PES UNIVERSITY
100 feet Ring Road, BSK 3rd Stage
Bengaluru 560085
Department of Computer Science and Engineering

Department of Computer Science and Engineering
B. Tech. CSE - 6th Semester
Jan – May 2025

UE22CS343BB3
DATABASE TECHNOLOGIES (DBT)

PROJECT REPORT
on

Twitter IPL Data Analysis

Submitted by : Team #: 3

Lohit Kumar Nagarur	6 E	PES2UG22CS280
M C Krishna Kumar	6 E	PES2UG22CS281
Nikhil Srivasta	6F	PES2UG22CS357

Class of Prof. Saranya Rubini

Table of Contents

Introduction

In recent years, social media platforms have emerged as dynamic sources of real-time public opinion and sentiment. Among them, Twitter stands out for its fast-paced updates, especially during major live events like sports tournaments. This project focuses on harnessing Twitter data related to the Indian Premier League (IPL), one of the most popular and widely followed cricket leagues globally, to perform both **batch and streaming data analysis**.

The primary goal of this project is to collect, process, and analyze IPL-related tweets in real time and in batches to extract valuable insights such as user engagement, sentiment, and team-specific trends. To achieve this, we employ a modern big data pipeline using **Apache Kafka**, **PySpark**, and **PostgreSQL**:

- **Kafka** serves as the backbone for ingesting real-time tweet streams.
- **PySpark** is used for processing both batch datasets (e.g., archived tweets) and real-time tweet streams.
- **PostgreSQL** acts as the storage layer, where structured and filtered tweet data is stored for further querying and visualization.

Additionally, the system includes logic to extract user information, tweet text, and specific hashtags such as team names (e.g., #RCB, #CSK), and filters out verified users or team-related mentions. The cleaned and structured data is then stored in a PostgreSQL database for downstream analysis and visualization.

This project not only demonstrates the application of distributed data processing tools in a real-world social media context but also provides a framework for extending the pipeline to include sentiment analysis, trend forecasting, and fan behavior analysis during large-scale events.

Installation of Software

1. Setting up Zookeeper and Kafka

Use the images from the Github or Docker Hub directly for zookeeper and kafka and make a docker-compose.yaml file

run

```
docker-compose up
```

to test the container run

```
docker exec -it tweeeeedbt-kafka-1 bash
```

2. Setting up Spark

Requirements `java 17 python 3.8.10`
We will be setting up Spark locally

To Download and Install run the following commands

```
wget
https://downloads.apache.org/spark/spark-3.5.5/spark-3.5.5-bin-hadoop3.
tgz
```

```
tar -xvzf spark-3.5.5-bin-hadoop3.tgz
mv spark-3.5.5-bin-hadoop3 ~/spark
```

```
vim ~/.bashrc # or zshrc
```

```
export SPARK_HOME=~/spark
export PATH=$SPARK_HOME/bin:$PATH
```

3. Setting up Database

```
sudo apt install postgresql postgresql-contrib
```

```
sudo systemctl status postgresql
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

```
sudo -i -u postgres
psql
```

This will install postgres on your system, now create the required database,

```
-- Create a database
CREATE DATABASE tweedbt;

-- Create a user with a password
CREATE USER <username> WITH PASSWORD '<password>'; -- these values
should go into your .env

-- Grant privileges
GRANT ALL PRIVILEGES ON DATABASE mydb TO <username>;
```

4. Installing required python libraries

```
pip install kafka-python pyspark pandas psycopg2-binary psycopg2
```

Input Data

a) Source

Data is taken from kaggle
https://www.kaggle.com/datasets/kaushiksuresh147/ipl2020-tweets?select=IPL_2022_tweets.csv

b) Description

The data totally consists of 13 columns. The description of the features is given below

Columns	Descriptions
user_name	The name of the user, as they've defined it.
user_location	The user-defined location for this account's profile.
user_description	The user-defined UTF-8 string describing their account.
user_created	Time and date, when the account was created.
user_followers	The number of followers an account currently has.
user_friends	The number of friends an account currently has.
user_favourites	The number of favorites an account currently has
user_verified	When true, indicates that the user has a verified account
dates	UTC time and date when the Tweet was created
text	The actual UTF-8 text of the Tweet
hashtags	All the other hashtags posted in the tweet along with #ipl2020
source	Utility used to post the Tweet, Tweets from the Twitter website have a source value - web
is_retweet	Indicates whether this Tweet has been Retweeted by the authenticating user.

Stream Mode Experiment

a) Description

Overview

This streaming pipeline processes live IPL tweets ingested via Apache Kafka using PySpark's Structured Streaming API. The purpose is to classify, filter, and aggregate incoming tweet data based on specific attributes such as team mentions, user verification status, and user location. Each filtered stream is routed to separate Kafka topics for further use in analytics, monitoring, and storage.

Streaming Workflow

1. **Spark Session Initialization**
    - A Spark session is started with the app name *"IPL Kafka Stream Processor"*.
    - Log level is set to WARN to reduce console noise.
2. **Kafka Integration**
    - Spark consumes live tweet data from the Kafka topic ipl_raw.
    - Kafka messages (JSON strings) are cast and parsed using a predefined schema.
3. **Schema Definition**
    - Schema includes 13 fields such as user_name, user_location, hashtags, text, user_verified, and date.
    - Data types are explicitly defined to support downstream transformations.
4. **Parsing and Transformation**
    - Incoming JSON strings are parsed into a structured format.
    - The date field is converted to a timestamp column for time-based operations.

5. **Filtering and Output Routing**
    - **Verified Users**: Filters users with "user_verified" = true and sends to VerifiedUserCheck Kafka topic.
    - **Geo-tagged Tweets**: Filters tweets where user_location contains "Delhi" or "Mumbai"; sent to GeoLocation.
    - **Team-Specific Mentions**: Extracts tweets mentioning teams like "RCB" or "CSK"; routed to TeamSpecific.
    - Each stream is written to Kafka using the write_to_kafka function.
6. **Windowed Aggregation**
    - For verified users, tweets are grouped into 10-second time windows.
    - The aggregation computes count and list of users per window.
    - Results are pushed to the VerifiedUserWindowedCount topic.

b) Windows

The **windowing** operation is applied to track verified users' tweet activity over fixed time intervals. Using PySpark's withWatermark and window functions, the stream of verified tweets is grouped into **10-second tumbling windows** based on the event_time derived from the tweet's timestamp. Watermarking with a **1-minute threshold** ensures late-arriving data within that time frame is still included, while discarding older data to manage state efficiently. Within each window, two key aggregations are performed: the **count of tweets** and the **collection of usernames** who tweeted during that interval. This produces a structured summary of verified user engagement over time, which is then serialized as JSON and sent to the Kafka topic VerifiedUserWindowedCount. This approach enables real-time temporal analytics and trend tracking across streaming IPL tweet data.

c) Results

Code :

stream_processing.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json, lower, trim,
to_timestamp, window, collect_list, date_format, count
from pyspark.sql.types import StructType, StringType

# Start Spark session
spark = SparkSession.builder \
      .appName("IPL Kafka Stream Processor") \
      .getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Define schema of your IPL tweet
schema = StructType() \
```

```
      .add("user_name", StringType()) \
      .add("user_location", StringType()) \
      .add("user_description", StringType()) \
      .add("user_created", StringType()) \
      .add("user_followers", StringType()) \
      .add("user_friends", StringType()) \
      .add("user_favourites", StringType()) \
      .add("user_verified", StringType()) \
      .add("date", StringType()) \
      .add("text", StringType()) \
      .add("hashtags", StringType()) \
      .add("source", StringType()) \
      .add("is_retweet", StringType())

# Read from the raw Kafka topic
df = spark.readStream \
```

```
      .format("kafka") \
      .option("kafka.bootstrap.servers", "localhost:9092") \
      .option("subscribe", "ipl_raw") \
      .option("startingOffsets", "earliest") \
      .option("failOnDataLoss", "false") \
      .load()

# Convert Kafka value to string and parse JSON
parsed_df = df.selectExpr("CAST(value AS STRING) as json") \
      .select(from_json(col("json"), schema).alias("data")) \
      .select("data.*")

# Convert date string to timestamp
parsed_df = parsed_df.withColumn("timestamp", to_timestamp(col("date"),
"yyyy-MM-dd HH:mm:ss"))

# Verified users only
verified_df = parsed_df.filter(lower(trim(col("user_verified"))) ==
"true")

# Function to write any DF to Kafka
def write_to_kafka(df, topic):
      df.selectExpr("to_json(struct(*)) AS value") \
      .writeStream \
```

```
.format("kafka") \
      .option("kafka.bootstrap.servers", "localhost:9092") \
      .option("topic", topic) \
      .option("checkpointLocation", f"/tmp/checkpoint_{topic}") \
      .start()

# Function: Verified User Windowed Count to Kafka
def verified_user_windowed_count_to_kafka(df, topic):
      df_with_event_time = df.withColumn("event_time",
to_timestamp(col("date"), "yyyy-MM-dd HH:mm:ss"))
```

```
      aggregated_df = df_with_event_time \
      .withWatermark("event_time", "1 minute") \
      .groupBy(window(col("event_time"), "10 seconds")) \
      .agg(
            count("*").alias("count"),
            collect_list("user_name").alias("usernames")
      ) \
      .select(
            date_format(col("window.start"), "yyyy-MM-dd
HH:mm:ss").alias("start_time"),
            date_format(col("window.end"), "yyyy-MM-dd
HH:mm:ss").alias("end_time"),
            col("count"),
            col("usernames")
      ) \
      .orderBy("start_time")

      aggregated_df \
      .selectExpr("to_json(struct(*)) AS value") \
      .writeStream \
      .format("kafka") \
      .option("kafka.bootstrap.servers", "localhost:9092") \
      .option("topic", topic) \
      .option("checkpointLocation", f"/tmp/checkpoint_{topic}") \
      .outputMode("complete") \
```

```
      .start()

# 1. Send verified user tweets to Kafka
write_to_kafka(verified_df, "VerifiedUserCheck")

# 2. Geo-tagged tweets
location_df = parsed_df.filter(

lower(col("user_location")).rlike(r"\b(new\s*)?delhi\b|\bmumbai\b")
)
```

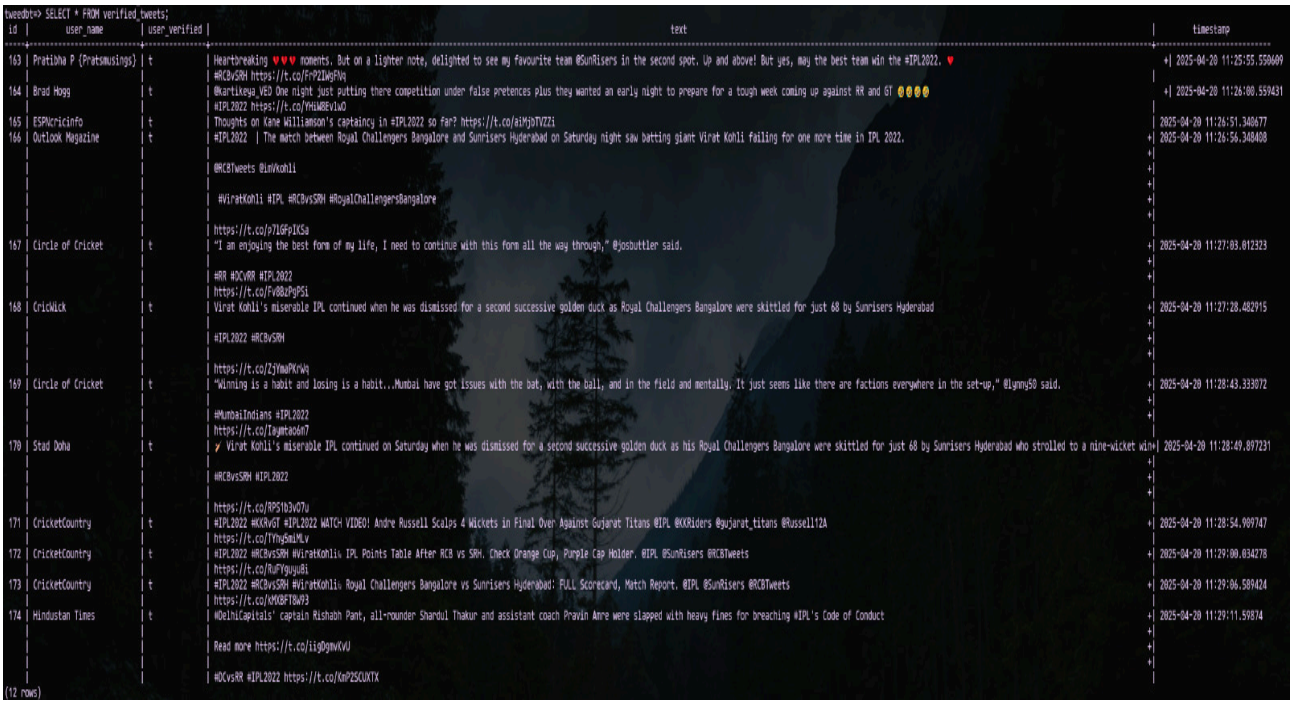
```
write_to_kafka(location_df, "GeoLocation")

# 3. Team-specific mentions (e.g., RCB, CSK)
team_df = parsed_df.filter(
      lower(col("hashtags")).rlike(r'\[.*?(rcb|csk).*?\]')
)
write_to_kafka(team_df, "TeamSpecific")

# 4. Windowed count of verified users to Kafka
verified_user_windowed_count_to_kafka(verified_df,
"VerifiedUserWindowedCount")

# Keep streaming job alive
spark.streams.awaitAnyTermination()
```

verified_users output

team_mentions output

geo_location output

data inserted into verified_tweets table



data inserted into team_mentions table

```
id |        user_name         |                                          text                                                        |    hashtags    |          timestamp
---+--------------------------+------------------------------------------------------------------------------------------------------+----------------+----------------------------
33 | Ranjan Kumar             | @ViCustomerCare Glenn Maxwell                                                                        +| [SRHvRCB]      | 2025-04-20 11:25:51.153388
   |                          |                                                                                                      +|                |
   |                          | #ViFanOfTheMatch                                                                                     +|                |
   |                          | #IPL2022 #SRHvRCB                                                                                    +|                |
   |                          | @ViCustomerCare                                                                                       |                |
34 | Ranjan Kumar             | @ViCustomerCare 27 Runs                                                                              +| [SRHvRCB]      | 2025-04-20 11:25:56.169982
   |                          |                                                                                                      +|                |
   |                          | #ViFanOfTheMatch #IPL2022 #SRHvRCB                                                                   +|                |
   |                          | @ViCustomerCare                                                                                       |                |
35 | Ranjan Kumar             | @ViCustomerCare A7) 2nd Over                                                                         +| [SRHvRCB]      | 2025-04-20 11:26:01.177508
   |                          |                                                                                                      +|                |
   |                          | #ViFanOfTheMatch #IPL2022 #SRHvRCB                                                                   +|                |
   |                          | @ViCustomerCare                                                                                       |                |
36 | Ranjan Kumar             | @ViCustomerCare 18 runs                                                                              +| [SRHvRCB]      | 2025-04-20 11:26:06.179536
   |                          | #ViFanOfTheMatch #IPL2022 #SRHvRCB                                                                   +|                |
   |                          | @ViCustomerCare                                                                                       |                |
37 | Pratibha P {Pratsmusings} | Heartbreaking ♥♥♥ moments. But on a lighter note, delighted to see my favourite team @SunRisers in the second spot. Up and above! But yes, may the best team win the #IPL2022. ♥+| [RCBvSRH]      | 2025-04-20 11:26:11.18544
   |                          | #RCBvSRH https://t.co/FrP2IWgFNq                                                                      |                |
38 | Sakshi Singh Rathour     | @LeadDigitalIn Sunrisers Hyderabad (SRH) ♥ will win                                                  +| [RCBvSRH]      | 2025-04-20 11:26:16.200297
   |                          |                                                                                                      +|                |
   |                          | #IPL2022 #TATAIPL #TATAIPL2022 #IPL #IPLT20 #RCBvSRH #IPLquiz #predictandwin                         +|                |
   |                          |                                                                                                      +|                |
   |                          | Tagging                                                                                              +|                |
   |                          | @Ankur_Raj_9168                                                                                      +|                |
   |                          | @mr_shubham31                                                                                        +|                |
   |                          | @HarshPandey860                                                                                      +|                |
   |                          | @shubhliv                                                                                            +|                |
   |                          | @Harshpr01180227                                                                                      |                |
39 | Parashu Uniyal           | #IPL                                                                                                 +| [RCBvsSRH]     | 2025-04-20 11:26:21.205154
   |                          | #IPL2022                                                                                             +|                |
   |                          | #RCBvsSRH                                                                                            +|                |
   |                          | IPL BATTERS WITH TWO BACK-TO-BACK GOLDEN DUCKS:                                                      +|                |
   |                          | 2022-Virat Kohli                                                                                     +|                |
   |                          | 2019; 2018-Ravichandran Ashwin                                                                       +|                |
   |                          | 2018-Aaron Finch                                                                                     +|                |
   |                          | 2008-Rahul Dravid                                                                                    +|                |
   |                          | _________________________________                                                                    +|                |
   |                          | *Nitish Rana scored 0, 81, 0, 87, 0 in 2020 [all ducks being Golden]                                  |                |
40 | Namita Sasmal            | @ViCustomerCare Q20A: 12 runs                                                                        +| [SRHvRCB]      | 2025-04-20 11:26:26.214717
   |                          | @ViCustomerCare                                                                                      +|                |
```

data inserted into geo_location table

```
 id |              user_name                |                                    text                                                                      | location |    timest
amp
----+---------------------------------------+--------------------------------------------------------------------------------------------------------------+----------+------------
-------------
  1 | Modern Mahapurush                     | And we expect #irfanpathan who was raised in a mosque by a mullah to have respect for Indian constitution or compassion for Hindus ? He is a just a hardcore jih@@di ! #IPL2022 #ChaloKarauli #jahagirpuri #HindusUnderAttackInIndia https://t.co/jsGVTJOok1 | Delhi    | 2025-04-20 11:
25:59.634832
  2 | M.abulkhair02                         | Or fact ye he ke playoffs bhi nhi chahta ke #SRH qualify kre                                                 +| Delhi    | 2025-04-20
11:26:08.579125
    |                                       | @tanay_chawda1                                                                                               +|          |
    |                                       | #IPL2022 https://t.co/A8t8aQL30u                                                                              |          |
  3 | TELEVISION REVIEWER WIZARD            | What do you think which #IPL teams will be qualified for playoffs?                                           +| Mumbai   | 2025-04-20 11
:26:13.589432
    |                                       | #IPL2022 #Cricket                                                                                             |          |
  4 | M.abulkhair02                         | Rishabh pant is the true leader                                                                              +| Delhi    | 2025-04-20 11:
26:18.600919
    |                                       |                                                                                                              +|          |
    |                                       | Rishabh Pant gets Rs 1.5 crore fine due to umpire's mistake                                                  +|          |
    |                                       | If you make a mistake, you have to empire and pay the fine to the player. Rishabh Pant gets Rs 1.5 crore fine due to umpire's mistake +|          |
    |                                       | If you make a mistake,                                                                                       +|          |
    |                                       | #IPL2022                                                                                                     +|          |
    |                                       | #ViratKohli                                                                                                   |          |
  5 | Pradhika Prat Panchal ( Poonam Panchal) | @ChaifullyYours Rajasthan Royals will win .                                                                +| Delhi    | 2025-04-20 11:
26:38.404654
    |                                       |                                                                                                              +|          |
    |                                       | #PBKSvGT #IPL2022                                                                                            +|          |
    |                                       | #enCHAIcricket                                                                                               +|          |
    |                                       | @ChaifullyYours                                                                                               |          |
  6 | Pradhika Prat Panchal ( Poonam Panchal) | @DelhiCapitals @LalitYadav03 Piece "A"                                                                     +| Delhi    | 2025-04-20 11:
26:43.41712
    |                                       |                                                                                                              +|          |
    |                                       | #FunWithDC #YehHaiNayiDilli #IPL2022 #TATAIPL #IPL #DelhiCapitals @DelhiCapitals                              |          |
  7 | Outlook Magazine                      | #IPL2022  | The match between Royal Challengers Bangalore and Sunrisers Hyderabad on Saturday night saw batting giant Virat Kohli failing for one more time in IPL 2022. +| Delhi    | 2025-04-20 11:
26:52.406413
    |                                       |                                                                                                              +|          |
    |                                       | @RCBTweets @imVkohli                                                                                         +|          |
    |                                       |                                                                                                              +|          |
    |                                       |  #ViratKohli #IPL #RCBvsSRH #RoyalChallengersBangalore                                                       +|          |
```

Batch Mode Experiment

a) Description

This experiment uses **Apache Spark in batch mode** to process tweets from the `IPL_2022_tweets.csv` dataset. The process performs filtering on the dataset to extract three specific categories of tweets:

1. **Verified User Tweets** – Tweets posted by users marked as verified.
2. **Geo-tagged Tweets** – Tweets from users located in Delhi or Mumbai.
3. **Team-specific Tweets** – Tweets containing hashtags related to RCB or CSK.

The filtered data is inserted into three corresponding PostgreSQL tables:

- `verified_tweets_batch`
- `geo_tweets_batch`
- `team_tweets_batch`

This is executed as a one-time batch job with the performance tracked via CPU, memory, and execution time.

b) Data size

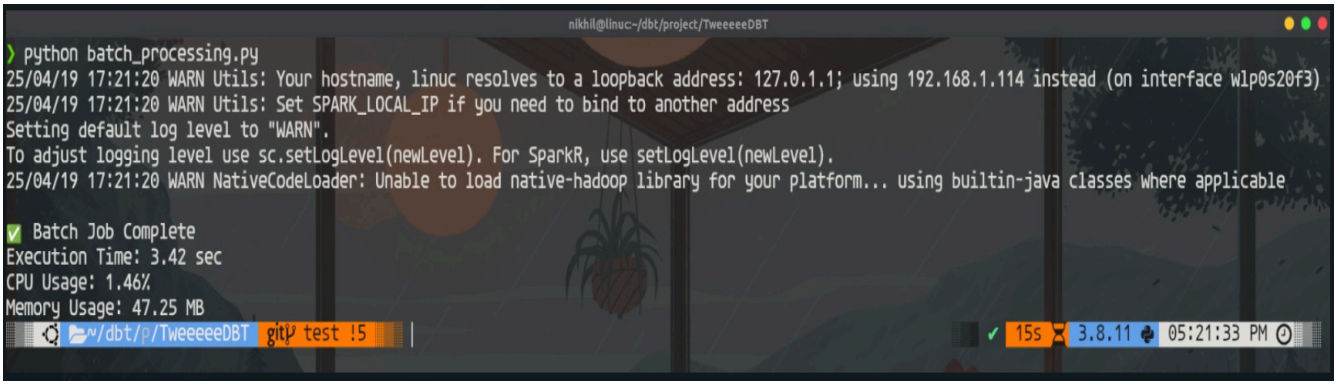
- The dataset is read from `IPL_2022_tweets.csv`.
- The script limits the data to the first 1500 rows using `df.limit(1500)`.
- These 1500 rows are then filtered into the three categories mentioned above (the exact split depends on data content, but processing is performed on all 1500 rows).

c) Results

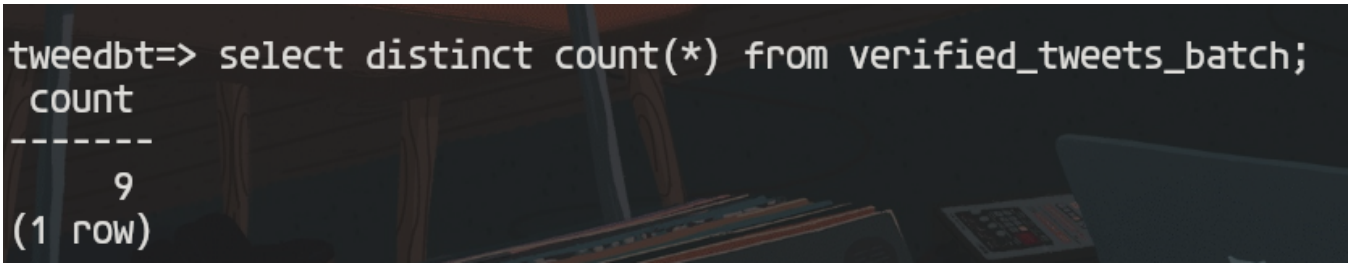
At the end of the batch processing:

- All qualifying tweets are inserted into their respective tables in a PostgreSQL database.
- The script prints:
    - **Execution Time:** Total wall time for the batch processing.
    - **CPU Usage:** Calculated manually from process user + system time over wall time.
    - **Memory Usage:** Memory consumed by the process at the end of execution.

batch_processing.py output



data inserted into verified_tweets_batch table



data inserted into team_tweets_batch table

```
tweedbt=> select distinct count(*) from team_tweets_batch;
 count
-------
    16
(1 row)
```

data inserted into geo_tweets_batch

```
tweedbt=> select distinct count(*) from geo_tweets_batch;
 count
-------
    28
(1 row)
```

Code

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower, trim, to_timestamp
import psycopg2
import time
import psutil
import os
from dotenv import load_dotenv

load_dotenv()

# Spark session setup
spark = SparkSession.builder \
      .appName("Independent Batch Processing") \
      .getOrCreate()

# Load CSV and limit to 1500 rows
df = spark.read.csv("IPL_2022_tweets.csv", header=True,
inferSchema=True)
df = df.limit(1500)
```

```
# Add timestamp column
df = df.withColumn("timestamp", to_timestamp(col("date"), "yyyy-MM-dd
HH:mm:ss"))

# Filter datasets
verified_df = df.filter(lower(trim(col("user_verified"))) == "true")
geo_df =
df.filter(lower(col("user_location")).rlike(r"\b(new\s*)?delhi\b|\bmumb
ai\b"))
team_df = df.filter(lower(col("hashtags")).rlike(r'.*?(rcb|csk).*?'))

# PostgreSQL DB config
DB_CONFIG = {
      'dbname': os.getenv('POSTGRES_DB'),
      'user': os.getenv('POSTGRES_USER'),
      'password': os.getenv('POSTGRES_PASSWORD'),
      'host': os.getenv('POSTGRES_HOST'),
      'port': os.getenv('POSTGRES_PORT')
}

# Check for missing env vars
required_vars = ['POSTGRES_DB', 'POSTGRES_USER', 'POSTGRES_PASSWORD',
'POSTGRES_HOST', 'POSTGRES_PORT']
missing_vars = [var for var in required_vars if not os.getenv(var)]

if missing_vars:
      raise EnvironmentError(f"Missing required environment variables:
{', '.join(missing_vars)}")

# Connect to PostgreSQL
try:
      conn = psycopg2.connect(**DB_CONFIG)
      cursor = conn.cursor()
except Exception as e:
      print(f"Error connecting to database: {e}")
      exit(1)
```

```
# Insert helper
def insert_data(df, table, cols):
      for row in df.collect():
      try:
            values = tuple(row[col] for col in cols)
            placeholders = ','.join(['%s'] * len(cols))
            query = f"INSERT INTO {table} ({','.join(cols)}) VALUES
({placeholders})"
            cursor.execute(query, values)
            conn.commit()
      except Exception as e:
            print(f"[ERROR] Inserting into {table}: {e}")
            conn.rollback()

# Start timing and resource tracking
start_time = time.time()
process = psutil.Process()
cpu_start = process.cpu_times()

# Insert data into respective tables
insert_data(verified_df, "verified_tweets_batch", ["user_name", "text",
"user_verified"])
insert_data(geo_df, "geo_tweets_batch", ["user_name", "text",
"user_location", "user_verified"])
insert_data(team_df, "team_tweets_batch", ["user_name", "text",
"hashtags", "user_verified"])

# End timing and calculate CPU usage manually
end_time = time.time()
cpu_end = process.cpu_times()
cpu_time_used = (cpu_end.user + cpu_end.system) - (cpu_start.user +
cpu_start.system)
wall_time = end_time - start_time
cpu_percent = (cpu_time_used / wall_time) * 100 if wall_time > 0 else 0
mem = process.memory_info().rss / 1024**2
```

```
# Report
print("\nBatch Job Complete")
print(f"Execution Time: {wall_time:.2f} sec")
print(f"CPU Usage: {cpu_percent:.2f}%")
print(f"Memory Usage: {mem:.2f} MB")

conn.close()
```

Comparison of Streaming & Batch Modes

a) Results

Batch mode and response time analysis were conducted independently by analyzing 1500 IPL tweet records. Different key performance indicators such as execution time, CPU usage, and memory were measured for every processing mode and category.

In the streaming mode, the work for authenticated users took a time of 788.75 seconds to complete, with a CPU usage of 0.88% and a memory usage of around 29.52 MB. For comparison, the geo-location stream took slightly less time at 766.7 seconds, with a CPU usage of 0.15% and a memory usage of 36.37 MB. The team reports that the stream took a comparable time of 776.42 seconds; however, it significantly showed the lowest CPU usage of 0.05%, with a memory usage of 33.79 MB.

Conversely, the batch processing approach ran all the 1500 rows of all three types — confirmed users, geo-tagged tweets, and team mentions — in a single uninterrupted run. Incurring a CPU load of 1.46% and a memory load of 47.25 MB, the batch job took only 3.42 seconds. While the batch mode used slightly higher memory compared to individual streaming operations, it far exceeded the streaming mode in time consumed and efficiency.

b) Discussion

These findings show that streaming mode is significantly slower for the same data because of its real-time processing design, dependency on Kafka buffering, and internal micro-batching. Batch processing, though slightly more memory-intensive, is far more efficient and quicker for bounded data. This renders batch mode extremely appropriate for offline analysis or processing past data, while streaming mode is more appropriate for real-time systems where responsiveness matters more than performance requirements.

```
tweedbt=> SELECT * FROM batch_metrics;
 id |   execution_time   |     cpu_percent     | memory_usage_mb |         timestamp
----+--------------------+---------------------+-----------------+----------------------------
  1 |  2.770416259765625 |   0.721913175664512 |         55.4375 | 2025-04-20 11:42:58.559876
  2 |  2.066192150115967 | 0.48398209234501877 |     55.90234375 | 2025-04-20 11:44:15.336609
  3 | 2.1284055709838867 |  0.9396705342560593 |      66.6015625 | 2025-04-20 11:44:30.87406
  4 | 2.0856893062591553 |  0.9589155939947528 |     53.10546875 | 2025-04-20 11:44:46.878842
  5 | 2.1444690227508545 |  0.9326317977932214 |     59.18359375 | 2025-04-20 11:45:11.617306
(5 rows)
```

```
tweedbt=> SELECT AVG(execution_time) AS avg_exec_time, AVG(cpu_percent) AS avg_cpu_percent, AVG(memory_usage_mb) AS avg_mem_usage FROM batch_metrics;
   avg_exec_time    |  avg_cpu_percent   | avg_mem_usage
--------------------+--------------------+---------------
 2.2390344619750975 | 0.8074226388107129 |   58.04609375
(1 row)
```

Conclusion

This project effectively utilized end-to-end streaming and batch processing pipelines for real-time and offline Twitter data processing of IPL. Utilizing Apache Kafka for message queuing, Apache Spark for data processing, and PostgreSQL for persistent storage, the system was able to consume, filter, and store the corresponding tweet data in a structured and scalable fashion.

In the streaming pipeline, live tweet data (or simulated stream data) was published to Kafka topics, consumed by Spark Structured Streaming, and filtered based on key conditions such as verified users, geo-tagged locations (Delhi and Mumbai), and team mentions (e.g., RCB and CSK). Each category was written to a distinct Kafka topic and then consumed by specialized Python consumer scripts which stored the results in PostgreSQL. The streaming pipeline was made fault-tolerant and continuous, using checkpointing mechanisms to make it reliable in real-time data ingestion.

At the same time, a batch processing pipeline was implemented to repeat the same logic over an unchanging data set. A random sample of 1500 rows was randomly selected from a larger IPL tweet data set and was processed in Apache Spark in batch mode. The same filters were used, and the results were then inserted into separate tables specific to the batch in PostgreSQL. This enabled direct comparison between both processing modes under the same logic and volume of data.

The performance test showed that batch processing exhibited much higher speed and efficiency over streaming in handling the same amount of data. Batch pipeline completion time was just a few seconds compared to the streaming pipeline, which took more than 12 minutes per category. While CPU usage in batch processing was slightly high, it remained within acceptable limits, while memory usage remained optimized in both processing modes. These findings endorse the hypothesis that batch processing is more resource-friendly for datasets that are classified as small to medium size, while streaming is suitable for use in applications where real-time updating of data or monitoring is required.

Accuracy analysis also supported that both modes yielded the same values, determining the correctness of implemented logic and transformations. The same SQL queries against both the streaming and batch output tables yielded equal counts and content, validating that both pipelines were functionally equivalent as output.

This project enabled the creation of a solid concept of distributed processing, stream vs. batch architecture differentiation, and end-to-end integration of data pipelines. The exposure to Kafka, Spark, and PostgreSQL in real-world scenarios enabled experiential work with modern big data technologies. Overall, the project served its purpose well and lays a good foundation for the scalability of similar architectures to real-world systems, such as live dashboards, real-time analysis, and mass-scale social media monitoring systems.

References

1. https://olympus.mygreatlearning.com/courses/64994/
2. https://kafka.apache.org/quickstart
3. https://youtu.be/EiMi11slVnY
4. https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html
5. https://spark.apache.org/docs/latest/api/python/getting_started/install.html
6. https://anishmahapatra.medium.com/apache-kafka-102-how-to-set-up-kafka-on-your-local-68f432dfb1ab
7. https://stackoverflow.com/questions/58961043/how-to-install-libpq-fe-h