Tipología y Ciclo de Vida de los datos PR2

Alex Stwart Porras Palma

May 2025

Índice

1. Descripción del Dataset **2**

2. Integración y selección **3**

3. Limpieza de datos **3**

3.1. Los datos contienen ceros, elementos vacíos u otros valores numéricos que indiquen la pérdida de datos? Gestiona cada uno de estos casos utilizando el método de imputación que consideres más adecuado 3

4. Análisis de los datos **6**

4.1. Aplica un modelo supervisado y uno no supervisado a los datos y comenta los resultados obtenidos. 6

4.2. Aplica una prueba por contraste de hipótesis. Ten en cuenta que algunas de estas pruebas requieren verificar previamente la normalidad y homocedasticidad de los datos. 10

5. Resolución del problema **12**

1. Descripción del Dataset

El dataset se trata de un conjunto de datos provenientes de la PR1 en la que se realiza web scraping en la página de Steam, para capturar la información sobre videojuegos que dado un precio objetivo de compra se encuentre por debajo o no de éste, su precio si había descuento o no etc.

```
client = MongoClient(mongo_uri)
db = client["Testing"]
collection = db["steam_bot"]
docs = list(collection.find())
df = pd.DataFrame(docs)
```

Title	Discount	Old Price	New Price	Source	Created At
Sons Of The Forest	-60 %	28,99€	11,59€	sales	2025-04-08 19:15:29.036
Devil May Cry HD Collection	-67 %	29,99€	9,89€	sales	2025-04-08 19:17:11.572
Baldur's Gate 3	-20 %	59,99€	47,99€	sales	2025-04-08 19:17:11.342
Red Dead Redemption 2	-75 %	59,99€	14,99€	sales	2025-04-08 19:17:11.415
Sons Of The Forest	-60 %	28,99€	11,59€	sales	2025-04-08 19:17:11.643

Tabla 1: Discounted Games Information

Como se puede observar en la tabla el dataset contiene información de:

- Title: Título del juego
- Discount: Porcentaje de descuento si lo hubiese
- Old Price: Precio antes del descuento
- New Price: Precio después del descuento
- Source: Comando realizado en el bot de telegram para obtener los datos mediante el scraping:
  - sales
  - search
  - bulksearch
- Created At: TimeStamp que indica cuando se agregó a la base de datos

2. Integración y selección

Por motivos de enriquecimiento de los datos, y para evitar un scraping masivo, se van a añadir una columna adicional "Bought", que contendrá la información de si el juego se ha comprado o no.

Statistic	Title	Discount	Old Price	New Price	Source	Created At	Price Count
Count	2549	6	6	6	2549	2549	6.000000
Unique	2388	4	3	5	3	NaN	NaN
Top	No Man's Sky	-60 %	28,99€	11,59€	bulksearch	NaN	NaN
Freq	61	2	2	2	2537	NaN	NaN
Mean	-	-	-	-	-	2025-05-25 16:51:08.10	17.256667
Min	-	-	-	-	-	2025-04-08 19:15:29.036	7.490000
25 %	-	-	-	-	-	2025-05-25 21:04:23.612	10.315000
50 %	-	-	-	-	-	2025-05-25 22:13:49.109	11.590000
75 %	-	-	-	-	-	2025-05-25 22:52:35.856	14.140000
Max	-	-	-	-	-	2025-05-26 00:14:17.850	47.990000
Std	-	-	-	-	-	-	15.254857

Tabla 2: Statistics Table

Se muestra un ejemplo de la descripción del dataset antes de su procesamiento.

3. Limpieza de datos

Como se pudo observar en el apartado anterior es necesario procesar los datos para poder crear un modelo en los apartados posteriores.

3.1. Los datos contienen ceros, elementos vacíos u otros valores numéricos que indiquen la pérdida de datos? Gestiona cada uno de estos casos utilizando el método de imputación que consideres más adecuado

```
df.drop_duplicates(subset='title',inplace=True)
df.drop(columns='_id',inplace=True)

df.isna().sum()

def parse_price(string):
    try:
        return float(string.replace("\euro{}", '').replace(',','.
           '))
    except:
        return None
df["new_price"] = df["new_price"].apply(parse_price)
df['price'] = df['price'].fillna(df['new_price'])
df.drop(columns=["discount", "old_price", "new_price"],inplace=
   True)

df.describe()
```

En este caso el dataset sin procesar contiene 2383 NaN en sus columnas de discount, old price y new price. Esto es debido a que la base de datos de estilo documento, no siempre contiene la misma estructura. Concretamente en este caso, se generaron muchos datos con el método bulksearch los cuales se almacenan con otra estructura de datos en las que directamente se obtiene solo el precio final es por ello, por lo que estas columnas contienen un gran número de NaNs, sin embargo la columna price no solo contiene 5, y esta información se puede obtener de la columna new price a la cual solo habría que quitarle el símbolo de euro. Finalmente, se eliminan las columnas.

Statistic	Created At	Price
Count	2388	2388.000000
Mean	2025-05-25 17:27:00.418834432	22.038044
Min	2025-04-08 19:15:29.036000	0.000000
25 %	2025-05-25 21:04:43.420249856	9.750000
50 %	2025-05-25 22:13:58.548000	18.990000
75 %	2025-05-25 22:52:53.411749888	29.990000
Max	2025-05-26 00:14:17.850000	99.990000
Std Dev	NaN	16.595385

Tabla 3: Summary Statistics

Por el bien del ejercicio y como se mencionó anteriormente se va a agregar una nueva columna llamada is bought, que contendrá la información sobre si un juego ha sido comprado o no. Adicionalmente se crea la columna price tier, que contiene diferentes rangos de precios, menor que 10 euros, entre 10 y 25 etc. Para ser convertida en dummies y adaptar el dataframe: Para continuar con el desarrollo del ejercicio, se va a realizar la

Index	Title	Source	Created At	Price	Is Bought	<10€	10-25€	25-50€	>50€
0	Sons Of The Forest	sales	2025-04-08 19:15:29.036	11.59	1	0	1	0	0
1	Devil May Cry HD Collection	sales	2025-04-08 19:17:11.572	9.89	0	1	0	0	0
2	Baldur's Gate 3	sales	2025-04-08 19:17:11.342	47.99	0	0	0	1	0
3	Red Dead Redemption 2	sales	2025-04-08 19:17:11.415	14.99	0	0	1	0	0
5	Devil May Cry 5	sales	2025-04-08 19:17:11.510	7.49	1	1	0	0	0

Tabla 4: Price and purchase information of various games

adición de otra columna llamada genre, que contendrá la información sobre el género del videojuego, para ello y continuando con el temario se hará uso en esta ocasión de una API, concreamente la API de rawg.io.

```
def fetch_genres(title):
 params = {
     'key': API_KEY,
     'search': title,
     'page_size': 1
 }
 try:
     response = requests.get(BASE_URL, params=params)
     response.raise_for_status()
     data = response.json()
     if data['results']:
         genres = [genre['name'] for genre in data['results'][0].
            get('genres', [])]
```

```
            return ', '.join(genres)
    except Exception as e:
        print(f"Error fetching genres for '{title}': {e}")
    return 'Genre not found'
```

Index	Title	Source	Created At	Price	Is Bought	Price Tier <10€	Price Tier 10-25€	Price Tier 25-50€	Price Tier >50€	Genre
0	Sons Of The Forest	sales	2025-04-08 19:15:29.036	11.59	1	0	1	0	0	Indie
1	Devil May Cry HD Collection	sales	2025-04-08 19:17:11.572	9.89	0	1	0	0	0	Adventure
2	Baldur's Gate 3	sales	2025-04-08 19:17:11.342	47.99	0	0	0	1	0	Strategy
3	Red Dead Redemption 2	sales	2025-04-08 19:17:11.415	14.99	0	0	1	0	0	Action
5	Devil May Cry 5	sales	2025-04-08 19:17:11.510	7.49	1	1	0	0	0	Action

Tabla 5: Price and purchase information of various games, including genre

Una vez realizado todos estos cambios, para completar el apartado de imputación, se va van a generar valores faltantes del 20 % del data set, y se van a imputar con la mediana. ¿Por qué con la mediana y no la media?, (u otros métodos), en este caso la mediana se ha escogido como mejor método de imputación, ya que al hacerlo con la media, ésta se ve altamente afectada por valores extremos, tendrá sesgos. Otros métodos de imputación como la moda, resulta inadecuado para valores faltantes aleatorios, imputar con los valores anteriores o posteriores sería útil para series temporales etc.

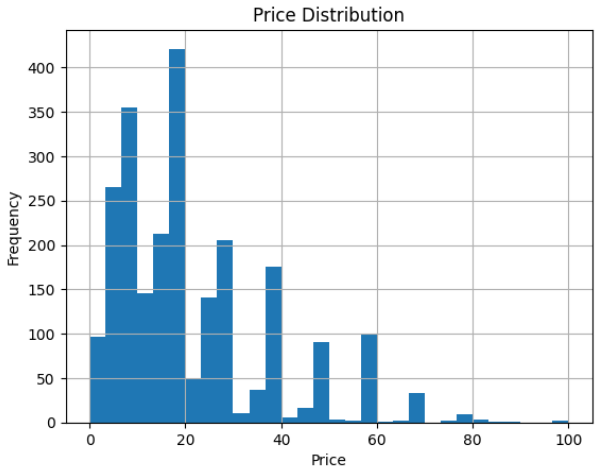

Figura 1: Datos antes de la imputación

Como se puede observar se pierden datos reales, pero se mantiene la distribución, así que dará como válido para entrenar modelos.
Con esta representación ya se puede observar la existencia de valores atípicos, pero aún así se va a representar un diagrama de cajas:
En este caso los se van a dejar los valores atípicos y se va a proceder a crear un modelo de clasificación binaria, es decir, si se compraría o no un videojuego, mediante el uso de Árboles de decisión.

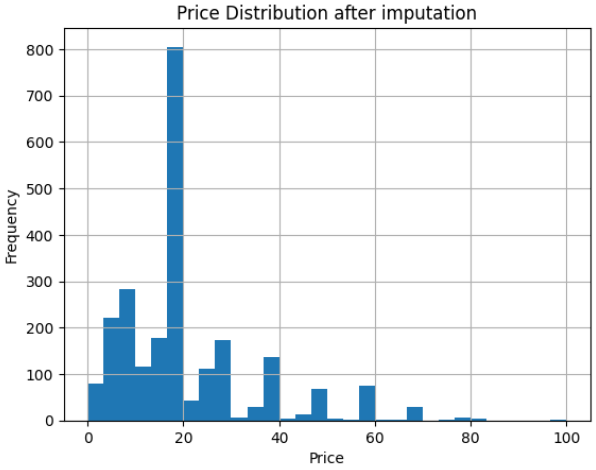

Figura 2: Datos después de la imputación

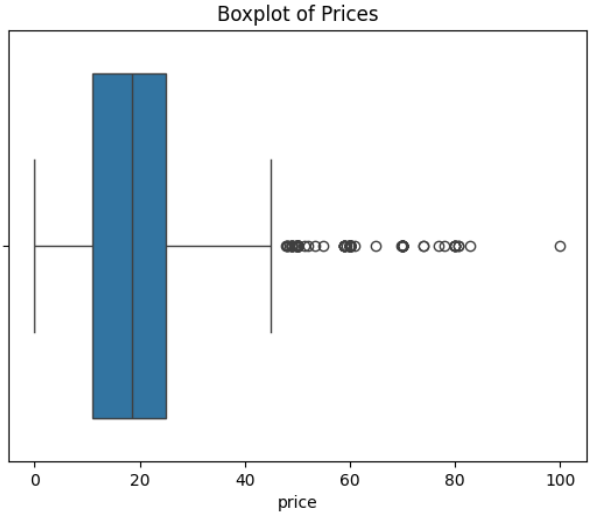

Figura 3: Diagrama de cajas y representación de valores atípicos

4. Análisis de los datos

4.1. Aplica un modelo supervisado y uno no supervisado a los datos y comenta los resultados obtenidos.

Para modelos supervisados se va a hacer uso de decision trees como se mencionó anteriormente y de random forest consecuentemente. Para ello es necesario preparar los datos de entrenamiento y test.

```
X = df.drop(columns=['title', 'source', 'is_bought'])
y = df["is_bought"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_tree
```

```
    ))
print("\nClassification Report:\n", classification_report(y_test,
    y_pred_tree))
```

Classification Report

	precision	recall	f1-score	support
0	0.70	0.83	0.76	283
1	0.66	0.49	0.56	195
accuracy			0.69	478
macro avg	0.68	0.66	0.66	478
weighted avg	0.68	0.69	0.68	478

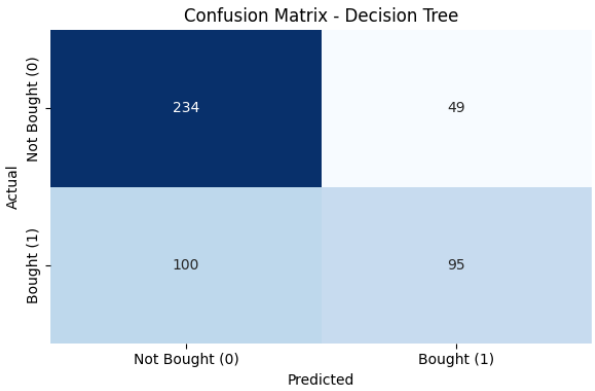

Figura 4: Matriz de confusión para el modelo de Árbol de decisión

```
rf = RandomForestClassifier(n_estimators=100, max_depth=6,
    random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
    y_pred))
```

Classification Report

	precision	recall	f1-score	support
0	0.71	0.81	0.76	283
1	0.65	0.51	0.57	195
accuracy			0.69	478
macro avg	0.68	0.66	0.66	478
weighted avg	0.68	0.69	0.68	478

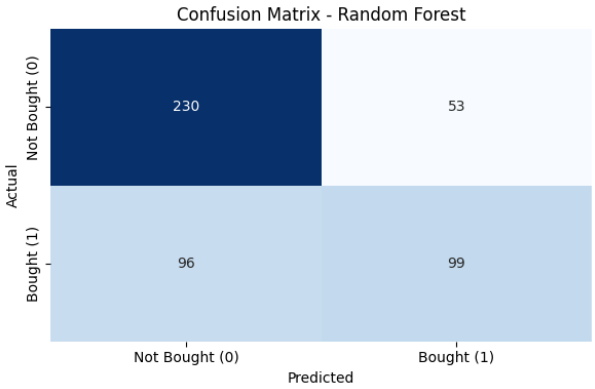

Figura 5: Matriz de confusión para el modelo de Random Forest

En cuanto al método no supervisado se ha realizado un clustering, y una proyección en 2D con PCA, debido a que hay que reducir el número de características. Para ello, primero identificamos el valor óptimo de K, mediante el método del codo.

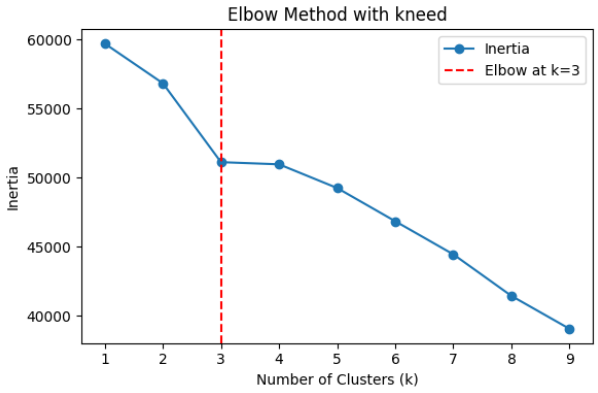

Figura 6: Método del codo para identificar k óptimo

```
kmeans = KMeans(n_clusters=optimal_k, random_state=100)
df['cluster'] = kmeans.fit_predict(X_scaled)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(6, 5))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=df['cluster'],
   palette='tab10')
plt.title('K-Means Clusters (PCA 2D Projection)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.tight_layout()
plt.show()
```

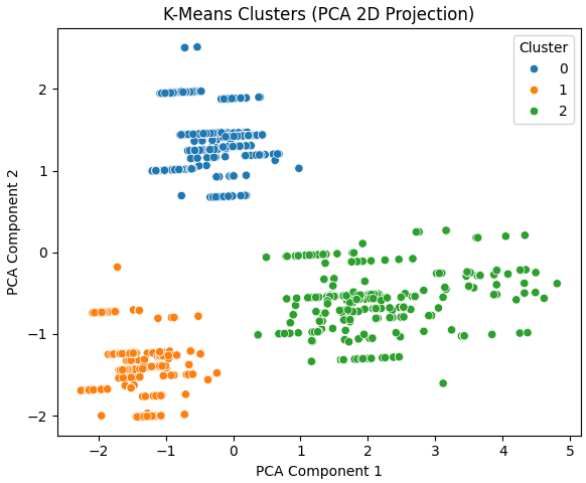

Figura 7: Proyección 2D y resultado de Clustering

Para interpretar los clusters es necesario explorar el dataset, para ello se creó la columna cluster sobre el dataset original para identificar a que cluster pertenece cada juego. De esta manera se hace representaciones de precio por cluster y de género mayoritario por juego, obteniendo los siguientes resultados.

Figura 8: Precio Medio de Videojuegos por Cluster

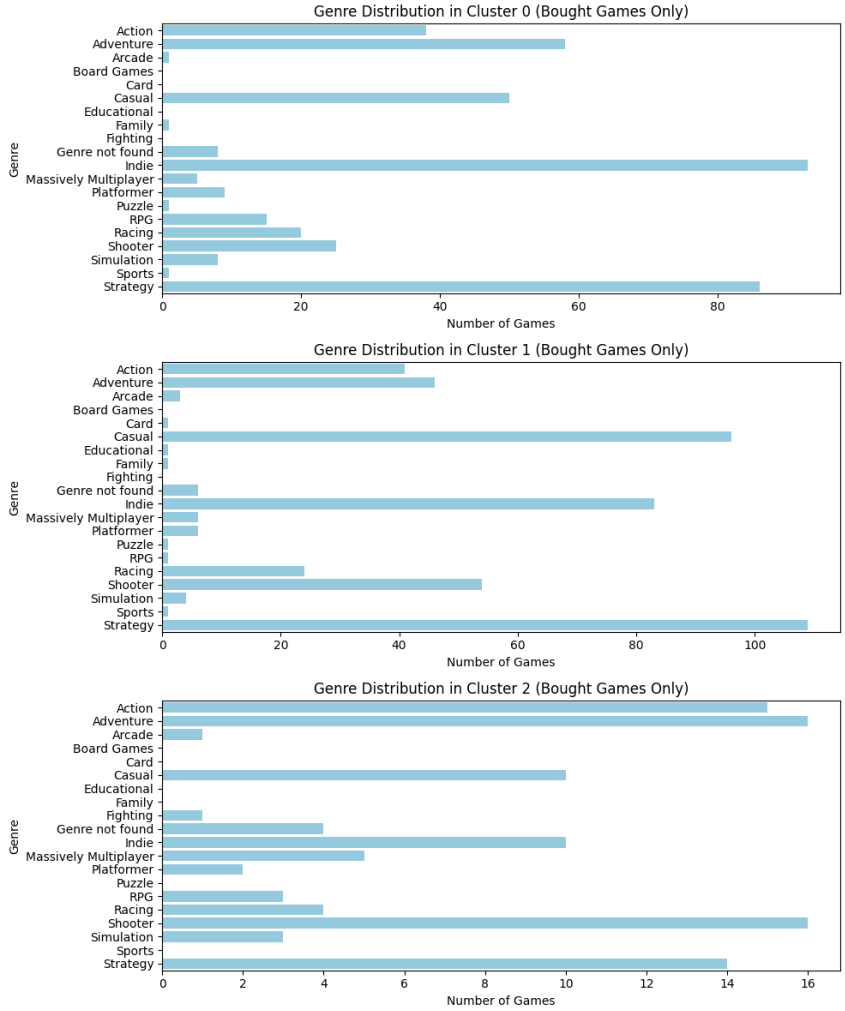

Figura 9: Distribución de género por cluster

Mediante la distribución de los géneros por cluster, no es claro un género decisivo por cada uno de ellos, teniendo en cuenta de que las características han sido los precios y géneros, entonces la representación de los cluster ha sido decisiva por los precios de los mismos

4.2. Aplica una prueba por contraste de hipótesis. Ten en cuenta que algunas de estas pruebas requieren verificar previamente la normalidad y homocedasticidad de los datos.

Para establecer un contraste por hipótesis, se va a estudiar si existe una diferencia significativa entre el precio medio de los juegos comprados o no.

$$H_0 : \mu_1 = \mu_2$$

$$H_1 : \mu_1 \neq \mu_2$$

- Hipótesis Nula H0: No hay una diferencia significativa entre la media de precios de los productos comprados y no comprados. Es decir, la media del precio de los productos comprados es igual a la media del precio de los productos no comprados

- Hipótesis Alternativa H1: Sí hay una diferencia significativa entre la media de precios de los productos comprados y no comprados. Esto significa que la media del precio de los productos comprados es diferente de la media del precio de los productos no comprados.

Ya sabemos de antemano que los datos no siguen una distribución normal, pero se puede hacer el test the Shapiro para verificar la normalidad.

```
bought_prices = df[df['is_bought'] == 1]['price']
not_bought_prices = df[df['is_bought'] == 0]['price']

shapiro_bought = shapiro(bought_prices)
shapiro_not_bought = shapiro(not_bought_prices)

print("Shapiro-Wilk Test:")
print(f"Bought: p-value = {shapiro_bought.pvalue:.4f}")
print(f"Not Bought: p-value = {shapiro_not_bought.pvalue:.4f}")
```

Shapiro-Wilk Test Results

```
Shapiro-Wilk Test:
Bought: p-value = 0.0000
Not Bought: p-value = 0.0000
```

Al ser p menor que 0.05, se rechaza la normalidad, continuamos con la homoestaciedad mediante el uso del test de Levene:

```
levene_test = levene(bought_prices, not_bought_prices)
print(f"\nLevene Test: p-value = {levene_test.pvalue:.4f}")
```

Levene Test Results

```
Levene Test: p-value = 0.0000
```

Como no se tiene ni homocedasticidad ni normalidad, se usa la prueba de Mann-Whitney U, que no requiere ni normalidad ni igualdad de varianzas

```
bought_prices = df[df['is_bought'] == 1]['price']
not_bought_prices = df[df['is_bought'] == 0]['price']

mann_test = mannwhitneyu(bought_prices, not_bought_prices,
    alternative='two-sided')
print(f"Mann-Whitney U Test: p-value = {mann_test.pvalue:.4f}")
```

Mann-Whitney U Test Results

```
Mann-Whitney U Test: p-value = 0.0000
```

Existe una diferencia estadísticamente significativa entre los precios que fueron comprados y los que no p es menor que 0.05, se rechaza la hipótesis nula de igualdad de distribuciones. El precio sí influye en la decisión de compra. Lo que tiene lógica y sentido, sobre todo porque al ser la base de datos personal (sin el bulksearch etc), los últimos 10 juegos que he comprado, han sido por menos de 15 euros.

5. Resolución del problema

Una vez terminado el estudio, se evalúa el modelo de random forest para predecir si un juego va a ser comprado o no, el cual es el objetivo de este estudio. Para ello, mediante una lista de 3 juegos su precio y su género correspondiente se predice el resultado.

```
predictions = rf.predict(games)
probabilities = rf.predict_proba(games)[:, 1]

titles = ['Sniper Elite 5', 'The Last of Us', 'Elden Ring']

for title, pred, prob in zip(titles, predictions, probabilities):
    status = "Bought" if pred == 1 else "Not Bought"
    print(f"{title}: {status} (Buy Probability: {prob:.2f})")
```

Purchase Predictions

- **Sniper Elite 5**: Bought (Buy Probability: 0.66)
- **The Last of Us**: Not Bought (Buy Probability: 0.22)
- **Elden Ring**: Not Bought (Buy Probability: 0.09)

Efectivamente el modelo realiza las predicciones de manera correcta, ya que efectivamente he comprado Sniper Elite 5, y los demás no.

Contribuciones

- **Firma**: Alex S P
- **Investigación previa**: Alex S P
- **Redacción de las respuestas**: Alex S P
- **Desarrollo del código**: Alex S P
- **Participación en el vídeo**: Alex S P
- **Repositorio**: https://github.com/aporraspa/TYCVD-PR2/tree/main