



NLP FINAL PROJECT

Movie Recommender System



Prepared by :

Andrew Adel

Hussien Amin

Phoebe Thabit

Sandy Adel

Problem Formulation :

The movie chatbot recommendation problem aims to build an interactive chatbot that can engage in a conversation with users and recommend movies based on their preferences. The chatbot should be able to understand the user's movie preferences, ask relevant questions to gather necessary information, and provide a list of movie recommendations tailored to the user's tastes.

Innovativeness :

The Movie recommendation chatbot offers a unique value proposition through two distinct recommendation approaches. Firstly, it suggests movies based on the user's favorite genre, ensuring personalized suggestions aligned with their cinematic preferences. Additionally, it employs a novel method by recommending movies related to the user's favorite movie. This dual approach provides users with a broader spectrum of movie choices, enhancing their viewing experience and catering to their specific tastes. By combining these two innovative recommendation strategies, our chatbot offers a comprehensive and tailored movie recommendation service that differentiates it from conventional systems.

Dataset Description :

We have two MovieLens datasets.

The Full Dataset: Consists of 26,000,000 ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users. Includes tag genome data with 12 million relevance scores across 1,100 tags.

The Small Dataset: Comprises of 100,000 ratings and 1,300 tag applications applied to 9,000 movies by 700 users.

Reading data:

```
[ ] df1.head()
```

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date	revenue	runtime
0	False	['id': 10194, 'name': 'Toy Story Collection', ...]	30000000	['id': 16, 'name': 'Animation', 'id': 35, ...]	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his	1995-10-30	373554033.0	81.0
1	False	NaN	65000000	['id': 12, 'name': 'Adventure', 'id': 14, ...]	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	1995-12-15	262797249.0	104.0
2	False	['id': 119050, 'name': 'Grumpy Old Men Collect...	0	['id': 10749, 'name': 'Romance', 'id': 35, ...]	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancient feud be...	...	1995-12-22	0.0	101.0
3	False	NaN	16000000	['id': 35, 'name': 'Comedy', 'id': 18, ...]	NaN	31357	tt0114885	en	Waiting to Exhale	Cheated on, mistreated and stepped on the wom...	...	1995-12-22	81452156.0	127.0

Data Preparation :

● Data Cleaning :

```
print("--- cleaning ---")
overview = re.sub(r'^[^\s]', '', str(overview).lower().strip())
```

```
--- cleaning ---
```

```
0      led by woody andys toys live happily in his
1      when siblings judy and peter discover an encha
2      a family wedding reignites the ancient feud be
3      cheated on mistreated and stepped on the wom
4      just when george banks has recovered from his

45461      rising and falling between a man and woman
45462      an artist struggles to finish his work while a
45463      when one of her hits goes wrong a professiona
45464      in a small town live two brothers one a minis
45465      50 years after decriminalisation of homosexual
name overview length 45466 dtype object
```

- **Data Tokenization :**

Tokenization

```
In [21]: overview = overview.split()  
print(overview)
```

```
['0', 'led', 'by', 'woody', 'andys', 'toys', 'live', 'happily', 'in', 'his', '1', 'when', 'siblings', 'judy', 'and', 'peter',  
'discover', 'an', 'encha', '2', 'a', 'family', 'wedding', 'reignites', 'the', 'ancient', 'feud', 'be', '3', 'cheated', 'on', 'm  
istreated', 'and', 'stepped', 'on', 'the', 'won', '4', 'just', 'when', 'george', 'banks', 'has', 'recovered', 'from', 'his', '4  
5461', 'rising', 'and', 'falling', 'between', 'a', 'man', 'and', 'woman', '45462', 'an', 'artist', 'struggles', 'to', 'finish',  
'his', 'work', 'while', 'a', '45463', 'when', 'one', 'of', 'her', 'hits', 'goes', 'wrong', 'a', 'professiona', '45464', 'in',  
'a', 'small', 'town', 'live', 'two', 'brothers', 'one', 'a', 'minis', '45465', '50', 'years', 'after', 'decriminalisation', 'o  
f', 'homosexual', 'name', 'overview', 'length', '45466', 'dtype', 'object']
```

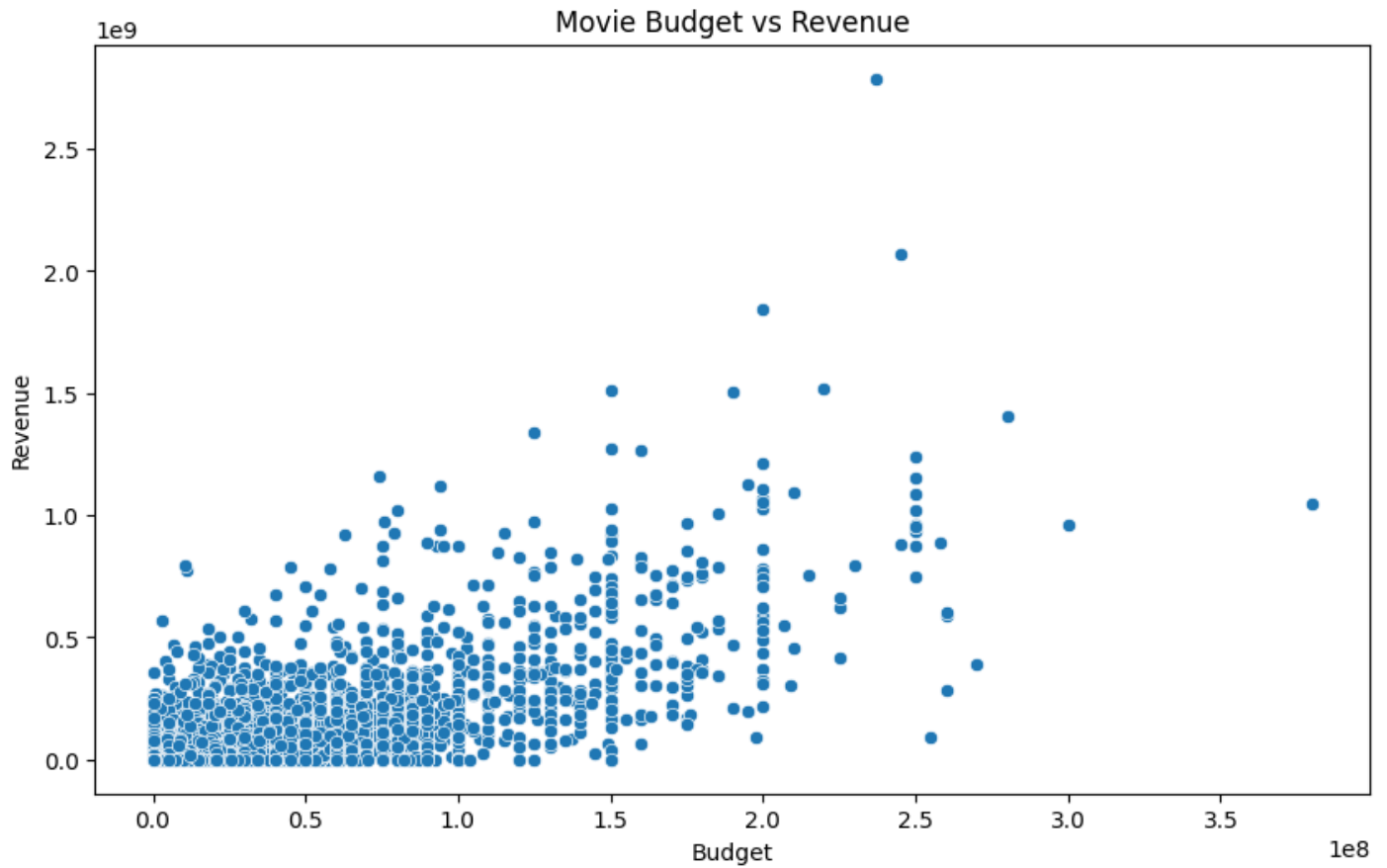
- **Data Lemmatization :**

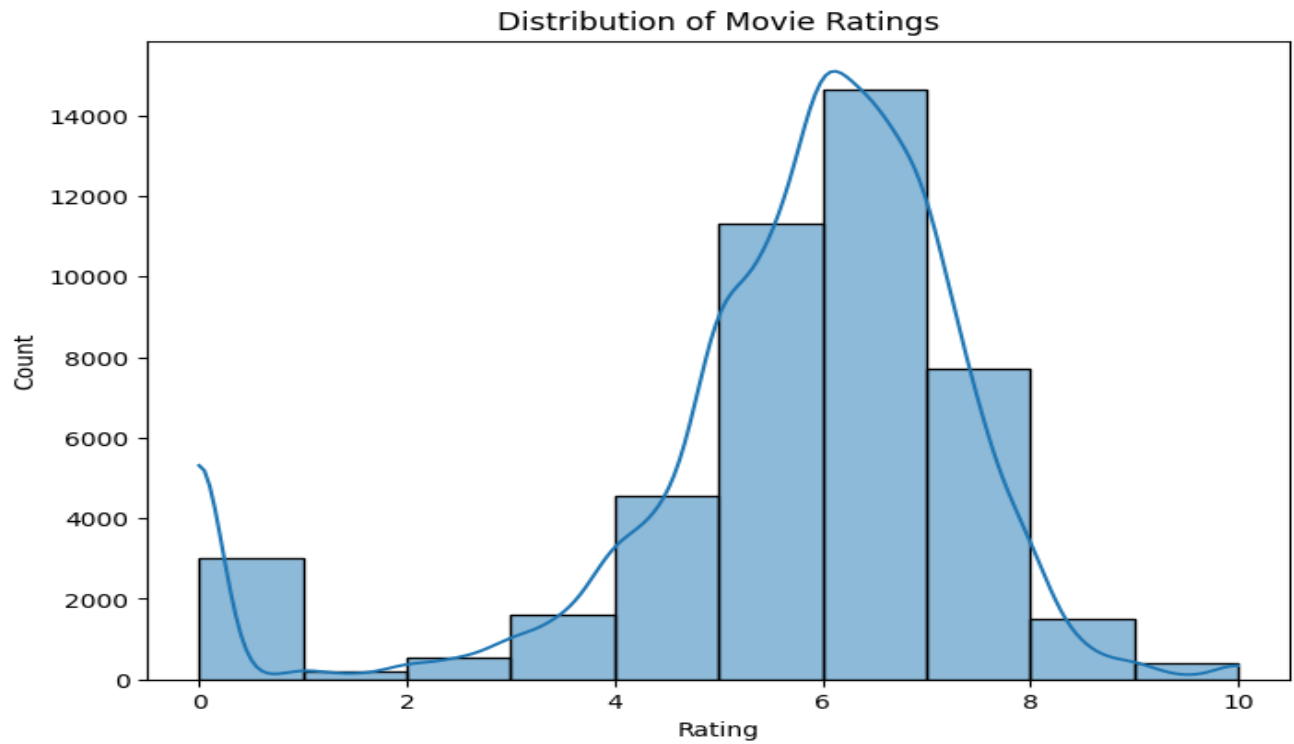
Lemmatiztion

```
In [23]: lem = nltk.stem.wordnet.WordNetLemmatizer()  
print([lem.lemmatize(word) for word in overview])
```

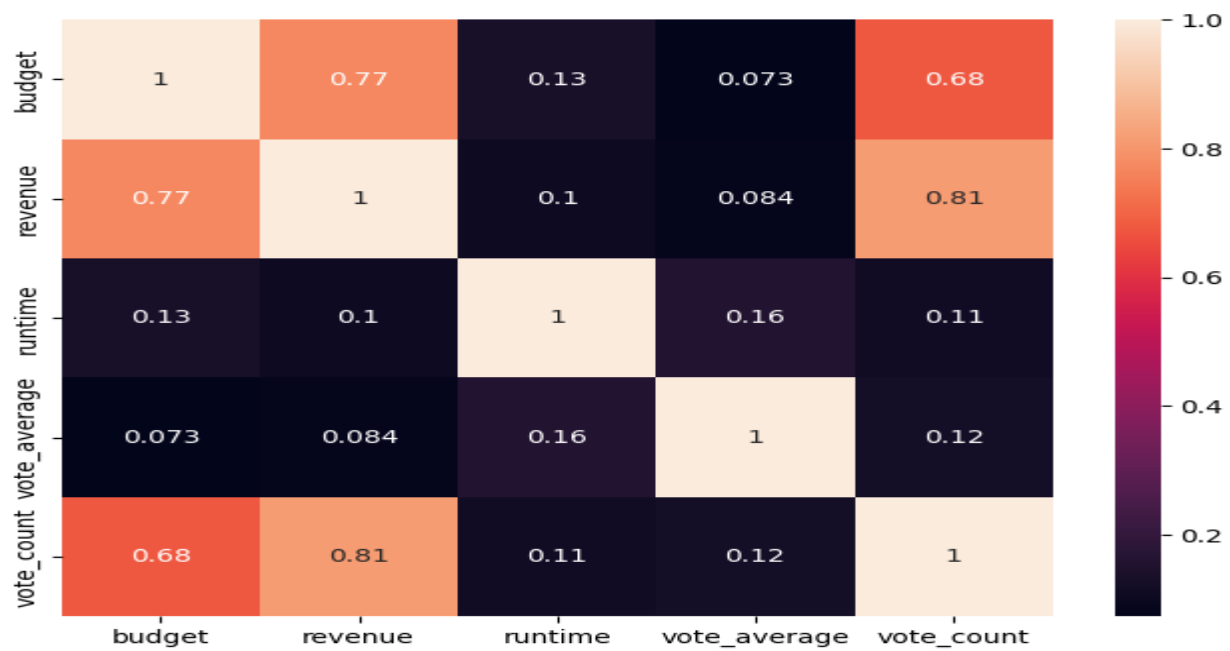
```
['0', 'led', 'by', 'woody', 'andys', 'toy', 'live', 'happily', 'in', 'his', '1', 'when', 'sibling', 'judy', 'and', 'peter', 'di  
scover', 'an', 'encha', '2', 'a', 'family', 'wedding', 'reignites', 'the', 'ancient', 'feud', 'be', '3', 'cheated', 'on', 'mist  
reated', 'and', 'stepped', 'on', 'the', 'won', '4', 'just', 'when', 'george', 'bank', 'ha', 'recovered', 'from', 'his', '4546  
1', 'rising', 'and', 'falling', 'between', 'a', 'man', 'and', 'woman', '45462', 'an', 'artist', 'struggle', 'to', 'finish', 'hi  
s', 'work', 'while', 'a', '45463', 'when', 'one', 'of', 'her', 'hit', 'go', 'wrong', 'a', 'professiona', '45464', 'in', 'a', 's  
mall', 'town', 'live', 'two', 'brother', 'one', 'a', 'mini', '45465', '50', 'year', 'after', 'decriminalisation', 'of', 'homose  
xual', 'name', 'overview', 'length', '45466', 'dtype', 'object']
```

Data Visualization :

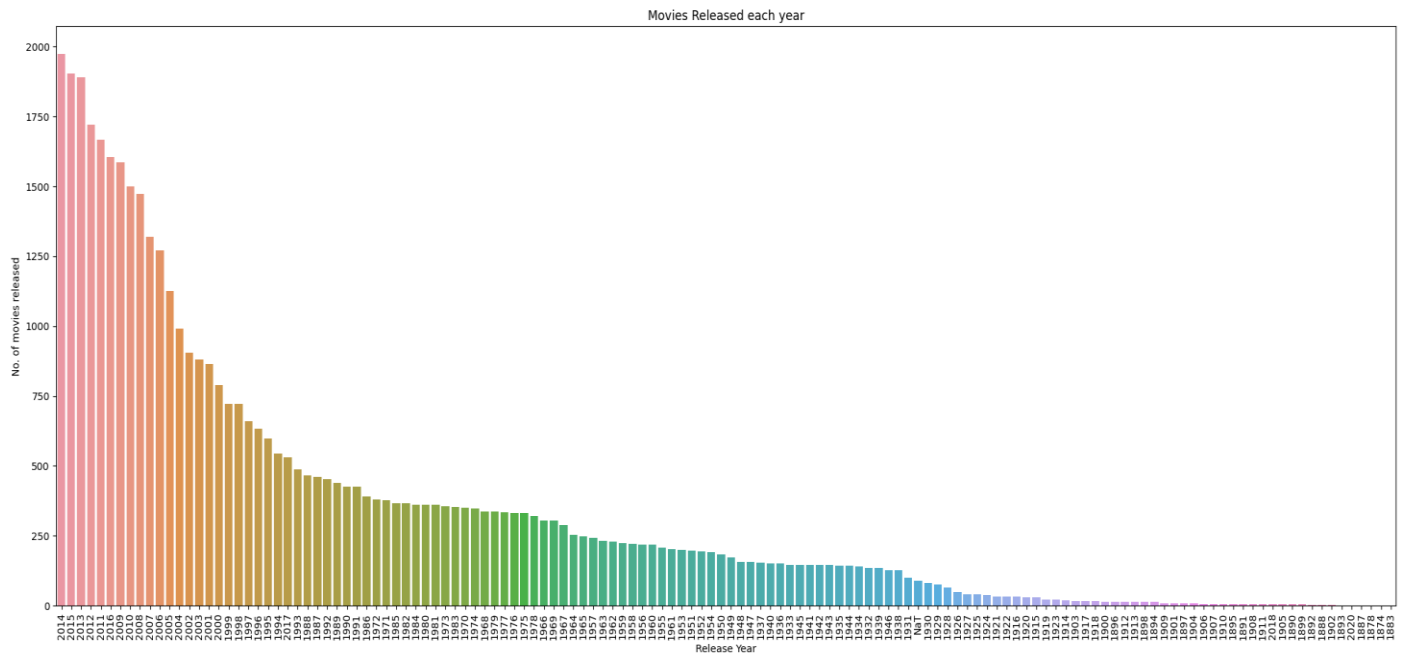




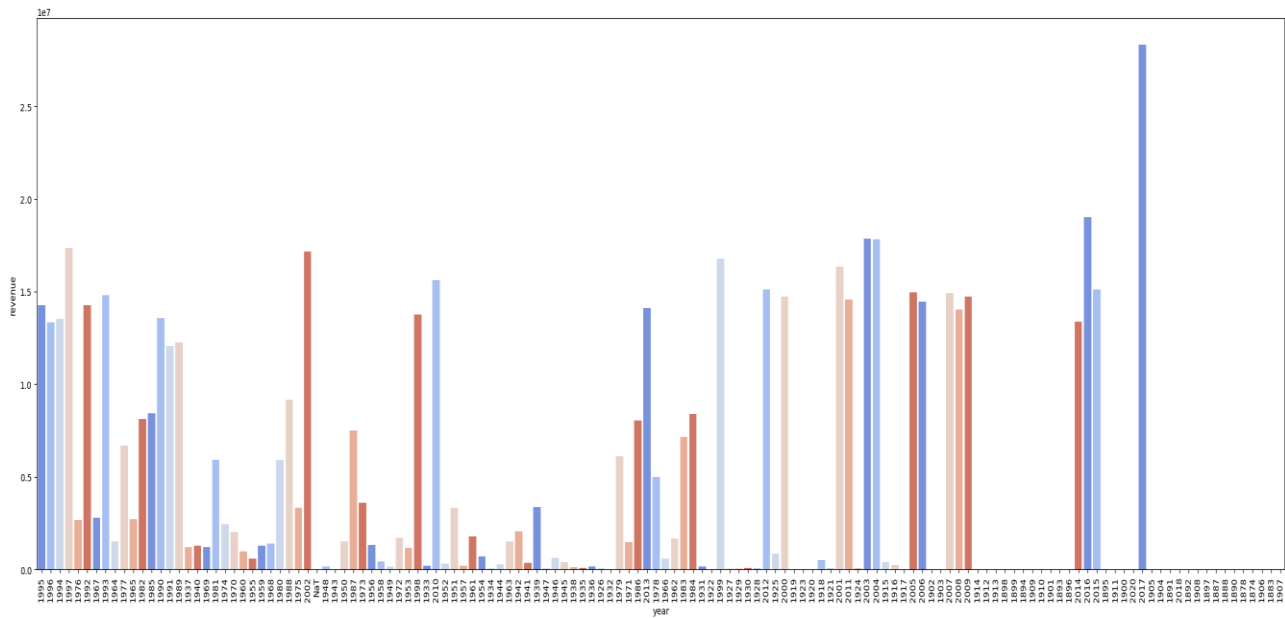
Correlation matrix for numeric variables :



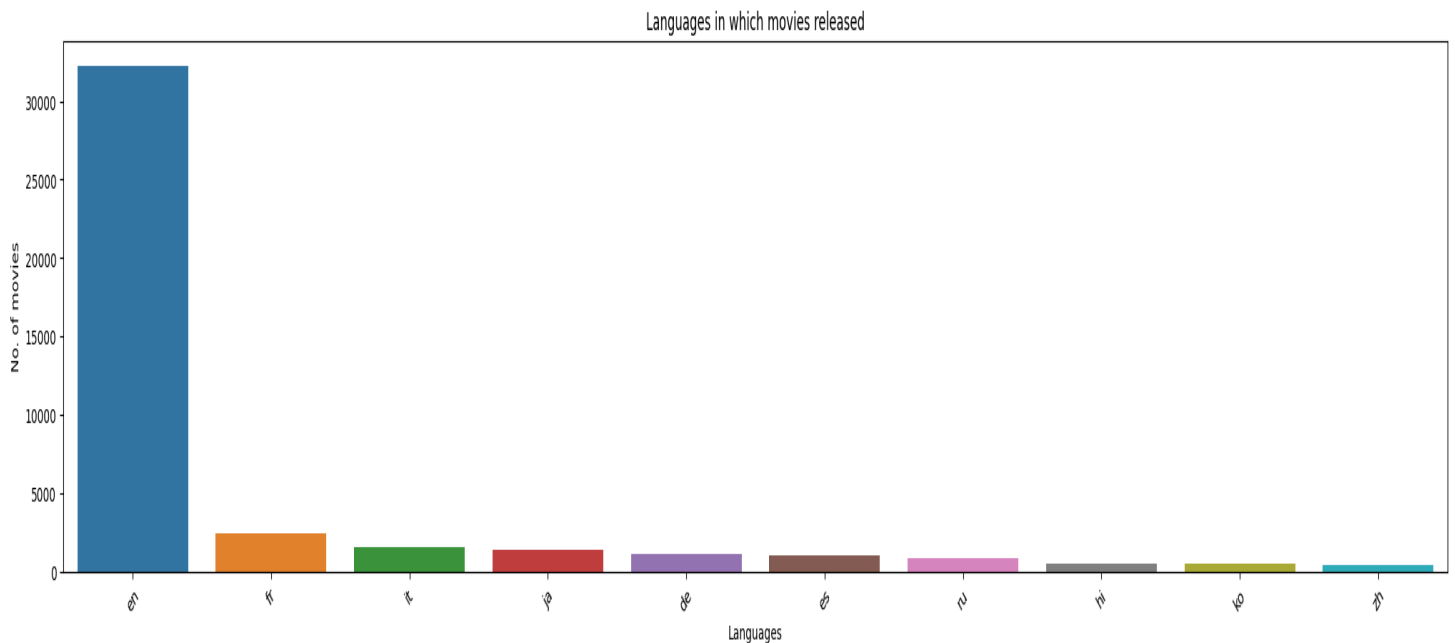
A figure that shows the number of movies that realeased every year :



A figure that shows the revenue increase every year :



A figure that shows the distribution of the languages of movies :



Code snippets :

We apply tfidf to overview after that we took this matrix and train kmeans on it to get labels

```
[ ] tfidf_df
```

	10	10 year	100	100 year	1000	10000	100000	10yearold	11	11 year	...	zack	zatoichi	zealand	zero	zhang	zijn	zoe	zombie	zone	zoo
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...
45461	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45462	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45463	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45464	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45465	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

45466 rows × 10000 columns

To extract the most frequent genre in each cluster in the data to be the title of this cluster :

```
[32] random_sample_0 = merged[merged["label"] == 0]
      random_sample_1 = merged[merged["label"] == 1]
      random_sample_2 = merged[merged["label"] == 2]
      random_sample_3 = merged[merged["label"] == 3]
      random_sample_4 = merged[merged["label"] == 4]
      random_sample_5 = merged[merged["label"] == 5]
      random_sample_6 = merged[merged["label"] == 6]
      random_sample_7 = merged[merged["label"] == 7]
      random_sample_8 = merged[merged["label"] == 8]
```

```
[36] genres = list()
      for genre in random_sample_3["genres"]:
          for i in range(len(genre)):
              genres.append(genre[i])
      print(Counter(genres))
```

Counter({'Drama': 7094, 'Comedy': 5313, 'Action': 4203, 'Thriller': 4060, 'Crime': 2455, 'Horror': 2455, 'Mystery': 2455, 'Romance': 2455, 'Sci-Fi': 2455, 'Western': 2455})

```
[37] genres = list()
      for genre in random_sample_4["genres"]:
          for i in range(len(genre)):
              genres.append(genre[i])
      print(Counter(genres))
```

Counter({'Drama': 1478, 'Romance': 1212, 'Comedy': 841, 'Thriller': 186, 'Action': 181, 'Horror': 181, 'Mystery': 181, 'Crime': 181, 'Western': 181, 'Sci-Fi': 181})

```
[36] genres = list()
      for genre in random_sample_3["genres"]:
          for i in range(len(genre)):
              genres.append(genre[i])
      print(Counter(genres))
```

Counter({'Drama': 7094, 'Comedy': 5313, 'Action': 4203, 'Thriller': 4060, 'Crime': 2455, 'Horror': 2455, 'Mystery': 2455, 'Romance': 2455, 'Sci-Fi': 2455, 'Western': 2455})

```
[37] genres = list()
      for genre in random_sample_4["genres"]:
          for i in range(len(genre)):
              genres.append(genre[i])
      print(Counter(genres))
```

Counter({'Drama': 1478, 'Romance': 1212, 'Comedy': 841, 'Thriller': 186, 'Action': 181, 'Horror': 181, 'Mystery': 181, 'Crime': 181, 'Western': 181, 'Sci-Fi': 181})

```
[38] genres = list()
      for genre in random_sample_5["genres"]:
          for i in range(len(genre)):
              genres.append(genre[i])
      print(Counter(genres))
```

we assign a genre to each cluster after we show the biggest number of this genre in this cluster

```
[42] merged.loc[merged["label"] == 0, "genres"] = "War"
merged.loc[merged["label"] == 1, "genres"] = "Crime"
merged.loc[merged["label"] == 2, "genres"] = "Horror"
merged.loc[merged["label"] == 3, "genres"] = "Drama"
merged.loc[merged["label"] == 4, "genres"] = "Romance"
merged.loc[merged["label"] == 5, "genres"] = "Comedy"
merged.loc[merged["label"] == 6, "genres"] = "Documentary"
merged.loc[merged["label"] == 7, "genres"] = "Thriller"
merged.loc[merged["label"] == 8, "genres"] = "Foreign"
```

output :

```
[43] merged.head()
```

budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	spoken_languages	status	tagline	title
30000000.0	Drama	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his	[{"iso_639_1": "en", "name": "English"}]	Released	NaN	Toy Story
65000000.0	Drama	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Roll the dice and unleash the excitement!	Jumanji
0.0	Foreign	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancient feud be...	...	[{"iso_639_1": "en", "name": "English"}]	Released	Still Yelling. Still Fighting. Still Ready for...	Grumpier Old Men
							Cheated on,				Friends are	

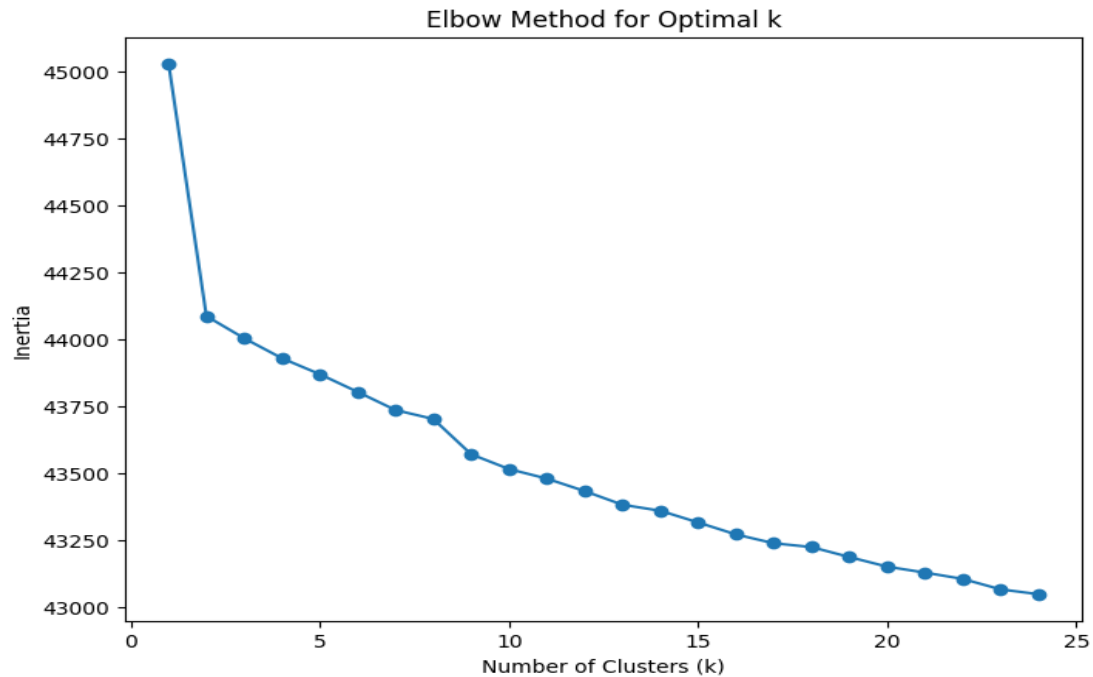
original_language	original_title	overview	...	spoken_languages	status	tagline	title	video	vote_average	vote_count	year	overview_filtered	label
en	Toy Story	Led by Woody, Andy's toys live happily in his room.	...	[{"iso_639_1": "en", "name": "English"}]	Released	NaN	Toy Story	False	7.7	5415.0	1995	led woody andys toy live happily room andys bi...	3
en	Jumanji	When siblings Judy and Peter discover an enchanted board game that opens the door to a magical world of adventure, a thrilling race begins.	...	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "es", "name": "Spanish"}]	Released	Roll the dice and unleash the excitement!	Jumanji	False	6.9	2413.0	1995	sibling judy peter discover enchanted board ga...	3
en	Grumpier Old Men	A family wedding reignites the ancient feud between two bickering families.	...	[{"iso_639_1": "en", "name": "English"}]	Released	Still Yelling. Still Fighting. Still Ready for war.	Grumpier Old Men	False	6.5	92.0	1995	family wedding reignites ancient feud nextdoor...	8
en	Waiting to Exhale	Cheated on, mistreated and stepped on, a woman finally finds love.	...	[{"iso_639_1": "en", "name": "English"}]	Released	Friends are the people who let you be who you are.	Waiting to Exhale	False	6.1	34.0	1995	cheated mistreated stepped woman holding breat...	7

The distinct genres :

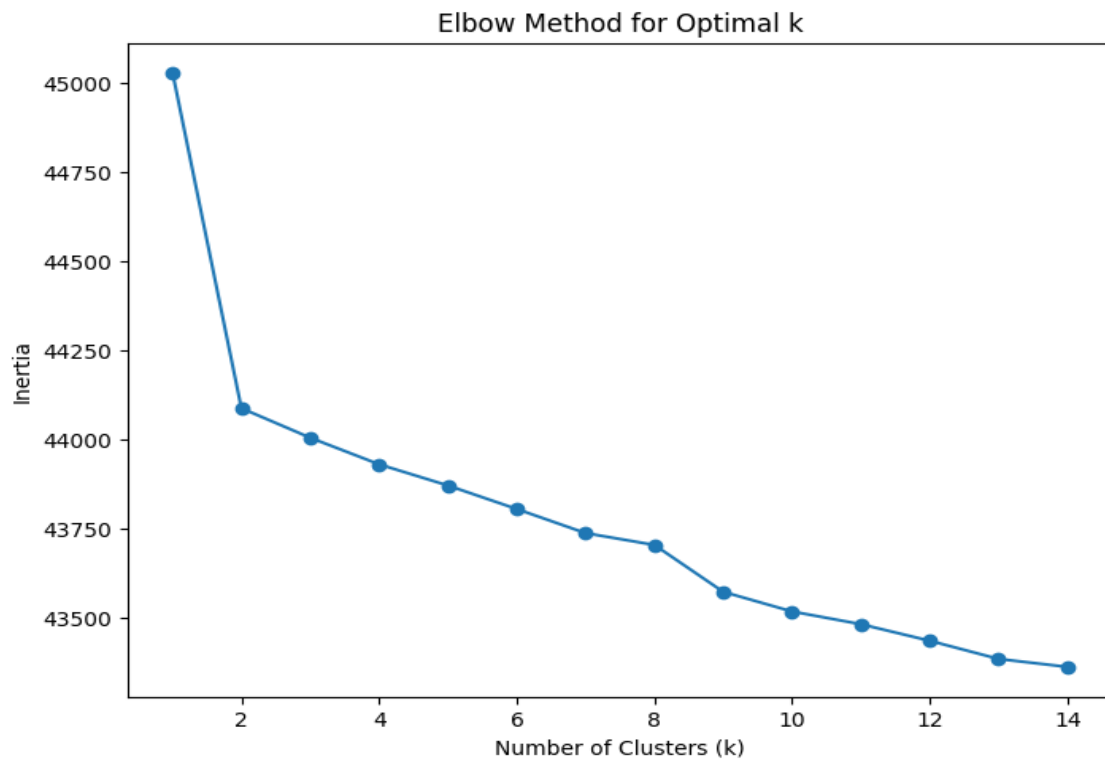
```
array(['Drama', 'Foreign', 'Thriller', 'Comedy', 'Documentary', 'Horror',
      'Crime', 'War', 'Romance'], dtype=object)
```

Error Analysis & Evaluation :

- Error Analysis For Kmeans Clustering : (k=25)



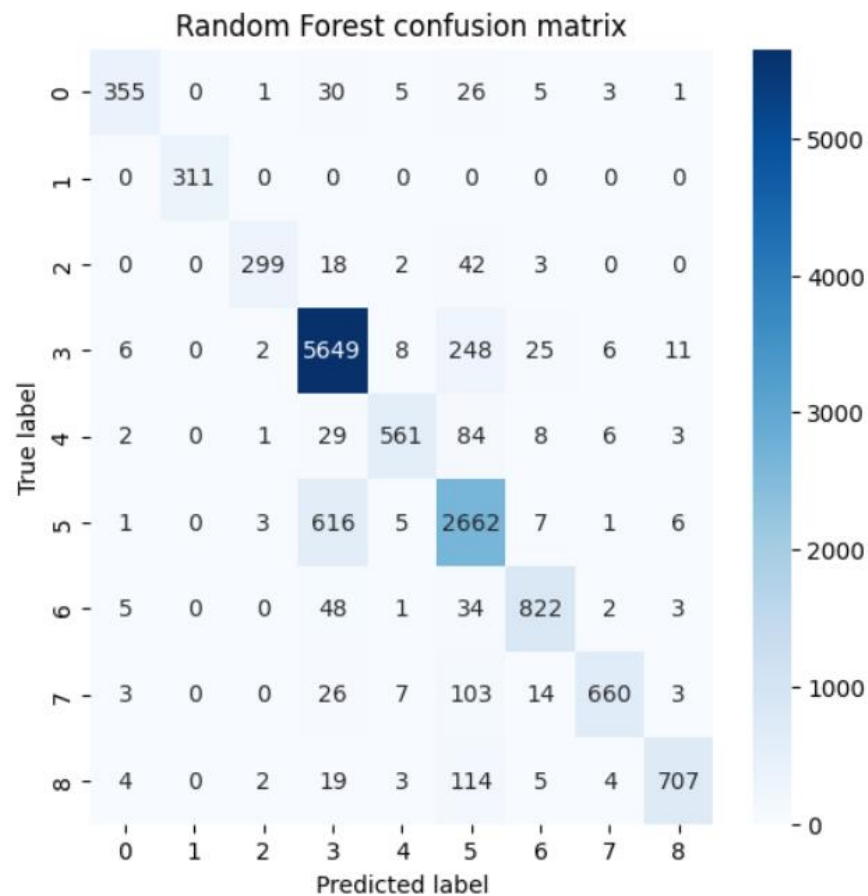
(k=15):



- **Classification (Random Forest) :**

Random Forest accuracy : 0.8834310850439883

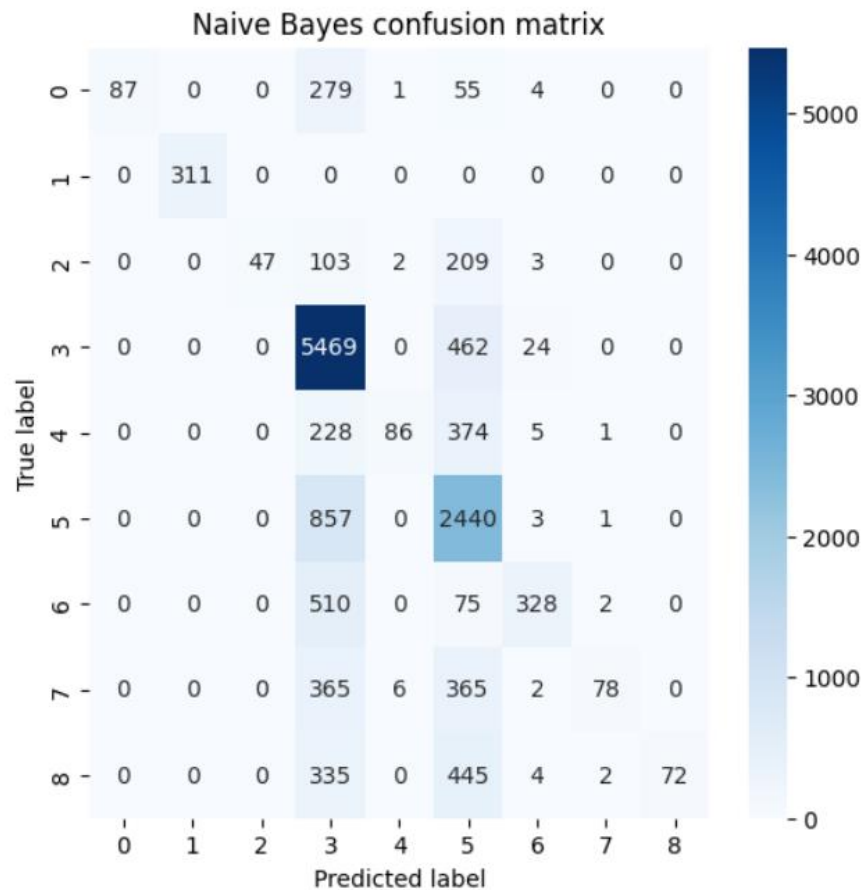
	precision	recall	f1-score	support
0	0.94	0.86	0.90	426
1	1.00	1.00	1.00	311
2	0.98	0.82	0.90	364
3	0.88	0.95	0.91	5955
4	0.94	0.84	0.88	694
5	0.81	0.80	0.81	3301
6	0.93	0.88	0.90	915
7	0.95	0.84	0.89	816
8	0.96	0.81	0.88	858
accuracy			0.88	13640
macro avg	0.93	0.87	0.90	13640
weighted avg	0.89	0.88	0.88	13640



- **Classification (Naïve Bayes):**

Naive Bayes Accuracy: 0.6538123167155425

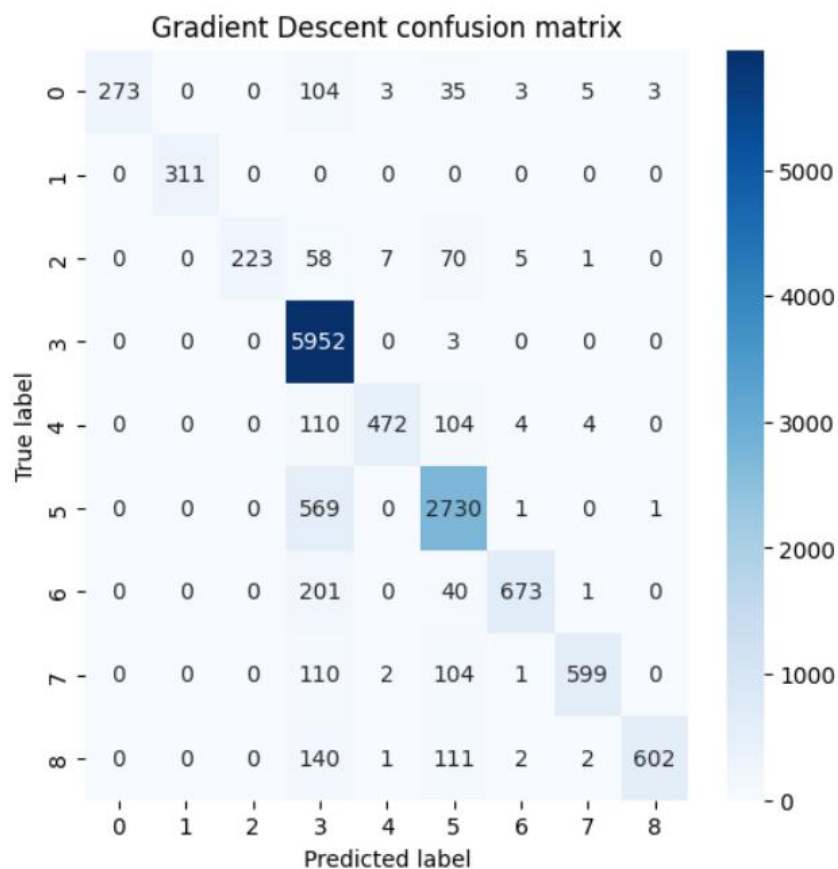
	precision	recall	f1-score	support
0	1.00	0.20	0.34	426
1	1.00	1.00	1.00	311
2	1.00	0.13	0.23	364
3	0.67	0.92	0.78	5955
4	0.91	0.12	0.22	694
5	0.55	0.74	0.63	3301
6	0.88	0.36	0.51	915
7	0.93	0.10	0.17	816
8	1.00	0.08	0.15	858
accuracy			0.65	13640
macro avg	0.88	0.41	0.45	13640
weighted avg	0.73	0.65	0.60	13640



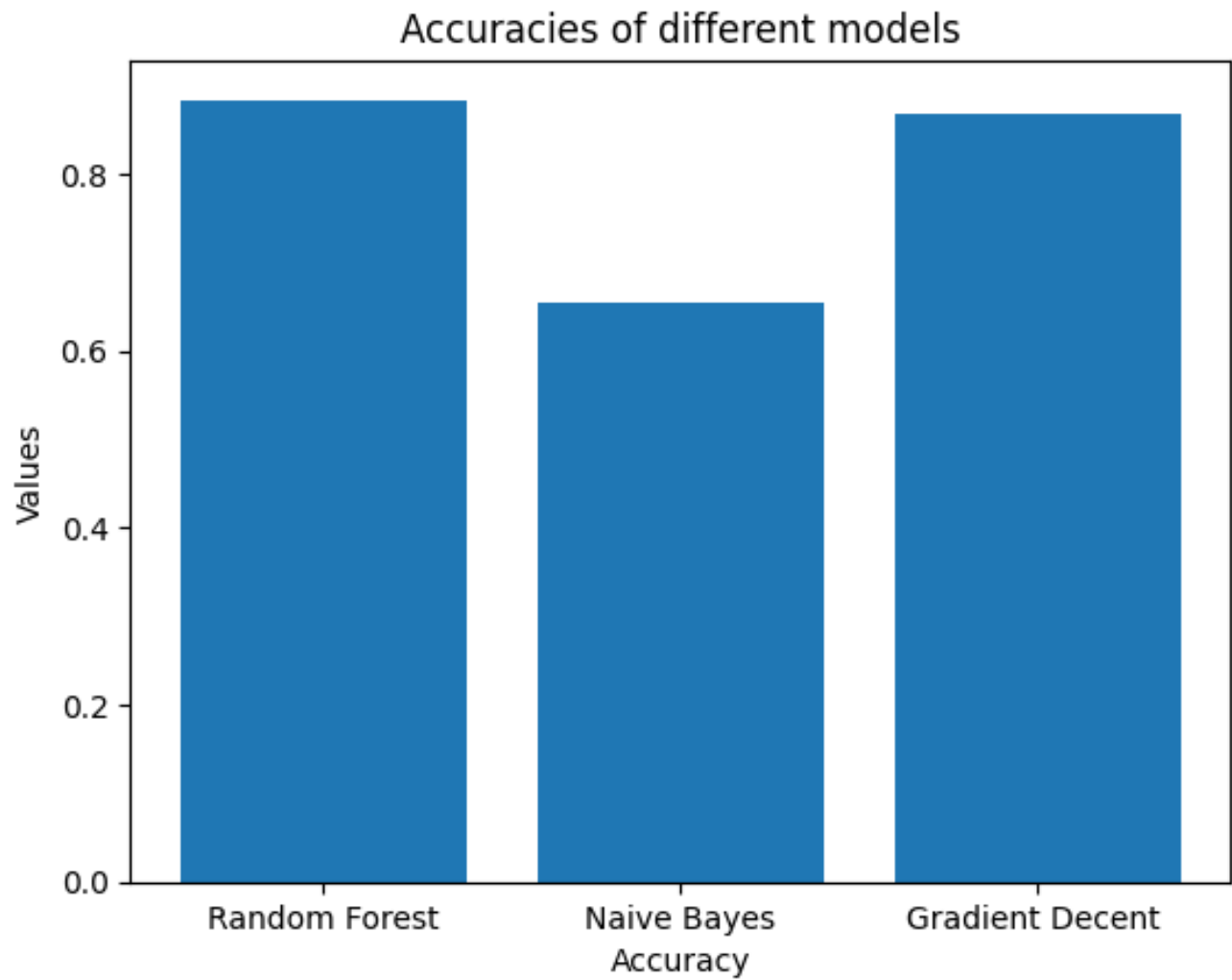
- **Classification (Gradient Descent):**

Stochastic Gradient Descent accuracy : 0.8676686217008798

	precision	recall	f1-score	support
0	1.00	0.64	0.78	426
1	1.00	1.00	1.00	311
2	1.00	0.61	0.76	364
3	0.82	1.00	0.90	5955
4	0.97	0.68	0.80	694
5	0.85	0.83	0.84	3301
6	0.98	0.74	0.84	915
7	0.98	0.73	0.84	816
8	0.99	0.70	0.82	858
accuracy			0.87	13640
macro avg	0.96	0.77	0.84	13640
weighted avg	0.88	0.87	0.86	13640



Champion Model :



Based on the previous graph, Our champion model is **Random Forest** .

Recommendation :

- **Type Based Recommendation :**

```
build_chart("Action")
```

	title	year	vote_count	vote_average	popularity	wr
15480	Inception	2010	14075	8	29.108149	7.955099
12481	The Dark Knight	2008	12269	8	123.167259	7.948610
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.070725	7.929579
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.324358	7.924031
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.423537	7.918382
...
21296	The Wolverine	2013	4110	6	3.918287	5.956636
13635	X-Men Origins: Wolverine	2009	4086	6	1.456541	5.956395
5244	Star Wars: Episode II - Attack of the Clones	2002	4074	6	14.072511	5.956273
18289	Mission: Impossible - Ghost Protocol	2011	4026	6	14.25843	5.955779
7917	I, Robot	2004	3889	6	14.43075	5.954308

- **Content Based Recommendation :**

```
get_recommendations('Fight Club').head(10)
```

```
4070          Delirious
980          Raging Bull
8200      Poor Little Rich Girl
3662          UHF
8718      The Borderlands
6901      Never Back Down
302    What's Eating Gilbert Grape
8219      Role/Play
6124      Unleashed
6979    Z Channel: A Magnificent Obsession
Name: title, dtype: object
```

- **Rating Based Recommendation :**

```
rating_recommendations('Fight Club')
```

	title	vote_count	vote_average	year	wr
2614	Twin Peaks: Fire Walk with Me	429	7	1992	6.117364
8088	Skyfall	7718	6	2012	5.959799
7870	Tinker Tailor Soldier Spy	856	6	2011	5.745957
2678	The Ninth Gate	768	6	1999	5.727359
149	Hackers	406	6	1995	5.609863
5946	The Interpreter	400	6	2005	5.607057
20	Get Shorty	305	6	1995	5.556543
6052	Be Cool	296	5	2005	5.145596
1683	Snake Eyes	331	5	1998	5.138935
7995	This Means War	1411	5	2012	5.057607

- **Collaboative Filtering :**

```
algo.fit(trainset)
ratings[ratings['userId'] == 1]
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191


- **Hybrid Recommendation :**


```
multi_recommend(3 , "Fight Club")
```

	title	vote_count	vote_average	year	id	est
5239	Cypher	196.0	6.7	2002	10133	3.769404
8088	Skyfall	7718.0	6.9	2012	37724	3.754768
713	Mother Night	13.0	7.1	1996	20318	3.679639
3824	Funeral in Berlin	31.0	6.1	1966	34388	3.570245
5756	The Tall Blond Man with One Black Shoe	58.0	6.9	1972	12089	3.554401
1599	Condorman	37.0	5.6	1981	19379	3.539049
6480	Compulsion	23.0	6.8	1959	35921	3.495012
8065	Cleanskin	118.0	5.6	2012	95516	3.491492
5946	The Interpreter	400.0	6.2	2005	179	3.461834
2614	Twin Peaks: Fire Walk with Me	429.0	7.3	1992	1923	3.434579

Testing Chatbot (Dialogflow) :

Genre based :

 MovieRecommender


POWERED BY  Dialogflow


hi


Good day! What can I do for you today?

Can you recommend me a movie to watch today?

Of course , I can recommend you a movie according to your preferred genre or a movie like your favorite one ..

Ask something... 

 MovieRecommender

POWERED BY  Dialogflow

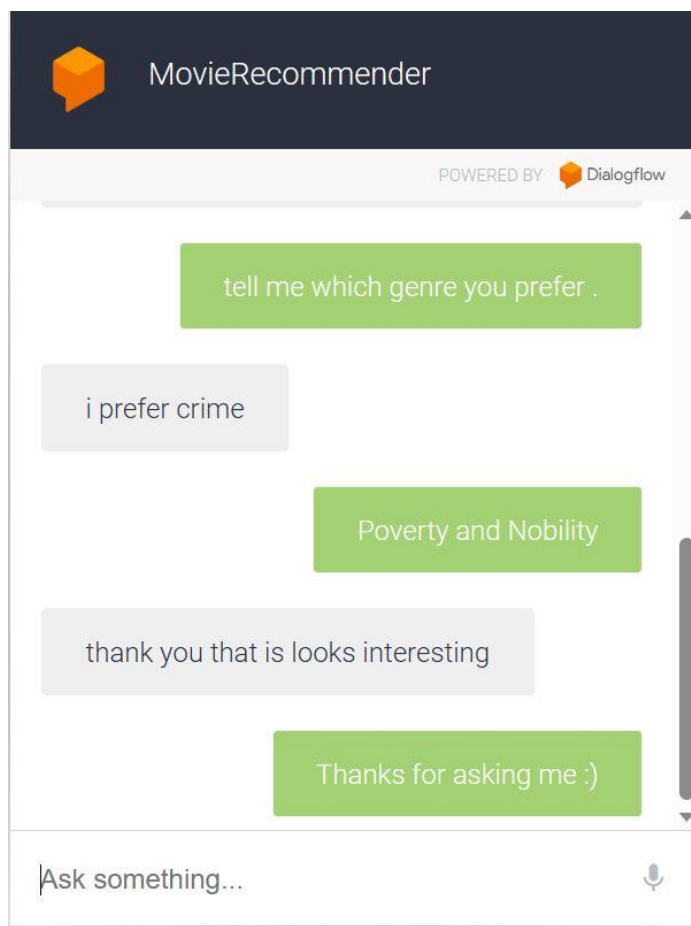
please , i need a movie related to ma favorite genre

tell me which genre you prefer .

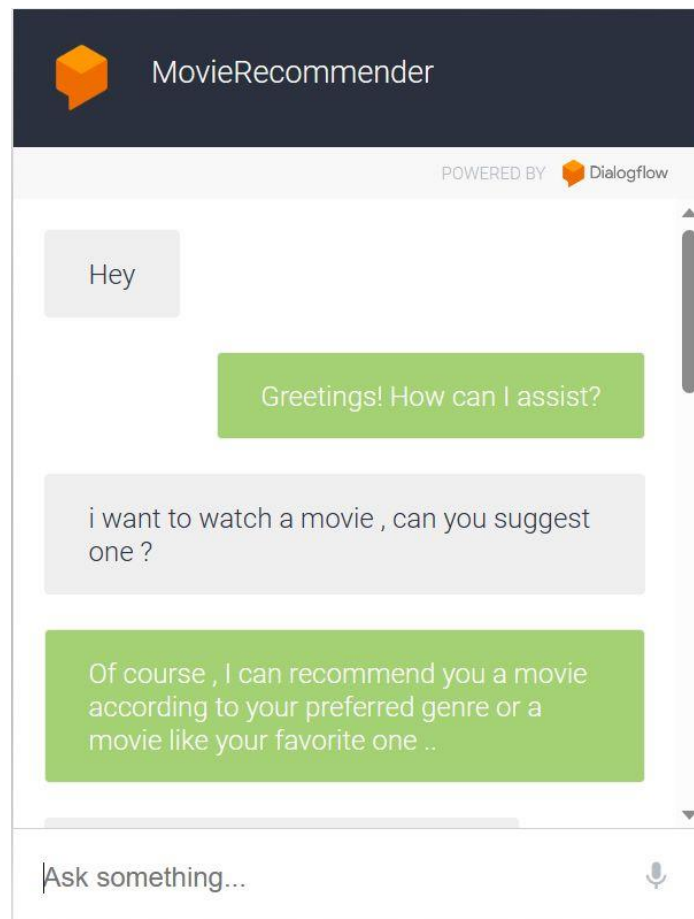
i prefer crime

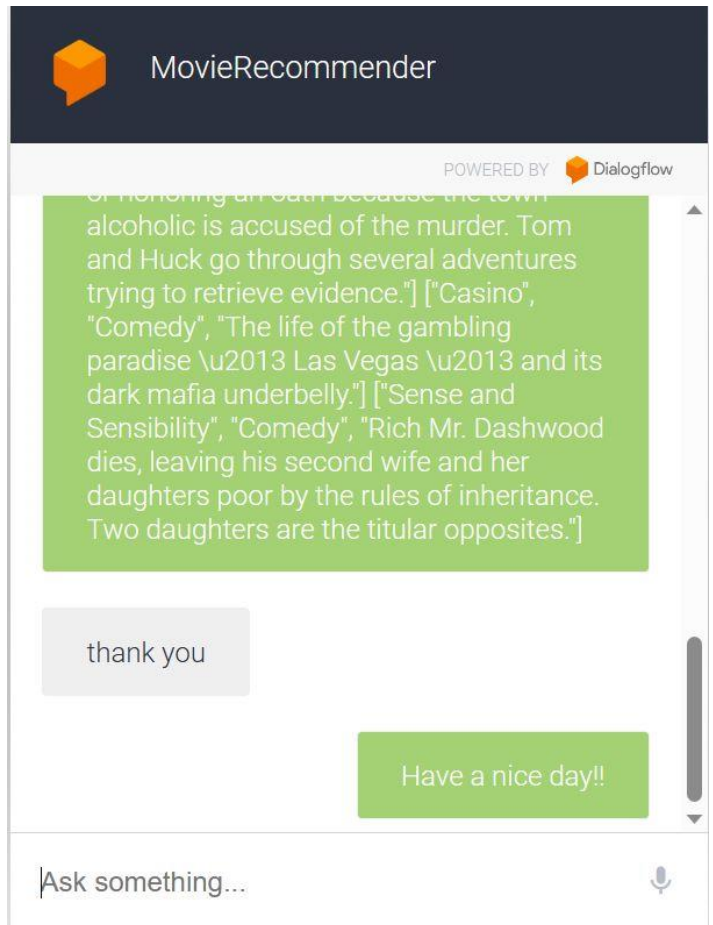
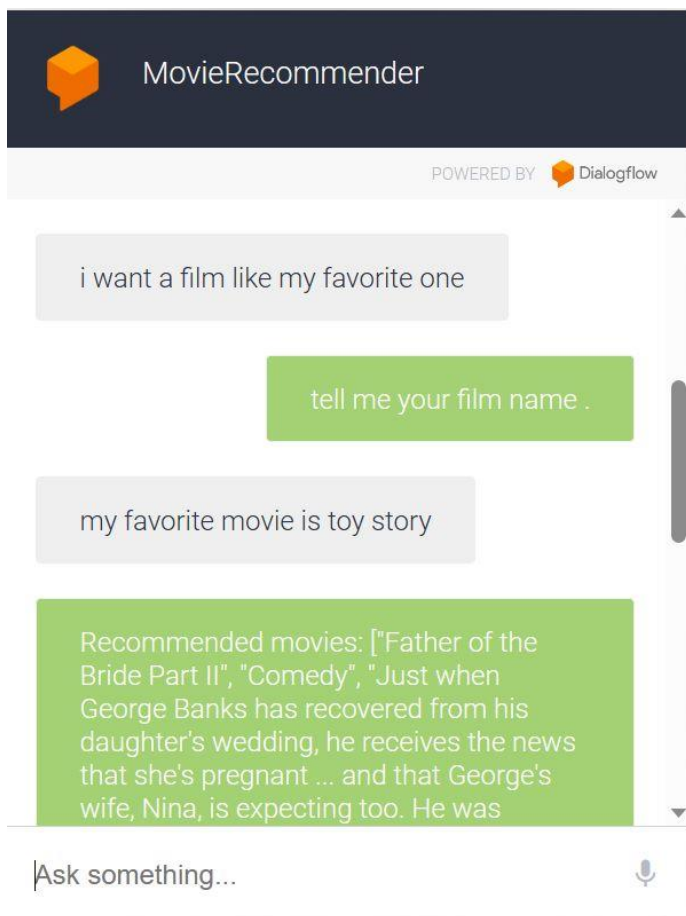
Poverty and Nobility

thank you that is looks interesting



Title based:





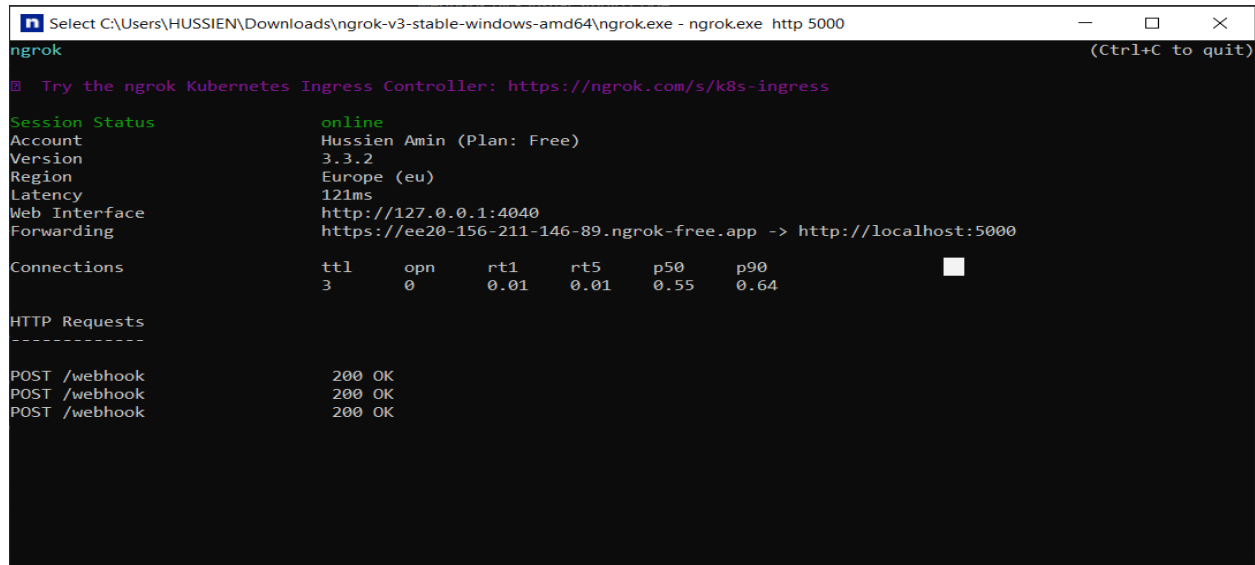
link of the web demo :

<https://bot.dialogflow.com/133b1fad-5eb0-4dc5-a5ad-8a58c71e0af5>

Deployment (Using Flask) :

- We use ngrok to connect flask with dialog flow :

These samples of requests in ngrok :



The screenshot shows a terminal window for ngrok. The title bar indicates the path to the ngrok.exe file and the listening port (http 5000). The terminal output includes a link to the ngrok Kubernetes Ingress Controller, session status (online), account details (Hussien Amin, Plan: Free), version (3.3.2), region (Europe (eu)), latency (121ms), web interface (http://127.0.0.1:4040), and forwarding URL (https://ee20-156-211-146-89.ngrok-free.app -> http://localhost:5000). A table of connections shows 3 total, 0 open, and various round-trip and processing times. Below, HTTP requests are listed as POST /webhook with 200 OK responses.

```
ngrok
Try the ngrok Kubernetes Ingress Controller: https://ngrok.com/s/k8s-ingress

Session Status      online
Account             Hussien Amin (Plan: Free)
Version             3.3.2
Region              Europe (eu)
Latency             121ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://ee20-156-211-146-89.ngrok-free.app -> http://localhost:5000

Connections
  ttl    opn    rt1    rt5    p50    p90
    3     0    0.01   0.01   0.55   0.64

HTTP Requests
-----
POST /webhook      200 OK
POST /webhook      200 OK
POST /webhook      200 OK
```

- We made route webhook that handles incoming POST requests from Dialogflow and processes the user's query, it retrieves query as JSON

```
import urllib
import json
import os
from flask import Flask , request , make_response , jsonify
from prediction import prediction , movies_recommender
from genre_prediction import build_chart
app = Flask(__name__)

@app.route('/webhook', methods = ['POST', 'GET'])
def webhook():
    if request.method == "POST":
        req = request.get_json(silent = True , force = True)
        res = req['queryResult']['parameters']
        res = processRequest(req)
        res = json.dumps(res , indent=4)
        r = make_response(res)
        r.headers['Content-Type'] = 'application/json'
        return r
```


- Retrieved query :

```

from genre_prediction import build_chart

* Debugger is active!
* Debugger PIN: 752-930-891

{'queryText': 'i have a preferd genre , which is crime', 'action': 'genres', 'parameters': {'genres_list': 'Crime'}, 'allRequiredParamsPresent': True, 'fulfillmentMessages': [{'text': {'text': ['']}}], 'outputContexts': [{'name': 'projects/movierecommender-rsqm/agent/sessions/527c314d-6215-d2b2-4f24-06e162fcee39/contexts/_system_counters_', 'parameters': {'no-input': 0.0, 'no-match': 0.0, 'genres_list': 'Crime', 'genres_list.original': 'crime'}}], 'intent': {'name': 'projects/movierecommender-rsqm/agent/intents/98af1dd8-b78a-4222-a821-d340d1f2dffb8', 'displayName': 'genres', 'intentDetectionConfidence': 0.698951, 'languageCode': 'en', 'sentimentAnalysisResult': {'queryTextSentiment': {'score': -0.4, 'magnitude': 0.4}}}]
127.0.0.1 - - [09/Aug/2023 00:13:16] "POST /webhook HTTP/1.1" 200 -

{'queryText': 'i like toy story', 'action': 'movie_ac', 'parameters': {}, 'allRequiredParamsPresent': True, 'fulfillmentMessages': [{'text': {'text': ['']}}], 'outputContexts': [{'name': 'projects/movierecommender-rsqm/agent/sessions/527c314d-6215-d2b2-4f24-06e162fcee39/contexts/_system_counters_', 'lifespanCount': 1, 'parameters': {'no-input': 0.0, 'no-match': 0.0}}], 'intent': {'name': 'projects/movierecommender-rsqm/agent/intents/f25863de-0c37-45d0-8f0c-b66fd9a6a4aa', 'displayName': 'movie_name', 'intentDetectionConfidence': 1.0, 'languageCode': 'en', 'sentimentAnalysisResult': {'queryTextSentiment': {'score': 0.9, 'magnitude': 0.9}}}]
127.0.0.1 - - [09/Aug/2023 00:14:40] "POST /webhook HTTP/1.1" 200 -

{'queryText': 'i really prefer toy story', 'action': 'movie_ac', 'parameters': {}, 'allRequiredParamsPresent': True, 'fulfillmentMessages': [{'text': {'text': ['']}}], 'outputContexts': [{'name': 'projects/movierecommender-rsqm/agent/sessions/527c314d-6215-d2b2-4f24-06e162fcee39/contexts/_system_counters_', 'parameters': {'no-input': 0.0, 'no-match': 0.0}}], 'intent': {'name': 'projects/movierecommender-rsqm/agent/intents/f25863de-0c37-45d0-8f0c-b66fd9a6a4aa', 'displayName': 'movie_name', 'intentDetectionConfidence': 0.7623979, 'languageCode': 'en', 'sentimentAnalysisResult': {'queryTextSentiment': {'score': 0.1, 'magnitude': 0.1}}}]
127.0.0.1 - - [09/Aug/2023 00:14:54] "POST /webhook HTTP/1.1" 200 -

```

- This function return response from flask to dialog flow :

If action_name = genres

It goes to genre_prediction file and return prediction according to genre

If action_name = movie_ac

It goes to prediction file and return prediction according to movie title , it return list of four movies that in the same genre of movie the user preferred

```

def processRequest(req):
    query_response = req['queryResult']
    action_name = query_response.get('action')
    print(query_response)
    text = query_response.get('queryText', None)
    parameters = query_response.get('parameters', None)

    if action_name == 'genres':
        speech1 = list(build_chart(text)['title'].head(1))
        return {"fulfillmentText": speech1}
    elif action_name == 'movie_ac':
        recommended_movies = prediction(text)
        recommended = movies_recommender(recommended_movies)
        # Prepare the response to be sent back to Dialogflow
        recommended_json_str = "\n".join(json.dumps(item) for item in recommended)
        speech = "Recommended movies: \n" + recommended_json_str
        return {"fulfillmentText": speech}

```

- Importing our new data set after preprocessing and made labels

```
data = pd.read_csv("C:/Users/HUSSIEN/Downloads/merged.csv")
```

✓ 1.1s

- This function made Recommendation of a movie according to genre

```
def build_chart(genre, percentile=0.85):
    full_text_lower = genre.lower()
    matching_titles = data[data['genres'].apply(lambda title: len(title) >= 3 and title.lower() in full_text_lower)]
    matching_titles_list = matching_titles['genres'].tolist()
    for title in matching_titles_list:
        new_title = title

    df = data[data['genres'] == new_title]
    vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(percentile)

    qualified = df[(df['vote_count'] >= m) & (df['vote_count'].notnull()) & (df['vote_average'].notnull())][['title', 'vote_count', 'vote_average', 'popul
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')

    qualified['wr'] = qualified.apply(lambda x: (x['vote_count']/(x['vote_count']+m) * x['vote_average']) + (m/(m+x['vote_count']) * C), axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(250)

    return qualified
```

✓ 0.0s

Python

- We imported pickle of tfidf to use it to convert movie overview and give matrix of this overview to our champion model

```
import pickle

# Load the TF-IDF matrix from the file
with open("C:/Users/HUSSIEN/Downloads/tfidf.pkl", "rb") as file:
    tf = pickle.load(file)
```

✓ 0.6s

- This function take title of movie from user and return overview from data set of this title

```
def overview(title):
    full_text_lower = title.lower()
    # Convert titles in DataFrame to lowercase and search for matching titles
    # Search for matching titles with at least 3 characters
    matching_titles = merged[merged['original_title'].apply(lambda title: len(title) >= 3 and title.lower() in full_text_lower)]
    # Collect matching titles in a list
    matching_titles_list = matching_titles['original_title'].tolist()
    for title in matching_titles_list:
        new_title = title
    index = merged[merged['title'] == new_title].index[0]
    overview = merged.at[index, 'overview_filtered']
    return overview
```

✓ 0.0s

- Import pickle of our champion model (random forest) to predict which cluster this title belong to

```
# load random forest model
with open("C:/Users/HUSSIEN/Downloads/random_forest_model.pkl", 'rb') as f:
    model = pickle.load(f)
```

✓ 1.8s

- this function predict which cluster this title belong to

```
def prediction(title):
    overview_list = [overview(title)]
    tfidf_data = pd.DataFrame(tf.transform(overview_list).toarray())
    # Get the column names
    column_names = tf.get_feature_names_out()

    # Assign the column names to the DataFrame
    tfidf_data.columns = column_names
    predictions = model.predict(tfidf_data)
    return predictions
```

✓ 0.0s

- this function recommends list of four movies according to the title that we took from user and these movies in the same cluster (genre) of this title

```
def movies_recommender(predictions):  
    for i in range(9):  
        if i == predictions:  
            condition = merged['label'] == i  
            selected_rows = merged.loc[condition]  
            selected = selected_rows[['original_title', 'genres', 'overview']].head(4)  
            selected_rows_list = selected.to_dict(orient='records')  
            selected_rows_list = selected.values.tolist()  
            # list_of_lists = [[dictionary] for dictionary in selected_rows_list]  
    return selected_rows_list
```

✓ 0.0s