

**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

***La Universidad Católica de Loja***

**ÁREA TÉCNICA**

**INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN**

**PRACTICUM 1.1**

**Por:**

Joan Manuel Briceño Quilambaqui

Alex Fernando Aguirre Rojas

**Director:**

Danilo Rubén Jaramillo Hurtado

**Tema:**

Análisis del Proyecto Final

Loja – Ecuador

Oct 2019 – Feb 2020

**Índice**

[Resumen v](#_Toc31271140)

[Problemática vi](#_Toc31271141)

[Objetivos vii](#_Toc31271142)

[General: vii](#_Toc31271143)

[Específicos vii](#_Toc31271144)

[Metodología viii](#_Toc31271145)

[Introducción 1](#_Toc31271146)

[Marco Referencial 2](#_Toc31271147)

[Antecedentes 2](#_Toc31271148)

[Definición de Términos 2](#_Toc31271149)

[Programación funcional 2](#_Toc31271150)

[Lenguaje de programación Scala. 3](#_Toc31271151)

[Base de datos. 3](#_Toc31271152)

[SQL. 4](#_Toc31271153)

[Modelo funcional. 4](#_Toc31271154)

[Vega-Lite. 5](#_Toc31271155)

[HTML. 5](#_Toc31271156)

[Solución 6](#_Toc31271157)

[Aplicación de Base de Datos 6](#_Toc31271158)

[Aplicación de programación funcional 23](#_Toc31271159)

[Validación 30](#_Toc31271160)

[Bibliografía 36](#_Toc31271161)

[Anexos 37](#_Toc31271162)

**Índice de figuras**

[Gráfico 1 Determinar las filas duplicadas 9](#_Toc31270980)

[Gráfico 2 Ordenar las filas duplicadas 9](#_Toc31270981)

[Gráfico 3 Filas duplicadas ordenadas 10](#_Toc31270982)

[Gráfico 4 Vaciamos hacia abajo 11](#_Toc31270983)

[Gráfico 5 Eliminamos las filas duplicadas 11](#_Toc31270984)

[Gráfico 6 Archivo sin datos duplicados 12](#_Toc31270985)

[Gráfico 7 Variaciones en el "user\_lang" 13](#_Toc31270986)

[Gráfico 8 Variaciones suprimidas 14](#_Toc31270987)

[Gráfico 9 Descargamos el archivo 15](#_Toc31270988)

[Gráfico 10 Importación de datos desde el csv 15](#_Toc31270989)

[Gráfico 11 Detección de los tipos de datos 16](#_Toc31270990)

[Gráfico 12 Datos cargados 16](#_Toc31270991)

[Gráfico 13 Modelado Lógico 19](#_Toc31270992)

[Gráfico 14 Creamos un usuario sysdba para la base de datos 20](#_Toc31270993)

[Gráfico 15 Tweets y Retweets por día. 30](#_Toc31270994)

[Gráfico 16 Tweets y ReTweets por hora. 30](#_Toc31270995)

[Gráfico 17 Aplicaciones más utilizadas para la publicación de Tweets. 31](#_Toc31270996)

[Gráfico 18 Distribución de Hashtags. 31](#_Toc31270997)

[Gráfico 19 Distribución de Menciones. 32](#_Toc31270998)

[Gráfico 20 Distribución de URLs. 32](#_Toc31270999)

[Gráfico 21 Distribución de Media. 33](#_Toc31271000)

[Gráfico 22 ¿Existe correlación entre amigos y seguidores? 33](#_Toc31271001)

[Gráfico 23 Comportamiento de Usuarios. 34](#_Toc31271002)

[Gráfico 24 Comportamiento de Menciones. 34](#_Toc31271003)

**Índice de tablas**

[Tabla 1 Diccionario de datos 6](#_Toc31265719)

# Resumen

El presente proyecto se centra en responder a diversas preguntas mediante la utilización de una gama de conocimientos adquiridos durante el ciclo, correspondientes a las materias de Fundamentos de Base de Datos, Programación Funcional y Reactiva, así como Prácticum 1.1, las incógnitas a responder serán las siguientes: Número de Tweets y Retweets por día, número de Tweets y Retweets por hora, aplicaciones más utilizadas para publicar Tweets, distribución de Hashtags, distribución de menciones, distribución de URL, distribución de media, ¿Existe una correlación entre el número de amigos y la cantidad de seguidores?, comportamiento de los usuarios; a su vez por cada usuario se debe presentar: la cantidad de seguidores y de amigos, también el número de Tweets y Retweets, y cuántas veces se ha mencionado a un usuario.

# Problemática

Las redes sociales han tomado un papel fundamental en la vida cotidiana de la mayoría de personas, así como Facebook se usa para el disfrute y la distribucion de los conocidos “memes” e Instagram para la publicación de fotos, la red social Twitter ha pasado a ser una de las redes sociales donde la mayoría de gente expresa opiniones a partir de hechos existentes en el mundo real, y a partir de esto se generan debates, produciendo respuestas a través de otros “tweets” o bien si alguien apoya algún punto de vista se aplican “Retweets” a publicaciones ya realizadas. Es debido a esto que existen gran cantidad de datos, en todo momento recopilados mediante diversas aplicaciones, como por ejemplo la Interfaz de Programación de Aplicaciones (API por sus siglas en ingles), del mismo Twitter. Teniendo en cuenta esto, surge una necesidad de poder lograr un análisis exploratorio de una parte de este conjunto de datos, el cual nos permita reducir, resumir y limpiar los datos hasta sus características principales, obteniendo diversos resultados esperados.

# Objetivos

## General:

* Realizar un análisis exploratorio de datos de un data set

## Específicos

* Construir un modelo relacional de base datos que represente a las entidades que se

encuentran presentes en el dataset.

* Construir un conjunto de consultas SQL para extraer datos almacenados en la base

de datos con el fin de proporcionar información relevante.

* Utilizar conceptos de programación funcional para realizar análisis exploratorio de

datos.

* Construir un conjunto de visualizaciones que permitan ver el resultado de los

análisis realizados.

# Metodología

Para el tercer ciclo de la carrera en Ciencias de la Computación se programó un Proyecto Integrador que integra a las materias de: Prácticum 1.1, Programación Funcional y Reactiva y Fundamentos de Base de Datos, para lo cual se pidió a los estudiantes formar grupos de dos personas a los cuales se le asignará un data set que deberá analizar y cumplir con los objetivos esperados.

La metodología ideal para el presente caso de estudio es el método deductivo, debido a que a partir del documento ods\_1\_2.csv se deberá analizar las entidades y sus atributos y en base a este análisis de procederá a identificar las relaciones específicas entre los campos que se manejan en redes sociales, en este caso en Twitter con el tema de: Objetivos de Desarrollo Sostenible o por sus siglas (ODS); se llevara un control de esto a través del repositorio en GitHub y su wiki (https://github.com/functional-programming-utpl-cc/proyectointegrador-testigos-de-timo).

# Introducción

La importancia de este proyecto integrador reside en, nosotros como estudiantes, prepararnos a futuro para poder afrontar las diversas situaciones de la vida real que nos encontraremos aplicando la carrera seguida, ya que deberemos de ser capaces de manejar a placer conjuntos de datos de este tamaño o incluso superiores, además de usar las bases de programación funcional y consultas SQL para poder obtener datos específicos de una manera rápida y eficaz.

Este proyecto insta a aplicar todos los conocimientos de tercer ciclo de la carrera en Ciencias de la Computación, así, en resumen, se deberá, a partir de un dataset normalizarlo para que llegue a ser una base de datos manejable, y así mismo este dataset deberá ser manejado mediante programación funcional para llegar a obtener los resultados requeridos.

# Marco Referencial

## Antecedentes

A medida que las popularidad de las redes sociales han ido en aumento, la cantidad de información que se almacena en las mismas igual ha ido en aumento, tal es el caso de Twitter, donde la escritura de "Tweets", la invención de "Hashtags" y demás servicios que ofrece la misma la han llevado hasta tal punto de fama donde se crean bases de datos bien mensuales, bien anuales, para recopilar y almacenar conjuntos de datos y que estos puedan ser reutilizados por los mismo usuarios de la aplicación en una fecha distinta a la publicación del "Tweet".

## Definición de Términos

### Programación funcional

La programación funcional es solo un paradigma de programación, es decir, es una manera en la cual podemos resolver un problema. Para García (2019) “Cuando nos encontramos desarrollamos software utilizando este paradigma, estaremos trabajando principalmente con funciones, evitaremos los datos mutables, así como el hecho de compartir estados entre funciones”. También la programación funcional o FP (Funtional Programming) por sus siglas en inglés puede ayudarnos a crear software más robusto, fácil de sostener y testear. Existen lenguajes que trabajan con programación funcional como son: Scala, Haskell, Scheme o Lisp e incluso java en su versión 8 se puede utilizar funciones sin abandonar su orientación a objetos (Becerro, 2016).

En este paradigma encontraremos conceptos como:

* **Funciones puras:** Son funciones que dado un valor de entrada siempre retorna el mismo de salida y estas no tienen efectos secundarios.
* **Composición de funciones:** Nos dice que se puede combinar dos o más funciones, y así ejecutarlas secuencialmente.
* **Mutabilidad:** Una característica de la programación funcional es que sus valores son inmutables, es decir, su valor no cambia una vez ha sido creado.
* **Efectos colaterales:** La programación funcional se basa en que las funciones no deberían tener efectos colaterales ya que estos crean cambios de estados en la aplicación fuera de la función llamada. (Pérez, 2019)

### Lenguaje de programación Scala.

**“**Scala es un lenguaje de programación moderno y multiparadigma diseñado para expresar patrones de programación comunes de una forma concisa, elegante, y de tipeado seguro. Integra fácilmente características de lenguajes orientados a objetos y funcionales” (Scala, 2018).

### Base de datos.

Se trata de un archivo o conjunto de archivos donde se almacenan ciertos datos de una forma específica, para que después los mismo puedan ser consultados, editados o eliminados según sea necesario (Borges, 2019). Existen varios tipos de bases de datos a su vez, como, por ejemplo:

- Bases de datos relacionales.

- Bases de datos orientadas a objetos.

- Bases de datos distribuidas.

- Almacenes de datos.

- Bases de datos NoSQL.

- Bases de datos orientadas a grafos.

- Bases de datos OLTP.

Estas son definidas como de función general, sin embargo, existen bases de datos para funciones específicas, dando de ejemplo los siguientes:

- Bases de datos de código abierto.

- Bases de datos en la nube.

- Base de datos multimodelo.

- Bases de datos de documentos/JSON.

- Bases de datos de autogestión.

### SQL.

Structured Query Language o “SQL” por sus siglas en inglés, (Prado, 2016) lo define como “un lenguaje estándar e interactivo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas, gracias a la utilización del álgebra y de cálculos relacionales”. SQL nos sirve para realizar consultas con la finalidad de obtener información de nuestra base de datos, este lenguaje de comandos hacer diversas acciones como: seleccionar, insertar, actualizar, etc.

### Modelo funcional.

Modelo funcional: se centra simple y llanamente en hacer de un conjunto de datos una base de datos manejable, limpiando impurezas de este, cómo podrían ser datos repetidos, valores en blanco o nulos u otros caracteres desconocidos que no permitirían manejar los datos de manera correcta.

### Vega-Lite.

Vega-Lite es una gramática de alto nivel que nos sirve para modelar graficas interactivas, ésta proporciona un soporte a la sintaxis de JSON, al estar en una forma declarativa se puede proporcionar cualquier especificación que desee que incluya la visualización a incluir, en términos de datos, marcas gráficas y canales de codificación, en lugar de tener que especificar cómo implementar la visualización en términos de bucles, comandos de dibujo de bajo nivel, etc. (UW Interactive Data Lab, 2016).

### HTML.

HTML o (HyperText Markup Language) es un lenguaje de marcas de hipertexto basado en SGML (Standard Generalized Markup Language) y es el formato estándar de los documentos de la web, normalizado por la W3C (World Wide Web Consortium) que es la encargada de normalizar los estándares y expansión de la web. (Lapuente, 2018)

# Solución

## Aplicación de Base de Datos

Primero se nos otorgó a cada grupo un dataset con 18000 filas separado por comas y que posee 18 columnas. La descripción de cada columna se muestra en la tabla 1:

Tabla 1 Diccionario de datos

|  |  |  |  |
| --- | --- | --- | --- |
| **Columna** | **Tipo** | **Descripción** | **Ejemplo** |
| id\_str | String | Representa al identificador único de un Tweet | 1038336680929964032 |
| from\_user | String | El nombre del usuario, tal como lo ha definido | MarielaAyerve |
| Text | String | El texto, en UTF-8, de la publicación. | RT @Salud\_CZ7: #Loja | Cristian Abendaño, psicólogo clinico, aborda sobre la prevención del suicido. Enfatiza como factores de protección la.comunicación, resiliencia, habilidades sociales, tiempo de calidad @lahoraecuador https://t.co/ CCAw6qB6SD |
| created\_at | String | Fecha UTC cuando se creó este Tweet | Sat Sep 08 08:02:10 +0000 2018 |
| Time | DateTime | Fecha | 08/09/2018 08:02:10 |
| geo\_coordinates | String | Puede ser nulo. Representa la ubicación geográfica de este Tweet según lo informado por el usuario o la aplicación del cliente. | loc: 18.501,-69.917 |
| user\_lang | String | Idioma con el que el usuario configuró su cuenta. | es |
| in\_reply\_to\_user\_id\_str | String | Puede ser nulo. Si el Tweet es una respuesta, este campo contendrá la el ID del autor del Tweet original. Esto no siempre será necesariamente el usuario mencionado directamente en el Tweet. | 6253282 |
| in\_reply\_to\_screen\_name | String | Puede ser nulo. Si el Tweet es una respuesta, este campo contendrá el nombre de pantalla del autor original del Tweet | twitterapi |
| from\_user\_id\_str | String | La cadena representa al identificador único para este usuario | 915406661157818368 |
| in\_reply\_to\_status\_id\_str | String | Puede ser nulo. Si el Tweet es una respuesta, este campo contendrá la representación de cadena del ID del Tweet original. | 1051222721923756032 |
| source | String | Aplicación utilizada para publicar el Tweet. (Formato HTML) | <a href="http:// twitter.com/download/ android" rel="nofollow">Twitter for Android</a> |
| profile\_image\_url | String | URL que apunta a la imagen de perfil del usuario | http://pbs.twimg.com/ profile\_images/ 915452554905038848/1 rVz0aNX\_normal.jpg |
| user\_followers\_count | Int | El número de seguidores que esta cuenta tiene actualmente. Bajo ciertas condiciones de coacción, este campo indicará temporalmente "0" | 41 |
| user\_friends\_count | Int | El número de usuarios que esta cuenta está siguiendo (también conocido como sus "seguidores"). Bajo ciertas condiciones de coacción, este campo indicará temporalmente "0" | 83 |
| user\_location | String | Puede ser nulo. La ubicación definida por el usuario en el perfil de su cuenta. | San Francisco, CA |
| status\_url | String | URL que apunta al tweet original | http://twitter.com/ MarielaAyerve/statuses/ 1038336680929964032 |
| entities\_str | String | Entidades que se han analizado del texto del Tweet. (Formato JSON). | {"hashtags": [{"text":"ODS","indices": [92,96]}],"symbols": [],"user\_mentions": [{"screen\_name":"aqua\_ vall","name":"AquaVall 💧 ","id":875691904196 907000,"id\_str":"875691 904196907008","indices ":[3,13]}, {"screen\_name":"aqua\_ vall","name":"AquaVall 💧 ","id":875691904196 907000,"id\_str":"875691 904196907008","indices ":[52,62]}],"urls": [{"url":"https://t.co/ LMUKvAPXhx","expand ed\_url":"https:// twitter.com/ VisionResponsab/ status/ 1108394166533394437" ,"display\_url":"twitter.co m/ VisionResponsa…","indices":[97,120]}]} |

Una vez descargado el CSV con los datos correspondientes del grupo, lo trabajamos a través de un programa externo “OpenRefine” que nos permitirá hacer más procesable al csv. Subimos el archivo a este programa y en la columna de *id\_str* utilizamos la opción de ver las “Duplicates facet” para visualizar las filas con datos repetidos, como se lo hace en el *gráfico 1*.

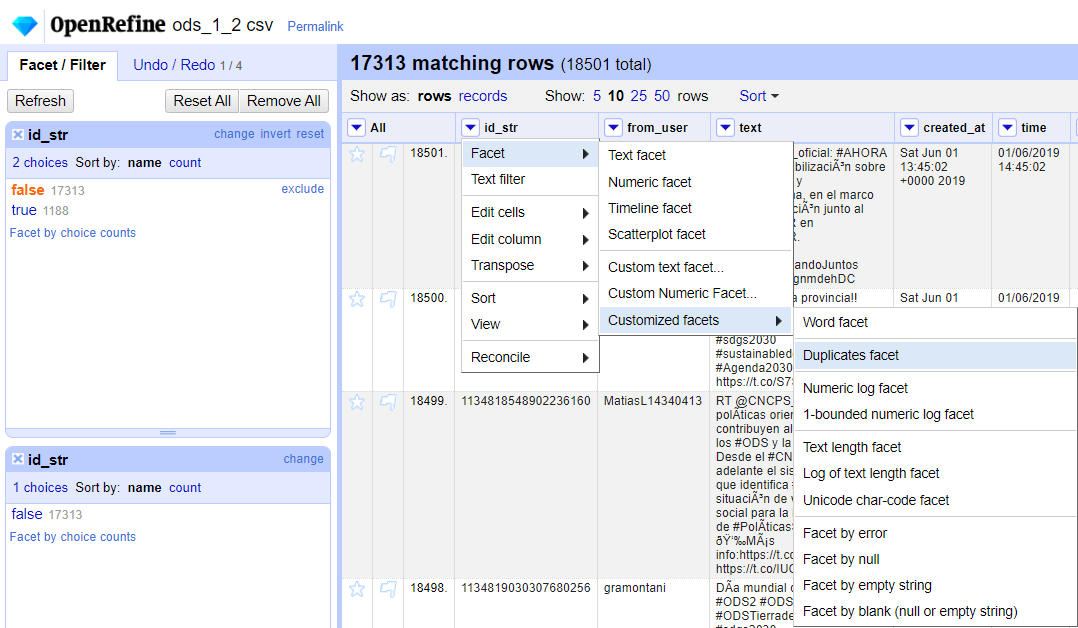


Gráfico 1 Determinar las filas duplicadas

Luego de determinar las filas duplicadas las ordenamos la columna de *id\_str* como si se tratase de números, como se especifica en el *gráfico 2.*

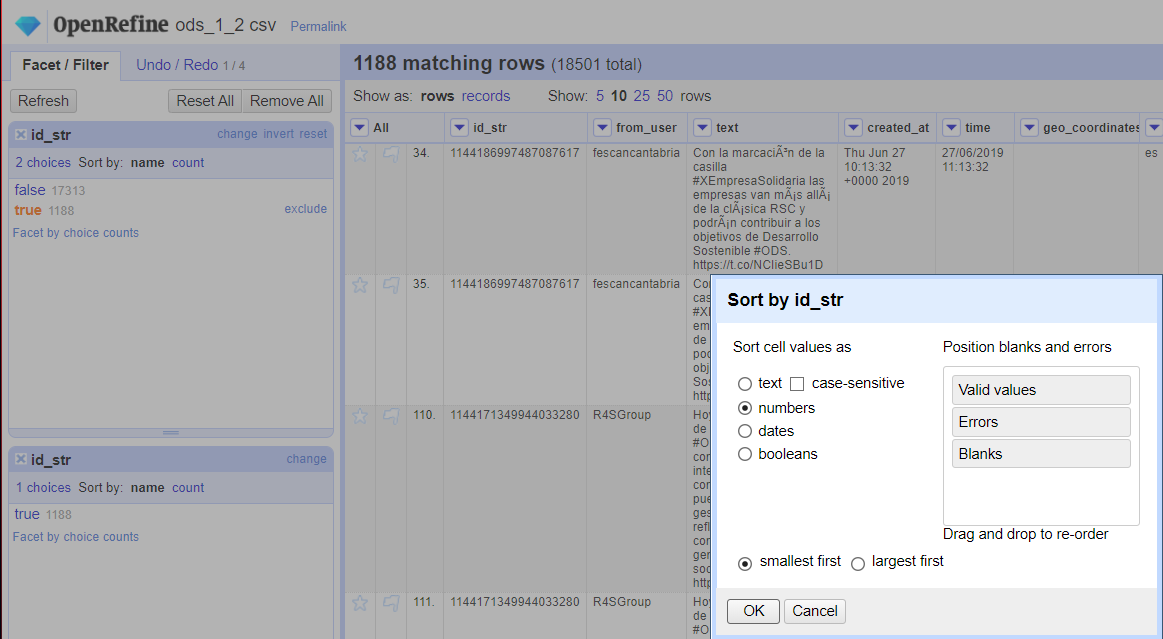


Gráfico 2 Ordenar las filas duplicadas

Ahora que tenemos las filas duplicadas adyacentes unas de otras, como se muestra en el *grafico 3* seleccionamos la pestaña de “All” y usamos la opción de “Blank down” que se encuentra dentro de “Edit cells” y ahora con los datos que nos quedaron usamos la opción de “remove all matching rows” para eliminar las filas que son repetidas de nuestro archivo csv, como se muestra el proceso en los *gráficos 3, 4 y 5.*

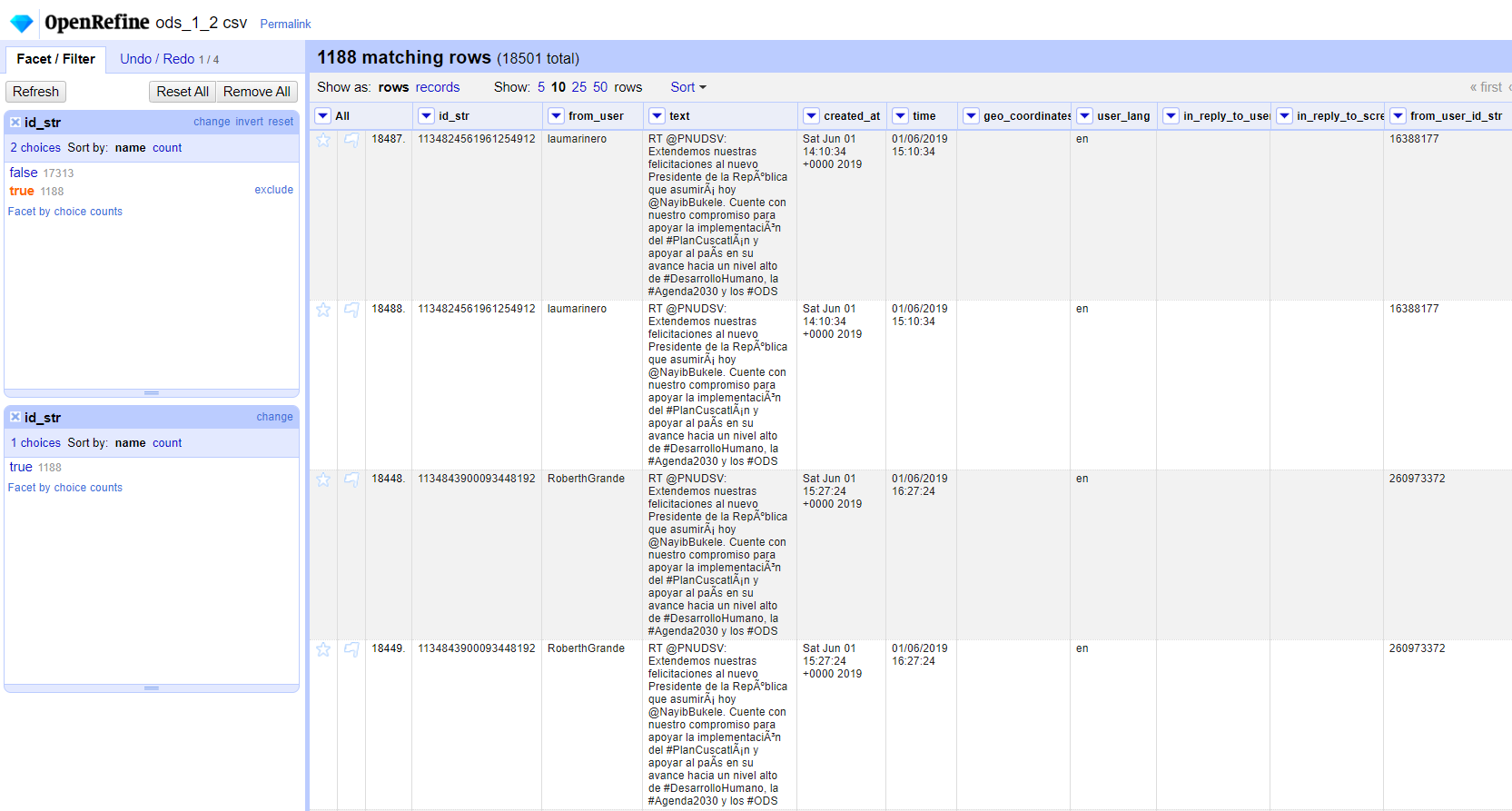


Gráfico 3 Filas duplicadas ordenadas

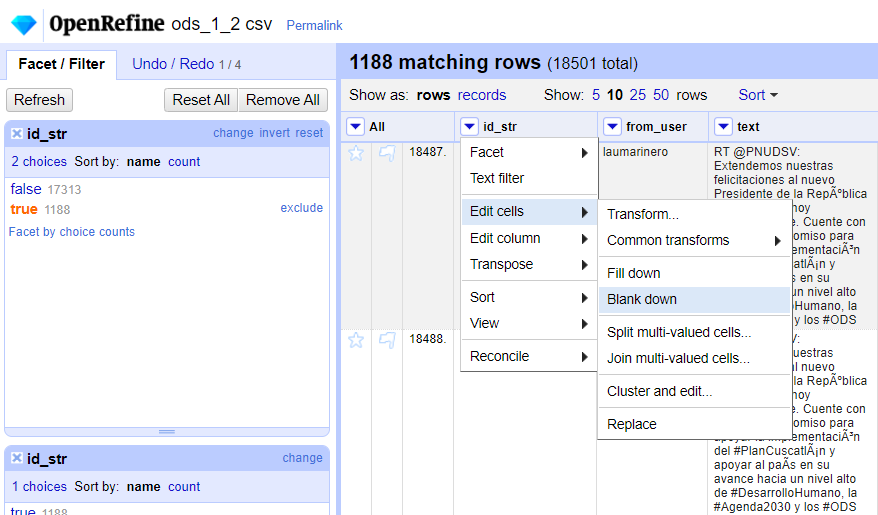


Gráfico 4 Vaciamos hacia abajo

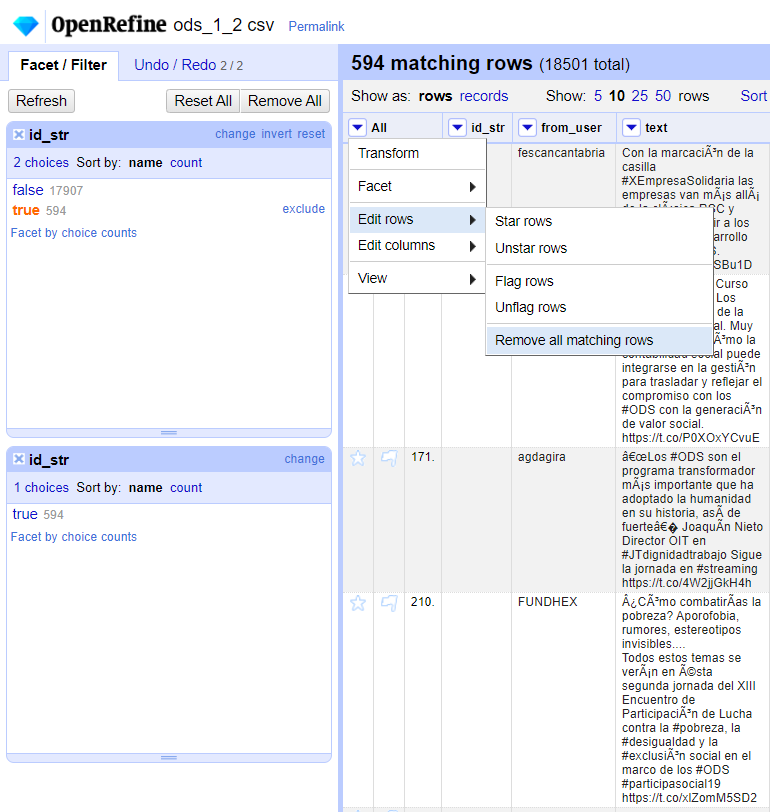


Gráfico 5 Eliminamos las filas duplicadas

Por último, para comprobar que nuestro archivo no tenga elementos repetidos hacemos el mismo proceso que hicimos en el grafico 1, y podemos observar en el *gráfico 6*, que ya no tenemos filas repetidas en nuestro archivo csv.

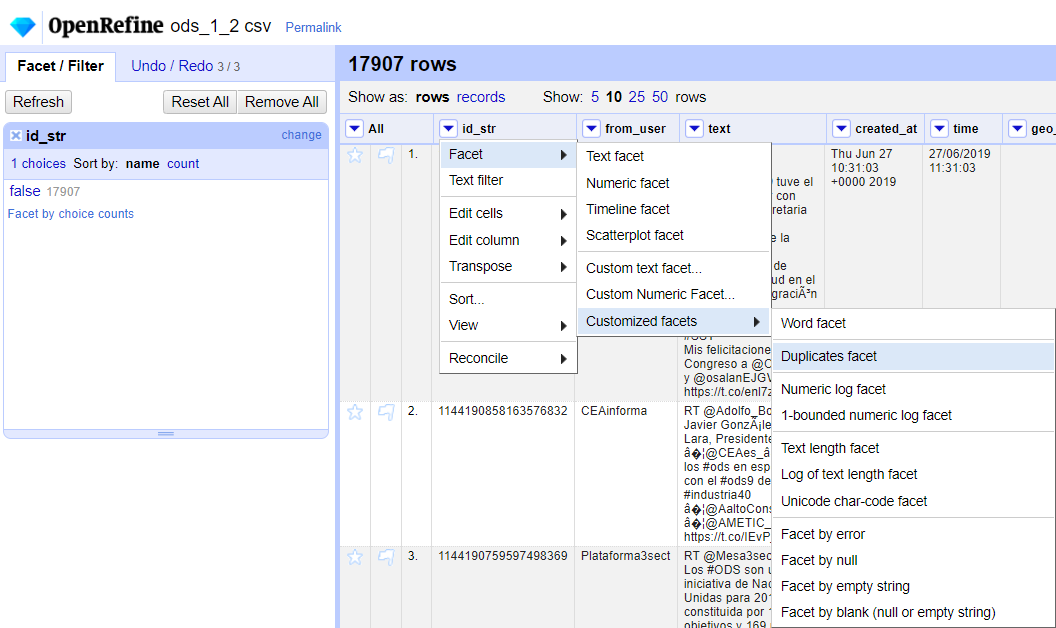


Gráfico 6 Archivo sin datos duplicados

A continuación, para simplificar un poco la información que nos da el usuario en la columna de “user\_lang”, como se observa en el *gráfico 7* el idioma proporcionado por el usuario suele presentar variaciones como es-mx que sigue siendo español o también en-gb que al final sigue siendo inglés.

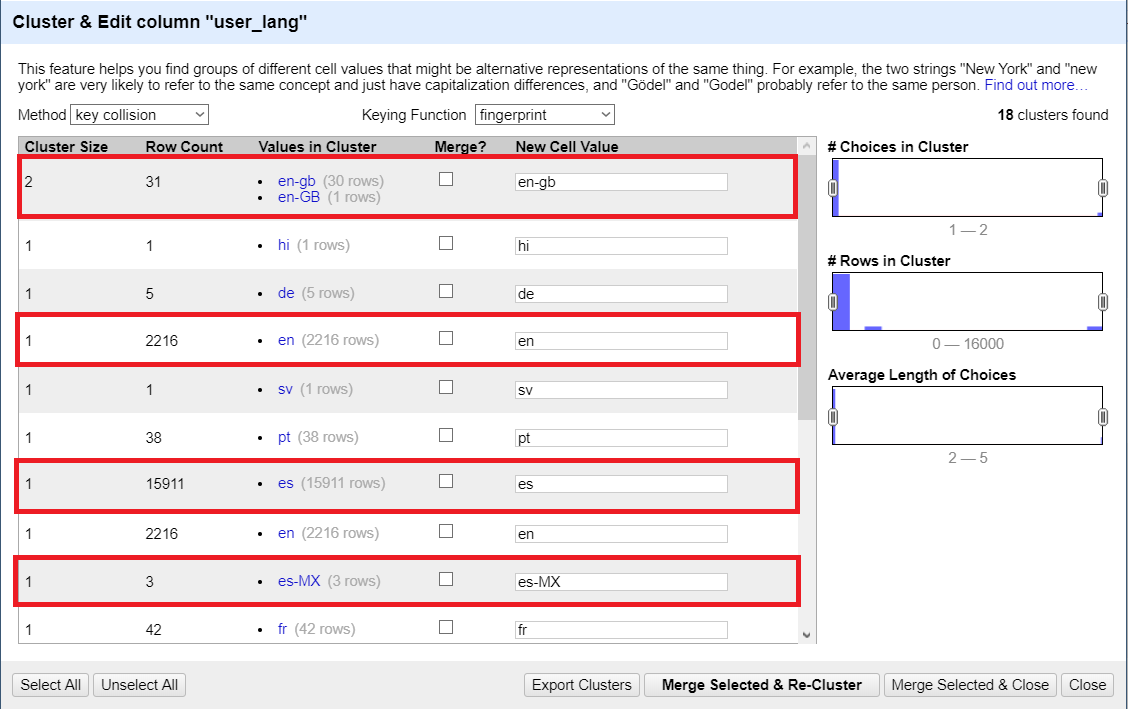
****

Gráfico 7 Variaciones en el "user\_lang"

Para esto vamos a la opción de “Cluster & Edit column” y esta nos agrupará los valores que son similares y luego solo editamos los nombres, seleccionamos y cerramos. En el *gráfico 8* podemos ver como ya se han agrupado todos los valores similares y ya no tenemos estas variaciones.

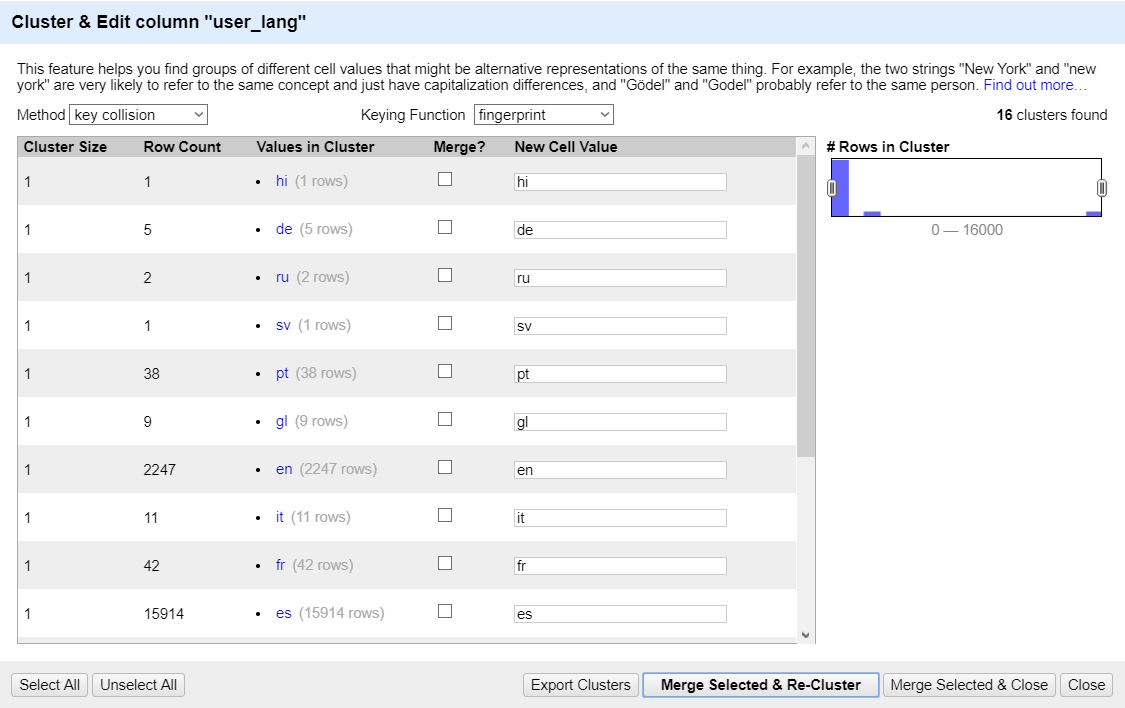


Gráfico 8 Variaciones suprimidas

Finalmente descargamos el archivo como un csv separado por comas, *gráfico 9*.

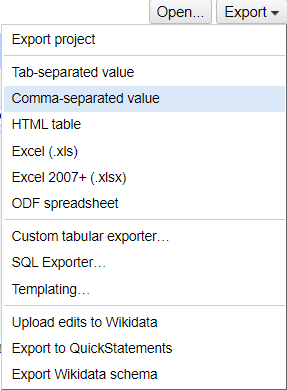


Gráfico 9 Descargamos el archivo

Una vez descargado el CSV con los datos correspondientes, los importamos a una hoja de Excel, con una codificación UNICODE(UTF-8) y con detección del tipo de dato basado en todo el conjunto de datos, como se muestra en el gráfico 1.

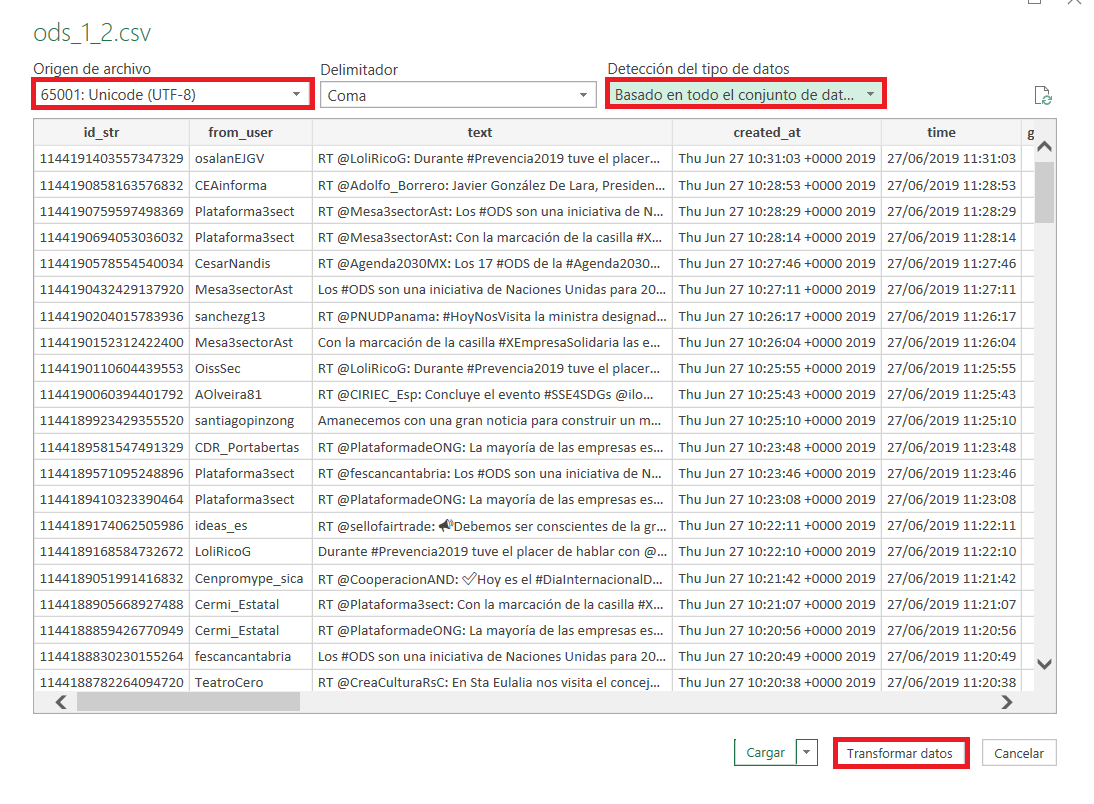


Gráfico 10 Importación de datos desde el csv

Luego seleccionamos en transformar datos revisamos y cambiamos los datos que estén con un tipo de dato incorrecto, como se puede identificar en el gráfico 2. En las columnas como el contador de seguidores y amigos las seleccionamos como tipo número entero y finalmente cerramos y cargamos los datos, como se muestra en el gráfico 3.

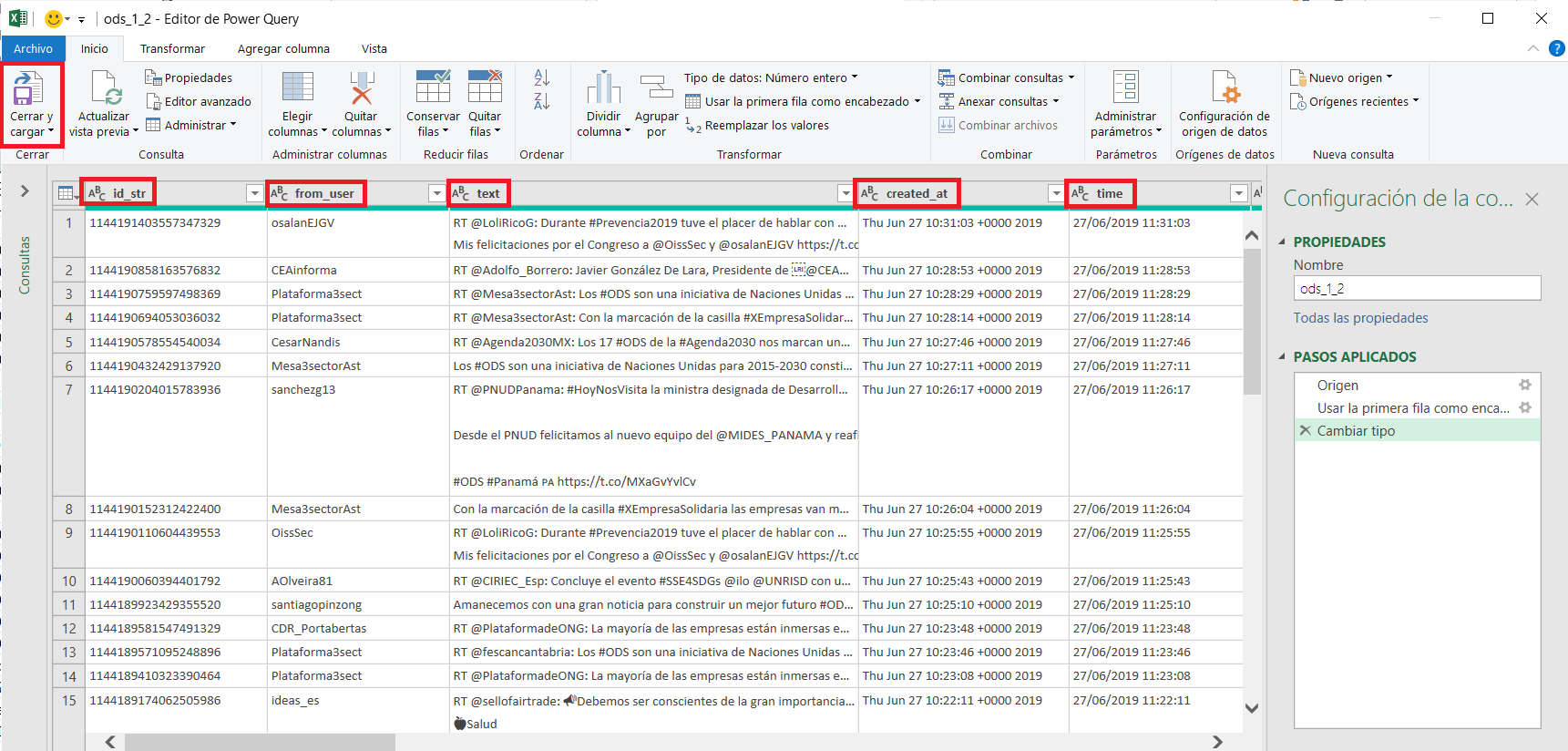
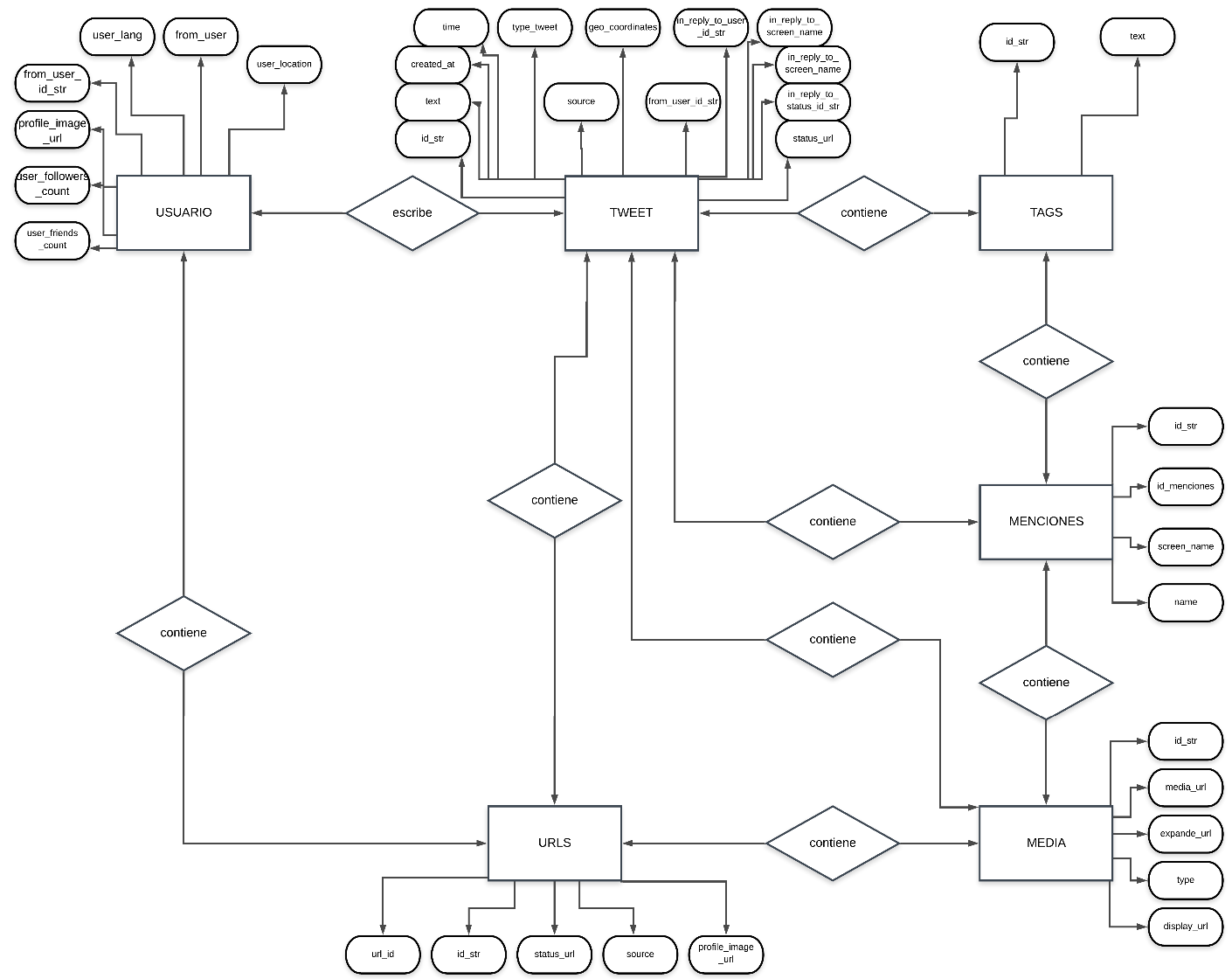


Gráfico 11 Detección de los tipos de datos



Gráfico 12 Datos cargados

1. **Modelado Conceptual**



Se obtuvieron las siguientes entidades luego del análisis del dataset:

**USUARIO:** El usuario es la persona u organización responsable de redactar y publicar un tweet. Algunos de los atributos más relevantes del usuario son el nombre que se muestra en pantalla, la foto de perfil, los seguidores, amigos, ubicación y su ID único de usuario que se relaciona con la tabla del tweet para indicar que usuario redacta el tweet.

**RE/TWEET:** Esta entidad es la principal en la base de datos, todas están relacionadas a ella y esta contiene todo lo que forma parte de un tweet o un retweet, el tweet es la publicación realizada por un usuario que puede contener letras, números, signos y enlaces, pero con un límite de 280 caracteres. Además, cada tweet contiene un ID único, el texto por el que está conformado, hora y fecha, y en ocasiones la localización de donde fue publicado el tweet.

**TAGS:** Los tags o etiquetas son escritos con el sigo # antepuesto que se utiliza para escribir palabras clave o temas específicos en Twitter.

Estos tags los podemos encontrar en la columna de “entities\_str” que poseen un formato JSON que forman parte de los Datos No Estructurados por lo que no se los puede visualizar con facilidad, pero se pueden visualizar por medio de la aplicación web JSON VIEWER

La entidad TAGS cuenta con el atributo id\_str para que se relacione con la entidad principal RE/TWEET ya que los tags son dependientes del tweet.

**URL:** El URL del tweet es la dirección especifica que se asigna a cada uno de los recursos disponibles en la red con la finalidad de que puedan ser localizados o identificados. La URL del tweet es distinta a la de la media debido a que esta entidad hace referencia directamente al tweet especifico. Se relaciona directamente con el tweet por lo que la clave foránea como en los demás casos es el id\_str, dependiendo directamente del tweet.

**MENCIONES:** Las menciones sirven para interactuar con otros usuarios y contiene el nombre del usuario que se quiere mencionar con la arroba (@) antepuesto, el ID de la mención y debido a que esta entidad también procede del JSON se utiliza la misma clave foránea id\_str para relacionar con la entidad tweet.

**MEDIA:** Esta entidad hace referencia a los elementos multimedia pueden ser utilizados un tweet y contiene un atributo tipo que especifica si es una imagen, video, gif, etc.; además del URL de la media. Estos son extraídos del JSON.

* 1. **Identificar los tipos de relación**

1. **Modelado Lógico**

En el apartado de modelo lógico se da a cada atributo un dominio, se colocan las claves primarías y foráneas en las tablas; se reevalúan las relaciones y si es necesario se eliminan las conexiones.

En el siguiente grafico se puede ver las relaciones que tiene las tablas entre sí, cada entidad tiene sus propios atributos los cuales se relacionan principalmente con tweet.

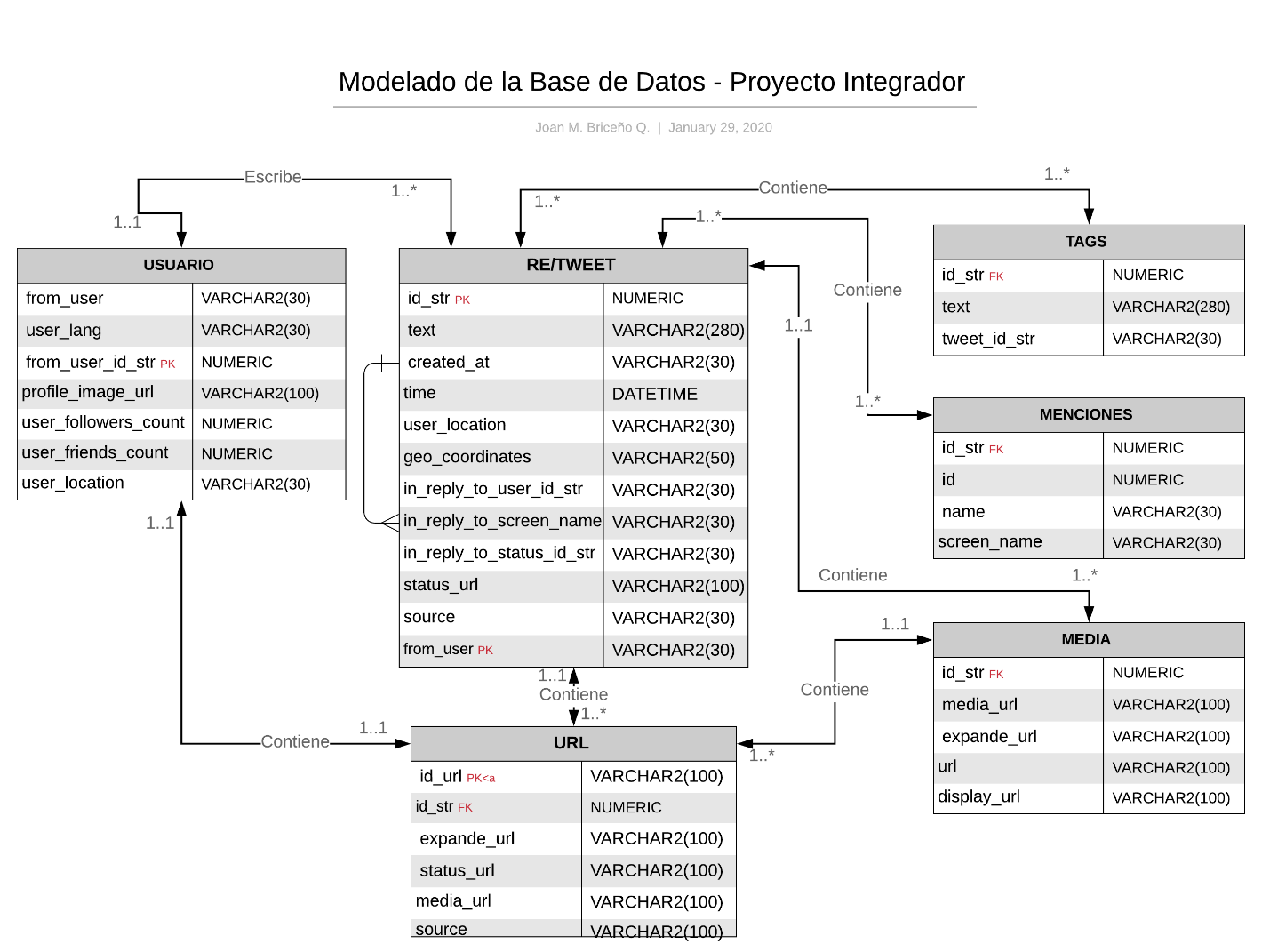


Gráfico 13 Modelado Lógico

Luego del modelado, seguimos con el levantamiento de la base de datos, creamos el usuario y lo nombramos “ProyectoIntegrador” con características de super usuario, como se muestra en el *gráfico 14*.

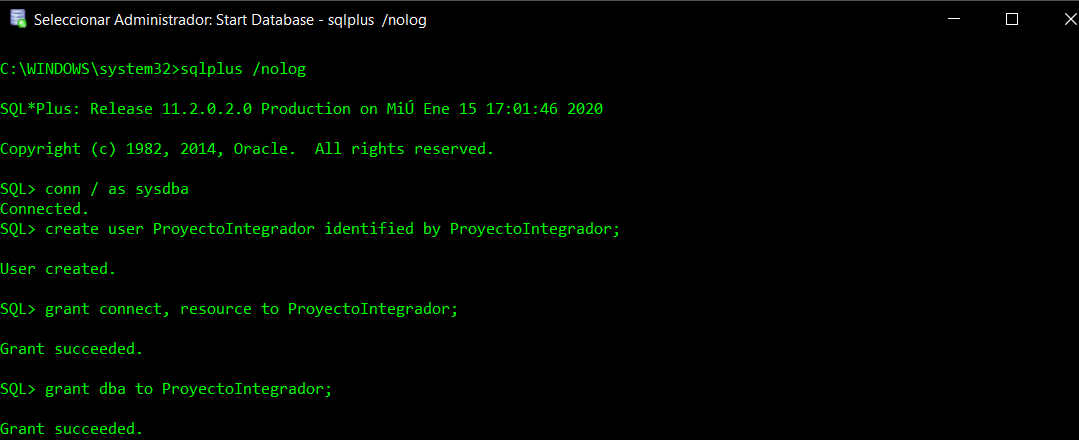
****

Gráfico 14 Creamos un usuario sysdba para la base de datos

1. **Modelado físico**

Después de validar las relaciones entre las entidades podemos crear las tablas de la base de datos, que poblaremos eventualmente mediante una tabla auxiliar “AUXODS”.

De igual manera debemos poblar las demás tablas, a través de diferentes tablas auxiliares obtenidas por medio de programación funcional. Una vez obtenidas las tablas auxiliares creamos scripts INSERT INTO para llenar las tablas creadas con valores de las tablas auxiliares.

* **Creación de Tablas.**

*DROP TABLE USUARIOS CASCADE CONSTRAINTS;*

*DROP TABLE TWEET CASCADE CONSTRAINTS;*

*DROP TABLE HASHTAG CASCADE CONSTRAINTS;*

*DROP TABLE MENCIONES CASCADE CONSTRAINTS;*

*DROP TABLE URL\_TWEET CASCADE CONSTRAINTS;*

*DROP TABLE MEDIA\_TWEET CASCADE CONSTRAINTS;*

*CREATE TABLE* ***USUARIOS*** *(*

*from\_user VARCHAR2(50),*

*user\_lang VARCHAR2(10),*

*from\_user\_id\_str NUMBER,*

*profile\_image\_url VARCHAR2(100),*

*user\_followers\_count NUMBER,*

*user\_friends\_count NUMBER,*

*user\_location VARCHAR2(100),*

*constraint pk\_from\_user\_id\_str PRIMARY KEY (from\_user\_id\_str)*

*);*

*CREATE TABLE* ***TWEET*** *(*

*id\_str NUMBER PRIMARY KEY,*

*text VARCHAR2(700),*

*created\_at VARCHAR2(600),*

*time VARCHAR2(100),*

*geo\_coordinates VARCHAR2(100),*

*in\_reply\_to\_user\_id\_str NUMBER,*

*in\_reply\_to\_screen\_name VARCHAR2(50),*

*in\_reply\_to\_status\_id\_str NUMBER,*

*status\_url VARCHAR2(400),*

*from\_user\_id\_str NUMBER,*

*constraint fk\_from\_user\_id\_str FOREIGN KEY(from\_user\_id\_str)*

*REFERENCES USUARIOS(FROM\_USER\_ID\_STR)*

*);*

*CREATE TABLE* ***HASHTAG*** *(*

*text\_h VARCHAR2(700),*

*id\_strH NUMBER,*

*constraint id\_strH FOREIGN KEY(id\_strH)*

*REFERENCES TWEET(id\_str)*

*);*

*CREATE TABLE* ***MENCIONES*** *(*

*id\_mentions NUMBER,*

*id\_strMen NUMBER,*

*screen\_name VARCHAR2(600),*

*name VARCHAR2(80),*

*constraint fk\_id\_strMen FOREIGN KEY(id\_strMen)*

*REFERENCES TWEET(id\_str)*

*);*

*CREATE TABLE* ***URL\_TWEET*** *(*

*id\_strU NUMBER,*

*status\_url VARCHAR2(600),*

*source VARCHAR2(80),*

*profile\_image\_url VARCHAR2(500),*

*constraint fk\_id\_strU FOREIGN KEY(id\_strU)*

*REFERENCES TWEET(id\_str)*

*);*

*CREATE TABLE* ***MEDIA\_TWEET*** *(*

*media\_url VARCHAR2(600),*

*id\_strM NUMBER,*

*display\_url VARCHAR2(600),*

*tipo VARCHAR2(80),*

*expanded\_url VARCHAR2(800),*

*constraint fk\_id\_strM FOREIGN KEY(id\_strM)*

*REFERENCES TWEET(id\_str)*

*);*

* **INSERTS**

*INSERT INTO* ***USUARIOS*** *(from\_user, user\_lang, from\_user\_id\_str, profile\_image\_url, user\_followers\_count, user\_friends\_count, user\_location)*

*SELECT DISTINCT from\_user, user\_lang, from\_user\_id\_str,PROFILE\_IMAGE\_URL, USER\_FOLLOWERS\_COUNT, USER\_FRIENDS\_COUNT, USER\_LOCATION*

*FROM AUX\_USUARIOS;*

*COMMIT;*

*INSERT INTO* ***TWEET*** *(id\_str, text, created\_at, time, geo\_coordinates, in\_reply\_to\_user\_id\_str, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id\_str, status\_url, from\_user\_id\_str)*

*SELECT DISTINCT id\_str, text, CREATED\_AT, TIME, GEO\_COORDINATES, IN\_REPLY\_TO\_USER\_ID\_STR, IN\_REPLY\_TO\_SCREEN\_NAME, IN\_REPLY\_TO\_STATUS\_ID\_STR, STATUS\_URL, FROM\_USER\_ID\_STR*

*FROM AUXODS;*

*COMMIT;*

*INSERT INTO* ***MENCIONES*** *(id\_mentions, id\_strMen, screen\_name, name)*

*SELECT id\_mentions, IDS, screen\_name, NAME*

*FROM AUX\_USER\_MEN x, TWEET t*

*WHERE x.ids = t.id\_str;*

*COMMIT;*

*INSERT INTO* ***URL\_TWEET*** *(ID\_STRU, STATUS\_URL, SOURCE, PROFILE\_IMAGE\_URL)*

*SELECT DISTINCT x.IDS\_STR, s.status\_url, s.SOURCE, s.PROFILE\_IMAGE\_URL*

*FROM AUX\_URLTWEET x, AUXODS s*

*WHERE x.IDS\_STR = s.id\_str;*

*COMMIT;*

*INSERT INTO* ***HASHTAG*** *(id\_strH, text\_h)*

*SELECT DISTINCT ids, HASHTAGS*

*FROM "AUX\_HASHTAGS" x, TWEET t*

*WHERE x.ids = t.id\_str;*

*COMMIT;*

*INSERT INTO* ***MEDIA\_TWEET*** *(media\_url, id\_strM, display\_url, tipo, expanded\_url)*

*SELECT DISTINCT MEDIA\_URL, IDS\_STR, DISPLAY\_URL, TYPE, EXPANDED\_URL*

*FROM "AUX\_MEDIA" x, TWEET t*

*WHERE x.IDS\_STR = t.id\_str;*

*COMMIT;*

## Aplicación de programación funcional

**Documentación**

Este proyecto insta a aplicar todos los conocimientos de tercer ciclo de la carrera en Ciencias de la Computación, así, en resumen, se deberá, a partir de un dataset, este deberá ser manejado mediante programación funcional para llegar a obtener los resultados propuestos:

* Número de Tweets y ReTweets por día.
* Número de Tweets y ReTweets por hora.
* Aplicaciones más utilizadas para publicar Tweets.
* Distribución de Hashtags.
* Distribución de menciones.
* Distribución de URLs
* Distribución de media.
* ¿Existe una correlación entre el número de amigos y la cantidad de seguidores?
* El comportamiento de los usuarios. Por cada usuario se debe presentar: la cantidad de seguidores y de amigos, también el número de Tweets y ReTweets.
* Cuántas veces se ha mencionado a un usuario.

**Herramientas**

Para el desarrollo del proyecto se utilizará el entorno [IntelliJ](https://www.jetbrains.com/es-es/idea/) utilizando el lenguaje de programación Scala.

Para el tratamiento de los datos en el csv usaremos la librería [Kantan](https://nrinaudo.github.io/kantan.csv/) que nos permitirá, importar y exportar archivos csv.

Para poder trabajar con la librería de Kantan, realizamos el "linkeo" correspondiente de Kantan que se encuentran en la página de GitHub [Kantan](https://nrinaudo.github.io/kantan.csv/), y pegarlos en el archivo con la extensión “.sbt” del proyecto en el entorno IntelliJ; esto nos ayudara a poder trabajar archivos csv en nuestro entorno, así como crear o leer este tipo de archivos. Luego de tener referenciada nuestra librería en la extensión sbt, importamos a nuestro proyecto principal con las siguientes sentencias:

import kantan.csv.ops.\_

import kantan.csv.ops.\_

import kantan.csv.generic.\_

import kantan.csv.java8.localDateDecoder.

Además, para realizar otros procesos requeridos para este proyecto, usaremos varias librerías extra, las cuales son:

import java.io.File

* Esta librería nos sirve para la lectura y escritura de los archivos csv.

import java.time.localDateTime

* Esta librería nos sirve para facilitar el trabajo de las fechas en los archivos csv.

import java.time.format.DateTimeFormatter

* Esta librería nos ayudará a definir un formato de tipo localDateTime.

import scala.util.{Try,Success, Failure}

* Esta librería nos permitirá realizar el manejo de excepciones gracias al operador Try

import scala.collection.immutable.ListMap

* Esta librería nos provee la herramienta “ListMap” para el filtrado de los datos.

**Tratamiento del dataset**

Después de realizar las importaciones de las librerías procedemos a crear una estructura con el dataset correspondiente del grupo:

val path2DataFile = "D:/TIMO/Desktop/UTPL/ods\_1\_2.csv"

* Se guarda la dirección del archivo local en el valor “path2DataFile” como una cadena de texto.

val formatDateTime = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss")

* Definimos un valor "formatDateTime", que conservará un formato de dato de tipo fecha/hora.

case class Tweet(

idStr: String,

fromUser: String,

text: String,

createdAt: String,

time: LocalDateTime,

geoCoordinates: String,

userLang: String,

inReply2UserId: String,

inReply2ScreenName: String,

fromUserId: String,

inReply2StatusId: String,

source: String,

profileImageURL: String,

userFollowersCount: Double,

userFriendsCount: Double,

userLocation: String,

statusURL: String,

entitiesStr: String

)

* Creamos nuestro case clase Tweet, el cual cada atributo será una de las columnas del dataset, y los valores serán los datos dentro de este. Se definen a la mayoría como tipo de dato String, a excepción de “time” que será del tipo localDateTime, y además “userFollowersCount” y “userFriendsCount” serán de tipo Double.

implicit val decoder : CellDecoder[LocalDateTime] = localDateTimeDecoder(formatDateTime)

* Ahora codificaremos el formato que definimos en “formatDateTime” en el tipo de dato “LocalDateTime”.

val dataSource = new File(path2DataFile).readCsv[List, Tweet](rfc.withHeader)

* Leemos el archivo, en el que se nos generará una estructura con objetos de tipo Tweet como lo definimos anteriormente.

val values = dataSource.collect({ case Right(tweet) => tweet }).toList

* Finalmente creamos un valor que colectará todos los datos de la estructura anterior, los cuales los obtendremos aplicando un “map” a la estructura creada en la anterior línea, aplicándole un case para extraer solamente los datos de tipo Right (que no presenten algún error).

**Solución**

**1. Número de Tweets y ReTweets por día.**

def numTweetsDia (ffData: List[Tuple3[Int,String,String]],isReTweet:Boolean):Int = {

val countersList = ffData.map(t3 => (t3.\_2,t3.\_3))

if(isReTweet)

(countersList.flatMap(t2 => List(t2.\_1)).filterNot(x=>x.startsWith("RT")).length)

else

(countersList.flatMap(t2 => List(t2.\_2)).filter(x=>x.startsWith("RT")).length)

}

val tweetsyRetweetsDia = values.map(tweet => (tweet.time.getDayOfMonth,tweet.text,tweet.text)).

groupBy(\_.\_1).map(kv=>(kv.\_1,

numTweetsDia(kv.\_2,true),

numTweetsDia(kv.\_2, false)

))

//Crear csv

val out = java.io.File.createTempFile("1TweetsyRetweetsDia",".csv")

val writer = out.asCsvWriter[(Int,Int,Int)](rfc.withHeader("days","tweets","retweets"))

tweetsyRetweetsDia.foreach(writer.write(\_))

writer.close()

**2. Número de Tweets y ReTweets por hora.**

def numTweetsHora (ffData: List[Tuple3[Int,String,String]],isReTweet:Boolean):Int = {

val countersList = ffData.map(t3 => (t3.\_2,t3.\_3))

if(isReTweet)

(countersList.flatMap(t2 => List(t2.\_1)).filterNot(x=>x.startsWith("RT")).length)

else

(countersList.flatMap(t2 => List(t2.\_2)).filter(x=>x.startsWith("RT")).length)

}

val tweetsyRetweetsHora = values.map(tweet => (tweet.time.getHour,tweet.text,tweet.text)).

groupBy(\_.\_1).

map(kv=>(kv.\_1,

numTweetsHora(kv.\_2,true),

numTweetsHora(kv.\_2, false)

))

//Crear csv

val out = java.io.File.createTempFile("2TweetsyRetweetsHora",".csv")

val writer = out.asCsvWriter[(Int,Int,Int)](rfc.withHeader("hours","tweets","retweets"))

tweetsyRetweetsHora.foreach(writer.write(\_))

writer.close()

**3. Aplicaciones más utilizadas para publicar Tweets.**

val remplazar = values.map(tweet => tweet.source.replace("</a>",""))

val appsUtilizadas = ListMap(remplazar.map(x => x.split(">").last)

.groupBy(identity).map({case(k,v) => (k, v.length)})

.toSeq.sortWith(\_.\_2 > \_.\_2):\_\*).filter(\_.\_2 > 10)

//Crear csv

val out = java.io.File.createTempFile("3AppsUtilizadas",".csv")

val writer = out.asCsvWriter[(String,Int)](rfc.withHeader("apps","count"))

appsUtilizadas.foreach(writer.write(\_))

writer.close()

**4. Distribución de Hashtags**

val distribucionHashtags = ListMap(values.map(tweet => ujson.read(tweet.entitiesStr).obj("hashtags").arr.length)

.groupBy(identity).map({case(k,v) => (k,v.length)})

.toSeq.sortWith(\_.\_2 > \_.\_2):\_\*)

//Crear csv

val out = java.io.File.createTempFile("4DistribucionDeHashtags", ".csv")

val writer = out.asCsvWriter[(Int, Int)] (rfc.withHeader("hashtags", "count"))

distribucionHashtags.foreach(writer.write(\_))

writer.close()

**5. Distribución de Menciones.**

val distribucionMentions = ListMap(values.map(tweet => ujson.read(tweet.entitiesStr).obj("user\_mentions").arr.length).

groupBy(identity).map({case(k,v) => (k,v.length)})

.toSeq.sortWith(\_.\_2 > \_.\_2):\_\*)

//Crear csv

val out = java.io.File.createTempFile("5DistribucionDeMenciones", ".csv")

val writer = out.asCsvWriter[(Int, Int)] (rfc.withHeader("mentions", "count"))

distribucionMentions.foreach(writer.write(\_))

writer.close()

**6. Distribución de URLs.**

val distribucionURLs = ListMap(values.map(tweet => ujson.read(tweet.entitiesStr).obj("urls").arr.length).

groupBy(identity).map({case(k,v)=> (k,v.length)})

.toSeq.sortWith(\_.\_2 > \_.\_2):\_\*)

//Crear csv

val out = java.io.File.createTempFile("6DistribucionDeURLs", ".csv")

val writer = out.asCsvWriter[(Int, Int)] (rfc.withHeader("urls", "count"))

distribucionURLs.foreach(writer.write(\_))

writer.close()

**7. Distribución de media.**

val distribucionMedia = ListMap(values.map(tweet => Try(ujson.read(tweet.entitiesStr).obj("media")) match {

case Success(v)=>"Tweets con multimedias"

case Failure(e)=>"Tweets sin multimedias"})

.groupBy(identity).map({case(k,v) => (k , v.length)})

.toSeq.sortWith(\_.\_2 > \_.\_2):\_\*)

//Crear csv

val out = java.io.File.createTempFile("7DistribucionDeMedia", ".csv")

val writer = out.asCsvWriter[(String, Int)] (rfc.withHeader("media", "count"))

distribucionMedia.foreach(writer.write(\_))

writer.close()

**8. ¿Existe una correlación entre el número de amigos y la cantidad de seguidores?**

val list= values.collect( { case tweet => (tweet.userFollowersCount,tweet.userFriendsCount) })

def Pearson (tabla: List[(Double, Double)] ): Double ={

val mediaX= (tabla. map(a => a.\_1).sum.toDouble/tabla.length)

val mediaY= (tabla. map(a => a.\_2).sum.toDouble/tabla.length)

val covarianza = ((tabla. map(a => a.\_1 \*a.\_2).sum )/ tabla.length) -(mediaX \*mediaY)

val desX= Math.sqrt((tabla. map(x => Math.pow(x.\_1,2)).sum/ tabla.length)- Math.pow(mediaX,2))

val desY =Math.sqrt((tabla. map(y => Math.pow(y.\_2,2)).sum/ tabla.length)- Math.pow(mediaY,2))

covarianza/(desX\*desY)

}

val correlacionP = Pearson(list)

**9. El comportamiento de los usuarios. Por cada usuario se debe presentar: la cantidad de seguidores y de amigos, también el número de Tweets y ReTweets.**

def processFFCounters(ffData: List[Tuple3[String,Double,Double]],isFollowers:Boolean):Int = {

val avg = (nums:List[Double])=> nums.sum / nums.length

val countersList = ffData.map(t3 => (t3.\_2,t3.\_3))

if(isFollowers)

avg(countersList.flatMap(t2 => List(t2.\_1))).toInt

else

avg(countersList.flatMap(t2 => List(t2.\_2))).toInt

}

def processFFCounters2(ffData: List[Tuple3[String,String,String]],isReTweet:Boolean):Int = {

val countersList = ffData.map(t3 => (t3.\_2,t3.\_3))

if(isReTweet)

(countersList.flatMap(t2 => List(t2.\_1)).filter(x=>x.startsWith("RT"))).length

else

(countersList.flatMap(t2 => List(t2.\_2)).filterNot(x=>x.startsWith("RT"))).length

}

def completo(ffData: List[Tuple5[String, Double, Double, String, String]],isTrue:Boolean,tweetOrFriends:Boolean):Int = {

val datos1 = ffData.map(x=>(x.\_1,x.\_2,x.\_3))

val datos2 = ffData.map(x=>(x.\_1,x.\_4,x.\_5))

if(tweetOrFriends:Boolean)

processFFCounters(datos1,isTrue)

else

processFFCounters2(datos2,isTrue)

}

val consulta = values.map(tweet => (tweet.fromUser,tweet.userFollowersCount, tweet.userFriendsCount,tweet.text,tweet.text)).

groupBy(\_.\_1).map(kv=>(kv.\_1,

completo(kv.\_2,true,true),

completo(kv.\_2, false,true),

completo(kv.\_2,true,false),

completo(kv.\_2, false,false)))

//Crear csv

val out = java.io.File.createTempFile("9Comportamiento",".csv")

val writer = out.asCsvWriter[(String,Int,Int,Int,Int)](rfc.withHeader("usuario","followers","friends","tweet","retweets"))

consulta.foreach(writer.write(\_))

writer.close()

**10. Cuántas veces se ha mencionado a un usuario.**

val mentions = values.flatMap(tweet => ujson.read(tweet.entitiesStr).obj("user\_mentions").arr)

.map(ht => ht.obj("screen\_name").str).groupBy(identity).map({case(k,v) => (k,v.length)})

//Crear csv

val out = java.io.File.createTempFile("10UsuariosMencionados",".csv")

val writer = out.asCsvWriter[(String,Int)](rfc.withHeader("user","mentions"))

mentions.foreach(writer.write(\_))

writer.close()

# Validación

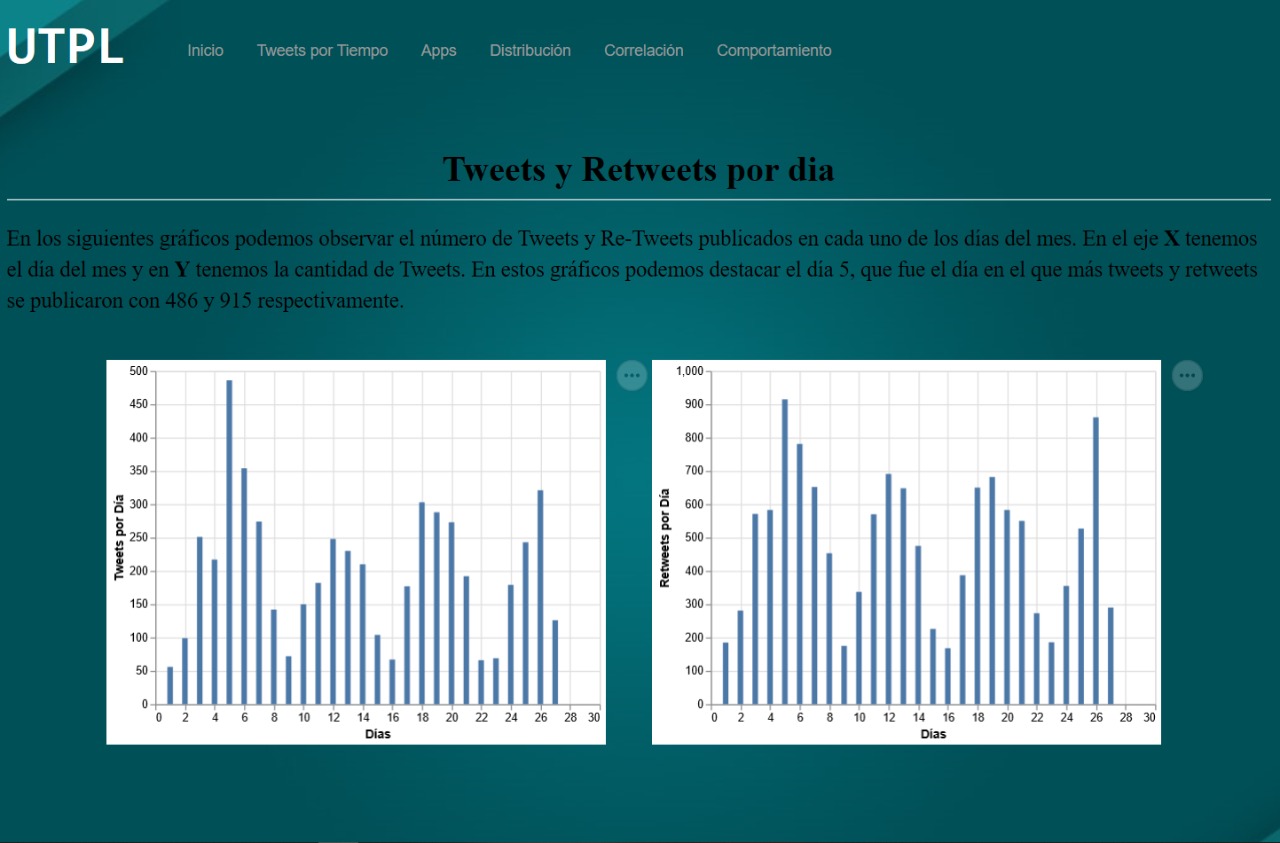


Gráfico 15 Tweets y Retweets por día.

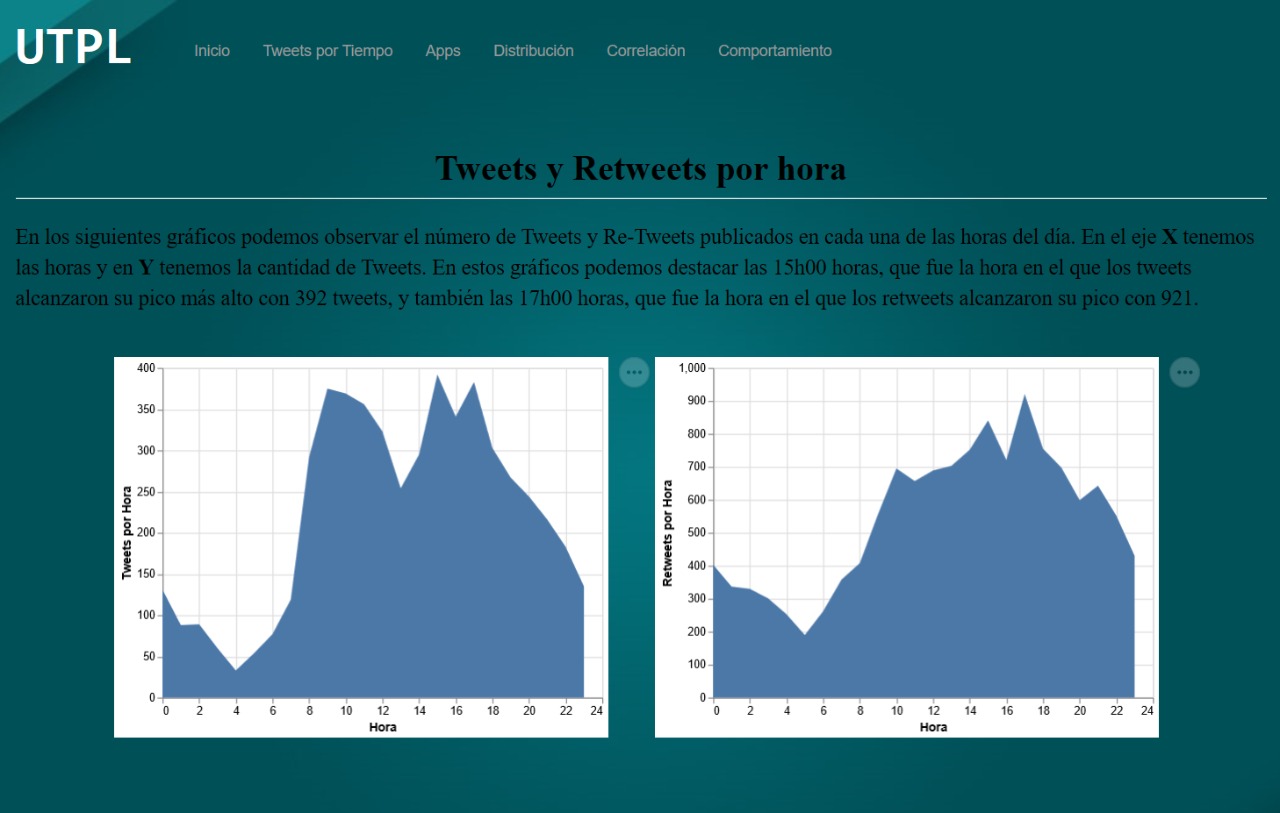


Gráfico 16 Tweets y ReTweets por hora.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 17 Aplicaciones más utilizadas para la publicación de Tweets.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 18 Distribución de Hashtags.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 19 Distribución de Menciones.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 20 Distribución de URLs.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 21 Distribución de Media.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 22 ¿Existe correlación entre amigos y seguidores?

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 23 Comportamiento de Usuarios.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Gráfico 24 Comportamiento de Menciones.

**Conclusiones**

- Una vez concluido el proyecto se observa la importancia de mantener siempre una buena gestión acerca de un conjunto de datos, si no, estos pueden convertirse en un verdadero quebradero de cabeza.

- Se puede observar que cada día se realizan más Retweets que Tweets, sin embargo, en determinadas horas se publican más Tweets que Retweets.

- Se determina que muchos de los Tweets publicados se realizan sin multimedia, es decir, solo texto, y sin hipervínculos.

- Se puede concluir que la mayoría de los tweets en los que se menciona a cualquier otro usuario se menciona a una sola otra cuenta, mas no a varias.

- Se puede determinar que si existe una correlación bastante alta entre el número de seguidores y numero de amigos.

- Mediante las imágenes se aprecia que la mayoría de las menciones son realizadas hacia empresas o corporaciones internacionales, dando por entendido que, Twitter, al ser una red social más seria que Facebook o Instagram se usa para informarse de determinadas situaciones en vez de ser usada para el entretenimiento como tal.

# Bibliografía

Becerro, R. G. (2016). *Paradigma Digital*. Obtenido de https://www.paradigmadigital.com/dev/la-programacion-funcional-deberias-usarla/

Borges, S. (2019). *Oracle*. Obtenido de ¿Qué es una base de datos?: https://www.oracle.com/es/database/what-is-database.html

Lapuente, M. J. (29 de Julio de 2018). *Hipertexto*. Obtenido de HTML: http://www.hipertexto.info/documentos/html.htm

Osorio, F. A. (2019). *Programacion Funcional: Conceptos y Perspectivas.* Bógota.

Pérez, E. I. (10 de Abril de 2019). *Codigo Facilito*. Obtenido de https://codigofacilito.com/articulos/programacion-funcional

Prado, C. E. (2016). *DevCode*. Obtenido de ¿Qué es y por qué aprender SQL?: https://devcode.la/blog/que-es-sql/

Scala. (Mayo de 2018). *Scala*. Obtenido de https://docs.scala-lang.org/tour/tour-of-scala.html

Soto, M. A., Burgueño, A. M., & Tostado, M. I. (2018). *PROTECCIÓN ANTE ATAQUES DE INYECCIÓN SQL EN APLICACIONES WEB.* Sinaloa.

UW Interactive Data Lab. (2016). *Vega*. Obtenido de Github: https://vega.github.io/vega-lite/docs/

# Anexos

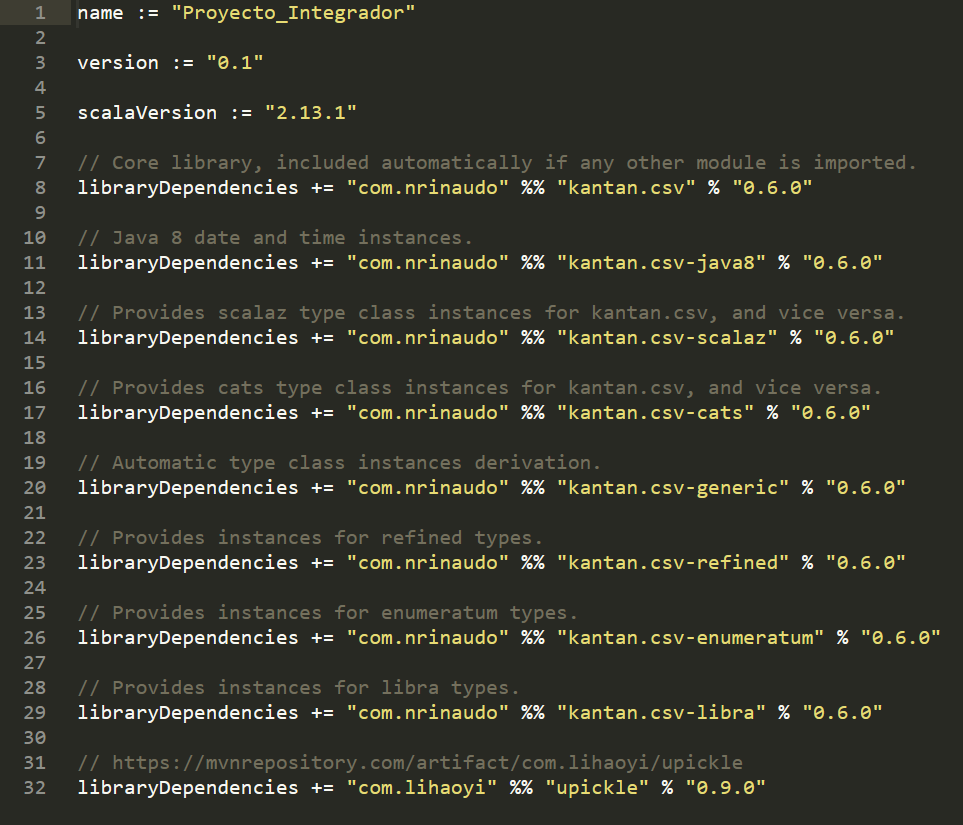


Gráfico 25 Librería importada "Kantan"

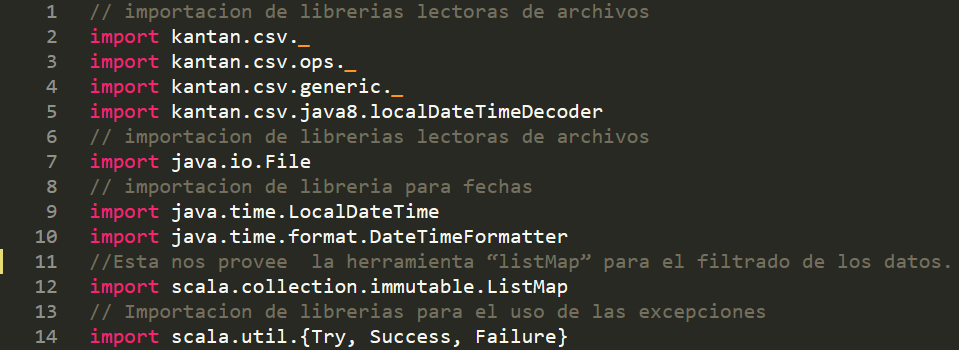


Gráfico 26 Importación de Librerías