GRUPO: Jesus Martin y Jorve Arevalo G13

Práctica 2

Vamos a ver en la práctica los conceptos vistos en clase sobre Recomendadores.

- En la primera parte veremos algunos conceptos de la librería Pandas de python ejemplificados en lo que hemos llamado Recomendador Simple, que ordena y muestra las películas en el TOP de popularidad pero no realiza recomendaciones personalizadas, es decir, a todos los usuarios les recomienda la misma lista de películas (como en el TOP 10 de Netflix).
- En la segunda parte veremos un recomendador personalizado basado en contenidos y
- En la tercera parte veremos los recomendadores por filtrado colaborativo usando la librería SURPRISE y haciendo hincapié en como evaluar los resultados de la recomendación.

Primera parte. Recomendacion simple por popularidad

Esta primera parte está inspirada en Rounak Banik, 2018 Recommender Systems in Python: Beginner Tutorial DE DATACAMP.COM Vamos a utilizar los dos datasets de MovieLens que se pueden descargar de: https://www.kaggle.com/rounakbanik/the-movies-dataset (https://www.kaggle.com/rounakbanik/the-movies-dataset)

- 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.

En esta primera parte solo tienes que ejecutar las celdas del tutorial y responder las preguntas del final. No se pide implementar nada extra

Recomendador simple basado en popularidad

Vamos a realizar una version simple de un Recomendador que ofrece recomendaciones generalizadas a todos los usuarios según la popularidad de la película y el género. La idea básica detrás de este recomendador es que las películas que son más populares y más aclamadas por la crítica tendrán una mayor probabilidad de gustar a la audiencia promedio. Este modelo no da recomendaciones personalizadas en base a los gustos de un usuario concreto.

La implementación de este modelo muy sencilla. Basta ordenar las películas según las calificaciones y la popularidad y mostrar las mejores películas de nuestra lista. Como paso adicional, podemos pasar un argumento de género para filtrar y mostrar solo las mejores películas de un género en particular.

Los pasos que vamos a realizar en esta primera versión son los siguientes:

- Decidir cual es el criterio que nos permite ordenar las peliculas recomendadas.
- Calcular esa métrica para cada pelicula de la lista total de películas.
- Ordenar las películas y mostrar las mejores.

Cargamos los datos del archivo csv de peliculas usando pandas. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

Pandas es un paquete de Python que proporciona estructuras de datos tabulares con columnas de tipo heterogéneo con etiquetas en columnas y filas. Estas tablas se conocen como DataFrames. Un dataframe es una estructura de datos con dos dimensiones en la cual se puede guardar datos de distintos tipos (como caractéres, enteros, valores de punto flotante, factores y más) en columnas. Es similar a una hoja de cálculo o una tabla de SQL

In [9]:
```
# Importamos Pandas
import pandas as pd
import warnings;
warnings.simplefilter('ignore')  # para evitar algunos warnings al cargar los dat
os.

# Load Movies Metadata in a dataframe
md = pd.read_csv('./data/movies_metadata.csv', low_memory=False)

# se puede imprimir solo las primeras 3 filas para ver como son los datos.
md.head(3)
#md.head()
```

Out[9]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt0113228

3 rows × 24 columns

Utilizo las clasificaciones de The Movie Database (TMDb) para crear nuestra tabla de películas principales. Vamos a utilizar la fórmula de calificación ponderada de IMDB Internet Movie Database.

Calificación ponderada o Weighted Rating (WR) = $(\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$ dónde,

- v es el número de votos para la película
- m son los votos mínimos requeridos para aparecer en la tabla
- R es la calificación promedio de la película
- C es el voto medio

El siguiente paso es determinar un valor apropiado para m, los votos mínimos requeridos para aparecer en la tabla. Usaremos el percentil 95 como nuestro límite. En otras palabras, para que una película aparezca en las listas, debe tener más votos que al menos el 95% de las películas de la lista.

Vamos a hacer una tabla Top 250 por popularidad y otra para seleccionar las películas de un genero en particular.

In [10]:
```
from ast import literal_eval
import numpy as np
import matplotlib.pyplot as plt
```

In [11]:
```
## obtenemos la media de las puntuaciones de las películas de la tabla.
vote_counts = md[md['vote_count'].notnull()]['vote_count'].astype('int')
vote_averages = md[md['vote_average'].notnull()]['vote_average'].astype('int')
C = vote_averages.mean()
C
```

Out[11]:
```
5.244896612406511
```

In [12]:
```
# cuantas cumplen el percentil 0,95
m = vote_counts.quantile(0.95)
m
```

Out[12]:
```
434.0
```

Para que una película sea elegida tiene que tener por los menos 434 votos en TMDB (95%). Tambien vemos que la media de rating de las peliculas en TMDB es de 5.244 en una escala de 10. Hay 2274 peliculas que cumplen estos criterios.

In [13]:
```
# Utilizo la columna release_date para construir una columna year, porque solo me
interesa el año.
# No es necesario comprender todos los detalles pero puedes consultar la document
ación de panda.

md['year'] = pd.to_datetime(md['release_date'], errors='coerce').apply(lambda x:
str(x).split('-')[0] if x != np.nan else np.nan)
```

In [14]:
```
# limpiamos la lista de generos para que se quede solo con el nombre en vez de la
estructura [{'id': 16, 'name': 'Animation'}. Sólo ejecutar una vez.
md['genres'] = md['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i[
'name'] for i in x] if isinstance(x, list) else [])
```

Vamos a mostrar el resultado despues de modificar la columna genres y añadir la columna year para el año.

In [15]:
```
md.head(3)
```

Out[15]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[Animation, Comedy, Family]	http://toystory.disney.com/toy-story	862	tt0114709
1	False	NaN	65000000	[Adventure, Fantasy, Family]	NaN	8844	tt0113497
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[Romance, Comedy]	NaN	15602	tt0113228

3 rows × 25 columns

In [16]:
```
seleccionadas = md[(md['vote_count'] >= m) & (md['vote_count'].notnull()) & (md['vote_average'].notnull())][['title', 'year', 'vote_count', 'vote_average', 'popularity', 'genres']]
seleccionadas['vote_count'] = seleccionadas['vote_count'].astype('int')
seleccionadas['vote_average'] = seleccionadas['vote_average'].astype('int')
seleccionadas.shape
# shape devuelve el numero filas y columnas de la tabla de peliculas seleccionadas.
```

Out[16]:
```
(2274, 6)
```

In [17]:
```
seleccionadas.head(3)
```

Out[17]:

	title	year	vote_count	vote_average	popularity	genres
0	Toy Story	1995	5415	7	21.946943	[Animation, Comedy, Family]
1	Jumanji	1995	2413	6	17.015539	[Adventure, Fantasy, Family]
5	Heat	1995	1886	7	17.924927	[Action, Crime, Drama, Thriller]

In [18]:
```
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [19]:
```
seleccionadas['wr'] = seleccionadas.apply(weighted_rating, axis=1)
```

In [20]:
```
seleccionadas = seleccionadas.sort_values('wr', ascending=False).head(250)
```

TOP 15 de peliculas

In [21]:
```
seleccionadas.head(15)
#mostramos el TOP 15
```

Out[21]:

	title	year	vote_count	vote_average	popularity	genres	wr
15480	Inception	2010	14075	8	29.108149	[Action, Thriller, Science Fiction, Mystery, A...	7.917588
12481	The Dark Knight	2008	12269	8	123.167259	[Drama, Action, Crime, Thriller]	7.905871
22879	Interstellar	2014	11187	8	32.213481	[Adventure, Drama, Science Fiction]	7.897107
2843	Fight Club	1999	9678	8	63.869599	[Drama]	7.881753
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.070725	[Adventure, Fantasy, Action]	7.871787
292	Pulp Fiction	1994	8670	8	140.950236	[Thriller, Crime]	7.868660
314	The Shawshank Redemption	1994	8358	8	51.645403	[Drama, Crime]	7.864000
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.324358	[Adventure, Fantasy, Action]	7.861927
351	Forrest Gump	1994	8147	8	48.307194	[Comedy, Drama, Romance]	7.860656
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.423537	[Adventure, Fantasy, Action]	7.851924
256	Star Wars	1977	6778	8	42.149697	[Adventure, Action, Science Fiction]	7.834205
1225	Back to the Future	1985	6239	8	25.778509	[Adventure, Comedy, Science Fiction, Family]	7.820813
834	The Godfather	1972	6024	8	41.109264	[Drama, Crime]	7.814847
1154	The Empire Strikes Back	1980	5998	8	19.470959	[Adventure, Action, Science Fiction]	7.814099
46	Se7en	1995	5915	8	18.45743	[Crime, Mystery, Thriller]	7.811669

Vemos que tres películas de Christopher Nolan: **Inception**, **The Dark Knight** e **Interstellar** son las preferidas por popularidad en la parte superior de nuestra lista de TOP 15. En la tabla se ve un fuerte sesgo de los usuarios de TMDB hacia géneros y directores particulares, pero de los sesgos no hablaremos en esta práctica.

Vamos a construir una funcion que permite filtrar y construir una tabla de peliculas recomendadas para generos particulares.

In [22]:
```
# Separamos las peliculas que tienen varios generos de forma que aparezcan varias filas de la misma película.
# una por cada genero.
# De nuevo no hace falta entender todos los detalles. Sólo observar el comportamiento en la tabla resultado.

s = md.apply(lambda x: pd.Series(x['genres']),axis=1).stack().reset_index(level=1, drop=True)
s.name = 'genre'
gen_md = md.drop('genres', axis=1).join(s)
```

In [23]:
```
gen_md.head(3)
```

Out[23]:

	adult	belongs_to_collection	budget	homepage	id	imdb_id	original_langua
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	http://toystory.disney.com/toy-story	862	tt0114709	
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	http://toystory.disney.com/toy-story	862	tt0114709	
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	http://toystory.disney.com/toy-story	862	tt0114709	

3 rows × 25 columns

In [24]:
```
# Metemos todo el código anterior en una función.
# Vamos a relajar nuestras condiciones predeterminadas al percentil 85 en lugar d
e 95 para tener más generos.

def construye_tabla(genre, percentile=0.85):
    df = gen_md[gen_md['genre'] == genre]
    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)

    seleccionadas = df[(df['vote_count'] >= m) & (df['vote_count'].notnull()) & (
df['vote_average'].notnull())][['title', 'year', 'vote_count', 'vote_average', 'p
opularity']]
    seleccionadas['vote_count'] = seleccionadas['vote_count'].astype('int')
    seleccionadas['vote_average'] = seleccionadas['vote_average'].astype('int')

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

    return seleccionadas
```

Veamos nuestro método en acción mostrando las 15 mejores películas románticas. Con esto forzamos el genero que nos interesa y resolvemos el problema de que el romance casi no figuraba en nuestro Top genérico a pesar de ser uno de los géneros de películas más populares.

TOP Películas románticas

In [25]:
```
construye_tabla('Romance').head(15)
```

Out[25]:

	title	year	vote_count	vote_average	popularity	wr
10309	Dilwale Dulhania Le Jayenge	1995	661	9	34.457024	8.565285
351	Forrest Gump	1994	8147	8	48.307194	7.971357
876	Vertigo	1958	1162	8	18.20822	7.811667
40251	Your Name.	2016	1030	8	34.461252	7.789489
883	Some Like It Hot	1959	835	8	11.845107	7.745154
1132	Cinema Paradiso	1988	834	8	14.177005	7.744878
19901	Paperman	2012	734	8	7.198633	7.713951
37863	Sing Street	2016	669	8	10.672862	7.689483
882	The Apartment	1960	498	8	11.994281	7.599317
38718	The Handmaiden	2016	453	8	16.727405	7.566166
3189	City Lights	1931	444	8	10.891524	7.558867
24886	The Way He Looks	2014	262	8	5.711274	7.331363
45437	In a Heartbeat	2017	146	8	20.82178	7.003959
1639	Titanic	1997	7770	7	26.88907	6.981546
19731	Silver Linings Playbook	2012	4840	7	14.488111	6.970581

Hasta aquí lo único que hemos hecho es ordenar el catalogo. Las recomendaciones son generales (no personalizadas) por popularidad, pero lo que nosotros queremos hacer son recomendaciones personalizadas para un usuario del que sabemos su perfil, sus preferencias, alguna película que le ha gustado. La siguiente parte corresponde al recomendador basado en contenido que dada una película (puede ser usada como perfil) recomienda peliculas similares. En la tercera parte veremos el filtrado colaborativo.

Para el Recomendador simple hemos usado las películas del dataset completo, mientras que para el sistema de recomendación personalizado haremos uso del segundo data set (Small)

Ejercicio parte 1.

Realiza algunas pruebas de recomendaciones por popularidad y comenta los resultados obtenidos. Incluye los resultados y los comentarios en este archivo.

¿Te parece útil esta forma de recomendación de contenidos? ¿Sería útil también para otras aplicaciones, por ejemplo, para recomendar libros o restaurantes?

In [26]:
```
construye_tabla('Comedy').head(10)
```

Out[26]:

	title	year	vote_count	vote_average	popularity	wr
10309	Dilwale Dulhania Le Jayenge	1995	661	9	34.457024	8.463024
351	Forrest Gump	1994	8147	8	48.307194	7.963363
1225	Back to the Future	1985	6239	8	25.778509	7.952358
18465	The Intouchables	2011	5410	8	16.086919	7.945207
22841	The Grand Budapest Hotel	2014	4644	8	14.442048	7.936384
2211	Life Is Beautiful	1997	3643	8	39.39497	7.919430
732	Dr. Strangelove or: How I Learned to Stop Worr...	1964	1472	8	9.80398	7.809073
3342	Modern Times	1936	881	8	8.159556	7.695554
883	Some Like It Hot	1959	835	8	11.845107	7.680781
1236	The Great Dictator	1940	756	8	9.241748	7.651762

In [27]:
```
construye_tabla('Adventure').head(10)
```

Out[27]:

	title	year	vote_count	vote_average	popularity	wr
15480	Inception	2010	14075	8	29.108149	7.906526
22879	Interstellar	2014	11187	8	32.213481	7.883426
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.070725	7.854939
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.324358	7.843867
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.423537	7.832647
256	Star Wars	1977	6778	8	42.149697	7.812801
1225	Back to the Future	1985	6239	8	25.778509	7.797828
1154	The Empire Strikes Back	1980	5998	8	19.470959	7.790329
5481	Spirited Away	2001	3968	8	41.048867	7.695056
9698	Howl's Moving Castle	2004	2049	8	16.136048	7.465435

In [28]:
```
construye_tabla('Crime').head(20)
```

Out[28]:

	title	year	vote_count	vote_average	popularity	wr
12481	The Dark Knight	2008	12269	8	123.167259	7.957677
292	Pulp Fiction	1994	8670	8	140.950236	7.940522
314	The Shawshank Redemption	1994	8358	8	51.645403	7.938355
834	The Godfather	1972	6024	8	41.109264	7.915273
46	Se7en	1995	5915	8	18.45743	7.913765
586	The Silence of the Lambs	1991	4549	8	4.307222	7.889007
289	Leon: The Professional	1994	4293	8	20.477329	7.882696
3030	The Green Mile	1999	4166	8	19.96678	7.879291
1057	Reservoir Dogs	1992	3821	8	12.22034	7.868957
1178	The Godfather: Part II	1974	3418	8	36.629307	7.854398
49	The Usual Suspects	1995	3334	8	16.302466	7.850946
1170	GoodFellas	1990	3211	8	15.424092	7.845585
4135	Scarface	1983	3017	8	11.299673	7.836299
109	Taxi Driver	1976	2632	8	14.092713	7.814116
5878	City of God	2002	1852	8	14.95927	7.743770
1184	Once Upon a Time in America	1984	1104	8	32.182851	7.597811
5157	Rashomon	1950	471	8	9.887355	7.223475
1215	M	1931	465	8	12.752421	7.216563
42015	The Invisible Guest	2016	395	8	13.90698	7.125783
1836	On the Waterfront	1954	368	8	18.211093	7.084882

Me parece una buena forma de recomendar para usuarios generales, como por ejemplo paginas web de recomendacion de peliculas, o para usuarios que se acaban de registrar en algun portal de peliculas (netflix, hbo...). Pero no parece tan buena para recomendaciones específicas para usuarios que ya han visto muchas peliculas, personas que tengan un gusto poco comun como en peliculas de cine alternativo en las que encontrariamos mejores recomendaciones comparando con usuarios similares a el.

Para recomendar libros o restaurantes sería igual de válido, tendría los mismos pros y contras que con las películas.

Segunda parte. Recomendacion basada en contenido

El recomendador que construimos en la sección anterior da la misma recomendación a todos, independientemente del gusto personal del usuario. Si una persona que le gustan las películas románticas (y odia la acción) mirara nuestro Top 15, probablemente no le gustarían la mayoría de las películas. Si esta persona mira nuestras listas por género, no estaría recibiendo las mejores recomendaciones.

Por ejemplo, considere a una persona que le gustan: "Dilwale Dulhania Le Jayenge", "My Name is Khan" y "Kabhi Khushi Kabhi Gham". Si miramos los datos de estas películas podríamos inferir que a esta persona le gusta el actor Shahrukh Khan y el director Karan Johar. Incluso si él / ella tuviera acceso a la tabla de películas románticas no las encontraría como las principales recomendaciones, porque no son películas populares.

Para personalizar más nuestras recomendaciones, voy a crear un motor que calcule la similitud entre películas en función de ciertas métricas y sugiera películas que son más similares a una película en particular que le gustó a un usuario. Como hemos visto en teoría este tipo de recomendaciones se conocen como **recomendación o filtrado basado en contenido**.

Vamos a ver cómo construir dos recomendadores basados en contenido:

- Usando los resúmenes textuales de las películas y taglines. Los taglines son las frases cortas que suelen aparecer en el cartel de la película, acompañando al título y a una imagen destacada y representativa de la historia que presenta.
- Usando el reparto de películas, equipo, palabras clave y género

In [29]:
```
# usaremos el conjunto pequeño de películas por eficiencia.
# Leemos los datos y limpiamos algunos valores nulos.
links_small = pd.read_csv('./data/links_small.csv')
links_small = links_small[links_small['tmdbId'].notnull()]['tmdbId'].astype('int'
)
md = md.drop([19730, 29503, 35587])
md['id'] = md['id'].astype('int')
## selecciono las peliculas de links_small con los id de la tabla de metadatos
smd = md[md['id'].isin(links_small)]
smd.shape
## shape devuelve el numero de filas y columnas.
## (9099, 25)
# Tenemos *9099* películas disponibles en nuestro conjunto de datos de metadatos
 de películas pequeñas, que es 5 veces más pequeño que nuestro conjunto de datos
 original de 45000 películas.
```

Out[29]: (9099, 25)

In [30]:
```
smd.head(5)
```

Out[30]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[Animation, Comedy, Family]	http://toystory.disney.com/toy-story	862	tt0114709
1	False	NaN	65000000	[Adventure, Fantasy, Family]	NaN	8844	tt0113497
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[Romance, Comedy]	NaN	15602	tt0113228
3	False	NaN	16000000	[Comedy, Drama, Romance]	NaN	31357	tt0114885
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[Comedy]	NaN	11862	tt0113041

5 rows × 25 columns

Tenemos que medir la similitud de las películas con las preferencias del usuario. En este caso usaremos como preferencia una pelicula que le haya gustado antes para recuperar (por similitud) otras películas parecidas. Elegimos usar similitud usando el atributo de descripción (textual) y el lema (tambien textual). Para calcular la similitud entre dos textos usamos el modelo del espacio vectorial.

In [31]:
```
smd['tagline'] = smd['tagline'].fillna('')
#El método fillna permite sustituir los valores nulos de una estructura pandas

smd['description'] = smd['overview'] + smd['tagline']
# creamos un campo llamado descrition con la
smd['description'] = smd['description'].fillna('')
```

In [32]:
```
smd.head(3)
```

Out[32]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[Animation, Comedy, Family]	http://toystory.disney.com/toy-story	862	tt0114709
1	False	NaN	65000000	[Adventure, Fantasy, Family]	NaN	8844	tt0113497
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[Romance, Comedy]	NaN	15602	tt0113228

3 rows × 26 columns

In [33]:
```
print(smd['description'])
```

```
0        Led by Woody, Andy's 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 ...
                               ...
40224    From the mind behind Evangelion comes a hit la...
40503    The band stormed Europe in 1963, and, in 1964,...
44821    When Molly Hale's sadness of her father's disa...
44826    All your favorite Pokémon characters are back,...
45265    While holidaying in the French Alps, a Swedish...
Name: description, Length: 9099, dtype: object
```

Similitud del coseno

Utilizamos el vectorizador TF-IDF Para calcular un valor de similitud se puede usar **cosine_similarity** o **linear_kernel** de sklearn. La segunda es más eficiente. https://scikit-learn.org/stable/modules/metrics.html (https://scikit-learn.org/stable/modules/metrics.html)

La medida de similitud del coseno calcula un valor numerico que representa la similitud entre dos vectores de términos. En este caso los vectores representan los textos de la descripción normalizados.

$$cosine(x, y) = \frac{x.y^\intercal}{||x||.||y||}$$

In [34]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='eng
lish')
tfidf_matrix = tf.fit_transform(smd['description'])
```

In [35]:
```
tfidf_matrix.shape
# filas, columnas
```

Out[35]:
```
(9099, 268124)
```

In [36]:
```
#cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [37]:
```
cosine_sim[0]
```

Out[37]:
```
array([1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913,
       0.        ])
```

Ahora tenemos una matriz de similitud de coseno por pares para todas las películas en nuestro conjunto de datos. El siguiente paso es escribir una función que devuelva las 30 películas más similares según la puntuación de similitud del coseno.

In [38]:
```
smd = smd.reset_index()
#reset_index resets the index column) to a regular column and uses a numeric sequ
ence as new index.

titles = smd['title']
indices = pd.Series(smd.index, index=smd['title'])
#Un objeto "Series" es un vector con datos indexados
```

In [39]:
```
indices.head(3)
```

Out[39]:
```
title
Toy Story           0
Jumanji             1
Grumpier Old Men    2
dtype: int64
```

In [40]:
```
def get_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

In [41]:
```
get_recommendations('The Godfather').head(10)
```

Out[41]:
```
973      The Godfather: Part II
8387                  The Family
3509                        Made
4196         Johnny Dangerously
29               Shanghai Triad
5667                        Fury
2412              American Movie
1582    The Godfather: Part III
4221                    8 Women
2159               Summer of Sam
Name: title, dtype: object
```

Ejercicio parte 2:

Prueba algunas consultas y comenta los resultados. Compara los resultados con el tipo de recomendaciones realizadas en la parte 1.

In [42]:
```
get_recommendations('Toy Story').head(20)
```

Out[42]:
```
2502                    Toy Story 2
7535                    Toy Story 3
6193         The 40 Year Old Virgin
2547                Man on the Moon
6627                   Factory Girl
4702         What's Up, Tiger Lily?
889           Rebel Without a Cause
6554         For Your Consideration
4988                Rivers and Tides
1599                       Condorman
994                        Manhattan
402                For Love or Money
5140                  Africa Screams
437                           Malice
6107                  Life Is Sweet
3097    The Ballad of Ramblin' Jack
1792              Indecent Proposal
8330     Woody Allen: A Documentary
4383              Maid in Manhattan
4935               A Midnight Clear
Name: title, dtype: object
```

In [43]:
```
get_recommendations('American Movie').head(20)
```

Out[43]:
```
7411                       Collapse
8623    Revenge of the Green Dragons
8283                     Pain & Gain
4612                  Silk Stockings
2973                    American Pop
7584                     The Joneses
2645                     Blue Collar
5847                  The Mambo Kings
2773                         La Bamba
692                     The Godfather
7017                   Disaster Movie
2639                      Boiler Room
2192                The Color Purple
121              Up Close & Personal
1129                         Tin Men
4497                    Stone Reader
7675                 The Company Men
6212                    Why We Fight
603                         Dead Man
1252                    Wild America
Name: title, dtype: object
```

In [44]:
```
get_recommendations('Woody Allen: A Documentary').head(20)
```

Out[44]:
```
4702                 What's Up, Tiger Lily?
1440                         Wild Man Blues
994                               Manhattan
2199                            Radio Days
3046                         Love and Death
7764         American: The Bill Hicks Story
5281               Smiles of a Summer Night
865                                  Sleeper
4077                       Hollywood Ending
7204    Stanley Kubrick: A Life in Pictures
6290                             Match Point
8669                             The Rewrite
1221                                 Nowhere
7770                  Louis C.K.: Chewed Up
7602                         I'm Still Here
3594                       The Big Picture
4956                 Jesus Christ Superstar
8098                      Sleepwalk with Me
8731                               Top Five
2222                             The Raven
Name: title, dtype: object
```

Esta forma de recomendaciones es más avanzada, ya que recomendar por género o simplemente el top produce unos resultados muy amplios, mientras que con estas recomendaciones tenemos mas aspectos en cuenta ya que compara a partir de datos y peliculas que sabe que son del gusto del usuario. Lo que produce recomendaciones mas especificas y mejores desde el punto de vista de perfil de usuario, ya que puedes llegar a todo tipo de usuario, tenga los gustos que tenga, no como en las recomendaciones basadas en popularidad.

In [ ]:
```
```