Gaza Ceasefire X (Twitter) Data Collection Documentation

Ilyas Ibrahim Mohamed

January 16, 2025

1 Introduction

This document outlines the process used to collect tweets related to the ongoing Gaza ceasefire discussions. The objective was to capture tweets in English that mention key terms such as 'gaza,' 'ceasefire,' 'Israel,' and 'Hamas,' over a 15-hour period.

We subdivided this 15-hour period into five 3-hour windows. Each window retrieved up to 1,450 tweets for a combined total of up to 7,250 tweets. This approach helps ensure coverage across multiple time segments and avoids skewing the dataset by pulling only the most recent tweets (which could happen if we tried to retrieve all 7,250 tweets in a single query)."

2 Environment Setup and Imports

In this section, we configure our Python environment and load necessary dependencies.

- `os` and `dotenv`. Used to securely access environment variables such as the Twitter Bearer Token.
- `requests`. For making HTTP GET requests to Twitter's API.
- `csv`. To store fetched tweets in CSV format.
- `datetime` and `timedelta`. For working with date ranges and time windows.
- `time`. Used to manage pause intervals (sleep) between consecutive Twitter API calls in order to respect rate limits.

We also define our search query and the Twitter API endpoint.

[33]:
```
# ========== Section 1: Environment Setup and Imports ==========

import os
import csv
import requests
import time
from datetime import datetime, timedelta
from dotenv import load_dotenv

# 1.1) Load environment variables from .env file
# This file contains the Twitter Bearer Token, ensuring it remains secure.
load_dotenv()
BEARER_TOKEN = os.getenv("BEARER_TOKEN")
```

```python
# 1.2) Define our search query
# This query searches for English tweets (lang:en) mentioning any of:
#   - "gaza" AND "ceasefire agreement"
#   - "israel" AND "hamas" AND "ceasefire agreement"
#   - "gaza" AND "ceasefire"
#   - Or simply "gaza"
# We can optionally exclude retweets or quotes by adding -is:retweet, etc.
query = (
    '(gaza "ceasefire agreement") OR '
    '(israel hamas "ceasefire agreement") OR '
    '(gaza ceasefire) OR gaza lang:en'
)

# 1.3) Define the Twitter API v2 Recent Search endpoint
SEARCH_URL = "https://api.twitter.com/2/tweets/search/recent"
```

3 Fetch Tweets Function

The function `fetch_tweets()` handles communication with the Twitter API's **Recent Search** endpoint. It uses Expansions to return additional fields about the original tweets and the users who authored them.

[34]:
```python
# ========== Section 2: Fetch Tweets Function ==========

def fetch_tweets(query, start_time, end_time, next_token=None, max_results=100):
    """
    Fetch tweets via Twitter's 'recent search' endpoint for a given time window.

    This function supports pagination through the 'next_token' parameter and
    retrieves expansions such as original tweet information and author details.

    Args:
        query (str): The full Twitter API query string (including operators).
        start_time (str): The start time in ISO 8601 format.
        end_time (str): The end time in ISO 8601 format.
        next_token (str, optional): The pagination token returned by Twitter if
            more results are available. Defaults to None.
        max_results (int): Maximum tweets per request (up to 100). Defaults to␣
 ↪100.

    Returns:
        dict: A parsed JSON response including:
            - 'data': the tweets in this page
            - 'includes': expansions (users, referenced tweets)
            - 'meta': metadata (e.g., next_token for pagination)
    """
    # 2.1) Prepare HTTP header with Bearer Token for authentication
    headers = {
        "Authorization": f"Bearer {BEARER_TOKEN}"
    }

    # 2.2) Prepare the request parameters
    # expansions=author_id -> referencing tweet's author
    # expansions=referenced_tweets.id -> original tweet object for quotes/
↪retweets
    # expansions=referenced_tweets.id.author_id -> author of the referenced␣
↪tweet
    # tweet.fields -> includes public metrics, creation time, language, etc.
    # user.fields -> includes follower counts and usernames for authors
    params = {
        "query": query,
        "max_results": max_results,
        "start_time": start_time,
        "end_time": end_time,
        "tweet.fields": (
            "author_id,public_metrics,created_at,lang,conversation_id,text,"
            "referenced_tweets"
        ),
        "expansions": "author_id,referenced_tweets.id,referenced_tweets.id.
↪author_id",
        "user.fields": "public_metrics,name,username"
    }

    # 2.3) If we have a next_token from a previous call, include it
    if next_token:
        params["next_token"] = next_token

    # 2.4) Make the GET request to the Twitter API
    response = requests.get(SEARCH_URL, headers=headers, params=params)
    response.raise_for_status()  # Raise an HTTPError if a bad request (e.g.,␣
↪429 rate limit)

    # 2.5) Return the parsed JSON
    return response.json()
```

4 Combine User Info from Multiple Pages

When fetching more than 100 tweets, we must paginate through multiple API responses. Each page includes some subset of user data in the `includes["users"]` field. The helper function `collect_all_user_info()` merges these user objects from all pages into a single dictionary keyed by `user_id`. This allows us to capture as many unique authors and their metrics (e.g., followers count) as possible.

[35]:
```python
# ========== Section 3: Combine User Info from Multiple Pages ==========

def collect_all_user_info(response_pages):
    """
    Combine user info from multiple response pages into a single dictionary.

    Args:
        response_pages (list): A list of JSON responses from 'fetch_tweets'
            calls, each containing possible 'includes["users"]' data.

    Returns:
        dict: A map of user_id -> {
            'username': str,
            'followers_count': int
        }
    """
    combined_user_info = {}

    for page_json in response_pages:
        includes = page_json.get("includes", {})
        if "users" in includes:
            for u in includes["users"]:
                combined_user_info[u["id"]] = {
                    "username": u.get("username", ""),
                    "followers_count": u.get("public_metrics", {}).
 ↪get("followers_count", 0)
                }

    return combined_user_info
```

5 Combine Original Tweets from Expansions

For tweets that reference other tweets (such as retweets or quotes), the original tweet object appears in `includes["tweets"]`. By merging data across pages, we ensure we have a single dictionary mapping `original_tweet_id -> original_tweet_object`.

[36]:
```python
# ========== Section 4: Combine Original Tweets from Expansions ==========

def collect_all_includes_tweets(response_pages):
    """
    Merge 'includes' tweet objects from all pages into a single dict,
    keyed by tweet_id -> tweet_object.

    Args:
        response_pages (list): List of JSON responses, each potentially
            containing 'includes["tweets"]' data.
    Returns:
        dict: A map of tweet_id -> original tweet object.
    """
    combined_includes = {}

    for page_json in response_pages:
        includes = page_json.get("includes", {})
        if "tweets" in includes:
            for tw in includes["tweets"]:
                combined_includes[tw["id"]] = tw

    return combined_includes
```

6 Store Tweets to CSV

Once we have:

1. The referencing tweet data (the tweet you directly pulled from `data`).
2. The user info for all authors.
3. The original tweets' info for any retweets/quotes.

We can assemble it all into a CSV. Each row contains:

- **Referencing Tweet.** ID, text, metrics, author info
- **Original Tweet.** ID, text, metrics, author info (if the referencing tweet is a retweet or quote)

If no referenced tweet exists, the original tweet columns remain blank.

[37]:
```
# ========== Section 5: Store Tweets to CSV ==========

def store_tweets_as_csv(tweets, user_map, includes_map, csv_filename):
    """
    Write referencing tweets (and if applicable, their original tweets) to CSV.

    Each row in the CSV will contain referencing tweet metadata and, if present,
    the corresponding original tweet's metadata.

    Args:
        tweets (list): List of referencing tweets from the 'data' field in the␣
↪JSON response.
        user_map (dict): Maps author_id to { 'username': str, 'followers_count':
↪ int }.
        includes_map (dict): Maps original_tweet_id to the original tweet␣
↪object.
        csv_filename (str): Filename for the output CSV.
    """

    fieldnames = [
        # Referencing tweet data
        "tweet_id",
        "author_id",
        "created_at",
        "text",
        "lang",
        "retweet_count",
        "reply_count",
        "like_count",
        "quote_count",

        # Original tweet data
        "orig_tweet_id",
        "orig_author_id",
        "orig_created_at",
        "orig_text",
        "orig_lang",
        "orig_retweet_count",
        "orig_reply_count",
        "orig_like_count",
        "orig_quote_count",

        # Referencing tweet's author info
        "author_followers_count",
        "author_username",

        # Original tweet's author info
        "orig_author_followers_count",
        "orig_author_username"
    ]

    with open(csv_filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()

        for tweet in tweets:
            # Referencing tweet
            this_metrics = tweet.get("public_metrics", {})
            author_id = tweet.get("author_id", "")
            author_info = user_map.get(author_id, {})

            # Basic referencing tweet fields
            author_followers_count = author_info.get("followers_count", 0)
            author_username = author_info.get("username", "")

            # Default placeholders for original tweet data
            orig_tweet_id = ""
            orig_author_id = ""
            orig_created_at = ""
            orig_text = ""
            orig_lang = ""
            orig_retweet_count = 0
            orig_reply_count = 0
            orig_like_count = 0
            orig_quote_count = 0
            orig_author_followers_count = 0
            orig_author_username = ""

            # Check if referencing another tweet (retweet or quote)
            referenced = tweet.get("referenced_tweets", [])
            if referenced:
                ref_id = referenced[0].get("id")
                if ref_id and ref_id in includes_map:
                    original_tweet = includes_map[ref_id]
                    orig_tweet_id = original_tweet.get("id", "")
                    orig_author_id = original_tweet.get("author_id", "")
                    orig_created_at = original_tweet.get("created_at", "")
                    orig_text = original_tweet.get("text", "").replace("\n", "␣
↪")

                    orig_lang = original_tweet.get("lang", "")

                    # Original tweet metrics
                    orig_metrics = original_tweet.get("public_metrics", {})
                    orig_retweet_count = orig_metrics.get("retweet_count", 0)
                    orig_reply_count = orig_metrics.get("reply_count", 0)
                    orig_like_count = orig_metrics.get("like_count", 0)
                    orig_quote_count = orig_metrics.get("quote_count", 0)

                    # Original tweet's author info
                    if orig_author_id in user_map:
                        oinfo = user_map[orig_author_id]
                        orig_author_followers_count = oinfo.
↪get("followers_count", 0)
                        orig_author_username = oinfo.get("username", "")

            # Construct CSV row
            row = {
                # Referencing tweet
                "tweet_id": tweet.get("id", ""),
                "author_id": author_id,
                "created_at": tweet.get("created_at", ""),
                "text": tweet.get("text", "").replace("\n", " "),
                "lang": tweet.get("lang", ""),
                "retweet_count": this_metrics.get("retweet_count", 0),
                "reply_count": this_metrics.get("reply_count", 0),
                "like_count": this_metrics.get("like_count", 0),
                "quote_count": this_metrics.get("quote_count", 0),

                # Original tweet
                "orig_tweet_id": orig_tweet_id,
                "orig_author_id": orig_author_id,
                "orig_created_at": orig_created_at,
                "orig_text": orig_text,
                "orig_lang": orig_lang,
                "orig_retweet_count": orig_retweet_count,
                "orig_reply_count": orig_reply_count,
                "orig_like_count": orig_like_count,
                "orig_quote_count": orig_quote_count,

                # Referencing tweet's author
                "author_followers_count": author_followers_count,
                "author_username": author_username,

                # Original tweet's author
                "orig_author_followers_count": orig_author_followers_count,
                "orig_author_username": orig_author_username,
            }

            writer.writerow(row)
```

7 Main Execution Flow (Five 3-Hour Windows)

Below is the `main()` function that orchestrates our data collection. The logic is:

1. **Time Windows.** We split a 15-hour range into five 3-hour segments:

- Window #1: Now minus 3 hours to Now (most recent 3 hours)
- Window #2: Now minus 6 hours to Now minus 3 hours
- Window #3: Now minus 9 hours to Now minus 6 hours
- Window #4: Now minus 12 hours to Now minus 9 hours
- Window #5: Now minus 15 hours to Now minus 12 hours

3. **Collection.** For each window, we collect up to 1,450 tweets. This ensures a broad coverage of tweets over the entire 15-hour period and avoids retrieving all tweets from only the most recent hours (which might be heavily skewed).

4. **Rate Limiting**

- Each request fetches up to 100 tweets.
- We add a 30-second sleep after each successful API call to avoid hitting the Basic plan rate limit (60 requests per 15 minutes).
- If we do encounter a 429 "Too Many Requests" error, we catch it, sleep for 3 minutes, and retry.

4. **Pagination.** We use `next_token` to fetch additional pages until we have either reached the maximum tweets for that window or no more results are available.

5. **Output.** We write each 3-hour window's tweets to a separate CSV file (`tweets_period_1.csv` through `tweets_period_5.csv`).

[38]:
```python
# ========== Section 6: Main Execution Flow (Five 3-Hour Windows) ==========

def main():
    """
    Orchestrate the data collection across five 3-hour windows (15 hours total).
    Each window retrieves up to 1,450 tweets, stored in a dedicated CSV file.
    """
    tweets_per_window = 1450
    # We slightly offset "now" by 15 seconds to reduce any edge-case issues␣
 ↪with
    # the Twitter API's recent search window. This is optional, but can help.
    now_utc = datetime.utcnow() - timedelta(seconds=15)

    for i in range(5):
        # Calculate start/end times for the i-th window
        # i=0 => from (now - 3h) to now
        # i=1 => from (now - 6h) to (now - 3h)
        # etc.
        end_dt = now_utc - timedelta(hours=3 * i)
        start_dt = now_utc - timedelta(hours=3 * (i + 1))

        # Build ISO 8601 strings (Twitter requires this format)
        start_time_str = start_dt.isoformat(timespec='seconds') + 'Z'
        end_time_str = end_dt.isoformat(timespec='seconds') + 'Z'

        print(f"\n[Window {i+1}] Collecting up to {tweets_per_window} tweets "
              f"from {start_time_str} to {end_time_str}...")

        all_json_pages = []
        all_tweets_collected = []
        next_token = None

        # Continue fetching until we have the desired number of tweets or no␣
 ↪more are available
        while len(all_tweets_collected) < tweets_per_window:
            try:
                resp_json = fetch_tweets(
                    query=query,
                    start_time=start_time_str,
                    end_time=end_time_str,
                    next_token=next_token,
                    max_results=100
                )
            except requests.exceptions.HTTPError as e:
                if e.response.status_code == 429:
                    # Rate limit hit: sleep 3 minutes, then retry
                    print("Rate limit hit (429). Sleeping for 3 minutes before␣
↪retry...")
                    time.sleep(180)
                    continue
                else:
                    # Unrelated HTTP error; re-raise
                    raise

            all_json_pages.append(resp_json)
            data = resp_json.get("data", [])
            all_tweets_collected.extend(data)

            meta = resp_json.get("meta", {})
            next_token = meta.get("next_token")

            if not next_token:
                # No more pages available
                print("No more pages available or fewer tweets than requested.")
                break

            # Sleep to respect rate limits (Basic plan: ~1 request every 15␣
↪seconds)
            time.sleep(30)

        # If we fetched more than needed, trim the list
        all_tweets_collected = all_tweets_collected[:tweets_per_window]

        # Merge user info and original tweets from expansions
        user_map = collect_all_user_info(all_json_pages)
        includes_map = collect_all_includes_tweets(all_json_pages)

        # Store results in a CSV
        csv_filename = f"tweets_period_{i+1}.csv"
        store_tweets_as_csv(all_tweets_collected, user_map, includes_map,␣
↪csv_filename)

        print(f" -> Stored {len(all_tweets_collected)} tweets in {csv_filename}.
↪")

    print("\nAll 5 windows completed successfully with Basic plan rate-limit␣
↪handling!")
```

8 Running the Script

By running the final code:

```python
[39]: if __name__ == "__main__":
          main()
```

```
[Window 1] Collecting up to 1450 tweets from 2025-01-16T07:47:49Z to
2025-01-16T10:47:49Z…
  -> Stored 1450 tweets in tweets_period_1.csv.

[Window 2] Collecting up to 1450 tweets from 2025-01-16T04:47:49Z to
2025-01-16T07:47:49Z…
  -> Stored 1450 tweets in tweets_period_2.csv.

[Window 3] Collecting up to 1450 tweets from 2025-01-16T01:47:49Z to
2025-01-16T04:47:49Z…
  -> Stored 1450 tweets in tweets_period_3.csv.

[Window 4] Collecting up to 1450 tweets from 2025-01-15T22:47:49Z to
2025-01-16T01:47:49Z…
  -> Stored 1450 tweets in tweets_period_4.csv.

[Window 5] Collecting up to 1450 tweets from 2025-01-15T19:47:49Z to
2025-01-15T22:47:49Z…
  -> Stored 1450 tweets in tweets_period_5.csv.

All 5 windows completed successfully (with Basic plan rate-limit logic)!
```

You will generate five CSV files named:

- `tweets_period_1.csv`
- `tweets_period_2.csv`
- `tweets_period_3.csv`
- `tweets_period_4.csv`
- `tweets_period_5.csv`

Each file contains up to 1,450 tweets collected during a distinct 3-hour time window, for a total of up to 7,250 tweets spanning the last 15 hours.

8.1 Important Notes

- **Rate Limits.** The Basic Twitter API plan currently allows 60 requests every 15 minutes for the `search/recent` endpoint. In our workflow, we add a delay after each call. If we exceed the limit, the code will catch a 429 error, pause for 3 minutes, and retry.
- **No Duplication.** Segmenting the 15-hour window into 3-hour slices helps ensure fewer duplicates and more representative coverage from each time slice. Otherwise, if we tried to pull 7,250 tweets in one go, Twitter's pagination might return mostly the latest tweets (which might be found in the last hour).

[ ]:
```
```

[ ]:
```
```