Q1. REGULAR EXPRESSION

PART A)

Methodology:

We split the tweet into sentences, by using regex.split with delimiters **!?.** and **-.** Further, we remove urls before this step, so as to avoid overcounting in the case of urls being divided into sentences. We tokenize the so obtained sentences into words by using whitespace delimiter ('\s+' to include multiple whitespaces) and choose a so obtained token only if it contains a \w i.e a word(alphanumeric or _).

We count the number of words that start with consonant or vowels by defining a vowel and consonant set, and then scanning through the text with the regex '\b{consonant_set}\w+'

We find usernames by using regex '@\w+' (this is the twitter username format'
We find urls by using regex to find https? or www at the start of the url or the present of a .com in the text. Further the regex explicitly allows only the occurence of special symbols that are allowed in a url which are - `[\w+=?/._\-:;&*^$#@~`!|]`

We find the number of tweets each day by parsing the datetime in the 'DATE_TIME' column in the data to find the day, and finding the number of entries for each unique day.

We find the occurences of a given word and the number of sentences it appears in, by using the regex f'\b{word}\b' and going through the list of tokens and sentences obtained in part 1 respectively. To find the number of sentences starting and ending with the word by replacing the preceding/succeeding \b in the previous regex with '^\s*' or '\s*$' respectively.

Output:

A. Average number of sentences and tokens.

```
Average Number of Sentence in Positive Text: 1.938
Average Number of Sentence in Negative Text: 1.85
Average Number of Tokens in Positive Text 12.70
Average Number of Tokens in Negative Text 13.51
```

B. Total number of words starting with consonants and vowels

```
Number of words starting with consonants in Positive
Text:  19069
```

```
Number of words starting with consonants in Negative
Text:  18400
Number of words starting with vowels in Positive Text:
5669
Number of words starting with vowels in Negative Text:
5445
```

C. Lowercase the text and report the number of unique tokens present before and after lower casing

```
Number of unique tokens before lowercasing in positive
text:  8410
Number of unique tokens before lowercasing in negative
text:  6870
Number of unique tokens after lowercasing in positive
text:  7400
Number of unique tokens after lowercasing in negative
text:  6089
```

D. Count and list all the usernames

```
Number of usernames in positive text: 1305
Number of usernames in negative text: 803
'List of usernames in positive text'
['@awaisnaseer',
 '@Marama',
 '@gfalcone601',
 '@mrstessyman',
 '@GetMeVideo',
 '@tb78',
 '@RealDeal32',
 '@yoginifoodie',
 '@mileycyrus',
 '@SCTunstall',
 '@IHauntWizards',
 '@soycamo',
 '@Liverpool_TX',...]

'List of usernames in negative text'

['@sokendrakouture',
 '@flyingbolt',
 '@digitallearnin',
 '@Luke',
 '@buckhollywood',
 '@alix_says',
 '@mykiaisosm',
 '@Sally_That_Girl',
 '@marginatasnaily',
```

```
 '@NewerDeal',
 '@meggles89',
 '@ferrite',
 '@karon', …]
```

E. Count and list all the urls

```
Number of urls in positive text: 144
Number of urls in negative text: 61

'List of urls in positive text'

['http://blip.fm/~4lfcc',
 'http://bit.ly/rwoHR',
 'http://su.pr/1rXuPY',
 'http://twitpic.com/6b03x',
 'http://tinyurl.com/dk5p94',
 'http://leo.lobato.org/Blipster/',
 'http://bit.ly/nZZQV',
 'http://bit.ly/etD3a',
 'http://dontkillspike.proboards.com/',
 'http://fuzz-ball.com/twitter',...]
'List of urls in negative text'

['http://bit.ly/AEbs3',
 'http://twitpic.com/3l589',
 'http://bit.ly/n4wL4',
 'http://twitpic.com/4ijt4',
 'http://tinyurl.com/ncbmmo',
 'http://twitpic.com/6u8ht',
 'http://bit.ly/47etHn',
 'http://bit.ly/i9lsr',
 'http://twitpic.com/5exx2',
 'http://mypict.me/2dG2',
 'http://twitpic.com/54r0g',
 'http://apps.facebook.com/dogbook/profile/view/6391349',
 'http://ustre.am/2FUW',...]
```

F. Count the number of tweets for each day of the week

```
Number of tweets on each day of the week
Fri:864
Thu:221
Sun:1328
Tue:286
Wed:299
Mon:872
Sat:417
```

PART B)

A. Word: Dog; Label =1
word_count=4
Sentences:

```
["I had to give my dog back to my Mom because he missed
her  but I'm getting a purebred golden retriever puppy!",
 'Smokers suck - even outdoors you still stink worse than
dog shit. Get with the program fuckwads!  Quit!',
 'My Bloody Valentine was great!  Reminded me of watching
it every Saturday on USA while growing up.  Missed the
hot dog scene though ',
 "My dog Died  FUCKING ZUCKY..i'll miss ya Cody :'(
finest malemute ever"]
```

B. Word: If; Label=0

```
['If you use StumbleUpon Please give my site The Thumbs
Up  http://su.pr/1rXuPY',
 'If not for @BingFutch keeping the music going  my
twitter stream would be empty. Where is everybody
tonight?',
 'If EVERYONE constantly retweeted that it would make it!
',
 'If you have a Nokia smartphone, jibjib is by far the
best twitter client for it. Supports twitpic directly
from the camera  #JAFFACAKES',
 'If I told you that you had a great body, would you hold
it against me? -- ']
```

C. Word= me; Label=1

```
['@TerrenceJ106 @KhloeKardashian I can get it online
maybe too many people are listening, which is good for
you &amp;amp; not so much for me',
 "@1045CHUMFM Oh ok,Thanks.Dont know where the courthouse
id either,I'll learn though.Any nkotb? Twitter stole my
pic again  it hates me"]
```

Q2. TEXT PREPROCESSING

Methodology:

We lowercase our text and then find HTML tags by using regex to determine '<...>' or '&.....;' type phrases and remove them. We remove url tags and usernames using the same regex parsers defined in Question 1. We then remove whitespaces using regex. We then expand contractions by using the contractions library. In the next step we remove punctuations using regex to remove all non word and non space characters from the text. We then tokenize the text using nltk's tweettokenizer which is especially useful in tokenizing tweet-type text. Thereafter we lemmatize the text by using nltk's WordNetLemmatizer. We use the helper function pos_tagger along with nltk's pos_tag function to feed the part of speech tag to the lemmatizer.
We then define our custom set of stopwords, in which we use only those stopwords which cannot affect the sentimental meaning of the text. We then use regex to find any of these stopwords and remove them. Finally we use Speller from the autocorrect library, for the purposes of spell checking.

Sentence:

```
"@Luke Oh Luke  , why don't you answer me? pleease try it .
Love your music ! Hear it the whole day, really trying ! </a>
/// http://bit.ly/etD3a ///  "
```

1. **Removing HTML tags (and lowercasing):**

```
 @luke oh luke  , why don't you answer me? pleease try it . love
your music ! hear it the whole day, really trying !  ///
http://bit.ly/etd3a ///
```

2. **Removing URL tags (and usernames):**

```
 oh luke  , why don't you answer me? pleease try it . love your
music ! hear it the whole day, really trying !  ///  ///
```

3. **Removing whitespace (and) expand contractions:**

```
 oh luke , why do not you answer me? pleease try it . love your
music ! hear it the whole day, really trying ! /// ///
```

4. **Removing punctuations:**

```
 oh luke  why do not you answer me pleease try it  love your music
hear it the whole day really trying
```

5. **Tokenize Text:**

['oh', 'luke', 'why', 'do', 'not', 'you', 'answer', 'me', 'pleease', 'try', 'it', 'love', 'your', 'music', 'hear', 'it', 'the', 'whole', 'day', 'really', 'trying']

6. **Lemmatize Text:**
   ['oh', 'luke', 'why', 'do', 'not', 'you', 'answer', 'me', 'pleease', 'try', 'it', 'love', 'your', 'music', 'hear', 'it', 'the', 'whole', 'day', 'really', 'try']
7. **Removing stop words Text:**
   ['oh', 'luke', 'not', 'answer', 'pleease', 'try', 'love', 'music', 'hear', 'whole', 'day', 'really', 'try']

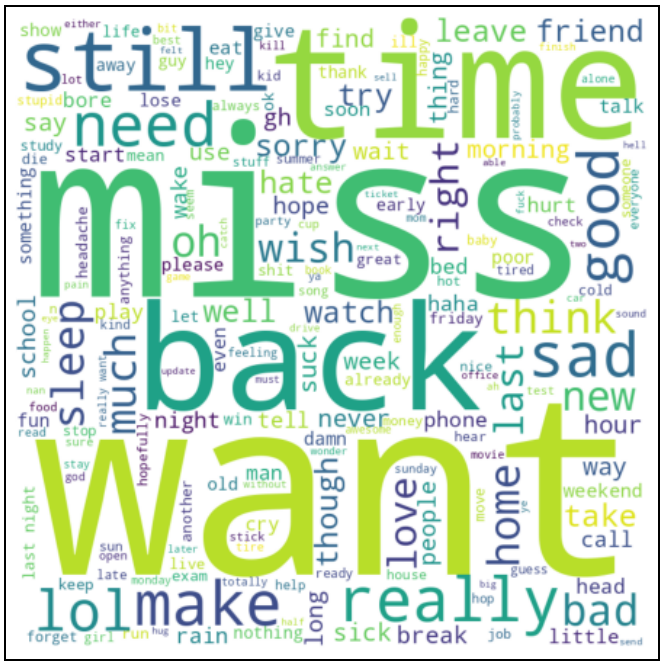
8. **Spelling corrected sentence:**
   oh luke not answer please try love music hear whole day really try

Q3. VISUALIZATION

A. Word Clouds

Positive Class

Negative Class



B. Comparison

The occurrence of positive sentiment words (such as love, thank, lol, haha, great, awesome etc.) is much higher in the positive word cloud.

The occurrence of negative sentiment words (such as miss, sad, bad, sick, bore, awesome etc.) is much higher in the negative word cloud.

However, there are exceptions. Eg- the word good which is a positive sentiment word, appears, in significantly high number in both negative and positive word clouds:
Positive sentence : 'What a good day'
Negative sentence : 'headache and not feeling good again'

Further both word clouds have occurences of words that hold different sentimental meaning in different contexts. Eg- think appears in both positive and negative word clouds as follows:
Positive sentence : 'I think this voice thing is working out'
Negative sentence : 'but they don't think of the consequence'

Q4. RULE-BASED SENTIMENT ANALYSIS

Methodology:

We feed our unprocessed and processed data to vader's SentimentIntensityAnalyser and obtain the polarity scores as a dictionary. We use the 'compound' metric to determine the sentiment class of the tweet (compound score>=0.05 means it is a positive tweet and vice versa). However, through sheer observation, we noticed that a large number of positive tweets in the processed data are assigned a high neutral score (>=0.8) and these tweets often receive a compound score<=0.05. Thus, if the tweet received a neutral score>=0.8, we assigned it the positive class. This is defined in the helper function *vader_class***.** Finally we define a helper function *accuracy* which inputs the predicted labels from the vader_class function and the true labels from the data, and outputs the percentage of correct predictions.

Results:

```
Accuracy of vader on unprocessed data : 65.68%
Accuracy of vader on processed data : 67.57%
```