Gaza Ceasefire X (Twitter) Data Preprocessing

Ilyas Ibrahim Mohamed

January 18, 2025

1 Introduction

This notebook focuses on preparing the tweet data collected from the Gaza ceasefire discussions for deeper analysis. By the end, we will have a single "master" dataset that is de-duplicated, enriched with new columns, cleaned of noise, and ready for use in visualization, modeling, or storytelling.

2 Environment Setup and Imports

In this section, we configure our Python environment and load essential dependencies:

- `pandas`. For DataFrame operations and CSV I/O.
- `numpy`. For array and numerical processing.
- `re`. For regex-based text cleaning.
- `nltk`. For advanced NLP tasks (tokenization, stopword removal, etc.).

Make sure you have the CSV files generated by the data collection process (e.g., `tweets_period_1.csv` through `tweets_period_5.csv`) in your working directory.

[15]:
```
# ========== Section 1: Environment Setup and Imports ==========

import os
import re
import numpy as np
import pandas as pd

# For optional advanced NLP (Step 7), we use NLTK:
import nltk
nltk.download('stopwords')  # Ensure NLTK stopwords are available
from nltk.corpus import stopwords

# Define a set of English stopwords (for advanced text cleaning)
stop_words = set(stopwords.words('english'))

# If needed, load other environment variables or credentials here
# from dotenv import load_dotenv
# load_dotenv()
# ...
```

```
[nltk_data] Downloading package stopwords to /Users/ilyas/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

3 Consolidate All CSV Files

We previously collected tweets in five CSV files—one for each 3-hour window over a total of 15 hours. Now, we load them all into separate DataFrames and concatenate them into a single DataFrame for easier processing.

[16]:
```
# ========== Section 2: Consolidate All CSV Files ==========

df_list = []
for i in range(1, 6):
    filename = f"tweets_period_{i}.csv"
    print(f"Reading {filename}...")
    temp_df = pd.read_csv(filename)  # Load each CSV into a DataFrame
    df_list.append(temp_df)

# Combine all DataFrames into one
combined_df = pd.concat(df_list, ignore_index=True)

print("\nCombined DataFrame shape:", combined_df.shape)
display(combined_df.head(5))  # Preview a small sample for data integrity
```

```
Reading tweets_period_1.csv…
Reading tweets_period_2.csv…
Reading tweets_period_3.csv…
Reading tweets_period_4.csv…
Reading tweets_period_5.csv…

Combined DataFrame shape: (7250, 22)
```

	tweet_id	author_id	created_at	\
0	1879842988714791211	751910658267029505	2025-01-16T10:47:48.000Z	
1	1879842986743455869	1514309714980401153	2025-01-16T10:47:48.000Z	
2	1879842986487619937	53293606	2025-01-16T10:47:48.000Z	
3	1879842985862713416	869624059	2025-01-16T10:47:48.000Z	
4	1879842985824932102	1088469191571722240	2025-01-16T10:47:48.000Z	

	text	lang	retweet_count	\
0	RT @missfalsteenia: A year later & he stil…	en	28155	
1	RT @DanScavino: Congratulations to President T…	en	6515	
2	RT @swilkinsonbc: Bereaved Palestinian parents…	en	31	
3	RT @Hind_Gaza: At least 71 Palestinians have b…	en	1463	
4	RT @zarahsultana: No mention of UK arms sales …	en	2886	

	reply_count	like_count	quote_count	orig_tweet_id	…	\
0	0	0	0	1.879223e+18	…	

```
1            0           0            0   1.879630e+18  …
2            0           0            0   1.879840e+18  …
3            0           0            0   1.879825e+18  …
4            0           0            0   1.879658e+18  …

                                          orig_text orig_lang  \
0  A year later &amp;amp; he still stands every singl…        en
1  Congratulations to President Trump and his Spe…        en
2  Bereaved Palestinian parents say goodbye to th…        en
3  At least 71 Palestinians have been killed and …        en
4  No mention of UK arms sales that have enabled …        en

  orig_retweet_count orig_reply_count  orig_like_count  orig_quote_count  \
0              28181              133           134723               103
1               6515              704            25518               215
2                 42                4               64                 3
3               1588               35             2066                62
4               2899              620            16443                39

   author_followers_count  author_username  orig_author_followers_count  \
0                   41177    nuelleduterte                        93133
1                    1183    LionHearted46                      2098520
2                    3679       steveirons                       324704
3                     452       ludaycrous                       259532
4                    4287     KloppEnjoyer                       365652

  orig_author_username
0       missfalsteenia
1           DanScavino
2         swilkinsonbc
3            Hind_Gaza
4         zarahsultana

[5 rows x 22 columns]
```

4 Removing Duplicates

It is possible for tweets to appear in multiple time windows or overlap in our search results. We drop duplicates based on the `tweet_id` column, keeping only the first occurrence of each unique tweet.

[17]:
```python
# ========== Section 3: De-Duplicate Rows by tweet_id ==========

before_dedup = combined_df.shape[0]
combined_df.drop_duplicates(subset=["tweet_id"], inplace=True)
after_dedup = combined_df.shape[0]
```

```
print(f"Rows before deduplication: {before_dedup}")
print(f"Rows after deduplication: {after_dedup}")
print(f"Duplicates removed: {before_dedup - after_dedup}")
```

```
Rows before deduplication: 7250
Rows after deduplication:  7250
Duplicates removed:        0
```

5 Handling Missing Values

Some columns, particularly those related to original (referenced) tweets—such as `orig_tweet_id`, `orig_author_id`, `orig_created_at`, `orig_text`—will be missing for truly original tweets.

In many cases, leaving these columns as NaN is helpful because it preserves the semantic meaning that "this tweet does not reference any original post." If you prefer to fill missing values with placeholders (e.g., `""` or "`N/A`"), you can modify the code accordingly.

[18]:
```
# ========== Section 4: Handle Missing Columns / Null Values ==========

print("Missing value counts for key columns:")
missing_counts = combined_df.isnull().sum().sort_values(ascending=False)
display(missing_counts.head(10))

# Example code to fill referencing columns with empty strings (commented out):
# placeholder_cols = [
#     "orig_created_at",
#     "orig_tweet_id",
#     "orig_lang",
#     "orig_text",
#     "orig_author_id",
#     "orig_author_username"
# ]
# for col in placeholder_cols:
#     if col in combined_df.columns:
#         combined_df[col].fillna("", inplace=True)
#
# After filling, re-check missing counts:
# missing_counts = combined_df.isnull().sum().sort_values(ascending=False)
# display(missing_counts.head(10))
```

```
Missing value counts for key columns:

orig_created_at         240
orig_tweet_id           240
orig_lang               240
orig_text               240
orig_author_id          240
orig_author_username    240
reply_count               0
```

```
like_count                0
quote_count               0
retweet_count             0
dtype: int64
```

6 Basic Text Cleaning

We perform initial text cleaning on the `text` and `orig_text` columns, removing URLs, converting to lowercase, and stripping leading/trailing whitespace. You can also remove hashtags, mentions, or punctuation if desired.

[19]:
```python
# ========== Section 5: Basic Text Cleaning ==========

def basic_text_cleaning(text):
    """
    Remove URLs, convert to lowercase, and strip extra whitespace.
    """
    if not isinstance(text, str):
        return text

    # Remove URLs
    text = re.sub(r"http\S+", "", text)
    # Convert to lowercase
    text = text.lower()
    # Strip leading/trailing whitespace
    text = text.strip()

    return text

# Apply basic cleaning to referencing and original text columns
combined_df["text"] = combined_df["text"].apply(basic_text_cleaning)
combined_df["orig_text"] = combined_df["orig_text"].apply(basic_text_cleaning)

print("Basic text cleaning complete. Sample of 'text' column:")
display(combined_df[["tweet_id", "text", "orig_text"]].head(5))
```

```
Basic text cleaning complete. Sample of 'text' column:
```

	tweet_id	text	orig_text
0	1879842988714791211	rt @missfalsteenia: a year later & he stil…	a year later & he still stands every singl…
1	1879842986743455869	rt @danscavino: congratulations to president t…	congratulations to president trump and his spe…
2	1879842986487619937	rt @swilkinsonbc: bereaved palestinian parents…	
3	1879842985862713416	rt @hind_gaza: at least 71 palestinians have b…	
4	1879842985824932102	rt @zarahsultana: no mention of uk arms sales …	

```
2  bereaved palestinian parents say goodbye to th…
3  at least 71 palestinians have been killed and …
4  no mention of uk arms sales that have enabled …
```

7 Labeling Tweets as Original or Referencing

Next, we add a tweet_type column to distinguish:

- `original` tweets that have no referenced tweet (i.e., `orig_tweet_id` is NaN or empty).
- `retweet_or_quote` tweets that reference another tweet's content.

This classification is useful for later steps (e.g., deciding whether to analyze the referencing tweet's text or the original tweet's text).

[20]:
```
# ========== Section 6: Label Tweets as Original or Referencing ==========

def label_tweet_type(row):
    """
    Return 'original' if 'orig_tweet_id' is missing/empty,
    else 'retweet_or_quote'.
    """
    if "orig_tweet_id" not in row or pd.isna(row["orig_tweet_id"]) or␣
 ↪row["orig_tweet_id"] == "":
        return "original"
    else:
        return "retweet_or_quote"

if "orig_tweet_id" in combined_df.columns:
    combined_df["tweet_type"] = combined_df.apply(label_tweet_type, axis=1)
else:
    combined_df["tweet_type"] = "unknown"

print("Tweet type distribution:")
display(combined_df["tweet_type"].value_counts())
```

```
Tweet type distribution:
```

```
tweet_type
retweet_or_quote    7010
original             240
Name: count, dtype: int64
```

8 Advanced NLP Preprocessing

We introduce a new column called analysis_text, which merges:

- `orig_text` if the tweet is referencing another tweet.
- `text` if the tweet is truly original.

After that, we apply advanced NLP techniques—tokenization, stopword removal, and optional stemming or lemmatization. The cleaned result is stored in `analysis_text_nlp`.

[21]:
```
# ========== Section 7: Advanced NLP Preprocessing ==========

# 7.1) Merge 'text' and 'orig_text' into 'analysis_text'
def get_analysis_text(row):
    """
    If the tweet references an original tweet, use 'orig_text'.
    Otherwise, use the tweet's own 'text'.
    """
    if row["tweet_type"] == "retweet_or_quote":
        return row["orig_text"]  # May be NaN if incomplete, which is acceptable
    else:
        return row["text"]

combined_df["analysis_text"] = combined_df.apply(get_analysis_text, axis=1)

# 7.2) Define an advanced NLP preprocessing function
def advanced_nlp_preprocess(text):
    """
    Tokenizes, lowercases, removes stopwords/punctuation, and optionally
    stems or lemmatizes.
    """
    if pd.isna(text):
        # Keep NaN if there's no text at all
        return np.nan

    # Tokenize by whitespace
    tokens = text.split()
    cleaned_tokens = []
    for tok in tokens:
        # Remove non-alphanumeric characters
        tok = re.sub(r'[^a-zA-Z0-9]', '', tok)
        # Convert to lowercase
        tok = tok.lower().strip()
        # Filter out empty tokens and English stopwords
        if tok and tok not in stop_words:
            cleaned_tokens.append(tok)

    # Optional Stemming or Lemmatization (commented out)
    # from nltk.stem import PorterStemmer
    # stemmer = PorterStemmer()
    # cleaned_tokens = [stemmer.stem(t) for t in cleaned_tokens]

    # Return tokens as a single string
    return " ".join(cleaned_tokens)
```

```
# 7.3) Apply advanced NLP to 'analysis_text'
combined_df["analysis_text_nlp"] = combined_df["analysis_text"].
 ↪apply(advanced_nlp_preprocess)

print("Advanced NLP preprocessing complete. Sample comparison:")
display(combined_df[["tweet_id", "tweet_type", "analysis_text",␣
 ↪"analysis_text_nlp"]].head(10))
```

```
Advanced NLP preprocessing complete. Here is a sample of 'analysis_text' vs.
'analysis_text_nlp':

             tweet_id        tweet_type  \
0  1879842988714791211  retweet_or_quote
1  1879842986743455869  retweet_or_quote
2  1879842986487619937  retweet_or_quote
3  1879842985862713416  retweet_or_quote
4  1879842985824932102  retweet_or_quote
5  1879842984772182413  retweet_or_quote
6  1879842984407302621  retweet_or_quote
7  1879842984356966669  retweet_or_quote
8  1879842984117825802  retweet_or_quote
9  1879842983958487511  retweet_or_quote

                                      analysis_text  \
0  a year later &amp;amp; he still stands every singl…
1  congratulations to president trump and his spe…
2  bereaved palestinian parents say goodbye to th…
3  at least 71 palestinians have been killed and …
4  no mention of uk arms sales that have enabled …
5  maybe this will draw your attention to my niec…
6  israel is ramping up their killing before the …
7  joe biden ended his presidency with:  2.9% inf…
8    stop  everyone has stopped talking about us…
9          long live palestine     long live gaza

                                   analysis_text_nlp
0  year later amp still stands every single day c…
1  congratulations president trump special envoy …
2  bereaved palestinian parents say goodbye belov…
3  least 71 palestinians killed 200 injured since…
4        mention uk arms sales enabled genocide gaza
5  maybe draw attention niece really finds alone …
6  israel ramping killing ceasefire takes effect …
7  joe biden ended presidency 29 inflation 41 une…
8  stop everyone stopped talking us everything co…
9                 long live palestine long live gaza
```

9 Creating Derived Features

We create additional columns to support deeper analysis:

1. **Time-based Features**

- Convert `created_at` to a `datetime` object and extract `hour_utc`.
- Convert `orig_created_at` similarly and extract `orig_hour_utc`.

2. **Unified "Analysis" Columns**

- For referencing tweets, we overwrite the tweet's own metrics (`retweet_count`, `like_count`, etc.) with the original tweet's metrics.
- This way, each row captures the original perspective of the tweet content—useful if you want to analyze the "true" engagement of the text.

3. **Derived Metrics**

- `analysis_engagement_score`. A sum of retweets, replies, likes, and quotes.
- `analysis_impact_score`. A product of the author's followers and the total engagement—an approximate measure of reach.
- `analysis_interaction_rate`. Ratio of engagement to the author's followers (helps normalize for user size).

[25]:
```
# ========== Section 8: Create Derived Features ==========

# 8.1) Convert date columns and extract hour of day
combined_df["created_at"] = pd.to_datetime(combined_df["created_at"],␣
 ↪errors="coerce")
combined_df["hour_utc"] = combined_df["created_at"].dt.hour.fillna(-1).
 ↪astype(int)

combined_df["orig_created_at"] = pd.to_datetime(combined_df["orig_created_at"],␣
 ↪errors="coerce")
combined_df["orig_hour_utc"] = combined_df["orig_created_at"].dt.hour.
 ↪fillna(-1).astype(int)

# 8.2) Default "analysis_" columns to the tweet's own metrics & author data
combined_df["analysis_author_id"] = combined_df["author_id"]
combined_df["analysis_created_at"] = combined_df["created_at"]
combined_df["analysis_retweet_count"] = combined_df["retweet_count"]
combined_df["analysis_reply_count"] = combined_df["reply_count"]
combined_df["analysis_like_count"] = combined_df["like_count"]
combined_df["analysis_quote_count"] = combined_df["quote_count"]
combined_df["analysis_author_followers_count"] =␣
 ↪combined_df["author_followers_count"]

# 8.3) Overwrite columns with original tweet data for referencing tweets
mask_ref = (combined_df["tweet_type"] == "retweet_or_quote")
```

```
combined_df.loc[mask_ref, "analysis_author_id"] = combined_df.loc[mask_ref,␣
 ↪"orig_author_id"]
combined_df.loc[mask_ref, "analysis_created_at"] = combined_df.loc[mask_ref,␣
 ↪"orig_created_at"]
combined_df.loc[mask_ref, "analysis_retweet_count"] = combined_df.loc[mask_ref,␣
 ↪"orig_retweet_count"]
combined_df.loc[mask_ref, "analysis_reply_count"] = combined_df.loc[mask_ref,␣
 ↪"orig_reply_count"]
combined_df.loc[mask_ref, "analysis_like_count"] = combined_df.loc[mask_ref,␣
 ↪"orig_like_count"]
combined_df.loc[mask_ref, "analysis_quote_count"] = combined_df.loc[mask_ref,␣
 ↪"orig_quote_count"]
combined_df.loc[mask_ref, "analysis_author_followers_count"] = combined_df.
 ↪loc[mask_ref, "orig_author_followers_count"]

# 8.4) Compute final aggregated metrics
combined_df["analysis_engagement_score"] = (
    combined_df["analysis_retweet_count"] +
    combined_df["analysis_reply_count"] +
    combined_df["analysis_like_count"] +
    combined_df["analysis_quote_count"]
)

combined_df["analysis_impact_score"] = (
    combined_df["analysis_author_followers_count"] *␣
 ↪combined_df["analysis_engagement_score"]
)

# 8.5) Compute interaction_rate
# This normalizes engagement by the author's follower count
combined_df["analysis_interaction_rate"] = (
    combined_df["analysis_engagement_score"] /␣
 ↪(combined_df["analysis_author_followers_count"] + 1)
)

print("Final 'analysis' columns have been created or overwritten for␣
 ↪referencing tweets.")
display(combined_df[[
    "tweet_id",
    "tweet_type",
    "analysis_author_id",
    "hour_utc",
    "orig_hour_utc",
    "analysis_created_at",
    "analysis_retweet_count",
    "analysis_like_count",
```

```
    "analysis_author_followers_count",
    "analysis_engagement_score",
    "analysis_impact_score",
    "analysis_interaction_rate"
]].head(5))
```

```
Final 'analysis' columns have been created/overwritten for referencing tweets.
```

	tweet_id	tweet_type	analysis_author_id	hour_utc	\
0	1879842988714791211	retweet_or_quote	950539834464010240	10	
1	1879842986743455869	retweet_or_quote	620571475	10	
2	1879842986487619937	retweet_or_quote	4249826728	10	
3	1879842985862713416	retweet_or_quote	1514633984185184256	10	
4	1879842985824932102	retweet_or_quote	3056307455	10	

	orig_hour_utc	analysis_created_at	analysis_retweet_count	\
0	17	2025-01-14 17:45:59+00:00	28181	
1	20	2025-01-15 20:43:15+00:00	6515	
2	10	2025-01-16 10:35:02+00:00	42	
3	9	2025-01-16 09:35:37+00:00	1588	
4	22	2025-01-15 22:31:09+00:00	2899	

	analysis_like_count	analysis_author_followers_count	\
0	134723	93133	
1	25518	2098520	
2	64	324704	
3	2066	259532	
4	16443	365652	

	analysis_engagement_score	analysis_impact_score	analysis_interaction_rate
0	163140	15193717620	1.751670
1	32952	69150431040	0.015702
2	113	36691552	0.000348
3	3751	973504532	0.014453
4	20001	7313405652	0.054699

10 Exporting "Master" Dataset to CSV File

The final step is to save our cleaned and feature-rich DataFrame to a master CSV file. This dataset consolidates:

- De-duplicated tweets.
- Basic text cleaning + advanced NLP columns (`analysis_text_nlp`).
- A `tweet_type` label indicating original vs. referencing.
- Derived engagement and impact features.

[27]:
```
# ========== Section 9: Save Final "Master" Dataset ==========
```

```
output_filename = "tweets_master_cleaned.csv"
combined_df.to_csv(output_filename, index=False, encoding="utf-8")

print(f"Master cleaned dataset saved to '{output_filename}' with shape␣
 ↪{combined_df.shape}.")
```

```
Master cleaned dataset saved to 'tweets_master_cleaned.csv' with shape (7250,
37).
```

With this master dataset in hand, you can proceed to any of the following advanced analyses:

- **Visualization.** Plot time-series of tweets, engagement, or user activity.
- **Sentiment/Topic Modeling.** Apply NLP techniques (e.g., sentiment analysis, LDA topic modeling) to `analysis_text_nlp`.
- **Influencer Analysis.** Identify high-impact users based on `analysis_impact_score` or `analysis_interaction_rate`.
- **Storytelling.** Combine data with external information sources, build dashboards, or perform deeper statistical modeling.