MOVIE RECOMMENDATION

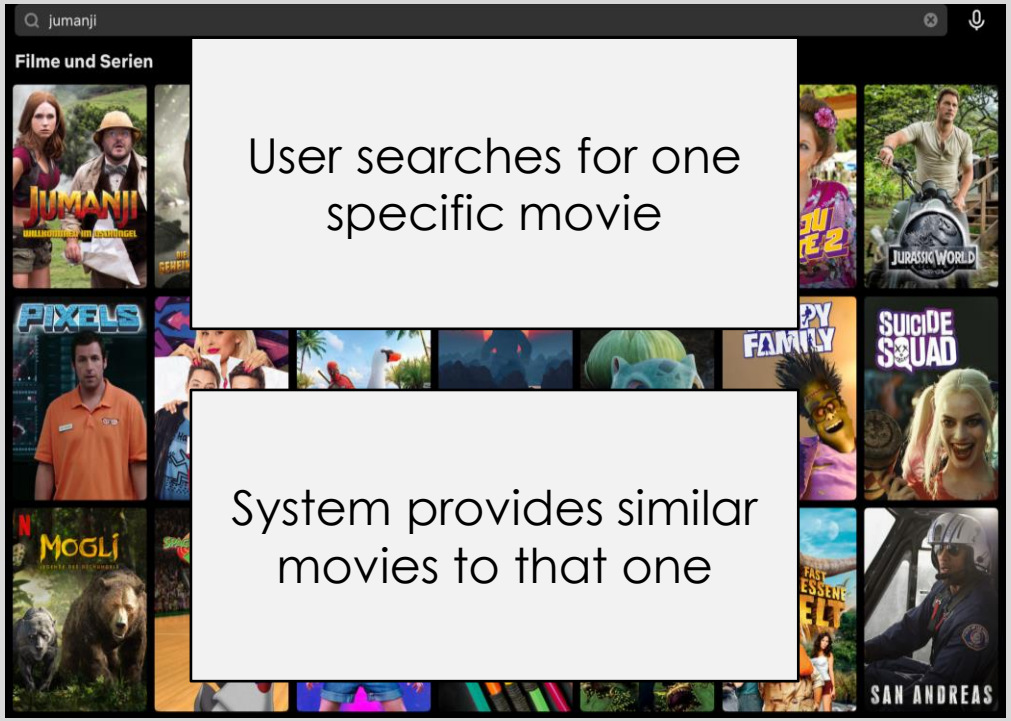
Data Exploration

13.07.2021

CONCEPT

Concept

movies get recommended in 2 different ways



User searches for one specific movie

System provides similar movies to that one

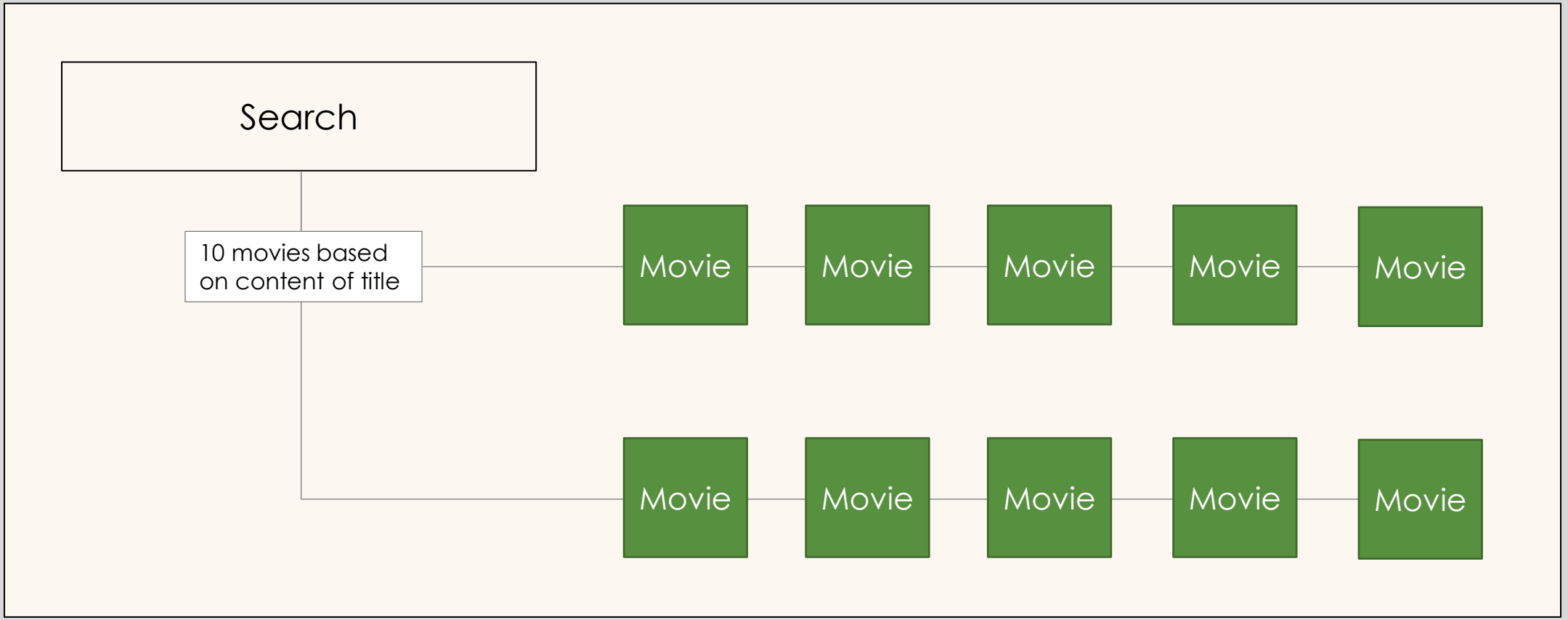
User watches movies

Enthalte
Film-Empfehlungen für Sie

System provides recommended movies in the background

Concept (1/2)

Recommendation by search result



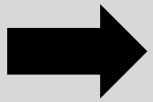
Concept (2/2)

Recommendation by user rating

User likes/dislikes movies

Movie 1	like	dislike
Movie 2	like	dislike
Movie 3	like	dislike
Movie 4	like	dislike
Movie 5	like	dislike

…



Recommendation

Movie Movie Movie Movie

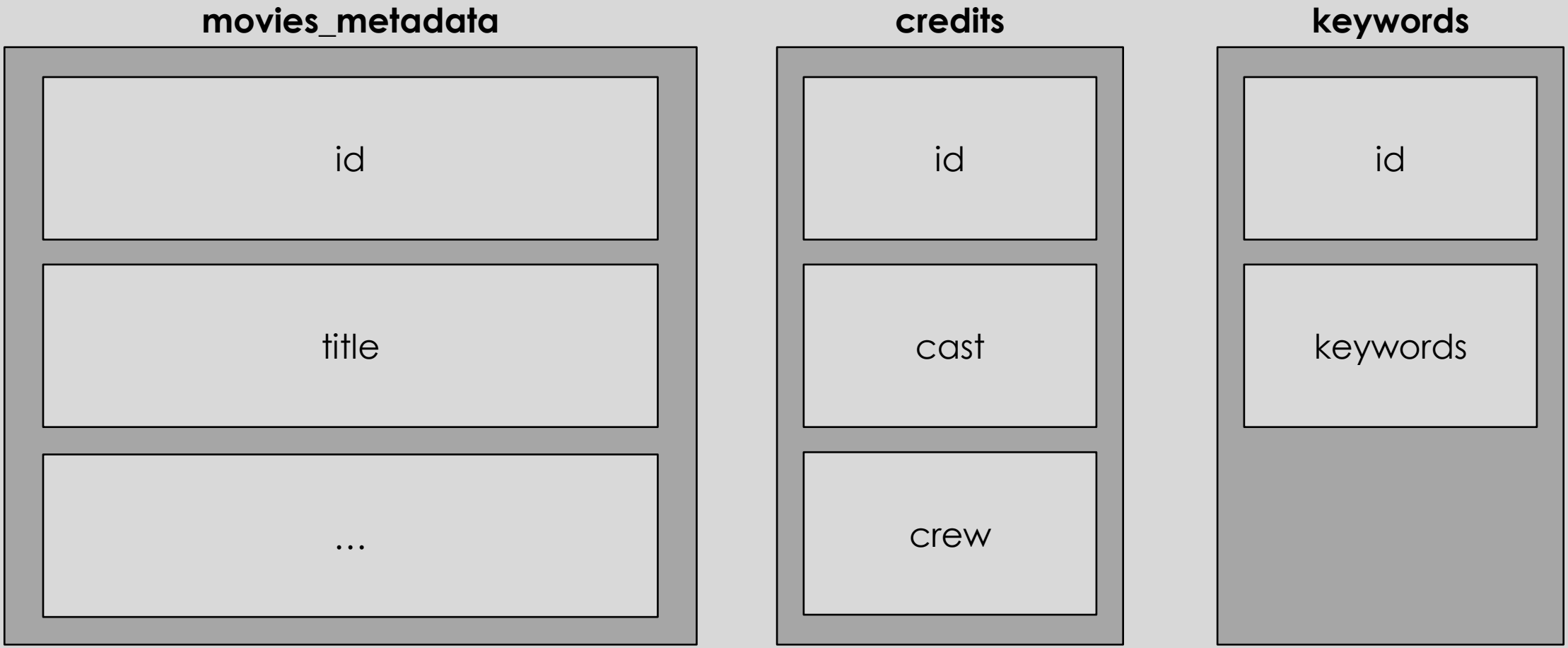
Movie Movie Movie Movie

…

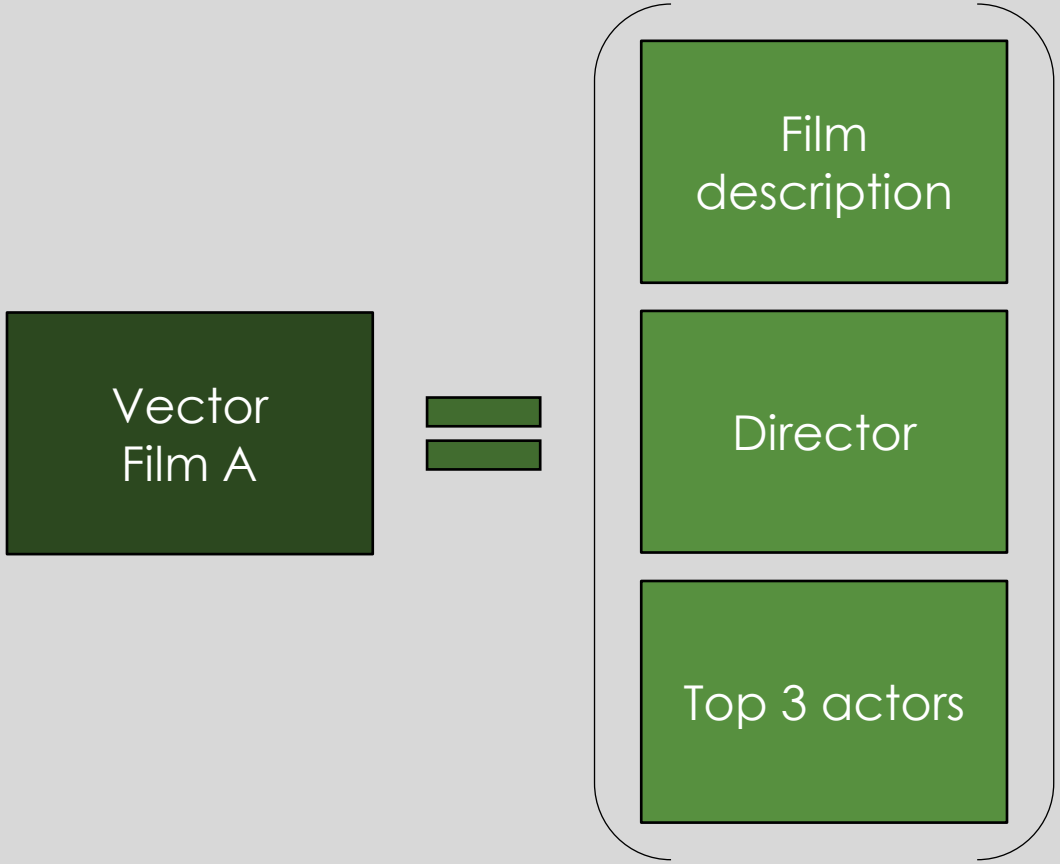
13.07.2021

CONTENT BASED

Dataset


movies_metadata
id
title
…
credits
id
cast
crew
keywords
id
keywords

Content-based recommendation

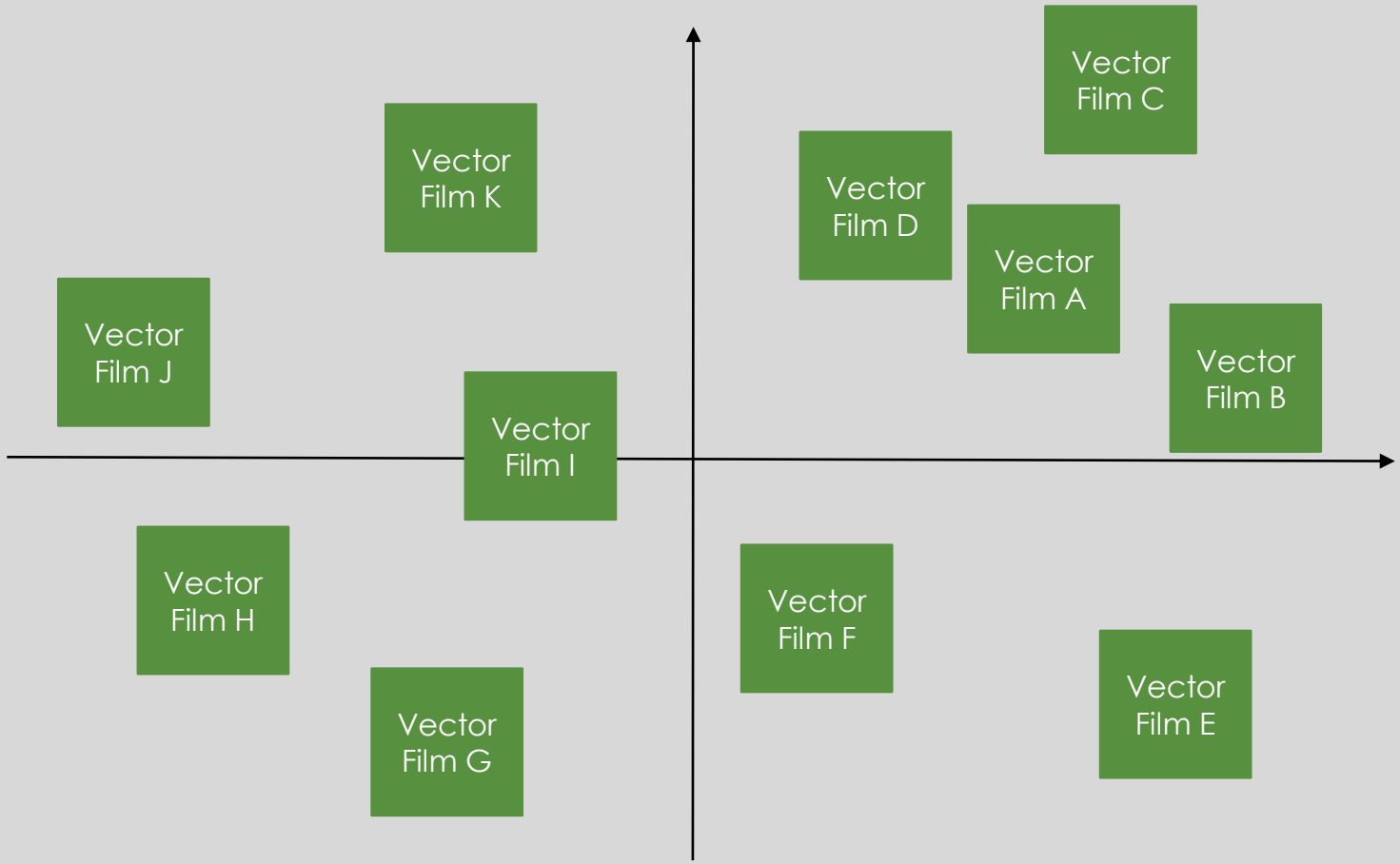


Attributes are joined as a lowercase string

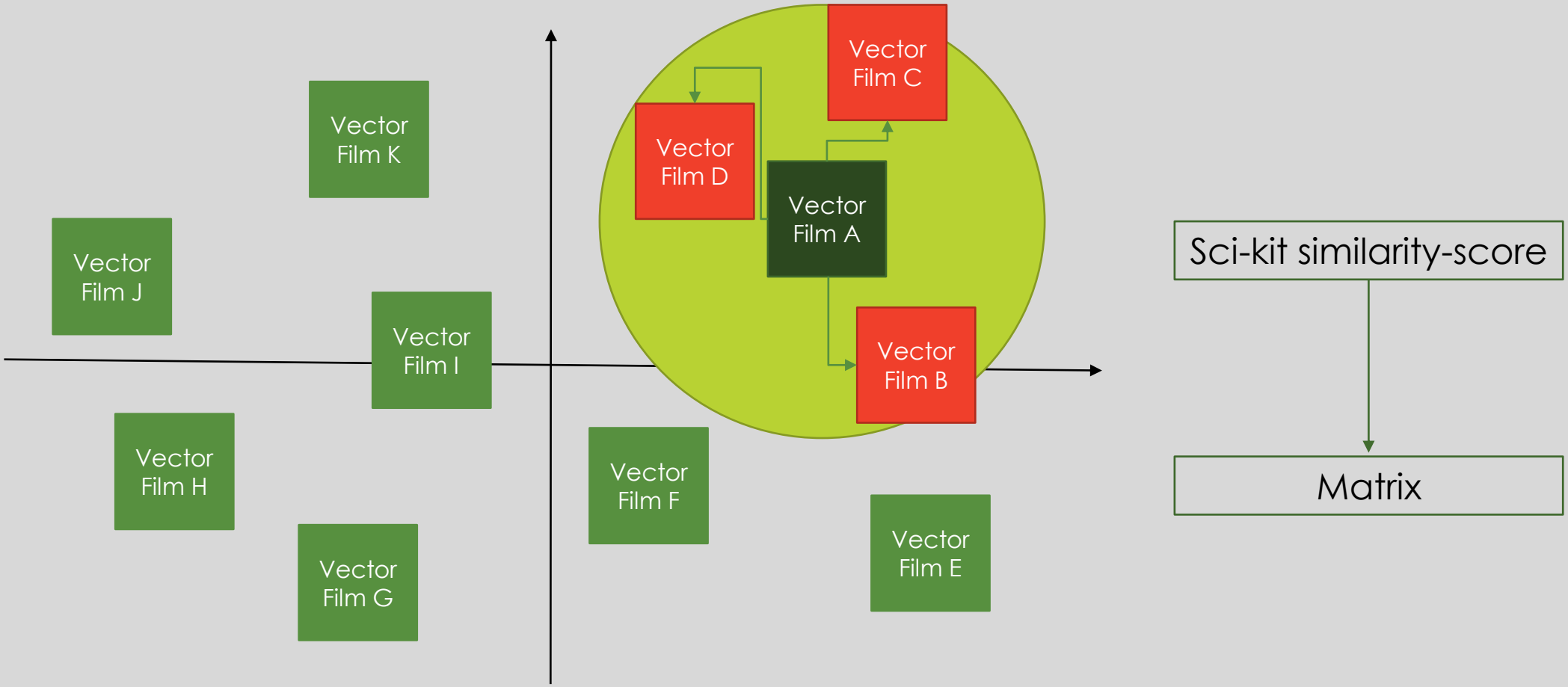
↓

Sci-kit CountVectorizer()

Content-based recommendation



Content-based recommendation



Content based - Packages

```
import numpy as np
```

```
import pandas as pd
```

```
from ast import literal_eval
```

```
import difflib
```

```
import sklearn
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

Content based – Datapreparation

```
metadata = metadata[~metadata.id.str.contains("-")]
```

```
metadata = metadata.merge(credits, on='id')
metadata = metadata.merge(keywords, on='id')
```

```
features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    metadata[feature] = metadata[feature].apply(literal_eval)
```

Content based – Datapreparation

```
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan

def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If no, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

```
metadata['director'] = metadata['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    metadata[feature] = metadata[feature].apply(get_list)
```

Content based – Datapreparation

```python
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''

# Apply clean_data function to your features.
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
    metadata[feature] = metadata[feature].apply(clean_data)
```

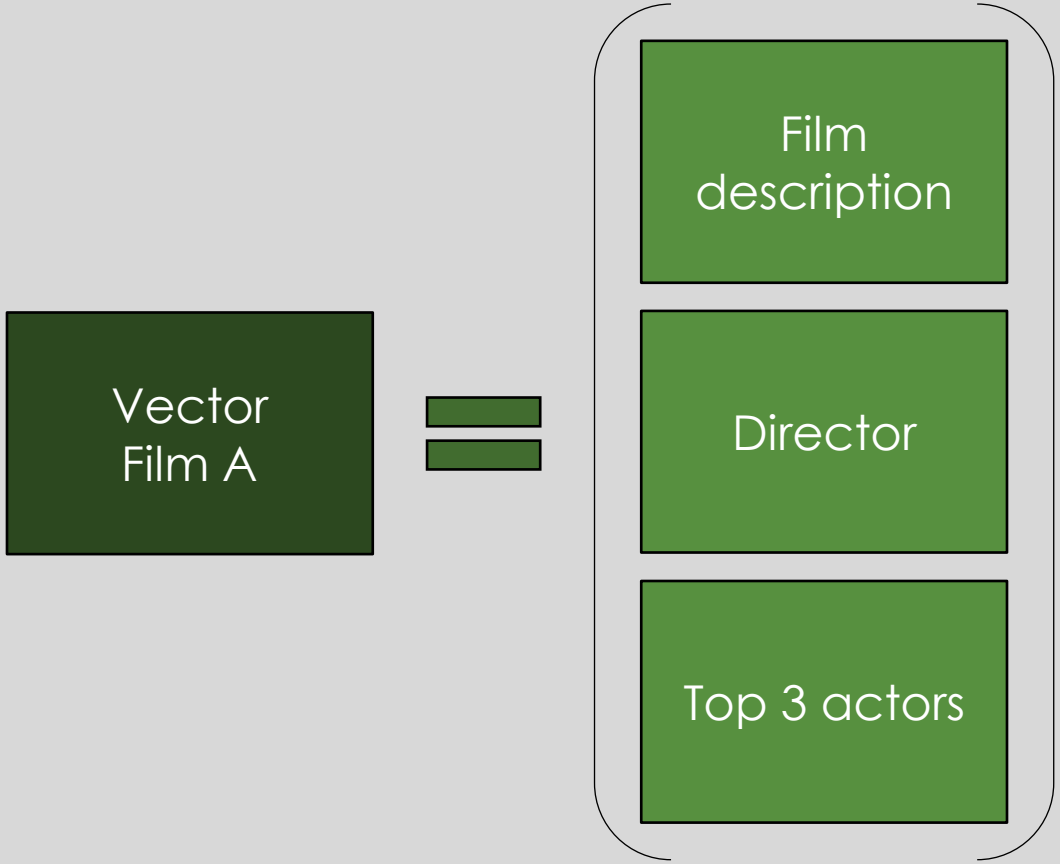
Content based – Datapreparation

```
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])

# Create a new soup feature
metadata['soup'] = metadata.apply(create_soup, axis=1)
```

	soup
0	jealousy toy boy tomhanks timallen donrickles ...
1	boardgame disappearance basedonchildren'sbook ...

Content-based recommendation

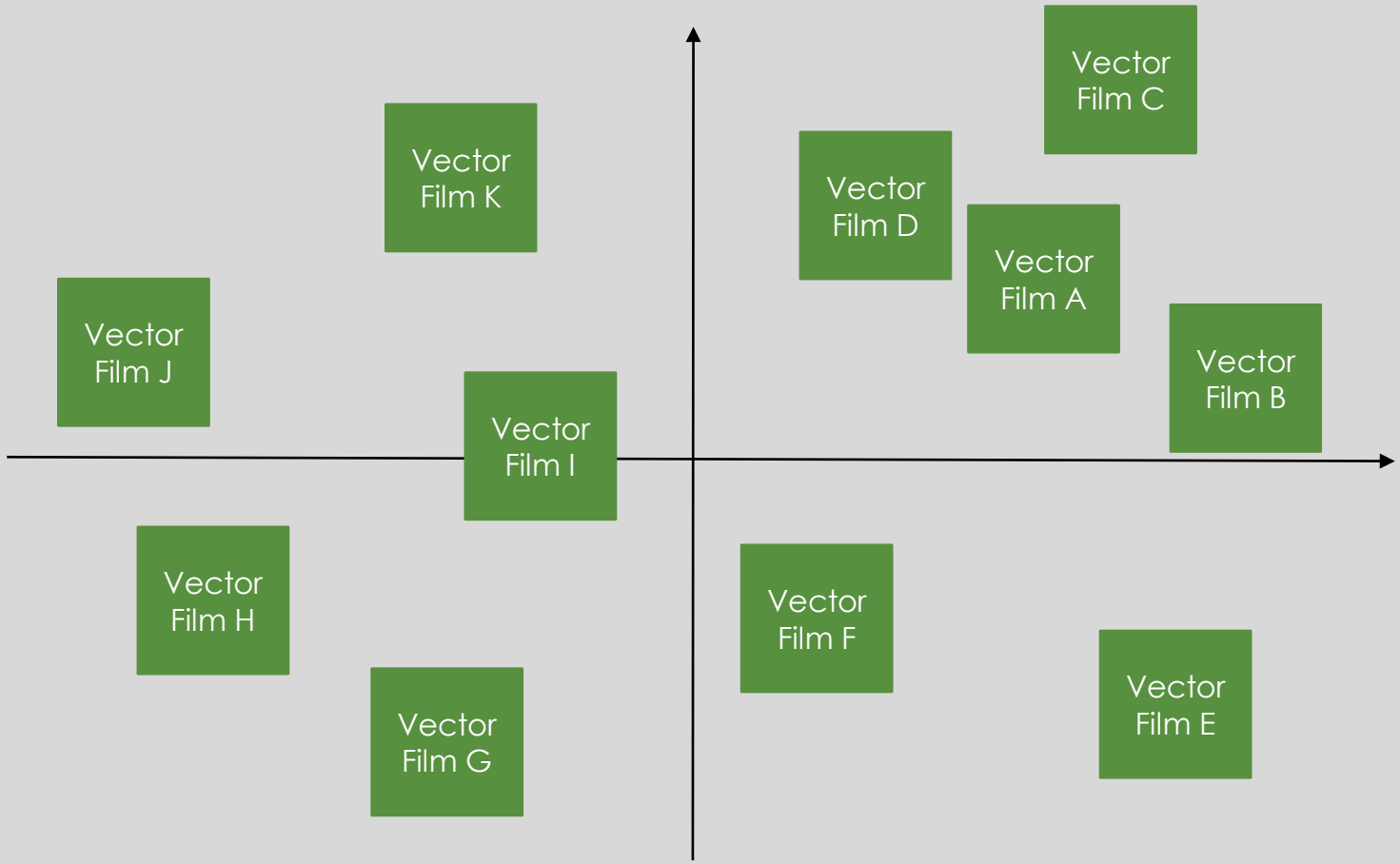


Attributes are joined as a lowercase string

↓

Sci-kit CountVectorizer()

Content-based recommendation

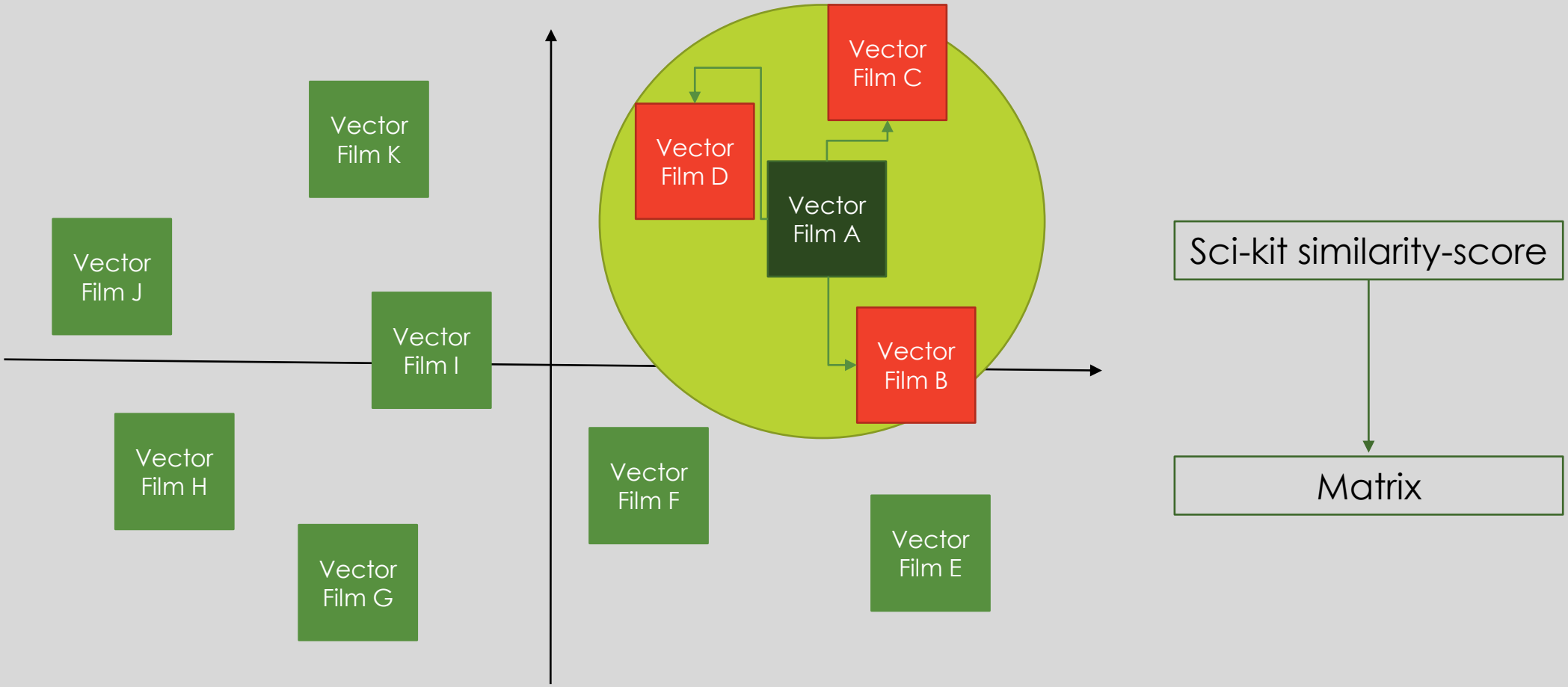


Content based – Calculation

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(metadata['soup'])
```

```
cosine_sim = cosine_similarity(count_matrix, count_matrix)
```

Content-based recommendation



Content based – From Search to Recommendation

```
input_title = input("Your title: ")
```

```
Your title: Toy Story
```

```
possible_titles = difflib.get_close_matches(input_title, metadata['original_title'].tolist(), 5)
print(possible_titles)
```

```
['Toy Story', 'Toy Story 3', 'Toy Story 2', 'True Story', 'Tall Story']
```

```
input_number = int(input("What's your film? Input 0 to 4: "))
chosen_title = possible_titles[input_number]
```

```
What's your film? Input 0 to 4: 0
```

```
chosen_id = metadata.loc[metadata['original_title'] == str(chosen_title), 'id'].array[0]
```

Content based – Recommendation

```python
def get_recommendations(id, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[id]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]
```

Content based – Recommendation

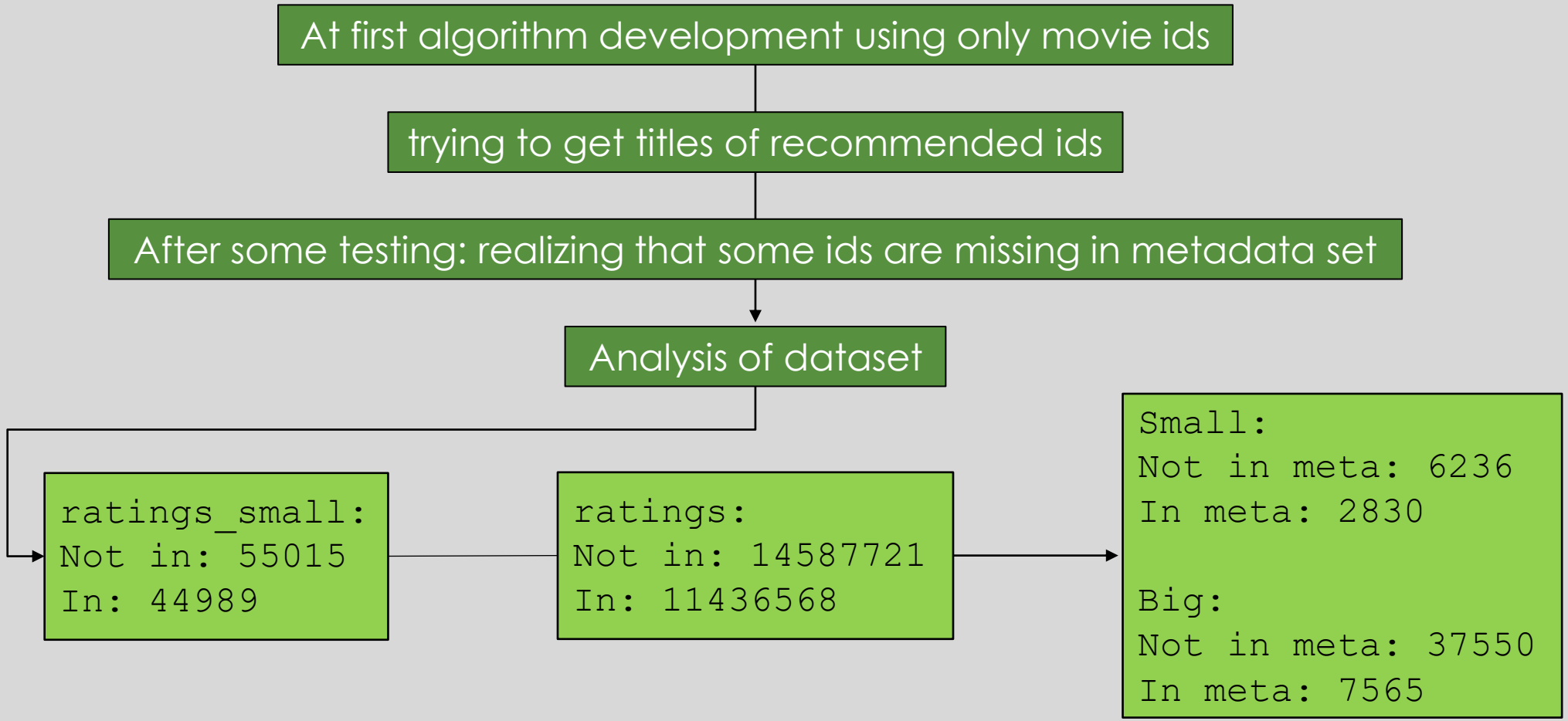
```
get_recommendations(chosen_id, cosine_sim)
```

3024	Toy Story 2
15519	Toy Story 3
29198	Superstar Goofy
26001	Toy Story That Time Forgot
22126	Toy Story of Terror!
3336	Creature Comforts
25999	Partysaurus Rex
27606	Anina
43071	Dexter's Laboratory: Ego Trip
28005	Radiopiratene

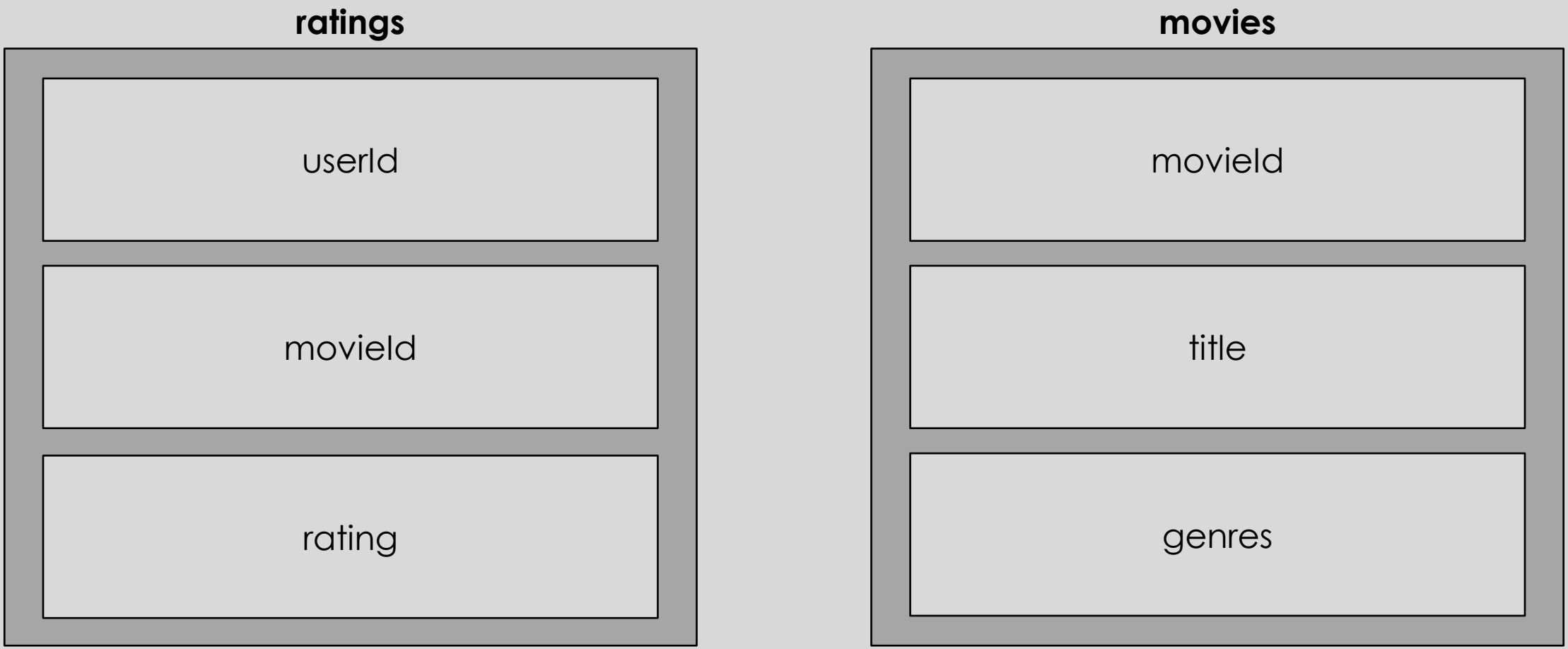
13.07.2021

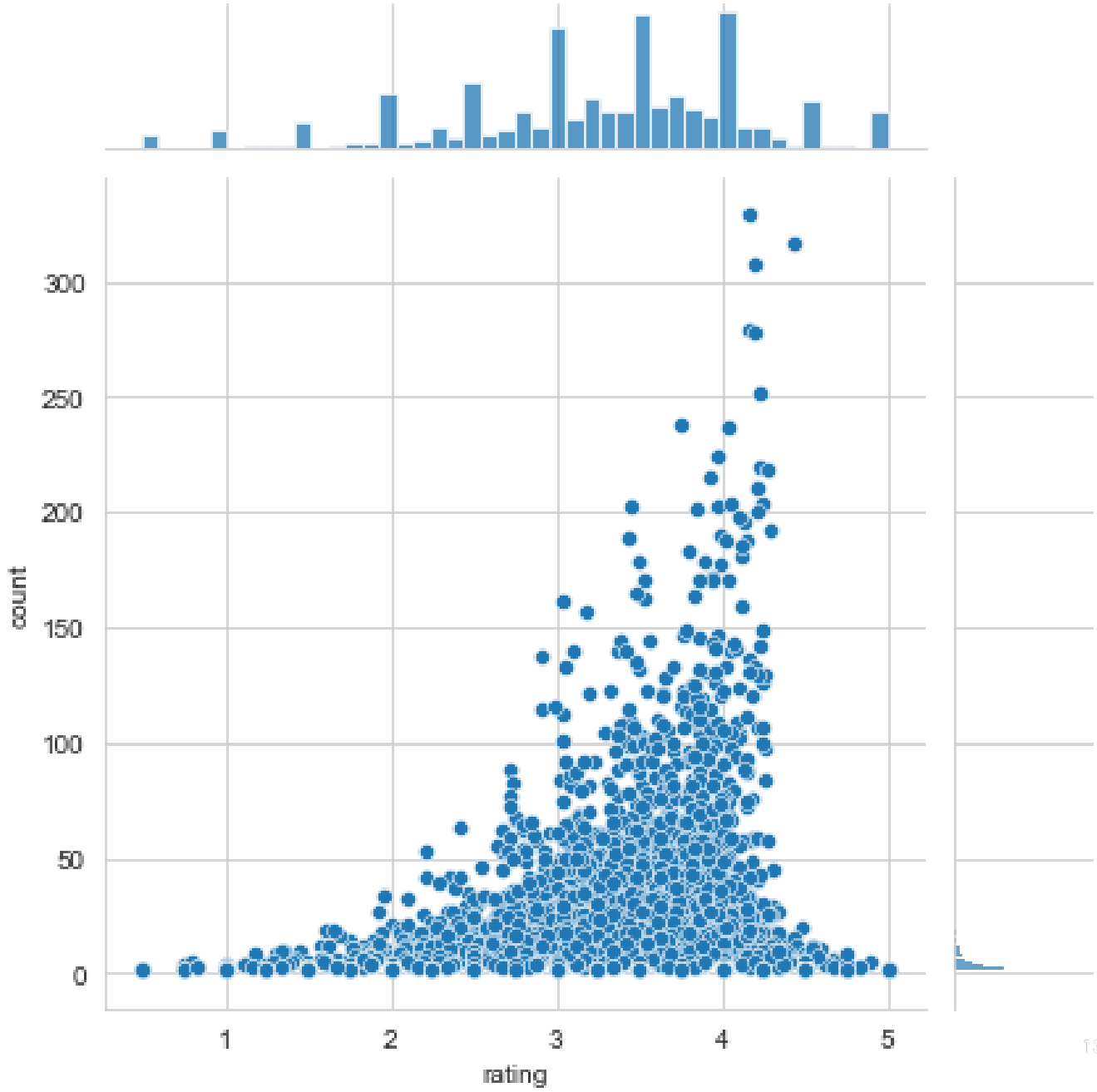
RATING BASED

Problem with original dataset



Dataset


ratings
userId
movieId
rating
movies
movieId
title
genres



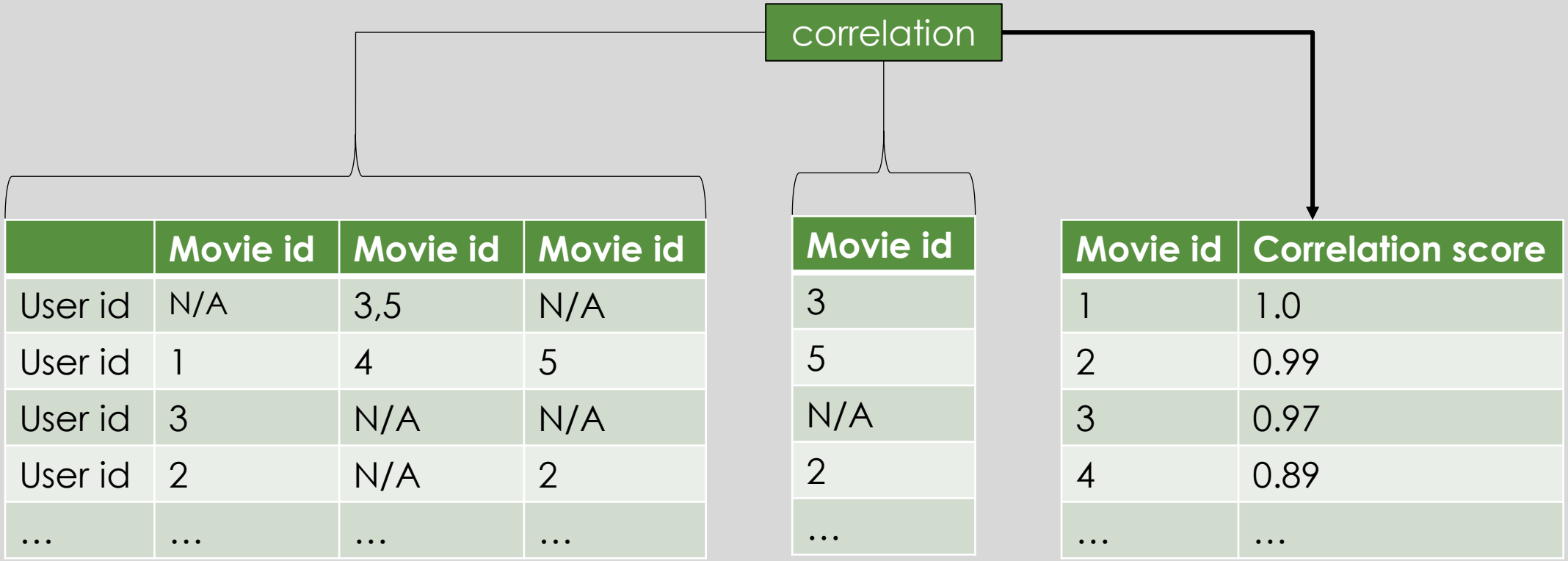
13.07.2021

Dataset

Viewing rating data

➔ Ratings are rudimentarily normally distributed

Rating-based recommendation



	Movie id	Movie id	Movie id
User id	N/A	3,5	N/A
User id	1	4	5
User id	3	N/A	N/A
User id	2	N/A	2
…	…	…	…

Movie id
3
5
N/A
2
…

Movie id	Correlation score
1	1.0
2	0.99
3	0.97
4	0.89
…	…

Content based - Packages

```
import pandas as pd
```

```
import collections
```

```
import copy
```

Rating-based – datapreparation

1) Creating table with **total count** of **ratings** for each movie id

```
rating_info = pd.DataFrame(ratings.groupby('movieId')['rating'].mean())
rating_info['count'] = pd.DataFrame(ratings.groupby('movieId')['rating'].count())
```

movieId	rating	count
1	3.920930	215
2	3.431818	110
3	3.259615	52
4	2.357143	7
5	3.071429	49

Rating-based – datapreparation

2) Create **pivot table**

```
rating_pivot = pd.pivot_table(ratings, index='userId', columns='movieId', values='rating')
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Rating-based – Algorithm

```
def recommendation(movie_id):
    #calculating correlation to other users
    correlation = rating_pivot.corrwith(rating_pivot[movie_id])
    #creating table
    corr_movie = pd.DataFrame(correlation, columns=['correlation'])
    #dropping movies non values => no correlation
    corr_movie.dropna(inplace=True)
    #adding total count of ratings for each movie to the table
    corr_movie = corr_movie.join(rating_info['count'])
    #listing movies by correlation score and just using movies with
     the total count of ratings per movie
    cutoff = rating_info['count'].quantile(0.90)
    corr_movie = corr_movie[corr_movie['count']>cutoff].sort_values
    #deleting movie that the recommendation is based on from recomm
    if movie_id in corr_movie.index:
        corr_movie = corr_movie.drop([movie_id])
    #changing index to ranking and movieId to column
    corr_movie['movieId'] = corr_movie.index
    corr_movie = corr_movie.set_index(pd.Index(list(range(len(corr_
    return corr_movie
```

	correlation	count	movieId
0	1.0	52	6870
1	1.0	31	2953
2	1.0	87	2011
3	1.0	48	2019
4	1.0	58	34162
...	...	...	...
400	-1.0	83	1101
401	-1.0	40	2541
402	-1.0	90	1610
403	-1.0	39	2746
404	-1.0	59	3535

Rating-based – Algorithm

```
def multi_recommendation
    #function to get rec
    rec_movies_list = []
    rec_movies_dict = {}
    for i in movie_ids:

        rec = recommendation(int(i))
        #skip if no correlating movies
        if rec.empty:
            continue


    return sorted(rec_movies_dict, key=rec_movies_dict.get, reverse=True)[:10]
```

```
#adding correlation score for top 5 recommended movies to dictionary
for k in range(5):
    id = rec.iloc[k].array[2]
    #checking if id has already been recommended
    if id in rec_movies_list:
        #if already recommended the correlation score of both cases are added up
        rec_movies_copy = copy.deepcopy(rec_movies_dict)
        rec_movies_copy.update({id: rec_movies_dict.get(id) + rec.iloc[k][0]})
        rec_movies_dict = rec_movies_copy
    else:
        rec_movies_dict[rec.iloc[k][2]] = rec.iloc[k][0]
        rec_movies_list.append(id)
```

old score + new correlation score = new score

13.07.2021

LIVE PRESENTATION