



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Department of Computer Science
Faculty of Computing

Project 2: Real-Time Sentiment Analysis using Apache Spark and Kafka

Programme : Bachelor of Computer Science (*Data Engineering*)

Subject Code : SECP3133

Subject Name : High Performance Data Processing

Session-Sem : 2024/2025-2

Prepared by : DANIAL HARRIZ BIN MOHD (A22EC0152)
ASINEH @MOHD ASNEH

TIEW CHUAN RONG (A22EC0112)

CHAI YU TONG (A22EC0145)

KOH SU XUAN (A22EC0060)

Section : 01

Lecturer : ASSOC. PROF. DR. MOHD SHAHIZAN BIN OTHMAN

Date : 07-07-2025

Table of Content

1.0 Introduction.....	1
1.1 Background of the project.....	1
1.2 Objectives.....	2
1.3 Project Scope.....	2
2.0 Data Acquisition & Preprocessing.....	4
2.1 Data Source.....	4
2.2 Data Acquisition Tools and Process.....	4
2.3 Data Preprocessing and Cleaning Steps.....	6
3.0 Sentiment Model Development.....	8
3.1 Model Choice.....	8
3.2 Training Process.....	9
4.0 Apache System Architecture.....	14
4.1 Component Roles and Configuration.....	14
4.2 Workflow Diagram and Data Flow.....	16
5.0 Analysis & Results.....	18
5.1 Key Findings.....	18
5.2 Visualization.....	20
6.0 Optimization & Comparison.....	26
6.1 Model Optimization.....	26
6.2 Model Comparison.....	27
7.0 Conclusion & Future Work.....	28
References.....	29
Appendix.....	30

1.0 Introduction

1.1 Background of the project

In our current digital era, social media platforms have truly transformed into key avenues where people express themselves and share their feelings, particularly during significant cultural and religious events. Hari Raya, as one of Malaysia's most celebrated festivals, consistently generates substantial online engagement through various social media platforms. Among these, YouTube notably emerges as a prominent platform for sharing festive content, music, and diverse cultural expressions.

The increase of user-generated content during Hari Raya celebrations shows an opportunity to understand public sentiment and cultural engagement patterns. However, manually analyzing thousands of comments across multiple videos is time consuming and not practical. This project addresses this challenge by implementing an automated real-time sentiment analysis pipeline that leverages cutting-edge big data technology to process and analyze YouTube comments related to Hari Raya.

The system integrates multiple Apache technologies to create a scalable, distributed architecture capable of handling large volumes of streaming data. By combining batch processing, data streaming, machine learning and real-time analysis, this project demonstrates how modern data engineering practices can be applied to extract meaningful insights from social media content that provides valuable perspective on public sentiment during cultural celebrations.

1.2 Objectives

- **Real time Data Processing**

Develop a streaming pipeline that can continuously collect and process YouTube comments related to Hari Raya videos using Apache Kafka for data ingestion and Apache Spark for distributed processing.

- **Sentiment Analysis Implementation**

Implement and deploy machine learning models to automatically classify comment sentiments as positive, negative, or neutral for enabling real-time understanding of public sentiment. In this project, we have used the Logistic Regression model.

- **Data Visualization and Monitoring**

Create interactive dashboards using Kibana to visualize sentiment trends, comment patterns and real-time analysis to provide actionable insights.

- **Model Performance Evaluation**

Evaluate and compare the performance of different machine learning models using multiple evaluation metrics including Accuracy, Precision, Recall and F1 Score analysis to identify the most effective approach for sentiment classification of Malay language content.

1.3 Project Scope

- **Data Sources:**

The system focuses on YouTube comments from selected Hari Raya-related videos, including traditional songs, modern interpretations, and cultural content. The dataset includes comments in multiple languages (primarily Malay, with some English and regional languages) from videos with varying popularity levels.

- **Containerization and Orchestration:** The project implements a fully containerized architecture using:

- a. **Docker Containers:** Individual containers for each service component ensuring isolation and portability
- b. **Docker Compose:** Multi-container orchestration for simplified deployment and service management

- c. **Service Isolation:** Each component (Zookeeper, Kafka, Spark, Elasticsearch, Kibana) runs in its own container
 - d. **Volume Management:** Persistent data storage and model sharing between containers
 - e. **Network Configuration:** Inter-container communication and external port exposure
- **Machine Learning Models:**

The scope includes training and deploying two primary sentiment classification models:

 - a. Naive Bayes classifier for baseline sentiment analysis
 - b. Logistic Regression model for comparative performance evaluation
 - c. TF-IDF vectorization for text feature extraction
- **System Architecture:**

The project implements a microservices architecture with:

 - a. Zookeeper for distributed coordination
 - b. Kafka for message streaming
 - c. Spark for stream processing
 - d. Elasticsearch for data storage and search
 - e. Kibana for visualization and monitoring
- **Model Evaluation and Comparison**
 - a. **Accuracy:** Overall classification accuracy for both models
 - b. **Precision:** Scores for positive, negative, and neutral sentiment classes
 - c. **Recall:** Scores to measure the ability to identify all instances of each sentiment class
 - d. **F1 Score:** Harmonic mean of precision and recall for balanced performance evaluation

2.0 Data Acquisition & Preprocessing

2.1 Data Source

The primary data source for this project was comments from YouTube videos. YouTube was chosen as a rich source of public opinion for several reasons. The reasons include

- **High Engagement**

YouTube is one of the most popular social media platforms in Malaysia, where users actively share their thoughts and feelings in the comments section.

- **Cultural Relevance**

For a topic like "Hari Raya," YouTube has tons of vlogs, advertisements and music videos that generate significant public discussion. This makes it an ideal platform to capture genuine sentiment.

- **Data Accessibility**

With the right tools, YouTube comments can be systematically collected and provide a large volume of text data for analysis.

The search query 'Hari Raya' was selected to ensure the collected data was directly related to a significant cultural event in Malaysia to align with the project's requirement, which is to analyze sentiment on a locally relevant topic.

2.2 Data Acquisition Tools and Process

The data acquisition process was automated using Python scripts and key libraries.

1. **googleapiclient**

```
from googleapiclient.discovery import build
```

The build function from this library was used to interact with the official YouTube Data API v3. This allowed us to programmatically search for videos relevant to our query.

2. **youtube-comment-downloader**

```
from youtube_comment_downloader import YoutubeCommentDownloader
```

A specialized Python library designed to efficiently scrape comments from specified

YouTube video URLs without violating YouTube's terms of service for public data access.

3. Pandas

```
import pandas as pd
```

Used for structuring the collected data into a DataFrame and saving it for offline processing.

The acquisition workflow was executed in mainly four steps.

1. Video Search

```
# === CONFIGURATION ===
API_KEY = '...' # Replace with your API key
SEARCH_QUERY = 'Hari Raya'
MAX_COMMENTS = 1000
RAW_CSV_FILE = 'youtube_comments.csv'

# === STEP 1: Search YouTube for "Hari Raya" videos ===
youtube = build('youtube', 'v3', developerKey=API_KEY)

search_response = youtube.search().list(
    q=SEARCH_QUERY,
    type='video',
    part='id,snippet',
    maxResults=10 # Fetch top 10 videos
).execute()
```

An API call was made to the YouTube Data API to search for the top 10 most relevant videos using the search query "Hari Raya".

2. Video ID Extraction

```
video_ids = [item['id']['videoId'] for item in search_response['items']]
print(f"🔍 Found {len(video_ids)} videos for '{SEARCH_QUERY}'")
```

From the search results, the unique videoId for each of the 10 videos was extracted.

3. Comment Scraping and Storage

```
# === STEP 2: Download Comments ===
downloader = YoutubeCommentDownloader()
all_comments = []

for vid in video_ids:
    try:
        url = f'https://www.youtube.com/watch?v={vid}'
        print(f"📄 Scraping comments from: {url}")
        comments = downloader.get_comments_from_url(url)
        for comment in comments:
            all_comments.append({'video_id': vid, 'comment': comment['text']})
            if len(all_comments) >= MAX_COMMENTS:
                break
        time.sleep(1) # Avoid being blocked
    except Exception as e:
        print(f"❌ Error scraping video {vid}: {e}")
    if len(all_comments) >= MAX_COMMENTS:
        break
```

The youtube-comment-downloader library was then used to iterate through the list of videoIds. For each video, the script fetched the associated public comments. A loop collected the comments until a predefined limit of 1,000 comments was reached. Each comment was stored along with its source videoId to maintain traceability.

4. Raw Data Export

```
# === STEP 3: Save raw comments to CSV ===
df = pd.DataFrame(all_comments[:MAX_COMMENTS])
df.to_csv(RAW_CSV_FILE, index=False)
print(f"✅ Saved {len(df)} raw comments to {RAW_CSV_FILE}")
```

The final collection of raw comments was structured into a Pandas DataFrame and saved as a CSV file named youtube_comments.csv.

2.3 Data Preprocessing and Cleaning Steps

Raw text from social media is inherently noisy, containing URLs, special characters, inconsistent capitalization and other elements that can decrease the performance of a sentiment analysis model. Therefore, a preprocessing and cleaning pipeline was implemented using the Pandas and re (Regular Expressions) libraries in Python.

A function, `clean_text`, was defined to apply the steps sequentially to each comment.

1. Lowercasing

```
text = str(text).lower()
```

All text was converted to lowercase to ensure uniformity. For example, "Happy", "happy" and "HAPPY" are treated as the same word.

2. URL Removal

```
text = re.sub(r"http\S+|www\S+", '', text) # Remove URLs
```

A regular expression (`re.sub(r"http\S+|www\S+", "", text)`) was used to identify and remove all hyperlinks from the comments.

3. Symbol and Emoji Removal

```
text = re.sub(r'^a-zA-Z0-9\s.,!?', '', text) # Remove symbols/emojis
```

To focus on the textual content, another regular expression (`re.sub(r'^a-zA-Z0-9\s.,!?', "", text)`) was applied. This removed most special characters and emojis while preserving essential punctuation that can convey sentiment, such as periods (.), commas (,), question marks (?) and exclamation marks (!).

4. Whitespace Normalization

```
text = re.sub(r'\s+', ' ', text).strip() # Normalize whitespace
```

Multiple consecutive spaces were collapsed into a single space and any leading or trailing whitespace was removed using `.strip()`. This standardizes the spacing within and around the text.

5. Removal of Empty Comments

```
df = df[df['clean_comment'].str.strip() != '']
```

After the cleaning process, some comments became empty strings (e.g., a comment that only contained a URL or emojis). These rows were identified and dropped from the dataset to prevent errors in the subsequent analysis stages.

The cleaned text was stored in a new column named `clean_comment` and this processed data was saved to a new file, `cleaned_youtube_comments.csv`, ready for the sentiment labeling and model development phase.

3.0 Sentiment Model Development

3.1 Model Choice

To fulfill the project requirement of exploring at least two different approaches, we selected two classic machine learning models for training and classification with the assistance of one deep learning-based method for data labeling.

1. Pre-trained Transformer Model (for Data Labeling)

```
from transformers import pipeline

# Load Indonesian sentiment model
sentiment_pipeline = pipeline(
    "sentiment-analysis",
)

# Define logic to convert to NEUTRAL based on confidence
def classify_sentiment(text):
    result = sentiment_pipeline(text)[0]
    label = result['label'].upper() # Make consistent (POSITIVE/NEGATIVE)
    score = result['score']

    if score < 0.90:
        return "NEUTRAL", score
    else:
        return label, score

# Apply to DataFrame
df[['sentiment', 'score']] = df['clean_comment'].apply(
    lambda x: pd.Series(classify_sentiment(x))
)

# Save result
df.to_csv("labeled_youtube_comments.csv", index=False)
print("✅ Comments labeled with POSITIVE, NEGATIVE, and NEUTRAL.")
```

To generate high-quality sentiment labels for our unlabeled YouTube comments, we used a pre-trained model from the Hugging Face Transformers library. The `pipeline("sentiment-analysis")` function was utilized, which by default loads a DistilBERT model fine-tuned on the SST-2 (Stanford Sentiment Treebank) dataset.

2. Supervised Machine Learning Models (for Training)

The labels generated by the transformer model were used to train two machine learning classifiers known for their effectiveness in text classification tasks.

a. **Multinomial Naive Bayes (MNB)**

A probabilistic classifier based on Bayes' theorem. It is a popular baseline for text classification due to its computational efficiency, simplicity and strong performance, especially with features like word counts or TF-IDF scores.

b. **Logistic Regression**

A linear model that is highly effective for binary and multi-class classification. It is more robust than Naive Bayes as it does not assume feature independence and often yields high accuracy. It serves as a strong, non-probabilistic counterpart to Multinomial Naive Bayes.

3.2 Training Process

The training workflow was carefully structured to ensure the models were trained on clean, relevant features and that the process was reproducible.

1. **Automated Data Labeling**

The cleaned comments from the preprocessing stage were fed into the pre-trained sentiment analysis pipeline. A custom function, `classify_sentiment`, was implemented with the logic below.

```
result = sentiment_pipeline(text)[0]
label = result['label'].upper() # Make consistent (POSITIVE/NEGATIVE)
score = result['score']
```

a. The pipeline classifies each comment, returning a label ('POSITIVE' or 'NEGATIVE') and a confidence score.

```
if score < 0.90:
    return "NEUTRAL", score
else:
    return label, score
```

b. A confidence threshold of 0.90 was established. If the model's confidence score for a prediction was below this threshold, the comment was classified as 'NEUTRAL'. This logic helps to isolate highly confident positive and negative sentiments by treating ambiguous comments as neutral.

```
# Apply to DataFrame
df[['sentiment', 'score']] = df['clean_comment'].apply(
    lambda x: pd.Series(classify_sentiment(x))
)

# Save result
df.to_csv("labeled_youtube_comments.csv", index=False)
print("✅ Comments labeled with POSITIVE, NEGATIVE, and NEUTRAL.")
```

c. The resulting labels ('POSITIVE', 'NEGATIVE', 'NEUTRAL') were saved to a new CSV file, labeled_youtube_comments.csv, creating the dataset for the next stage.

2. Feature Engineering with TF-IDF

Before training the ML models, the text comments were converted into numerical feature vectors using the TfidfVectorizer from scikit-learn.

```
vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

- TF-IDF (Term Frequency-Inverse Document Frequency) was chosen because it weighs the importance of a word not only by its frequency in a single document but also by how rare it is across the entire corpus. This helps the model focus on more meaningful and discriminative words.
- The vectorizer was configured with max_features=5000 to limit the vocabulary to the top 5,000 most important terms, which helps control dimensionality and reduce noise.

3. Data Splitting

The labeled dataset was split into training and testing sets using scikit-learn's train_test_split function.

```
X_train, X_test, y_train, y_test = train_test_split(
    df['clean_comment'], df['label'], test_size=0.2, random_state=42, stratify=df['label']
)
```

- 80% of the data was allocated for training and 20% was reserved for testing.
- stratify=df['label'] was used to ensure that the distribution of 'POSITIVE', 'NEGATIVE' and 'NEUTRAL' labels was the same in both the training and testing sets. This is important for reliable evaluation, especially if the classes are imbalanced.

4. Model Fitting

Both the Multinomial Naive Bayes and Logistic Regression models were trained using the `.fit()` method on the TF-IDF vectors of the training data (`X_train_tfidf`) and their corresponding sentiment labels (`y_train`).

```
# Naïve Bayes
nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)

# Logistic Regression
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_tfidf, y_train)
```

3.3 Evaluation Strategy

To assess the performance of the trained models, an evaluation strategy was planned. This includes applying a custom prediction logic and using standard classification metrics.

1. Prediction with Thresholding

For the primary evaluation, a custom thresholding function (`apply_threshold`) was defined. When making predictions on the test set, the models' probability outputs (`predict_proba`) are used.

```
def apply_threshold(probs, threshold=0.70):
    preds = []
    for row in probs:
        max_prob = max(row)
        if max_prob < threshold:
            preds.append(2) # Neutral
        else:
            preds.append(row.argmax())
    return preds

# Step 7: Predictions with threshold
nb_probs = nb_model.predict_proba(X_test_tfidf)
lr_probs = lr_model.predict_proba(X_test_tfidf)

nb_preds = apply_threshold(nb_probs, threshold=0.70)
lr_preds = apply_threshold(lr_probs, threshold=0.70)
```

- a. If the highest probability for any class for a given comment is below a threshold of 0.70, the comment is classified as 'NEUTRAL'.

- b. Otherwise, it is assigned the class with the highest probability.
- c. This approach aims to reduce false positives by classifying low-confidence predictions as neutral, thereby increasing the reliability of the 'POSITIVE' and 'NEGATIVE' labels.

2. Quantitative Metrics

The performance of both models will be measured using a `classification_report` from `scikit-learn`, which provides the key metrics below for each class.

```
label_names = ['NEGATIVE', 'POSITIVE', 'NEUTRAL']

print("🔍 Naive Bayes Report (with threshold 0.70):\n", classification_report(y_test,
                                                                           nb_preds,
                                                                           target_names=label_names))

print("🔍 Logistic Regression Report (with threshold 0.70):\n", classification_report(y_test,
                                                                           lr_preds,
                                                                           target_names=label_names))
```

a. Precision

The ratio of correctly predicted positive observations to the total predicted positive observations.

b. Recall (Sensitivity)

The ratio of correctly predicted positive observations to all observations in the actual class.

c. F1-Score

The weighted average of Precision and Recall, which is a good measure of a model's overall accuracy.

3. Visual Evaluation

A Confusion Matrix will be generated for both models. This provides a visual representation of the model's performance, showing the counts of true vs. predicted labels. It is particularly useful for identifying where the model gets confused such as

misclassifying 'NEUTRAL' as 'POSITIVE'.

```
# Naïve Bayes Confusion Matrix
nb_cm = confusion_matrix(y_test, nb_preds)
nb_disp = ConfusionMatrixDisplay(confusion_matrix=nb_cm,
                                  display_labels=label_names)
nb_disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Naive Bayes (Threshold 0.70)")
plt.show()

# Logistic Regression Confusion Matrix
lr_cm = confusion_matrix(y_test, lr_preds)
lr_disp = ConfusionMatrixDisplay(confusion_matrix=lr_cm,
                                  display_labels=label_names)
lr_disp.plot(cmap='Greens')
plt.title("Confusion Matrix - Logistic Regression (Threshold 0.70)")
plt.show()
```

4.0 Apache System Architecture

4.1 Component Roles and Configuration

The entire system is defined in a docker-compose.yml file, which has the services below.

1. **Apache Zookeeper (confluentinc/cp-zookeeper:7.6.1)**

Acts as a centralized coordination service for the distributed components, primarily managing the state of the Apache Kafka cluster. It is a critical prerequisite for Kafka.

2. **Apache Kafka (confluentinc/cp-kafka:7.6.1)**

Serves as the high-throughput, distributed messaging system or "data backbone" of the pipeline. It decouples the data source (producer) from the data processor (consumer).

- a. Implementation in project

A Python script, producer.py, continuously fetches comments from YouTube using the youtube-comment-downloader library and publishes them as JSON messages to a Kafka topic named youtube-comments. This creates a live stream of data for analysis.

3. **Apache Spark (bitnami/spark:3.4.2)**

The central processing engine of the pipeline. It is responsible for consuming data from Kafka in real-time, applying transformations and running the sentiment analysis model.

- a. Implementation in project

The spark_stream.py script utilizes Spark Structured Streaming. It connects to the Kafka youtube-comments topic, reads the data stream and performs the operations below.

- i. Data Cleaning

Applies the same text cleaning logic developed in the preprocessing stage.

- ii. Model Inference

Loads the pre-trained Logistic Regression model (lr_sentiment_model.pkl) and the TfidfVectorizer (tfidf_vectorizer.pkl). It uses these to predict the sentiment of each incoming comment. The models are broadcasted to Spark's worker nodes for efficiency.

iii. **Data Sinking**

Writes the final, enriched data (original comment, cleaned text, predicted sentiment and other metadata) directly to Elasticsearch.

4. **Elasticsearch (docker.elastic.co/elasticsearch:8.13.4)**

A distributed search and analytics engine used as the primary data store for the processed results. Its powerful indexing capabilities allow for fast querying and aggregation, which is ideal for powering dashboards.

a. **Implementation in project**

It receives the structured data stream from Spark and stores it in an index named youtube-comments6-index.

5. **Kibana (docker.elastic.co/kibana:8.13.4)**

The visualization layer of the stack. It provides a web-based interface for exploring and visualizing the data stored in Elasticsearch.

a. **Implementation in project**

Kibana connects to the Elasticsearch instance. This allows the creation of real-time dashboards that can display sentiment trends over time, show the distribution of positive/negative/neutral comments and enable interactive exploration of the results.

4.2 Workflow Diagram and Data Flow

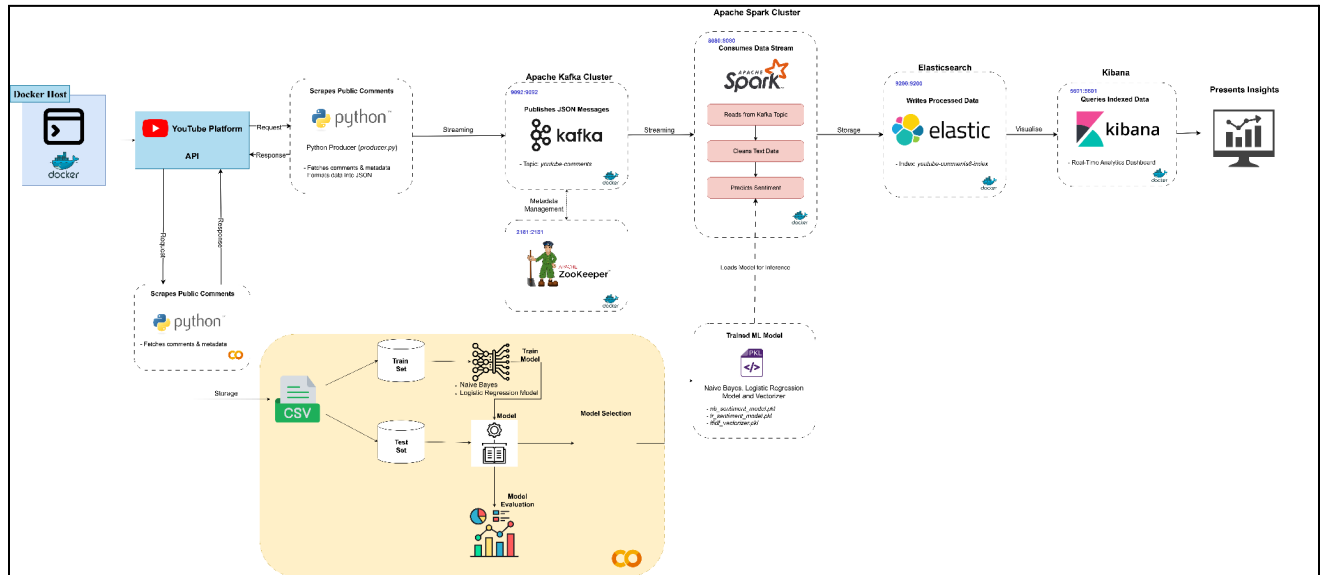


Figure 4.2.1: Workflow Diagram

Higher Resolution Image: Workflow Diagram.png

Step-by-Step Data Flow:

1. Data Ingestion

The producer.py script acts as the Kafka Producer. It continuously scrapes new comments from specific YouTube videos. Each comment, along with metadata like video_id and title, is packaged as a JSON object.

2. Data Streaming

The JSON object is sent to the Apache Kafka broker and published to the youtube-comments topic. Kafka buffers these messages, making them available for consumption.

3. Real-Time Consumption & Processing

The Apache Spark Structured Streaming job (spark_stream.py) acts as the Kafka Consumer. It connects to the youtube-comments topic and reads the data in micro-batches as a continuous stream.

4. Transformation and Analysis

For each record in the stream, Spark performs the actions below.

- a. The raw comment text is cleaned.
- b. The cleaned text is vectorized using the pre-loaded TfidfVectorizer (tfidf_vectorizer.pkl).
- c. The sentiment is predicted using the pre-trained Logistic Regression model (lr_sentiment_model.pkl).
- d. A new, structured record containing the original data and the predicted sentiment is created.

5. **Data Persistence**

The enriched data from Spark is streamed to Elasticsearch. Elasticsearch indexes this data, making it immediately available for searching and analysis.

6. **Visualization and Insight**

Kibana is used to query the youtube-comments6-index in Elasticsearch. Users can build and view dashboards that visualize the sentiment analysis results in real time, providing immediate insights into public opinion regarding "Hari Raya" on YouTube.

5.0 Analysis & Results

5.1 Key Findings

After setting up the real-time sentiment analysis system in Docker containers that run Apache Kafka, Apache Spark, Elasticsearch and Kibana together, the system had successfully collected, processed, streamed and visualized a total of 6829 data from 7 different Hari Raya-related YouTube videos. The videos included popular artists like Siti Nurhaliza, Iman Troye, Aisha Retno, and more. These videos were chosen because they had high view counts and were related to the Hari Raya celebration. This system helped us understand how people feel about Hari Raya content shared online in real-time.

To begin, we collect real-time comments from selected YouTube videos in `producer.py`. For every video, we collect information like video title, publish date, number of views and likes using the `yt_dlp` library. To get the comments, we used another tool called `youtube-comment-downloader`. This data collection will collect real-time data, which means it will keep running and collect new data on YouTube. Each piece of data collected will then be packed into a small message in JSON format and then sent into Kafka to handle large amounts of streaming data. Each comment was sent to a specific topic in Kafka called `youtube-comments`.

Next, we will run `spark_stream.py` to run Apache Spark to read data from Kafka. Spark is also running in real-time, which will keep reading new data from Kafka. In Spark, it will process the data, such as cleaning the comments by removing links, special characters and spaces. After cleaning the text, each comment was passed to a Logistic Regression sentiment analysis model that we trained earlier using TF-IDF features. This model is to predict whether the comments are positive, neutral or negative for sentiment analysis.

The final record is then stored in an index called `youtube-comments6-index` in Elasticsearch. Elasticsearch is a database designed for fast searching and analysis. Then we can open `localhost:5061` to open our Kibana to create a visualized dashboard that shows the result in an interactive way. Kibana will get data that is stored in Elasticsearch for visualization.

In Kibana, we had a dashboard that displayed the total number of comments collected, the number of different videos analyzed, and a pie chart showing the sentiment analysis breakdown. We also included a word cloud showing the most common words used in the comments, a table with video details like video title, publish_at, viewer, likes and number of comments and a graph that compares sentiment across videos. This visualization allows us to observe which video has the most positive sentiment or negative sentiment and how people used the word in comments on the Hari Raya video.

5.2 Visualization

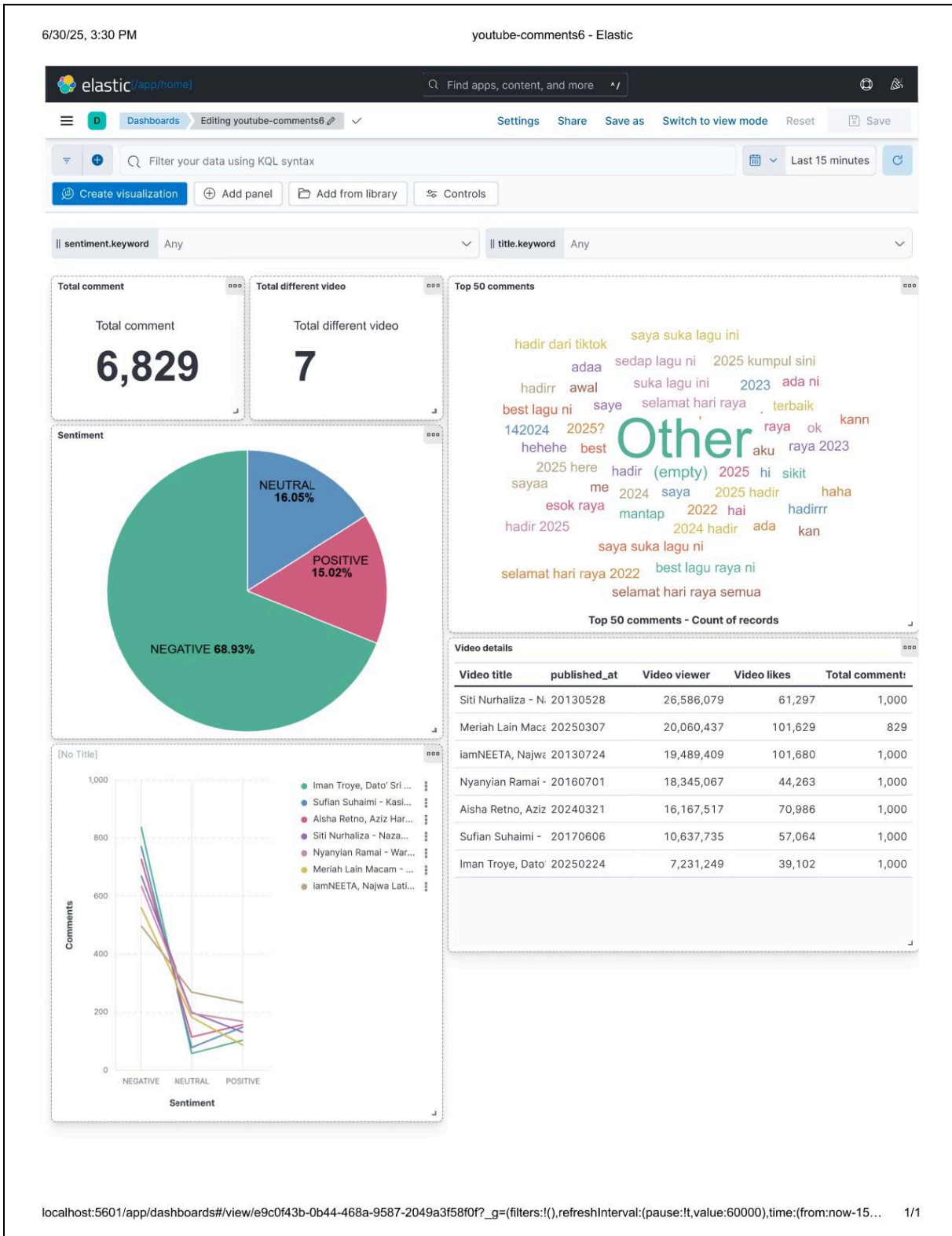


Figure 5.2.1: Sentiment Analysis Dashboard

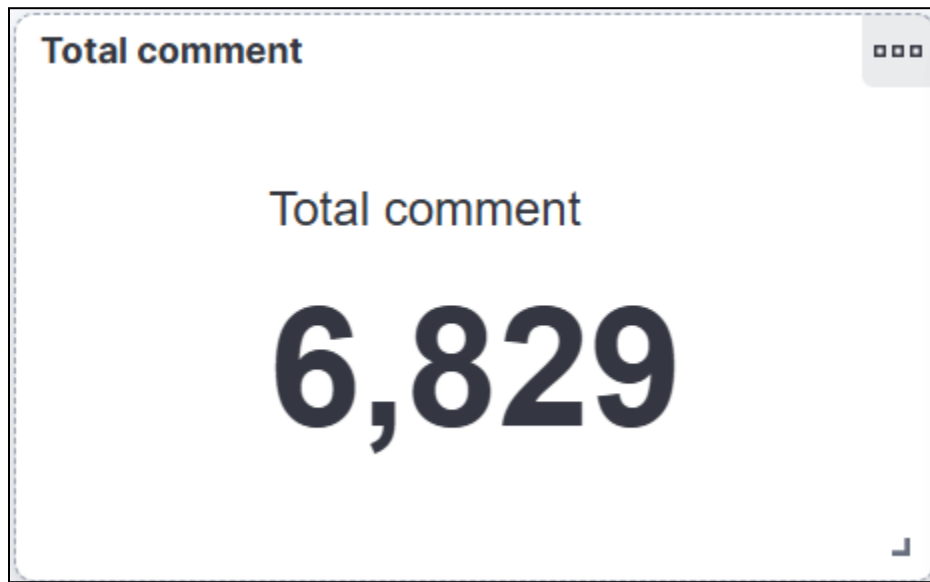


Figure 5.2.2: Total Comment

This dashboard shows the total number of comments that are collected by the real-time sentiment analysis into the Elasticsearch. It collected 6829 data from YouTube videos.

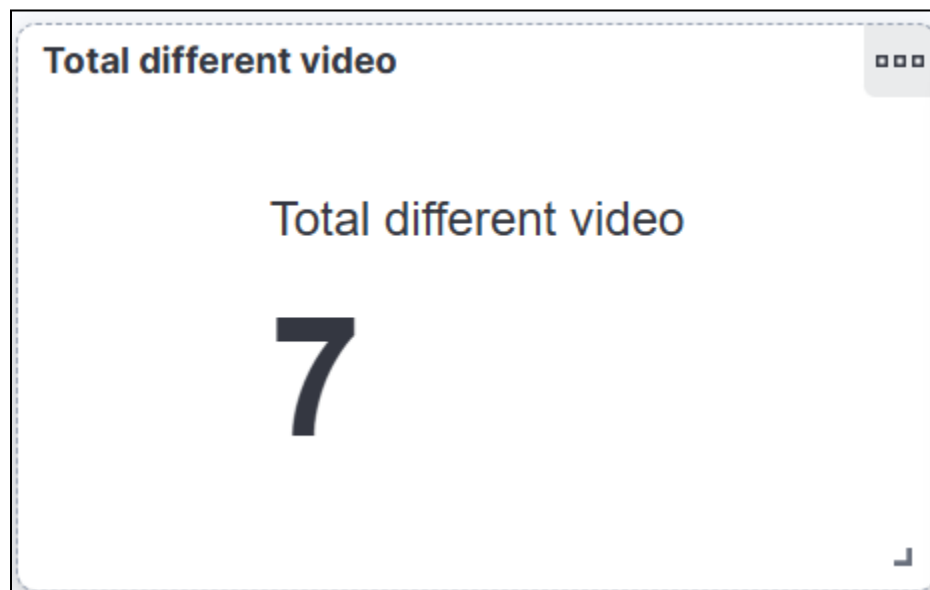


Figure 5.2.3: Total Different Video

This dashboard shows that the comments are collected from 7 different YouTube videos.

Video details				
Video title	published_at	Video viewer	Video likes	Total comments
Siti Nurhaliza - Nazam I	20130528	26,586,079	61,297	1,000
Meriah Lain Macam - H	20250307	20,060,437	101,629	829
iamNEETA, Najwa Latif,	20130724	19,489,409	101,680	1,000
Nyanyian Ramai - Warn	20160701	18,345,067	44,263	1,000
Aisha Retno, Aziz Harur	20240321	16,167,517	70,986	1,000
Sufian Suhaimi - Kasih I	20170606	10,637,735	57,064	1,000
Iman Troye, Dato' Sri Ali	20250224	7,231,249	39,102	1,000

Figure 5.2.4: Video Details Table

This dashboard shows the details of the YouTube video that was collected from YouTube in table format. The details of the video are video title, publication date, viewer count, likes, and comment count. From the table we can see that the video title with Siti Nurhaliza has the most viewers, which is around 26 million views. The video with the most likes is iamNEETA and Meriah Lain Macam, which has around 10K likes. Besides that, we can see that the number of comments every video consistently gets is around 1000 comments.

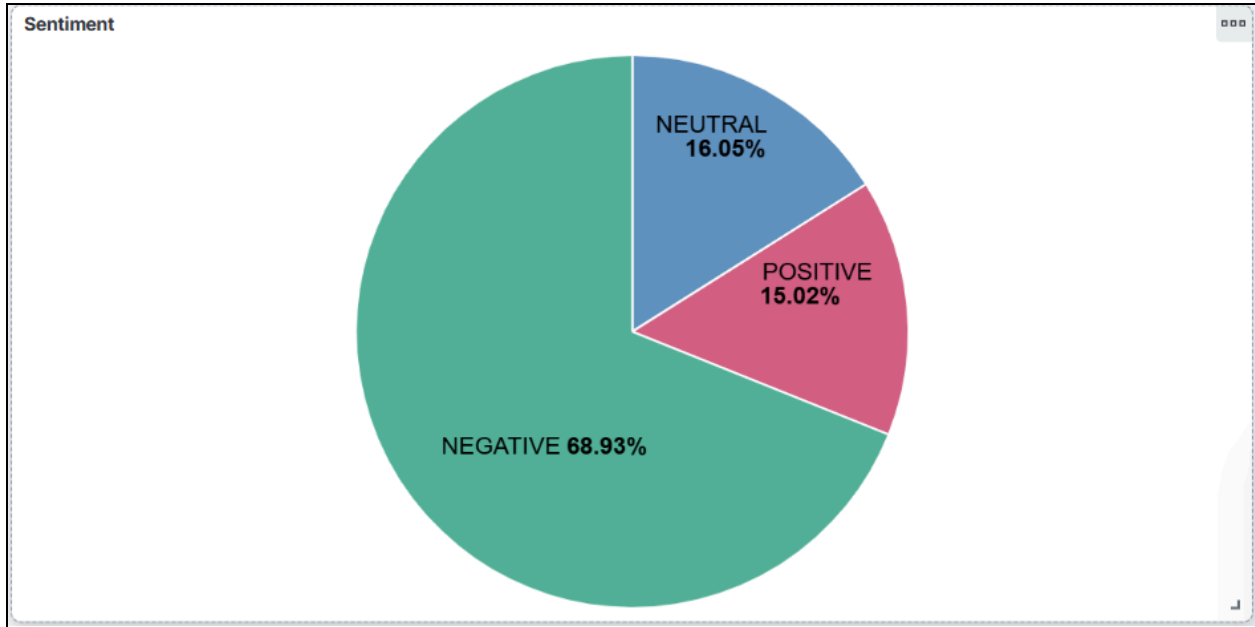


Figure 5.2.5: Pie Chart for Sentiment Analysis

This pie chart visualizes the sentiment analysis of all 6829 collected comments. The comments are categorized into 3 sentiment classification which are positive, negative and neutral. From the pie chart, we can see that the majority of comments are negative sentiment, which is 68.93% of the comments. About 16.05% of the comments are neutral and 15.02% of the comments are positive. As a result, this pie chart tells us that most people give bad comments and are dissatisfied across the analyzed video.



Figure 5.2.6: Word Cloud with Top 50 Comments

This word cloud visualizes the top 50 comments that will frequently exist in the 7 YouTube videos analyzed. The word size indicates the frequency of the word appearing in the comments. The larger the size of the word, the more often it appears in the comments of the YouTube video. From this word cloud, we can see that the comments appear to be primarily in Malay. Through this word cloud, we can see that the phrases that appear frequently in these 7 videos are “selamat hari raya,” “best lagu raya ni,” “sedap lagu ni,” “suka lagu ni,” and more. The YouTube videos we scrape are Hari Raya-related music videos so the comments are mostly relevant to the Hari Raya music comments.

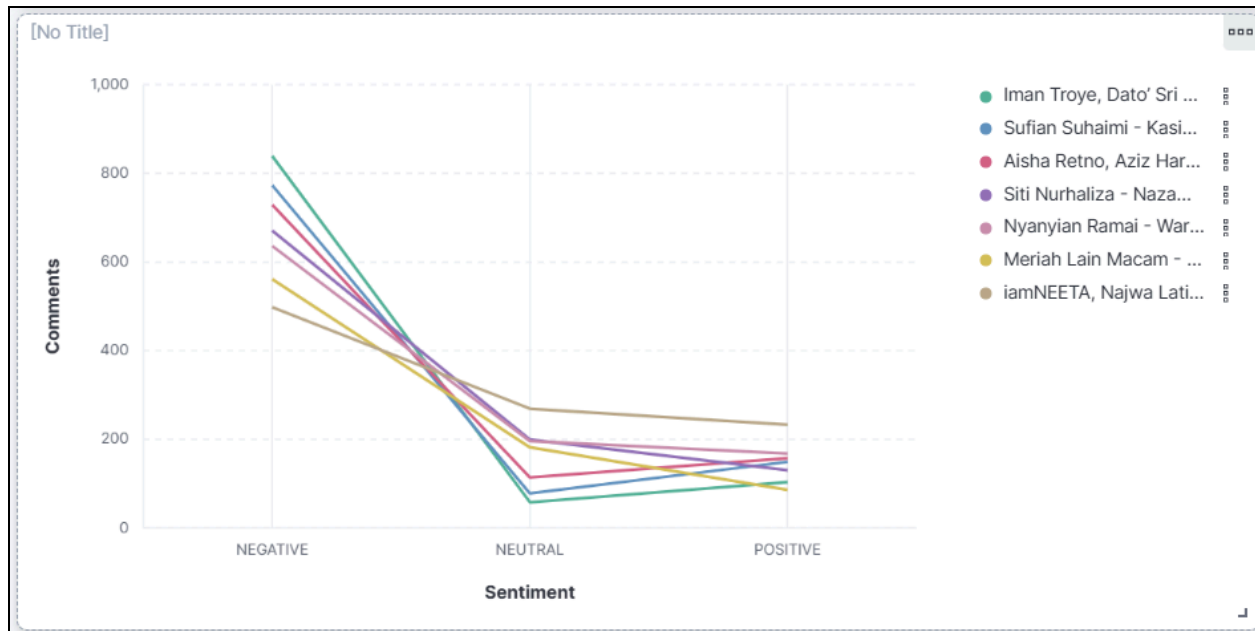


Figure 5.2.7: Comparison of Sentiment between Different Videos

This line graph displays sentiment analysis of YouTube comments across 7 Malaysian Hari Raya music videos. This line graph compares the sentiment analysis between the 7 YouTube videos. Based on the legend, we can see that it includes video from Iman Troye, Sufian Suhaimi, Aisha Retno, Siti Nurhaliza, Nyanyian Ramai, Meriah Lain Macam and iamNEETA. The comments are categorized into 3 sentiment classifications. The analysis reveals that the negative sentiment appears to be highest across all the videos, with comments ranging from 450 to 820. While the positive sentiment reveals itself to be the lowest across all the videos, with comments ranging from 100 to 250. Besides that, this line chart tells us that the Iman Troye YouTube video has a bad sentiment, as it contains the highest negative sentiment and lowest neutral and positive sentiment. This line chart shows that iamNEETA has a very good review due to it having the lowest negative sentiment and highest neutral and positive sentiment among all the videos.

6.0 Optimization & Comparison

6.1 Model Optimization

Improving the performance and usability of the sentiment analysis system needed us to implement several optimization:

- **Text Preprocessing:** Cleaned YouTube comments where punctuations, emojis, URLs, and also stop words were removed. This helped by reducing the noise and improving model learning quality.
- **Sentiment Labelling with Neutral Class:** A specific confidence threshold approach was introduced to handle neutral sentiments in the initial data label as the Hugging Face transformer used did not account for neutral sentiments. If the model's prediction probability was below 0.90 or 90%, the comment will be labeled as **NEUTRAL**. This implementation improved the accuracy for comments that were ambiguous or lacked sentiment cues.
- **Efficient Feature Extraction:** Both models used **TF-IDF vectorization** in order to convert text into numerical form. This approach captures the importance of terms while also reducing dimensionality.
- **Model Serialization:** Both the model that were trained and the TF-IDF vectorizer were saved using the library joblib, which allows quick loading and also reusability during classification, especially real-time classification.

6.2 Model Comparison

Two traditional ML models were trained for the sentiment analysis system:

- Naive Bayes
- Logistic Regression

These two models were trained using the transformer-labeled dataset (which includes the neutral class using the 0.90 or 90% threshold as mentioned previously in this documentation) using TF-IDF features.

Model	Accuracy	Precision	Recall	F1-Score	Notes
Logistic Regression	0.66	0.70	0.63	0.63	Balanced performance across all classes
Naive Bayes	0.62	0.69	0.58	0.58	Better recall for NEGATIVE class

Insights:

- Logistic Regression outperforms Naive Bayes overall as the performance in labelling across all classes is well distributed and balanced.
- Naive Bayes had a better recall for NEGATIVE comments but struggled with NEUTRAL.
- NEUTRAL sentiment was the hardest to classify for both models.

7.0 Conclusion & Future Work

In conclusion, this project successfully implemented a real-time sentiment system that is able to classify YouTube comments based on 3 main sentiments which are positive, neutral and negative. This is achieved by using Apache Kafka, PySpark, and also machine learning models such as Naive Bayes and Logistic Regression. By implementing a systematic data processing pipeline, intelligent labelling by using the Hugging Face transformer model with a confidence threshold, and also an optimized TF-IDF vectorization, the pipeline was able to process and also analyse the sentiments of comments efficiently.

The two classification models, Naive Bayes and Logistic Regression were trained and evaluated. Out of the two, Logistic Regression demonstrated plausible performance as it achieved a higher accuracy and better macro-averaged precision, recall and F1-score compared to the Naive Bayes model. However, what is obvious is that both models faced challenges in accurately classifying neutral sentiments, highlighting the difficulty of handling uncertain and ambiguous comments.

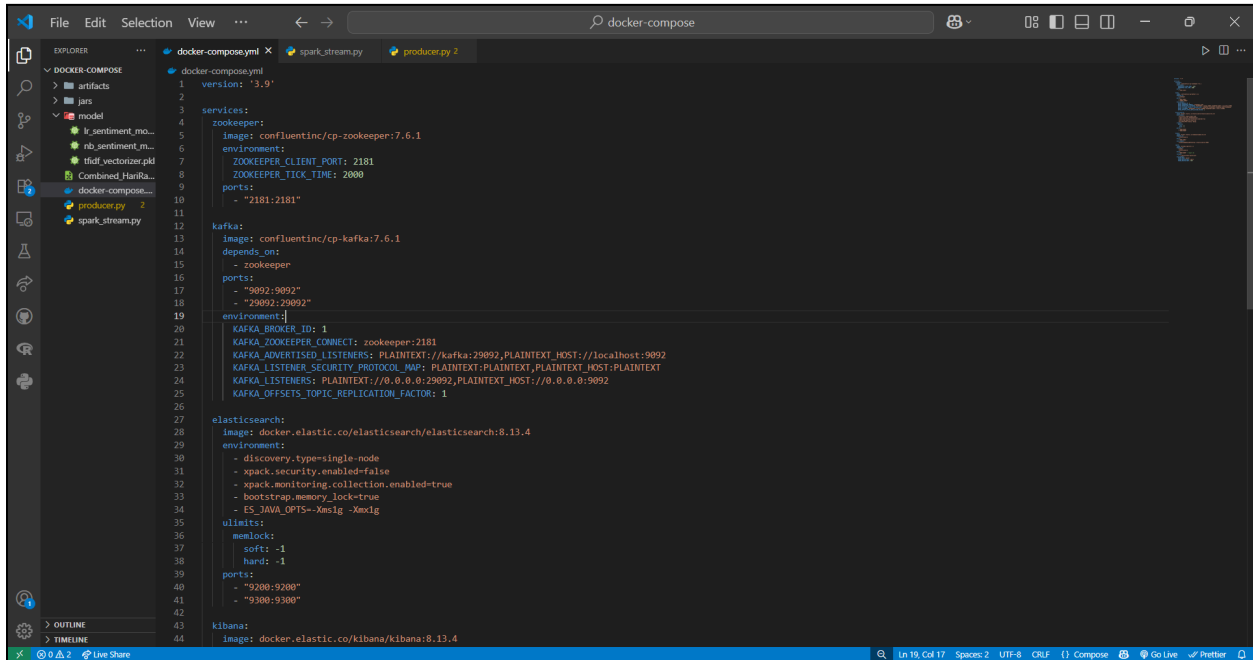
Overall, the system shows a promising look to be applied in real-world applications such as being used in social media monitoring, content moderation, and also brand sentiment tracking. With further enhancement and specific tuning, integrating with deep learning models, and also trained with a larger and richer volume of data, the accuracy and efficiency of the system can be further enhanced.

References

1. The Apache Software Foundation. (n.d.). *Apache Kafka*. Retrieved June 16, 2025, from <https://kafka.apache.org/>
2. The Apache Software Foundation. (n.d.). *Apache Spark*. Retrieved June 16, 2025, from <https://spark.apache.org/>
3. Docker, Inc. (n.d.). *Docker Documentation*. Retrieved June 16, 2025, from <https://docs.docker.com/>
4. Elastic. (n.d.). *Elasticsearch: The Official Distributed Search and Analytics Engine*. Retrieved June 16, 2025, from <https://www.elastic.co/elasticsearch/>
5. Elastic. (n.d.). *Kibana: Explore, Visualize, Discover Data*. Retrieved June 16, 2025, from <https://www.elastic.co/kibana/>

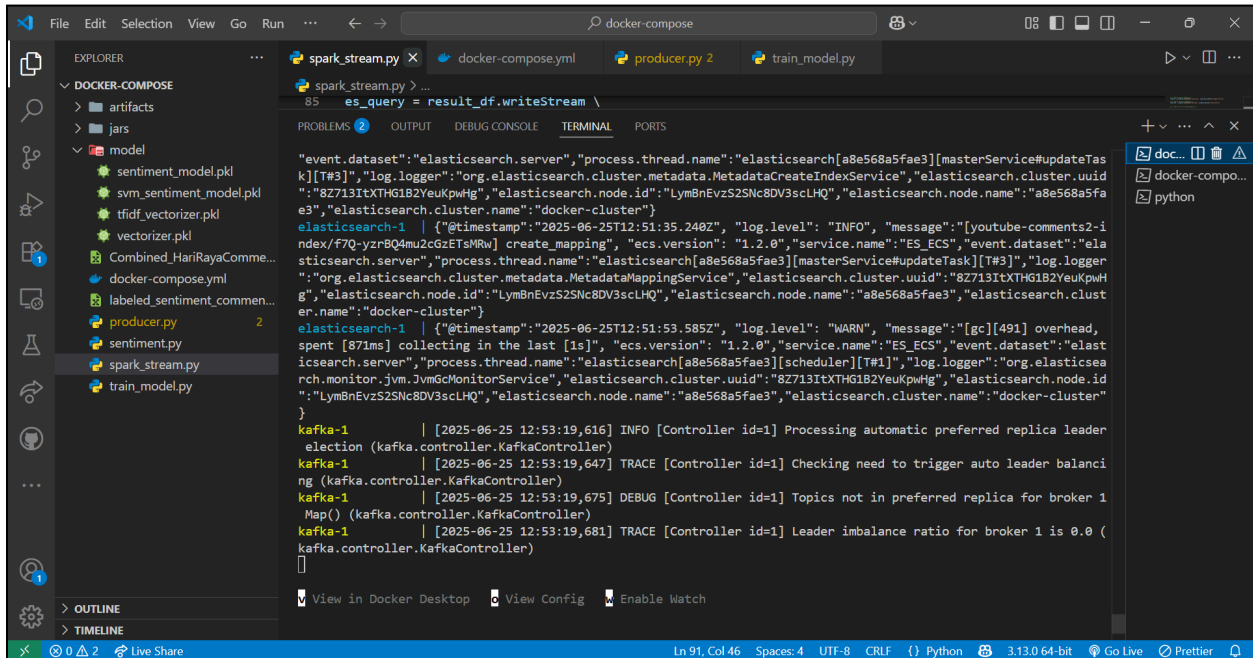
Appendix

1. Docker-compose.yml source code.



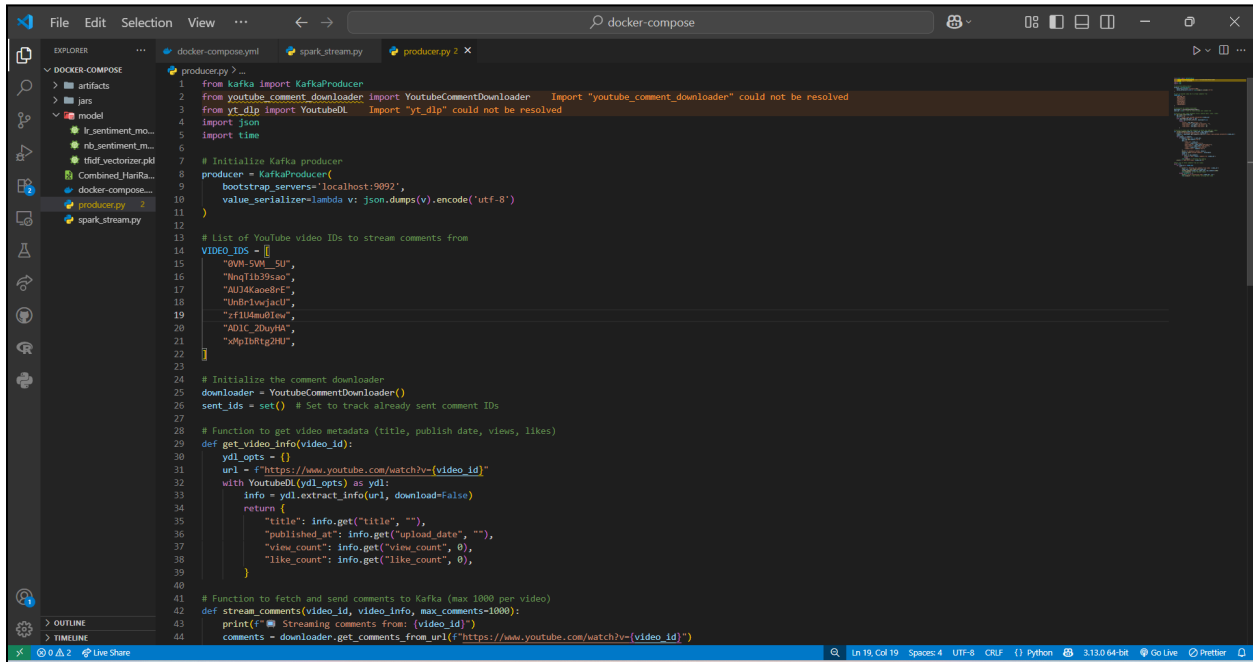
```
1 version: '3.9'
2
3 services:
4   zookeeper:
5     image: confluentinc/cp-zookeeper:7.6.1
6     environment:
7       ZOOKEEPER_CLIENT_PORT: 2181
8       ZOOKEEPER_TICK_TIME: 2000
9     ports:
10      - "2181:2181"
11
12   kafka:
13     image: confluentinc/cp-kafka:7.6.1
14     depends_on:
15       - zookeeper
16     ports:
17       - "9092:9092"
18       - "29092:29092"
19     environment:
20       KAFKA_BROKER_ID: 1
21       KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
22       KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
23       KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
24       KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:29092,PLAINTEXT_HOST://0.0.0.0:9092
25       KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
26
27   elasticsearch:
28     image: docker.elastic.co/elasticsearch/elasticsearch:8.13.4
29     environment:
30       - discovery.type=single-node
31       - xpack.security.enabled=false
32       - xpack.monitoring.collection.enabled=true
33       - bootstrap.memory_lock=true
34       - ES_JAVA_OPTS=-Xms1g -Xmx1g
35     ulimits:
36       memlock:
37         soft: -1
38         hard: -1
39     ports:
40       - "9200:9200"
41       - "9300:9300"
42
43   kibana:
44     image: docker.elastic.co/kibana/kibana:8.13.4
```

2. Running docker-compose.yml to run Docker containers that have tools like Apache Kafka, Apache Spark, Elasticsearch and Kibana.



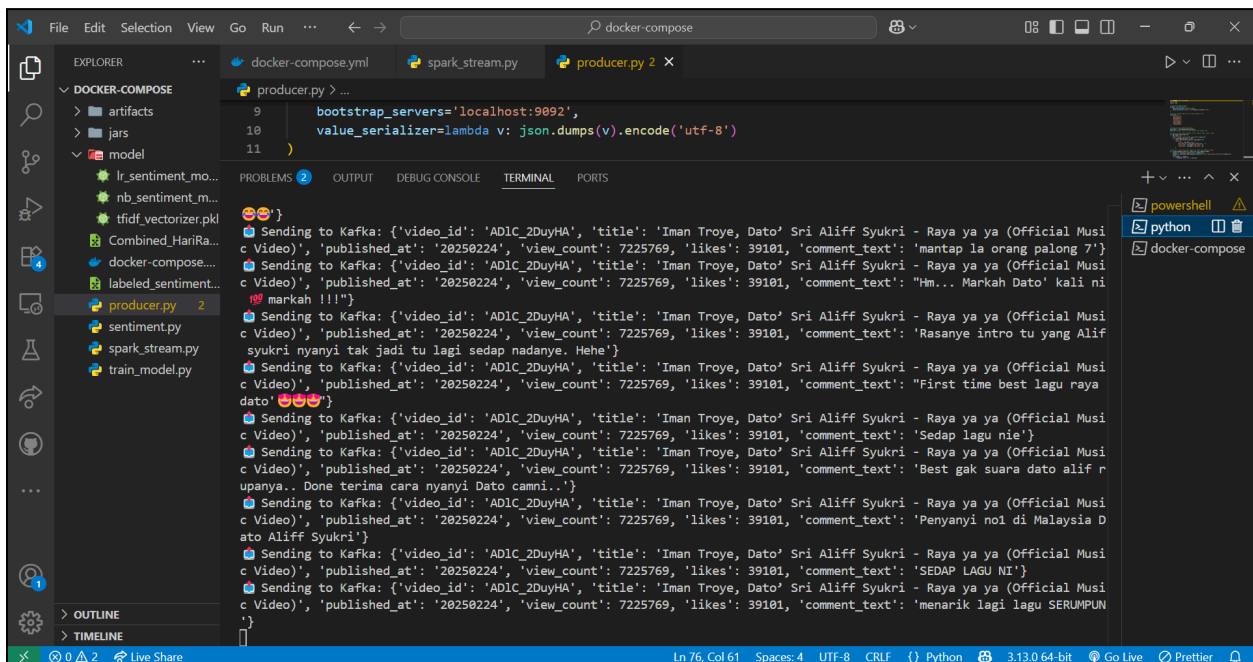
```
85 es_query = result_df.writeStream \
"event.dataset":"elasticsearch.server","process.thread.name":"elasticsearch[a8e568a5fae3][masterService#updateTask][T#3]","log.logger":"org.elasticsearch.cluster.metadata.MetadataCreateIndexService","elasticsearch.cluster.uuid":"827131tXTHG182YeuKpwHg","elasticsearch.node.id":"LymBnEvzS2Snc8DV3scLHQ","elasticsearch.node.name":"a8e568a5fae3","elasticsearch.cluster.name":"docker-cluster"}
elasticsearch-1 | {"@timestamp":"2025-06-25T12:51:35.240Z","log.level":"INFO","message":"[youtube-comments2-index/f7Q-yzrBQ4mu2cGzETsMRw] create_mapping","ecs.version":"1.2.0","service.name":"ES_ECS","event.dataset":"elasticsearch.server","process.thread.name":"elasticsearch[a8e568a5fae3][masterService#updateTask][T#3]","log.logger":"org.elasticsearch.cluster.metadata.MetadataMappingService","elasticsearch.cluster.uuid":"827131tXTHG182YeuKpwHg","elasticsearch.node.id":"LymBnEvzS2Snc8DV3scLHQ","elasticsearch.node.name":"a8e568a5fae3","elasticsearch.cluster.name":"docker-cluster"}
elasticsearch-1 | {"@timestamp":"2025-06-25T12:51:53.585Z","log.level":"WARN","message":"[gc][491] overhead, spent [871ms] collecting in the last [1s]","ecs.version":"1.2.0","service.name":"ES_ECS","event.dataset":"elasticsearch.server","process.thread.name":"elasticsearch[a8e568a5fae3][scheduler][T#1]","log.logger":"org.elasticsearch.monitor.jvm.JvmGcMonitorService","elasticsearch.cluster.uuid":"827131tXTHG182YeuKpwHg","elasticsearch.node.id":"LymBnEvzS2Snc8DV3scLHQ","elasticsearch.node.name":"a8e568a5fae3","elasticsearch.cluster.name":"docker-cluster"}
kafka-1 | [2025-06-25 12:53:19,616] INFO [Controller id=1] Processing automatic preferred replica leader election (kafka.controller.KafkaController)
kafka-1 | [2025-06-25 12:53:19,647] TRACE [Controller id=1] Checking need to trigger auto leader balancing (kafka.controller.KafkaController)
kafka-1 | [2025-06-25 12:53:19,675] DEBUG [Controller id=1] Topics not in preferred replica for broker 1 Map() (kafka.controller.KafkaController)
kafka-1 | [2025-06-25 12:53:19,681] TRACE [Controller id=1] Leader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
[]
View in Docker Desktop View Config Enable Watch
```


3. producer.py source code.



```
1 from kafka import KafkaProducer
2 from youtube_comment_downloader import YoutubeCommentDownloader
3 from yt_dlp import YoutubeDL
4 import json
5 import time
6
7 # Initialize Kafka producer
8 producer = KafkaProducer(
9     bootstrap_servers='localhost:9092',
10     value_serializer=lambda v: json.dumps(v).encode('utf-8'))
11
12 # List of YouTube video IDs to stream comments from
13 VIDEO_IDS = [
14     "8VM-5VM_SU",
15     "Mnq1B39sao",
16     "AD1C2DuyHA",
17     "Ud1r1wjacl",
18     "zfl1dau0Lew",
19     "AD1C2DuyHA",
20     "Mnq1B39sao",
21 ]
22
23 # Initialize the comment downloader
24 downloader = YoutubeCommentDownloader()
25 sent_ids = set() # Set to track already sent comment IDs
26
27 # Function to get video metadata (title, publish date, views, likes)
28 def get_video_info(video_id):
29     ydl_opts = {}
30     url = f"https://www.youtube.com/watch?v={video_id}"
31     with YoutubeDL(ydl_opts) as ydl:
32         info = ydl.extract_info(url, download=False)
33     return {
34         "title": info.get("title", ""),
35         "published_at": info.get("upload_date", ""),
36         "view_count": info.get("view_count", 0),
37         "like_count": info.get("like_count", 0),
38     }
39
40 # Function to fetch and send comments to Kafka (max 1000 per video)
41 def stream_comments(video_id, video_info, max_comments=1000):
42     print(f"Streaming comments from: {video_id}")
43     comments = downloader.get_comments_from_url(f"https://www.youtube.com/watch?v={video_id}")
```

4. Running [producer.py](#) to read and stream data from YouTube into Apache Kafka.

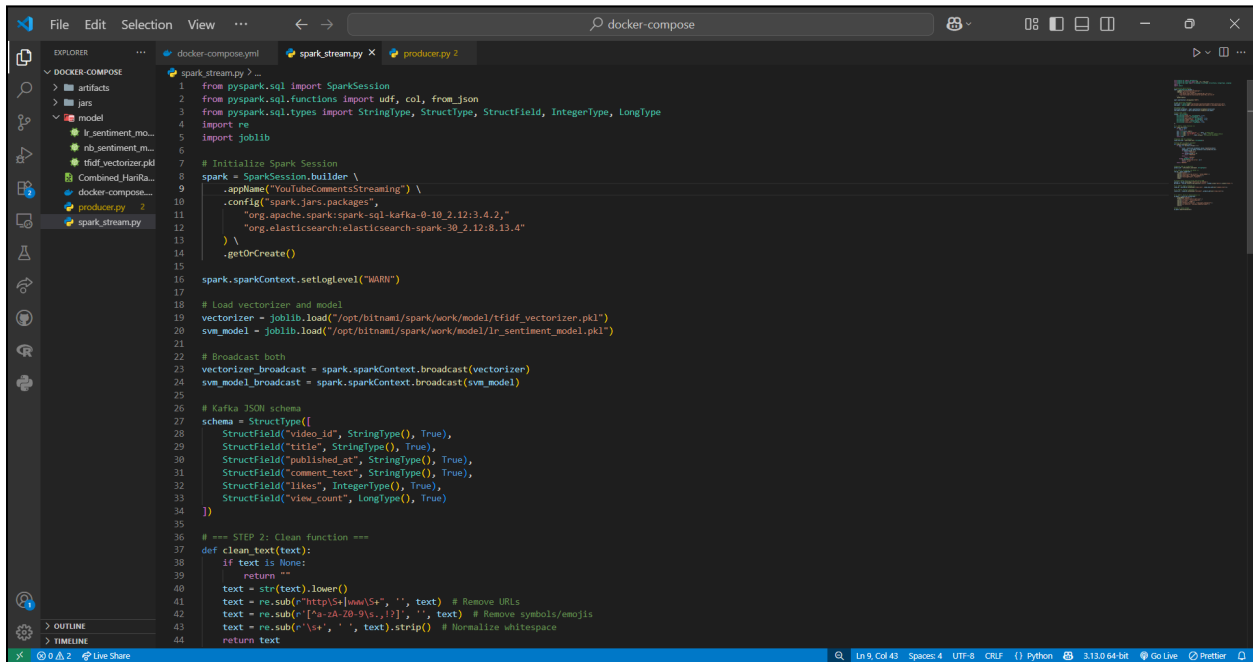


```
9     bootstrap_servers='localhost:9092',
10     value_serializer=lambda v: json.dumps(v).encode('utf-8'))
11 ]
12
13 # Initialize the comment downloader
14 downloader = YoutubeCommentDownloader()
15 sent_ids = set() # Set to track already sent comment IDs
16
17 # Function to get video metadata (title, publish date, views, likes)
18 def get_video_info(video_id):
19     ydl_opts = {}
20     url = f"https://www.youtube.com/watch?v={video_id}"
21     with YoutubeDL(ydl_opts) as ydl:
22         info = ydl.extract_info(url, download=False)
23     return {
24         "title": info.get("title", ""),
25         "published_at": info.get("upload_date", ""),
26         "view_count": info.get("view_count", 0),
27         "like_count": info.get("like_count", 0),
28     }
29
30 # Function to fetch and send comments to Kafka (max 1000 per video)
31 def stream_comments(video_id, video_info, max_comments=1000):
32     print(f"Streaming comments from: {video_id}")
33     comments = downloader.get_comments_from_url(f"https://www.youtube.com/watch?v={video_id}")
```

Terminal Output:

```
{
  "video_id": "AD1C2DuyHA",
  "title": "Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)",
  "published_at": "20250224",
  "view_count": 7225769,
  "likes": 39101,
  "comment_text": "mantap la orang palong 7"}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'Hm... Markah Dato' kali ni markah !!!'}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'Rasanya intro tu yang Alif syukri nyanyi tak jadi tu lagi sedih nadanye. Hehe!'}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'First time best lagu raya dato' }
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'Sedap lagu nie'}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'Best gak suara dato alif r upanya.. Done terima cara nyanyi Dato camli..'}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'Penyanyi no1 di Malaysia D ato Aliff Syukri'}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'SEDAP LAGU NI'}
Sending to Kafka: {'video_id': 'AD1C2DuyHA', 'title': 'Iman Troye, Dato' Sri Aliff Syukri - Raya ya ya (Official Music Video)', 'published_at': '20250224', 'view_count': 7225769, 'likes': 39101, 'comment_text': 'menarik lagi lagu SERUMPUN'}
```

5. `spark_stream.py` source code.



- Running `spark_stream.py` to consume data from Kafka into Spark and preprocess the data in Spark, then save it into Elasticsearch.

