ENTREGA 2c: INTRODUCCIÓN A ANALISIS DE DATOS CON PANDAS

Contenido

[INTRODUCCIÓN 2](#_Toc181548760)

[¿QUÉ ES EL ANÁLISIS DE DATOS? 2](#_Toc181548761)

[Herramientas necesarios 3](#_Toc181548762)

[ELECCIÓN DEL DATASET 4](#_Toc181548763)

[ESTUDIO DEL DATASET 4](#_Toc181548764)

[IMPORTAR DATASET 5](#_Toc181548765)

[LIMPIAR DATOS DEL DATASET 6](#_Toc181548766)

[CREAR ENUMERADOR PARA AUTORES 7](#_Toc181548767)

[GUARDAR Y EXPORTAR DATASET TRATADOS 8](#_Toc181548768)

[ENSUCIAR EL DATASET 9](#_Toc181548769)

[ORDENAR DATASET 10](#_Toc181548770)

[FLUJO DEL TRABAJO (MAIN DEL ARCHIVO) 11](#_Toc181548771)

[DOCUMENTOS UTILIZADOS Y QUE SE VAN A CREAR 13](#_Toc181548772)

[ARCHIVOS PROPORCIONADOS 14](#_Toc181548773)

[SALIDA DEL FLUJO PRINCIPAL DE TRABAJO (MAIN) 14](#_Toc181548774)

# INTRODUCCIÓN

Este documento tiene como objetivo recabar la información recopilada por el estudiante Sergio Martín Ledesma y Daniel Mocinha Sanchez para introducirse al análisis de datos con la librería de Pandas con el lenguaje de programación Python.

En esta práctica se realizarán y aprenderán las siguientes cuestiones:

1. Maneras de importar y exportar archivos comunes en al análisis de datos, como .csv, hojas de cálculo o .json.
2. Tratamiento de datos, utilizando “Dataframes”, una clase que nos ofrece la librería de Pandas, útiles para tratamiento de datos.
3. Limpieza del dataset
   1. Valores nulos
   2. Duplicados
4. Tratamiento de errores en las lecturas e importación de un dataset.
5. Cómo ensuciar nuestro dataset.
6. Normalizar datos
7. Ordenar nuestro dataset
8. Exportar el dataset tratado a diferentes archivos.

Debo explicar que el apartado de “Cómo ensuciar nuestro dataset” está establecido para podernos poner en situación sobre una práctica habitual en el entorno del Análisis de datos, debido a que se trata de una práctica realizada en un entorno controlado.

Como toda introducción, se debe realizar un pequeña explicación al campo en el que nos estamos centrando: el análisis de datos.

## ¿QUÉ ES EL ANÁLISIS DE DATOS?

El análisis de datos es el proceso de examinar, limpiar, transformar e interpretar datos para extraer información significativa y tomar decisiones informadas. Este proceso incluye diversas técnicas, como la recopilación de datos, la limpieza para eliminar inconsistencias y la aplicación de métodos estadísticos y algoritmos que ayudan a descubrir patrones, tendencias y relaciones.

**Finalidad del Análisis de Datos**

El análisis de datos tiene como finalidad convertir grandes volúmenes de datos en información útil. Esto permite a las organizaciones y a los individuos:

1. **Tomar decisiones informadas**: A través del análisis, se pueden identificar patrones que ayuden a realizar previsiones y mejorar la toma de decisiones. Por ejemplo, en marketing, el análisis de datos permite comprender el comportamiento de los clientes y personalizar estrategias para atraerlos de manera más efectiva.
2. **Optimizar procesos**: En la industria y la gestión empresarial, el análisis ayuda a identificar áreas de mejora y reducir ineficiencias. Mediante el análisis de datos de producción, por ejemplo, una empresa puede identificar cuellos de botella y optimizar sus procesos.
3. **Identificar oportunidades y riesgos**: El análisis predictivo y la minería de datos permiten anticipar oportunidades de mercado y detectar riesgos potenciales. Esto es útil en áreas como las finanzas, donde el análisis puede ayudar a predecir tendencias de inversión o riesgos de crédito.
4. **Comprender mejor los fenómenos**: En ciencias sociales, medicina o climatología, el análisis de datos ayuda a comprender fenómenos complejos, desde patrones de enfermedades hasta cambios en el clima global, y a desarrollar teorías y modelos predictivos.

En resumen, el análisis de datos se utiliza para transformar datos en información valiosa que guíe decisiones estratégicas, optimice operaciones, identifique riesgos y oportunidades, y mejore la comprensión de fenómenos complejos.

## Herramientas necesarios

Para poder realizar esta práctica en condiciones, será necesario configurar nuestro entorno con el software necesario. En este caso, se necesaritará los siguientes requisitos:

1. Tener instalado en nuestro sistema operativo, el compilador de Python. Es recomendable tenerlo actualizado al último o la versión más estable del momento.
2. La librería “Pandas”: Para ello se debe realizar el siguiente comando en vuestra consola: Pip install Pandas

Texto

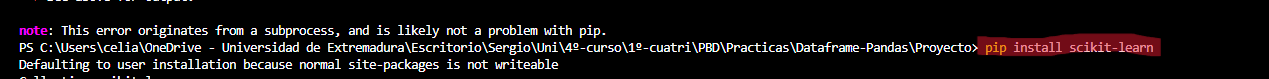
Descripción generada automáticamente

1. La librería “csv”: Esta librería es necesaria, para poder leer, crear y modificar archivos csv. Además, es la librería que utiliza “Pandas” a la hora de realizar las lecturas y escrituras sobre este tipo de archivos.
2. La librería “openpyxl”: Esta librería es necesaria, para poder leer, crear y modificar archivos de tipo Hojas de cálculo. Además, es la librería que utiliza “Pandas” a la hora de realizar las lecturas y escrituras sobre este tipo de archivos.

Captura de pantalla de computadora

Descripción generada automáticamente

1. La librería scikit-learn: Scikit-Learn ofrece herramientas para limpiar y preparar los datos antes de crear modelos. Nosotros la utilizaremos únicamente para poder normalizar los datos escogidos de nuestro dataset.



## ELECCIÓN DEL DATASET

# IMPORTAR DATASET

La función cargar\_datos carga el dataset desde un archivo CSV, permitiendo manejar errores en caso de rutas incorrectas o problemas en el archivo:

* **df = pd.read\_csv(ruta\_csv, on\_bad\_lines='skip'):** Se trata de la función que tiene la librería “Pandas” para leer archivos csv.
* **Validación de ruta**: Si el archivo no se encuentra, se muestra un mensaje de error.
* **Líneas incorrectas**: Con el parámetro on\_bad\_lines='skip', se omiten las filas que tienen errores de formato.

En el caso de que no se encuentre dicho documento pasado por parámetros, se lanzará un error.

**def** **cargar\_datos**(ruta\_csv):  
 """  
 Carga un archivo CSV en un DataFrame de pandas.  
  
 :param ruta\_csv: Ruta al archivo CSV.  
 :return: DataFrame de pandas.  
 """  
 **try**:  
 df = pd.read\_csv(ruta\_csv, on\_bad\_lines='skip')  
   
 print("Datos cargados exitosamente.")  
 **return** df  
 **except** FileNotFoundError:  
 print("Archivo no encontrado. Verifica la ruta.")  
 **return** None

# LIMPIAR DATOS DEL DATASET

La función limpiar\_datos se asegura de eliminar datos duplicados y manejar valores nulos de forma apropiada:

* **Eliminación de duplicados**: Se eliminan registros duplicados. La función “drop\_duplicated” nos viene muy bien en estos casos.

df = df.drop\_duplicates()

* **Tratamiento de valores nulos**: Luego, en nuestro “for”, se iteran cada columna para poder tratar los valores nulos de nuestro dataset. Dependiendo del caso, se rellenará con la moda o la media.
  + Si una columna es categórica, se rellena con la **moda**.
  + Si es numérica, se rellena con la **media** de la columna.

**for** column **in** df.columns:  
 **if** df[column].isnull().sum() > 0:  
 **if** df[column].dtype == "object":  
 df[column].fillna(df[column].mode()[0], inplace=True) # Rellenar con la moda  
 **else**:  
 df[column].fillna(df[column].mean(), inplace=True) # Rellenar con la media

Finalmente, se muestra por pantalla que los datos nulos y duplicados han sido eliminados. La función finalmente quedaría de la siguiente forma:

# Función para limpiar datos nulos y duplicados  
**def** **limpiar\_datos**(df):  
 """  
 Limpia datos nulos y duplicados en el DataFrame.  
  
 :param df: DataFrame a limpiar.  
 :return: DataFrame limpio.  
 """  
 # Eliminar duplicados  
 df = df.drop\_duplicates()  
   
 # Rellenar valores nulos con métodos adecuados (media, moda, etc.)  
 **for** column **in** df.columns:  
 **if** df[column].isnull().sum() > 0:  
 **if** df[column].dtype == "object":  
 df[column].fillna(df[column].mode()[0], inplace=True) # Rellenar con la moda  
 **else**:  
 df[column].fillna(df[column].mean(), inplace=True) # Rellenar con la media  
 print("Datos nulos y duplicados eliminados.")  
  
 **return** df

# CREAR ENUMERADOR PARA AUTORES

La función procesar\_authors permite convertir la columna authors en una lista de identificadores únicos de autores. Para múltiples autores en un mismo registro, se divide la cadena y se reemplaza por un enumerador único.

Una vez procesada, se creará un nuevo fichero en el que se guardarán dichos enumeradores, para que en un futuro, se pueda realizar búsquedas indexadas.

Se pondrá especial atención en la llamada a .dropna(). Una función de la librería “Pandas” que nos permite ignorar los valores nulos que se encuentran en un dataframe o iterador.

Por cada iteración, se divide el String de cada instancia de la columna “authors” por el carácter “/”, ya que, según el autor del dataset, se utiliza dicho carácter para poder dividir los autores, cuando los libros estén creados por más de un autor.

lista\_autores = []  
 **for** df\_autores **in** df['authors'].dropna(): # Ignorar valores nulos  
   
 lista\_autores.extend(df\_autores.split('/')) # Dividir y añadir cada autor

Por cada autor, se guarda en la lista “lista\_autores”. Una vez finalizada el “for”, se pasa dicha lista a un conjunto (set) para poder eliminar los duplicados de una forma sencilla, aplicando las propiedades de los conjuntos.

# Obtener autores únicos  
 lista\_autores\_unicos = list(set(lista\_autores))

Para finalizar, se establece el formato del enumerador, siendo la primera columna para el identifcador y la segunda para el nombre.

# Crear el enumerador de autores (diccionario)  
 enumerador\_autores = {autor: idx **for** idx, autor **in** enumerate(lista\_autores\_unicos)}  
   
   
 # Reemplazar en la columna 'authors' los nombres por sus respectivos IDs  
 df['authors'] = df['authors'].apply(**lambda** x: [enumerador\_autores[autor] **for** autor **in** x.split('/')] **if** pd.notna(x) **else** [])

# GUARDAR Y EXPORTAR DATASET TRATADOS

La función guardar\_datos exporta el DataFrame limpio y procesado en los formatos Excel y JSON. Para ello, se utiliza las funciones “to\_excel” y “to\_json”, respectivamente. Estas se tratan de funciones específicas de la librería de “Pandas”, dándonos una facilidad para poder exportar nuestros datasets a distintos formatos.

Para las funciones “to\_excel”, se debe especificar la ruta y/o nombre del documento con una extensión de tipo hoja de cálculo. Lo mismo con la función “to\_json”.

Sino, te saltará una excepción de que el formato destino no es correcto.

# Función para guardar el DataFrame en distintos formatos  
**def** **guardar\_datos**(df, ruta\_excel, ruta\_json):  
 """  
 Guarda el DataFrame en archivos Excel y JSON.  
  
 :param df: DataFrame a guardar.  
 :param ruta\_excel: Ruta para guardar el archivo Excel.  
 :param ruta\_json: Ruta para guardar el archivo JSON.  
 """  
 # Guardar en Excel  
 df.to\_excel(ruta\_excel, index=False)  
 print(f"Datos guardados en formato Excel en {ruta\_excel}.")  
  
 # Guardar en JSON  
 df.to\_json(ruta\_json, orient="records", lines=True)  
 print(f"Datos guardados en formato JSON en {ruta\_json}.")

La opción de “index=False” se establece con el objetivo de no establecer una columna con los índices que les corresponde en el dataset en el que estamos tratando. Si se pone a “True”, se guardará y creará en la primera columna, su índice correspondiente en el dataset.

# ENSUCIAR EL DATASET

Con el objetivo de ponenos en la piel de un Analista de Datos, vamos a ponernos en situación de que nos encontramos ante un dataset no controlado, proporcionando campos con tipos erróneos, campos nulos o sin rellenar. Para ello, hemos creado una función, llamada “ensuciar\_dataset”.

**def** **ensuciar\_dataset**(df, prob\_nulos=0.1, prob\_duplicados=0.05, prob\_tipografia=0.05, prob\_formato=0.1):  
 """  
 Ensucia un dataset con valores nulos, duplicados, errores tipográficos y variaciones de formato.  
  
 :param df: DataFrame original a ensuciar.  
 :param prob\_nulos: Probabilidad de insertar valores nulos en cada columna.  
 :param prob\_duplicados: Probabilidad de insertar filas duplicadas en el dataset.  
 :param prob\_tipografia: Probabilidad de agregar errores tipográficos en valores de texto.  
 :param prob\_formato: Probabilidad de aplicar variaciones de formato en ciertas columnas.  
 :return: DataFrame con datos ensuciados.  
 """  
 # 1. Introducir valores nulos  
 **for** col **in** df.columns:  
 df.loc[df.sample(frac=prob\_nulos).index, col] = np.nan  
  
 # 2. Introducir duplicados  
 num\_duplicados = int(len(df) \* prob\_duplicados)  
 duplicados = df.sample(n=num\_duplicados)  
 df = pd.concat([df, duplicados], ignore\_index=True)  
  
 # 3. Errores tipográficos en columnas de texto  
 **def** **insertar\_error\_tipografico**(texto):  
 **if** pd.isnull(texto) **or** len(texto) < 2:  
 **return** texto # Dejar el valor si está vacío o tiene menos de 2 caracteres  
 index = random.randint(0, len(texto) - 1)  
 # Intercambiar dos letras adyacentes para simular un error tipográfico  
 texto\_list = list(texto)  
 texto\_list[index], texto\_list[index - 1] = texto\_list[index - 1], texto\_list[index]  
 **return** ''.join(texto\_list)  
   
 columnas\_texto = ['title', 'authors', 'publisher']  
 **for** col **in** columnas\_texto:  
 df[col] = df[col].apply(**lambda** x: insertar\_error\_tipografico(x) **if** random.random() < prob\_tipografia **else** x)  
   
 # 4. Variaciones en el formato de columnas numéricas o de fecha  
 # Para "average\_rating", introducir algunos valores en strings  
 df['average\_rating'] = df['average\_rating'].apply(**lambda** x: str(x) **if** random.random() < prob\_formato **else** x)  
   
 # Para "publication\_date", cambiar el formato en algunas filas  
 **def** **cambiar\_formato\_fecha**(fecha):  
 **if** pd.isnull(fecha):  
 **return** fecha  
 **try**:  
 # Convertir la fecha a diferentes formatos de cadena  
 fecha = pd.to\_datetime(fecha)  
 formatos = ['%Y-%m-%d', '%d/%m/%Y', '%m-%d-%Y']  
 **return** fecha.strftime(random.choice(formatos))  
 **except**:  
 **return** fecha # Dejar sin cambio si no es una fecha válida  
  
 df['publication\_date'] = df['publication\_date'].apply(**lambda** x: cambiar\_formato\_fecha(x) **if** random.random() < prob\_formato **else** x)  
   
 # Para "ratings\_count" y "text\_reviews\_count", agregar variación de formato  
 **for** col **in** ['ratings\_count', 'text\_reviews\_count']:  
 df[col] = df[col].apply(**lambda** x: str(x) **if** random.random() < prob\_formato **else** x)  
   
 print("Dataset ensuciado con éxito.")  
 **return** df

# ORDENAR DATASET

Para ordenar el dataset, se utiliza la función que nos proporciona “Pandas”, “sort\_values()”. Esta función acepta distinto parámetros, pero los más importantes, y los que nosotros utilizamos son:

* **By:** En este parámetro, hay que establecer el nombre de la columna que se quiere utilizar para ordenar el dataset. En nuestro caso, se utiliza la columna de rating. También acepta una lista de String como parámetros, para así, utilizar distintas columnas para realizar el ordenamiento.
* **Ascending:** Este parámetro configura si se desea que el orden sea ascendente o no. Si está a True, te lo ordenará de forma ascendente. Sino, de forma descendente. En nuestro caso, queremos que en las primeras posiciones se encuentren los libros mejores valorados. Por eso, establecemos este parámetro a “False”.

df = df.sort\_values(by='average\_rating', ascending=False)

# FLUJO DEL TRABAJO (MAIN DEL ARCHIVO)

En el main, a la hora de trabajar con datasets, siempre se suele seguir la misma estructura:

1. Establecer un parámetro para la ruta del fichero/dataset

ruta\_csv = "dataset\_ensuciado.csv"

1. Llamar a la función de lectura del fichero elegido en el parámetro del apartado anterior

# Cargar los datos  
 df = cargar\_datos(ruta\_csv)

1. Condicional de si se ha cargado el fichero o no
   1. Entra solamente al bloque condicional si se ha cargado correctamente el fichero.

**if** df **is** **not** None:

* 1. Se trabaja con dicho dataset cargado. En nuestro caso, se guarda en la variable “df”.
     1. Limpiar dataset

# Limpiar datos  
 df = limpiar\_datos(df)

* + 1. Ordenar valores

#ORDENAR LIBROS POR SU RATING  
 df = df.sort\_values(by='average\_rating', ascending=False)

print(f"Primeras filas ordenados de mayor a menor por rating {df.head(10)[['title','average\_rating']]}")  
 print(f"ÚLtimas filas ordenados de mayor a menor por rating {df.tail(10)[['title','average\_rating']]}")

* + 1. Tratar y procesar datos y columnas deseadas

#Se procesan los autores, creando un enmerador de autores únicos y guardandolos en un csv aparte. Además, se cambian la columna "authors" por los correspondientes IDs del enumerador de autores  
 df,list\_authors = procesar\_authors(df)  
 pd.DataFrame(enumerate(list\_authors)).to\_csv("autores.csv",index=False,header=["ID-Author","Author-name"])

* + 1. Establecer las rutas destino del dataset tratado
       1. Tantos como se desean. En nuestro caso, se establecen 2
          1. Para el formato Excel o hoja de cálculo
          2. Para el formato .json

# Guardar datos en Excel y JSON  
 ruta\_excel = "libros\_limpios.xlsx"  
 ruta\_json = "libros\_limpios.json"

* + 1. Exportar el dataset tratados a distintos ficheros

guardar\_datos(df, ruta\_excel, ruta\_json)

El flujo final del main quedaría de la siguiente forma:

# Ejemplo de flujo completo  
**if** \_\_name\_\_ == "\_\_main\_\_":  
 # Ruta del archivo CSV de entrada ensucdiado  
   
 ruta\_csv = "dataset\_ensuciado.csv"  
  
 # Cargar los datos  
 df = cargar\_datos(ruta\_csv)  
  
 **if** df **is** **not** None:  
 # Limpiar datos  
 df = limpiar\_datos(df)  
  
 #ORDENAR LIBROS POR SU RATING  
 df = df.sort\_values(by='average\_rating', ascending=False)  
  
 print(f"Primeras filas ordenados de mayor a menor por rating {df.head(10)[['title','average\_rating']]}")  
 print(f"ÚLtimas filas ordenados de mayor a menor por rating {df.tail(10)[['title','average\_rating']]}")  
   
 # #LISTA DE AUTORES SIN DUPLICADOS  
 # list\_authors = []  
 # df['authors'].map(lambda autores: list\_authors.extend(separarAutores(autores)))  
 # print(f"Número total de autores con duplicados {len(list\_authors)}")  
 # list\_authors = set(list\_authors)  
 # print(f"Número total de autores sin duplicados {len(list\_authors)}")  
  
 #Se procesan los autores, creando un enmerador de autores únicos y guardandolos en un csv aparte. Además, se cambian la columna "authors" por los correspondientes IDs del enumerador de autores  
 df,list\_authors = procesar\_authors(df)  
 pd.DataFrame(enumerate(list\_authors)).to\_csv("autores.csv",index=False,header=["ID-Author","Author-name"])  
  
 # Guardar datos en Excel y JSON  
 ruta\_excel = "libros\_limpios.xlsx"  
 ruta\_json = "libros\_limpios.json"  
 guardar\_datos(df, ruta\_excel, ruta\_json)

# DOCUMENTOS UTILIZADOS Y QUE SE VAN A CREAR

Para que no nos dé ningún error, hay que asegurarse que la ruta de origen del dataset esté bien establecido, ya que sino, nos saltará una excepción de que dicho documento no existe. En mi caso, se utilizará uno, que he creado yo, llamado “dataset\_ensuciado.csv”. Este documento se proporcionará al profesor.

ruta\_csv = "dataset\_ensuciado.csv"

Aún así, si se desea utilizar otro documento, será necesario cambiar dicho nombre por el deseado por el usuario. Este documento se llama así, ya que ha sido “ensuciado” mediante la función que se ha proporionado anteriormente y se le ha dado anteriormente.

El dataset de origen se llama “books.csv”, que también se le proporcionará al profesor.

Una vez termiando de ejecutar nuestro script, se nos crearán 2 nuevos ficheros, uno de formato hoja de cálculo y otro formato JSON, con el nombre de “libros\_limpios”.

Este nombre, obviamente se puede cambiar, poniendo otro nombre en las variables de “ruta\_excel” y “ruta\_json”, respectivamente.

# Guardar datos en Excel y JSON  
 ruta\_excel = "libros\_limpios.xlsx"  
 ruta\_json = "libros\_limpios.json"

Por último, se creará un dataset relacionado con los enumeradores creados a partir de los autores, que se nombrará como “autores.csv”. Obviamente, este nombre se puede cambiar, poniendo un nombre diferente en la siguiente línea de código. En vez de “autores.csv”, poner el nombre deseado.

pd.DataFrame(enumerate(list\_authors)).to\_csv("autores.csv",index=False,header=["ID-Author","Author-name"])

# ARCHIVOS PROPORCIONADOS

En dicha entrega, se van a poder observar los siguientes documentos:

1. Main.py: archivo principal, en el que se presenta el flujo principal de trabajo explicado en el documento. Es el archivo que se debe ejecutar para realizar todo el tema de tratamiento del dataset.
2. ensuciarDataset.py: Archivo relacionado con las funciones de “ensuciamento” del dataset, para podernos poner en la piel de un Analista de Datos de verdad.
3. Books.csv: Se trata del dataset original, recogido de la página web de kaggle.
4. Dataset\_ensuciado.csv: Se trata del dataset ensuciado, utilizando las funciones implementadas en el archivo de “ensuciarDataset.py”. Además, se trata del dataset que se utilizará en el flujo principal de trabajo. Es decir, en el main.py

# SALIDA DEL FLUJO PRINCIPAL DE TRABAJO (MAIN)

Datos cargados exitosamente.  
C:\Users\celia\OneDrive - Universidad de Extremadura\Escritorio\Sergio\Uni\4º-curso\1º-cuatri\PBD\Practicas\Dataframe-Pandas\Proyecto\main.py:40: FutureWarning: A value **is** trying to be set on a copy of a DataFrame **or** Series through chained assignment using an inplace method.  
The behavior will change **in** pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves **as** a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', **try** using 'df.method({col: value}, inplace=True)' **or** df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df[column].fillna(df[column].mean(), inplace=True) # Rellenar con la media  
C:\Users\celia\OneDrive - Universidad de Extremadura\Escritorio\Sergio\Uni\4º-curso\1º-cuatri\PBD\Practicas\Dataframe-Pandas\Proyecto\main.py:40: SettingWithCopyWarning:  
A value **is** trying to be set on a copy of a slice **from** a DataFrame  
  
See the caveats **in** the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df[column].fillna(df[column].mean(), inplace=True) # Rellenar con la media  
C:\Users\celia\OneDrive - Universidad de Extremadura\Escritorio\Sergio\Uni\4º-curso\1º-cuatri\PBD\Practicas\Dataframe-Pandas\Proyecto\main.py:38: FutureWarning: A value **is** trying to be set on a copy of a DataFrame **or** Series through chained assignment using an inplace method.  
The behavior will change **in** pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves **as** a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', **try** using 'df.method({col: value}, inplace=True)' **or** df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df[column].fillna(df[column].mode()[0], inplace=True) # Rellenar con la moda  
C:\Users\celia\OneDrive - Universidad de Extremadura\Escritorio\Sergio\Uni\4º-curso\1º-cuatri\PBD\Practicas\Dataframe-Pandas\Proyecto\main.py:38: SettingWithCopyWarning:  
A value **is** trying to be set on a copy of a slice **from** a DataFrame  
  
See the caveats **in** the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df[column].fillna(df[column].mode()[0], inplace=True) # Rellenar con la moda  
Datos nulos y duplicados eliminados.  
Primeras filas ordenados de mayor a menor por rating title average\_rating  
4934 The Brothers Karamazov 5.0  
855 Literature Circle Guide: Bridge to Terabithia:... 5.0  
786 Willem de Kooning: Late Paintings 5.0  
4126 Zone of the Enders: The 2nd Runner Official St... 5.0  
4789 The Diamond Color Meditation: Color Pathway to... 5.0  
10263 Bill Gates: Computer Legend (Famous Lives) 5.0  
1243 Middlesex Borough (Images of America: New Jersey) 5.0  
9283 Oliver Wendell Holmes **in** Paris: Medicine Theo... 5.0  
9325 Tyranonsaurus Wrecks (Stanley #1) 5.0  
6776 Delwau Duon: Peintiadau Nicholas Evans = Symph... 5.0  
ÚLtimas filas ordenados de mayor a menor por rating title average\_rating  
9332 Random House Crossword Mega Omnibus Volume 1 0.0  
7635 Your Child **and** Jesus: AF amily Activity Book 0.0  
7574 The Berlin Phenomenology 0.0  
265 Out to Eat London 2002 (Lonely Planet Out to Eat) 0.0  
2533 Insights: Talks On The Nature Of Existence 0.0  
375 The Brothers Karamazov 0.0  
2758 Venacs onetnih venaca; Puževa srma 0.0  
2532 How To Meditate: An Anthology Of Talks On Medi... 0.0  
11493 Random House Crossword Mega Omnibus Volume 1 0.0  
11543 Canopy: A Work **for** Voice **and** Light **in** Harvard ... 0.0  
Enumerador de autores creado y asignado en el DataFrame.  
Datos guardados en formato Excel en libros\_limpios.xlsx.  
Datos guardados en formato JSON en libros\_limpios.json.