# Metacritic - Análisis de Reseñas de Videojuegos Históricos

## Integrantes:

* Sergio Renato Zegarra Villanueva (a20190028)
* Máximo Alejandro Guzmán Huamán (f1446907)

## Definición del Problema

La percepción de los videojuegos varía significativamente entre los críticos y los jugadores. Mientras que Metacritic proporciona una puntuación promedio basada en reseñas de críticos especializados, las opiniones de los usuarios reflejan una experiencia subjetiva más cercana a la comunidad. En muchos casos, las diferencias entre ambas evaluaciones pueden ser sustanciales y responder a diversos factores, como expectativas, actualizaciones del juego, errores técnicos o elementos de jugabilidad.

Este proyecto busca responder a la siguiente pregunta clave: ¿**Existe una correlación entre la puntuación de los videojuegos en Metacritic y el sentimiento expresado en las reseñas de los jugadores?** En caso de discrepancias, se analizarán las posibles causas que explican por qué ciertos juegos pueden ser bien evaluados por la crítica, pero mal recibidos por los jugadores (o viceversa).

## Objetivos del Proyecto

1. Extracción de Datos: Desarrollar un scraper automatizado para recopilar información clave de los videojuegos y sus reseñas en Metacritic.
2. Análisis de Sentimiento: Aplicar técnicas de procesamiento de lenguaje natural (NLP) para determinar el sentimiento predominante en las reseñas de los jugadores.
3. Comparación con Puntuaciones Críticas: Evaluar la relación entre la puntuación de Metacritic y el sentimiento expresado en las reseñas.
4. Identificación de Patrones: Clasificar los videojuegos según el grado de coincidencia o discrepancia entre la puntuación de críticos y el sentimiento de los jugadores.
5. Visualización de Resultados: Presentar análisis gráficos sobre la relación entre las calificaciones de los juegos y la percepción del público.
6. Generación de Insights: Determinar las posibles razones detrás de las discrepancias encontradas y ofrecer conclusiones sobre la percepción de los videojuegos.

## Descripción del Proyecto

Este proyecto tiene como objetivo recopilar y analizar las reseñas de los mejores videojuegos de todos los tiempos utilizando la plataforma Metacritic (https://www.metacritic.com/). Para lograrlo, se desarrolló un scraper automatizado con Selenium y BeautifulSoup, permitiendo la extracción estructurada de datos clave de los videojuegos y sus respectivas reseñas de usuarios.

El proceso consta de dos fases principales:

**1. Extracción de Datos de los Mejores Videojuegos:** Se recolectaron información relevante de los juegos en las primeras 50 páginas del sitio web, con un total de 1,200 juegos registrados (24 juegos por página). Para cada juego, se extrajeron los siguientes atributos:

* Nombre del juego
* Fecha de lanzamiento
* Calificación de los usuarios
* Descripción
* Metascore (puntaje de críticos)
* Enlace directo a la página del juego en Metacritic

**2. Extracción de Reseñas de Usuarios:** A partir de los enlaces recopilados en la primera fase, se ingresó a cada juego para obtener hasta 1,000 reseñas por título. Debido a que no todos los juegos cuentan con una gran cantidad de reseñas, la cantidad final de datos recopilados fue de 188,720 reseñas.

## Detalles Técnicos

* **Automatización con Selenium**: Se configuró un WebDriver para navegar en el sitio, interactuar con la página y esperar la carga dinámica de los elementos.
* **Extracción con BeautifulSoup:** Se utilizó esta librería para analizar el HTML y extraer la información requerida.
* **Manejo de Paginación:** Se recorrieron 50 páginas de Metacritic para obtener el listado de juegos.
* **Extracción de Datos con Robustez:** Se implementaron esperas explícitas para garantizar la carga de elementos y evitar bloqueos en la navegación.
* **Almacenamiento Eficiente:** Los datos se guardaron en archivos CSV dividiendo la información en partes de máximo 20MB, optimizando el rendimiento en el manejo de grandes volúmenes de datos.

## Repositorio del Proyecto

El código fuente del scraper y los datos recopilados están disponibles en el siguiente repositorio de GitHub: Metacritic Game Scraper (<https://github.com/MaximoGuzmanH/metacritic-game-scraper.git>).

Este proyecto proporciona una base de datos extensa para el análisis de la percepción de los videojuegos a lo largo de la historia y la correlación entre las puntuaciones de críticos y usuarios.

### Scraper para Obtener los Mejores Juegos y sus Reseñas

El proceso de scraping consta de los siguientes pasos:

1. **Configuración del WebDriver:** Se establece un WebDriver de Selenium con opciones para mejorar el rendimiento y evitar restricciones de acceso.
2. **Extracción de Videojuegos:**
   * Se recorren 50 páginas del sitio Metacritic.
   * Se identifican los juegos y se extrae su información clave.
   * Se almacenan los datos en un archivo CSV, segmentando los archivos para no superar los 20MB.
3. **Extracción de Reseñas de Usuarios:**
   * Se accede a cada juego individualmente a través de su enlace.
   * Se recogen hasta 1,000 reseñas por juego, extrayendo usuario, puntaje, fecha y contenido de la reseña.
   * Se almacena la información en CSV de manera eficiente.

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from bs4 import BeautifulSoup

import time

import os

import pandas as pd

# Configuración inicial del driver

def setup\_driver():

    """

    Configura el WebDriver de Selenium con opciones específicas para mejorar el rendimiento y evitar problemas.

    """

    options = webdriver.ChromeOptions()

    options.add\_argument("--disable-gpu")

    options.add\_argument("--no-sandbox")

    options.add\_argument("--start-maximized")

    driver\_path = r"C:\\Users\\Maximo Guzman\\Desktop\\Metacritic - Scraper\\chromedriver.exe"

    driver = webdriver.Chrome(service=webdriver.chrome.service.Service(driver\_path), options=options)

    return driver

# Función para dividir y guardar la data en archivos CSV de máximo 20MB

def save\_to\_csv(filename, data):

    """

    Guarda los datos en archivos CSV de máximo 20 MB cada uno.

    """

    max\_size = 20 \* 1024 \* 1024  # 20MB en bytes

    file\_count = 1

    chunk\_size = 10000  # Cantidad de filas por bloque (ajustable)

    df = pd.DataFrame(data)

    while True:

        file\_path = f"{filename}\_{file\_count}.csv"

        df\_chunk = df.iloc[(file\_count - 1) \* chunk\_size : file\_count \* chunk\_size]

        if df\_chunk.empty:

            break

        df\_chunk.to\_csv(file\_path, index=False, encoding="utf-8")

        # Si el archivo supera los 20MB, reducimos el chunk size para los siguientes archivos

        if os.path.getsize(file\_path) > max\_size:

            chunk\_size = int(chunk\_size \* 0.8)

        file\_count += 1

    print(f"Datos guardados en {file\_count - 1} archivo(s) '{filename}\_\*.csv'.")

# Función principal para realizar el scraping de Metacritic

def scrape\_metacritic():

    start\_time = time.time()

    driver = setup\_driver()

    base\_url = "https://www.metacritic.com/browse/game/?releaseYearMin=1958&releaseYearMax=2025&page={}"

    all\_data = []  # Lista para almacenar la información de los juegos

    # Iterar a través de 50 páginas del sitio web

    for page in range(1, 51):

        url = base\_url.format(page)

        print(f"Procesando: {url}")

        driver.get(url)

        WebDriverWait(driver, 20).until(

            EC.presence\_of\_element\_located((By.CLASS\_NAME, "c-productListings"))

        )

        soup = BeautifulSoup(driver.page\_source, "html.parser")

        section\_id = f"detailed|{page}"

        main\_div = soup.find("div", {"section": section\_id, "class": "c-productListings"})

        if not main\_div:

            print(f"No se encontró el div especificado en la página {page}.")

            continue

        game\_cards = main\_div.find\_all("div", class\_="c-finderProductCard")

        for card in game\_cards:

            try:

                title = card.find("div", class\_="c-finderProductCard\_title")

                title = title.get\_text(strip=True) if title else "N/A"

                release\_date = card.find("span", class\_="u-text-uppercase")

                release\_date = release\_date.get\_text(strip=True) if release\_date else "N/A"

                rating = card.find("span", class\_="u-text-capitalize")

                rating = rating.find\_next\_sibling(string=True).strip() if rating else "N/A"

                description = card.find("div", class\_="c-finderProductCard\_description")

                description = description.get\_text(strip=True) if description else "N/A"

                metascore = card.find("div", class\_="c-siteReviewScore")

                metascore = metascore.get\_text(strip=True) if metascore else "N/A"

                link = card.find("a", href=True)

                link = f"https://www.metacritic.com{link['href']}" if link else "N/A"

                all\_data.append({

                    "Title": title,

                    "Release Date": release\_date,

                    "Rating": rating,

                    "Description": description,

                    "Metascore": metascore,

                    "Link": link

                })

            except AttributeError as e:

                print(f"Error al procesar una tarjeta de juego: {e}")

    # Guardar la lista de juegos en archivos CSV

    if all\_data:

        save\_to\_csv("metacritic\_detailed\_games\_all\_time", all\_data)

    else:

        print("No se encontraron datos para guardar.")

    # Scraping de reseñas

    scrape\_reviews(all\_data, driver)

    driver.quit()

    print(f"Tiempo total de ejecución: {time.time() - start\_time:.2f} segundos.")

# Función para extraer reseñas de los juegos

def scrape\_reviews(games\_data, driver):

    reviews\_data = []

    for game in games\_data:

        link = game.get("Link")

        if link == "N/A":

            continue

        print(f"Extrayendo reseñas de: {link}")

        driver.get(link)

        try:

            user\_reviews\_tab = WebDriverWait(driver, 15).until(

                EC.element\_to\_be\_clickable((By.LINK\_TEXT, "User Reviews"))

            )

            user\_reviews\_tab.click()

            WebDriverWait(driver, 15).until(

                EC.presence\_of\_element\_located((By.CLASS\_NAME, "c-siteReview\_main"))

            )

        except Exception as e:

            print(f"No se pudo acceder a las reseñas para {game['Title']}: {e}")

            continue

        try:

            scroll\_pause\_time = 2

            last\_height = driver.execute\_script("return document.body.scrollHeight")

            while True:

                driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

                time.sleep(scroll\_pause\_time)

                new\_height = driver.execute\_script("return document.body.scrollHeight")

                if new\_height == last\_height:

                    break

                last\_height = new\_height

            soup = BeautifulSoup(driver.page\_source, "html.parser")

            reviews = soup.find\_all("div", class\_="c-siteReview\_main")

            if not reviews:

                print(f"No se encontraron reseñas para el juego '{game['Title']}'.")

                continue

            for review in reviews[:1000]:

                try:

                    user = review.find("a", class\_="c-siteReviewHeader\_username")

                    user = user.get\_text(strip=True) if user else "Anonymous"

                    score = review.find("div", class\_="c-siteReviewScore")

                    score = score.get\_text(strip=True) if score else "N/A"

                    date = review.find("div", class\_="c-siteReviewHeader\_reviewDate")

                    date = date.get\_text(strip=True) if date else "N/A"

                    content = review.find("div", class\_="c-siteReview\_quote")

                    content = content.get\_text(strip=True) if content else "N/A"

                    reviews\_data.append({

                        "Game": game["Title"],

                        "User": user,

                        "Score": score,

                        "Date": date,

                        "Content": content,

                        "Link": link

                    })

                except AttributeError as e:

                    print(f"Error al procesar una reseña: {e}")

        except Exception as e:

            print(f"Error al cargar reseñas para {game['Title']}: {e}")

    # Guardar las reseñas en archivos CSV

    if reviews\_data:

        save\_to\_csv("metacritic\_reviews\_all\_time", reviews\_data)

    else:

        print("No se encontraron datos para guardar.")

if \_\_name\_\_ == "\_\_main\_\_":

    scrape\_metacritic()

**Datos Relevantes del Proceso de Extracción**

Durante la ejecución del scraper, se observaron los siguientes puntos clave:

* División de Archivos: Dado el gran volumen de datos, se implementó una estrategia para dividir las reseñas en archivos CSV que contuvieran un máximo de 10,000 registros cada uno. Como resultado, se generaron un total de 19 archivos con las reseñas recopiladas de los juegos.
* Tiempo de Ejecución: La ejecución completa del proceso de scraping tomó 18,061.66 segundos, lo que equivale aproximadamente a 301.02 minutos o 5.01 horas.
* Estos datos reflejan la magnitud del proyecto y la cantidad de información extraída, asegurando que la base de datos obtenida sea lo suficientemente amplia y detallada para futuros análisis.

### Carga y Consolidación de Datos desde GitHub

Después de extraer y procesar las reseñas de los videojuegos, decidimos almacenar los 19 archivos generados en un repositorio público de GitHub: <https://github.com/MaximoGuzmanH/metacritic-game-scraper.git>

El objetivo de esta acción fue permitir que cualquier persona con acceso al Notebook pudiera ejecutar el código sin necesidad de descargar y gestionar los archivos localmente.

**Proceso de Consolidación y Limpieza de Datos**

Para facilitar el análisis, realizamos los siguientes pasos:

1. Carga de los Archivos desde GitHub
   * Se definió la URL base del repositorio y se iteró sobre los 19 archivos disponibles.
   * Se cargaron los archivos en un proceso secuencial, asegurando la correcta estructuración de los datos.
2. Concatenación de los Archivos
   * Una vez cargados, los archivos se unieron en un único dataset consolidado.
   * Se eliminó cualquier registro vacío o duplicado para mantener la integridad de los datos.
3. Corrección de Problemas de Codificación
   * Se aplicó una función de limpieza para corregir caracteres especiales mal codificados en la columna de contenido (Content).
   * Se utilizó una conversión de latin1 a utf-8 para asegurar que los textos fueran legibles y coherentes.
4. Generación del Dataset Final
   * Se guardó una copia del dataset consolidado en un archivo CSV (df\_original\_cleaned.csv).
   * Se imprimieron estadísticas clave sobre los datos procesados:
   * Registros antes de limpieza: cantidad total de reseñas cargadas inicialmente.
   * Registros después de limpieza: cantidad final tras la eliminación de filas vacías o duplicadas.
   * Diferencia: cantidad de registros eliminados en el proceso de depuración.

**Tiempo de Ejecución**

El tiempo total necesario para la carga, consolidación y limpieza de datos fue de 24.08 segundos.

Este proceso asegura que los datos estén estructurados correctamente y listos para su posterior análisis, sin errores de codificación ni redundancias.

import pandas as pd

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# Lista de URLs de los archivos en GitHub

base\_url = "https://raw.githubusercontent.com/MaximoGuzmanH/metacritic-game-scraper/adc408e8064ca3cd97c1960192eabec0ad41d4b4/metacritic\_reviews\_all\_time\_"

file\_count = 19  # Total de archivos

# Lista para almacenar los DataFrames

dataframes = []

initial\_record\_count = 0  # Contador para registros originales

# Función para corregir caracteres mal codificados en la columna 'Content'

def clean\_text(text):

    if isinstance(text, str):

        try:

            text = text.encode('latin1').decode('utf-8')  # Corregir problemas de codificación

        except UnicodeDecodeError:

            pass  # Si no se puede corregir, dejar el texto como está

    return text

# Cargar los archivos en un solo proceso iterativo

for i in range(1, file\_count + 1):

    file\_url = f"{base\_url}{i}.csv"

    print(f"📥 Cargando archivo {i}/{file\_count}: {file\_url}")

    # Leer el CSV, asegurando nombres correctos y eliminando filas vacías

    df = pd.read\_csv(file\_url, skiprows=1 if i > 1 else 0, names=dataframes[0].columns if i > 1 else None, encoding='latin1').dropna(how='all')

    # Aplicar limpieza de caracteres raros en la columna 'Content'

    if 'Content' in df.columns:

        df['Content'] = df['Content'].apply(clean\_text)

    initial\_record\_count += len(df)  # Contar registros iniciales

    dataframes.append(df)

# Unir todos los DataFrames en uno solo

df\_final = pd.concat(dataframes, ignore\_index=True)

final\_record\_count = len(df\_final)  # Contar registros después de la limpieza

# Guardar una copia del dataset original antes de modificaciones

original\_csv\_filename = "df\_original\_cleaned.csv"

df\_final.to\_csv(original\_csv\_filename, index=False, encoding='utf-8')

print(f"\n✅ Se ha guardado una copia del dataset limpio como '{original\_csv\_filename}'.")

# Mostrar información sobre los datos

print(f"\n📊 Información sobre los datos:")

print(f"   🔹 Registros cargados inicialmente: {initial\_record\_count}")

print(f"   🔹 Registros después de limpieza: {final\_record\_count}")

print(f"   🔹 Diferencia: {initial\_record\_count - final\_record\_count} registros eliminados (vacíos o duplicados)")

# Mostrar una vista previa del dataset original

print("\n🔍 Vista previa del dataset:")

print(df\_final[['Game', 'Content']].head(10))  # Se enfoca en 'Game' y 'Content'

# Detener el cronómetro y mostrar el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"\n⏱️ Tiempo total de ejecución: {execution\_time:.2f} segundos")

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

### Detección de Idioma en las Reseñas

Dado que los usuarios de Metacritic pueden escribir sus reseñas en distintos idiomas, se realizó un análisis para detectar el idioma de cada review y facilitar el procesamiento posterior de los datos.

**Proceso de Detección de Idioma**

1. Revisión Preliminar
   * Se observó que las reseñas pueden estar en inglés, español, francés, alemán, italiano, portugués, neerlandés, entre otros idiomas.
   * Se decidió implementar un proceso automático para detectar el idioma de cada reseña.
2. Implementación del Algoritmo
   * Se utilizó la librería **\*\*langdetect\*\*** para identificar el idioma de cada comentario.
   * Se creó un mapeo de códigos de idioma a nombres completos para facilitar la interpretación de los resultados.
   * Se aseguraron resultados consistentes utilizando DetectorFactory.seed = 0.
3. Aplicación sobre el Dataset
   * Se aplicó la detección de idioma a la columna Content que contiene las reseñas.
   * Se generaron dos nuevas columnas en el dataset:
     + Language\_code: Código del idioma detectado (ej. en para inglés, es para español).
     + Language\_name: Nombre completo del idioma detectado (ej. English, Spanish).
   * En caso de que la detección fallara, se asignó la etiqueta "Unknown".
4. Generación del Nuevo Dataset
   * Se guardó un nuevo archivo CSV con todas las reseñas y sus respectivos idiomas detectados: 📂 df\_final\_languages.csv

**Resumen del Procesamiento**

* Registros analizados: cantidad total de reseñas procesadas.
* Registros después del procesamiento: cantidad final tras la detección de idioma.
* Registros afectados: diferencia entre la cantidad inicial y final.
* Tiempo total de ejecución: 352.49 segundos o 5.87 minutos.

Este análisis permite segmentar las reseñas por idioma y facilita estudios específicos basados en el idioma en que fueron escritas.

from langdetect import detect, DetectorFactory, LangDetectException

import pandas as pd

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# Asegurar resultados consistentes en langdetect

DetectorFactory.seed = 0

# Mapeo de códigos de idioma a nombres completos

LANGUAGE\_MAPPING = {

    "en": "English",

    "es": "Spanish",

    "fr": "French",

    "de": "German",

    "it": "Italian",

    "pt": "Portuguese",

    "nl": "Dutch",

    "unknown": "Unknown"

}

# Función optimizada para detectar idioma (evita aplicar str conversion varias veces)

def detect\_language(text):

    try:

        return detect(text) if isinstance(text, str) else "unknown"

    except LangDetectException:

        return "unknown"  # Si falla la detección

# 📌 Contar registros iniciales

initial\_count = len(df\_final)

# Aplicar la detección de idioma usando .progress\_apply() para ver barra de progreso si tqdm está disponible

print("🚀 Detectando idiomas en las reviews...")

df\_final['Language\_code'] = df\_final['Content'].astype(str).apply(detect\_language)

df\_final['Language\_name'] = df\_final['Language\_code'].map(LANGUAGE\_MAPPING).fillna("Other")

# 📌 Contar registros después de detección de idioma

final\_count = len(df\_final)

# Guardar un archivo CSV con TODAS las columnas

csv\_filename = "df\_final\_languages.csv"

df\_final.to\_csv(csv\_filename, index=False)  # Guarda todas las columnas

print(f"\n✅ Archivo CSV generado con TODAS las columnas: '{csv\_filename}'")

# Detener el cronómetro y mostrar el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información del procesamiento

print("\n📊 Resumen de procesamiento de detección de idiomas:")

print(f"   🔹 Registros analizados: {initial\_count}")

print(f"   🔹 Registros después del procesamiento: {final\_count}")

print(f"   🔹 Registros afectados: {final\_count - initial\_count}")  # Diferencia si hay cambios inesperados

# ⏱ Mostrar tiempo total de ejecución

print(f"\n✅ Tiempo total de ejecución para la detección de idioma: {execution\_time:.2f} segundos")

# 🔍 Mostrar una vista previa del dataset actualizado

print("\n🔍 Vista previa de las primeras 10 filas del dataset actualizado:")

print(df\_final.head(10))

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

### Distribución de Idiomas en las Reseñas

Para comprender mejor la composición del dataset en términos de idioma, realizamos un análisis cuantitativo de la cantidad de reseñas escritas en inglés, español y otros idiomas.

**Resumen de Cálculo**

1. Conteo de Reseñas por Idioma
   * Se calculó el número total de reseñas disponibles en el dataset.
   * Se utilizó la columna Language\_code para contar la cantidad de reseñas en cada idioma.
2. Cálculo de Porcentajes
   * Se calcularon los porcentajes de reseñas escritas en inglés (en), español (es) y otros idiomas.
   * Se organizó la información en una estructura clara y fácil de interpretar.
3. Resultados Finales
   * Total de reseñas analizadas: 188,720
   * Reseñas en inglés: 160,282 (84.93%)
   * Reseñas en español: 10,431 (5.53%)
   * Reseñas en otros idiomas: 18,007 (9.54%)
4. Tiempo de Ejecución
   * El proceso de conteo y cálculo de porcentajes tomó aproximadamente 0.0082 segundos, asegurando una ejecución eficiente.

Este análisis proporciona una visión general sobre la distribución de idiomas en el dataset, facilitando futuras segmentaciones y análisis específicos por idioma.

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# Contar el número total de reviews

total\_reviews = len(df\_final)

# Contar las reviews por idioma usando value\_counts() para mejor rendimiento

language\_counts = df\_final['Language\_code'].value\_counts().to\_dict()

# Obtener los valores específicos de inglés y español, si no existen, asignar 0

english\_reviews = language\_counts.get('en', 0)

spanish\_reviews = language\_counts.get('es', 0)

# Calcular las reviews en otros idiomas

other\_reviews = total\_reviews - (english\_reviews + spanish\_reviews)

# Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar los resultados

print("\n📊 \*\*Resumen de conteo de reviews por idioma:\*\*")

print(f"   🔹 Total de reviews: {total\_reviews}")

print(f"   🔹 Reviews en inglés: {english\_reviews} ({round(100 \* english\_reviews / total\_reviews, 2)}%)")

print(f"   🔹 Reviews en español: {spanish\_reviews} ({round(100 \* spanish\_reviews / total\_reviews, 2)}%)")

print(f"   🔹 Reviews en otros idiomas: {other\_reviews} ({round(100 \* other\_reviews / total\_reviews, 2)}%)")

# ⏱ Mostrar tiempo total de ejecución

print(f"\n✅ Tiempo total de ejecución para contar las reviews: {execution\_time:.4f} segundos")

Texto

El contenido generado por IA puede ser incorrecto.

### Análisis del Promedio de Palabras en Reseñas en Inglés

Para definir la longitud óptima de los resúmenes generados a partir de las reseñas, realizamos un análisis para calcular el promedio de palabras utilizadas en las reviews escritas en inglés.

**Proceso de Análisis**

1. Filtrado de Reseñas en Inglés
   * Se seleccionaron únicamente las reseñas que fueron detectadas como escritas en inglés (Language\_code = 'en').
   * Se calculó la cantidad total de reseñas en inglés disponibles en el dataset.

b. Cálculo del Conteo de Palabras

* + Se utilizó una técnica optimizada con pandas para calcular la cantidad de palabras en cada review.
  + Se descartaron valores nulos y se aseguraron formatos numéricos para mayor precisión.

c. Cálculo del Promedio de Palabras

* + Se obtuvo la media de palabras por reseña en inglés, proporcionando un valor de referencia para la generación de resúmenes.

d. Resultados Finales

* + Total de reseñas en el dataset: 188,720
  + Cantidad de reseñas en inglés analizadas: 160,282
  + Promedio de palabras por review en inglés: 97.04 palabras

e. Tiempo de Ejecución

* + El cálculo de conteo y promedio de palabras tomó aproximadamente 1.3442 segundos, garantizando eficiencia en el procesamiento.

Este análisis servirá como una métrica clave para establecer un límite adecuado en la cantidad de palabras generadas en los resúmenes automáticos, asegurando que reflejen con precisión el contenido original sin ser excesivamente largos o cortos.

import pandas as pd

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# 📌 Contar registros antes del filtrado

total\_reviews = len(df\_final)

# Filtrar solo las reseñas en inglés

df\_english = df\_final[df\_final['Language\_code'] == 'en']

# Contar el número de reviews en inglés

english\_reviews = len(df\_english)

print(f"📌 Total de reviews en el dataset: {total\_reviews}")

print(f"📌 Reviews en inglés analizadas: {english\_reviews}")

# Contar palabras usando pandas (más rápido que apply)

df\_english['Word\_Count'] = df\_english['Content'].str.split().str.len().fillna(0).astype(int)

# Calcular el promedio de palabras por review en inglés

average\_word\_count = df\_english['Word\_Count'].mean()

# Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar el resultado

print(f"\n📊 Promedio de palabras por review en inglés: {average\_word\_count:.2f}")

print(f"⏱ Tiempo total de ejecución para contar palabras: {execution\_time:.4f} segundos")

Texto

El contenido generado por IA puede ser incorrecto.

### Preprocesamiento de Datos para Reseñas en Inglés y Español

Luego de identificar el idioma de cada reseña, se realizó un proceso de limpieza y preprocesamiento del texto para normalizar los datos y optimizar futuras tareas de análisis y modelado.

**Proceso de Preprocesamiento**

1. Definición de Stopwords por Idioma
   * Se utilizaron las listas de palabras vacías (stopwords) de nltk para eliminar términos comunes sin valor semántico en inglés y español.
   * Se aseguraron descargas automáticas de los recursos necesarios (punkt, stopwords) en caso de no estar disponibles.

b. Tokenización y Limpieza de Texto

* + Se aplicó word\_tokenize de nltk para dividir cada reseña en palabras individuales.
  + Se eliminaron caracteres no alfabéticos y se convirtió todo el texto a minúsculas.
  + Se filtraron las palabras vacías (stopwords) según el idioma de cada reseña.

c. Optimización del Procesamiento

* + Se intentó utilizar swifter para acelerar el procesamiento con múltiples núcleos de CPU o GPU.
  + En caso de que swifter no estuviera disponible, se aplicó un método estándar utilizando pandas.apply().

d. Resultados del Preprocesamiento

* + Se generó una nueva columna en el dataset (content\_tokens) con las reseñas preprocesadas.
  + Se conservaron únicamente las palabras relevantes para análisis posteriores.

e. Tiempo de Ejecución

* + El proceso completo de tokenización y limpieza tomó aproximadamente 50.6931 segundos, optimizando el rendimiento del preprocesamiento.

Este preprocesamiento garantiza que los datos textuales estén en un formato adecuado para análisis de sentimientos, resúmenes automáticos y otros modelos de procesamiento de lenguaje natural.

import time  # Para medir el tiempo de ejecución

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

# Iniciar el cronómetro

start\_time = time.time()

# Descargar recursos de nltk solo si no están disponibles

nltk\_resources = ["punkt", "stopwords"]

for resource in nltk\_resources:

    try:

        nltk.data.find(f"tokenizers/{resource}")

    except LookupError:

        print(f"📥 Descargando {resource}...")

        nltk.download(resource)

# 📌 Definir stopwords por idioma (inglés y español)

stop\_words\_en = set(stopwords.words("english"))

stop\_words\_es = set(stopwords.words("spanish"))

# Función de preprocesamiento optimizada

def text\_preprocess(text, language):

    if isinstance(text, str):  # Verificar si es texto

        if language == "English":

            stop\_words = stop\_words\_en

        elif language == "Spanish":

            stop\_words = stop\_words\_es

        else:

            return text  # Si el idioma no es reconocido, devolver el texto original

        tokens = [

            token.lower() for token in word\_tokenize(text)

            if token.isalpha() and token.lower() not in stop\_words

        ]

        return ' '.join(tokens)

    return text  # Si el texto es nulo, devolverlo sin cambios

# Aplicar la función usando swifter si está disponible (aceleración en CPU/GPU)

try:

    import swifter

    print("⚡ Usando swifter para acelerar la tokenización...")

    df\_final["content\_tokens"] = df\_final.swifter.apply(lambda row: text\_preprocess(row["Content"], row["Language\_name"]), axis=1)

except ImportError:

    print("⏳ Ejecutando tokenización sin swifter (puede ser más lento)...")

    df\_final["content\_tokens"] = df\_final.apply(lambda row: text\_preprocess(row["Content"], row["Language\_name"]), axis=1)

# Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar el tiempo total de ejecución

print(f"\n✅ Tiempo total de ejecución para la tokenización de texto: {execution\_time:.4f} segundos")

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

### Agrupación y Concatenación de Reseñas por Juego

Luego del preprocesamiento de las reseñas, el siguiente paso fue consolidar la mayor cantidad posible de comentarios para cada videojuego, permitiendo un análisis más eficiente y mejorando la calidad de los resúmenes generados en futuras etapas.

**Proceso de Agrupación**

1. Filtrado de Reseñas en Inglés
   * Se seleccionaron exclusivamente las reseñas en inglés (Language\_code = 'en') para garantizar uniformidad en el análisis.
2. Limpieza y Concatenación de Reseñas
   * Se eliminaron valores nulos y se limpiaron espacios innecesarios dentro del texto.
   * Se concatenaron todas las reseñas de un mismo juego en un solo bloque de texto.
   * Se estableció un límite de 32,000 caracteres por juego para evitar sobrecargas en futuras etapas de procesamiento.
3. Agrupación de Datos
   * Se agruparon todas las reseñas por el nombre del juego (Game).
   * Se almacenaron las reseñas concatenadas en un nuevo dataset.
4. Resultados del Proceso
   * Total de reseñas procesadas: 160,282
   * Número de juegos antes de agrupar: 1,145
   * Número de juegos con reseñas agrupadas: 1145
5. Generación del Nuevo Dataset

* Se guardó un nuevo archivo CSV con las reseñas agrupadas por juego: 📂 grouped\_reviews\_all.csv

1. Tiempo de Ejecución
   * El proceso completo de agrupación y limpieza tomó aproximadamente 3.0607 segundos, garantizando un rendimiento eficiente.

Este paso permite trabajar con un conjunto de datos más compacto y útil para tareas como generación de resúmenes automáticos y análisis de tendencias dentro de las reseñas de los videojuegos.

import pandas as pd

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# 📌 Filtrar solo las reviews en inglés dentro de df\_final

df\_english = df\_final[df\_final['Language\_code'] == 'en'].copy()

# 📊 Contar registros antes de la agrupación

total\_reviews = len(df\_english)

unique\_games\_before = df\_english['Game'].nunique()

# Función para limpiar y concatenar reviews correctamente

def clean\_and\_join\_reviews(reviews, max\_length=32000):

    cleaned\_reviews = reviews.dropna().astype(str).str.replace(r'\s+', ' ', regex=True)  # Eliminar espacios extraños

    full\_text = cleaned\_reviews.str.cat(sep=". ")  # Concatenar usando str.cat() (más rápido)

    # Si la longitud excede el límite, se trunca

    return full\_text[:max\_length]

# 🚀 Agrupar los reviews por juego y limpiar el texto antes de concatenar

grouped\_reviews = df\_english.groupby('Game', as\_index=False)['Content'].agg(clean\_and\_join\_reviews)

# 📊 Contar juegos después de la agrupación

total\_games = grouped\_reviews.shape[0]

# Guardar el DataFrame corregido con todos los juegos

csv\_filename = "grouped\_reviews\_all.csv"

grouped\_reviews.to\_csv(csv\_filename, index=False)

# Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📌 Mostrar información clave del proceso

print("\n✅ Sección completada: Se han agrupado y guardado los reviews de todos los juegos en 'grouped\_reviews\_all.csv'.")

print(f"⏱ Tiempo total de ejecución para agrupar y limpiar reviews: {execution\_time:.4f} segundos")

print(f"📊 Total de reviews procesadas: {total\_reviews}")

print(f"🎮 Total de juegos antes de agrupar: {unique\_games\_before}")

print(f"🎮 Total de juegos con reviews agrupadas: {total\_games}")

Texto

El contenido generado por IA puede ser incorrecto.

### Generación de Resúmenes de Reseñas por Juego

Una vez que se han concatenado múltiples reseñas en un solo texto por juego, el siguiente paso fue generar un resumen automático para cada videojuego, utilizando un modelo de lenguaje basado en Transformers.

**Proceso de Resumen Automático**

1. Carga de Datos
   * Se utilizó el archivo previamente generado con las reseñas agrupadas por juego (grouped\_reviews\_all.csv).
   * Se contabilizó el número total de videojuegos a procesar.
2. Uso del Modelo T5-Base
   * Se seleccionó el modelo t5-base de Hugging Face Transformers para realizar la tarea de resumen automático.
   * Se utilizó T5Tokenizer para tokenizar el texto y T5ForConditionalGeneration para generar resúmenes.
   * Se aprovechó la aceleración en GPU (CUDA) en caso de estar disponible, mejorando el rendimiento del procesamiento.
3. Generación de Resúmenes
   * Cada reseña concatenada por juego se procesó en lotes (batches) de 50 registros, optimizando el uso de memoria.
   * Se aplicó el modelo T5 con los siguientes parámetros:
     + Longitud máxima del resumen: 100 palabras
     + Longitud mínima: 30 palabras
     + Penalización de longitud: 2.0
     + Estrategia de búsqueda: beam search con 4 beams
   * Cada lote procesado se guardó en archivos CSV individuales para evitar pérdidas de información.
4. Unificación de Resultados
   * Se consolidaron todos los archivos batch en un solo dataset final.
   * Se generó el archivo grouped\_reviews\_summaries\_all.csv, que contiene el resumen de cada juego basado en sus reseñas agrupadas.
5. Resultados del Proceso
   * Total de juegos procesados: 1145
   * Tiempo total de ejecución: 1401.4975 segundos o 23.35 minutos
   * Archivo final generado: 📂 grouped\_reviews\_summaries\_all.csv

Este paso permite obtener una versión resumida de las reseñas más relevantes de cada juego, facilitando el análisis de opinión y la identificación de patrones en la percepción de los jugadores.

import pandas as pd

from transformers import T5Tokenizer, T5ForConditionalGeneration

import torch

import glob

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# 📌 Verificar si CUDA está disponible y usar GPU si es posible

device = "cuda" if torch.cuda.is\_available() else "cpu"

print(f"🚀 Usando dispositivo: {device.upper()}")

# Cargar el DataFrame agrupado con TODOS los juegos

grouped\_reviews = pd.read\_csv("grouped\_reviews\_all.csv")

# 📊 Contar total de juegos a procesar

total\_games = len(grouped\_reviews)

print(f"📌 Total de juegos a procesar: {total\_games}")

# Cargar el modelo T5-Base en GPU si está disponible

model\_name = "t5-base"

tokenizer = T5Tokenizer.from\_pretrained(model\_name)

model = T5ForConditionalGeneration.from\_pretrained(model\_name).to(device)

# 🚀 Función optimizada para generar resumen con T5-Base

def generate\_summary(text, model, tokenizer, max\_length=100):

    input\_text = "summarize: " + text

    inputs = tokenizer.encode(input\_text, return\_tensors="pt", truncation=True, max\_length=512).to(device)

    with torch.no\_grad():  # Desactivar gradientes para ahorrar memoria en GPU

        summary\_ids = model.generate(inputs, max\_length=max\_length, min\_length=30, length\_penalty=2.0, num\_beams=4, early\_stopping=True)

    return tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)

# 📌 Procesar los resúmenes en batches para evitar bloqueos

batch\_size = 50  # Ajusta este número según tu capacidad de GPU

total\_batches = (len(grouped\_reviews) + batch\_size - 1) // batch\_size  # Redondeo para que incluya todos los datos

for i in range(total\_batches):

    start = i \* batch\_size

    end = min((i + 1) \* batch\_size, len(grouped\_reviews))

    batch = grouped\_reviews.iloc[start:end].copy()  # Extraer una parte

    print(f"📝 Procesando batch {i+1}/{total\_batches} (Registros {start}-{end})...")

    # Aplicar resumen con T5 en GPU

    batch['Summary\_t5-base'] = batch['Content'].apply(

        lambda x: generate\_summary(x, model, tokenizer) if isinstance(x, str) else ""

    )

    # Guardar el batch procesado

    batch.to\_csv(f"batch\_{i}.csv", index=False)

    print(f"✅ Batch {i+1} guardado.")

print("\n✅ Sección 2 completada: Resúmenes generados en batches y guardados.")

# 📌 Unir todos los archivos batch en un solo CSV final

batch\_files = glob.glob("batch\_\*.csv")

df\_summaries = pd.concat([pd.read\_csv(file) for file in batch\_files], ignore\_index=True)

# Guardar el archivo consolidado

csv\_filename = "grouped\_reviews\_summaries\_all.csv"

df\_summaries.to\_csv(csv\_filename, index=False)

# Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información clave

print("\n✅ Sección 3 completada: Archivo final generado como 'grouped\_reviews\_summaries\_all.csv'.")

print(f"⏱ Tiempo total de ejecución para generar resúmenes: {execution\_time:.4f} segundos")

print(f"📌 Total de juegos procesados: {total\_games}")

Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

### Análisis de Sentimiento en los Resúmenes de Reseñas

Una vez generados los resúmenes de reseñas para cada videojuego, se aplicó un análisis de sentimiento para evaluar la percepción general de los jugadores con respecto a cada título.

**Proceso de Análisis de Sentimiento**

1. Carga del Dataset de Resúmenes
   * Se utilizó el archivo previamente generado con los resúmenes de cada juego (grouped\_reviews\_summaries\_all.csv).
   * Se contabilizó el número total de videojuegos a analizar.
2. Uso del Modelo VADER
   * Se utilizó el analizador de sentimiento VADER (Valence Aware Dictionary and sEntiment Reasoner) de nltk.
   * Este modelo es ideal para evaluar sentimiento en textos cortos, como los resúmenes generados.
   * Se descargó el lexicón de VADER en caso de no estar disponible.
3. Cálculo de la Polaridad de Sentimiento
   * Para cada resumen de juego, se obtuvo un puntaje de sentimiento (compound score) en un rango de [-1,1]:
     + Valores cercanos a -1 → Sentimiento negativo
     + Valores cercanos a 0 → Sentimiento neutro
     + Valores cercanos a 1 → Sentimiento positivo
4. Normalización del Puntaje de Sentimiento
   * Para facilitar la interpretación, los valores se transformaron a un rango de 0 a 10:
     + Sentiment Score = (compound score + 1) × 5
   * Esta conversión permite visualizar mejor las diferencias entre juegos con evaluaciones negativas, neutras y positivas.
5. Generación del Dataset Final
   * Se guardó el archivo con los resultados del análisis de sentimiento: 📂 grouped\_reviews\_summaries\_with\_sentiment.csv
6. Resultados del Proceso
   * Total de juegos analizados: 1145
   * Tiempo total de ejecución: 0.8017 segundos
   * Archivo final generado: 📂 grouped\_reviews\_summaries\_with\_sentiment.csv

Este análisis permite obtener una visión global del sentimiento predominante en las reseñas de cada videojuego, proporcionando información útil para estudios de percepción y tendencias en la industria del gaming.

import pandas as pd

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import time  # Para medir el tiempo de ejecución

# 📌 Iniciar el cronómetro

start\_time = time.time()

# 📥 Cargar el dataset con los resúmenes generados

csv\_filename = "grouped\_reviews\_summaries\_all.csv"

df\_summaries = pd.read\_csv(csv\_filename)

# 📊 Contar total de juegos en el dataset

total\_games = len(df\_summaries)

print(f"📌 Total de juegos en el análisis de sentimiento: {total\_games}")

# 📌 Descargar el lexicón de VADER si no está disponible

nltk.download('vader\_lexicon')

# 🚀 Cargar el analizador de sentimiento VADER

vader = SentimentIntensityAnalyzer()

# 🎭 Función para calcular el sentimiento de los resúmenes

def calculate\_vader\_score(text):

    if isinstance(text, str):  # Verificar que sea texto

        vader\_score = vader.polarity\_scores(text)['compound']  # Score en rango [-1, 1]

        return vader\_score

    return 0  # Si no hay texto, devolver 0

# 📝 Aplicar la función de sentimiento a la columna de resúmenes

df\_summaries['Summary\_Sentiment\_Score'] = df\_summaries['Summary\_t5-base'].apply(calculate\_vader\_score)

# 🔄 Normalizar el sentimiento a un rango de 0 a 10

df\_summaries['Summary\_Sentiment\_Score'] = (df\_summaries['Summary\_Sentiment\_Score'] + 1) \* 5  # Convierte [-1,1] a [0,10]

# 📥 Guardar el DataFrame con el análisis de sentimiento

output\_csv = "grouped\_reviews\_summaries\_with\_sentiment.csv"

df\_summaries.to\_csv(output\_csv, index=False)

# ⏳ Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información clave

print("\n✅ Análisis de sentimiento completado en los resúmenes.")

print(f"📌 Total de juegos analizados: {total\_games}")

print(f"⏱ Tiempo total de ejecución: {execution\_time:.4f} segundos")

print(f"📁 Archivo generado: '{output\_csv}'")

# 🔍 Vista previa del dataset actualizado

print("\n🔹 Vista previa con Sentiment Score:")

print(df\_summaries[['Game', 'Summary\_t5-base', 'Summary\_Sentiment\_Score']].head(10))

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

### Análisis de Sentimiento en las Reseñas Originales

Después de evaluar el sentimiento en los resúmenes generados por juego, el siguiente paso fue calcular el sentimiento para cada reseña individual en su estado original. Este análisis permite obtener una visión más detallada sobre la distribución del sentimiento en los comentarios de los usuarios.

**Proceso de Análisis de Sentimiento**

1. Carga del Dataset de Reseñas Originales
   * Se utilizó el dataset completo con todas las reseñas originales (Content).
   * Se contabilizó el número total de reseñas a procesar.
2. Uso del Modelo VADER
   * Se utilizó el analizador de sentimiento VADER (Valence Aware Dictionary and sEntiment Reasoner) de nltk, optimizado para evaluar sentimiento en texto corto.
   * Se descargó el lexicón de VADER en caso de no estar disponible.
3. Cálculo de la Polaridad de Sentimiento
   * Para cada reseña en la columna Content, se calculó un Sentiment Score en el rango de [-1,1]:
     + Valores cercanos a -1 → Sentimiento negativo
     + Valores cercanos a 0 → Sentimiento neutro
     + Valores cercanos a 1 → Sentimiento positivo
4. Normalización del Puntaje de Sentimiento
   * Para facilitar la interpretación, los valores se transformaron a un rango de 0 a 10:
     + Sentiment Score=(compound score+1)×5
   * Esta conversión permite visualizar mejor las diferencias entre reseñas con evaluación negativa, neutra y positiva.
5. Optimización del Procesamiento
   * Se intentó utilizar swifter para acelerar el cálculo de sentimiento mediante paralelización en CPU/GPU.
   * En caso de no estar disponible swifter, se ejecutó el análisis con pandas.apply().
6. Generación del Dataset Final
   * Se guardó el archivo con los resultados del análisis de sentimiento para cada reseña individual: 📂 df\_final\_with\_sentiment.csv
7. Resultados del Proceso
   * Total de reseñas analizadas: 188,720
   * Tiempo total de ejecución: 93.6080 segundos
   * Archivo final generado: 📂 df\_final\_with\_sentiment.csv

Este análisis permite examinar el sentimiento a nivel de reseña individual, proporcionando información granular sobre la percepción de los usuarios y facilitando análisis más profundos en el comportamiento y preferencias de los jugadores.

import pandas as pd

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import time  # Para medir el tiempo de ejecución

# Iniciar el cronómetro

start\_time = time.time()

# 📌 Descargar el lexicón de VADER solo si no está disponible

try:

    nltk.data.find("sentiment/vader\_lexicon.zip")

except LookupError:

    print("📥 Descargando lexicón de VADER...")

    nltk.download("vader\_lexicon")

# 📌 Cargar el analizador de sentimiento VADER

vader = SentimentIntensityAnalyzer()

# 🚀 Función optimizada para calcular el score de sentimiento con VADER

def calculate\_vader\_score(text):

    if isinstance(text, str):  # Verificar que sea texto

        vader\_score = vader.polarity\_scores(text)['compound']  # Score en rango [-1, 1]

        return vader\_score

    return 0  # Si no hay texto, devolver 0

# 📌 Aplicar la función a la columna 'Content' con swifter para acelerar (si está disponible)

df\_final['Content'] = df\_final['Content'].fillna('')  # Reemplazar NaN para evitar errores

try:

    import swifter

    print("⚡ Usando swifter para análisis de sentimiento...")

    df\_final['Sentiment\_Score'] = df\_final['Content'].swifter.apply(calculate\_vader\_score)

except ImportError:

    print("⏳ Ejecutando análisis de sentimiento sin swifter (puede ser más lento)...")

    df\_final['Sentiment\_Score'] = df\_final['Content'].apply(calculate\_vader\_score)

# 📊 Normalizar el score a un rango de 0 a 10

df\_final['Sentiment\_Score'] = (df\_final['Sentiment\_Score'] + 1) \* 5  # Convierte [-1,1] a [0,10]

# 📌 Guardar el DataFrame con la nueva columna de sentimiento

csv\_filename = "df\_final\_with\_sentiment.csv"

df\_final.to\_csv(csv\_filename, index=False)

# 📊 Contar registros procesados

total\_reviews = len(df\_final)

# 📌 Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información clave del proceso

print(f"\n✅ Sección completada: Se ha agregado el sentimiento al dataset original y guardado como '{csv\_filename}'.")

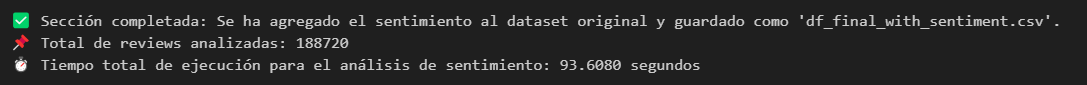
print(f"📌 Total de reviews analizadas: {total\_reviews}")

print(f"⏱ Tiempo total de ejecución para el análisis de sentimiento: {execution\_time:.4f} segundos")

# 📊 Mostrar una vista previa del dataset con la nueva columna añadida

print("\n📌 Vista previa de los datos con Sentiment\_Score:")

print(df\_final[['Game', 'Content', 'Sentiment\_Score']].head(10))



Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

### Cálculo del Promedio de Sentimiento por Juego

Después de calcular el sentimiento para cada reseña individual, se realizó un análisis agregado para obtener el promedio de sentimiento por cada videojuego. Este cálculo permite evaluar la percepción general de los usuarios sobre cada título.

**Proceso de Cálculo del Promedio de Sentimiento**

1. Carga del Dataset con Sentimiento por Reseña
   * Se utilizó el dataset df\_final\_with\_sentiment.csv, que contiene la puntuación de sentimiento (Sentiment\_Score) para cada reseña individual.
   * Se verificó que no existieran valores nulos en la columna de sentimiento y se reemplazaron con 0 en caso necesario.
2. Cálculo del Promedio de Sentimiento
   * Se agrupó la información por videojuego (Game), calculando la media de sentimiento de todas sus reseñas.
   * Se utilizó GPU (CUDA) si estaba disponible, optimizando la ejecución mediante el uso de tensores en PyTorch.
   * Si GPU no estaba disponible, se utilizó pandas y, en caso de estar instalado, swifter para acelerar el cálculo.
3. Normalización y Almacenamiento de Resultados
   * Se generó un nuevo dataset con la estructura:
     + Game: Nombre del videojuego.
     + Average\_Sentiment\_Score: Promedio del sentimiento de todas sus reseñas.
   * Se aseguraron nombres de columnas claros y legibles para facilitar el análisis posterior.
4. Resultados del Proceso
   * Total de juegos procesados: 1151
   * Tiempo total de ejecución: 0.1269 segundos
   * Archivo generado: 📂 average\_sentiment\_per\_game.csv

Este análisis permite obtener una visión global del sentimiento promedio de cada videojuego, proporcionando información útil para estudios de percepción y tendencias en la industria del gaming.

import time  # Para medir el tiempo de ejecución

import pandas as pd

import torch

# 📌 Verificar si CUDA está disponible y usar GPU si es posible

device = "cuda" if torch.cuda.is\_available() else "cpu"

print(f"🚀 Usando dispositivo: {device.upper()}")

# Iniciar el cronómetro

start\_time = time.time()

# 📌 Asegurar que no haya valores NaN en 'Sentiment\_Score'

df\_final['Sentiment\_Score'] = df\_final['Sentiment\_Score'].fillna(0)

# 📊 Contar juegos antes del cálculo

total\_games = df\_final['Game'].nunique()

# Convertir 'Game' a valores numéricos

df\_final['Game\_ID'], game\_labels = pd.factorize(df\_final['Game'])

# 🚀 Calcular el promedio de sentimiento por juego en GPU (si está disponible)

if device == "cuda":

    print("⚡ Ejecutando en GPU...")

    # Convertir a tensores

    game\_ids\_tensor = torch.tensor(df\_final['Game\_ID'].values, dtype=torch.int64, device="cuda")

    sentiment\_tensor = torch.tensor(df\_final['Sentiment\_Score'].values, dtype=torch.float32, device="cuda")

    # Contar ocurrencias de cada juego

    games\_unique = torch.unique(game\_ids\_tensor, return\_inverse=True)[1]

    avg\_sentiment\_gpu = torch.zeros(games\_unique.max() + 1, dtype=torch.float32, device="cuda").scatter\_add\_(0, games\_unique, sentiment\_tensor)

    counts\_gpu = torch.zeros\_like(avg\_sentiment\_gpu, dtype=torch.float32, device="cuda").scatter\_add\_(0, games\_unique, torch.ones\_like(sentiment\_tensor, dtype=torch.float32, device="cuda"))

    avg\_sentiment\_gpu = (avg\_sentiment\_gpu / counts\_gpu).cpu().numpy()

    # Crear DataFrame con resultados

    average\_sentiment\_scores = pd.DataFrame({'Game': game\_labels, 'Average\_Sentiment\_Score': avg\_sentiment\_gpu})

else:

    print("⏳ Ejecutando en CPU (puede ser más lento)...")

    try:

        import swifter

        print("⚡ Usando swifter para acelerar el cálculo del promedio de sentimiento...")

        average\_sentiment\_scores = df\_final.groupby('Game', as\_index=False)['Sentiment\_Score'].swifter.mean()

    except ImportError:

        print("⏳ Ejecutando sin swifter...")

        average\_sentiment\_scores = df\_final.groupby('Game', as\_index=False)['Sentiment\_Score'].mean()

# 📌 Renombrar la columna para mayor claridad

average\_sentiment\_scores.rename(columns={'Sentiment\_Score': 'Average\_Sentiment\_Score'}, inplace=True)

# 📊 Contar juegos después del cálculo

games\_with\_sentiment = len(average\_sentiment\_scores)

# ⏳ Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información clave del proceso

print("\n✅ Sección completada: Se ha calculado el promedio de sentimiento por juego.")

print(f"📌 Total de juegos procesados: {games\_with\_sentiment} de {total\_games}")

print(f"⏱ Tiempo total de ejecución: {execution\_time:.4f} segundos")

# 📊 Mostrar los primeros 10 resultados

print(average\_sentiment\_scores.head(10))

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

### Cálculo del Promedio de Puntuación por Juego (Score de Metacritic)

Además del análisis de sentimiento en las reseñas de los jugadores, es importante considerar el promedio de puntuación (Score) de los juegos, un dato extraído directamente desde Metacritic. Este análisis nos permite comparar la percepción de los usuarios con la evaluación crítica proporcionada en la plataforma.

**Proceso de Cálculo del Promedio de Puntuación**

1. Carga del Dataset con los Scores de Metacritic
   * Se utilizó el dataset df\_final, que contiene la columna Score con las puntuaciones originales extraídas de Metacritic.
   * Se verificó que no existieran valores nulos en la columna Score, reemplazándolos por 0 en caso necesario.
2. Cálculo del Promedio de Puntuación
   * Se agruparon los datos por videojuego (Game) y se calculó la media de puntuación basada en las evaluaciones de la página de Metacritic.
   * Se utilizaron tensores en GPU (CUDA) si estaban disponibles, acelerando la ejecución con PyTorch.
   * Si GPU no estaba disponible, se utilizó pandas y, en caso de estar instalado, swifter para optimizar el cálculo.
3. Normalización y Almacenamiento de Resultados
   * Se generó un nuevo dataset con la estructura:
     + Game: Nombre del videojuego.
     + Average\_Score: Promedio de la puntuación obtenida de Metacritic.
   * Se aseguraron nombres de columnas claros y legibles para facilitar la interpretación de los resultados.
4. Resultados del Proceso
   * Total de juegos procesados: 1151
   * Tiempo total de ejecución: 0.0530 segundos
   * Archivo final generado: 📂 average\_scores\_per\_game.csv

Este análisis proporciona una métrica objetiva para comparar la percepción de los usuarios con la calificación crítica de cada juego, permitiendo estudios detallados sobre la correlación entre ambos valores.

import time  # Para medir el tiempo de ejecución

import pandas as pd

import torch

# 📌 Verificar si CUDA está disponible y usar GPU si es posible

device = "cuda" if torch.cuda.is\_available() else "cpu"

print(f"🚀 Usando dispositivo: {device.upper()}")

# Iniciar el cronómetro

start\_time = time.time()

# 📌 Asegurar que no haya valores NaN en 'Score'

df\_final['Score'] = df\_final['Score'].fillna(0)

# 📊 Contar juegos antes del cálculo

total\_games = df\_final['Game'].nunique()

# 🚀 Calcular el promedio de Score por juego en GPU (si está disponible)

if device == "cuda":

    print("⚡ Ejecutando cálculos en GPU...")

    # Convertir datos a tensores

    game\_indices = {game: idx for idx, game in enumerate(df\_final['Game'].unique())}  # Mapeo de juegos a índices numéricos

    df\_final['Game\_Index'] = df\_final['Game'].map(game\_indices)  # Convertir juegos a índices

    df\_tensor = torch.tensor(df\_final[['Game\_Index', 'Score']].values, dtype=torch.float32, device="cuda")

    # Obtener índices únicos de juegos

    games\_unique = torch.unique(df\_tensor[:, 0], return\_inverse=True)[1]

    # Calcular sumas de Score y conteo de reviews por juego

    sum\_scores\_gpu = torch.zeros(len(game\_indices), dtype=torch.float32, device="cuda").scatter\_add\_(0, games\_unique, df\_tensor[:, 1])

    count\_reviews\_gpu = torch.zeros(len(game\_indices), dtype=torch.float32, device="cuda").scatter\_add\_(0, games\_unique, torch.ones\_like(games\_unique, dtype=torch.float32, device="cuda"))

    # Calcular el promedio de Score por juego

    avg\_score\_gpu = (sum\_scores\_gpu / count\_reviews\_gpu).cpu().numpy()

    # Crear DataFrame con los resultados

    average\_scores = pd.DataFrame({'Game': list(game\_indices.keys()), 'Average\_Score': avg\_score\_gpu})

else:

    print("⏳ Ejecutando cálculos en CPU (puede ser más lento)...")

    try:

        import swifter

        print("⚡ Usando swifter para acelerar el cálculo del promedio de Score...")

        average\_scores = df\_final.groupby('Game', as\_index=False)['Score'].swifter.mean()

    except ImportError:

        print("⏳ Ejecutando sin swifter...")

        average\_scores = df\_final.groupby('Game', as\_index=False)['Score'].mean().reset\_index()

# 📊 Contar juegos después del cálculo

games\_with\_scores = len(average\_scores)

# ⏳ Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información clave del proceso

print("\n✅ Sección completada: Se ha calculado el promedio de Score por juego.")

print(f"📌 Total de juegos procesados: {games\_with\_scores} de {total\_games}")

print(f"⏱ Tiempo total de ejecución: {execution\_time:.4f} segundos")

# 📌 Mostrar los primeros 10 resultados

print("\n📌 Vista previa de los promedios por juego:")

print(average\_scores.head(10))

Texto

El contenido generado por IA puede ser incorrecto.

### Unificación de Datos: Puntuaciones y Sentimientos por Juego

Para obtener una visión completa y unificada de los datos recopilados, realizamos la fusión de tres datasets clave, combinando la información de puntuaciones de Metacritic y el análisis de sentimiento en un único dataframe.

**Proceso de Fusión de los Datasets**

1. Datasets Utilizados
   * average\_scores: Contiene el promedio de puntuaciones (Score) extraídas de Metacritic para cada juego.
   * average\_sentiment\_scores: Contiene el promedio de sentimiento (Sentiment\_Score) calculado a partir de las reseñas individuales de los jugadores.
   * df\_summaries: Contiene el sentimiento asociado al resumen de las reseñas generadas para cada juego (Summary\_Sentiment\_Score).
2. Proceso de Unificación
   * Se verificó la existencia de los datasets antes de proceder con la fusión.
   * Se realizó una unión interna (inner join) entre average\_scores y average\_sentiment\_scores basada en la columna Game, asegurando que solo los juegos presentes en ambos conjuntos de datos fueran considerados.
   * Posteriormente, se agregó la información de df\_summaries, incluyendo el sentimiento del resumen generado para cada juego.
3. Resultados de la Consolidación
   * Total de juegos procesados antes de la fusión:
   * Juegos en average\_scores: 1151
   * Juegos en average\_sentiment\_scores: 1151
   * Juegos en df\_summaries: 1151
   * Total de juegos en el dataset final: 1151 (tras la fusión de los tres conjuntos de datos).
4. Generación del Dataset Final
   * Se guardó el dataset consolidado en un archivo CSV: 📂 merged\_average\_scores.csv
5. Tiempo de Ejecución
   * La unificación de datos se completó en 0.0151 segundos, garantizando un proceso eficiente.

**Vista Previa del Dataset Final**

El dataset final incluye las siguientes columnas clave:

* Game: Nombre del videojuego.
* Average\_Score: Puntuación promedio obtenida de Metacritic.
* Average\_Sentiment\_Score: Promedio del sentimiento basado en las reseñas individuales.
* Summary\_Sentiment\_Score: Sentimiento del resumen generado por juego.

Este dataset consolidado permite realizar un análisis completo de la relación entre la calificación otorgada en Metacritic y la percepción general de los jugadores a través de sus reseñas.

import pandas as pd

import time  # Para medir el tiempo de ejecución

# 📌 Iniciar el cronómetro

start\_time = time.time()

# 📌 Verificar que los datasets existen

if 'average\_scores' not in locals() or 'average\_sentiment\_scores' not in locals() or 'df\_summaries' not in locals():

    raise ValueError("❌ Error: Los datasets 'average\_scores', 'average\_sentiment\_scores' y 'df\_summaries' deben estar definidos antes de ejecutar este código.")

# 📊 Mostrar la cantidad de juegos en cada dataset antes de la fusión

print(f"📊 Juegos en average\_scores: {len(average\_scores)}")

print(f"📊 Juegos en average\_sentiment\_scores: {len(average\_sentiment\_scores)}")

print(f"📊 Juegos en df\_summaries: {len(df\_summaries)}")

# 🔗 Unir 'average\_scores' y 'average\_sentiment\_scores' en base a 'Game'

merged\_df = pd.merge(average\_scores, average\_sentiment\_scores, on="Game", how="inner")

# 🔗 Unir con 'df\_summaries' para agregar el sentimiento del resumen

merged\_df = pd.merge(merged\_df, df\_summaries[['Game', 'Summary\_Sentiment\_Score']], on="Game", how="left")

# 📊 Contar juegos después de la fusión

total\_games\_merged = len(merged\_df)

# 📁 Guardar el resultado en un archivo CSV

csv\_filename = "merged\_average\_scores.csv"

merged\_df.to\_csv(csv\_filename, index=False)

# ⏳ Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

execution\_time = end\_time - start\_time

# 📊 Mostrar información sobre el proceso

print("\n✅ Sección completada: Se han combinado los datasets de puntuaciones y sentimientos.")

print(f"📌 Total de juegos en dataset final: {total\_games\_merged}")

print(f"📁 Archivo guardado como: {csv\_filename}")

print(f"⏱ Tiempo total de ejecución: {execution\_time:.4f} segundos")

# 📊 Mostrar una vista previa del dataset final

print("\n📌 Vista previa de los primeros 10 registros del dataset combinado:")

print(merged\_df.head(10))

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

### Comparación de Puntuaciones y Sentimientos en los Mejores Videojuegos

El propósito de este análisis es comparar la puntuación otorgada por los críticos en Metacritic (Average\_Score) con el sentimiento expresado en las reseñas de los jugadores (Average\_Sentiment\_Score) y los resúmenes generados (Summary\_Sentiment\_Score).

Para visualizar esta relación, se seleccionaron los 10 juegos con mejor puntuación en Metacritic, ordenados de mayor a menor en función de Average\_Score. Luego, se generó un gráfico de barras comparativo que permite identificar si las puntuaciones de los críticos y el sentimiento de los jugadores están alineados o si existen discrepancias significativas.

**Metodología del Análisis**

1. Carga del Dataset
   * Se utilizó el archivo merged\_average\_scores.csv, que consolida la puntuación de Metacritic y los análisis de sentimiento.
   * Se aseguraron valores no nulos en las tres métricas principales (Average\_Score, Average\_Sentiment\_Score, Summary\_Sentiment\_Score).
2. Selección de los 10 Juegos Mejor Puntuados
   * Se ordenaron los juegos en función de su puntuación en Metacritic.
   * Se seleccionaron los 10 con la mayor puntuación para el análisis.
3. Visualización Comparativa con Gráfico de Barras
   * Se generó un gráfico de barras para mostrar las tres métricas en cada juego:
     + Puntuación de Metacritic (barra azul).
     + Sentimiento Promedio en Reseñas de Jugadores (barra roja).
     + Sentimiento Promedio en Resúmenes Generados (barra verde).

**Interpretación del Gráfico**

📊 Casos en los que las puntuaciones y los sentimientos están alineados

* Si las tres barras están a la misma altura, significa que los críticos y los jugadores tienen una percepción similar sobre el juego.

***📊 Casos en los que la puntuación de Metacritic es superior al sentimiento de los jugadores***

* ***Si la barra azul (puntuación de Metacritic) es más alta que las barras roja y verde, indica que los críticos valoraron mejor el juego en comparación con la opinión de los jugadores.\****

***✅ Posibles explicaciones:***

* ***Expectativas no cumplidas por parte de los jugadores.***
* ***Problemas técnicos o bugs que afectan la experiencia del usuario.***
* ***Contenido adicional (DLCs, microtransacciones) que pudo generar una percepción negativa.***

📊 Casos en los que el sentimiento de los jugadores es más alto que la puntuación de Metacritic

* Si las barras de sentimiento (roja y verde) son más altas que la de puntuación, significa que el juego tiene una calificación baja en Metacritic, pero los jugadores lo valoran positivamente.

✅ Posibles explicaciones:

* Juegos de nicho que no fueron bien recibidos por los críticos, pero que tienen una comunidad leal.
* Actualizaciones posteriores al lanzamiento que mejoraron la experiencia del juego.

📊 Casos con discrepancias marcadas

* Si hay una gran diferencia entre las tres métricas, es posible que el juego haya generado opiniones polarizadas.

✅ Posibles explicaciones:

* Cambios en la jugabilidad o la historia que dividieron a la comunidad.
* Controversias o estrategias comerciales que afectaron la percepción del público.

**Conclusiones del Análisis**

📌 Si las puntuaciones de Metacritic y los sentimientos de los jugadores están alineados

* Se confirma que la percepción crítica y la experiencia de los jugadores coinciden en términos de calidad.

***📌 Si las puntuaciones y los sentimientos presentan discrepancias***

* ***Es posible que factores externos, como expectativas, problemas técnicos o estrategias comerciales, hayan influido en la percepción de los jugadores.***

📌 Si los sentimientos de los jugadores son más positivos que la puntuación de Metacritic

* Se evidencia que algunos juegos pueden haber sido subestimados por la crítica pero apreciados por la comunidad.

**Tiempo de Ejecución**

* El análisis completo tomó aproximadamente 0.1389 segundos, asegurando un procesamiento eficiente de los datos.

Este análisis proporciona una visión clara de cómo los juegos mejor evaluados por la crítica son percibidos por los jugadores, permitiendo detectar tendencias y posibles factores que influyen en la percepción de los videojuegos.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import time  # Para medir el tiempo de ejecución

# 📌 Iniciar el cronómetro

start\_time = time.time()

# 📌 Cargar el dataset

average\_scores = pd.read\_csv("merged\_average\_scores.csv")

# 📌 Asegurar que no haya valores NaN en los scores

average\_scores['Average\_Score'] = average\_scores['Average\_Score'].fillna(0)

average\_scores['Average\_Sentiment\_Score'] = average\_scores['Average\_Sentiment\_Score'].fillna(0)

average\_scores['Summary\_Sentiment\_Score'] = average\_scores['Summary\_Sentiment\_Score'].fillna(0)

# 📊 Mostrar estadísticas básicas

total\_games = len(average\_scores)

print(f"📌 Total de juegos analizados: {total\_games}")

# 🎨 Configuración de visualización

sns.set\_theme(style="whitegrid")

# 📌 \*\*Gráfico de Barras Comparativo\*\*

plt.figure(figsize=(12, 6))

average\_scores\_sorted = average\_scores.sort\_values(by="Average\_Score", ascending=False).head(10)  # Top 10 juegos mejor puntuados

ax = average\_scores\_sorted.plot(

    x="Game",

    y=["Average\_Score", "Average\_Sentiment\_Score", "Summary\_Sentiment\_Score"],

    kind="bar", figsize=(12, 6),

    color=["blue", "red", "green"]

)

plt.xlabel("Juegos")

plt.ylabel("Valor")

plt.title("📊 Comparación de Puntuaciones y Sentimientos (Top 10 Juegos)")

plt.xticks(rotation=45, ha="right")

plt.legend(["Puntuación", "Sentimiento Reseñas", "Sentimiento Resúmenes"])

plt.show()

# 📢 \*\*Conclusión del Gráfico\*\*

print("\n📢 Conclusión del Gráfico:")

print("- Si las tres barras están alineadas, significa que la puntuación del juego y los sentimientos en reseñas y resúmenes son similares.")

print("- Si la barra azul (puntuación) es más alta que las rojas y verdes, indica que el juego tiene una buena calificación, pero los jugadores expresan sentimientos más negativos en las reseñas o resúmenes.")

print("- Si las barras de sentimiento son más altas que la puntuación, sugiere que, aunque el juego tiene una puntuación baja, los jugadores lo describen con un sentimiento más positivo.")

print("- Juegos con diferencias marcadas entre puntuaciones y sentimientos pueden ser controversiales, polarizando la opinión de los jugadores.")

# ⏳ Detener el cronómetro y calcular el tiempo de ejecución

end\_time = time.time()

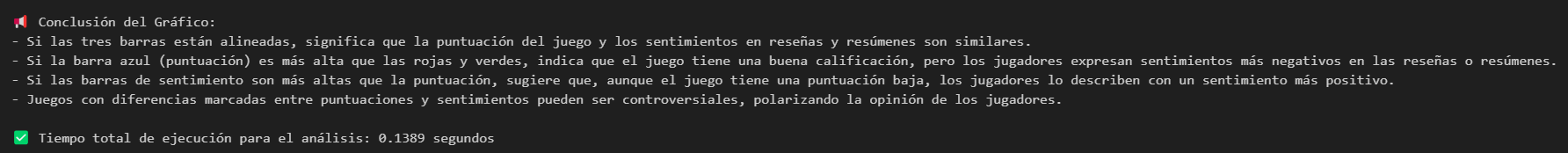
execution\_time = end\_time - start\_time

print(f"\n✅ Tiempo total de ejecución para el análisis: {execution\_time:.4f} segundos")



Gráfico, Gráfico en cascada

El contenido generado por IA puede ser incorrecto.



### Ranking de Juegos según Sentimiento Promedio

Para evaluar cuáles videojuegos han sido mejor percibidos por los jugadores, se ordenaron los juegos en función de su sentimiento promedio en las reseñas individuales (Average\_Sentiment\_Score). Esto permite identificar cuáles títulos han generado una respuesta más positiva dentro de la comunidad de jugadores.

**Proceso de Análisis**

1. Se ordenaron los videojuegos de mayor a menor según el sentimiento promedio en las reseñas.
2. Se generó un archivo sorted\_games\_by\_sentiment.csv con la lista completa de juegos ordenados según su percepción positiva.
3. Se mostraron los 10 juegos con mejor Sentimiento Promedio, permitiendo analizar cuáles títulos han sido los más valorados por los jugadores en términos de emociones expresadas en sus reseñas.

**Interpretación de Resultados**

* Los juegos en las primeras posiciones del ranking son aquellos que han recibido las reseñas más positivas en términos de emoción y percepción general.
* Un sentimiento promedio alto indica que la mayoría de las reseñas expresaron una experiencia positiva con el juego, destacando aspectos como jugabilidad, historia, mecánicas y satisfacción general.
* Este análisis es útil para detectar títulos altamente recomendados por la comunidad, independientemente de su puntuación en Metacritic.

Los resultados obtenidos pueden ser de gran valor para entender la percepción de los jugadores y comparar si estos juegos también tienen una alta puntuación crítica o si existen diferencias entre la evaluación de los críticos y la experiencia de los usuarios.

# Ordenar juegos según Sentimiento Promedio

sorted\_games = average\_scores.sort\_values(by='Average\_Sentiment\_Score', ascending=False)

# Guardar los resultados

sorted\_csv\_filename = "sorted\_games\_by\_sentiment.csv"

sorted\_games.to\_csv(sorted\_csv\_filename, index=False)

# Mostrar los primeros 10 juegos con mejor sentimiento promedio

print("\n📌 Top 10 juegos con mejor Sentimiento Promedio:")

print(sorted\_games[['Game', 'Average\_Sentiment\_Score']].head(10))

Texto

El contenido generado por IA puede ser incorrecto.

## Conclusión Final

Este análisis nos permite comprender mejor la relación entre la evaluación crítica y la percepción de los jugadores. En casos donde hay discrepancias significativas, podemos inferir que la experiencia de juego puede diferir de la expectativa generada por los críticos, lo que puede deberse a múltiples factores, como bugs, cambios en la jugabilidad, o la subjetividad de la comunidad de jugadores.

Los resultados obtenidos pueden ser útiles para desarrolladores y analistas, permitiéndoles comprender mejor cómo sus juegos son percibidos por el público y qué factores pueden influir en su recepción.