PROCESAMIENTO DE

LENGUAJE NATURAL

(NLP)

Trabajo Práctico N°1

Clasificador de Recomendaciones Recreativas

Año 2024 - Segundo Semestre

**Profesores:**

Geary, Alan

Manson, Juan Pablo

**Integrantes:**

García, Julián

Herrera, Francisca

**Fecha de entrega:**

Miércoles 06/11/24

# ÍNDICE

[ÍNDICE 2](#_2hsvbpbhjdep)

[INTRODUCCIÓN 3](#_ydig4xn928qq)

[DESARROLLO 4](#_hunvbi5trga2)

[Datos utilizados 4](#_hy0q6dy5wdru)

[Consideraciones 5](#_elfis1iq2gpd)

[Procesamiento de datos 5](#_qvzcnbd0vuls)

[Funciones de carga y procesamiento 5](#_v9tla72bfft8)

[Aplicación 6](#_sfnztzsyeeh6)

[Filtrado parental y por cantidad de jugadores 6](#_uq8ryfsg47vc)

[Filtrado por estado de ánimo 7](#_p97xgpz6ah6j)

[Ejemplo de uso 8](#_5ujjxt63dt1w)

[Filtrado por NER 9](#_wpks09z2zqpn)

[Ejemplo de uso 9](#_r14pygbk8ang)

[Búsqueda semántica por interés 10](#_fndv43t3355q)

[Ejemplo de uso 11](#_awkwup3eqmh)

[Puesta en marcha 11](#_rivtlwhledru)

[RESULTADOS 13](#_myxqi1y8oyow)

[CONCLUSIONES 13](#_ptx6uyz97afo)

[ANEXOS 14](#_556vr68unmk)

# 

# 

# 

# 

# 

# INTRODUCCIÓN

**Contexto**

*Saldremos de vacaciones a la playa por 15 días. Se pronostica lluvia durante al menos cuatro de esos días, lo que limitará las actividades al aire libre. Para esos días de mal clima, proponemos una solución que facilite la recreación según nuestro estado de ánimo del momento.*

**Objetivo**

1. Crear un programa de NLP que, basado en el estado de ánimo del usuario, sugiera entre ver una película, jugar un juego de mesa o leer un libro (o múltiples opciones si es adecuado).
   1. Desarrollar un clasificador que categorice el estado de ánimo del usuario (a partir del input #1).
   2. A partir de una frase de preferencia proporcionada por el usuario, generar un conjunto de recomendaciones personalizadas. Ejemplo: "una historia de amor en la selva".
   3. Búsqueda de Opciones: Vincular la frase ingresada por el usuario con las bases de datos de libros, películas y juegos.

**Requerimientos**

1. Utilizar clasificadores para determinar el estado de ánimo (ejemplo: métodos de clasificación supervisada).
2. Aplicar técnicas de embeddings y comparación de similitud semántica para encontrar las mejores coincidencias en los datasets.
3. Utilizar reconocimiento de entidades nombradas (NER).
4. Realizar pruebas con diferentes ejemplos para mostrar la efectividad del programa.
5. Entregar el dataset de libros generado con web scraping.
6. Opcional: Presentación de una aplicación para el programa desarrollado, utilizando Google Colab con una interfaz sencilla basada en widgets (ip widgets).

# 

# DESARROLLO

## Datos utilizados

Los datos empleados en este trabajo se obtuvieron de dos datasets y una página web.

■ Base de datos de juegos de mesa: bgg\_database.csv

■ Base de datos de películas: IMDB-Movie-Data.csv

■ Libros del Proyecto Gutenberg: <https://www.gutenberg.org/browse/scores/top1000.php#books-last1>

## Consideraciones

Separamos el código en distintos capítulos para un mayor orden y claridad del desarrollo. Dentro de cada uno se explicarán las funciones creadas y la ejecución de las mismas con ejemplos.

Al final se unirán las funciones en la función app para la ejecución de las recomendaciones.

## Procesamiento de datos

En esta sección se implementa un sistema de procesamiento y consolidación de datos de diferentes fuentes (juegos, películas y libros) con el objetivo de construir un dataframe unificado que permita un análisis conjunto. A continuación, se describe cada parte de su funcionamiento.

### Funciones de carga y procesamiento

Se clona el repositorio NLP-TUIA-Garcia-Herrera y se cambia el directorio de trabajo al repositorio clonado.

**get\_games:** Procesa un archivo CSV de juegos de mesa, ajusta el tipo de datos de las columnas y selecciona sólo las relevantes (nombre del juego, descripción, número mínimo y máximo de jugadores, edad mínima y categorías). Se añaden columnas de información fija (por ejemplo, si el juego es adecuado para niños y la clase games). Devuelve el dataframe df\_games procesado.

**get\_films:** Procesa un archivo CSV de películas, selecciona columnas relevantes (título, género, descripción, director y actores), combina los campos de director y actores en una columna People, y añade una columna Class con valor fijo films. Además, establece un rango fijo de jugadores (de 1 a 99) y una columna allow\_kids basada en el género (si contiene ‘Family’ es V). El criterio de establecer un rango de jugadores aplica a la lógica establecida en df\_games, para poder luego unir ambos dataframes.

Devuelve el dataframe df\_films procesado.

**get\_books:** Lee un archivo CSV de libros previamente generado mediante web scraping (ver función scrap\_books) y lo devuelve como el dataframe df\_books. Se asume que la cantidad de participantes será 1 (uno).

**scrap\_books:** Extrae información de libros de un sitio web, utilizando una combinación de requests y BeautifulSoup para parsear el contenido HTML. La función extrae los libros más populares y guarda la información en un archivo CSV. Los datos incluyen título, URL, autor y temas. Agrega la columna fija de clase books, rango de jugadores y apto niños.

**get\_all:** Combina los tres dataframes generados (juegos, películas y libros) en un solo dataframe df\_all. Se realiza un proceso de limpieza en columnas específicas (título, personas, descripción y género) para eliminar caracteres no deseados.

Se crea además la columna Data, que es una concatenación de los valores de otras columnas para obtener una representación unificada que luego servirá para la creación de embeddings.

**clean\_genre:** Esta función elimina caracteres especiales, palabras innecesarias (como "of" y "and"), números romanos, y convierte los textos a minúsculas para un procesamiento homogéneo.

Se obtiene como resultado el dataframe df\_all consolidado con los datos de películas, libros y juegos, ya estructurado y listo para las aplicaciones de NLP y filtrado que se detallan a continuación.

#### Aplicación



## Filtrado parental y por cantidad de jugadores

En esta sección se crean dos funciones de filtrado que serán utilizadas más adelante.   
A continuación se describe cada una de ellas:

**parental\_filter**: Esta función recibe como argumentos el dataframe df\_all y una variable input\_kids que indica si se debe filtrar contenido apto para niños.

Si input\_kids es "S", se filtran las filas en las que la columna allow\_kids es igual a 1, es decir, sólo se seleccionan las filas donde el contenido es apto.

Si input\_kids es distinto de "S", la función devuelve el dataframe sin aplicar ningún filtro específico.

**players\_filter**: Esta función recibe como argumentos el dataframe df\_all y un número input\_players que representa la cantidad de participantes.

Filtra las filas donde el número de jugadores está en el rango permitido, comparando input\_players con las columnas minplayers y maxplayers: selecciona las filas en las que input\_players es mayor o igual a minplayers y menor o igual a maxplayers.

Esto permite obtener actividades o juegos que sean apropiados para el número de jugadores especificado.

## Filtrado por estado de ánimo

Se utiliza un modelo de clasificación basado en *embeddings* de frases para identificar los estados de ánimo asociados con frases. El modelo entrenado tiene la capacidad de predecir las emociones expresadas en cada input, permitiendo un filtrado según las emociones del usuario.

**train\_sentiments:** Para entrenar el modelo de clasificación de sentimientos, primero cargamos un archivo de frases etiquetadas, donde cada frase tiene asignada una etiqueta que describe la emoción asociada. El proceso consiste en los siguientes pasos.

* **Carga y preproceso** del archivo de texto, dividiendo los datos en un conjunto de entrenamiento y otro de prueba.
* **Uso de un modelo preentrenado de embeddings** (DistilBERT) para generar representaciones vectoriales de las frases.
* **Entrenamiento de un modelo de regresión logística** para clasificar las frases en diferentes categorías emocionales.

**class\_sentiment:** Esta función asigna una etiqueta a la frase que recibe como argumento, según el estado de ánimo detectado por el modelo.

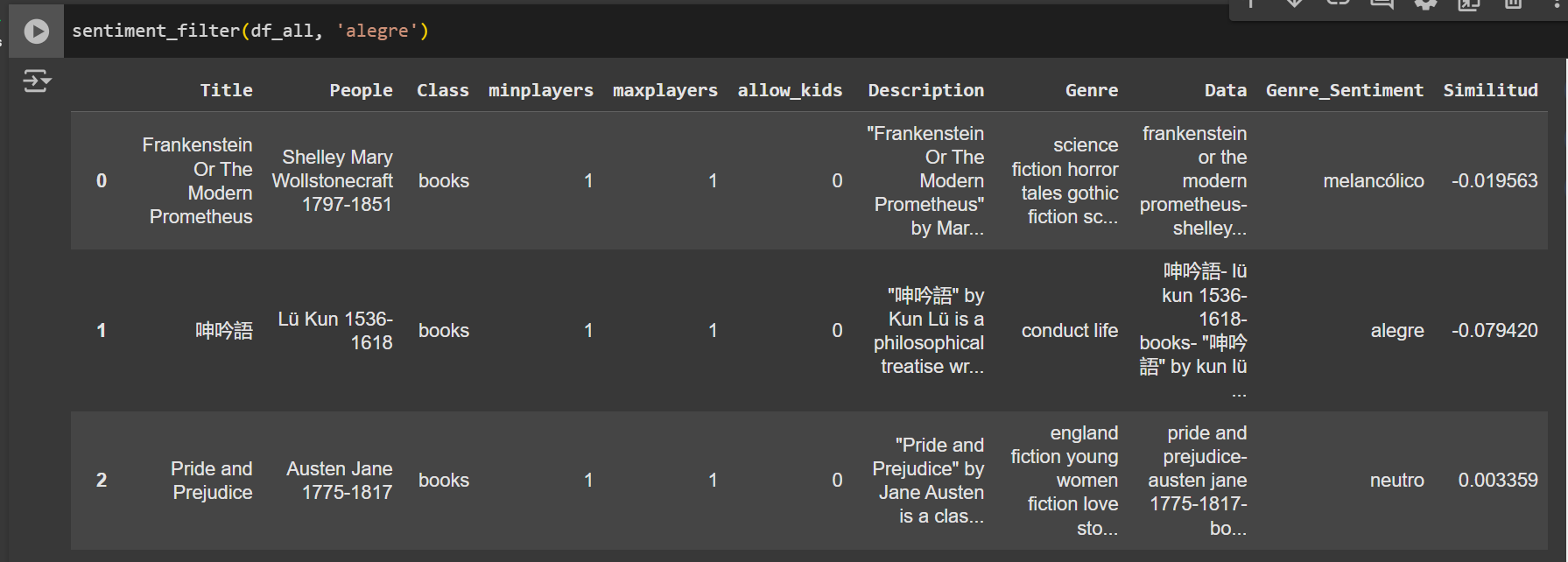
* **Generación del embedding** de la frase proporcionada.
* **Predicción del sentimiento** (como alegre, melancólico, neutro).
* **Asignación de la etiqueta "neutro"** cuando la probabilidad de acierto de la predicción es baja (por debajo del umbral 0.65).

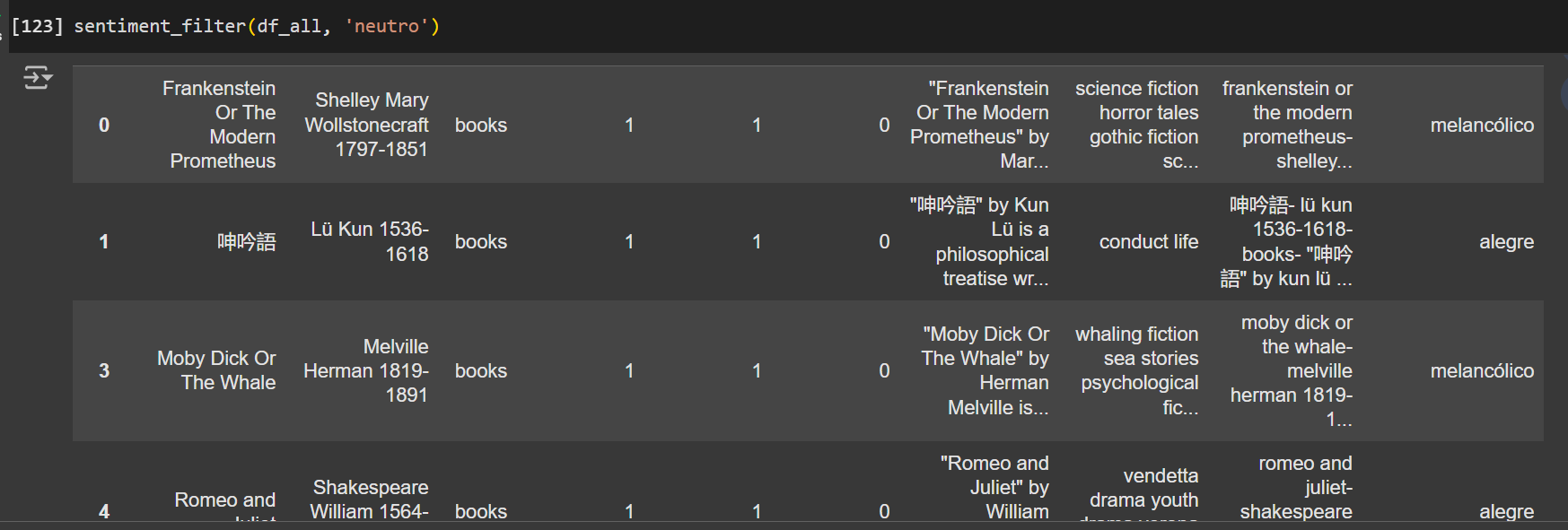
**class\_genre:** Esta función recorre cada registro de la columna "Genre" y asigna a cada uno una etiqueta de emoción correspondiente. Luego, devuelve el dataframe df\_all con una nueva columna llamada 'Genre\_Sentiment', que incluye las etiquetas asignadas.

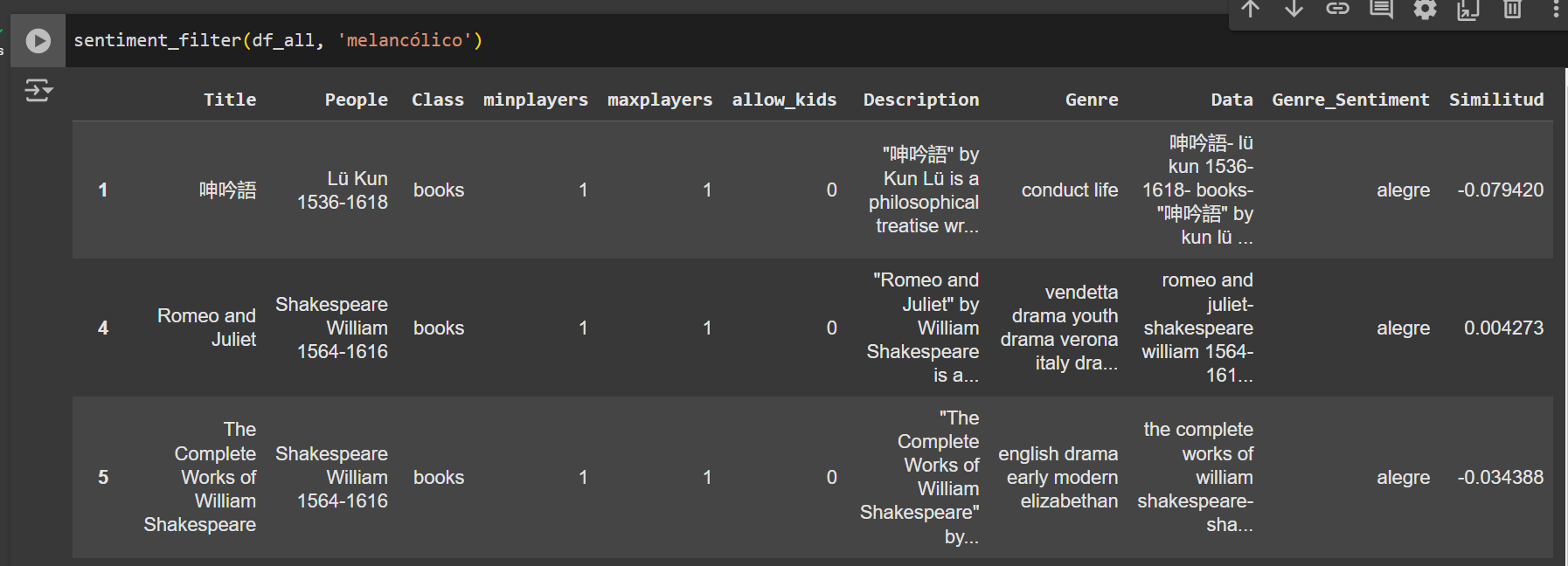
#### Ejemplo de uso

#### 









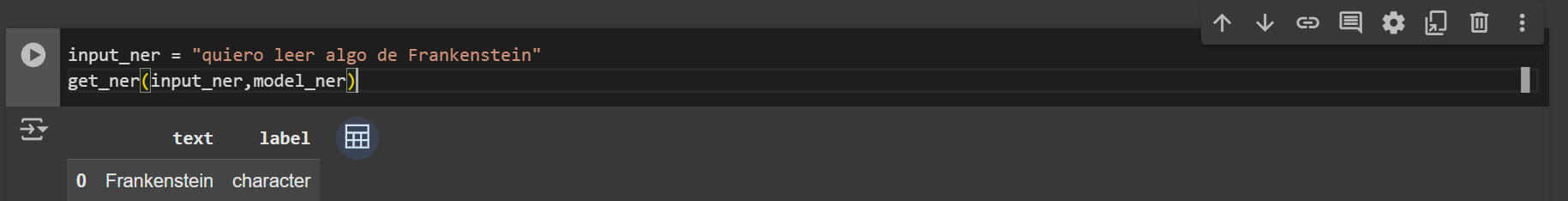
## Filtrado por NER

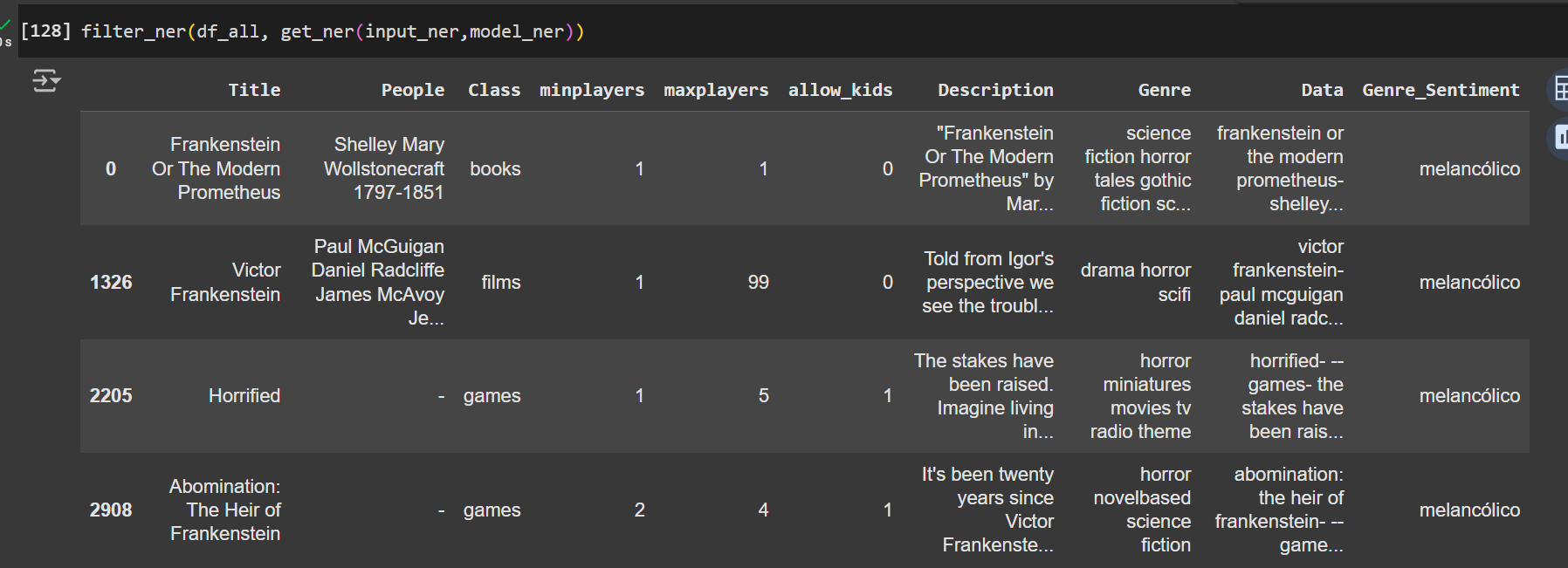
En esta sección, se utiliza un modelo preentrenado de reconocimiento de entidades nombradas (NER) para identificar entidades específicas en un conjunto de datos. El modelo de NER utilizado es GLiNER (gliner\_multi-v2.1) de Hugging Face, que permite detectar diversas entidades como personas, ubicaciones, actores y personajes.

**get\_ner:** Esta función recibe un texto y utiliza el modelo GLiNER para detectar las entidades especificadas en el texto, como personas, ubicaciones, actores y personajes, aplicando un umbral mínimo de confianza para la detección. Devuelve un dataframe con las entidades detectadas y sus respectivas etiquetas. Si no se detecta ninguna entidad, devuelve un dataframe vacío con las columnas ‘text’ y ‘label’.

**filter\_ner:** Esta función se encarga de filtrar el dataframe original para que sólo contenga filas donde la columna ‘data’ contenga alguna de las entidades detectadas.

#### Ejemplo de uso

****

****

## Búsqueda semántica por interés

En esta sección se desarrollan las funciones para realizar la búsqueda semántica basada en los embeddings de la columna ‘data’ de df\_all, con el objeto de recomendar actividades en base a un texto de entrada. A continuación se detallan las funciones creadas:

**precomputar\_embeddings**: Genera los embeddings de la columna Data de df\_all.

La función toma entonces como entrada el df, el nombre de la columna a vectorizar, y adicionalmente se puede ingresar el tamaño del lote par el entrenamiento, fijado en 64 por defecto, y el argumento device para ejecutar el modelo en CPU o GPU en caso de estar disponible. Los embeddings generados se concatenan en un solo tensor.

**guardar\_embeddings**: Se guardan los embeddings generados en la función anterior en un archivo CSV para su uso posterior, evitando la necesidad de recomputarlos en cada ejecución.

**semantic\_search**: esta función realiza la búsqueda semántica para encontrar filas en df\_all que sean similares a un texto de entrada input\_text.

En primer lugar la función carga los embeddings guardados en CSV con la función anterior y los convierte en tensores.

Codifica el texto de entrada input\_text en un embedding de entrada usando el mismo modelo.

Calcula la similitud coseno entre el embedding de entrada y los embeddings precomputados.

Añade la similitud calculada a df\_all en una nueva columna Similitud.

Ordena el dataframe por similitud, de mayor a menor, de modo que los elementos más similares al input\_text aparezcan primero en el dataframe ordenado (df\_ordenado) y lo retorna.

#### Ejemplo de uso

****

## Puesta en marcha

**1- app:** La función app se encarga de procesar una serie de inputs del usuario y realizar filtrados y clasificaciones sobre df\_all. Su objetivo es retornar un conjunto de datos filtrados y ordenados de acuerdo con diferentes criterios, tales como el estado de ánimo, entidades nombradas detectadas, cantidad de jugadores y control parental.

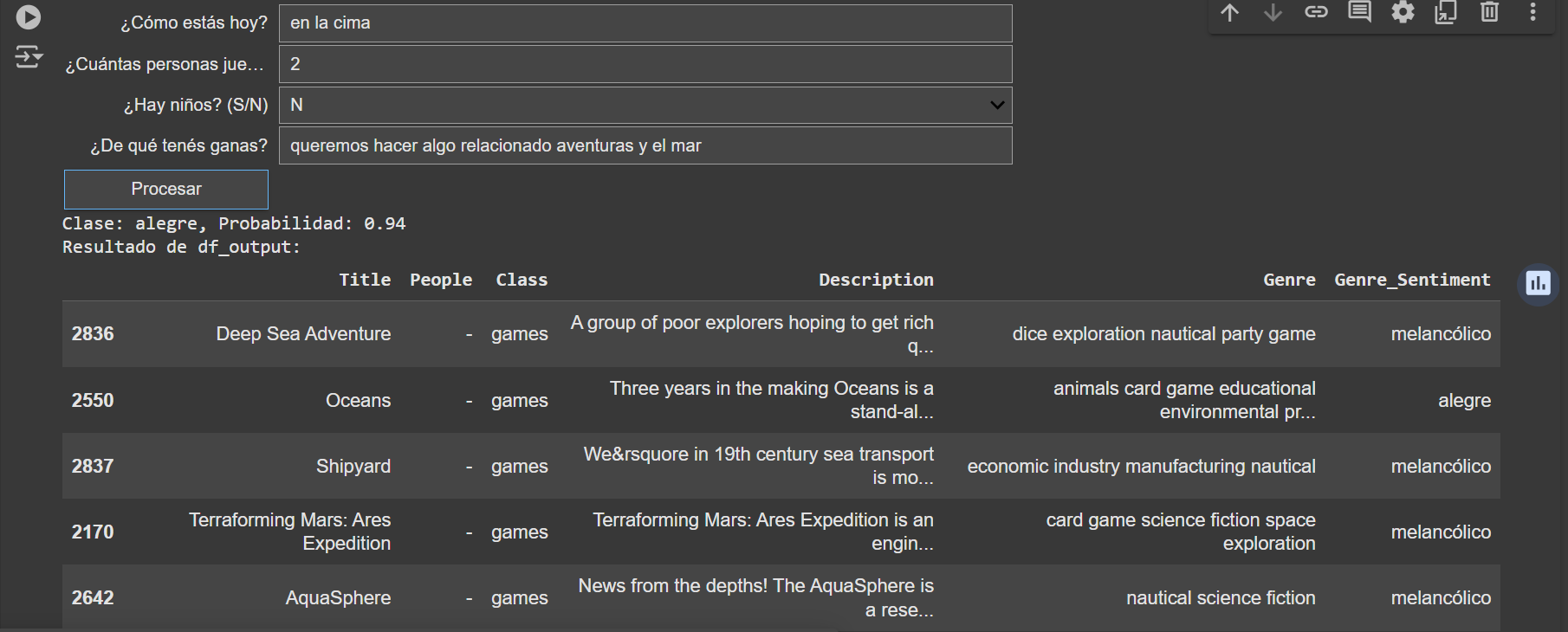
* **Entrada:**
  + df\_all: El dataframe que contiene todos los datos a procesar.
  + input\_1: Entrada de texto que representa el estado de ánimo del usuario.
  + input\_2: Número de jugadores.
  + input\_3: Indicador de si hay niños o no.
  + input\_4: Texto relacionado con los interés del usuario.
* **Proceso:**
  + **class\_sentiment:** Esta función se utiliza para clasificar el estado de ánimo del usuario.
  + **semantic\_search:** Se realiza una búsqueda semántica sobre el texto input\_4 para encontrar actividades similares en el conjunto de datos.
  + **get\_ner:** Se emplea la función para identificar entidades relevantes en el input\_4, y luego se aplica **filter\_ner** para filtrar las actividades que contengan estas entidades.
  + **players\_filter:** Se aplica el filtro para seleccionar las actividades que se ajusten al número de jugadores especificado (input\_2).
  + **parental\_filter:** Se filtran las actividades según la presencia de niños o no, utilizando el input\_3.
  + **sentiment\_filter:** Si no se detectaron entidades relevantes en el paso anterior, se realiza un filtrado adicional de acuerdo al estado de ánimo.
  + **Orden por similitud**: Se ordena el dataframe resultante por la columna de similitud, de mayor a menor.
* **Salida:** La función retorna los siguientes dataframes.
  + df\_4: Resultado de la búsqueda semántica.
  + df\_2: Resultado del filtrado por cantidad de jugadores.
  + df\_3: Resultado del filtrado por control parental.
  + df\_1: Resultado del filtrado final (por estado de ánimo u otras condiciones).
  + df\_output: Un dataframe final con las 5 mejores actividades filtradas y ordenadas por similitud.

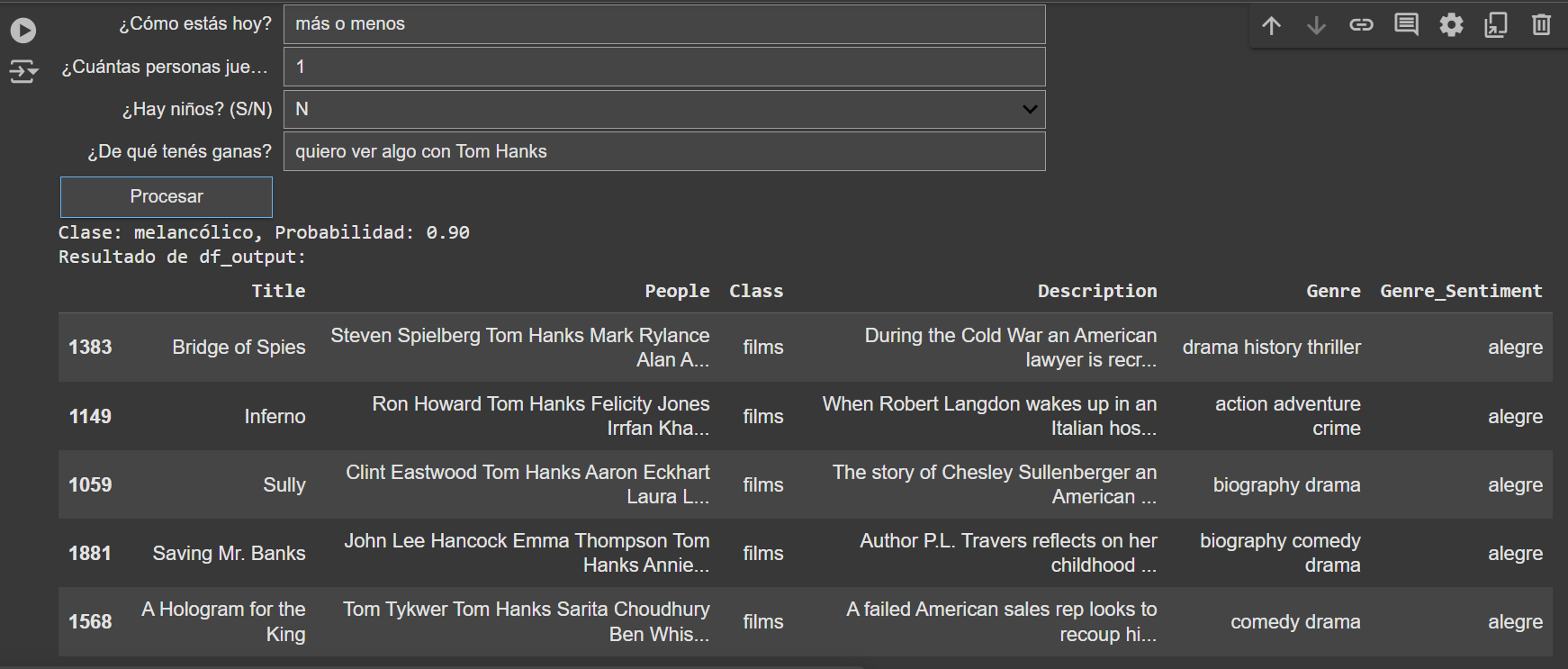
**2- recomendador:** Esta función crea una interfaz de usuario interactiva utilizando widgets, permitiendo que el usuario ingrese los inputs comentados en la función app.  
A partir de esta información se llama a la función **app** para procesar y mostrar las actividades recomendadas.

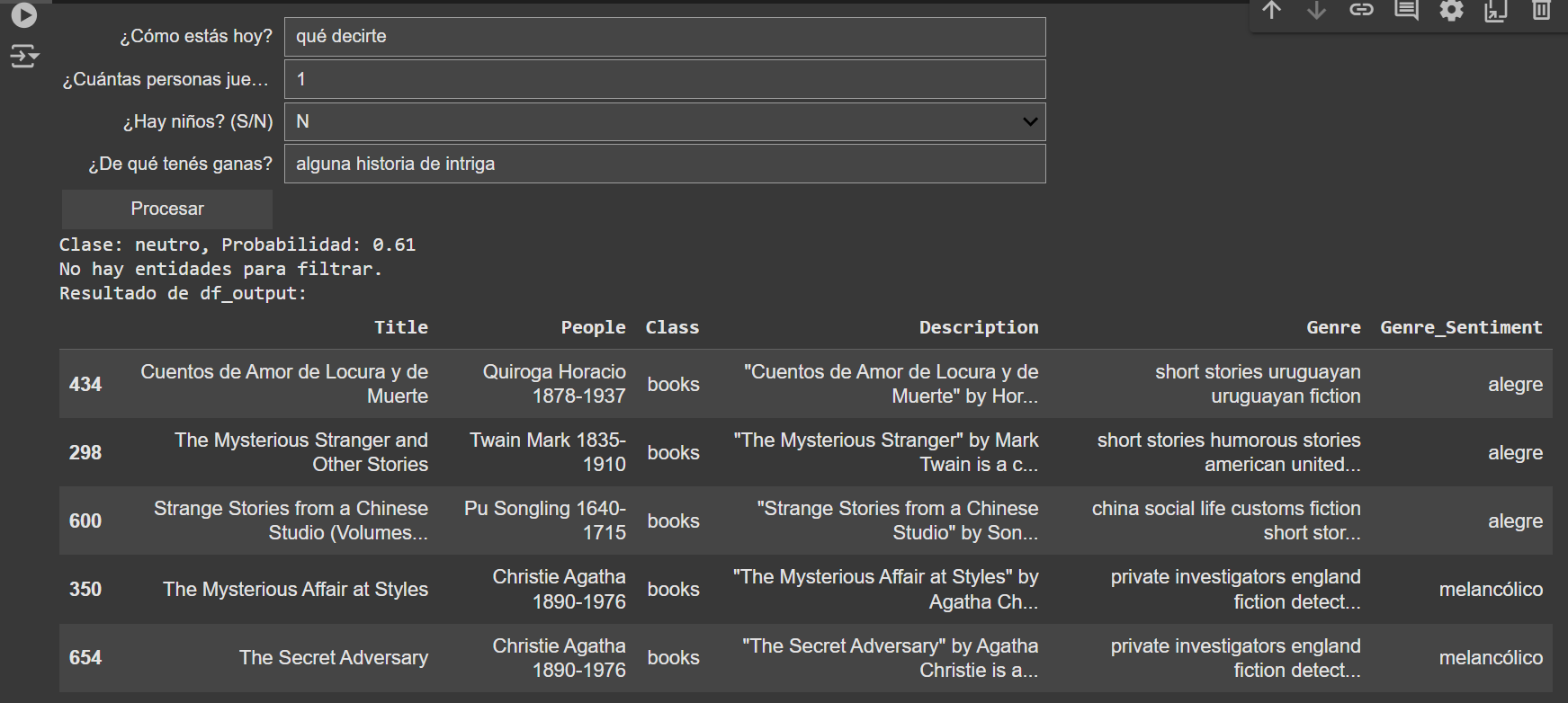
* **Entrada:**
  1. df\_all: El dataframe con todos los datos disponibles.
  2. app: La función previamente definida que realiza el filtrado y procesamiento de los datos.
* **Proceso:**
  1. **Creación de widgets**: Se crean cuatro widgets para que el usuario ingrese los datos:
     + Un campo de texto para el estado de ánimo (input\_1\_widget).
     + Un campo numérico para la cantidad de jugadores (input\_2\_widget).
     + Un desplegable para indicar si hay o no niños (input\_3\_widget).
     + Un campo de texto para que el usuario ingrese sus preferencias (input\_4\_widget).
  2. **Botón de procesamiento**: Se crea un botón que al ser presionado, llama a la función on\_button\_click. La misma recoge los valores de los widgets, llama a la función app para procesar los datos y muestra los resultados en un widget de salida (result\_output).
  3. **Mostrar resultados**: Cuando el usuario hace clic en el botón, la función on\_button\_click procesa los datos, ejecuta la función app y muestra el dataframe df\_output, que contiene las actividades recomendadas.
* **Salida:** La interfaz interactiva permite al usuario ver los resultados de la función app de manera visual, mostrando las actividades recomendadas según las entradas proporcionadas.

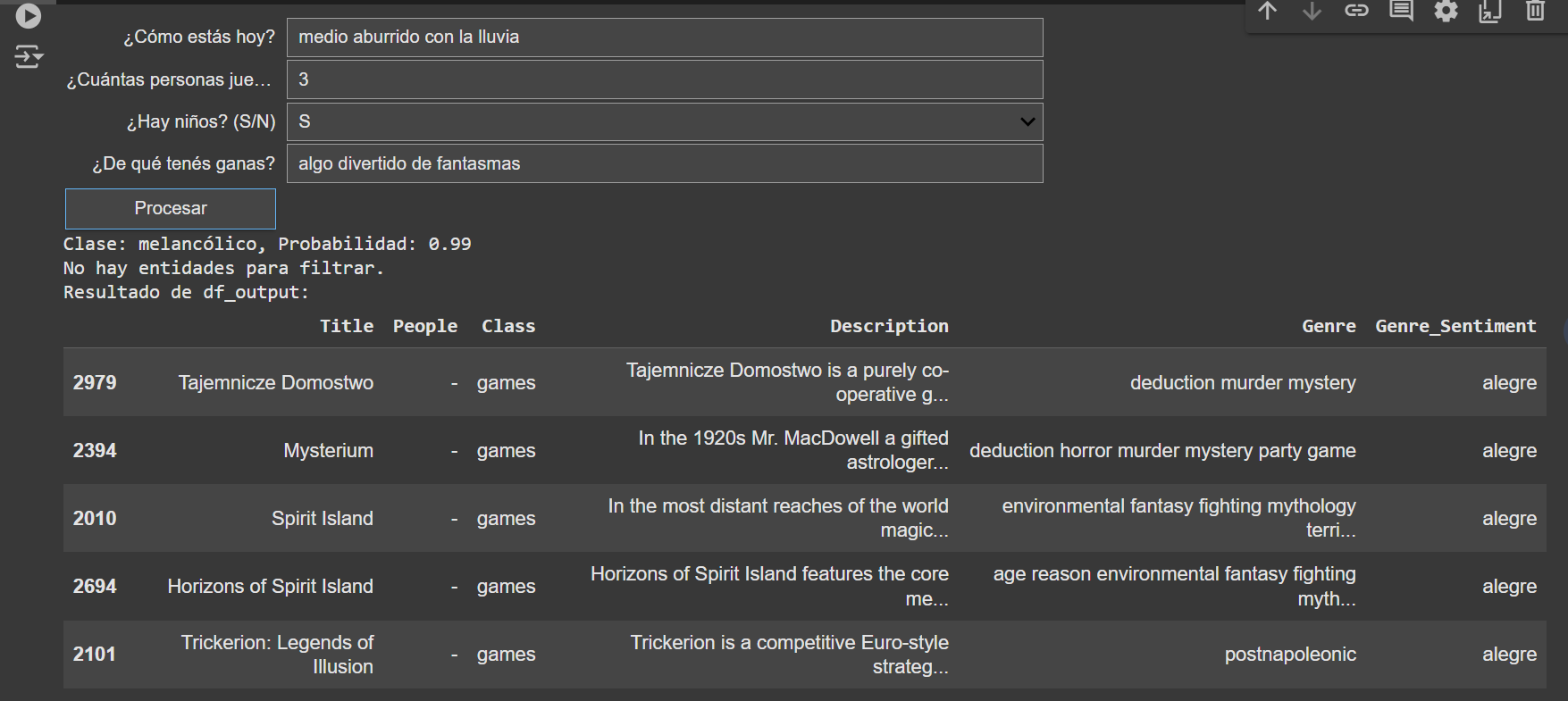
# RESULTADOS

Presentación de los resultados obtenidos.









# CONCLUSIONES

Se obtiene un programa que permite sugerir actividades de manera robusta en base a las entradas del usuario.

El análisis de sentimientos fue la parte más desafiante del trabajo. Se abordaron diferentes estrategias, como por ejemplo la implementación del modelo para análisis de sentimientos basado en BERT "nlptown/bert-base-multilingual-uncased-sentiment", sin obtener grandes resultados.

El entrenamiento de una red de regresión logística en base a los embeddings generados del dataset\_estado\_animo.txt permitió resolver este problema con una cantidad de ejemplos limitada de forma más que satisfactoria.

Se consiguió entonces cumplir con todos los requerimientos solicitados.

# ANEXOS

Dentro del repositorio <https://github.com/juliangg17/NLP-TUIA-Garcia-Herrera> se encuentran las bases de datos, datasets, embeddings y el notebook.

* IMDB-Movie-Data.csv
* NLP\_TP\_1.ipynb
* bgg\_database.csv
* books.csv
* dataset\_estado\_animo.txt
* embeddings\_data.csv
* README.md