Estoy ejecutando el siguiente código y genera un error:

dfsteam\_games = dfsteam\_games.drop(['publisher', 'app\_name', 'url', 'tags', 'reviews\_url', 'specs', 'price', 'early\_access', 'developer'], axis=1)

KeyError: "['publisher', 'app\_name', 'url', 'tags', 'reviews\_url', 'specs', 'price', 'early\_access', 'developer'] not found in axis"

git config --global user.email "jaimeajl@hotmail.com"

git config --global user.name "jaimeajl"

cd C:\Users\jaime\Desktop\Henry\Proyectopersonal1/apiproject2

uvicorn nombre\_del\_archivo:app --host 0.0.0.0 --port $PORT

cd C:\Users\jaime\Desktop\Henry\Proyectopersonal1

git add .

git commit -m "Descripción de los cambios"

git push origin main

http://localhost:8000/playtime\_genre?genre=Action

dfsteam\_games\_cleaned:('C:\Users\jaime\Desktop\Henry\Proyectopersonal1\dfsteam\_games\_cleaned.csv', index=False)

dfusers\_items: ('C:\Users\jaime\Desktop\Henry\Proyectopersonal1\dfusers\_items.csv', index=False)

df\_user\_reviews:

('C:\Users\jaime\Desktop\Henry\Proyectopersonal1\user\_reviews.csv', index=False)

Análisis exploratorio de los datos: (Exploratory Data Analysis-EDA)

Que tipo de análisis se podría hacer y cual es la recomendación para un conjunto de datos de uso de videojuegos

Dame tu concepto para investigar las relaciones que hay entre las variables del dataset, ver si hay outliers o anomalías y ver si hay algún patrón que valga la pena explorar en un análisis posterior.

Las nubes de palabras dan una buena idea de cuáles palabras son más frecuentes en los títulos, ¡podría ayudar al sistema de predicción!

No se puede usar librerías para hacer EDA automático

Dame el código en python para generar y entrenar un modelo de machine learning para crear un sistema de recomendación que tenga una relación ítem-ítem, esto es se toma un item, en base a que tan similar esa ese ítem al resto, se recomiendan similares. Aquí el input es un juego y el output es una lista de juegos recomendados

El join se debe hacer con la columna: id

Origen de datos e información a utilizar:

Dataframe:

dfsteam\_games\_cleaned = pd.read\_csv('dfsteam\_games\_cleaned.csv')

Columnas:

genres: Genero del video juego

title: Que contiene el titulo del videojuego

Id: identificador unico de contenido

Dataframe:

dfusers\_items = pd.read\_csv('dfusers\_items.csv')

Columnas:

playtime\_forever: tiempo de uso del juego

id: identificador unico de contenido

Dame el código en python para generar y entrenar un modelo de machine learning para crear un sistema de recomendación que tenga una relación ítem-ítem, esto es se toma un item, en base a que tan similar esa ese ítem al resto, se recomiendan similares. Aquí el input es un juego y el output es una lista de juegos recomendados

El join se debe hacer con la columna: id

Origen de datos e información a utilizar:

Dataframe:

dfsteam\_games\_cleaned = pd.read\_csv('dfsteam\_games\_cleaned.csv')

Columnas:

Title: Que contiene el titulo del videojuego

Id: identificador unico de contenido

Dataframe:

dfusers\_items = pd.read\_csv('dfusers\_items.csv')

Columnas:

playtime\_forever: tiempo de uso del juego

Id: identificador unico de contenido

# Histograma de la columna "playtime\_forever"

plt.figure(figsize=(10, 6))

sns.histplot(data=dfusers\_items, x='playtime\_forever', bins=50, kde=True)

plt.title('Distribución de Tiempo de Juego')

plt.xlabel('Tiempo de Juego')

plt.ylabel('Frecuencia')

plt.show()

Interfaz de usuario gráfica

Descripción generada automáticamente

# Boxplot para identificar outliers

plt.figure(figsize=(8, 5))

sns.boxplot(x=dfusers\_items['playtime\_forever'])

plt.title('Boxplot de Tiempo de Juego')

plt.xlabel('Tiempo de Juego')

plt.show()

Texto

Descripción generada automáticamente con confianza media

Dataframe:

dfsteam\_games\_cleaned = pd.read\_csv('dfsteam\_games\_cleaned.csv')

Columnas:

Title: Que contiene el titulo del videojuego

Id: identificador unico de contenido

Dame el código para averiguarl el tipo de dato en la columna: Title del dataframe: dfsteam\_games\_cleaned

NameError: name 'sns' is not defined

>>> plt.title('Boxplot de Tiempo de Juego')

Text(0.5, 1.0, 'Boxplot de Tiempo de Juego')

>>> plt.xlabel('Tiempo de Juego')

Text(0.5, 0, 'Tiempo de Juego')

>>> plt.show()

>>> playtime\_stats = dfusers\_items['playtime\_forever'].describe()

>>> print(playtime\_stats)

count 88310.000000

mean 2915.249236

std 10927.103790

min 0.000000

25% 0.000000

50% 132.000000

75% 1282.000000

max 388621.000000

Name: playtime\_forever, dtype: float64

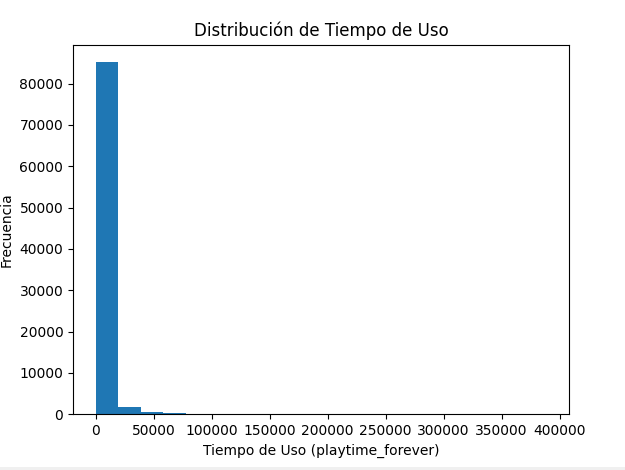
plt.hist(dfusers\_items['playtime\_forever'], bins=20)

plt.xlabel('Tiempo de Uso (playtime\_forever)')

plt.ylabel('Frecuencia')

plt.title('Distribución de Tiempo de Uso')

plt.show()



plt.boxplot(dfusers\_items['playtime\_forever'])

plt.ylabel('Tiempo de Uso (playtime\_forever)')

plt.title('Boxplot de Tiempo de Uso')

plt.show()

Imagen que contiene Gráfico de cajas y bigotes

Descripción generada automáticamente

Q1 = dfusers\_items['playtime\_forever'].quantile(0.25)

Q3 = dfusers\_items['playtime\_forever'].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = dfusers\_items[(dfusers\_items['playtime\_forever'] < lower\_bound) | (dfusers\_items['playtime\_forever'] > upper\_bound)]

print("Número de outliers:", len(outliers))

> print("Número de outliers:", len(outliers))

Número de outliers: 13465

from wordcloud import WordCloud

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(' '.join(dfsteam\_games\_cleaned['title']))

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.show()



Análisis exploratorio de los datos: (Exploratory Data Analysis-EDA)

Que tipo de análisis se podría hacer y cual es la recomendación para un conjunto de datos de uso de videojuegos

Dame tu concepto para investigar las relaciones que hay entre las variables del dataset, ver si hay outliers o anomalías y ver si hay algún patrón que valga la pena explorar en un análisis posterior.

Las nubes de palabras dan una buena idea de cuáles palabras son más frecuentes en los títulos, ¡podría ayudar al sistema de predicción!

No se puede usar librerías para hacer EDA automático

Dame el código en python para generar y entrenar un modelo de machine learning para crear una función def recomendacion\_usuario( id de usuario ): con la cual Ingresando el user\_id, deberíamos recibir una lista con 5 juegos recomendados para dicho usuario.

El join se debe hacer con la columna: id

Origen de datos e información a utilizar:

Dataframe:

dfsteam\_games\_cleaned = pd.read\_csv('dfsteam\_games\_cleaned.csv')

Columnas:

title: Que contiene el titulo del videojuego

Id: identificador unico de contenido

Dataframe:

dfusers\_items = pd.read\_csv('dfusers\_items.csv')

Columnas:

user\_id: identificador unico de usuario

playtime\_forever: tiempo de uso del juego

id: identificador unico de contenido

Dame el código en python para generar y entrenar un modelo de machine learning para crear un sistema de recomendación que tenga una relación ítem-ítem, esto es se toma un item, en base a que tan similar es ese ítem al resto, se recomiendan similares y se debe aplicar la similitud del coseno

La función a crear es def recomendacion\_usuario( id de usuario ): con la cual Ingresando el id: 273350, deberíamos recibir una lista con 5 juegos recomendados para dicho usuario obtenidos de la columna “title” del dataframe dfsteam\_games\_cleaned

La función creada que se consumirá en la API, deben tene un decorador por cada una (@app.get(‘/’)) con el código para ejecutar la api:

Ejemplo:

# Ejecutar la aplicación FastAPI

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

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)

El join se debe hacer con la columna: id

Origen de datos e información a utilizar:

Dataframe:

dfsteam\_games\_cleaned = pd.read\_csv('dfsteam\_games\_cleaned.csv')

Columnas:

title: Que contiene el titulo del videojuego

id: identificador unico de contenido

Dataframe:

dfusers\_items = pd.read\_csv('dfusers\_items.csv')

Columnas:

playtime\_forever: tiempo de uso del juego

id: identificador unico de contenido

Ajusta el código para cumplir lo siguiente:

Desarrollo API: Disponibilizar usando el framework FastAPI el cual se compartira a través de render

La función creada que se consumirá en la API, deben tene un decorador por cada una (@app.get(‘/’)).

http://localhost:8000/user\_id= 273350

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CODIGO FINAL FUNCIONAL

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from fastapi import FastAPI

# Cargar los datos

dfsteam\_games\_cleaned = pd.read\_csv('dfsteam\_games\_cleaned.csv')

dfusers\_items = pd.read\_csv('dfusers\_items.csv')

# Calcular la matriz TF-IDF para la columna 'title' en dfsteam\_games\_cleaned

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english')

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(dfsteam\_games\_cleaned['title'])

# Calcular la similitud del coseno entre los ítems basado en los títulos

cosine\_sim\_titles = cosine\_similarity(tfidf\_matrix, tfidf\_matrix)

# Inicializar FastAPI

app = FastAPI()

# Definir la función de recomendación de usuario

def recomendacion\_usuario(user\_id):

    # Filtrar los juegos que el usuario ha jugado y obtener sus IDs

    user\_items = dfusers\_items[dfusers\_items['id'] == user\_id]['id'].tolist()

    # Crear un diccionario para almacenar la puntuación total de similitud de juegos

    game\_scores = {}

    # Recorrer los juegos jugados por el usuario

    for game\_id in user\_items:

        game\_idx = dfsteam\_games\_cleaned[dfsteam\_games\_cleaned['id'] == game\_id].index[0]

        sim\_scores = list(enumerate(cosine\_sim\_titles[game\_idx]))

        # Ordenar por similitud y obtener los 5 juegos más similares

        sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)[1:6]

        # Calcular la puntuación total de similitud para recomendación

        for sim\_game in sim\_scores:

            if sim\_game[0] not in game\_scores:

                game\_scores[sim\_game[0]] = sim\_game[1]

            else:

                game\_scores[sim\_game[0]] += sim\_game[1]

    # Ordenar los juegos recomendados por puntuación de similitud

    recommended\_games = sorted(game\_scores.items(), key=lambda x: x[1], reverse=True)

    # Obtener los títulos de los juegos recomendados

    recommended\_titles = [dfsteam\_games\_cleaned.iloc[i[0]]['title'] for i in recommended\_games]

    return recommended\_titles

# Definir la ruta de la API

@app.get("/")

async def get\_recommendations(user\_id: int):

    recommended\_games = recomendacion\_usuario(user\_id)

    return {"user\_id": user\_id, "recommended\_games": recommended\_games}

# Ejecutar la aplicación FastAPI

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

    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=8000)