(https://colab.research.google.com/github/unt-iialab/INFO5731_Spring2020/blob/master/Assignments/INFO5731_Assignment_Two.ipynb)

INFO5731 Assignment Two

In this assignment, you will try to gather text data from open data source via web scraping or API. After that you need to clean the text data and syntactic analysis of the data.

Question 1

(40 points). Write a python program to collect text data from **either of the following sources** and save the data into a **csv file**:

(1) Collect all the customer reviews of the product Apple iPhone 11 (https://www.amazon.com/Apple-iPhone-11-64GB-Unlocked/dp/B07ZPKF8RG/ref=sr_1_13?dchild=1&keywords=iphone+12&qid=1631721363&sr=8-13) on amazon.

(2) Collect the top 10000 User Reviews of the film Shang-Chi and the Legend of the Ten Rings (https://www.imdb.com/title/tt9376612/reviews?ref_=tt_sa_3) from IMDB.

(3) Collect all the reviews of the top 100 most popular software from G2 (https://www.g2.com/) or Capterra (https://www.capterra.com/)

(4) Collect the abstracts of the top 10000 research papers by using the query natural language processing (https://citeseerx.ist.psu.edu/search?q=natural+language+processing&submit.x=0&submit.y=0&sort=rlv&t=doc) from CiteSeerX.

(5) Collect all the information of the 904 narrators in the Densho Digital Repository (https://ddr.densho.org/narrators/).

(6) Collect the top 10000 tweets by using hashtag "#blacklivesmatter" (https://twitter.com/hashtag/blacklivesmatter) from Twitter.

In [25]:
```
# !pip install tweepy
```

In [1]:
```
import pandas as pd
import tweepy
```

In [11]:
```
tweets_list = []

# Write your code here


api_key = "c3fWWuuvMyUYOkWfHaad1A4mn"
api_key_secret = "RpAwtZ4IuEUcNyXxbXRDjfegzAtUUdFpX18acdAYDRMNfh5pfJ"

acess_token = "2824470606-tfb1oZ4GZDQRH4qjt4MuT1MI5FJlxsvIPpDZdR3"
access_token_secret = "Lhp2FXgoYGniidXq7nP9sGkcJhOCL71oPzY5GhLed8qlh"



auth = tweepy.OAuthHandler(api_key, api_key_secret)
auth.set_access_token(acess_token, access_token_secret)

api = tweepy.API(auth,wait_on_rate_limit=False)

hash_tag = "#blacklivesmatter"
tweet_items = tweepy.Cursor(api.search_tweets,
                            q = hash_tag + "-filter:retweets",
                            count = 10000,
                            lang="en",
                            since_id=0).items()

for tweet in tweet_items:
    text_tweet = tweet.text.encode('utf-8')
    tweets_list.append(str(text_tweet,'utf-8'))

    if len(tweets_list)>=10000:
        break
```

In [15]:
```
tweets_df = pd.DataFrame(tweets_list[:10000],columns=['tweet_text'])
```

In [16]:
```
tweets_df.to_csv("tweets_data.csv",index = False)
```

In [2]:
```
tweets_df = pd.read_csv("tweets_data.csv")
```

Question 2

(30 points). Write a python program to **clean the text data** you collected above and save the data in a new column in the csv file. The data cleaning steps include:

(1) Remove noise, such as special characters and punctuations.

(2) Remove numbers.

(3) Remove stopwords by using the stopwords list (https://gist.github.com/sebleier/554280).

(4) Lowercase all texts

(5) Stemming.

(6) Lemmatization.

In [3]:
```
#Remove noise, such as special characters and punctuations.

import re
def remove_noise(tweet):
    return re.sub('[^0-9a-zA-Z]+ ', ' ', tweet)
```

In [4]:
```
tweets_df["cleaned_text"] = tweets_df.tweet_text.apply(lambda x: remov
```

In [5]:
```
tweets_df["cleaned_text"] = tweets_df.cleaned_text.apply(lambda x: re.
```

In [6]:
```
stop_words_list = ['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 'your',
 'yours',
 'yourself',
```

```
'yourselves',
'he',
'him',
'his',
'himself',
'she',
'her',
'hers',
'herself',
'it',
'its',
'itself',
'they',
'them',
'their',
'theirs',
'themselves',
'what',
'which',
'who',
'whom',
'this',
'that',
'these',
'those',
'am',
'is',
'are',
'was',
'were',
'be',
'been',
'being',
'have',
'has',
'had',
'having',
'do',
'does',
'did',
'doing',
'a',
'an',
'the',
'and',
'but',
'if',
'or',
'because',
'as',
```

```
    'until',
    'while',
    'of',
    'at',
    'by',
    'for',
    'with',
    'about',
    'against',
    'between',
    'into',
    'through',
    'during',
    'before',
    'after',
    'above',
    'below',
    'to',
    'from',
    'up',
    'down',
    'in',
    'out',
    'on',
    'off',
    'over',
    'under',
    'again',
    'further',
    'then',
    'once',
    'here',
    'there',
    'when',
    'where',
    'why',
    'how',
    'all',
    'any',
    'both',
    'each',
    'few',
    'more',
    'most',
    'other',
    'some',
    'such',
    'no',
    'nor',
    'not',
    'only'
```

```
 'only',
 'own',
 'same',
 'so',
 'than',
 'too',
 'very',
 's',
 't',
 'can',
 'will',
 'just',
 'don',
 'should',
 'now']
```

In [7]:
```
def remove_stop_words(tweet):
    result_string = ""
    for each_word in tweet.split(" "):
        if each_word in stop_words_list:
            pass
        else:
            result_string = result_string + each_word + " "

    return result_string
```

In [8]:
```
tweets_df.cleaned_text = tweets_df.cleaned_text.apply(lambda x: remove
```

In [9]:
```
tweets_df.cleaned_text = tweets_df.cleaned_text.apply(lambda x: x.lowe
```

In [10]:
```
import nltk
nltk.download("wordnet")
nltk.download('punkt')
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/porteruser/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/porteruser/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [11]:
```
my_stemmer =PorterStemmer()
def stemmer(tweet):
    root_word_list = []
    for each_word in tweet.split(" "):
        root_word = my_stemmer.stem(each_word)
        root_word_list.append(root_word)
    return " ".join(root_word_list)
```

In [12]:
```
tweets_df["cleaned_text"] = tweets_df.cleaned_text.apply(lambda x: ste
```

In [13]:
```
my_wordnet_lemmatizer = WordNetLemmatizer()

def lemmatizer(tweet):
    lemmatized_word_list = []
    for each_word in tweet.split(" "):
        lemmatized_word = my_wordnet_lemmatizer.lemmatize(each_word)
        lemmatized_word_list.append(lemmatized_word)
    return " ".join(lemmatized_word_list)
```

In [14]:
```
tweets_df["cleaned_text"] = tweets_df.cleaned_text.apply(lambda x: lem
```

Question 3

(30 points). Write a python program to conduct **syntax and structure analysis** of the clean text you just saved above. The syntax and structure analysis includes:

(1) Parts of Speech (POS) Tagging: Tag Parts of Speech of each word in the text, and calculate the total number of N(oun), V(erb), Adj(ective), Adv(erb), respectively.

(2) Constituency Parsing and Dependency Parsing: print out the constituency parsing trees and dependency parsing trees of all the sentences. Using one sentence as an example to explain your understanding about the constituency parsing tree and dependency parsing tree.

(3) Named Entity Recognition: Extract all the entities such as person names, organizations, locations, product names, and date from the clean texts, calculate the count of each entity.

In [15]:
```
from collections import Counter
```

In [16]:
```
text = tweets_df.iloc[1][0]
```

In [17]:
```
noun_tags = ['NN','NNS','NNP','NNPS']
verb_tags = ["VB","VBD","VBG","VBN","VBP","VBZ" ]
adverb_tags = ['RB','RBR','RBS']
adjective_tags = ['JJ','JJR','JJS']
```

In [18]:
```
def pos_counts(tweet):
    tokens = nltk.word_tokenize(tweet.lower())
    text = nltk.Text(tokens)
    tags = nltk.pos_tag(text)
    counts = Counter(tag for word,tag in tags)
    counts = dict(counts)

    n_nouns = 0
    for each_noun_tag in noun_tags:
        if each_noun_tag in counts:
            n_nouns = n_nouns + counts[each_noun_tag]
    n_verbs = 0
    for each_verb_tag in verb_tags:
        if each_verb_tag in counts:
            n_verbs = n_verbs + counts[each_verb_tag]

    n_adverbs = 0
    for each_adverb_tag in adverb_tags:
        if each_adverb_tag in counts:
            n_adverbs = n_adverbs + counts[each_adverb_tag]

    n_adjectives = 0
    for each_adjective_tag in adjective_tags:
        if each_adjective_tag in counts:
            n_adjectives = n_adjectives + counts[each_adjective_tag]

    return_s = pd.Series([n_nouns, n_verbs, n_adjectives, n_adverbs])
    return pd.Series(return_s).rename({0:'n_nouns',1:'n_verbs',2:"n_ad
```

In [19]:
```
counts_df = tweets_df.cleaned_text.apply(lambda x: pos_counts(x) )
```

In [20]:
```
tweets_df = pd.concat([tweets_df, counts_df], axis=1)
```

In [21]:
```
#Constituency Parsing and Dependency Parsing
```

In [27]:
```
from spacy import displacy
import spacy
nlp=spacy.load('en_core_web_sm')
```

In [45]:
```
def get_cp_dp(tweet):
    for each_word in nlp(tweet):
        print("Dependency Tag: ", each_word.text,"Head: ",each_word.he
```

In [46]:
```
tweets_df.cleaned_text.apply(lambda x: get_cp_dp(x))
```

```
Dependency Tag:  @varieti Head:  inf Dependency:  nsubj
Dependency Tag:  @potu Head:  countri Dependency:  compound
Dependency Tag:  @vp Head:  countri Dependency:  nmod
Dependency Tag:  @speakerpelosi Head:  countri Dependency:  nmod
Dependency Tag:  @theblackcaucu Head:  countri Dependency:  nmod
Dependency Tag:  @thejusticedept Head:  countri Dependency:  nmod
Dependency Tag:  @naacp Head:  repair Dependency:  compound
Dependency Tag:  repair Head:  @thejusticedept Dependency:  dative
Dependency Tag:  damag Head:  countri Dependency:  compound
Dependency Tag:  countri Head:  @varieti Dependency:  appos
Dependency Tag:  inf Head:  inf Dependency:  ROOT
Dependency Tag:  https://t.co/pczdwkhkil (https://t.co/pczdwkhkil)
Head:  inf Dependency:  npadvmod
Dependency Tag:  sen Head:  tubervil Dependency:  compound
Dependency Tag:  tommi Head:  tubervil Dependency:  compound
Dependency Tag:  tubervil Head:  suggest Dependency:  nsubj
Dependency Tag:  suggest Head:  suggest Dependency:  ROOT
Dependency Tag:  all Head:  peopl Dependency:  det
Dependency Tag:  black Head:  peopl Dependency:  amod
```

In [60]:
```
from spacy import displacy
displacy.render(nlp(tweets_df.cleaned_text.iloc[10]),jupyter=True)
```

black PROPN america PROPN never ADV need VERB organ NOUN slogan NOUN inform PROPN u PRON black ADJ live NOUN matter NOUN a DET fellow NOUN human NOUN https://t.co/gsqktycn ADV compound nsubj neg compound compound dobj dative amod advmod ccomp det amod dobj punct

Write your explanations of the constituency parsing tree and dependency parsing tree here (Question 3-2):

A sample explanation of the above image is given below

The word 'black' is used for compounding the word 'america', which will make the one dependent on the latter. The two words compound together and gives more precise sentence, i.e it helps in understanding whom exactly we are referring in our example.

The word 'America' is a subject word for the verb, need thus denoted by nsubj which stands for nominal subject

The word 'Never' negates the sentence, thus labelled as neg