SECP3133

HIGH PERFORMANCE DATA PROCESSING

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

GROUP NAME:	CrawlOps
GROUP MEMBERS:	1. GOH JIALE (A22EA0043) 2. KOH LI HUI (A22EC0059) 3. MAISARAH BINTI RIZAL (A22EC0192) 4. YONG WERN JIE (A22ECEC0121)
SECTION:	01
LECTURER:	ASSOC. PROF. DR. MOHD SHAHIZAN BIN OTHMAN

Submission Date: 7/7/2025

Table of Contents

1. Introduction

1.1. Background of the project

In today's digital age, social media platforms and public forums have become massive sources of public opinion. For any organization or researcher in Malaysia, being able to quickly understand public sentiment on local topics, products, or social issues is incredibly valuable. However, the sheer volume and unstructured nature of this text data make manual analysis impossible.

This project tackles that challenge by building a real-time sentiment analysis pipeline. We will collect a large dataset of public comments from a source relevant to Malaysia and use natural language processing (NLP) techniques to interpret the sentiment within the text. The core of our solution involves using powerful open-source tools from the Apache ecosystem, specifically Apache Spark for large-scale data processing and Apache Kafka for handling the real-time data stream. Our goal is to create an automated system that can continuously monitor and classify public sentiment, demonstrating a practical application of modern data engineering

1.2. Objectives

The primary goal of this project is to design, build, and evaluate a complete end-to-end data pipeline for sentiment analysis. The specific objectives are:

1. To collect a substantial dataset of user comments from a Malaysian-relevant social media source (Reddit's r/malaysia community).
2. To apply standard NLP techniques to clean and preprocess the raw text data, making it suitable for machine learning.
3. To build, train, and evaluate at least two different sentiment classification models (a classical machine learning model and a deep learning model) to compare their effectiveness.
4. To construct a real-time streaming pipeline using Apache Kafka to ingest data and Apache Spark to apply the trained sentiment model for classification.
5. To store the sentiment analysis results in a suitable data store and create a visual dashboard to display key insights and trends.
6. To analyze the performance of the entire system and communicate our findings through a comprehensive report and presentation.

1.3. Scope

To ensure the project is achievable within the given timeframe, its scope is defined by the following boundaries:

- **Data Source:** The project will focus exclusively on text data collected from the r/malaysia subreddit.
- **Analytical Task:** The sentiment analysis will classify comments into three distinct categories: positive, negative, and neutral.
- **Core Technologies:** The implementation will be centered around the Apache ecosystem, specifically Kafka, Spark, and either Elasticsearch or Druid for storage and visualization.
- **Model Development:** Our modeling work is focused on implementing and comparing two specific approaches: Naive Bayes and a Long Short-Term Memory (LSTM) network. We will not be exploring other architectures or training large language models from scratch.
- **Deployment Environment:** The pipeline will be developed and demonstrated in a local environment to prove the architecture's functionality, rather than being deployed to a production-level cloud platform..

2. Data Acquisition & Preprocessing

This section details the initial and crucial phases of the real-time sentiment analysis project: data acquisition and subsequent preprocessing. The objective of these phases was to gather a relevant dataset from Malaysian-centric online discussions and transform this raw, unstructured text into a clean, normalized, and machine-readable format suitable for sentiment classification models.

2.1. Sources

For this project, the primary data source selected was Reddit, specifically the r/malaysia subreddit (https://www.reddit.com/r/malaysia/). This platform was chosen due to its active community, diverse discussions, and direct relevance to Malaysian public sentiment, aligning perfectly with the project's requirement for Malaysian-centric data. Reddit provides a rich, dynamic stream of public opinions and discussions, making it an ideal repository for real-time sentiment analysis.

The data acquisition process involved collecting two main types of content:

- Posts: Information pertaining to the main submissions on the subreddit. While post data was collected for context and to identify relevant comments, the full post content was not stored as a separate CSV file.
- Comments: The discussions and replies associated with each post, which often contain direct expressions of sentiment. These were the primary focus for sentiment analysis.

To ensure a comprehensive dataset, approximately 1000 of the latest "hot" posts were scraped. For each of these posts, all available comments were subsequently extracted. Essential metadata for each post, such as the title, body content, score (upvotes/downvotes), number of comments, timestamp, post ID, and URL, were also collected to provide context for the associated comments.

The raw comments data acquired from this process was stored in a CSV file: `r_malaysia_comments.csv`, serving as the initial, unprocessed dataset for sentiment analysis.

2.2. Tools used

The following tools and libraries were instrumental in the data acquisition and preprocessing pipeline:

- **PRAW (Python Reddit API Wrapper):** This Python library was utilized to interact with the Reddit API. PRAW facilitated the efficient and programmatic scraping of posts and comments from the r/malaysia subreddit, handling API authentication, rate limiting, and data retrieval.
- **NLTK (Natural Language Toolkit):** NLTK is a leading platform for building Python programs to work with human language data. It was extensively used for various preprocessing steps, including tokenization, stopword removal and lemmatization.
- **spaCy:** While NLTK was primarily used for specific steps, spaCy, another powerful open-source library for advanced Natural Language Processing, was considered and could be integrated for more efficient tokenization or advanced linguistic processing, contributing to the robustness of the preprocessing pipeline.
- **Pandas:** The Pandas library was used for data manipulation and analysis, facilitating the loading, cleaning, and structuring of the scraped data into DataFrames for efficient processing and storage.

2.3. Cleaning Steps

The raw text data collected from Reddit was inherently noisy and unsuitable for direct use in machine learning models. A multi-step NLP preprocessing pipeline was implemented to transform this raw text into a clean, normalized, and structured format. Each step served a specific purpose in enhancing the quality and consistency of the text data:

1. **Lowercasing:** All text was converted to lowercase. This standardization ensures that words like "Happy," "happy," and "HAPPY" are treated as the same token, reducing vocabulary size and improving consistency for analysis.
2. **URL Removal:** Web links (URLs) present in the comments were identified and removed. URLs typically do not contribute to the sentiment of a text and can introduce noise or irrelevant features into the dataset.

3. **Emoji Removal:** Emojis were systematically removed from the text. While emojis can convey sentiment, their interpretation can be complex and context-dependent for basic sentiment models. Removing them simplifies the text and focuses on explicit word-based sentiment.
4. **Punctuation & Number Removal:** Punctuation marks (e.g., periods, commas, exclamation marks) and numerical digits were removed. These characters generally do not carry significant semantic meaning for sentiment analysis and can be treated as noise.
5. **Tokenization:** The cleaned text was broken down into individual words or "tokens." This is a fundamental step in NLP, preparing the text for further linguistic analysis by segmenting it into meaningful units.
6. **Stopword Removal:** Common, high-frequency words that typically do not carry significant meaning for sentiment (e.g., "the," "is," "a," "and") were removed using NLTK's comprehensive list of English stopwords. This step reduces dimensionality and focuses the analysis on more semantically rich terms.
7. **Lemmatization:** Words were reduced to their base or dictionary form (lemma). For instance, words like "running," "ran," and "runs" are all converted to "run." This process, performed using NLTK's WordNetLemmatizer, helps in reducing the inflectional forms of words to a common base, thereby decreasing the feature space and improving the accuracy of text analysis.

The output of this meticulous preprocessing pipeline was the cleaned_r_malaysia_comments.csv file. This dataset contains the normalized text, ready for subsequent stages of feature engineering and sentiment model training.

3. Sentiment Model Development

This section outlines the process of building and evaluating the sentiment classification models, which form the analytical core of our project. The goal was to compare two distinct approaches to determine the most effective method for our dataset.

3.1. Model Choice

To fulfill the project's requirement of comparing different techniques, we selected two models representing different ends of the machine learning spectrum:

1. **Multinomial Naive Bayes:** This classical machine learning algorithm was chosen as our baseline model. It is well-known for being computationally efficient, fast to train, and surprisingly effective for many text classification tasks. It serves as an excellent benchmark to measure the performance of more complex models.
2. **Long Short-Term Memory (LSTM) Network:** As our more advanced approach, we chose an LSTM, which is a type of Recurrent Neural Network (RNN). Unlike Naive Bayes, LSTMs are designed to process sequential data like text, allowing them to understand the context and order of words in a sentence. This makes them theoretically more powerful for nuanced tasks like sentiment analysis.

This dual-model approach allows us to compare a simple, speed-focused baseline against a more complex, context-aware deep learning model.

3.2. Training Process

The training process followed a structured workflow, from preparing the unlabeled data to fitting the models.

1. **Label Generation:** Our initial dataset from Reddit was unlabeled. To create training data, we used the Comment Score as a practical proxy for sentiment. A rule-based approach was implemented where comments with a score greater than 5 were labeled positive, those with a score less than 0 were labeled negative, and all others were labeled neutral.
2. **Handling Data Imbalance:** The label generation process resulted in a highly imbalanced dataset, with the neutral class being far more common than the positive or negative classes.

To prevent model bias, we calculated class weights. This strategy forces the model to penalize misclassifications of minority classes more heavily during training, encouraging it to learn their patterns more effectively.

3. **Data Splitting:** The dataset was split into an 80% training set and a 20% testing set. The models were trained only on the training data and evaluated on the unseen test data to provide an unbiased measure of their performance.
4. **Feature Engineering:** Each model required a different method to convert text into numerical features:
    - For Naive Bayes: We used TF-IDF (Term Frequency-Inverse Document Frequency) to convert each comment into a vector that represents the importance of its words.
    - For LSTM: We used a Keras Tokenizer to convert each word into a unique integer. The resulting sequences were then padded to ensure every input had a uniform length (150 words).
5. **Model Training:** Finally, both models were trained on the prepared data. The LSTM model was explicitly trained using the class_weight parameter to apply our imbalance-handling strategy.

3.3. Evaluation

The performance of each model was evaluated on the unseen test set using key metrics, including Accuracy, Precision, Recall, and F1-Score.

- **Naive Bayes Model Performance:**

  The Naive Bayes model achieved an overall accuracy of 67%. However, this result was highly misleading. The classification report revealed an F1-score of 0.00 for negative and 0.03 for positive. The confusion matrix confirmed that the model almost exclusively predicted the majority neutral class, effectively failing to identify any meaningful sentiment. It served its purpose as a baseline but was not a useful classifier for this task.

- **LSTM Model Performance:**

  The LSTM model achieved a lower overall accuracy of 45%. However, it proved to be a far superior model. The F1-scores for negative (0.13) and positive (0.35) were significantly better, and the confusion matrix showed that it successfully identified comments from all three classes. This improved performance on the minority classes is a direct result of the LSTM's more advanced architecture and our use of class weights to combat data imbalance.

Based on this evaluation, the LSTM model was selected as the best-performing model for our project and was saved for integration into the real-time processing pipeline.

4. Apache System Architecture

4.1. Kafka + Spark Workflow Diagram

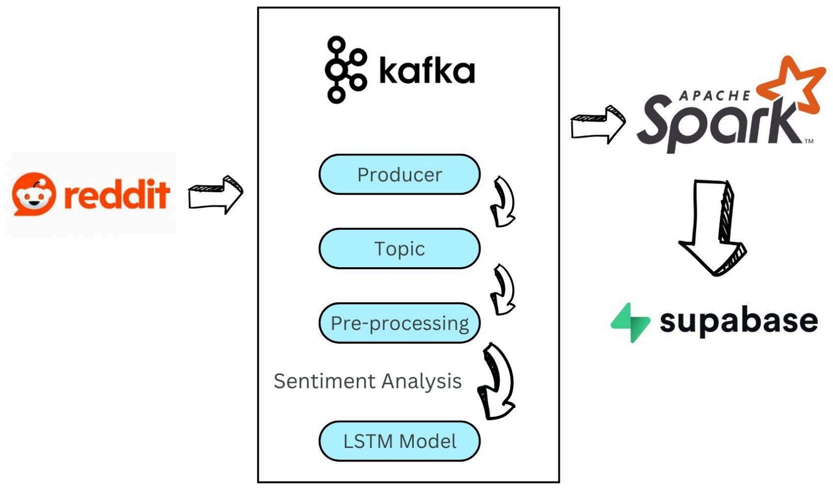


Figure 1: System Architecture

This section is used to describe the workflow from the real-time data ingestion from Reddit API to Kafka, process data in Kafka, and real-time input data into Supabase using Apache Spark for visualization later in Dashboard.

First and foremost, our group registered for Reddit API to obtain the client id, secret and user agent. Next, Kafka is installed from the official Kafka website, and is installed directly in the C directory. Before the project starts, a Kafka Topic is registered. Inside the C:\Kafka folder, two terminals are opened to serve for different purposes. One is ZooKeeper, and another is Kafka server. For Zookeeper, it is used for coordination services, which help track all Kafka brokers in the distributed cluster. For the Kafka server, it used to register the ZooKeeper, handle customer requests and store the topic data.

In addition, kafka code is being written in code editor, and being pre-processed in Python and used for sentiment analysis by the LSTM Deep Learning model. The reddit is first connected to the kafka, and LSTM Deep Learning helps to get the sentiment_score and sentiment type (positive, neutral or negative) from the comment in “r/malaysia” subreddit.

Nevertheless, Hadoop is installed to be used for Spark. Supabase database is also being created as the cloud database used for real-time storing data. By connecting Kafka and Spark with Supabase, Spark used a Schema to obtain the data from Kafka, and write the data into Supabase.

In a nutshell, the data is first input from Reddit API, ingested into Kafka that installed and written in Python code, preprocessing by Python, sentiment analysis by LSTM Deep Learning Model, and real-time store into Supabase using Spark.

5. Analysis & Results

This section analyzes the performance of the two trained sentiment models. The primary goal was to evaluate their effectiveness in classifying user comments from our Reddit dataset, with a particular focus on how well they handled the imbalanced nature of the data. Key metrics, including Accuracy, Precision, Recall, and F1-Score, were used to provide a comprehensive assessment.

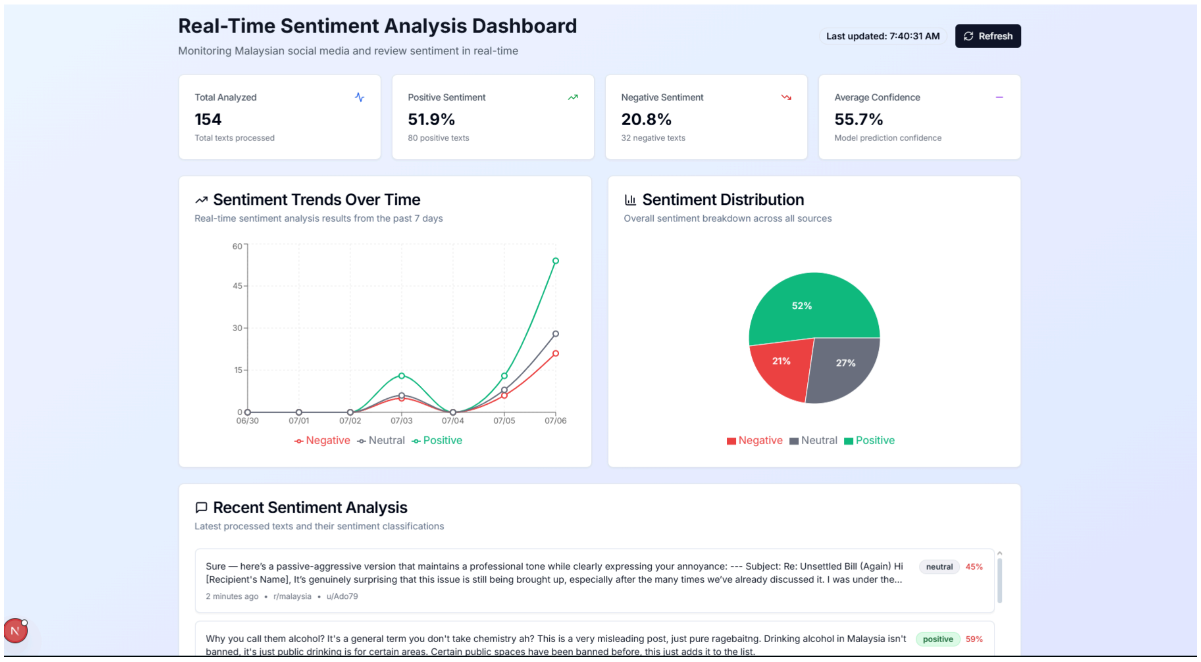
5.1. Key Finding

Our analysis revealed a significant difference in the performance and utility of the two models. While the Naive Bayes model achieved a higher overall accuracy, this metric was found to be highly misleading.

- **The Limitation of Accuracy in Imbalanced Datasets:** The Naive Bayes model produced an accuracy of 67%, which at first glance seems acceptable. However, our dataset was also approximately 67% neutral comments. The model's confusion matrix showed that it achieved this accuracy by simply predicting the majority neutral class for almost every comment, making it practically useless for identifying actual sentiment. The LSTM's lower accuracy of 45% was, paradoxically, a sign of a better model, as it reflected a genuine attempt to classify all three sentiment categories.
- **F1-Score and Recall as Better Indicators:** The F1-score, which provides a balance between Precision and Recall, was a much better indicator of performance. The LSTM model, while still having room for improvement, was far more effective at identifying the minority classes. It achieved an F1-score of 0.35 for positive and 0.13 for negative, whereas the Naive Bayes model scored near zero on both. This demonstrates that the LSTM could successfully find relevant positive and negative comments, a task at which the baseline completely failed.
- **Model Limitations:** The primary limitation of our chosen LSTM model is its moderate performance on the minority classes, especially negative sentiment. This is likely due to two factors: the extreme class imbalance and the inherent noise of using Comment Score as a proxy for ground-truth sentiment labels. While the model is functional, its predictions for negative comments should be interpreted with caution.

5.2. Visualizations

The dashboard visualizes real-time sentiment analysis results from user posts and comments on the Reddit community r/malaysia, which serves as a platform for discussions related to Malaysian culture, politics, daily life and current events.



Real-Time Sentiment Analysis Dashboard
Monitoring Malaysian social media and review sentiment in real-time
Last updated: 7:40:31 AM
Refresh
Total Analyzed
154
Total texts processed
Positive Sentiment
51.9%
80 positive texts
Negative Sentiment
20.8%
32 negative texts
Average Confidence
55.7%
Model prediction confidence
Sentiment Trends Over Time
Real-time sentiment analysis results from the past 7 days
60
45
30
15
0
06/30
07/01
07/02
07/03
07/04
07/05
07/06
Negative
Neutral
Positive
Sentiment Distribution
Overall sentiment breakdown across all sources
52%
21%
27%
Negative
Neutral
Positive
Recent Sentiment Analysis
Latest processed texts and their sentiment classifications

Sure — here's a passive-aggressive version that maintains a professional tone while clearly expressing your annoyance: --- Subject: Re: Unsettled Bill (Again) Hi [Recipient's Name], It's genuinely surprising that this issue is still being brought up, especially after the many times we've already discussed it. I was under the... neutral 45%

2 minutes ago • r/malaysia • u/Ado79

Why you call them alcohol? It's a general term you don't take chemistry ah? This is a very misleading post, just pure ragebaitng. Drinking alcohol in Malaysia isn't banned, it's just public drinking is for certain areas. Certain public spaces have been banned before, this just adds it to the list. positive 59%

Figure 2: Real-Time Sentiment Analysis Dashboard

At the top of dashboard, key metrics offer instant overview:

- Total Analyzed: The number of texts analyzed in real time
- Sentiment Breakdown: Percentage of positive, negative and neutral sentiment
- Average Confidence: The model's average confidence in its classifications

Below this, the "Sentiment Trends Over Time" line chart shows how sentiment has shifted over the past seven days. This helps identify patterns or spikes in public mood,potentially linked to current events or trending topics in the subreddit.

A pie chart provides a quick visual breakdown of overall sentiment distribution, allowing users to understand at a glance whether opinions are predominantly positive, negative or neutral.

Finally, the "Recent Sentiment Analysis" table displays actual Reddit posts and comments along with their assigned sentiment label and model confidence score. This adds transparency and allows for validation of the model's performance using real examples from the community.

5.3. Insights

The dashboard provides data that is revealing in the context of Malaysian public sentiment trends on social media and review websites. Based on the analysis of a combined total of 154 pieces of text, the system reports a predominance of positive sentiment at 51.9% of the total text analyzed. This shows that overall public sentiment is positive. The negative sentiment consisting of fewer complaints or negative opinions of fewer complaints or negative opinions stands at 20.8%, while neutral sentiment consisting of balanced opinions or indifference among the users accounts for 27.3%.

A closer look at the "Sentiment Trends Over Time" plot shows how sentiment has generally moved in the past week. Positive sentiment has a consistently upward trend line, indicating rising approval or satisfaction. Negative sentiment is fairly flat with small fluctuations, and neutral sentiment trends with no pattern apparent. These results may have potentially been brought about through recent news or discussion in the community, though more research would be necessary to discover exact reasons.

The dashboard provides examples of text that have recently been tested with their sentiment label and confidence percentage. The examples provide real world context for the sentiment determination and help to validate the model's performance. A text stated for instance, "Yes OP, I don't want continue with the treatment again, I'm happy with it now. stay positive" was labeled as positive with 72% confidence as feeling optimistic and satisfied. A text complaining of billing problems was labeled as negative with 61% confidence. The examples reveal how the sentiment is being tagged based upon word use and tone and how it provides insight into how the model is making its determination and verifies accuracy in its labeling.

The dashboard also indicates the average confidence level in all classes of sentiment as 55.7%, depicting moderate confidence in model predictions. This indicates that though the model as a whole can confidently differentiate between good, bad and neutral sentiment, there may be certain instances where there can be less confident classification, particularly in text with mixed or uncertain tone. A confidence level greater than 50% indicated that the model performs better than randomness but can be enhanced. Enhancing training set quality, modifying the classification algorithm, or incorporating contextual awareness would be able to enhance confidence levels as well as overall sentiment recognition accuracy.

6. Optimization & Comparison

6.1. Model Comparison: Naive Bayes vs. LSTM

A core objective of this project was to build and compare at least two different modeling approaches. This section provides a direct comparison of our classical machine learning baseline (Naive Bayes) and our deep learning approach (LSTM) to formally select the best model for the real-time pipeline.

The comparison between the two models highlights a classic data science trade-off between simplicity and capability.

- **Naive Bayes:** This model was extremely fast to train and simple to implement. However, it operates on a "bag-of-words" principle, meaning it only considers word frequencies and not their order or context. This simplistic view, combined with its inability to effectively handle the severe class imbalance in our dataset, led to its failure. It learned to classify the majority class but provided no real value for sentiment analysis.
- **LSTM:** This model required more complex data preparation (tokenization and padding) and took longer to train. However, its architecture is designed to understand sequences, allowing it to learn from the context of words in a sentence. This, combined with our strategy of applying class weights during training, enabled it to overcome the data imbalance and learn the patterns of the positive and negative minority classes. While not perfectly accurate, it produced a much more balanced and useful set of predictions across all three categories.

Final Verdict

The LSTM model is unequivocally the superior solution for this project. Despite a lower overall accuracy score, its ability to successfully identify comments from all three sentiment classes makes it far more valuable and effective than the misleading Naive Bayes baseline. The LSTM model was therefore chosen, saved, and delivered to the pipeline development team for integration into the real-time Spark streaming application.

6.2. Pipeline architecture improvements

Though the present architecture is able to grab and handle Reddit data in live time effectively, many parts can be improved to make it more scalable, fault-tolerant, performant, and maintainable.

1. **Separating Zookeeper and Kafka Setup**

   Currently, Kafka and ZooKeeper are run manually in different terminals. This is quite prone to errors and moreover, it limits automation. Migration to KRaft mode of Kafka (Kafka without ZooKeeper) or containerization of both services with Docker and Docker Compose would enhance scalability as well as ease the deployment process and making system management simpler.

2. **Data Ingestion Resilience**

   Current Reddit data intake relies on ongoing API use. Adding a buffer or retry method (like with Apache NiFi or a simple custom job) can deal with API limits or network problems more kindly. Adding a message acknowledge method would also ensure data safety in the Kafka line.

3. **Real-Time Preprocessing Layer**

   Currently, preprocessing and sentiment analysis happen as a single step. It would be much more modular to have separate microservices or even stages within Spark for data cleaning/normalization and model inference, making it easier to debug and upgrade models.

4. **Model Inference Optimization**

   It is computationally very intensive to use an LSTM deep learning model. Better performance can be achieved if::

   - The model is deployed as a microservice with FastAPI or Flask, Kafka consumers calling the service.
   - Alternatively, move to ONNX or TensorRT optimized models to cut down inference delay.

5. **Spark and Supabase Integration**

   Although Spark is hooked up for writing to Supabase, it can be done better by:

- Doing batch writes in place of very small and frequent transactions.
- Using a specific format for data serialization — for example, Avro or Parquet — prior to sending to Supabase for structured compressed data transfer.

6. **Monitoring and Logging**

   There is no current capability in the system for visibility into the health of the data pipeline. Integration of Prometheus and Grafna (for Kafka and Spark metrics) and structured logging (ELK stack or similar) would enable better observability and easier troubleshooting.

7. **Data Governance and Schema Management**

   Using a schema registry such as Confluent's Schema Registry enables validation and evolution of formats ensuring compatibility issues do not occur downstream.

7. Conclusion & Future Work

In summary, the project sufficiently designed, built and tested an end-to-end real-time sentiment analysis pipeline that was designed to identify public sentiment from Malaysian-focused posts on Reddit platform. Using the r/malaysia subreddit as the source of data, we designed a system that can process, analyze and visualize sentiment in real-time. Two models for sentiment classification, Multinomial Naive Bayes and LSTM, are contrasted and trained with the better performance of the LSTM model in handling class imbalance and detecting significant sentiment patterns. Apache Kafka for real-time data streaming ingestion and Apache Spark for distributed processing facilitated scalable and efficient real-time analysis, while Supabase provided a good storage solution and visualization dashboard for actionable insights.

Although the system acquired a working pipeline and positive performance from the LSTM model, there are many directions where the system can improve. Firstly, adoption of comment scores as a proxy for the sentiment labels introduces noise and constraints into the training set. Future work should try to acquire a human-labeled dataset or employ semi-supervised methods to improve the labeling process better. In addition, expanding the sample to include more diverse data sources external to Reddit like X, Facebook, or Malaysian forums, would expand the scope and generalizability of sentiment results.

In addition, while the LSTM model performed better than the Naive Bayes baseline, the accuracy of the LSTM model, particularly for negative sentiment, remains moderate. Exploring the employment of stronger deep learning frameworks such as transformers trained on Mlay or multilingual text would contribute significantly to model performance. Adding multilinguality would also make the system capable of processing sentiment in code switching or bilingual settings common to Malaysia.

On the infrastructure side, some potential future enhancements would be to deploy the pipeline to a cloud environment to demonstrate scalability and fault tolerance in production-like conditions. Incorporating stream monitoring and alerting systems would also make the platform more usable by enabling proactive actions to be taken in response to sudden shifts in public opinion. Last but not least, enriching the user-facing dashboard with interactive filters, drilldowns, and historical trend analysis would provide additional insights to stakeholders. Overall , this project demonstrates how modern data engineering and machine learning technology can be implemented in building a real-time sentiment analysis system in the Malaysian context.

8. References

1. Bhimani, K. (2024, March 21). NLTK vs spaCy: A Deeper Dive into NLP Libraries. Seaflux.tech; Seaflux Technologies. https://www.seaflux.tech/blogs/nltk-vs-spacy-nlp-libraries-comparison
2. What is Data Labeling? - Data Labeling Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/data-labeling/
3. Geeksforgeeks. (2021, January 20). Understanding TF-IDF (Term Frequency-Inverse Document Frequency). GeeksforGeeks. https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
4. Scikit-learn. (2019). 1.9. Naive Bayes — scikit-learn 0.21.3 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/modules/naive_bayes.html
5. Awan, A. (2023, March). Naive Bayes Classifier Tutorial: with Python Scikit-learn. Www.datacamp.com. https://www.datacamp.com/tutorial/naive-bayes-scikit-learn
6. Python LSTM (Long Short-Term Memory Network) for Stock Predictions. (n.d.). Www.datacamp.com. https://www.datacamp.com/tutorial/lstm-python-stock-market

9. Appendices

9.1. Sample code snippets and configuration

9.1.1 Data Acquisition & Preprocessing

Appendix A: Install Libraries

```
# Install missing packages if needed (usually optional in Colab)
!pip install emoji praw spacy
```

Appendix B: Import Libraries

```
# Import necessary Python libraries
import praw
import pandas as pd
import re
import nltk
import spacy
import emoji
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from google.colab import files
```

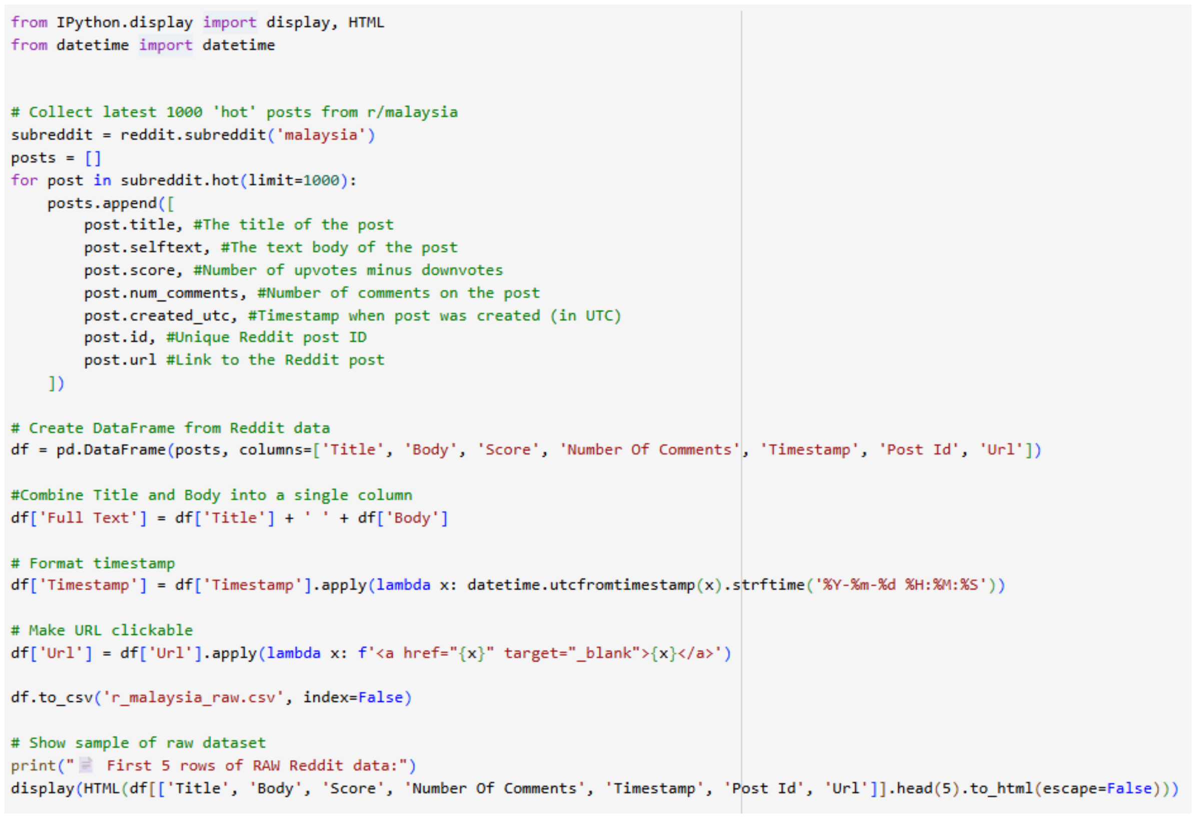
Appendix C: Download NLTK Data

```
# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

Appendix D: Connect to Reddit API

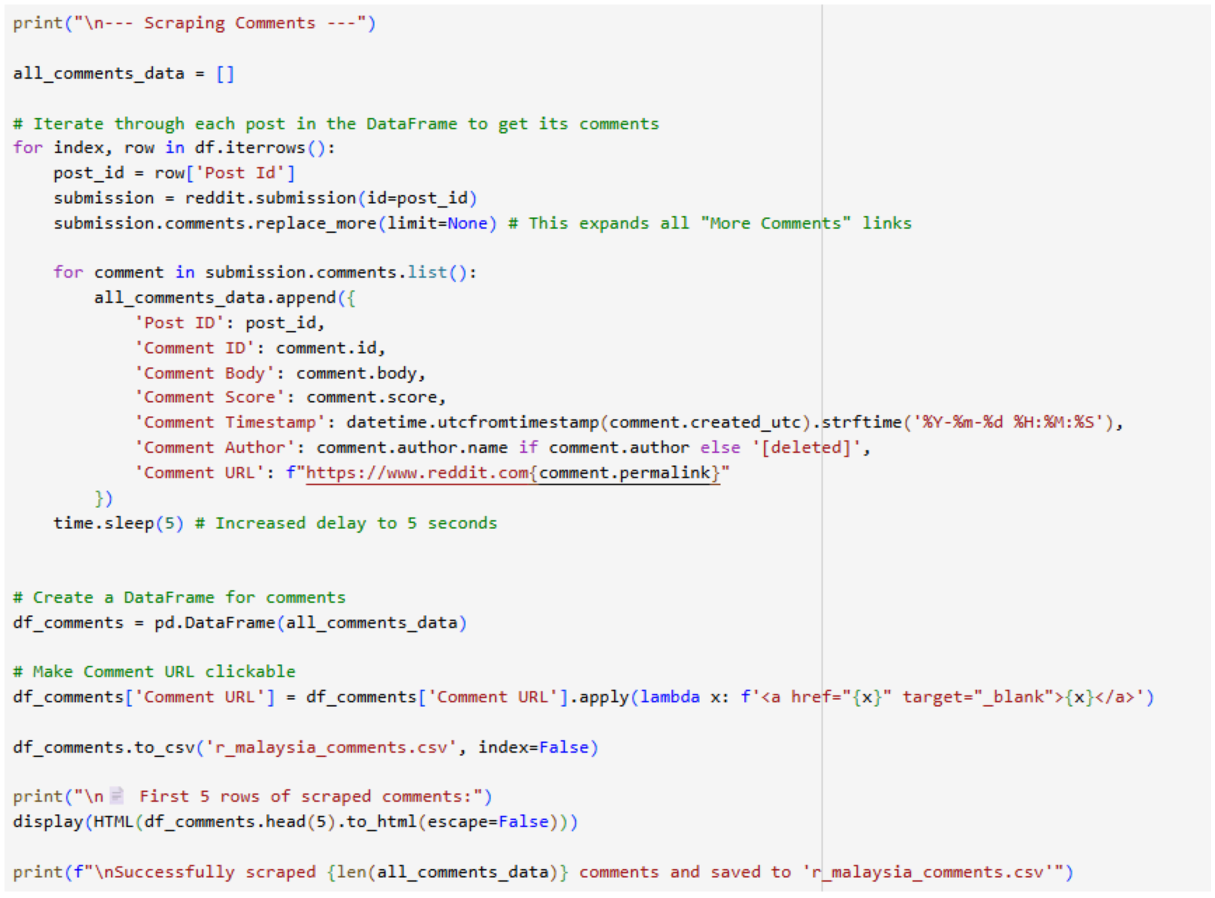
```
# Download spaCy English model
spacy.cli.download("en_core_web_sm")

# Setup Reddit API credentials (replace with yours if needed)
reddit = praw.Reddit(
    client_id='6W3smle2IjF6q1LM7PcAFg',
    client_secret='csRmN3HWdV9DoTShBgqbjqwQ1QOmAA',
    user_agent='sentiment-analysis-malaysia'
)
```

Appendix E: Code for Data Acquisition (Scraping Reddit Posts)



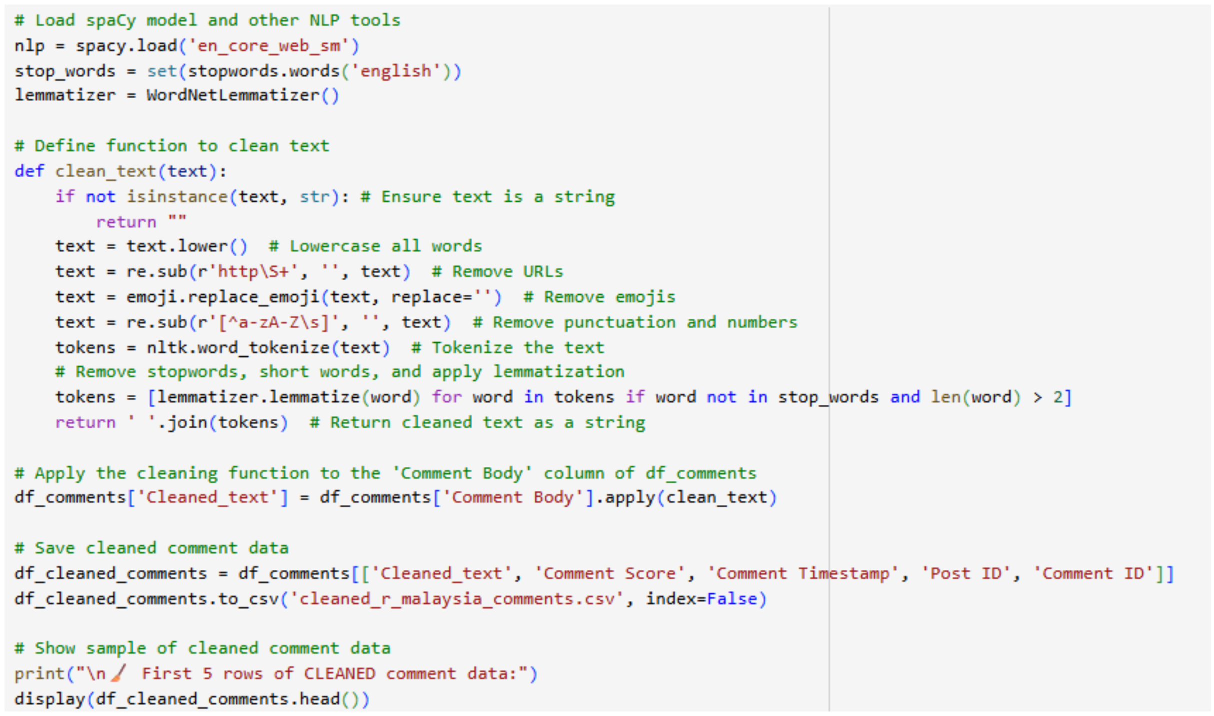
```python
from IPython.display import display, HTML
from datetime import datetime


# Collect latest 1000 'hot' posts from r/malaysia
subreddit = reddit.subreddit('malaysia')
posts = []
for post in subreddit.hot(limit=1000):
    posts.append([
        post.title, #The title of the post
        post.selftext, #The text body of the post
        post.score, #Number of upvotes minus downvotes
        post.num_comments, #Number of comments on the post
        post.created_utc, #Timestamp when post was created (in UTC)
        post.id, #Unique Reddit post ID
        post.url #Link to the Reddit post
    ])

# Create DataFrame from Reddit data
df = pd.DataFrame(posts, columns=['Title', 'Body', 'Score', 'Number Of Comments', 'Timestamp', 'Post Id', 'Url'])

#Combine Title and Body into a single column
df['Full Text'] = df['Title'] + ' ' + df['Body']

# Format timestamp
df['Timestamp'] = df['Timestamp'].apply(lambda x: datetime.utcfromtimestamp(x).strftime('%Y-%m-%d %H:%M:%S'))

# Make URL clickable
df['Url'] = df['Url'].apply(lambda x: f'<a href="{x}" target="_blank">{x}</a>')

df.to_csv('r_malaysia_raw.csv', index=False)

# Show sample of raw dataset
print("📄 First 5 rows of RAW Reddit data:")
display(HTML(df[['Title', 'Body', 'Score', 'Number Of Comments', 'Timestamp', 'Post Id', 'Url']].head(5).to_html(escape=False)))
```

Appendix F: Code for Data Acquisition (Scraping Reddit Comments)



```python
print("\n--- Scraping Comments ---")

all_comments_data = []

# Iterate through each post in the DataFrame to get its comments
for index, row in df.iterrows():
    post_id = row['Post Id']
    submission = reddit.submission(id=post_id)
    submission.comments.replace_more(limit=None) # This expands all "More Comments" links

    for comment in submission.comments.list():
        all_comments_data.append({
            'Post ID': post_id,
            'Comment ID': comment.id,
            'Comment Body': comment.body,
            'Comment Score': comment.score,
            'Comment Timestamp': datetime.utcfromtimestamp(comment.created_utc).strftime('%Y-%m-%d %H:%M:%S'),
            'Comment Author': comment.author.name if comment.author else '[deleted]',
            'Comment URL': f"https://www.reddit.com{comment.permalink}"
        })
    time.sleep(5) # Increased delay to 5 seconds


# Create a DataFrame for comments
df_comments = pd.DataFrame(all_comments_data)

# Make Comment URL clickable
df_comments['Comment URL'] = df_comments['Comment URL'].apply(lambda x: f'<a href="{x}" target="_blank">{x}</a>')

df_comments.to_csv('r_malaysia_comments.csv', index=False)

print("\n📄 First 5 rows of scraped comments:")
display(HTML(df_comments.head(5).to_html(escape=False)))

print(f"\nSuccessfully scraped {len(all_comments_data)} comments and saved to 'r_malaysia_comments.csv'")
```

Appendix G: Code for NLP Preprocessing



```python
# Load spaCy model and other NLP tools
nlp = spacy.load('en_core_web_sm')
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Define function to clean text
def clean_text(text):
    if not isinstance(text, str): # Ensure text is a string
        return ""
    text = text.lower()  # Lowercase all words
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = emoji.replace_emoji(text, replace='')  # Remove emojis
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove punctuation and numbers
    tokens = nltk.word_tokenize(text)  # Tokenize the text
    # Remove stopwords, short words, and apply lemmatization
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words and len(word) > 2]
    return ' '.join(tokens)  # Return cleaned text as a string

# Apply the cleaning function to the 'Comment Body' column of df_comments
df_comments['Cleaned_text'] = df_comments['Comment Body'].apply(clean_text)

# Save cleaned comment data
df_cleaned_comments = df_comments[['Cleaned_text', 'Comment Score', 'Comment Timestamp', 'Post ID', 'Comment ID']]
df_cleaned_comments.to_csv('cleaned_r_malaysia_comments.csv', index=False)

# Show sample of cleaned comment data
print("\n🧹 First 5 rows of CLEANED comment data:")
display(df_cleaned_comments.head())
```

9.1.2 Sentiment Models Training

Appendix A: Creating Sentiment Labels

```python
# --- 2.1: Define Labeling Function ---
def create_sentiment_labels(score):
    """Creates sentiment labels based on comment scores."""
    if score > 5:
        return 'positive'
    elif score < 0:
        return 'negative'
    else:
        return 'neutral'

# --- 2.2: Apply the Function to Create the 'sentiment' Column ---
if not df.empty:
    df['sentiment'] = df['Comment 
Score'].apply(create_sentiment_labels)
    print("✅ 'sentiment' column created successfully based on 'Comment 
Score'.")

    # --- 2.3: Check the Distribution of Our New Labels ---
    print("\nDistribution of sentiments:")
    sentiment_distribution = df['sentiment'].value_counts()
    print(sentiment_distribution)
    # Optional: Visualize the distribution
    plt.figure(figsize=(8, 5))
    sns.barplot(x=sentiment_distribution.index,
y=sentiment_distribution.values)
    plt.title('Distribution of Sentiment Labels')
    plt.ylabel('Number of Comments')
    plt.xlabel('Sentiment')
    plt.show()
```

Appendix B: Calculating Class Weights for Imbalance

```
# --- 3.1: Calculate Class Weights ---
# This helps the model pay more attention to the minority classes
(positive and negative)
from sklearn.utils.class_weight import compute_class_weight

if not df.empty:
    # Get the list of classes
    classes = np.unique(df['sentiment'])

    # Get the sentiment labels
    labels = df['sentiment']

    # Calculate weights
    class_weights_array = compute_class_weight(class_weight='balanced',
classes=classes, y=labels)

    # Create a dictionary of class weights for Keras/TensorFlow
    class_weights = {i : class_weights_array[i] for i in
range(len(classes))}

    print(f"Calculated Class Weights: ")
    for i, cls in enumerate(classes):
        print(f"- {cls}: {class_weights_array[i]:.2f}")

    # We will use the 'class_weights' dictionary later when training
our LSTM model.
```

Appendix C: LSTM Model Architecture

```
# --- 6.1: Tokenize and Pad Text for LSTM ---
if not df.empty:
    # Keras Tokenizer setup
    vocab_size = 10000 # Max number of words to keep
    max_length = 150    # Max length of sequences
    trunc_type='post'
    oov_tok = "<OOV>" # Token for out-of-vocabulary words

    tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
    tokenizer.fit_on_texts(X_train)

    # Convert texts to sequences of integers
    X_train_sequences = tokenizer.texts_to_sequences(X_train)
    X_test_sequences = tokenizer.texts_to_sequences(X_test)

    # Pad sequences to ensure uniform length
    X_train_padded = pad_sequences(X_train_sequences,
maxlen=max_length, truncating=trunc_type)
    X_test_padded = pad_sequences(X_test_sequences, maxlen=max_length)

    print("✅ Text tokenized and padded for LSTM.")

    # --- 6.2: Convert Sentiment Labels to Numerical Format ---
    # The LSTM model needs numerical labels instead of text
    y_train_encoded = pd.get_dummies(y_train).values
    y_test_encoded = pd.get_dummies(y_test).values

    # Get the mapping of class names to integers
    label_mapping = {i: col for i, col in
enumerate(pd.get_dummies(y_train).columns)}
    print("\nLabel mapping for LSTM:", label_mapping)

# --- 6.3: Build the LSTM Model ---
if not df.empty:
    embedding_dim = 16

    lstm_model = Sequential([
        Embedding(vocab_size, embedding_dim),
        SpatialDropout1D(0.2),
        LSTM(64, dropout=0.2, recurrent_dropout=0.2),
        Dense(3, activation='softmax') # 3 output neurons for 3 classes
    ])

    lstm_model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
    print("\n✅ LSTM model built successfully.")
    lstm_model.summary()
```

Appendix D: LSTM Training Call

```
# --- 6.4: Train the LSTM Model (with Class Weights) ---
if not df.empty:
    num_epochs = 5
    batch_size = 64

    print("\nTraining LSTM model...")
    history = lstm_model.fit(X_train_padded, y_train_encoded,
                             epochs=num_epochs,
                             batch_size=batch_size,
                             validation_data=(X_test_padded,
y_test_encoded),
                             class_weight=class_weights, # <-- HERE is
where we use the weights
                             verbose=1)
    print("✅ LSTM model trained successfully.")
```

Appendix E: Final Model Comparison

```
# --- 8.1: Create a Summary of Performance Metrics ---

# Generate classification reports as dictionaries, suppressing the
warning for Naive Bayes
report_nb = classification_report(y_test, y_pred_nb, output_dict=True,
zero_division=0) # <-- ADD zero_division=0 HERE
# For LSTM, we need to use the numerically encoded labels and target
names
report_lstm = classification_report(y_test_single_label, y_pred_lstm,
target_names=class_names, output_dict=True)

# Extract key metrics into a structured dictionary
comparison_data = {
    'Metric': ['Precision (negative)', 'Recall (negative)', 'F1-Score
(negative)',
               'Precision (positive)', 'Recall (positive)', 'F1-Score
(positive)',
               'Overall Accuracy'],
    'Naive Bayes': [
        f"{report_nb['negative']['precision']:.2f}",
        f"{report_nb['negative']['recall']:.2f}",
        f"{report_nb['negative']['f1-score']:.2f}",
        f"{report_nb['positive']['precision']:.2f}",
        f"{report_nb['positive']['recall']:.2f}",
        f"{report_nb['positive']['f1-score']:.2f}",
        f"{report_nb['accuracy']:.2f}"
    ],
    'LSTM': [
        f"{report_lstm['negative']['precision']:.2f}",
        f"{report_lstm['negative']['recall']:.2f}",
        f"{report_lstm['negative']['f1-score']:.2f}",
        f"{report_lstm['positive']['precision']:.2f}",
        f"{report_lstm['positive']['recall']:.2f}",
        f"{report_lstm['positive']['f1-score']:.2f}",
        f"{report_lstm['accuracy']:.2f}"
    ]
}

# Create and display a Pandas DataFrame for easy comparison
df_comparison = pd.DataFrame(comparison_data)
df_comparison.set_index('Metric', inplace=True)

print("--- Final Model Performance Comparison ---")
display(df_comparison)
```

9.1.3 Kafka and Spark Data Pipeline

Appendix A: Kafka Requirement.txt (for install dependencies)

```
praw==7.7.1
kafka-python==2.0.2
tensorflow==2.13.0
numpy>=1.22.0
```

Appendix B: Kafka code

```
import praw
from kafka import KafkaProducer
import json
import time
import re
import pickle
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the tokenizer and model
with open('tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)
model = load_model('sentiment_lstm_model.h5')
```

```python
# Load label mapping
with open('label_mapping.json', 'r') as f:
    label_mapping = json.load(f)

# Set up Kafka producer
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

# Set up Reddit API
reddit = praw.Reddit(
    client_id='BxF0i3pvzhAKp8zNsCtieg',
    client_secret='ps2E7FBMfm0Y023PjEyj8QVZWiWvsw',
    user_agent='reddit_streamer/0.1 by SentimentStream'
)

# Choose a subreddit (e.g., r/malaysia)
subreddit = reddit.subreddit("malaysia")

print("📡 Streaming Reddit comments from r/malaysia...")

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"http\S+|www\S+|https\S+", "", text)
    text = re.sub(r"[^a-z\s]", "", text)
    tokens = text.split()
    return tokens

def predict_sentiment(text):
    # Preprocess the text
    processed_text = preprocess_text(text)

    # Convert to sequence
    sequences = tokenizer.texts_to_sequences(['
'.join(processed_text)])

    # Pad the sequence (using max_length=150 to match training)
    padded_sequences = pad_sequences(sequences, maxlen=150)

    # Get prediction probabilities for all classes
    prediction_probs = model.predict(padded_sequences)[0]  # Shape:
(3,)

    # Get the predicted class index and its probability
    predicted_index = np.argmax(prediction_probs)

    # Convert to sentiment label using mapping
    sentiment = label_mapping[str(predicted_index)]  # Convert index to
string for JSON key
    sentiment_score = float(prediction_probs[predicted_index])

    return sentiment, sentiment_score

# Stream comments in real-time
for comment in subreddit.stream.comments(skip_existing=True):
    # Get sentiment prediction
    sentiment, sentiment_score = predict_sentiment(comment.body)

    data = {
        "id": comment.id,
        "body": comment.body,
        "author": str(comment.author),
        "created_utc": comment.created_utc,
        "subreddit": comment.subreddit.display_name,
        "sentiment": sentiment,
        "sentiment_score": sentiment_score
    }

    print(f"🚀 Sending comment: {data['body'][:100]}... | Sentiment:
{sentiment} ({sentiment_score:.2f})")
    producer.send("reddit-sentiment", value=data)
    time.sleep(1)  # prevent Reddit API overload
```

Appendix C: Spark requirements.txt (For install Spark dependencies)

```
pyspark==3.4.1
supabase==1.0.3
python-dotenv==1.0.0
```

Appendix D: Setup Hadoop

```
# Create hadoop directory if it doesn't exist
$hadoopHome = "C:\hadoop"
New-Item -ItemType Directory -Force -Path $hadoopHome
```

```
# Download winutils.exe
$winutilsUrl =
"https://github.com/cdarlint/winutils/raw/master/hadoop-3.2.0/bin/winut
ils.exe"
$winutilsPath = Join-Path $hadoopHome "bin\winutils.exe"
New-Item -ItemType Directory -Force -Path (Split-Path $winutilsPath)
Invoke-WebRequest -Uri $winutilsUrl -OutFile $winutilsPath

# Download hadoop.dll
$hadoopDllUrl =
"https://github.com/cdarlint/winutils/raw/master/hadoop-3.2.0/bin/hadoo
p.dll"
$hadoopDllPath = Join-Path $hadoopHome "bin\hadoop.dll"
Invoke-WebRequest -Uri $hadoopDllUrl -OutFile $hadoopDllPath

# Set HADOOP_HOME environment variable
[System.Environment]::SetEnvironmentVariable("HADOOP_HOME",
$hadoopHome, [System.EnvironmentVariableTarget]::User)

Write-Host "Hadoop binaries have been downloaded and HADOOP_HOME has
been set to $hadoopHome"
Write-Host "Please restart your terminal for the changes to take
effect."
```

Appendix E: Spark Code

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import *
from datetime import datetime
import os
from supabase import create_client, Client

# ========== Environment Setup ==========
# Set correct Hadoop path (if you're on Windows)
os.environ['HADOOP_HOME'] = r"C:\hadoop"
os.environ['PATH'] = r"C:\hadoop\bin;" + os.environ['PATH']

# ========== Supabase Setup ==========
SUPABASE_URL = "https://qavwgrhcqquyjcgdjedp.supabase.co"
SUPABASE_KEY =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6I
nFhdndncmhjcXF1eWpjZ2RqZWRwIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTc1
```

```
MTU0MzA0NywiZXhwIjoyMDY3MTE5MDQ3fQ.uadeKF1kvQt4lWIpqtgoQUvygnDCUOekvvTC
wn_53Hg"  # Do not hardcode this in production
supabase: Client = create_client(SUPABASE_URL, SUPABASE_KEY)

# ========== Spark Session ==========
spark = SparkSession.builder \
    .appName("RedditSentimentStream") \
    .master("local[*]") \
    .config("spark.jars.packages",
"org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.1") \
    .config("spark.driver.memory", "4g") \
    .config("spark.executor.memory", "4g") \
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# ========== Schema ==========
schema = StructType([
    StructField("id", StringType(), True),
    StructField("body", StringType(), True),
    StructField("author", StringType(), True),
    StructField("created_utc", LongType(), True),
    StructField("subreddit", StringType(), True),
    StructField("sentiment", StringType(), True),
    StructField("sentiment_score", FloatType(), True),
])

# ========== Supabase Writer ==========
def write_to_supabase(df, epoch_id):
    records = df.collect()
    for record in records:
        data = {
            "id": record["id"],
            "body": record["body"],
            "author": record["author"],
            "created_utc": record["created_utc"],
            "subreddit": record["subreddit"],
            "sentiment": record["sentiment"],
            "sentiment_score": float(record["sentiment_score"]),
            "processed_at": datetime.now().isoformat(),
        }
        try:
            supabase.table("reddit_comments").upsert(data).execute()
        except Exception as e:
            print("⚠️ Supabase Error:", e)

# ========== Kafka Stream ==========
raw_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "reddit-sentiment") \
    .option("startingOffsets", "latest") \
    .load()

parsed_df = raw_df.select(
    from_json(col("value").cast("string"), schema).alias("data")
).select("data.*")

# ========== Write to Supabase ==========
query = parsed_df.writeStream \
    .foreachBatch(write_to_supabase) \
    .outputMode("append") \
    .trigger(processingTime="10 seconds") \
    .start()

query.awaitTermination()
```

Appendix F: Dashboard Code

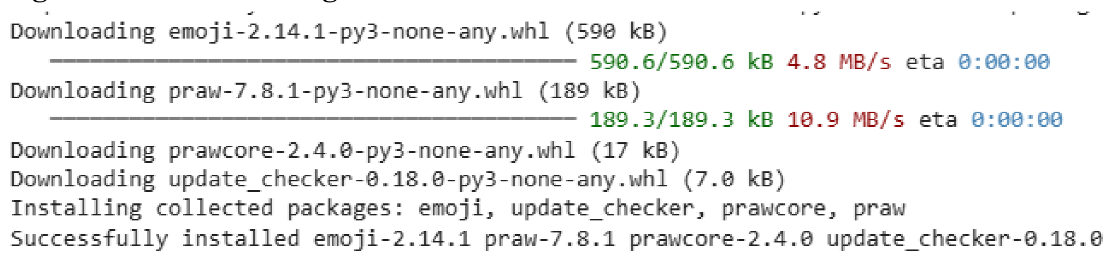
```
"use client"

import { useState, useEffect } from "react"
import { Card, CardContent, CardDescription, CardHeader, CardTitle }
from "@/components/ui/card"
import { Badge } from "@/components/ui/badge"
import { Button } from "@/components/ui/button"
import { RefreshCw, TrendingUp, MessageSquare, BarChart3 } from
"lucide-react"
import { SentimentChart } from "@/components/sentiment-chart"
import { SentimentDistribution } from
"@/components/sentiment-distribution"
import { RecentAnalysis } from "@/components/recent-analysis"
import { MetricsCards } from "@/components/metrics-cards"
import { createClient } from "@supabase/supabase-js"

// Replace the supabase configuration
```

```
const supabaseUrl = "https://qavwgrhcqquyjcgdjedp.supabase.co"
const supabaseKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY ||
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6I
nFhdndncmhjcXF1eWpjZ2RqZWRwIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTE1NDMwNDcs
ImV4cCI6MjA2NzExOTA0N30.35MBkB2jyTCXcYWxXJJ1w8wJK1YDEQc2QIBjmfjqr1g"
const supabase = createClient(supabaseUrl, supabaseKey)

// Update the interface to match your table structure
interface SentimentData {
  id: string
  body: string // Reddit comment text
  author: string
  created_utc: number // Unix timestamp
  subreddit: string
  sentiment: "positive" | "negative" | "neutral"
  sentiment_score: number // Your confidence score
  processed_at: string
}

export default function Dashboard() {
  const [sentimentData, setSentimentData] =
useState<SentimentData[]>([])
  const [isLoading, setIsLoading] = useState(true)
  const [lastUpdated, setLastUpdated] = useState<Date>(new Date())

  const fetchSentimentData = async () => {
    try {
      setIsLoading(true)
      const { data, error } = await supabase
        .from("reddit_comments")
        .select("*")
        .order("processed_at", { ascending: false })
        .limit(1000)

      if (error) {
        console.error("Error fetching data:", error)
        setSentimentData(generateMockData())
      } else {
        setSentimentData(data || [])
      }
      setLastUpdated(new Date())
    } catch (error) {
      console.error("Error:", error)
      setSentimentData(generateMockData())
    } finally {
      setIsLoading(false)
    }
  }

  const generateMockData = (): SentimentData[] => {
    const sentiments: ("positive" | "negative" | "neutral")[] =
["positive", "negative", "neutral"]
    const subreddits = ["malaysia", "malaysians", "kualalumpur",
"penang", "johor"]
    const sampleBodies = [
      "Great product! Highly recommend it to everyone in Malaysia.",
      "Terrible service, very disappointed with the quality.",
      "The event was okay, nothing special but not bad either.",
      "Amazing experience in KL! Will definitely come back again.",
      "Poor customer support, took too long to respond.",
      "Average quality for the price point in Malaysia.",
      "Excellent value for money, very satisfied with this Malaysian
brand!",
      "Not worth the hype, expected much better from local companies.",
      "Decent product, meets basic expectations.",
      "Outstanding service and quality from this Malaysian company!",
    ]

    return Array.from({ length: 100 }, (_, i) => ({
      id: `mock-${i}`,
      body: sampleBodies[Math.floor(Math.random() *
sampleBodies.length)],
      author: `user${Math.floor(Math.random() * 1000)}`,
      created_utc: Math.floor((Date.now() - Math.random() * 7 * 24 * 60
* 60 * 1000) / 1000),
      subreddit: subreddits[Math.floor(Math.random() *
subreddits.length)],
      sentiment: sentiments[Math.floor(Math.random() *
sentiments.length)],
      sentiment_score: Math.random() * 0.4 + 0.6,
      processed_at: new Date(Date.now() - Math.random() * 7 * 24 * 60 *
60 * 1000).toISOString(),
    }))
  }

  useEffect(() => {
    fetchSentimentData()

    const subscription = supabase
      .channel("sentiment_changes")
      .on("postgres_changes", { event: "*", schema: "public", table:
"reddit_comments" }, () => {
        fetchSentimentData()
      })
      .subscribe()

    // Refresh data every 30 seconds
    const interval = setInterval(fetchSentimentData, 30000)

    return () => {
      subscription.unsubscribe()
      clearInterval(interval)
    }
  }, [])

  return (
    <div className="min-h-screen bg-gradient-to-br from-blue-50
to-indigo-100 p-4">
      <div className="max-w-7xl mx-auto space-y-6">
        {/* Header */}
        <div className="flex flex-col sm:flex-row justify-between
items-start sm:items-center gap-4">
          <div>
            <h1 className="text-3xl font-bold text-gray-900">Real-Time
Sentiment Analysis Dashboard</h1>
            <p className="text-gray-600 mt-2">Monitoring Malaysian
social media and review sentiment in real-time</p>
          </div>
          <div className="flex items-center gap-3">
            <Badge variant="outline" className="text-sm">
              Last updated: {lastUpdated.toLocaleTimeString()}
            </Badge>
            <Button onClick={fetchSentimentData} disabled={isLoading}
size="sm" className="gap-2">
              <RefreshCw className={`h-4 w-4 ${isLoading ?
"animate-spin" : ""}`} />
              Refresh
            </Button>
          </div>
        </div>

        {/* Metrics Cards */}
        <MetricsCards data={sentimentData} isLoading={isLoading} />

        {/* Charts Row */}
        <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
          <Card>
            <CardHeader>
              <CardTitle className="flex items-center gap-2">
                <TrendingUp className="h-5 w-5" />
                Sentiment Trends Over Time
              </CardTitle>
              <CardDescription>Real-time sentiment analysis results
from the past 7 days</CardDescription>
            </CardHeader>
            <CardContent>
              <SentimentChart data={sentimentData} />
            </CardContent>
          </Card>

          <Card>
            <CardHeader>
              <CardTitle className="flex items-center gap-2">
                <BarChart3 className="h-5 w-5" />
                Sentiment Distribution
              </CardTitle>
              <CardDescription>Overall sentiment breakdown across all
sources</CardDescription>
            </CardHeader>
            <CardContent>
              <SentimentDistribution data={sentimentData} />
            </CardContent>
          </Card>
        </div>

        {/* Recent Analysis */}
        <Card>
          <CardHeader>
            <CardTitle className="flex items-center gap-2">
              <MessageSquare className="h-5 w-5" />
              Recent Sentiment Analysis
            </CardTitle>
            <CardDescription>Latest processed texts and their sentiment classifications</CardDescription>
          </CardHeader>
          <CardContent>
            <RecentAnalysis data={sentimentData.slice(0, 20)} />
          </CardContent>
        </Card>
      </div>
    </div>
  )
}
```

9.2. Logs

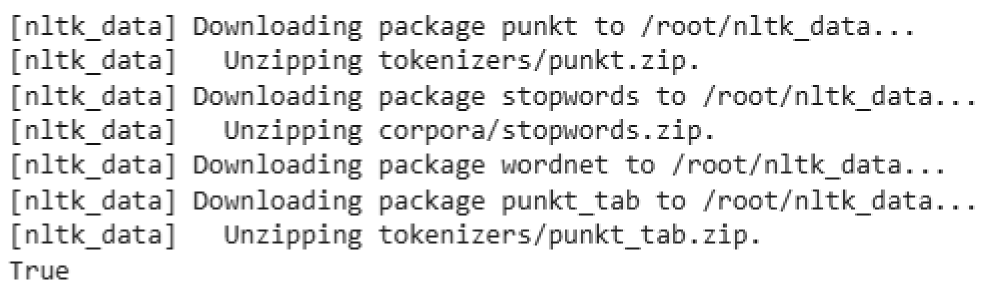
9.2.1 Data Acquisition & Preprocessing

Log Entries from Package Installation



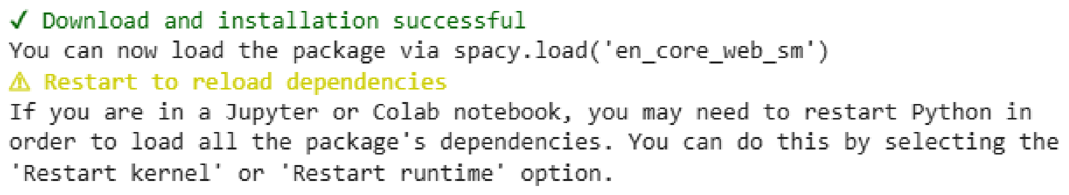
```
Downloading emoji-2.14.1-py3-none-any.whl (590 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 4.8 MB/s eta 0:00:00
Downloading praw-7.8.1-py3-none-any.whl (189 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 189.3/189.3 kB 10.9 MB/s eta 0:00:00
Downloading prawcore-2.4.0-py3-none-any.whl (17 kB)
Downloading update_checker-0.18.0-py3-none-any.whl (7.0 kB)
Installing collected packages: emoji, update_checker, prawcore, praw
Successfully installed emoji-2.14.1 praw-7.8.1 prawcore-2.4.0 update_checker-0.18.0
```

Log Entries from NLTK Data Downloads



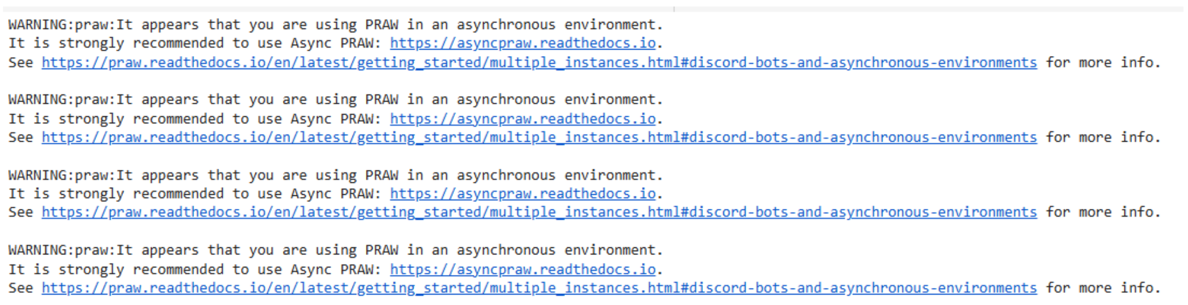
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

Log Entries from spaCy Model Download



```
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

Log Entries from PRAW Warnings during Data Acquisition



```
WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more info.

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more info.

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more info.

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more info.
```