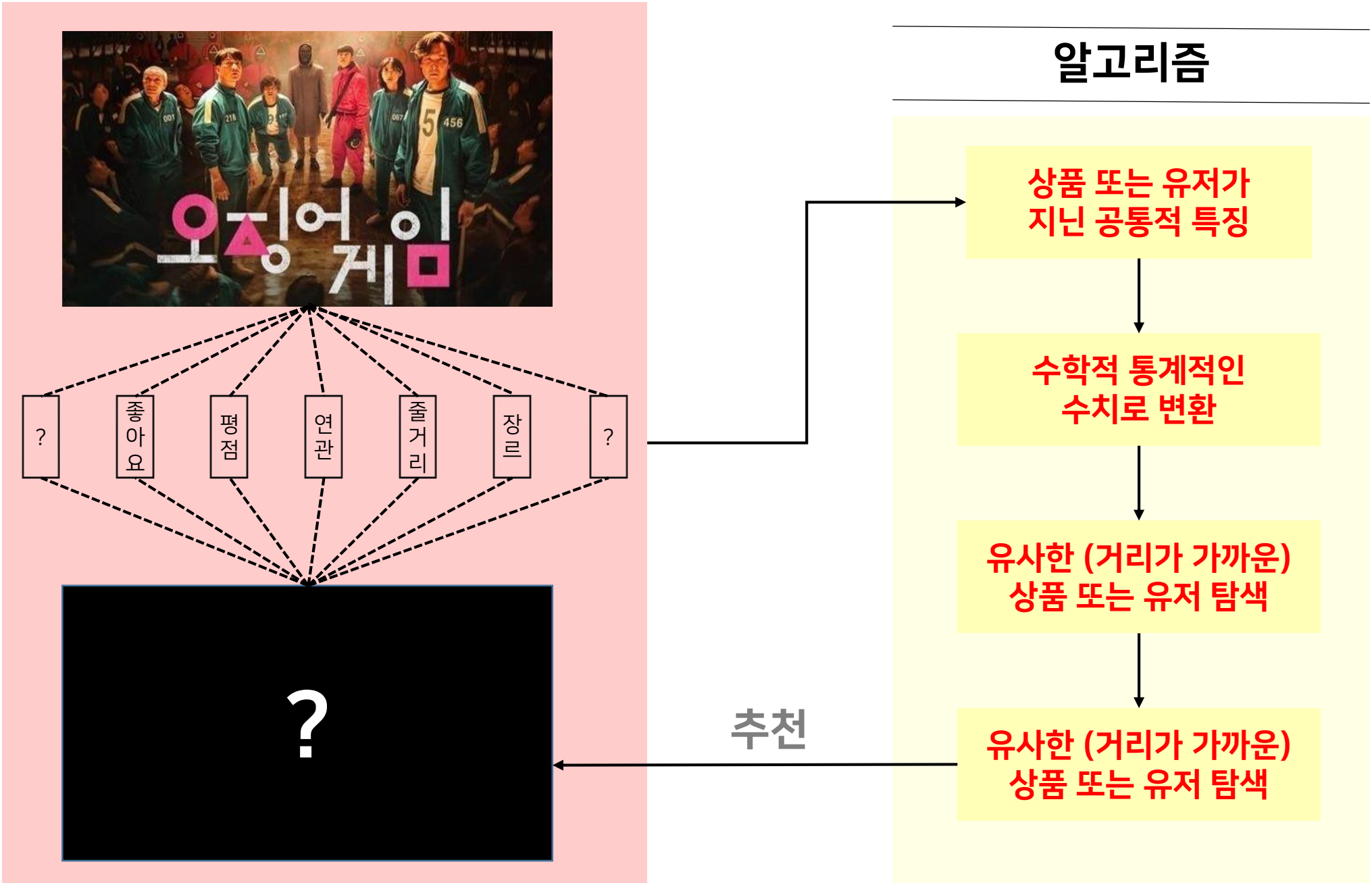


추천 시스템이란?



추천 알고리즘 트렌드

추천 알고리즘은 Hadoop Ecosystem과 함께 발전한 추세를 보임



Apriori 연관규칙 알고리즘

두 아이템 집합에 대한 일련의 규칙을 생성하는 알고리즘

- 지지도(support)
- 신뢰도(confidence)
- 향상도(lift)

수학과 통계에 기반한 알고리즘으로 아래와 같은 한계를 지님

- ① 연산 속도의 한계
- ② 상관성은 있으나, 인과성이 없음

협업 필터링(CF) / 콘텐츠 필터링(CB)

협업 필터링 (비슷한 유저 탐색)

유저 행동 정보를 분석하여 해당 유저와 비슷한 성향의 유저들이 좋아했던 상품을 추천

- Matrix Factorization
- K-Nearest Neighbor algorithm

콘텐츠 기반 필터링 (비슷한 상품 탐색)

유저 행동 정보를 분석하여 해당 유저와 비슷한 성향의 유저들이 좋아했던 상품을 추천

- TF-IDF
- Word2Vec

빅데이터 추천 시스템

Spark ML 활용한 알고리즘

- FP-growth algorithm
- Apriori (BigData Ver)
- ALS, SGD
- Matrix Factorization

계산 속도 및 BigData Scale 병렬 처리에 있어 효과적인 알고리즘

Recommendation System Trend

딥러닝 추천 시스템

α + 협업필터링

다양한 벡터화(Vectorization)와 협업 필터링(CF)을 결합한 알고리즘

- Doc2Vec + CF
- User2Vec + CF
- Item2Vec + CF

1. 프로젝트 개요

1) 프로젝트 선정

영화 추천 시스템 구축하기
Movie Recommendation System

2) 분석 데이터 설명

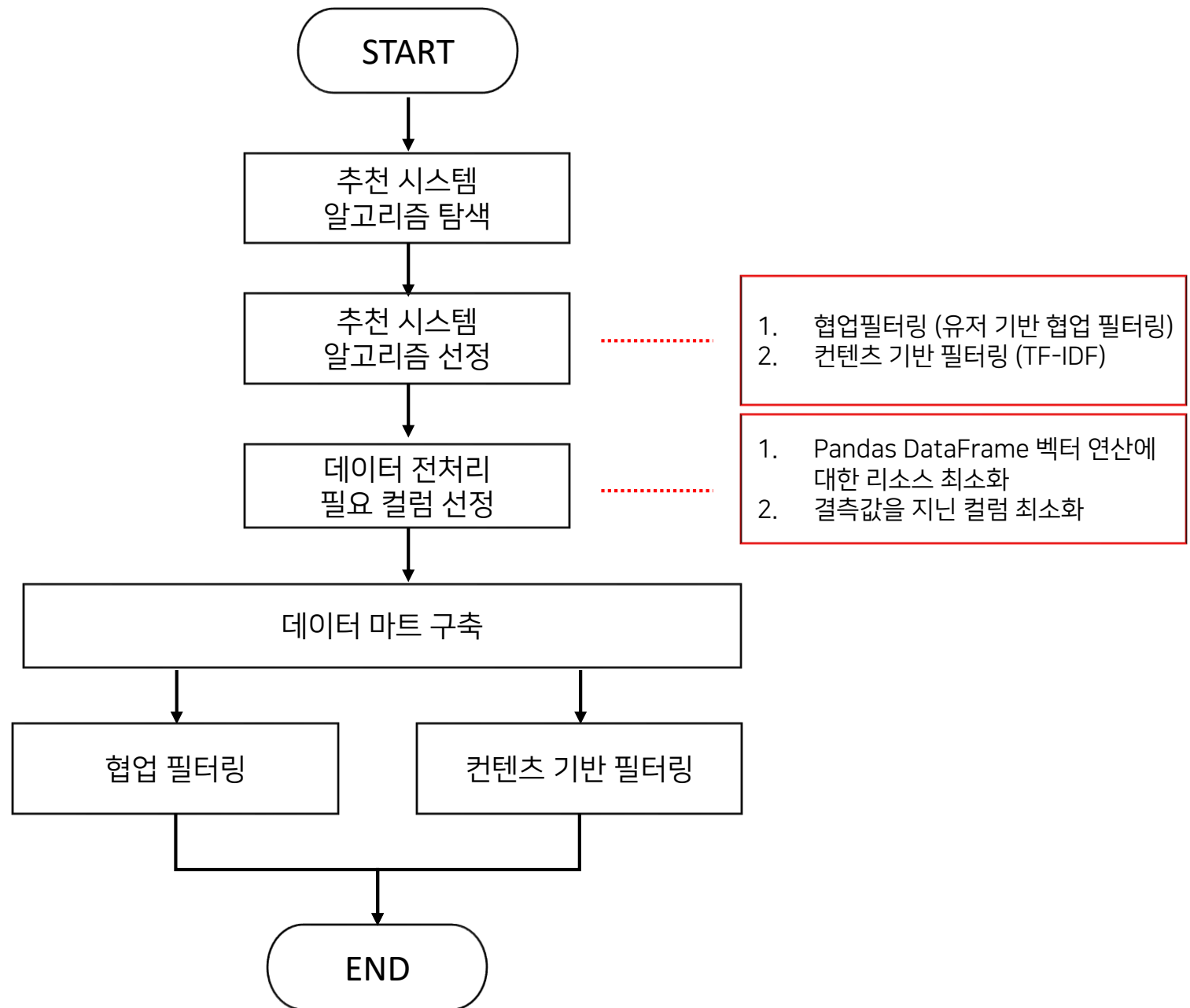
데이터	내용	크기
tmdb_5000_credits	영화 출연/조연출	(4803, 4)
tmdb_5000_movies	영화 정보	(4803, 19)
ratings.csv	소비자 평가	

3) 목표

- ① 영화 데이터 셋을 이용하여 추천 알고리즘 구현
(협업 필터링 & 콘텐츠 필터링)
- ② 알고리즘에 대한 수학적 개념 정리 및 소개

1. 프로젝트 개요

4) 분석 프로세스



2. 전처리

1) 문자열 전처리

- 문자열 결측치 "[]" => np.nan으로 대체

코드 -> 텍스트, 규격 통합

movie_id	cast	crew
43630	[]	[]

```
merge.replace('[]', np.nan, inplace=True)
```

movie_id	cast	crew
4401	43630	NaN

- 리스트 내부의 Key:Value 딕셔너리 => 컬럼을 대표할 수 있는 핵심 Value 만 추출 (대표성)

keywords	production_countries
[{"id": 1463, "name": "culture clash"}, {"id": ...}]	[{"iso_3166_1": "US", "name": "United States"}, {"id": ...}]

```
apply(lambda x: re.findall(r'"name":\s"(.+?)"', str(x)))
```

keywords	production_countries
[culture clash, future, space war, space colon...]	[United States of America, United Kingdom]

- 날짜 데이터 (string) => 년월일 사이 "-" 을 제거하고 (int)로 변환

0	2009-12-10
1	2007-05-19
2	2015-10-26
3	2012-07-16
4	2012-03-07
...	...
4798	1992-09-04
4799	2011-12-26
4800	2013-10-13
4801	2012-05-03
4802	2005-08-05

```
map(lambda x : int(x.replace('-', '')) if isinstance(x, str) else '')
```

0	20091210
1	20070519
2	20151026
3	20120716
4	20120307
...	...
4798	19920904
4799	20111226
4800	20131013
4801	20120503
4802	20050805

2. 전처리

2) 결측값 전처리

- 결측값 비율이 높은 컬럼 (homepage, tagline) 제거

```
[(item[0], item[1]) for item in merge.isnull().sum().items() if item[1] != 0]
```

✓ 0.4s

개수	비율
('cast', 43),	('cast', 0.008952737872163231),
('crew', 28),	('crew', 0.005829689777222569),
('genres', 28),	('genres', 0.005829689777222569),
('homepage', 3091),	('homepage', 0.6435561107641058),
('keywords', 412),	('keywords', 0.08577972100770352),
('overview', 3),	('overview', 0.0006246096189881324),
('production_companies', 351),	('production_companies', 0.0730793254216115),
('production_countries', 174),	('production_countries', 0.03622735790131168),
('release_date', 1),	('release_date', 0.00020820320632937748),
('runtime', 2),	('runtime', 0.00041640641265875496),
('spoken_languages', 86),	('spoken_languages', 0.017905475744326462),
('tagline', 844)]	('tagline', 0.17572350614199458)]

결측 비율 50% 이상 컬럼 제거

	title	특성 반영? X	tagline
0	Avatar	←	Enter the World of Pandora.
1	Pirates of the Caribbean: At World's End		At the end of the world, the adventure begins.

결측 비율 15% 이상, tagline이
상품 특성을 나타내기엔 모호성이 있음

('cast', 43),	출연진
('crew', 28),	조연출
('genres', 28),	장르
('keywords', 412),	키워드
('overview', 3),	줄거리
('production_companies', 351),	
('production_countries', 174),	
('release_date', 1),	
('runtime', 2),	
('spoken_languages', 86)]	

컬럼

컨텐츠 기반 필터링 (유사도)로 활용할 수 있으나 모든 영화의 주인공, 제작 인원을 벡터화 하는 것은 비효율적, 연산의 제약 발생가능성(메모리 에러)이 높음 **제거**

컨텐츠 기반 필터링 (유사도)로 활용, 영화를 범주화 할 수 있는 특징 **채택**

컨텐츠 기반 필터링 (TF-IDF)로 활용, 영화를 대표할 수 있는 Tagging 했기에 대표성이 있음. 하지만 전체 행수 대비 결측 비율이 높으므로 (8%), overview 에서 불용어 처리를 진행하여 keywords에 대한 결측값 보간을 진행 **채택**

평균 투표점수와의 산점도, 상관분석, 대상의 설명 변수로 부적절 판단 **제거**

2. 전처리

2) 결측값 전처리

- ① 알고리즘별 동일한 데이터 마트를 구축하기 위해 대체불가능한 컬럼을 중심으로 결측행 우선 제거 실행

```
merge.dropna(subset=['genres'], inplace=True)
```

 장르 결측행 제거

- ② Keywords와 Overview가 모두 결측 값인 행 (보간 불가능) 제거

```
merge = merge[~(merge.keywords.isnull() & merge.overview.isnull())]
```

- ③ Overview 전처리

```
merge['overview'] = merge['overview'].apply(lambda x: re.sub(r"[^a-zA-Z0-9]", ' ', x) if isinstance(x, str) else x) # 특수문자 제거  
merge['overview'] = merge['overview'].apply(lambda x: re.sub(r"\d+[a-zA-Z]+", ' ', x) if isinstance(x, str) else x) # (숫자 + 문자) 제거  
merge['overview'] = merge['overview'].apply(lambda x: re.sub(r"\d+", ' ', x) if isinstance(x, str) else x) # 숫자 제거
```

특수문자, 숫자(키워드가 아닌 단어) 제거

- ④ nltk (자연어 처리 패키지)의 불용어 사전을 이용하여 불용어 제거

```
merge['overview'] = merge['overview'].apply(lambda x: word_tokenize(x) if isinstance(x, str) else x)
```

토큰나이징(Tokenizing) : 문장을 쪼개는 행위

nltk 불용어 사전

```
['i',  
'me',  
'my',  
'myself',  
'we',  
'our',  
'ours',  
'ourselves',  
'you',  
"you're",
```

불용어 제거

merge['overview']

```
0 [In, century, paraplegic, Marine, dispatched, ...  
1 [Captain, Barbossa, long, believed, dead, come...  
2 [A, cryptic, message, Bond, past, sends, trail...  
3 [Following, death, District, Attorney, Harvey,...  
4 [John, Carter, war, weary, former, military, c...
```

merge['keywords']

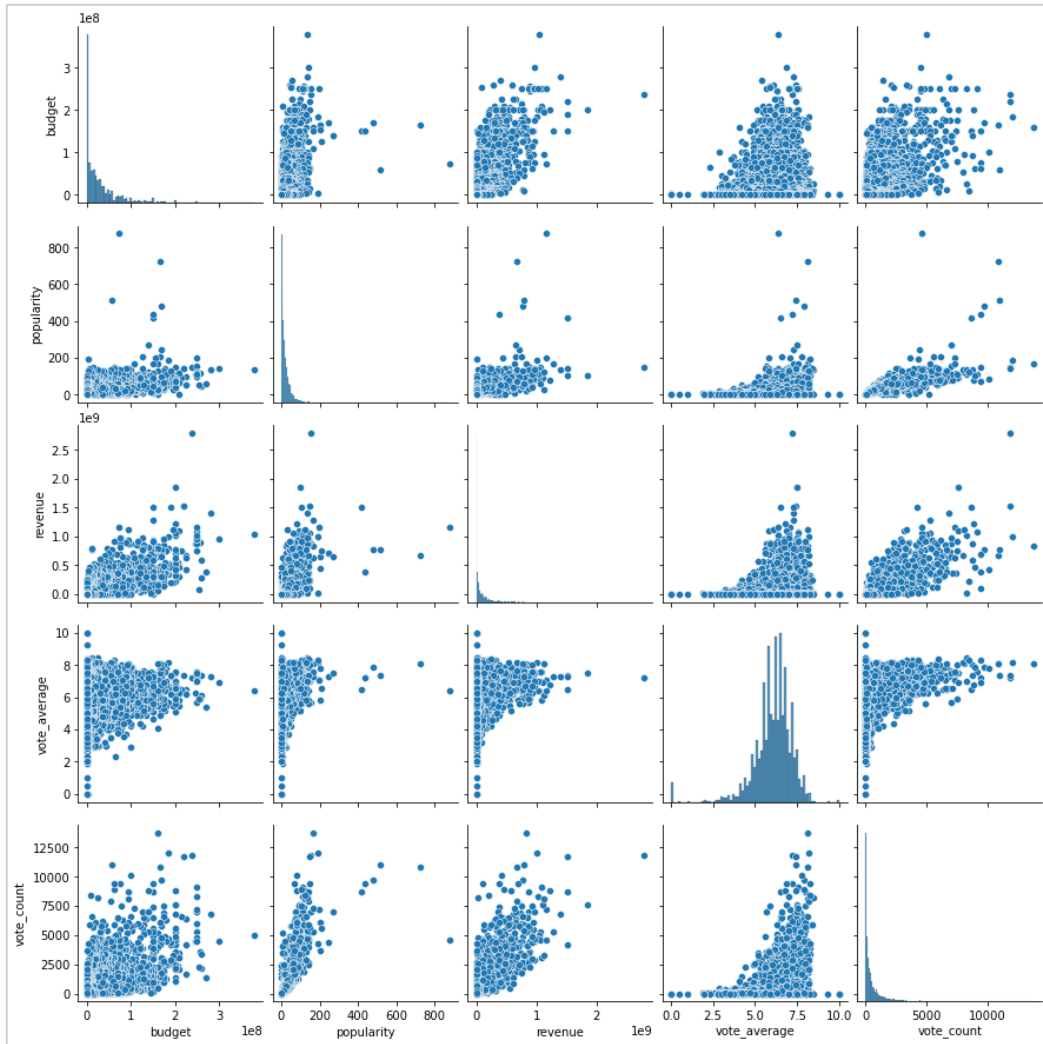
```
71 NaN  
83 NaN  
323 NaN  
381 NaN
```

결측치 보간

3. EDA

1) 데이터 분포

- 연속형 변수 간의 산점도(Scatter Plot)



- 추천에서의 만족도와 상관관계가 있을까?

① 만족도에 대한 정의

- (1) 평균 평점이 높은 영화는 만족도가 높은 것인가?
일반적으로 그러하다. 그럴 가능성이 높다. (기본전제)
- (2) 사람들이 많이 본 영화는 만족도가 높은 것인가?
그렇지 않다. (평균투표점수와 투표 수와의 pearson 상관관계 비교)

```
stats.pearsonr(merge.vote_average, merge.vote_count)  
✓ 0.5s  
(0.3129974039957597, 1.1909204293002478e-109)
```


4. 콘텐츠 기반 필터링

1)) What is Contents-based Filtering?

- 사용자가 **선택한 상품**과 **비슷한 콘텐츠(속성)**을 가진 **상품을 찾아 추천**하는 방법

- **Cold Start** 문제와 **데이터 희소성** 문제로부터 자유

* **Cold Start**

- 새로운 상품에 대한 유저들에 대한 충분한 정보가 수집된 상태가 아니라서 해당 유저들에게 제품을 추천해주지 못하는 문제

* **데이터 희소성** 문제

- 전체 상품 중 일부의 상품에만 평점을 부여하여, 사용자와 상품 관계를 정확히 이해하기 힘든 것 (sparse하다, 행렬 매트릭스에서 0이 많다)

2) Main Point

- **상품의 텍스트 콘텐츠를 어떻게 수학적으로 표현** 할 것인가?

3) 선정 알고리즘 : TF - IDF

- 여러 문서에서 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치

- 특징 추출(Feature Extraction)의 일종의 기법

$$TF - IDF = TF * IDF$$

TF : Term Frequency

단어가 한 문서에 얼마나 빈번하게 나오는가? (가중치)
한 문서에서 자주 나오는 단어는 중요 단어

IDF : Inverse-Document Frequency

모든 문서에서 빈번한(흔한) 단어 나오는가? (패널티)
모든 문서에서 자주 나오는 단어는 흔한 단어



We can go home home



I want to go now

TF

- Doc ① : We (1), can (1), go (1), home (2)
- Doc ② : I (1), want(1), to (1), go(1), now(1)

DF

- Doc ① ② : We (1), can (1), go (2), **home (1)**, I (1), want(1), to (1), now(1)

$$IDF (Ex: home) = \ln \left(\frac{1+N}{1+df} \right) + 1 = \ln \left(\frac{1+2}{1+1} \right) + 1$$

총 문서 개수 (N) → 단어의 DF (df)

TF-IDF 구현하기

1) Keywords (리스트) 요소를 결합하여 단일 문장으로 생성

```
def join_words(lst: list) -> list:
    try:
        result = ' '.join(lst)
    except:
        result = lst
    return result

merge['keywords'] = merge.apply(lambda x : join_words(x['keywords']), axis=1)
merge['keywords']
```

```
0      culture clash future space war space colony so...
1      ocean drug abuse exotic island east india trad...
2      spy based on novel secret agent sequel mi6 bri...
3      dc comics crime fighter terrorist secret ident...
4      based on novel mars medallion space travel pri...
...
4769   Adam security guard travels California Philipp...
4770   united states\u2013mexico barrier legs arms pa...
4771   A newlywed couple honeymoon upended arrivals r...
4772   date love at first sight narration investigati...
4773   obsession camcorder crush dream girl
```

2) sklearn.feature_extraction에서 TfidfVectorizer 활용하기

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(min_df=2, analyzer='word', stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(merge['keywords']) 문서 벡터화
```

```
tfidf_matrix.toarray()
0.85 벡터화된 단어 사전 인덱스 순서
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
1번 영화
2번 영화
3번 영화
```

```
sorted(tfidf.vocabulary_.items())
```

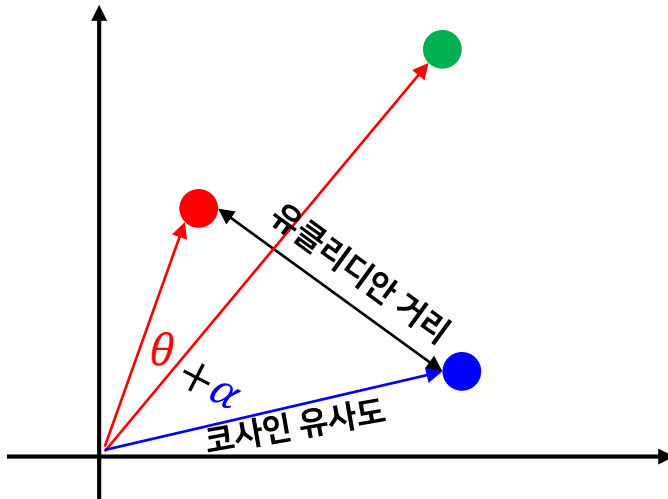
벡터화시킨 단어 사전

단어 → 인덱스

```
('abandoned', 23),
('abandonment', 24),
('abby', 25),
('abducted', 26),
('abduction', 27),
('ability', 28),
('able', 29),
('abolitionist', 30),
```

TF-IDF 구현하기

3) Cosine 유사도를 이용해 (N차원 상의 거리 또는 각도가) 유사한 대상 탐색하기



● 와 최단 거리에 있는 점은 ● 이다. (유클리디안 거리)

● 와 내각이 작은 점은 ● 이다. (코사인 유사도)



Why to use cosine similarity over Euclidean similarity?

A 문서에서 'home' 단어가 5번 나왔고,
B 문서에서 'home' 단어가 3번 나왔다.

A가 B보다 'home' 와 더 밀접한 문서라고 볼 수 있는가?


No. 두 문서의 길이 (문장 수)가 다를 수 있기 때문이다.
코사인 유사도는 문장의 수가 다를 때,
내각으로 유사함을 판단하여 가까움을 평가한다.

TF-IDF 구현하기

3) Cosine 유사도를 이용해 (N차원 상의 거리 또는 각도가) 유사한 대상 탐색하기

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)
pd.DataFrame(np.round(cosine_matrix, 4))
```

sklearn.metrics.pairwise의 cosine_similarity 불러오기



	0	1	2	3	4	5	6	7	8	9	10	11
0	1.000000	0.012500	0.000000	0.000000	0.226000	0.014700	0.024400	0.026300	0.000000	0.000000	0.000000	0.000000
1	0.012500	1.000000	0.000000	0.030600	0.000000	0.040700	0.011800	0.000000	0.000000	0.000000	0.000000	0.466800
2	0.000000	0.000000	1.000000	0.050700	0.041800	0.048000	0.017300	0.072200	0.000000	0.022100	0.077400	0.529600
3	0.000000	0.030600	0.050700	1.000000	0.000000	0.054800	0.025700	0.059500	0.000000	0.204500	0.126400	0.067000
4	0.226000	0.000000	0.041800	0.000000	1.000000	0.000000	0.044500	0.043300	0.000000	0.013400	0.132900	0.000000

1번 영화와 2번 영화의 코사인유사도
인덱스로 (0, 1) 행렬에 기록

```
title_idx_arr = pd.Series(range(0, len(merge)), index=merge.title)
idx = title_idx_arr["Pirates of the Caribbean: At World's End"]
```

```
tfidf_matrix.toarray()
0.8s
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

1번 영화
2번 영화

↓ 검색

```
[("Pirates of the Caribbean: Dead Man's Chest", 0.4667916407766871),
 ('Pirates of the Caribbean: The Curse of the Black Pearl',
 0.4023777905001075),
 ('Anna and the King', 0.28844958789136815),
 ('Cutthroat Island', 0.27792134737031793),
 ('Nim's Island', 0.21922127852887277),
 ('Swept Away', 0.21637837554577352),
 ('Shipwrecked', 0.19627413002229457),
 ('Pirates of the Caribbean: On Stranger Tides', 0.19072303531161638),
 ('Sahara', 0.18666820846908622),
 ('Half Baked', 0.18162205287789343)]
```