaggle/input/tse-spacy-model/models2/model_neg/vocab/vectors
/kaggle/input/tse-spacy-model/models2/model_neg/ner/cfg
/kaggle/input/tse-spacy-model/models2/model_neg/ner/model
/kaggle/input/tse-spacy-model/models2/model_neg/ner/moves
```

```
/kaggle/input/masks-for-wordclouds/twitter_mask.png
/kaggle/input/tweet-sentiment-extraction/train.csv
/kaggle/input/tweet-sentiment-extraction/test.csv
/kaggle/input/tweet-sentiment-extraction/sample_submission.csv
```

[ ]:
```
def random_colours(number_of_colors):
    '''
    Simple function for random colours generation.
    Input:
        number_of_colors - integer value indicating the number of colours which␣
 ↪are going to be generated.
    Output:
        Color in the following format: ['#E86DA4'] .
    '''
    colors = []
    for i in range(number_of_colors):
        colors.append("#"+''.join([random.choice('0123456789ABCDEF') for j in␣
 ↪range(6)]))
    return colors
```

4 Reading the Data

[ ]:
```
train = pd.read_csv('/kaggle/input/tweet-sentiment-extraction/train.csv')
test = pd.read_csv('/kaggle/input/tweet-sentiment-extraction/test.csv')
ss = pd.read_csv('/kaggle/input/tweet-sentiment-extraction/sample_submission.
 ↪csv')
```

[ ]:
```
print(train.shape)
print(test.shape)
```

```
(27481, 4)
(3534, 3)
```

So We have 27486 tweets in the train set and 3535 tweets in the test set

[ ]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27481 entries, 0 to 27480
Data columns (total 4 columns):
textID          27481 non-null object
text            27480 non-null object
selected_text   27480 non-null object
sentiment       27481 non-null object
dtypes: object(4)
memory usage: 858.9+ KB
```

[ ]:
```
train.dropna(inplace=True)
```

[ ]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3534 entries, 0 to 3533
Data columns (total 3 columns):
textID       3534 non-null object
text         3534 non-null object
sentiment    3534 non-null object
dtypes: object(3)
memory usage: 83.0+ KB
```

There are no null Values in the test set

5 Exploratory Data Analysis.

[ ]:
```
train.head()
```

[ ]:

	textID	text	selected_text	sentiment
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
2	088c60f138	my boss is bullying me…	bullying me	negative
3	9642c003ef	what interview! leave me alone	leave me alone	negative
4	358bd9e861	Sons of ****, why couldn`t they put them on t…	Sons of ****,	negative

[ ]:
```
train.describe()
```

[ ]:

	textID	text	selected_text	sentiment
count	27480	27480	27480	27480
unique	27480	27480	22463	3
top	609f4a0832	can`t wait to crack it open and no doubt will…	good	neutral
freq	1	1	199	11117

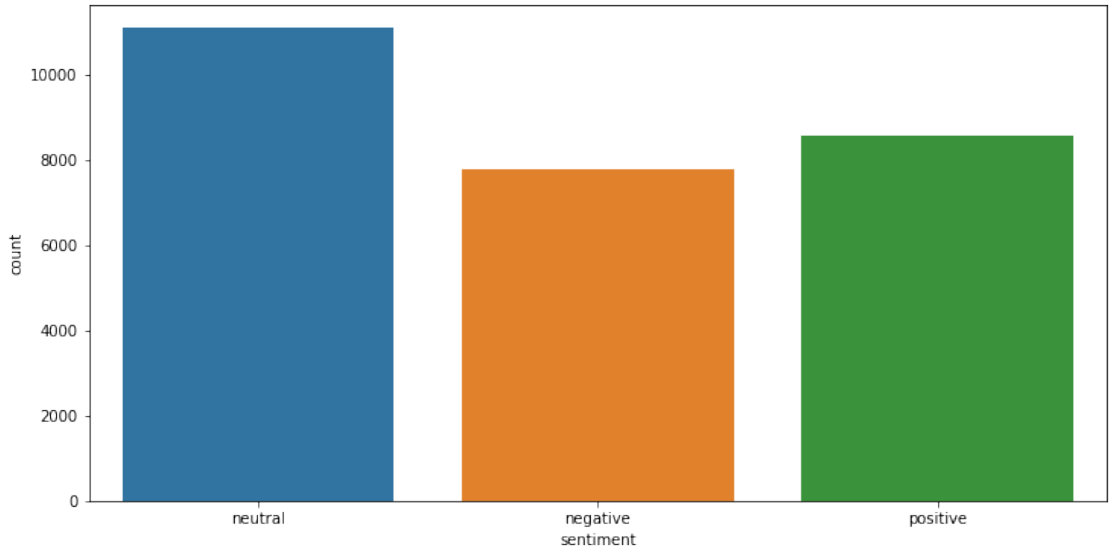
Lets look at the distribution of tweets in the train set

```
[ ]: temp = train.groupby('sentiment').count()['text'].reset_index().
      ↪sort_values(by='text',ascending=False)
     temp.style.background_gradient(cmap='Purples')
```

```
[ ]: <pandas.io.formats.style.Styler at 0x7f996bb18f60>
```

```
[ ]: plt.figure(figsize=(12,6))
     sns.countplot(x='sentiment',data=train)
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f99b80710b8>
```



Let's draw a Funnel-Chart for better visualization

```
[ ]: fig = go.Figure(go.Funnelarea(
         text =temp.sentiment,
         values = temp.text,
         title = {"position": "top center", "text": "Funnel-Chart of Sentiment␣
      ↪Distribution"}
         ))
     fig.show()
```

5.1 Generating Meta-Features

```
[ ]: def jaccard(str1, str2):
         a = set(str1.lower().split())
         b = set(str2.lower().split())
         c = a.intersection(b)
         return float(len(c)) / (len(a) + len(b) - len(c))
```

[ ]:
```
results_jaccard=[]

for ind,row in train.iterrows():
    sentence1 = row.text
    sentence2 = row.selected_text

    jaccard_score = jaccard(sentence1,sentence2)
    results_jaccard.append([sentence1,sentence2,jaccard_score])
```

[ ]:
```
jaccard = pd.
 ↪DataFrame(results_jaccard,columns=["text","selected_text","jaccard_score"])
train = train.merge(jaccard,how='outer')
```

[ ]:
```
train['Num_words_ST'] = train['selected_text'].apply(lambda x:len(str(x).
 ↪split())) #Number Of words in Selected Text
train['Num_word_text'] = train['text'].apply(lambda x:len(str(x).split()))␣
 ↪#Number Of words in main text
train['difference_in_words'] = train['Num_word_text'] - train['Num_words_ST']␣
 ↪#Difference in Number of words text and Selected Text
```

[ ]:
```
train.head()
```

[ ]:

	textID	text	selected_text	sentiment	jaccard_score	Num_words_ST	Num_word_text	difference_in_words
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral	1.000000	7	7	0
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative	0.200000	2	10	8
2	088c60f138	my boss is bullying me…	bullying me	negative	0.166667	2	5	3
3	9642c003ef	what interview! leave me alone	leave me alone	negative	0.600000	3	5	2
4	358bd9e861	Sons of ****, why couldn`t they put them on t…	Sons of ****,	negative	0.214286	3	14	11

Let's look at the distribution of Meta-Features

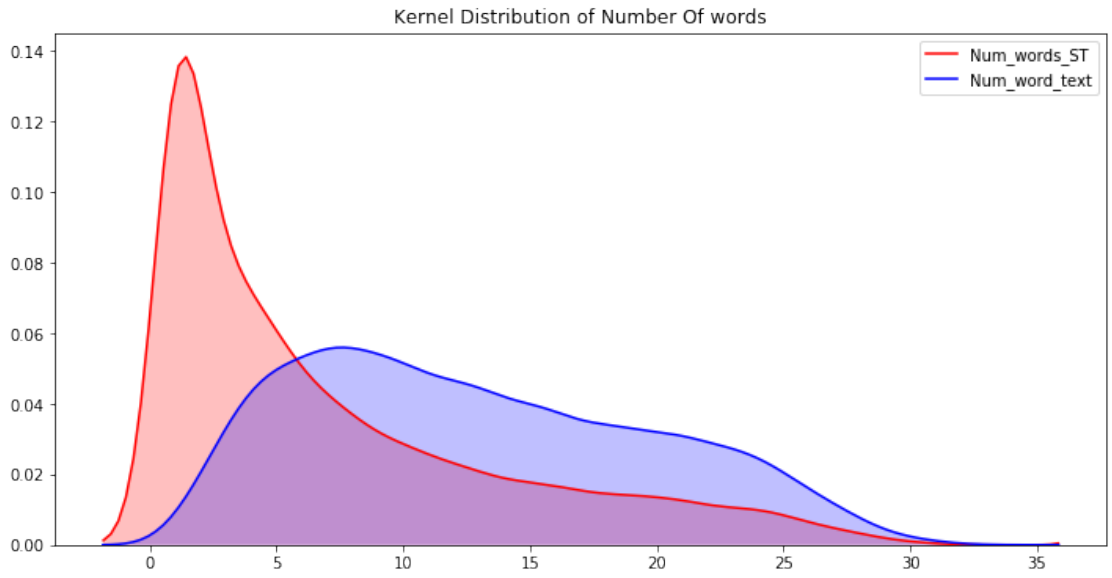
[ ]:
```
hist_data = [train['Num_words_ST'],train['Num_word_text']]

group_labels = ['Selected_Text', 'Text']
```

```
# Create distplot with custom bin_size
fig = ff.create_distplot(hist_data, group_labels,show_curve=False)
fig.update_layout(title_text='Distribution of Number Of words')
fig.update_layout(
    autosize=False,
    width=900,
    height=700,
    paper_bgcolor="LightSteelBlue",
)
fig.show()
```

- The number of words plot is really interesting ,the tweets having number of words greater than 25 are very less and thus the number of words distribution plot is right skewed

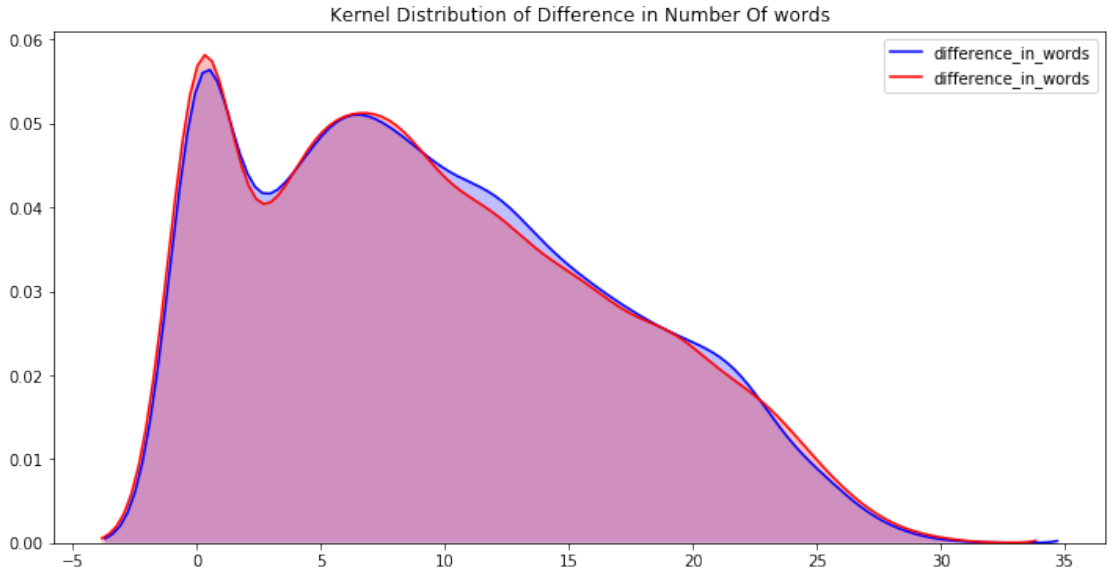
[ ]:
```
plt.figure(figsize=(12,6))
p1=sns.kdeplot(train['Num_words_ST'], shade=True, color="r").set_title('Kernel␣
↪Distribution of Number Of words')
p1=sns.kdeplot(train['Num_word_text'], shade=True, color="b")
```



Now It will be more interesting to see the differnce in number of words and jaccard_scores across different Sentiments

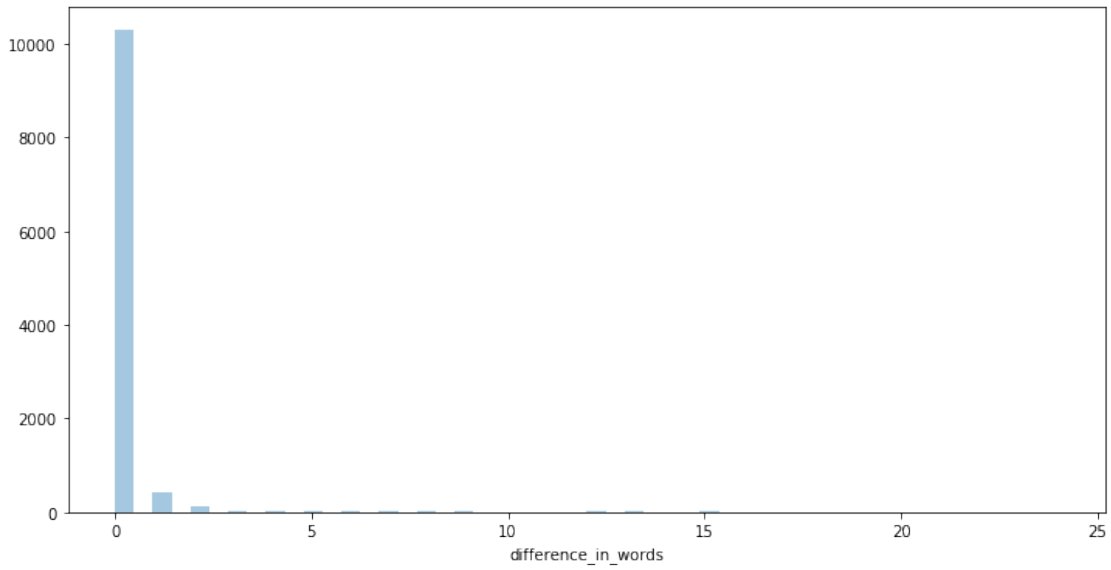
[ ]:
```
plt.figure(figsize=(12,6))
p1=sns.kdeplot(train[train['sentiment']=='positive']['difference_in_words'],␣
↪shade=True, color="b").set_title('Kernel Distribution of Difference in␣
↪Number Of words')
```

```
p2=sns.kdeplot(train[train['sentiment']=='negative']['difference_in_words'],␣
↪shade=True, color="r")
```



[ ]:
```
plt.figure(figsize=(12,6))
sns.
↪distplot(train[train['sentiment']=='neutral']['difference_in_words'],kde=False)
```

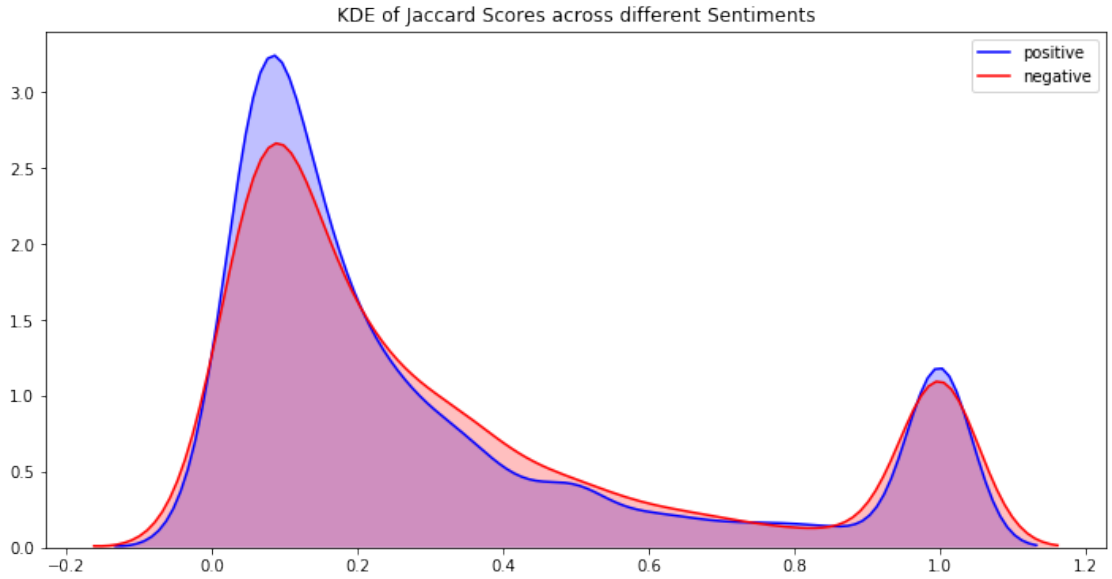
[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9968f22ac8>
```



I was not able to plot kde plot for neutral tweets because most of the values for difference in number of words were zero. We can see it clearly now ,if we had used the feature in the starting we would have known that text and selected text are mostly the same for neutral tweets,thus its always important to keep the end goal in mind while performing EDA

[ ]:
```
plt.figure(figsize=(12,6))
p1=sns.kdeplot(train[train['sentiment']=='positive']['jaccard_score'],␣
 ↪shade=True, color="b").set_title('KDE of Jaccard Scores across different␣
 ↪Sentiments')
p2=sns.kdeplot(train[train['sentiment']=='negative']['jaccard_score'],␣
 ↪shade=True, color="r")
plt.legend(labels=['positive','negative'])
```

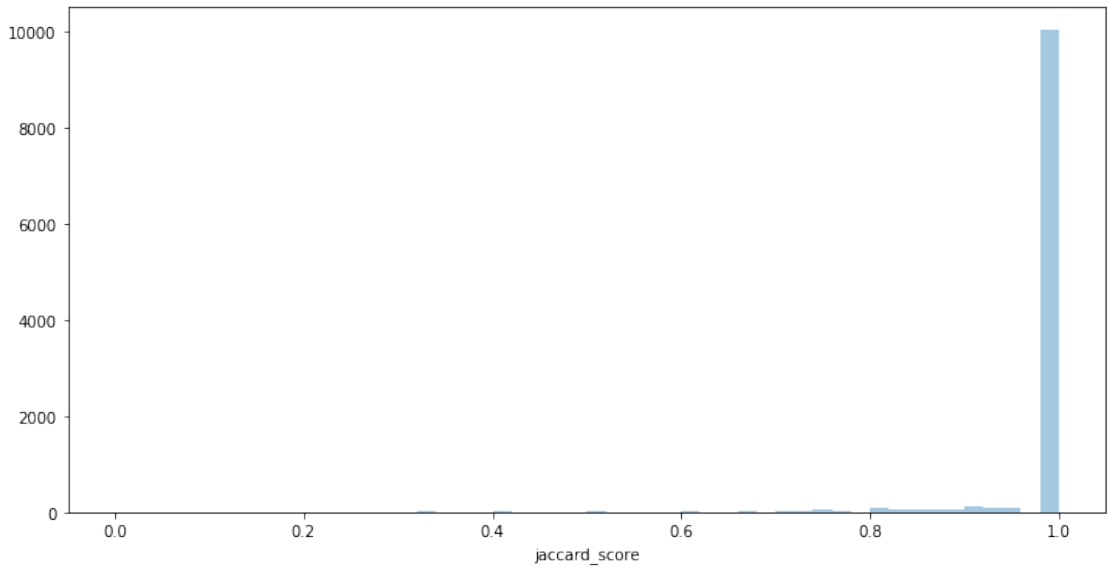
[ ]:
```
<matplotlib.legend.Legend at 0x7f99682eac50>
```



I was not able to plot kde of jaccard_scores of neutral tweets for the same reason,thus I will plot a distribution plot

[ ]:
```
plt.figure(figsize=(12,6))
sns.distplot(train[train['sentiment']=='neutral']['jaccard_score'],kde=False)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99682fce80>
```



Here, we observe notable patterns:

Tweets classified as positive and negative exhibit a high kurtosis, indicating that their values are densely clustered within two narrow regions. Conversely, neutral tweets demonstrate a lower kurtosis, revealing a noticeable increase in density around the value of 1. For clarification:

Kurtosis quantifies the sharpness of a distribution's peak and the extent of its spread around that peak. Skewness assesses the degree to which a distribution deviates from being normally distributed.

6 EDA Summary

The Jaccard score plot reveals a notable peak for both negative and positive tweets around a score of 1. This suggests a grouping of tweets with a high degree of similarity between the text and the selected texts. Identifying these clusters could enable us to accurately predict the selected texts for these tweets, regardless of their category. Exploring this further, an intriguing approach could be to examine tweets with fewer than three words in the text, as these instances might be using the entire text as the selected text.

[ ]:
```
k = train[train['Num_word_text']<=2]
```

[ ]:
```
k.groupby('sentiment').mean()['jaccard_score']
```

[ ]:
```
sentiment
negative    0.788580
neutral     0.977805
positive    0.765700
Name: jaccard_score, dtype: float64
```

We can see that there is similarity between text and selected text .Let's have closer look

```
[ ]: k[k['sentiment']=='positive']
```

```
[ ]:           textID                                text  \
68     fa2654e730                              Chilliin
80     bbbc46889b   THANK YYYYYYYYYOOOOOOOOOOUUUUU!
170    f3d95b57b1                         good morning
278    89d5b3f0b5                               Thanks
429    a78ef3e0d0                           Goodmorning
…           …                                     …
26689  e80c242d6a                           Goodnight;
26725  aad244f37d                                *hug*
26842  a46571fe12                           congrats!
26959  49a942e9b1                       Happy birthday.
27292  47c474aaf1                           Good choice

                         selected_text sentiment  jaccard_score  Num_words_ST  \
68                             Chilliin  positive            1.0             1
80     THANK YYYYYYYYYOOOOOOOOOOUUUUU!  positive            1.0             2
170                        good morning  positive            1.0             2
278                              Thanks  positive            1.0             1
429                          Goodmorning  positive            1.0             1
…                                    …       …             …            …
26689                         Goodnight;  positive            1.0             1
26725                              *hug*  positive            1.0             1
26842                        congrats!  positive            1.0             1
26959                    Happy birthday.  positive            1.0             2
27292                               Good  positive            0.5             1

       Num_word_text  difference_in_words
68                 1                    0
80                 2                    0
170                2                    0
278                1                    0
429                1                    0
…                …                    …
26689              1                    0
26725              1                    0
26842              1                    0
26959              2                    0
27292              2                    1

[207 rows x 8 columns]
```

Thus its clear that most of the times , text is used as selected text.We can improve this by preprocessing the text which have word length less than 3.We will remember this information and use it in model building

6.0.1 Cleaning the Corpus

Now Before We Dive into extracting information out of words in text and selected text,let's first clean the data

[ ]:
```
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove␣
 ↪punctuation
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

[ ]:
```
train['text'] = train['text'].apply(lambda x:clean_text(x))
train['selected_text'] = train['selected_text'].apply(lambda x:clean_text(x))
```

[ ]:
```
train.head()
```

[ ]:

	textID	text	selected_text	sentiment	jaccard_score	Num_words_ST	Num_word_text	difference_in_words
0	cb774db0d1	id have responded if i were going	id have responded if i were going	neutral	1.000000	7	7	0
1	549e992a42	sooo sad i will miss you here in san diego	sooo sad	negative	0.200000	2	10	8
2	088c60f138	my boss is bullying me	bullying me	negative	0.166667	2	5	3
3	9642c003ef	what interview leave me alone	leave me alone	negative	0.600000	3	5	2
4	358bd9e861	sons of why couldnt they put them on the rel…	sons of	negative	0.214286	3	14	11

6.1 Most Common words in our Target-Selected Text

[ ]:
```
train['temp_list'] = train['selected_text'].apply(lambda x:str(x).split())
top = Counter([item for sublist in train['temp_list'] for item in sublist])
temp = pd.DataFrame(top.most_common(20))
temp.columns = ['Common_words','count']
temp.style.background_gradient(cmap='Blues')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f995c117320>
```

[ ]:
```
fig = px.bar(temp, x="count", y="Common_words", title='Commmon Words in␣
 ↪Selected Text', orientation='h',
             width=700, height=700,color='Common_words')
fig.show()
```

OOPS!While we cleaned our dataset we didnt remove the stop words and hence we can see the most coomon word is 'to' . Let's try again after removing the stopwords

[ ]:
```
def remove_stopword(x):
    return [y for y in x if y not in stopwords.words('english')]
train['temp_list'] = train['temp_list'].apply(lambda x:remove_stopword(x))
```

[ ]:
```
top = Counter([item for sublist in train['temp_list'] for item in sublist])
temp = pd.DataFrame(top.most_common(20))
temp = temp.iloc[1:,:]
temp.columns = ['Common_words','count']
temp.style.background_gradient(cmap='Purples')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f995c116898>
```

[ ]:
```
fig = px.treemap(temp, path=['Common_words'], values='count',title='Tree of␣
 ↪Most Common Words')
fig.show()
```

7 Most Common words in Text

Let's also look at the most common words in Text

[ ]:
```
train['temp_list1'] = train['text'].apply(lambda x:str(x).split()) #List of␣
 ↪words in every row for text
train['temp_list1'] = train['temp_list1'].apply(lambda x:remove_stopword(x))␣
 ↪#Removing Stopwords
```

[ ]:
```
top = Counter([item for sublist in train['temp_list1'] for item in sublist])
temp = pd.DataFrame(top.most_common(25))
temp = temp.iloc[1:,:]
temp.columns = ['Common_words','count']
temp.style.background_gradient(cmap='Blues')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f994688feb8>
```

So the first two common word was I'm so I removed it and took data from second row

[ ]:
```
fig = px.bar(temp, x="count", y="Common_words", title='Commmon Words in Text',␣
 ↪orientation='h',
             width=700, height=700,color='Common_words')
fig.show()
```

SO we can see the Most common words in Selected text and Text are almost the same,which was obvious

8 Most common words Sentiments Wise

Let's look at the most common words in different sentiments

[ ]:
```
Positive_sent = train[train['sentiment']=='positive']
Negative_sent = train[train['sentiment']=='negative']
Neutral_sent = train[train['sentiment']=='neutral']
```

[ ]:
```
#MosT common positive words
top = Counter([item for sublist in Positive_sent['temp_list'] for item in␣
 ↪sublist])
temp_positive = pd.DataFrame(top.most_common(20))
temp_positive.columns = ['Common_words','count']
temp_positive.style.background_gradient(cmap='Greens')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f9946dfe9e8>
```

[ ]:
```
fig = px.bar(temp_positive, x="count", y="Common_words", title='Most Commmon␣
 ↪Positive Words', orientation='h',
             width=700, height=700,color='Common_words')
fig.show()
```

[ ]:
```
#MosT common negative words
top = Counter([item for sublist in Negative_sent['temp_list'] for item in␣
 ↪sublist])
temp_negative = pd.DataFrame(top.most_common(20))
temp_negative = temp_negative.iloc[1:,:]
temp_negative.columns = ['Common_words','count']
temp_negative.style.background_gradient(cmap='Reds')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f99472bb908>
```

[ ]:
```
fig = px.treemap(temp_negative, path=['Common_words'],␣
 ↪values='count',title='Tree Of Most Common Negative Words')
fig.show()
```

```
[ ]: #MosT common Neutral words
top = Counter([item for sublist in Neutral_sent['temp_list'] for item in␣
 ↪sublist])
temp_neutral = pd.DataFrame(top.most_common(20))
temp_neutral = temp_neutral.loc[1:,:]
temp_neutral.columns = ['Common_words','count']
temp_neutral.style.background_gradient(cmap='Reds')
```

```
[ ]: <pandas.io.formats.style.Styler at 0x7f9946dff710>
```

```
[ ]: fig = px.bar(temp_neutral, x="count", y="Common_words", title='Most Commmon␣
 ↪Neutral Words', orientation='h',
             width=700, height=700,color='Common_words')
fig.show()
```

```
[ ]: fig = px.treemap(temp_neutral, path=['Common_words'],␣
 ↪values='count',title='Tree Of Most Common Neutral Words')
fig.show()
```

- We can see words like get,go,dont,got,u,cant,lol,like are common in all three segments . That's interesting because words like dont and cant are more of negative nature and words like lol are more of positive nature.Does this mean our data is incorrectly labelled , we will have more insights on this after N-gram analysis
- It will be interesting to see the word unique to different sentiments

8.1 Let's Look at Unique Words in each Segment

We will look at unique words in each segment in the Following Order: * Positive * Negative * Neutral

```
[ ]: raw_text = [word for word_list in train['temp_list1'] for word in word_list]
```

```
[ ]: def words_unique(sentiment,numwords,raw_words):
    '''
    Input:
        segment - Segment category (ex. 'Neutral');
        numwords - how many specific words do you want to see in the final␣
 ↪result;
        raw_words - list  for item in train_data[train_data.segments ==␣
 ↪segments]['temp_list1']:
    Output:
        dataframe giving information about the name of the specific ingredient␣
 ↪and how many times it occurs in the chosen cuisine (in descending order␣
 ↪based on their counts)..

    '''
    allother = []
    for item in train[train.sentiment != sentiment]['temp_list1']:
```

```
        for word in item:
            allother .append(word)
    allother  = list(set(allother ))

    specificnonly = [x for x in raw_text if x not in allother]

    mycounter = Counter()

    for item in train[train.sentiment == sentiment]['temp_list1']:
        for word in item:
            mycounter[word] += 1
    keep = list(specificnonly)

    for word in list(mycounter):
        if word not in keep:
            del mycounter[word]

    Unique_words = pd.DataFrame(mycounter.most_common(numwords), columns =␣
↪['words','count'])

    return Unique_words
```

8.1.1 Positive Tweets

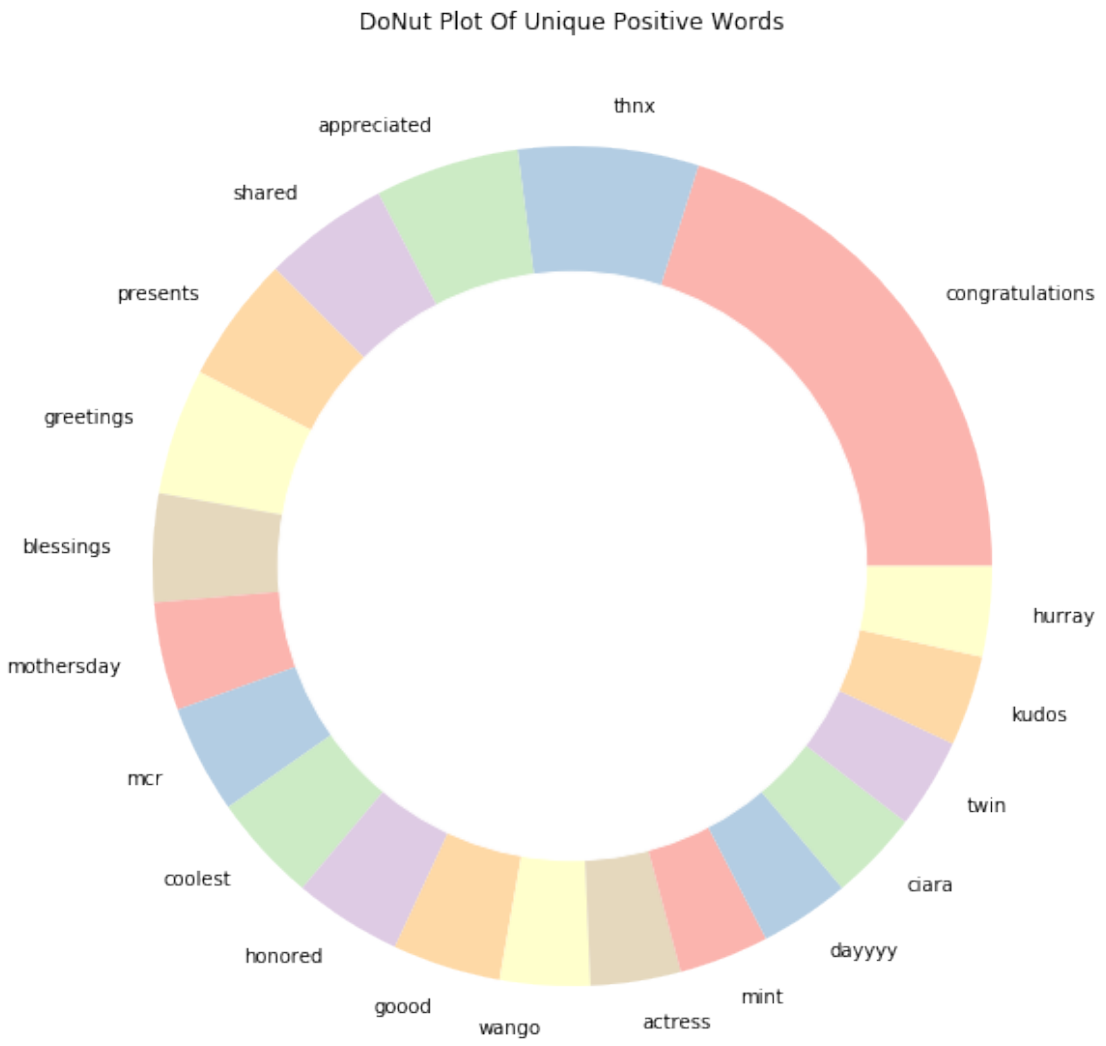
[ ]:
```
Unique_Positive= words_unique('positive', 20, raw_text)
print("The top 20 unique words in Positive Tweets are:")
Unique_Positive.style.background_gradient(cmap='Greens')
```

The top 20 unique words in Positive Tweets are:

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f994741d940>
```

[ ]:
```
fig = px.treemap(Unique_Positive, path=['words'], values='count',title='Tree Of␣
↪Unique Positive Words')
fig.show()
```

[ ]:
```
from palettable.colorbrewer.qualitative import Pastel1_7
plt.figure(figsize=(16,10))
my_circle=plt.Circle((0,0), 0.7, color='white')
plt.pie(Unique_Positive['count'], labels=Unique_Positive.words,␣
↪colors=Pastel1_7.hex_colors)
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.title('DoNut Plot Of Unique Positive Words')
plt.show()
```



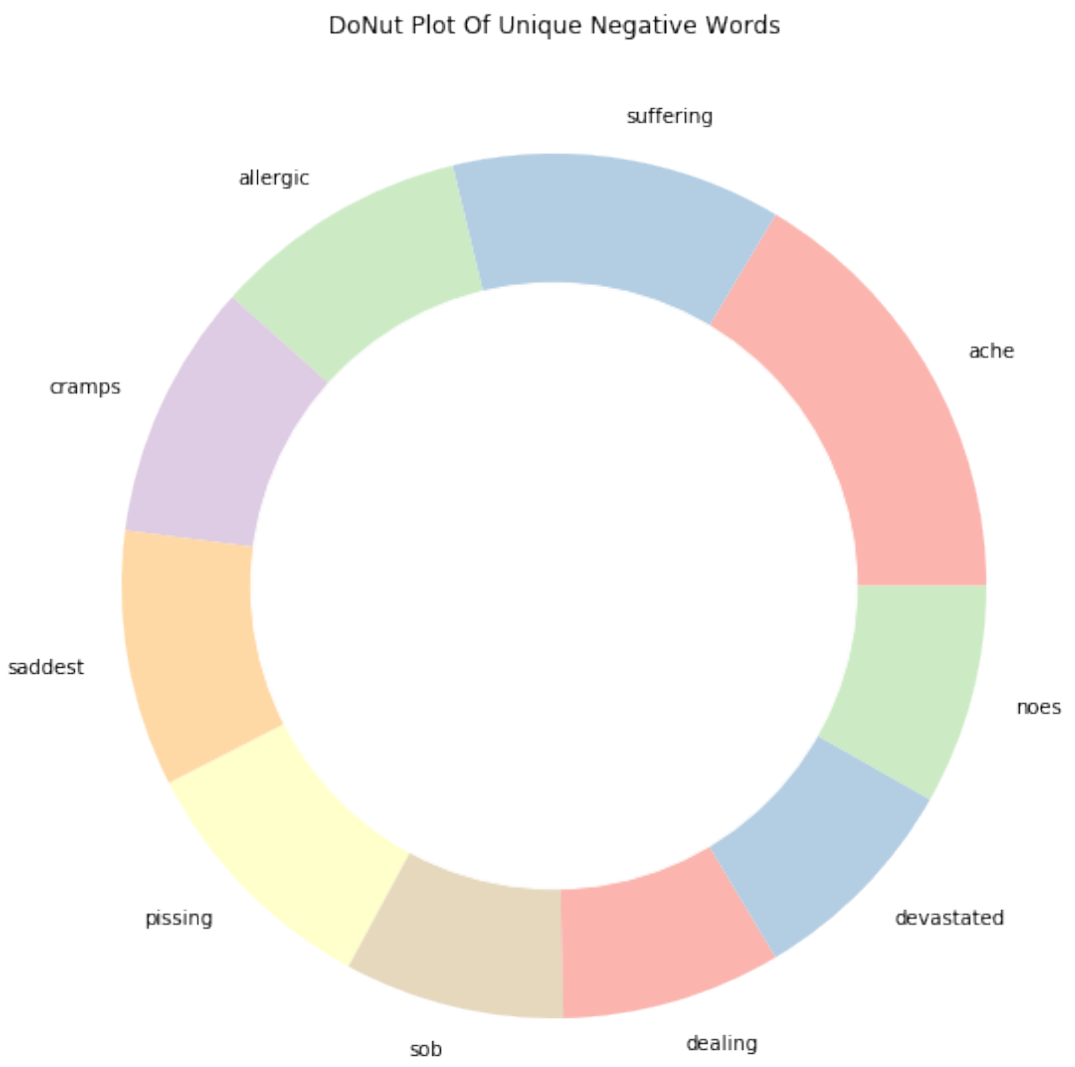
[ ]:
```
Unique_Negative= words_unique('negative', 10, raw_text)
print("The top 10 unique words in Negative Tweets are:")
Unique_Negative.style.background_gradient(cmap='Reds')
```

```
The top 10 unique words in Negative Tweets are:
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f994741c278>
```

[ ]:
```
from palettable.colorbrewer.qualitative import Pastel1_7
plt.figure(figsize=(16,10))
my_circle=plt.Circle((0,0), 0.7, color='white')
plt.rcParams['text.color'] = 'black'
plt.pie(Unique_Negative['count'], labels=Unique_Negative.words,␣
 ↪colors=Pastel1_7.hex_colors)
```

```
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.title('DoNut Plot Of Unique Negative Words')
plt.show()
```

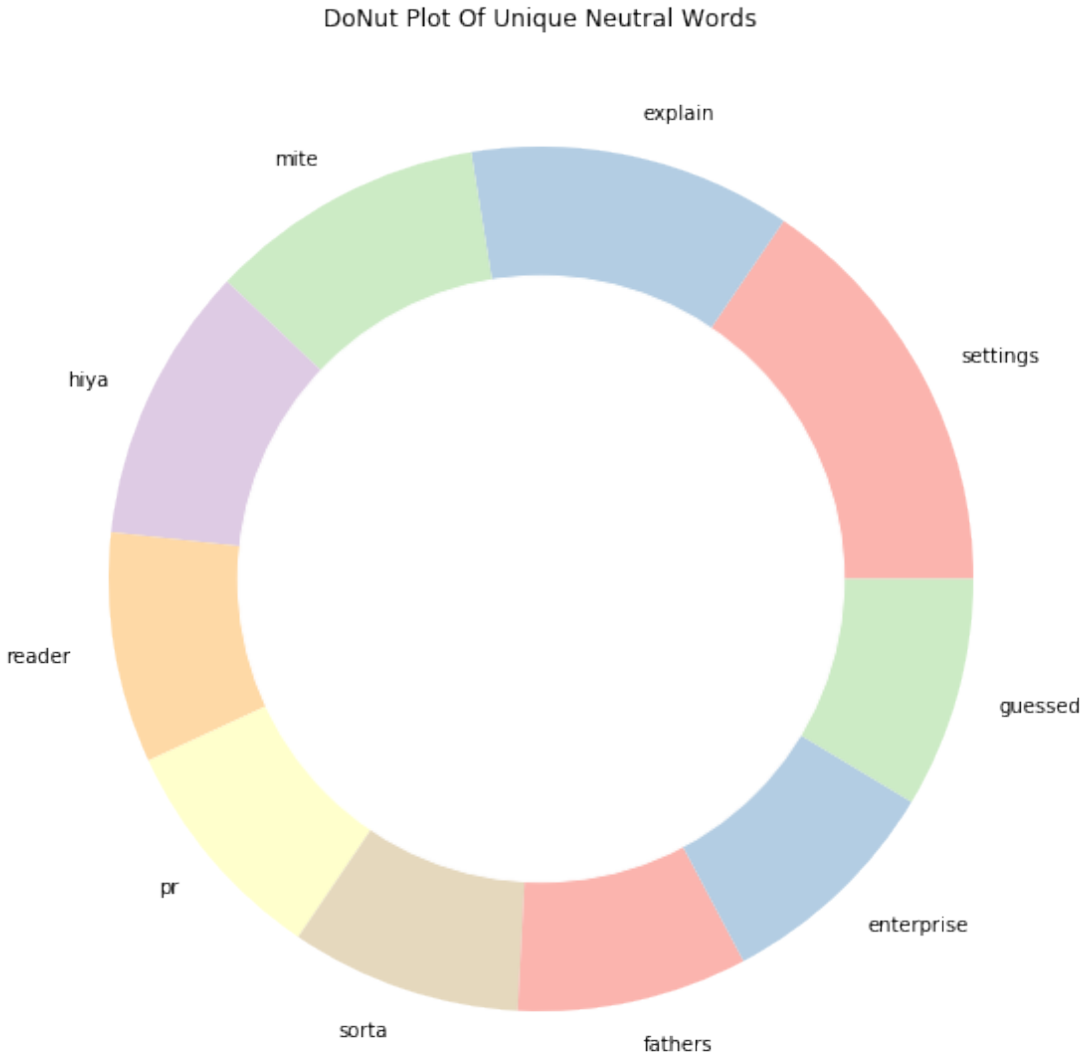


[ ]:
```
Unique_Neutral= words_unique('neutral', 10, raw_text)
print("The top 10 unique words in Neutral Tweets are:")
Unique_Neutral.style.background_gradient(cmap='Oranges')
```

```
The top 10 unique words in Neutral Tweets are:
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f9946a65898>
```

[ ]:
```
from palettable.colorbrewer.qualitative import Pastel1_7
plt.figure(figsize=(16,10))
my_circle=plt.Circle((0,0), 0.7, color='white')
plt.pie(Unique_Neutral['count'], labels=Unique_Neutral.words, colors=Pastel1_7.
  ↪hex_colors)
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.title('DoNut Plot Of Unique Neutral Words')
plt.show()
```



By Looking at the Unique Words of each sentiment,we now have much more clarity about the data,these unique words are very strong determiners of Sentiment of tweets

8.2 It's Time For WordClouds

We will be building wordclouds in the following order:

- WordCloud of Neutral Tweets
- WordCloud of Positive Tweets
- WordCloud of Negative Tweets

[ ]:
```
def plot_wordcloud(text, mask=None, max_words=200, max_font_size=100,␣
 ↪figure_size=(24.0,16.0), color = 'white',
                   title = None, title_size=40, image_color=False):
    stopwords = set(STOPWORDS)
    more_stopwords = {'u', "im"}
    stopwords = stopwords.union(more_stopwords)

    wordcloud = WordCloud(background_color=color,
                    stopwords = stopwords,
                    max_words = max_words,
                    max_font_size = max_font_size,
                    random_state = 42,
                    width=400,
                    height=200,
                    mask = mask)
    wordcloud.generate(str(text))

    plt.figure(figsize=figure_size)
    if image_color:
        image_colors = ImageColorGenerator(mask);
        plt.imshow(wordcloud.recolor(color_func=image_colors),␣
 ↪interpolation="bilinear");
        plt.title(title, fontdict={'size': title_size,
                                   'verticalalignment': 'bottom'})
    else:
        plt.imshow(wordcloud);
        plt.title(title, fontdict={'size': title_size, 'color': 'black',
                                   'verticalalignment': 'bottom'})
    plt.axis('off');
    plt.tight_layout()
d = '/kaggle/input/masks-for-wordclouds/'
```

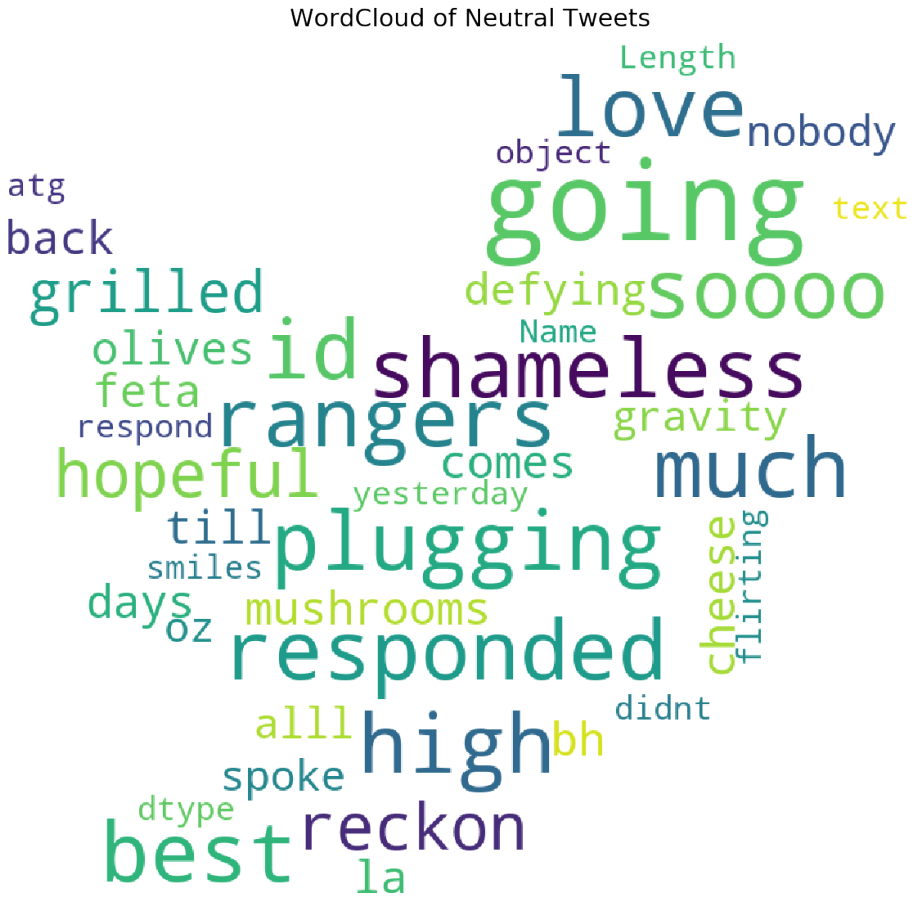
I have added more words like im , u (that we say were there in the most common words,disturbing our analysis) as stopwords

WORDCLOUD OF NEUTRAL TWEETS We Have already visualized our Most Common Negative words ,but Wordclouds Provide us much more clarity

[ ]:
```

```

```
pos_mask = np.array(Image.open(d+ 'twitter_mask.png'))
plot_wordcloud(Neutral_sent.
  ↪text,mask=pos_mask,color='white',max_font_size=100,title_size=30,title="WordCloud␣
  ↪of Neutral Tweets")
```



[ ]:
```
plot_wordcloud(Positive_sent.text,mask=pos_mask,title="Word Cloud Of Positive␣
  ↪tweets",title_size=30)
```

Word Cloud Of Positive tweets

[ ]:
```
plot_wordcloud(Negative_sent.text,mask=pos_mask,title="Word Cloud of Negative␣
↪Tweets",color='white',title_size=30)
```

Word Cloud of Negative Tweets


```
[ ]: df_train = pd.read_csv('/kaggle/input/tweet-sentiment-extraction/train.csv')
     df_test = pd.read_csv('/kaggle/input/tweet-sentiment-extraction/test.csv')
     df_submission = pd.read_csv('/kaggle/input/tweet-sentiment-extraction/
      ↪sample_submission.csv')
```

```
[ ]: df_train['Num_words_text'] = df_train['text'].apply(lambda x:len(str(x).
      ↪split())) #Number Of words in main Text in train set
```

```
[ ]: df_train = df_train[df_train['Num_words_text']>=3]
```

```
[ ]: def save_model(output_dir, nlp, new_model_name):
         ''' This Function Saves model to
         given output directory'''

         output_dir = f'../working/{output_dir}'
         if output_dir is not None:
             if not os.path.exists(output_dir):
                 os.makedirs(output_dir)
```

```
        nlp.meta["name"] = new_model_name
        nlp.to_disk(output_dir)
        print("Saved model to", output_dir)
```

[ ]:
```
# pass model = nlp if you want to train on top of existing model

def train(train_data, output_dir, n_iter=20, model=None):
    """Load the model, set up the pipeline and train the entity recognizer."""
    ""
    if model is not None:
        nlp = spacy.load(output_dir)  # load existing spaCy model
        print("Loaded model '%s'" % model)
    else:
        nlp = spacy.blank("en")  # create blank Language class
        print("Created blank 'en' model")

    # create the built-in pipeline components and add them to the pipeline
    # nlp.create_pipe works for built-ins that are registered with spaCy
    if "ner" not in nlp.pipe_names:
        ner = nlp.create_pipe("ner")
        nlp.add_pipe(ner, last=True)
    # otherwise, get it so we can add labels
    else:
        ner = nlp.get_pipe("ner")

    # add labels
    for _, annotations in train_data:
        for ent in annotations.get("entities"):
            ner.add_label(ent[2])

    # get names of other pipes to disable them during training
    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
    with nlp.disable_pipes(*other_pipes):  # only train NER
        # sizes = compounding(1.0, 4.0, 1.001)
        # batch up the examples using spaCy's minibatch
        if model is None:
            nlp.begin_training()
        else:
            nlp.resume_training()


        for itn in tqdm(range(n_iter)):
            random.shuffle(train_data)
            batches = minibatch(train_data, size=compounding(4.0, 500.0, 1.001))
            losses = {}
            for batch in batches:
                texts, annotations = zip(*batch)
```

```
                nlp.update(texts,  # batch of texts
                           annotations,  # batch of annotations
                           drop=0.5,   # dropout - make it harder to memorise␣
 ↪data
                           losses=losses,
                           )
            print("Losses", losses)
    save_model(output_dir, nlp, 'st_ner')
```

[ ]:
```
def get_model_out_path(sentiment):
    '''
    Returns Model output path
    '''
    model_out_path = None
    if sentiment == 'positive':
        model_out_path = 'models/model_pos'
    elif sentiment == 'negative':
        model_out_path = 'models/model_neg'
    return model_out_path
```

[ ]:
```
def get_training_data(sentiment):
    '''
    Returns Trainong data in the format needed to train spacy NER
    '''
    train_data = []
    for index, row in df_train.iterrows():
        if row.sentiment == sentiment:
            selected_text = row.selected_text
            text = row.text
            start = text.find(selected_text)
            end = start + len(selected_text)
            train_data.append((text, {"entities": [[start, end,␣
 ↪'selected_text']]}))
    return train_data
```

Training models for Positive and Negative tweets

[ ]:
```
sentiment = 'positive'

train_data = get_training_data(sentiment)
model_path = get_model_out_path(sentiment)
# For DEmo Purposes I have taken 3 iterations you can train the model as you␣
 ↪want
train(train_data, model_path, n_iter=3, model=None)
```

```
Created blank 'en' model

 33%|      | 1/3 [00:52<01:45, 52.84s/it]
```

```
Losses {'ner': 33900.41114743876}
 67%|       | 2/3 [01:45<00:52, 52.68s/it]
Losses {'ner': 30517.31436416012}
100%|      | 3/3 [02:37<00:00, 52.62s/it]
Losses {'ner': 28912.80112953766}
Saved model to ../working/models/model_pos
```

[ ]:
```
sentiment = 'negative'

train_data = get_training_data(sentiment)
model_path = get_model_out_path(sentiment)

train(train_data, model_path, n_iter=3, model=None)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Created blank 'en' model
 33%|        | 1/3 [00:50<01:41, 50.56s/it]
Losses {'ner': 31926.559298905544}
 67%|       | 2/3 [01:40<00:50, 50.32s/it]
Losses {'ner': 28526.67285699508}
100%|      | 3/3 [02:30<00:00, 50.02s/it]
Losses {'ner': 27071.952286979005}
Saved model to ../working/models/model_neg
```

8.2.1 Predicting with the trained Model

[ ]:
```
def predict_entities(text, model):
    doc = model(text)
    ent_array = []
    for ent in doc.ents:
        start = text.find(ent.text)
        end = start + len(ent.text)
        new_int = [start, end, ent.label_]
        if new_int not in ent_array:
            ent_array.append([start, end, ent.label_])
    selected_text = text[ent_array[0][0]: ent_array[0][1]] if len(ent_array) >␣
 ↪0 else text
    return selected_text
```

[ ]:
```
selected_texts = []
MODELS_BASE_PATH = '../input/tse-spacy-model/models/'

if MODELS_BASE_PATH is not None:
    print("Loading Models  from ", MODELS_BASE_PATH)
    model_pos = spacy.load(MODELS_BASE_PATH + 'model_pos')
    model_neg = spacy.load(MODELS_BASE_PATH + 'model_neg')

    for index, row in df_test.iterrows():
        text = row.text
        output_str = ""
        if row.sentiment == 'neutral' or len(text.split()) <= 2:
            selected_texts.append(text)
        elif row.sentiment == 'positive':
            selected_texts.append(predict_entities(text, model_pos))
        else:
            selected_texts.append(predict_entities(text, model_neg))

df_test['selected_text'] = selected_texts
```

```
Loading Models  from  ../input/tse-spacy-model/models/
```

[ ]:
```
df_submission['selected_text'] = df_test['selected_text']
df_submission.to_csv("submission.csv", index=False)
display(df_submission.head(10))
```

	textID	selected_text
0	f87dea47db	Last session of the day http://twitpic.com/67ezh
1	96d74cb729	exciting
2	eee518ae67	Recession
3	01082688c6	happy bday!
4	33987a8ee5	I like it!!
5	726e501993	visitors!
6	261932614e	HATES
7	afa11da83f	blocked
8	e64208b4ef	and within a short time of the last clue all …
9	37bcad24ca	What did you get? My day is alright.. haven`…

[ ]:
```

```