```
#Document Clustering and Topic Modeling


!pip install -U -q PyDrive#identity varification

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_applicati
drive = GoogleDrive(gauth)
```

```
ERROR: Invalid requirement: 'PyDrive#identity'
```

```
file = drive.CreateFile({'id':'192JMR7SIqoa14vrs7Z9
file.GetContentFile('data.tsv')


import numpy as np
import pandas as pd
import nltk
# import gensim

from sklearn.feature_extraction.text import TfidfVe
import matplotlib.pyplot as plt

nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /r
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
df = pd.read_csv('data.tsv',sep = '\t',error_bad_li
```

```
<ipython-input-8-f636447d6c36>:1: FutureWarning


  df = pd.read_csv('data.tsv',sep = '\t',error_
Skipping line 8704: expected 15 fields, saw 22
Skipping line 16933: expected 15 fields, saw 22
Skipping line 23726: expected 15 fields, saw 22

Skipping line 85637: expected 15 fields, saw 22
```

watch_reviews.tsv

文件过大，无法显示。

下载

```
Skipping line 132136: expected 15 fields, saw 2
Skipping line 158070: expected 15 fields, saw 2
Skipping line 166007: expected 15 fields, saw 2
Skipping line 171877: expected 15 fields, saw 2
Skipping line 177756: expected 15 fields, saw 2
Skipping line 181773: expected 15 fields, saw 2
Skipping line 191085: expected 15 fields, saw 2
Skipping line 196273: expected 15 fields, saw 2
Skipping line 196331: expected 15 fields, saw 2

Skipping line 197000: expected 15 fields, saw 2
Skipping line 197011: expected 15 fields, saw 2
Skipping line 197432: expected 15 fields, saw 2
Skipping line 208016: expected 15 fields, saw 2
Skipping line 214110: expected 15 fields, saw 2
Skipping line 244328: expected 15 fields, saw 2
Skipping line 248519: expected 15 fields, saw 2
Skipping line 254936: expected 15 fields, saw 2

Skipping line 272057: expected 15 fields, saw 2
Skipping line 293214: expected 15 fields, saw 2
Skipping line 310507: expected 15 fields, saw 2
Skipping line 312306: expected 15 fields, saw 2
Skipping line 316296: expected 15 fields, saw 2

Skipping line 336028: expected 15 fields, saw 2
Skipping line 344885: expected 15 fields, saw 2
Skipping line 352551: expected 15 fields, saw 2

Skipping line 408773: expected 15 fields, saw 2
Skipping line 434535: expected 15 fields, saw 2

Skipping line 581593: expected 15 fields, saw 2

Skipping line 652409: expected 15 fields, saw 2
```

```
df.head()
```

	marketplace	customer_id	review_id	
0	US	3653882	R3O9SGZBVQBV76	I
1	US	14661224	RKH8BNC3L5DLF	B
2	US	27324930	R2HLE8WKZSU3NL	E
3	US	7211452	R31U3UH5AZ42LL	E
4	US	12733322	R2SV659OUJ945Y	E

```
df.dropna(subset = ['review_body'],inplace = True)
```

```
df.reset_index(inplace = True,drop = True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 960056 entries, 0 to 960055
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   marketplace        960056 non-null  object
 1   customer_id        960056 non-null  int64 
 2   review_id          960056 non-null  object
 3   product_id         960056 non-null  object
 4   product_parent     960056 non-null  int64 
 5   product_title      960054 non-null  object
 6   product_category   960056 non-null  object
 7   star_rating        960056 non-null  int64 
 8   helpful_votes      960056 non-null  int64 
```

```
 9   total_votes        960056 non-null  int64
 10  vine               960056 non-null  object
 11  verified_purchase  960056 non-null  object
 12  review_headline    960049 non-null  object
 13  review_body        960056 non-null  object
 14  review_date        960052 non-null  object
dtypes: int64(5), object(10)
memory usage: 109.9+ MB
```

```
data = df.loc[:999,'review_body'].tolist()#change t
```

```
data
```

```
Bracelet Watch did not, reviewed
separately.",
 "I'm late getting to the party, but after
discovering Invicta watches I just can't
get enough of them. There is a watch for
every situation from dress to casual. After
acquiring several of these in a short time,
I can honestly say that I have not been let
down in style or performance. I don't think
I'll ever buy another watch that isn't an
Invicta!!!!!",
 'Wear it all the time!',
 'very good.',
 'Watch is exactly as it is shown in the
picture..would definitely recommend..',
 "Really large on the arm but that's what I
wanted - thx!"]
```

```
#Part2 Tokenizing and Stemming
```

```
stopwords = nltk.corpus.stopwords.words('english')
stopwords.append("'s")
stopwords.append("'m")
stopwords.append("br")
stopwords.append("watch")
print("We use" + str(len(stopwords)) + "stop-words
print(stopwords[:10])
```

```
We use183stop-words from nltk library.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours'
```

```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
def tokenization_and_stemming(text):
    tokens = []
    for word in nltk.word_tokenize(text):
        if word.lower() not in stopwords:
            tokens.append(word.lower())

    filtered_tokens = []

    # filter out any tokens not containing letters
    for token in tokens:
        if token.isalpha():
            filtered_tokens.append(token)

    # stemming
    stems = [stemmer.stem(t) for t in filtered_toke
    return stems
```

```
data[0]
```

```
'Absolutely love this watch! Get compliments a
lmost every time I wear it. Dainty.'
```

```
tokenization_and_stemming(data[0])
```

```
['absolut',
 'love',
 'get',
 'compliment',
 'almost',
 'everi',
 'time',
 'wear',
 'dainti']
```

```
#Part3:TF-IDF
```

```
from sklearn.feature_extraction.text import TfidfVe
tfidf_model = TfidfVectorizer(max_df = 0.99,max_fea
            min_df = 0.01,stop_words = 'english',
                              use_idf = True,tokeni
tfidf_matrix = tfidf_model.fit_transform(data) #fit

print ("In total, there are " + str(tfidf_matrix.sh
      " reviews and " + str(tfidf_matrix.shape[1])
```

```
In total, there are 1000 reviews and 239 terms.
```

```
tfidf_matrix
```

```
<1000x239 sparse matrix of type '<class
'numpy.float64'>'
        with 6891 stored elements in
Compressed Sparse Row format>
```

```
tfidf_matrix.toarray()
```

```
array([[0.       , 0.5125863, 0.       , ...,
0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
        0.       ],
       ...,
       [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
        0.       ]])
```

```
data[0]
```

```
'Absolutely love this watch! Get compliments a
lmost every time I wear it. Dainty.'
```

```
tfidf_matrix.toarray()[0]
```

```
array([0.        , 0.5125863 , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
```

```
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.46967673, 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.49124588,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.26515821, 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
0.        , 0.        ,
       0.        , 0.        , 0.        ,
```

```
tfidf_matrix.todense()#transform the sparse to dens
```

```
matrix([[0.       , 0.5125863, 0.       , ...,
0.       , 0.       ,
         0.       ],
        [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
         0.       ],
        [0.       , 0.       , 0.       , ...,
0.       , 0.       ,
```

```
         0.        ],
        ...,
        [0.        , 0.        , 0.        , ...,
0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ...,
0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ...,
0.        , 0.        ,
         0.        ]])
```

```
print(type(tfidf_matrix.toarray()))
```

```
<class 'numpy.ndarray'>
```

```
print(type(tfidf_matrix.todense()))
```

```
<class 'numpy.matrix'>
```

```
tf_selected_words = tfidf_model.get_feature_names_c
```

```
tf_selected_words
```

```
array(['abl', 'absolut', 'accur', 'actual',
'adjust', 'alarm', 'alreadi',
       'alway', 'amaz', 'amazon', 'anoth',
'arm', 'arriv', 'automat',
       'awesom', 'bad', 'band', 'batteri',
'beauti', 'best', 'better',
       'big', 'bit', 'black', 'blue',
'bought', 'box', 'bracelet',
       'brand', 'break', 'bright', 'broke',
'button', 'buy', 'ca', 'came',
       'case', 'casio', 'chang', 'cheap',
'clasp', 'classi', 'clock',
       'color', 'come', 'comfort',
'compliment', 'cool', 'cost', 'crown',
       'crystal', 'dark', 'date',
'daughter', 'day', 'deal', 'definit',
       'deliveri', 'design', 'dial',
'differ', 'difficult', 'disappoint',
       'display', 'dress', 'durabl',
'easi', 'easili', 'end', 'everi',
       'everyday', 'everyth', 'exact',
'excel', 'expect', 'expens',
       'face', 'fair', 'far', 'fast',
'featur', 'feel', 'fell', 'fine',
       'finish', 'fit', 'function', 'gave',
'gift', 'gold', 'good', 'got',
       'great', 'hand', 'happi', 'hard',
'heavi', 'high', 'hold',
```

```
       'honest', 'hope', 'hour', 'howev',
'husband', 'includ', 'instruct',
       'invicta', 'issu', 'item', 'kept',
'know', 'larg', 'leather',
       'light', 'like', 'link', 'littl',
'long', 'look', 'lot', 'love',
       'low', 'make', 'mani', 'metal',
'minut', 'model', 'money', 'month',
       'movement', 'need', 'new', 'nice',
'night', 'normal', 'number',
       'old', 'open', 'oper', 'order',
'origin', 'overal', 'packag',
       'paid', 'pay', 'perfect', 'perform',
'person', 'pictur', 'piec',
       'pin', 'place', 'plastic', 'pleas',
'point', 'press', 'pretti',
       'price', 'problem', 'product',
'purchas', 'qualiti', 'quick',
       'quit', 'rate', 'read', 'real',
'realli', 'reason', 'receiv',
       'recommend', 'red', 'remov',
'replac', 'resist', 'return',
       'review', 'right', 'run', 'said',
'say', 'screw', 'second',
       'seiko', 'seller', 'send', 'sent',
'set', 'sever', 'ship', 'short',
       'simpl', 'sinc', 'size', 'small',
'smaller', 'solid', 'someth',
       'somewhat', 'son', 'star', 'start',
'stop', 'strap', 'sturdi',
```

```
#Part4:K_means clustering
```

```
from sklearn.cluster import KMeans
num_clusters = 5
km = KMeans(n_clusters = num_clusters)
km.fit(tfidf_matrix)
clusters = km.labels_.tolist()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn
  warnings.warn(
```

```
product = {'review':df[:1000].review_body,'cluster'
frame = pd.DataFrame(product,columns = ['review','c
```

```
frame.head(10)
```

	review	cluster
0	Absolutely love this watch! Get compliments al...	2
1	I love this watch it keeps time wonderfully.	2
2	Scratches	0
3	It works well on me. However, I found cheaper ...	0
4	Beautiful watch face. The band looks nice all...	0
5	i love this watch for my purpose, about the pe...	2
6	for my wife and she loved it, looks great and ...	1

Next steps: View recommended plots

```
print('Number of reviews included in each cluster:'
frame['cluster'].value_counts().to_frame()
```

```
Number of reviews included in each cluster:
```

	cluster
0	656
2	115
1	92
3	75
4	62

```
km.cluster_centers_.shape
```

```
(5, 239)
```

```
print ("<Document clustering result by K-means>")

#km.cluster_centers_ denotes the importances of eac
#We need to sort it in decreasing-order and get the
order_centroids = km.cluster_centers_.argsort()[:,

Cluster_keywords_summary = {}
for i in range(num_clusters):
    print ("Cluster " + str(i) + " words:", end='')
    Cluster_keywords_summary[i] = []
    for ind in order_centroids[i, :6]: #replace 6 w
        Cluster_keywords_summary[i].append(tf_selec
        print (tf_selected_words[ind] + ",", end='' 
    print ()

    cluster_reviews = frame[frame.cluster==i].revie
    print ("Cluster " + str(i) + " reviews (" + str
    print (", ".join(cluster_reviews))
    print ()
```

```
<Document clustering result by K-means>
Cluster 0 words:look,like,band,work,time,beauti
Cluster 0 reviews (656 reviews):
Scratches, It works well on me. However, I foun

Cluster 1 words:great,look,price,work,product,d
Cluster 1 reviews (92 reviews):
for my wife and she loved it, looks great and a

Cluster 2 words:love,wife,look,husband,beauti,a
Cluster 2 reviews (115 reviews):
Absolutely love this watch! Get compliments alm

Cluster 3 words:good,product,price,qualiti,love
Cluster 3 reviews (75 reviews):
very good, It's a good value, and a good functi

Cluster 4 words:nice,price,look,realli,simpl,go
Cluster 4 reviews (62 reviews):
Nice watch, on time delivery from seller., It w
```

```
#Part5:Topic Modeling-Latent Dirichlet Allocation


from sklearn.decomposition import LatentDirichletAl
lda = LatentDirichletAllocation(n_components = 5)
```

```
lda_output = lda.fit_transform(tfidf_matrix)
print(lda_output.shape)
print(lda_output)
```

```
(1000, 5)
[[0.06068135 0.06191326 0.06062595 0.56073702 0
 [0.0838654  0.08650942 0.0829156  0.08581821 0
 [0.2        0.2        0.2        0.2        0
 ...
 [0.1000069  0.10017928 0.10085872 0.5989492  0
 [0.72562263 0.07201201 0.06825222 0.06674086 0
 [0.0674704  0.06814299 0.06816504 0.06723496 0
```

```
topic_word = lda.components_
print(topic_word.shape)
print(topic_word)
```

```
(5, 239)
[[1.74267471 0.20355186 0.20187854 ... 4.932304
 [2.75005334 0.20306839 1.8559635  ... 1.398607
 [0.200609   0.20014691 0.20023819 ... 0.200344
 [0.20095532 7.68455278 0.2001556  ... 0.203248
 [0.20103471 0.2044429  1.09551925 ... 0.201193
```

```
# column names
topic_names = ["Topic" + str(i) for i in range(lda.

# index names
doc_names = ["Doc" + str(i) for i in range(len(data

df_document_topic = pd.DataFrame(np.round(lda_outpu

# get dominant topic for each document
topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['topic'] = topic

df_document_topic.head(10)
```

	Topic0	Topic1	Topic2	Topic3	Topic4	to
Doc0	0.06	0.06	0.06	0.56	0.26	
Doc1	0.08	0.09	0.08	0.09	0.66	
Doc2	0.20	0.20	0.20	0.20	0.20	
Doc3	0.06	0.76	0.06	0.06	0.06	
Doc4	0.26	0.28	0.04	0.04	0.37	
Doc5	0.70	0.07	0.07	0.08	0.08	
Doc6	0.06	0.06	0.45	0.36	0.07	
Doc7	0.06	0.07	0.06	0.74	0.06	
Doc8	0.82	0.04	0.05	0.04	0.04	

Next steps: View recommended plots

```
df_document_topic['topic'].value_counts().to_frame(
```

	topic
1	303
4	225
0	197
2	148
3	127

```
print(lda.components_)
df_topic_words = pd.DataFrame(lda.components_)
df_topic_words.columns = tfidf_model.get_feature_na
df_topic_words.index = topic_names
df_topic_words.head()
```

```
[[1.74267471 0.20355186 0.20187854 ... 4.932304
 [2.75005334 0.20306839 1.8559635  ... 1.398607
 [0.200609   0.20014691 0.20023819 ... 0.200344
 [0.20095532 7.68455278 0.2001556  ... 0.203248
 [0.20103471 0.2044429  1.09551925 ... 0.201193
```

	abl	absolut	accur	actual	a
Topic0	1.742675	0.203552	0.201879	0.202845	0.2