Proyecto Data Science:

“Global sales prediction”

Autor: Gambin Mariano
Profesor: Germán Rodriguez
Tutor: Mateo Bongiorno
Comisión: 60895

Video Games Dataset

El dataset elegido contiene una lista de videojuegos. Cada uno de los mismos están caracterizados por un género, un año de lanzamiento, publisher(editor), plataforma, así también como las ventas que representaron esos juegos en correspondientes regiones (NA, EU, JP, otros) y también encontraremos las valoraciones promedio de los críticos y de los usuarios.

Descripción columnas del dataset

[Name]: Nombre del juego.

[Year]: Año en el que fue lanzado.

[Genre]: Género oficial del juego.

[Publisher]: Editora del juego.

[Platform]: Plataforma para la que el juego fue lanzado

[NA_Sales]: Ventas en Norte América por millón.

[EU_Sales]: Ventas en la Unión Europea por millón.

[JP_Sales]: Ventas en Japón por millón.

[Other_Sales]: Ventas del resto del mundo por millón.

[Global_Sales]: Ventas totales por millón (Suma [NA_Sales], [EU_Sales], [JP_Sales], [Other_Sales]).

[Critic_Score]: Promedio de los puntajes del juego recibido por los críticos.

[Cricitc_Count]: Cantidad de críticos que realizaron una valoración.

[User_Score]: Promedio de los puntajes del juego recibido por los usuarios.

[User_Count]: Cantidad de usuarios que realizaron una valoración.

[Developer]: Entidad que desarrolló el juego.

[Rating]: Clasificación del juego (E, Everyone, T-Teens, etc).

Columnas interesantes para análisis

[NA_Sales], +[EU_Sales], +[JP_Sales], +[Other_Sales]*= [Global_Sales]. Las columnas de ventas pueden sernos de mucha utilidad, por ejemplo, al cruzarla con el género o los años, podríamos obtener insights sobre cómo evolucionaron las ventas de las distintas regiones por año, como evolucionó la cantidad de géneros por año.

[Platform]. Si sumamos esta columna a las ventas podríamos determinar que plataformas fueron las más rentables por región y también cuantos juegos han salido por plataforma por año y la diferencia, si es que la hay, de los géneros más vendidos entre las plataformas.

[Developer]. Puede ayudarnos a recolectar insights sobre la relación entre los desarrolladores y las ventas globales ¿Cuales son los desarrolladores que más ventas globales lograron?

Abstract:

En un mercado tan competitivo como el de los videojuegos, comprender las tendencias de ventas y los factores que influyen en ellas es crucial para el éxito empresarial. Un modelo predictivo preciso puede proporcionar a los desarrolladores y editores una ventaja al permitirles anticipar la demanda y ajustar estrategias de lanzamiento, marketing y desarrollo de productos de manera más efectiva.

El análisis de datos en esta industria puede revelar patrones y comportamientos del consumidor que pueden ser difíciles de detectar de otra manera. Al comprender mejor las preferencias de los jugadores y cómo estas se relacionan con las características de los juegos, las empresas pueden tomar decisiones más informadas sobre qué tipos de juegos desarrollar y cómo dirigirse a diferentes segmentos de mercado.

La aplicación de técnicas de aprendizaje automático en este contexto puede proporcionar valiosos conocimientos sobre la relación entre variables, permitiendo a las empresas optimizar sus procesos de toma de decisiones y maximizar su retorno de inversión en el desarrollo y comercialización de videojuegos.

Este proyecto puede ayudar a identificar oportunidades de inversión y crecimiento dentro de la industria de los videojuegos. Al analizar qué características y géneros de juegos tienden a tener un mejor desempeño en términos de ventas globales, los inversores pueden tomar decisiones más fundamentadas sobre dónde asignar recursos y capital.

Como objetivo buscamos predecir las ventas globales por millon utilizando un algoritmo de regresión, teniendo en cuenta los puntos ya establecidos como afirmaciones que guían nuestra investigación.

Exploración y limpieza

Queres determinar cuál es el % de los valores NaN de cada columna para poder tener un ideal de proporción y así especificar las estrategias de imputación para cada uno de los casos

<Fragmento de código>

```
#Porcentaje de valores nulos por columna (Sumo los valores y los divido por el largo del dataframe * 100)
percentage_null = round(df_games.isnull().sum() / len(df_games)*100,2)

percentage_null
```

Nombre columna	%Nulos
Name	0.01%
Platform	0.00%
Year_of_Release	1.61%
Genre	0.01%
Publisher	0.32%
NA_Sales	0.00%
EU_Sales	0.00%
JP_Sales	0.00%
Other_Sales	0.00%
Global_Sales	0.00%
Critic_Score	51.33%
Critic_Count	51.33%
User_Score	40.10%
User_Count	54.60%
Developer	39.61%

Rating	40.49%

Podemos observar con esta variable aplicada el porcentaje de valores nulos por columna. Existen columnas en nuestro data frame que cuentan con más del 35% de sus valores nulos. Dependiendo su tipo de dato e importancia utilizaremos distintos métodos para imputar dichos valores descritos más adelante.

Imputación de valores *NaN* con sklearn

Considerando que la idea es predecir las ventas globales existen columnas como "Year_of_Release", "Genre" y las distintas de "Sales", para poder hacer una exploración y buscar patrones a través de dichas columnas.

- Las columnas **[Critic_Score], [Critic_Count], [User_Count], [Rating], [Developer]** tienen más de un 35% de valores nulos. Estas columnas pueden asistirnos para la predicción que estamos buscando, por ende, optamos por imputar los valores nulos utilizando distintos métodos proporcionados por la librería sklearn

- La columna **[User_Score]** tiene una situación un poco más particular. La misma columna se encuentra como tipo objeto ya que cuenta con un valor que es 'tbd' (to be determined, "ha ser determinado") para recuperar esos valores vamos a realizar 3 acciones:

    1) Reemplazar los valores 'tbd' por 0 utilizando un bucle for y el método .replace

    2) Cambiar el tipo de dato de la columna a float

    3) Por último utilizar Sklearn para imputar los valores por la mediana ya que la media es más sensible a los valores extremos y el reemplazo por 0 afectaría dicha media

- La columna "Year_of_Release" que podría darnos insights sobre cómo se comportaron las ventas y los géneros a lo largo del tiempo, presenta un problema al detectar que, además de los valores *NaN*, la misma contiene valores flotantes. Vamos a imputar la poca cantidad de valores *NaN* utilizando la media y convertir el tipo de dato en entero (int)

- Las columnas **[Publisher], [Rating], [Developer]**, los valores *NaN* serán reemplazados por un valor "Otro", entendiendo que el mismo no podemos asumirlo o aproximarlo con los métodos utilizados en las columnas numéricas anteriores.

Imputacion valores columnas ['Critic_Count'], ['Critic_Score'], ['User_Count'], ['Year_of_Release']

<Fragmento de código>

```
#Instanciamos la estrategia de nuestro imputador en una variable
llamada "imputer" (esta misma es para las columnas con variables
numéricas)
imputer_mean = SimpleImputer(missing_values=np.nan,strategy='mean')

#Hacemos reshape a cada una de las columnas a las que vamos a pasar por
esta estrategia
Critic_Count = df_games['Critic_Count'].values.reshape(-1,1)
Critic_Score = df_games['Critic_Score'].values.reshape(-1,1)
User_Count = df_games['User_Count'].values.reshape(-1,1)
Year_of_Release = df_games['Year_of_Release'].values.reshape(-1,1)

#Asignamos en un variable los valores de la columna imputada
Critic_Count_imputed = imputer_mean.fit_transform(Critic_Count)
Critic_Score_imputed = imputer_mean.fit_transform(Critic_Score)
User_Count_imputed = imputer_mean.fit_transform(User_Count)
Year_of_Release_imputed = imputer_mean.fit_transform(Year_of_Release)

#Asignó cada columna imputada con su columna correspondiente
df_games['Critic_Count'] = Critic_Count_imputed
df_games['Critic_Score'] = Critic_Score_imputed
df_games['User_Count'] = User_Count_imputed
df_games['Year_of_Release'] = Year_of_Release_imputed
```

Transformación columna ['User Score]

<Fragmento de código>

```
#Busco los valores de la columna user score

user_score_valores = df_games['User_Score'].values

user_score_valores_reemplazados = [] #Nueva lista para los valores
reemplazados

for valor in user_score_valores:
  if valor == 'tbd': #Si el valor es tbd
    nuevo_valor = valor.replace('tbd','0') #Reemplazalo por '0'
    user_score_valores_reemplazados.append(nuevo_valor) #Apendealo a la
nueva lista
  else:
    user_score_valores_reemplazados.append(valor)  #Si no, apendea el
valor original

df_games['User_Score'] = user_score_valores_reemplazados
```

Al hacer esta transformación y efectivamente cambiar el valor 'tbd' por '0', vemos que hay una gran cantidad de valores en '0', reemplazar por la media los valores nan podría afectar negativamente ya que la misma es sensible a los valores extremos. Por ello optamos por reemplazar los valores 'NaN' por la mediana

<Fragmento de código>

```
#Instanciamos la estrategia de nuestro imputador en una variable
llamada "imputer" (esta misma es para las columnas con variables
númericas)
imputer_median = SimpleImputer(missing_values=np.nan,strategy='median')

#Hacemos reshape a cada una de las columnas a las que vamos a pasar por
esta estrategia
User_Score = df_games['User_Score'].values.reshape(-1,1)

#Asignamos en un variable los valores de la columna imputada
User_Score_imputed = imputer_median.fit_transform(User_Score)

df_games['User_Score'] = User_Score_imputed
```

Imputación de las columnas categóricas

```
#Imputación manual de las variables categóricas

df_games["Rating"].fillna("Otro", inplace=True)
df_games["Developer"].fillna("Otro", inplace=True)
df_games["Publisher"].fillna("Otro", inplace=True)
df_games["Genre"].fillna("Misc", inplace=True)
df_games["Name"].fillna("Otro", inplace=True)
```

Volvemos a mostrar nuestro dataframe para poder identificar si, efectivamente, hemos imputado la totalidad de los valores NaN.

Name	0
Platform	0
Year_of_Release	0
Genre	0
Publisher	0
NA_Sales	0
EU_Sales	0
JP_Sales	0
Other_Sales	0
Global_Sales	0
Critic_Score	0
Critic_Count	0
User_Score	0
User_Count	0
Developer	0
Rating	0

EDA (Exploratory Data Analysis)

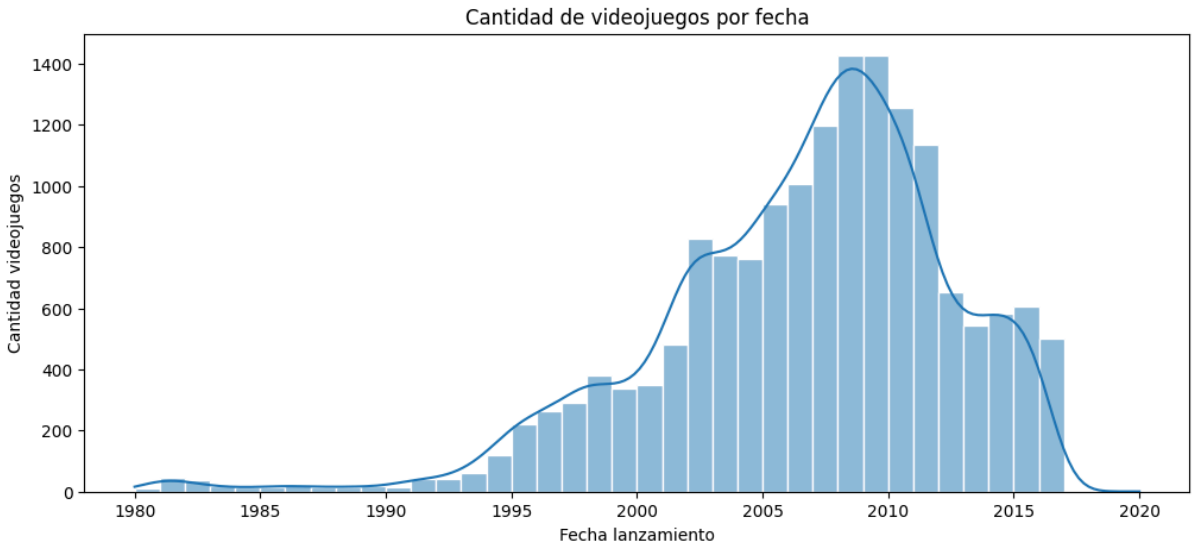
Gráficos y observaciones

Cantidad de videojuegos por fecha de lanzamiento

1) Como primera observación quisiera ver cómo se distribuyen los videojuegos según su fecha de lanzamiento. Creo una variable que va a ser mi histograma, donde le voy a pasar mi df, en el eje x quiero los años de lanzamiento y muestra el conteo por año

<Fragmento de código>

```
fig, ax = plt.subplots(figsize=(12,5))
frecuencia_años = sns.histplot(data=df_games, x="Year_of_Release",
binwidth = 1, edgecolor = "white", linewidth = 1, kde="True")
frecuencia_años.set(xlabel="Fecha lanzamiento", ylabel="Cantidad
videojuegos" ,title="Cantidad de videojuegos por fecha")
plt.show()
```



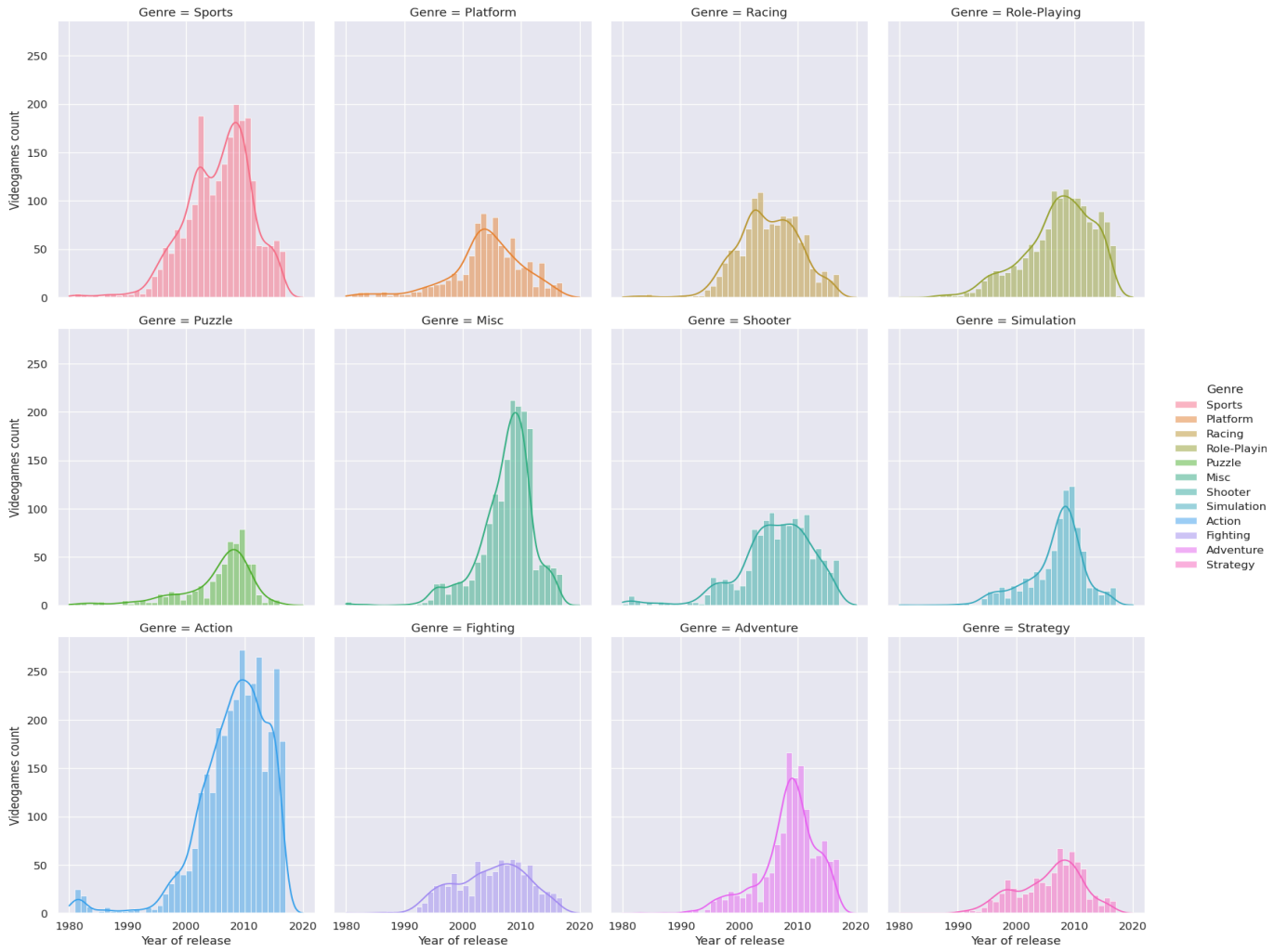
Podemos observar una distribución asimétrica positiva y podríamos decir en base a este gráfico que la mayoría de mis videojuegos aparecen entre el 2005 y el 2012 aproximadamente

Distribución de la cantidad de géneros por fecha de lanzamiento

2) Podríamos evaluar como se comportan los distintos géneros por año.

Creamos un plot donde graficamos. Estipulamos el tamaño y creamos un gráfico para determinar la frecuencia
Pasaremos el "año de lanzamiento" al eje x, el conteo será el eje y, y dividiremos los gráficos por la columna "genre".
La misma al tener 12 géneros nos mostrará 12 gráficos distintos. Pasamos col wrap para que solo muestre 4 gráficos por columna
y algunos aspectos más de formato.
<Fragmento de código>

```
plt.figure(figsize=(16, 16))
#Tamaño de cada uno de los gráficos
sns.set(rc={'figure.figsize': (25, 15)})
displot_for_genre= sns.displot(data=df_games, x="Year_of_Release",
hue="Genre", col="Genre", col_wrap=4, edgecolor = "white",binwidth = 1,
aspect=.8, kde="True")
# Itera a través de cada gráfico y establece etiquetas para los ejes
for ax in displot_for_genre.axes:
  ax.set_ylabel("Videogames count")
  ax.set_xlabel("Year of release")
plt.show()
```



Cada uno de los géneros parecen estar comportándose bastante parecido donde alcanzan un pico entre los años 2005 y 2012. Y además parecen presentar similitudes en su distribución, marcando una clara asimetría positiva.

Cantidad de videojuegos por género

3) Quisiera ver cómo se distribuyen mis videojuegos por géneros ¿Cuantos videojuegos tengo por género? Creo una variable que va a almacenar en el índice los géneros y en una columna el conteo total de ese género.

<Fragmento de código>

```
#Conteo de la cantidad de veces que aparece cada género
genre_count = df_games["Genre"].value_counts()

genre_count
```

Genre	count
Action	3370
Sports	2348
Misc	1750
Role-Playing	1500
Shooter	1323
Adventure	1303
Racing	1249
Platform	888
Simulation	874
Fighting	849
Strategy	683
Puzzle	580

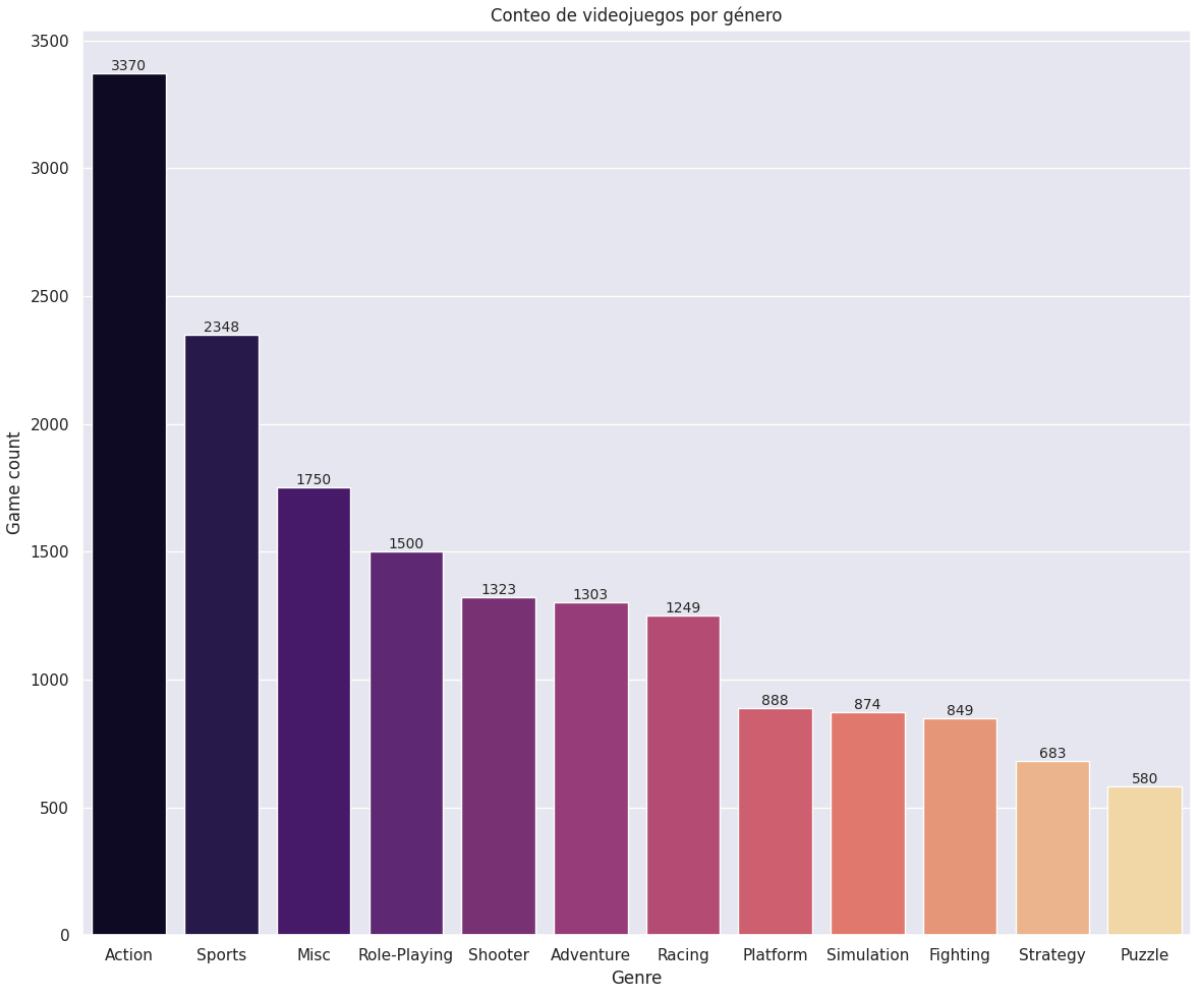
Con esta información creamos un gráfico de barras para visualizar de mejor manera la cantidad de videojuegos según el género.

Creamos un plot donde se graficara. Con seaborn creamos el gráfico de barras, pasamos los géneros como eje x y el conteo de

los mismos como eje y. Agregamos una distinción de color por cada género para hacer mas amena la lectura.

<Fragmento de código>

```
#Como la columna género y videojuegos coinciden en su cantidad de datos puedo asumir que según la cantidad de veces que aparece un género es la cantidad de videojuegos que tengo del mismo
plt.figure(figsize=(12,10))
barplot_genre = sns.barplot(x=genre_count.index, y=genre_count, hue=genre_count.index, palette="magma")
barplot_genre.set(xlabel="Genre", ylabel="Game count", title="Conteo de videojuegos por género")

for container in barplot_genre.containers: #Para cada columna, en el grafico
    barplot_genre.bar_label(container, fontsize=10) #Ingreso el valor y el tamaño de la fuente
plt.show()
```



Se puede observar una predominancia de videojuegos del género de "acción" dentro de este dataset, seguido de "deportes" y "variados"

Ventas globales por género

4) Quiero entender cuántas fueron las ventas globales por género y ver cuales han generado el mayor número de ventas.

<Fragmento de código>

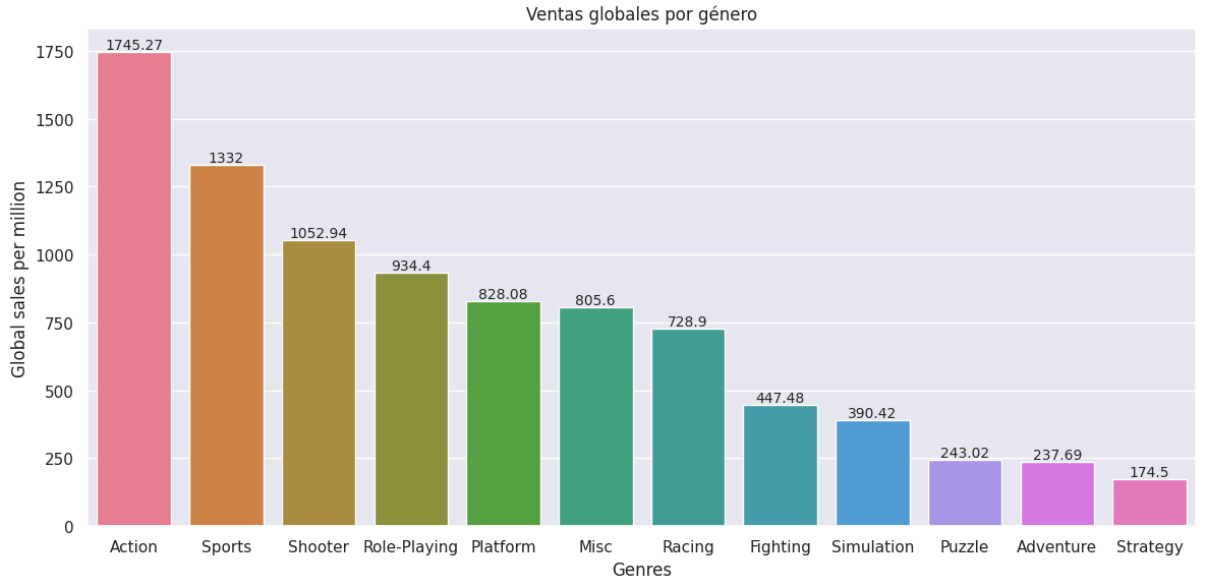
```
#Utilizo el metodo groupby para agrupar las ventas totales por género
genre_global_sales =
df_games.groupby("Genre")["Global_Sales"].sum().sort_values(ascending=F
alse)

genre_global_sales
```

Genre	Global_Sales
Action	1745.27
Sports	1332.00
Shooter	1052.94
Role-Playing	934.40
Platform	828.08
Misc	805.60
Racing	728.90
Fighting	447.48
Simulation	390.42
Puzzle	243.02
Adventure	237.69

<Fragmento de código>

```
plt.figure(figsize=(12,10))
barplot_global_sales = sns.barplot(x=genre_global_sales.index,
y=genre_global_sales, hue=genre_global_sales.index)
barplot_global_sales.set(xlabel= "Genres", ylabel="Global sales per
million", title="Ventas globales por género")
```

```
for container in barplot_global_sales.containers:
  barplot_global_sales.bar_label(container, fontsize=10)
plt.tight_layout()
plt.show()
```



Rápidamente podemos identificar como los géneros de acción, deportes, disparos y de rol son los que más ventas globales han tenido

Proporción de las ventas por región

5) ¿Cómo se comportó el porcentaje de las ventas por región?

Agrupamos las ventas de cada región por año utilizando el método .groupby

<Fragmento de código>

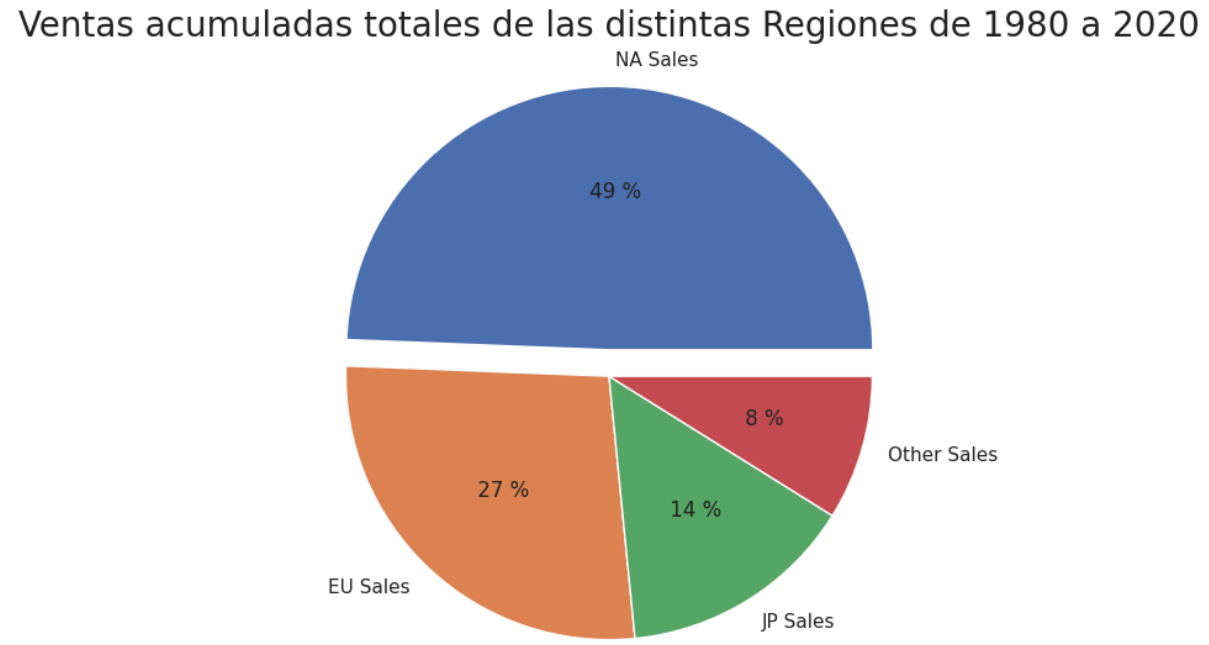
```
sum_sales_region = df_games.groupby("Year_of_Release")[["NA_Sales",
"EU_Sales", "JP_Sales", "Other_Sales"]].sum()
sum_sales_region
```

Creamos una variable que va a sumar la cantidad de ventas, que se llamara "total_sales"
y luego creamos el gráfico de torta. Pasaremos las ventas totales y que los cortes sean las columnas de mi agrupamiento anterior,
en este caso las regiones

<Fragmento de código>

```
fig, ax = plt.subplots(figsize=(10,6))
total_sales = sum_sales_region.sum()
```

```
# Cambiar nombres de los labels
labels = ["NA Sales", "EU Sales", "JP Sales", "Other Sales"]
plt.pie(total_sales, labels=labels, explode=(0.1,0,0,0), autopct="% 0i
%%")
ax.set_title('Ventas acumuladas totales de las distintas Regiones de
1980 a 2020', fontsize=20)
ax.set_label(labels)
plt.tight_layout()
plt.show()
```



Podemos observar que el 49% de las ventas corresponden a la región de NorteAmérica seguido de la Unión Europea.

Ventas por año por región

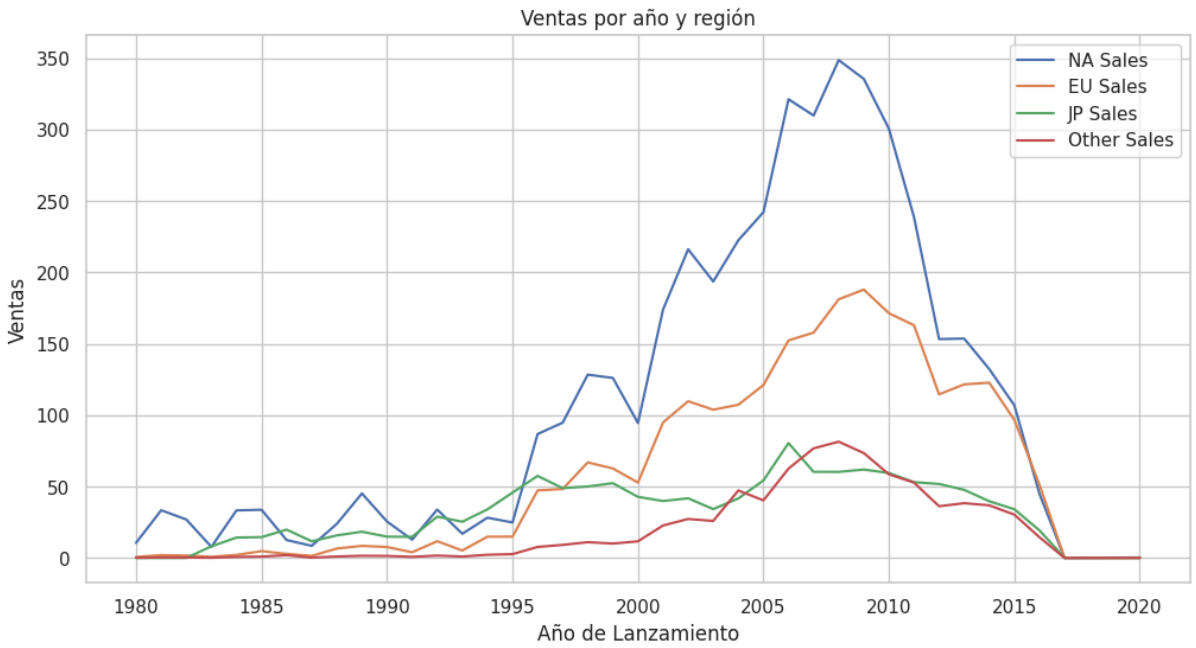
6)¿Cómo han evolucionado las ventas de las distintas regiones por año?

Antes analizamos el porcentaje total de ventas ¿Que región obtuvo la mayor cantidad de ventas?
Ahora nos puede interesar cómo se han movido las ventas por región a lo largo de los años

Establecemos nuestra figura, y un grid para formato. Creamos 4 líneas donde pasamos los años de lanzamiento en el eje x y cada una de suma de las ventas de las regiones en el eje y, aplicamos una etiqueta para cada línea. Sumamos a esto un título y una etiqueta para cada eje.

<Fragmento de código>

```
plt.figure(figsize=(12, 6))
sns.set(style="whitegrid")



sns.lineplot(data=sum_sales_region, x="Year_of_Release", y="NA_Sales",
label="NA Sales")
sns.lineplot(data=sum_sales_region, x="Year_of_Release", y="EU_Sales",
label="EU Sales")
sns.lineplot(data=sum_sales_region, x="Year_of_Release", y="JP_Sales",
label="JP Sales")
sns.lineplot(data=sum_sales_region, x="Year_of_Release",
y="Other_Sales", label="Other Sales")

plt.title("Ventas por año y región")
plt.xlabel("Año de Lanzamiento")
plt.ylabel("Ventas")

plt.show()
```



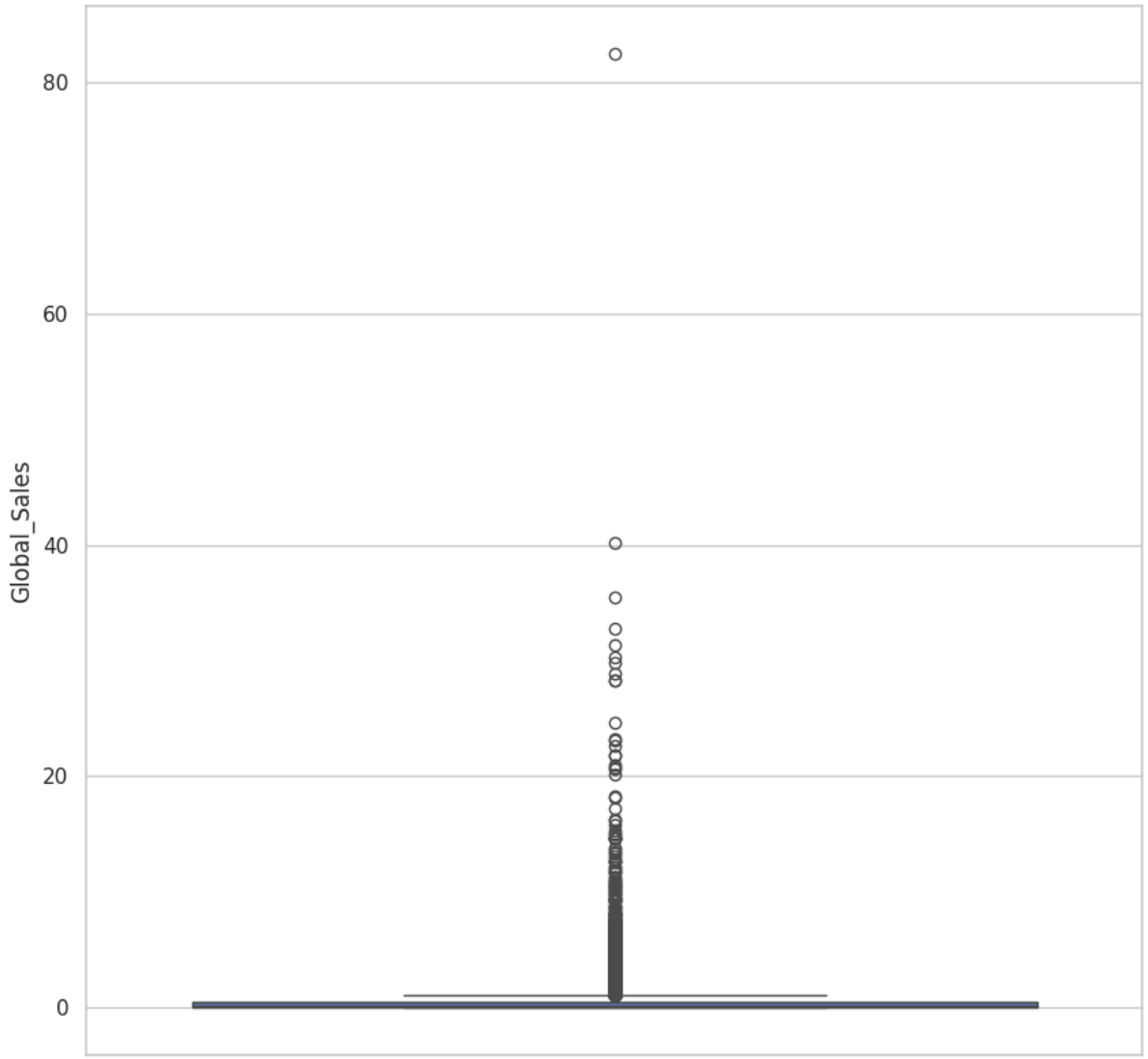
Podemos observar cómo si bien las 4 regiones parecen cumplir un mismo patrón, en donde la mayor cantidad de ventas se da entre el 2000 y el 2012, el volumen de ventas ha sido mayor en la región de Norteamérica y la Unión Europea que en las otras dos en ese mismo periodo

Outliers

Creamos un gráfico de cajas para identificar la cantidad de valores outliers presentes en nuestra columna [Global_Sales]

<Fragmento de código>

```
plt.figure(figsize=(10,10))
sns.boxplot(data=df_games['Global_Sales'])
plt.show()
```



Observamos una gran cantidad de valores outliers en nuestra variable a predecir. Crearemos dos funciones, una que calcule los límites superiores e inferiores del boxplot para determinar los valores outliers y otra que traiga los índices de dichos outlier para que los elimine.

<Fragmento de código>

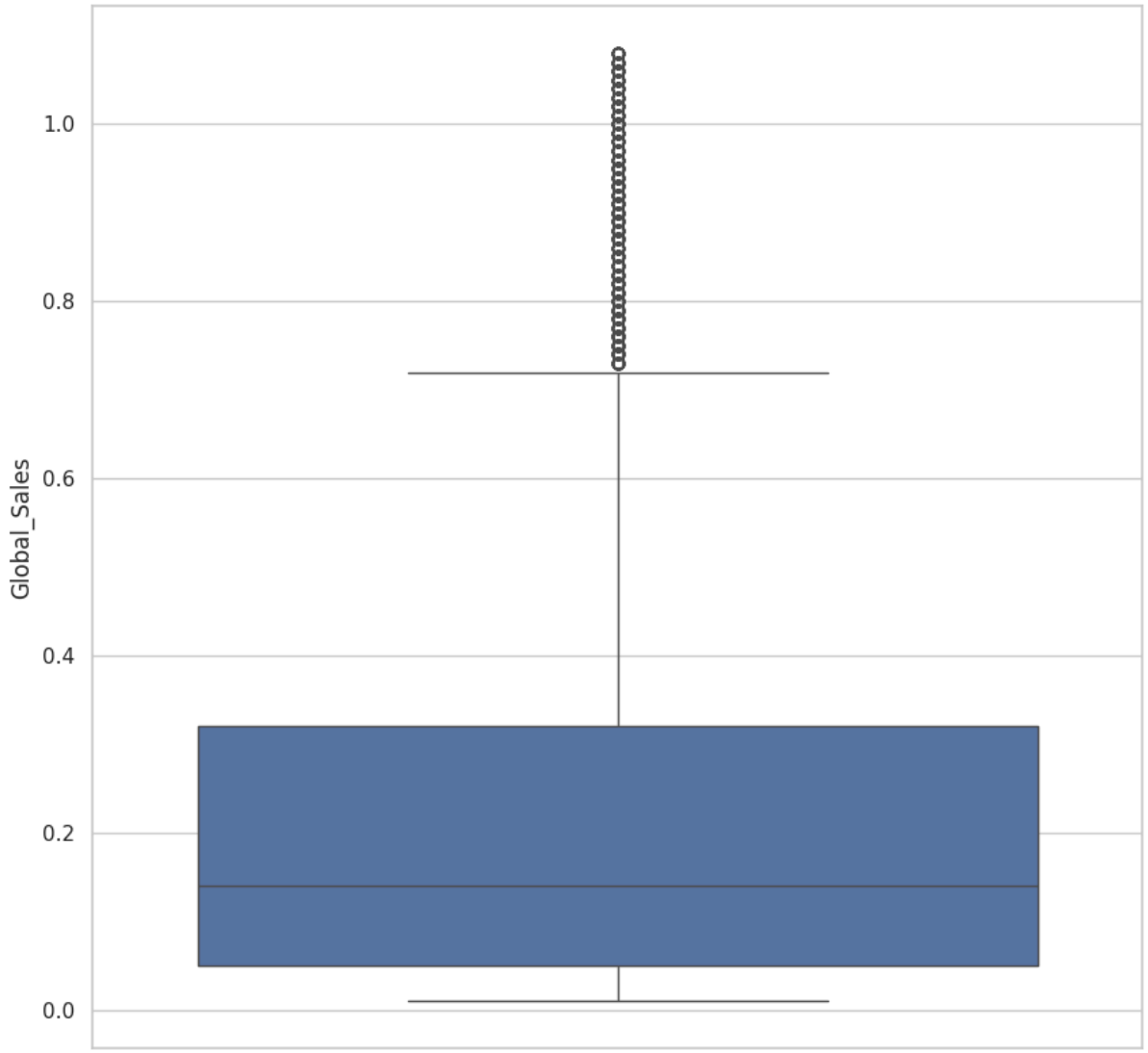
```
# Quiero crear una función que calcule los límites del boxplot y me
devuelva el límite superior e inferior en una variable con nombre ls
def outliers(data,feature):
    q1 = data[feature].quantile(0.25)
    q3 = data[feature].quantile(0.75)
    iqr = q3 - q1

    lower_limit = q1 - iqr *1.5
    upper_limit = q3 + iqr *1.5
```

```
    ls = data.index[(data[feature]<lower_limit) |
(data[feature]>upper_limit)]
    return ls

#Y otra función para que remueva los valores que están por encima o por
debajo de esos límites. Como parámetros va a tomar el data frame y el
ls calculado anteriormente
def remove(df_games,ls):
    ls = sorted(set(ls))
    df_games = df_games.drop(ls)
    return df_games

#obtención de outliers
index_list_global = []
index_list_global.extend(outliers(df_games,'Global_Sales'))

#Borramos dichos outliers y los asigno a una nueva variable que va a
almacenar nuestro dataset limpio
df_games_clean = remove(df_games,index_list_global)
```

Al volver a graficar la los valores de columna en un diagrama de cajas notamos la diferencia en el gráfico con la eliminación de dichos outliers

<Fragmento de código>

```
fig, ax = plt.subplots(figsize=(10,10))
sns.boxplot(df_games_clean['Global_Sales'])
plt.show()
```



Aproximación Modelos

Para esta aproximación a nuestros modelos de regresión vamos a eliminar las columnas de ventas (sales) que no van a participar de nuestro algoritmo y encodear el resto de los valores de nuestras columnas que sí lo van a hacer.

<Fragmento de código>

```
[36] df_games_clean.drop(['NA_Sales','EU_Sales','JP_Sales', 'User_Score','Other_Sales'], axis='columns', inplace=True)
```

Las columnas ['Platform','Genre','Name','Developer','Publisher','Rating']**.
Se encuentran con valores categóricos. Vamos a utilizar LabelEnconder para codificar cada uno de estos valores.

<Fragmento de código>

```
38] #En una variable llamada column, asigno las columnas que voy a pasar por el encoder
    column=['Platform','Genre','Name','Developer','Publisher','Rating']


    #Aplico el Label Encoder a cada una de las columnas guardadas en la variable 'column' anterior
    df_games_clean[column]=df_games_clean[column].apply(LabelEncoder().fit_transform)
```

Modelos

Utilizaremos 3 modelos de regresión.

- Regresión Lineal
- KNN
- XGBRegressor

Haremos una primera prueba con estos 3 algoritmos y en función de los resultados evaluaremos los próximos pasos a seguir

Regresión Lineal

1) Dividimos la data de entrenamiento y testing:

<Fragmento de código>

```
#Separo los datos de entrenamiento y test 70/30
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=16)

print('x_train size',X_train.shape)
print('y_train size',y_train.shape)
print('x_test size',X_test.shape)
print('y_test size',y_test.shape)
```

2) Escalamos los datos:

<Fragmento de código>

```
#Escalo los datos
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

3) Instanciamos el modelo

<Fragmento de código>

```
model = LinearRegression()
```

4) Entrenamos el modelo con la data de training

<Fragmento de código>

```
model.fit(X_train, y_train)
```

```
LinearRegression()
```

5) Predicción del modelo

<Fragmento de código>

```
# prediccion
y_pred = model.predict(X_test)
```

6) Evaluamos el modelo

<Fragmento de código>

```
# metrica R2
r2_rl = r2_score(y_test, y_pred)
r2_rl
```

```
0.10260671756716333
```

Esta métrica nos permite entender que la eficacia en la predicción de nuestro modelo utilizando un método de regresión lineal es del 10%

KNN

1) Instanciamos el modelo (como hay pasos que se repiten con respecto al anterior, solo dejamos aquellos que son distintos)

<Fragmento de código>

```
[49] # Instancio modelo KNN
     model = KNN(n_neighbors=10,weights="uniform",metric="manhattan")
```

2)

<Fragmento de código>

```
# Metrica R2 predicción
r2_knn = r2_score(y_test, y_pred)
r2_knn
```

```
0.23027966898544694
```

Esta métrica nos permite entender que la eficacia en la predicción de nuestro modelo utilizando un algoritmo de tipo KNN es del 23%

XGBRegressor

1) Instancio modelo XGBRegressor

<Fragmento de código>

```
#Instancio modelo XGBRegresor
model = XGBRegressor(n_estimators= 2000 , max_depth= 8, learning_rate = 0.01)
```

2) Métricas de evaluación

<Fragmento de código>

```
#Metrica R2 predicción
r2_xgb = r2_score(y_test, y_pred)
r2_xgb
```

```
0.38249224171468765
```

```
#Metrica MAE predicción
print("MAE",mean_absolute_error(y_test,y_pred))
```

```
MAE 0.13300263691805198
```

```
#Metrica MSE predicción
print("MSE",mean_squared_error(y_test,y_pred))
```

```
MSE 0.035731344577055 22
```

Esta métrica nos permite entender que la eficacia en la predicción de nuestro modelo utilizando un modelo XGBRegressor es del 38%

Validación Hipótesis / Ajuste de variables

Bajo el experimento anterior y utilizando estos 3 modelos de regresión para predecir las ventas globales de videojuegos podríamos decir que el más efectivo es el XGBregressor con un 38% de efectividad.

El 38% es muy bajo para decir que nuestro modelo obtuvo los resultados esperados.

Con esto podríamos decir, con los datos utilizados, ninguno de los modelos se acerca a un 60% - 65% esperado de predicción y necesitaríamos más datos para poder mejorar la exactitud de la predicción. Además, sumado a esto nuestro MAE y MSE que tomamos para determinar qué tan lejos están nuestras predicciones de nuestro valores reales es de 0,13 y 0,035 respectivamente, siendo valores bastante altos que condice con la pobre efectividad del modelo

En el siguiente paso vamos a sumar la columna *[Other_Sales]*, que representa el menor porcentaje de ventas de todas las regiones.
La idea es, sumando esta columna poder ver si con un muy bajo % de ventas, logramos predecir, cómo serán las ventas en las otras regiones (Norteamérica, Europa, Japón) y así llegar a las ventas globales (Global_Sales)

Hipótesis 2

Si sumo a mis datos anteriores una columna que traiga el menor % de ventas ¿Puedo predecir cómo se va a comportar en el resto de las regiones?

En esta hipótesis vamos a dejar una sola de las columnas de ventas [‘Other_Sales’] para poder ver si dicho cambio mejora la predicción de nuestra variable [‘Global_Sales’]

Omitiremos la recepción de los pasos anteriores y pasaremos a las métricas de predicción de cada uno de los algoritmos con dicho cambio

Hipótesis 2 ***Regresión Lineal***

<Fragmento de código>

```
#Metrica R2
r2_rl_2 = r2_score(y_test_2, y_pred_2)
r2_rl_2
```

```
0.5449235461010843
```

Para nuestras segunda hipotesis hemos obtenido un r2 de 0.54, 54% de efectividad para la predicción contra un 11% con los datos anteriores.

Hipótesis 2 ***KNN***

<Fragmento de código>

```
# metrica R2
r2_knn_2 = r2_score(y_test_2, y_pred_2)
r2_knn_2
```

```
0.6400547201378567
```

Para nuestras segunda hipótesis hemos obtenido un r2 de 0.64, 64% de efectividad para la predicción contra un 23% con nuestros datos anteriores en el mismo algoritmo KNN. Básicamente no hubo diferencia con este

Hipótesis 2 ***XGBRegressor***

<Fragmento de código>

```
#Metrica R2 predicción
r2_xgb_2 = r2_score(y_test_2, y_pred_2)
r2_xgb_2
```

```
0.8377876049277614
```

```
#Metrica MAE predicción
print("MAE",mean_absolute_error(y_test_2,y_pred_2))
```

```
MAE 0.0602415679306159
```

```
#Metrica MSE predicción
print("MSE",mean_squared_error(y_test_2,y_pred_2))
```

```
MSE 0.0093862253635323
```

Para nuestras segunda hipótesis hemos obtenido un r2 de 0.83, 83% de efectividad para la predicción contra un 38% con nuestros datos anteriores en el mismo algoritmo XGBRegressor. Aquí sí encontramos una diferencia significativa en el porcentaje de predicción. Además de tener un MAE de 0,06 y un MSE de 0,009 que indica que nuestras predicciones están muy cerca de los valores reales.

PCA

Aplicaremos para esta última hipótesis, PCA para poder ver si mejora la efectividad de nuestras predicciones.

<Fragmento de código>

```
from sklearn.decomposition import PCA
pca = PCA()
X_train_3 = pca.fit_transform(X_train_3)
X_test_3 = pca.transform(X_test_3)
```

```
# Explicación de la varianza de las componentes
varianza_explicada = pca.explained_variance_ratio_
varianza_explicada
```

```
array([0.15156212, 0.11542707, 0.11355041, 0.10217622, 0.09627367,
       0.08849107, 0.08088809, 0.07250522, 0.06998436, 0.0595608 ,
       0.04958099])
```

Nuestros datos parecen tener una varianza bastante pareja, casi 8 componentes mantienen una varianza cercana al 10% y los últimos 3 por debajo del 7%

Como próximo estableceremos un pca de 8 componentes para ver cuánto varía la predicción de nuestros 3 algoritmos

PCA Regresión Lineal

<Fragmento de código>

```
pca = PCA(n_components=8) # cantidad de componentes
X_train_3 = pca.fit_transform(X_train_3)
X_test_3 = pca.transform(X_test_3)
```

<Fragmento de código>

```
#Metrica R2
r2_rl_3 = r2_score(y_test_3, y_pred_3)
r2_rl_3
```

```
0.543294406893178
```

PCA KNN

<Fragmento de código>

```
# metrica R2
r2_knn_3 = r2_score(y_test_3, y_pred_3)
r2_knn_3
```

```
0.6445604873744659
```

PCA XGBRegressor

<Fragmento de código>

```
#Metrica R2 predicción
r2_xgb_3 = r2_score(y_test_3, y_pred_3)
r2_xgb_3
```

```
0.7026799554720213
```

Búsqueda de parámetros óptimos

<Fragmento de código>

```
#Parametros utilizados para el xgbregressor
params_grid = {
        'n_estimators': [2000, 2500, 3000],
        'max_depth': [6,7,8],
        'learning_rate':[0.01, 0.10, 0.1]
        }
```

```
#Prueba con halvingGridSearchCV
halving_cv = HalvingGridSearchCV(model_3_xgb, params_grid, scoring="r2", factor=3)
halving_cv.fit(X_train_3, y_train_3)

print("Mejores parametros", halving_cv.best_params_)
print("Mejor Score CV", halving_cv.best_score_)
print(f'Accuracy del modelo R2 = {round(r2_score(y_test_3, halving_cv.predict(X_test_3)), 2)}')
```

```
Mejores parametros {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 2000}
Mejor Score CV 0.6789626223414388
Accuracy del modelo R2 = 0.7
```

Utilizando HalvingGridSearchCV confirma que los hiperparametros utilizados en nuestra tercera prueba con el PCA incluido son los óptimos en función del r2 obtenido.

Conclusión

En conclusión, se observó que el algoritmo XGBRegressor demostró un desempeño superior en ambas hipótesis, alcanzando la mejor métrica de R2. En la primera hipótesis, donde no se disponía de ningún dato sobre las ventas, la capacidad de predicción de las ventas globales fue limitada, con una efectividad del 38%. En contraste, en la segunda hipótesis, al incorporar los datos de ventas en otras regiones, aunque representaban un porcentaje reducido del total, la capacidad predictiva mejoró significativamente, alcanzando un 85% con un MSE de 0.0090 y un MAE de 0.06.

Cuando se utilizó PCA para reducir la dimensionalidad de nuestro conjunto de datos, utilizando 8 componentes la eficacia de nuestro algoritmo de regresión se mantuvo al 54%, pero las de nuestros algoritmo de KNN y XGBRegressor cayeron en sus predicciones

aunque se mantuvieron más altas que las primeras pruebas. Por esto se decide quedarse con los resultados expresados sin la utilización de PCA.

Estos resultados sugieren que, en la primera hipótesis, la falta de datos limitó la precisión de las predicciones de ventas globales, destacando la necesidad de obtener más información para mejorar la exactitud del modelo. Por otro lado, en la segunda hipótesis, se evidenció que incluso con un porcentaje bajo de ventas en una región, fue posible predecir con un alto grado de certeza (85%) el comportamiento de las ventas en globales.