Proyecto Final BBDD

Miguel Martín Vieira 2ºB iMAT

202203898

Jorge Vančo Sampedro 2ºB iMAT

202200473

1.Introducción:

A lo largo de este proyecto se ha trabajo con los conjuntos de datos de Reviews de Amazon, disponibles en este enlace. Para el proyecto, se va a trabajar con los siguientes conjuntos de datos:

- Toys and Games 5-core (Toys_and_Games_5.json, 167,597 reviews).
- Video Games 5-core (Video_Games_5.json, 231,780 reviews).
- Digital Music 5-core (Digital_Music_5.json, 64,706 reviews).
- Musical Instruments 5-core (Musical_Instruments_5.json, 10,261 reviews)

Más tarde, en el punto 5, se añadirán:

- Amazon Instant Video 5-core (Amazon_Instant_Video_5.json, 37,127 reviews)
- Office Products 5-core (Office_Products_5.json, 53,258 reviews)

Este trabajo ha sido realizado por dos alumnos, Miguel Martín Vieira y Jorge Vančo Sanpedro, por tanto, el trabajo se ha dividido para realizarlo de manera independiente y luego juntarlo a posteriori.

A la hora de programa se ha usado el estilo Python Enhancement Proposals 8, (PEP-8) para una mayor legibilidad del código.

2.Diseño Y Carga De Datos

A la hora de decidir la manera de almacenar los datos hemos seguido el siguiente esquema relacional:

SQL:

REVIEWERS (reviewerId, reviewerName)

ITEMS (asin, type_id)

- type_id FK REFERS TYPES (id)

TYPES (id, type)

MONGODB:

{reviewerID, asin, helpful, summary, overall, type_id, reviewText, reviewTime}

Donde type_id es la clave primaria de TYPES.

Se ha elegido este esquema para obtener la mejor distribución posible de datos y aprovechar al máximo las características de cada base de datos. MongoDB, al ser una base de datos no estructurados, es perfecto para añadir campos de texto como summary. Además, como se tratan con grandes cantidades de reseñas y muchas queries, la alta velocidad de este es muy beneficiosa de cara a la implementación. En SQL se guardan valores para evitar algo de redundancia en MongoDB y proporcionar unas tablas normalizadas. El campo de reviewTime se ha calculado usando unixReviewTime, es por esto que no se sube ningún otro campo de fecha, pues ya se puede obtener toda la información que se requiera con este.

En el fichero load_data.py se muestra el proceso de carga de datos. Este programa recorre los ficheros guardados en el directorio especificado en configuracion.ini. Los datos de estos ficheros son distribuidos adecuadamente entre las bases de datos comentadas anteriormente. En estos ficheros hay reseñas a las que les falta el reviewerName. Estos reviewerIDs son guardados hasta el final de la ejecución por si son encontrados más adelante en algún documento.

Además, el programa se ha creado con el objetivo de facilitar al máximo la inserción de nuevos datos, por lo que no es necesario ejecutar otro script para ello. Si se desea crear las bases de datos desde cero, se debe establecer el valor de create_new_db a true en configuracion.ini. Por el contrario, si se requiere insertar nuevos datos a los ya existentes, se deberá poner el valor de false.

El fichero **configuracion.ini** es de la forma:

```
[MONGODB]
connection = mongodb://localhost:27017
collection = reviews_prueba
database = reviews_prueba

[SQL]
host = localhost
```

```
user = root
password = Mike314FireM!
database = reviews

[DATA_UPLOAD]
path = data_upload/
create_new_db = true

[NEO4J]
limite_usuarios_reviews = 30
fichero_similitud = similarity.txt
usuario = neo4j
password = aaaaaaaa
connection = neo4j://localhost:7687
max_cache_size = 30
```

Esto permite una fácil modificación de todos todos los campos necesarios para lo correcta conexión a las bases de datos y archivos.

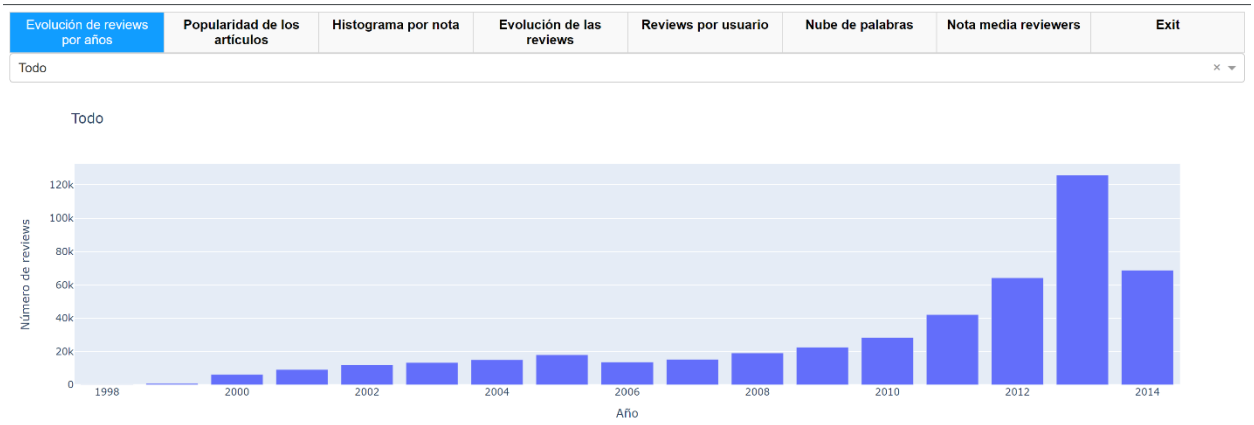
3. Aplicación Python Para El Acceso Y Visualización De Los Datos

- Mostrar la evolución de reviews por años

Para ello a través de los datos almacenados en MondoDB agrupamos las reviews por año, en concreto usando la siguiente query que modificamos en caso de que el usuario quiera solicitar todos los datos.

```
def Query_1_Evolucion_Reviews_Por_Año(tipo_review:str) -> list:
    """Devuelve el número de reviews por categoria y año
    Args:
        tipo_review (str): tipo de review a buscar
    Returns:
        list: lista de diccionarios con el año y el número de reviews de ese
año"""
    if tipo_review != "Todo":
        pipeline = [
            {"$match": {"type_id": tipo_review}},
            {"$group": {"_id": {"$year": "$reviewTime"}, "count": {"$sum": 1}}},
            {"$sort": {"_id": 1}},
        ]
```

```
    else:
        pipeline = [
            {"$group": {"_id": {"$year": "$reviewTime"}, "count": {"$sum": 1}}},
            {"$sort": {"_id": 1}},
        ]

    result = collection.aggregate(pipeline)

    return list(result)
```



- Evolución de la popularidad de los artículos:

Los datos con los que trabajamos en la gráfica es la suma de todas las reviews para cada artículo. Para ello la query utilizada en MongoDB es la siguiente. Con la que conseguimos que se nos de el asin del producto junto al número de veces que se ha hecho una review del mismo.

```
def Query_2_Evolucion_Popularidad_Articulos(tipo_review):
    """Devuelve el número de reviews por año
    Args:
        tipo_review (str): tipo de review a buscar
    Returns:
        list: lista de diccionarios con el año y el número de reviews de ese
año"""

    if tipo_review != "Todo":
        pipeline = [
            {"$match": {"type_id": tipo_review}},
            {"$group": {"_id": "$asin", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
        ]
    else:
```

```
        pipeline = [
            {"$group": {"_id": "$asin", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
        ]
    result = collection.aggregate(pipeline)

    return list(result)
```

- Histograma por nota

En este caso la agrupación que se lleva a cabo en MongoDB es con el Overall. La query de MongoDB es la siguiente:

```
def Query_3_Histograma_Por_Nota(asin=None, type_id=None):
    """Devuelve el número de reviews por nota
    Args:
        asin (str): asin del producto a buscar
        type_id (str): tipo de review a buscar
    Returns:
        list: lista de diccionarios con la nota y el número de reviews de esa
nota"""
    if asin is None or asin == "Todo":
        pipeline = [
            {"$group": {"_id": "$overall", "count": {"$sum": 1}}},
            {"$sort": {"_id": 1}},
        ]

    else:
        pipeline = [
            {"$match": {"asin": asin, "type_id": type_id}},
            {"$group": {"_id": "$overall", "count": {"$sum": 1}}},
            {"$sort": {"_id": 1}},
        ]
```

```
    result = collection.aggregate(pipeline)

    if result == None:
        print("No se encontraron datos")
        return []

    return list(result)
```

- Evolución de las reviews a lo largo del tiempo para todas las categorías

Cabe recalcar que en esta query la gestión de la fecha fue lo más tedioso pero tras múltiples pruebas se consiguió algo que cuadraba y nos permitía crear el histograma de una manera modular y efectiva.

```
def Query_4_Evolucion_Reviews_Tiempo_Todas_Categorias() -> list:
    """Muestra la evolución de las reviews a lo largo del tiempo
    Args:
        None
    Returns:
        list: lista de diccionarios con la fecha y el número de reviews de ese
día"""
    pipeline = [
        {
            "$group": {
                "_id": {
                    "year": {"$year": "$reviewTime"},
                    "month": {"$month": "$reviewTime"},
                    "day": {"$dayOfMonth": "$reviewTime"},
                },
                "count": {"$sum": 1},
            }
        },
        {"$sort": {"_id": 1}},
    ]
```

```
    # Ejecutar la operación de agregación
    result = list(collection.aggregate(pipeline))

    # Imprimir los resultados
    suma = 0
    for doc in result:
        count_dia = doc["count"]
        doc["count"] += suma
        suma += count_dia
        date = doc["_id"]
        doc["fecha"] = f"{date['year']}-{date['month']:02d}-{date['day']:02d}"
      return result
```

- Histograma de reviews por usuario

```
def Query_5_Reviews_Por_Usuario() -> list:
    """Devuelve el número de reviews por usuario
    Args:
        None
    Returns:
        list: lista de diccionarios de los usuarios con más reviews junto con el
número de reviews que tienen cada uno"""
    pipeline = [
        {"$group": {"_id": "$reviewerID", "count": {"$sum": 1}}},
        {"$group": {"_id": "$count", "number_of_users": {"$sum": 1}}},
        {"$sort": {"count": -1}},
    ]

    result = collection.aggregate(pipeline)

    return list(result)
```

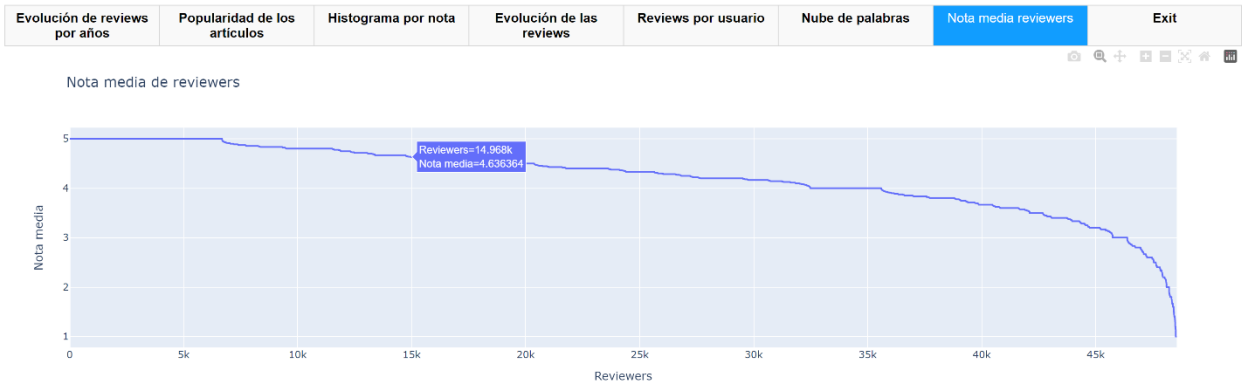
- Una opción para obtener una nube de palabras en función de la categoría

```
def Query_6_Nube_Palabras_Por_Categoria(tipo_review : str) -> Counter:
    """Devuelve el numero de palabras en los resúmenes de las reviews
    Args:
        tipo_review (str): tipo de review a buscar
    Returns:
        Counter: contador de palabras en los resúmenes de las reviews"""
    documentos = collection.find({"type_id": tipo_review})

    # Concatenar resúmenes
    resumenes = " ".join(doc["summary"] for doc in documentos)

    # Eliminar conectores y palabras cortas
    palabras = [
        clean_word(palabra.lower()) for palabra in resumenes.split() if
len(palabra) > 3
    ]
    frequencia_palabras = count_words(palabras)
    # for palabra, frequencia in frequencia_palabras.items():
    #     print(f"{palabra}, {frequencia}")
    return frequencia_palabras
```

- Analisis de los reviewers

Todas las reviews son relativas, cada cliente tiene un nivel de exigencia o unas espectativas, etc... Por eso mismo creamos esta query que analiza la nota media de cada uno de los reviewers. Con ello lo que buscamos es tener un parámetro, extremadamente sencillo, que nos permita analizar si un reviewer tiende a ser generoso o no. Como se puede ver en la gráfica de abajo, un 90% de los reviewers pone una nota media superior a 3.

```
def Query_7_Libre_Reviewers_Generosos() -> list:
    """La nota media que un reviewer pone a las cosas que valora
    Args:
        None
    Returns:
        list: lista de diccionarios con el reviewer y la nota media que pone a
las cosas que valora"""

    pipeline = [
        {"$group": {"_id": "$reviewerID", "averageRating": {"$avg":
"$overall"}}},
        {"$sort": {"averageRating": -1}},
    ]

    result = collection.aggregate(pipeline)

    return list(result)
```

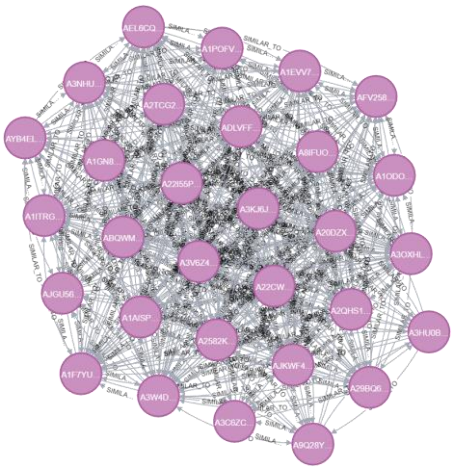


Se decidió crear un dashboard para la visualización de los datos usando la librería de python dash. El programa se puede probar ejecutando dashboard.py. Se mostrará por pantalla la dirección en la que se encuentra hosteado el dashboard. En él, hay múltiples pestañas con las distintas gráficas pedidas, las cuales son creadas con los datos obtenidos de las queries mencionadas anteriormente.

4. Aplicación Python Junto Neo4J

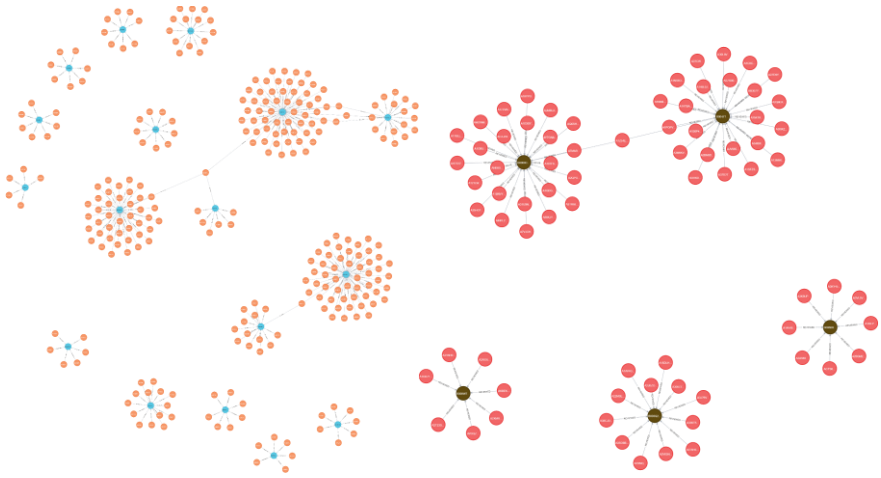
4.1 Obtener similitudes entre usuarios y mostrar los enlaces en Neo4J

Para crear este gráfico son necesarias 170 líneas de código y 7 funciones que se encargan de obtener los artículos con mayor número de reviews, evitar duplicados, calcular la similitud de jaccard y generar un documento de texto con dicha información que después es lanzado a Neo4j y visualizado ahí.



4.2 **Obtener enlaces entre usuarios y artículos.**

Para generar este gráfico es importante procesar la información a través de Python ya que se tienen que encontrar las relaciones y las propiedades que tendrán estas antes de cargar los datos en Neo4j. Es por esto mismo que es necesario el uso de una clase en la que generar las relaciones y una vez los elementos han sido correctamente guardados en el documento de resultados_reviews.txt se puede abrir con una segunda función ese mismo documento y cargar desde ahí los datos a Neo4J. La primera imagen muestra la información para 30 artículos aleatorios de la base de datos de Video_Game y la segunda muestra 5 artículos aleatorios en la base de datos de Toys_and_Game.



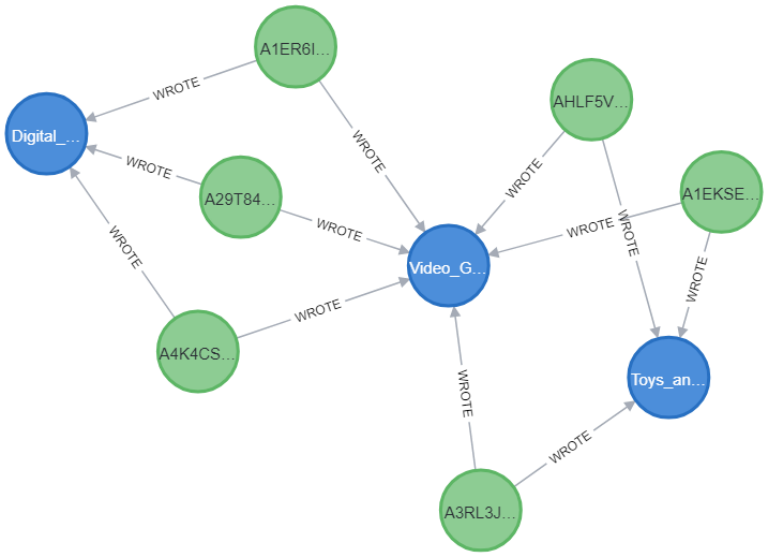
4.3 Obtener algunos usuarios que han visto más de un determinado tipo de artículo

```
def apartado_4_3(connection, collection : Collection, driver) -> None:
    """Carga en Neo4j los usuarios que han escrito a más de dos tipos de
productos distintos

    Args:
        connection: conexión a la base de datos
        collection (Collection): colección de MongoDB
        driver: driver de la conexión a Neo4j

    Returns:
        None (aunque carga datos en la base de datos de Neo4j)
    """

    sql = """
            SELECT reviewerID, reviewerName
            FROM reviewers
            ORDER BY reviewerName
            LIMIT 400;
        """
```

```
    neo4j_query = """
                MERGE (reviewer: User {user_id: $reviewer_id, reviewer_name:
$reviewer_name})
                MERGE (type: ProductType {product_type: $product_type})
                CREATE (reviewer) - [:WROTE {number_of_articles: $count}] ->
(type)
                """

    cursor = connection.cursor()
    cursor.execute(sql)
    reviewer_names = cursor.fetchall()

    # Se consigue un diccionario de los ids y nombres de productos
    product_types = dict(get_product_types())

    with driver.session() as session:
        # Se recorren los reviewers
        for id, name in reviewer_names:
            docs = collection.aggregate(
                [
                    {"$match": {"reviewerID": id}},
                    {"$group": {"_id": "$type_id", "count": {"$sum": 1}}},
                ]
            )
            docs = list(docs)

            # Si han escrito a más de dos tipos de productos distintos
            # Se suben las relaciones a Neo4j
            if len(list(docs)) >= 2:
                for doc in docs:
                    product_type = product_types[doc["_id"]]
                    count = doc["count"]
                    # Se ejecuta la query
                    session.run(
                        neo4j_query,
                        reviewer_id=id,
                        reviewer_name=name,
                        product_type=product_type,
                        count=count,
                    )
    print("Se ha finalizado la carga en Neo4j")
```

El código que se puede ver en la parte superior permite encontrar los usuarios que han llevado a cabo reviews en diferentes categorías. Para ello se hace uso de búsquedas tanto en MongoDb como en Neo4j,

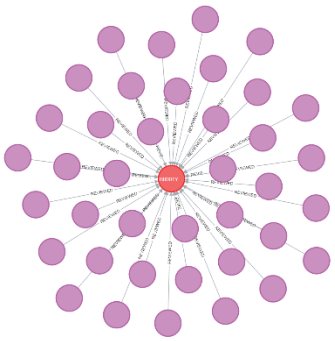
lo que nos permite agilizar el proceso y llevar a cabo las operaciones en las bases de datos en las que las consultas son más cómodas. Principalmente con el objetivo de acabar mostrando toda la información en Neo4J de manera ordenada y estructurada.



4.4 Artículos populares y artículos en común entre usuarios

En la imagen se muestra un ejemplo de lo que presenta para **un** artículo en la consulta 4.4, para ello y con los datos en MongoDB accede a los 5 artículos que cumplen las condiciones dadas por el enunciado (los 5 primeros que no tengan 40 reviews) y acumula su asin y los reviewersIDs relacionado con ellos en el documento resultados_reviews_2. Desde allí una vez más se lanza una función que sube los datos a Neo4J. Una vez con los datos en Neo4J ejecutamos la query que da lugar a la lista que encontramos a la derecha y que muestra los reviewers que presentan artículos en común.

```
MATCH (u1:User)-[:REVIEWED]->(a:Article)<-[:REVIEWED]-(u2:User)
WHERE id(u1) < id(u2)
WITH u1, u2, COUNT(a) AS sharedReviews
WHERE sharedReviews > 1
RETURN u1.reviewerID AS User1, u2.reviewerID AS User2, sharedReviews
ORDER BY sharedReviews DESC
```



```
User A5C9UZCS7X6DO and User APN6DO7VHDLTN have 1 shared reviews.
User A1QWGPBP61DPQP and User APN6DO7VHDLTN have 1 shared reviews.
User A29ECIGILCOG5Y and User APN6DO7VHDLTN have 1 shared reviews.
User APLQ0Q4VXS3J9 and User APN6DO7VHDLTN have 1 shared reviews.
User A33D1LPCYUANWE and User APN6DO7VHDLTN have 1 shared reviews.
User A2TAPL67U2A5HM and User APN6DO7VHDLTN have 1 shared reviews.
User A3HNTTBLZS3L1O and User APN6DO7VHDLTN have 1 shared reviews.
User A3KJ6JAZPH382D and User APN6DO7VHDLTN have 1 shared reviews.
Se han encontrado 3705 parejas de reviewers que tienen reviews en común
```

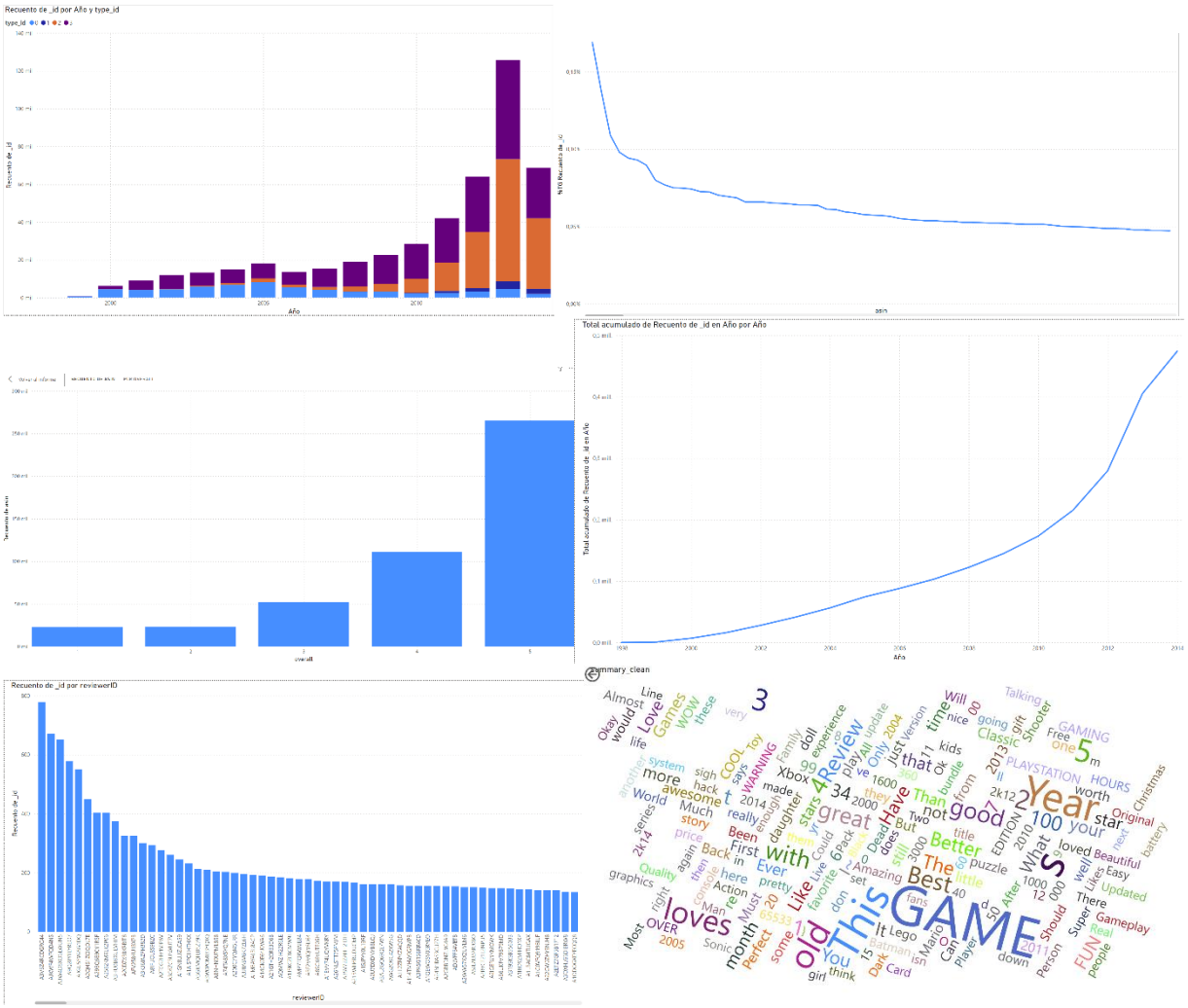
5. Nuevos Datos

Esta función implementada de carga de datos quisimos llevarla a la excelencia creando una versión modular que parte del mismo archivo que la carga de datos original. El único procedimiento que se debe llevar a cabo es cargar tus documentos en una carpeta que se deberá indicar en configuracion.ini y cambiar create_new_db a false en el fichero de configuracion.ini.

Esta función tendrá en cuenta las categorías actuales con las que cuenta tu base de datos, los asins, etc… y generará un nuevo identificador para la categoría recién creada. También consigue evitar la duplicación de datos mediante funciones de checkeo, etc… Es importante destacar que se debe tener en cuenta el path que está configurado en el documento de configuración.ini.

6. Más Visualización Y Relación Con Machine Learning.

Haciendo uso de PowerBI, se han replicado, en la medida de lo posible, las gráficas hechas anteriormente con python. En algunos casos, para que la imagen pudiese salir bien, se tuvo que prescindir del uso de la leyenda, pues esta hacía que se estratificasen los datos y llegasen a ser incomprensibles algunos gráficos. Dado que la gráfica original empieza por muchísimos datos con overall medio de 5, no se consiguió visualizar bien en PowerBI. Es por ello que se realizó la nube de palabras en vez de esa gráfica.



7. Opcional: Proyecto bases de datos. Implementación de un pequeño modelo de Machine Learning.

Como primer modelo, se trató de usar PCA para completar los valores de una matriz que contenía los ratings de todos los usuarios. Sin embargo, a pesar de tener un código completamente funcional, no se pudo realizar con una matriz tan grande y con tantos valores faltantes. Debido a este problema, se decidió hacer uso de KNN para realizar la estimación de los ratings que daría un usuario.

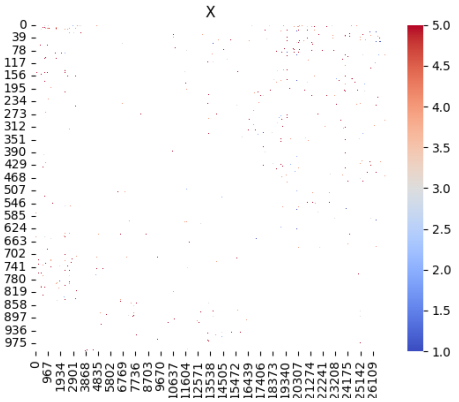
La idea bajo el modelo es usar la similitud de jaccard, la cual se calculó anteriormente para elegir a los usuarios que tienen algún artículo en común con el que se va a tratar de completar. De entre estos, se cogen a los 20 reviewers más similares al objetivo. Es decir, que sus votos son similares. El ranking que le daría el reviewer se calcula como la media que han votado estos 20 reviewers más cercanos.

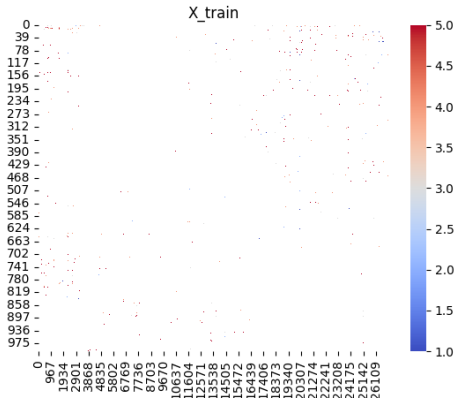
Se ha comprobado el funcionamiento de este algoritmo subiendo a neo4j las similitudes entre 1000 reviewers y tratando de completar las votaciones a todos los productos. Con las similitudes, se crea una matriz de similitudes que contiene las similitudes de cada usuario con cada usuario. Además, se crea la matriz de ratings que tiene por columnas a los productos y por filas a los usuarios. Cada entrada de esta matriz es el rating que le da el usuario al producto. Es esta matriz la que se trata de completar usando métodos de machine learning. Tras obtener todos los datos de neo4j, se crea un dataset de entrenamiento quitando el 10% de los valores de la matriz creada. Este 10% es luego usado para comprobar el rendimiento del modelo.

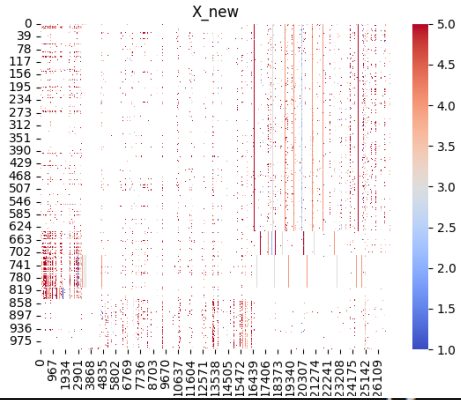
Se han observado muy buenos resultados con este modelo. De media, se completan un 58% de los votos de un reviewer, lo cual es más que suficiente para poder recomendar productos. Además, el modelo se equivoca en promedio tan solo 0.58 puntos del valor real del rating. Este valor es bastante cercano a 0, lo que permitirá dar recomendaciones muy acertadas.

Las matrices creadas para entrenar el modelo se figuran en las siguientes imágenes. Se puede observar lo dispersas que es la matriz de los ratings (reviewers en las filas y productos en las columnas). La gran mayoría de las entradas son valores nulos. En cambio, tras usar el modelo de machine learning, se puede observar cómo se ha completado parte de la matriz. Las regiones con más valores completados son de usuarios que deben de tener bastantes reviews en común.

Todo el código de esta parte se encuentra en recommender.py. Se debe subir primero las similitudes a Neo4J, de donde se obtienen todos los datos para entrenar el modelo. Este modelo se podría ejecutar en batch todos los días para obtener un listado de las películas a recomendar a cada usuario sin perjudicar el rendimiento de la aplicación, pues la inferencia puede ser lenta con tantos datos.








```
Número de reviewers obtenidos: 1000
Número de productos: 27064
Se van a tapar 7005 ratings
Porcentaje de no imputados: 57.87%
MAE: 0.587
```