Facultad de Ingeniería de la Universidad de Buenos Aires

Ingeniería en informática

Materia: **Base de Datos**

Año: 2024, 1er Cuatrimestre

Grupo:

Integrantes:

Alumno	Padrón	Mail
Juan Biancuzzo	106005	jbiancuzzo@fi.uba.ar
Clara Ruano Frugoli	106835	cruano@fi.uba.ar
Francisco Ezequiel Martínez	108460	femartinez@fi.uba.ar
Ramiro Fernandez	105595	rfernandezm@fi.uba.ar
Valentina Lanzillotta	108257	vlanzillotta@fi.uba.ar

Introducción **3**
Análisis Exploratorio **4**
Caso 1 4
Caso 2 5
Caso 3 5
Workflow **7**
Paso 1: Limpieza de datos y verificación de la integridad de los mismos 7
Paso 2: Carga en la base de datos 8
Apéndice **9**

Introducción

El objetivo de este trabajo práctico es desarrollar un workflow eficiente para la limpieza y carga de datos en una base de datos SQLite3. Este proceso permitirá transformar datos crudos en información estructurada y accesible, lista para análisis posteriores. Para este propósito, utilizaremos un conjunto de datos que contiene una serie de reseñas de la aplicación de Netflix, el cual se puede encontrar en el apéndice.

Para lograrlo, primero realizamos un análisis exploratorio de los datos, usando la biblioteca Pandas con Python. En el presente informe se pueden visualizar gráficos y conclusiones derivados de dicho análisis. Un link al colab se puede encontrar en el apéndice.

Nuestro workflow presenta 2 pasos claves: limpieza y carga de datos. Para la limpieza de datos se eligió realizar un programa en Rust, ya que es un lenguaje de programación altamente eficiente para este propósito. Para la carga de datos, se diseñaron las tablas y se realizó un programa en Python que realiza la carga de los datos ya limpios. Es decir que la entrada del segundo programa es la salida del primero. Esto permite que se pueda correr solo la parte del programa donde hubo fallas (en caso de haberlas). También permite que se agreguen pasos intermedios al workflow en caso de ser necesario, encadenando salidas, siempre y cuando se respeten las condiciones de entrada y salida propuestas por los módulos actuales.

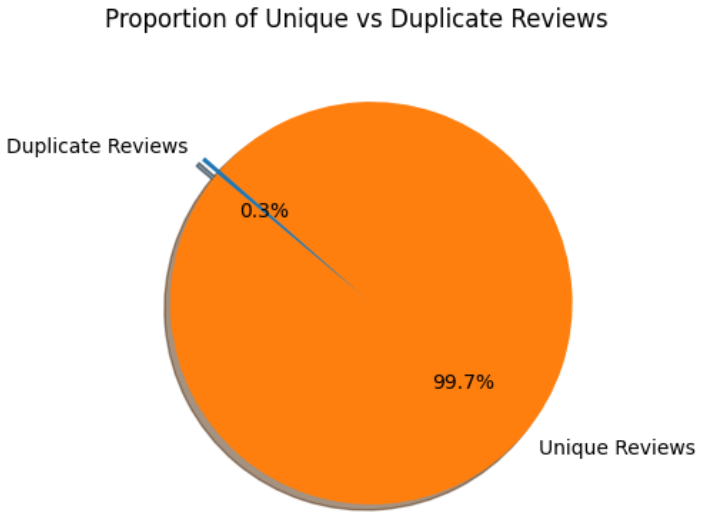
Para la carga de datos en la base de datos se eligió Python por su simplicidad. Se evaluó el rendimiento y se llegó a la conclusión de que utilizar Python no agrega demasiado tiempo adicional a la tarea, el tiempo de demora de la tarea ocurre mayormente por el uso de consultas SQL.

Para el desarrollo del presente informe, se utilizó ChatGPT como herramienta para corregir y mejorar las explicaciones en algunos puntos, así como para realizar correcciones mínimas en el código.

Análisis Exploratorio

Un primer análisis de los datos puede encontrarse en el apéndice. A continuación se detalla lo recopilado a partir de la exploración y cuáles fueron las decisiones tomadas respecto de la limpieza de datos.

El siguiente gráfico ilustra la cantidad porcentual de valores únicos respecto del total de valores. Esto significa que hay claves que no son únicas.



A continuación detallamos un análisis de distintos casos de repetición y las decisiones t tomadas para la limpieza y verificación de la integridad de los datos.

Caso 1

Por un lado, encontramos el caso en el que el `reviewId` se repite. El usuario es el mismo, pero cambia la versión y la fecha de la review. Tiene la misma cantidad de thumbs up, lo que nos indicaría que podría ser una re-edición del comentario.
Esto sucede, por ejemplo, para el id `700110d1-56f7-42d3-a17c-d1be8990ae7b`.

	reviewId	userName	content	score	thumbsUpCount	reviewCreatedVersion	at	appVersion
6	700110d1-56f7-42d3-a17c-d1be8990ae7b	Trevor S	Buggy. Not only do subtitles frequently show o...	3	39	8.115.2 build 6 50688	2024-05-28 21:40:46	8.115.2 build 6 50688
6019	700110d1-56f7-42d3-a17c-d1be8990ae7b	Trevor S	Not only do subtitles frequently show only a f...	3	39	8.101.1 build 7 50601	2024-01-31 01:24:22	8.101.1 build 7 50601

Evidentemente en este caso hubo una modificación del comentario para una versión más nueva de la aplicación. Cuando este sea el caso, nos quedaremos con ambos datos.

Caso 2

Por otro lado, tenemos casos de id repetidos donde el usuario pasa a ser anónimo, tal como sucede con el id `783f611f-6f2a-4bd3-8a0e-567cf1152721`. En este caso nos quedamos con ambos datos.

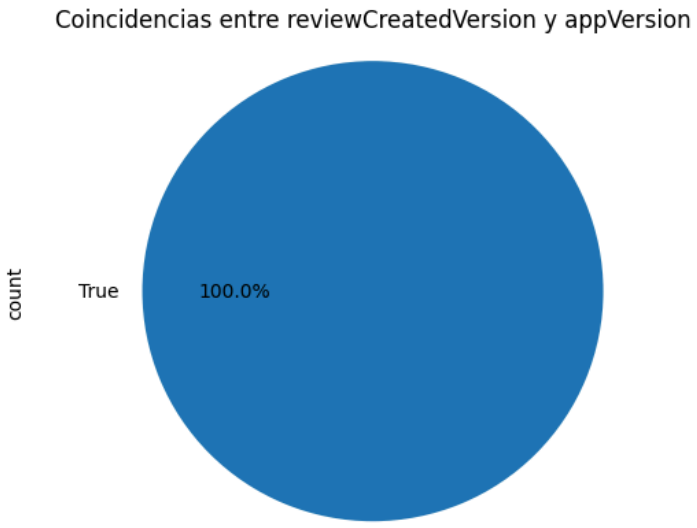
	reviewId	userName	content	score	thumbsUpCount	reviewCreatedVersion	at	appVersion
1348	783f611f-6f2a-4bd3-8a0e-567cf1152721	Maureene Bermejo (Gigi)	Very well satisfied.. Best app and good qualit...	5	3	8.50.0 build 9 50318	2024-05-21 10:06:34	8.50.0 build 9 50318
108920	783f611f-6f2a-4bd3-8a0e-567cf1152721	A Google user	Very well satisfied.. Best app and good qualit...	5	3	7.26.1 build 27 34479	2019-10-02 13:39:32	7.26.1 build 27 34479

Caso 3

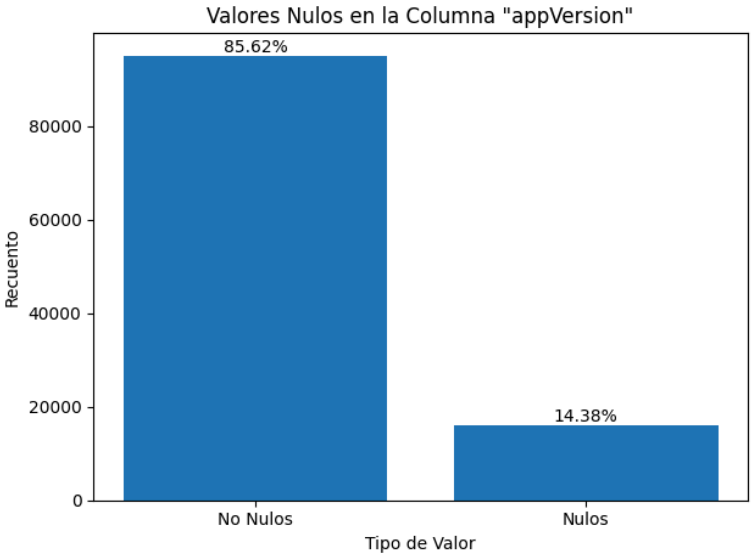
Hay un tercer caso, como ocurre con el id `ef3c7dcf-587d-44ed-9b2e-ef000115fe6d`, donde lo único que cambia es la fecha y la `app version`, pero todo lo demás se mantiene. Como este caso no aporta mucha información de valor, eliminaremos el comentario más viejo.

	reviewId	userName	content	score	thumbsUpCount	reviewCreatedVersion	at	appVersion
12	ef3c7dcf-587d-44ed-9b2e-ef000115fe6d	Mark "The Dr." Steward	The wife and I absolutely love Netflix. Been w...	5	37	8.52.2 build 14 50335	2024-05-28 20:44:38	8.52.2 build 14 50335
12258	ef3c7dcf-587d-44ed-9b2e-ef000115fe6d	Mark "The Dr." Steward	The wife and I absolutely love Netflix. Been w...	5	37	7.71.0 build 9 35074	2020-09-01 08:18:20	7.71.0 build 9 35074

Terminado el análisis de la columna de `reviewId`, notamos que parecería haber una coincidencia entre las columnas `reviewCreatedVersion` y `appVersion`, ya que en Kaggle nos indica 'app version' en ambas y la cantidad de nulos coinciden.
Para poder hacer el análisis, se reemplazan los nulos por 0 y se verifica, a través del siguiente gráfico, que efectivamente las columnas son coincidentes. Luego, procedemos a eliminar la columna `reviewCreatedVersion`, que tiene un nombre menos claro.



En cuanto a los nulos, evidentemente vemos que existe un 14.38% de casos donde la app versión no se pudo verificar.



Al representar un porcentaje alto sobre el total de los datos, decidimos dejarlos ya que sigue siendo información útil. Le vamos a dar una categoría especial: versión ‘0.0.0 build 0 0’. Como estos datos van a tener una fecha válida, siguen siendo útiles para un potencial análisis sobre la evolución de las reseñas de la aplicación a través del tiempo.

Workflow

Paso 1: Limpieza de datos y verificación de la integridad de los mismos.

Teniendo en cuenta las consideraciones del análisis exploratorio de datos, el primer paso de nuestro workflow consistirá en un programa escrito en Rust que realizará las siguientes tareas

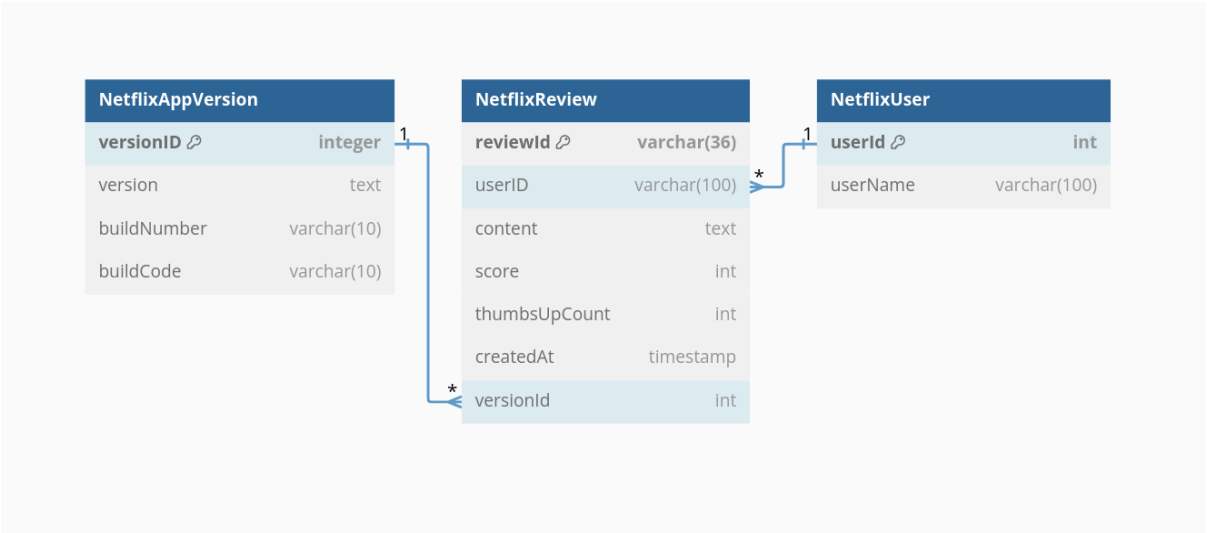
- Reemplazo de valores nulos en app versión por '0.0.0 build 0 0', ya que representan una parte importante de los datos (15%) y consideramos que se gana más de lo que se pierde guardando la información. También, con los datos ya guardados en la base de datos, se podría hacer algún análisis con las fechas de las reviews para poder rellenar inteligentemente aquellas reviews que no tienen versión.
- Verificación de formatos:
    - Las fechas deben ser del estilo YYYY-MM-DD HH:MM:SS.
    - La `app_version` debe tener el siguiente formato: [numero].[numero].[numero] build [numero] [numero]. Por ejemplo: 8.76.0 build 8 50457.
    - `Score` y `ThumbsUpCount` son columnas que deben tener valores mayores o iguales que 0.

  Cualquier dato que no cumpla con el formato, será eliminado del conjunto de datos.
- Eliminación de datos duplicados. Una review está duplicada cuando su uuid (id único que identifica la review) aparece más de una vez. Existen varios casos donde esto puede ocurrir, como fue explicado en el análisis exploratorio. Se decide conservar siempre las más recientes.

Para realizar esta tarea, se desarrolló un programa en Rust, un lenguaje conocido por su eficiencia y seguridad en la gestión de memoria. Rust es capaz de manejar grandes volúmenes de datos con un alto rendimiento y menor riesgo de errores en comparación con otros lenguajes como Python. Esto es especialmente beneficioso para la limpieza y verificación de datos, donde la velocidad y la precisión son cruciales para garantizar la integridad del conjunto de datos.
Se optó por manejar el archivo sin cargar todas las filas en memoria a la vez. Para esto se hacen dos pasadas, la primera para identificar uuids duplicados y la segunda para escribir en un nuevo archivo aquellas filas que resultan ser válidas.

Paso 2: Carga en la base de datos.

Se levantará una base de datos SQLite3. Si la base de datos no existe, se podrá levantar mediante un script. Si existe, se utilizará la existente. La base de datos presentará el siguiente formato que se encuentra en 3FN (Tercera forma normal)



Con esta distribución de los datos se evita repetir valores de versiones, ya que solo hay 350 valores distintos aproximadamente. Además se pueden obtener fácilmente datos sobre reviews de determinadas versiones.

Se optó por conservar el `reviewID`, que es un uuid, como primary key para evitar añadir reviews repetidas.

En este paso se decidió utilizar Python debido a que no perjudica la performance. El costo más importante en este paso es la inserción en la base de datos SQLite, hacer uso de Rust no resultaba en una mejora notable en el tiempo que tarda este proceso por las características del problema ya que se necesita hacer una inserción secuencial de cada uno de los datos.

Apéndice

Conjunto de datos: Netflix Reviews - Kaggle

Repositorio de GitHub: Repositorio del TP

Colab con análisis exploratorio: Análisis Exploratorio