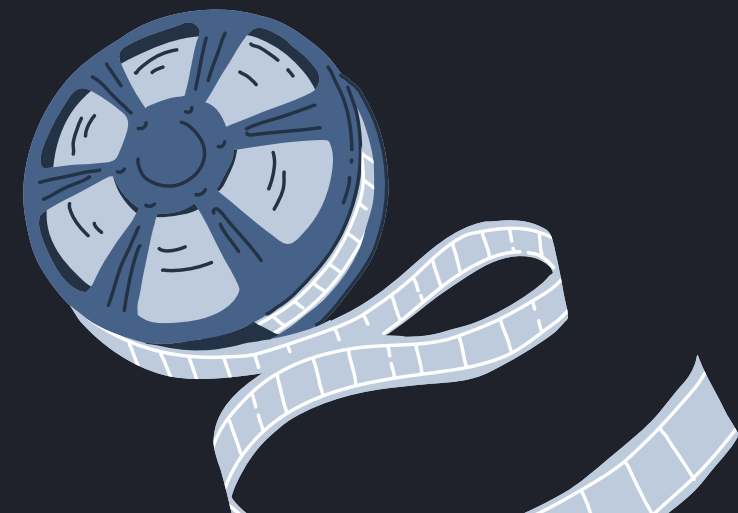
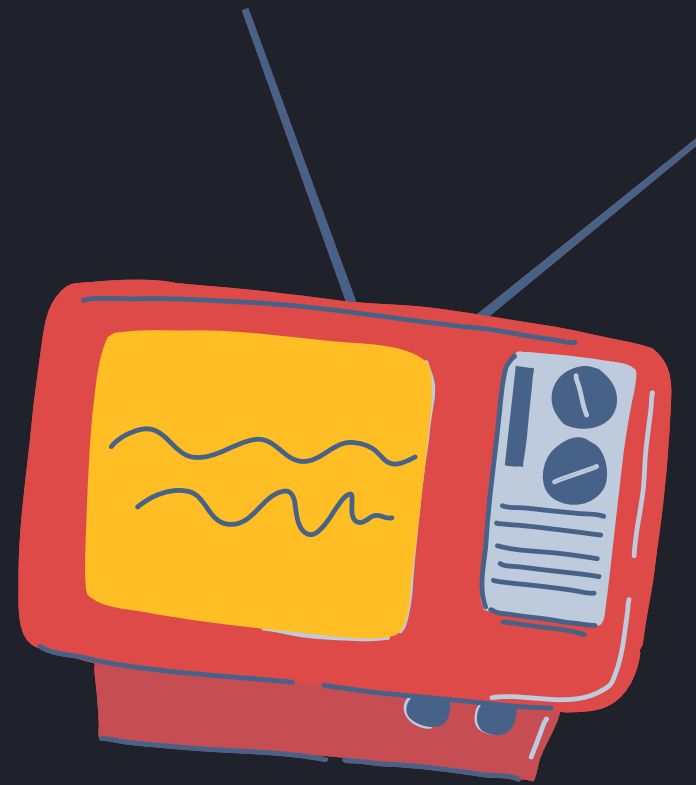
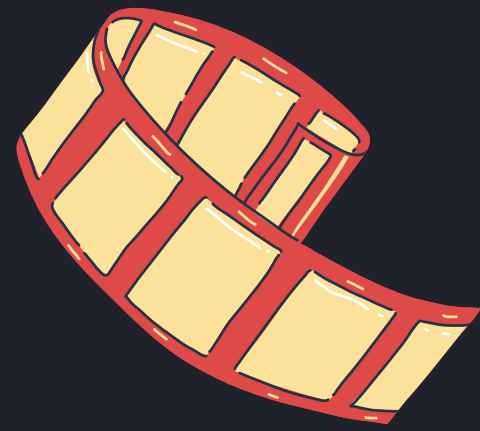


영화 추천 프로그램

KDT 5기 변주영



INDEX

1

주제 선정 이유, 데이터셋 소개

2

영화 추천 알고리즘 구현 과정

3

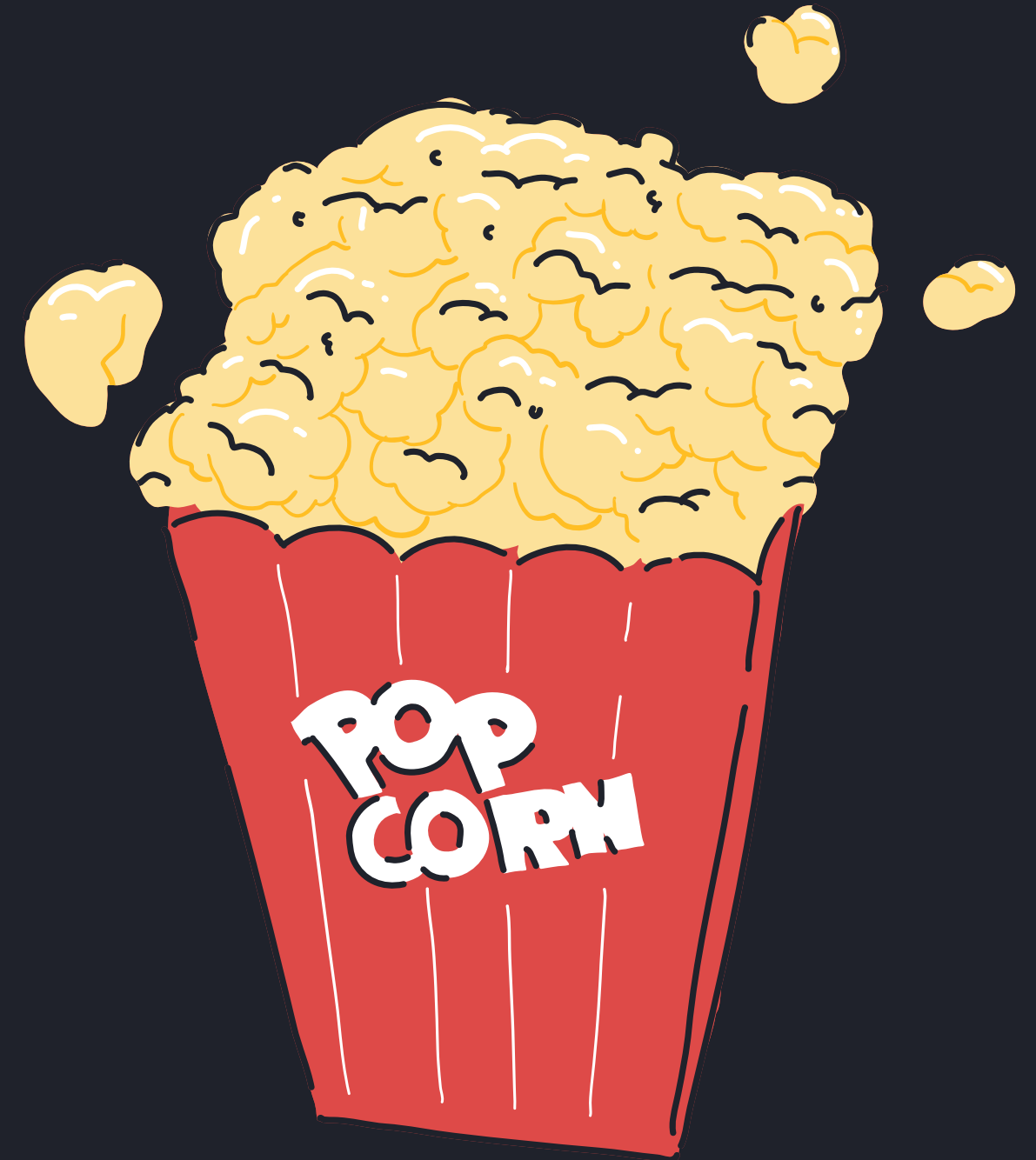
Flask 웹 서비스 개발 과정

4

Flask 웹 서비스 시연

5

느낀 점 & TODO



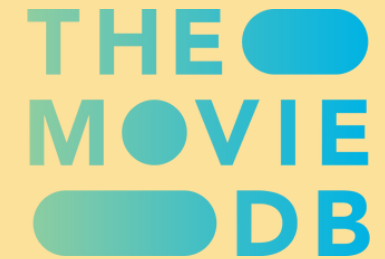
CATEGORY 1

주제 선정 이유, 데이터 소개

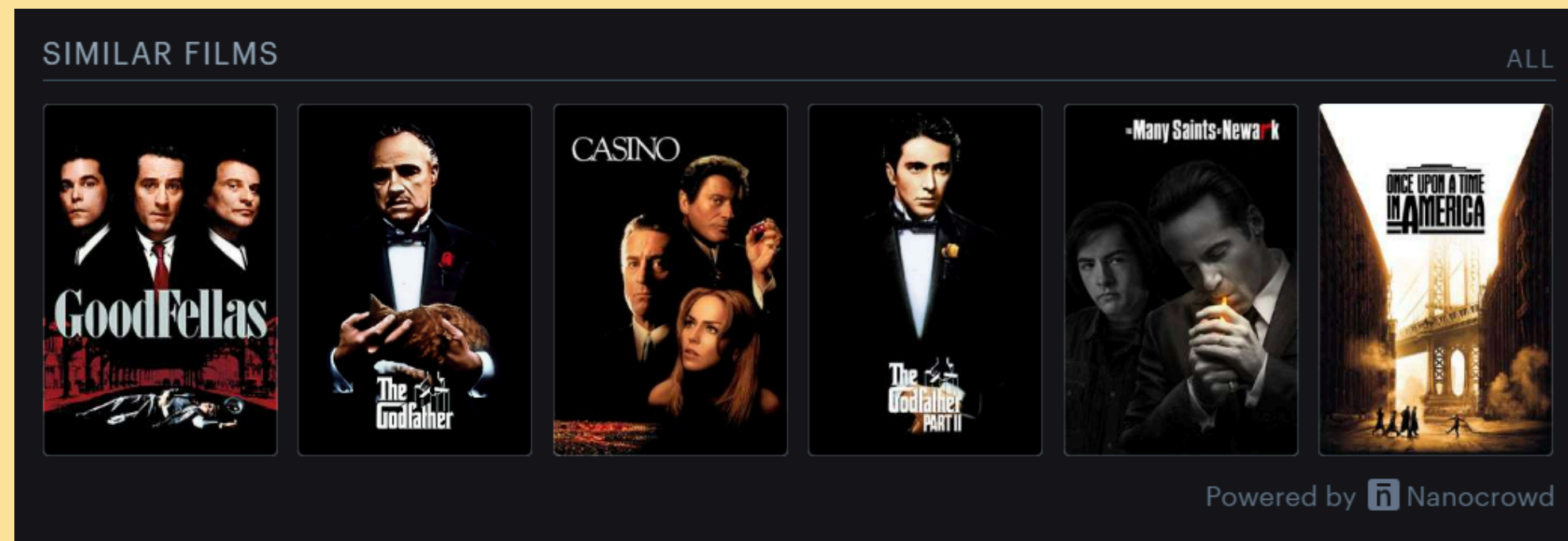
A brief overview

주제 선정 이유

영화를 매우 좋아하는 만큼,
영화 평가/기록 사이트에 많이 익숙하다!



이런 사이트들에서 공통적으로 지원하는 영화 추천 기능을 보면서
나도 만들어 보고 싶다는 생각을 쪽 해 왔었고, 이번 기회에 도전해보고 싶었다.



예시: Letterboxd

데이터셋 소개

TMDB 5000 Movie Dataset

Metadata on ~5,000 movies from TMDb



TMDB 5000 Movie Dataset | [Kaggle](#)

Full TMDB Movies Dataset 2024 (1M Movies)

Complete dataset containing movie data from TMDb. Updated Daily



Full TMDB Movies Dataset 2024 (1M Movies) | [Kaggle](#)

데이터셋 소개

TMDB 5000 Movie Dataset | Kaggle

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   movie_id    4803 non-null   int64
1   title       4803 non-null   object
2   cast        4803 non-null   object
3   crew        4803 non-null   object
dtypes: int64(1), object(3)
memory usage: 150.2+ KB
```

약 5000 편의 영화 데이터
마지막 업데이트 : 2017년
출연진, 연출진 정보 존재

Full TMDB Movies Dataset 2024 (1M Movies)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028972 entries, 0 to 1028971
Data columns (total 24 columns):
#   Column      Non-Null Count  Dtype
---  -
0   id          1028972 non-null int64
1   title       1028960 non-null object
2   vote_average 1028972 non-null float64
3   vote_count  1028972 non-null int64
4   status      1028972 non-null object
5   release_date 900122 non-null object
6   revenue     1028972 non-null int64
7   runtime     1028972 non-null int64
8   adult       1028972 non-null bool
9   backdrop_path 286782 non-null object
10  budget      1028972 non-null int64
11  homepage    111303 non-null object
12  imdb_id     578353 non-null object
13  original_language 1028972 non-null object
14  original_title 1028960 non-null object
15  overview    839370 non-null object
16  popularity  1028972 non-null float64
17  poster_path 735332 non-null object
18  tagline     147027 non-null object
19  genres      642901 non-null object
20  production_companies 484109 non-null object
21  production_countries 602888 non-null object
22  spoken_languages 614891 non-null object
23  keywords    293972 non-null object
dtypes: bool(1), float64(2), int64(5), object(16)
```

공통

— : 주요 컬럼

CATEGORY 2

영화 추천 알고리즘 구현 과정

추천 알고리즘 종류 3가지

1. Demographic Filtering

- 개인적인 추천이 아닌, 일반적인 추천.
- 더 유명할 수록, 더 평점이 좋을수록 상위권에 위치한다.

2. Content Based Filtering

- 특정 영화에 근거해서 비슷한 영화들을 추천.
- 메타데이터(줄거리, 장르, 배우, 감독 등)를 사용해서 추천한다.

3. Collaborative Filtering

- 비슷한 관심사를 가진 사람을 매칭해서 이 매칭에 기반하여 추천.
- 메타데이터를 많이 요구하지는 않는다.

Demographic Filtering

일반적인 추천을 어떻게 할 수 있을까?

- 평균 점수가 높으면 대다수가 좋아할 것이다.
- 그러나, 평균 점수는 높지만 평가수가 적은 경우는? 신뢰할 수 있을까?



왼쪽?
오른쪽?



Demographic Filtering

이를 해결하기 위한 방법 : 가중치 점수 (Weighted Rating)

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m} \cdot R \right) + \left(\frac{m}{v+m} \cdot C \right)$$

출처 : [Getting Started with a Movie Recommendation System](#) | Kaggle

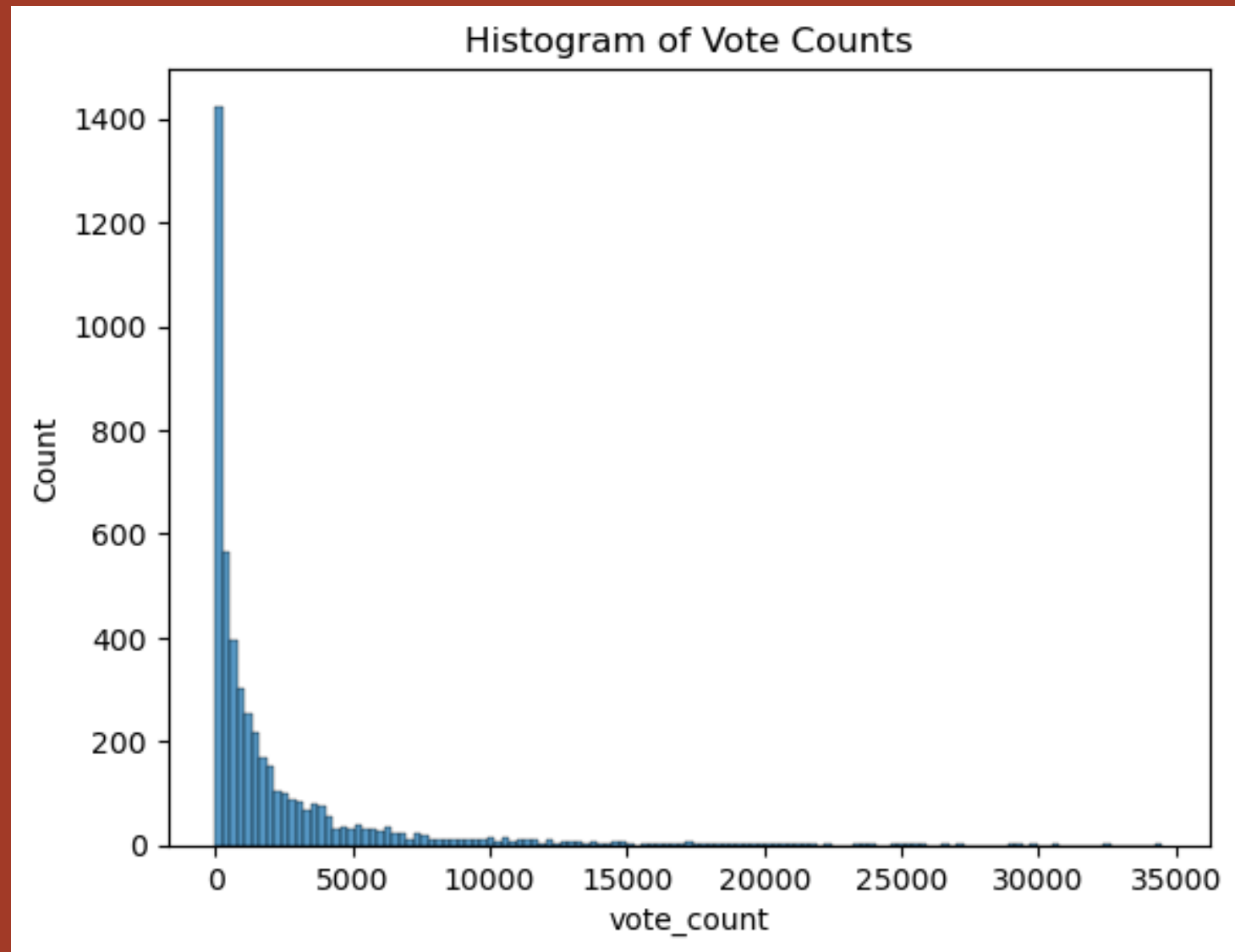
v : 해당 영화의 평가 수

m : 차트 진입 가능한 최소 평가 개수

R : 해당 영화의 평균 점수

C : 전체 영화들을 대상으로 산출한 평균 점수

Demographic Filtering



```
### m : 차트 진입 가능한 최소 평점수 (상위 25%까지)
m = movieDF['vote_count'].quantile(q=0.75) # 2456.25
```

```
### c : 전체 영화 평점 평균
C = movieDF['vote_average'].mean() # 6.3089376563803174
```

```
def weighted_rating(x, m=m, C=C):
    v = x['vote_count'] # 평점 개수
    R = x['vote_average'] # 각 영화의 평균 평점

    return (v / (v + m)) * R + (m / (v + m)) * C
```

가중치 점수를 기준으로 내림차순 정렬 → Top 100 영화 인덱스 저장

Content Based Filtering

한 영화를 기준으로 다른 영화를 추천하려면?

- 영화와 영화 간의 유사도를 측정해서 높은 값을 가진 영화를 추천!
- 유사도는 무엇을 기준으로 할까?

줄거리 기반

&

출연진, 연출진 기반

Content Based Filtering

코사인 유사도 (1) - 줄거리 기반

결측치 전처리

```
### contractions 라이브러리를 사용해 축약어 확장 실행
movieDF['overview'] = movieDF['overview'].apply(func=lambda x: contractions.fix(s=x))
✓ 0.0s
```

영어 축약어 전처리

```
### TF-IDF 행렬 생성 (=> 19823 단어 존재)
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(raw_documents=movieDF['overview']) # scipy.sparse._csr.csr_matrix
print(f'TF-IDF 행렬의 크기(shape) : {tfidf_matrix.shape}')
✓ 0.1s
```

TF-IDF 행렬의 크기(shape) : (4796, 19823)

TF-IDF 행렬 생성

```
### 메모리 부족 주의 !!! (MemoryError 주의)
cosine_sim = cosine_similarity(X=tfidf_matrix, Y=tfidf_matrix) # ndarray
print(f'코사인 유사도 연산 결과 : {cosine_sim.shape}')
✓ 0.1s
```

코사인 유사도 연산 결과 : (4796, 4796)

코사인 유사도 행렬 생성

Content Based Filtering

코사인 유사도 (2) - 출연진, 연출진 기반

출연진 정보 추출

- 각 영화 별 출연 배우 정보는 배역의 중요도 순으로 정렬되어 있음
- 배역 중요도가 높은 순서대로 3명씩 추출

연출진 정보 추출

- 연출진에 대한 정보 : 각 인원 별 이름, 직업(job)에 대한 정보가 존재.
- 내가 상정한 중요 연출진 : 감독 > 각본가 > 프로듀서, 촬영감독, ...
- 감독명부터 먼저 추출 (가장 중요하다고 생각하기 때문)
 - 감독이 나머지 역할까지 겸직하는 경우 처리
- 감독 제외한 나머지 중요 연출직군에서 1명씩 추출 (→ 약 3명)
- 추출한 이름 정규화 : 소문자화, 공백 제거

TF-IDF 행렬 생성

코사인 유사도 행렬 생성

Content Based Filtering

코사인 유사도 (2) - 출연진, 연출진 기반

출연진 정보 추출

연출진 정보 추출

TF-IDF 행렬 생성

코사인 유사도 행렬 생성

```
def get_crews(x: pd.Series):
    ## x는 movieDF의 한 행(Series)
    if not isinstance(obj/x, class_or_tuple/pd.Series):
        return ""

    crewDF = str_to_df(input_str=x['crew'])
    if crewDF.shape[0] == 0:
        ## 연출진 정보가 없다면 빈 문자열 반환
        return ""

    mask1 = crewDF['job'] == 'Director'
    directorList = [director for director in crewDF[mask1].name]

    mask2 = crewDF['job'].isin(values=['Writer', 'Producer', 'Screenplay', 'Executive Producer'])
    mask3 = crewDF['name'].isin(values=directorList)
    crewDF = crewDF[mask2 & ~mask3] # 감독을 제외한 나머지 연출진들
    crewDF.drop_duplicates(subset=['name'], inplace=True) # 중복된 인물 제거

    crew_num = len(obj/directorList) + crewDF.shape[0]

    if crew_num <= 3:
        ## 다 합쳐서 3명 이하일 경우 전체 반환
        directorList = [director.lower().replace(" ", "") for director in directorList]
        crewList = [name.lower().replace(" ", "") for name in crewDF['name']]
        return " ".join(iterable/directorList + crewList)

    ## 다 합쳐서 3명 보다 많을 경우 (감독 + 작가 1~2명 + 제작자 1~2명)
    crewDF.drop_duplicates(subset=['job'], inplace=True)
    directorList = [director.lower().replace(" ", "") for director in directorList]
    crewList = [name.lower().replace(" ", "") for name in crewDF['name']]
    return " ".join(iterable/directorList + crewList)
```


CATEGORY 3

FLASK 웹 서비스 구현 과정

Flask 작동 흐름

View File (rileyFilm_views.py)

- 두 가지 데이터셋을 병합한 영화 csv 파일 불러오기
- Top 100 차트에 해당하는 영화들의 인덱스 불러오기
- 코사인 유사도 행렬 불러오기 (2가지 종류)

Home Page
(homepage.html)



Top 100 Chart Page
(top100.html)

Movie Info Page
(show_movie_info.html)

구현 예정

Flask 페이지 구성 (1)

Home Page

 RileyFilm - 영화 추천 프로그램 

영화 제목 :

[Top 100 영화 리스트 보기](#)

Flask 페이지 구성 (2)

Top 100 Chart Page

[종합] Top 100 리스트

Ranking	title	year	genres	original_language	vote_average	vote_count
1	The Shawshank Redemption	1994	Drama, Crime	en	8.702	24649
2	The Godfather	1972	Drama, Crime	en	8.707	18677
3	The Dark Knight	2008	Drama, Action, Crime, Thriller	en	8.512	30619
4	Pulp Fiction	1994	Thriller, Crime	en	8.488	25893
5	Forrest Gump	1994	Comedy, Drama, Romance	en	8.477	25409
6	Interstellar	2014	Adventure, Drama, Science Fiction	en	8.417	32571
7	Fight Club	1999	Drama	en	8.438	27238
8	The Lord of the Rings: The Return of the King	2003	Adventure, Fantasy, Action	en	8.474	22334

Flask 페이지 구성 (3)

Movie Info Page

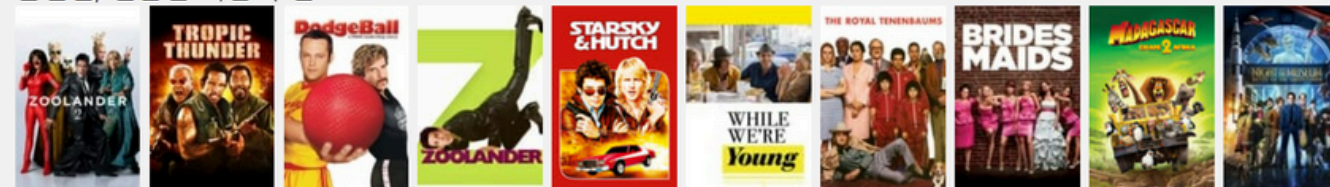


The Secret Life of Walter Mitty (2013) United States of America, United Kingdom

즐거리 기반 추천 :



출연진, 연출진 기반 추천 :



CATEGORY 4

FLASK 웹 사이트 시연

CATEGORY 5

느낀 점 & TODO

느낀 점

평소 궁금해 하던 가중치 평점 산출을 직접 수행해 보아서 좋았고,
내가 만든 차트가 실제 사이트와 비슷한 결과가 나오는 것이 신기했다.

TODO

사용자 리뷰, 키워드 등을 사용해서 추천 알고리즘을 더 발전 시키는 것
비슷한 관심사를 가진 유저 매칭 시스템 (Collaborative Filtering) 구현
더 넓은 영화 DB 구축, 더욱 유기적인 Flask 웹 사이트 구현

THE END

**THANK
YOU!**