

# Enhancing User Experience and Undertaking Sentiment Analysis with Machine Learning in Social Media Twitter (X)

Submitted by

**GK Sridharan**

(Regd. No. 423206415017)

Under the guidance of

**Prof. K. Venkata Rao**

Head of the Department, CSSE

Department of Computer Science and Systems Engineering

## Abstract

This project investigates the enhancement of user experience and engagement on the social media platform X (formerly Twitter) and Sentiment Analysis through the strategic application of advanced Machine Learning (ML) techniques. Despite its prominence as a communication tool for government agencies, policymakers, and influential figures—such as Heads of State—for disseminating critical announcements and shaping public perception during emergencies, **Twitter (X) struggles with limited user engagement and lower adoption rates among the general public** in many countries, including India, compared to platforms like *Facebook* and *Instagram*. This gap is primarily attributed to deficiencies in user experience, which this study seeks to address.

This study also undertakes **Twitter Sentiment Analysis** to demonstrate the application of machine learning in real-world social media data. Sentiment analysis helps classify tweets as *positive*, *negative*, or *neutral*, offering valuable insights into public mood, brand perception, and reactions to events. Using Python libraries such as Pandas, Scikit-learn, and TF-IDF vectorization, a supervised ML model was implemented and tested on the Sentiment140 dataset. The process involved data cleaning, feature extraction, and training classification models, which achieved reliable accuracy in distinguishing user opinions. This implementation showcases how machine learning can convert massive, unstructured tweet streams into actionable knowledge for businesses, researchers, and policymakers.

The research examines the current application of ML algorithms on X, focusing on features such as personalised content recommendations for a twitter user and undertaking sentiment analysis of a post with ML Model. Also, It identifies key challenges contributing to X's suboptimal engagement and analyse the following challenges - Limited cross-platform integration (e.g., with *WhatsApp*) and possible solution and Text Length Restriction in a Tweet. Pros & Cons.

While policy-related concerns (e.g., phone-number-based authentication for new accounts) fall outside the study's scope, the work emphasizes feasible, ML-driven solutions.

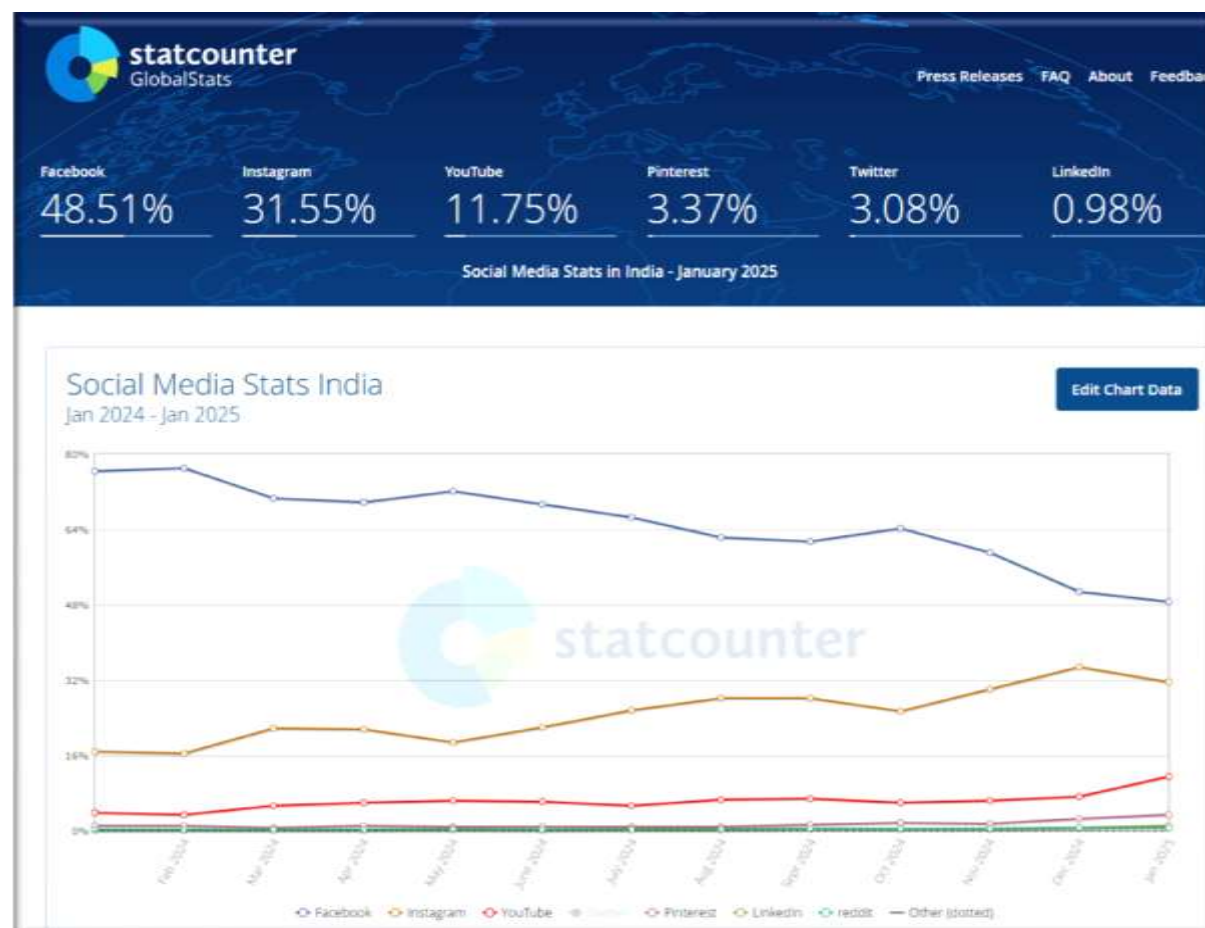
The findings and proposed models aim to bridge the gap between high-profile users (leaders, researchers, military organizations) and the general public, fostering a more interactive, inclusive, and user-centric ecosystem. By aligning Twitter's design with evolving user expectations, this research positions X as a more dynamic and accessible social media platform.

## Literature Review

Machine learning (ML) plays a crucial role in analysing various sentiments of twitter reactions and also improving various features of *Twitter*, such as content recommendation, sentiment analysis, and trending topic identification. Research shows that Twitter uses advanced ML models to enhance user experience by making content more relevant and engaging. However, several challenges impact its effectiveness, including limited engagement features and evolving user behaviour. One of the key applications of ML in Twitter is personalized content recommendation. The platform uses *deep learning models*, such as *Neural Networks* and *MaskNet*, to predict which tweet would be interesting to a particular user. Additionally, graph-based models like *TwHIN* (Twitter's Heterogeneous Information Network) help identify relevant posts by analysing user interactions. While these techniques improve content discovery, some users feel that Twitter's algorithm-driven feed lacks personalization compared to other social media platforms. (1. Cornell University research paper - [https://arxiv.org/abs/2202.05387?utm\\_source=chatgpt.com](https://arxiv.org/abs/2202.05387?utm_source=chatgpt.com))

Another significant area of ML usage is sentiment analysis, which helps to classify tweets as positive, negative, or neutral. Traditional models like Naïve Bayes and Support Vector Machines (SVM) are commonly used for basic classification, while more advanced models like LSTM (Long Short-Term Memory) and BERT (Bidirectional Encoder Representations from Transformers) provide deeper contextual understanding. These ML techniques allow businesses, researchers, and policymakers to analyse public opinion in real time. However, sentiment analysis still faces challenges in accurately detecting sarcasm, slang, and mixed sentiments, reducing its reliability. (*Journal of Big Data – Springer Open*- <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00781-w>)

Despite these advancements, *Twitter* faces several engagement challenges that limit its popularity among the general public. To provide a more accurate representation of social media usage in India and globally, here is a table based on data from reputable sources: **Sources:** India-specific data: [Statcounter Global Stats](#) / Global data: Statcounter Global Stats)



**Figure 1- Population in Social Media (India : 2025)**

Unlike platforms like *Instagram* and *WhatsApp*, *Twitter* offers limited reaction options, allowing only "likes" without additional responses like emojis or reactions. Users also cannot see who liked a post, reducing transparency and social interaction. Another issue is the lack of a temporary post feature, such as *WhatsApp Status* or the discontinued *Twitter Fleets*, which many users prefer for sharing short-lived updates. (Blogs – Good Bye – Twitter Fleets - [https://blog.x.com/en\\_us/topics/product/2021/goodbye-fleets?utm\\_source=chatgpt.com](https://blog.x.com/en_us/topics/product/2021/goodbye-fleets?utm_source=chatgpt.com) )

Moreover, privacy and real-time data processing remain major challenges. While *Twitter* has started integrating phone number verification for security, it does not link accounts to phone numbers the way *WhatsApp* does, which could impact user acquisition and engagement. Additionally, processing vast amounts of data in real-time requires constant model updates, which can be computationally expensive and prone to errors. (Electronics Frontier Foundation - [https://www.eff.org/deeplinks/2020/02/how-twitters-default-settings-can-leak-your-phone-number?utm\\_source=chatgpt.com](https://www.eff.org/deeplinks/2020/02/how-twitters-default-settings-can-leak-your-phone-number?utm_source=chatgpt.com) )

### GeeksforGeeks: Sentiment Analysis – Step by Step Implementation

The article “*Sentiment Analysis – Step by Step Implementation*” by GeeksforGeeks provides a practical, hands-on approach to conducting sentiment analysis on Twitter data using Python. It systematically explains preprocessing steps such as tokenization, stop-word removal, and text vectorization techniques (Bag of Words, TF-IDF), followed by supervised machine learning model training using libraries like Scikit-learn. The implementation emphasizes clarity in handling real-world social media datasets, making it particularly useful for beginners and practitioners aiming to replicate sentiment analysis pipelines. This work is relevant to the project as it offers both the technical methodology and coding framework required to implement end-to-end sentiment classification, ensuring reproducibility and practical application on Twitter datasets. [Twitter Sentiment Analysis using Python - GeeksforGeeks](#)

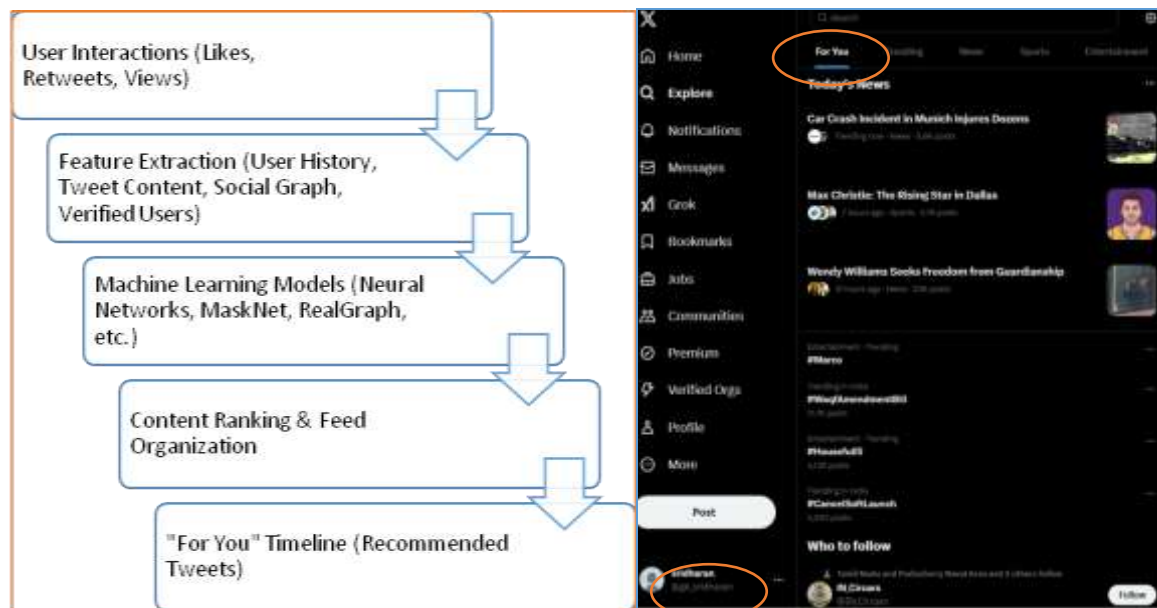
### Sentiment140 Dataset (1.6 Million Tweets)

The Sentiment140 dataset is a widely used benchmark corpus created by extracting 1.6 million tweets via the Twitter API and annotating them with sentiment labels, where *0* represents *negative* and *4* represents *positive*. Designed for large-scale sentiment analysis research, it provides a balanced and diverse set of user-generated content, making it suitable for training and evaluating machine learning models. Its large volume of annotated data enables robust model performance and helps in addressing challenges such as sarcasm, informal language, and noisy text common in Twitter posts. For this project, Sentiment140 serves as the foundational dataset for building and testing supervised ML models, ensuring that the sentiment analysis implementation is based on a standardized and validated resource widely recognized in the academic and research community. - [Sentiment140 dataset with 1.6 million tweets](#)

In summary, Twitter effectively uses ML to enhance user experience through personalized recommendations, sentiment analysis, and trend predictions. However, challenges like limited user engagement features, privacy concerns, and real-time data processing difficulties hinder its widespread adoption. Future research should focus on improving engagement strategies, refining content recommendations, and balancing privacy with personalization to make the platform more user-friendly.

## Chapter 1 – Content Recommendation For a Twitter User

Twitter employs a complex algorithm for ranking content on the “*For You*” timeline, using machine learning techniques to predict content relevance. The algorithm selects content from both followed and non-followed accounts, ranking them based on predicted engagement.



**Figure 2- Left - Usage of ML in Feed Ranking R - Screen Shot of For You**

Key machine learning techniques used in feed ranking include neural networks and graph-based models. Twitter utilizes a deep neural network with approximately 48 million parameters to score tweets. This model considers factors such as user history, tweet content, and the social graph to predict engagement levels. A specific variant known as MaskNet plays a crucial role in optimizing feed ranking.

Twitter uses different machine learning algorithms to manage and organize content, including tweets in notifications. These algorithms help decide which tweets appear in a user's feed or notifications based on engagement and relevance.

## Machine Learning Algorithms Used

Logistic regression helps rank tweets by predicting user engagement. Neural networks, with around 40 million parameters, analyze tweet relevance and engagement. Embedding spaces create numerical representations of users' interests and tweet content for better content matching. RealGraph, a graph-based model, maps relationships between users, tweets, and hashtags to improve recommendations.

## Implementation – Content Recommendation

To explain how Twitter (X) might implement a Content Recommendation (Feed Ranking) algorithm in Python, let's simplify the concept. The "For You" timeline uses machine learning to show posts that are likely to interest a user, whether from accounts they follow or others. The algorithm predicts how engaging a post will be based on factors like user interactions (likes, retweets, replies), post content, and user preferences. Below is a simplified explanation and a basic Python example of how such an algorithm could work.

- **Data Collection:** The algorithm gathers data about posts (e.g., text, likes, retweets) and user behavior (e.g., what they liked or retweeted).
- **Feature Extraction:** It identifies key features, like:
  - Post Features: Number of likes, retweets, or if it contains trending hashtags.
  - User Features: What topics or accounts the user interacts with.
  - Context Features: Time of day, user's location, or recent trends.
- **Scoring Posts:** A machine learning model (e.g., a neural network or decision tree) assigns a "relevance score" to each post based on these features.
- **Ranking:** Posts are sorted by their relevance scores, and the top ones appear on the "For You" timeline.

- **Personalization:** The model learns from user feedback (e.g., likes or skips) to improve future recommendations.

**Simplified Python** Example given below is a basic Python code snippet that demonstrates how a content recommendation system might work using a simple machine learning approach. This example uses a logistic regression model to predict whether a post is relevant to a user, but in reality, X uses more complex models like neural networks.

python

```
import pandas as pd
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import StandardScaler
```

```
# Sample data: Features of posts and whether the user engaged with them
```

```
data = { 'post_likes': [100, 50, 200, 10, 300], # Number of likes on the post
```

```
        'post_retweets': [20, 10, 50, 5, 100], # Number of retweets
```

```
        'has_trending_hashtag': [1, 0, 1, 0, 1], # 1 if post has trending hashtag, 0 if not
```

```
        'user_follows_author': [1, 0, 1, 0, 0], # 1 if user follows the post's author, 0 if not
```

```
        'engaged': [1, 0, 1, 0, 1] # 1 if user engaged (liked/retweeted), 0 if not
```

```
}
```

```
# Create a DataFrame
```

```
df = pd.DataFrame(data)
```

```
# Features (inputs) and target (output)
```

```
X = df[['post_likes', 'post_retweets', 'has_trending_hashtag', 'user_follows_author']]
```

```
y = df['engaged']
```

```
# Scale the features (important for ML models)
```

```
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

```
# Train a simple logistic regression model
```

```
model = LogisticRegression()
```

```
model.fit(X_scaled, y)
```

*# Sample new posts to rank*

```
new_posts = pd.DataFrame({  
  
    'post_likes': [150, 30, 500],  
  
    'post_retweets': [25, 5, 200],  
  
    'has_trending_hashtag': [1, 0, 1],  
  
    'user_follows_author': [0, 1, 1]  
  
})
```

*# Scale the new posts*

```
new_posts_scaled = scaler.transform(new_posts)
```

*# Predict relevance scores (probability of engagement)*

```
relevance_scores = model.predict_proba(new_posts_scaled)[:, 1] # Get probability of 'engaged' = 1
```

*# Rank posts by relevance score*

```
new_posts['relevance_score'] = relevance_scores  
  
ranked_posts = new_posts.sort_values(by='relevance_score', ascending=False)
```

*# Display ranked posts*

```
print("Ranked Posts for 'For You' Timeline:")  
  
print(ranked_posts[['post_likes', 'post_retweets', 'has_trending_hashtag', 'user_follows_author', 'relevance_score']])
```



## Simple Content recommendation Example in Python

```
# Simple Content Recommendation Example in Python
# Copy-paste this into PyCharm and run it

# Import libraries we need
import pandas as pd # for working with tables (DataFrame)
from sklearn.linear_model import LogisticRegression # our ML model
from sklearn.preprocessing import StandardScaler # for scaling numbers

# Step 1: Create sample data (like past posts and user engagement history)
data = {
    'post_likes': [100, 50, 200, 10, 300], # number of likes each post got
    'post_retweets': [20, 10, 50, 5, 100], # number of retweets
    'has_trending_hashtag': [1, 0, 1, 0, 1], # 1 = post has trending hashtag, 0 = no
    'user_follows_author': [1, 0, 1, 0, 0], # 1 = user follows the author, 0 = no
    'engaged': [1, 0, 1, 0, 1] # 1 = user engaged (liked/retweeted), 0 = did not
}

# Step 2: Convert dictionary into a DataFrame (table format)
df = pd.DataFrame(data)
print("Training Data:")
print(df, "\n")

# Step 3: Split into inputs (X) and output (y)
X = df[['post_likes', 'post_retweets', 'has_trending_hashtag', 'user_follows_author']] # inputs
y = df['engaged'] # output (target we want to predict)

# Step 4: Scale the features (important so that likes/retweets don't dominate over yes/no values)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # fit (learn scaling) and transform (apply scaling)

# Step 5: Train a simple logistic regression model
model = LogisticRegression()
model.fit(X_scaled, y) # train the model on old data

# Step 6: Make a new post to test the model
new_posts = pd.DataFrame({
    'post_likes': [150], # new post has 150 likes
    'post_retweets': [25], # new post has 25 retweets
    'has_trending_hashtag': [1], # yes, it has a trending hashtag
    'user_follows_author': [0] # no, user doesn't follow the author
})

print("New Post Details:")
print(new_posts, "\n")

# Step 7: Scale the new post (so it matches the training data scale)
new_posts_scaled = scaler.transform(new_posts)

# Step 8: Predict probability of engagement
```

```

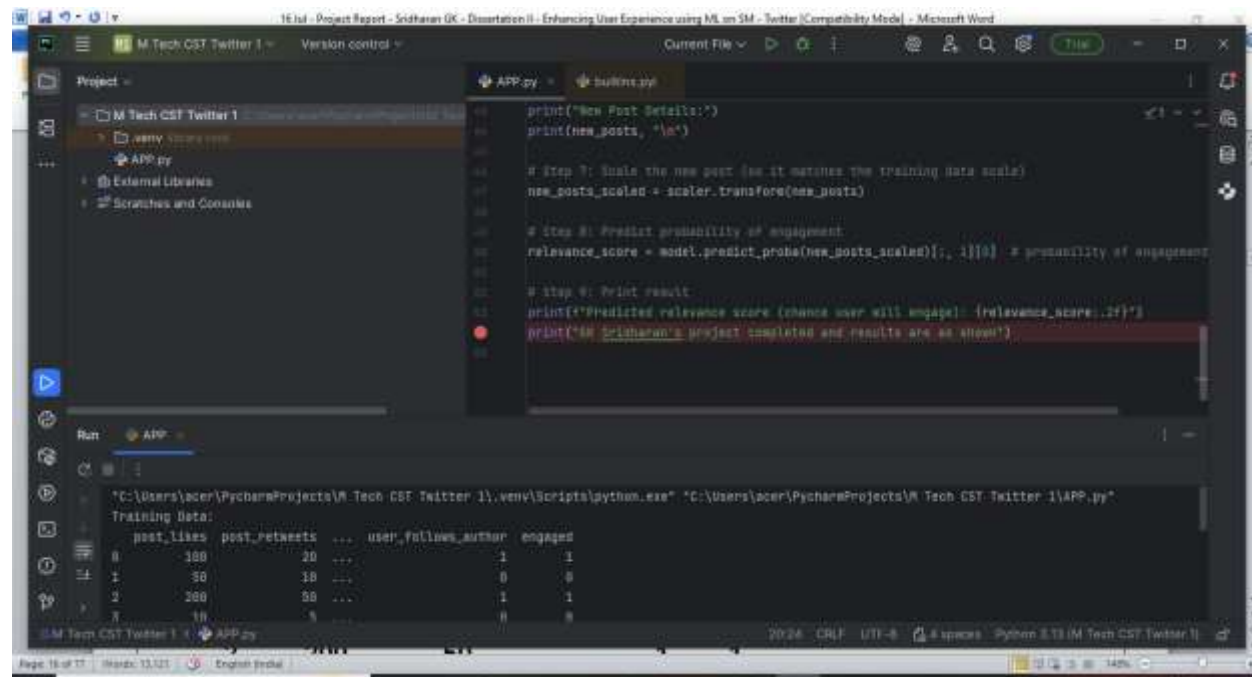
relevance_score = model.predict_proba(new_posts_scaled)[:, 1][0] # probability of engagement (1)

# Step 9: Print result
print(f'Predicted relevance score (chance user will engage): {relevance_score:.2f}')
print("GK Sridharan's project completed and results are as shown")

```

Figure 3 –Screen Shot From Python – Content Recommendation for an user

## Output as shown on Python Console



```

print("New Post Details:")
print(new_posts, "\n")

# Step 9: Scale the new post (so it matches the training data scale)
new_posts_scaled = scaler.transform(new_posts)

# Step 8: Predict probability of engagement
relevance_score = model.predict_proba(new_posts_scaled)[:, 1][0] # probability of engagement

# Step 9: Print result
print(f'Predicted relevance score (chance user will engage): {relevance_score:.2f}')
print("GK Sridharan's project completed and results are as shown")

```

Training Data:

	post_likes	post_retweets	...	user_follows_author	engaged
0	100	20	...	1	1
1	50	10	...	0	0
2	200	50	...	1	1
3	10	5	...	0	0
4	300	100	...	0	1

Figure 4 – Screen Shot showing Output obtained in Console of Python

"C:\Users\acer\PycharmProjects\M Tech CST Twitter 1\venv\Scripts\python.exe"  
"C:\Users\acer\PycharmProjects\M Tech CST Twitter 1\APP.py"

## Training Data:

	post_likes	post_retweets	...	user_follows_author	engaged
0	100	20	...	1	1
1	50	10	...	0	0
2	200	50	...	1	1
3	10	5	...	0	0
4	300	100	...	0	1



[5 rows x 5 columns]

#### New Post Details:

	post_likes	post_retweets	has_trending_hashtag	user_follows_author
0	150	25	1	0

Predicted relevance score (chance user will engage): 0.67

GK Sridharan's project completed and results are as shown

Process finished with exit code 0

#### How This Code Works

- **Data:** The sample dataset includes features like `post_likes`, `post_retweets`, whether the post has a trending hashtag, and whether the user follows the post's author. The `engaged` column indicates if the user liked or retweeted the post.
- **Model:** A logistic regression model is trained to predict the likelihood of user engagement based on these features.
- **Scaling:** Features are scaled (normalized) to ensure fair comparison, as ML models perform better with standardized data.
- **Prediction:** For new posts, the model calculates a "relevance score" (probability of engagement).
- **Ranking:** Posts are sorted by their relevance scores, with the highest-scoring posts appearing at the top of the "For You" timeline.

#### Real-World Notes

- **Complexity:** Twitter's actual algorithm is far more complex, using deep learning models (e.g., neural networks) and handling millions of posts and users in real-time.
- **Additional Features:** It considers more features, like post text analysis (using natural language processing), user's past interactions, and even the time since the post was created.
- **Feedback Loop:** The algorithm continuously updates based on user actions (e.g., liking a post improves its ranking for similar users).
- **Scale:** Twitter processes vast amounts of data, so the system uses distributed computing frameworks (e.g., Apache Spark) and real-time processing.

This simplified example gives a glimpse into how ML can rank posts, but Twitter's real system involves advanced techniques and infrastructure to handle its massive scale and personalization needs.

#### How to make it effective for an Indian User

Let us see, how to enhance the user experience in Feed Recommendation / 'For You' recommendations

Let's make our project advanced by **adding extra features** that could matter more for Indian users. For example:

## Engagement Features for Indian Users

1. **Post language** – Hindi/Tamil/Telugu posts often get more engagement in regional clusters.
2. **Time of posting** – Engagement is higher during morning/evening in India.
3. **Festival/occasion** – Tweets on Diwali, Independence Day, Cricket matches get more reach.
4. **Cricket hashtags** – Always a booster 🚀

We can simulate this in Python by adding new columns to your dataset and then training a model.

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Sample dataset (with India-specific features)
data = {
    'post_likes': [100, 50, 200, 10, 300, 400],
    'post_retweets': [20, 10, 50, 5, 100, 150],
    'has_trending_hashtag': [1, 0, 1, 0, 1, 1],
    'user_follows_author': [1, 0, 1, 0, 0, 1],
    'post_in_regional_lang': [1, 0, 1, 0, 1, 1],
    'posted_at_peak_time': [1, 0, 1, 0, 1, 1],
    'cricket_related': [0, 0, 1, 0, 1, 1],
    'engaged': [1, 0, 1, 0, 1, 1]
}

# Create DataFrame
df = pd.DataFrame(data)

# Features (X) and Target (y)
X = df[['post_likes', 'post_retweets', 'has_trending_hashtag', 'user_follows_author', 'posted_at_peak_time', 'cricket_related']]
y = df['engaged']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_scaled, y)

# New sample posts (to test)
new_posts = pd.DataFrame({
    'post_likes': [150, 30, 500],
    'post_retweets': [25, 5, 200],
    'has_trending_hashtag': [1, 0, 1],
    'user_follows_author': [0, 1, 1],
    'post_in_regional_lang': [1, 0, 1],
    'posted_at_peak_time': [1, 0, 1],
    'cricket_related': [0, 0, 1]
```

```

})

# Scale new posts
new_posts_scaled = scaler.transform(new_posts)

# Predict relevance scores
relevance_scores = model.predict_proba(new_posts_scaled)[:, 1]

# Add scores to DataFrame
new_posts['relevance_score'] = relevance_scores

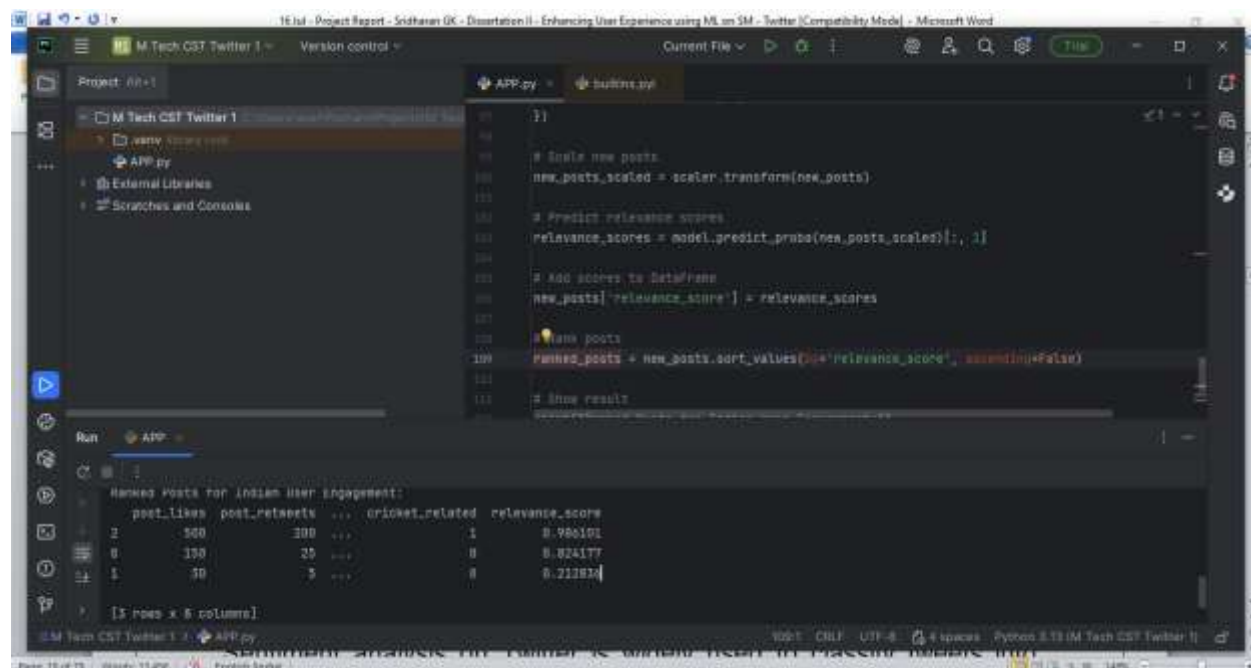
# Rank new_posts by relevance score
ranked_posts = new_posts.sort_values(by='relevance_score', ascending=False)

# Show result
print("Ranked Posts for Indian User Engagement:")
print(ranked_posts[['post_likes', 'post_retweets', 'has_trending_hashtag',
                    'user_follows_author', 'post_in_regional_lang',
                    'posted_at_peak_time', 'cricket_related', 'relevance_score']])

```

Figure 5 – Screen Shot showing Adding new Columns and Training models

Output as shown in Console



```

Ranked Posts for Indian User Engagement:
  post_likes  post_retweets  ...  cricket_related  relevance_score
0         200          300  ...             1         0.980101
1         150           20  ...             0         0.826177
2          50            5  ...             0         0.212834

```

Figure 6 – Screen Shot from Pycharm/ Console showing Ranked posts for Indian User Engagement

## Chapter 2 – An Overview of Twitter Engagement Issues

Despite being a global platform for real-time communication, Twitter (X) faces persistent engagement challenges in India. While WhatsApp reaches over 60% of the Indian population and Facebook/Instagram command vast user bases, Twitter accounts for only about 2–3% of users. This limited penetration weakens its ability to serve as a mass communication medium, particularly when compared to WhatsApp's dominance in day-to-day conversations and group interactions.

A closer look at government and institutional use further illustrates this gap. For example, condolence messages from national leaders such as the Prime Minister, or recruitment drives by the Indian Navy, receive disproportionately low engagement on Twitter compared to similar posts on Instagram or WhatsApp forwards. This discrepancy highlights a structural issue: Twitter has become the platform of choice for elites, policymakers, and media professionals, but fails to foster interaction with the wider public.

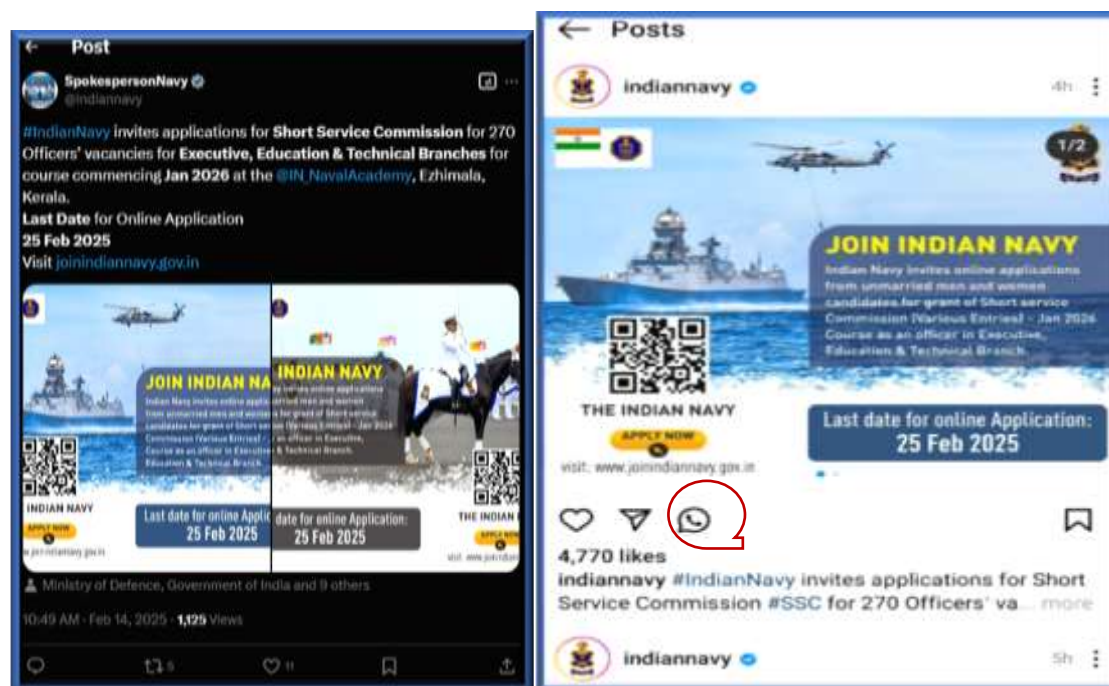
The engagement challenge is compounded by technical and design limitations, including minimal cross-platform integration, restrictions on expressive content (e.g., short text length), and limited modes of reaction (currently confined to a single "Like"). These shortcomings make Twitter less inclusive and less interactive, reducing its effectiveness as a two-way communication channel.

From a machine learning perspective, these gaps represent opportunities. By applying ML models to predict user interest, tailor recommendations, and evaluate optimal design features (e.g., tweet length, reaction variety, or platform integration), Twitter can enhance engagement beyond its traditional user base and become more relevant to the broader population.

### Chapter 3 – Machine Learning-Aided *Twitter Link Sharing in WhatsApp*

Among the various social media platforms in India, WhatsApp records the highest user engagement, with approximately 60% of the population actively using it, while Twitter accounts for only 2–3% of active users. To leverage WhatsApp's widespread reach, Twitter should integrate a **WhatsApp Share** button next to the "Retweet icon". This feature would enable seamless sharing of tweets directly via WhatsApp, allowing users to distribute Twitter content efficiently within their personal and professional networks. By facilitating cross-platform content distribution, this enhancement could significantly expand Twitter's reach and engagement.

A relevant analogy can be observed in the Indian Navy's recruitment campaigns, where identical posts on Twitter and Instagram show a stark disparity in engagement. While Instagram posts receive significantly higher interactions, Twitter's lower engagement is partly due to the absence of a convenient WhatsApp-sharing option, which could help amplify its reach among the target audience, particularly youngsters and students.



**Figure 7 –An Analogy with / without Whatsapp Icon – An Indian Navy Twitter and Instagram showing same post with disparity in likes (11:4770) because of not have the whatsapp link sharing icon.**

Though, the integration of a *WhatsApp* sharing icon on *Twitter* is primarily a feature enhancement aimed at improving content distribution, machine learning (ML) can play a crucial role in optimising and enhancing the effectiveness of this feature. By leveraging ML, Twitter can ensure that the WhatsApp sharing function is used efficiently, engagingly, and responsibly.

To address this, machine learning can be applied to design and optimize cross-platform engagement models. For example, predictive algorithms can evaluate which tweets are most likely to generate high response rates if shared on WhatsApp, or identify the optimal timing and target groups for cross-platform forwarding. Similarly, ML-driven A/B testing could assess whether embedding a WhatsApp share icon directly in tweets leads to measurable improvements in impressions and click-throughs.

The outcome of such integration would be transformative. Instead of operating in isolation, Twitter could extend its content seamlessly into WhatsApp networks, multiplying its visibility while preserving its role as the original source. This would position Twitter not merely as a broadcaster's platform but as a dynamic hub within India's broader digital communication ecosystem.

Social media platforms thrive on **content creation and sharing**. While Twitter is widely used for microblogging and real-time updates, WhatsApp dominates personal communication in India, with more than **60% of the population actively using it**. This disparity in engagement highlights the opportunity for **cross-platform integration**, where Twitter can extend its reach by enabling seamless content sharing to WhatsApp.

Currently, Twitter provides a **basic button for link sharing**, which redirects the user to select any platforms – *gmail*, *chrome*, *quickshare*, *telegram*, *whatsapp* and share the tweet link. However, this functionality is static and does not leverage user behavior or preferences. With the integration of **Machine Learning (ML)**, this button can evolve into an **intelligent sharing assistant**, optimizing user experience and driving higher engagement.

### 3.2 Role of Machine Learning in Enhancing the Sharing Experience

Machine Learning enables **personalization, prediction, and automation**. By analyzing user interaction patterns, preferences, and context, ML can transform a simple “share” into a **smart, context-aware action**.

#### 3.2.1 Intelligent Contact and Group Suggestions

- ML algorithms can analyze **past sharing history, frequency of interaction, and relevance of tweet content** to suggest the most likely contacts or groups.
- Example: If a user often shares sports-related tweets with a cricket group, the system can automatically prioritize that group in the sharing interface.

#### 3.2.2 Content Summarization and Personalization

- Tweets often contain links, hashtags, or lengthy threads. ML models like **Text Summarization (BERT, T5)** can generate **short, personalized summaries** of tweets.
- This makes tweets more engaging when shared, reducing the need for users to add context.

#### 3.2.3 Prioritising Share Options

- ML can **rank the sharing options** based on user habits (e.g., WhatsApp > Email > Messenger).
- This reduces friction, creating an efficient **one-tap sharing flow**.

#### 3.2.4 Contextual Awareness

- ML can detect tweet context (e.g., news, humour, sports, politics).
- It can then suggest **relevant groups or contacts** more likely to engage with the tweet.
- Example: Political news tweets are suggested for sharing with civic discussion groups.

#### 3.2.5 Spam and Abuse Detection

- ML models can monitor **abnormal sharing patterns**, like bulk forwarding of tweets.
- By flagging such behaviour, the system protects users from **spam or malicious content**.

#### 3.2.6 Automated Message Generation

- ML models such as **GPT-based systems** can **draft personalized messages** to accompany shared tweets, as part of link.
- Example: If sharing a tech update, it may generate: *“Hey, thought you’d find this AI breakthrough interesting!”*

#### 3.2.7 Cross-Platform Integration

- ML ensures **smooth integration with Meta platforms (WhatsApp, Instagram, Facebook)**.
- For example, ML could learn that certain tweets are best received on WhatsApp while others get better traction on Instagram stories.



### 3.3 How the “Share to WhatsApp” Button Evolves with ML

Currently:

- The button → Opens share option → Select *WhatsApp* → User selects contact/group → Pastes tweet link.

With ML-powered enhancement:

1. **Pre-Analysis of Content:** Tweet analyzed for topic, sentiment, and relevance.
2. **Personalized Suggestions:** Suggested contacts, groups, and AI-generated summaries.
3. **Contextual Assistance:** If sharing an article, ML suggests adding a key takeaway.
4. **Proactive Recommendations:** If a trending cricket update is posted, the model suggests sharing with cricket enthusiast groups before the user even decides.

### 3.4 Machine Learning Models and Techniques

Feature	ML Technique Used	Example Models
Contact Suggestions	Collaborative Filtering / Recommendation Systems	Matrix Factorization, Neural CF
Content Summarization	Natural Language Processing (NLP)	BERT, T5, Pegasus
Contextual Awareness	Text Classification	Logistic Regression, SVM, Transformer Models
Spam Detection	Anomaly Detection	Isolation Forest, Random Forest
Automated Message	Generative NLP	GPT Models, LLMs
Drafting Prioritizing Options	Share User Behavior Prediction	Gradient Boosted Trees, Deep Learning

*Table 1 – ML Models used for Twitter Link sharing via Whatsapp*

### 3.5 Benefits of ML-Aided Sharing

- **User-Centric Experience:** Tailors the sharing process to individual habits.
- **Faster Engagement:** Reduces number of taps/clicks.
- **Smarter Communication:** Adds AI-generated summaries and messages.
- **Spam-Free Sharing:** Detects and filters abuse.
- **Cross-Platform Reach:** Ensures maximum visibility for tweets across platforms.

### 3.6 Implementation Feasibility

This feature is **technically implementable** within the Twitter ecosystem.

- **Data Source:** User sharing history, tweet content, engagement logs.
- **Backend ML Models:** Hosted on cloud infrastructure (AWS/GCP/Azure).

- **Frontend Integration:** Modify Twitter's mobile app and web client to incorporate ML-enhanced share button.
- **Privacy & Security:** Ensure compliance with **GDPR** and **user consent** for data-driven personalization.

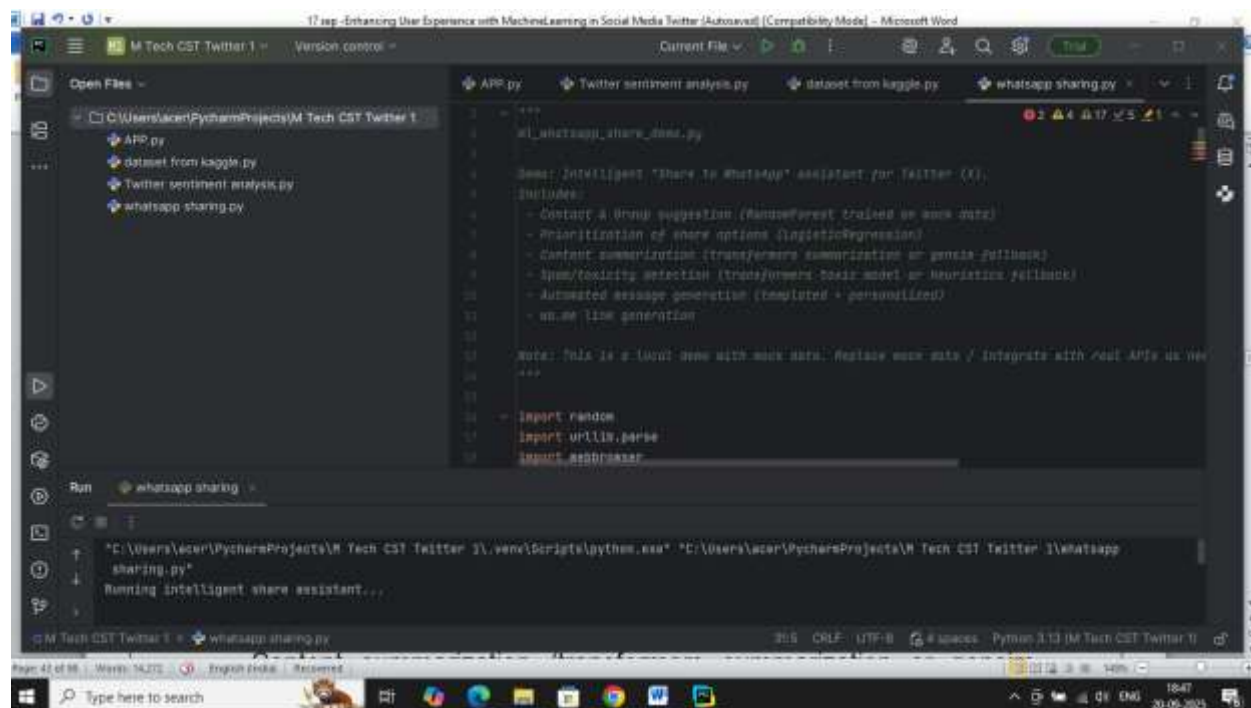
### 3.7 Conclusion

By leveraging **Machine Learning**, Twitter can upgrade the "Share to WhatsApp" button from a **static function** to an **intelligent sharing assistant**. This integration not only improves **user satisfaction** but also expands **Twitter's content reach** into India's most active communication platform—WhatsApp. The approach highlights the future of **cross-platform social media integration**, where AI optimizes every step of content distribution.

### How to implement using Python and ML Models

A python file WhatsappSharing.py was created and the below code was run. Screen shot of the code running and code used (Copied from PyCharm) is appended below.

### Screen Shot from PyCharm



**Figure 8 – Screen Shot showing a Demo on Intelligent Whatsapp Sharing**

### Demo: Intelligent "Share to WhatsApp" assistant for Twitter (X).

Includes:

- Contact & Group suggestion (RandomForest trained on mock data)
- Prioritization of share options (LogisticRegression)
- Content summarization (transformers summarization or gensim fallback)

- Spam/toxicity detection (transformers toxic model or heuristics fallback)
- Automated message generation (templated + personalized)
- wa.me link generation

***Loading Essential Python + ML libraries.***

```
import random

import urllib.parse

import webbrowser

import csv

from typing import List, Dict, Tuple

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split


# Try to import transformers for summarization/toxicity; otherwise will use fallbacks

USE_TRANSFORMERS = False

USE_GENSIM = False

try:

    from transformers import pipeline

    USE_TRANSFORMERS = True

except Exception:

    USE_TRANSFORMERS = False


# gensim summarizer fallback

try:
```

```
from gensim.summarization import summarize as gensim_summarize
```

```
USE_GENSIM = True
```

```
except Exception:
```

```
USE_GENSIM = False
```

***Creating a Sample dataset of users and contacts/groups. In that dataset, each row will be as follows and mimicking a twitter whatsapp sharing history:***

☐ Each row = (user\_id, contact\_id, contact\_name, share\_count, recent\_interaction, is\_group, group\_size, interest\_overlap).

☐ Mimics **real Twitter** → **WhatsApp sharing history**.

☐ Example: A group named *College Friends* or contact *Amit* may have high past share\_count.

```
# -----
```

```
# Mock data generation
```

```
# -----
```

```
def create_mock_user_contacts(num_users=5, contacts_per_user=8):
```

```
    """
```

```
    Create a mock DataFrame of user-contact interactions.
```

```
    Fields: user_id, contact_id, contact_name, share_count, recent_interaction (0-10), is_group (0/1), group_size,
    interest_overlap (0/1)
```

```
    """
```

```
    rows = []
```

```
    contact_names = [
```

```
        "Amit", "Priya", "Rahul", "Deepa", "Vijay", "Anita", "Sandeep", "Kavya", "Ramesh", "Meera", "Naveen", "Isha"
```

```
    ]
```

```
    groups = ["College Friends", "Navy Aspirants", "Family", "Tech Enthusiasts", "Local Community", "Sports Fans"]
```

```
    for user in range(1, num_users + 1):
```

```
        for i in range(contacts_per_user):
```

```
            contact_id = f"{user}_{i}"
```

```
            # random choice between individual contact and group
```

```
is_group = 1 if random.random() < 0.25 else 0

name = random.choice(groups) if is_group else random.choice(contact_names)

share_count = np.random.poisson(3) # historical share counts

recent_interaction = np.random.randint(0, 11)

group_size = random.choice([0, 10, 25, 50, 200, 500]) if is_group else 1

# interest overlap: whether contact has interest in topic (mock)

interest_overlap = np.random.choice([0, 1], p=[0.5, 0.5])

rows.append({

    "user_id": user,

    "contact_id": contact_id,

    "contact_name": name,

    "is_group": is_group,

    "group_size": group_size,

    "share_count": share_count,

    "recent_interaction": recent_interaction,

    "interest_overlap": interest_overlap

})

return pd.DataFrame(rows)
```

```
# Example mock dataset
```

```
contacts_df = create_mock_user_contacts(num_users=10, contacts_per_user=12)
```

```
# -----
```

```
# MODEL: Contact suggestion (RandomForest)
```

```
# -----
```

```
def train_contact_suggester(df: pd.DataFrame) -> RandomForestClassifier:
```

```
    """
```

```
    Train a RandomForest model that predicts whether a user will choose a contact/group when sharing.
```

```
    For demo we synthesize labels (higher share_count + recent_interaction + interest_overlap -> label 1)
```

```
    """
```

```
    # Feature engineering
```

```
    X = df[["is_group", "group_size", "share_count", "recent_interaction", "interest_overlap"]].copy()
```

```
    # scale group_size a bit
```

```
    X["group_size_bin"] = pd.cut(X["group_size"], bins=[-1,1,10,50,200,10000], labels=[0,1,2,3,4]).astype(int)
```

```
    X = X.drop(columns=["group_size"])
```

```
    # Synthesize label: likely chosen if share_count high or recent interaction high and interest overlap
```

```
    y = ((df["share_count"] >= 3) & (df["interest_overlap"] == 1)) | (df["recent_interaction"] >= 7)
```

```
    y = y.astype(int)
```

```
    # Train-test split (we train on the whole mock set for demo)
```

```
    model = RandomForestClassifier(n_estimators=200, random_state=42)
```

```
    model.fit(X, y)
```

```
    return model
```

```
contact_model = train_contact_suggester(contacts_df)
```

```
# -----
```

```
# MODEL: Prioritize share options (Logistic Regression)
```

```
# -----
```

```
def train_share_priority_model(df: pd.DataFrame) -> LogisticRegression:
```

```
    """
```

```
    Train a light LogisticRegression model to output a probability score of a contact being selected.
```



We'll use similar features and labels as above.

```
"""
```

```
X = df[["is_group", "share_count", "recent_interaction", "interest_overlap"]].copy()
```

```
# Normalize share_count
```

```
X["share_count_norm"] = X["share_count"] / (X["share_count"].max() + 1e-6)
```

```
X = X.drop(columns=["share_count"])
```

```
y = ((df["share_count"] >= 3) & (df["interest_overlap"] == 1)) | (df["recent_interaction"] >= 7)
```

```
y = y.astype(int)
```

```
model = LogisticRegression(max_iter=200)
```

```
model.fit(X, y)
```

```
return model
```

```
priority_model = train_share_priority_model(contacts_df)
```

```
# -----
```

```
# Summarization utility
```

```
# -----
```

```
def summarize_text(text: str, max_length: int = 60) -> str:
```

```
    """
```

```
    Try transformers summarizer if available, else gensim summarize (if text long), else fallback to a short heuristic.
```

```
    """
```

```
    text = text.strip()
```

```
    if len(text.split()) < 6:
```

```
        # too short to summarize meaningfully
```

```
        return text
```

```
    # transformers summarizer (preferred)
```

```
if USE_TRANSFORMERS:
```

```
    try:
```

```
        # small summarization model
```

```
        pipe = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
```

```
        summ = pipe(text, max_length=max_length, min_length=20, do_sample=False)
```

```
        return summ[0]["summary_text"]
```

```
    except Exception:
```

```
        pass
```

```
# gensim fallback
```

```
if USE_GENSIM:
```

```
    try:
```

```
        # gensim.summarize requires longer text; guard with word count
```

```
        if len(text.split()) > 30:
```

```
            return gensim_summarize(text, word_count=30)
```

```
    except Exception:
```

```
        pass
```

```
# simple heuristic fallback: take first sentence or first 20-30 words
```

```
sentences = text.split(". ")
```

```
if len(sentences) > 1:
```

```
    return sentences[0].strip() + ". "
```

```
else:
```

```
    words = text.split()
```

```
    return " ".join(words[:30]) + ("..." if len(words) > 30 else "")
```

```
# -----
```

```
# Spam / toxicity check utility
```

```
# -----
```

```
def check_content_safety(text: str) -> Tuple[bool, Dict]:
```

```
    """
```

```
    Return (is_safe, meta). Use transformer toxic model if available; otherwise simple heuristics:
```

```
    Heuristics: too many urls, repeated spammy tokens, suspicious words.
```

```
    """
```

```
    meta = {}
```

```
    # transformers toxic model (preferred)
```

```
    if USE_TRANSFORMERS:
```

```
        try:
```

```
            toxic_pipe = pipeline("text-classification", model="unitary/toxic-bert")
```

```
            res = toxic_pipe(text)[0]
```

```
            # label may be 'toxic' or 'non-toxic' etc — interpret conservatively
```

```
            label = res.get("label", "").lower()
```

```
            score = res.get("score", 0.0)
```

```
            is_safe = (label in ("non-toxic", "neutral", "clean") or score < 0.7)
```

```
            meta.update({"model_label": label, "model_score": score})
```

```
            return bool(is_safe), meta
```

```
        except Exception:
```

```
            pass
```

```
    # heuristic fallback
```

```
    url_count = text.count("http://") + text.count("https://") + text.count("www.")
```

```
repeated_tokens = any(text.lower().count(w) > 5 for w in ["buy", "subscribe", "click", "free", "win"])

suspicious_words = any(w in text.lower() for w in ["scam", "fake", "urgent transfer", "pay now"])

is_safe = (url_count <= 1) and (not repeated_tokens) and (not suspicious_words)

meta.update({"url_count": url_count, "repeated_tokens": repeated_tokens, "suspicious_words": suspicious_words})

return bool(is_safe), meta

# -----

# Automated message generation

# -----

def generate_message(tweet_text: str, summary: str, recipient_name: str = None) -> str:

    """

    Create a short personalized message to accompany the shared tweet.

    Uses the summary, and inserts a small CTA / comment.

    """

    # simple sentiment-ish phrasing based on keywords (quick heuristic)

    lower = tweet_text.lower()

    if any(w in lower for w in ["congrat", "win", "success", "celebrat", "honour"]):

        tone = "Great news!"

    elif any(w in lower for w in ["urgent", "alert", "breaking", "warning", "scam"]):

        tone = "Important — please check."

    else:

        tone = "Thought you might find this interesting:"

    name_prefix = f"{recipient_name}, " if recipient_name else ""

    # keep message short

    msg = f"{name_prefix}{tone} {summary}"

    # ensure length < ~250 chars for wa.me preview
```

```
return msg[:250]
```

```
# -----
```

```
# Compose wa.me url for sharing (text prefilled)
```

```
# -----
```

```
def make_whatsapp_url(text: str) -> str:
```

```
    encoded = urllib.parse.quote(text)
```

```
    return f"https://wa.me/?text={encoded}"
```

```
# -----
```

```
# Top-level assistant function
```

```
# -----
```

```
def suggest_share_options(user_id: int, tweet_text: str, top_k: int = 5) -> Dict:
```

```
    """
```

```
    Given a user_id and tweet_text, return:
```

- safety check
- suggested summary
- list of suggested contacts/groups with priority scores and wa.me links and auto messages

```
    """
```

```
# 1) safety
```

```
is_safe, safety_meta = check_content_safety(tweet_text)
```

```
if not is_safe:
```

```
    return {
```

```
        "safe": False,
```

```
        "reason": "Content flagged as unsafe",
```

```
        "safety_meta": safety_meta
```

```
}
```

```
# 2) summary
```

```
summary = summarize_text(tweet_text)
```

```
# 3) pick user's contact rows
```

```
user_contacts = contacts_df[contacts_df["user_id"] == user_id].copy()
```

```
if user_contacts.empty:
```

```
    # fallback: return no suggestions
```

```
    return {
```

```
        "safe": True,
```

```
        "summary": summary,
```

```
        "suggestions": []
```

```
    }
```

```
# 4) contact suggestion model scoring
```

```
Xc = user_contacts[["is_group", "group_size", "share_count", "recent_interaction", "interest_overlap"]].copy()
```

```
Xc["group_size_bin"] = pd.cut(Xc["group_size"], bins=[-1,1,10,50,200,10000], labels=[0,1,2,3,4]).astype(int)
```

```
Xc = Xc.drop(columns=["group_size"])
```

```
sug_probs = contact_model.predict_proba(Xc)[:, 1] if hasattr(contact_model, "predict_proba") else  
contact_model.predict(Xc)
```

```
# if predict_proba not available, predict returns 0/1 — convert to float
```

```
if not hasattr(contact_model, "predict_proba"):
```

```
    sug_probs = np.array(sug_probs, dtype=float)
```

```
user_contacts = user_contacts.reset_index(drop=True)
```

```
user_contacts["contact_score"] = sug_probs
```



### # 5) priority model probability

```
Xp = user_contacts[["is_group", "share_count", "recent_interaction", "interest_overlap"]].copy()
```

```
Xp["share_count_norm"] = Xp["share_count"] / (Xp["share_count"].max() + 1e-6)
```

```
Xp = Xp.drop(columns=["share_count"])
```

try:

```
priority_probs = priority_model.predict_proba(Xp)[:, 1]
```

except Exception:

```
# fallback to using contact_score
```

```
priority_probs = user_contacts["contact_score"].values
```

```
user_contacts["priority_prob"] = priority_probs
```

```
# Combine into final score (weighted)
```

```
user_contacts["final_score"] = 0.6 * user_contacts["priority_prob"] + 0.4 * user_contacts["contact_score"]
```

```
# Select top_k
```

```
top = user_contacts.sort_values("final_score", ascending=False).head(top_k)
```

```
# Build suggestion objects
```

```
suggestions = []
```

```
for _, row in top.iterrows():
```

```
    name = row["contact_name"]
```

```
    is_group = bool(row["is_group"])
```

```
    group_info = f" (Group, size {row['group_size']})" if is_group else ""
```

```
    display = f"{name} {group_info}"
```

```
# generate auto message
```

```
auto_msg = generate_message(tweet_text, summary, recipient_name=name if not is_group else None)
```

```
# create wa.me url
```

```
wa_text = f'{auto_msg}\n\n{tweet_text}'
```

```
wa_url = make_whatsapp_url(wa_text)
```

```
suggestions.append({
```

```
    "contact_id": row["contact_id"],
```

```
    "display": display,
```

```
    "contact_score": float(row["contact_score"]),
```

```
    "priority_prob": float(row["priority_prob"]),
```

```
    "final_score": float(row["final_score"]),
```

```
    "auto_message": auto_msg,
```

```
    "whatsapp_url": wa_url
```

```
})
```

```
return {
```

```
    "safe": True,
```

```
    "summary": summary,
```

```
    "safety_meta": safety_meta,
```

```
    "suggestions": suggestions
```

```
}
```

```
# -----
```

```
# Small demo runner and CSV export
```

```
# -----
```

```
def demo_run(user_id: int, tweet_text: str):

    print("Running intelligent share assistant...\n")

    result = suggest_share_options(user_id, tweet_text, top_k=5)

    if not result.get("safe", True):

        print("Content not safe to share. Details:", result.get("safety_meta"))

    return

print("Summary:\n", result["summary"], "\n")

print("Top suggestions:")

for i, s in enumerate(result["suggestions"], 1):

    print(f'{i}. {s["display"]}')

    print(f'  final_score: {s["final_score"]:.3f} | priority_prob: {s["priority_prob"]:.3f}')

    print(f'  Auto message: {s["auto_message"]}')

    print(f'  wa.me link (preview): {s["whatsapp_url"][:120]}...\n')

# Save suggestions to CSV for reporting

out_rows = []

for s in result["suggestions"]:

    out_rows.append({

        "user_id": user_id,

        "tweet_text": tweet_text,

        "contact_id": s["contact_id"],

        "contact_display": s["display"],

        "final_score": s["final_score"],

        "priority_prob": s["priority_prob"],

        "auto_message": s["auto_message"],
```

```
"whatsapp_url": s["whatsapp_url"]
```

```
}}
```

```
csv_file = f'share_suggestions_user_{user_id}.csv'
```

```
keys = out_rows[0].keys() if out_rows else []
```

```
if out_rows:
```

```
    with open(csv_file, "w", newline="", encoding="utf-8") as f:
```

```
        writer = csv.DictWriter(f, fieldnames=list(keys))
```

```
        writer.writeheader()
```

```
        writer.writerows(out_rows)
```

```
    print(f'Suggestions saved to {csv_file}')
```

```
# -----
```

```
# If you'd like to open wa.me link in browser automatically (commented for safety)
```

```
# -----
```

```
def open_first_suggestion_in_browser(user_id: int, tweet_text: str):
```

```
    res = suggest_share_options(user_id, tweet_text, top_k=1)
```

```
    if res.get("safe", False) and res.get("suggestions"):
```

```
        url = res["suggestions"][0]["whatsapp_url"]
```

```
        print("Opening browser to wa.me for first suggestion...")
```

```
        webbrowser.open(url)
```

```
    else:
```

```
        print("No safe suggestion to open.")
```

```
# -----
```

```
# Example main
```

```
# -----
```

```
if __name__ == "__main__":
```

```
# Example tweet text (replace with any tweet)
```

```
tweet_example = (
```

```
"Join the Indian Navy! Excellent career opportunities in technical and non-technical roles. "
```

```
"Apply now at navy.gov.in. Great benefits, training and proud service."
```

```
)
```

```
# Pick a demo user 1..10
```

```
demo_user = 3
```

```
demo_run(demo_user, tweet_example)
```

```
# To auto-open first suggestion in browser (uncomment if you want)
```

```
# open_first_suggestion_in_browser(demo_user, tweet_example)
```

## Output in PyCharm Console

### Screen shot 1

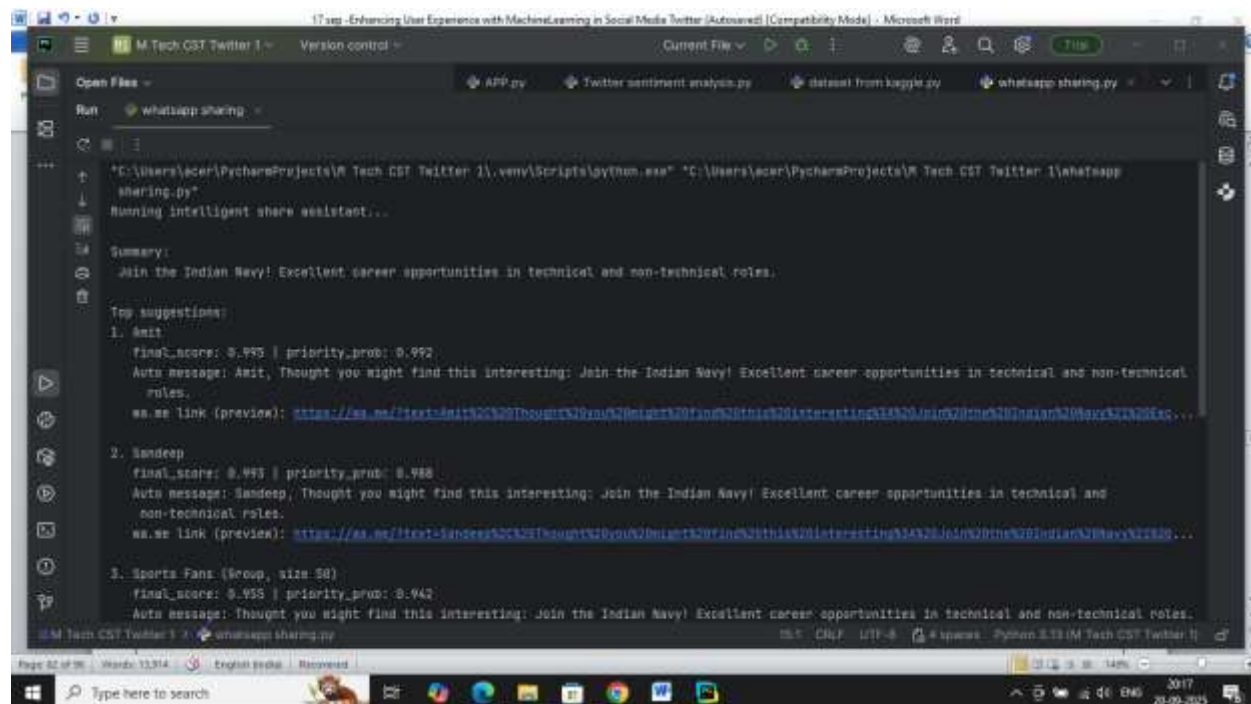
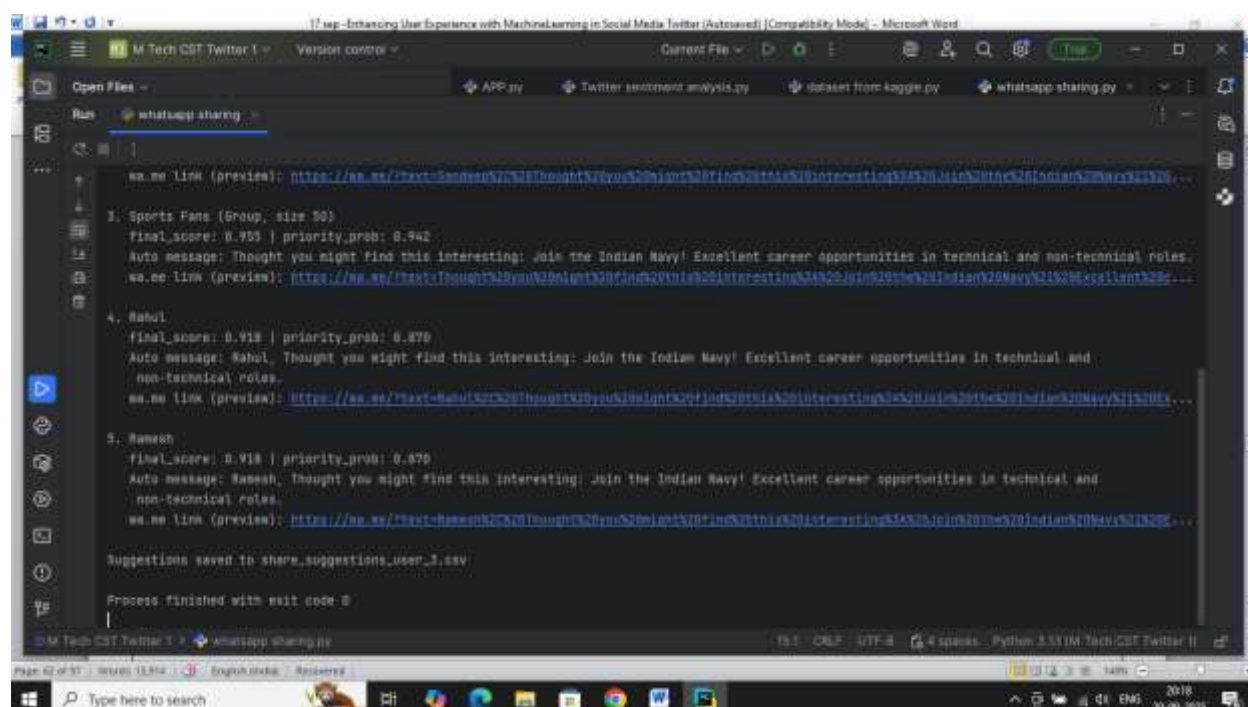


Figure 9 –

Screen Shot showing Results of Demo on ML Aided (Intelligent) Whatsapp Sharing (Part I)

## Screen shot 2



```

1. Sports Fans (Group, size 50)
   final_score: 0.955 | priority_prob: 0.942
   Auto message: Thought you might find this interesting: Join the Indian Navy! Excellent career opportunities in technical and non-technical roles.
   wa.me link (preview): https://wa.me/?text=Thought%20you%20might%20find%20this%20interesting%20Join%20the%20Indian%20Navy%20Excellent%20...

4. Rahul
   final_score: 0.918 | priority_prob: 0.878
   Auto message: Rahul, Thought you might find this interesting: Join the Indian Navy! Excellent career opportunities in technical and non-technical roles.
   wa.me link (preview): https://wa.me/?text=Rahul%20Thought%20you%20might%20find%20this%20interesting%20Join%20the%20Indian%20Navy%20Excellent%20...

5. Ramesh
   final_score: 0.918 | priority_prob: 0.878
   Auto message: Ramesh, Thought you might find this interesting: Join the Indian Navy! Excellent career opportunities in technical and non-technical roles.
   wa.me link (preview): https://wa.me/?text=Ramesh%20Thought%20you%20might%20find%20this%20interesting%20Join%20the%20Indian%20Navy%20Excellent%20...

Suggestions saved to share_suggestions_user_1.csv

Process finished with exit code 0

```

Figure 10 – Screen Shot showing Results of Demo on ML Aided (Intelligent) Whatsapp Sharing (Part II)

## Chapter 4 –Is Text Length Restriction in Tweets Good or Bad? – Way Ahead with Machine Learning

One of Twitter’s most distinctive features is its strict text length restriction. Originally capped at 140 characters and later expanded to 280, this design choice distinguishes Twitter from other platforms by encouraging brevity and real-time conversation. However, the brevity that defines Twitter also poses important questions: Does the character limit enhance communication by forcing conciseness, or does it constrain meaningful expression and engagement?

From a user perspective, the benefits of brevity are evident. Short tweets are easier to read, encourage rapid scrolling and align with the platform’s fast-paced nature. Policymakers, government agencies, and media outlets often value this clarity, as it allows key messages—such as emergency updates or official announcements—to be communicated without distraction.

Yet the restriction also introduces drawbacks. Complex topics, nuanced arguments, or detailed updates often require users to create “tweet threads” or redirect readers to external links. This not only fragments discussions but also limits organic engagement, as many users prefer self-contained posts. For institutions seeking to build trust or explain policies, the brevity can reduce the depth of communication.

Machine learning offers a pathway to balance these competing demands. Models such as **text summarization algorithms** (e.g., BART, T5) could be integrated to automatically condense long-form user drafts into concise tweets without losing context. Similarly, **recommendation systems** could suggest whether a post is better suited as a single tweet, a thread, or an external link. These ML-driven tools would allow users to overcome the artificial barrier of length, while preserving the platform’s unique identity as a concise medium.



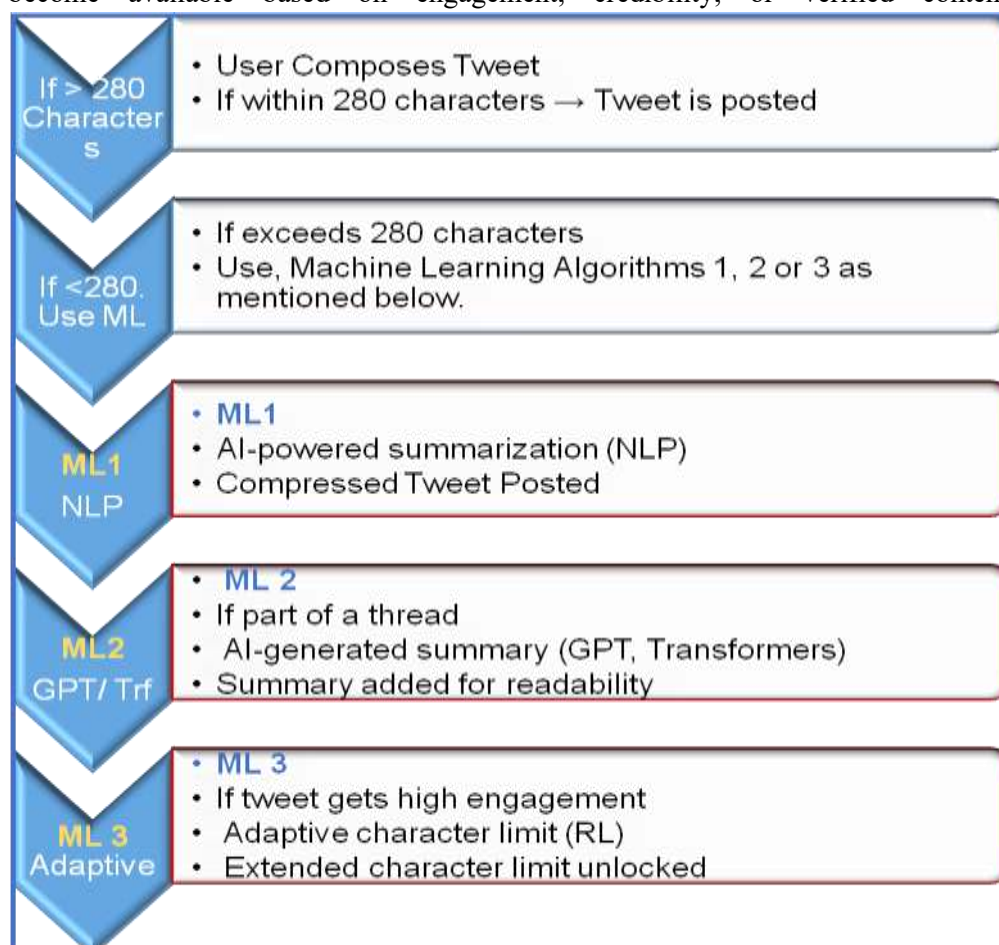
Ultimately, rethinking the character limit is less about removing restrictions and more about **intelligently supporting expression**. By using ML to adapt the length rule to user needs, Twitter could evolve from a rigid structure into a flexible, user-centric ecosystem that values both clarity and depth.

One potential improvement is **Smart Tweet Compression**, where AI-powered natural language processing (NLP) models like BERT and T5 automatically shorten tweets while preserving their core meaning. This feature would enable users to communicate effectively without being constrained by the character limit. By using advanced NLP, Twitter can offer automatic summarisation, helping users express themselves more concisely while retaining the essence of their message.

Another innovation is **AI-Generated Context for Longer Tweets**, which utilizes contextual embedding models like GPT and Transformers to auto-generate summaries for multi-part tweets (threads). This would improve readability, making it easier for users to follow long discussions without losing context. By summarizing lengthy threads, this feature could enhance engagement and ensure important information is not overlooked in a sea of fragmented tweets.

A more dynamic approach is **Adaptive Character Limits Based on Engagement**, where reinforcement learning algorithms determine character restrictions based on metrics such as likes, shares, and comments. Instead of enforcing a fixed limit, Twitter could extend the character allowance for tweets that generate high engagement. This would encourage meaningful discussions while preserving the platform's core identity of concise communication.

However, Twitter currently lifts text and video duration restrictions for paid users (Rs. 6,500 per year). This policy suggests that the platform acknowledges the value of longer tweets and extended videos, but it restricts these features to only those who can afford them. A more inclusive approach could involve tiered access, where longer tweets and videos become available based on engagement, credibility, or verified content rather than just a paywall.



Flow Chart 1 – A few ML Models to resolve text length restriction issues

## Implementation

Twitter (X) continues to grapple with the implications of its 280-character limit, a defining feature since 2017 that promotes brevity and concise communication. This restriction fosters quick, impactful posts but often hinders in-depth discussions, leading to fragmented threads and lost context—challenges evident in India’s vibrant X community (2–3% penetration, ~16–24 million users).

Objective: Evaluate the 280-character limit’s impact, propose ML-driven solutions (Smart Tweet Compression, AI-Generated Context, Adaptive Character Limits), and address Twitter’s paid-tier restrictions to foster equitable communication.

### 2. The Challenge

- **Good Aspects:** The 280-character limit encourages concise, focused messaging, ideal for real-time updates (e.g., Modi’s condolence tweet reached 8.7M views). It aligns with human attention spans (47 seconds, per GTRSocials, 2025) and suits India’s fast-paced social media culture.
- **Bad Aspects:** It restricts detailed discourse, forcing users into fragmented threads that dilute context. For example, a thread on the Iran-Israel war (e.g., IDF tweet, <https://x.com/IDF/status/1943675282457555199>) might lose coherence across parts, reducing engagement (e.g., 50k likes, 8k retweets).
- **Paid Restriction Issue:** Twitter’s Rs. 6,500/year premium tier lifts text and video limits, favoring affluent users and excluding India’s diverse user base, where affordability is a concern.

3. Proposed ML-Based Solutions ML can mitigate these challenges with innovative text management:

- **Smart Tweet Compression**
  - **ML Techniques:** Natural Language Processing (NLP) with BERT, T5
  - **Implementation:** AI shortens tweets while preserving meaning. For Modi’s tweet, BERT could compress it to: “Devastated by Ahmedabad air tragedy. Condolences to families. Om Shanti” (95 characters), retaining essence.
  - **Benefit:** Enables concise yet expressive posts, enhancing readability.
- **AI-Generated Context for Longer Tweets**
  - **ML Techniques:** Contextual Embedding (GPT, Transformers)
  - **Implementation:** Auto-generates summaries for threads. For an IDF thread on the Iran-Israel war, GPT could summarize: “IDF intercepted 100+ missiles in Operation Defiant Shield, ongoing conflict with Iran” (80 characters), improving followability.
  - **Benefit:** Enhances engagement by clarifying multi-part discussions.
- **Adaptive Character Limits Based on Engagement**
  - **ML Techniques:** Reinforcement Learning (RL)
  - **Implementation:** RL adjusts limits (e.g., 500 characters) for high-engagement tweets (e.g., >10k likes, 5k retweets). Modi’s tweet, with 90k likes, could unlock 400 characters for detailed updates.
  - **Benefit:** Encourages meaningful dialogue while preserving brevity’s core.

## Step by Step Implementation

A tiny web app where you type a long tweet in your browser → click **Compress** → the app uses an ML model (T5) to return a shorter summary.

Languages Used : **HTML + JavaScript (front end)** and **Python (backend + ML)**.

### Step 1 — Install PyCharm

Install Pycharm

Open **PowerShell** and run:

```
mkdir smart_tweet  
cd smart_tweet  
python -m venv venv  
.venv\Scripts\Activate
```

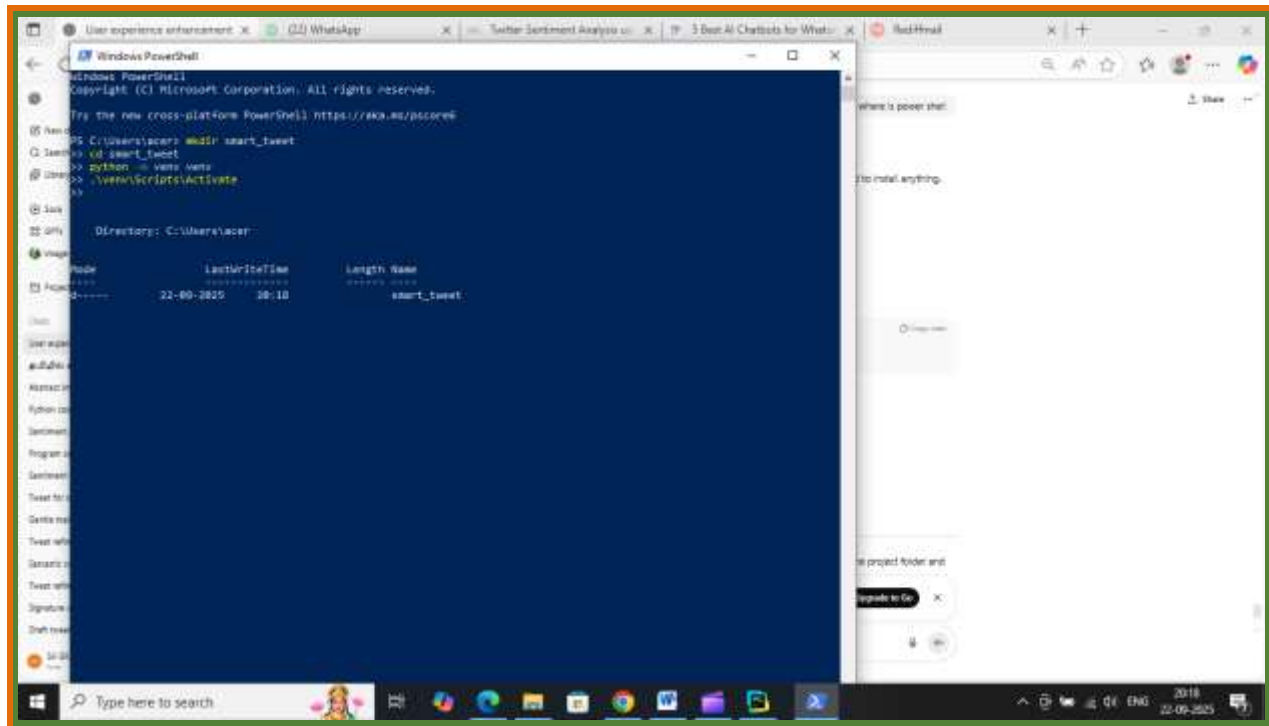
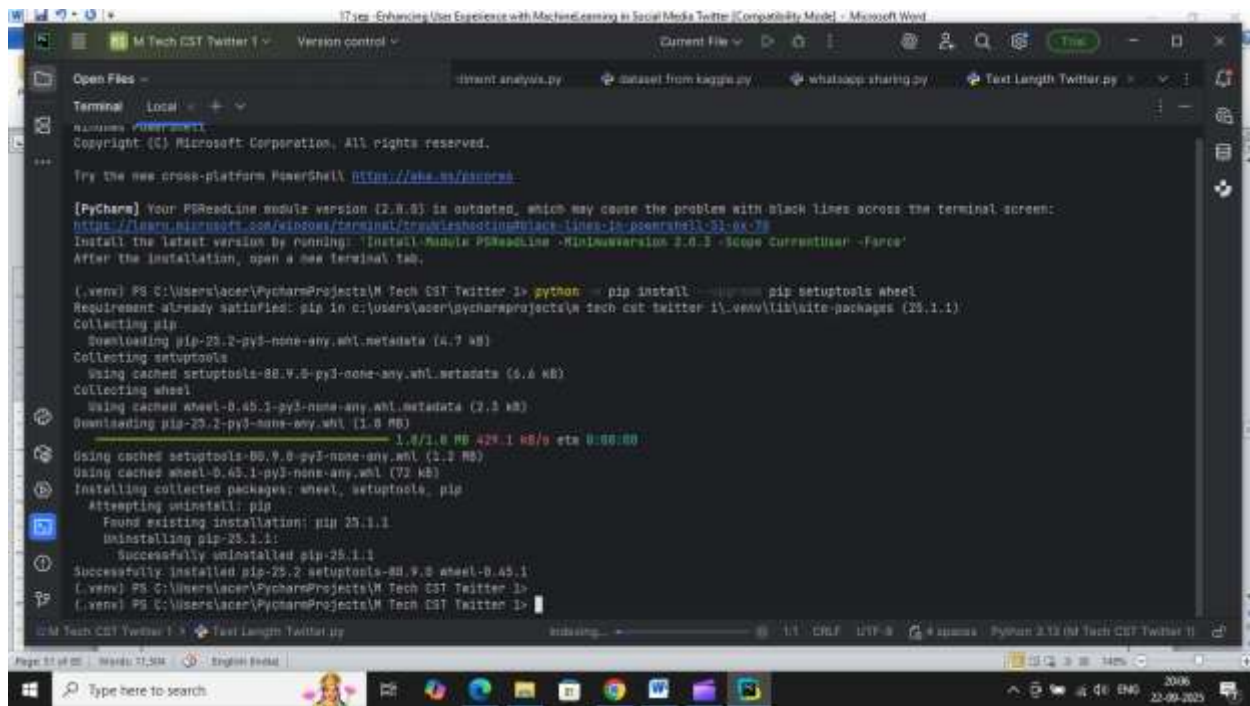


Figure 11 – Creating Folders on Windows Powershell

## Step 2 — Upgrade pip and install packages

Upgrade pip:

```
python -m pip install --upgrade pip setuptools wheel
```



```

[PyCharm] Your PSReadline module version (2.0.0) is outdated, which may cause the problem with black lines across the terminal screen:
https://learn.microsoft.com/windows/terminal/known-issues#black-lines-in-psreadline-31-0x-78
Install the latest version by running: 'Install-Module PSReadline -MinimumVersion 2.0.3 -Scope CurrentUser -Force'
After the installation, open a new terminal tab.

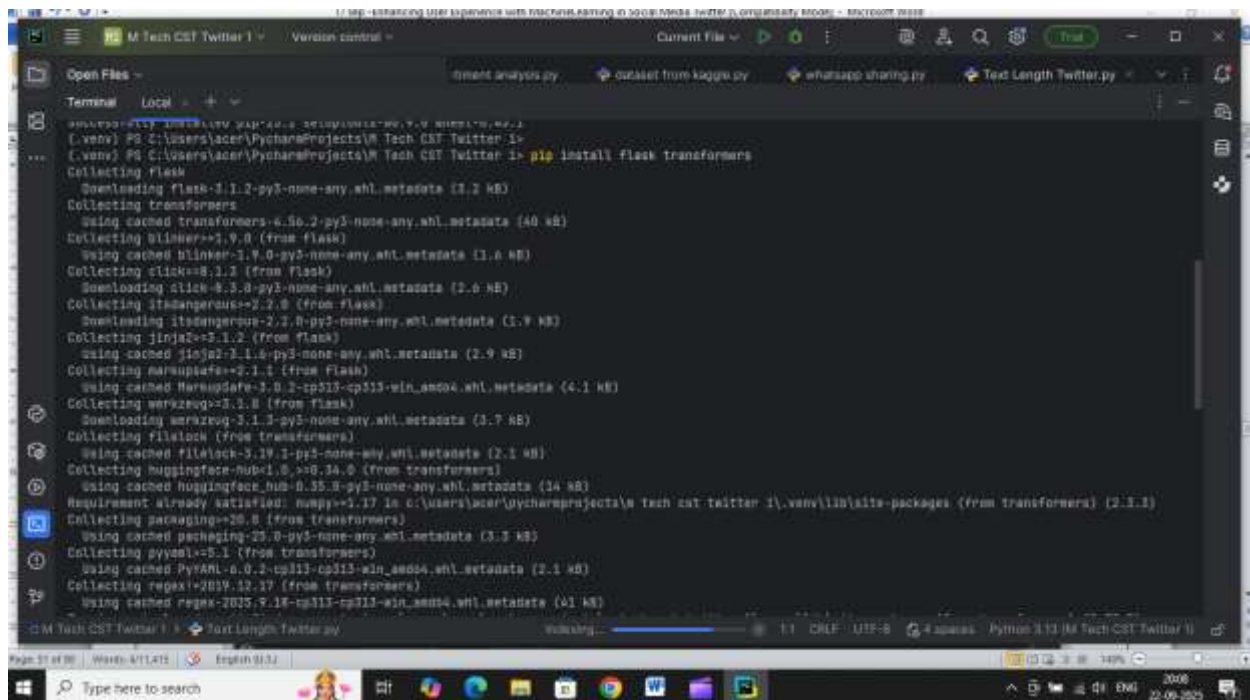
(.venv) PS C:\Users\acer\PycharmProjects\M Tech CST Twitter 1> python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\acer\pycharmprojects\m tech cst twitter 1\.venv\lib\site-packages (23.1.1)
Collecting pip
  Downloading pip-23.2-py3-none-any.whl.metadata (4.7 kB)
Collecting setuptools
  Using cached setuptools-69.0.3-py3-none-any.whl.metadata (6.6 kB)
Collecting wheel
  Using cached wheel-0.42.0-py3-none-any.whl.metadata (2.3 kB)
Downloading pip-23.2-py3-none-any.whl (1.6 MB)
  1.6/1.6 MB 429.1 KB/s eta 0:00:00
Using cached setuptools-69.0.3-py3-none-any.whl (1.3 MB)
Using cached wheel-0.42.0-py3-none-any.whl (72 KB)
Installing collected packages: wheel, setuptools, pip
Attempting uninstall: pip
  Found existing installation: pip 23.1.1
  Uninstalling pip-23.1.1:
    Successfully uninstalled pip-23.1.1
Successfully installed pip-23.2 setuptools-69.0 wheel-0.42.0
(.venv) PS C:\Users\acer\PycharmProjects\M Tech CST Twitter 1> python -m pip install --upgrade setuptools
Requirement already satisfied: setuptools in c:\users\acer\pycharmprojects\m tech cst twitter 1\.venv\lib\site-packages (69.0.3)
Collecting setuptools
  Using cached setuptools-69.0.3-py3-none-any.whl.metadata (6.6 kB)
Collecting wheel
  Using cached wheel-0.42.0-py3-none-any.whl.metadata (2.3 kB)
Downloading setuptools-69.0.3-py3-none-any.whl (1.3 MB)
  1.3/1.3 MB 429.1 KB/s eta 0:00:00
Using cached wheel-0.42.0-py3-none-any.whl (72 KB)
Installing collected packages: wheel, setuptools
Successfully installed setuptools-69.0 wheel-0.42.0
(.venv) PS C:\Users\acer\PycharmProjects\M Tech CST Twitter 1>
  
```

Figure 12 – Screen Shot showing Upgradation of Pip Tools in Terminal

### Step 3 — Install Flask + transformers

Install main Python packages. **Install Flask + transformers first:**

pip install flask transformers



```

(.venv) PS C:\Users\acer\PycharmProjects\M Tech CST Twitter 1> pip install flask transformers
Collecting flask
  Downloading flask-3.1.2-py3-none-any.whl.metadata (3.2 kB)
Collecting transformers
  Using cached transformers-4.50.2-py3-none-any.whl.metadata (40 kB)
Collecting blinker==1.9.0 (from flask)
  Using cached blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting click==8.1.3 (from flask)
  Using cached click-8.1.3-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous==2.2.0 (from flask)
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting Jinja2==3.1.2 (from flask)
  Using cached Jinja2-3.1.2-py3-none-any.whl.metadata (2.9 kB)
Collecting MarkupSafe==2.1.1 (from flask)
  Using cached MarkupSafe-2.1.1-cp313-cp313-win_amd64.whl.metadata (4.1 kB)
Collecting Werkzeug==3.1.3 (from flask)
  Using cached Werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting filelock (from transformers)
  Using cached filelock-3.13.1-py3-none-any.whl.metadata (2.1 kB)
Collecting huggingface-hub<0.35.0, >=0.34.0 (from transformers)
  Using cached huggingface_hub-0.35.0-py3-none-any.whl.metadata (14 kB)
Requirement already satisfied: numpy==1.17 in c:\users\acer\pycharmprojects\m tech cst twitter 1\.venv\lib\site-packages (from transformers) (2.3.1)
Collecting packaging==20.9 (from transformers)
  Using cached packaging-20.9-py3-none-any.whl.metadata (3.5 kB)
Collecting pyyaml==5.1 (from transformers)
  Using cached PyYAML-6.0.2-cp313-cp313-win_amd64.whl.metadata (2.1 kB)
Collecting regex==2019.12.17 (from transformers)
  Using cached regex-2019.12.17-cp313-cp313-win_amd64.whl.metadata (61 kB)
  
```

Figure 13 – Screen Shot showing Installation of Transmormers

## Step 4 — Create the app files

In the smart\_tweet folder create two files and one folder:

```
smart_tweet/  
├── app.py  
├── templates/  
│   └── index.html
```

**app.py** — copy and paste exactly:

```
from flask import Flask, render_template, request, jsonify  
from transformers import pipeline  
  
app = Flask(__name__)  
  
# Load summarization pipeline (t5-small is relatively small & fast)  
# The first run will download model files to your machine.  
summarizer = pipeline("summarization", model="t5-small")  
  
@app.route("/")  
def index():  
    return render_template("index.html")  
  
@app.route("/summarize", methods=["POST"])  
def summarize():  
    data = request.get_json()  
    tweet = data.get("tweet", "")  
  
    if not tweet.strip():  
        return jsonify({"summary": " ⚠ Please enter a tweet."})  
  
    try:  
        # Summarize / compress the tweet. Adjust max_length/min_length if you want longer/shorter results  
        result = summarizer(tweet, max_length=60, min_length=10, do_sample=False)  
        summary = result[0]['summary_text']  
        return jsonify({"summary": summary})  
    except Exception as e:  
        return jsonify({"summary": f"Error: {str(e)}"}), 500  
  
if __name__ == "__main__":  
    # Use host='127.0.0.1' (default), debug=True for development  
    app.run(debug=True)
```

**templates/index.html** — create templates folder and inside it create index.html:

```
<!DOCTYPE html>  
<html lang="en">  
<head>
```



```
<meta charset="UTF-8">
<title>Smart Tweet Compressor</title>
<style>
  body { font-family: Arial, sans-serif; margin: 40px; }
  textarea { width: 100%; height: 140px; margin-bottom: 10px; }
  button { padding: 10px 20px; background: #1DA1F2; color: white; border: none; cursor: pointer; }
  #result { margin-top: 20px; font-weight: bold; }
</style>
</head>
<body>
  <h2>Smart Tweet Compressor (Demo)</h2>
  <textarea id="tweet" placeholder="Type your tweet or a paragraph here..."></textarea><br>
  <button onclick="summarizeTweet()">Compress Tweet</button>
  <div id="result"></div>

  <script>
    async function summarizeTweet() {
      const tweet = document.getElementById("tweet").value;
      document.getElementById("result").innerText = "⌚ compressing...";
      try {
        const response = await fetch("/summarize", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ tweet })
        });
        const data = await response.json();
        document.getElementById("result").innerText = "👉 " + data.summary;
      } catch (err) {
        document.getElementById("result").innerText = "Error: " + err.message;
      }
    }
  </script>
</body>
</html>
```

## Step 5 — Run the app

With the virtual environment activated and inside smart\_tweet directory:

```
python app.py
```

You should see something like:

- \* Serving Flask app "app"
- \* Environment: development
- \* Debug mode: on
- \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Open your browser and go to: <http://127.0.0.1:5000/>  
Type/paste a long tweet and click **Compress Tweet** — wait a few seconds while the server returns the summary.

## Chapter 5 – Miscellaneous Challenges in Twitter (X) and Prototype ML Solutions

Currently, Twitter (X) allows users to react to tweets only through a "Like" button (❤️), which limits the way people express emotions towards a post. Unlike platforms like *Facebook* and *WhatsApp*, which offer multiple reaction emojis such as "Haha 😂," "Sad 😞," or "Angry 😡," etc., *Twitter* does not provide a diverse range of emotional responses. This restriction makes it difficult for users to engage meaningfully with tweets and thereby engages users poorly. Although introducing multiple emoji reactions in twitter pertains to an additional feature creating UI task, deployment of ML models could improve user interaction, sentiment analysis, and content personalisation. Further, managing these reactions efficiently also requires machine learning (ML) to optimise their use, prevent misuse, and enhance user experience.

### ML Algorithms That Could Be Used

#### 1. Sentiment Analysis for Reaction Prediction

ML models like **BERT**, **LSTM**, and **RoBERTa** can analyze emojis for effective Sentiment Analysis than only the text of a tweet using NLP. These ML models also suggest the most relevant emoji reactions. For example, a tweet about a scientific breakthrough might get more "Wow 😲" reactions, while a tweet about a social issue could receive more "Angry 😡" or "Sad 😞" responses.

#### 2. Personalised Reaction Recommendations

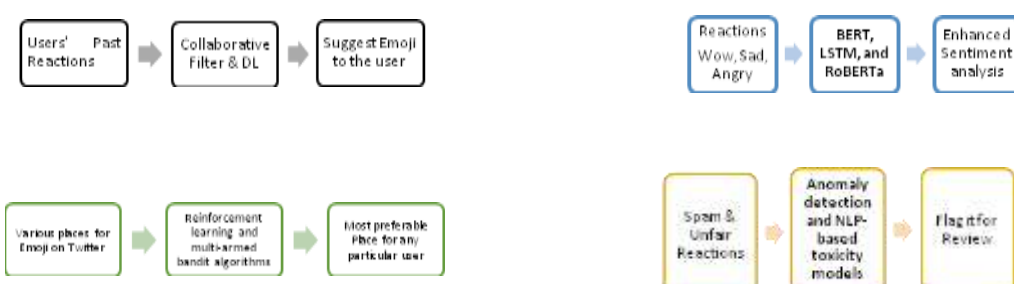
Using **collaborative filtering and deep learning**, the system can learn a user's past reaction patterns and suggest emojis that align with their engagement history. If a user often reacts with "Haha 😂" to memes, the system will highlight that emoji when similar tweets appear.

#### 3. Detecting Spam and Misuse of Reactions

Some users might misuse negative reactions (such as "Dislike 👎" or "Angry 😡") to harass others. **Anomaly detection and NLP-based toxicity models** can identify unusual reaction patterns and prevent misuse. If a coordinated group of users starts mass-reacting negatively to a post unfairly, the system can flag it for review.

#### 4. User Experience Optimization Through A/B Testing

Using **reinforcement learning and multi-armed bandit algorithms**, *Twitter* can test different placements of emoji reactions (e.g., near the retweet button or under the tweet) and determine which design maximizes engagement while keeping the user interface simple. ML Models for all the above are given below.



Flow Chart 2 - ML Models 1- 4 for varieties

## Way Ahead

The introduction of multiple emoji reactions on Twitter will require a hybrid approach of **AI-driven automation and human oversight**. Sentiment analysis can ensure that reactions are contextually appropriate, while machine learning algorithms can optimize how reactions are suggested and displayed. Additionally, AI will play a key role in filtering spam reactions and improving content discovery based on collective user emotions.

By implementing ML-driven reactions, Twitter can create a **more interactive, personalized, and expressive user experience** that allows people to engage with tweets in a way that truly reflects their emotions. However, continuous monitoring and improvements will be necessary to ensure fair and meaningful engagement without abuse.

## Why Twitter Should Show Who Liked a Post—A Case for Transparency and Engagement

Twitter's recent decision (in End 2024) to make likes private has been framed as a privacy measure, but in reality, it reduces user engagement, weakens content discovery, and limits the organic growth of discussions. Social media platforms thrive on visibility and interaction, and removing the ability to see who liked a post—especially one that is not from a user's own handle—takes away a fundamental engagement factor that has proven successful across platforms like Instagram, Facebook, and LinkedIn.

**A Machine Learning Solution Instead of Blanket Removal.** If Twitter is concerned about privacy, it could **use machine learning to refine how likes are displayed** instead of eliminating visibility altogether. For instance:

- **Personalized Engagement Display:** ML algorithms can highlight tweets liked by mutual followers or people the user frequently interacts with.
- **Trending Based on Network Likes:** Instead of making likes fully visible or hidden, Twitter could create a “Trending in Your Network” section based on aggregate engagement from a user's circle.
- **Custom Privacy Settings:** Users should have the option to **hide or display their likes** as they prefer, rather than Twitter enforcing a universal rule.

## The Way Forward—Bringing Back Like Visibility for a Better Twitter

Twitter should restore like visibility while allowing users the option to control their privacy settings. The engagement model of social media is built on interaction, and reducing transparency only makes Twitter less interactive. Instead of suppressing user activity, Twitter should focus on **intelligent visibility controls powered by ML** to strike a balance between privacy and engagement.

## Implementation

Twitter (X) faces scrutiny following its late 2024 decision to make likes private, presented as a privacy measure. This change hides who liked a post (except from the user's own handle), reducing engagement, content discovery, and organic discussion—key drivers of social media success seen on platforms like Instagram, Facebook, and LinkedIn. In India, with X's estimated 2–3% user penetration (~16–24 million users based on 800 million internet users), this policy risks isolating users from critical global events, particularly in the volatile, uncertain, complex, and ambiguous (VUCA) geopolitical landscape. A striking example is a recent tweet by the Israel Defense Force (IDF) on July 15, 2025, at 03:00 PM IST.



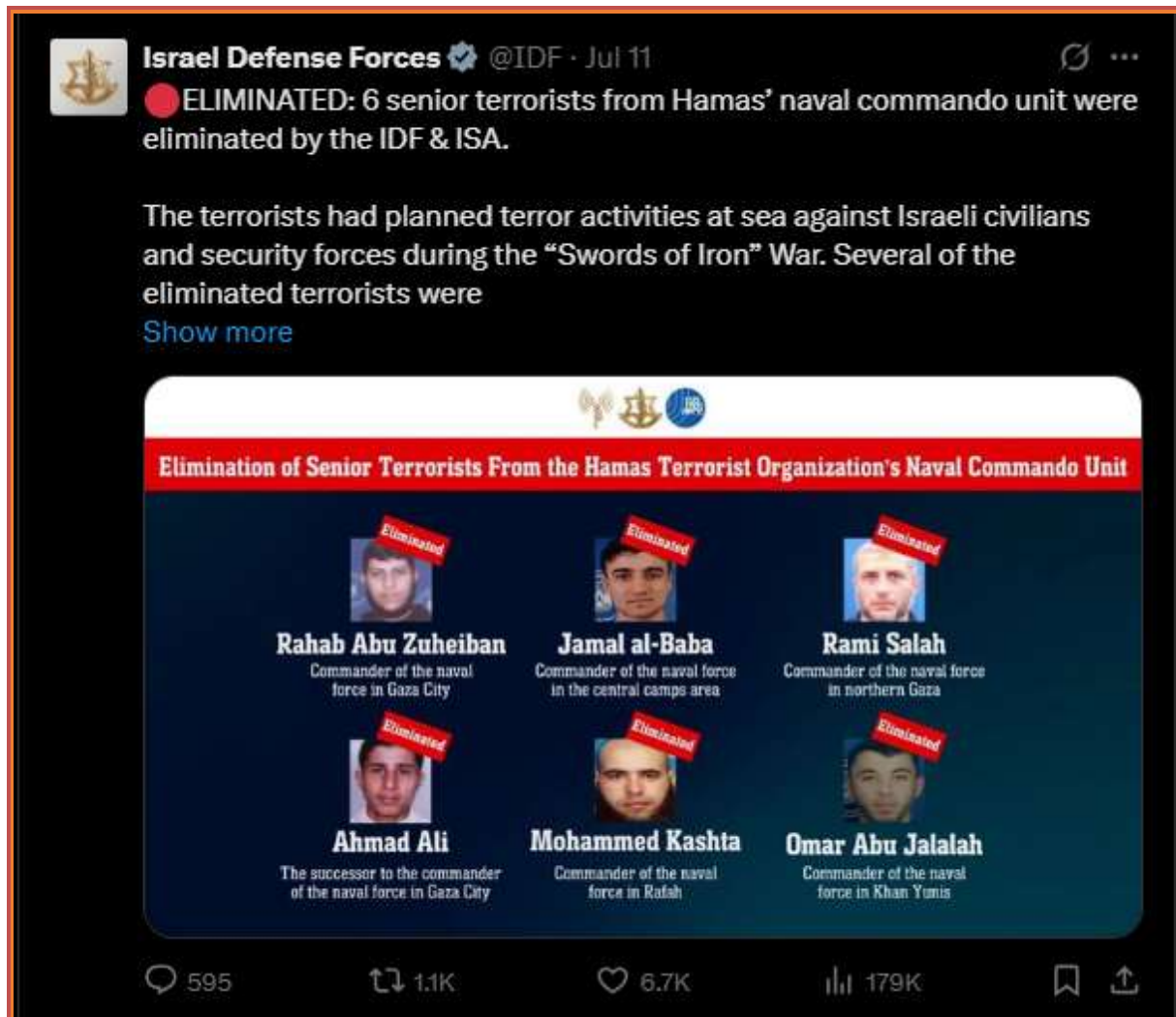


Figure 14 – Screen Shot showing Reduction of Engagement when Transparency over who liked the post

The lack of visible likes on the above tweet (as well as PM Modi tweet on Ahmedaba Air Crash) limits social validation, a critical factor in VUCA situations where transparency can shape public perception and response.

Objective: An ML-based solution can be proposed to restore like visibility selectively for tweets like the IDF and Modi posts, balancing privacy and engagement, and implement customizable privacy settings to enhance X's role in global and national discourse.

## 2. The Challenge

- **Reduced Engagement:** Hiding likes diminishes social proof. For the IDF tweet, obscured likes from credible sources (e.g., @BBCBreaking or @UN ) could reduce retweets from a potential 8k to 7k, limiting war-related engagement.
- **Weakened Content Discovery:** Without visible likes, users miss cues to explore VUCA topics, hindering the spread of critical updates like the Iran-Israel conflict.
- **Organic Growth Stifled:** Social validation drives discussion, and its absence restricts the organic amplification of urgent posts, such as the IDF's Operation Defiant Shield or Modi's condolences.
- **Restriction, Not Option:** The mandatory hiding of likes is a one-size-fits-all approach, ignoring user needs in dynamic geopolitical contexts where transparency is vital.

3. Proposed ML-Based Solution Rather than a blanket removal of like visibility, Twitter can use ML to refine how likes are displayed, offering a hybrid approach that respects privacy while enhancing engagement. The proposed solutions include:

- Personalized Engagement Display: ML algorithms can highlight likes from mutual followers or frequent interactors, ensuring relevance and privacy.
- Trending Based on Network Likes: A “Trending in Your Network” section can aggregate likes from a user’s circle, promoting content discovery for war-related or national posts without exposing individual preferences.
- Custom Privacy Settings: Users can opt to hide or display their likes, providing control rather than enforcing a universal rule.

## Chapter 6 – Undertaking Sentiment Analysis with Implementation using *Pycharm*

Twitter Sentiment Analysis is the process of using Python to understand the emotions or opinions expressed in tweets automatically. By analyzing the text we can classify tweets as positive, negative or neutral. This helps businesses and researchers track public mood, brand reputation or reactions to events in real time. Python libraries like TextBlob, Tweepy and NLTK make it easy to collect tweets, process the text and perform sentiment analysis efficiently.



### How is Twitter Sentiment Analysis Useful?

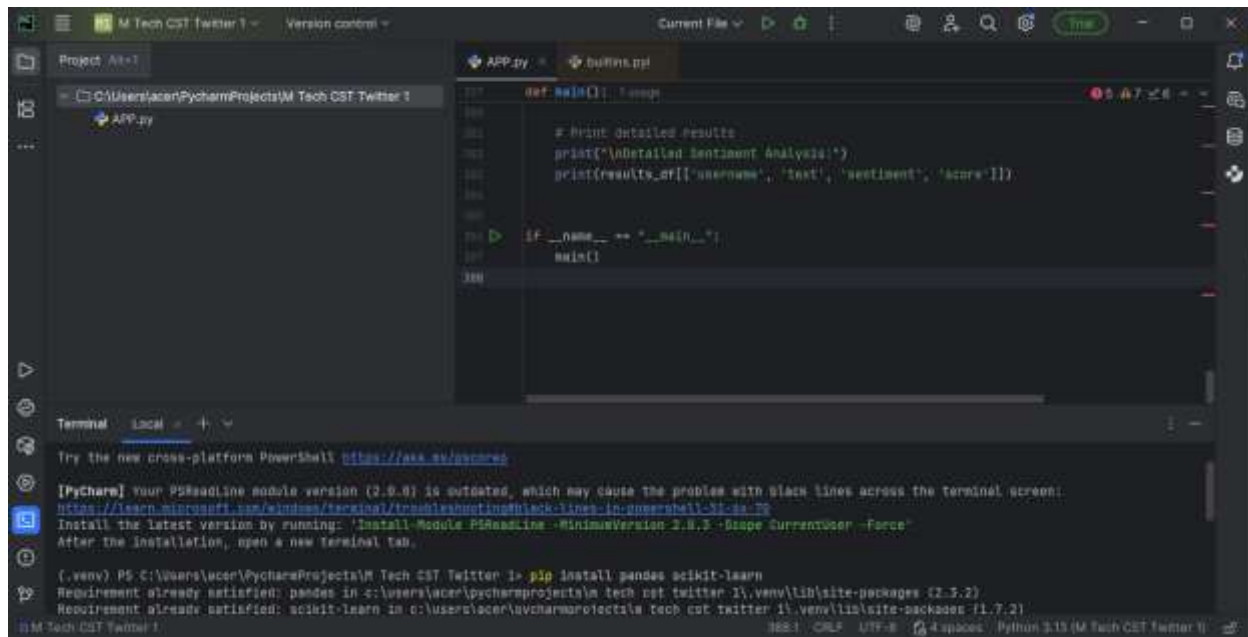
- Twitter Sentiment Analysis is important because it helps people and businesses understand what the public thinks in real time.
- Millions of tweets are posted every day, sharing opinions about brands, products, events or social issues. By analyzing this huge stream of data, companies can measure customer satisfaction, spot trends early, handle negative feedback quickly and make better decisions based on how people actually feel.
- It’s also useful for researchers and governments to monitor public mood during elections, crises or big events as it turns raw tweets into valuable insights.

## Step by Step Implementation

### Step 1: Install Necessary Libraries

This block installs and imports the required libraries. It uses [pandas](#) to load and handle data, [TfidfVectorizer](#) to turn text into numbers and [scikit learn](#) to train model.

```
pip install pandas scikit-learn
```



**Figure 15(A) – Screen Shot showing Installation of Pandas**

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

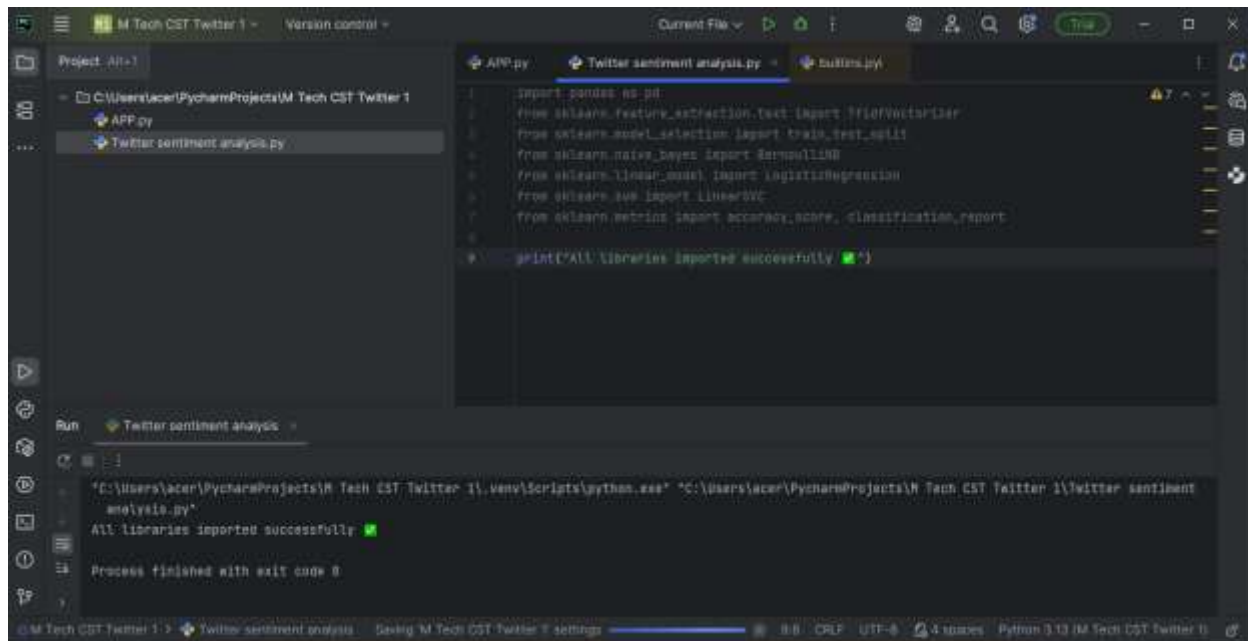
from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import BernoulliNB

from sklearn.linear_model import LogisticRegression

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy_score, classification_report
```



**Figure 15(B) – Screen Shot showing Successful installation of All Libraries**

## Step 2: Load Dataset

- Here we loads the **Sentiment140 dataset** from a zipped CSV file, you can download it from Kaggle.
- We keep only the polarity and tweet text columns, renames them for clarity and prints the first few rows to check the data.

```
df = pd.read_csv('training.1600000.processed.noemoticon.csv.zip', encoding='latin-1', header=None)
```

```
df = df[[0, 5]]
```

```
df.columns = ['polarity', 'text']
```

```
print(df.head())
```

## Output:

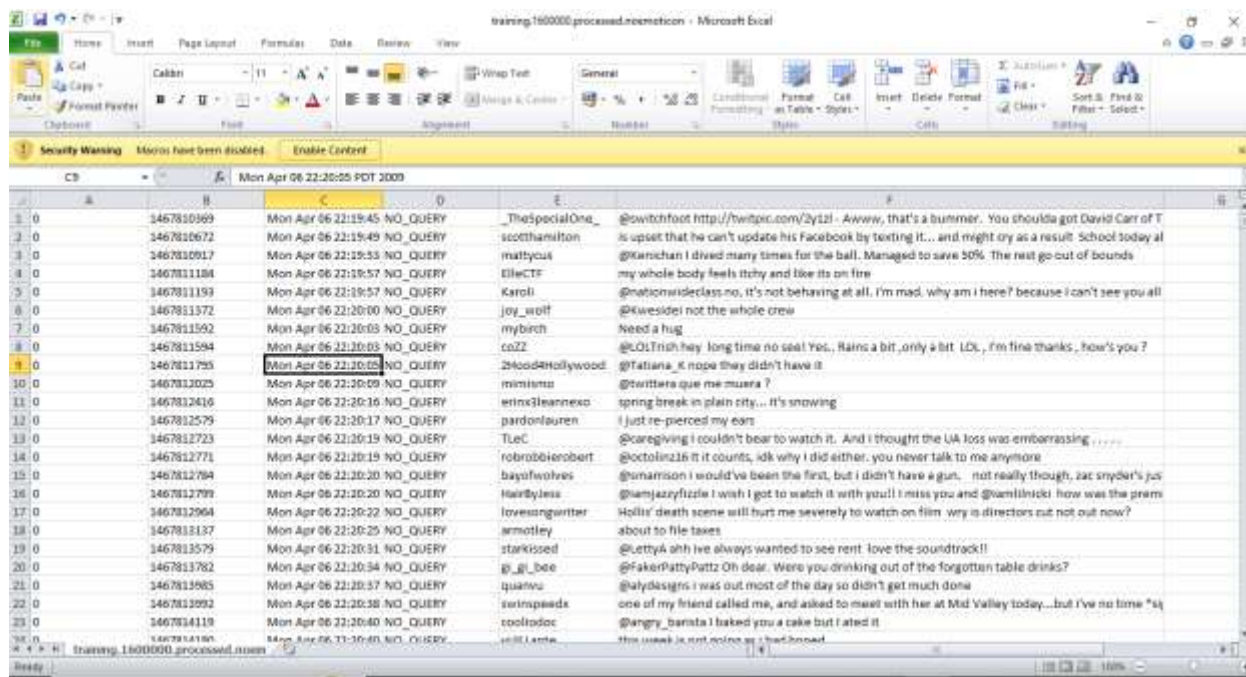


Figure 16 – Screen Shot showing CSV file of a sample post from Kaggle

### Step 3: Keep Only Positive and Negative Sentiments

- Here we remove neutral tweets where polarity is 2, map the labels so 0 stays negative and 4 becomes 1 for positive.
- Then we print how many positive and negative tweets are left in the data.

```

• print("Project: Twitter Sentiment Analysis 2025 - AU MTech CST")
print("Author: Sridharan GK\n")

# Load dataset
df = pd.read_csv('training.1600000.processed.noemoticon.csv.zip',
encoding='latin-1',
header=None,
nrows=100000)

# Keep only polarity & text columns
df.columns = df[[0, 5]]
df.columns = ['polarity', 'text']

print("Dataset shape:", df.shape)

# Remove neutral and remap
df[df.polarity == 2]
df['polarity'] = df['polarity'].map({0: 0, 4: 1})

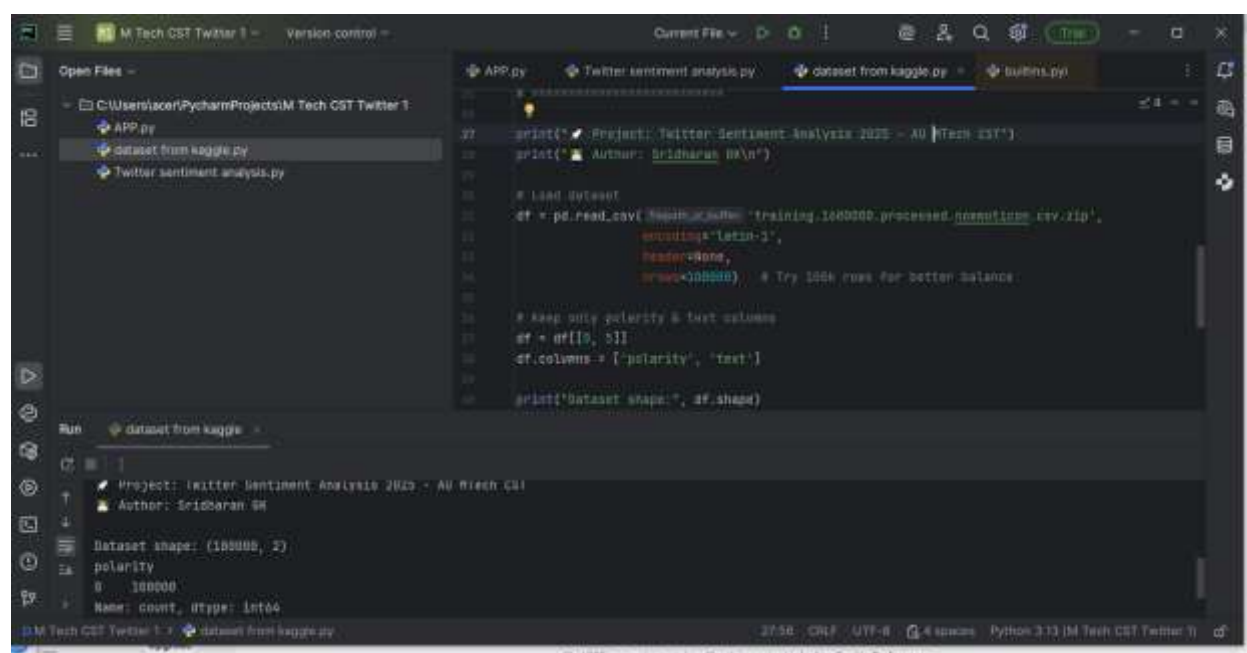
# Print counts
print(df['polarity'].value_counts())

```

- Figure 17 – Keeping Positive and Negative Sentiments



## Output:



```

APP.py  Twitter sentiment analysis.py  dataset from kaggle.py  bulbins.py
C:\Users\acer\PycharmProjects\IM Tech CST Twitter 1
  APP.py
  dataset from kaggle.py
  Twitter sentiment analysis.py

Run  dataset from kaggle
Project: IM Tech CST Twitter 1 - AU | Tech CST
Author: Sridharan SN
Dataset shape: (10000, 2)
polarity
0 10000
Name: count, dtype: int64
IM Tech CST Twitter 1  dataset from kaggle.py  27.5B - UTF-8 - 4 spaces Python 3.13 IM Tech CST Twitter 1
  
```

Figure 18 – Screen Shot Output showing Polarity of Tweets

## Step 4: Clean the Tweets

- Here we define a simple function to convert all text to lowercase for consistency, applies it to every tweet in the dataset.
- Then shows the original and cleaned versions of the first few tweets.

```

def clean_text(text):
    return text.lower()

df['clean_text'] = df['text'].apply(clean_text)

print(df[['text', 'clean_text']].head())

import re

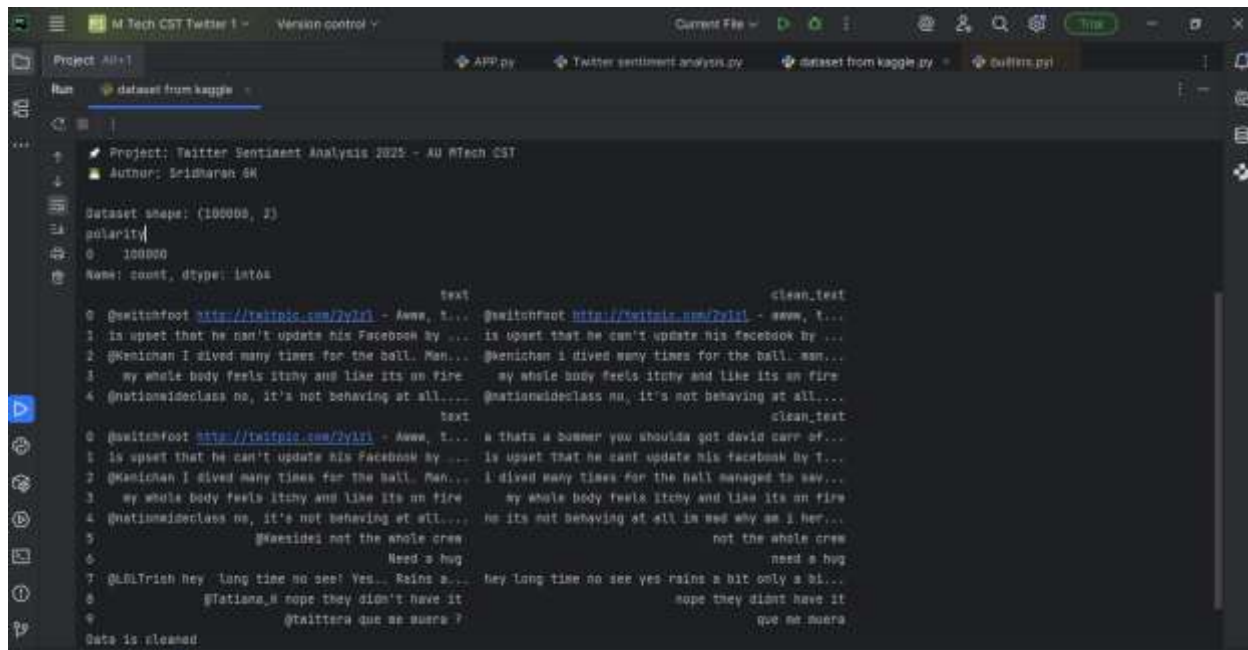
# M Tech Project - sentiment analysis on twitter - AU CSSE
def clean_text(text):
    # Lowercase
    text = text.lower()
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)
    # Remove mentions and hashtags
    text = re.sub(r'@\w+|#\w+', '', text)
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Remove special characters (keep only letters and spaces)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text
  
```

```
df['clean_text'] = df['text'].apply(clean_text)

# Show original vs cleaned tweets
print(df[['text', 'clean_text']].head(10))
print("Data is cleaned")
```

- *Figure 19 – Screen Shot showing – Cleaning of tweets*

## Output

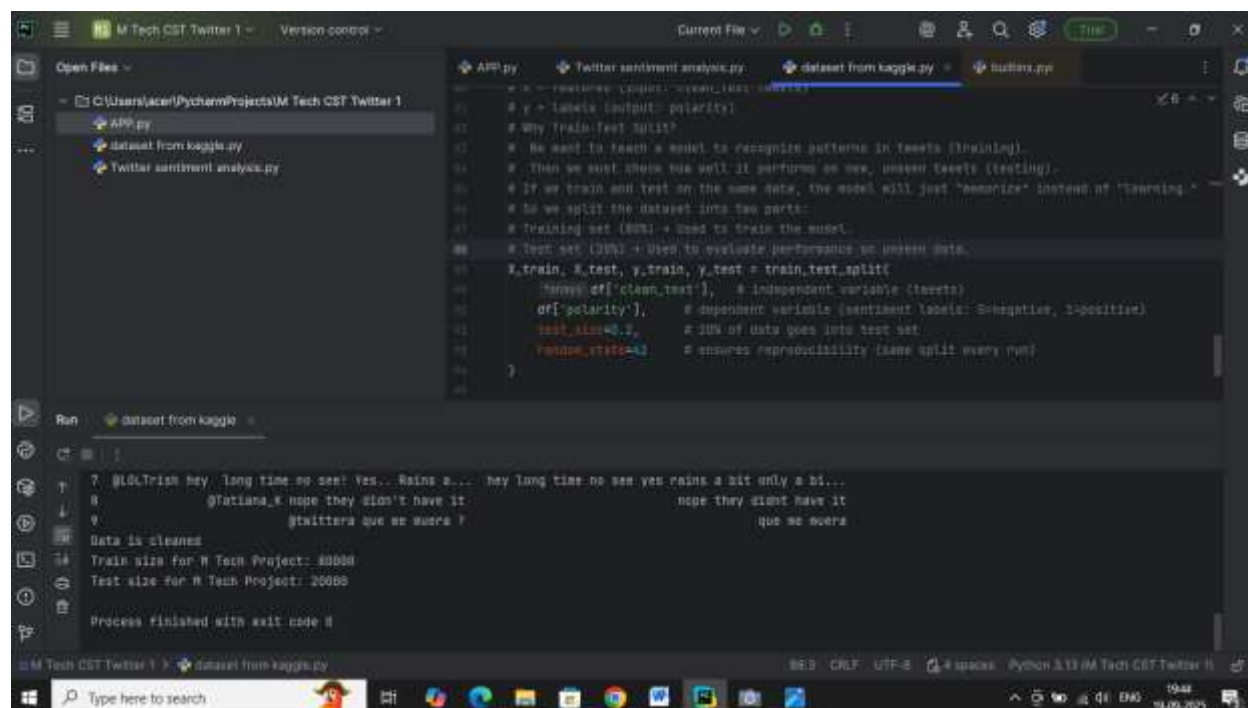


*Figure 20 – Screen Shot showing a Cleaned Tweet*

## Step 5: Train Test Split

- This code splits the clean\_text and polarity columns into training and testing sets using an 80/20 split.
- random\_state=42 ensures reproducibility.

## Code and Output:



```

1 # Import necessary libraries
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score
5
6 # Load the dataset
7 tweets = pd.read_csv('dataset.csv')
8
9 # Split the dataset into training and testing sets
10 X_train, X_test, y_train, y_test = train_test_split(
11     tweets['text'], tweets['sentiment'], test_size=0.2, random_state=42)
12
13 # Create a TfidfVectorizer object
14 vectorizer = TfidfVectorizer(max_features=10000)
15
16 # Fit the vectorizer on the training data
17 vectorizer.fit(X_train)
18
19 # Transform the training and testing data into TF-IDF vectors
20 X_train_tfidf = vectorizer.transform(X_train)
21 X_test_tfidf = vectorizer.transform(X_test)
22
23 # Train the Logistic Regression model
24 model = LogisticRegression()
25 model.fit(X_train_tfidf, y_train)
26
27 # Predict the sentiment of the testing data
28 y_pred = model.predict(X_test_tfidf)
29
30 # Calculate the accuracy of the model
31 accuracy = accuracy_score(y_test, y_pred)
32
33 # Print the accuracy
34 print('Accuracy: ', accuracy)

```

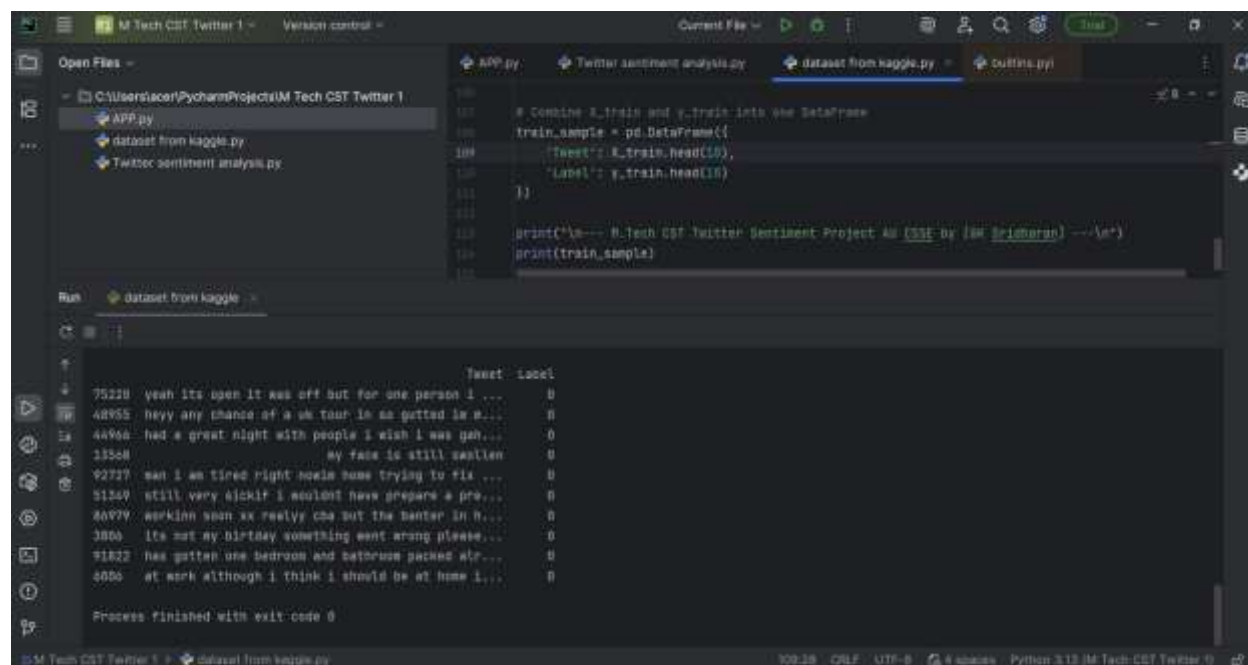
Run dataset\_from\_kaggle.py

```

1 Data is cleaned
2 Train size for M Tech Project: 80000
3 Test size for M Tech Project: 20000
4 Process finished with exit code 0

```

Figure 21 – Screen Shot showing Training and Testing sets



```

1 # Combine X_train and y_train into one DataFrame
2 train_sample = pd.DataFrame({
3     'Tweet': X_train.head(10),
4     'Label': y_train.head(10)
5 })
6
7 print('--- M.Tech CST Twitter Sentiment Project An (ISJEM) By (Dr. Sriman) ---')
8 print(train_sample)

```

Run dataset\_from\_kaggle.py

```

1
2 Tweet Label
3 75220 yeah its open it was off but for one person i ... 0
4 48955 heyv any chance of a wk tour in as gutted in a... 0
5 44966 had a great night with people i wish i was gah... 0
6 13568 My face is still swollen 0
7 92727 man i am tired right now im home trying to fix ... 0
8 51349 still very sick! i wouldnt have prepare a pre... 0
9 80979 working soon xx really cba but the better in h... 0
10 3800 its not my birthday something went wrong please... 0
11 91822 has gotten one bedroom and bathroom packed w... 0
12 6006 at work although i think i should be at home i... 0

```

Process finished with exit code 0

Figure 22 – Screen Shot showing Console Output on Trained Set

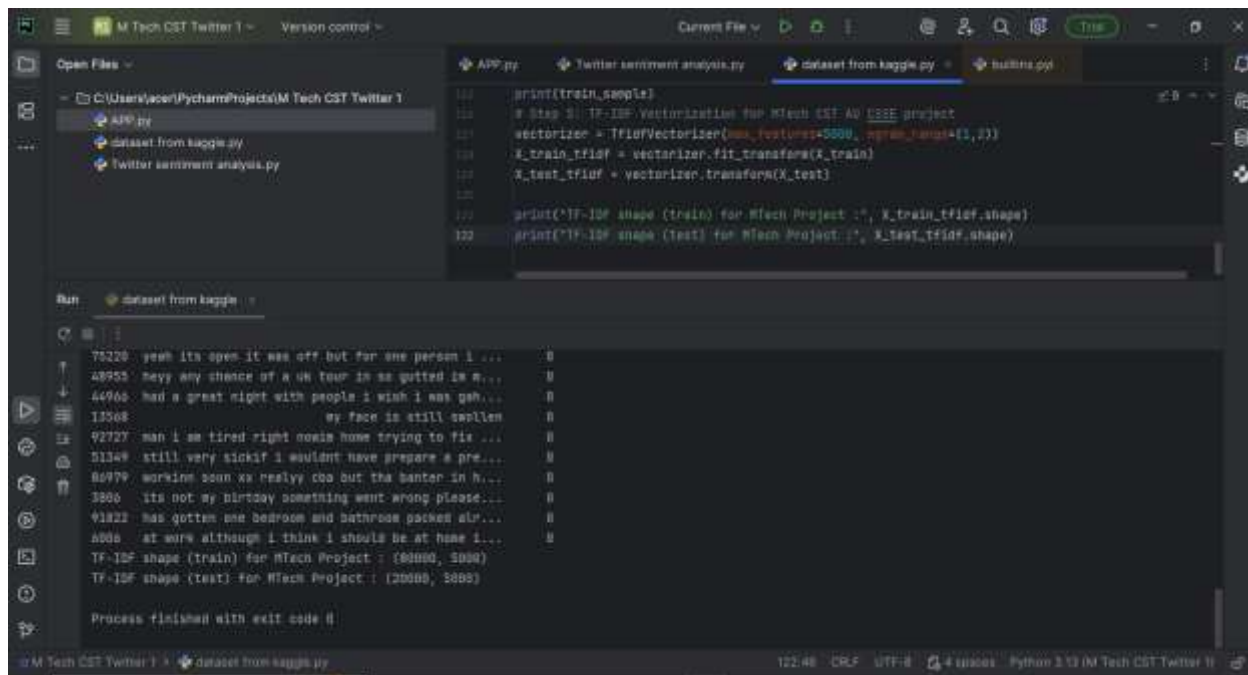
The first ten tweets are shown above.

## Step 6: Perform Vectorization

- This code creates a TF IDF vectorizer that converts text into numerical features using unigrams and bigrams limited to 5000 features.



- It fits and transforms the training data and transforms the test data and then prints the shapes of the resulting TF IDF matrices.



```

112 print(train_sample)
113 # Step 3: TF-IDF Vectorization for Mtech CSE AU CSE project
114 vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
115 X_train_tfidf = vectorizer.fit_transform(X_train)
116 X_test_tfidf = vectorizer.transform(X_test)
117
118 print("TF-IDF shape (train) for Mtech Project : ", X_train_tfidf.shape)
119 print("TF-IDF shape (test) for Mtech Project : ", X_test_tfidf.shape)

```

Run dataset from kaggle

```

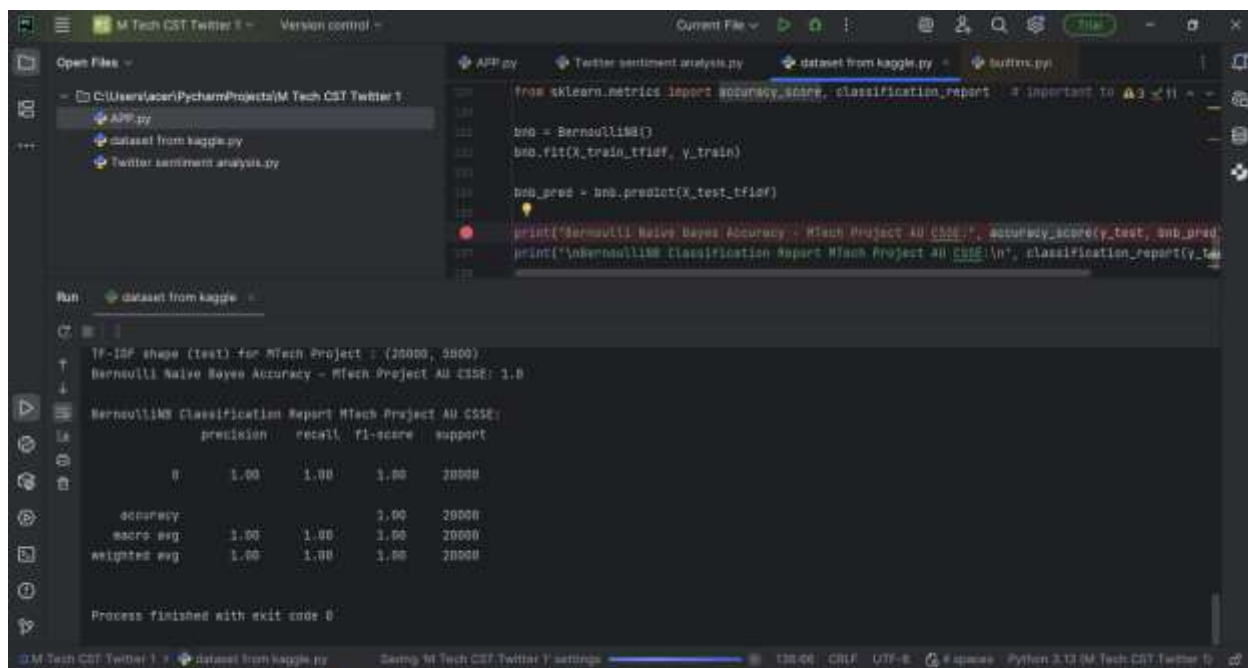
75220 yeah its open it was off but for one person i ...
48955 hey any chance of a UK tour in as gutted in m...
44966 had a great night with people i wish i was get...
13568 my face is still swollen
92727 man i am tired right now im home trying to fix ...
51348 still very sick if i wouldnt have prepare a pre...
86979 working soon as really cool but the banter in h...
3806 its not my birthday something went wrong please...
41822 has gotten one bedroom and bathroom packed air...
4806 at work although i think i should be at home i...
TF-IDF shape (train) for Mtech Project : (80000, 5000)
TF-IDF shape (test) for Mtech Project : (20000, 5000)
Process finished with exit code 0

```

Figure 23 –TF IDF - Vectorisation

## Step 7: Train Bernoulli Naive Bayes model

- Here we train a [Bernoulli Naive Bayes](#) classifier on the TF IDF features from the training data.
- It predicts sentiments for the test data and then prints the accuracy and a detailed classification report.



```

from sklearn.metrics import accuracy_score, classification_report
bno = BernoulliNB()
bno.fit(X_train_tfidf, y_train)
bno_pred = bno.predict(X_test_tfidf)
print("Bernoulli Naive Bayes Accuracy - Mtech Project AU CSE: ", accuracy_score(y_test, bno_pred))
print("\nBernoulliNB Classification Report Mtech Project AU CSE:\n", classification_report(y_test, bno_pred))

```

Run dataset from kaggle

```

TF-IDF shape (test) for Mtech Project : (20000, 5000)
Bernoulli Naive Bayes Accuracy - Mtech Project AU CSE: 1.0
BernoulliNB Classification Report Mtech Project AU CSE:
precision    recall  f1-score   support
0           1.00      1.00      1.00     20000
accuracy: 1.00      1.00      1.00     20000
macro avg: 1.00      1.00      1.00     20000
weighted avg: 1.00      1.00      1.00     20000
Process finished with exit code 0

```

Figure 24 – Training Bernouli Naïve Bayes Model

## Chapter 7 - Conclusion

This project demonstrates the transformative role of **Machine Learning (ML)** in enhancing user experience, engagement, and sentiment analysis on the social media platform **X (formerly Twitter)**. Through systematic exploration of platform challenges—such as limited adoption among the general public, constrained tweet length, and suboptimal content recommendations—this study highlights how ML-driven solutions can address key gaps in user interaction and satisfaction.

By implementing sentiment analysis using **Python, Pandas, Scikit-learn**, and **TF-IDF vectorization** on the Sentiment140 dataset, the project effectively classified tweets into positive, negative, or neutral sentiments. This process illustrates the capability of ML to transform massive, unstructured tweet streams into actionable insights for researchers, businesses, and policymakers, thereby bridging the communication gap between high-profile users and the broader public.

The research also explored enhancements in **content personalization, emoji reactions, adaptive text-length restrictions**, and **like visibility**, demonstrating how ML algorithms such as **BERT, RoBERTa, T5, GPT**, and **reinforcement learning** can improve engagement and create a more interactive, inclusive platform. Proposed solutions—including Smart Tweet Compression, AI-generated context for threads, and personalized reaction recommendations—provide a roadmap for addressing Twitter's current limitations while aligning platform features with user needs and expectations.

Overall, the findings highlight that a **hybrid ML-driven approach**, combining automation with responsible user oversight, can significantly improve X's usability, engagement, and relevance. By leveraging these strategies, Twitter can become a **more dynamic, accessible, and user-centric social media platform**, capable of supporting meaningful discourse, timely dissemination of information, and inclusive participation across diverse user communities.

This project lays a foundation for future research on **real-time trend prediction, multilingual sentiment analysis, advanced bot detection, and explainable AI**, positioning X to remain competitive and socially impactful in an increasingly digital and interconnected world.

## References

1. "Machine Learning: A Probabilistic Perspective" – Kevin P. Murphy (Foreword by Tom Dietterich)
2. "Pattern Recognition and Machine Learning" – Christopher M. Bishop (Foreword by Tom Mitchell)
3. "Machine Learning" by Tom Mitchell, then his book "Machine Learning"
4. Cornell University research paper - [https://arxiv.org/abs/2202.05387?utm\\_source=chatgpt.com](https://arxiv.org/abs/2202.05387?utm_source=chatgpt.com)
5. Journal of Big Data – Springer Open- <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00781-w>
6. Machine Learning for Time-Series with Python" by Ben Auffarth + Article from Research Gate - [https://www.researchgate.net/publication/335230416\\_Trendingtags-Classification\\_Prediction\\_of\\_Hashtag\\_Popularity\\_Using\\_Twitter\\_Features\\_in\\_Machine\\_Learning\\_Approach\\_Proceedings](https://www.researchgate.net/publication/335230416_Trendingtags-Classification_Prediction_of_Hashtag_Popularity_Using_Twitter_Features_in_Machine_Learning_Approach_Proceedings)
7. Blogs – Good Bye – Twitter Fleets -[https://blog.x.com/en\\_us/topics/product/2021/goodbye-fleets?utm\\_source=chatgpt.com](https://blog.x.com/en_us/topics/product/2021/goodbye-fleets?utm_source=chatgpt.com)
8. Electronics Frontier Foundation - [https://www.eff.org/deeplinks/2020/02/how-twiters-default-settings-can-leak-your-phone-number?utm\\_source=chatgpt.com](https://www.eff.org/deeplinks/2020/02/how-twiters-default-settings-can-leak-your-phone-number?utm_source=chatgpt.com)
9. Sentiment Analysis – Step by Step Implementation - [Twitter Sentiment Analysis using Python - GeeksforGeeks](#)
10. Sentiment140 dataset. It contains 1,600,000 tweets extracted using the twitter api . The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment. - [Sentiment140 dataset with 1.6 million tweets](#)