CS4624

Multimedia, Hypertext, and Information Access

Final Report

December 16, 2021

Library6Btweets

Virginia Tech, Blacksburg VA 24061

Instructor: Dr. Edward A. Fox
Client: Xinyue Wang
Team: Yash Bhargava, Daniel Burdisso, Pranav Dhakal, Anna Herms, Kenneth Powell

Table of Contents

List of Figures

List of Tables

1.0 Abstract

The Digital Library Research Laboratory (DLRL) has collected billions of tweets over the course of years. They have been organized into collections. This data was collected from various research projects and using different data collection methods. The DLRL aims to consolidate this data so the Library can provide a service that allows the campus to easily access and use this data.

Our job was to make new files, to convert these tweets into a unified format, and to provide suitable collection information as well as mappings between tweets and collections by converting these tweets into a unified JSON format. To do this, we devised an individual tweet schema that all of the data collection types can be converted to. There were three data collection types that each store the data in different ways: Social Feed Manager (SFM), yourTwapperKeeper (YTK), and Digital Methods Initiative Twitter Capture and Analysis Toolset (DMI-TCAT). We designed this schema to take into account the three different tweet types, as well as the official Twitter version 2 schema. Additionally, we were asked to come up with a collection-level schema that includes a list of all of the tweet IDs that are included in a collection. This allows one to determine which tweets belong to each collection. We made the collection level schema to reflect the one used by the events archive site, with an additional list that stores all of the tweet IDs in the collection.

We were also tasked with writing scripts to convert the old data to the new schema that we devise. After taking our client's input and our team's experience into consideration, we determined that Python would be the best language to use. Thus, we made a Python script for each data type that would convert each collected tweet into the new unified schema. We also made separate scripts that went through all of the tweet collections and generated a collection level schema. Separate scripts had to be made because SFM, YTK, and DMI-TCAT all stored their data differently. For example, SFM exports into JSON files, while YTK and DMI-TCAT data is stored in databases. Additionally, the fields and data that YTK and DMI-TCAT store are different in both content and scope.

In total, we have six conversion scripts, and one script to aid in the reading of DMI-TCAT data due to its complexity. These scripts, along with the schemas, are our deliverables. They are available for use in a Gitlab repository that is hosted by Virginia Tech. We think the work we have done can be built on to fully complete this project. The individual tweet converters were completely finished this semester. This leaves only the collection-level converters to be completed, by finding a way to include the collection metadata in the SFM and DMI-TCAT collection converters.

2.0 Introduction

DLRL has collected billions of tweets from various research projects that cover a diverse range of collections. The data were collected using different tools, each of which generates a different style of data representation. The Library hopes to provide a service that can afford easy access to all the data.

2.1 Objective

Our primary objective was to unify the existing data from different sources into a common standard that can be flexibly digested for different service models. The goal of this project was to convert tweet data that exists in MySQL databases (YTK, DMI-TCAT) or JSON files (SFM) to comply with a newly designed JSON schema [1][2][3]. This new schema would need to be based on the Twitter tweet object standard. Another objective was to devise a JSON schema that describes each collection, and to provide a way to programmatically create these collection-level JSONs. Finally, we were to provide the Library with converted tweet data and created collection metadata.

2.2 Deliverables

The purpose of this project is to allow for ease of access by all students and faculty at Virginia Tech to the tweet data collected across various projects over the previous decade. To achieve this, the following deliverables were produced:

1. An individual tweet schema that works for all of the different tweets, in three different systems and their respective set of collections, that describes many aspects of the tweets
2. A tweet collection schema that describes the collection level information and a mapping schema to preserve the connection between tweet and collection
3. Three Python scripts that can convert SFM, YTK, and DMI-TCAT tweets to the new individual tweet schema
4. Three Python scripts that can summarize SFM, YTK, and DMI-TCAT tweets on the collection level

Additional details about each script can be found in Sections 5 and 7.

2.3 Client

The client for our project is Xinyue Wang, a research assistant at Virginia Tech for the Digital Library Research Laboratory. He has a background in digital library systems. He has been

guiding us throughout this project by providing us resources and directions. This project will ultimately be delivered to Bill Ingram, Assistant Dean and Director of Information Technology services for University Libraries at Virginia Tech.

2.4 Team

Our team consists of Yash Bhargava, Daniel Burdisso, Pranav Dhakal, Anna Herms, and Kenneth Powell. Yash and Daniel are seniors in Data Science. Pranav, Anna, and Kenneth are seniors in Computer Science.

3.0 Requirements

We needed to fulfill several requirements in order to satisfactorily complete the project. These requirements are detailed in the subsections below.

3.1 Individual Tweet Schema

For the individual tweet schema to be useful, several important requirements must be met. One important requirement is that the most important and most used fields in each of the three input data sources (YTK, SFM, and DMI-TCAT) should be present in the final schema. If important fields are missing, this removes a great deal of the benefit from compiling these input data sources into a final schema. A few examples of the many fields the client considers important include the full text of the tweet and a significant portion of the user data, and it is mandatory that these fields be included in the final schema.

For both the individual tweet schema and the collection-level tweet schema, the schema should be represented in a readable, standardized format. Additionally, for each field it should specify the field name, the data type that can be stored in the field, and whether the field is nullable (i.e., can be set to null). Subfields of JSON objects and elements of JSON lists should be displayed as well.

3.2 Collection-Level Tweet Schema

One important requirement for this project is to produce a collection-level schema. This is a specification for the structure of a JSON object designed to store the important information related to a single tweet collection. The collection-level tweet schema will be much simpler than the individual tweet schema, but there are still important requirements that a successful implementation of this part must meet. The project specification explicitly states that this schema must "describe collection level information and contain tweet IDs." Collection level information can include the number of tweets in the collection, the collection type, the collection description, and similar data. As requested by the client, a list of all of the tweet IDs that are included in the collection will also be in the schema. This will allow us to keep track of which tweet belongs to which collection.

3.3 Converted Tweet Data

There are several requirements that successfully converted tweet data must meet. One important one is that the tweet data should be consistent with the schemas, such that fields are included in

the tweets if and only if the schema mentions them. If this requirement is not met, using our converted data will not meet user needs.

Additionally, we should minimize the amount of effort required to successfully access attributes in our data. One important way we can do this is to consistently represent missing fields as being in the JSON object but assigned a value of null, rather than simply omitting them. This way, the user can use a simpler check to determine if a field is missing and avoid accidental exceptions that would occur when trying to access a field that does not exist.

4.0 Design

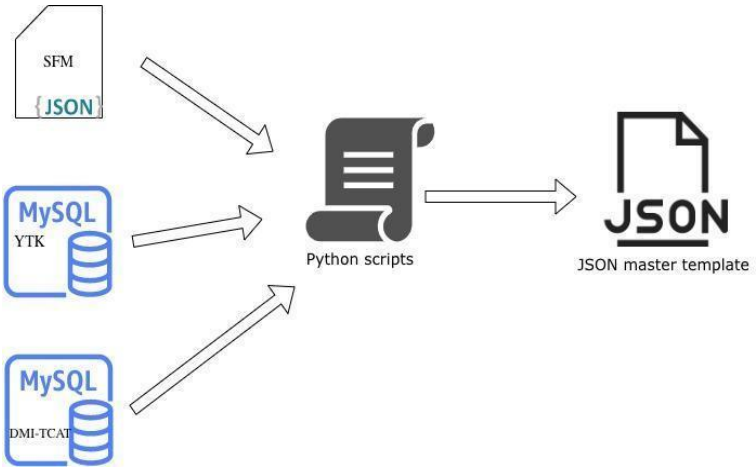


Figure 1: Basic design of our project

The goal of this project is to convert data that exists in MySQL databases or JSON files into a newly designed JSON schema. We needed to design a new tweet schema as a master template, as well as a collection-level JSON schema that describes collection level information. The team then needed to extract and convert the existing data according to these newly designed schemas.

4.1 Schema Design

As shown in Figure 1, our design required a JSON master template. To come up with this schema, it was recommended to us that we look at the fields in both the SFM schema and the Twitter v2 schema [4]. This allows us to keep important SFM fields that may not be present in the smaller YTK and DMI-TCAT forms. Additionally, by also looking at Twitter's official schema, we can be sure that our schema includes fields that Twitter uses in their API and deems important. However, some fields were not included despite being in the Twitter v2 schema. Examples of these non-included fields are promoted metrics, organic metrics, and non-public metrics [4]. These were not included in the final schema because none of these fields were present in any of the collected tweet forms. The final individual tweet schema we came up with, and the input fields from which each of the final fields is derived, can be seen in Table 1.

SFM schema	DMI-TCAT schema	YTK schema	Final schema
contributors			contributors
created_at	created_at	created_at	created_at
display_text_range[0]			display_text_range.start
display_text_range[1]			display_text_range.end

entities.hashtags			entities.hashtags
entities.media			entities.media
entities.media.url			entities.media.url
entities.media.id_str			entities.media.id
entities.urls			entities.urls
entities.user_mentions			entities.user_mentions
place.country			geo.country
geo.coordinates[0]	geo.lat	geo_coordinates_0	geo.latitude
geo.coordinates[1]	geo.lng	geo_coordinates_1	geo.longitude
geo.type		geo.type	geo.type
id_str	id (BigInteger)	Id	id
in_reply_to_screen_name			in_reply_to_screen_name
in_reply_to_user_id_str			in_reply_to_user_id
in_reply_to_status_id_str			in_reply_to_status_id
is_quote_status			is_quote_status
lang	lang		lang
metadata.iso_language_code		iso_language_code	lang_iso_code
metadata.result_type			result_type
favorite_count (is this for the tweet or user?)	favorite_count		metrics.favorite_count
retweet_count	retweet_count		metrics.retweet_count
source (HTML tag - will have to extract the exact source)	source	source	source
full_text	text	Text	text
user.contributors_enabled			user.contributors_enabled
user.created_at			user.created_at
user.default_profile			user.default_profile
user.default_profile_image			user.default_profile_image
user.description			user.description
user.follow_request_sent			user.follow_request_sent
user.geo_enabled			user.geo_enabled
user.id_str	from_user_id (BigInteger)	from_user_id	user.id
user.favorites_count			user.metrics.favorites_count
user.followers_count			user.metrics.followers_count
user.friends_count			user.metrics.friends_count
user.statuses_count			user.metrics.statuses_count
user.listed_count			user.metrics.listed_count
user.has_extended_profile			user.has_extended_profile
user.is_translation_enabled			user.is_translation_enabled
user.is_translator			user.is_translator
user.lang			user.lang

user.location			user.location
user.name	from_user_real_name		user.real_name
user.notifications			user.notifications
user.screen_name	from_user_name		user.screen_name
user.time_zone			user.time_zone
user.translator_type			user.translator_type
user.url			user.url
user.utc_offset			user.utc_offset
user.verified			user.verified
user.withheld_in_countries			user.withheld_in_countries

Table 1: Individual tweet schema.

The final column of Table 1 lists the fields included in our final JSON schema, while each of the other columns lists the field in each data source mapped to the corresponding final schema field. In JSON schemas, the notation 'field1.field2' refers to the field named 'field2' nested within the JSON object field named 'field1' (e.g., "field1": {"field2": field2val}). An empty cell means that there is no corresponding field in that input format, and during conversion each tweet object from that input format will receive a value of null for that field.

In addition to the individual tweet schema, we also designed a collection-level JSON schema for storing the data relevant to an entire collection of tweet objects; see Table 2. This collection-level schema will store the name, unique ID, and description of the collection. Additionally, it will store a list of all of the tweet IDs that are included in the collection, and a count of how many there are. This allows one to find which tweet belongs to which collection.

SFM schema	**YTK schema**	**DMI-TCAT schema**	**Final schema**
	Found in linked collection table csv file		id
	Found in linked collection table csv file		description
Total # of tweets found	Total # of tweets found	Total # of tweets found	count
List of all found "tweet_id" fields	List of all found "tweet_id" fields	List of all found "tweet_id" fields	tweet_ids
Deduped list of all found "hashtag" fields	List of keywords from archives table	Deduped list of all found "hashtag" fields	collection_terms
	Found in linked collection table csv file		wikipedia
	Found in linked collection table		create_time

	csv file		
Total retweet count of all tweets		Total retweet count of all tweets	metrics.retweet_count
Total like count of all tweets		Total like count of all tweets	metrics.like_count

Table 2: Collection-level schema.

The conventions used in Table 2 are the same as those used for Table 1. However, in some cases here the value for the final schema wasn't simply copied from an input data field, so we gave a short verbal description of how the value was calculated.

4.2 Script Design

The chosen design of this project is described in Figure 1. Our design focuses on Python scripts that will convert the collected data into the new schema we create. While we could have chosen any method of converting the data, we chose to use Python because it was recommended by the client and we were all comfortable using it already.

We also chose to use six separate Python scripts, one individual tweet script and one collection-level script for each input data format, even though the data being output by the scripts is in the same two schemas. This is because most of the code in the scripts is used to access fields within the SFM, YTK, or DMI-TCAT input tweet objects. This attribute access code differs significantly across the three formats due to their widely different structures (for instance, one is in JSON while the other two are database tables). Therefore, the approach of three separate scripts can be accomplished with minimal code redundancy.

4.3 Discarded Fields

The SFM collection had significantly more fields than both the DMI-TCAT and YTK collections with many of these fields not providing much use in the individual schema. YTK also had some fields that were not needed or provided little information. These fields were determined to not be included and were discarded as a result. Table 3 shows a list of these discarded fields along with the reason they were discarded.

Field	**Source**	**Reason**
archivesource	YTK	Field that indicates the collection is not necessary due to the collection level metadata.
to_user_id	YTK / DMI-TCAT	Field that indicates to what user a tweet is replying to is not relevant to the project.
profile_image_url	YTK / DMI-TCAT	Field indicating the URL of where the image originated not relevant to the project.
time	YTK	This is a duplicate of a YTK's created_time field in a different format
display_text_range	SFM	Field representing the number of spaces the tweet text took up is not relevant to the project.
entities.symbols	SFM	Contained fields with extraneous information not relevant to the project.
extended_entities	SFM	Many fields (entities, quoted status) had an additional "extended_entities" subfield that had fields that were repeated elsewhere, or were extraneous information.

Table 3: Discarded fields from individual schema

5.0 Implementation

Our implementation of this project involves several important software components. For each of the three input data formats (SFM, YTK, and DMI-TCAT), there is one converter for individual tweets. Each of the converters takes the form of a Python script that accepts an input data file or MySQL database and outputs a set of converted JSON tweets in the new schema. This is shown in Figure 2.

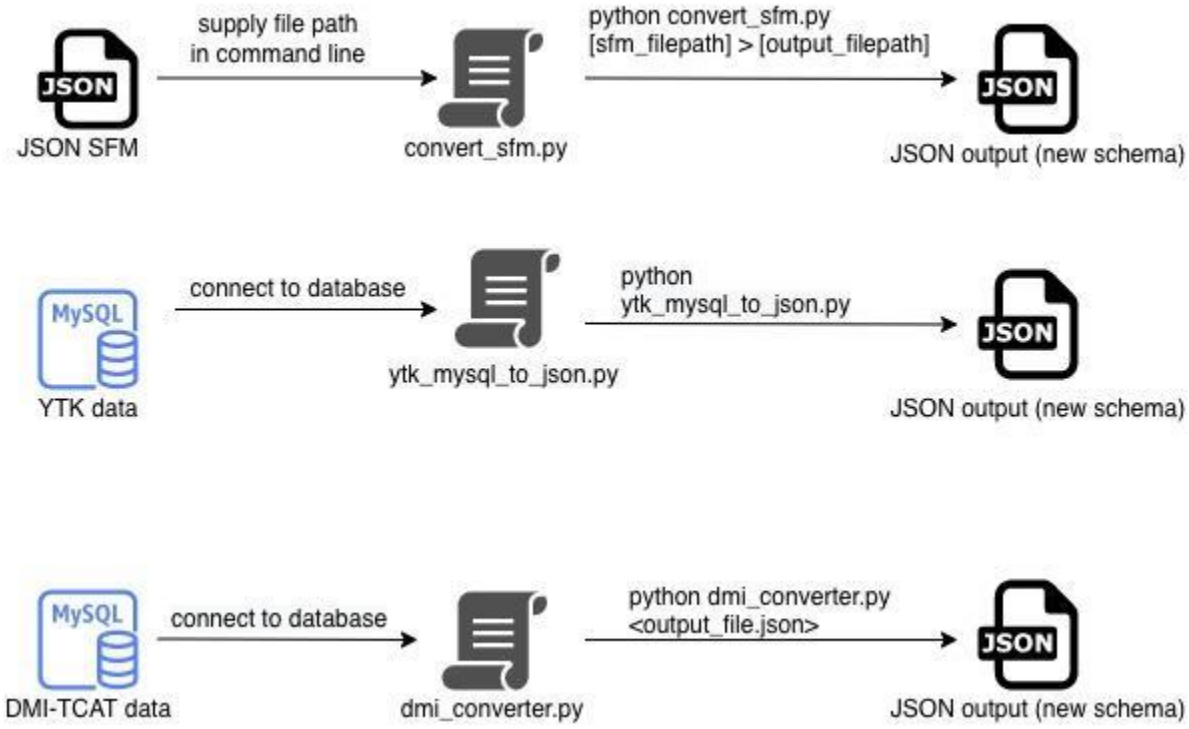


Figure 2: Visual workflow of individual tweet converters

We also have collection-level scripts that can take as input a tweet collection or set of tweet collections and, for each input collection, output one JSON object containing fields that summarize that collection. How these collection-level scripts work is outlined in Figure 3. This object will contain fields like a collection ID and a description if this information can be deduced from the input collections, as well as other data such as a list of included tweet IDs.

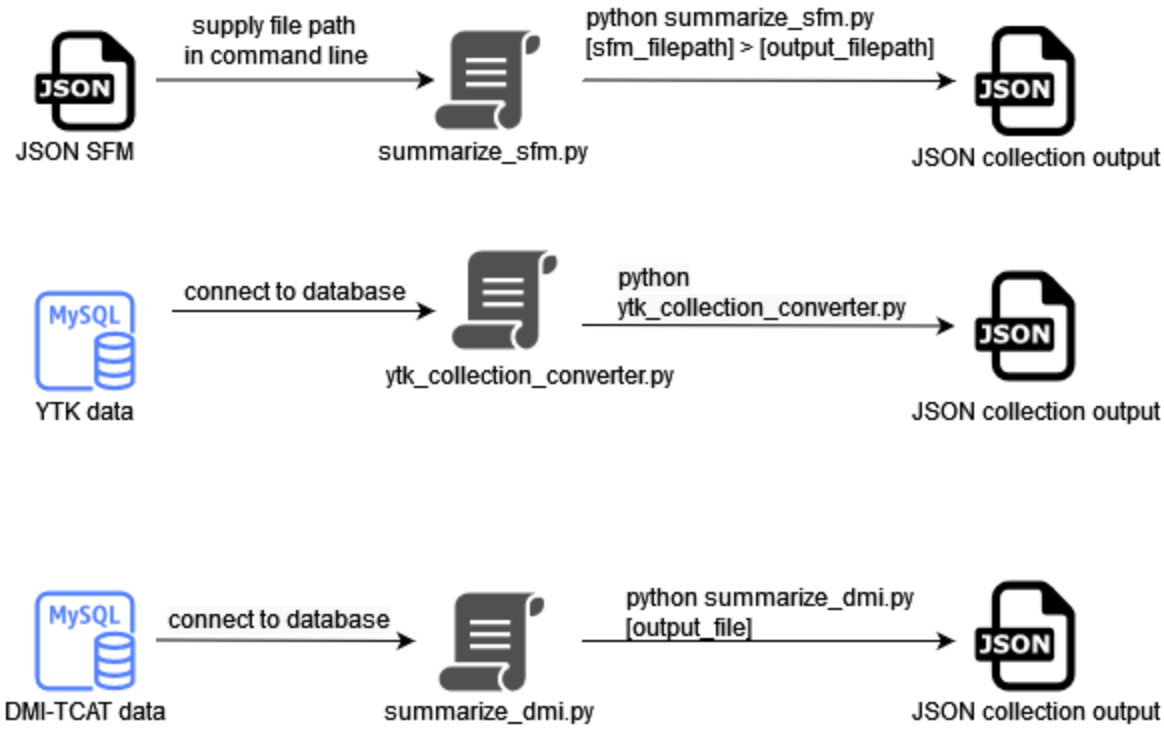


Figure 3: Visual workflow of collection-level converters

One useful technique to reduce the amount of boilerplate code in these scripts is to create empty template JSON objects that have the correct structure and field names but have their fields set to null. To avoid having to set many fields to null manually if they were missing from the input data, we had our scripts copy an empty object and then fill in only the non-null fields. This was especially useful for the YTK and DMI-TCAT scripts because these schemas are missing many of the fields present in our final schema.

Additionally, we wrote the converter scripts to consistently output their JSON data with a single JSON object per line. There are a number of advantages to this approach:

- An arbitrary tweet object can easily be parsed simply by seeking to a line in the output JSON file, reading in the line, and parsing it.
- Corruption in an input data file over a small number of lines has a localized effect, only making the tweet objects in the affected lines unreadable. In contrast, if all the tweet objects were stored in one JSON list, the entire file would essentially become unreadable because the attempt to read in the list would fail.
- Our output datasets can easily be combined by simply appending one file to the end of another.
- Since each line in the file is a syntactically valid JSON object, one can move the first *k* objects to another file by copying the first *k* lines of the source file to the destination file.

- The one-object-per-line format is already how SFM represents their JSON data, so we know there is good precedent for this format, and SFM users will be accustomed to it.

5.1 SFM Individual Tweet Converter Implementation

This project's SFM converter module takes an input file containing JSON SFM tweet objects and outputs a stream of tweet objects in the new schema. It consists of a single Python script. To perform the conversion, the converter script reads in the JSON file it was passed. The SFM file contains one JSON object per line, so the script uses the Python JSON library to load in each line of the input file as an SFM JSON tweet object. Each relevant field is copied over to a new tweet object in the new schema.

In the original SFM tweet object there are often missing fields, which are either represented as fields assigned the value null or as fields that do not appear in the original JSON object at all. For each of these cases, the SFM converter specifies that the corresponding field in the new schema is missing by including a field in the output JSON object but assigning it to null. For example, if the "contributors" field is missing from the input SFM tweet or it is null, the SFM converter will include a null field called "contributors" in the output JSON object. Missing list fields are assigned an empty list instead of null so that a special case in user code isn't needed for an empty/missing list.

After conversion, the dump() method of the JSON object is invoked while ensuring that the keyword argument ensure_ascii is set to false. This is to allow Unicode characters to be rendered correctly. Finally, each new schema JSON object is written to standard output, while standard output is redirected to a file designated for converted JSON tweets.

Figure 4 is a screenshot showing part of the sample SFM input without pretty-printing:

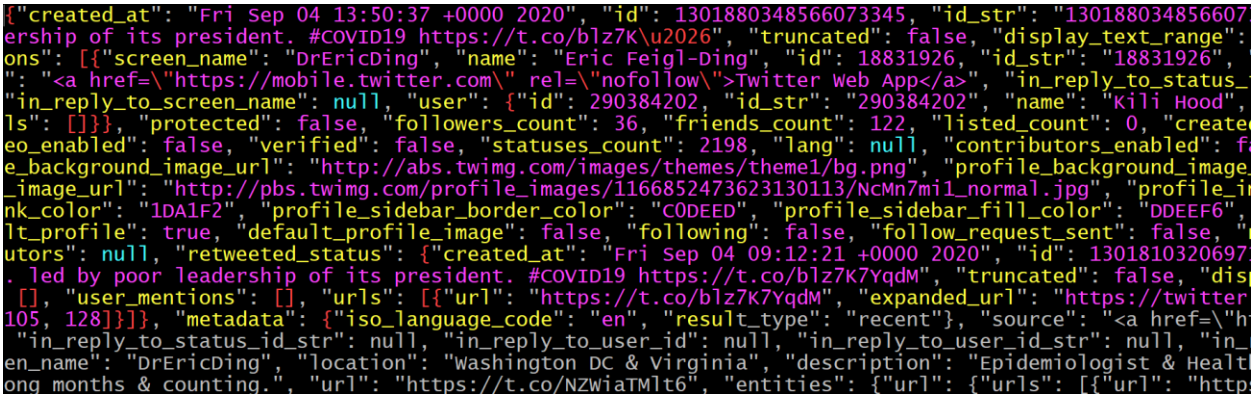


Figure 4: SFM input screenshot

Figure 5 is a screenshot showing part of the current output of the SFM converter without pretty-printing, when a different SFM tweet other than that depicted in Figure 4 was converted.

Figure 5: New schema output screenshot

They look similar, since both are in JSON format, but there are differences in the field structure.

5.2 SFM Collection-Level Converter Implementation

The collection-level converter implementation is simpler than the implementation of the individual tweet converter, since the collection-level schema is less complex. The script reads in the input SFM file it was passed. It iterates through each tweet in the input file. For each tweet, the script increments the count of the number of tweets it has seen, appends the tweet ID to a list, and updates the cumulative count of certain metrics such as the total number of retweets and likes. It also tracks the number of unique hashtags it has seen, using a set to efficiently store a duplicate-free collection. These fields are then assigned to the appropriate collection-level fields in an output JSON object and the object is dumped to standard output. The user can use output redirection to store the object in a file.

5.3 YTK Individual Tweet Converter Implementation

The YTK converter converts each collection of tweets in YTK and outputs a stream of individual tweet objects in the new schema, in a JSON file containing each tweet in that collection. YTK data is originally stored in MySQL, so because of that the data file is read in using MySQL, and then the information is pulled from each individual tweet, and then it is converted into the new JSON schema. Each relevant field is copied over to a new tweet object in the new schema. Any value that is specified in the new schema but could not be found in the YTK tweet is set to a null

value in the new tweet object. Most of the fields in the output are null due to the limited number of fields available from YTK, and some fields are not useful for the output.

Figure 6 is a screenshot of one example of a collection with what fields it contains and its type.

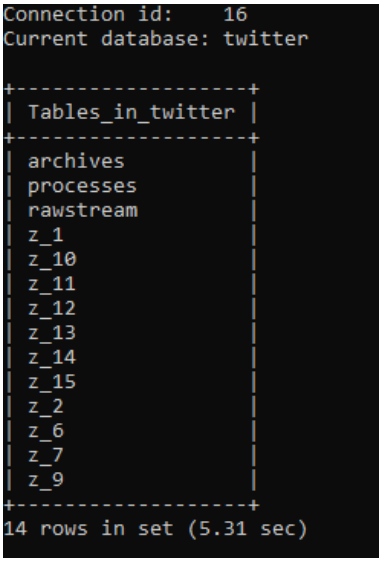
```
mysql> describe z_1;
+-------------------+---------------+------+-----+---------+-------+
| Field             | Type          | Null | Key | Default | Extra |
+-------------------+---------------+------+-----+---------+-------+
| archivesource     | varchar(100)  | NO   |     | NULL    |       |
| text              | varchar(1000) | NO   | MUL | NULL    |       |
| to_user_id        | varchar(100)  | NO   |     | NULL    |       |
| from_user         | varchar(100)  | NO   | MUL | NULL    |       |
| id                | varchar(100)  | NO   | MUL | NULL    |       |
| from_user_id      | varchar(100)  | NO   |     | NULL    |       |
| iso_language_code | varchar(10)   | NO   | MUL | NULL    |       |
| source            | varchar(250)  | NO   |     | NULL    |       |
| profile_image_url | varchar(250)  | NO   |     | NULL    |       |
| geo_type          | varchar(30)   | NO   | MUL | NULL    |       |
| geo_coordinates_0 | double        | NO   |     | NULL    |       |
| geo_coordinates_1 | double        | NO   |     | NULL    |       |
| created_at        | varchar(50)   | NO   |     | NULL    |       |
| time              | int           | NO   | MUL | NULL    |       |
+-------------------+---------------+------+-----+---------+-------+
14 rows in set (0.00 sec)
```

Figure 6: Screenshot of the YTK collection's table organization

5.4 YTK Collection-Level Converter Implementation

The YTK collection-level conversion script is a simpler implementation than the individual conversion script. This is because the collection level converter does not need to access any fields from the individual tweets with the exception of the tweet ID. YTK, unlike SFM and DMI-TCAT, contains collection level information linking tables to collection terms. This information can be found in the YTK "archives" table with each collection listed therein referenced by an ID corresponding to a separate table. The script takes one argument and outputs all collections to a single JSON file.

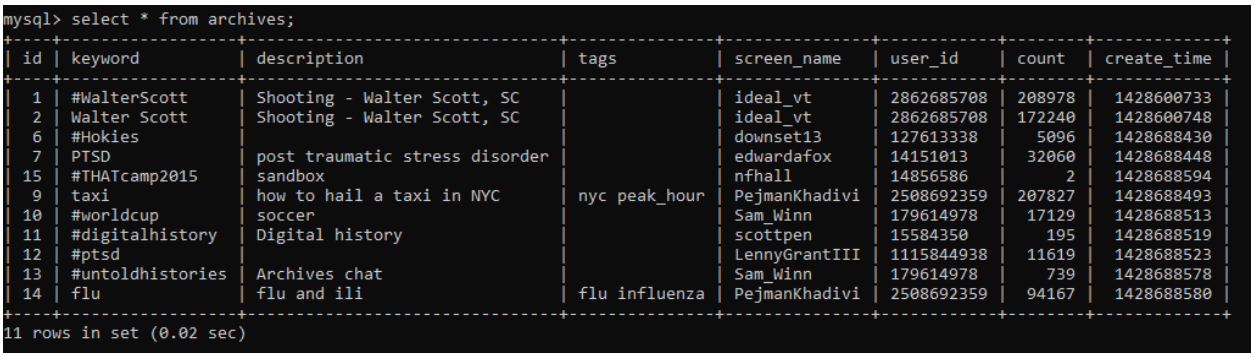
The script takes advantage of the additional tables contained by YTK describing collection level information for each table. The tables containing the actual tweet objects are labeled with a z followed by an underscore and an ID number. The three other tables available are called archives, processes, and rawstream. The archives table lists the keywords used to gather each of the separate data collections as well as the corresponding ID used in naming the tweet collection tables. Figure 7 shows these tables from the YTK tweet data stored in MySQL. Figure 8 shows the archives table (one of those listed in Figure 7) showing the initial list of keywords used for the initial set of tweet collections.



```
Connection id:    16
Current database: twitter

+-------------------+
| Tables_in_twitter |
+-------------------+
| archives          |
| processes         |
| rawstream         |
| z_1               |
| z_10              |
| z_11              |
| z_12              |
| z_13              |
| z_14              |
| z_15              |
| z_2               |
| z_6               |
| z_7               |
| z_9               |
+-------------------+
14 rows in set (5.31 sec)
```

Figure 7: Screenshot of tables in YTK sample

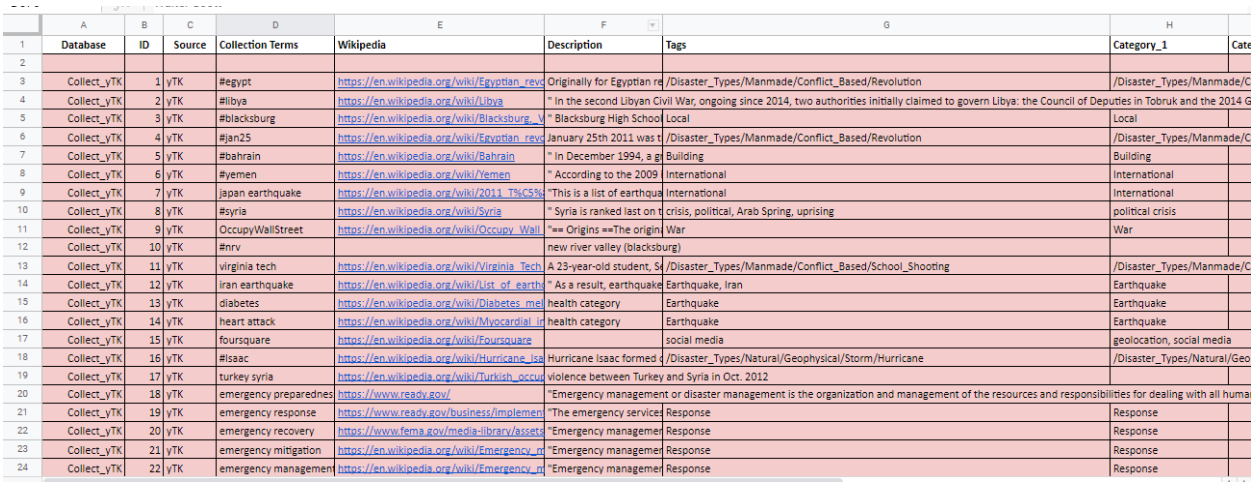


```
mysql> select * from archives;
+----+-----------------+-------------------------------+--------------+--------------+------------+--------+-------------+
| id | keyword         | description                   | tags         | screen_name  | user_id    | count  | create_time |
+----+-----------------+-------------------------------+--------------+--------------+------------+--------+-------------+
|  1 | #WalterScott    | Shooting - Walter Scott, SC   |              | ideal_vt     | 2862685708 | 208978 |  1428600733 |
|  2 | Walter Scott    | Shooting - Walter Scott, SC   |              | ideal_vt     | 2862685708 | 172240 |  1428600748 |
|  6 | #Hokies         |                               |              | downset13    | 127613338  |   5096 |  1428688430 |
|  7 | PTSD            | post traumatic stress disorder |              | edwardafox   | 14151013   |  32060 |  1428688448 |
| 15 | #THATcamp2015   | sandbox                       |              | nfhall       | 14856586   |      2 |  1428688594 |
|  9 | taxi            | how to hail a taxi in NYC     | nyc peak_hour | PejmanKhadivi | 2508692359 | 207827 |  1428688493 |
| 10 | #worldcup       | soccer                        |              | Sam_Winn     | 179614978  |  17129 |  1428688513 |
| 11 | #digitalhistory | Digital history               |              | scottpen     | 15584350   |    195 |  1428688519 |
| 12 | #ptsd           |                               |              | LennyGrantIII | 1115844938 |  11619 |  1428688523 |
| 13 | #untoldhistories | Archives chat                 |              | Sam_Winn     | 179614978  |    739 |  1428688578 |
| 14 | flu             | flu and ili                   | flu influenza | PejmanKhadivi | 2508692359 |  94167 |  1428688580 |
+----+-----------------+-------------------------------+--------------+--------------+------------+--------+-------------+
11 rows in set (0.02 sec)
```

Figure 8: Screenshot of archives table in ytk2-eom_all_mysql.sql

The YTK collection level conversion uses another file, Collection_Table_for_IA_-_rich_collections.csv, which contains a list of known collections available across all three collection sources and is the file that must be passed as the argument for the script. This file contains several fields not found in the YTK data such as the collection ID, description, wikipedia page, and create_time. Figure 9 shows a screenshot of this file. This file can also be

found at the Virginia Tech Events Archive site or http://eventsarchive.org/ [5]. The script retrieves these fields from this file as well as the tweet IDs from each of the tweet data tables to create the final collection level data for each collection. Each keyword from the archives table is checked to see if available in the list of known collections. For collections with keywords not found in the list of known collections, collection level data is still created with missing fields for the collection ID and wikipedia page. The description and created_time fields are filled with the corresponding fields, description and starting_date respectively, found in the archives table if available.



	A	B	C	D	E	F	G	H	
1	Database	ID	Source	Collection Terms	Wikipedia	Description	Tags	Category_1	Cate
2									
3	Collect_yTK	1	yTK	#egypt	https://en.wikipedia.org/wiki/Egyptian_revo	Originally for Egyptian re	/Disaster_Types/Manmade/Conflict_Based/Revolution	/Disaster_Types/Manmade/C	
4	Collect_yTK	2	yTK	#libya	https://en.wikipedia.org/wiki/Libya	" In the second Libyan Civil War, ongoing since 2014, two authorities initially claimed to govern Libya: the Council of Deputies in Tobruk and the 2014 G			
5	Collect_yTK	3	yTK	#blacksburg	https://en.wikipedia.org/wiki/Blacksburg,_V	" Blacksburg High School	Local	Local	
6	Collect_yTK	4	yTK	#jan25	https://en.wikipedia.org/wiki/Egyptian_revo	January 25th 2011 was t	/Disaster_Types/Manmade/Conflict_Based/Revolution	/Disaster_Types/Manmade/C	
7	Collect_yTK	5	yTK	#bahrain	https://en.wikipedia.org/wiki/Bahrain	" In December 1994, a gr	Building	Building	
8	Collect_yTK	6	yTK	#yemen	https://en.wikipedia.org/wiki/Yemen	" According to the 2009	International	International	
9	Collect_yTK	7	yTK	japan earthquake	https://en.wikipedia.org/wiki/2011_T%C5%8	"This is a list of earthqua	International	International	
10	Collect_yTK	8	yTK	#syria	https://en.wikipedia.org/wiki/Syria	" Syria is ranked last on t	crisis, political, Arab Spring, uprising	political crisis	
11	Collect_yTK	9	yTK	OccupyWallStreet	https://en.wikipedia.org/wiki/Occupy_Wall_	"== Origins ==The origin	War	War	
12	Collect_yTK	10	yTK	#nrv		new river valley (blacksburg)			
13	Collect_yTK	11	yTK	virginia tech	https://en.wikipedia.org/wiki/Virginia_Tech	A 23-year-old student, Se	/Disaster_Types/Manmade/Conflict_Based/School_Shooting	/Disaster_Types/Manmade/C	
14	Collect_yTK	12	yTK	iran earthquake	https://en.wikipedia.org/wiki/List_of_earthq	" As a result, earthquake	Earthquake, Iran	Earthquake	
15	Collect_yTK	13	yTK	diabetes	https://en.wikipedia.org/wiki/Diabetes_mel	health category	Earthquake	Earthquake	
16	Collect_yTK	14	yTK	heart attack	https://en.wikipedia.org/wiki/Myocardial_in	health category	Earthquake	Earthquake	
17	Collect_yTK	15	yTK	foursquare	https://en.wikipedia.org/wiki/Foursquare		social media	geolocation, social media	
18	Collect_yTK	16	yTK	#isaac	https://en.wikipedia.org/wiki/Hurricane_Isa	Hurricane Isaac formed d	/Disaster_Types/Natural/Geophysical/Storm/Hurricane	/Disaster_Types/Natural/Geo	
19	Collect_yTK	17	yTK	turkey syria	https://en.wikipedia.org/wiki/Turkish_occup	violence between Turkey and Syria in Oct. 2012			
20	Collect_yTK	18	yTK	emergency preparednes	https://www.ready.gov/	"Emergency management or disaster management is the organization and management of the resources and responsibilities for dealing with all huma			
21	Collect_yTK	19	yTK	emergency response	https://www.ready.gov/business/implemen	"The emergency services	Response	Response	
22	Collect_yTK	20	yTK	emergency recovery	https://www.fema.gov/media-library/assets/	"Emergency managemen	Response	Response	
23	Collect_yTK	21	yTK	emergency mitigation	https://en.wikipedia.org/wiki/Emergency_m	"Emergency managemen	Response	Response	
24	Collect_yTK	22	yTK	emergency management	https://en.wikipedia.org/wiki/Emergency_m	"Emergency managemen	Response	Response	

Figure 9: Screenshot of known tweet collections [5]

5.5 DMI-TCAT Individual Tweet Converter Implementation

The DMI-TCAT file consists of tweet data broken up into seven different parts, stored in seven different tables, each with its own unique identifier for that table as well as a tweet ID. DMI-TCAT data is stored in MySQL, and as a result of that, the data file is read into MySQL, and afterwards information from each tweet is pulled from the individual tweets, and then converted into JSON schema. The file that reads in DMI-TCAT data consists of a Python script that takes in an SQL file and executes queries in that file in order to store the data to a database. After, another Python script, which connects to the MySQL database, runs some queries in order to retrieve the necessary fields of a tweet from its respective table. Figure 10 shows a sample database used and the tables the database contains, from which the script extracts its information.

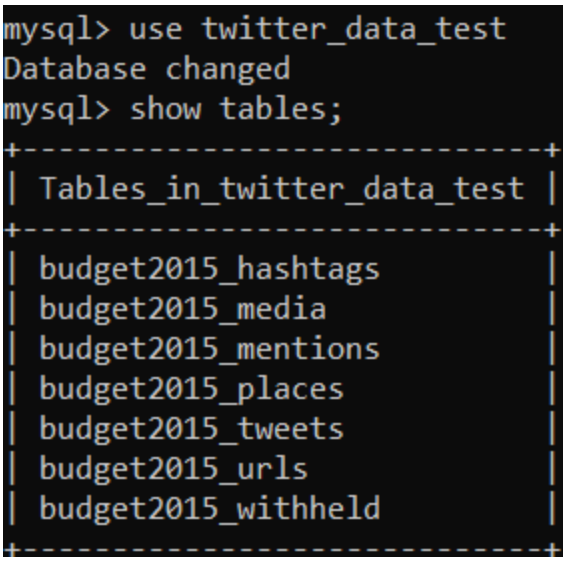


Figure 10: Screenshot of tables in DMI_TCAT_sample_Budget2015.gz

Any value that is specified in the schema but could not be found is set to the null value. The script outputs individual tweet objects following our schema format in a file chosen by the user. The converter script for DMI-TCAT does take a while for conversion, most likely due to accessing all the different tables that make up a single tweet object. Figure 11 shows what the input looks like as well as a sample run of converting 100 tweets.

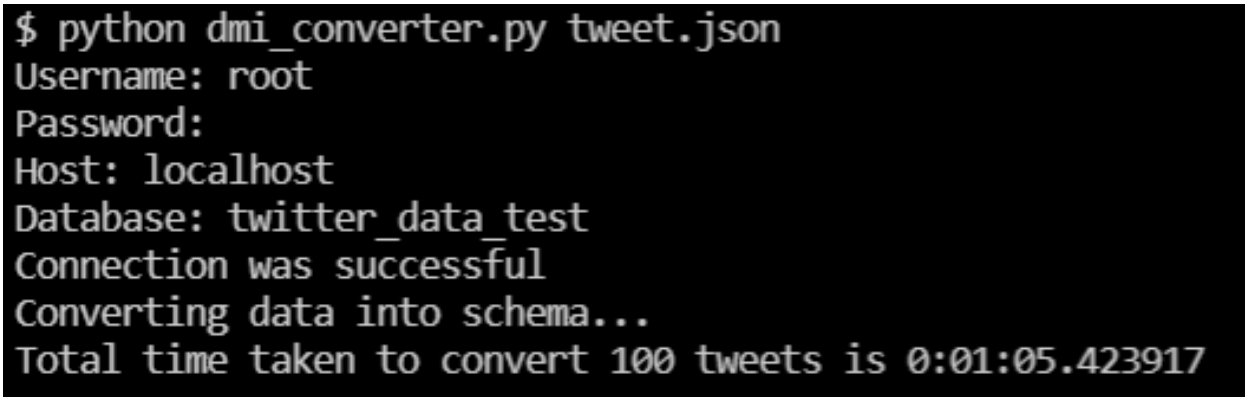


Figure 11: DMI-TCAT input and time taken screenshot

As shown in Figure 11, the time taken to convert 100 tweets to our schema format was 65 seconds. The converted tweets go to a file called tweet.json. Figure 11 also shows that when the converter script is run, it asks for username, password, host, and database. These input fields are specified to MySQL, and the data is read and extracted from the database entered, in this case, twitter_data_test.

5.6 DMI-TCAT Collection-Level Converter Implementation

The DMI-TCAT summarizer, summarize_dmi.py, has the same high-level design as the DMI-TCAT individual tweet converter. It has a simpler implementation because we are only parsing the tweet IDs and any hashtags or keywords that are included in the tweets. These hashtags and keywords are stored in the collection_terms field, so they can be accessed easily for any future research that may be done on the collection. The implementation stores these terms as a set to ensure that no duplicates are listed. Note that this script is only capable of determining the tweet IDs and number of tweets for each collection. This is because DMI-TCAT does not store any collection-level data on its own. This must be added manually or the data should be pulled from the events archive spreadsheet [5].

6.0 Testing

We used three distinct techniques for testing the correctness of our converter scripts: unit tests, validator scripts, and manual inspection. Unit tests were utilized on individual tweets to test whether an individual input tweet was converted by our code; the resulting JSON object exactly matched what we expected based on our schema. Validator scripts are scripts that are run on entire output data files to try and find problems such as incorrect tweet structure and incorrect types or values for the fields. Finally, we used manual inspection in some cases to further verify that our tweet conversion produced the expected results.

6.1 SFM Unit Test Implementation

To help ensure the correctness of the SFM converter implementation, we wrote a unit test file to test specific tweet conversion cases. We used a Python file that utilizes the unittest module so that the tests could be run on the command line. Figure 12 is an example of the unit test output when all test cases pass:

```
$ python test/test_convert_sfm.py
...............
----------------------------------------------------------------------
Ran 15 tests in 0.002s

OK
```

Figure 12: Command-line output of SFM unit test run

The test cases are fairly simple, with each consisting of an input SFM tweet object, an output JSON tweet in the new schema, and an assertion that the former is converted correctly into the latter. See Figure 13 for an example of such a test case.

```
def test_sample_3(self):
    in_datum = {
        "full_text": "RT @stLogesh: Sir my name is Logeshwaran and I'm passed out BE student in anna
        "lang": "English",
        "favorite_count": 10000,
        "retweet_count": -1,
        "metadata": {
            "iso_language_code": "en",
            "result_type": "recent"
        },
        "source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Andro
    }
    out_datum = {
        "created_at": None,
        "id": None,
        "text": "RT @stLogesh: Sir my name is Logeshwaran and I'm passed out BE student in anna unive
        "entities": None,
        "lang": "English",
        "lang_iso_code": "en",
        "source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Andro
        "in_reply_to_user_id": None,
        "user": None,
        'geo': {
            'country': None,
            'latitude': None,
            'longitude': None
        },
        'metrics': {
            'favorite_count': 10000,
            'retweet_count': -1
        },
    }
    self.assertEqual(out_datum, convert(in_datum))
```

Figure 13: SFM unit test case example

Our test cases primarily focus on testing that the code works properly when fields are excluded, testing that it works correctly when fields are misspelled, testing that it correctly adds fields that were missing in the SFM tweet, and making sure that data is copied over correctly on real tweets with lots of non-null data.

6.2 Individual Tweet Validator Script

As part of testing, we wrote a script that takes a file containing our converted JSON individual tweet data as input and performs sanity checks on each object in the file using Python assert statements. For each of the fields that are expected to be in the tweet object, we check the type of the field, ensuring that it is either the expected field type or null (for missing fields). For example, the “is_quote_status” field must be Boolean or null. A few fields, such as the hashtags field, are not permitted to be null (although hashtags can be an empty list), so we check that these fields are non-null. We also check the type of each element in the list fields. Additionally, we check that the latitude and longitude values are in their appropriate ranges. This is necessary to ensure we didn’t accidentally mix up latitude and longitude.

The validator script also checks the tweet structure at the same time it performs the type checks. This is because, as it accesses each field it tries to type check, it allows any exceptions that might

result from incorrect structure (e.g., accessing a field that doesn't exist in the requested part of the tweet object) to propagate up and terminate the script. The lack of such exceptions indicates that every tweet object had all the expected fields, organized in the correct way.

6.3 Collection-Level Validator Script

The collection-level validator works similarly to the individual tweet validator in that it takes a file containing converted data as input and performs sanity checks on each JSON object. However, the collection-level validator works on collection-level JSON objects rather than ones for individual tweets. Like the individual tweet validator, this validator ensures that each field is either of the expected type or null. Similarly, it uses the same tactic of attempting field accesses and allowing any resulting exceptions to propagate up, so it tests the structure of the tweet object as well.

In addition to type and structure checks, the collection-level validator checks that the tweet count field is at least one and that the number of tweet IDs in the tweet ID list matches the tweet count. It checks that every tweet ID is the string representation of an integer, since every input tweet ID from our current data is in integer form (although this might have to be edited if anyone generalizes the ID field in the future). Additionally, it checks that the list of hashtags in the collection is free of duplicates.

6.4 Manual Inspection

There is another technique we used to test the correctness of our output JSON data. Using multiple random smaller sets of data from the database or input JSON file, we can manually compare the input and the intended output of the script to see if it outputs the correct values and data type for each field for specific tweets. Using many tweets pulled from the database, with different edge cases, we compare with the output that the script gave us to inspect the expected result.

7.0 User's Manual

7.1 Use Environment Discussion

The command lines specified by the following requirements use the Linux/GitBash versions of commands; if the user is on a different operating system, alter the command line commands to that operating system's equivalent of the commands.

7.2 Use Cases/Tasks Supported

Our codebase is capable of supporting a variety of different tasks, but these can mostly be put into three categories: data conversion, data validation, and data utilization.

7.2.1 Data Conversion Use Cases/Tasks Supported

The following instructions give detailed instructions on how to perform tweet/collection conversion using our scripts.

7.2.1.1 SFM Individual Tweet Conversion Use Cases

Given an input SFM file with one JSON object per line (like the sample SFM data we were given), individual SFM tweets can be converted and stored in a new file using the following steps:

1. Decide on a file path to which the output JSON objects should be written.
2. Move to the directory of the 'convert_sfm.py' script, if you are not there already.
3. Enter the command line 'python convert_sfm.py [sfm_filepath] > [output_filepath]', where [sfm_filepath] is the path to the input SFM file and [output_filepath] is the path to the new file to which output JSON objects should be written.
4. Open [output_filepath]. There should now be JSON tweet objects in the new schema in this file, with one object per line.

To read SFM data and add the converted tweet objects to an existing file without overwriting the tweet objects in that file, perform the same steps as above, but enter 'python convert_sfm.py [sfm_filepath] >> [output_filepath]' at the command line instead.

7.2.1.2 SFM Collection-Level Conversion Use Cases

Given an input SFM file with one JSON object per line (like the sample SFM data we received), collection-level metadata can be converted and stored in a new file using the following steps:

1. Get all the SFM tweet data for which you want to create a JSON tweet metadata object into one file:
    a. If the desired tweet data is already in one file, do nothing.
    b. If the tweet data is spread across multiple files, use file concatenation to put all the SFM data into one file. This works because the SFM data is in one object per line format. There are many possible ways to do file concatenation, but one method is 'cat [file2] >> [file1]'. This concatenates [file2] onto the end of [file1].
    c. If the tweet data is in a subset of one file, use some form of line selection to put all the SFM data into one file. This works because the SFM data is in one object per line format. For one example of how to do this, one can use 'head -n [nlines] [file1] > [file2]' to put the first [nlines] SFM objects from [file1] into [file2]. Note that this overwrites [file2].
2. Decide on a file path to which the output JSON objects should be written.
3. Move to the directory of the 'summarize_sfm.py' script, if you are not there already.
4. Enter the command line 'python summarize_sfm.py [sfm_filepath] > [output_filepath]', where [sfm_filepath] is the path to the input SFM file and [output_filepath] is the path to the new file to which output JSON objects should be written.
5. Open [output_filepath]. There should now be a JSON collection object in the new schema in this file on the first line.

Since SFM data comes without collection metadata beyond what can be gleaned from analyzing the provided tweet objects themselves, the SFM conversion script assumes that the SFM tweets together in its input file constitute one SFM collection, and the output JSON object it produces is arranged accordingly.

One important thing to note is that although one run of the SFM collection conversion script only produces a single JSON object representing a collection, multiple JSON collection-level objects in the final schema can be moved into one file (by file concatenation, for instance), and the

‘validate_col_data.py’ script will continue to work. This means that a large amount of collection metadata can be stored compactly in a way that still facilitates the use of the validator script.

7.2.1.3 YTK Individual Tweet Conversion Use Cases

This script requires YTK data to be stored in a MySQL database. If not already done, the Python script read_sql.py must be run using “python read_sql.py <ytk_file>”. To use the ytk_mysql_to_json.py script, the following steps should be followed:

1. Navigate to the directory in which the converter script is located in. Make sure MySQL is installed.
2. In the terminal, type the command ‘python ytk_mysql_to_json.py’. It will automatically generate the output files based on each table in the YTK data.
3. When the command is entered, the script will ask for username, password, hostname, and database. The username and password are for the account of MySQL. Hostname is the hostname of the connection, and database is the name of where the tweet data is stored. If all information is entered correctly, the script will print in the terminal “Connection was successful.” If not, then there will be an error.
4. Once a connection is established, a message will be printed in the terminal saying “Fetching all the table names...”. The script does take some time depending on the number of tweets converting as that information is given on Table 11. The script is finished when the total time taken to do the conversion is displayed along with a message affirming that this script is finished running.
5. In that same directory, the number of output files in JSON will be generated, with the name “table_<tablename>” giving the table name that it extracted data from.

7.2.1.4 YTK Collection-Level Conversion Use Cases

This script requires YTK data to be stored in a MySQL database. If not already done, the Python script read_sql.py must be run with “python read_sql.py <ytk_file>”. To use the ytk_collection converter.py script, the following steps should be followed:

1. Navigate to the directory where the script is located. Make sure MySQL is installed.
2. Make sure the file Collection_Table_for_IA_-_rich_collections.csv is also in the same directory.
3. In the terminal, run the command ‘python ytk_collection_converter.py Collection_Table_for_IA_-_rich_collections.csv’. It will generate an output file with the collection level data.

4. When the command is entered, the script will ask for a username, password, and database. The username and password are for the account for MySQL. Make sure that the database entered is the same as the database used for the YTK Individual Tweet Conversion. If all information is entered correctly, the script will print in the terminal "Going through each collection:" with a percentage next to it. Once this percentage reaches 100, the script will complete and an output file will be created. Otherwise, an error message will appear.

7.2.1.5 DMI-TCAT Individual Tweet Conversion Use Cases

This script requires DMI-TCAT data to be stored in a MySQL database. If not already done, the Python script dmi_reader.py must be run with "python dmi_reader.py <dmi_file>". To use the dmi_converter.py script, the following steps should be followed:

1. Navigate to the directory in which the converter script is located in. Make sure MySQL is installed.
2. In the terminal, run the command 'python dmi_converter.py <output_file.json>' where <output_file.json> represents the file name in which the data is printed. The command 'python dmi_converter.py' can also be used, in which case a file with the name 'dmi-tcat_ouput.json' will be automatically generated and will store all the converted data. The data in this file will be overwritten if it exists.
3. When the command is entered, the script will ask for username, password, hostname, and database. The username and password are for the MySQL account. Hostname is the hostname of the connection, and database is the name of where the tweet data is stored. If all information is entered correctly, the script will print in the terminal "Connection was successful" and continue to run, otherwise an error message will be raised and printed in the terminal.
4. Once a connection is established, a message will be printed in the terminal saying "Converting data into schema…"
    a. The script is finished when the total time taken to do the conversion is displayed.
    b. The script has a limit on the number of tweets to convert from a given data table, set in place for testing purposes. To convert all tweets in the database, limits can be removed by commenting out lines 362-364, and the script will run normally.

It has been assumed that the formats of all databases are the same, meaning that tables in databases have keywords in order to extract information from.

7.2.1.6 DMI-TCAT Collection-Level Conversion Use Cases

Like with the DMI-TCAT individual use cases, this script requires the DMI-TCAT data to be stored in a MySQL database. If not already done, the Python script dmi_reader.py must be run with "python dmi_reader.py <dmi_file>". To run the summarize_dmi.py script, the following steps need to be taken:

1. Navigate to the directory in which the converter script is located in. Make sure MySQL is installed.
2. In the terminal, run the command 'python summarize_dmi.py <output_file.json>' where <output_file.json> represents the file name in which the data is printed. The command 'python dmi_converter.py' can also be used, in which case a file with the name 'dmi-tcat_collection_output.json' will be automatically generated, storing all the converted data. The data in this file will be overwritten if it exists.
3. When the command is entered, the script will ask for username, password, hostname, and database. The username and password are for the account of MySQL. Hostname is the hostname of the connection, and database is the name of where the tweet data is stored. If all information is entered correctly, the script will print in the terminal "Connection was successful" and continue to run, otherwise an error message will be raised and printed in the terminal.
4. Once a connection is established, a message will be printed in the terminal saying "Converting data into schema…"
    c. The script is finished when the total time taken to do the conversion is displayed.

7.2.2 Data Validation Use Cases/Tasks Supported

In addition to generating data, users might be interested in validating the data for correctness. This is important to detect data corruption or other issues that could be present in a large dataset without users being aware of it. Additionally, these validator scripts can be used to ensure that future versions of this codebase continue to produce correct, schema-following tweet and collection JSON objects.

Suppose the user is interested in validating a file containing JSON objects representing individual tweets that are supposed to follow the individual tweet schema we created. To validate the final schema JSON tweet objects in a given file, perform the following steps:

1. Move to the directory containing the 'validate_ind_data.py' Python script, if you are not there already.
2. On the command line, execute 'python validate_ind_data.py [data_file]'
3. If there is no output, the validator script detected no problems with any of the tweet objects in the file.

4. If an assertion failure or exception occurs:
    a. At least one tweet object in the input file had a field that didn't match the schema in some way. This could be due to the field being missing, being of the wrong type, being in the wrong place in the JSON object hierarchy, or in some cases (such as with latitude and longitude) having a value inconsistent with the field's logically possible range.
    b. Examine the line in the validator script where the exception occurred to determine what type of failure there was.

Note that for this script to pass without exceptions, the input data must follow the format of one JSON object per line and contain objects that follow the final individual tweet schema. The file can contain an arbitrary number of JSON tweet objects. Importantly, the script will work regardless of what original data source (SFM, YTK, or DMI-TCAT) the tweets were originally converted from, as long as all the tweets follow our final schema properly. In fact, this script is a very helpful way to check that future modifications to our conversion scripts result in JSON objects that have consistent formatting regardless of the source from which they were converted.

Suppose the user is interested in validating a file containing JSON objects representing collection-level metadata that are supposed to follow the collection-level schema we created. To validate the final schema JSON collection in a given file, perform the following steps:

1. Move to the directory containing the 'validate_col_data.py' Python script, if you are not there already.
2. On the command line, execute 'python validate_col_data.py [data_file]'
3. If there is no output, the validator did not detect any problems with any of the collection-level data in the file.
4. If an assertion failure or exception occurs:
    c. At least one tweet object in the input file had a field that didn't match the schema in some way. This could be due to the field being missing, being of the wrong type, being in the wrong place in the JSON object hierarchy, or in some cases having a value inconsistent with the field's logically possible range.
    d. Examine the line in the validator script where the exception occurred to determine what type of failure there was.

Note that for this script to pass without exceptions, the input data must follow the format of one JSON object per line and contain objects that follow the final collection-level schema. The file can contain an arbitrary number of JSON collection-level objects. Importantly, the script will work regardless of what original data source (SFM, YTK, or DMI-TCAT) the collection metadata objects were originally converted from, as long as the data follow our final schema

properly. In fact, this script is a very helpful way to check that future modifications to our conversion scripts result in JSON objects that have consistent formatting regardless of the source from which they were converted.

In addition to using the validator scripts, users might be interested in running the unit tests for the SFM individual tweet converter. This requires the following steps:

1. Move to the parent of the 'test' directory, if not already there.
2. Execute 'python test/test_convert_sfm.py' on the command line.
3. If the last word of output is 'OK', all the tests passed.
4. If a test failed, open the 'test_convert_sfm.py' file, find the test case that failed, and diagnose the problem.

7.2.3 Data Utilization Use Cases/Tasks Supported

In addition to the data conversion and data validation use cases, there are a few different use cases supported related to the utilization of the output data. One example would be to perform sentiment analysis on the text of the tweets, and would proceed as follows:

1. Import the Python JSON library.
2. Read in the data file using the open() function.
3. Pass each line of the file contents to the json.loads() function to create JSON objects.
4. Access the text field of each tweet object.
5. Pass each of these text fields to a preferred sentiment analysis model (e.g., deep neural network).
6. Generate and display output.

Another example use case could be to determine the concentration of Twitter users in each geographic area. This use case will start the same way as the first three steps of the previous use case, and will then proceed as follows:

1. Filter out tweet objects with null data in the geo dict.
2. For each geographic region of interest, tally the number of distinct users encountered among the tweets within that geographic region. The geographic region can be determined either as a function of the coordinate data or through the country field in the geo component of the JSON object.
3. Display the tallies for each geographic region.

Finally, an additional use case could be to track the prevalence of tweet hashtags over time. This use case will start the same way as the first three steps of the first use case, and will then proceed as follows:

1. Compile a list of hashtags of interest to track over time.
2. For each tweet, access its list of hashtags through the ‘hashtags’ list field in the ‘entities’ dictionary.
3. Iterate through the hashtag list, tallying which (if any) hashtags of interest the tweet contains.
4. Find the date when the tweet was created using the ‘created_at’ field. Generate a record giving two values, hashtag and date, for each such pair identified. Sort and tabulate all of the generated records into what can be used for a histogram or plot.
5. Generate a plot with tweet dates on the x-axis and the number of tweets containing each hashtag of interest on the y-axis.

Many other use cases are possible, but these should be sufficient to show the usefulness of our deliverables.

7.3 Tutorials on Use

The only library required to make use of our deliverables is the JSON library, although if one wants to produce more data using the YTK or DMI-TCAT converters, it is also necessary to install MySQL. Fortunately, JSON is already part of the Python standard library, so any users that already have Python installed don’t need to install JSON separately. However, MySQL requires a separate installation.

Our provided data is in the JSON format, which makes it easy to read in and do analysis on. Just like with SFM, each line in our data files contains a single JSON object. Each JSON object refers to an individual tweet in the individual tweet data and a collection in the collection-level data.

Regardless of which data type is being utilized, the process for reading it is similar. Use the file handling API of your preferred programming language to iterate through the lines in the JSON file, and pass each line to a JSON object constructor like Python’s json.loads() function to obtain a JSON object representing the tweet or collection data. The data can then be accessed as attributes within the object returned. Some values may be null, so remember to check for those during parsing.

For help on understanding the structure of our JSON objects and the properties of each field, consult the JSON schemas we have provided as deliverables to this project. To get specific

examples of JSON objects that appear in our data, one can simply read in and print out lines from our data files. To make this output easier to read, a pretty printer can be used, for instance by passing indent=4 to the Python json.dump() or json.dumps() function when the JSON object is serialized for printing.

8.0 Developer's Manual

8.1 Inventory of All Files

Tables 4-10 detail all of the files that are present in our repository.

Table 4: Input data files

Filename	Description	Format
sfm_sample_collection_covid_library.json	Sample file of SFM data.	Each line contains an JSON object following the SFM schema representing a single tweet, and there are exactly 200,171 tweet objects represented.
ytk2-eom_all_mysql.sql	Sample file of YTK data.	The file takes the form of a series of SQL commands. These form a YTK database.
DMI_TCAT_sample_Budget2015.gz	Sample file of DMI-TCAT data.	The file takes the form of a series of SQL commands, like a .sql file. These form a DMI-TCAT database.
tweet_json_response_examples.json	Example TwitterV2 tweets we moved into the repository early on when we were considering what fields to include in the schema.	A JSON list containing two JSON tweet objects that follow the TwitterV2 schema.
Collection_Table_for_IA_-_rich_collections.csv	File of known tweet collections found in the tweet data across all three sources	Comma Separated Values

Table 5: Output data files

Filename	Description	Format
dmi_sample_output.json	Sample output created from running dmi_converter.py script.	The output file contains JSON tweets in the new schema, printed with an indent level of 4.

ytk_collection.json	JSON file of the outputted collection level data from the YTK tweet data	The output file contains 1 JSON tweet object in the new schema per YTK archive per line.

Table 6: SFM script files

Filename	**Command to use**	**Description**
test/test_convert_sfm.py	$ python test/test_convert_sfm.py	Unit test file for the SFM tweet converter.
read_sfm.py	$ python read_sfm.py <json_filename>	Python script that reads in an SFM JSON file and prints out the first two JSON objects.
convert_sfm.py	$ python convert_sfm.py <sfm_filename> > <output_file>	Python script that converts an SFM tweet data file into individual tweet data in the new schema.
summarize_sfm.py	$python summarize_sfm.py <sfm_filename> > <output_file>	Python script that takes an SFM tweet file as input and creates a collection-level JSON object describing the collection.

Table 7: YTK Script Files

Filename	**Command to use**	**Description**
read_sql.py	$ python read_sql.py <ytk_filename>	Python script that reads in and prints a YTK SQL file, including the tables in the database and the first few entries of the ‘archives’ table. This script is supplemental for debugging purposes. It is a good way to test whether you are using the right MySQL username and password when troubleshooting.

		In most cases, this command should run fine as it is, but in some environments (such as Windows Git Bash), it must be run with 'winpty python' instead of just 'python' to prevent the script from hanging on the call to getpass(). This script requires entering a MySQL username and password.

ytk_mysql_to_json.py	$ python ytk_mysql_to_json.py
	
	OR
	
	$ winpty python ytk_mysql_to_json.py
		Python script that converts an YTK file of tweet data into tweet data in the new schema.

ytk_collection_converter.py	$ python ytk_collection_converter.py Collection_Table_for_IA_-_rich_collections.csv	The file will ask for the username and password of the same MySQL user for the read_sql.py file The file will also ask for a database in MySQL to read tweet data from.

Table 8: DMI-TCAT Script Files

Filename	**Command to use**	**Description**
dmi_reader.py	$ python dmi_reader.py <dmi_tcat_filename>	Stores data into MySQL database. Asks for username, password, hostname, and database to store in.

dmi_converter.py	$ python dmi_converter.py <output_file>
	OR
	$ python dmi_converter.py
		Python script that connects to a MySQL database and converts given data into the new schema. Asks for username, password, hostname, and database of tweets.

summarize_dmi.py	$ python summarize_dmi.py <output_file>	Python script that is used to collect all of the tweet IDs in a collection. It outputs a single JSON object that follows the collection level schema to a file that compiles a list of all of the tweets in a collection.

Table 9: Validator Script Files

Filename	**Command to use**	**Description**
validate_ind_data.py	$ python validate_ind_data.py <json_filename>	Python script that is used for validating output JSON tweets in the final schema we designed. If this script produces an exception or an assertion fails, then a tweet field was missing, in the wrong location, had the wrong type, or had some other problem. Check the script to find the source of the exception/assertion failure to diagnose the issue.
validate_col_data.py	$ python validate_col_data.py <json_filename>	Python script that is used for validating output JSON collection objects in the final schema we designed. If this script produces an exception or an assertion fails, then a tweet field

		was missing, in the wrong location, had the wrong type, or had some other problem. An assertion will also fail if the tweet count didn't match the number of tweet ids in a collection or if there were duplicates in the hashtags list (duplicates here unnecessarily waste space). Check the script to find the source of the exception/assertion failure to diagnose the issue.

Table 10: Miscellaneous code files

Filename	**Command to use**	**Description**
README.md	N/A	Brief explanation of the project
SFM_tweet_schema.zeppelin.output	N/A	JSON schema file describing the SFM format followed by the provided SFM tweets.
read_tweet_examples.py	$ python read_tweet_examples.py <twitter_json_filename>	Python script that reads in JSON tweets in the TwitterV2 schema
util/empty.py	N/A	Contains three JSON objects with all empty fields. After importing and deep copying these JSON objects, users can fill in only the non-null fields to avoid having to manually assign all unused fields to null. For an example of this pattern, see test/test_convert_sfm.py lines 422-452. The tweet objects are: ● empty_collection (empty collection-level tweet object) ● empty_output (empty individual tweet object)

		• empty_user (empty user object to be nested in the individual tweet object) When copying an empty object, use the function deepcopy() from the Python 'copy' module instead of the copy() method of the empty object to perform a deep copy rather than a shallow copy.

8.2 Dependencies

All of the scripts were written in Python 3. The scripts include dependencies to certain Python libraries outside of the Python Standard Library that will need to be installed via pip from the Python Package Index [6]. The complete list of installation dependencies is as follows:

- mysql-connector-python 8.0.27
- pandas 1.3.4
- PyMySql 1.0.2

8.3 Repository Structure

Our code repository is relatively simple, but there are a few conventions we have been following that might make it easier for future developers if they continue to follow them.

There are only two directories in the repository, 'test' and 'util'. The 'test' directory is for unit tests and now only contains the SFM individual tweet converter unit test, 'test_convert_sfm.py'. If future developers create more unit tests, we recommend that they be placed here. The 'util' directory is for auxiliary functions and data and now only contains 'empty.py', which has empty JSON objects representing tweet collections, individual tweets, and Twitter users. Importing and deep copying objects from this file can help reduce the need to manually set a lot of unused fields to null when performing data conversions.

Right now, all other files are stored right under the repository root.

9.0 Lessons Learned

Following are some lessons we have learned and challenges we have faced. Section 9.1 gives the initial timeline we set up at the beginning of the semester. It involves having 8 milestones that represent our progress in this project. We planned to meet the client once every two weeks, but once per week in the last few weeks of the schedule. We kept regular contact with the client through emails if we needed directions in the project.

9.1 Timeline and Schedule

- **Week of 9/7:** Figure out how to read in collected data (YTK and SFM) into a Python program; review v2 schema; come up with schema options, and put together presentation; assign team roles and responsibilities
- **Week of 9/13:** First presentation detailing project outline, timeline, and deliverables
- **Week of 9/20:** Complete script for reading in YTK data, begin developing format for JSON collection-level schema. Milestone 1 completion
- **Week of 9/27:** Complete script for reading in SFM data, continue developing JSON collection-level schema. Milestone 2 completion
- **Week of 10/4:** Complete script for reading in DMI-TCAT data, continue developing JSON collection-level schema. Milestone 3 completion
- **Week 10/11:** Finalize JSON collection-level schema, begin developing format for JSON individual tweet schema. Milestone 4 completion
- **Week of 10/18:** Begin working on how the current tweet data translates into the prototype schema, continue developing JSON individual tweet schema
- **Week of 10/25:** Continue developing JSON unified tweet master schema, begin developing script for converting tweet data to new schema
- **Week of 11/1:** Second presentation detailing progress on project and what still needs to be completed, make necessary changes to timeline, prepare and complete interim report
- **Week of 11/8:** Complete development of JSON individual tweet schema, complete scripts for converting tweet data to new schemas. Milestones 5, 6 and 7 completion
- **Week of 11/15:** Test script for converting tweet data to the JSON master schemas, create set of converted JSON tweets data. Milestone 8 completion
- **Week of 11/29:** Prepare final presentation detailing project outcomes and providing the deliverables, prepare final report

- **Week of 12/6:** Complete final report, project completion

Although we generally completed tasks in the order listed on the timeline, things came up that resulted in exceptions to this order:

- We experienced challenges as mentioned in Section 9.2 with converting our data, so we had to delay completing the conversion scripts and testing the data. Thus, from the initial timeline above, the weeks from 11/15 to 12/6 had to be adjusted.
- We expected to create and complete the collection level schema first, however, the individual tweet schema was actually completed first before moving onto collection level schema.
- When actually completing the project, the SFM individual and collection-level tweet conversion script ended up being completed earlier than the other conversion scripts. The DMI-TCAT individual and collection-level tweet conversation script ended up being done much later in the schedule due to complexity of it compared to other scripts we had to work on.

9.2 Challenges

The challenges faced by the team include the actual conversion of the data to a new schema. The team faced the challenge of having to convert MySQL to JSON as well as extracting all information for a single tweet from MySQL due to tweet data being broken into multiple tables. Specifically, for DMI-TCAT, the main challenge was putting the tables together as tweet data was broken up into sections. As a result of this challenge, the approach taken to gather all information of a single tweet was to access each table individually and pull the information out based on tweet ID. Because of this, data for each tweet was accessed directly from the MySQL database and converted to the JSON schema.

For SFM (which is already in JSON), one challenge was that almost all of the fields are nullable including fields that have subfields and list elements. This means that the parser code has to constantly perform null checks when attempting to access fields. Additionally, there appeared to be a lot of cases where the fields were entirely missing rather than null, which also requires an explicit check to avoid KeyErrors upon attempting to access the field value.

Another challenge was that each SFM tweet object is very large and therefore a large number of fields must be copied over into the new tweet schema to preserve all the important information in SFM. This makes writing unit tests cumbersome because the input and output example tweets may have a very large number of fields each.

We also faced challenges when coming up with the JSON individual tweet schema. SFM had significantly more fields than both the DMI-TCAT and YTK formats, which means that preserving the important SFM data requires including a lot of setting fields to null in the DMI-TCAT and YTK conversion scripts. This effort can be mitigated by providing an example of an empty tweet in the new schema, so that DMI-TCAT and YTK scripts can each import this tweet and only fill in the fields that are actually present in their respective input data formats.

10.0 Future Work

While there has been progress made in this project, there are possible improvements and future work that could be done. Further improvements in the performance of the scripts could be implemented to help improve the rate at which the data can be converted. Table 11 shows the current runtimes of our scripts on the sample data we were given by our client:

Script	**Runtime**	**Tweets Processed**	**Environment**
SFM Individual	27.999s	200,171	CentOS Linux 7, bash, 15 GB RAM
SFM Collection	13.969s	200,171	CentOS Linux 7, bash, 15 GB RAM
YTK Individual	2m 59.065s	750,052	CentOS Linux 7, bash, 15 GB RAM
YTK Collection	41.804s	750,052	CentOS Linux 7, bash, 15 GB RAM
DMI-TCAT Individual	73m 18.707s	98,171	CentOS Linux 7, bash, 15 GB RAM
DMI-TCAT Collection	3m 30.895s	98,171	CentOS Linux 7, bash, 15 GB RAM

Table 11: Script runtimes

This would help with converting at least 6 billion tweets that the library should receive from this effort, and possibly more future tweet data from the three collection systems if data continues being collected.

There is still work that needs to be done to finish the implementation of the collection level converters. This is because during the semester we were unaware we had to tie each collection to the spreadsheet found on the events archive site [5]. At this time, our scripts only compile a list of the tweet IDs and the rest of the collection level schema is left blank (i.e., null). To remedy this, some strategy must be devised to link the rows of the spreadsheet that give the collection level schema information such as collection ID, collection name, collection description, wikipedia article, etc. to the list of compiled tweet IDs that are extracted from the data. One possible strategy that was discussed with the client is to use the keywords that are found in the tweets to match them up to which collection they belong in.

We were informed that it would be cleaner to have a third schema that maps the collection IDs to tweet IDs, rather than storing the collection ID-tweet IDs mapping in the collection-level schema via the tweet IDs field in that schema. To make this change, one could delete the tweet ID field from the collection-level schema, modify the collection-level conversion scripts to not process this field, and add new scripts that, given a tweet collection, can collect the tweet IDs in the collection and output a collection ID-tweet IDs mapping. This mapping could be stored in JSON format, like the current collection objects, or in a MySQL file.

The next task for the future would be to use our scripts we have implemented and start converting the tweet data into the new schema.

11.0 Acknowledgements

Xinyue Wang: Client
Email: xw0078@vt.edu

Dr. Edward Fox: Professor
Email: fox@vt.edu

12.0 References

[1] yourTwapperKeeper. (2013, May). yourTwapperKeeper - Archive Your Social Media. Retrieved December 8, 2021, from https://github.com/540co/yourTwapperKeeper

[2] DMI-TCAT. (2016, September). Digital Methods Initiative - Twitter Capture and Analysis Toolset. Retrieved December 8, 2021, from https://github.com/digitalmethodsinitiative/dmi-tcat

[3] Prom, C. (2017). Tool Report: Social Feed Manager. *MAC Newsletter, 45*(2). Retrieved December 8, 2021, from https://dr.lib.iastate.edu/server/api/core/bitstreams/0a0cc6eb-8843-47fd-bcc1-aa193059244d/content

[4] Twitter. (2021). *Tweet object | docs | Twitter developer platform*. Twitter. Retrieved November 11, 2021, from https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/tweet.

[5] Virginia Tech Digital Library Research Laboratory. (2021) *Events archiving:* Events Archive Invitation, Funding | Events Archiving. Retrieved November 11, 2021, from http://eventsarchive.org/.

[6] Python Package Index - PyPI. (2021). Python Software Foundation. Retrieved December 1, 2021, from https://pypi.org/