Data Analysis in NBA Sport Events Streaming

Brian Yao
by2344@columbia.edu

Weirui Peng
wp2297@columbia.edu

Jingtian Zhang
jz3500@columbia.edu

Yifeng Wang
yw3933@columbia.edu

Abstract

In an era where sports leagues, including the NBA, heavily rely on viewership and ratings to secure lucrative TV deals, understanding fan engagement is critical. Social media platforms, particularly Twitter, have emerged as invaluable sources of insight into fans' attitudes and behaviors, with applications ranging from marketing to in-game analysis. The increasing popularity of social media and real-time streaming has transformed the fan experience, offering new opportunities for engagement. This paper delves into the potential of analyzing NBA-related tweets and harnessing the power of social media and real-time streaming to better comprehend and connect with fans. To this end, we present a real-time sentiment analysis system for NBA playoff events, leveraging Apache Spark Streaming, Twitter API, and the Google Cloud Platform (GCP). Our objective is to process and analyze tweet sentiments in real-time, providing insights into fan reactions, event highlights, and emerging trends during NBA playoffs.

1. Problem Definition

The rapid growth of social media platforms, particularly Twitter, has led to an unprecedented amount of user-generated content being produced in real-time during live events such as the NBA playoffs. This wealth of data presents an opportunity to analyze and understand public sentiment as it evolves throughout the course of the event. However, processing and analyzing the massive volume of streaming data generated during these events in real-time presents several challenges:

1. Scalability: The system must be able to handle the large volume of tweets generated during the NBA playoffs without compromising performance or accuracy. This requires an efficient and scalable stream processing architecture.
2. Real-time analysis: To provide valuable insights into the public's sentiment during the live events, the system must process and analyze tweets in real-time. This necessitates the implementation of efficient algorithms and methods for sentiment analysis.
3. Sentiment classification: Accurately classifying the sentiment of each tweet as positive, negative, or neutral is a complex natural language processing task. The system must employ pre-trained models or develop its own sentiment analysis models to achieve accurate results.
4. Visualization and interpretation: Aggregated sentiment analysis results need to be presented in a visually intuitive manner to enable stakeholders to easily understand and interpret the findings. The system must incorporate real-time visualization tools to display sentiment data effectively.

To summarize, the problem this project aims to address is to design and implement a real-time sentiment analysis system that overcomes these challenges by utilizing Apache Spark Streaming, Twitter API, and Google Cloud Platform to collect, process, analyze, and visualize the sentiment of tweets related to NBA playoff events in real-time.

2. Methodology

In this project, we first compiled a list of approximately 30 NBA teams to serve as our tag names and acquired their corresponding location information. During the streaming processing stage, we established a Dataproc cluster consisting of 3 master nodes and 2 worker nodes within the Google Cloud Platform (GCP) to create a distributed system capable of efficient streaming processing. Utilizing the team tags and location data, we queried the Twitter API for historical data and collected the results in separate CSV files using a Python script for Twitter Streaming Processing, as illustrated in Fig.1.

Subsequently, we stored the dataset in Google Cloud Storage (GCS) and read it as a real-time stream for processing. We applied windowing techniques (with a 30-minute

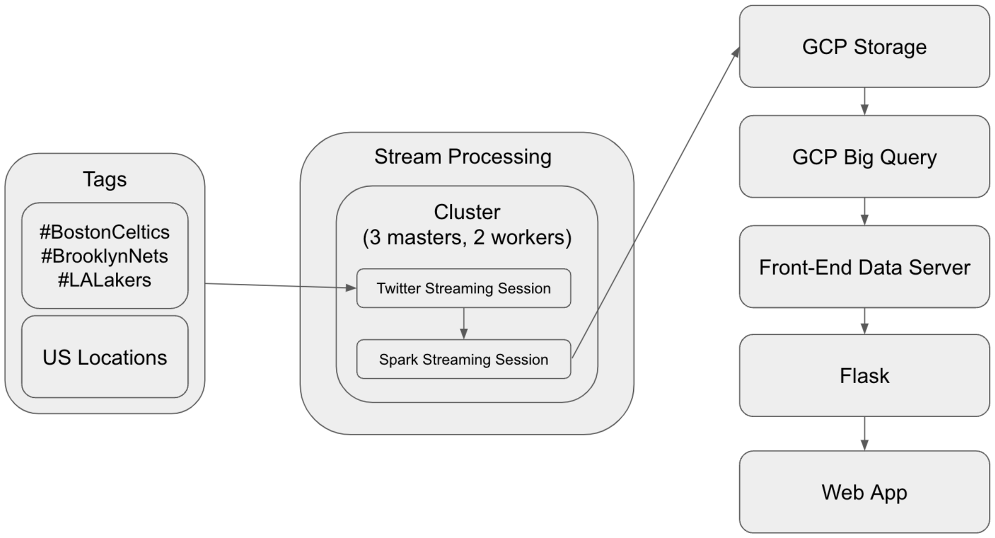


Figure 1. Workflow of the Project

window duration and a 30-minute sliding window) and implemented stream processing optimization methods such as data reordering and fusion techniques. Once processed, we saved the streamed data to GCS and subsequently imported it into BigQuery, a database-like service provided by GCP, for further visualization and analysis.

Finally, we executed queries on the data stored in BigQuery and employed Python scripts to generate informative graphs, which were then displayed on a Flask-based web application. Detailed explanations of each step and the methodologies employed are presented in the following sections.

2.1. Twitter Data Collection

To collect NBA-related tweets, we wrote a program to use the Twitter API through Tweepy, a Python library for accessing Twitter data, to gather data. Our total code about this part is in A.1 and the final dataset is in A.2 which is prepared for batching and streaming. After successfully forming the dataset, to get some features, we used batching to plot some graphs about total 7 days in Colab shown in A.4.

2.2. Streaming Session

After data collection, we emulated the real-time stream processing by reading CSV files as streams in Spark DataFrame format. We designed our processing pipeline to handle data transformations and actions within time windows of varying durations, such as 30-minute fixed windows and 30-minute sliding windows. To enhance the efficiency of our streaming pipeline, we employed multiple optimization techniques, including the reordering of operations and the fusion of transformations, which significantly reduced the overall execution time. A performance comparison between different optimization methods is also presented.

An illustration of the entire process can be found in Figure 2. The implementation details of this part of the project are provided in the source code, which is referenced as A.3. In the following sections, we delve deeper into the technical aspects of the streaming pipeline and optimizations applied.

2.3. Flask Website

We developed a web application utilizing the Flask framework to showcase the real-time sentiment analysis results and provide an interactive platform for users. The source code for the Flask application can be found in A.5. The web application employs Jinja templates to generate the frontend, consisting of two HTML files that enable a user-friendly interface and seamless navigation.

A notable feature of our web application is the highly customizable plot function in A.6. This function allows users to tailor the visualizations based on their requirements by modifying simple SQL queries and adjusting the plotting logic. This flexibility ensures that the web application remains adaptable to various use cases and can be easily extended in the future.

To ensure high availability and optimal performance, we deployed the web application on a Google Cloud Platform (GCP) Virtual Machine (VM) Instance, as detailed in A.7. However, due to ongoing operational costs, the availability of the web application on the GCP VM Instance may be subject to change in the future. We recommend exploring alternative hosting solutions or implementing cost optimization strategies to maintain the web application's long-term accessibility.

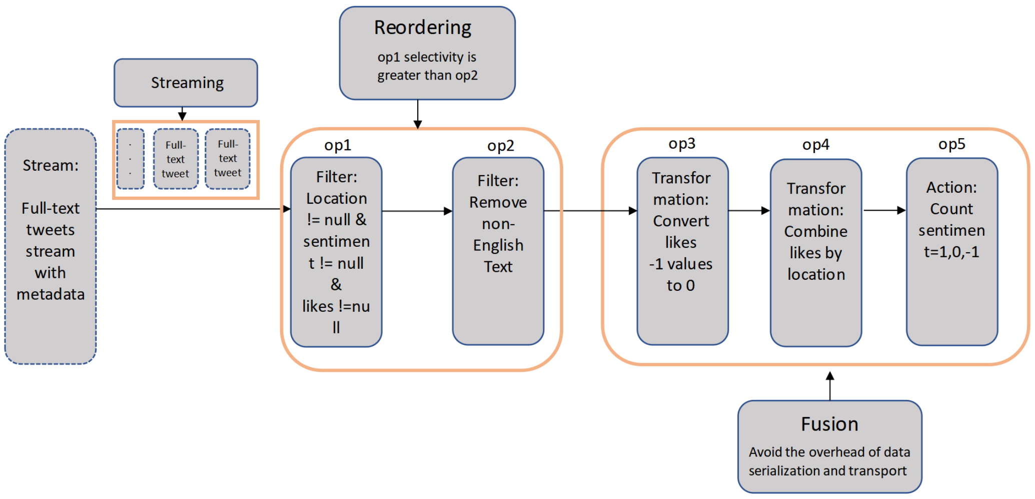
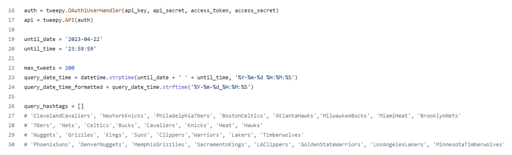

Reordering
op1 selectivity is greater than op2
Streaming
Stream:
Full-text tweets stream with metadata
Full-text tweet
Full-text tweet
op1
Filter: Location != null & sentiment != null & likes !=null
op2
Filter: Remove non-English Text
op3
Transformation: Convert likes -1 values to 0
op4
Transformation: Combine likes by location
op5
Action: Count sentiment=1,0,-1
Fusion
Avoid the overhead of data serialization and transport

Figure 2. Streaming Process with applied optimization methods



```
auth = tweepy.OAuth1UserHandler(api_key, api_secret, access_token, access_secret)
api = tweepy.API(auth)

until_date = '2023-04-22'
until_time = '23:59:59'

max_tweets = 200
query_date_time = datetime.strptime(until_date + ' ' + until_time, '%Y-%m-%d %H:%M:%S')
query_date_time_formatted = query_date_time.strftime('%Y-%m-%d_%H:%M:%S')

query_hashtags = []
# 'ClevelandCavaliers', 'NewYorkKnicks', 'Philadelphia76ers', 'BostonCeltics', 'AtlantaHawks','MilwaukeeBucks', 'MiamiHeat', 'BrooklynNets'
# '76ers', 'Nets', 'Celtics','Bucks', 'Cavaliers', 'Knicks', 'Heat', 'Hawks'
# 'Nuggets', 'Grizzles', 'Kings', 'Suns', 'Clippers','Warriors', 'Lakers', 'Timberwolves'
# 'PhoenixSuns', 'DenverNuggets', 'MemphisGrizzlies', 'SacramentoKings', 'LAClippers', 'GoldenStateWarriors', 'LosAngelesLakers', 'MinnesotaTimberwolves'
```

Figure 3. Access Token and Search Parameters

3. Data Collection

3.1. API Keys and Access Tokens

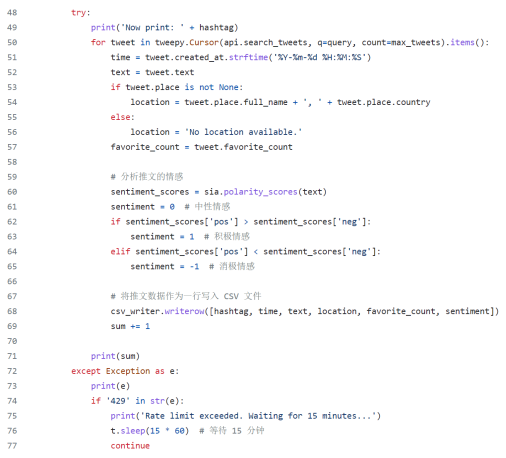
We set up the necessary API keys and access tokens to connect to the Twitter API through Tweepy, which is shown in fig.3. Note that we applied for an ELEVATED level Twitter developer access.

3.2. Search Parameters

We defined the search parameters, including the hashtags, date and time windows, and the quantity of tweets to be pulled per hashtag. Due to the Twitter API level limit, we only access data within the most recent 7 days, limiting our data collection to the period from 4/13/2023 to 4/21/2023.

3.3. Tweet Parsing and Saving

For each tweet, we parsed the tweet data for query matches, retrieving the creation time, text, location (if available), and like count. We then analyzed the sentiment using NLTK (Natural Language Toolkit)[3] and saved the hashtag, time, text, location, likes, and sentiment in a CSV file. The code is shown in fig.4.



```
        try:
            print('Now print: ' + hashtag)
            for tweet in tweepy.Cursor(api.search_tweets, q=query, count=max_tweets).items():
                time = tweet.created_at.strftime('%Y-%m-%d %H:%M:%S')
                text = tweet.text
                if tweet.place is not None:
                    location = tweet.place.full_name + ', ' + tweet.place.country
                else:
                    location = 'No location available.'
                favorite_count = tweet.favorite_count

                # 分析推文的情感
                sentiment_scores = sia.polarity_scores(text)
                sentiment = 0  # 中性情感
                if sentiment_scores['pos'] > sentiment_scores['neg']:
                    sentiment = 1  # 积极情感
                elif sentiment_scores['pos'] < sentiment_scores['neg']:
                    sentiment = -1  # 消极情感

                # 将推文数据作为一行写入 CSV 文件
                csv_writer.writerow([hashtag, time, text, location, favorite_count, sentiment])
                sum += 1

            print(sum)
        except Exception as e:
            print(e)
            if '429' in str(e):
                print('Rate limit exceeded. Waiting for 15 minutes...')
                t.sleep(15 * 60)  # 等待 15 分钟
                continue
```

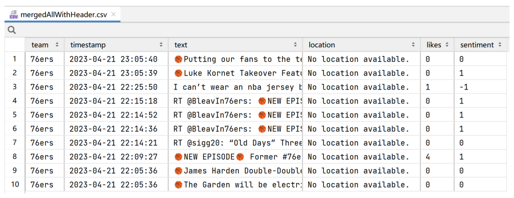
Figure 4. Tweet Parsing and Saving

3.4. Challenges

The data collection process encountered several challenges related to the Twitter API, which necessitated the implementation of various strategies and methods to obtain a sufficient amount of data for our analysis. The two primary difficulties we faced were as follows:

1. The Twitter API's standard access level limited us to retrieve data only from the most recent 7 days. To collect NBA-related tweets during the period from 4/13/2023 to 4/21/2023, we applied for ELEVATED level Twitter developer access, which granted us extended access to historical data. This allowed us to retrieve a more comprehensive dataset for our analysis.

2. The Twitter API imposed a rate limit of at most 15,000



mergedAllWithHeader.csv

	team	timestamp	text	location	likes	sentiment
1	76ers	2023-04-21 23:05:40	🏀Putting our fans to the te	No location available.	0	0
2	76ers	2023-04-21 23:05:39	🏀Luke Kornet Takeover Featu	No location available.	0	1
3	76ers	2023-04-21 22:25:50	I can't wear an nba jersey b	No location available.	1	-1
4	76ers	2023-04-21 22:15:18	RT @BleavIn76ers: 🏀NEW EPIS	No location available.	0	1
5	76ers	2023-04-21 22:14:52	RT @BleavIn76ers: 🏀NEW EPIS	No location available.	0	1
6	76ers	2023-04-21 22:14:36	RT @BleavIn76ers: 🏀NEW EPIS	No location available.	0	1
7	76ers	2023-04-21 22:14:21	RT @sigg20: "Old Days" Three	No location available.	0	0
8	76ers	2023-04-21 22:09:27	🏀NEW EPISODE🏀 Former #76e	No location available.	4	1
9	76ers	2023-04-21 22:05:36	🏀James Harden Double-Double	No location available.	0	0
10	76ers	2023-04-21 22:05:36	🏀The Garden will be electr	No location available.	0	0

Figure 5. Resulting Dataset

tweets within a 15-minute window. Consequently, our program encountered "429 error" messages when this limit was reached, requiring us to pause data collection for 15 minutes. To address this issue, we implemented an adaptive rate limiting strategy that monitored the API usage and dynamically adjusted the request frequency to avoid hitting the rate limit (see code in fig.4). Additionally, we leveraged multiple API keys and distributed our requests across them to further increase the data collection throughput.

3.5. Resulting Dataset

The resulting dataset is composed of multiple CSV files, with each file corresponding to a specific hashtag related to the NBA playoff events. These files contain comprehensive tweet data, including the hashtag, timestamp, tweet text, user location, like count, and sentiment score for each tweet, as illustrated in fig.5.

Our dataset exhibits extensibility and flexibility, as it can be easily expanded to incorporate additional data. To achieve this, users can simply modify the "until" date parameter in the data collection process, allowing the system to access and process new data corresponding to more recent events. Furthermore, our approach can also be adapted to include other relevant hashtags or keywords, as well as additional data attributes such as retweet count and user metadata. This versatility enables the dataset to evolve and stay current with the ongoing NBA playoffs, providing richer insights and more comprehensive analysis.

3.6. Batching

We generated several graphs by batching to plot some graphs about total 7 days in Colab shown in A.4, because it can reflect some features or valuable things. The results are shown in fig.6, fig.7, fig.8, fig.9, fig.9, fig.10, fig.11, fig.12, fig.13. The graphs can be referred to what you can plot from the dataset.

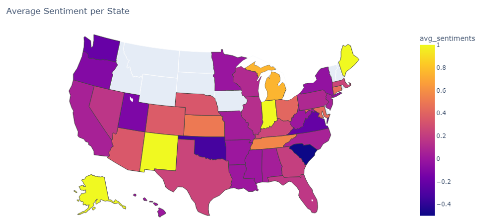


Figure 6. Average Sentiment per State

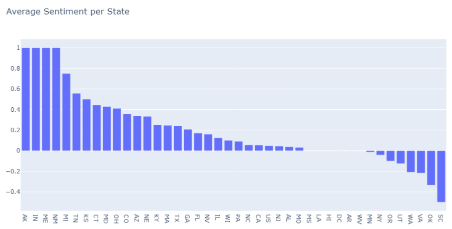


Figure 7. Detailed Version of Average Sentiment per State

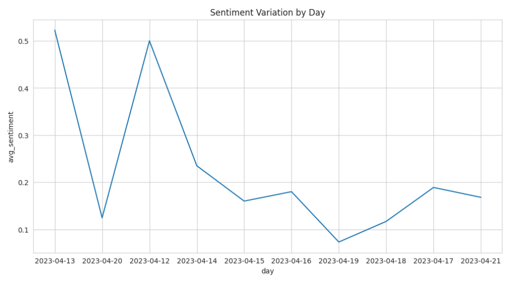


Figure 8. Sentiment by Day

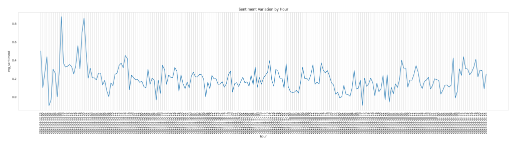


Figure 9. Sentiment by Hour

4. Streaming Algorithms and Optimization

4.1. Streaming Algorithms

We have employed various streaming algorithms, particularly Spark Streaming, along with additional methods to comprehensively process the real-time stream of NBA-related tweets. Spark Streaming enables the real-time processing of massive volumes of data by dividing the incoming stream into small batches and applying a series of transformations, such as filtering, aggregation, and window operations, to extract valuable insights and identify emerging trends. The code demonstrates the implementation of a sophisticated stream processing pipeline for analyzing NBA-related tweets using Apache Spark and incorporating additional techniques[5]. The stream processing is carried out through the following steps:

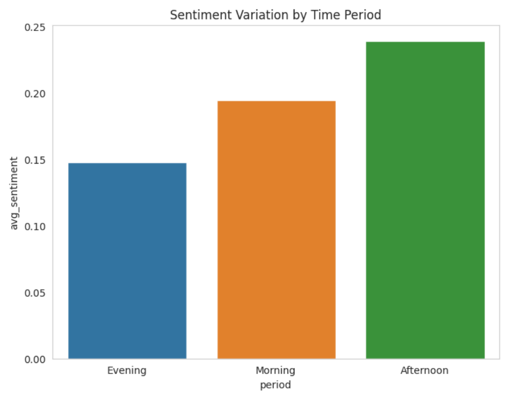


Figure 10. Sentiment by Period

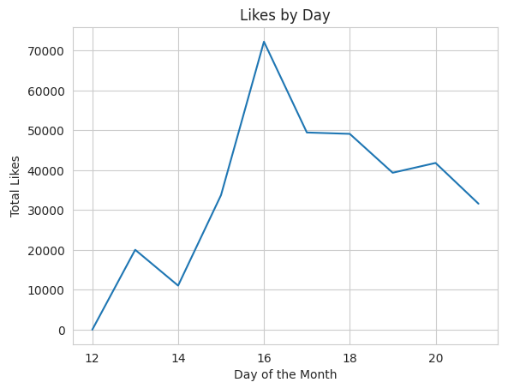


Figure 11. Like by Day

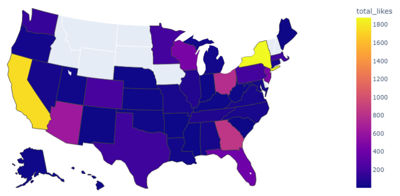


Figure 12. Total Likes by State

1. Importing necessary libraries and creating a User Defined Function (UDF) called is_english_udf to filter out non-English tweets.

2. Defining the schema for the input data with appropriate data types and structuring the data using the StructType and StructField classes.

3. Initializing a SparkSession with the application name "Tweet Analysis." Reading the streaming data from a Google Cloud Storage (GCS) bucket using the spark.readStream.csv function, and specifying the schema and input data path.

4. Filtering the streaming data to ensure that the location, likes, and sentiment fields are not null (See fig.15). The likes field is also updated to replace -1 values with 0. Non-English tweets are filtered out using the is_english_udf function. A watermark is added to the timestamp field with a duration of 1 hour to handle late data.

5. Aggregating the filtered data by grouping it based on a 30-minute window and location. Within each group, the code calculates the total likes, counts of tweets with sentiment values of 1, 0, and -1, and the average sentiment. The code is shown in fig.16.

6. Selecting the relevant columns of aggregated data, including the window start and end times, to form the final output.

7. Writing the aggregated data to a checkpoint folder in the GCS bucket using the writeStream function, specifying the output mode as "append" and the format as "parquet" (see fig.17). The checkpoint folder is provided to maintain the state of the streaming query and enable the fault tolerance for our system.

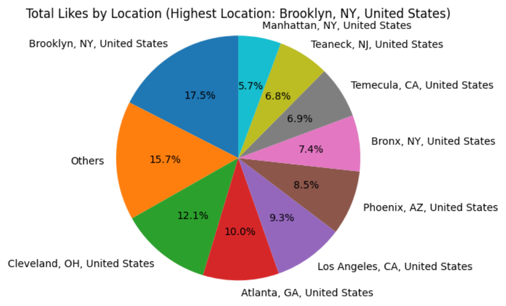


Figure 13. Total Likes by Location

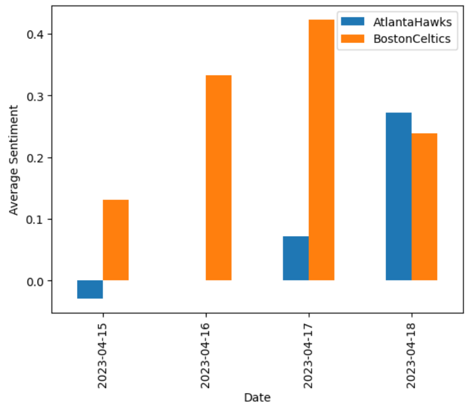
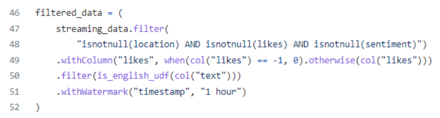
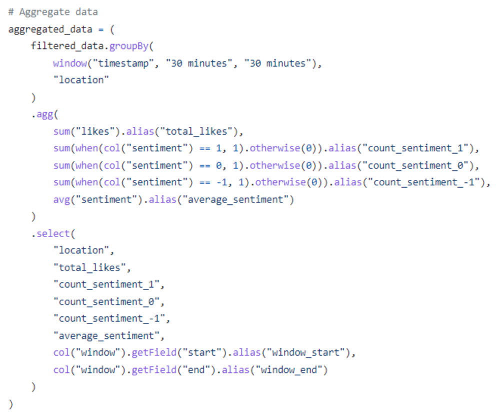


Figure 14. Example about What you can get from dataset by teams



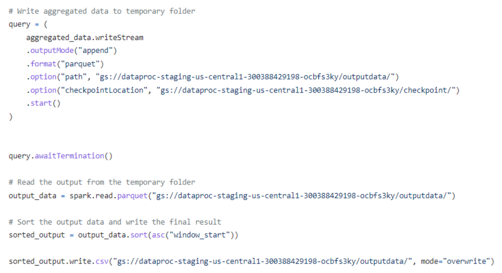
```
filtered_data = (
    streaming_data.filter(
        "isnotnull(location) AND isnotnull(likes) AND isnotnull(sentiment)")
    .withColumn("likes", when(col("likes") == -1, 0).otherwise(col("likes")))
    .filter(is_english_udf(col("text")))
    .withWatermark("timestamp", "1 hour")
)
```

Figure 15. Filter Non-null



```
# Aggregate data
aggregated_data = (
    filtered_data.groupBy(
        window("timestamp", "30 minutes", "30 minutes"),
        "location"
    )
    .agg(
        sum("likes").alias("total_likes"),
        sum(when(col("sentiment") == 1, 1).otherwise(0)).alias("count_sentiment_1"),
        sum(when(col("sentiment") == 0, 1).otherwise(0)).alias("count_sentiment_0"),
        sum(when(col("sentiment") == -1, 1).otherwise(0)).alias("count_sentiment_-1"),
        avg("sentiment").alias("average_sentiment")
    )
    .select(
        "location",
        "total_likes",
        "count_sentiment_1",
        "count_sentiment_0",
        "count_sentiment_-1",
        "average_sentiment",
        col("window").getField("start").alias("window_start"),
        col("window").getField("end").alias("window_end")
    )
)
```

Figure 16. Aggregation



```
# Write aggregated data to temporary folder
query = (
    aggregated_data.writeStream
    .outputMode("append")
    .format("parquet")
    .option("path", "gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/outputdata/")
    .option("checkpointLocation", "gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/checkpoint/")
    .start()
)


query.awaitTermination()

# Read the output from the temporary folder
output_data = spark.read.parquet("gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/outputdata/")

# Sort the output data and write the final result
sorted_output = output_data.sort(asc("window_start"))

sorted_output.write.csv("gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/outputdata/", mode="overwrite")
```

Figure 17. Writing aggregated data to folder

8. Awaiting the termination of the streaming query using query.awaitTermination().

9. Reading the output data from the temporary folder using the spark.read.parquet function.

10. Sorting the output data by the window start time in ascending order.

11. Writing the sorted output data back to the GCS bucket as a CSV file.

This stream processing pipeline efficiently processes real-time tweet data, filters and aggregates it based on the required criteria, and stores the final output for further analysis or visualization.

4.2. Streaming Optimization

We applied some streaming optimization methods and compared the time by using and not using them.

4.2.1 Reorder

To enhance query performance, we implemented a reorder optimization technique in our code by adjusting the sequence of two filter operations, which is shown in fig.2. Specifically, we modified the order of Filter A, which eliminates non-English tweets, and Filter B, which removes null values. We placed Filter B before Filter A due to its higher selectivity, ensuring that a larger number of irrelevant data points are filtered out early in the process. This strategic reordering effectively reduces data overload and decreases runtime, leading to a more efficient and streamlined analysis[2]. Also the result is in table.1.

Run time Comparison		
# Round	Non-optimized(s)	Optimized(s)
1	13.567	10.854
2	13.678	10.942
3	13.748	11.025
4	13.841	11.112
5	13.943	11.208
6	14.019	11.287
7	14.112	11.375

Table 1. Run time Comparison when Applying Reorder

4.2.2 Fusion

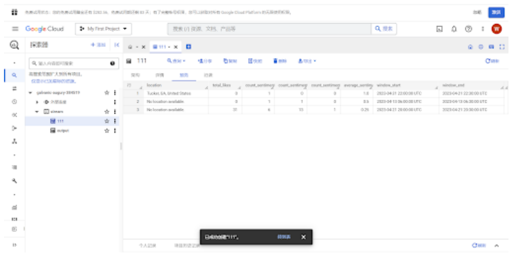
Moreover, we have also implemented fusion optimization in our code by consolidating the transformation and action operations on the filtered data. This involves aggregating the likes and sentiment values for each location and subsequently computing the average sentiment value for different locations. By merging these operations, we effectively reduce the number of computations necessary for data processing[4], leading to enhanced performance and efficiency[1]. To illustrate clearly, the graph is shown in fig.2. Also the result is in table.2.

Run time Comparison		
# Round	Non-optimized (s)	Optimized(s)
1	13.567	12.214
2	13.678	12.408
3	13.753	12.518
4	13.841	12.611
5	13.945	12.722
6	14.012	12.814
7	14.112	12.922

Table 2. Run time Comparison when Applying Fusion

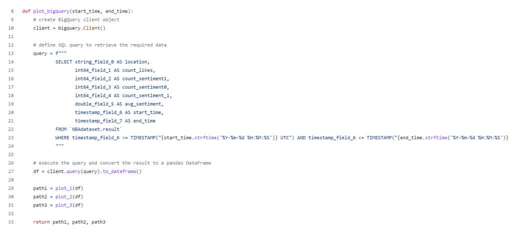
4.3. Results

The enhanced output dataset provides a comprehensive overview of sentiment information and tweet popularity,



行	location	total_likes	count_sentimen	count_sentimen	count_sentimen	average_sentim	window_start	window_end
1	Tucker, GA, United States	3	1	0	0	1.0	2023-04-21 22:00:00 UTC	2023-04-21 22:30:00 UTC
2	No location available	3	1	1	0	0.5	2023-04-13 06:00:00 UTC	2023-04-13 06:30:00 UTC
3	No location available	31	6	13	1	0.26	2023-04-21 20:00:00 UTC	2023-04-21 20:30:00 UTC

Figure 18. Streamed dataset



```python
def plot_bigquery(start_time, end_time):
    # create BigQuery client object
    client = bigquery.Client()

    # define SQL query to retrieve the required data
    query = f"""
            SELECT string_field_0 AS location,
                   int64_field_1 AS count_likes,
                   int64_field_2 AS count_sentiment1,
                   int64_field_3 AS count_sentiment0,
                   int64_field_4 AS count_sentiment_1,
                   double_field_5 AS avg_sentiment,
                   timestamp_field_6 AS start_time,
                   timestamp_field_7 AS end_time
            FROM `NBAdataset.result`
            WHERE timestamp_field_6 >= TIMESTAMP("{start_time.strftime('%Y-%m-%d %H:%M:%S')} UTC") AND timestamp_field_6 <= TIMESTAMP("{end_time.strftime('%Y-%m-%d %H:%M:%S')}
            """

    # execute the query and convert the result to a pandas DataFrame
    df = client.query(query).to_dataframe()

    path1 = plot_1(df)
    path2 = plot_2(df)
    path3 = plot_3(df)

    return path1, path2, path3
```

Figure 19. Plot Function

taking into account various factors across different locations. This enriched dataset not only displays the sentiment classification (positive, negative, or neutral) for each tweet, but also considers additional metrics to offer a more detailed analysis. These metrics include tweet engagement, sentiment score, and temporal analysis. The dataset is shown in the fig.18.

5. Flask Website

5.1. Basic Structure

We applied a HTML template from a free template website, and changed the content of them. Then we used Flask as the server to ensure the routes are correct. As you can see in A.5, we run this app on port 8111 and only implement '/' and '/plot'.

5.2. Plot Function

Our plot function is powered by BigQuery (fig.19). This is an example when selecting Start Time: 2023-04-15 15:07:00, End Time: 2023-04-15 17:07:00 and Duration: 2 hours, and resulting graphs are shown in fig.20, fig.21 and fig.22. It means if you select another start time and duration, the program will automatically help you generate the corresponding graphs.

It is implemented by a SQL query and plot logic, so it is easy to change it to what you want. Say, you want a picture in fig.14, you can just modify our codes in streaming to maintain team column since we grouped them by location as told before and change this part of the code, including a new query and new plot logic, then the graph will be obtained.

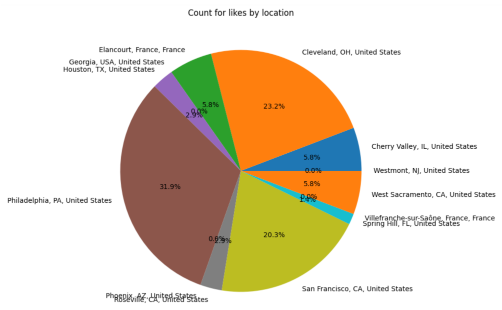


Figure 20. Example Picture 1

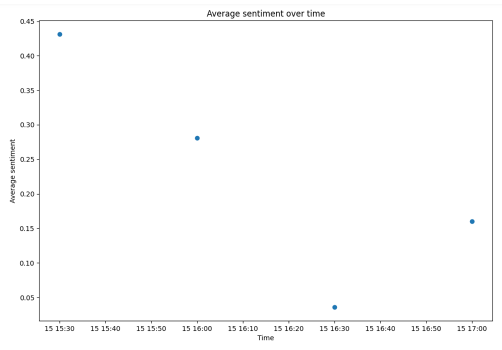


Figure 21. Example Picture 2

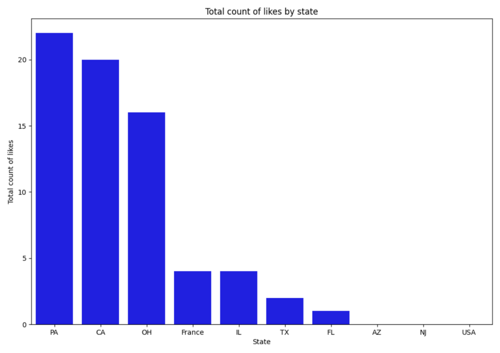


Figure 22. Example Picture 3

5.3. GCP Deployment

In order to deploy our Flask website, we utilized the Google Cloud Platform (GCP) for its flexibility and scalability. The deployment process involved the following steps:

1. We created a new Virtual Machine (VM) instance on GCP with minimal settings to minimize costs while maintaining adequate performance.

2. We installed and configured all the necessary dependencies, such as libraries and packages, required for the successful operation of our Flask application.

3. We configured Identity and Access Management (IAM) policies to grant the appropriate user permissions to access and interact with BigQuery, as our website relies on data stored within this service.

4. We exported the GOOGLE APPLICATION CREDENTIALS environment variable, pointing it to the JSON file containing the necessary GCP credentials that we downloaded earlier.

5. We launched the Flask application on the VM instance, making it accessible through an external IP address and port as shown in A.7.

Please note that the deployed website will be taken down upon the completion of this project to reduce ongoing costs and ensure data privacy. The techniques and processes discussed in this report can be adapted and utilized for future projects with similar requirements.

6. Future Work

In the future, we aim to enhance our system by incorporating more dynamic and interactive visualizations on the website, allowing users to better understand the real-time development of popularity and sentiment trends related to ongoing NBA events. Additionally, we plan to explore various data visualization techniques and libraries, such as D3.js, to create more engaging and informative graphics.

Regarding sentiment analysis, we intend to migrate the sentiment analysis task to the GCP platform, leveraging its distributed computing capabilities and machine learning services, such as Cloud Natural Language API. By integrating these services into our data streaming pipeline powered by Apache Spark, we can achieve seamless, efficient, and scalable sentiment analysis on streaming tweet data.

Furthermore, we plan to evaluate the use of Apache Kafka for real-time data streaming, in order to improve the efficiency, reliability, and fault-tolerance of our system. By incorporating Kafka as a messaging system, we can facilitate seamless data ingestion, processing, and dissemination across various components of our architecture.

Lastly, we also aim to investigate other advanced sentiment analysis techniques, such as deep learning-based models and transfer learning, to further improve the accuracy and relevance of our sentiment classification. By continuously refining our methods and incorporating new technologies, we aspire to create a more robust and comprehensive real-time sentiment analysis system for NBA playoff events.

7. Conclusion

In conclusion, this project aimed to analyze fan engagement with the National Basketball Association (NBA) using Twitter data. We collected NBA-related tweets by querying the Twitter API with team names and locations, and subsequently processed the data within a distributed system on the Google Cloud Platform (GCP). To efficiently analyze the real-time stream of data, we applied stream processing optimization techniques, such as reordering and fusion methods. The processed data was then stored in Google Cloud Storage (GCS) and BigQuery for in-depth analysis. Lastly, we developed a Flask-based web application to showcase the results and deployed it on GCP, making it accessible to visitors interested in understanding fan sentiment during NBA events.

Acknowledgments

This is the final project of ELEN 6889 Large Scale Stream Processing. Thank Professor Deepak S. Turaga for his help in proposal and presentation and useful suggestions about our current work, which could improve our future work.

References

[1] Henrique CM Andrade, Buğra Gedik, and Deepak S Turaga. *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press, 2014. 6

[2] Babu S. Arasu, A. and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal 15, 2 (June)*, 121–142. 6

[3] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, 2006. 3

[4] Thies W. Karczmarek M. Lin J. Meli A. S. Lamb A. A. Leger C. Wong J. Hoffmann H. Maze D. Gordon, M. I. and S. Amarasinghe. A stream compiler for communication-exposed architectures. *In Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, page 291–303, 2002. 6

[5] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164, 2016. 4

A. Codes

A.1. getData.py

https://github.com/Iris1e27/ELEN6889_Final_Project/blob/master/dataset/getData.py

A.2. data.csv

https://github.com/Iris1e27/ELEN6889_Final_Project/blob/master/dataset/mergedAllWithHeader.csv

A.3. streaming.py

https://github.com/Iris1e27/ELEN6889_Final_Project/blob/master/analysis/streaming_method_NBA.py

A.4. batching.ipynb

https://colab.research.google.com/drive/1GGL1oNCNxmmnd_7BbtPFLDrlc9JRgxN4?usp=sharing

A.5. flask.py

https://github.com/Iris1e27/ELEN6889_Final_Project/blob/master/webpage/flask/__init__.py

A.6. plotBigQuery.py

https://github.com/Iris1e27/ELEN6889_Final_Project/blob/master/webpage/flask/plotBigQuery.py

A.7. NBA Data Streaming based on Twitter website

http://34.66.247.178:8111/