Benemérita Universidad Autónoma de Puebla

Facultad de ciencias de la computación (FCC)

Reporte de limpieza de la Base de datos

Introducción a la ciencia de datos

Responsable: Manzanarez Peña Victor Hugo

Docente: Jaime Alejandro Romero Sierra

Link al repositorio de GitHub: https://github.com/shiro0416r-wq/Prograyecienciadedatos.git

Índice

Descripción inicial de la base de datos

Saludos, cordial lector. El día de hoy en este documento se hará el reporte completo de la limpieza a la base de datos "Video game sales" en la cual se analizan los datos de venta de más de 16,500 videojuegos. A continuación una descripción breve del significado cada columna:

- 1.-Rank: Ranking en base a las ventas totales
- 2.-Name: Nombre del videojuego
- 3.-Platform: Plataforma en la cual se lanzó el videojuego
- 4.-Year: Año en el que se lanzó el videojuego
- 5.-Genre: Genero del videojuego
- 6.-Publisher: editorial del videojuego
- 7.-NA_Sales: Ventas en Norteamérica (en millones)
- 8.-EU_Sales: Ventas en Europa (en millones)
- 9.-JP_Sales: Ventas en Japón (en millones)
- 10.-Other_Sales: Ventas en el resto del mundo (en millones)
- 11.-Global_Sales: Ventas mundiales totales

Sin más que mencionar, procedemos con el proceso de limpieza.

Proceso de limpieza

A continuación se documentara como fue que se realizó la limpieza de la base de datos:

1.- Revisión de datos faltantes

Lo primero que se comprobó al iniciar la limpieza, fue la cantidad de datos nulos en la base de datos, para lo cual se hizo uso del comando **df.isnull().sum(),** para posteriormente utilizar el comando **df.info()** para conocer el tipo de dato de cada columna.

```
#Verificamos cuantos datos nulos tenemos en cada columna
df.isnull().sum()
```

```
Rank            587
Name            587
Platform        587
Year            898
Genre           587
Publisher       654
NA_Sales        587
EU_Sales        587
JP_Sales        587
Other_Sales     587
Global_Sales    970
dtype: int64
```

```
#Verificamos cuales son los tipos de datos de cada columna
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19585 entries, 0 to 19584
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Rank          18998 non-null  object 
 1   Name          18998 non-null  object 
 2   Platform      18998 non-null  object 
 3   Year          18687 non-null  object 
 4   Genre         18998 non-null  object 
 5   Publisher     18931 non-null  object 
 6   NA_Sales      18998 non-null  float64
 7   EU_Sales      18998 non-null  float64
 8   JP_Sales      18998 non-null  float64
 9   Other_Sales   18998 non-null  float64
 10  Global_Sales  18615 non-null  float64
dtypes: float64(5), object(6)
memory usage: 1.6+ MB
```

2.- Detección y manejo de duplicados

Para la limpieza de esos datos simplemente se utilizó la función **drop_duplicates()**. Una cosa importante es que si bien logro deshacerse de una cantidad importante de datos duplicados, no logro deshacerse de todos, puesto que algunos de ellos contenían datos nulos (NaN) en algunas de sus columnas, pero esos datos los eliminamos más adelante.

```
df = df.drop_duplicates()
df
```

	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
0	1	NaN	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37
...	...	...	...	...	...	...	...	...	...	...	...
19580	NaN	MotoGP 07	PS2	2007.0	Racing	Capcom	0.05	0.04	0.00	0.01	0.10
19581	12130	Secret Service: Ultimate Sacrifice	PS2	2008.0	NaN	Activision	0.03	0.03	0.00	0.01	0.07
19582	5887	Monster Truck Madness 64	N64	1999.0	Racing	Take-Two Interactive	0.24	0.06	NaN	0.00	0.30
19583	10408	NaN	PSV	2014.0	Role-Playing	Ubisoft	0.00	0.07	0.02	0.02	0.11
19584	13608	The Monkey King: The Legend Begins	NaN	2007.0	Shooter	Starfish	0.04	NaN	0.00	0.00	0.04

18301 rows × 11 columns

3.- Corrección de valores atípicos o inconsistentes

Para este caso se utilizaron diferentes soluciones según la columna que se limpió, a continuación una breve descripción de lo que se hizo con los datos atípicos en cada columna:

- Columna Rank

Se identificaron los datos atípicos y posteriormente, al no ser un dato crucial para el análisis (lo que significa que el registro no pierde valor en caso de no tener el dato) se reemplazaron por datos NaN para seguidamente rellenarlos con el numero 0

```
# 1.- Verificar los datos invalidos con los que se cuentan
df["Rank"].value_counts()
```

```
Rank
Auto%#    381
11937       3
12869       3
10521       3
9250        3
         ...
8213        1
11938       1
28          1
11          1
10          1
Name: count, Length: 15916, dtype: int64
```

```
#2.- Una vez identificados los datos invalidos con los que se cuentan, procedemos a reemplazarlos por datos nulos
df["Rank"] = df["Rank"].replace("Auto%#", np.nan)
```

.

```
#3.- Una vez sin datos invalidos, procedemos a rellenar los datos NaN con el numero 0
df["Rank"] = df["Rank"].fillna(0)
```

- Columna Name

Para esta columna se aplicó una estrategia similar (ubicar los datos atípicos y reemplazarlos por datos nulos), sin embargo al ser esta columna de **SUMA** importancia para el análisis (puesto que sin este dato, el registro queda inutilizado) se tomó la decisión de usar el comando **dropna** para eliminar todos esos registros.

```
#identificamos datos invalidos
df["Name"].unique()
```

```
array(['Super Mario Bros.', 'Mario Kart Wii', 'Wii Sports Resort', ...,
       'Doom 64', 'The Sims 2: Nightlife',
       'Naruto Shippuden: Ninja Destiny 2'], shape=(10449,), dtype=object)
```

```
#1.- Una vez identificados los datos invalidos con los que se cuentan, procedemos a reemplazarlos por datos nulos
df["Name"] = df["Name"].replace("Auto%#", np.nan)
```

```
#2.- Dropeamos los datos NaN
df = df.dropna(subset=['Name'])
```

- **Columnas de ventas (NA_Sales, EU_Sales, JP_Sales, Other_Sales, Global_Sales)**

La limpieza de estas columnas fue un caso particular ya que se pudieron recuperar la mayoría de los datos nulos y atípicos.

Al ser estas columnas directamente influenciadas por las demás, se optó por obtener los datos mediante cálculos simples.

Lo que se hizo con estas columnas fue lo siguiente:

Se eliminaron todos los registros que contenían más de 1 dato nulo en alguna de estas columnas (porque de otra forma, el dato no se puede calcular) y posteriormente se obtuvieron los registros faltantes mediante fórmulas básicas de despeje (A + X = B, se despeja X= B-A)

```
# 1.- Crear una lista con las columnas de ventas
columnas_ventas = ['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales']



#2.- Creamos una columna que contara cuantos datos NaN tiene un registro en las columnas de ventas
df["NaN_Counts"] = df[columnas_ventas].isna().sum(axis=1)
```

```
#3.- Ahora conservamos solo los registros que contengan 1 o menos datos NaN en las columnas de ventas
df = df[df['NaN_Counts'] <= 1]



#4.- Nos deshacemos de la columna que nos auxilio
df = df.drop(columns="NaN_Counts")
```

```
#1.- Crear una lista con los registros NaN de global sales
global_nan = df["Global_Sales"].isna()



#2.- Tomamos los registros nulos con .loc y aplicamos la formula
#(Se añade .loc a cada uno de los registros a sumarse para aegurarnos que solo se sumen los registros de una misma fila)
df.loc[global_nan, "Global_Sales"] = (
    df.loc[global_nan, 'EU_Sales'] +
    df.loc[global_nan, 'JP_Sales'] +
    df.loc[global_nan, 'NA_Sales'] +
    df.loc[global_nan, 'Other_Sales']
)
```

```python
#3.- Procedemos a hacer lo mismo  con el resto de las columnas de ventas
eu_nan = df["EU_Sales"].isna()

df.loc[eu_nan, "EU_Sales"] = (
    df.loc[eu_nan, "Global_Sales"]-
    df.loc[eu_nan,"Other_Sales"]-
    df.loc[eu_nan,"JP_Sales"]-
    df.loc[eu_nan, "NA_Sales"]
)



jp_nan = df["JP_Sales"].isna()

df.loc[jp_nan, "JP_Sales"] = (
    df.loc[jp_nan, "Global_Sales"]-
    df.loc[jp_nan,"Other_Sales"]-
    df.loc[jp_nan,"EU_Sales"]-
    df.loc[jp_nan, "NA_Sales"]
)



na_nan = df["NA_Sales"].isna()

df.loc[na_nan, "NA_Sales"] = (
    df.loc[na_nan, "Global_Sales"]-
    df.loc[na_nan,"Other_Sales"]-
    df.loc[na_nan,"EU_Sales"]-
    df.loc[na_nan, "JP_Sales"]
)



other_nan = df["Other_Sales"].isna()

df.loc[other_nan, "Other_Sales"] = (
    df.loc[other_nan, "Global_Sales"]-
    df.loc[other_nan,"NA_Sales"]-
    df.loc[other_nan,"EU_Sales"]-
    df.loc[other_nan, "JP_Sales"]
)
```

.

- **Columna Publisher y columna Genre**

(Se incluirán ambas columnas en una sección ya que se utilizó la misma técnica para ambas.)

Directamente no podemos obtener este dato, sin embargo al haber juegos que se lanzaron varias veces en diferentes plataformas, podemos obtener ambos datos de esos registros. Pero evidentemente no todos los registros contaban con más de un lanzamiento, entonces lo que se hizo fue llenar los datos faltantes en base a los que ya se tenía y dropear a los que no se podían recuperar.

```
df['Genre'] = df.groupby('Name')['Genre'].transform(lambda x: x.fillna(method='ffill').fillna(method='bfill'))
#df.groupby('Name')['Genre']: agrupa los registros por el nombre del juego.
#.transform: aplica una función y devuelve un resultado alineado con el DataFrame original.
#fillna(method='ffill') y fillna(method='bfill'):Rellenan los NaN con valores válidos del mismo grupo (mismo juego), tanto hacia adelante como hacia atrás.
```

```
#Procedemos a borrar los datos restantes
df = df.dropna(subset=['Genre'])
```

(Se hizo lo mismo con la columna “Publisher”)

- **Columnas restantes (Year y platform)**

Ambas columnas presentaban el mismo problema: los datos contenidos son de suma importancia y no hay forma de recuperarlos. Por lo tanto aunque en este caso la columna platform representaba menos del 5% de los datos y se podía dropear sin problemas, no sucedía lo mismo con la columna

year. Ya que esta columna representaba cerca del 7% de los datos, pero al no haber una manera concreta de poder recuperarlos y además asegurarse de que estos datos fueran correctos, se tomó la decisión de eliminarlos todos (tanto los datos atípicos como los datos nulos).

Columna genre:

```
#Procedemos a dropear los registros
df = df.dropna(subset=["Platform"])
```

Columna Year:

```
#Reemplazando los datos invalidos por datos nulos
df["Year"] = df["Year"].replace("Auto%#", np.nan)
```

```
#Dropeamos todos los datos nulos
df = df.dropna(subset=["Year"])
```

4.- Cambio de nombres a columnas

Este paso fue bastante sencillo, y aunque no afectaba en nada a la base de datos, se optó por traducir los nombres de las columnas

```
df.rename(columns={
    'Name': 'Nombre',
    'Platform': 'Plataforma',
    'Year': 'Año',
    'Genre': 'Género',
    'Publisher': 'Editorial',
    'NA_Sales': 'Ventas_NA',
    'EU_Sales': 'Ventas_EU',
    'JP_Sales': 'Ventas_JP',
    'Other_Sales': 'Ventas_Otras',
    'Global_Sales': 'Ventas_Globales'
}, inplace=True)
```

```
df.columns
```

0.0s

```
dex(['Rank', 'Nombre', 'Plataforma', 'Año', 'Género', 'Editorial',
     'Ventas_NA', 'Ventas_EU', 'Ventas_JP', 'Ventas_Otras',
     'Ventas_Globales'],
    dtype='object')
```

5.- Conversión de tipos de datos

Para este momento, todas las columnas contaban con su respectivo dtype correcto a excepción de la columna year, la cual fue corregida (de obj a int)

```
#Al ser un object, primero lo convertimos a float
df['Year'] = df['Year'].astype(float)
```

```
#Convertimos a int
df['Year'] = df['Year'].astype(int)
```

5.- Validación final

Finalmente, mostramos como fue que nuestra base quedo 100% libre de datos tanto nulos como atípicos

```
#Comprobando que en efecto, ya no se cuenta ni con datos nulos ni datos invalidos
df.isnull().sum()
```

```
Rank              0
Nombre            0
Plataforma        0
Año               0
Género            0
Editorial         0
Ventas_NA         0
Ventas_EU         0
Ventas_JP         0
Ventas_Otras      0
Ventas_Globales   0
dtype: int64
```

Conclusiones

Los principales problemas que se encontraron en la base de datos fueron los siguientes

- **Datos duplicados**
- **Datos nulos**
- **Datos atípicos**

Y las técnicas que se aplicaron respectivamente fueron las siguientes

- **Para datos duplicados**: comando drop_duplicates
- **Para datos nulos**: Relleno (comando fillna) y dropeo (comando dropna)
- **Para datos atípicos**: Reemplazo (comando replace), relleno (comando fillna) y dropeo (comando dropna)

¿Que se aprendió con esta limpieza?

Desde la experiencia personal, se aprendió más de lo esperado. Lo que más gane fue un mejor dominio del uso de los comandos para la correcta limpieza de los datos.

También se aprendió a encontrar soluciones correctas para recuperar la mayor cantidad de datos posibles sin tener que recurrir a medidas drásticas (como el comando dropna)

Y por último pero no por eso menos importante, se aprendió el uso de nuevos comandos útiles que podrían ser utilizados en un futuro para encontrar mejores soluciones a problemas que de cualquier otra manera serían muy difícil de solucionar (los casos más remarcados son el uso del comando .loc y el comando groupby)