# Práctica 2: Análisis de Datos en un Sistema de Información Cinematográfico

Ivan Morillas, Yani Aici

Fecha de entrega: 5 de mayo de 2024

# Contents

# 1 Introducción

Esta documentación detalla el desarrollo de una explotación de datos a partir de todos los datos que habíamos obtenido en la base de datos realizada en la práctica anterior. Se aborda diferentes técnicas y métodos analíticos para poder interpretar los datos y obtener conocimiento.

# 2 Fase 1: Análisis exploratoria de datos

Para empezar a hacer el estudio, primero hemos identificado el tipo de cada una de las variables de los datos los cuales vamos a estudiar. Empezando por los datos categóricos los cuales son: **genres**, **productionCountries**, **ageCertification** y **type**. Por otro lado tenemos los datos numéricos los cuales tenemos dos tipos: los discretos y continuos, donde de los datos numéricos discretos tenemos: **releaseYear**, **runtime** y **seasons**. Y de los datos numéricos continuos tenemos: **imdbVotes**, **imdbScore**, **tmdbPopularity** y **tmdbScore**.

Después de identificar las variables a estudiar, las hemos relacionado entre ellas para sacar conclusiones y saber si tienen alguna relación y poder agrandar el estudio mediante unas preguntas que nos hemos propuesto.

## 2.1 ¿Influye el país con la cantidad de retransmisiones?

Para saber la respuesta a esta pregunta, hemos hecho una query en Python de la Base de Datos para que calcule y muestre la cantidad de retransmisiones en cada país.

```python
def countryTitles():
    query = """
    SELECT c.country_name, COUNT(t.title_id) AS title_count
    FROM titles t
    INNER JOIN titles_countries tc ON t.title_id = tc.title_id
    INNER JOIN countries c ON tc.country_id = c.country_id
    GROUP BY c.country_name
    ORDER BY title_count DESC
    LIMIT 5;
    """
    return query
```

Figure 1: Query Country Ranking

En esta query hacemos que muestre el **countryName** juntamente con el contador de **titles**, seguidamente hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos a partir del **countryName** y en orden descendente y un limite de 5 para que muestre solo los 5 **Countries** con más retransmisiones.

El resultado de la query lo hemos representado en una gráfica de barras donde se muestra claramente la cantidad total de retransmisiones en cada país.

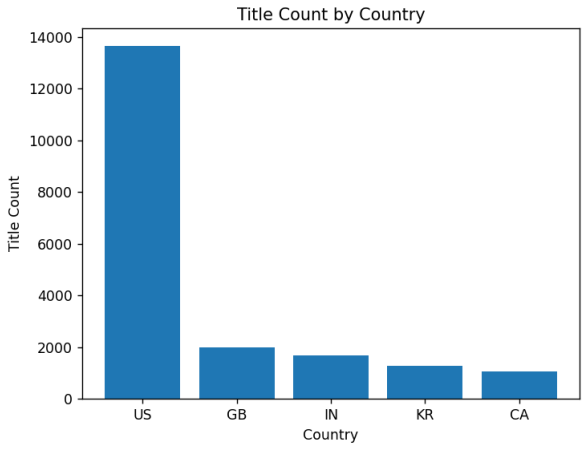



Figure 2: Country Ranking

Donde vemos, que por una gran diferencia, el país **US** tiene una gran cantidad de retransmisiones en comparación al resto de países, por lo que concluimos que podría tener una industria cinematográfica más grande y desarrollada que otros países, con mayor capacidad de producción y distribución de películas y series.

## 2.2 ¿Hay países que tienen una industria ambientada a películas?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la cantidad de películas por país.

```
def countryMovies():
    query = """
    SELECT
        c.country_name,
        COUNT(CASE WHEN t.title_type = 'MOVIE' THEN 1 END) AS movie_count
    FROM
        titles t
    JOIN
        titles_countries tc ON t.title_id = tc.title_id
    JOIN
        countries c ON tc.country_id = c.country_id
    GROUP BY
        c.country_name
    ORDER BY
        movie_count DESC
    LIMIT 5;
    """
    return query
```

Figure 3: Query Movies Country

En esta query hacemos que muestre el **countryName** y la cantidad de **Movies** que contiene, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos a partir del **countryName** y en orden descendente con un limite de 5 **Countries**.

El resultado de la query lo hemos representado en una gráfica de barras donde se muestra claramente la cantidad total de Movies en cada país.

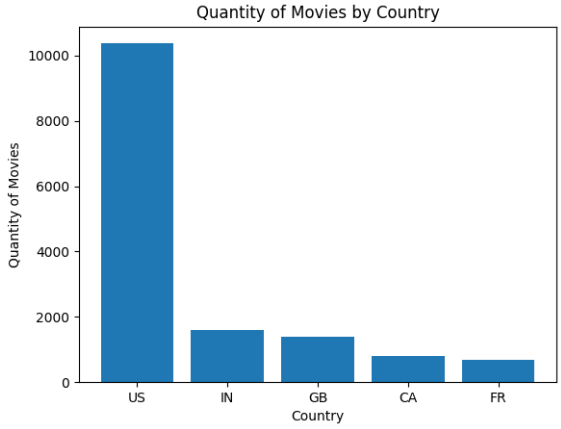



Figure 4: Movies Country Ranking

En la figure 4, se puede apreciar que el país con más retransmisiones es también el país con más películas por diferencia, por lo que podemos concluir que su industria cinematográfica de **US** se basa más en películas que en series.

## 2.3 ¿Hay países que tienen una industria ambientada a series?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la cantidad de series por país.

```
def countryShows():
    query = """
    SELECT
        c.country_name,
        COUNT(CASE WHEN t.title_type = 'SHOW' THEN 1 END) AS show_count
    FROM
        titles t
    JOIN
        titles_countries tc ON t.title_id = tc.title_id
    JOIN
        countries c ON tc.country_id = c.country_id
    GROUP BY
        c.country_name
    ORDER BY
        show_count DESC
    LIMIT 5;
    """
    return query
```

Figure 5: Query Shows Country

En esta query hacemos que muestre el **countryName** y la cantidad de **Shows** que contiene, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos a partir del **countryName** y en orden descendente con un limite de 5 **Countries**.

El resultado de la query lo hemos representado en una gráfica de barras donde se muestra claramente la cantidad total de series en cada país.

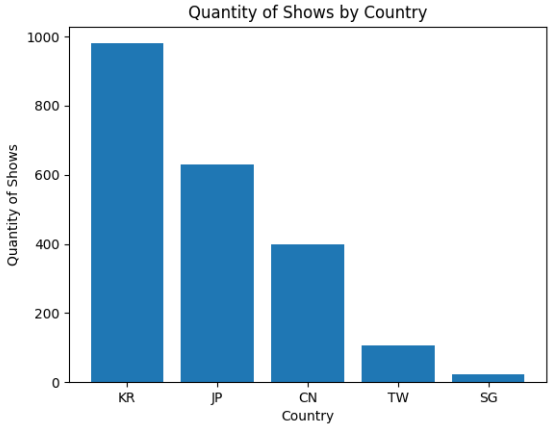



Figure 6: Shows Country Ranking

En la figure 6, se puede apreciar que el país con más series es **KR** y viendo que en la figure 2 se veia que **KR** está en la cuarta posición podemos concluir que la mayoria de países con más retransmisiones contienen más películas que series viendo que **KR** se producen más series que peliculas. También vemos que el TOP 5 son de Asia, por lo que también podemos decir que en Asia se producen más series que películas.

## 2.4 ¿Las películas tienden a ser de algún género en concreto?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la popularidad media de cada género en las películas.

```
def genrePopularityMovie():
    query = """
    WITH movie_genres AS (
    SELECT t.title, g.genre_name, t.title_tmdb_popularity
    FROM Titles t
    JOIN titles_genres tg ON t.title_id = tg.title_id
    JOIN Genres g ON tg.genre_id = g.genre_id
    WHERE t.title_type = 'Movie'
    )

    SELECT genre_name, AVG(title_tmdb_popularity) AS average_popularity
    FROM movie_genres
    GROUP BY genre_name
    ORDER BY average_popularity DESC
    LIMIT 5;
    """
    return query
```

Figure 7: Query Movie Genre

En esta query hacemos que muestre el **genreName** juntamente con la media de todas las **Movies** en base a la **tmdbPopularity**, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **genreName** y lo ordenamos por **averagePopularity** y ponemos de limite 5.

El resultado de la query lo hemos representado en una gráfica de barras donde se muestra claramente la media de popularidad de cada género de las películas.

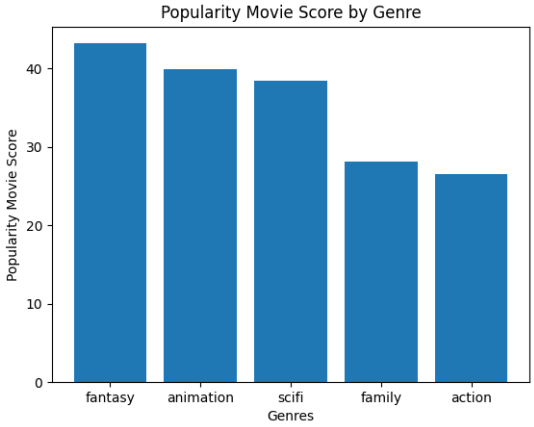



Figure 8: Genre Popularity Movie

En la figure 8, podemos apreciar que el género fantasia es el más popular entre las películas que hay en todas las plataformas de streaming.

## 2.5 ¿Las series tienden a ser de algún género en concreto?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la popularidad media de cada género en las películas.

```
def genrePopularityShow():
    query = """
    WITH movie_genres AS (
    SELECT t.title, g.genre_name, t.title_tmdb_popularity
    FROM Titles t
    JOIN titles_genres tg ON t.title_id = tg.title_id
    JOIN Genres g ON tg.genre_id = g.genre_id
    WHERE t.title_type = 'Show'
    )

    SELECT genre_name, AVG(title_tmdb_popularity) AS average_popularity
    FROM movie_genres
    GROUP BY genre_name
    ORDER BY average_popularity DESC
    LIMIT 5;
    """
    return query
```

Figure 9: Query Show Genre

En esta query hacemos que muestre el **genreName** juntamente con la media de todos los **Shows** en base a la **tmdbPopularity**, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **genreName** y lo ordenamos por **averagePopularity** y ponemos de limite 5.

El resultado de la query lo hemos representado en una gráfica de barras donde se muestra claramente la media de popularidad de cada género de las series.

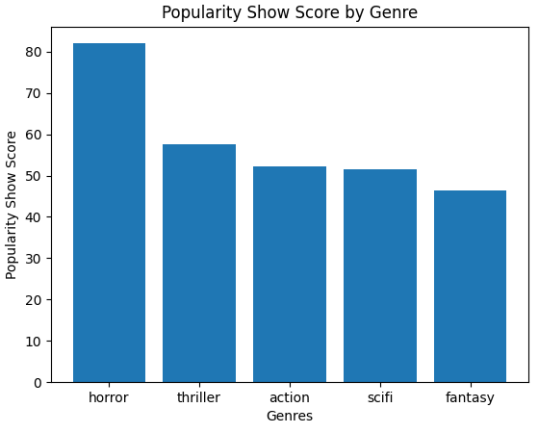



Figure 10: Genre Popularity Show

En la figure 10, podemos apreciar que el género terror es el más popular entre las series que hay en todas las plataformas de streaming.
En ambas gráficas, mostrando la popularidad de los géneros de películas y series, no hay ningún género que esté en la misma posición de popularidad, por lo que el género depende de las películas y series.

## 2.6 ¿Qué géneros son los más populares?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la popularidad media de cada género juntamente con la cantidad total de retransmisiones por género.

```
def genreScorePopularity():
    query = """
    SELECT g.genre_name,
        AVG(COALESCE(t.title_imdb_score, 0) + COALESCE(t.title_tmdb_score, 0)) AS avg_score,
        AVG(t.title_tmdb_popularity) AS avg_popularity
    FROM genres g
    INNER JOIN titles_genres tg ON g.genre_id = tg.genre_id
    INNER JOIN titles t ON tg.title_id = t.title_id
    WHERE t.title_type = 'MOVIE'
    GROUP BY g.genre_name;
    """
    return query
```

Figure 11: Query Genre Score Popularity

En esta query hacemos que muestre el **genreName** juntamente con la media de su **tmdbPopularity** y la cantidad total de **titles** de dicho género, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **genreName**.

El resultado de la query lo hemos representado en una gráfica de barras que representa la cantidad total de retransmisiones y una linea que representa la popularidad media de cada género.

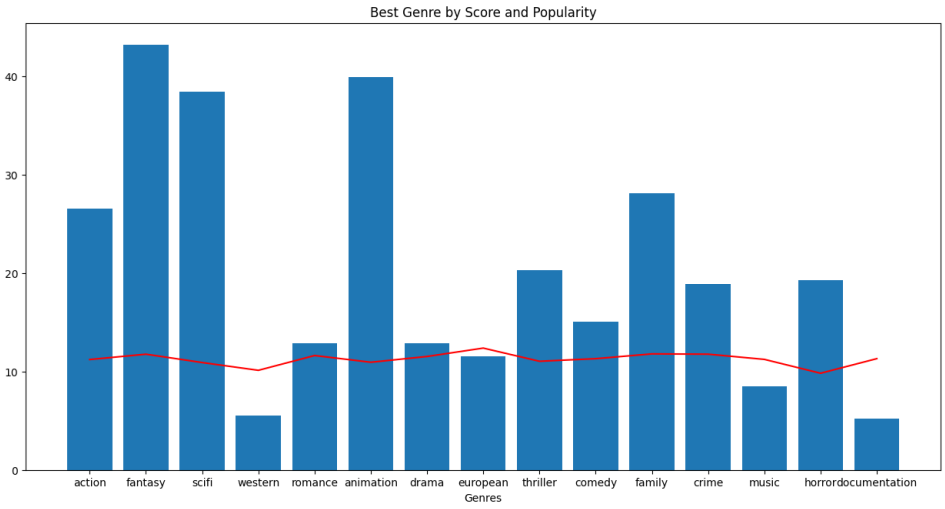



Figure 12: Genre Score Popularity

En la figure 12, podemos apreciar la comparación de ambas gráficas, donde se pueden ver algunos picos de diferencia en la popularidad que podría dar el caso a la cantidad de retransmisiones pero hay otros géneros que, por muchas o pocas retransmisiones que tenga, tienen una popularidad medianamente normal en comparación a otros géneros.

## 2.7 ¿Qué géneros son los más restringidos?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la cantidad de géneros restringidos juntamente con la cantidad total de retransmisiones por género.

```
def genreRestrictedQuantity():
    query = """
    SELECT g.genre_name,
        COUNT(t.title_id) AS total_titles,  -- Added to count total titles per genre
        COUNT(CASE WHEN a.age_name IN ('NC-17', 'TV-MA') THEN 1 END) AS restricted_titles_count
    FROM titles t
    JOIN titles_genres tg ON t.title_id = tg.title_id
    JOIN genres g ON tg.genre_id = g.genre_id
    JOIN titles_ages ta ON t.title_id = ta.title_id
    JOIN ages a ON ta.age_id = a.age_id
    GROUP BY g.genre_name
    ORDER BY restricted_titles_count DESC
    LIMIT 5;
    """
    return query
```

Figure 13: Query Genre Restriction

En esta query hacemos que muestre el **genreName** juntamente con la cantidad de **titles** con una restricción de edad los cuales tenemos **NC-17**, que significa que es para mayores de 17 años y también tenemos **TV-MA**, que es para mayores de 18 años, y la cantidad total de **titles** de dicho género, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **genreName**.

El resultado de la query lo hemos representado en una gráfica de barras que representa la cantidad total de retransmisiones y una linea que representa la cantidad de retransmisiones restringidas de cada género.

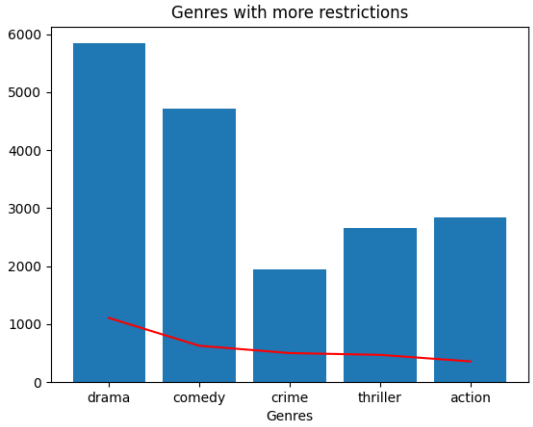



Figure 14: Genre Restriction

En la figure 14, podemos apreciar la comparación de ambas gráficas, donde se pueden ver que el género más restringido es el drama, ya por su contenido sexual y también por la cantidad de retransmisiones, hace que sea el más restringido.

## 2.8 ¿Las plataformas Streaming tienen contenido de varios países?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la cantidad de países que contienen cada plataforma Streaming juntamente con la cantidad total de retransmisiones tanto de películas como de series.

```
def countryStreamingQuantity():
    query = """
    SELECT s.streaming_name,
        COUNT(DISTINCT c.country_id) AS num_countries,
        COUNT(*) AS num_titles
    FROM streamings s
    INNER JOIN titles_streamings ts ON s.streaming_id = ts.streaming_id
    LEFT JOIN titles_countries tc ON ts.title_id = tc.title_id
    LEFT JOIN countries c ON tc.country_id = c.country_id
    GROUP BY s.streaming_name;
    """
    return query
```

Figure 15: Query Streaming Country

En esta query hacemos que muestre el **streamingName** juntamente con su cantidad de **countries** y su cantidad de **titles** que tiene, para poder ver la proporción de ambos, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **streamingName**.

El resultado de la query lo hemos representado en una gráfica de barras que representa la cantidad total de retransmisiones y una linea que representa la cantidad de países de cada plataforma Streaming.

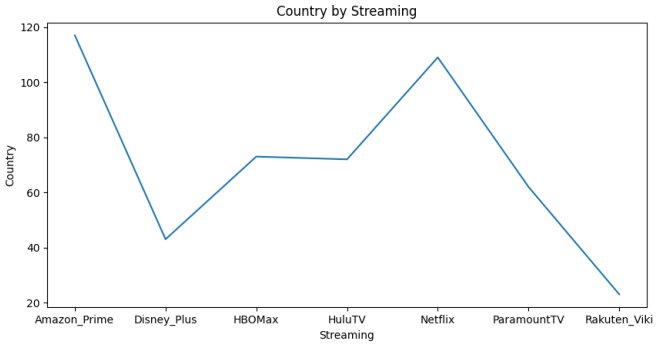



Figure 16: Streaming Country

En la figure 16, podemos apreciar la cantidad de países que tiene cada plataforma Streaming, donde podemos ver que Amazon Prime contiene más países seguido de Netflix.

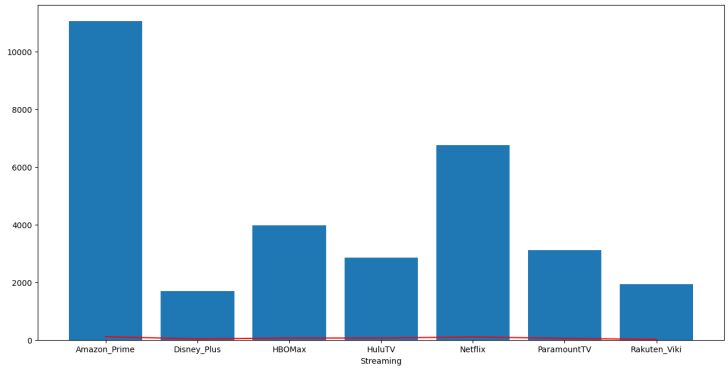



Figure 17: Streaming Country Titles

En la figure 17, podemos saber porqué Amazon Prime y Netflix contienen tanto países en comparación a las otras plataformas Streamings, y la respuesta es porque son los que más cantidad de retransmisiones tienen y tienen más posibilidad de tener retransmisiones de diferentes países.

## 2.9 ¿Influye la plataforma Streaming con la duración?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la duración media de todas las retransmisiones de cada plataforma Streaming.

```
def streamingRuntimeMovie():
    query = """
    SELECT s.streaming_name,
        AVG(t.title_runtime) AS total_movie_runtime
    FROM titles t
    INNER JOIN titles_streamings ts ON t.title_id = ts.title_id
    INNER JOIN streamings s ON ts.streaming_id = s.streaming_id
    WHERE t.title_type = 'MOVIE'
    GROUP BY s.streaming_name;
    """
    return query
```

Figure 18: Query Streaming Runtime

En esta query hacemos que muestre el **streamingName** juntamente con la media de **titleRuntime** de las **MOVIES**, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **streamingName**. Para las series sería la misma query pero cambiando **MOVIE** por **SHOW** y tendriamos otra gráfica con los **SHOWS**.

El resultado de la query lo hemos representado en dos gráficas lineales que representa la media de duración de las retransmisiones tanto en peliculas como en series.

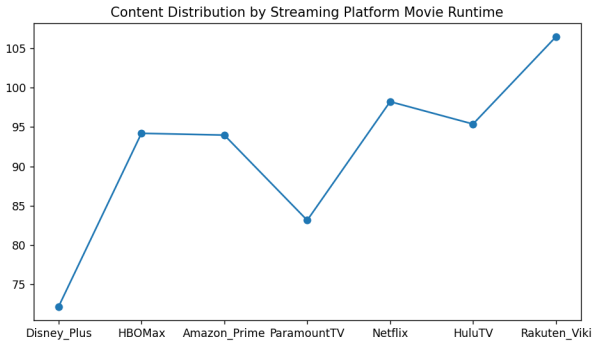



Figure 19: Streaming Runtime Movie

En la figure 19, podemos apreciar la duración media de las películas de cada plataforma Streaming, donde podemos ver que Rakuten Viki tiene una duración media bastante alta, superando los 100 minutos, en cambio, Disney Plus tiene una duración media de 1 hora.

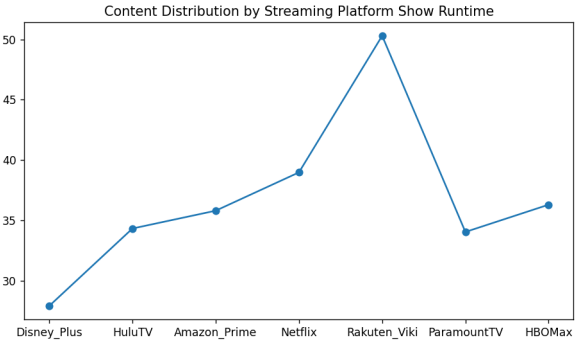



Figure 20: Streaming Runtime Show

En la figure 20, podemos apreciar la duración media de las series de cada plataforma Streaming, donde podemos ver que Rakuten Viki tiene una duración media bastante alta, superando los 50 minutos, en cambio, Disney Plus tiene una duración media de menos de 30 minutos.

Con la información de ambas gráficas, podemos concluir que Rakuten Viki suele tener películas y series con una duración superior a la media y Disney Plus una duración inferior a la media entre todas las plataformas Streamings.

## 2.10 ¿Ha mejorado la industria del cine al paso de los años?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la media de popularidad entre todas las retransmisiones emitidas de cada año desde 1900 hasta 2022.

```
def yearPopularity():
    query = """
    SELECT title_release_year, AVG(title_tmdb_popularity) AS avg_popularity
    FROM titles
    GROUP BY title_release_year
    ORDER BY title_release_year;
    """
    return query
```

Figure 21: Query Year Popularity

En esta query hacemos que muestre el **releaseYear** juntamente con la media del **TmdbPopularity**, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **releaseYear**.

El resultado de la query lo hemos representado en una gráfica de barras que representa la media de popularidad entre las retransmisiones de cada año.

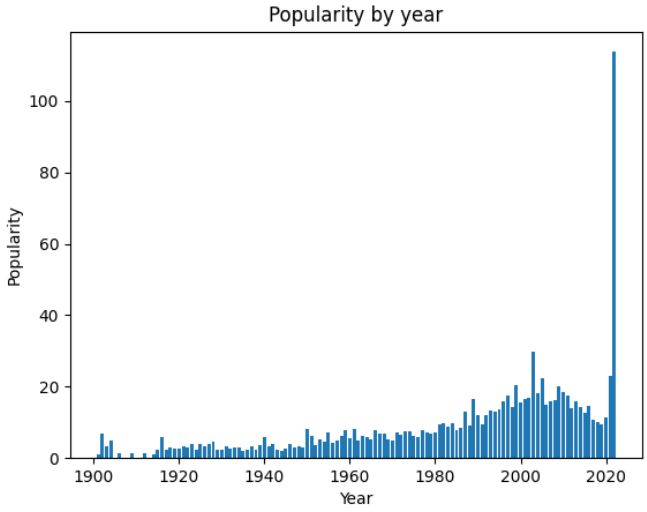



Figure 22: Year Popularity

En la figure 22, podemos apreciar la popularidad entre todas las retransmisiones que han habido en cada año y podemos ver que a medida que han pasado los años, la popularidad ha ido aumentando llegando a tener un pico a principios del siglo, después ha habido una pequeña bajada llegando al 2022 donde tiene la mejor media de popularidad en estos ultimos 100 años.

## 2.11 ¿Las plataformas Streaming están ambientadas a cierto año?

Para saber la respuesta a esta pregunta, hemos hecho una query que calcule y muestre la cantidad de retransmisiones de cada año desde 1900 hasta 2022.

```
def yearStreaming():
    query = """
    SELECT s.streaming_name, t.title_release_year
    FROM streamings s
    LEFT JOIN titles_streamings ts ON s.streaming_id = ts.streaming_id
    LEFT JOIN titles t ON ts.title_id = t.title_id
    ORDER BY s.streaming_name, t.title_release_year;
    """
    return query
```

Figure 23: Query Year Streaming

En esta query hacemos que muestre el **streamingName** juntamente con los **releaseYear**, después hacemos la unión de tablas con sus respectivas variables y luego lo agrupamos por **streamingName**.

El resultado de la query lo hemos representado en una gráfica de dispersión que representa la cantidad de retransmisiones que tienen cada plataforma Streaming basandose en el año de salida.

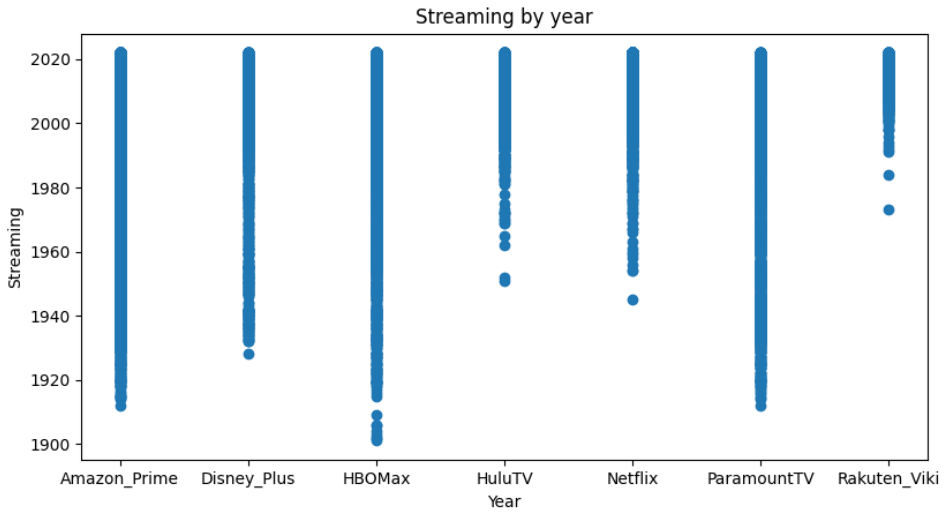



Figure 24: Year Streaming

En la figure 24, podemos apreciar la cantidad de años que están ambientadas cada plataforma Streaming. Podemos llegar a ver que HBO Max engloba más años en comparación a Rakuten Viki que es el que engloba menos, posiblemente porqué estará ambientado para cierto tipos de retransmisiones más modernas y HBO Max podría estar ambientado para todos los años.

## 2.12 ¿Tienen alguna correlación las puntuaciones de Imdb y Tmdb?

Para saber la respuesta a esta pregunta, hemos hecho una query que muestre las puntuaciones tanto de ImdbScore como TmdbScore de todas las retransmisiones.

```
def correlacionScores():
    query = """
    SELECT
        title_imdb_score,
        title_tmdb_score
    FROM
        titles;
    """
    return query
```

Figure 25: Query Correlation Imdb/Tmdb

En esta query hacemos que muestre las puntuaciones **imdbScore** juntamente con las puntuaciones**tmdbScore** de cada **title**.

El resultado de la query lo hemos representado en una gráfica de dispersión que representa las puntuaciones tanto de ImdbScore como TmdbScore de todas las retransmisiones.

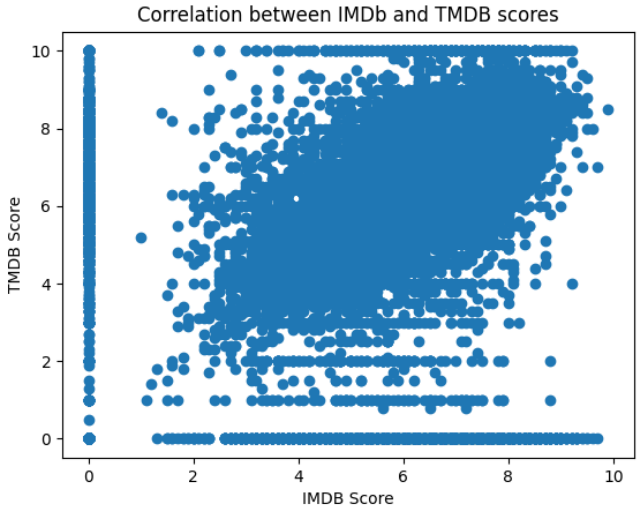



Figure 26: Correlation Imdb/Tmdb

En la figure 26, podemos apreciar varias cosas, una de ellas es que hay retransmisiones que no tienen ambas puntuaciones donde se puede ver en la línea vertical que se forma en ImdbScore: 0 que significa que solo hay puntuacion Tmdb y, por otro lado, la línea horizontal que se forma en Tmdb: 0 significa que solo hay puntuacion Imdb. El resto de puntos representa la correlación que hay en ambas puntuaciones teniendo una correlación de 0.315607 indica que la correlación es positiva moderada, lo que significa que si una puntuación es elevada, la otra puntuación tiende a estar elevada también.

# 3 Fase 2: Sistema de recomendación de contenido

El objetivo principal de este sistema de recomendación es proporcionar a los usuarios sugerencias personalizadas de películas y series que se alineen con sus intereses y preferencias, incluso en ausencia de datos explícitos de las preferencias del usuario.

## 3.1 Diseño del algoritmo

El diseño de esta aplicación de recomendación de películas y series se basa en el uso de la biblioteca **Tkinter** de Python para crear una interfaz gráfica de usuario (**GUI**) simple y fácil de usar.


Figure 27: Tkinter

Ahora vamos a explicar el diseño:
La función main crea una instancia de la clase **MovieRecommendationApp** y la ejecuta en un bucle infinito (**mainloop**) hasta que el usuario cierra la ventana principal.

```
def main():
    root = tk.Tk()
    app = MovieRecommendationApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Figure 28: Main

La clase **MovieRecommendationApp** representa la aplicación en sí y contiene toda la lógica necesaria para ejecutarla.
Al inicializar la clase, se crea una ventana principal (**root**) con un tamaño específico y un título.

```python
class MovieRecommendationApp:
    def __init__(self, master):
        self.master = master
        master.title("Movie Recommendation App")
        master.geometry("400x200")
        master.configure(bg="#333")
```

Figure 29: MovieRecommendationApp

Se definen diccionarios (**questionMap** y **answerOptions**) que mapean preguntas y opciones de respuesta respectivamente.

```python
self.question_map = {
    "What do you prefer movies or series?": {
        "MOVIE": "Do you want it to have any film age restrictions?",
        "SHOW": "Do you want it to have any TV age restrictions?"
    },
    "Do you want it to have any TV age restrictions?":"What genre do you like?",
    "Do you want it to have any film age restrictions?":"What genre do you like?",
    "What genre do you like?": "Do you prefer it to last a maximum of 2 hours?",
    "Do you prefer it to last a maximum of 2 hours?": "What streaming platform do you want it to be on?",
    "What streaming platform do you want it to be on?": "What country do you want it to be from?",
    "What country do you want it to be from?": "What decade do you want it to be?",
    "What decade do you want it to be?": "Who do you want to star in it? (Actor or Director)",
    "Who do you want to star in it? (Actor or Director)": "Last Question"
}
```

Figure 30: QuestionMap


Figure 31: QuestionMap

Se inicializan variables para mantener el estado actual de la aplicación, como la pregunta actual y las respuestas del usuario.

```
self.current_question = "What do you prefer movies or series?"
self.user_answers = {}  # Diccionario para almacenar las respuestas del usuario
```

Figure 32: Inicialización de variables

Se crean widgets de interfaz de usuario, como etiquetas, menús desplegables y botones, para mostrar las preguntas y las opciones de respuesta al usuario.

```
self.question_label = tk.Label(master, text=self.current_question, fg="white", bg="#333")
self.question_label.pack()

self.answer_var = tk.StringVar(master)
self.answer_var.set(self.answer_options[self.current_question][0])
self.answer_menu = tk.OptionMenu(master, self.answer_var, *self.answer_options[self.current_question])
self.answer_menu.config(bg="#666", fg="white")
self.answer_menu.pack()

self.button = tk.Button(master, text="Next", command=self.next_question, bg="#666", fg="white")
self.button.pack()
```

Figure 33: Creación de widgets

Los métodos **nextQuestion**, **handleUserInput**, **showRecommendations**, **agreeRecommendations** y **disagreeRecommendations** se utilizan para manejar la navegación entre preguntas, procesar las respuestas del usuario y mostrar las recomendaciones de películas.

```
def next_question(self):
    answer = self.answer_var.get()
    self.user_answers[self.current_question] = answer  # Almacenar la respuesta del usuario
```

Figure 34: Función nextQuestion

Se crean variables para almacenar las respuestas del usuario para poder trabajar con ellas más adelante.

```
if self.current_question == "What do you prefer movies or series?":
    next_question_text = self.question_map[self.current_question][answer]
    self.current_question = next_question_text
else:
    next_question_text = self.question_map[self.current_question]
    self.current_question = next_question_text
```

Figure 35: Condición titleType

Con esta condición hacemos que, dependiendo de la respuesta del usuario, se diriga a una pregunta especificada en **questionMap**.

```
if self.current_question == "Who do you want to star in it? (Actor or Director)":
    input_frame = tk.Frame(self.master, bg="#333")
    input_frame.pack(pady=10)
    input_label = tk.Label(input_frame, text="Who do you want to star in it? (Actor or Director):", fg="white", bg="#333")
    input_label.pack()
    input_var = tk.StringVar()
    input_entry = tk.Entry(input_frame, textvariable=input_var, bg="#666", fg="white")
    input_entry.pack()
    input_button = tk.Button(input_frame, text="Next", command=lambda: self.handle_user_input(input_var), bg="#666", fg="white")
    input_button.pack()
    self.question_label.pack_forget()
    self.answer_menu.pack_forget()
    self.button.pack_forget()
else:
    self.answer_var.set(self.answer_options[self.current_question][0])
    self.answer_menu['menu'].delete(0, 'end')
    for option in self.answer_options[self.current_question]:
        self.answer_menu['menu'].add_command(label=option, command=tk._setit(self.answer_var, option))
```

Figure 36: Condición Actor/Director

Con esta condición hacemos que, cuando llegue a la pregunta de si quiere que participe un **Actor** en especifico o que lo haya dirigido algún **Director**, se muestre una ventana con un frame llamando a la función **handleUserInput()** para que pueda escribir por teclado el nombre y apellido de la persona.
Si no se cumple la condición, es decir, cualquier pregunta creada en el **questionMap**, se creará una ventana con la pregunta y sus respuestas.

```
def get_recommendation(self, user_responses):
    cursor = connection.connection.cursor()

    default_types = ["MOVIE", "SHOW"]
    default_ages = ["G", "PG", "PG-13", "R", "NC-17", "TV-Y", "TV-Y7", "TV-G", "TV-PG", "TV-14", "TV-MA"]
    default_genres = ["comedy", "family", "animation", "action", "fantasy", "horror", "drama", "war", "western", "european",
                      "romance", "thriller", "crime", "history", "sport", "scifi", "documentation", "music", "reality"]
    default_streamings = ["Amazon_Prime", "Disney_Plus", "HBOMax", "Hulu", "Netflix", "ParamountTV", "Rakuten_Viki"]
    default_countries = ["US", "GB", "MX", "CA", "DE", "SU", "IN", "XX", "IT", "JP", "FR", "HK", "ES", "IL", "AU", "CH",
                         "IE", "GR", "CN", "PH", "NL", "YU", "CI", "PR", "LI", "KR", "XC", "HU", "TW", "AN", "MC", "CO",
                         "RO", "EG", "TR", "BE", "ZA", "PT", "CL", "SE", "BR", "DK", "NZ", "RU", "LU", "CZ", "FI", "AT",
                         "SK", "AR", "VE", "TH", "PL", "AE", "SI", "BA", "ID", "NO", "AF", "IR", "IS", "BG", "JM", "RS",
                         "SZ", "LT", "TC", "SG", "UY", "BO", "UA", "MY", "TN", "QA", "NG", "KZ", "GQ", "MT", "SO", "KE",
                         "UnitedStatesofAmerica", "MA", "VN", "BD", "FJ", "MN", "UG", "TT", "PK", "XK", "PE", "DO", "SV",
                         "GE", "PS", "HR", "LV", "AQ", "LB", "KH", "CR", "BM", "JO", "PA", "AL", "CY", "CU", "PY", "EE",
                         "ET", "PF", "EC", "IO", "AM", "SY", "CM", "LY", "SUHH", "KI", "BW", "XG", "DZ", "SN", "AO", "RW",
                         "GT", "ZW", "KW", "CS", "MK", "BY", "GH", "BF", "BS", "Lebanon", "SA", "CD", "GL", "IQ", "VA",
                         "TZ", "NP", "KG", "BT", "MW", "LK", "MU", "NA", "KN", "FO", "HN", "SB", "ZM", "NC", "MO"]
    default_years = ["1900","1910","1920","1930", "1940", "1950", "1960", "1970", "1980","1990","2000","2010", "2020"]

    type_choice = [default_types] if user_responses[0] == "undefined" else [user_responses[0]]
    age_choice = [default_ages] if user_responses[1] == "undefined" else [user_responses[1]]
    genre_choice = [default_genres] if user_responses[2] == "undefined" else [user_responses[2]]
    runtime_choice = [[120]] if user_responses[3] == "Yes" else [[600]]
    streaming_choice = [default_streamings] if user_responses[4] == "undefined" else [user_responses[4]]
    country_choice = [default_countries] if user_responses[5] == "undefined" else [user_responses[5]]
    year_choice = [default_years] if user_responses[6] == "undefined" else [user_responses[6]]
    person_name = user_responses[7]
```

Figure 37: Función de recomendación

Primero creamos conexión con la base de datos e inicializamos unas variables que tendrán un valor por defecto que son todas las respuestas de una pregunta por si el usuario no selecciona ninguna respuesta en especifico, seleccionando la respuesta "**undefined**".

```
    query, parameters = querys.recommendation2(type_choice, age_choice, genre_choice, runtime_choice, streaming_choice,
                                               country_choice, year_choice, person_name)

    cursor.execute(query, parameters)
    resultados = cursor.fetchall()
    cursor.close()

    # Solo las primeras 5 recomendaciones
    recommendations = [resultado[0] for resultado in resultados[:5]]

    if not recommendations:
        messagebox.showwarning("Warning", "No recommendations were found in the database.")
        self.master.destroy()

    return recommendations
```

Figure 38: Función de recomendación

Seguidamente, llamamos a la funcion **recommendation()** que es una query para buscar los titles que vamos a recomendar. Almacenamos la query en su respectiva variable y los parametros, que son las respuestas del usuario tratadas anteriormente, en otra variable.
Ejecutamos la query con los parametros y lo almacenamos en la variable resultados, donde almacenaremos 5 recomendaciones.

Si no obtenemos ninguna recomendación, saldrá un mensaje de aviso diciendo que no se ha encontrado ningún title con los parametros que ha introducido el usuario.

```
def handle_user_input(self, input_var):
    input_value = input_var.get()
    if input_value.strip():
        updated_answers = self.user_answers.copy()
        updated_answers["Who do you want to star in it? (Actor or Director)"] = input_value.strip()
        self.user_answers = updated_answers
        self.next_question()
    else:
        messagebox.showwarning("Warning", "Please enter a valid name.")
```

Figure 39: Función para introducir Actor/Director

Aquí comprovamos que el valor introducido por teclado sea correcto. Básicamente, con la función **strip()** verificamos si el usuario ha introducido un **String** en el frame, en caso contrario, saldría una ventana de aviso diciendo que introduzca un **String**.
Seguidamente, almacenamos el **String** introducido por el usuario en la variable donde almacenamos las respuestas.

```python
def show_recommendations(self, recommendations):
    self.recommendations_window = tk.Toplevel(self.master)
    self.recommendations_window.title("Movie Recommendations")
    self.recommendations_window.geometry("800x500")
    self.recommendations_window.configure(bg="#333")

    recommendation_label = tk.Label(self.recommendations_window, text="Recommended Titles:", fg="white", bg="#333")
    recommendation_label.pack()

    recommendations_text = "\n".join(recommendations)
    recommendations_display = tk.Text(self.recommendations_window, bg="#666", fg="white")
    recommendations_display.insert(tk.END, recommendations_text)
    recommendations_display.pack()

    agree_button = tk.Button(self.recommendations_window, text="I agree", command=self.agree_recommendations, bg="#666", fg="white")
    agree_button.pack(side=tk.LEFT, padx=10, pady=10)

    disagree_button = tk.Button(self.recommendations_window, text="I disagree", command=self.disagree_recommendations, bg="#666", fg="white")
    disagree_button.pack(side=tk.RIGHT, padx=10, pady=10)
```

Figure 40: Función para mostrar las recomendaciones

Cuando el usuario responde todas las preguntas, se muestra una ventana con las recomendaciones de películas y se le da al usuario la opción de aceptar o rechazar las recomendaciones.

```python
def agree_recommendations(self):
    messagebox.showinfo("Feedback", "Thank you for your feedback! We're glad you found the recommendations helpful.")
    self.recommendations_window.destroy()

def disagree_recommendations(self):
    #messagebox.showinfo("Feedback", "Thank you for your feedback! We'll work on improving our recommendations.")
    cursor = connection.connection.cursor()
    recommendation = querys.recommendationDefault()
    cursor.execute(recommendation)
    resultados = cursor.fetchall()
    cursor.close()
    recommendations = []
    for resultado in resultados:
        title = resultado[0]
        recommendations.append(title)

    self.recommendations_window = tk.Toplevel(self.master)
    self.recommendations_window.title("Movie Recommendations")
    self.recommendations_window.geometry("800x500")
    self.recommendations_window.configure(bg="#333")

    recommendation_label = tk.Label(self.recommendations_window, text="Recommended Titles:", fg="white", bg="#333")
    recommendation_label.pack()

    recommendations_text = "\n".join(recommendations)
    recommendations_display = tk.Text(self.recommendations_window, bg="#666", fg="white")
    recommendations_display.insert(tk.END, recommendations_text)
    recommendations_display.pack()
```

Figure 41: Función para dar acción a los botones agree y disagree

Si el usuario selecciona el botón **agree**, automáticamente se muestra una ventana emergente con un mensaje de agradecimiento.
En cambio, si elige **disagree**, significa que no está conforme con las recomendaciones, ya sea porque ya ha las ha visto todas o no le llama mucho la atención el titulo, entonces hemos decidido recomendar 5 **titles** mediante una query diferente a la usada anteriormente.

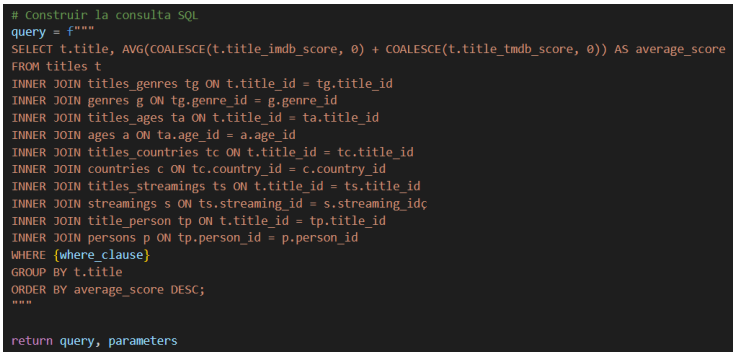
```python
def recommendation(type, age, genre, runtime, streaming, country, year, name):
    # Construir la cláusula WHERE dinámicamente
    where_conditions = []
    parameters = []

    # Generar los placeholders y recoger los parámetros para cada lista
    def placeholders(values):
        return ",".join(["%s"] * len(values))
    if type:
        where_conditions.append(f"t.title_type IN ({placeholders(type)})")
        parameters.extend(type)
    if age:
        where_conditions.append(f"a.age_name IN ({placeholders(age)})")
        parameters.extend(age)
    if genre:
        where_conditions.append(f"g.genre_name IN ({placeholders(genre)})")
        parameters.extend(genre)
    if runtime:
        where_conditions.append("t.title_runtime <= %s")
        parameters.append(runtime[0])  # Asumiendo que runtime es una lista con un solo elemento
    if streaming:
        where_conditions.append(f"s.streaming_name IN ({placeholders(streaming)})")
        parameters.extend(streaming)
    if country:
        where_conditions.append(f"c.country_name IN ({placeholders(country)})")
        parameters.extend(country)
    if year:
        where_conditions.append(f"t.title_release_year IN ({placeholders(year)})")
        parameters.extend(year)
    if name:
        where_conditions.append("p.person_name LIKE %s")
        parameters.append(name)

    where_clause = " AND ".join(where_conditions)
```

Figure 42: Función para crear la query de recomendaciones

En esta función, empezamos tratando los parametros que recibe, los cuales son una array de las respuestas del usuario. Los tratamos como array porque puede darse el caso de que haya elegido una respuesta "**undefined**" y en ese caso deberiamos hacer que lea todos los elementos de la array. Despues los almacenamos en la variable **whereClause**.



```python
# Construir la consulta SQL
query = f"""
SELECT t.title, AVG(COALESCE(t.title_imdb_score, 0) + COALESCE(t.title_tmdb_score, 0)) AS average_score
FROM titles t
INNER JOIN titles_genres tg ON t.title_id = tg.title_id
INNER JOIN genres g ON tg.genre_id = g.genre_id
INNER JOIN titles_ages ta ON t.title_id = ta.title_id
INNER JOIN ages a ON ta.age_id = a.age_id
INNER JOIN titles_countries tc ON t.title_id = tc.title_id
INNER JOIN countries c ON tc.country_id = c.country_id
INNER JOIN titles_streamings ts ON t.title_id = ts.title_id
INNER JOIN streamings s ON ts.streaming_id = s.streaming_idç
INNER JOIN title_person tp ON t.title_id = tp.title_id
INNER JOIN persons p ON tp.person_id = p.person_id
WHERE {where_clause}
GROUP BY t.title
ORDER BY average_score DESC;
"""

return query, parameters
```

Figure 43: Query para las recomendaciones

Con esta query mostramos los **titles** juntamente con la media de sus puntuaciones ya que será la clave para saber cuales mostrar primeramente. Accedemos a sus respectivas tablas de **MySQL** y con **WHERE** accedemos a los filtros que ha elegido el usuario para encontrar los **titles**. Despues los agrupamos y los ordenamos descendientemente por la puntuación media.

```
def recommendationDefault():
    query = """
    (
        SELECT title, title_type, title_release_year, title_imdb_score, title_tmdb_score
        FROM titles
        WHERE title_release_year = 2022 AND title_type = 'MOVIE'
        ORDER BY title_imdb_score + title_tmdb_score DESC
        LIMIT 5
    )
    UNION
    (
        SELECT title, title_type, title_release_year, title_imdb_score, title_tmdb_score
        FROM titles
        WHERE title_release_year = 2022 AND title_type = 'SHOW'
        ORDER BY title_imdb_score + title_tmdb_score DESC
        LIMIT 5
    );
    """
    return query
```

Figure 44: Query para las recomendaciones por defecto

Si el usuario no le han gustado las recomendaciones que les hemos mostrado, cuando pulse el botón **Disagree**, con esta query mostrará las 5 series y películas mejores puntuadas en este ultimo año, ya que serán las que estarán en el **TOP**.

## 3.2 Juego de pruebas

### 3.2.1 Prueba 1: Recomendación de una película

Vamos a verificar que el programa puede recomendar una película correctamente.

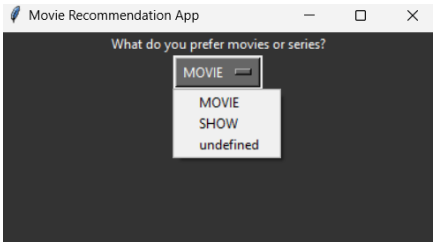



Figure 45: Opciones Type

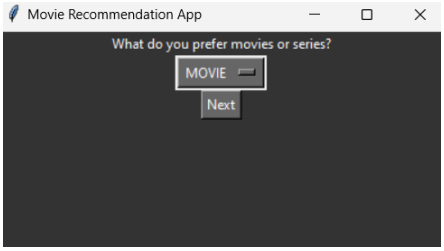



Figure 46: Respuesta Type

En esta parte elegimos el tipo del **title**, que en este caso es una **MOVIE**.

Acto seguido, nos muestra la siguiente pantalla de las restricciones de edad.

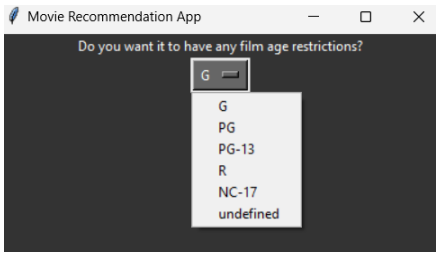



Figure 47: Opciones Age Restrictions

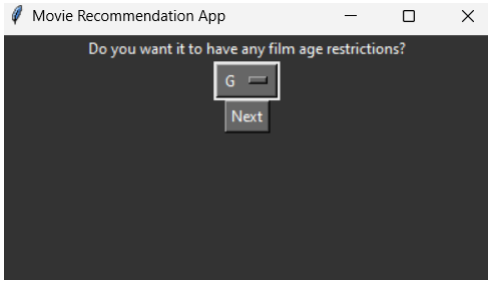



Figure 48: Respuesta Age Restrictions

Elegimos por ejemplo la '**G**' ya que es para un público general, es decir, está diseñado para ser adecuado para todas las edades y puede ser que sea el más común entre todas.

Después nos sale esta pantalla de los géneros.

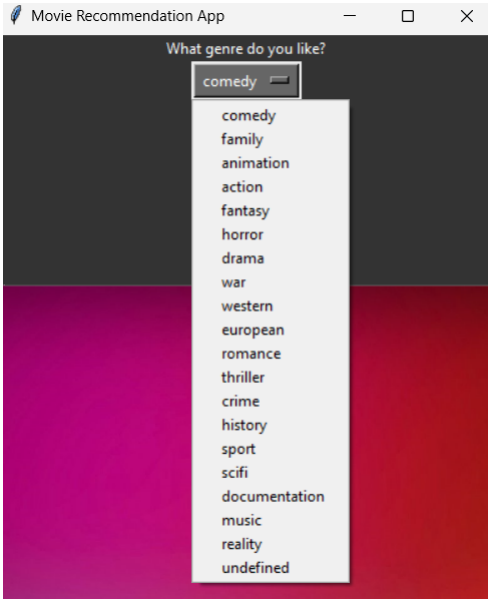



Figure 49: Opciones Genre

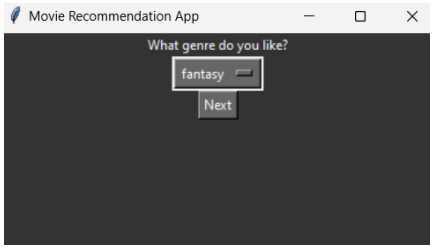



Figure 50: Respuesta Genre

Elegimos el género **Fantasy** ya que, en el estudio de la **Fase 1**, es el más común.

La siguiente pantalla es del tiempo de duración, si prefiere que sean de máximo 2 horas o que no haya límite.

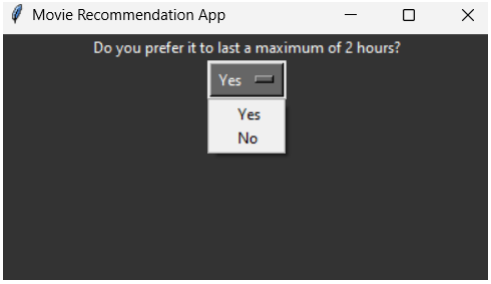



Figure 51: Opciones Runtime

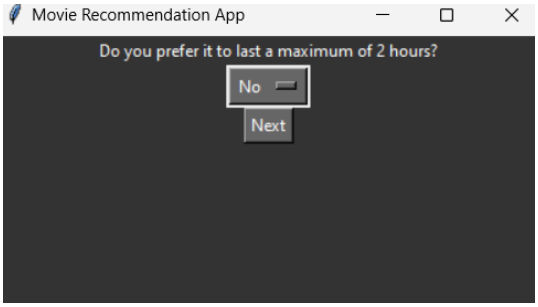



Figure 52: Respuesta Runtime

Elegimos que no tenga un límite de duración para que acceda a más películas para recomendar.

Después se nos muestra la pantalla de elegir en que **Streaming** la queremos.

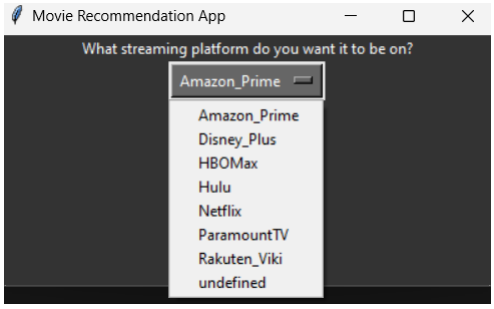



Figure 53: Opciones Streaming

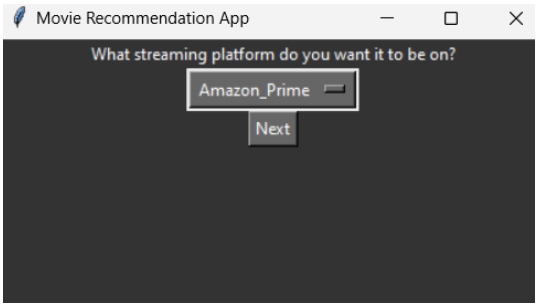



Figure 54: Respuesta Streaming

Elegimos **AmazonPrime** ya que es la plataforma **Streaming** con más retransmisiones en comparación con las otras plataformas.

La siguiente pantalla es la de elegir de que país te gustaria que fuera.

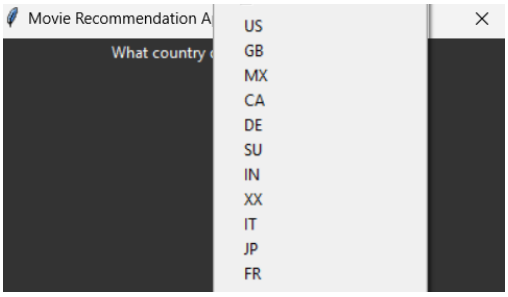



Figure 55: Opciones Country

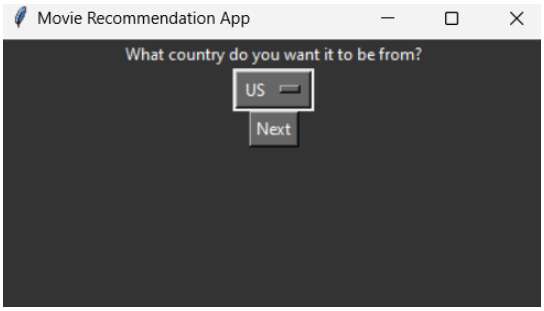



Figure 56: Respuesta Country

En este caso, como hay 155 países, no los podemos mostrar en una imagen, peró se podría seleccionar cualquiera, como aquí, que elegimos el país **US** ya que es el que más películas tiene.

Después vemos la pantalla del año de salida, que nosotros lo hemos hecho por décadas.

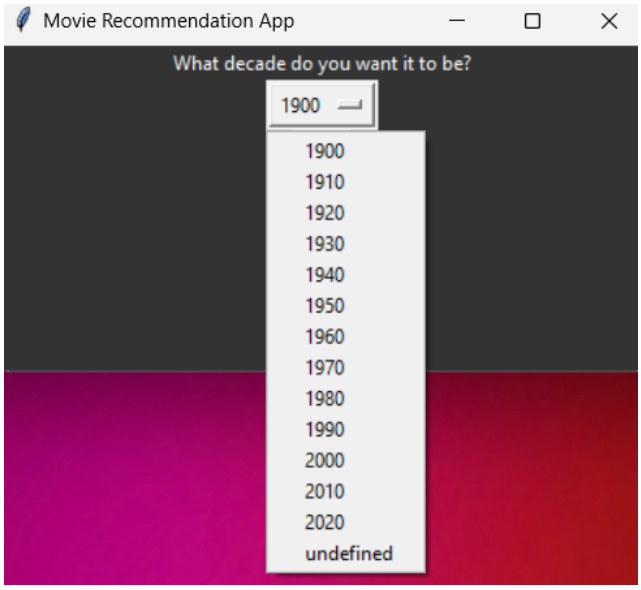



Figure 57: Opciones Year

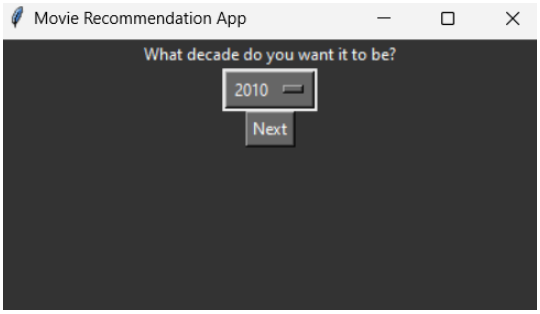



Figure 58: Respuesta Year

Elegimos el año 2010 y, el rango de años en que se va a basar, será entre 2010 y 2020 ambos incluidos.

Y por ultima pregunta, tenemos la de si quiere que aparezca algún **Actor** o que la haya dirigido algún **Director**.

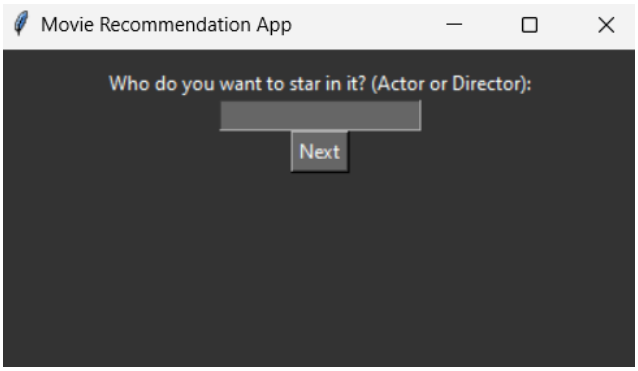



Figure 59: Frame Actor/Director

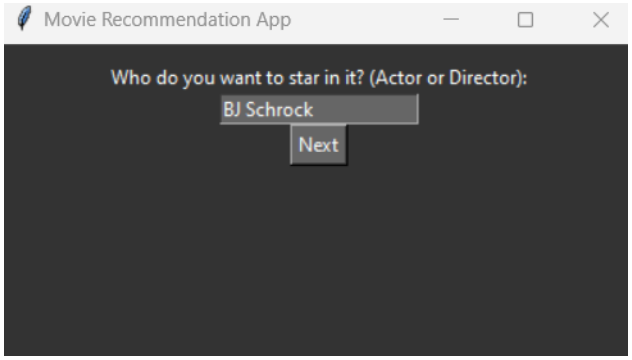



Figure 60: Respuesta Actor/Director

En esta pregunta, la hemos hecho tal que, tienes que introducir algún **String** si o si, sino sale una ventana emergente dando el aviso de que tiene que introducir un texto como este.

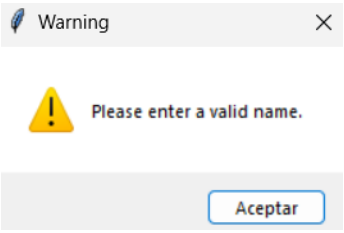



Figure 61: Ventana emergente

Después de haber elegido todas las respuestas, el programa accede a la base de datos y busca todos los titles que coincidan con las respuestas del usuario y se mostrará esta ventana.

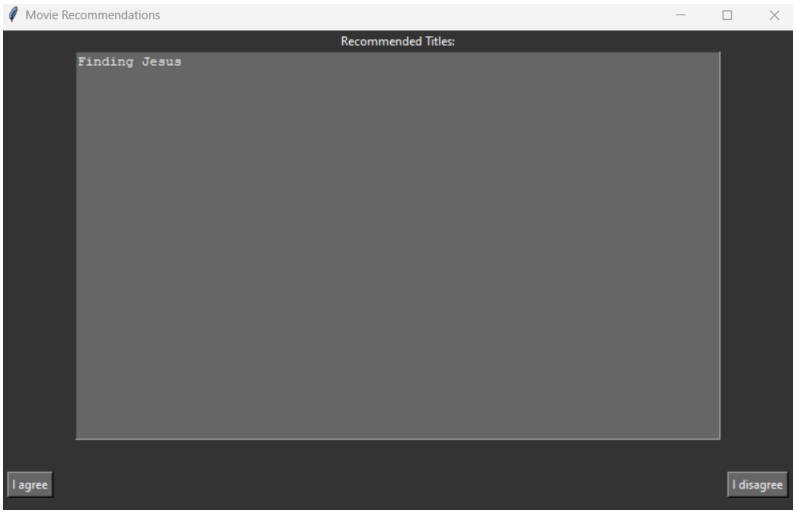



Figure 62: Titles Recomendados

En la ventana de **titles** recomendados, podemos ver 2 botones que ponen "**I agree**" y "**I disagree**". Si presionamos el botón "**I agree**", digamos que el usuario está conforme con los titles recomendados y se muestra esta ventana.

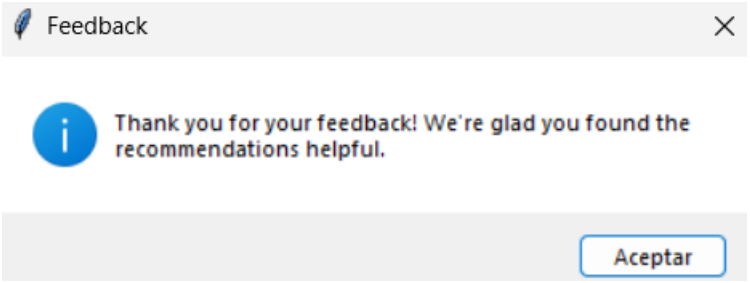



Figure 63: Ventana Agree

Si elegimos el botón "**I disagree**", podemos decir que el usuario no está conforme con los **titles** recomendados y automáticamente se muestra esta ventana con unos titles recomendados por defecto.

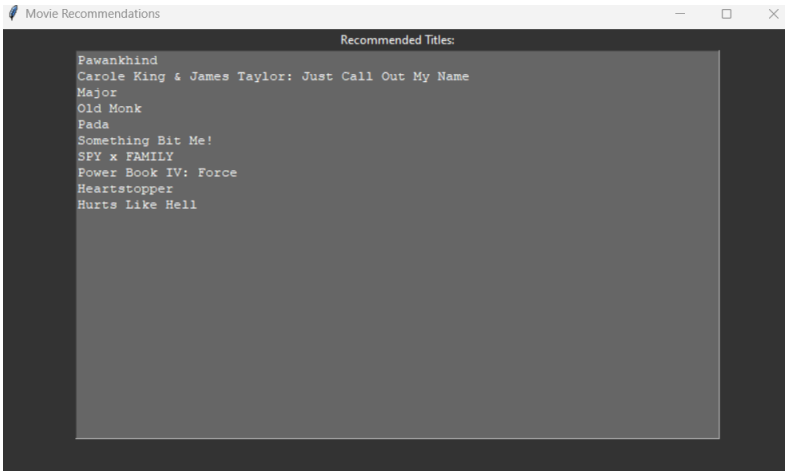



Figure 64: Titles recomendados por defecto

Donde podemos ver que hay 5 películas y 5 series que son las que hemos mencionado anteriormente que íbamos a recomendar si el usuario no está conforme con los **titles** recomendados al principio.

### 3.2.2 Prueba 2: Sin recomendaciones encontradas

En esta prueba vamos a poner respuestas para que no encuentre ningún **title** que pueda recomendar, es decir, que no esté en la base de datos.

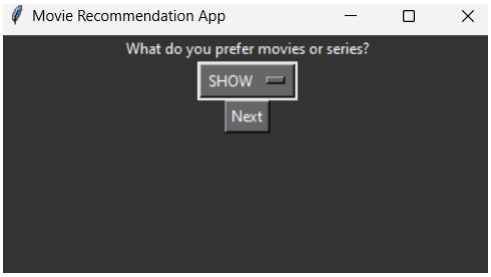



Figure 65: Respuesta type: SHOW

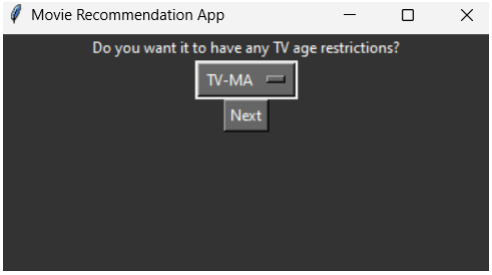



Figure 66: Respuesta age restriction: TV-MA

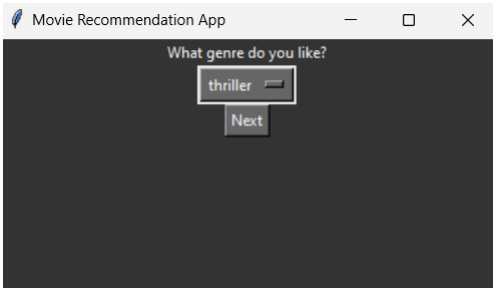



Figure 67: Respuesta genre: thriller

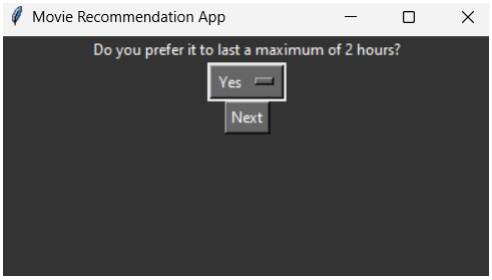



Figure 68: Respuesta runtime: Yes

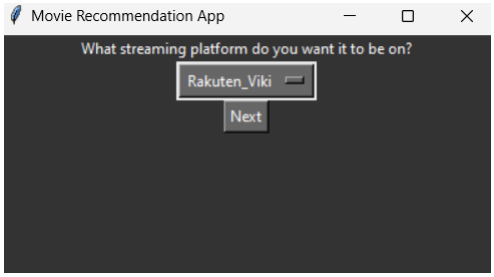



Figure 69: Respuesta Streaming: Rakuten Viki

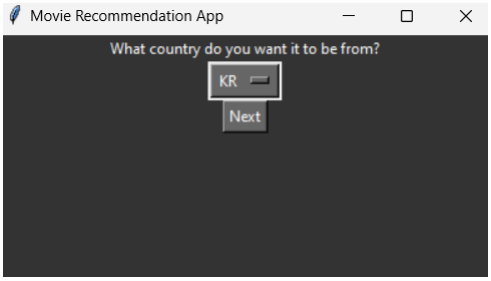



Figure 70: Respuesta country: KR

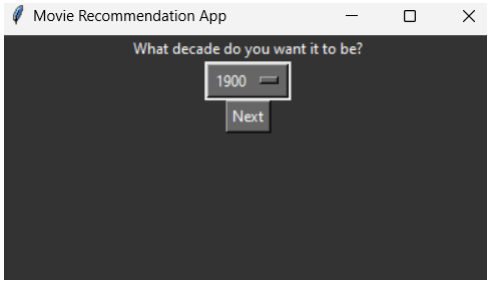



Figure 71: Respuesta Decade: 1900

Al introducir estas respuestas en la query para la búsqueda de **titles** recomendados, el programa, al no encontrar ningún **title** que cumpla con las respuestas, muestra esta ventana emergente.

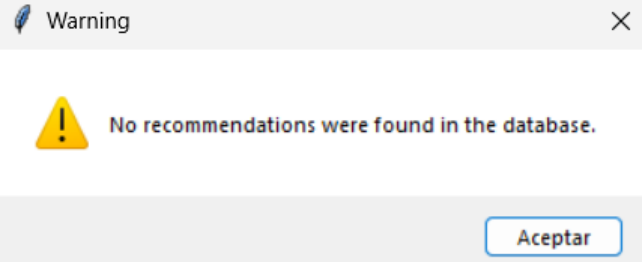



Figure 72: Ventana emergente: No recommendation

### 3.2.3 Prueba 3: Usuario sin gustos preferenciales

Ahora vamos a probas cuando al usuario no tiene ningún tipo en especifico y le da igual que **title** salga, es decir, que todas las respuestas son "**undefined**".

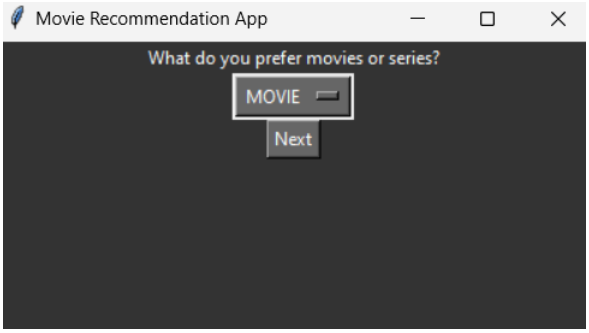



Figure 73: Respuesta type: MOVIE

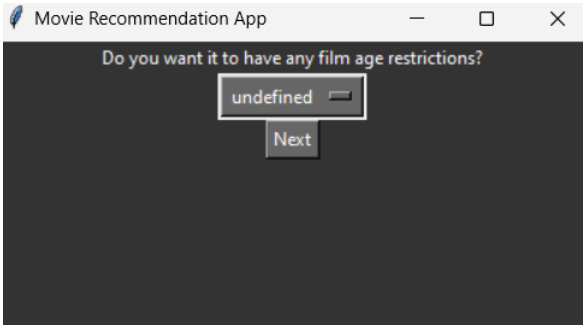



Figure 74: Respuesta age restriction: undefined

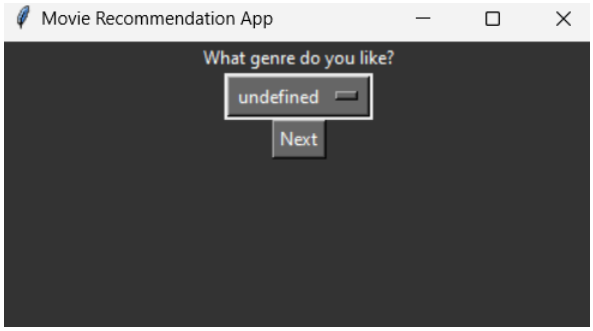



Figure 75: Respuesta genre: undefined

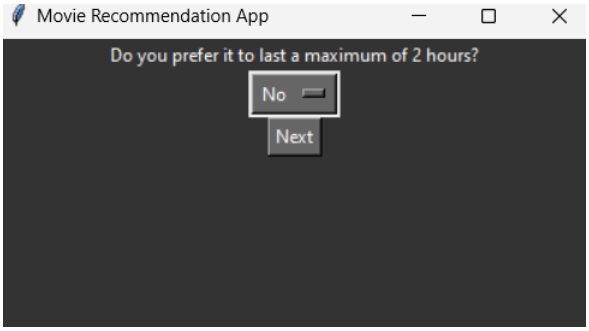



Figure 76: Respuesta runtime: No

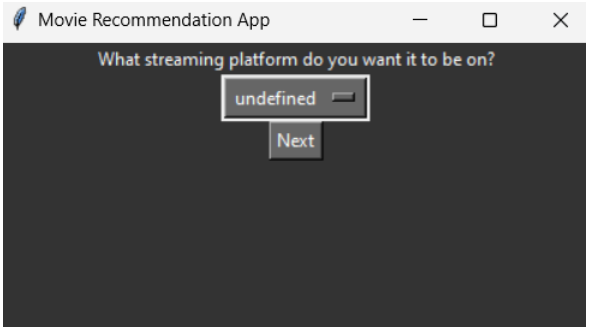



Figure 77: Respuesta streaming: undefined

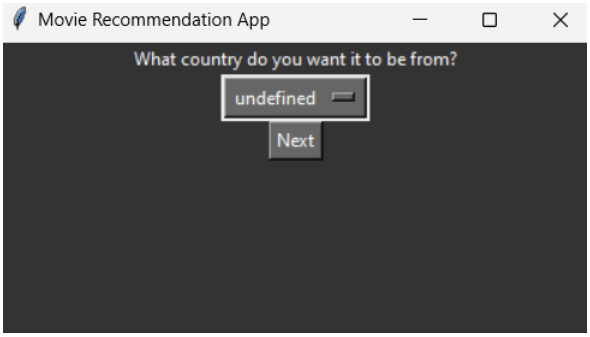



Figure 78: Respuesta country: undefined

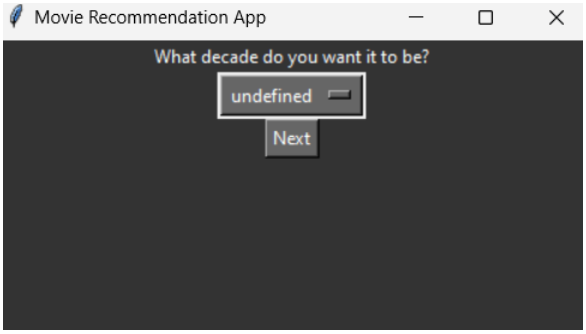



Figure 79: Respuesta year: undefined

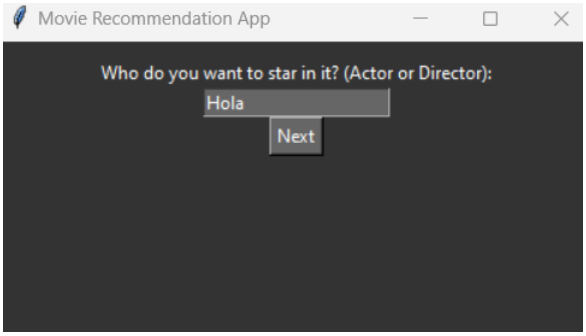



Figure 80: Respuesta actor/director: random

Despues de que el usuario sin preferencia de gustos, haya puesto sus respuestas, el programa accederá a la base de datos y mirará las 5 mejores películas valoradas, ya que el usuario solo quiere una película sin restricción alguna.

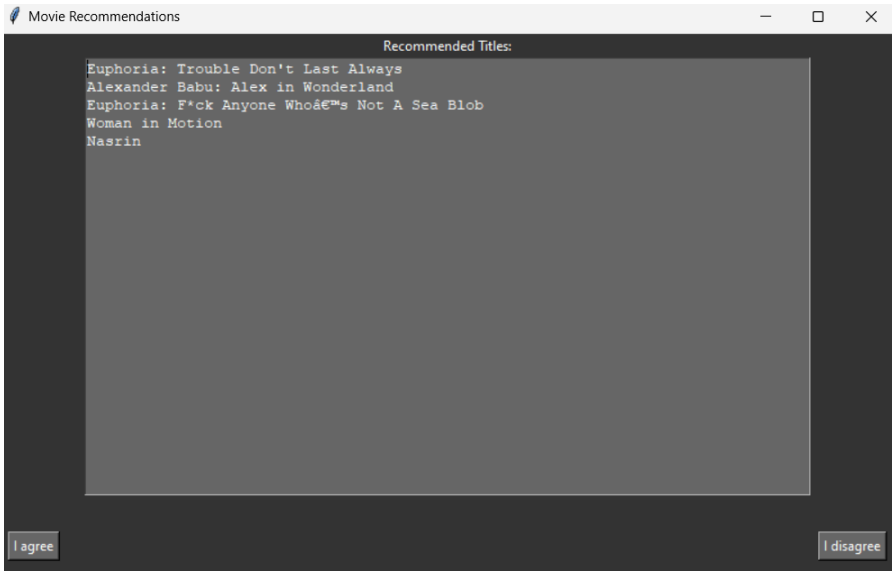



Figure 81: Titles recomendados