캡스톤
디자인

유튜브 급상승영상 분석

산업데이터사이언스학부 201804236 이혜규

INDEX

유튜브 급상승영상 분석

주 제 유튜브 급상승영상 분석

급상승 영상 분석

- 급상승 영상 가능 분류 분석
- 급상승 영상 군집화 분석
- 급상승 영상의 특징

주제 선정 이유

유튜브? 2005년에 시작한 동영상 공유 플랫폼

현재 구글에서 운영 중, 전세계 최대 규모

모든 연령대가 시청하고 누구나 쉽게 접근가능

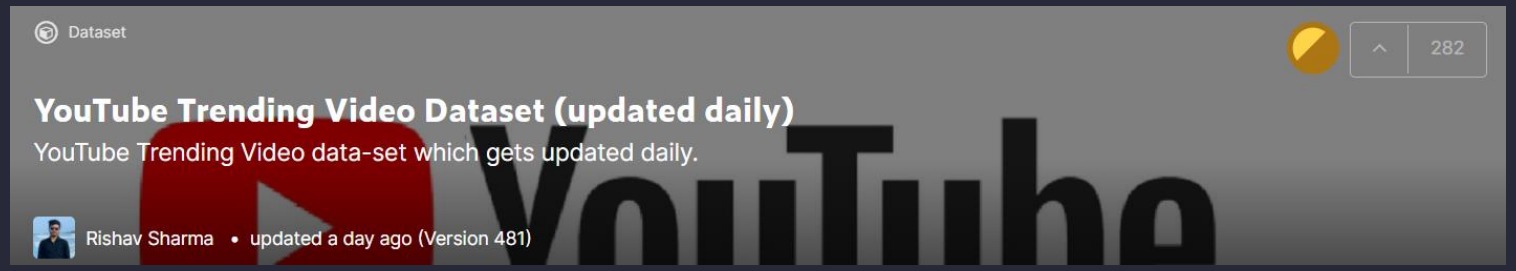
다양한 분야와 주제를 다루고 있음

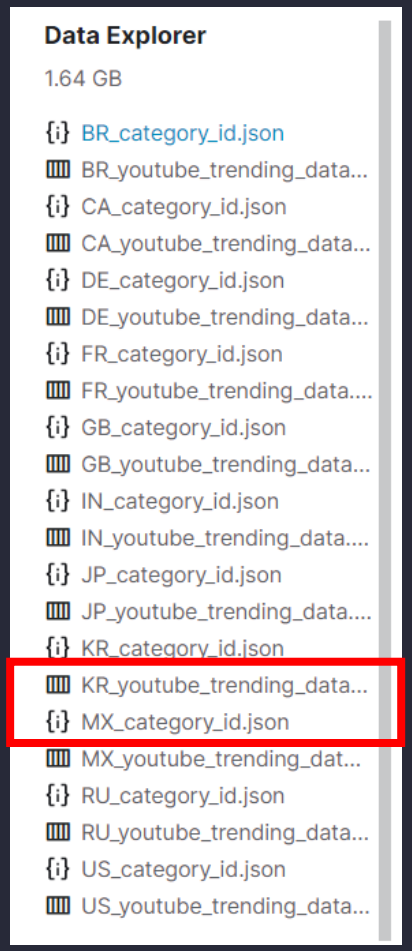
유튜버가 직업이라 생길만큼 유튜브의 시장은 점점 성장 중

데이터 설명

캐글 데이터

https://www.kaggle.com/rsrishav/youtube-trending-video-dataset





2020년 8월 12일 ~ 2021년 9월 20일 (약 1년간의 데이터)

총 79554개의 매일 자정 기준 인기 급상승 동영상에 있던 목록

급상승 영상 ?

광범위한 시청자가 관심을 보일만한 동영상을 먼저 보여주는 것을 목표로 만든 기준

맞춤설정과는 무관한 기준

각 국의 모든 사용자에게 동일한 목록의 영상을 표시, 약 15분 주기로 업데이트

인기 급상승 동영상의 동영상 순위를 결정하는 요소는 무엇인가요?

특정 기간에 YouTube에 업로드된 여러 신규 동영상 중에 인기 급상승 동영상은 극히 일부의 신규 동영상만 표시합니다. 인기 급상승 동영상은 다음과 같은 동영상을 노출하는 것을 목표로 합니다.

- 다양한 시청자의 관심을 끄는 동영상
- 현혹적이거나 클릭을 유도하거나 선정적이지 않은 동영상
- YouTube와 전 세계에서 일어나고 있는 일들을 다루는 동영상
- 크리에이터의 다양성을 보여주는 동영상
- 흥미와 새로움을 느낄 만한 동영상

인기 급상승 동영상에서는 이 조건을 모두 고려하고자 노력합니다. 이를 위해 인기 급상승 동영상은 다음을 포함하여 다양한 신호를 고려합니다.

- 조회수
- 동영상 조회수 증가 속도(즉, '온도')
- YouTube 외부를 포함하여 조회수가 발생하는 소스
- 동영상 업로드 기간
- 해당 동영상을 같은 채널에 최근 업로드한 다른 동영상과 비교한 결과

혜규

유튜브 급상승 영상 분석

캡스톤디자인 데이터사이언스전공 201804236 이혜규

데이터 이해

```
df.info()
```

```
df.iloc[:5, :8]
```

	video_id	title	publishedAt	channelId	channelTitle	categoryId	trending_date	tags
0	uq5LClQN3cE	안녕하세요 보겸입니다	2020-08-09T09:32:48Z	UCu9BCtGIEr73LXZ... KmoujKw	보겸 BK	24	2020-08-12T00:00:00Z	보겸|bokyem
1	I-ZbZCHsHD0	부락토스의 계획 [총몇명 프리퀄]	2020-08-12T09:00:08Z	UCRuSxVu4iqTK5kC... untAgA	총몇명	1	2020-08-12T00:00:00Z	총몇명|새겼는 판화|부락토스|투시퍼|총몇명 프리퀄|총몇명 스토리
2	9d7jNUjBoss	평생 반성하면서 살겠습니다.	2020-08-10T09:54:13Z	UCMVC92EOs9yDJC... CMesQ	양팡 YangPang	22	2020-08-12T00:00:00Z	양팡|양팡유튜브|양팡튜브|가족시트콤|양팡가족|양팡가족시트콤|양팡언니|현실남매|현실 자매...
3	3pl_L3-sMVg	안녕하세요 꽈뚜룹입니다.	2020-08-11T15:00:58Z	UCkQCwnkQfgSuP... Tnw_Y7v7w	꽈뚜룹 Quaddurup	24	2020-08-12T00:00:00Z	꽈뚜룹|한국여행기|quaddurup|뚜룹이|korealsouthkorealvlog|
4	zrsBjYukE8s	박진영 (J.Y. Park) When We Disco (Duet with 선미) M/V	2020-08-11T09:00:13Z	UCaO6TYtlC8U5ttz... 2hTZgg	JYP Entertainment	10	2020-08-12T00:00:00Z	JYP Entertainment|JYP|J.Y.Park|JYPark|박진영|선미|S...

```
df.iloc[:5, 8:]
```

	view_count	likes	dislikes	comment_count	thumbnail_link	comments_disabled	ratings_disabled	description
0	5947503	53326	105756	139946	https://i.ytimg.com/vi/uq5LClQN3cE/default.jpg	False	False	NaN
1	963384	28244	494	3339	https://i.ytimg.com/vi/I-ZbZCHsHD0/default.jpg	False	False	오늘도 정말 감사드립니다!!총몇명! 스토리 총몇명 십제곱, 5G민, MOVE혁...
2	2950885	17974	68898	50688	https://i.ytimg.com/vi/9d7jNUjBoss/default.jpg	False	False	NaN
3	1743374	36893	1798	8751	https://i.ytimg.com/vi/3pl_L3-sMVg/default.jpg	False	False	앞으로 좀 더 깔끔한 영상제작 약속 드리겠습니다.늘 감사드립니다
4	3433885	353337	9763	23405	https://i.ytimg.com/vi/zrsBjYukE8s/default.jpg	False	False	MetQm http://kko.to/TWyXd7zYjS potifolio https://...

```
 13  comments_disabled   79554 non-null  bool
 14  ratings_disabled    79554 non-null  bool
 15  description         78109 non-null  object
dtypes: bool(2), int64(5), object(9)
memory usage: 8.6+ MB
```

	속성명	속성 뜻
0	video_id	영상 코드
1	title	영상 제목
2	publishedAt	게시 일자
3	channelId	채널 코드
4	channelTitle	채널 제목
5	categoryId	카테고리
6	trending_date	급상승 영상 일자
7	tags	태그
8	view_count	조회수
9	likes	좋아요 수
10	dislikes	싫어요 수
11	comment_count	댓글 수
12	thumbnail_link	썸네일
13	comments_disabled	댓글 비공개 여부
14	ratings_disabled	좋아요/싫어요 버튼 비공개 여부
15	description	설명

마지막 열 Null 값 존재. 설명이 없는 부분이 null값 처리 됨

```
df["description"]=df["description"].fillna(0)
```

데이터 전처리

#	Column	Non-Null Count	Dtype	속성 뜻	데이터 변경사항
0	video_id	79554 non-null	object	영상 코드	
1	title	79554 non-null	object	영상 제목	
2	publishedAt	79554 non-null	object	게시 일자	게시날짜, 게시시간, 게시요일 추가
3	channelId	79554 non-null	object	채널 코드	
4	channelTitle	79554 non-null	object	채널 제목	
5	categoryId	79554 non-null	int64	카테고리	
6	trending_date	79554 non-null	object	급상승 영상 일자	급상승 날짜, 급상승 요일 추가
7	tags	79554 non-null	object	태그	태그 리스트로 변환, 각 태그 개수 추가
8	view_count	79554 non-null	int64	조회수	
9	likes	79554 non-null	int64	좋아요 수	
10	dislikes	79554 non-null	int64	싫어요 수	
11	comment_count	79554 non-null	int64	댓글 수	
12	thumbnail_link	79554 non-null	object	썸네일	
13	comments_disabled	79554 non-null	bool	댓글 비공개 여부	TRUE는 1, FALSE는 0으로 변경
14	ratings_disabled	79554 non-null	bool	좋아요/싫어요 비공개 여부	TRUE는 1, FALSE는 0으로 변경
15	description	78109 non-null	object	설명	설명이 있으면 1, 없으면 0으로 변경

데이터 전처리

```
from pytube import YouTube

play_time=[]
for i in range(len(count)):
    url = "https://www.youtube.com/watch?v=" + df2["index"][i]
    yt = YouTube(url)
    try:
        play_time.append(yt.length)
    except TypeError:
        play_time.append(0)
    print(i, "번째 시간 :", play_time[i])
play_time
```

```
0 번째 시간 : 61
1 번째 시간 : 228
2 번째 시간 : 0
3 번째 시간 : 81
4 번째 시간 : 257
5 번째 시간 : 20
6 번째 시간 : 0
7 번째 시간 : 258
8 번째 시간 : 486
9 번째 시간 : 835
10 번째 시간 : 930
```

재생 시간 변수 추가

```
import operator
sorted(count.items(), key=operator.itemgetter(1), reverse=True)
```

```
[('BhxadmfbFkg', 24),
 ('JoR9u44dNjI', 24),
 ('fQwkO_pjMP4', 23),
 ('GBcd8_5rFYk', 23),
 ('fANsFnkaX-U', 23),
 ('6cZfUAdTe9Y', 23),
 ('YLXfyHsfFz0', 22),
 ('SqIBCdGNj0g', 22),
 ('v4cUkiR1gms', 22),
 ('9pDSxezPTvE', 22),
 ('415YskuV4Lg', 22),
 ('cHezzE-fPNc', 22),
 ('q5KDJqPqzP8', 21),
 ('mYITh9E43s8', 21),
 ('8b9WiId0gMw', 21),
```

```
df2 = pd.DataFrame.from_dict(count, orient='index')
df2.reset_index(drop=False, inplace=True)
df2
```

	index	0
0	uq5LClQN3cE	7
1	I-ZbZCHsHD0	5
2	9d7jNUjBoss	7
3	3pl_L3-sMVg	7
4	zrsBjYukE8s	8
...	...	...
10918	esQPl9_rA6U	1
10919	WL4ipAjcspE	1
10920	yqLla5LbpH8	1
10921	n028FLMfsSY	1
10922	2NEVBW3eQm8	1

10923 rows × 2 columns

며칠동안 목록에 있었는지

데이터 전처리

```
df=df.drop_duplicates(['video_id'], keep = 'first')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10921 entries, 0 to 79362
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   video_id           10921 non-null  object
 1   video_title        10921 non-null  object
 2   channel_id         10921 non-null  object
 3   channel_title      10921 non-null  object
 4   category           10921 non-null  int64
 5   trending_date      10921 non-null  object
 6   trending_week      10921 non-null  object
 7   publishedAt_date   10921 non-null  object
 8   publishedAt_time   10921 non-null  object
 9   publishedAt_week   10921 non-null  object
 10  duration           10921 non-null  int64
 11  tags_split         10921 non-null  object
 12  tags_len           10921 non-null  int64
 13  view_count         10921 non-null  int64
 14  play_time          10921 non-null  int64
 15  likes              10921 non-null  int64
 16  dislikes           10921 non-null  int64
 17  comment_count      10921 non-null  int64
 18  comments_disabled  10921 non-null  int64
 19  ratings_disabled   10921 non-null  int64
 20  description        10921 non-null  int64
dtypes: int64(11), object(10)
memory usage: 1.8+ MB
```

첫번째로 인기동영상이 된 데이터가 의미 있다고 판단

=> 영상 코드가 겹치는 데이터 중 맨 처음을 남김

종속변수 선택

```
df["duration"].describe()
```

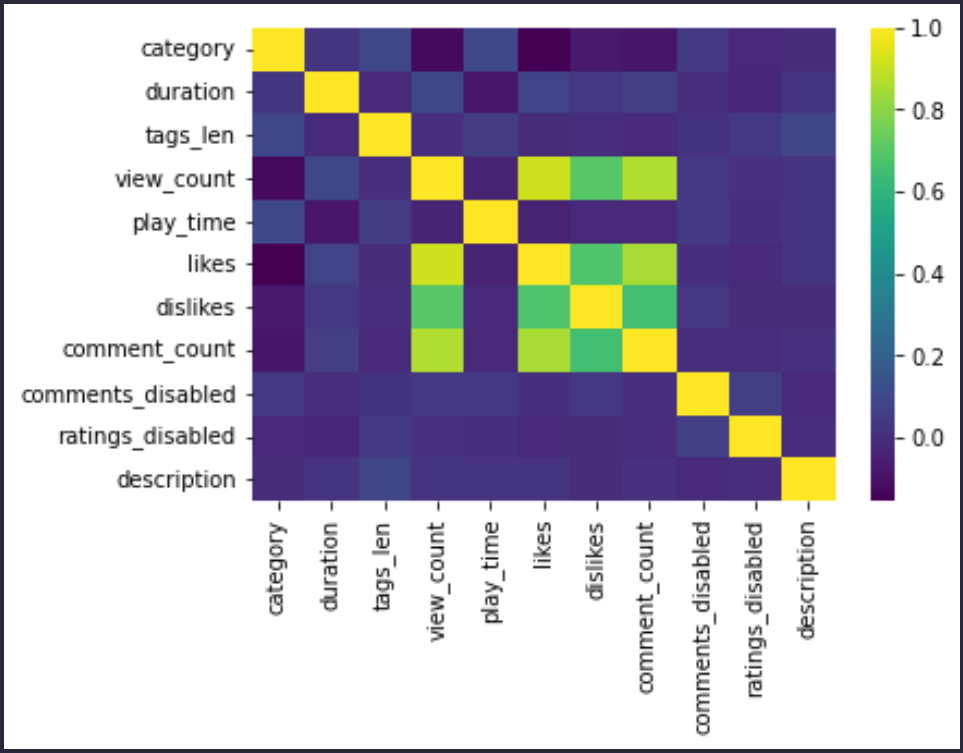
```
count   10921.00
mean        7.28
std         3.07
min         1.00
25%         5.00
50%         7.00
75%         9.00
max        24.00
Name: duration, dtype: float64
```

```
df1=df[df["duration"]>5]
df2=df[df["duration"]<=5]
df1["pred"]=1
df2["pred"]=0
df=pd.concat([df1, df2])
df.reset_index(inplace=True, drop=True)
df
```

임의로 정한 변수

데이터 시각화

상관관계

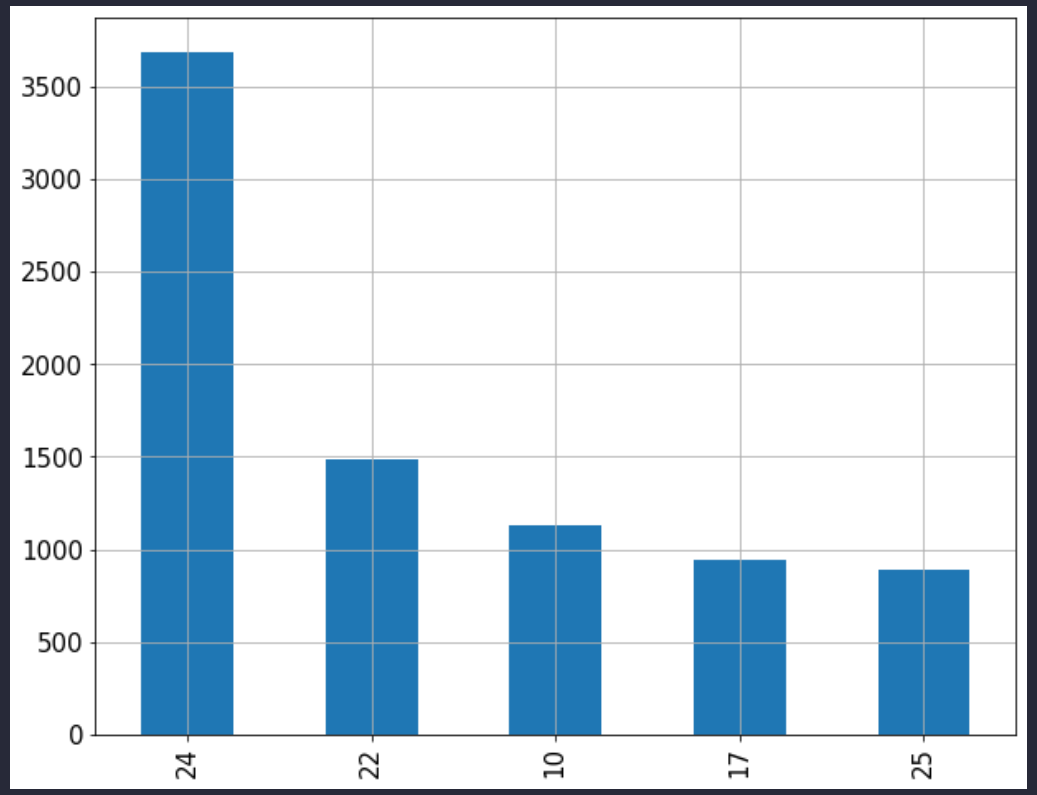


조회수와 상관관계가 높은 변수 (-+ 0.3이상)

데이터 시각화

태그에서 자주 보이는 단어 75개

자주 급상승 영상 목록에 올라가는 카테고리



혜규

유튜브 급상승 영상 분석

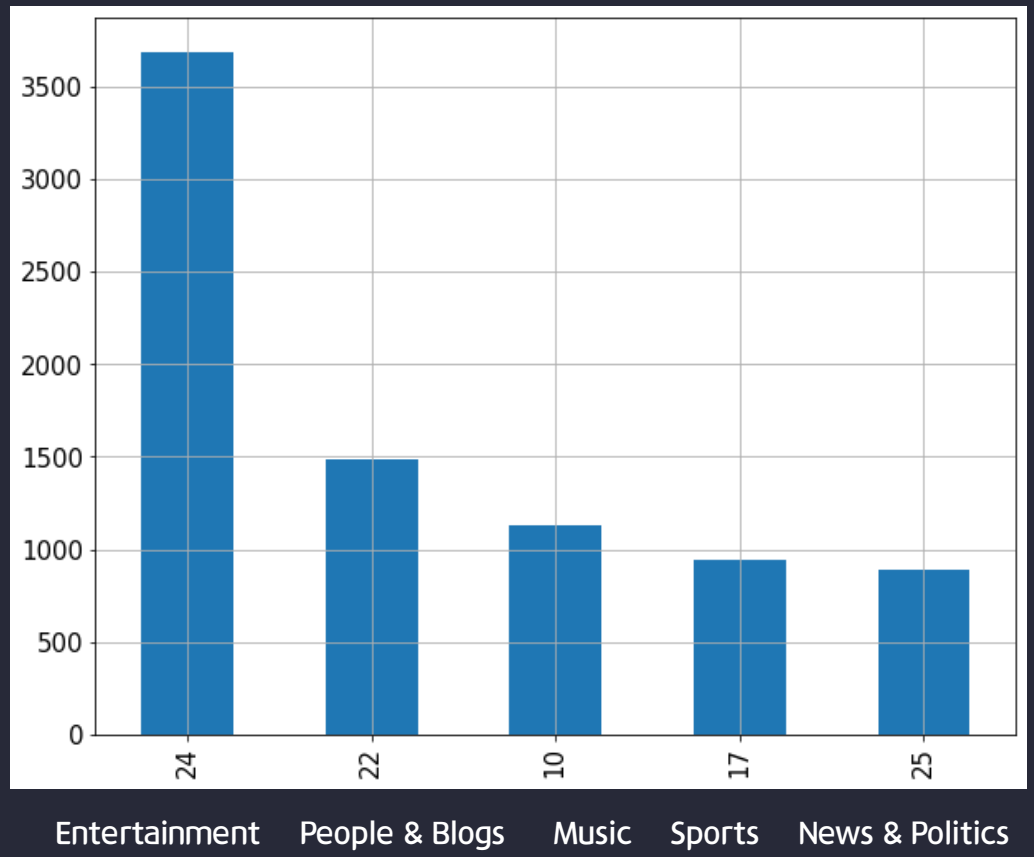
캡스톤디자인 데이터사이언스전공 201804236 이혜규

```
len(df["category"].unique())
```

```
15
```

유튜브의 카테고리는 총 44개이지만,
15개만의 카테고리만 올라갔었다.

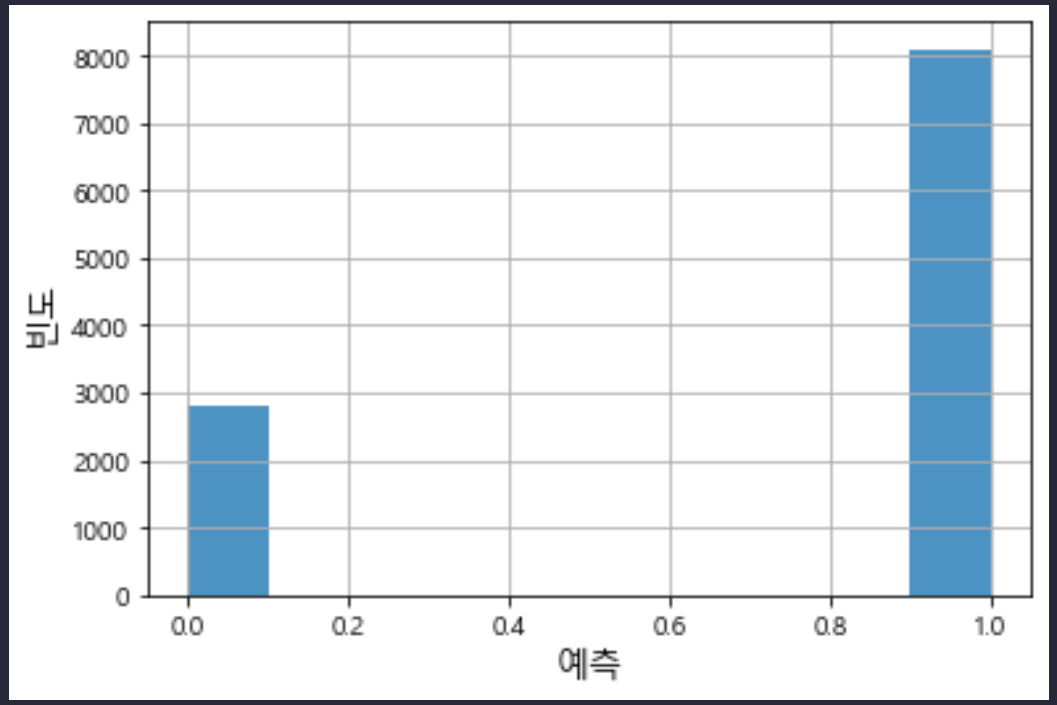
자주 급상승 영상 목록에 올라가는 카테고리



데이터 시각화

종속변수 시각화

```
plt.hist(df["pred"], bins=10, alpha=.8)
plt.ylabel("빈도", fontsize=13)
plt.xlabel("예측", fontsize=13)
plt.grid()
plt.show()
```



데이터 샘플링

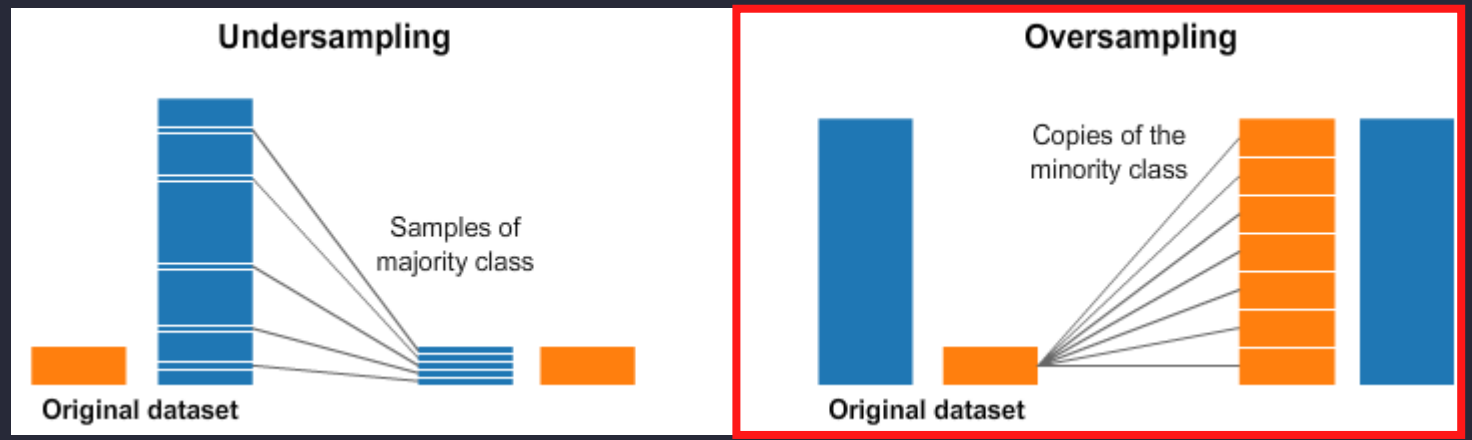
Imbalanced data (불균형 데이터)

- 불균형인 데이터를 학습시키면 한 쪽으로 치우져져, 제대로 분석 불가
- 해결방안 : Oversampling VS Undersampling

```
df["pred"].value_counts()
```

```
1    8097
0    2824
Name: pred, dtype: int64
```

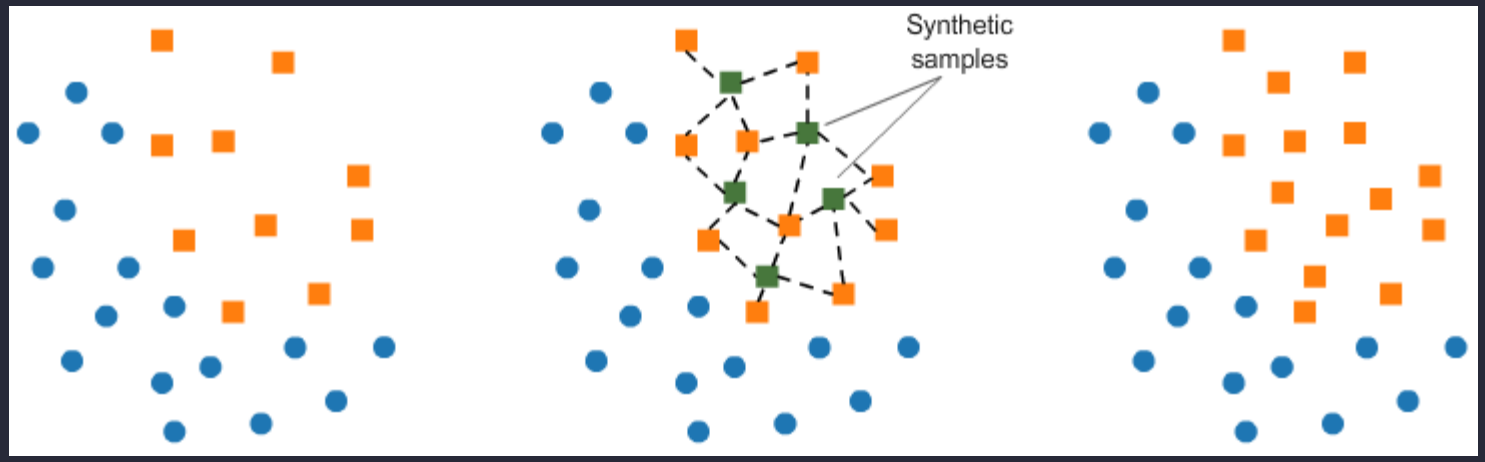
3 : 1의 비율



데이터 샘플링

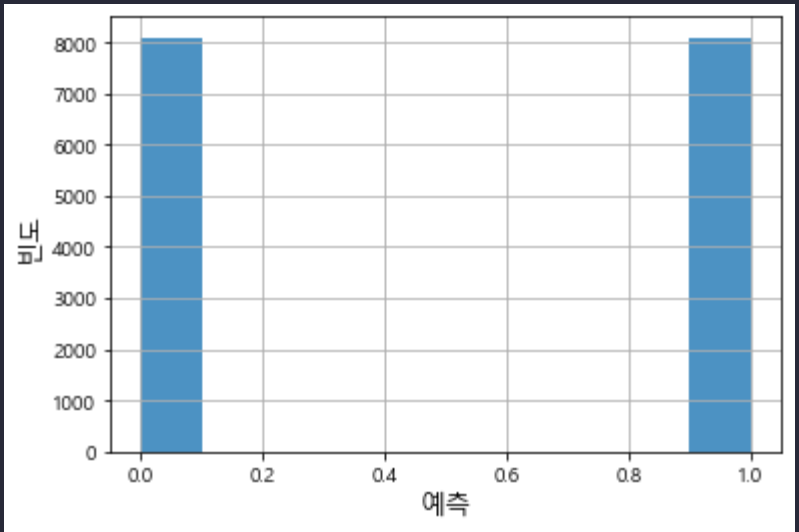
OverSampling

- 대표적인 알고리즘 SMOTE 사용
- 개별 데이터의 K 최근접 이웃(KNN)을 찾아 기존 데이터와 차이가 덜 나게 샘플링

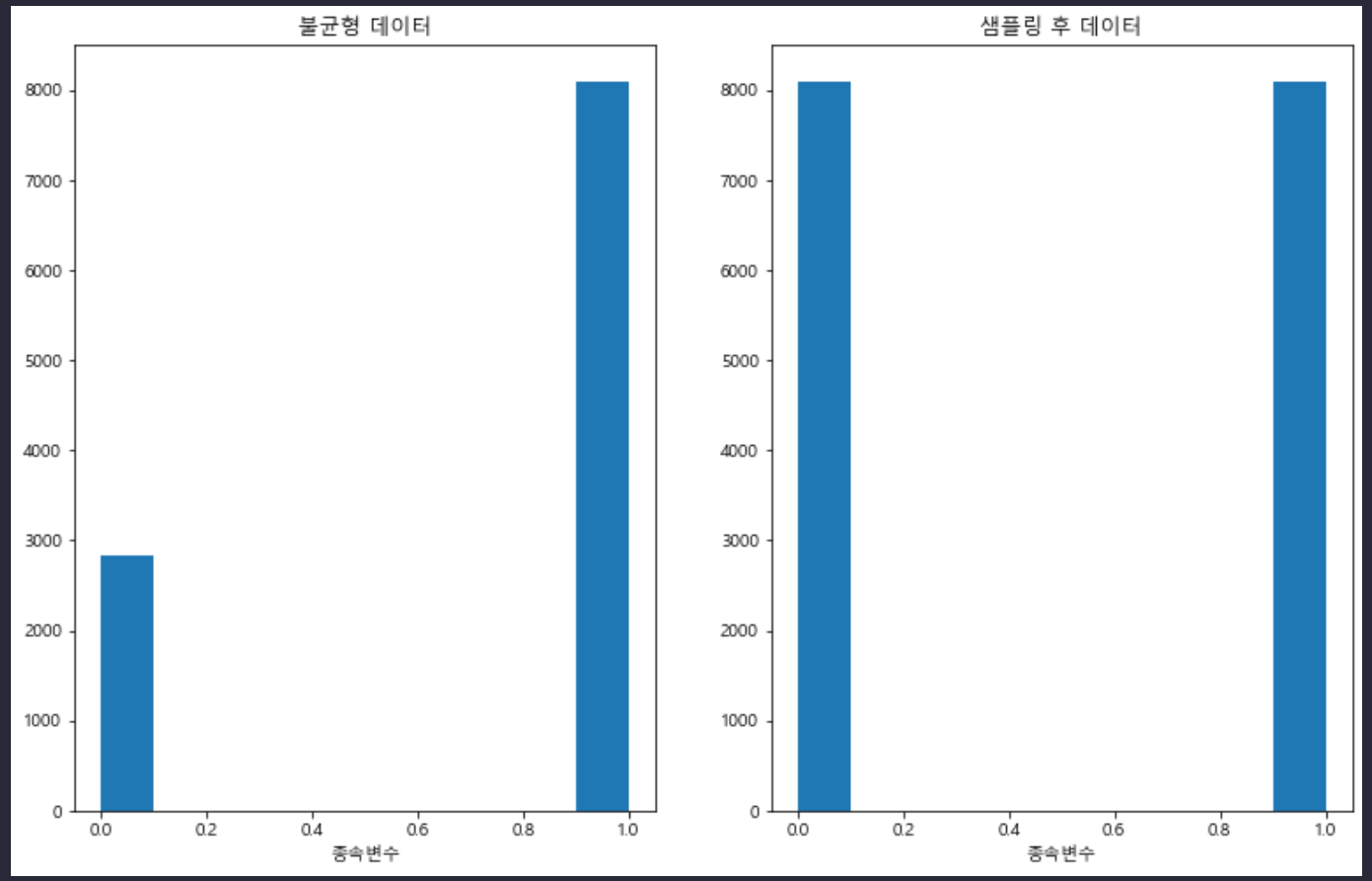


데이터 샘플링

```
from imblearn.over_sampling import SMOTE

X_resampled, y_resampled = SMOTE(random_state=0).fit_resample(X, Y)
```



데이터 샘플링


불균형 데이터
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
종속변수
샘플링 후 데이터
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
종속변수

분류 : Train / Test 분리

```
# 수치형 데이터 전체
X=df[["category", "publishedAt_week", "tags_len", "view_count", "play_time", "likes",
     "dislikes",comment_count", "comments_disabled", "ratings_disabled", "description",
     "video_len", "channel_len"]]
Y=df["pred"]

# Train / Test 분리
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X_resampled, y_resampled)
```

분류 : 데이터 스케일

StandardScaler

- 서로 다른 변수들의 값을 일정한 수준으로 표준화
- 평균 0, 분산이 1인 정규분포표를 가진 값으로 변환

$$x_i\_new = \frac{x_i - mean(x)}{std(x)}$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

분류 : 데이터 스케일

```
x_train.head()
```

	category	publishedAt_week	tags_len	view_count	play_time	likes	dislikes	comment_count	comn
12062	12	1	18	519191	578	23906	350	1562	0
1649	1	6	1	513918	0	6684	199	761	0
10239	24	0	29	4474680	218	132366	1782	11124	0
9454	17	6	1	166168	587	2078	53	242	0
15714	21	3	41	250015	651	7778	136	1112	0

```
x_train_scaled
```

```
array([[-1.3842798 , -0.99687924, -0.05758397, ...,  0.18214657,
         0.77297891, -0.32084706],
       [-3.2022663 ,  1.73745798, -0.99233316, ...,  0.18214657,
        -0.98372559,  1.18205363],
       [ 0.59897821, -1.54374668,  0.54725374, ...,  0.18214657,
        -0.31902659,  1.51603157],
       ...,
       [ 0.59897821,  0.64372309, -0.22253971, ...,  0.18214657,
         1.58011341,  2.85194329],
       [ 0.59897821,  0.64372309,  0.65722423, ...,  0.18214657,
         0.10827991, -1.15579189],
       [ 0.59897821,  0.64372309, -0.05758397, ...,  0.18214657,
         0.44062941, -0.32084706]])
```

분류 : KNN

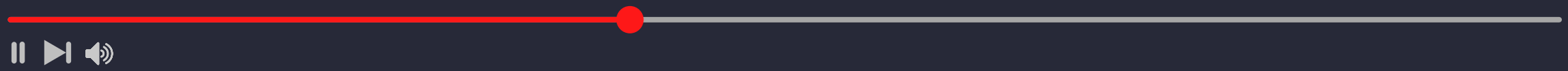
K 최근접 이웃 알고리즘

- 지도학습의 분류 알고리즘

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train_scaled, y_train)
knn_train_pred = knn.predict(x_train_scaled)
knn_test_pred = knn.predict(x_test_scaled)
```

```
from sklearn.metrics import accuracy_score
print("knn train 정확도 : {0:.3f}".format(accuracy_score(y_train, knn_train_pred)))
print("knn test  정확도 : {0:.3f}".format(accuracy_score(y_test, knn_test_pred)))
```

```
knn train 정확도 : 0.783
knn test  정확도 : 0.662
```



분류 : KNN

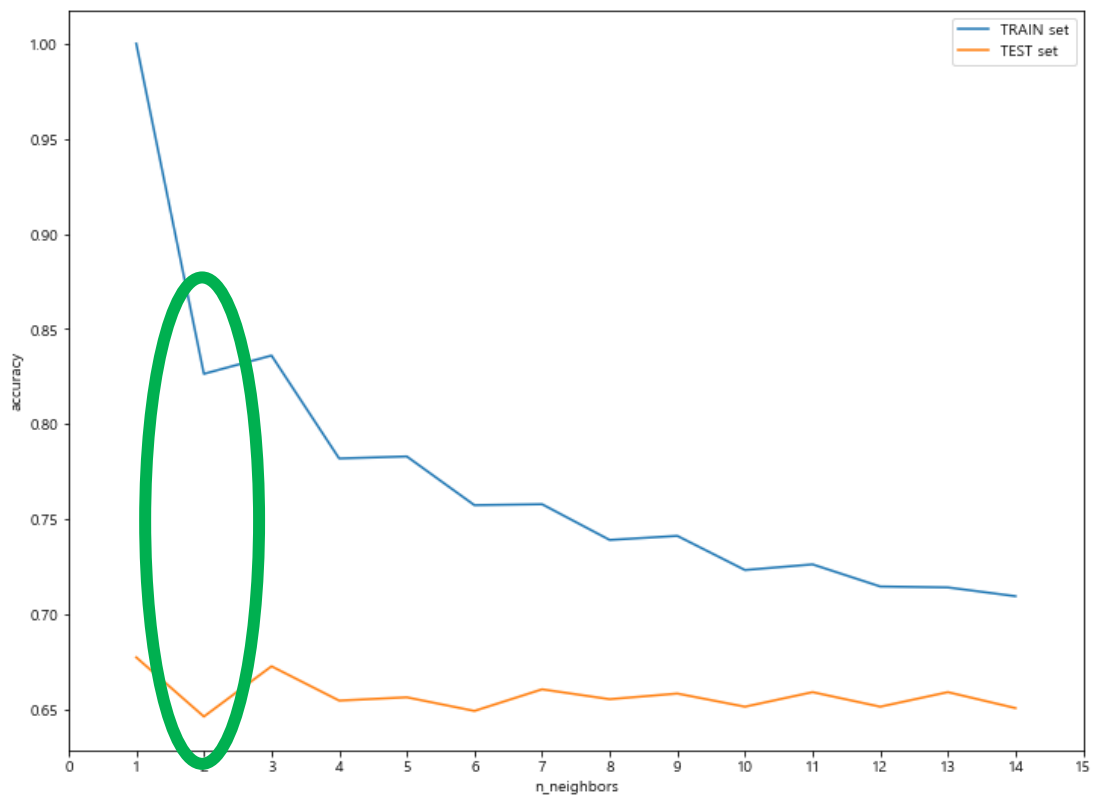
오차행렬을 통한 결과

```
from sklearn.metrics import confusion_matrix

print("KNN train\n", confusion_matrix(y_train, knn_train_pred))
print()
print("KNN test\n", confusion_matrix(y_test, knn_test_pred))
```

```
KNN train
 [[4905 1208]
 [1429 4603]]

KNN test
 [[1362  622]
 [ 748 1317]]
```

분류 : KNN

KNN의 주요 파라미터인 n_neighbors 을 1부터 15까지 정확도를 기준으로 검증했을 때,
k값이 2까지 급격하게 떨어졌다가 이후 점차 천천히 떨어진다.



분류 : KNN

k 값을 설정하여 괜찮은 성능을 봄

```
from sklearn.metrics import accuracy_score
print("knn train 정확도 : {0:.3f}".format(accuracy_score(y_train, knn_train_pred)))
print("knn test  정확도 : {0:.3f}".format(accuracy_score(y_test, knn_test_pred)))
```

```
knn train 정확도 : 0.783
knn test  정확도 : 0.662
```

```
from sklearn.metrics import confusion_matrix

print("KNN train\n", confusion_matrix(y_train, knn_train_pred))
print()
print("KNN test\n", confusion_matrix(y_test, knn_test_pred))
```

```
KNN train
 [[4905 1208]
 [1429 4603]]

KNN test
 [[1362  622]
 [ 748 1317]]
```

```
knn1 = KNeighborsClassifier(n_neighbors=2)
knn1.fit(x_train_scaled, y_train)
knn_train_pred1 = knn1.predict(x_train_scaled)
knn_test_pred1 = knn1.predict(x_test_scaled)
```

```
print("KNN k값 train 정확도 : {0:.3f}".format(accuracy_score(y_train, knn_train_pred1)))
print("KNN k값 test  정확도 : {0:.3f}".format(accuracy_score(y_test, knn_test_pred1)))
```

```
KNN k값 train 정확도 : 0.827
KNN k값 test  정확도 : 0.644
```

```
print("KNN train\n", confusion_matrix(y_train, knn_train_pred1))
print()
print("KNN test\n", confusion_matrix(y_test, knn_test_pred1))
```

```
KNN train
 [[6005    0]
 [2102 4038]]

KNN test
 [[1768  324]
 [1118  839]]
```

분류 : LDA

LDA, 선형판별분석(Linear Discriminant Analysis)

- 지도학습의 분류 알고리즘
- 2개 이상의 클래스를 최대한 멀리 분리해주는 방식
- 2개이상 범주때 사용, 이 범주가 명확할수록 좋음

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
lda.fit(x_train_scaled, y_train)
lda_train_pred = lda.predict(x_train_scaled)
lda_test_pred = lda.predict(x_test_scaled)
```

```
print("LDA train 정확도 : {0:.3f}".format(accuracy_score(y_train, lda_train_pred)))
print("LDA test  정확도 : {0:.3f}".format(accuracy_score(y_test, lda_test_pred)))
```

```
LDA train 정확도 : 0.564
LDA test  정확도 : 0.555
```

```
print("LDA 전체 train\n", confusion_matrix(y_train, lda_train_pred))
print()
print("LDA 전체 test\n", confusion_matrix(y_test, lda_test_pred))
```

```
LDA 전체 train
 [[3263 2850]
 [2440 3592]]

LDA 전체 test
 [[1050  934]
 [ 866 1199]]
```

분류 : SVM

SVM, 서포트 벡터 머신 (Support Vector Machine)

- 지도학습의 분류 알고리즘
- 2개 이상의 클래스 마진을 최대화하는 방식
- 어느 한쪽에 치우져지지 않은 데이터에 사용

```
from sklearn import svm
from sklearn.svm import SVC

svm = SVC()
svm.fit(x_train_scaled, y_train)
svm_train_pred = svm.predict(x_train_scaled)
svm_test_pred = svm.predict(x_test_scaled)
```

```
from sklearn.metrics import accuracy_score
print("svm train 정확도 : {0:.3f}".format(accuracy_score(y_train, svm_train_pred)))
print("svm test  정확도 : {0:.3f}".format(accuracy_score(y_test, svm_test_pred)))
```

```
svm train 정확도 : 0.663
svm test  정확도 : 0.649
```

```
from sklearn.metrics import confusion_matrix

print("SVM train\n", confusion_matrix(y_train, svm_train_pred))
print()
print("SVM test\n", confusion_matrix(y_test, svm_test_pred))
```

```
SVM train
 [[3893 2220]
 [1869 4163]]

SVM test
 [[1200  784]
 [ 637 1428]]
```

분류 알고리즘 정리

	KNN	LDA	SVM
정확도	train 정확도 1.000 test 정확도 0.641	train 정확도 0.564 test 정확도 0.555	train 정확도 0.663 test 정확도 0.649
오차행렬	KNN train [[6005 0] [0 6140]] KNN test [[1379 713] [742 1215]]	LDA 전체 train [[3263 2850] [2440 3592]] LDA 전체 test [[1050 934] [866 1199]]	SVM train [[3893 2220] [1869 4163]] SVM test [[1200 784] [637 1428]]

군집 : K-Means

K-Means, K-평균 (K-means clustering)

- 주어진 데이터를 k개의 클러스터로 묶는 알고리즘
- 각 클러스터와 거리 차이의 분산을 최소화하는 방식

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(df_sc)
```

```
KMeans(n_clusters=3)
```

```
km_df.groupby(["target", "cluster"])["view_count"].count()
```

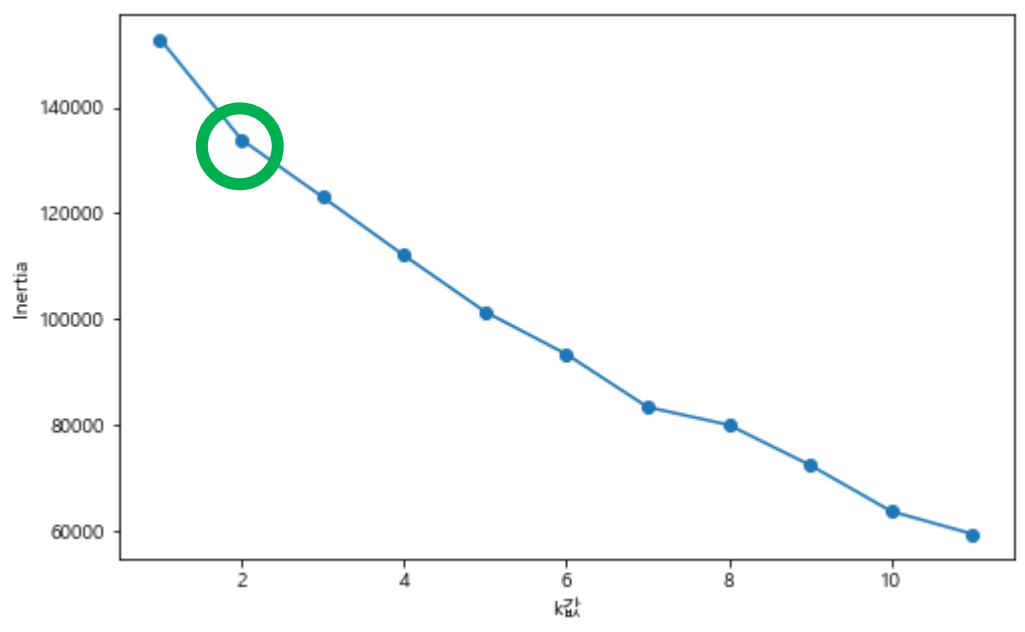
```
target  cluster
0       0          1587
        1          1235
        2             2
1       0          5155
        1          2923
        2            19
Name: view_count, dtype: int64
```

```
#데이터가 퍼진 정도. 작을수록 좋음
print("데이터 퍼진 정도 : {0:.3f}".format(kmeans.inertia_))
```

```
데이터 퍼진 정도 : 123072.675
```

군집 : K-Means

K-Means 의 주요 파라미터인 n_cluster 값을 1부터 12까지 검증했을 때,
k값이 2까지 좀 더 급하게 떨어졌다가 이후 점차 천천히 떨어진다.



군집 : K-Means

데이터의 분산 정도를 보았을 때 큰 차이가 없음

```
print("데이터 퍼진 정도 : {0:.3f}".format(kmeans.inertia_))
```

```
데이터 퍼진 정도 : 123072.675
```

전

```
print("데이터 퍼진 정도 : {0:.3f}".format(s_kmeans.inertia_))
```

```
데이터 퍼진 정도 : 133913.844
```

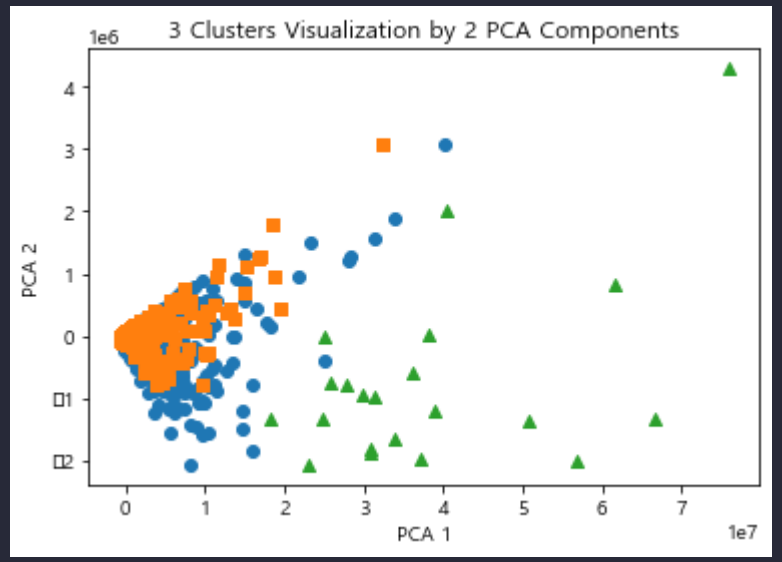
후

군집 : K-Means, PCA

 # 혹시나 피처가 많아서 잘 진행이 안될까 하여 주성분분석 시행.

=> 그 중 2개만 골라서 진행, 그래도 큰 효과는 없었음

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(km_df)

km_df['pca_x'] = pca_transformed[:,0]
km_df['pca_y'] = pca_transformed[:,1]
```



군집 : K-Means, PCA

다른 피처들도 어떤 분산 비율을 보이는지 확인.

- 누적 기여율이 80%가 넘어가는 순간이 10번째 피처

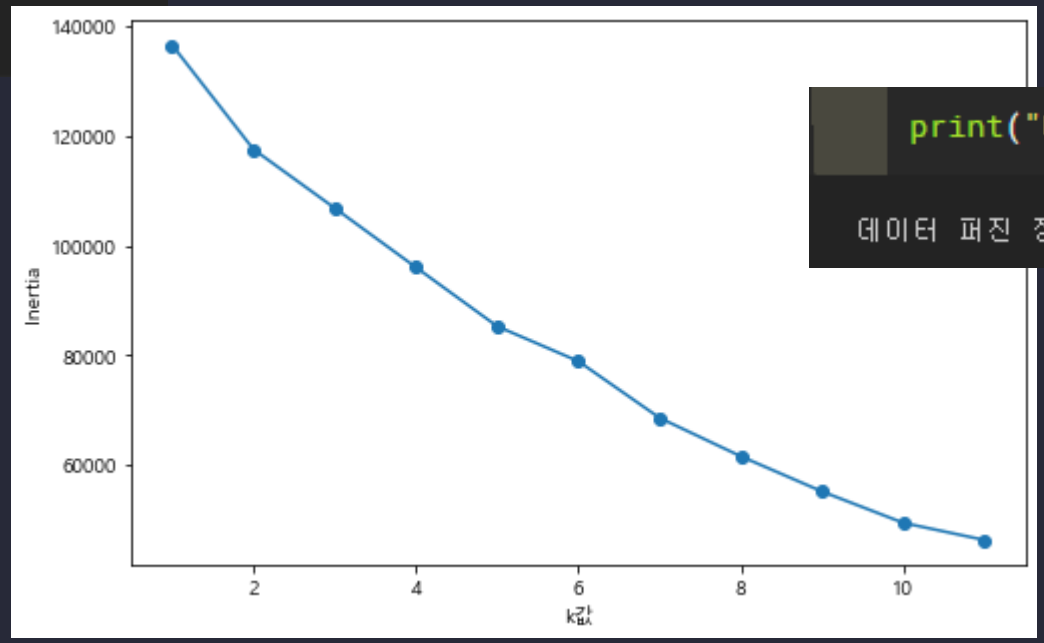
	설명가능한 분산 비율(고윳값)	기여율	누적기여율
pca1	2.99	0.20	0.20
pca2	1.47	0.10	0.30
pca3	1.24	0.08	0.38
pca4	1.18	0.08	0.46
pca5	1.07	0.07	0.53
pca6	1.02	0.07	0.60
pca7	0.99	0.07	0.66
pca8	0.90	0.06	0.72
pca9	0.85	0.06	0.78
pca10	0.79	0.05	0.83
pca11	0.78	0.05	0.89
pca12	0.68	0.05	0.93
pca13	0.62	0.04	0.97
pca14	0.27	0.02	0.99
pca15	0.14	0.01	1.00

군집 : K-Means, PCA

추가로 진행.

```
kmeans1 = KMeans(n_clusters=3)
kmeans1.fit(pca_df1)
```

```
KMeans(n_clusters=3)
```




```
print("데이터 퍼진 정도 : {0:.3f}".format(kmeans1.inertia_))
```

```
데이터 퍼진 정도 : 46342.557
```

군집 : K-Means, PCA

```
print("데이터 퍼진 정도 : {0:.3f}".format(kmeans.inertia_))
```

```
데이터 퍼진 정도 : 123072.675
```

스케일만 진행된 전체 피처

```
print("데이터 퍼진 정도 : {0:.3f}".format(s_kmeans.inertia_))
```

```
데이터 퍼진 정도 : 133913.844
```

스케일만 진행된 전체 피처에서 k값만 조절.
=> 더 증가

```
print("데이터 퍼진 정도 : {0:.3f}".format(kmeans1.inertia_))
```

```
데이터 퍼진 정도 : 46342.557
```

그나마 가장 나은 성능을 보임

군집 : DB-SCAN

DBSCAN, 밀도 기반 클러스터링(Density-based spatial clustering of applications with noise)

- 세밀하게 몰려 있어서 밀도가 높은 부분을 클러스터링 하는 방식

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(min_samples=5)
dbs_pred = pd.DataFrame(dbscan.fit_predict(df_sc))
dbs_pred.columns=['predict']

df_dbs = pd.concat([XX, dbs_pred],axis=1)
df_dbs
```

```
df_dbs["predict"].value_counts()
```

```
-1      8831
 41      202
 12      187
 9       163
 16      152
        ...
 60        4
 40        4
 109       4
 101       3
 88        3
Name: predict, Length: 115, dtype: int64
```

```
pd.crosstab(Y, df_dbs['predict'])
```

predict	-1	0	1	2	3	4	5	6	7	8	...	104	105	106	107	108	109	110	111	112	113
pred																					
0	2481	3	6	13	16	2	9	11	15	0	...	0	0	0	0	0	0	0	0	0	1
1	6350	3	0	1	46	8	142	8	8	50	...	8	6	8	6	6	4	5	5	10	4

2 rows × 115 columns

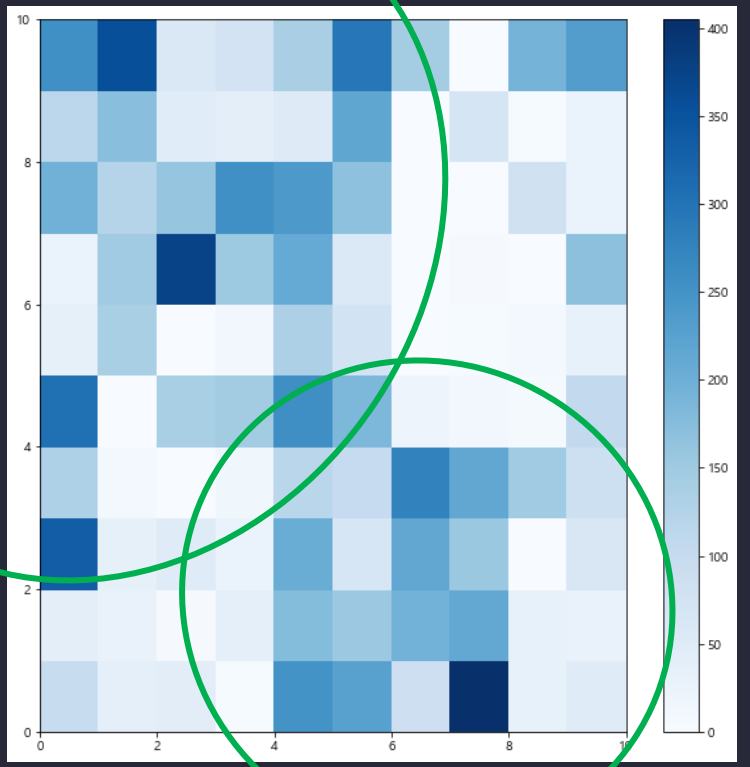
너무 많이 군집화됨

군집 : SOM

SOM, 자기조직화 지도 (Self-Organizing Maps)

- 비지도 신경망으로 고차원의 데이터를 이해하기 저차원으로 정렬하여 지도 형태로 형상화
- 입력 변수의 위치 관계를 보존함

```
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 14, sigma = 1, learning_rate = 0.5)
som.random_weights_init(df_sc)
som.train_random(data = df_sc, num_iteration = 100)
```



이상데이터 확인

```
mappings = som.win_map(df_sc)
frauds = np.concatenate((mappings[(1,4)], mappings[(2,9)]), axis = 0)
frauds = scaler.inverse_transform(frauds)
frauds.shape
```

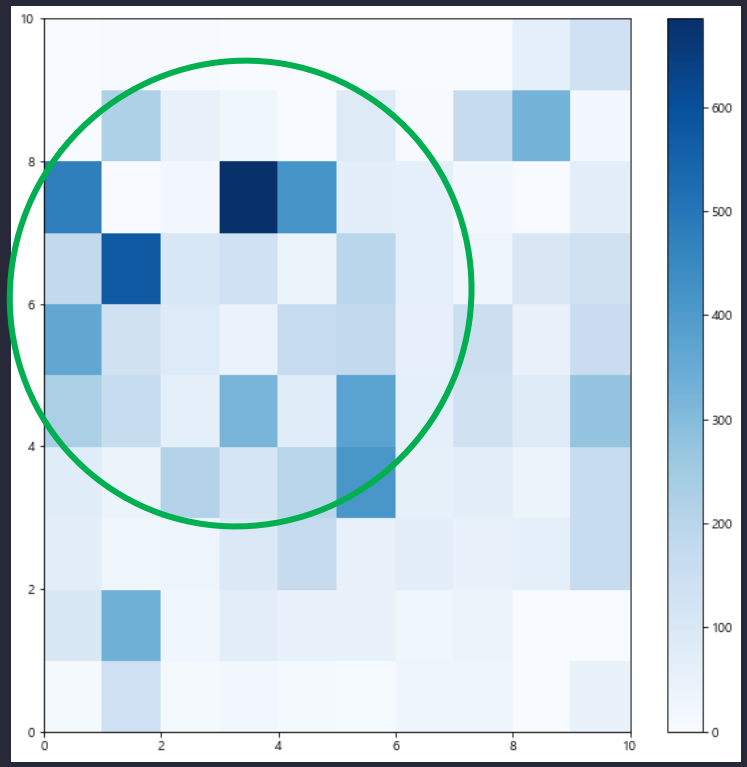
```
(179, 14)
```

군집 : SOM

pca적용

```
som1 = MiniSom(x = 10, y = 10, input_len = 14, sigma = 1, learning_rate = 0.5)
som1.pca_weights_init(df_sc)
som1.train_random(data = df_sc, num_iteration = 100)
```

```
plt.figure(figsize=(10, 10))
frequencies = np.zeros((10, 10))
for position, values in som1.win_map(df_sc).items():
    frequencies[position[0], position[1]] = len(values)
plt.pcolor(frequencies, cmap='Blues')
plt.colorbar()
plt.show()
```



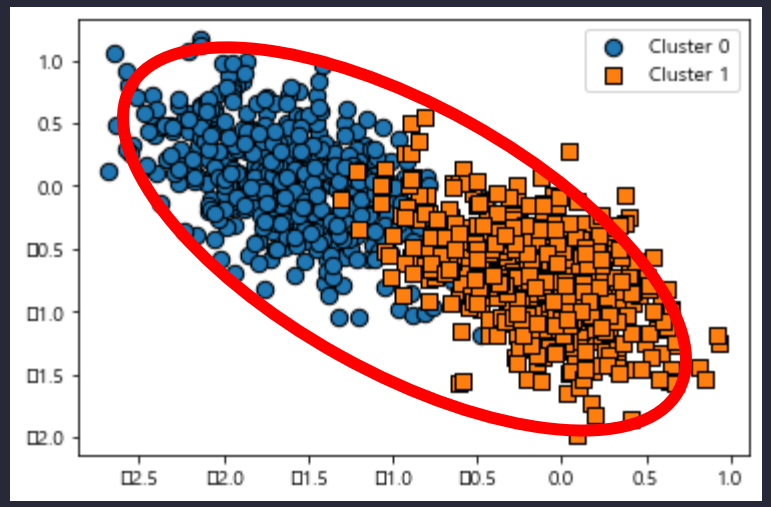
이상데이터 확인

```
mappings = som1.win_map(df_sc)
frauds = np.concatenate((mappings[(1,4)], mappings[(2,9)]), axis = 0)
frauds = scaler.inverse_transform(frauds)
frauds.shape
```

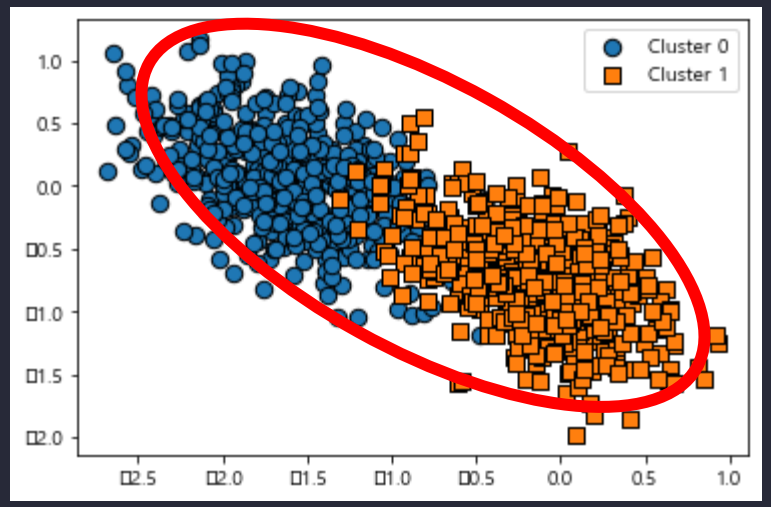
```
(217, 14)
```

군집 : SOM

랜덤으로 했을 때



PCA를 했을 때



두 그래프 모두 서로 얽혀있고 대각선 상에 있음

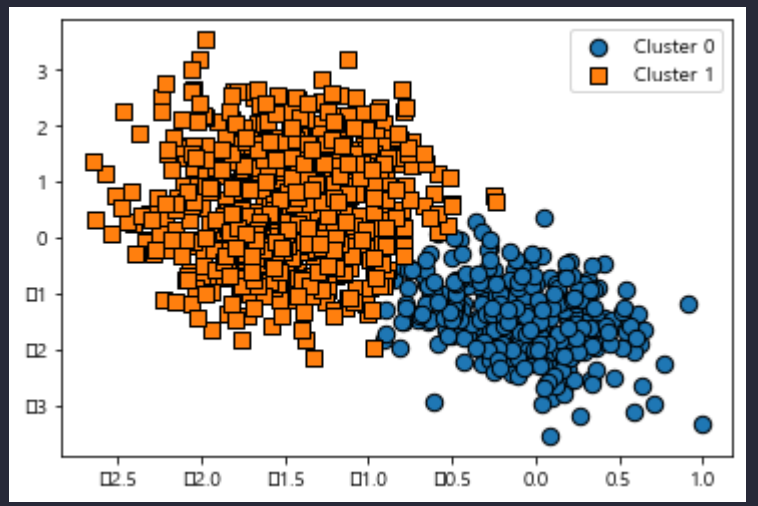
군집 : GMM

GMM, 가우시안 혼합모델(Gaussian Mixture Model)

-각각의 분포에 속할 확률이 높은 데이터끼리 클러스터링

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components = 2, random_state = 0)
gmm.fit(df_sc)
gmm_cluster_labels = gmm.predict(df_sc)

XX['gmm_cluster'] = gmm_cluster_labels
XX['target'] = Y

XX1 = XX.groupby(['target'])['gmm_cluster'].value_counts()
XX1
```

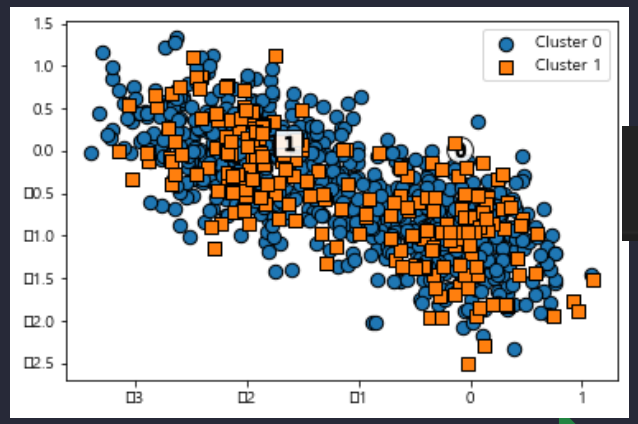
```
target  gmm_cluster
0       0              2536
        1               288
1       0              7284
        1               813
Name: gmm_cluster, dtype: int64
```



군집 알고리즘 정리

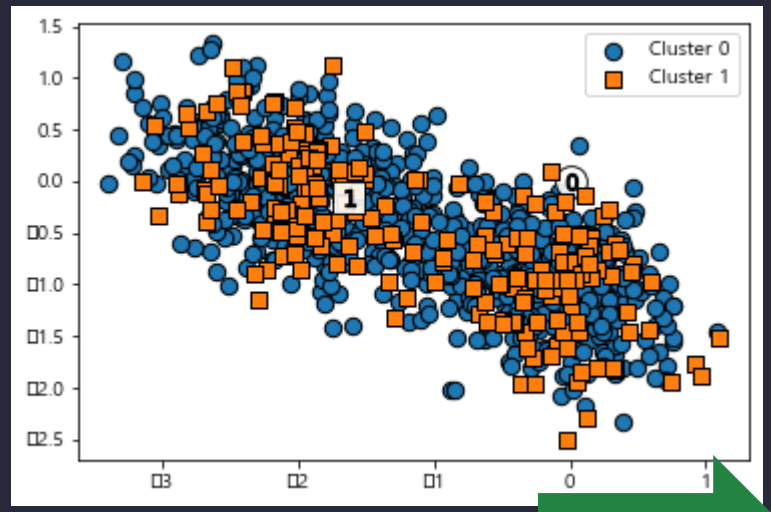
대체로 군집화가 안됨.

K-Means 관련 추가



1. 스케일만 진행된 전체 피처
=> 서로 얽혀있음

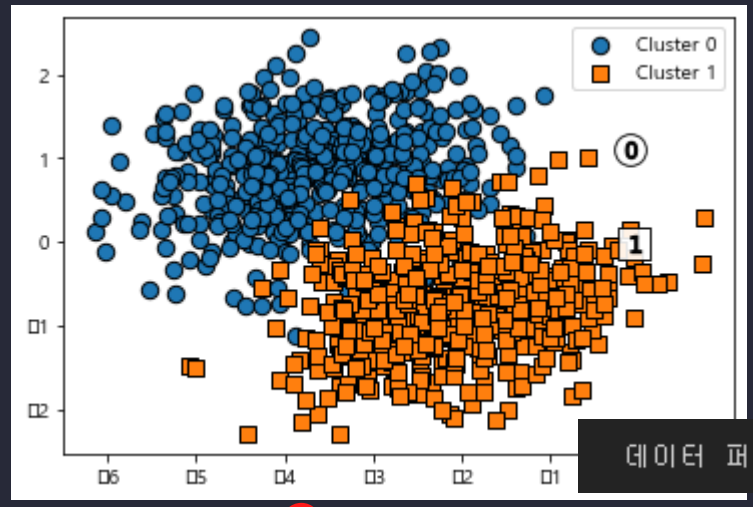
```
데이터 퍼진 정도 : 123072.675
```



2. 스케일만 진행된 전체 피처에서 k값만 조절.
=> 비슷하게 복잡

```
데이터 퍼진 정도 : 133913.844
```

3. 주성분분석 후 k값 조절
그나마 가장 나음 . 하지만 붙어있음




```
데이터 퍼진 정도 : 46342.557
```

결론1. 알고리즘

분류

- KNN, LDA, SVM 진행
- 오버 샘플링의 영향으로 진행해서 과적합이 발생
- 그나마 셋 중 SVM의 성능이 가장 나았음

종합 : SVM > KNN > LDA

군집화

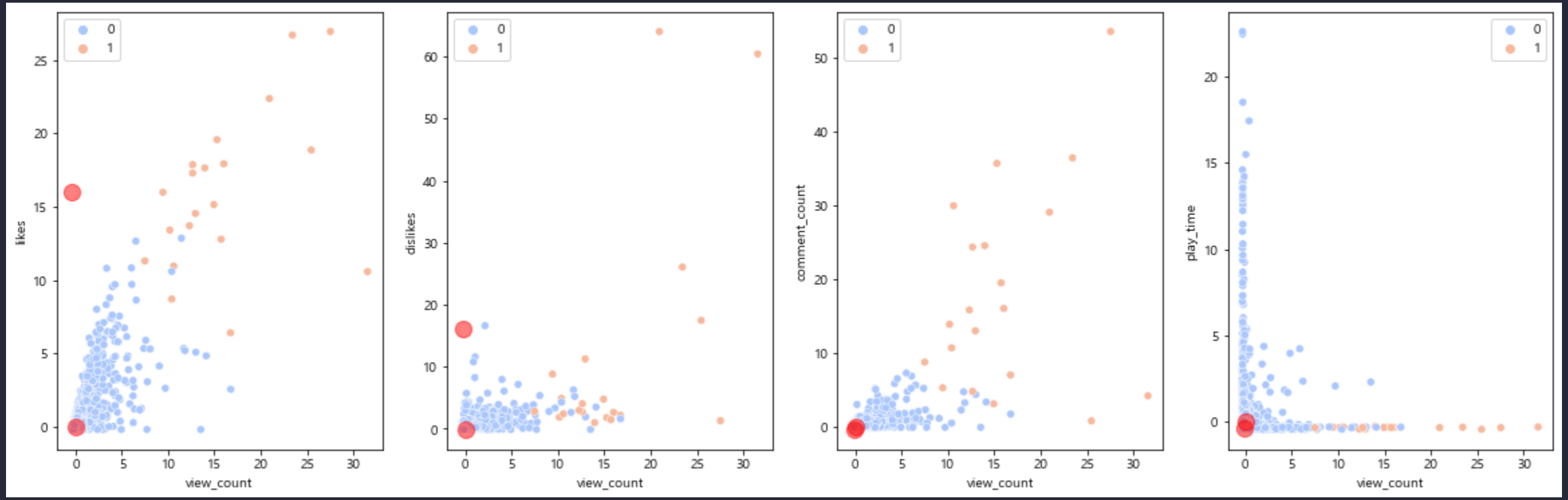
- K-means, PCA, DB-SCAN, SOM, GMM 진행
- 확률분포기반인 GMM, 거리기반 K-Means의 성능이 상대적으로 좀 더 좋았음
- 밀도기반인 DB-SCAN은 너무 안 맞았음
- SOM은 PCA와 같이 차원축소에 유용한 알고리즘이라 두 알고리즘 모두 데이터와 맞지 않았음

종합 : K-Means > GMM > SOM > DB-SCAN

결론2. 피처 간의 분석

#조회수와 관련되어 k-means 분석

조회수와 좋아요수 관계 | 조회수와 싫어요수 관계 | 조회수와 댓글수 관계 | 조회수와 재생시간 관계



= 군집분석이랑 안 맞는 데이터였다고 생각. 각 특성의 상관관계를 제외한 다른 인과관계는 없다고 생각.

활용방안

좀 더 데이터 양이 많으면 과적합 문제가 해결되기 때문에 더 다양한 분석 가능

유튜버라는 직업이 새로 나오는데 이를 서포터할 데이터 분석가가 있다면 큰 도움 가능

현재 데이터는 많지만 분석에 뛰어드는 사람은 거의 없는 분야

아쉬운점

관련 데이터의 종류가 적어서 직접 수집해야하는 부분이 많았다.

- 좀 더 다양한 데이터가 있었으면 더 좋은 결과가 나왔을 것이다.

비슷한 시계열 데이터 중 주식과 관련된 데이터는 넘치고 기록이 많지만 데이터가 시계열 데이터고 기록이 남지 않아 과거의 데이터를 찾을 수 없었다.

프로젝트를 진행하면서 방향설정에 대해 고민하느라 시간을 많이 놓쳐서 좋은 분석을 많이 놓쳐서 아쉽다.

감사합니다.