```
!pip install -U spacy
```

```
!python -m spacy download en_core_web_lg
```

```
# Main libraries
import pandas as pd
import numpy as np

# Preprocessing
#!pip install gensim # i think it is already there in colab
import gensim
from gensim.utils import simple_preprocess
import re

import matplotlib.pyplot as plt
%matplotlib inline

#to increase the space of displaying texts
pd.options.display.max_colwidth = 200
```

Load extracted submissions and comments

```
# use this as many Comments files you have, just change the dataframe number and file name
cdf= pd.read_csv('comments_ve2y.csv', encoding = "utf-8", on_bad_lines='skip')
cdf1= pd.read_csv('comments_ca1y.csv', encoding = "utf-8", on_bad_lines='skip')
```

```
cdf_ve = cdf[cdf['created_utc']> 1606780799].copy() # selecting comments after 1/12/2020 [a
print(cdf_ve.shape)
```

```
# Combine all Comments dataframes in one dataframe
cdf_master = cdf_ve.append([cdf1], ignore_index= True)  #ignore_index= True
```

```
# use this as many Submissions files you have, just change the dataframe number and file na
```

```
sdf1= pd.read_csv('submissions_ca1y.csv', encoding = "utf-8", on_bad_lines='skip');


# Combine all Submissions dataframes in one dataframe
sdf_master = sdf.append([sdf1], ignore_index = True)


#check number of submissions and comments
print('Total number of submissions' + " " + str(len(sdf_master)))
print('Total number of comments' + " " + str(len(cdf_master)))
```

Deal with removed/deleted submissions and comments

```
#noticed that some "removed" submissions have some text beside the word "removed", so this
sdf_master['selftext'] = sdf_master['selftext'].replace(r"\[removed\].*", value='[removed]'


#Count the number of deleted or removed submissions/comments from the corpus
print(len(sdf_master[(sdf_master['selftext'] == '[removed]') | (sdf_master['selftext'] ==
print(len(cdf_master[(cdf_master['body'] == '[removed]') | (cdf_master['body'] == '[deleted


#Count the number of deleted or removed submissions which has no comments from the corpus
print(len((sdf_master[(((sdf_master['selftext'] == '[removed]') | (sdf_master['selftext'] =

#Count the number of deleted or removed comments from the corpus
print(len(cdf_master[(cdf_master['body'] == '[removed]') | (cdf_master['body'] == '[deleted


# Remove deleted and removed comments/submissions
sdf_master_clean = sdf_master.loc[(((sdf_master['selftext'] == '[removed]') | (sdf_master[
print('Total number of submissions = ', len(sdf_master_clean))

cdf_master_clean = cdf_master.loc[(cdf_master['body'] != '[removed]') & (cdf_master['body']
print('Total number of comments = ', len(cdf_master_clean))
#for .copy, check https://www.dataquest.io/blog/settingwithcopywarning/ (I used .loc, but i



sdf_master_clean["selftext"].replace(r'\[removed\]', " ", regex=True, inplace=True)
sdf_master_clean["selftext"].replace(r'\[deleted\].*', " ", regex=True, inplace=True)
#second answer : https://stackoverflow.com/questions/37593550/replace-method-not-working-or
```

Append Submissions and Comments

```
# Note that both submission's title and text are considered becuase there are a lot of subm
sdf_master_clean['combined_text'] = sdf_master_clean["title"].astype(str) + ". " + sdf_mast
```

```
#Extract the text of comments/submissions and combine them, calling them posts this point f
posts_texts = sdf_master_clean['combined_text'].append(cdf_master_clean['body'])
print('Total number of posts = ', len(posts_texts))
```

```
#to handle emoji encoding, as vader recognizes emojies
posts_texts = posts_texts.str.encode('latin', errors= 'ignore').str.decode('utf-8', errors=
```

Documents' test Preprocessing

```
#'nan' and newline charachter removal

posts_texts = posts_texts.replace(r'\.+', ".", regex=True) # replace multiple dots with sir
posts_texts = posts_texts.replace(r'\\n',' ', regex=True) # several newlines charachers wer
posts_texts = posts_texts.replace(r'\b([Nn][Aa][Nn])\b',' ', regex=True) # 'nan' happended
posts_texts = posts_texts.replace(r'([Hh][Aa]){2,}',' ', regex=True) #  remove hahaha relat
posts_texts = posts_texts.replace(r'([lL][Oo][lL]){2,}',' ', regex=True) # remove lollol re
posts_texts = posts_texts.replace(r'(ID4|ID.4)','VWIDIV', regex=True) # to handle Volkswage
posts_texts = posts_texts.replace(r'(ID3|ID.3)','VWIDIII', regex=True) # to handle Volkswag
posts_texts = posts_texts.replace(r'(ID2|ID.2)','VWIDII', regex=True) # to handle Volkswage
posts_texts = posts_texts.replace(r'(ID6|ID.6)','VWIDVI', regex=True) # to handle Volkswage
```

```
#check if we have empty documents
print(len(posts_texts[posts_texts == ''])) # answer should be 0 if no empty documents
posts_texts[14:16]
```

```
#removing links
removed_links = posts_texts.replace(r"[\[]?[hH][tT][tT][Pp]\S+", value='', regex=True).copy
#note: added [] to the regex, because there was some cells with only the URL put inside [],
```

```
print(len(removed_links[removed_links == '']))
```

```
removed_links.replace('', np.nan, inplace=True)
```

```
removed_links.shape[0] - removed_links.dropna().shape[0]
```

```
removed_links.dropna(inplace=True)
```

```
print(len(removed_links[removed_links == '']))
```

```
print(len(removed_links[removed_links ==  ]))


# Reseting index
removed_links.reset_index(drop= True, inplace=True)
```

Calculate word2vec

```
#import selected aspects
topics_words_df= pd.read_csv('aspects17.csv', on_bad_lines='skip')
topics_words_df.columns =('TopicNo', 'TopicName', 'TopicCategory', 'TopicKeyWords')
topics_words_df.head()


nlp2 = spacy.load('en_core_web_lg')
#Define function to get word2vec for a sentence
def get_vec(x):
  doc = nlp2(x)
  vec = doc.vector
  return vec


#calculate vec for aspects keywords
topics_words_df['key_words_vec'] = topics_words_df['TopicKeyWords'].apply(lambda x: get_vec


#Prepare posts for word2vec-similarity calculation

import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize


#tokenize sentences
posts_sent = removed_links.apply(lambda x : sent_tokenize(x)).apply(pd.Series,1).stack()
Post_sent_df = pd.DataFrame(posts_sent)#change serise to dataframe
Post_sent_df.columns = ['Sent_text']
Post_sent_df.shape
Post_sent_df.head()


#remove deaccent
from gensim.utils import deaccent
def clean_text (text):
    text = deaccent(text)
    return text

Post_sent_df['Sent_text'].apply(lambda x : clean_text(x))


#calculate vec for sentences
Post_sent_df['Sent_vec'] = Post_sent_df['Sent_text'].apply(lambda x: get_vec(x))
```

```
#calculate similarity values
for index, row in Post_sent_df.iterrows():
    row = Post_sent_df['Sent_vec'][index]
    for index_tp, row_tp in topics_words_df.iterrows():
        row_tp = topics_words_df['key_words_vec'][index_tp]
        cosine_similarity = np.dot(row, row_tp)/(np.linalg.norm(row)* np.linalg.norm(row_tp
    #    new_col.append(sim_value)
        col_name = str(index_tp)
        Post_sent_df.at[index , index_tp] = cosine_similarity


#delete sentences vector column as it causes large size/ memory crash in further processing
Post_sent_df.drop(['Sent_vec'], axis = 1, inplace = True)


#to check columns
Post_sent_df.iloc[:, 1]


#Find the most similar aspect (the aspect associated with each sentence )
Post_sent_df['Topic'] = Post_sent_df.values[:,1:18].argmax(1) # as similarity columns start
#https://stackoverflow.com/questions/43330555/pandas-python-max-in-columns-define-new-value

#second tried case, #to ignore if similarity is less tha 0.5
Post_sent_df['Topic'] = (
   Post_sent_df.iloc[:, 1:18].mask(~Post_sent_df.iloc[:, 1:18].ge(0.65).any(1), other=pd.NA
           .idxmax(1).fillna('none'))



len(Post_sent_df['Topic'].value_counts()) # check value
```

Sentiment Analysis , Thanks to https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()



#sentiment calculation
Post_sent_df['scores'] = Post_sent_df['Sent_text'].apply(lambda T: sid.polarity_scores(T))
Post_sent_df['compoundSco'] = Post_sent_df['scores'].apply(lambda score_dict: score_dict['c
Post_sent_df['comp_score'] = Post_sent_df['compoundSco'].apply(lambda c: 'pos' if c >0 else
```

```
Post_sent_df['comp_score'].value_counts()
```

```
Post_sent_df['comp_score'].value_counts().plot(kind = 'bar', title = 'Count of sentences pe
```

```
# to sum sentiment per dominant topic
Post_sent_df_grp_sen = Post_sent_df.groupby('Topic', as_index=False)['compoundSco'].mean()
```

```
Post_sent_df_grp_sen.plot(x='Topic', y='compoundSco', kind = 'bar')
```