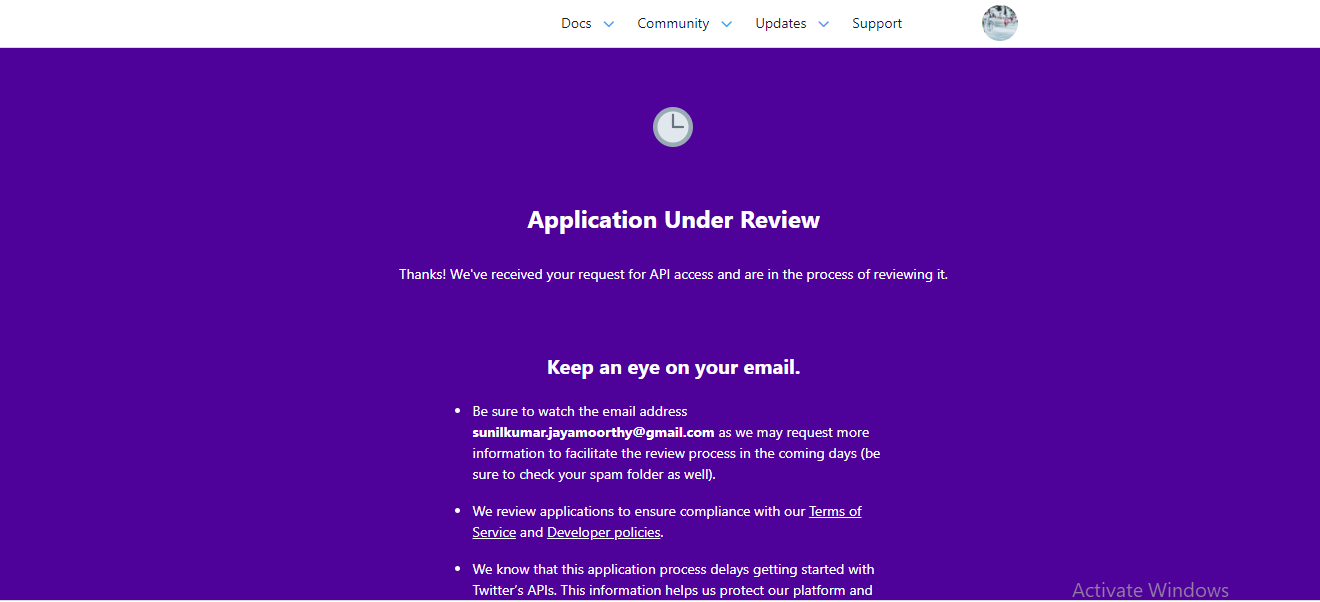
For Text Mining assignment

**ONE:**

**1) Extract tweets for any user (try choosing a user who has more tweets)**

**2) Perform sentimental analysis on the tweets extracted from the above**

Still application is under review which I applied before 1 month.



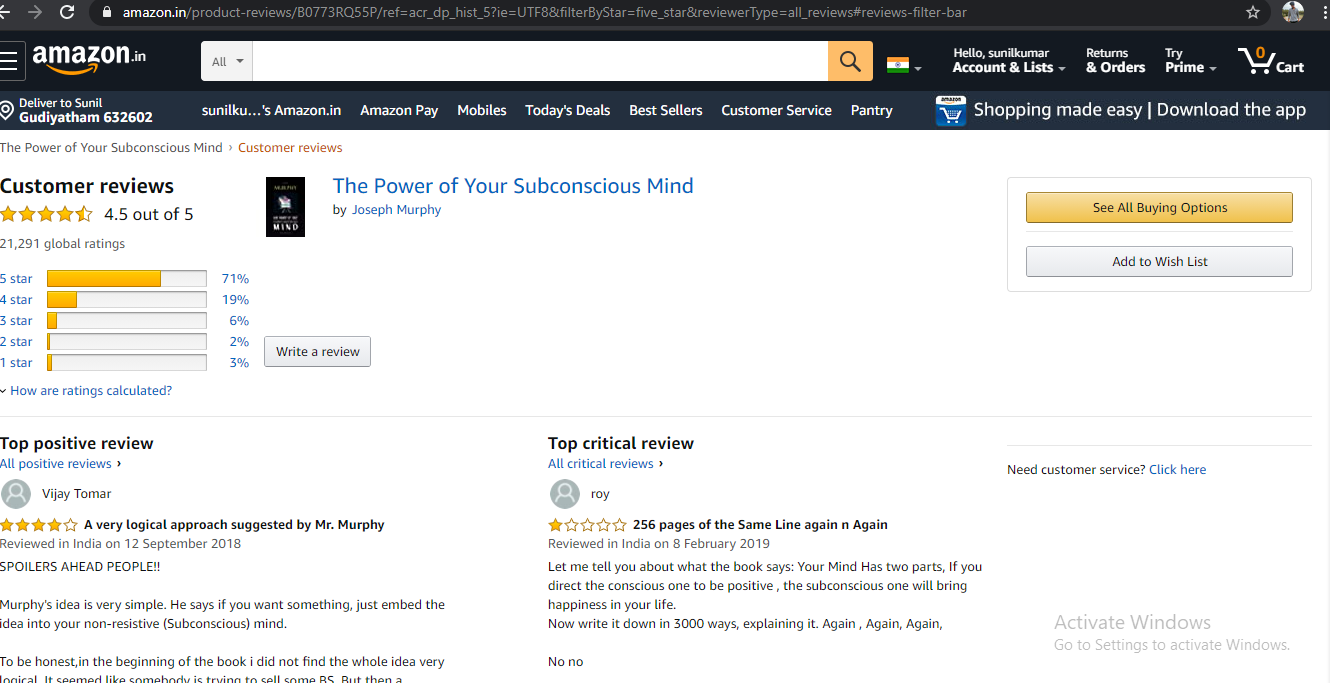
**TWO:**

**1) Extract reviews of any product from ecommerce website like amazon**

**2) Perform emotion mining**

Performing the amazon review for the book

Link: https://www.amazon.in/product-reviews/B0773RQ55P/ref=acr\_dp\_hist\_5?ie=UTF8&filterByStar=five\_star&reviewerType=all\_reviews#reviews-filter-bar



Code:

# -\*- coding: utf-8 -\*-

"""

Created on Mon Nov 23 09:43:53 2020

@author: sunil

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import re

import time

from datetime import datetime

import matplotlib.dates as mdates

import matplotlib.ticker as ticker

from urllib.request import urlopen

from bs4 import BeautifulSoup as bs

import requests

# In[2]:

wm\_title=[]

wm\_date = []

wm\_content = []

wm\_rating = []

for i in range(1,150):

link = "https://www.amazon.in/product-reviews/B0773RQ55P/ref=acr\_dp\_hist\_5?ie=UTF8&filterByStar=five\_star&reviewerType=all\_reviews#reviews-filter-bar"+str(i)

response = requests.get(link)

soup = bs(response.content,"html.parser")# creating soup object to iterate over the extracted content

# extracting Review Title

title = soup.find\_all('a',class\_='review-title-content')

review\_title = []

for i in range(0,len(title)):

review\_title.append(title[i].get\_text())

review\_title[:] = [titles.lstrip('\n') for titles in review\_title]

review\_title[:] = [titles.rstrip('\n') for titles in review\_title]

wm\_title = wm\_title + review\_title

## Extracting Ratings

rating = soup.find\_all('i',class\_='review-rating')

review\_rating = []

for i in range(2,len(rating)):

review\_rating.append(rating[i].get\_text())

#review\_rating.pop(0)

#review\_rating.pop(0)

review\_rating[:] = [reviews.rstrip(' out of 5 stars') for reviews in review\_rating]

wm\_rating = wm\_rating + review\_rating

#Extracting Content of review

review = soup.find\_all("span",{"data-hook":"review-body"})

review\_content = []

for i in range(0,len(review)):

review\_content.append(review[i].get\_text())

review\_content[:] = [reviews.lstrip('\n') for reviews in review\_content]

review\_content[:] = [reviews.rstrip('\n') for reviews in review\_content]

wm\_content = wm\_content + review\_content

#Extracting dates of reviews

dates = soup.find\_all('span',class\_='review-date')

review\_dates = []

for i in range(2,len(rating)):

review\_dates.append(dates[i].get\_text())

review\_dates[:] = [reviews.lstrip('Reviewed in India on') for reviews in review\_dates]

#review\_dates.pop(0)

#review\_dates.pop(0)

wm\_date = wm\_date + review\_dates

# In[3]:

print(len(wm\_title))

print(len(wm\_rating))

print(len(wm\_content))

print(len(wm\_date))

# In[4]:

df = pd.DataFrame()

df['Title'] = wm\_title

df['Ratings'] = wm\_rating

df['Comments'] = wm\_content

df['Date'] = wm\_date

df.head(5)

# In[5]:

df.head(2)

# In[6]:

df['Date'] = pd.to\_datetime(df['Date'])

df['Ratings'] = df['Ratings'].astype(float)

df.head(2)

# ## Text Cleaning

#

# 1. lower the text

# 2. tokenize the text (split the text into words) and remove the punctuation

# 3. remove useless words that contain numbers

# 4. remove useless stop words like ‘the’, ‘a’ ,’this’ etc.

# 5. Part-Of-Speech (POS) tagging: assign a tag to every word to define 6. if it corresponds to a noun, a verb etc. using the WordNet lexical database

# 7. lemmatize the text: transform every word into their root form (e.g. rooms -> room, slept -> sleep)

# In[ ]:

df.head(2)

# In[7]:

from nltk.corpus import wordnet

def get\_wordnet\_pos(pos\_tag):

if pos\_tag.startswith('J'):

return wordnet.ADJ

elif pos\_tag.startswith('V'):

return wordnet.VERB

elif pos\_tag.startswith('N'):

return wordnet.NOUN

elif pos\_tag.startswith('R'):

return wordnet.ADV

else:

return wordnet.NOUN

import string

import nltk

from nltk import pos\_tag

from nltk.corpus import stopwords

from nltk.tokenize import WhitespaceTokenizer

from nltk.stem import WordNetLemmatizer

nltk.download('stopwords')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('wordnet')

nltk.download('vader\_lexicon')

# In[8]:

def clean\_text(text):

# lower text

text = text.lower()

# tokenize text and remove puncutation

text = [word.strip(string.punctuation) for word in text.split(" ")]

# remove words that contain numbers

text = [word for word in text if not any(c.isdigit() for c in word)]

# remove stop words

stop = stopwords.words('english')

text = [x for x in text if x not in stop]

# remove empty tokens

text = [t for t in text if len(t) > 0]

# pos tag text

pos\_tags = pos\_tag(text)

# lemmatize text

text = [WordNetLemmatizer().lemmatize(t[0], get\_wordnet\_pos(t[1])) for t in pos\_tags]

# remove words with only one letter

text = [t for t in text if len(t) > 1]

# join all

text = " ".join(text)

return(text)

# In[9]:

# clean text data

df["Comments"] = df["Comments"].apply(lambda x: clean\_text(x))

# In[10]:

df['Title'] = df['Title'].astype(str)

df['Title'] = df['Title'].apply(lambda x: clean\_text(x))

# In[14]:

df.head(5)

# ## Feature Engineering

# In[23]:

# add sentiment anaylsis columns

from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

df["sentiments"] = df["Comments"].apply(lambda x: sid.polarity\_scores(x))

df = pd.concat([df.drop(['sentiments'], axis=1), df['sentiments'].apply(pd.Series)], axis=1)

'''

sid = SentimentIntensityAnalyzer()

df["sentiments\_title"] = df["Title"].apply(lambda x: sid.polarity\_scores(x))

df = pd.concat([df.drop(['sentiments\_title'], axis=1), df['sentiments\_title'].apply(pd.Series)], axis=1)

'''

df

# In[12]:

# add number of characters column

df["nb\_chars"] = df["Comments"].apply(lambda x: len(x))

# add number of words column

df["nb\_words"] = df["Comments"].apply(lambda x: len(x.split(" ")))

''''

# add number of characters column

df["nb\_chars\_title"] = df["Title"].apply(lambda x: len(x))

# add number of words column

df["nb\_words\_title"] = df["Title"].apply(lambda x: len(x.split(" ")))

'''

# In[13]:

# create doc2vec vector columns

from gensim.test.utils import common\_texts

from gensim.models.doc2vec import Doc2Vec, TaggedDocument

documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(df["Comments"].apply(lambda x: x.split(" ")))]

# train a Doc2Vec model with our text data

model = Doc2Vec(documents, vector\_size=5, window=2, min\_count=1, workers=4)

# transform each Comment into a vector data

doc2vec\_df = df["Comments"].apply(lambda x: model.infer\_vector(x.split(" "))).apply(pd.Series)

doc2vec\_df.columns = ["doc2vec\_vector\_" + str(x) for x in doc2vec\_df.columns]

df = pd.concat([df, doc2vec\_df], axis=1)

'''

# transform each Title into a vector data

doc2vec\_df\_title = df["Title"].apply(lambda x: model.infer\_vector(x.split(" "))).apply(pd.Series)

doc2vec\_df\_title.columns = ["doc2vec\_vector\_" + str(x) for x in doc2vec\_df.columns]

df = pd.concat([df, doc2vec\_df\_title], axis=1)

'''

# In[14]:

# add tf-idfs columns

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(min\_df = 10)

tfidf\_result = tfidf.fit\_transform(df["Comments"]).toarray()

tfidf\_df = pd.DataFrame(tfidf\_result, columns = tfidf.get\_feature\_names())

tfidf\_df.columns = ["word\_" + str(x) for x in tfidf\_df.columns]

tfidf\_df.index = df.index

df = pd.concat([df, tfidf\_df], axis=1)

'''

##TF-IDF for Titles

tfidf = TfidfVectorizer(min\_df = 10)

tfidf\_result = tfidf.fit\_transform(df["Title"]).toarray()

tfidf\_df\_title = pd.DataFrame(tfidf\_result, columns = tfidf.get\_feature\_names())

tfidf\_df\_title.columns = ["word\_" + str(x) for x in tfidf\_df\_title.columns]

tfidf\_df\_title.index = df.index

df = pd.concat([df, tfidf\_df\_title], axis=1)

'''

# In[15]:

from wordcloud import WordCloud

import matplotlib.pyplot as plt

def show\_wordcloud(data, title = None):

wordcloud = WordCloud(

background\_color = 'white',

max\_words = 200,

max\_font\_size = 40,

scale = 3,

random\_state = 42

).generate(str(data))

fig = plt.figure(1, figsize = (20, 20))

plt.axis('off')

if title:

fig.suptitle(title, fontsize = 20)

fig.subplots\_adjust(top = 2.3)

plt.imshow(wordcloud)

plt.show()

# print wordcloud

show\_wordcloud(df["Comments"])

# In[16]:

# print wordcloud

show\_wordcloud(df["Title"])

# In[21]:

df.head(2)

# In[17]:

# highest positive sentiment reviews (with more than 5 words)

df[df["nb\_words"] >= 5].sort\_values("pos", ascending = False)[["Comments", "pos"]].head(10)

# In[18]:

# lowest negative sentiment reviews (with more than 5 words)

df[df["nb\_words"] >= 5].sort\_values("neg", ascending = False)[["Comments", "neg"]].head(10)

# In[19]:

df['Month'] = df['Date'].dt.month

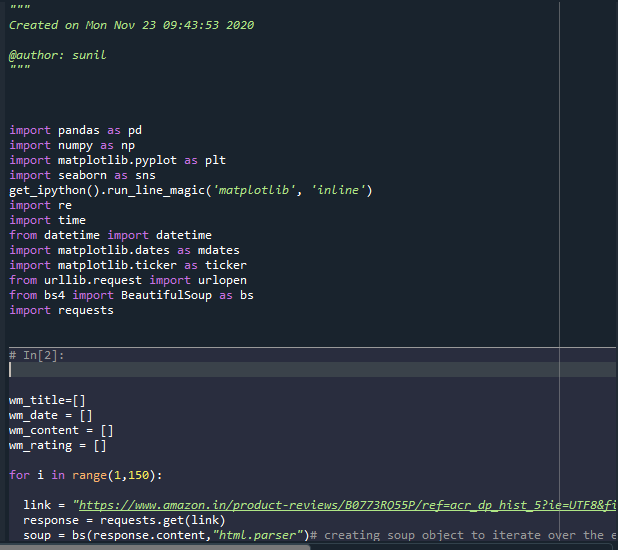
df['Year'] = df['Date'].dt.year

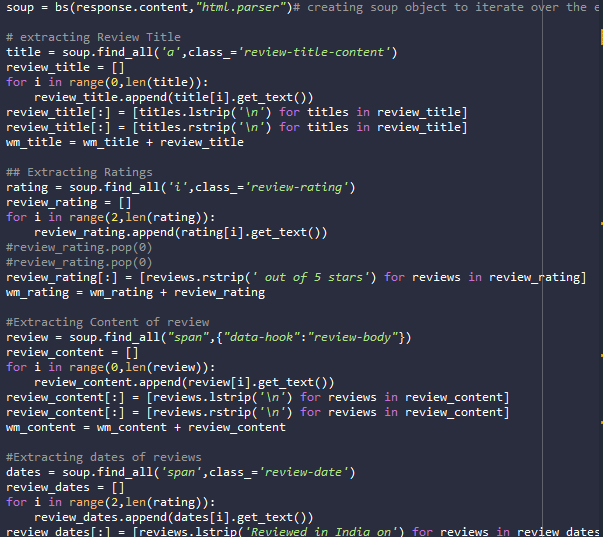
df.head(2)

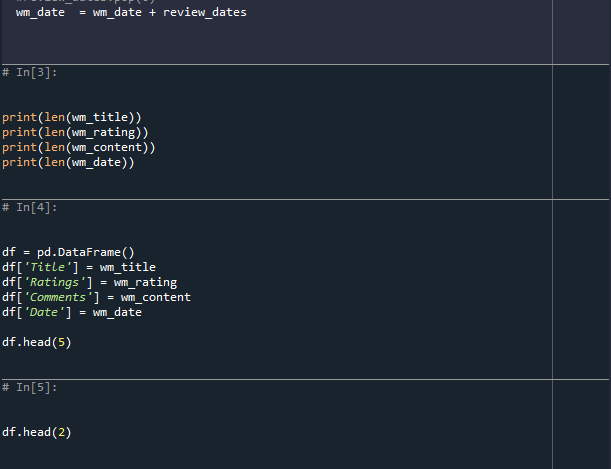
# In[20]:

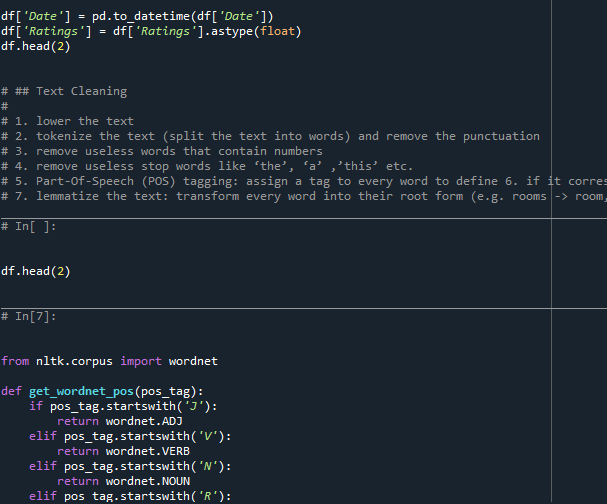
df\_recent = df[(df['Year']== 2020) & (df['Month'] != 8)]

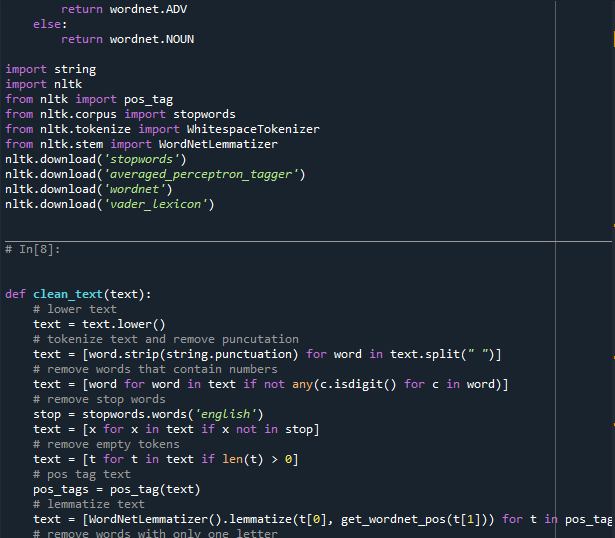
df\_recent.head(2)

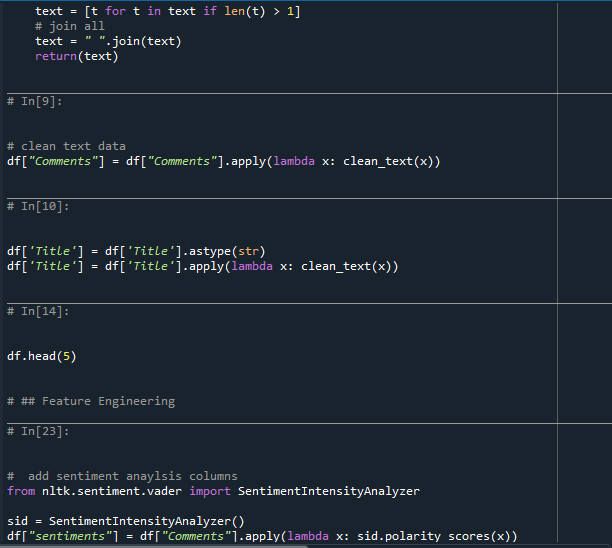


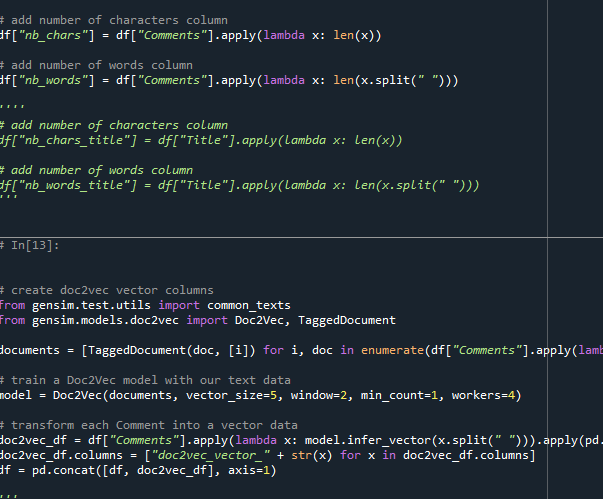


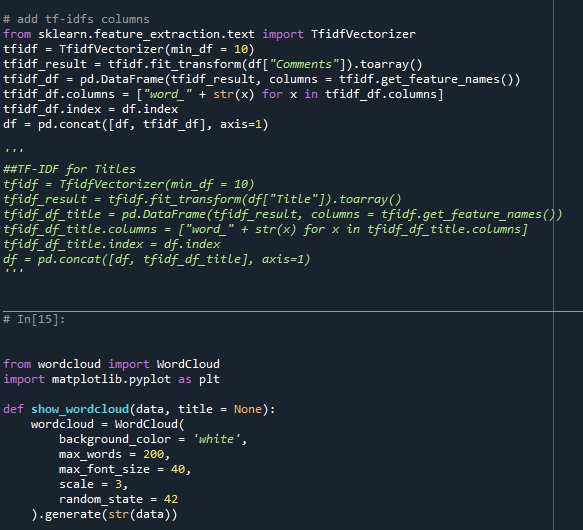


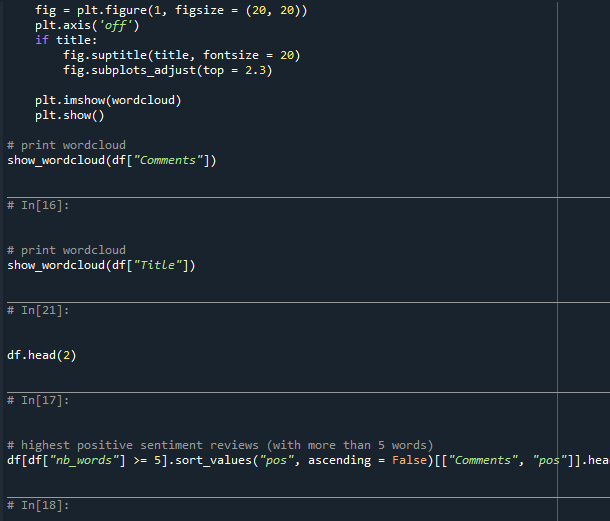


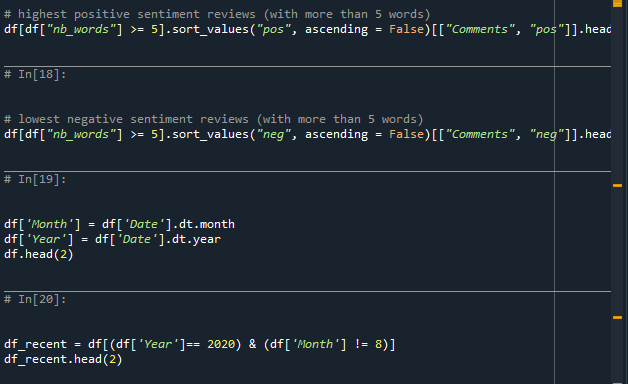












Output:

