Python Programming and Practice

Put on your song!
: Music Recommendation System

Final Report

Date : 2023.12.23

Name : Seokyoung Kim

ID : 182207

1. Introduction

1) Background

According to the International Federation of the Phonographic Industry (IFPI)'s Global Music Report 2023, the global music industry generated $25.9 billion in revenue in 2022, up 18.5% from 2021. These revenues have been reaching new highs every year, with this year's growth being the largest since the 1990s. In addition, the 2021 Music User Survey report released by the Korea Creative Content Agency found that 71.5% of respondents said they have spent more time listening to music since the coronavirus (COVID-19) pandemic. 88.3% of all respondents said they listen to music "at least once a week," with 51.7% saying they listen to music "almost every day. The number of people streaming music has also been on the rise, with 60.5% in 2019, 63.2% in 2020, and 65% in 2021. To compete in this rapidly growing music market, differentiated service technologies are being developed by streaming companies such as Melon, Genie, and YouTube Music.

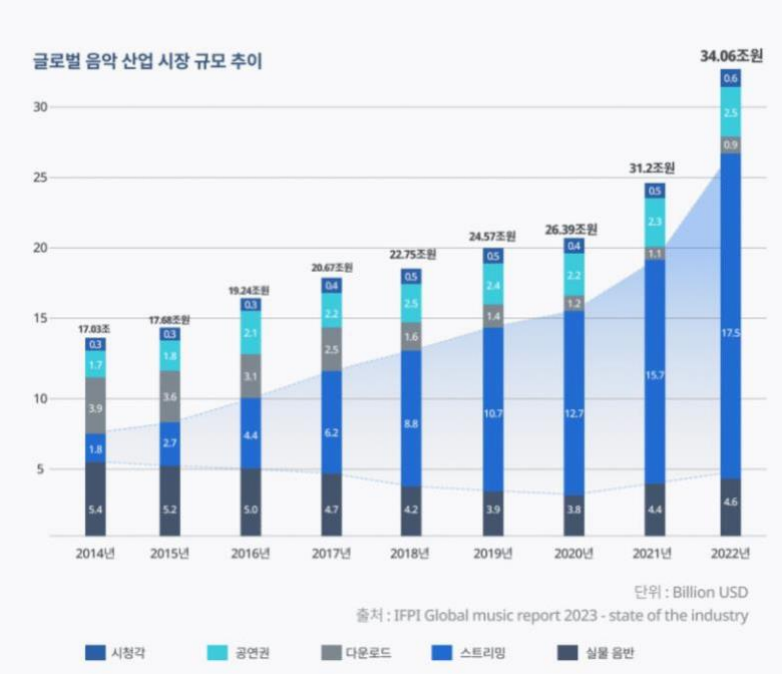


Figure 1. Global music industry market size trends

2) Project goal

In this project, we aim to build a system that recommends music to users by analyzing existing music data. It also aims to automatically create new playlists based on the user's playlists and generate titles for them.

3) Differences from existing programs

Traditional recommendation systems use other users with similar tastes to the user to recommend songs that the user has listened to. However, these systems simply recommend songs based on what you've listened to recently, rather than what you've listened to in the past, even on the same day, depending on your mood and the time of day. In addition, this requires a large enough number of users to collect data, and this limitation makes it difficult to recommend music from new users or independent artists. Especially with newer music, it can be difficult to recommend it to users before we have enough information to recommend it. In addition, the music recommendation systems currently used by many streaming apps are mainly limited to paid subscribers or song purchasers, and free users have limited access to them.

To solve these problems, this project proposes a system that recommends songs without existing user data based on a systematic analysis of music information. This is expected to reduce the gap in the music industry caused by the imbalance of user data.

2. Functional Requirement

*** The following functions can be changed during implementation.**

1) Load Dataset Function

- Download the dataset for analysis

  (1) Load dataset – music genre and file

2) Data Preprocessing Function

- Preprocessing data to extract clean features

  (1) Analyzing singer-sepecific data

  (2) Analyzing data by gender

  (3) Analyzing data by genre

  (4) Analyzing of the frequency of lyrics

  (5) Top-ranked frequency analysis

  (6) Analysis of the frequency of songwriting

3) Remove specific words Function

- Remove strange words/english combinations

  (1) Analyzing singer-sepecific data

  (2) Delete outlier or incorrect word

4) Tokenize Function

- Tokenize the words in sentences

  (1) Using tokenize library in tensorflow

  (2) Tokenize words and divide to ‘singer’, ‘genre’, etc

5) Clustering Data Function

- Clustering the data with traditional algorithm

  (1) K-means Clustering

  (2) Agglomerative Clustering

  (3) Analysis the morpheme

6) Main Function

- Set the epoch, save path, load path, samples, data file paths, batch size

  (1) Download the model (ex. GPT2LMHeadModel, kogpt2)

  (2) Download vocabulary

  (3) Call the tokenizer

  (4) Train the model using dataset

  (5) Evaluate the trained model

  (5) Save the results

7) Get a song/singer/playlist input and recommend system Function

- Recommended by receiving the song title and checking if the song is in the database you own.

(1) Recall 'doc2vec' model and run similarity

(2) Find the cluster to which each song belongs

(3) Call the tokenizer

(4) Choose randomly selected songs from songs in the cluster

(5) If the input is a singer, check the cluster that contains the most songs by the singer and select the randomly selected song

(6) Run similarly if the input is a playlist.

3. Implementation

*** The part where i applied what we learned is marked in red.**

1) Load dataset function

- Download the dataset for analysis

(1) Load font

```
import matplotlib.font_manager as fm
import os
import matplotlib.pyplot as plt

fe = fm.FontEntry(
    fname=r'./NanumSquareNeo-Variable.ttf',   # ttf 파일이 저장되어 있는 경로
    name='NanumGothic')                        # 이 폰트의 원하는 이름 설정
fm.fontManager.ttflist.insert(0, fe)              # Matplotlib에 폰트 추가
plt.rcParams.update({'font.size': 10, 'font.family': 'NanumGothic'}) # 폰트 설정
```

(2) Load dataset

- read dataset file(csv) using pandas library.

```
data = pd.read_csv("1992_2020_kpop.csv", engine='python',encoding='CP949')

train_data = pd.DataFrame(data)
```

(3) Verifying the data

```
train_data.head()
```

	년도	가수	제목	성별	장르	최고순위	작사	작곡	소속사	가사
0	19920000	쟴	난 멈추지 않는다	mixed	댄스	1	조진호	조진호	DSP 미디어	이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌이킬수...
1	19920000	쟴	우리 모두 사랑하자	mixed	댄스	7	조진수	조진수	DSP 미디어	워 우리 모두 사랑하자 우리의 젊은날을 위하여 우리 모두 춤을 추자 가벼운 인스텝 ...
2	19920000	쟴	이유	mixed	발라드	0	조진호	조진호	DSP 미디어	어둠은 드리워지고 이제는 우리들만의 시간이 지친듯이 내 가슴속에 밀려와 나를 재우고...
3	19920000	쟴	18번가의 비밀	mixed	댄스	0	조진호	조진호	DSP 미디어	끝없는 어둠 속을 천천히 걷고 있어 어디서 본 듯한 야릇한 느낌이야 거리는 비에 젖...
4	19920000	쟴	요즘 친구들	mixed	댄스	0	조진수	조진호	DSP 미디어	요즘 친구들은 흥 정말 진정한 친구가 뭔지 잘 몰라 말로만 떠들고 자기들이 다만 그...

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7235 entries, 0 to 7234
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   년도      7235 non-null   int64 
 1   가수      7235 non-null   object
 2   제목      7235 non-null   object
 3   성별      7235 non-null   object
 4   장르      7235 non-null   object
 5   최고순위    7235 non-null   object
 6   작사      7194 non-null   object
 7   작곡      7183 non-null   object
 8   소속사     7026 non-null   object
 9   가사      7233 non-null   object
dtypes: int64(1), object(9)
memory usage: 565.4+ KB
```

- Check the data shape and information

```
train_data.shape # 7235개 곡, 10개 속성
```

```
(7235, 10)
```

```
train_data.columns
```

```
Index(['년도', '가수', '제목', '성별', '장르', '최고순위', '작사', '작곡', '소속사', '가사'], dtype='object')
```

```
train_data['소속사'].describe()
```

```
count            7026
unique            251
top       (주)SM엔터테인먼트
freq              716
Name: 소속사, dtype: object
```

(4) Confirm the not available values in data using 'for' statements

```
# 각 attribute에 따른 N/A값의 수
for attribute in train_data.columns:
    print(train_data[attribute].isnull().sum())
```

```
0
0
0
0
0
0
41
52
209
2
```

2) Data Preprocessing Function

- Preprocessing data to extract clean features

(1) Analyzing singer-sepecific data based on index

```
train_data['가수'].describe()
```

```
count      7235
unique      455
top       방탄소년단
freq        317
Name: 가수, dtype: object
```

```
train_data['소속사'].describe()
```

```
count            7026
unique            251
top       (주)SM엔터테인먼트
freq              716
Name: 소속사, dtype: object
```

```
#groupby 가수: 빈도수 계산
singer_count=train_data.groupby('가수',sort = False).count().rename(columns = {'년도': 'count'})
singer_count=singer_count.reset_index().rename(columns={"index": "singer"})
singer_count=singer_count[['가수','count']]

topsinger=singer_count.sort_values('count',ascending = False).head(50)
topsinger.head(10)
```

	가수	count
267	방탄소년단	317
63	동방신기	280
334	TWICE (트와이스)	196
319	몬스타엑스	176
72	슈퍼주니어	155
47	오렌지	143
296	Red Velvet (레드벨벳)	141
291	마마무 (Mamamoo)	139
34	쥬얼리	134
74	빅뱅	134

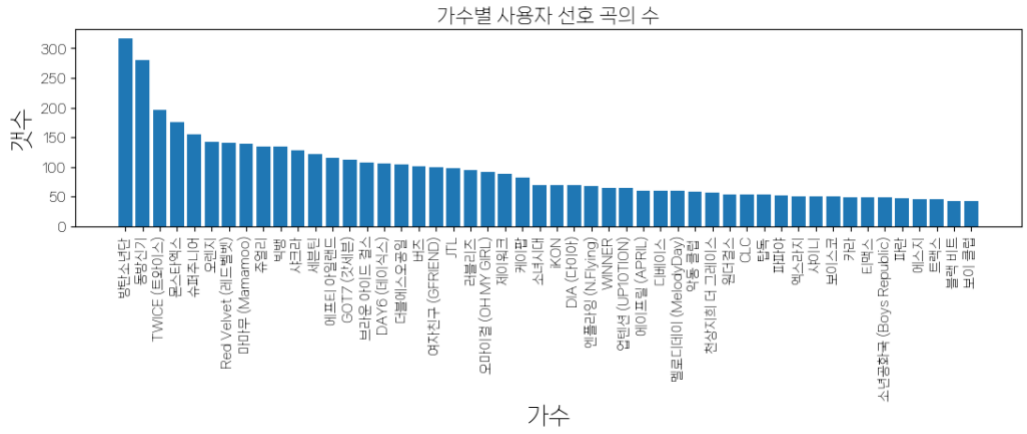
```
#groupby 소속사: 빈도수 계산
E_count=train_data.groupby('소속사',sort = False).count().rename(columns = {'년도': 'count'})
E_count=E_count.reset_index().rename(columns={"index": "소속사"})
E_count=E_count[['소속사','count']]

topE=E_count.sort_values('count',ascending = False).head(50)

topE.head(10)
```

	소속사	count
43	(주)SM엔터테인먼트	716
179	JYP 엔터테인먼트	401
121	빅히트 엔터테인먼트	290
0	DSP 미디어	212
5	SM 엔터테인먼트	211
77	스타쉽 엔터테인먼트	201
50	(주)YG엔터테인먼트	171
39	YG 엔터테인먼트	148
25	스타제국	147
36	오렌지엔터테인먼트	143

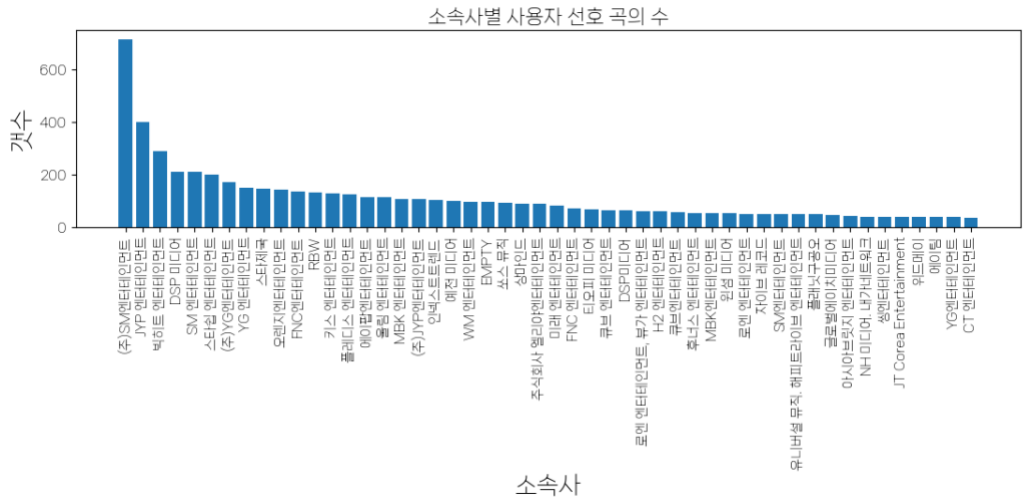
- Visualization using matplotlib library and index.

```
#시각화
index = np.arange(topsinger.shape[0])
x_name = topsinger['가수'].tolist()
plt.figure(figsize=(13,6))
plt.subplot(211)
plt.bar(index, topsinger['count'])
plt.title('가수별 사용자 선호 곡의 수', fontsize=15)
plt.xlabel('가수', fontsize=18)
plt.ylabel('갯수', fontsize=18)
plt.xticks(index, x_name, rotation= 90)
plt.rc('font', family='NanumGothic')

index = np.arange(topE.shape[0])
x_name = topE['소속사'].tolist()
plt.figure(figsize=(13,6))
plt.subplot(212)
plt.bar(index, topE['count'])
plt.title('소속사별 사용자 선호 곡의 수', fontsize=15)
plt.xlabel('소속사', fontsize=18)
plt.ylabel('갯수', fontsize=18)
plt.xticks(index, x_name, rotation= 90)
plt.rc('font', family='NanumGothic')

plt.show()
```

- Number of user favorite songs by singer.



- Number of user favorite songs by agency.



(2) Analyzing data by gender

- check the data of gender based on index.

```
train_data['성별'].describe()
```

```
count      7235
unique       15
top        male
freq       2543
Name: 성별, dtype: object
```

- group by gender using count function.

```
#groupby 성별
gender_count=train_data.groupby('성별',sort = True).count()

train_data.loc[train_data["성별"] == "MALE","성별"] = "male"
train_data.loc[train_data["성별"] == "Male","성별"] = "male"
train_data.loc[train_data["성별"] == "male","성별"] = "male"
train_data.loc[train_data["성별"] == "남","성별"] = "male"

train_data.loc[train_data["성별"] == "Famale","성별"] = "female"
train_data.loc[train_data["성별"] == "Female","성별"] = "female"
train_data.loc[train_data["성별"] == "female","성별"] = "female"
train_data.loc[train_data["성별"] == "여","성별"] = "female"

train_data.loc[train_data["성별"] == "Mixed","성별"] = "mixed"
train_data.loc[train_data["성별"] == "Mixed voices","성별"] = "mixed"
train_data.loc[train_data["성별"] == "all","성별"] = "mixed"
train_data.loc[train_data["성별"] == "mixed voices","성별"] = "mixed"
train_data.loc[train_data["성별"] == "mix","성별"] = "mixed"
train_data.loc[train_data["성별"] == "mixed","성별"] = "mixed"
```

```
gender_count=train_data.groupby('성별',sort = True).count()
gender_count #남자 가수 4366, 여자가수 2786, 혼성 83
```

성별	년도	가수	제목	장르	최고순위	작사	작곡	소속사	가사
female	1	1	1	1	1	1	1	1	1
female	2785	2785	2785	2785	2785	2782	2782	2674	2785
male	4366	4366	4366	4366	4366	4331	4320	4268	4364
mixed	83	83	83	83	83	80	80	83	83

(3) Analyzing data by genre

- check the data of genre.

```
train_data['장르'].describe()
```

```
count      7235
unique       54
top          댄스
freq       3419
Name: 장르, dtype: object
```

- group by genre using count function.

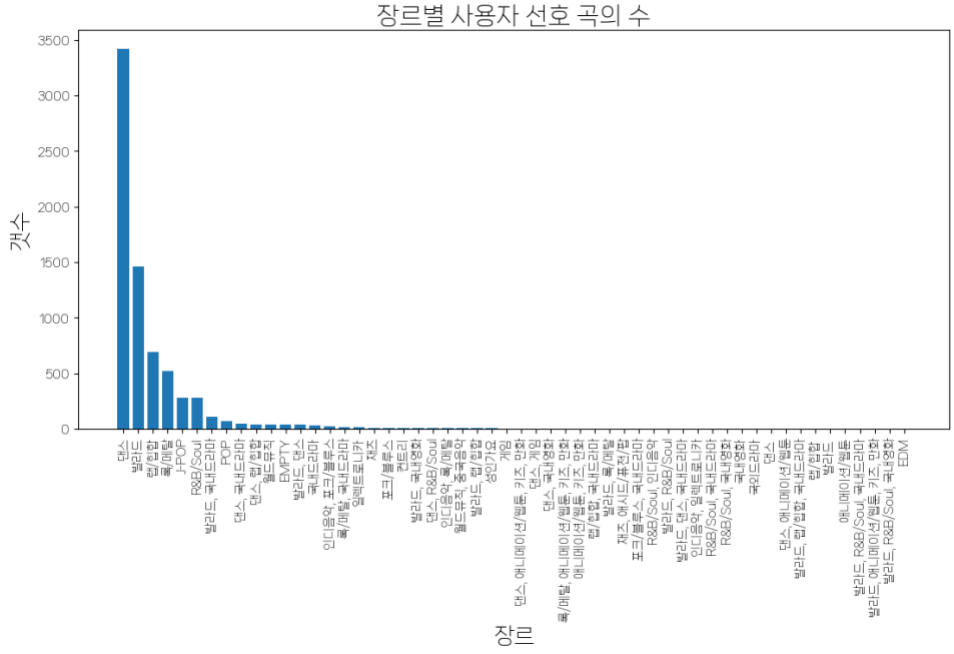
```
#장르
gnr_count=train_data.groupby('장르').count().rename(columns = {'년도': 'count'})
gnr_count.loc[:,'비율(%)'] = round(gnr_count['count']/sum(gnr_count['count'])*100,2)
gnr_count=gnr_count[['count','비율(%)']]
gnr_count=gnr_count.reset_index().rename(columns={"index": "gnr"})
gnr_list=gnr_count.sort_values('count',ascending = False)

gnr_count.sort_values('count',ascending=False).head(10)
```

	장르	count	비율(%)
12	댄스	3419	47.26
27	발라드	1458	20.15
21	랩/힙합	692	9.56
24	록/메탈	517	7.15
2	J-POP	280	3.87
4	R&B/Soul	277	3.83
32	발라드, 국내드라마	111	1.53
3	POP	69	0.95
16	댄스, 국내드라마	49	0.68
18	댄스, 랩/힙합	41	0.57

- Visualization using matplotlib library and index.

```
index = np.arange(gnr_list.shape[0])
x_name = gnr_list['장르'].tolist()

plt.figure(figsize=(13,6))
plt.bar(index, gnr_list['count'])
plt.title('장르별 사용자 선호 곡의 수', fontsize=20)
plt.xlabel('장르', fontsize=18)
plt.ylabel('갯수', fontsize=18)
plt.xticks(index, x_name,rotation=90)
plt.rc('font', family='NanumGothic')

plt.show()
```

- Number of user favorite songs by genre.



(4) Analyzing of the frequency of lyrics

- Check out the song without lyrics based on isnull and sum function : 2.

```
print(train_data['가사'].isnull().sum()) #2 가사 NULL
```

```
2
```

- Check the overlapping lyrics

```
# 중복된 가사 확인
train_data.shape[0] - train_data['가사'].nunique()
if (train_data.duplicated('가사')).any == True:
  print("중복된 가사 확인")
```

- Calling morpheme analyzer (Okt) and save the lyrics in variable 'okja' list based on 'for' statements and 'append' function.

```
#객체 호출
okt = Okt()
```

```
# 변수 okja에 전체가사 다시저장
words=train_data['가사']
okja = []
for line in words:
    okja.append(line)
```

- Separating morpheme by sentence using 'for' statements, 'if' statements, 'append' function, 'len' function, and try-except.

```
# 각 문장별로 형태소 구분하기
sentences_tag = []
i=0
for sentence in okja:
  try:
    morph = okt.pos(sentence)
    sentences_tag.append(morph)
    if i %1000 == 0:
      print("[{}/{}]done".format(i,len(okja)))
    i +=1
  except:
    print("NULL")
print(len(sentences_tag))
```

```
[0/7235]done
[1000/7235]done
[2000/7235]done
[3000/7235]done
[4000/7235]done
[5000/7235]done
[6000/7235]done
NULL
NULL
[7000/7235]done
7233
```

- Select and list only parts of speech that are nouns or adjectives and remove one word length based on list, 'for' statements, 'pop', 'append' function.

```
# 명사 혹은 형용사인 품사만 선별해 리스트에 담기 ### (이후 다른 품사에도 적용 가능)
noun_adj_list = []
for sentence1 in sentences_tag:
    for word, tag in sentence1:
        if tag in ['Noun','Adjective']:
            noun_adj_list.append(word)

# 단어의 길이가 하나인거 제거
for i,v in enumerate (noun_adj_list):
  if len(v) <2:
    noun_adj_list.pop(i)
```

- Calculate the frequency of each selected part of the speech & output top 10 frequencies.

```
# 선별된 품사별 빈도수 계산 & 상위 빈도 10위 출력
counts = Counter(noun_adj_list)
print(counts.most_common(10))
```

```
[('사랑', 16281), ('너', 8689), ('그대', 7764), ('내', 7072), ('우리', 5778), ('나', 5713), ('이제', 4504), ('없어', 4310), ('다시', 4159), ('맘', 3974)]
```

- Visualize with WordCloud.

```
#WordCloud 사용하여 시각화
wc = WordCloud(font_path='./NanumSquareNeo-Variable.ttf', background_color='white', width=800, height=600)
print(dict(counts))
cloud = wc.generate_from_frequencies(dict(counts))
plt.figure(figsize=(10, 8))
plt.axis('off')
plt.imshow(cloud)
plt.show()
```

(5) Top-ranked frequency analysis

- Check the Top-ranked frequency and confirm null values using sum function.

```
train_data['최고순위'].describe()
```

```
count      7235
unique      104
top       EMPTY
freq       3128
Name: 최고순위, dtype: object
```

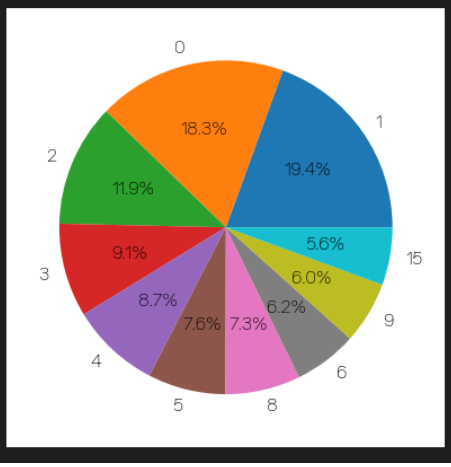
```
train_data['최고순위'].isnull().sum()
```

```
0
```

- Top-ranked frequency visualization.

```
Top_count=train_data.groupby('최고순위').count().rename(columns = {'년도': 'count'})
Top_count=Top_count.reset_index().rename(columns={"index": "최고순위"})
Top_count=Top_count[['최고순위','count']]

Top_count = Top_count.drop(index=[103,0])
Top_count=Top_count.sort_values('count',ascending=False).head(10)
```

```
plt.pie(Top_count['count'], labels=Top_count['최고순위'],autopct='%1.1f%%')
plt.show()
```



(6) Analysis of the frequency of songwriting

- Check the songwriting, songcomposing.

```
print(train_data['작사'].describe(),"\n=============")
print(train_data['작곡'].describe())
```

```
count      7194
unique     2186
top       EMPTY
freq       1246
Name: 작사, dtype: object
=============
count      7183
unique     2476
top       EMPTY
freq       1148
Name: 작곡, dtype: object
```

- Group by songwriter and count the values using count function.

```
ma_count=train_data.groupby('작사').count().rename(columns = {'년도': 'count'})
ma_count=ma_count.reset_index().rename(columns={"index": "작사"})
ma_count=ma_count[['작사','count']]
ma_count.sort_values('count',ascending=False).head(10)

mu_count=train_data.groupby('작곡').count().rename(columns = {'년도': 'count'})
mu_count=mu_count.reset_index().rename(columns={"index": "작곡"})
mu_count=mu_count[['작곡','count']]
```

```
ma_count.sort_values('count',ascending=False).head(10)
```

	작사	count
173	EMPTY	1246
1243	박진영	79
197	G-DRAGON	72
2121	한성호	67
708	Young K	61
918	김이나	58
1631	유영진	53
1368	서지음	51
905	김영아	50
322	KENZIE	49

```
mu_count.sort_values('count',ascending=False).head(10)
```

	작곡	count
292	EMPTY	1148
1599	박진영	61
1882	용감한 형제	57
1937	유영진	49
1295	김승수	45
2120	이현도	38
1541	박근태	37
2006	이민수	36
2401	한상원	34
2458	황성제	34

3) Remove specific words(Preprocessing) Function

- In this function refines the data and preprocessing process

(1) Analyzing singer-sepecific data

- After storing data in train_data, check the data information with the head function, 'len' function, 'for' statements, index.

```
train_data = pd.DataFrame(data)
```

```
train_data.head(3) # 데이터 구성요소
```

	년도	가수	제목	성별	장르	최고순위	작사	작곡	소속사	가사
0	19920000	쟴	난 멈추지 않는다	mixed	댄스	1	조진호	조진호	DSP 미디어	이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌이킬수...
1	19920000	쟴	우리 모두 사랑하자	mixed	댄스	7	조진수	조진수	DSP 미디어	워 우리 모두 사랑하자 우리의 젊은날을 위하여 우리 모두 춤을 추자 가벼운 인스텝 ...
2	19920000	쟴	이유	mixed	발라드	0	조진호	조진호	DSP 미디어	어둠은 드리워지고 이제는 우리들만의 시간이 지친듯이 내 가슴속에 밀려와 나를 재우고...

```
train_data.shape[0] # 곡 수
```

```
5619
```

```
len(train_data['가수'].unique()) # 유니크한 가수의 수
```

```
454
```

```
train_data.columns # 특성 목록
```

```
Index(['년도', '가수', '제목', '성별', '장르', '최고순위', '작사', '작곡', '소속사', '가사'], dtype='object')
```

- Check the number of N/A values for each attribute and delete them using 'for' statements, 'sum' function.

```
# 각 attribute에 따른 N/A값의 수
for attribute in train_data.columns:
    print(train_data[attribute].isnull().sum())

# 아래 결과에서 소속사가 없는 가수에 대한 정보가 다수 포함되어 있음을 알 수 있음.
# 클러스터링 단계에서는 가사만 사용되므로, 가사가 N/A값인 데이터만 삭제.
```

```
0
0
0
0
0
0
41
52
99
1
```

```
train_data = train_data[train_data['가사'].notnull()]
```

```
train_data.head(3)
```

	년도	가수	제목	성별	장르	최고순위	작사	작곡	소속사	가사
0	19920000	쟴	난 멈추지 않는다	mixed	댄스	1	조진호	조진호	DSP 미디어	이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌이킬수...
1	19920000	쟴	우리 모두 사랑하자	mixed	댄스	7	조진수	조진수	DSP 미디어	워 우리 모두 사랑하자 우리의 젊은날을 위하여 우리 모두 춤을 추자 가벼운 인스텝 ...
2	19920000	쟴	이유	mixed	발라드	0	조진호	조진호	DSP 미디어	어둠은 드리워지고 이제는 우리들만의 시간이 지친듯이 내 가슴속에 밀려와 나를 재우고...

```
print(train_data['가사'].isnull().sum())
```

```
0
```

(2) Delete outlier or incorrect word

- Remove duplicate lyrics.

```
# 중복된 가사 제거
train_data.shape[0] - train_data['가사'].nunique()
train_data.drop_duplicates(subset=['가사'], inplace=True)
```

- Check the total number of samples.

```
print('총 샘플의 수 :',len(train_data))
```

```
총 샘플의 수 : 5618
```

4) Tokenize Function

- Tokenize the words in sentences

(1) Specify an unspecified term and calling the Okt object. and then check the train data using index.

```
# 불용어 지정
stopwords = ['의','가','이','은','들','는','좀','잘','걍','과','도','를','으로','자','에','와','한','하다']
```

```
okt = Okt()
```

```
train_data['가사'][0]
```

```
'이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌이킬수 없는 후회도 하고  그러나 우리 잊어선 안돼 지금의 나는 내가
```

(2) Use Okt, classify words by morphemes, and perform some level of normalization with the 'for' statements, 'append' function.

```
# Okt를 사용, 형태소 단위로 단어 분류, 일정 수준의 정규화 실행
X_train = []
for sentence in train_data['가사']:
    temp_X = []
    temp_X = okt.morphs(sentence, stem=True) # 토큰화
    temp_X = [word for word in temp_X if not word in stopwords] # 불용어 제거
    X_train.append(temp_X)
```

(3) Save the tokenized X_test as a .npy file.

```
# 토큰화한 X_test를 저장

X_train_nparray = np.array(X_train, dtype='object')
np.save('./X_train',X_train_nparray)
```

```
X_train = np.load('./X_train.npy',allow_pickle=True).tolist()
```

(4) Doc2vec based weight training code is implemented, but I will write down in function 7.

5) Clustering Data Function

- Clustering the data with traditional algorithm

(1) K-means Clustering

- Two different tokenizers are used on '가수' and '장르' columns of the train_data_all DataFrame. The fit_on_texts method is used to update the internal vocabulary based on the given texts, and texts_to_sequences converts each text to a sequence of integers.

- A new list X2 is created to build input sequences. Using 'for' statements entry in the tokenized '장르' (X_genre) and '가수' (X_singer) lists, the code appends the first element of each to a temporary list (temp). This forms pairs of integers representing the genre and singer for each entry.

- The list X2 is converted into a NumPy array named X3. This array could be used as input data for clustering algorithms.

```
# 클러스터링 전처리
import copy
from gensim.models import Doc2Vec
from keras.preprocessing.text import Tokenizer
import numpy as np

# Assuming you have already trained a Doc2Vec model and stored it in the variable 'model'
# If not, you need to train a Doc2Vec model before using it.

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data_all['가수'])
X_singer = tokenizer.texts_to_sequences(train_data_all['가수'])

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data_all['장르'])
X_genre = tokenizer.texts_to_sequences(train_data_all['장르'])

# Assuming 'model' is your trained Doc2Vec model
X2 = []
for n, x_genre in enumerate(X_genre):
    temp = []

    temp.append(x_genre[0])
    temp.append(X_singer[n][0])

    X2.append(temp)

X3 = np.array(X2)
```

- Perform K-means clustering based on 8 clusters. (using dictionary and append function, for statements)

```
print("K-Means Clustering")


M_KMeans = KMeans(n_clusters=8, random_state=0)
X = X3 # document vector 전체를 가져옴.
M_KMeans.fit(X)# fitting

'''
M_KMeans = KMeans(n_clusters=8, random_state=0)
X = model.docvecs.vectors_docs # document vector 전체를 가져옴.
M_KMeans.fit(X)# fitting
'''

cluster_dict = {i:[] for i in range(0,8)}
for text_tags, label in zip(common_texts_and_tags, M_KMeans.labels_):
    text, tags = text_tags
    cluster_dict[label].append([tags, text])

'''
for label, lst in cluster_dict.items():
    print(f"Cluster {label}")
    for x in lst:
        print(x)
'''
```

- Create an empty lyrics list, turn cluster_dict into a for statement, and for each title, singer, append the lyrics that correspond to the cluster.

```
gasas = []
cluster_n = []
for clster in range(len(cluster_dict)):
    #print('cluster ' + str(clster))
    cluster_n_song = []
    gasa = []
    for song in cluster_dict[clster]:
        temp1 = list(train_data_all['제목'][train_data_all['제목']==song[0]][0].index)
        temp2 = list(train_data_all['가수'][train_data_all['가수']==song[0]][1].index)

        for title in temp1:
            x = -1
            if title in temp2:
                x = title
                break

        #print(train_data_all['제목'][x])
        #print(train_data_all['장르'][x])
        #print(train_data_all['가수'][x])
        gasa.append(train_data_all['가사'][x])
        cluster_n_song.append([train_data_all['장르'][x],train_data_all['가수'][x],train_data_all['소속사'][x],int(train_data_all['년도'][x]/10000),train_data_all['작곡'][x]])
    cluster_n.append(cluster_n_song)
    gasas.append(gasa)
```

(2) Agglomerative Clustering

- Perform Agglomerative clustering based on 10 clusters.

```
print("Agglomerative Clustering")

n_clusters = 10

M_Agglo = AgglomerativeClustering(linkage='ward',
        connectivity=None, n_clusters=n_clusters)

# Assuming 'model' is my trained Doc2Vec model
X = model.docvecs.vectors_docs

result = M_Agglo.fit_predict(X)
```

- Populating a list of lists (cluster_dict1), where each sublist corresponds to a cluster identified by a clustering algorithm. After the 'for' loop completes, cluster_dict1 is a list where each element (sublist) corresponds to a cluster, and each sublist contains a list of tags and texts belonging to that cluster. The structure is essentially a way to organize the data by clusters.

```
cluster_dict1 = []
for i in range(n_clusters):
    cluster_dict1.append([])

for n, i in enumerate(result):
    text, tags = common_texts_and_tags[n]
    cluster_dict1[i].append([tags, text])
```

- Create an empty lyrics list, turn cluster_dict into a for statement, and for each title, singer, append the lyrics that correspond to the cluster (same as K-means clustering).

```
gasas = []
cluster_n = []
for clster in range(len(cluster_dict1)):
    #print('cluster ' + str(clster))
    cluster_n_song = []
    gasa = []
    for song in cluster_dict1[clster]:
        temp1 = list(train_data_all['제목'][train_data_all['제목']==song[0]][0]].index)
        temp2 = list(train_data_all['가수'][train_data_all['가수']==song[0][1]].index)

        for title in temp1:
            x = -1
            if title in temp2:
                x = title
                break

        #print(train_data_all['제목'][x])
        #print(train_data_all['장르'][x])
        #print(train_data_all['가수'][x])
        gasa.append(train_data_all['가사'][x])
        cluster_n_song.append([train_data_all['장르'][x],train_data_all['가수'][x],train_data_all['소속사'][x],int(train_data_all['년도'][x]/10000),train_data_all['작곡'][x]])
    cluster_n.append(cluster_n_song)
    gasas.append(gasa)
```

(3) Analysis the morpheme and clustering data will be covered in function 7.

6) Main data check Function

(1) Confirm the Clustering morpheme and morpheme.

```
sentences_tag_n = []
for n, cluster in enumerate(gasas):
    sentences_tag = []
    for sentence in cluster:
        morph = okt.pos(sentence)
        sentences_tag.append(morph)
    sentences_tag_n.append(sentences_tag)
print(len(sentences_tag_n))
```

```
10
```

- process a collection of clusters using the Korean morphological analyzer library called "okt" using for loop and list append function.

- It performs morphological analysis on each sentence in each cluster using the "okt" library, and stores the results in a nested list structure. And then the length of is printed.

[43]

```
noun_adj_list_n = []
for i in range(len(sentences_tag_n)):
    noun_adj_list = []
    for sentence1 in sentences_tag_n[i]:
        for word, tag in sentence1:
            if tag in ['Noun','Adjective']:
                noun_adj_list.append(word)
    noun_adj_list_n.append(noun_adj_list)
```

[44]

```
banlist = ['사랑','우리','그대','나','너','내']
for n in (noun_adj_list_n):
    for i,v in enumerate(n):
        if len(v) <2:
            n.pop(i)
            continue
        if v in banlist:
            n.pop(i)
            continue
```

- And then upper code filters and processes the morphological analysis results to create a list of nouns and adjectives for each cluster. It also removes specific words based on length and a predefined banlist. The final result is printed for the first cluster. (list, for, if, continue, pop, append, range, len)

```
from collections import Counter
from wordcloud import WordCloud
for n in (noun_adj_list_n):
    counts = Counter(n)
    print(counts.most_common(10))
```

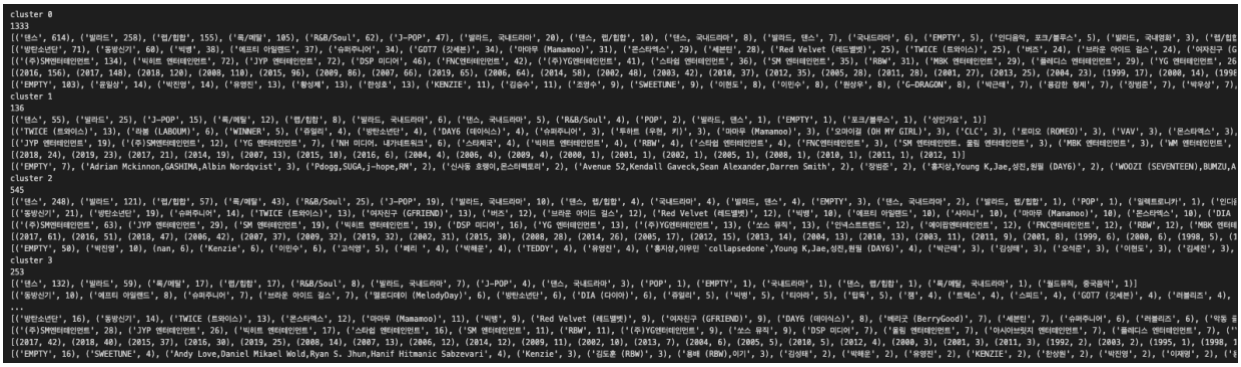
```
[('너', 1777), ('내', 1498), ('사랑', 1272), ('나', 1163), ('이제', 827), ('없어', 826), ('마음', 796), ('맘', 786), ('난', 768), ('사람', 756)]
[('내', 164), ('너', 151), ('나', 144), ('사랑', 102), ('맘', 94), ('이제', 86), ('없어', 81), ('오늘', 72), ('사람', 71), ('마음', 69)]
[('내', 645), ('너', 644), ('나', 578), ('사랑', 513), ('사람', 367), ('다시', 339), ('난', 328), ('마음', 323), ('생각', 321), ('이제', 320)]
[('너', 404), ('내', 283), ('사랑', 267), ('나', 246), ('이제', 214), ('없어', 179), ('다시', 179), ('모두', 159), ('사람', 150), ('생각', 149)]
[('너', 709), ('내', 648), ('나', 465), ('사랑', 438), ('없어', 348), ('맘', 328), ('사람', 313), ('난', 312), ('다시', 309), ('날', 297)]
[('너', 1241), ('내', 992), ('사랑', 923), ('나', 908), ('이제', 610), ('다시', 599), ('맘', 594), ('마음', 575), ('없어', 573), ('시간', 541)]
[('너', 156), ('내', 153), ('지금', 91), ('나', 90), ('맘', 85), ('마음', 82), ('날', 74), ('오늘', 74), ('없는', 67), ('이제', 61)]
[('너', 1065), ('내', 920), ('사랑', 813), ('나', 678), ('이제', 649), ('없어', 556), ('맘', 533), ('마음', 495), ('다시', 461), ('생각', 452)]
[('너', 475), ('내', 276), ('사랑', 265), ('나', 236), ('이제', 204), ('다시', 175), ('지금', 169), ('없어', 167), ('마음', 161), ('모두', 157)]
[('너', 383), ('내', 357), ('나', 281), ('사랑', 259), ('없어', 217), ('다시', 186), ('시간', 172), ('맘', 172), ('지금', 170), ('마음', 167)]
```

- It processes the lists of nouns and adjectives for each cluster, counts the occurrences of each word, and prints the ten most common words for each list. It seems to be part of a larger process that involves visualizing word frequencies using a word cloud.

(2) Confirm the Clustering data and morpheme.

- And then, it is a collection of clusters, where each cluster contains lists of tags related to genre, singer, house, time, and composer. It uses dictionaries to count the occurrences of each tag within each category for each cluster. The results are then printed, sorted by frequency in descending order for each category.

```
import operator

for n, cluster in enumerate(cluster_n):
    tags_genre = {}
    tags_singer = {}
    tags_house = {}
    tags_time = {}
    tags_composer = {}

    for tag_n in cluster:
        if tag_n[0] not in tags_genre.keys():
            tags_genre[tag_n[0]] = 1
        else:
            tags_genre[tag_n[0]] += 1

        if tag_n[1] not in tags_singer.keys():
            tags_singer[tag_n[1]] = 1
        else:
            tags_singer[tag_n[1]] += 1

        if tag_n[2] not in tags_house.keys():
            tags_house[tag_n[2]] = 1
        else:
            tags_house[tag_n[2]] += 1

        if tag_n[3] not in tags_time.keys():
            tags_time[tag_n[3]] = 1
        else:
            tags_time[tag_n[3]] += 1

        if tag_n[4] not in tags_composer.keys():
            tags_composer[tag_n[4]] = 1
        else:
            tags_composer[tag_n[4]] += 1

    print("cluster " + str(n))
    print(len(cluster))
    print(sorted(tags_genre.items(), key=operator.itemgetter(1),reverse=True))
    print(sorted(tags_singer.items(), key=operator.itemgetter(1),reverse=True))
    print(sorted(tags_house.items(), key=operator.itemgetter(1),reverse=True))
    print(sorted(tags_time.items(), key=operator.itemgetter(1),reverse=True))
    print(sorted(tags_composer.items(), key=operator.itemgetter(1),reverse=True))
```



- Append the specific keywords in cluster_tag and save the weight files.

```
cluster_tags = []
cluster_tags.append(['다시 한번','숨은'])
cluster_tags.append(['지나간','옛날','추억의'])
cluster_tags.append(['그리움','아이들'])
cluster_tags.append(['신나는','메탈'])
cluster_tags.append(['드라마틱','발라드','밤'])
cluster_tags.append(['대중적','아이돌','다양한'])
cluster_tags.append(['2세대','노인돌'])
cluster_tags.append(['1세대','들어본'])
cluster_tags.append(['힘찬','감각적인'])
cluster_tags.append(['랩','유영진','이국적인'])
```

```
cluster_tags_save = np.array(cluster_tags, dtype = 'object')
```

```
cluster_dict1 = np.array(cluster_dict1, dtype = 'object')
```

```
np.save('./cluster_tags_save',cluster_tags_save)
```

```
np.save('./cluster_dict',cluster_dict1)
```

- Confirm the cluster tags and cluster dictionary.

```
cluster_tags
```

```
[['다시 한번', '숨은'],
 ['지나간', '옛날', '추억의'],
 ['그리움', '아이들'],
 ['신나는', '메탈'],
 ['드라마틱', '발라드', '밤'],
 ['대중적', '아이들', '다양한'],
 ['2세대', '노인들'],
 ['1세대', '들어본'],
 ['힘한', '감각적인'],
 ['랩', '유영진', '이국적인']]
```

```
cluster_dict[0][0][1]
```

```
['사실',
 '기다리다',
 '있다',
 '날',
 '좋아하다',
 '너',
 '고백',
 '일부러',
 '튕겼었',
 '지만',
 '그건',
 '나',
 '진심',
 '아니다',
 '혹시',
 '포기',
 '어떡하다',
 '얼마나',
 '가슴',
 '졸이다',
 '넌',
 '아니다',
 '원래',
 '여자',
 '란',
...
 '날',
 '바라보다',
 '주기',
 '바',
 '래']
```

- It performs hierarchical clustering using the 'ward' method on document vectors obtained from a Doc2Vec model. The results of the clustering are stored in the 'linked' variable, which typically contains information about the hierarchical relationships between clusters.

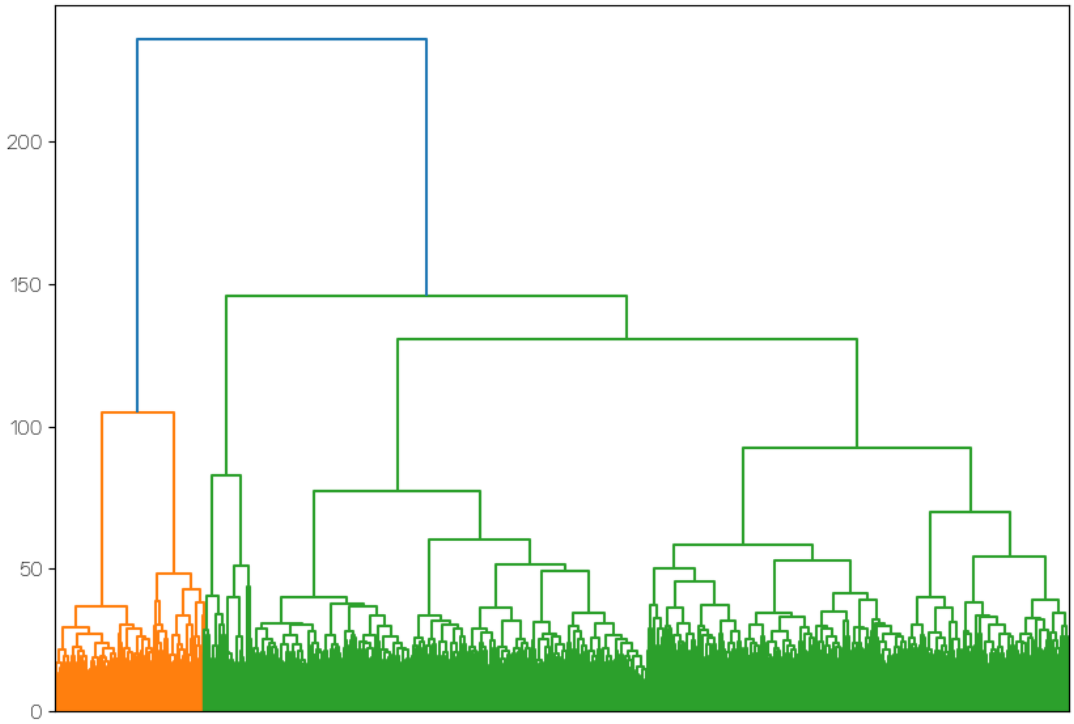
```
sys.setrecursionlimit(10000)

print("hierarchical Clustering")

X = model.docvecs.vectors_docs

linked = linkage(X, 'ward')
```

- Check the linked array.

```
linked
```

```
array([[4.82500000e+03, 4.82600000e+03, 3.34051772e-02, 2.00000000e+00],
       [2.16100000e+03, 2.16200000e+03, 3.42934017e-02, 2.00000000e+00],
       [3.94800000e+03, 3.94900000e+03, 3.54100036e-02, 2.00000000e+00],
       ...,
       [1.04140000e+04, 1.04160000e+04, 1.30454881e+02, 4.19500000e+03],
       [1.04150000e+04, 1.04180000e+04, 1.45956813e+02, 4.44600000e+03],
       [1.04170000e+04, 1.04190000e+04, 2.36109789e+02, 5.21100000e+03]])
```

- Visualize the linked using pyplot diagram.



7) Get a song/singer/playlist input and recommend system Function

- Recommended by receiving the song title and checking if the song is in the database you own.

(1) Function 1 : Take a song input and recommend a playlist.

- Input : name of song, ex) IU - 'Blueming'

```
input = ["Blueming"]
```

- Make sure the songs you've entered are in your database. If not, relax the search criteria (such as removing) to search. (list, for, try/except)

```
input_lyrics = []
for i in input:
    try :
        input_lyrics.append(train_data_all['가사'][train_data_all['제목']==i].values[0])
    except :
        try :
            # 검색 조건을 완화하여 검색
            print("입력한 제목과 정확히 일치하는 곡이 데이터베이스에 없습니다.")
            print("검색 조건을 완화하여 가사를 검색합니다.")
        except :
            print("입력한 제목과 일치하는 곡이 데이터베이스에 없습니다.")
            print("외부 데이터베이스에서 가사를 검색합니다.")
```

- Check the input lyrics.

```
input_lyrics[0]
```

```
''뭐해?'라는 두 글자에 '네가 보고 싶어' 나의 속마음을 담아 우 이모티콘 하나하나 속에 달라지는 내 미묘한 심리를 알까 우
```

- Tokenization and preprocessing the removing bull terms. (list, for, if, lambda)

```
input_lyrics_tokenized = []
for i in input_lyrics:
    temp_X = []
    temp_X = okt.morphs(i, stem=True) # 토큰화
    temp_X = [word for word in temp_X if not word in stopwords] # 불용어 제거
input_lyrics_tokenized.append(temp_X)
```

- Check the tokens.

```
input_lyrics_tokenized
```

```
[['',
  '뭐',
  '?',
  '',
  '라는',
  '두',
  '글자',
  '',
  '네',
  '보고',
  '싶다',
  '',
  '나',
  '속마음',
  '을',
  '담다',
  '우',
  '이모티콘',
  '하나',
  '하나',
  '속',
  '달라지다',
  '내',
  '미묘하다',
  '심리',
...
  '너',
  '에게',
  '송이',
  '더',
  '보내다']]
```

- Gets the doc2vec model and runs a similarity comparison with the songs in the database. (for, list, append function)

```
similarity_list = []
for text in input_lyrics_tokenized:
    inferred_v = model.infer_vector(text)
    print(f"vector of {text}: {inferred_v}")

    most_similar_docs = model.docvecs.most_similar([inferred_v], topn=5)
    for index, similarity in most_similar_docs:
        similarity_list.append(index)
```

```
vector of ['', '뭐', '?', '', '라는', '두', '글자', '', '네', '보고', '싶다', '', '나', '속마음', '을', '담다', '우', '이모티콘', '하나', '하나', '속', '달라지다', '내', '미묘하다', '심리', '일다',
 -1.9815159   1.362109   -2.367803   -0.23654045  1.6496413  -0.19876924
 -0.4494174  -0.4388416   1.0720826   2.0494323  -3.204669   -0.03340163
 -2.301487    0.462839    0.96016526 -0.7694015   0.49322462 -1.5152911
  0.41661695 -0.58827895  0.92162204 -0.84028304 -1.4572337   0.174985
 -0.40522024  2.422465    0.28833693  0.2927587  -0.71295065 -1.8860952
  1.4885858  -1.563536   -4.4920344  -0.98534817 -1.8535775  -0.94505566
 -0.62469876 -0.05438897 -2.2264335   0.15543476  0.752867   -1.1086013
 -0.2702848   0.4823291  -0.521387   -0.83014524  1.8622584   0.48444948
 -2.3866284   1.106595    0.44674808  0.13413216  0.28044143  0.78964514
  0.10021608  1.5585043   2.2702303   0.29993618  0.0672126  -1.0424107
  1.6577204   1.1067163   1.6731604  -0.3323773   0.49616376  0.81152093
  0.42504695  1.1726879  -0.8942435   1.4117634   1.2813181   1.0117211
  1.929888   -1.1561311  -0.72479975 -0.4167035   1.5907537   1.2113606
  0.42861795  0.04080913 -0.36686212 -0.9844595   3.1340714   1.0252876
  1.2293779  -1.1153218   0.59463555 -1.2489951   0.6706856   0.6791192
  2.335978    2.2826588   0.35578892 -0.03226048]
```

- Locate the cluster to which each song belongs. -> Cluster 9

(for, try/except, if, index, append function)

```
similarity_cluster = []

cluster_find = -1
for each_song in similarity_list:
    for n, cluster in cluster_dict.items():
            try:
                if each_song in cluster[0]:
                    cluster_find = n
            except:
                break
    similarity_cluster.append(cluster_find)
```

```
cluster_cnt = [0,0,0,0,0,0,0,0,0,0]
for i in similarity_cluster:
    cluster_cnt[i] += 1

max = -1
max_n = -1
for n, i in enumerate(cluster_cnt):
    if i > max:
        max = i
        max_n = n

print(max_n)
```

```
9
```

- Recommend randomly extracted songs among the songs belonging to cluster 9.

(for, random function, while, len, append function)

```
songint = []
for i in range(10):
    num = random.randrange(len(cluster_dict[0]))
    while(num in songint):
        num = random.randrange(len(cluster_dict[0]))
    songint.append(train_data_all['제목'][num])

print(f"아이유의 'Blueming'를 좋아한다면 이 노래도 들어보세요! : {songint}")
```

```
아이유의 'Blueming'를 좋아한다면 이 노래도 들어보세요! : ['아니라기에', '나쁜 마음을 먹게돼 (Ballad Ver.)', '뭇어본다 (대성 Solo)', '두번째 만남', '소원을 말해봐 (Genie)', 'Sweetest Love', '천사와 나무꾼', 'Phantom – 환영 (幻影)', '마법', '댄스! 댄스! 댄스!']
```

(2) Function 2 : Take a singer input and recommend a playlist.

- Input : name of singer, ex) BTS

```
input2 = ["방탄소년단"]
```

- Check the cluster that contains the most songs by the singer. -> Cluster 1 (for, list, dictionary, index)

```
for artist in input2:
    cnt = [0,0,0,0,0,0,0,0,0,0]
    for n, cluster in cluster_dict.items():
        for songs in cluster:
            if songs[0][1] == artist:
                cnt[n] += 1

print(cnt)
```

```
[268, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- Select a randomly extracted song from the songs in cluster 1.
(for, len, dictionary, index, random function, while)

```
songint = []
for i in range(10):
    num = random.randrange(len(cluster_dict[0]))
    while(num in songint):
        num = random.randrange(len(cluster_dict[0]))
    songint.append(train_data_all['제목'][num])
print(f"가수 '방탄소년단'를 기반으로 추천해드릴 플레이리스트입니다 : {songint}")
```

```
가수 '방탄소년단'를 기반으로 추천해드릴 플레이리스트입니다 : ['중독', 'Love', '내 곁에만 있어 (Best Place)', 'Pretty Girl (Bani Ver.)', 'Final Way', '시나브로', 'Secret', '몰래몰래', 'Stand By Me', 'I'll Be There']
```

4. Test results

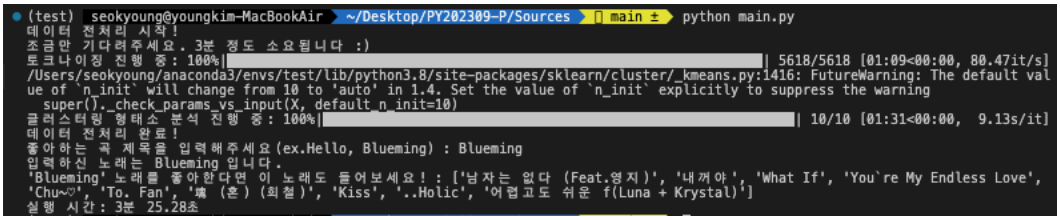
(1) Take a song input and recommend a playlist

- To finalize the functionality, i converted main.ipynb to main.py to meet the evaluation criteria. Main.py performs the main function of this project, which is to recommend song playlists by entering song titles. All the code in main.py is similar to the existing main.ipynb, and the visualization function was removed due to the characteristic of python files. In addition, considering the time required to perform data preprocessing, i used tqdm, a library built into Python. Through tqdm, i was able to check the time it takes to perform data preprocessing in real time. And the differences from the existing main.ipynb are as follows.

1) You can check the progress of preprocessing in real time.
2) You can check the recommended playlist by entering the song title directly.
3) If the song you entered does not exist in the database, exit the program.

* **Note**: The execution time of main.py is about 3 minutes, and the Python package and natural language processing library must be installed to run.

- Test result screenshot



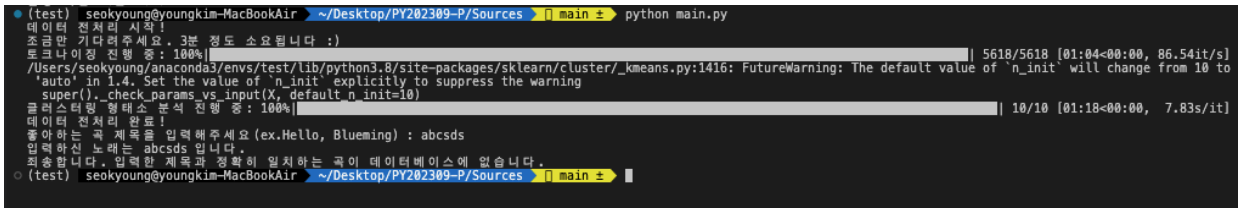
Input song : Blueming

Recommend Playlist : ['남자는 없다 (Feat.영지)', '내꺼야', 'What If', 'You`re My Endless Love', 'Chu~♡', 'To. Fan', '魂 (혼) (희철)', 'Kiss', '..Holic', '어렵고도 쉬운 f(Luna + Krystal)']

Time Required : 3m 25s

- **Exception Case**

: If the song entered by the user is not in the database, an error message will appear and exit the program.



Input song : abcsds

Recommend Playlist : None. song 'abcsds' is not in song database, so exit the program with prompts messages

(2) Take a singer input and recommend a playlist.

- A system that recommends playlists by entering a singer also has the same code structure as upper function. The only difference is the search index in the database, so please check this feature in the implementation phase of 3.

5. Changes in Comparison to the Plan

- None

6. Lessons Learned & Feedback

- Thank you so much for giving me a good lecture during the semester. I've learned Python before, but this class was a great reminder of what I'd forgotten, and I really satisfied the details of the lessons. The one thing that's unfortunate is that I can't take your class anymore as I will graduate next year. Thank you again for the great python lecture!